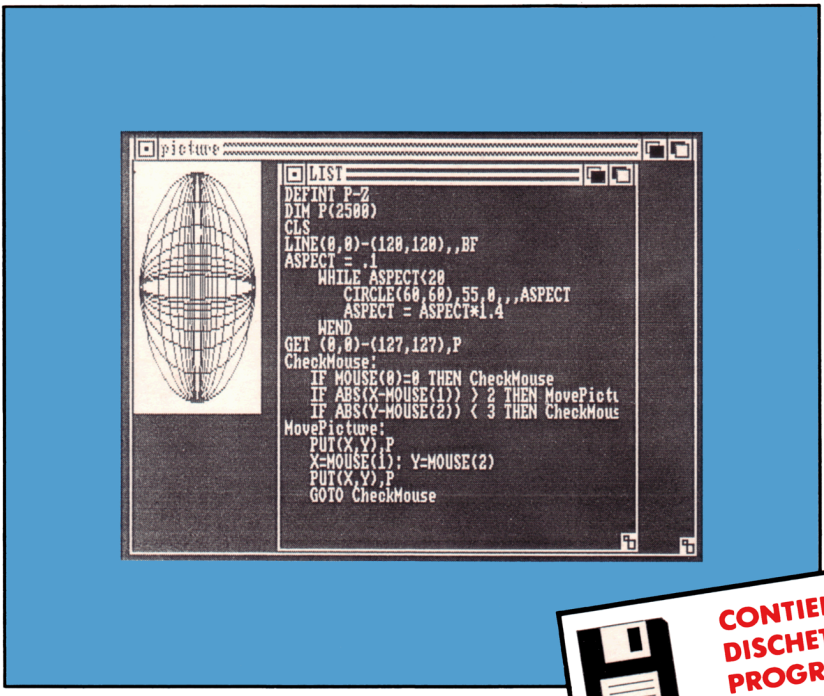
FREE TIME EDITIONS

AmigaBASIC



CONTIENE DISCHETTO PROGRAMMI

Ruegheimer/Spanik

AMIGA

Un libro **DATA BECKER**

Hannes Ruegheimer - Christian Spanik

AmigaBASIC

Traduzione ed impaginazione:
Enrico Barichella - Maurizio Beretta - Mario Dell'Oca

Stampa: Iniziative Grafiche Srl, Via XXV Aprile 3, S. Donato Milanese (MI)

Titolo originale: "AmigaBASIC", Ruegheimer - Spanik

Indice Generale

INTRODUZIONE

Argomenti del libro

Nella stesura di questo libro sull'AmigaBASIC si è tenuto conto sia delle esigenze dei principianti che di quelle dei programmatori più esperti. A tale proposito sono state realizzate due diverse introduzioni tra le quali è possibile scegliere a seconda del diverso livello di esperienza.

I capitoli che seguono l'introduzione sono invece stati scritti per qualsiasi tipo di lettore. Chi non ha esperienza nel campo dei computer dovrebbe leggere il libro attentamente, capitolo per capitolo, e non saltare a capitoli successivi in quanto le informazioni in ognuno di essi si basano sull'utilizzo di quelle fornite in precedenza. Chi invece ha già una discreta esperienza probabilmente conosce già alcuni degli argomenti trattati. Tuttavia, anche i programmatori esperti dovrebbero leggere in sequenza ogni capitolo in quanto molte caratteristiche di Amiga differiscono leggermente da quelle di altri computer.

Nel testo compariranno spesso messaggi preceduti dal simbolo • come il seguente:

• Accendere il monitor e l'unità centrale.

Tali messaggi rappresentano operazioni da eseguire per produrre particolari effetti.

Nel testo sono riportati anche numerosi termini tecnici; essi compariranno scritti in *corsivo* quando vengono utilizzati per la prima volta. E' possibile trovarne la definizione nel breve glossario incluso nell'appendice E.

Le appendici

A proposito delle appendici, è necessario specificare che esse contengono numerose informazioni di utilità pratica.

Nei primi approcci alla programmazione si tende a commettere un numero elevato di errori. AmigaBASIC fornisce, attraverso *messaggi di errore*, informazioni che consentono di comprendere il tipo di errore commesso. Spesso si tratta di errori di battitura o di valori inesatti forniti dall'utente. Nell'appendice A vengono elencati, in ordine alfabetico, i casi più frequenti. Non è stato comunque possibile includere tutti gli errori dato che ne esistono molti e di vario tipo ed inoltre essi dipendono dal programma in esecuzione e dallo stato della macchina. L'appendice A fornisce indicazioni su ciò che ha causato l'errore e su cosa è possibile fare per correggerlo.

Terminata la lettura di questo libro, si dovrebbe essere in grado di realizzare autonomamente programmi in BASIC. E' possibile però che talvolta sorgano dubbi e domande sull'uso di un particolare comando e sulla sua utilità all'interno di un programma. A tale proposito nel libro è stata inclusa l'appendice B che riporta una breve descrizione di tutti i comandi disponibili; dato che non tutti i comandi e le opzioni vengono utilizzate nei programmi presentati nel libro, questa appendice risulterà essere un valido punto di riferimento fin dall'inizio.

E' possibile che, in fase di pubblicazione, si siano verificati degli errori tipografici per cui alcuni dei lunghi programmi presentati possono contenere qualche errore di stampa, benchè essi siano stati controllati molto attentamente prima di essere mandati in tipografia.

Per fornire almeno una copia esatta dei programmi presentati nel libro, allo stesso è stato allegato un dischetto contenente tutti i programmi privi di errori. Ognuno di questi programmi è stato direttamente provato su Amiga; per cui nei listati contenuti nel dischetto non dovrebbero esserci errori. Se quindi dovesse sorgere qualche dubbio circa la correttezza di una linea o di un comando, è consigliabile effettuare una verifica utilizzando il dischetto allegato al libro.

La Commodore fornisce un certo numero di programmi di esempio nel dischetto Extras. L'appendice D illustra brevemente ciò che essi fanno e cosa è possibile fare utilizzandoli.

I differenti modelli Amiga

Il testo ed i programmi contenuti in questo libro sono stati scritti per i modelli Amiga 500, 1000 e 2000. Come noto, Amiga è stata sottoposta a numerose modifiche da quando fu presentato il modello originale (il 1000). A partire dal 1987, la Commodore ha iniziato a produrre due nuove macchine: il modello Amiga 500 (un unico dispositivo contenente tastiera, unità centrale e disk drive) ed il modello Amiga 2000 (un "sistema aperto" che offre anche la compatibilità MS-DOS). Benchè il modello Amiga 1000 non venga più prodotto, ancor oggi un gran numero di essi viene utilizzato.

Questo libro è stato realizzato in modo tale da risultare utile ai possessori di qualunque modello Amiga. Grazie alla piena compatibilità esistente tra i tre modelli Amiga, non è stato troppo difficile perseguire questo obiettivo; in ogni caso, quando necessario, vengono incluse note particolari riguardo le istruzioni da eseguire solamente su un determinato modello Amiga.

Nota per i possessori del modello Amiga 1000

Nel caso si possieda il modello Amiga 1000 con 256K di RAM (Random Access Memory, memoria ad accesso casuale) senza espansione di memoria, è necessario porre particolare attenzione nell'utilizzo della stessa (Questa nota non si applica ai modelli 500 e 2000 in quanto essi possiedono direttamente una quantità superiore di memoria). Un modello Amiga 1000 senza espansione si mostra carente dal punto di vista delle capacità elaborative, rispetto agli altri modelli. E' quindi consigliabile acquistare una espansione di memoria da 256K (la maggior parte dei programmi descritti in questo libro sono rivolti a modelli Amiga con 512K di memoria, corrispondente alla quantità di memoria fornita con i modelli 500 e 2000). E' preferibile acquistare un'espansione di memoria prima di pensare di espandere il sistema con un secondo disk drive o altre periferiche.

In ogni caso, anche se non si possiedono 512K di memoria, è possibile utilizzare AmigaBASIC, oltre ad eseguire la maggior parte dei programmi contenuti in questo libro.

Si suppone inoltre che, avendo a disposizione un modello 1000, si stia utilizzando la versione 1.2 dei dischetti Kickstart, Workbench ed Extras (i modelli 500 e 2000 non utilizzano il dischetto Kickstart poichè i dati di inizializzazione sono contenuti nella memoria ROM del computer). Nel caso non si possiedano tali dischetti, rivolgersi al proprio rivenditore per otte-

nerli il più presto possibile. Se si possiede un modello 500 e 2000, i dischetti a disposizione sono sicuramente quelli corretti.

Una ulteriore nota sul dischetto Extras: la Commodore ha utilizzato nomi diversi per lo stesso dischetto, generalmente Extras o Extras 1.2. Per motivi di chiarezza all'interno del libro è stato utilizzato sempre lo stesso nome; se il dischetto Extras che si possiede ha un diverso nome, è consigliabile rinominarlo con il nome qui utilizzato per evitare confusione. La sezione 3.1 riporta le istruzioni per effettuare tale operazione.

Visualizzazione dei dati sullo schermo
Qualunque riferimento ai colori utilizzati si basa sui colori standard del Workbench. Inoltre, tutti i programmi descritti nel libro sono stati realizzati per un utilizzo su uno schermo in cui possono venire visualizzati 80 caratteri. Ciò implica che, per ottenere una corretta visualizzazione dei dati sul monitor, è necessario selezionare il valore 80 nel gadget Text della finestra Preferences. Ciò può essere fatto effettuando un click sul numero 80 del gadget Text seguito da un click sul gadget Save o Use.

NOTA:
Per avere a disposizione una maggior quantità di memoria è possibile rimuovere il dischetto RAM effettuando le seguenti operazioni:

- Accendere Amiga.
- Inserire, quando è richiesto, il dischetto Workbench.
- Non appena compare lo schermo azzurro, premere i tasti <Ctrl> <D> contemporaneamente per interrompere l'escuzione della sequenza di inizializzazione (startup-sequence) ed attivare il CLI (command line interface).
- Quando compare il simbolo 1>, battere loadwb e premere il tasto <Return>. In tal modo viene effettuato il caricamento del Worbench.
- Battere endcli e premere il tasto <Return> per disattivare il Cli e restituire il controllo al Workbench.

Scelta dell'introduzione

Si ha già una discreta esperienza nell'utilizzo dei computer? Si conosce già Amiga? Rispondendo a queste domande si dovrebbe essere in grado di determinare il proprio livello di abilità e quindi la categoria di lettori a cui si appartiene.

Nel libro sono presenti capitoli introduttivi diversi da cui è possibile iniziare la lettura. La prima introduzione è dedicata a chi non ha una grande esperienza di computer ed ha appena acquistato Amiga. La seconda introduzione è invece rivolta a coloro i quali possiedono già molta esperienza nel campo dei computer ed hanno già utilizzato Amiga o ne conoscono le caratteristiche principali. Si consiglia di valutare bene la propria preparazione prima di scegliere l'introduzione da cui partire.

In ogni caso, la decisione che si sta prendendo potrebbe essere tradotta in Basic, come segue:

```
IF NOT esperto THEN GOTO Introduzione 1
IF esperto THEN GOTO Introduzione 2
```

Non è comunque il caso di far prendere questa decisione ad Amiga. Battere e provare ad eseguire queste linee porterebbe al messaggio d'errore Syntax error. Come mai? Le risposte a questa ed a altre domande saranno fornite nelle pagine seguenti. Dopo aver letto questo libro rimarranno pochi dubbi relativi ad AmigaBASIC e ci si potrà divertire utilizzando a fondo Amiga.

Introduzione 1: il mouse, le finestre ed il Workbench

Avendo scelto di leggere questo capitolo, si pensa di non aver sufficenti conoscenze a proposito del Workbench e delle principali caratteristiche di Amiga. Oppure si ritiene che un piccolo ripasso non faccia poi così male. Per questi motivi rivedremo le azioni fondamentali per avviare ed utilizzare Amiga.

• Accendere il monitor ed Amiga.

Inizializzazione di Amiga: il Kickstart
Appena viene acceso Amiga, il monitor presenta uno schermo grigio: Amiga sta effettuando alcune operazioni di controllo sul buon funzionamento di tutti i suoi dispositivi. Successivamente lo schermo diventa bianco e compare un disegno raffigurante una mano che impugna un dischetto. Ecco il primo incontro con una delle più importanti caratteristiche di Amiga: un linguaggio basato su simboli o *icone*. Se si possiede il modello Amiga 1000, l'icona sullo schermo indica di inserire il dischetto Kickstart necessario al modello 1000 prima di poter eseguire qualunque operazione.

Questa è una delle principali differenze fra i tre modelli Amiga: i modelli 500 e 2000 non necessitano del dischetto Kickstart per il loro avviamento. Se si possiede il modello 500 o il modello 2000, è possibile saltare la parte seguente e riprendere la lettura dalla descrizione delle operazioni riguardanti il Workbench.

• Se si possiede Amiga 1000, inserire il dischetto Kickstart.

Assicurarsi di inserire il dischetto esattamente come mostrato dal disegno sullo schermo, in modo tale quindi che la parte metallica sia rivolta verso il disk drive, mentre l'etichetta dovrebbe venire a trovarsi tra le dita.

Il sistema operativo

Il dischetto Kickstart contiene il *sistema operativo* di Amiga 1000, un programma che informa il computer su come eseguire le operazioni. Senza un programma di tale tipo, anche il più avanzato computer risulta praticamente inutilizzabile. Il sitema operativo di Amiga, ad esempio, indica al computer come reagire agli input forniti mediante la tastiera o tramite il mouse oppure come visualizzare sullo schermo testo e grafica. Le informazioni del sistema operativo vengono memorizzate in un'area speciale di memoria e non vengono cancellate finchè il computer rimane acceso. Il caricamento del sistema operativo contenuto nel Kickstart da parte di Amiga avviene in circa 20 secondi. In seguito Amiga, attraverso un'altra icona, richiede l'inserimento di un dischetto di nome Workbench.

- Dopo che si è spenta la luce del disk drive, premere il pulsante, posto sotto al disk drive, per estrarre il dischetto.

Il dischetto Kickstart viene parzialmente espulso dal drive.

- Estrarre il dischetto Kickstart.

Il Workbench

- Inserire il dischetto Workbench nel drive.

Il drive si mette in funzione ed Amiga carica il Workbench. Viene visualizzato uno schermo azzurro nella cui parte superiore compare una linea bianca (chiamata *riga Titolo*) sotto alla quale viene visualizzato un disegno di un disco: si tratta dell'icona del Workbench che si riferisce al dischetto presente nel drive. Se viene inserito nel drive un nuovo dischetto, comparirà sullo schermo Workbench la relativa icona.

- Spostare il mouse.

E' possibile notare come, in seguito allo spostamento del mouse, si ottenga un corrispondente spostamento della freccia rossa presente sullo schermo. Questa freccia viene detta *puntatore* del mouse.

- Spostare il mouse in modo da posizionare il puntatore sull'icona del dischetto Workbench e premere una volta il pulsante sinistro del mouse.

Come si può vedere l'icona cambia colore diventando nera. Si verifica questo in quanto il pulsante sinistro del mouse consente di attivare le icone presenti sullo schermo. Amiga modifica il colore dell'icona in modo che sia possibile notare immediatamente quale oggetto sia attualmente attivo. Premendo nuovamente il pulsante sinistro del mouse, dopo aver posizionato il puntatore nella parte azzurra dello schermo, si disattiva l'icona del dischetto il cui colore ritorna ad essere bianco. In generale è possibile attivare solamente un'icona per volta.

- Spostare il mouse in modo da posizionare il puntatore sull'icona del dischetto Workbench. Premere due volte in rapida successione il pulsante sinistro del mouse.

L'atto di premere due volte in rapida sucessione il pulsante sinistro del mouse viene detto *doppio-click*. Il puntatore del mouse modifica la propria forma, trasformandosi in due nuvolette su cui compare la scritta Zzz. Questo nuovo puntatore viene detto *puntatore d'attesa*. Amiga sta utilizzando un linguaggio iconico per comunicare che sta effettuando alcune operazioni ed è quindi necessario attendere fino al termine di tali attività. Compare a questo punto sullo schermo una cornice contenente quattro diversi tipi di icona, come mostrato nella figura 1.

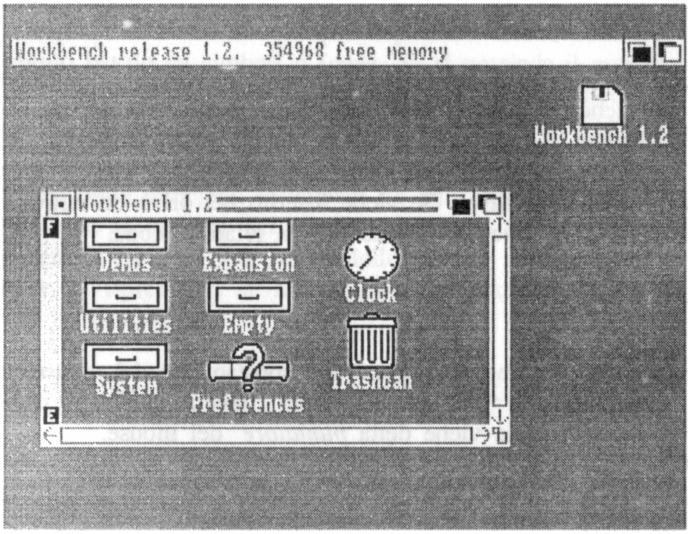


Figura 1: lo schermo Workbench in cui è presente la finestra omonima.

Sullo schermo del modello Amiga 500 potrebbe essere presente anche un'icona di un dischetto di nome RAM Disk (si veda il paragrafo 3.5 per ulteriori informazioni). Nella riga Titolo viene visualizzata la quantità di memoria libera (che dipende dal modello di Amiga che si possiede e dalla memoria installata).

La cornice contenente le varie icone viene detta *finestra*. Alcune finestre vengono utilizzate per visualizzare i contenuti dei dischetti, come nel caso della figura 1 che mostra il contenuto del dischetto Workbench.

La riga Titolo

Ogni finestra possiede un nome, nel presente caso Workbench. E' possibile che il nome compaia leggermente sfumato e che non si riesca a leggerlo bene; questo significa che la finestra non è stata attivata. Per attivarla è necessario posizionare il puntatore all'interno della finestra ed effettuare un click. Ora il nome sarà visualizzato con caratteri più marcati e risulterà facilmente leggibile.

- Posizionare il puntatore del mouse sulla riga Titolo della finestra Workbench, premere il pulsante sinistro del mouse e, tenendolo premuto, spostare il mouse.

Come si può notare, compare un rettangolo arancione delle stesse dimensioni della finestra che si sposta sullo schermo in seguito al movimento del mouse. Non appena si rilascia il pulsante sinistro del mouse, la finestra viene riposizionata sullo schermo in corrispondenza della posizione del puntatore.

I gadget Back e Front

I piccoli rettangoli sovrapposti posti nell'angolo in alto a destra della finestra consentono di scegliere quale finestra debba essere posta in primo piano nel caso in cui vi siano più finestre aperte contemporaneamente. Essi vengono chiamati gadget back e front. Per comprenderne il funzionamento si consiglia di provare ad utilizzarli.

- Effettuare un doppio click sull'icona a forma di orologio di nome Clock.

Dopo pochi secondi compare sullo schermo un orologio; è probabile che

l'ora visualizzata non sia corretta. Per ora questo non è molto importante; è possibile comunque regolare l'ora di questo orologio utilizzando il programma Preferences (si consulti il manuale di Amiga, oppure il libro "Amiga! Primi Passi" edito dalla F.T.E., per le istruzioni relative all'uso delle Preferences).

L'orologio dovrebbe coprire parte della finestra Workbench. In caso contrario spostare la finestra Clock in modo tale da sovrapporla parzialmente a quella del Workbench.

- Posizionare il puntatore del mouse sul simbolo raffigurante i due rettangoli sovrapposti collocato nell'angolo in alto all'estrema destra della finestra Workbench ed effettuare un click su tale simbolo.

L'orologio scompare dietro alla finestra Workbench.

Effettuando un click sul simbolo analogo al precedente posto leggermente più a sinistra, si riporta l'orologio in primo piano. Questi due simboli vengono detti *gadget back e front* e risultano molto utili quando sullo schermo sono aperte numerose finestre per collocarle in primo od in secondo piano.

Il gadget di dimensionamento

Spostando il *gadget di dimensionamento* di una finestra (posto nell'angolo in basso a destra), si modifica la dimensione della finestra stessa. Spostare il gadget di dimensionamento significa posizionare il puntatore del mouse sul corrispondente simbolo, premere il pulsante sinistro del mouse e, tenendolo schiacciato, spostare il mouse. Come nel precedente caso relativo allo spostamento di una finestra, la nuova dimensione della finestra, fissata non appena si rilascia il pulsante sinistro del mouse, viene indicata da un rettangolo arancione.

La barra di scorrimento

La *barra di scorrimento* segnala quando una finestra non è sufficientemente grande per visualizzare tutti i suoi contenuti. In tal caso, in questa riga, compare un rettangolo azzurro che consente di rilevare la porzione di finestra attualmente visibile. Spostando la riga di scorrimento è possibile visualizzare le altre parti della finestra. In alternativa, è possibile effettua-

re un click sulle *frecce di scorrimento*, poste a destra e a sinistra delle righe di scorrimento, per spostarsi lungo la finestra. Le finestre possiedono barre di scorrimento sia orizzontali che verticali.

L'indicatore del disco

Sul lato sinistro della finestra compare un rettangolo arancione detto *indicatore del disco* il quale specifica la quantità di memoria del dischetto utilizzata. La lettera E sta per vuoto (E=Empty), mentre F sta per pieno (F=Full). Più alto è il rettangolo arancione più pieno risulta il dischetto.

Il gadget di chiusura

Effettuando un click sul *gadget di chiusura*, posto nell'angolo in alto a sinistra della finestra, è possibile chiudere la finestra facendola scomparire dallo schermo.

I cassetti

I *cassetti* costituiscono utili luoghi in cui riporre i propri programmi. Effettuando un doppio-click sull'icona di un cassetto, si provoca l'apertura di una nuova finestra contenente altri programmi o cassetti. E' quindi possibile immaginare un dischetto come uno schedario con molti cassetti. Si possono radunare i vari programmi in diversi cassetti a seconda della loro categoria. E' possibile poi visualizzare il contenuto di tali cassetti aprendoli. Per aprire uno dei cassetti del dischetto Workbench eseguire la seguente procedura:

- Effettuare un doppio-click su uno dei cassetti contenuti nella finestra Workbench

Si ottiene l'apertura di una nuova finestra associata al cassetto selezionato in cui è possibile trovare alcune icone.

- Chiudere tale finestra effettuando un click sul suo gadget di chiusura.

E' possibile inserire in un cassetto qualsiasi cosa; si possono avere cassetti all'interno di cassetti contenuti in altri cassetti, ecc.

Il Trashcan

Il *cestino* (Trashcan) consente di "gettare via" programmi di cui non si ha più bisogno. Anche in questo caso l'icona del cestino evoca un oggetto di uso quotidiano. Una volta posto nel cestino un programma non viene immediatamente cancellato, ma rimarrà nel cestino finchè esso non viene *svuotato* (secondo quanto mostrato più avanti). E' possibile aprire il cestino con la stessa procedura utilizzata per un cassetto e recuperare quindi qualunque cosa in esso contenuta.

• Effettuare un doppio-click sull'icona Trashcan.

Il cestino dovrebbe essere attualmente vuoto.

• Chiudere la finestra Trashcan.

I menù

Come è possibile cancellare dati e programmi? La risposta a questa domanda porta ad introdurre il concetto di *menu' a tendina* (pull-down menu).

• Premere e tenere schiacciato il pulsante destro del mouse.

E' possibile notare come, nella riga Titolo dello schermo, compaiano tre parole:

Workbench Disk Special

• Spostare il puntatore del mouse, tenendo sempre premuto il pulsante destro, sulla parola Special.

Non appena il puntatore del mouse si sovrappone alla parola Special, compare il corrispondente menù a tendina contenente un elenco di opzioni. Questo menù contiene, ad esempio, le opzioni Cleanup, Last Error, Redraw, Snapshot e Version. Alcune di queste opzioni non sono, per ora, interessanti; è inoltre possibile che alcune di esse siano inattive.

• Tenendo il pulsante destro sempre premuto, spostare verso il basso il puntatore del mouse.

Mentre il puntatore del mouse viene spostato lungo il menù, si può notare come le opzioni attive vengano evidenziate in nero. Il pulsante destro del mouse viene detto *pulsante di menù* poichè viene usato per accedere ai menù a tendina. Il pulsante sinistro del mouse viene invece detto *pulsante di selezione* in quanto consente di selezionare icone. I termini "pulsante di selezione", "pulsante di azione" o "pulsante di attivazione", eventualmente utilizzati in altri libri, sono tutti sinonimi per riferirsi al pulsante sinistro del mouse.

- Rilasciare il pulsante destro del mouse e chiudere tutte le finestre sullo schermo.

- Attivare il dischetto Workbench (effettuando un click su di esso).

- Selezionare l'opzione Open dal menù a tendina Workbench.

Divertirsi con le finestre

Le istruzioni appena fornite possono venire eseguite anche nel seguente modo:

- Attivare il dischetto Workbench; premere il pulsante destro del mouse e, tenendolo schiacciato, muovere il puntatore sulla parola Workbench; sempre tenendo premuto il pulsante destro del mouse, spostare il puntatore sulla parola Open e, non appena lo sfondo della parola Open diventa nero, rilasciare il pulsante.

Si può notare come tali operazioni abbiano lo stesso effetto di un doppio-click effettuato sull'icona del dischetto Workbench: si ottiene infatti l'apertura della finestra Workbench. I metodi per accedere al contenuto di un dischetto sono ora due.

- Attivare l'icona del dischetto Workbench, nel caso non lo sia già, effettuando un click su di essa.

- Selezionare l'opzione Cleanup dal menù a tendina Special.

L'opzione Cleanup consente di riordinare le icone all'interno di una finestra; questa operazione si rivela molto utile quando si hanno numerose icone poste disordinatamente in una finestra. Cleanup effettua in modo

automatico la loro sistemazione.

E' stata precedentemente lasciata in sospeso la spiegazione su come effettuare lo "svuotamento" del cestino cancellando dal dischetto i programmi di cui non si ha più bisogno. Prima di fornire la relativa risposta, è necessario rilevare come i programmi contenuti nel dischetto Workbench siano tutti fondamentali per il buon funzionamento di Amiga e che quindi nessuno di essi dovrebbe essere cancellato.

Copie di lavoro

E' sempre preferibile lavorare utilizzando una *copia di lavoro* del dischetto Workbench originale. Il manuale fornito con Amiga ed il libro "Amiga! Primi Passi" spiegano dettagliatamente come effettuare copie di dischetti. E' consigliabile effettuare copie di tutti i dischetti forniti con Amiga. Nel resto del libro sarà dato per scontato che tali copie di lavoro siano già state effettuate, per cui si consiglia di procedere ora a questa operazione.

La procedura per cancellare i programmi contenuti nel cestino è molto semplice: quando si desidera "gettare" tutto ciò che è presente nel cestino, è sufficiente attivare l'icona Trashcan e selezionare l'opzione Empty Trash dal menù a tendina Disk: i file verranno persi per sempre ed il cestino risulterà vuoto.

Si conclude qui la prima introduzione al Workbench. Ulteriori caratteristiche e precisazioni sul Workbench vengono trattate in dettaglio nel libro Amiga! Primi Passi disponibile presso la F.T.E. Free Time Editions.

Dopo questo primo capitolo introduttivo, si dovrebbe essere in grado di addentrarsi nella conoscenza di AmigaBASIC. Naturalmente non dovrebbe recare alcun danno leggere anche il capitolo introduttivo dedicato ai più esperti, senza preoccuparsi se non si comprendono tutte le informazioni fornite poichè le cose più importanti sono già state esaminate in questa introduzione.

Introduzione 2: Workbench: un breve riepilogo per i lettori più esperti

Avendo scelto di iniziare con questa introduzione, si ritiene di avere una sufficiente esperienza con Amiga. Verrà quindi spiegato solo brevemente quanto accade dopo l'avviamento di Amiga e verranno riepilogate le caratteristiche fondamentali del Workbench. Se si incontra qualcosa di incomprensibile, è forse utile tornare all'introduzione 1 per ulteriori chiarimenti.

Non appena viene acceso, Amiga esegue un test interno. Nel caso si possieda il modello 1000, tale test comprende un controllo sull'uscita audio che produrrà un breve suono. Nel caso non si rilevi tale rumore è consigliabile controllare la connessione fra Amiga e la presa audio del monitor.

Il Kickstart

Dopo questo test interno il modello 1000 visualizza l'icona di un dischetto di nome Kickstart. In tale dischetto è contenuto il sistema operativo di Amiga, detto anche *nucleo* (kernel). Il kernel include diverse routine utilizzate da Amiga: ad esempio il controller per l'uscita video, per l'input e l'output, la gestione del multitasking. I contenuti del dischetto Kickstart vengono caricati in una particolare porzione della memoria di Amiga. Tale spazio di memoria viene poi trattato come memoria ROM (Read Only Memory) risultando inaccessibile all'utente. I sistemi operativi dei modelli Amiga 500 e 2000 sono invece contenuti in chip ROM per cui tali modelli non necessitano del dischetto Kickstart.

Il modello Amiga 1000 possiede comunque un vantaggio: è possibile infatti aggiornare il sistema operativo semplicemente cambiando un dischetto, il dischetto Kickstart, cosa non possibile con gli altri due modelli Amiga, per l'aggiornamento dei quali è invece richiesta la sostituzione di un chip.

Il Workbench

Dopo il caricamento del Kickstart (per il modello 1000) o la procedura automatica di inizializzazione (per i modelli 500 e 2000), Amiga necessita di un altro dischetto: il Workbench. Il *Banco di Lavoro* (traduzione del termine Workbench) su cui operare viene appunto caricato dal dischetto Workbench; esso contiene i comandi AmigaDOS, i programmi di sistema (come ad esempio i driver per le varie stampanti) ed altri programmi di utilità. Dopo aver effettuato il caricamento del Workbench, ci si trova di fronte ad un'interfaccia grafica che consente di controllare agevolmente Amiga mediante il mouse e le finestre.

Il puntatore del mouse viene sincronizzato con il movimento del mouse stesso su una superficie liscia. Il pulsante sinistro del mouse viene utilizzato per attivare icone ed eseguire programmi, mentre il pulsante destro viene usato per accedere ai menù a tendina.

Le icone

Il simbolo raffigurante un dischetto, visualizzato nell'angolo in alto a destra dello schermo, rappresenta il dischetto Workbench; sullo schermo compare spesso anche un altro dischetto: il dischetto RAM. Questi due disegni vengono detti *icone*.

Muovendo il puntatore del mouse sull'icona del dischetto Workbench ed effettuando un doppio-click con il pulsante sinistro del mouse, si provoca l'apertura della finestra Workbench al cui interno compaiono altre icone che rappresentano il contenuto del dischetto stesso. Effettuando, ad esempio, un doppio-click sull'icona Clock, si dovrebbe attivare un programma che visualizza l'orologio di sistema.

Le icone associate ai cassetti Demos, Utilities, System ed Empty rappresentano directory che possono anche essere nidificate all'interno di altri cassetti. E' necessario effettuare un doppio-click su tali icone per esaminare i contenuti delle varie directory presenti nel dischetto Workbench.

Lungo i bordi di una finestra compaiono numerosi simboli, molto importanti nella gestione e nell'utilizzo delle finestre stesse, il cui significato viene ampiamente trattato all'interno dell'Introduzione 1.

I menù

Premendo, e tenendo schiacciato, il pulsante destro del mouse è possibile accedere ai menù a tendina; per questo motivo tale pulsante viene detto *pulsante di selezione* o di menù. Effettuando questa operazione compare infatti, nella riga superiore dello schermo, l'elenco dei titoli dei diversi menù a tendina disponibili. Per selezionare una opzione contenuta in uno dei menù, è necessario spostare il puntatore, tenendo sempre premuto il pulsante destro del mouse, sull'opzione desiderata e rilasciare il pulsante stesso. In ogni istante sono comunque disponibili solo alcune delle opzioni, quelle il cui nome è facilmente leggibile; vi sono, infatti, alcune *opzioni mascherate* , il cui nome è di difficile lettura, che risultano inattive e quindi non disponibili. Una cosa simile accade per le finestre; è quindi possibile individuare una finestra attiva dalla facilità con cui è possibile leggerne il titolo.

Il menù Workbench

Le opzioni Open e Close del menù a tendina Workbench consentono di aprire e chiudere icone e finestre. Duplicate consente di effettuare copie di dischetti e programmi. Rename permette di definire un nuovo nome per un programma od un dischetto. Discard fornisce la possibilità di cancellare programmi. Info produce l'apertura di una finestra contenente informazioni relative ad un programma riguardanti, ad esempio, la dimensione, il tipo di programma, le protezioni, etc.

Il menù Disk

L'opzione Empty Trash del menù a tendina Disk rimuove i programmi contenuti nel cestino, cancellandoli dal dischetto. Initialize consente di effettuare la formattazione dei dischetti.

Il menù Special

L'opzione Cleanup del menù a tendina Special effettua un riordinamento delle icone all'interno di una finestra. Last Error visualizza, nella riga Titolo dello schermo, l'ultimo errore rilevato dal sistema operativo. Redraw ridisegna lo schermo. Snapshot "ancora" le icone attive in una particolare posizione all'interno di una finestra. Version visualizza il numero di versione del Workbench che si sta utilizzando.

Il Trashcan

Dopo questa breve analisi dei diversi menù a tendina disponibili, è utile soffermarsi un attimo su uno strumento un po' particolare: il Trashcan (cestino). Il cestino viene utilizzato per "mettere da parte" files che non verranno più usati e che si intendono cancellare. Il cestino non è nient'altro che una directory speciale. I programmi in esso contenuti vengono rimossi dal dischetto attivando l'icona Trashcan e selezionando l'opzione Empty Trash dal menù a tendina Disk. E' importante rilevare come tale operazione sia irreversibile e come sia quindi importante lavorare SEMPRE con copie di lavoro dei dischetti originali, per non trovarsi poi senza un file importante a causa di una cancellazione erronea.

Nei manuali forniti con Amiga e nel libro "Amiga! Primi Passi" viene ampiamente spiegato come effettuare copie dei dischetti e perciò, nel resto del libro, si dà per scontato che siano già state effettuate le copie dei dischetti originali.

1 Realizzare oggetti in movimento

Nei capitoli seguenti verranno esaminate le grandi capacità grafiche di Amiga. Molto probabilmente si è già avuto occasione di vedere alcuni programmi dimostrativi ed è possibile che proprio la grafica di Amiga sia stato uno dei motivi che hanno spinto all'acquisto di questo tipo di computer. Dopo aver letto i prossimi capitoli si sarà in grado di creare programmi BASIC che realizzino la maggior parte dei sofisticati effetti grafici presenti nei vari programmi dimostrativi.

Le pagine seguenti costituiscono una valida introduzione all'animazione con il computer realizzata tramite il linguaggio BASIC. In questo primo capitolo si vedrà ciò che Amiga consente di fare nel campo dell'animazione grafica. Unendo tutti i listati di esempio si otterrà, alla fine del capitolo, un breve programma in grado di creare titoli grafici animati che possono essere eventualmente utilizzati, collegando Amiga ad un videoregistratore, per titolare registrazioni. Anche se non si possede un videoregistratore, ci si potrà ugualmente rallegrare per i risultati ottenuti dal primo programma BASIC realizzato.

1.1 Il dischetto Extras

La prima cosa da fare è caricare dal dischetto Extras tutto ciò di cui Amiga necessita per lavorare in Basic.

ATTENZIONE:
Poichè d'ora in poi i possessori di un modello Amiga con un solo disk drive dovranno spesso cambiare il dischetto inserito nel drive stesso, è necessario fornire alcune importanti avvertenze. **Estrarre un dischetto dal drive solo quando la luce del drive stesso è spenta**, poichè, in caso contario, si potrebbe danneggiare irrimediabilmente il dischetto. Tale precauzione dovrebbe essere tenuta in considerazione soprattutto da chi ha in precedenza lavorato su sistemi nei quali era possibile estrarre un disco anche quando la luce del drive era accesa. Amiga non permette questo.

La prima azione da eseguire consiste nell'effettuare una copia del dischetto Extras. Se non si sa come effettuare tale operazione, riferirsi al manuale fornito con Amiga, oppure consultare il libro "Amiga! Primi Passi".

Terminata tale operazione, se si possiede un secondo drive, inserire il dischetto Extras nel drive esterno; altrimenti, estrarre dal drive interno il dischetto Workbench ed inserire il dischetto Extras.

NOTA:
Ricordarsi di utilizzare il programma Preferences per configurare la larghezza di testo a 80 colonne (si veda l'Introduzione 1 per le relative informazioni) poichè i programmi in questo libro funzionano correttamente solo in tale modalità.

Anche se si estrae il dischetto Workbench, la relativa icona rimane visualizzata sullo schermo poichè Amiga utilizza una zona speciale di memoria in cui annota le informazioni relative al dischetto Workbench.

• Aprire il dischetto Extras.

Il dischetto Extras

Posizionare il puntatore del mouse sull'icona del dischetto Workbench ed effettuare un doppio-click. Sullo schermo compare una finestra contenente sette icone di nome AmigaBASIC, BasicDemos, FD1.2, PCUtil, ReadMe, Tools e Trashcan, delle quali soltanto le prime due risultano per il momento interessanti. Per quanto riguarda le altre icone, ReadMe (leggibile con il Notepad contenuto nel dischetto Workbench) contiene alcune note relative alla versione del dischetto Extras che si sta utilizzando, i cassetti Tools e PCUtil contengono alcuni programmi descritti in dettaglio all'interno dei cassetti stessi, mentre i contenuti del cassetto FD1.2 verranno discussi in seguito. In ogni caso, è consigliabile non rimuovere nessun file contenuto nel dischetto Extras. Il parallepipedo bianco contenente simboli arancioni rappresenta l'icona associata ad AmigaBASIC.

• Effettuare un click su tale icona.

Alla fine del caricamento di AmigaBASIC, come si può vedere in Figura 2, compaiono due finestre di nome BASIC e LIST.

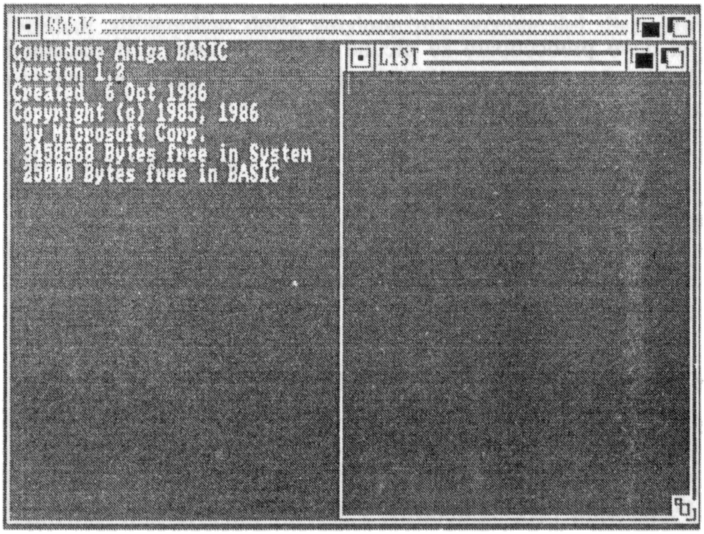


Figura 2: Lo schermo come appare dopo il caricamento di AmigaBASIC

1.2 Introduzione ad AmigaBASIC

Come si puo' notare, dopo il caricamento di AmigaBASIC, compaiono sullo schermo due finestre di nome BASIC e LIST. La finestra LIST è quella attiva, come è possibile rilevare dal suo titolo facilmente leggibile, mentre la finestra BASIC visualizza le seguenti linee:

```
Commodore Amiga BASIC
Version 1.2
Created 6 Oct 1986
Copyright (c) 1985, 1986
by Microsoft Corp.
220536 Bytes free in System
25000 Bytes free in BASIC
```

Non ci si deve preoccupare se alcune di queste linee differiscono leggermente da ciò che viene visualizzato sul proprio monitor, in quanto non vi è alcuna sostanziale differenza.

Il numero di versione
Il numero visualizzato nella seconda linea della finestra BASIC costituisce il numero della versione del programma; quello presentato sopra potrebbe essere completamente differente da quello presente sul proprio monitor.

Le case produttrici di software, come la Microsoft Corp., associano infatti ai loro prodotti un particolare numero che muta in seguito agli aggiornamenti apportati alle versioni originali. Il numero di versione costituisce quindi un buon indice per stabilire l'"età" di un particolare programma. In seguito al continuo rilascio di nuove versioni di AmigaBASIC, nel libro verranno evidenziati gli eventuali errori che si possono incontrare nel caso si utilizzi una versione piuttosto vecchia.

La prima versione ufficiale di un programma è generalmente etichettata Version 1.00. E' possibile tuttavia incontrare, talvolta, versioni numerate 0.99; si tratta di programmi non ancora completamente terminati, di cui non è quindi ancora stata realizzata la versione 1.00.

In seguito alla correzione di piccoli errori contenuti nella versione originale, vengono generalmente rilasciate le versioni 1.01 e 1.02 a cui fa seguito, in alcuni casi, la versione 1.1. Nel caso, invece, di sostanziali modifiche alla versione originale di un programma, il produttore rilascia la versione 2.0.

Questo processo di aggiornamento può però continuare ulteriormente per cui, molto spesso, si ha una versione 2.01 seguita da altre successive versioni. Per alcuni programmi si è giunti, ad esempio, al rilascio della versione 5.21, il che non significa necessariamente che il programma originale fosse pieno di errori.

La versione di AmigaBASIC che si sta utilizzando dovrebbe essere la 1.2. Per adattare AmigaBASIC alle versioni 1.2 del Kickstart e del Workbench, la versione 1.1 è stata riscritta in quanto vi erano alcuni problemi di compatibilità. Non si dovrebbe utilizzare la versione 1.0 di AmigaBASIC poichè essa risulta inadatta. Se non si possiede nessun'altra versione, è consigliabile procedere all'acquisto della versione più recente del dischetto Extras (Extras 1.2).

Data di completamento
La linea successiva a quella relativa alla versione, riporta la scritta Created 6 Oct 1986, data corrispondente al giorno in cui i programmatori di AmigaBASIC hanno inserito l'ultima linea di programma. Tale data risulta ovviamente differente tra le varie versioni.

Copyright
La linea Copyright (c) 1985, 1986 by Microsoft Corp. indica che il BASIC è stato scritto dalla Microsoft Corp., creatrice, tra l'altro, di molte versioni di successo del BASIC. I PC IBM, l'Apple Macintosh e quasi tutti i computer Commodore (dal vecchio PET al Commodore 128) utilizzano versioni BASIC realizzate da Microsoft. Poichè realizzare un interprete BASIC costituisce un grosso impegno finanziario e lavorativo, Microsoft ha brevettato

il suo lavoro riservandosi i diritti di commercializzazione del prodotto.

La memoria disponibile

Le ultime due linee riportano informazioni relative alla memoria disponibile misurata in *byte* (si veda il glossario tecnico contenuto nell'appendice E per la definizione di byte).

Il primo numero fornisce informazioni sul numero di byte liberi nella memoria di Amiga, cioè sulla quantità di memoria che può venire utilizzata dai vari programmi (sia BASIC che no). Tale valore può essere diverso da quello riportato in figura 2, in quanto, oltre che dalle eventuali espansioni di memoria installate nel proprio computer, dipende anche dal numero di finestre e dischetti aperti prima del caricamento di AmigaBASIC. Il numero contenuto nella successiva linea riporta la quantità di memoria disponibile ad AmigaBASIC per i suoi file e programmi. Tale valore è, generalmente, di 25000 byte. Ci si potrebbe allora chiedere come mai AmigaBASIC abbia a disposizione così poca memoria.

Vi sono diverse risposte a tale domanda: innanzitutto, Amiga opera in *multitasking* ed è perciò necessario che vi sia un'adeguata quantità di memoria disponibile agli eventuali altri programmi che possono essere eseguiti contemporaneamente ad AmigaBASIC; in secondo luogo, la grafica a colori che Amiga riesce a generare richiede grandi quantità di memoria (ad esempio, per visualizzare i 4096 colori di cui dispone, Amiga necessita di circa 128k di memoria). Questi ed altri motivi hanno sconsigliato ai progettisti di riservare tutta la memoria ad AmigaBASIC.

1.3 Utilizzare AmigaBASIC

LIST
Analizzando in maggior dettaglio la finestra LIST è possibile notare come essa non sia completamente vuota: nell'angolo in alto a sinistra si può notare un piccolo trattino verticale arancione che rappresenta il cursore BASIC.

Si può attivare la finestra BASIC posizionando il puntatore del mouse in un qualunque punto al suo interno ed effettuando un click con il pulsante destro. Il computer fornisce una conferma dell'avvenuta operazione visualizzando la scritta Ok. E' ora attiva la finestra BASIC; in ogni momento, è comunque possibile riattivare la finestra LIST effettuando un click al suo interno. Finora Amiga si comporta esattamente come quando si interagisce con il Workbench: in un dato istante una sola finestra può essere attiva.

I cursori
E' possibile comunque riscontrare una prima differenza nel numero di cursori disponibili, in quanto ora se ne hanno a disposizione due: il puntatore del mouse ed il cursore BASIC che indica il punto nel quale verranno inseriti i caratteri forniti da tastiera; tale cursore verrà utilizzato spesso in BASIC.

PRINT
Si è ora finalmente in grado di cominciare ad utilizzare AmigaBASIC. Battere la seguente linea:

```
print "ciao"
```

La linea appena inserita dovrebbe comparire sullo schermo nello stesso modo in cui compare sul libro.

NOTA:
In alcune tastiere del modello Amiga 1000, nell'angolo in alto a destra, si trova il tasto <BackSpace>, mentre generalmente sulla tastiera del modello 500 si ha, nella stessa posizione, il tasto (<-). Questi tasti svolgono la stessa funzione e, nel resto del libro si utilizzerà la terminologia "tasto <BackSpace>" per riferirsi ad essi. Allo stesso modo, su alcune tastiere del modello 1000, sotto al tasto <BackSpace> si ha il tasto <Return>, siglato invece generalmente come tasto (↵) sulla tastiera del modello 500. Nel resto del libro ci si riferirà a tale tasto con il nome "tasto <Return>".

Il tasto <Return> costituisce il tasto di invio: Amiga non riconosce i caratteri battuti su una linea finchè non viene premuto tale tasto. Premendo il tasto <Return> si comunica ad Amiga di aver finito di battere la linea corrente, che può ora essere interpretata. Nel caso si commettano errori di battitura, è possibile utilizzare il tasto <BackSpace> per cancellare i caratteri alla sinistra del cursore. E' consigliabile premere il tasto <Return> solo dopo aver controllato che la linea inserita sia senza errori.

Amiga analizza la linea, la esegue e, di conseguenza, visualizza la parola ciao seguita dal consueto messaggio Ok. Il comando impartito ad Amiga richiedeva infatti di visualizzare il messaggio contenuto tra le virgolette; a sua volta Amiga, dopo aver eseguito correttamente il compito assegnatole, si predispone ad accettare nuovi comandi comunicando la propria disponibilità all'utente mediante la visualizzazione del messaggio OK.

E' indifferente battere un'istruzione in caratteri maiuscoli o minuscoli poichè Amiga la interpreta correttamente in qualunque caso: sarebbe stato così possibile battere anche pRiNt CiaO per ottenere lo stesso risultato.

Non dovrebbe essere difficile immaginare ciò che si ottiene fornendo l'istruzione:

```
print "Come stai?"¶
```

Nella linea precedente compare lo strano carattere grafico ¶; nel resto del libro tale simbolo sarà utilizzato per indicare la necessità di premere, alla fine di un'istruzione, il tasto <Return>.

Per rispondere alla precedente domanda, inserire:

```
Grazie, io sto bene¶
```

I messaggi d'errore
Dialogare con il computer non è così semplice. In risposta all'inserimento dell'ultima linea si otterrà un suono seguito dalla visualizzazione di un riquadro (requester) contenente il messaggio "Undefined subprogram" ed un gadget con la scritta OK. Come si può intuire, tale messaggio rappresenta un *messaggio d'errore* con il quale Amiga comunica di non essere riuscita ad interpretare correttamente l'istruzione fornita, in quanto "Grazie, io sto bene" non è un comando BASIC. Dopo aver rilevato il messaggio d'errore è necessario effettuare un click sul gadget OK.

E' possibile chiedersi a cosa servano le virgolette tra cui è compresa la parola ciao.

• Provare a battere la stessa istruzione senza virgolette:

```
print ciao¶
```

In risposta Amiga visualizzerà

```
0
Ok
```

LET
Qual è il significato di quel valore 0?

• Battere:

```
let ciao = 100¶
print ciao¶
```

Le variabili
Si otterrà in questo modo la visualizzazione del numero 100, (invece del numero 0). In questo caso ciao rappresenta una *variabile numerica* analoga alle variabili incontrate in matematica i cui valori risultano spesso incogniti.

Una delle cose che il BASIC consente di fare è l'esecuzione di calcoli matematici. Per poter utilizzare i valori ottenuti da tali calcoli, il BASIC necessita di un luogo in cui memorizzarli. Si ricorre, a tale scopo, all'uso

delle variabili. Il comando LET consente di assegnare un particolare valore ad una variabile. Battere la seguente linea:

```
print 7+5¶
```

Amiga visualizzerà 12 come risultato. Se si batte:

```
let ciao = 7+5¶
print ciao¶
```

Amiga visualizzerà nuovamente 12. Tuttavia, in questo secondo caso, il risultato della somma è stato memorizzato nella variabile ciao.
Battendo:

```
print ciao¶
```

si ottiene nuovamente 12.

Se viene richiesto di visualizzare il contenuto di una variabile a cui non è stato ancora assegnato alcun valore, si ottiene la visualizzazione del numero 0, poichè tutte le variabili vengono inizializzate automaticamente con il valore 0.

Se si desidera abbreviare le operazioni di battitura, è possibile utilizzare un piccolo trucco. I due comandi:

```
let ciao = 5¶
ciao = 5¶
```

hanno lo stesso effetto. Inoltre, il comando PRINT può venire sostituito dal carattere ?.

```
print "ciao"¶
```

e

```
? "ciao"¶
```

sono equivalenti. Nel secondo caso si risparmia però tempo nell'operazione di inserimento.

E' possibile riassumere le istruzioni viste finora in una singola istruzione:

```
? "La somma di 5 e 7 è "5+7¶
```

Il gadget front

E' possibile incontrare qualche problema per inserire quest'ultima istruzione nella finestra BASIC in quanto può capitare che il cursore finisca dietro la finestra LIST. Utilizzando i gadget front e back della finestra BASIC è comunque possibile ovviare a tale inconveniente (i gadget front e back sono rappresentati dai piccoli rettangoli sovrapposti collocati nell'angolo in alto a destra). Non appena si effettua un click sul gadget front della finestra BASIC, essa diventa visibile a tutto schermo, fornendo così lo spazio sufficiente ad inserire l'intera istruzione precedente.

E' possibile "ripulire" abbastanza facilmente la finestra Basic utilizzando il comando:

```
cls¶
```

CLS

Il comando CLS (CLear Screen) ripulisce lo schermo posizionando il cursore nell'angolo in alto a sinistra. Tale comando è molto utile nel caso in cui si intendano inserire molte istruzioni e si vuole avere più spazio possibile sullo schermo.

Se si vuole ripulire lo schermo subito prima di visualizzare qualcosa, si può utilizzare la seguente istruzione:

```
cls:? "ciao"¶
```

Le istruzioni multiple

Quest'ultima istruzione contiene qualcosa di nuovo: è possibile inserire più di un'istruzione sulla stessa linea separando le singole istruzioni con il carattere : (due punti).

E' comunque possibile fare qualcosa di ancor più interessante. Come primo assaggio di ciò che si può fare verrà presentato un programma che consente di visualizzare delle sfere che si spostano sullo schermo.

Il primo passo da eseguire è quello di fornire un nome al programma:

```
? "Sfere  volanti"¶
```

In questo modo viene visualizzato il titolo del programma. Si rivela a questo punto molto utile la finestra LIST che consente di non dover sempre riscrivere una linea dopo la sua esecuzione.

1.4 Il primo programma: volare è bello

La finestra LIST è parzialmente nascosta dalla finestra BASIC. Effettuando un click sul gadget back della finestra BASIC (posto leggermente più a sinistra nell'angolo in alto a destra), si riporta in primo piano la finestra LIST ponendo in secondo piano la finestra BASIC. E' ora possibile vedere due finestre contemporaneamente: la finestra LIST e la finestra associata al dischetto Extras; non essendo quest'ultima di alcuna utilità (per il momento), è consigliabile chiuderla effettuando un click sul suo gadget di chiusura.

L'inserimento di programmi
Risulta ora possibile cominciare a scrivere il primo programma BASIC.

- Attivare la finestra LIST effettuando un click al suo interno affinchè il cursore BASIC si sposti dalla finestra BASIC alla finestra LIST.

- Battere la seguente linea:

```
? "Sfere  volanti"¶
```

Amiga replica in modo diverso rispetto a quanto precedentemente visto: invece di eseguire l'istruzione fornita, modifica il carattere ? nella scritta PRINT posizionando il cursore all'inizio della linea seguente.

- Battere:

```
? "di John Doe"¶
```

Nuovamente Amiga cambia il carattere ? nella scritta PRINT e posiziona il cursore sulla linea succesiva. La finestra LIST si differenzia dalla finestra BASIC anche in altri aspetti. Ad esempio, premendo una delle frecce cursore (poste sulla tastiera di fianco al tasto <Return>) il cursore si sposterà nella direzione indicata dalla freccia; cercando di spostare il cursore in una direzione nella quale non è possibile effettuare alcuno spostamento, si otterrà un messaggio sonoro di avvertimento. Utilizzando i tasti cursore ed il

tasto <BackSpace> è quindi possibile muoversi liberamente all'interno della finestra LIST modificando e correggendo qualunque parte del testo in essa contenuto. Per questo motivo la finestra LIST viene considerata simile ad uno *screen editor*, a differenza della finestra BASIC che viene invece considerata come un *line editor* in quanto le correzioni possono essere effettuate solo sulla linea corrente (si veda l'Appendice E per la definizione precisa di screen e line editor).

NOTA:
Si osservi la prima linea del programma presentato in seguito. Se il dischetto contenente AmigaBASIC utilizzato ha un nome diverso da Extras (ad esempio, Copy of Extras), tale programma non funzionerà. Risulta allora necessario rinominare il dischetto come Extras, utilizzando l'opzione Rename del menù a tendina Workbench.

• Battere le seguenti linee nella finestra LIST:

```
p$ = "Extras:BasicDemos/Ball"¶
open p$ for input as 1 ¶
object.shape 1,nmput$(lof(1),1)¶
close 1 ¶
¶
Inizio: ¶
for x=2 to 5 ¶
object.shape x,1 ¶
object.x x,320 ¶
object.y x,60 ¶
object.hit x,0,0¶
object.ax x, ((x=2 or x=4)+.5)*6¶
object.ay x, ((x>3+.5)*1.5)¶
next x ¶
¶
for x=2 to 5 ¶
object.on x : object.start x¶
next x ¶
¶
for x=1 to 3000 : next x ¶
```

Ecco il primo programma in AmigaBASIC. Si consiglia di eseguire un attento controllo per assicurarsi di aver inserito le istruzioni correttamente. Come si puo' notare come, dopo ogni linea, Amiga converte le parole dei comandi BASIC in caratteri maiuscoli, lasciando invece le parole e le lettere che non rappresentano comandi AmigaBASIC in caratteri minuscoli. Nel caso si dovesse rilevare un errore, è possibile correggerlo utilizzando i tasti cursore ed il tasto <BackSpace>.

• Effettuare un click all'interno della finestra BASIC e battere:

```
run¶
```

Esecuzione del programma
Se il programma è stato inserito correttamente, compariranno sullo schermo quattro piccole sfere che si metteranno in movimento partendo dal centro dello schermo. Nel caso vi sia invece un errore Amiga emette un lieve suono visualizzando un requester nella parte superiore dello schermo: muovere allora il puntatore del mouse sul gadget OK ed effettuare un click. Esaminare poi attentamente il programma appena inserito confrontandolo con quello presente sul libro.

E' molto probabile che la maggior parte delle istruzioni appena inserite siano al momento attuale incomprensibili; ciò non deve tuttavia rappresentare un problema in quanto il programma costituisce un esempio di ciò che si riesce ad ottenere con un minimo sforzo di programmazione. Inoltre i comandi e le funzioni utilizzate verranno ampiamente trattati nel capitolo 2.

Save as
Selezionare ora l'opzione Save as del menù a tendina Project. Per effettuare tale operazione è necessario premere il pulsante di menù (il pulsante destro) e muovere il puntatore del mouse sul titolo Project. Spostare por, tenendo schiacciato il pulsante destro, il puntatore del mouse sulla parola Save as e rilasciare il pulsante stesso. Compare, nell'angolo in alto a sinistra dello schermo, un requester al cui interno vi è un rettangolo etichettato Save program as:. Spostare il puntatore del mouse su tale rettangolo ed effettuare un click al suo interno. Battere poi:

```
Programma sfere volanti¶
```

Salvataggio dei programmi
Il requester scompare ed il drive si mette in funzione. Allo spegnimento della luce del drive, l'operazione di salvataggio su dischetto del programma contenuto nella finestra LIST è stata completata. Tale operazione riveste un'importanza fondamentale in quanto consente di salvare un programma BASIC che, altrimenti, verrebbe completamente perso allo spegnimento di Amiga. L'unico modo per memorizzare un programma per poterlo successivamente riutilizzare è quello di salvarlo su un dischetto.

• Effettuare un click sulla finestra BASIC e battere:

```
new¶
```

Entrambe le finestre LIST e BASIC vengono ripulite ripristinando quindi la situazione che si aveva all'inizio della sessione di lavoro con AmigaBASIC.

1.5 Le finestre BASIC e LIST

Le prove precedenti sono state effettuate in una modalità di lavoro particolare, detta *modo diretto*, in cui Amiga analizza ed esegue le varie istruzioni inserite nella finestra Basic immediatamente dopo aver premuto il tasto <Return>. Nella finestra LIST Amiga consente invece di memorizzare le diverse istruzioni ed il programma non viene eseguito fino a quando non viene fornito il comando RUN.

I numeri di linea

Se si è già avuta occasione di vedere il listato di un programma BASIC su altri computer, sarà stata probabilmente notata la presenza di numeri che precedono le varie istruzioni. In AmigaBASIC tali numeri di linea non vengono utilizzati in quanto essi non sono necessari ad Amiga. In altri computer è invece richiesto che ad ogni linea di un programma sia associato un numero identificativo per diversi motivi: una prima ragione è dovuta al fatto che la presenza dei numeri di linea viene utilizzata per distinguere fra il modo diretto ed il *modo programma* (un' istruzione preceduta dal numero di linea non viene immediatamente eseguita dopo aver premuto il tasto <Return>). Per distinguere fra i due modi, Amiga utilizza invece due diverse finestre. Ciò non significa però che sia vietato utilizzare i numeri di linea e successivamente si vedrà come fare per associare ad ogni istruzione un numero.

Il programma di video titolazione

Dopo aver visto un primo esempio di programma BASIC, è ora possibile passare all'impostazione di un programma più complesso. In particolare si cercherà di realizzare un programma per la titolazione video. E' consigliabile, a tale proposito, inserire tutte le linee presentate: esse, alla fine del capitolo, consentiranno di formare un unico programma. Effettuare un click all'interno della finestra LIST e battere:

```
? "Videotitolatrice"¶
? "di Hannes Ruegheimer"¶
```

Queste due linee rappresentano le prime linee del nuovo programma e la loro interpretazione non dovrebbe più costituire un problema. Esse producono infatti la visualizzazione del nome del programma e del suo autore.

RUN

Per ottenere l'esecuzione del programma è necessario utilizzare il comando RUN, che deve essere fornito in modo diretto e quindi all'interno della finestra BASIC. Effettuare perciò un click all'interno della finestra BASIC e battere:

```
run¶
```

LIST

Nel caso in cui tutto funzioni regolarmente, scompare la finestra LIST, mentre nella finestra BASIC viene visualizzato il testo che nelle precedenti istruzioni era compreso fra le virgolette, seguito dal messaggio Ok. I risultati generati da un programma BASIC, cioè l'output del programma stesso, vengono quindi visualizzati nella finestra BASIC. Vi sono diversi modi per riottenere il listato del programma precedentemente battuto, il più semplice dei quali è rappresentato dall'inserimento del comando:

```
list¶
```

che ha l'effetto di portare in primo piano la finestra LIST.

I menù a tendina del BASIC

In alternativa all'inserimento da tastiera di alcuni comandi è possibile utilizzare quattro menù a tendina disponibili in AmigaBASIC:

Project Edit Run Windows.

In particolare, il menù Project consente di memorizzare, recuperare e cancellare programmi BASIC, il menù Edit facilita le operazioni di modifica dei programmi, il menù Run permette di avviare od interrompere l'esecuzione di un programma ed il menù Windows viene utilizzato per la gestione delle finestre BASIC e LIST.

Per eseguire un programma è sufficiente selezionare l'opzione Start del menù a tendina Run che provoca la scomparsa dal video del listato del programma (viene chiusa la finestra List). Per poter visualizzare nuovamente il listato, è necessario selezionare l'opzione Show List dal menù Windows che, come indicato dal nome stesso, apre nuovamente sullo schermo la finestra LIST.

Scorciatoie da tastiera

E' possibile notare, alla destra delle opzioni Start e Show List, una lettera A su sfondo blu seguita dalle lettere R ed L. Tali sequenze di lettere rappresentano metodi alternativi per selezionare le corrispondenti opzioni.

Sulla destra della tastiera, fra la barra spaziatrice ed il tasto <Alt> vi è un tasto siglato con una lettera A vuota. Questo tasto, che rappresenta uno dei due tasti Amiga, nel resto del libro verrà chiamato tasto <Amiga destro>; il corrispondente tasto Amiga posto sulla sinistra della tastiera (siglato con una lettera A piena o con il simbolo C=) verrà invece chiamato tasto <Amiga sinistro>. E' possibile richiamare le opzioni Start o Show List premendo il tasto <Amiga destro> e, contemporaneamente, il tasto L o R. Tali scorciatoie da tastiera potranno quindi essere utilizzate per richiamare alcune opzioni dei vari menù a tendina.

Menù	Combinazione di tasti
Menu' Edit:	
Cut	<Amiga destro> <X>
Copy	<Amiga destro> <C>
Paste	<Amiga destro> <P>
Menu' Run:	
Start	<Amiga destro> <R>
Stop	<Amiga destro> <.>
Step	<Amiga destro> <T>
Menu' Windows:	
ShowList	<Amiga destro> <L>

1.6 I comandi e le funzioni BASIC

Dopo aver esaminato le finestre utilizzate da AmigaBASIC ed i metodi per eseguire programmi, si può ora procedere a scrivere il programma di titolazione elettronica.

• Battere le seguenti linee nella finestra LIST:

```
? "Scegli:"¶
? "1 Immissione Testo"¶
? "2 Lettura dati Oggetto"¶
? "3 Movimento Oggetto"¶
? "4 Definizione Colore"¶
? "5 Visualizzazione Titolo"¶
```

Queste sei linee di programma, che dovrebbero risultare di facile comprensione, dimostrano che il BASIC non è così difficile come si può pensare a prima vista.

Come si può vedere il comando PRINT è un comando di utilizzo assai frequente. In questo caso è stato ad esempio utilizzato per creare le voci di un menù.

L'esecuzione del programma

Cosa si ottiene eseguendo il programma appena inserito? Si provi a lanciare l'esecuzione, utilizzando il comando RUN, per vederne i risultati. Un ulteriore consiglio prima di eseguire il programma: lasciando la finestra LIST attiva (senza cioè effettuare un click all'interno della finestra BASIC) e selezionando Start dal menù Run (o la combinazione di tasti <Amiga destro> <R>), si otterrà, alla fine dell'esecuzione del programma, la visualizzazione del listato del programma stesso. Per non ottenere questo, è sufficiente effettuare un click all'interno della finestra BASIC attivandola prima di lanciare l'esecuzione del programma.

Inserire linee

Eseguendo il programma ci si rende conto di come il suo output risulti abbastanza confuso, e non di immediata comprensione. Per migliorare un poco le cose, si può effettuare la procedura che segue:

- Riattivare la finestra LIST, posizionare il cursore alla fine della linea contenente il nome dell'autore del programma e premere il tasto <Return> inserendo una linea vuota. Battere ora:

```
?
```

Eseguendo nuovamente il programma, si può notare una linea vuota inserita fra il nome dell'autore e la scritta Scegli:. Il comando PRINT senza argomenti produce quindi la visualizzazione di una linea vuota.

Posizionando il cursore BASIC nella finestra LIST alla fine della linea contenente il titolo del programma ed inserendo il carattere ; (punto e virgola) è possibile ottenere l'effetto opposto:

```
PRINT "Videotitolatrice";¶
PRINT "di Hannes Ruegheimer"¶
```

Eseguendo il programma si può notare come il nome dell'autore venga ora visualizzato sulla stessa linea del titolo del programma. Il carattere ; alla fine di un'istruzione PRINT fa in modo che quanto visualizzato dalla successiva istruzione PRINT sia posizionato immediatamente al termine di quella corrente. L'unico problema che sorge a questo punto è la mancanza di uno spazio fra le parole "videotitolatrice" e "di"; tale spazio può essere inserito subito prima delle virgolette finali nella prima istruzione PRINT, posizionando il cursore in corrispondenza delle virgolette e premendo la barra spaziatrice.

Sullo schermo compaiono quindi le opzioni di menù disponibili. E' ora necessario istruire Amiga a riconoscere la selezione effettuata. Battere nella finestra LIST:

```
?¶
? "Immettere un numero:";¶
input a¶
```

INPUT
Si esegua ora il programma per vedere il funzionamento del comando INPUT finora inutilizzato. Si può notare che il cursore viene posizionato subito dopo la scritta "Immettere un numero:?" al termine della quale compare un punto interrogativo assente all'interno delle istruzioni inserite precedentemente. La presenza del punto interrogativo indica all'u ente che Amiga è in attesa dell'inserimento di un numero, numero che deve essere fornito seguito dal tasto <Return>. Se in seguito a questa operazione si ottiene un lampeggio dello schermo, è necessario effettuare un click all'interno della finestra BASIC per attivarla. Ogni volta che viene richiesto input, la finestra BASIC deve essere attiva. Il numero inserito viene memorizzato nella variabile di nome a.

Quando il programma termina ricompare nella finestra BASIC il messaggio Ok.

Il comando INPUT consente di inserire dati durante l'esecuzione di un programma offrendo l'opportunità di "dialogare" con Amiga.

• Inserire:

```
? a¶
```

Amiga visualizza il numero precedentemente fornito che era stato memorizzato nella variabile a.

Guru meditation
Durante l'esecuzione di un programma si verificano spesso malfunzionamenti (*crash*) del computer che possono essere dovuti, ad esempio, ad un errore nel codice del programma, ad un input scorretto o ad una caduta di tensione. Quando avviene una situazione di questo tipo (crash) Amiga visualizza, nella parte superiore dello schermo, un rettangolo nero bordato di rosso contenente il messaggio d'errore *Guru meditation* che avverte dell'immediata sospensione di ogni attività del computer.

Se alla richiesta di input del programma precedente si fornisce un numero superiore a 5 si ottiene un errore meno grave di una Guru meditation (che identifica invece un errore "irreparabile"). Se invece viene fornito un numero compreso tra 1 e 5 non si ottiene alcun errore. Nella stesura di un programma è importante tenere conto di tutti gli input che l'utente può

fornire e realizzare quindi una opportuna procedura per il controllo della loro correttezza. Per effettuare un opportuno controllo sull'input fornito è necessario aggiungere al programma (all'interno della finestra LIST) l'istruzione:

```
IF a<1 OR a>5 THEN RUN¶
```

IF ... THEN
Il comando BASIC IF ... THEN consente di controllare il flusso di esecuzione di un programma. Inserendo, tra le parole IF e THEN, una condizione, il computer controlla se la condizione sia o meno verificata ed in base a questo decide quale azione intraprendere. Si ha in tal modo:

IF (condizione verificata) THEN (effettua alcune operazioni)

OR
Nell'esempio descritto in precedenza la condizione era (a<1 OR a>5). La parola OR specifica che, affinchè sia intrapresa l'operazione seguente, è sufficiente che una sola delle due condizioni sia vera. Non esistono, del resto, numeri reali contemporaneamente inferiori ad 1 e superiori a 5. Il comando RUN posto subito dopo la parola THEN permette di eseguire nuovamente il programma nel caso in cui la condizione sia verificata. Il comando RUN è già stato descritto in precedenza ed utilizzato all'interno della finestra BASIC. Ora lo si è invece utilizzato ed inserito anche all'interno del listato di un programma.

Effettuata questa correzione al programma è ora possibile inserire qualunque numero, anche superiore a 5; Amiga continuerà a richiedere l'input fino a quando non viene fornito un numero corretto.

BEEP
Si potrebbe pensare di generare un piccolo suono nel caso in cui venga fornito un numero errato, in modo da avvertire della non correttezza dell'input. Per far questo è necessario modificare la linea di condizione inserendo il comando BEEP seguito dal carattere : (due punti) ottenendo la linea seguente:

```
IF a<1 OR a>5 THEN BEEP : RUN¶
```

In caso di errore, il computer genererà ora, oltre ad un lampeggio dello schermo, un messaggio sonoro (il carattere due punti consente, come già accennato, di separare più comandi posti sulla stessa linea).

GOTO

In un programma BASIC vi è la possibilità di trasferire il flusso di esecuzione in un particolare punto del programma stesso mediante il comando GOTO che consente di saltare, durante l'esecuzione del programma, ad una precisa linea di codice. Per effettuare questa operazione è comunque necessario avere a disposizione un metodo per indicare al comando GOTO dove saltare. Nei tradizionali linguaggi BASIC viene semplicemente indicato un numero di linea. In AmigaBASIC, non utilizzando numeri di linea, si ricorre ad alcune parole, dette *etichette*, associate alle linee a cui si desidera saltare.

- Visualizzare ora la finestra LIST e, dopo aver spostato il cursore all'inizio della linea contenente il comando INPUT, creare una nuova linea, premendo il tasto <Return>, e battere:

```
richiesta:¶
```

Il carattere due punti, posto dopo la parola "Richiesta", è fondamentale in quanto consente di identificare la parola stessa come etichetta. Si potrebbero usare altre parole, come ad esempio ciao:, la.de.da:, hum.de.dum: o inizio.programma:, ma è consigliabile utilizzare un termine che fornisca un'idea precisa di ciò che l'istruzione etichettata realizza. E' quindi consigliabile utilizzare sempre etichette che rievochino informazioni sulla parte di programma cui vengono associate.

- Modificare ora la parola RUN dell'ultima linea del programma in:

```
GOTO richiesta¶
```

Per effettuare tale sostituzione è necessario posizionare il cursore dopo la parola RUN, premere per tre volte il tasto <BackSpace> ed inserire il nuovo testo. E' possibile oltrepassare il bordo destro della finestra LIST durante l'inserimento del testo: in questo caso è necessario provvedere ad allargare la finestra stessa utilizzando l'apposito gadget di dimensionamento.

Alcuni considerazioni sui nomi delle variabili e delle etichette
I nomi delle variabili e delle etichette possono contenere numeri e lettere, ma non è possibile utilizzare al loro interno spazi vuoti; dovrebbe essere inoltre evitato l'uso dei simboli di punteggiatura, che possono risultare ambigui e generare errori. La lunghezza massima di un'etichetta è fissata in 40 caratteri; Amiga ignorerà automaticamente ulteriori caratteri. Le limitazioni viste valgono anche nel caso delle variabili. Per quanto riguarda il tipo di carattere da utilizzare per variabili ed etichette, è indifferente utilizzare maiuscole o minuscole in quanto cIAO, ciao e Ciao sono riconosciuti allo stesso modo.

• Inserire la linea:

```
GOTO Richiesta¶
```

In tale linea è stata posta una lettera maiuscola come iniziale dell'etichetta utilizzata. Premere ora il tasto <Return> ed osservare attentamente cosa si verifica nelle due linee successive in cui la parola "richiesta" compare scritta in caratteri minuscoli. AmigaBASIC sostituisce automaticamente tutte le occorrenze di "richiesta" con "Richiesta".

E' anche possibile inserire etichette per intestare sezioni diverse del programma al fine di strutturarlo meglio. Si potrebbe, ad esempio, marcare l'inizio del programma con l'etichetta Partenza.

Il programma dovrebbe ora apparire come segue:

```
Partenza:¶
PRINT "Videotitolatrice";¶
PRINT "di Hannes Ruegheimer"¶
PRINT¶
¶
? "Scegli:"¶
? "1 Immissione Testo"¶
? "2 Lettura dati Oggetto"¶
? "3 Movimento Oggetto"¶
? "4 Definizione Colore"¶
? "5 Visualizzazione Titolo"¶
PRINT¶
PRINT  "Immettere un numero:";¶
```

```
Richiesta:¶
INPUT a¶
IF a<1 OR a>5 THEN BEEP : GOTO Richiesta¶
```

Errori dell'utente

E' necessario discutere ora brevemente come il programma si comporta in presenza di errori di input commessi dall'utente. Battendo una lettera invece di un numero in seguito alla richiesta di input del programma, si ottiene la visualizzazione del messaggio "?Redo from Start" che invita ad inserire nuovamente un input. Questo si verifica in quanto si sta cercando di assegnare parole o singole lettere ad una variabile che può memorizzare solo valori numerici.

Le variabili stringa

Per memorizzare caratteri o parole è necessario utilizzare un tipo particolare di variabile, la *variabile stringa*. Una variabile di questo tipo può essere vista come una sequenza di caratteri. Le variabili stringa sono identificate dalla presenza del carattere $ immediatamente dopo il loro nome: alla variabile Ciao$, è quindi possibile assegnare qualunque sequenza di caratteri. Per effettuare un assegnamento ad una variabile stringa all'interno di un programma è necessario includere tra virgolette la sequenza di caratteri da assegnare. Modificando le ultime linee della sezione identificata dall'etichetta Richiesta, anche l'eventuale input di un carattere da parte dell'utente viene isolato e non costituisce causa d'errore. Nel programma dovrebbero ora comparire le linee seguenti:

```
INPUT a$¶
IF a$<"1" OR a$>"5" THEN BEEP : GOTO Richiesta¶
```

Prima di concludere questo capitolo è importante ribadire nuovamente quanto sia importante gestire all'interno del programma gli eventuali errori commessi dall'utente

1.7 Inserire testo nel programma di videotitolazione

Emerge a questo punto un potenziale errore: se l'utente continua ad inserire una risposta errata ben presto lo schermo si riempirà di messaggi d'errore provocando la scomparsa della richiesta iniziale. Per porre rimedio a questo eventuale problema è necessario migliorare la sezione di programma dedicata alla gestione del menù.

A proposito della finestra LIST si può rilevare come fino ad ora sono stati utilizzati i tasti cursore (le frecce rivolte nelle quattro direzioni) per spostarsi all'interno di tale finestra. E' però possibile notare che, quando si attiva la finestra LIST, il cursore BASIC compare nel punto in cui è stato effettuato il click con il mouse. Il cursore può essere spostato anche utilizzando il mouse, consentendo in tal modo spostamenti più veloci rispetto all'uso dei tasti cursore.

Evidenziazione
Premendo il pulsante sinistro del mouse e, tenendolo schiacciato, muovendo il mouse stesso si ottiene un effetto particolare: compare un rettangolo arancione, detto *evidenziatore* , che si estende dalla linea in cui si trova il cursore BASIC fino alla posizione del puntatore del mouse. Per provare meglio questa nuova funzione dello screen editor, inserire del testo in una qualunque posizione all'interno del programma, spostare successivamente il puntatore del mouse all'inizio del testo inserito e, premendo e tenendo schiacciato il pulsante sinistro del mouse, spostare il puntatore alla fine di tale testo facendo in modo che venga evidenziata solo la linea di programma modificata. Quando il testo è completamente evidenziato, rilasciare il pulsante sinistro del mouse.

Come usare l'evidenziatore
Se si prova ad inserire del testo, si verifica la scomparsa del testo evidenziato che viene sostituito con il testo che si sta inserendo.

• Evidenziare ora il testo appena inserito e premere il tasto <BackSpace>.

Questo provoca la cancellazione automatica dell'intero blocco evidenziato. Quest'ultima operazione rappresenta un modo veloce per cancellare intere sezioni di programma. E' comunque necessario utilizzare tale funzione con cautela in quanto è molto facile includere incidentalmente nel blocco marcato linee che non si vorrebbero cancellare e che poi, dato che il testo cancellato non è più recuperabile, dovrebbero essere nuovamente riscritte.

Cut e Paste

Oltre alla cancellazione sono possibili numerose altre operazioni che possono essere effettuate utilizzando la funzione di evidenziazione. Una di queste è l'opzione Cut and Paste (taglia e incolla).

• E' necessario innanzitutto evidenziare una parte qualsiasi del testo presente nella finestra LIST, e selezionare poi l'opzione Cut del menù a tendina Edit.

In alcuni casi, se si possiede un solo drive, potrebbe essere richiesto l'inserimento del dischetto Workbench in quanto AmigaBASIC necessita di un particolare dispositivo (Clipboard device) contenuto in tale dischetto.

• Al termine del caricamento dal dischetto Workbench reinserire il dischetto Extras.

Il dispositivo di cui AmigaBASIC necessitava è ora presente in memoria. Dopo che Amiga ha eseguito tutte le operazioni associate all'opzione Cut, è possibile rilevare come la parte di testo evidenziata sia stata rimossa dal programma. A differenza di quanto compiuto nell'operazione precedente di cancellazione, il testo rimosso è stato ora memorizzato in una particolare porzione di memoria detta *clipboard* (blocco appunti).

• Senza modificare la posizione del cursore BASIC, selezionare ora l'opzione Paste del menù a tendina Edit.

Questo provoca la ricomparsa del testo precedentemente rimosso. E' possibile quindi recuperare il testo memorizzato nella clipboard ed inserirlo in qualunque posizione all'interno del testo presente nella finestra LIST.

Copy
L'opzione Copy del menù a tendina Edit agisce in modo analogo all'opzione precedente. Il testo evidenziato non viene però rimosso dal programma, ma solamente memorizzato nella clipboard da cui può essere poi recuperato per effettuarne duplicazioni.

- Utilizzando eventualmente anche le opzioni appena descritte, battere nella finestra LIST il programma in modo tale che assuma questa forma:

```
Partenza:¶
PRINT "Videotitolatrice";¶
PRINT "di Hannes Rugheimer"¶
¶
PRINT "Scegli:"¶
PRINT "1 Immissione Testo"¶
PRINT "2 Lettura Dati Oggetto"¶
PRINT "3 Movimento Oggetto"¶
PRINT "4 Definizione Colore"¶
PRINT "5 Visualizzazione Titolo"¶
PRINT¶
¶
Richiesta:¶
LOCATE 11,1¶
PRINT "Immettere un numero:";¶
INPUT a$¶
IF a$<"1" OR a$>"5" THEN BEEP : GOTO Richiesta¶
IF a$="1" THEN ImmettiTesto¶
PRINT "L'opzione" a$ "non esiste ancora."¶
GOTO Richiesta¶
```

Poichè il programma è ancora in fase di realizzazione, si è pensato di inserire alcune linee che informino l'utente su quali funzioni non sono ancora state completate. Ad eccezione della porzione di programma (routine) ImmettiTesto (associata alla scelta 1), che verrà definita in questo paragrafo, le altre scelte non sono ancora implementate. La selezione di un numero fra 2 e 5 porterà quindi alla visualizzazione del messaggio "L'opzione ... non esiste ancora" in cui è stata inserita la visualizzazione del contenuto della variabile a$. In questo esempio è quindi possibile notare come testo e variabili possano essere incluse all'interno dello stesso comando PRINT.

LOCATE

L'istruzione LOCATE 11,1 non è stata ancora discussa. La visualizzazione di un messaggio attraverso il comando PRINT viene generalmente effettuata sulla linea successiva a quella corrente. LOCATE consente invece di specificare un punto preciso dello schermo da cui partire nella visualizzazione del messaggio. In particolare, nel programma che si sta realizzando, per fare in modo che il messaggio "Immettere un numero:" compaia sempre nella stessa posizione dello schermo, è stata specificata la prima colonna della undicesima riga.

- Provare a battere un comando LOCATE in modo diretto, ossia all'interno della finestra BASIC, ad esempio:

```
LOCATE 5,20 : ? "ciao"6
```

Tornando ora al programma di video titolazione, è possibile osservare ciò che accade in seguito alle ultime modifiche. L'output del programma risulta ora molto più chiaro ed ordinato. A questo punto solamente la selezione 1 causa un errore, in quanto l'etichetta ImmettiTesto non è ancora stata definita. AmigaBASIC informa l'utente relativamente al punto in cui si è verificato un errore visualizzando la finestra LIST ed evidenziando la linea errata. Si ha inoltre la comparsa di un messaggio d'errore nella parte superiore dello schermo. Per proseguire è a questo punto necessario effettuare un click sul gadget OK visualizzato insieme al messaggio d'errore.

Per correggere l'errore che si verifica selezionando l'opzione 1 vi è un solo modo: aggiungere al programma la porzione mancante associata all'etichetta ImmettiTesto.

NOTA:

Nei successivi listati compaiono linee in fondo alle quali è presente il simbolo ¶ ed altre in cui tale simbolo è invece assente. A causa dell'impaginazione effettuata, è stato infatti necessario spezzare su due righe alcune linee di programma. Una riga che non contiene il simbolo ¶ deve essere inserita senza battere il tasto <Return> che deve essere premuto solo in corrispondenza del simbolo ¶.

```
ImmettiTesto:¶
CLS : INPUT "Quante righe di testo (1-15)": NumLinee$¶
IF NumLinee$="" THEN CLS : GOTO Partenza¶
NumLinee=VAL(NumLinee$)¶
IF NumLinee<1 OR NumLinee>15 THEN BEEP :
GOTO ImmettiTesto¶
DIM Testo$(NumLinee)¶
FOR x=1 TO NumLinee¶
LINE INPUT "Testo:";Testo$(x)¶
NEXT x : CLS : GOTO Partenza¶
```

In queste linee di programma compaiono alcuni nuovi comandi. Prima di passare alla loro analisi, un suggerimento: non è necessario battere tutte le linee che seguono; avendo esse solo scopo dimostrativo dei vari comandi a disposizione, non faranno parte del programma di titolazione.

VAL

E' gia stata rilevata la differenza fra una variabile numerica (a) ed una variabile stringa (a$). In un programma si ha, talvolta, la necessità di convertire il tipo di una variabile da stringa in numerica o viceversa. Il comando VAL consente di trasformare le variabili stringa in variabili numeriche. Esso viene utilizzato nella forma:

```
z = VAL (z$)¶
```

DIM

Il comando DIM consente ad introdurre un terzo tipo di variabile, il *vettore o matrice* (array), utilizzato per memorizzare con un unico nome più dati. Prima di utilizzare una matrice è necessario indicare il numero di elementi in esso contenuti, ossia la *dimensione* della matrice stessa. Il comando DIM consente di dimensionare una matrice.

Vettori e matrici

Per chiarire meglio le idee è utile soffermarsi su un esempio di matrice la cui dimensione è fissata attraverso l'istruzione DIM t$(10). Ciò significa che la matrice t$ può contenere fino ad 11 elementi a cui ci si può riferire con

t$(0), t$(1), t$(2) e così via fino a t$(10). Ogni singolo elemento è una variabile stringa e può quindi contenere una stringa di caratteri. E' importante comunque rilevare come non vi sia alcun legame tra la matrice t$(z) e la variabile stringa t$.

Le matrici multidimensionali

Una matrice può anche essere definita a più dimensioni. L'istruzione DIM a$(2,2), ad esempio, definisce una matrice contenente i seguenti nove elementi:

```
a$(0,0)  a$(0,1)  a$(0,2)
a$(1,0)  a$(1,1)  a$(1,2)
a$(2,0)  a$(2,1)  a$(2,2)
```

Anche in questo caso ogni elemento può contenere una sequenza di caratteri definita indipendentemente da un'altra. E' possibile definire anche matrici a più di due dimensioni; Amiga consente di definire matrici fino a 255 dimensioni.

FOR ... NEXT

Questo comando consente di incrementare il contenuto di una variabile a partire da un determinato valore iniziale fino ad un particolare valore finale. Il seguente esempio consente di visualizzare tutti i numeri compresi fra 1 e 50:

```
FOR x=1 TO 50¶
? x¶
NEXT x¶
```

Alla variabile x viene inizialmente assegnato il valore 1; l'istruzione seguente visualizza il valore di x; il comando NEXT trasferisce il controllo al comando FOR incrementando il valore di x (che quindi vale ora 2). Tale sequenza di operazioni viene ripetuta finchè x assume il valore 50; a questo punto l'esecuzione (invece di tornare al comando FOR) prosegue con l'eventuale linea successiva al comando NEXT.

LINE INPUT
Il comando INPUT a$ possiede numerose restrizioni (non è possibile, ad esempio, utilizzare virgole o punti interrogativi). Per superare queste limitazioni, è possibile ricorrere al comando LINE INPUT il quale consente di assegnare ad una variabile stringa qualunque sequenza di caratteri terminata dal tasto <Return>.

A questo punto tutte le istruzioni che compongono la sezione di programma etichettata ImmettiTesto sono state analizzate.

D'ora in poi non verranno più utilizzati per le variabili nomi insignificanti come z o t$ in quanto è meglio utilizzare nomi che forniscano un'idea generale della funzione svolta da una particolare variabile (ad esempio NumLinee o Testo$).

Funzionamento della parte di programma per l'inserimento di testo
Viene richiesto innanzitutto il numero di linee che si desidera inserire (da un minimo di 1 ad un massimo di 15) e viene dimensionata la matrice Testo$ con il valore fornito. In base a questo stesso numero viene poi eseguito il ciclo FOR ... NEXT per memorizzare nelle varie celle della matrice Testo$ le linee di testo. Viene infine ripulito lo schermo ed il programma ritorna al menù principale.

Questa sezione di codice consente di inserire il testo che verrà poi visualizzato davanti agli oggetti in movimento. Se si desidera realizzare una titolazione, si consiglia di inserire il titolo di un film, ad esempio Guerre stellari nella prima linea ed eventuali sottotitoli, ad esempio Parte 1, nella seconda.

1.8 Memorizzare un programma

Il programma per la titolazione elettronica comincia ad essere abbastanza lungo. Per il momento si tratta però di un programma temporaneo: se accidentalmente si spegne il computer, il programma viene perso per sempre e si deve rifare tutto il lavoro. E' quindi consigliabile, durante la stesura del codice, effettuare frequenti operazioni di salvataggio del programma su dischetto.

Save e Save As

AmigaBASIC dispone delle due opzioni Save e Save As, contenute nel menù a tendina Project, per memorizzare su dischetto il programma contenuto in memoria. Perchè vi sono due comandi che effettuano la stessa operazione?

- Selezionare l'opzione Save As (se si possiede un solo drive ed in precedenza si è estratto il dischetto Extras, Amiga ne richiede l'inserimento).

Compare, nell'angolo superiore dello schermo, un *requester* (finestra di dialogo) con il messaggio Save program as: che invita ad inserire il nome con cui si intende memorizzare il programma.

- Spostare il puntatore all'interno del rettangolo vuoto posto sotto al messaggio ed effettuare un click provocando la comparsa di un cursore.

Ai programmi memorizzati su dischetto è necessario attribuire un nome che Amiga utilizza come riferimento. Nel caso presente si potrebbe fornire al programma il nome VideoTitolatrice; questo nome deve essere inserito da tastiera all'interno del rettangolo contenente il cursore. Se, durante la scrittura, si commettono errori, è possibile correggerli utilizzando i tasti <BackSpace> e <Del> per cancellare rispettivamente il carattere alla sinistra del cursore ed il carattere su cui si trova il cursore. Dopo aver inserito il nome corretto è necessario premere il tasto <Return> od effettuare un

click sul gadget OK per far scomparire dallo schermo il requester ed avviare l'operazione di memorizzazione su dischetto del programma.

L'opzione Save viene invece utilizzata quando il programma da memorizzare possiede già un nome, nel qual caso è sufficiente selezionare l'opzione stessa senza dover specificare nuovamente il nome da utilizzare. Se il programma possiede già un nome si ricorre quindi all'utilizzo di Save As soltanto nel caso in cui se ne voglia modificare il nome, operazione che non porta alla cancellazione del programma con il vecchio nome.

AmigaBASIC fornisce comunque un metodo alternativo per memorizzare un programma, attraverso il comando SAVE fornito direttamente nella finestra BASIC.

```
SAVE¶
```

produce lo stesso effetto della corrispondente opzione del menù Project. L'istruzione:

```
SAVE "nome"¶
```

in cui all'interno delle virgolette viene inserito il nome con cui memorizzare il programma, è l'analogo dell'opzione Save As del menù Project.

A questo punto il programma scritto è stato salvato su dischetto e quindi, anche in caso di spegnimento accidentale del computer, risulta ora possibile recuperarlo.

1.9 Cancellare programmi BASIC

Dopo aver memorizzato su dischetto il programma, è ora possibile cancellarlo dalla memoria interna di Amiga senza problemi. Il comando per effettuare tale operazione è il comando NEW che può essere selezionato dal menù a tendina Project od inserito direttamente nella finestra BASIC. Amiga ripulisce le finestre BASIC e LIST posizionando il cursore nella finestra attiva. Il programma cancellato non è più presente nella memoria interna del computer (il termine *memoria interna* si riferisce ai chip contenuti all'interno di Amiga, a differenza del termine *memoria di massa* usato invece per indicare dischetti, dischi fissi, nastri, ecc.).

NEW
AmigaBASIC è ora pronto a ricevere nuovi comandi. Ma cosa accade selezionando NEW dimenticandosi di salvare il lavoro precedente? Per fortuna AmigaBASIC effettua un particolare controllo.

• Scrivere un breve programma nella finestra LIST costituito dalle linee:

```
FOR x=1 TO 100¶
PRINT x¶
NEXT x¶
```

Le spiegazioni fornite nell'ultimo paragrafo dovrebbero consentire di comprendere completamente queste istruzioni: il programma visualizza i numeri da 1 a 100 (per verificarlo eseguire il programma con il comando RUN).

• Cancellare ora il programma con il comando NEW.

Prima di cancellare la propria memoria, Amiga visualizza un requester con un messaggio che indica che il programma non è ancora stato salvato e chiede se si desidera effettuare un salvataggio prima di proseguire. Vi sono tre gadget da utilizzare per la risposta che può essere fornita effettuando

un click su uno di essi:

YES porta alla comparsa di un requester in cui viene chiesto il nome con cui memorizzare il programma

NO prosegue nell'esecuzione del comando NEW

CANCEL annulla il comando NEW ripristinando la situazione precedente alla sua selezione

Tale controllo, che si applica sia ai nuovi programmi che a programmi già esistenti, viene effettuato ogni volta che sono state apportate modifiche al programma e non è stata effettuata una registrazione. Nel caso del programma precedente è possibile selezionare NO in quanto il programma, scritto solo a scopo dimostrativo, può tranquillamente essere cancellato.

Uscire dal BASIC

Il comando NEW non rappresenta l'unico metodo per cancellare un programma BASIC. Un altro metodo possibile è quello di uscire completamente dal BASIC. Vi sono diversi modi per effettuare tale operazione: inserendo il comando SYSTEM in modo diretto (nella finestra BASIC); selezionando l'opzione Quit del menù a tendina Project; chiudendo le finestre BASIC e LIST (effettuando un click sui loro gadget di chiusura).

In tutti i casi, se non si è proceduto ad una memorizzazione delle modifiche apportate al programma, prima della visualizzazione dello schermo Workbench, verrà visualizzato il requester di controllo precedentemente descritto.

• Uscire da AmigaBASIC utilizzando uno dei tre metodi descritti.

Nota d'uso 1

Nel resto del libro si troveranno alcune note d'uso come questa all'interno dei vari paragrafi. Tali note d'uso presentano importanti informazioni che non possono venire classificate all'interno di nessuna sezione del libro.

In questa prima nota d'uso si creerà un cassetto dedicato a contenere i programmi BASIC realizzati durante la lettura del libro. Questo argomento non ha molto a che fare con il resto del capitolo, ma è necessario per il buon proseguimento del lavoro.

Lo schermo del Workbench sembra vuoto paragonato alla finestra BASIC su cui si stava lavorando in precedenza. L'unica cosa che compare è infatti la finestra associata al dischetto Extras.

- Chiudere anche questa finestra, effettuando un click sul suo gadget di chiusura, e riaprirla subito dopo.

Tale operazione, che sembra assurda, è necessaria per aggiornare il contenuto della finestra Extras, nella quale compaiono ora nove icone: le sette originali (Tools, FD1.2, ReadMe, AmigaBASIC, PCUtil, Trashcan e BasicDemos, già presenti in precedenza) e due nuove icone, associate al programma Sfere volanti ed al programma di video titolazione. Tali icone assomigliano a fogli di carta per stampante con dei simboli arancioni simili a quelli presenti sull' icona AmigaBASIC. Icone di questa forma identificano i programmi BASIC.

I cassetti

Molte di queste icone sono presenti anche nel cassetto BasicDemos (che verrà esaminato più avanti).

I cassetti vengono utilizzati per organizzare le informazioni memorizzate sui dischetti. Il dischetto Extras contiene due programmi ed una serie di

informazioni ben organizzate. E' quindi consigliabile creare un cassetto separato dedicato a contenere i programmi che si realizzeranno.

ATTENZIONE:
Come detto in precedenza, viene dato per scontato che si stia lavorando con una copia di lavoro del dischetto Extras dato che non si deve assolutamente memorizzare nessun programma sul dischetto Extras originale. Per informazioni sulle operazioni relative alla creazione di copie di lavoro consultare il manuale fornito insieme ad Amiga. Per proteggere da scritture accidentali il dischetto Extras originale, è necessario spostare lo *scorrevole di protezione* posto nell'angolo del dischetto in posizione tale da poter vedere attraverso la corrispondente fessura.

- Aprire il Workbench.

Come creare un cassetto
Se si possiede un solo drive, estrarre, dopo essersi assicurati di aver aperto la finestra Extras, il dischetto Extras ed inserire il dischetto Workbench; dopo aver aperto la finestra Workbench, si può notare al suo interno un cassetto di nome Empty creato appositamente per consentire di realizzare agevolmente nuovi cassetti.

- Effettuare un click sul cassetto Empty, spostarlo (tenendo schiacciato il pulsante sinistro del mouse e muovendo il mouse stesso) all'interno della finestra Extras e rilasciare il pulsante sinistro del mouse.

Se si possiede un solo drive, sarà necessario effettuare tre cambi di dischetti per completare l'operazione di copia. Amiga indicherà, di volta in volta, quale dischetto inserire. Come più volte detto, è necessario porre particolare attenzione alla luce del drive prima di estrarre un dischetto: attendere sempre lo spegnimento di tale luce prima di procedere all'estrazione del dischetto inserito!

- Al termine dell'operazione di copia, chiudere la finestra Workbench, attivare, nella finestra Extras, il nuovo cassetto e selezionare l'opzione Rename del menù a tendina Workbench.

Compare al centro dello schermo un requester che riporta il nome attua-

le del cassetto ed un cursore arancione. Per inserire il nuovo nome si deve innanzitutto cancellare il nome visualizzato (utilizzando il tasto <Del>) e successivamente battere il nuovo nome (ad esempio Programmi) terminato dal tasto <Return>.

Riempimento del cassetto
E' ora possibile spostare le icone del programma Sfere volanti e del programma di videotitolazione in questo nuovo cassetto.

- Sovrapporre l'icona del programma Sfere volanti all'icona del cassetto appena creato.

L'operazione eseguita costituisce il metodo più semplice per inserire icone in un cassetto, in un dischetto o nel cestino. Non appena il puntatore riassume la sua forma originale, è possibile ripetere l'operazione per il secondo programma.

Clean Up
Prima di passare al prossimo paragrafo, è consigliabile ordinare il cassetto utilizzando l'opzione Clean Up che consente di allineare le icone all'interno del cassetto stesso.

Per ordinare le icone all'interno di una finestra si può anche utilizzare l'opzione Snapshot del menù a tendina Special che permette di "ancorare" un'icona in una particolare posizione. Per fissare le posizioni precise in cui devono apparire all'apertura di una finestra le varie icone in essa contenute, è innanzitutto necessario attivare contemporaneamente le icone stesse. Per far questo si deve premere il tasto Shift (uno dei due tasti siglati con una freccia rivolta verso l'alto e collocati alle due estremità della tastiera) e, tenendolo schiacciato, effettuare un click sulle icone che si intendono attivare. Si deve infine selezionare l'opzione Snapshot che realizza una "istantanea" della finestra memorizzandola sul dischetto.

1.10 Principi di movimento: bob e sprite

Molto probabilmente si è già provato, per curiosità, a dare uno sguardo al contenuto del cassetto BasicDemos; esso contiene una collezione di interessanti programmi scritti in AmigaBASIC. Ci sono parecchie cose da vedere che verranno esaminate nei prossimi paragrafi. Non bisogna comunque preoccuparsi se non si comprende pienamente il funzionamento di tali programmi dimostrativi: essi saranno dettagliatamente spiegati in questo manuale.

ObjEdit
Per il momento riveste una certa importanza l'esame di un solo programma.

- Aprire il cassetto BasicDemos e cercare il programma ObjEdit. Dopo averlo rinvenuto, mandarlo in esecuzione effettuando un doppio-click sulla sua icona.

Il nome ObjEdit costituisce l'abbreviazione di Object Editor. Probabilmente è noto cosa consenta di fare un editor, ma cosa si intende per oggetti e quali operazioni si possono effettuare su di essi?

Il caricamento del programma ObjEdit richiede una certa quantità di tempo poichè, ogni volta che si esegue un programma scritto in BASIC, Amiga deve provvedere a caricare anche il BASIC stesso. Questa operazione automatica semplifica l'utilizzo dei programmi BASIC che possono venire eseguiti anche senza conoscere nulla relativamente all'AmigaBASIC.

Alla fine del caricamento Amiga esegue il programma ObjEdit, il quale inizialmente visualizza le linee seguenti:

```
Enter 1 if you want to edit sprites
Enter 0 if you want to edit bobs
```

Gli oggetti
Il programma si pone in attesa di una risposta. Amiga consente di avere *oggetti* che si possono muovere indipendentemente sullo schermo. Fino ad ora si è preso in considerazione un solo oggetto di questo tipo, il puntatore del mouse, che può essere spostato in qualunque direzione sullo schermo. Indipendentemente dal tipo di sfondo presente sullo schermo (la finestra Workbench, una finestra AmigaBASIC o qualunque altra finestra), è possibile posizionare il puntatore del mouse in un qualsiasi punto. La possibilità di disporre di oggetti in grado di muoversi indipendentemente dallo sfondo e' fondamentale nell'animazione computerizzata.

Le origini dell'animazione al computer
Cosa si intende per "indipendenza dallo sfondo"? I computer meno recenti non erano in grado di gestire gli oggetti in movimento. Per simulare il movimento di un oggetto era allora necessario disegnare l'oggetto in un punto, cancellarlo, disegnarlo in un secondo punto, cancellarlo nuovamente e così via ripetendo velocemente tali operazioni per dare l'illusione del movimento.

Tale metodo aveva tuttavia un grosso problema: lo sfondo, che avrebbe dovuto rimanere immutato, cambiava insieme al disegno poichè il punto in cui l'oggetto veniva cancellato diventava bianco. Se accadesse questo anche nel caso del movimento del puntatore del mouse, ogni cosa lungo il cammino del puntatore stesso scomparirebbe.

I primi esempi di animazione, molto semplici, trovarono la loro naturale applicazione nei video-giochi. Uno dei primi video-giochi a comparire sul mercato è stato il tennis: vi era una linea bianca centrale (che separava le due parti del campo), un piccolo quadrato bianco (la pallina) e due rettangoli (le racchette dei due giocatori) mossi mediante due manopole. Quando la pallina entrava nella propria metà campo si doveva cercare di colpirla con la racchetta per rispedirla nella metà campo avversaria. Il gioco ebbe un discreto successo ed il suo inventore, Nolan Bushnell, ottenne in seguito un ulteriore successo fondando la società Atari.

La necessità della velocità
Il gioco del tennis non possedeva uno sfondo ben curato poichè nessun computer era allora sufficientemente veloce. Per un lungo periodo di tempo, la velocità rimase l'unica cosa che consentiva di effettuare delle

animazioni con un particolare sfondo. Invece di cancellare continuamente gli oggetti sullo schermo, si procedeva alla memorizzazione dello sfondo, che veniva ridisegnato ogni volta che l'oggetto veniva spostato, con tempi di elaborazione molto più elevati. Per far sì che l'animazione sembrasse reale, era necessario disporre di computer molto veloci. Ci si rese quindi conto che era indispensabile realizzare particolari chip dedicati alla gestione della grafica. Tali chip permisero quindi di muovere gli oggetti sullo schermo senza doversi preoccupare dello sfondo.

Nei vari computer agli oggetti grafici vengono associati nomi diversi. I nomi più utilizzati sono player missile graphics (nei modelli Atari 400 e 800) e sprites (nel Commodore 64). Anche Amiga dispone di questi oggetti grafici, divisi però in due grandi categorie: *sprite*, che funzionano in modo analogo a quelli del Commodore 64 e *bob* (blitter object blocks) di cui si discuterà più avanti.

Dopo questa discussione dovrebbe essere abbastanza chiaro cosa consente di fare il programma ObjEdit: definire gli oggetti grafici (bob e sprite). La prima cosa che il programma richiede è l'indicazione del tipo di oggetto che si intende realizzare.

1.11 E' nata una stella: l'object editor

E' possibile a questo punto aggiungere al programma di video titolazione le istruzioni per la creazione di oggetti in movimento. Come esempio si cercherà, utilizzando un bob, di creare una stella che si muova intorno al titolo definito nei precedenti paragrafi.

Battendo 0 ed il tasto <Return> compaiono sullo schermo una finestra, quattro rettangoli di diverso colore ed il testo "Bob size X: 31 Y: 31 Pen".

A questo punto, premendo e tenendo schiacciato il pulsante destro del mouse, compaiono nella linea superiore dello schermo i seguenti titoli di menù:

File Tools Enlarge

Posizionando il puntatore del mouse sul titolo di menù Tools si rendono visibili le seguenti opzioni:

Pen
Pen, lo strumento attivato all'inizio del programma ObjEdit (in ogni istante lo strumento attivo viene indicato alla fine della linea in cui sono riportate le dimensioni della finestra), consente di disegnare a mano libera, analogamente a quanto è possibile fare con programmi di disegno quali DeluxePaint o Graphicraft.

Line
Poichè è difficile disegnare una linea retta con l'opzione Pen, viene fornita la possibilità di tracciare una linea fissandone i punti iniziale e finale. Per far ciò è sufficiente effettuare un click nel punto iniziale della linea e, tenendo premuto il pulsante sinistro del mouse, spostare il puntatore al punto finale e rilasciare il pulsante.

Oval
Questa opzione consente di disegnare cerchi ed ellissi definendo un rettangolo al cui interno verrà disegnata la forma circolare. Il rettangolo viene fissato effettuando un click in corrispondenza del suo angolo in alto a sinistra e, tenendo premuto il pulsante sinistro del mouse, spostando il puntatore fino a fissarne l'angolo destro rilasciando il pulsante.

Rectangle
Questa opzione funziona in modo analogo all'opzione Oval permettendo di disegnare un rettangolo.

Eraser
E' molto facile commettere errori mentre si disegna. Questa opzione consente perciò di cancellare porzioni di linee del disegno realizzato. La cancellazione si effettua tenendo premuto il pulsante sinistro del mouse mentre si muove il puntatore.

Paint
Con questa opzione è possibile colorare aree chiuse, delimitate da linee aventi colore identico a quello di riempimento; per effettuare tale operazione è necessario posizionare il puntatore del mouse all'interno dell'area da colorare ed effettuare un click con il pulsante sinistro. E' comunque opportuno fare molta attenzione nell'utilizzo di tale funzione in quanto è facile rovinare il disegno realizzato: Paint riempie infatti solo aree delimitate da linee chiuse ed è quindi sufficiente un piccolo spazio vuoto nella linea per far sì che vengano riempite anche aree non desiderate. Quando si lavora con l'opzione Paint, è quindi consigliabile effettuare frequenti salvataggi del disegno che si sta realizzando.

Cambiamento dei colori
Nella parte inferiore della finestra vi sono quattro rettangoli colorati (blu, bianco, nero ed arancione) preceduti dalla parola "Color:" scritta nel colore attualmente selezionato. E' possibile cambiare questo colore effettuando un click su uno degli altri rettangoli colorati, causando una corrispondente modifica nella colorazione della parola "Color:".

- Selezionare come colore attivo il bianco.

- Per ripulire completamente la finestra nella quale sono state eseguite le prove, è necessario selezionare l'opzione New del menù a tendina File; si ottiene la visualizzazione del seguente messaggio:

```
Current file is not saved
Do you want to save it?
Press Y if you want to save it
Press N if you don't want to save it
Press C if you want to cancel command
```

Ripensando a cosa succede quando in Basic viene selezionata l'opzione New senza aver memorizzato il programma, questo messaggio dovrebbe risultare familiare. L'unica differenza sta nel fatto che in questo caso la risposta deve essere fornita da tastiera. Se non si desidera effettuare il salvataggio del disegno è necessario premere il tasto N. Amiga ripulisce la finestra e visualizza nuovamente la domanda:

```
Enter 1 if you want to edit sprites
Enter 0 if you want to edit bobs
```

Tutte queste operazioni sono comunque inutili nel caso non sia stato disegnato nulla.

- Allargare ora la finestra. La stella che si intende disegnare deve essere contenuta in una finestra di dimensioni tali che X ed Y valgano rispettivamente circa 140 e 70.

Il disegno di una stella

Per disegnare il contorno della stella che dovrà essere successivamente colorata, è necessario selezionare l'opzione Line del menù a tendina Tool.

Non appena si riesce a disegnare una stella più o meno decente, simile a quella contenuta in figura 3, è consigliabile, prima di procedere alla sua colorazione, effettuare un salvataggio del disegno. Quando si utilizzano

programmi di grafica è infatti opportuno prendere tutte le possibili precauzioni. Per memorizzare il disegno: selezionare l'opzione Save As del menù a tendina File e, in seguito alla richiesta "Enter filename>", battere il nome Stella.bob seguito da <Return>. Si accende la luce del drive ed il bob appena disegnato viene salvato.

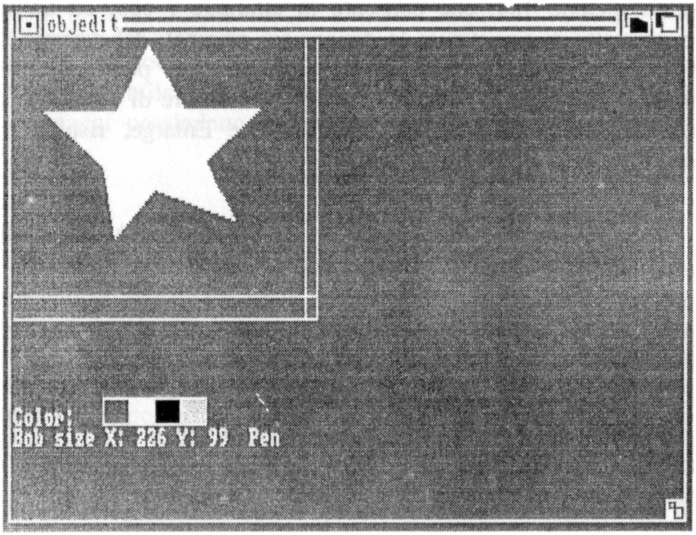


Figura 3: Realizzazione di un bob stella utilizzando l'object editor

La fase di colorazione

Prima di iniziare la colorazione è necessario controllare attentamente che le linee di contorno siano completamente chiuse; in caso contrario, procedere a chiudere i contorni utilizzando il comando Pen. Selezionare poi l'opzione Paint del menù a tendina Tool e, dopo aver posizionato il puntatore del mouse al centro della stella, effettuare un click con il pulsante sinistro. La stella dovrebbe colorarsi di bianco; nel caso invece in cui le linee tracciate non siano perfettamente combacianti fra loro, si colora di bianco l'intera finestra. E' allora necessario selezionare l'opzione Open del menù File per ricaricare il disegno con la stella, eliminare gli eventuali spazi fra le linee e riprovare ad utilizzare Paint (ecco uno dei motivi per cui è importante salvare il disegno frequentemente).

Dopo aver colorato la stella, è opportuno effettuare un ulteriore salvataggio del disegno mediante l'opzione Save del menù a tendina File.

L'allargamento di un bob

Completato il disegno della stella colorata rimangono ancora da analizzare alcune caratteristiche dell'Object Editor. Il menù a tendina Enlarge contiene un'opzione 4x4, che consente di allargare i bob, ed un'opzione 1x1, che permette di ripristinare le dimensioni originali del disegno. E' comunque necessario tener presente il fatto che è possibile utilizzare l'opzione 4x4 solo se Y è minore di 31 ed X è minore di 100 ed in secondo luogo che, dopo la selezione dell'opzione Enlarge, risulta abilitata solamente l'opzione di disegno Pen.

Per uscire da ObjEdit

Per uscire dall'object editor effettuare un click sui gadget di chiusura delle finestre ObjEdit e LIST (quest'ultima visualizzata solamente nel caso siano stati commessi errori durante l'utilizzo del programma).

Per poter definire un oggetto, Amiga consente quindi di disegnare l'oggetto stesso invece di obbligare, come avviene nella maggior parte dei casi, a dover ricorrere ai comandi PEEK e POKE.

Dopo aver terminato la sessione di lavoro con l'object editor ci si ritrova nuovamente di fronte alla schermo Workbench. Prima di richiamare il BASIC, è consigliabile fare un po' di ordine.

- Chiudere e riaprire la finestra BasicDemos per far comparire l'icona del file Stella.bob.

- Spostare poi tale icona, e tutte le icone associate agli eventuali programmi creati, nel cassetto Programmi.

- Richiudere nuovamente la finestra BasicDemos dopo averla eventualmente riordinata mediante l'opzione Snapshot.

1.12 Ulteriori informazioni sugli oggetti grafici

Dopo aver introdotto i due tipi diversi di oggetti grafici gestiti da Amiga, bob e sprite, è utile vedere come possono essere utilizzati e quali vantaggi e svantaggi possiedono.

La storia di Amiga
La storia dello sviluppo di Amiga aiuta a comprendere il motivo per cui vi siano due tipi diversi di oggetti grafici. In origine la macchina doveva nascere come computer da gioco (game-machine) più sofisticato rispetto a quanto presente allora sul mercato. La piccola società Amiga Corporation contattò Jay Miner per progettare i chip grafici speciali (Jay Miner merita di essere citato poichè si devono a lui la maggior parte delle capacità grafiche di Amiga). Miner aveva già progettato i chip grafici per i computer Atari 400 e 800 e la sua grande esperienza gli permise quindi di non incontrare grosse difficoltà nella progettazione del chip video di Amiga (chiamato *Denise*) che possiede capacità grafiche molto più elevate rispetto ai chip Atari. Denise è in grado anche di generare sprite.

Oltre a Denise, responsabile della maggior parte delle funzionalità grafiche, Amiga possiede altri chip dedicati alla gestione della grafica. Uno di questi, il chip *Agnus*, si divide in due componenti, *Blitter* e *Copper*, che hanno suscitato i maggiori entusiasmi alla presentazione di Amiga. La funzione principale del Blitter è la manipolazione ad alta velocità della memoria e delle locazioni dello schermo, processo nel quale viene coinvolto anche il Copper. Dal punto di vista della programmazione in BASIC non è necessario approfondire ulteriormente questi argomenti tecnici.

Il funzionamento del Blitter è basato sui concetti precedentemente illustrati per la realizzazione di animazione al computer. Se è possibile spostare e copiare rapidamente gli schermi, perchè non dovrebbe essere possibile disegnare, cancellare, ridisegnare e ricostruire gli sfondi? In quest'ottica sono stati sviluppati oggetti come i bob (blitter object blocks) che rappre-

sentano oggetti grafici controllati e manipolati dal Blitter.

Differenze tra bob e sprite

Nella valutazione dei vantaggi e degli svantaggi dei due tipi di oggetti grafici, è necessario considerare come essi siano generati e manipolati da due chip completamente differenti. Entrambi possiedono quindi vantaggi e svantaggi a seconda dell'applicazione in cui si devono utilizzare.

Per quanto riguarda gli sprite essi sono limitati, nella dimensione orizzontale, a 16 punti di schermo (corrispondenti alla larghezza di due caratteri), mentre non vi sono limitazioni nella dimensione verticale. Vi sono tuttavia altri limiti, come il numero dei colori disponibili contemporaneamente (quattro) o il numero massimo di sprite visualizzabili contemporaneamente (otto), limiti comunque superabili con alcuni particolari trucchi.

I bob non possiedono nessuna limitazione per quanto riguarda le dimensioni, consentono di avere a disposizione 32 diversi colori ed il loro numero massimo è limitato solo dalla quantità di memoria disponibile. I bob hanno tuttavia lo svantaggio di risultare più lenti degli sprite, soprattutto nel caso in cui ne vengano utilizzati parecchi contemporaneamente. La tabella seguente riassume vantaggi e svantaggi dei due oggetti grafici.

Oggetto grafico	Sprite	Bob
Controllato da:	Denise	Agnes/Blitter
Numero massimo:	8 (raddoppiabili)	illimitato (limitato solo dalla memoria)
Colori:	fino a 4 (per 2 sprite dello stesso colore)	fino a 32 (scelti fra i colori disponibili)
Velocità:	molto veloce	lento

Tabella 1: confronto tra bob e sprite.

AmigaBASIC rende abbastanza semplice muovere bob e sprite con l'uso di comandi validi per entrambi i tipi di oggetto. L'unica differenza riguarda quindi la creazione dell'oggetto stesso con l'object editor.

1.13 Caricare il bob Stella

Per la stella disegnata e memorizzata in precedenza è stato utilizzato un bob in quanto le dimensioni limite di uno sprite risultavano troppo ridotte per contenere il disegno.

E' necessario ora scrivere la routine BASIC che definisca il movimento del bob Stella sullo schermo. Prima di poter effettuare tale operazione deve però essere ricaricato in memoria il programma di videotitolazione.

- Lanciare il BASIC (se non è già stato attivato) e selezionare l'opzione Open del menù a tendina Project.

Nell'angolo superiore sinistro dello schermo viene visualizzato un requester con cui Amiga richiede il nome del programma da caricare ("Name of program to load:").

- Effettuare un click sul rettangolo contenuto nel requester e battere il nome del cassetto in cui è contenuto il programma da caricare seguito dal carattere / e dal nome del programma stesso terminato dal tasto <Return>:

```
Programmi/VideoTitolatrice¶
```

Se Amiga visualizza il messaggio d'errore "File not found", significa che il nome del cassetto o del programma indicato è errato. Se non si ricordano precisamente questi nomi, è necessario rimpicciolire le finestre BASIC e LIST ed esaminare i contenuti del dischetto Extras.

Una volta caricato il programma il suo listato verrà visualizzato nella finestra LIST.

LOAD
Il metodo di caricamento appena descritto costituisce un'alternativa all'effettuazione di un doppio-click sull'icona associata al programma stesso.

Essendovi in BASIC due metodi per il salvataggio di un programma, è lecito sospettare che vi siano anche due metodi di caricamento: quello appena visto, con l'ausilio quindi di un menù a tendina, ed un secondo metodo utilizzando, in modo diretto, il comando LOAD (che possiede la stessa sintassi del requester visto sopra). Per caricare il programma di titolazione si può quindi anche battere nella finestra BASIC la seguente istruzione:

```
LOAD "Programmi/VideoTitolatrice"¶
```

Avendo a disposizione due diversi metodi per effettuare la stessa operazione, diventa molto soggettiva la scelta di quello più adatto da utilizzare. Se si preferisce usare la tastiera è meglio ricorrere al modo diretto, mentre, se ci si trova maggiormente a proprio agio con i menù a tendina, si tenderà ad utilizzare il primo metodo descritto. In entrambi i casi, comunque, AmigaBASIC effettuerà l'identica azione.

Dopo aver scritto, nei precedenti paragrafi, la routine ImmettiTesto si deve ora scrivere una routine associata alla seconda voce del menù iniziale. Tale routine, etichettata "LeggiOggetto:", sarà quindi richiamabile selezionando il numero 2 sul menù principale.

Dopo la linea di programma

```
IF a$="1" THEN ImmettiTesto¶
```

inserire la linea:

```
IF a$="2" THEN LeggiOggetto¶
```

Muoversi lungo il programma

Si provi ora a premere la combinazione di tasti <Alt> <Cursore in basso> (cioè il tasto <Alt> e, tenendolo schiacciato, il tasto con la freccia rivolta verso il basso), rilasciandoli poi contemporaneamente. Questa azione consente di spostare il cursore direttamente all'ultima linea del programma. In modo analogo, la combinazione di tasti <Alt> <Cursore in alto> consente di spostarsi all'inizio del programma. Le combinazioni <Alt> <Cursore a sinistra> e <Alt> <Cursore a destra> consentono invece di spostare il cursore all'inizio ed alla fine di una linea di programma. Queste combinazioni di tasti forniscono quindi la possibilità di effettuare veloci spostamenti all'interno del programma senza dover ricorrere al mouse.

A questo punto è possibile inserire, alla fine del programma, la routine LeggiOggetto:

```
LeggiOggetto:¶
CLS¶
PRINT "Immetti il NOME dell'oggetto che vuoi
caricare"¶
INPUT NomeOgg$¶
IF NomeOgg$="" THEN CLS : GOTO Partenza¶
OPEN NomeOgg$ FOR INPUT AS 1¶
OBJECT.SHAPE 1,INPUT$(LOF(1),1)¶
CLOSE 1¶
IndicatoreOgg=1 : CLS : GOTO Partenza¶
```

In questa routine, che acquisisce un oggetto grafico dal dischetto, vi sono tre linee contenenti nuovi comandi:

```
OPEN NomeOgg$ FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$LOF(1),1)
CLOSE 1
```

File

La prima di queste tre linee (la cui spiegazione, in riferimento alla gestione di *file* e dischetti, viene trattata nel terzo capitolo del libro) consente di "aprire" il file specificato, in modo tale da poter caricare i dati in esso contenuti. L'esecuzione di questa istruzione porta alla lettura dei dati contenuti nel file il cui nome viene fornito dall'utente (in seguito alla richiesta del programma effettuata nelle prime tre linee della routine LeggiOggetto). Come si può notare, esaminando la quarta linea, viene effettuato anche un controllo sull'input fornito: se l'utente preme il tasto <Return> senza aver inserito un nome, il programma riprende l'esecuzione dall'inizio chiedendo nuovamente il nome del file da aprire.

I dati acquisiti da dischetto vengono assegnati ad una particolare variabile stringa, INPUT$ (del tutto simile alle variabili stringa Ciao$ o a$) alla quale possono essere assegnate stringhe lunghe fino a 32767 caratteri, spazio sufficiente per i file contenenti i dati di uno sprite o di un bob. Il file aperto dal programma contiene tutte le informazioni essenziali per la definizione dell'oggetto, nel caso specifico una stella. L'object editor gestisce

la composizione della stringa che conterrà tutti i dati relativi alla stella in un determianto ordine.

OBJECT. SHAPE

Il comando OBJECT.SHAPE, inserito nella seconda delle tre linee, assegna la stringa all'oggetto grafico numero 1, fornendo ad Amiga tutte le informazioni necessarie per la visualizzazione dell'oggetto, come il colore, l'altezza, la larghezza ed altri parametri. La terza linea chiude invece il file precedentemente aperto.

Viene poi assegnato il valore 1 alla variabile IndicatoreOgg utilizzata per specificare che è già stato caricato un oggetto. Dopo questa operazione il programma torna all'inizio.

Dopo aver stabilito come deve apparire l'oggetto, è necessario definire il suo movimento.

- Inserire dopo le due linee che costituiscono la routine Richiesta la seguente linea:

```
IF a$="3" THEN MuoviOggetto¶
```

Inserire poi le seguenti linee alla fine del programma:

```
MuoviOggetto:¶
CLS : IF IndicatoreOgg=0 THEN BEEP ELSE Mossiere¶
PRINT "Non c'è alcun oggetto attualmente in memoria!"¶
PRINT "Premi un tasto"¶
Pausa:¶
a$=INKEY$¶
IF a$="" THEN Pausa¶
CLS : GOTO Partenza¶
```

Le linee appena scritte contengono l'inizio del programma per il movimento dell'oggetto. Viene innanzitutto effettuato un controllo sul fatto che un oggetto sia già stato caricato e, nel caso in cui la variabile IndicatoreOgg valga 0 (non si è ancora aperto nessun file), Amiga emette un suono, visualizza un messaggio d'errore e si pone in attesa indicando di premere un tasto per continuare e tornare al menù principale.

ELSE

Nelle ultime linee descritte vi sono due nuovi comandi: ELSE e INKEY$. Il comando ELSE rappresenta un'estensione del comando IF ... THEN, utilizzato finora nella forma IF (condizione vera) THEN (sequenza di comandi).

```
IF IndicatoreOgg=0 THEN BEEP ELSE Mossiere¶
```

La prima parte di questa linea contiene la forma più semplice del comando IF: se la variabile IndicatoreOgg è uguale a 0, Amiga emette un suono. La seconda parte dell'istruzione specifica un'azione che verrà eseguita solo se IndicatoreOgg è diverso da 0, nel qual caso il programma effettua un salto alla routine etichettata Mossiere. IF ... THEN ... ELSE consente di eseguire determinate istruzioni se una particolare condizione è vera ed altre istruzioni se la condizione non è verificata, cioè:

```
IF (condizione vera) THEN (sequenza di comandi 1)
ELSE (sequenza di comandi 2)
```

Questa forma di IF ... THEN è particolarmente utile quando si devono eseguire dei comandi se una condizione viene verificata o effettuare un salto nel programma se la condizione non è verificata.

INKEY$

Il comando INKEY$ consente di memorizzare un carattere inserito da tastiera. Nel programma in esame, se la variabile a$ risulta vuota, cioè non viene premuto alcun tasto, il programma effettua un salto all'etichetta Pausa, salto ripetuto finchè non viene premuto un tasto. In questo modo è perciò possibile realizzare un ciclo di ritardo che si interrompe solamente quando la variabile a$ contiene un valore diverso dalla stringa vuota.

1.14 Per evidenziare gli errori: la modalità TRACE

• Effettuare un click all'interno della finestra BASIC e battere:

```
GOTO MuoviOggetto¶
```

Il modo trace

L'istruzione appena fornita consente di far partire l'esecuzione di un programma dal punto specificato. Nel caso in esame Amiga riscontra però un errore, in quanto non è stato ancora acquisito nessun oggetto, ed emette un suono visualizzando il testo del programma battuto in precedenza (se la finestra LIST non dovesse essere visualizzata, selezionare l'opzione Show List del menù a tendina Window).

• Selezionare l'opzione Trace On del menù Run.

Compare nella finestra LIST la routine etichettata MuoviOggetto ed un piccolo rettangolo arancione che si sposta lentamente da un comando a quello seguente (per essere più precisi, il rettangolo si sposta ciclicamente fra tre linee). E' stato attivato il modo trace nel quale Amiga evidenzia, passo per passo, le linee del programma in esecuzione. Questa modalità si rivela molto utile quando si incontrano problemi durante l'esecuzione di un programma e risulta praticamente indispensabile quando si devono analizzare programmi scritti da qualcun altro.

Quindi, nel caso la finestra LIST dovesse scomparire, ad esempio in seguito ad un comando INPUT, selezionare nuovamente l'opzione Show List del menù Windows. Se le linee del programma sono molto lunghe, AmigaBASIC modificherà il formato della linea visualizzata nella finestra LIST; spesso è quindi necessario allargare la finestra LIST. Per utilizzare al meglio il modo trace è consigliabile cercare di trovare un adeguato posizionamento sullo schermo delle finestre BASIC e LIST. Poichè il programma che si sta utilizzando visualizza i dati nella parte superiore della

finestra BASIC, si può pensare di posizionare la finestra LIST nella parte inferiore dello schermo. AmigaBASIC adatta automaticamente il listato del programma alle nuove dimensioni della finestra LIST.

Per chiarire eventuali dubbi rimasti sulle routine scritte, si consiglia di utilizzare un poco la modalità trace. Come si può notare, attivando tale modalità, il programma viene eseguito più lentamente rispetto al normale; questo si verifica in quanto Amiga deve eseguire un numero superiore di operazioni fra un comando e l'altro e l'esecuzione stessa deve poter essere seguita dall'utente.

Per disattivare la modalità trace, è necessario selezionare l'opzione Trace Off del menù a tendina Run.

TRON e TROFF

Per consentire di attivare la modalità trace solo per particolari parti di programma, AmigaBASIC mette a disposizione i due comandi TRON (TRace ON) e TROFF (TRace OFF) che possono essere utilizzati all'interno del programma come qualunque altro comando BASIC.

1.15 I comandi OBJECT

Come è possibile rilevare, la routine ImmettiTesto contiene un piccolo errore: se la si richiama una sola volta non vi sono problemi, mentre, se viene utilizzata più volte in successione, si ottiene il messaggio d'errore "Duplicate Definition". Questo errore viene generato dal comando DIM di dimensionamento della matrice stringa (dedicato al contenimento delle linee che comporranno il titolo). Non è infatti possibile dimensionare con il comando DIM la stessa variabile più di una volta.

Il modo più semplice per correggere questo errore è quello di spostare tutte le istruzioni contenenti il comando DIM all'inizio del programma in modo tale che esse vengano eseguite una sola volta. Per poter utilizzare questo metodo è però necessario sapere a priori il numero di elementi che comporranno le matrici che si devono definire. Poichè nel programma in questione la matrice Testo$ era composto da un numero massimo di 15 elementi, procedere a dimensionarlo con questo valore.

• Cancellare dalla routine ImmettiTesto la linea:

```
DIM Testo$(NumLinee)¶
```

DIM

E' possibile dimensionare contemporaneamente più vettori con un singolo comando DIM. E' quindi consigliabile dimensionare tutte le variabili che verranno utilizzate nel programma di titolazione all'inizio del programma. Per far ciò è necessario spostare il cursore all'inizio del programma e battere la linea seguente:

```
Inizializzazione:¶
d=15¶
DIM Testo$(d),MatriceColore(d,3), Mossa(d),
Velocita(d)¶
```

NOTA:
Si consiglia di procedere spesso alla memorizzazione del programma.

• Spostare il cursore alla fine del programma e battere le seguenti linee:

```
Mossiere:¶
PRINT "Muovere l'oggetto verso il suo punto di
partenza"¶
PRINT "usando i tasti freccia."¶
PRINT "Dopo averlo sistemato premere <RETURN>"¶
ox=100 : oy=100 : Destinazione=0¶
OBJECT.HIT 1,0,0¶
OBJECT.ON 1¶
OBJECT.STOP 1¶
Ciclo:¶
a$=INKEY$¶
IF a$=CHR$(13) THEN DestDef¶
IF a$=CHR$(28) THEN oy=oy-2¶
IF a$=CHR$(31) THEN ox=ox-5¶
IF a$=CHR$(30) THEN ox=ox+5¶
IF a$=CHR$(29) THEN oy=oy+2¶
OBJECT.X 1,ox : OBJECT.Y 1,oy¶
GOTO Ciclo¶
¶
DestDef:¶
CLS¶
Mossa(Destinazione*2+1)=ox :
Mossa(Destinazione*2+2)=oy¶
Destinazione=Destinazione+1 :
Mossa(0)=Destinazione¶
IF Destinazione=7 THEN FineDef¶
PRINT "Muovere l'oggetto verso la locazione"
Destinazione¶
PRINT "<RETURN> = Impostare un'altra locazione"¶
PRINT "<ESC> = Fine"¶
Ciclo2:¶
a$=INKEY$¶
```

```
IF a$=CHR$(13) THEN DestDef¶
IF a$=CHR$(27) THEN FineDef¶
IF a$=CHR$(28) THEN oy=oy-2¶
IF a$=CHR$(31) THEN ox=ox-5¶
IF a$=CHR$(30) THEN ox=ox+5¶
IF a$=CHR$(29) THEN oy=oy+2¶
OBJECT.X 1,ox : OBJECT.Y 1,oy¶
GOTO Ciclo2¶
¶
FineDef:¶
Mossa(0)=Destinazione¶
OBJECT.OFF 1¶
CLS : GOTO Partenza¶
```

In questa sezione del programma compaiono alcuni nuovi comandi che possono essere riuniti in due grandi categorie: comandi OBJECT ed altri tipi di comandi.

Bit, Byte e ASCII

Per spiegare il comando CHR$(x), l'unico appartenente alla seconda categoria, è necessario aprire una breve parentesi su alcuni concetti base di informatica.

Visto dall'esterno sembra che Amiga elabori numeri e lettere senza alcuno sforzo. Si potrebbe quindi affermare che Amiga fa parte di una generazione di computer che non ha niente a che fare con il passato. Tuttavia, nelle funzionalità di base non molto è cambiato rispetto ai tempi di ENIAC, il primo computer realizzato dall'uomo. I chip del microprocessore (il cervello di Amiga) possono infatti lavorare considerando solo due stati: acceso e spento. Tutte le funzionalità del computer sono basate su questi due stati di funzionamento.

Queste due condizioni vengono rappresentate con le cifre 0 ed 1 ed ogni singola cifra costituisce la quantità minima di informazione per un computer, il *bit*. Sfruttando alcuni accorgimenti tecnici e l'adozione della numerazione binaria, un computer può elaborare numerosi bit contemporaneamente potendo in tal modo lavorare non solo con zeri ed uno, ma anche con numeri più elevati.

Per quanto riguarda la manipolazione delle lettere da parte del computer, è necessario introdurre un altro concetto: ad ogni lettera viene associato un numero fra 0 e 255, dove 255 rappresenta un limite imposto dalle capacità elaborative dei primi microprocessori che non erano in grado di gestire numeri più grandi ad una velocità accettabile (il numero più grande che il microprocessore 68000 di Amiga può elaborare è 4.294.967.294). L'ordine con cui vengono associati alle varie lettere i numeri fra 0 e 255 è stato uniformato in modo tale che computer diversi possano comprendere informazioni identiche. A questo assegnamento standard è stato imposto il nome di *codice ASCII* (American Standard Code for Information Interchange).

CHR$

Il comando CHR$ viene utilizzato per convertire un numero nel corrispondente carattere del codice ASCII. Per analizzare meglio il funzionamento di questo comando, battere, all'interno della finestra BASIC, la seguente linea:

```
WIDTH 70: FOR x=0 TO 255 : ?CHR$(x) : NEXT x¶
```

Ulteriori informazioni sul codice ASCII

Durante l'esecuzione di questa istruzione, Amiga emette un suono. Finora tale comportamento è sempre stato associato ad un segnale d'errore; in questo caso non è invece necessario preoccuparsi poichè il suono emesso è associato ad un particolare carattere e quindi deriva da un numero fornito in codice ASCII. Il codice ASCII, oltre ai caratteri alfa-numerici, include infatti anche alcuni caratteri speciali detti caratteri di controllo. CHR$(7) genera ad esempio un suono, CHR$(12) ripulisce lo schermo. I caratteri di controllo non vengono tutti visualizzati; sullo schermo dovrebbero comunque comparire numerosi segni di punteggiatura, numeri, lettere maiuscole e minuscole e diversi rettangolini associati a caratteri speciali che Amiga non visualizza con lo stesso simbolo che li identifica sulla tastiera (del resto è già stato osservato che lo stesso tasto può essere identificato differentemente su tastiere diverse).

Dopo questa spiegazione dovrebbe essere abbastanza chiaro il funzionamento del comando CHR$. Esso viene generalmente utilizzato per controllare gli inserimenti effettuati. Nell'esempio seguente è presentata una routine che si pone in attesa della pressione del tasto <Esc>. Dato che tale routine non farà parte del programma di titolazione, prima di proce-

dere all'inserimento delle sue istruzioni, è consigliabile selezionare l'opzione NEW del menù File, per cancellare dalla memoria qualunque programma precedentemente caricato, ed effettuare un click all'interno della finestra LIST.

• Battere:

```
Pausa:¶
IF INKEY$ <> CHR$(27) THEN Pausa¶
```

Il computer si pone in attesa della pressione del tasto <Esc>.

OBJECT.SHAPE

I comandi OBJECT posti all'inizio della routine dedicata al movimento degli oggetti fanno parte di un particolare insieme di comandi (caratterizzati dalla parola OBJECT seguita da un punto e da un suffisso di specificazione) utilizzati per definire il movimento di bob e sprite. Il comando OBJECT.SHAPE viene usato per assegnare ad un oggetto una particolare forma. La sintassi del comando è la seguente:

```
OBJECT.SHAPE numero oggetto, stringa di definizione¶
```

Syntax error

Con il termine *sintassi* in BASIC ci si riferisce alla corretta composizione dei comandi. Se si battesse, ad esempio, nella finestra BASIC la linea:

```
OBJECT.SHAPE 1¶
```

si violerebbe la sintassi del comando in quanto è stato omesso il secondo valore (la stringa di definizione). Se si prova ad inserire questa istruzione si otterrà il messaggio d'errore "Syntax Error" utilizzata da Amiga per indicare che in un comando vi è qualcosa di errato o mancante.

I parametri

I valori da specificare in un comando vengono detti *parametri*. In tal modo "numero oggetto" del precedente esempio costituisce un parametro. Nel programma di videotitolazione questo parametro assumerà sempre il valore 1, dato che viene utilizzato soltanto l'oggetto numero 1.

OBJECT.X ed OBJECT.Y

OBJECT.X ed OBJECT.Y sono altri due comandi appartenenti alla classe OBJECT che consentono di posizionare un oggetto in un particolare punto dello schermo specificandone le posizioni orizzontale (X) e verticale (Y). Il punto d'origine cui le coordinate fanno riferimento si trova nell'angolo in alto a sinistra dello schermo. Il valore che può assumere X varia tra 0 e 617, mentre Y può variare tra 0 e 185. Specificando valori di X ed Y superiori ai valori massimi ammissibili la visualizzazione dell'oggetto non viene effettuata. Per specificare una coppia di coordinate, è quindi necessario fornire le seguenti istruzioni:

```
OBJECT.X object number, x-coordinate¶
OBJECT.Y object number, y-coordinate¶
```

OBJECT.ON

Un oggetto definito con le precedenti istruzioni non sarà comunque visibile finchè non si fornisce il comando OBJECT.ON la cui sintassi è:

```
OBJECT.ON object number¶
```

OBJECT.OFF

La controparte al comando OBJECT.ON è il comando OBJECT.OFF

```
OBJECT.ON object number¶
```

che causa la scomparsa di un oggetto dallo schermo.

OBJECT.VX ed OBJECT.VY

Poichè gli oggetti grafici sono "oggetti movibili" vi sono comandi dedicati alla definizione del loro spostamento. Due di questi comandi, OBJECT.VX ed OBJECT.VY (nei quali la lettera "V" sta per velocità) consentono di specificare la velocità di spostamento degli oggetti lungo le direzioni X (OBJECT.VX) ed Y (OBJECT.VY) in termini di punti di schermo al secondo.

Nella risoluzione standard utilizzata da AmigaBASIC si hanno a disposizione 640 X 256 punti (pixel) di schermo. Sottraendo i punti che costituiscono i bordi della finestra si ottiene la regione visibile dello schermo nella qua-

le agiscono i comandi OBJECT.X ed OBJECT.Y.

Una velocità orizzontale (VX) uguale ad 1 indica uno spostamento dell'oggetto verso destra di un punto al secondo; una velocità verticale (VY) pari a 2 indica uno spostamento verso il basso di due punti al secondo. Specificando sia la velocità orizzontale che quella verticale si ottiene uno spostamento dell'oggetto lungo la diagonale che va dall'angolo in alto a sinistra all'angolo in basso a destra dello schermo. Per invertire le direzioni di spostamento è sufficiente premettere al numero di punti al secondo il segno - per indicare una velocità negativa. La figura 4 risulta molto utile come schema di riferimento da utilizzare quando si devono fornire parametri di velocità nelle istruzioni per lo spostamento degli oggetti.

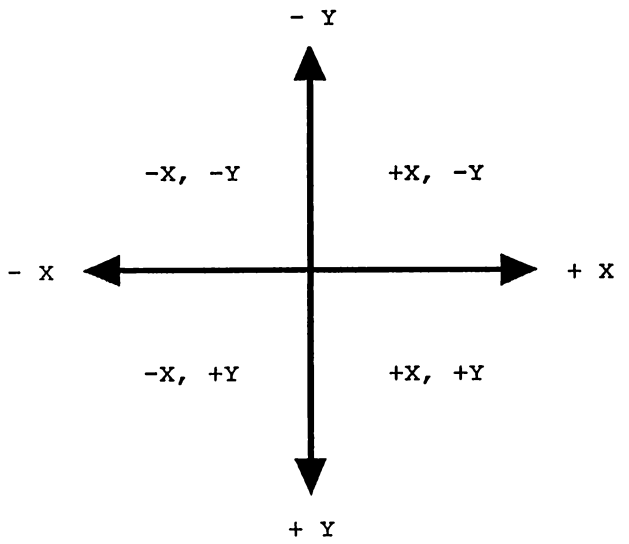


Figura 4: la polarità relativa alle velocità per il comando OBJECT

Se si vuole effettuare uno spostamento dall'angolo in alto a destra verso l'angolo in basso a sinistra, la velocità VX dovrà essere negativa, mentre VY dovrà essere positiva.

Un ultimo suggerimento: quando, in uno spostamento diagonale, la velocità VX è molto più elevata della velocità VY, lo spostamento risulta per lo più in senso orizzontale. Analogamente, vale il viceversa nel caso in cui

VY sia molto più grande di VX.

La sintassi dei due comandi OBJECT.VX ed OBJECT.VY è quindi:

```
OBJECT.VX object number, x-velocity¶
OBJECT.VY object number, y-velocity¶
```

OBJECT.START

Così come i comandi OBJECT.X ed OBJECT.Y, anche i comandi che definiscono la velocità di spostamento di un oggetto necessitano di un ulteriore comando di inizializzazione, il comando OBJECT.START:

```
OBJECT.START object number¶
```

OBJECT.STOP

L'esecuzione del comando OBJECT.START, che inizializza lo spostamento di un oggetto sullo schermo, non necessita di nessun altro intervento da parte dell'utente. Anche in questo caso esiste il comando per effettuare l'azione opposta, il comando OBJECT.STOP, che consente di fermare un oggetto nella posizione raggiunta:

```
OBJECT.STOP object number¶
```

E' necessario specificare il numero dell'oggetto per indicare, nel caso si abbiano più oggetti in movimento, a quale di essi ci si stia riferendo. Vi sono molti altri comandi dedicati al controllo degli oggetti in movimento che non verranno tuttavia utilizzati nel programma per la titolazione video e che verranno quindi trattati più avanti. E' possibile comunque trovare una completa descrizione di tutti i comandi appartenenti alla classe OBJECT nell'Appendice B.2 del libro.

OBJECT.HIT

Nella routine per il movimento degli oggetti rimane da esaminare un ultimo comando: OBJECT.HIT. AmigaBASIC consente di controllare se un oggetto sta per entrare in *collisione* con un altro sullo schermo. La collisione fra oggetti rappresenta uno degli eventi che possono influenzare l'esecuzione di un programma. Questo argomento è trattato in dettaglio nell'appendice B.2 sotto la voce ON COLLISION GOSUB. Senza questo comando, un oggetto dovrebbe fermarsi non appena si scontra con un al-

tro. Ciò può essere utile nel caso di un programma per il controllo del traffico, ma risulta inadeguato nel caso di un programma di videotitolazione. Il comando:

```
OBJECT.HIT object number,0,0¶
```

consente ad un oggetto di entrare in collisione sia con altri oggetti grafici che con i bordi dello schermo. Nel programma di videotitolazione viene effettuato un controllo sulla presenza o meno di un oggetto in memoria. Se è stato caricato in memoria un oggetto, le variabili ox ed oy, utilizzate per memorizzare le coordinate della posizione dell'oggetto, assumono il valore 100. La routine "Mossiere" attiva lo spostamento dell'oggetto numero 1, disattivando qualunque altro spostamento.

Per spostare un oggetto

E' ora possibile controllare gli spostamenti di un oggetto attraverso i tasti cursore causando variazioni nei valori delle coordinate X ed Y. La routine "Ciclo" consente di ripetere queste operazioni fino alla pressione del tasto <Return> che consente di indicare la posizione attuale come punto iniziale. Il programma esegue poi la routine "Destdef" (definizione della destinazione) in cui viene utilizzato la matrice Mossa (inizializzato all'inizio del programma) per memorizzare le coordinate X ed Y dei punti (in numero massimo di sette) nei quali dovrà avvenire lo spostamento. In particolare il numero di questi valori viene memorizzato nel primo elemento della matrice, Mossa(0).

La definizione dei punti di destinazione avviene in modo analogo alla definizione dei punti di partenza: vengono utilizzati i tasti cursore per definire lo spostamento ed il tasto <Return> per fissare i punti desiderati. L'inserimento si conclude dopo la memorizzazione del sesto punto o non appena viene premuto il tasto <Esc>. La routine "FineDef" effettua l'aggiornamento del numero di punti di destinazione, rimuove l'oggetto dallo schermo e ritorna poi al menù principale.

E' ora possibile eseguire il programma selezionando l'opzione 2 (Lettura dati Oggetto) per caricare in memoria i dati relativi alla stella definita e memorizzata nei precedenti paragrafi. L'opzione 3 (Movimento Oggetto) consente di definire il movimento che l'oggetto effettuerà. E' molto utile provare ad utilizzare quest'ultima funzione per trovare eventuali errori.

1.16 Il controllo dei colori

Amiga può visualizzare contemporaneamente 4096 diversi colori, numero inimmaginabile fino a pochi anni fa quando i computer più avanzati avevano già notevoli problemi per visualizzare più di 16 colori.

RGB

Uno dei fattori determinanti nella possibilità di disporre di un insieme tanto vasto di colori è rappresentato dal tipo di monitor supportato da Amiga, un monitor RGB *analogico*.

E' abbastanza noto il modo in cui funziona una televisione: un fascio di elettroni, che scandisce lo schermo linea per linea, colpisce uno strato di fosforo illuminandone alcuni punti. Ciò avviene tanto velocemente che l'occhio umano non riesce a rilevare la continua scansione del fascio di elettroni e quindi non si accorge di come una videata venga effettivamente costruita punto per punto o linea per linea.

Comunque, se lo sviluppo tecnologico si fosse fermato a questo stadio, si avrebbero oggi soltanto televisioni in bianco e nero. Infatti i monitor monocromatici a fosfori verdi, commercializzati per anni da società di computer, sono tecnicamente meno evoluti di un televisore in bianco e nero nel quale l'intensità del fascio di elettroni viene variata per visualizzare sullo schermo i diversi toni di grigio.

L'invenzione del televisore a colori è basata sull'utilizzo di tre fasci elettronici affiancati fra loro; ad ognuno dei tre fasci viene assegnato un particolare insieme di punti sullo schermo. Il primo di questi fasci è dedicato ai punti di colore rosso (red), il secondo al colore blu (blue) ed il terzo al verde (green). Da qui l'abbreviazione RGB (red-green-blue).

A partire da questi tre colori base si possono poi ottenere tutti i colori dello spettro. I tre fasci elettronici possono inoltre variare in intensità producendo quindi anche gradazioni differenti di uno stesso colore.

L'uso di questa nuova tecnologia venne immediatamente introdotto nella realizzazione di televisori a colori, mentre passò più tempo prima che l'idea fosse applicata nel campo dei computer.

I produttori di monitor monocromatici hanno cominciato recentemente a produrre monitor con tre fasci di elettroni di intensità costante. A causa della filosofia di base dei computer (acceso/spento, 0/1) l'applicazione di questa tecnologia risulta abbastanza semplice: vi sono tre connessioni diverse fra il monitor ed il computer per le bande R, G e B. Questa tecnica, che consente di avere a disposizione otto colori (bianco, nero, rosso, verde, blu, giallo, ciano, viola), viene detta controllo del colore *digitale*.

Amiga e RGB

Cosa c'è allora di diverso nella tecnologia del monitor di Amiga? Anche Amiga utilizza solo tre connessioni per il monitor, ma ognuna di esse possiede 16 diversi livelli di intensità consentendo di ottenere una qualità del colore quasi identica a quella di un televisore. La maggior parte dei programmi consente di regolare il colore attraverso variazioni delle intensità di rosso, verde e blu. Da questo punto di vista un programma di disegno come DeluxePaint si comporta in modo molto simile alle Preferences. E' possibile rendersi conto dell'effettiva esistenza di 4096 colori riducendo le dimensioni delle finestre BASIC e LIST e richiamando le Preferences del Workbench. Nella parte inferiore dello schermo delle Preferences compaiono tre gadget che consentono di regolare le tonalità dei tre colori base. Modificando le intensità di rosso, verde e blu si modificano i quattro colori base del Workbench.

NOTA:

I colori selezionati possono essere utilizzati effettuando un click sul gadget Use e rimangono validi fino ad un loro successivo cambiamento o fino allo spegnimento del computer.

PALETTE

Il cambiamento dei colori può essere effettuato anche in BASIC mediante l'uso del comando PALETTE al quale devono essere forniti come parametri quattro valori numerici. Il primo di questi numeri rappresenta il colore che si intende modificare. AmigaBASIC utilizza generalmente quattro co-

lori i cui numeri corrispondenti sono riassunti nella tabella seguente:

Colore	Numero
blu	0
bianco	1
nero	2
arancione	3

Gli altri tre parametri del comando PALETTE specificano le percentuali di rosso, verde e blu che compongono il nuovo colore e devono essere compresi fra 0 (corrispondente allo 0%) ed 1 (100%).

• Battere nella finestra Basic la seguente linea per modificare il colore blu:

```
PALETTE 0,0.6,0.2,0.4¶
```

In ogni caso, i valori percentuali possono anche essere forniti senza specificare lo zero prima del punto decimale per cui la precedente istruzione può essere fornita anche nella seguente forma:

```
PALETTE 0, .2, .3, .6¶
```

E' ora possibile includere nel programma di videotitolazione l'opzione "Definisci Colore". Per prima cosa è necessario inserire la seguente linea nella routine etichettata "Richiesta:" in modo tale da avere a disposizione un'altra opzione:

```
IF a$="4" THEN DefiniSciColore¶
```

Devono ora essere inserite altre istruzioni nella routine "Inizializzazione:" in modo che risulti:

```
Inizializzazione:¶
Colori=2¶
d=15 : MaxColori=(2^Colori)-1¶
ColoreTesto=1¶

DIM Testo$(d),MatriceColore(d,3),Mossa(d),
Velocita(d)¶
Riempitore$=STRING$(16,"-")¶
MatriceColore(1,1)=15¶
MatriceColore(1,2)=15¶
MatriceColore(1,3)=15¶
```

In fondo al programma inserire le linee di codice che seguono:

```
DefinisciColore:¶
CLS:PRINT "Valori del colore:"¶
ColoriPresenti:¶
FOR x=0 TO MaxColori¶
COLOR -(x=0),x¶
LOCATE 5,(x*4) + 1¶
PRINT x;CHR$(32);CHR$(32)¶
NEXT x¶
¶
CambioColore:¶
LOCATE 7,1:COLOR ColoreTesto,Sfondo¶
PRINT "Immetti il numero del colore da cambiare."¶
PRINT "(f = Fine)     " : BEEP¶
LOCATE 8,11 : INPUT Risposta$¶
IF UCASE$(Risposta$)="F" THEN AssegnaColore¶
Risposta$=LEFT$(Risposta$,2)¶
NumeroColore=VAL(Risposta$)¶
IF NumeroColore<0 OR NumeroColore>MaxColori
THEN BEEP: GOTO CambioColore¶
¶
RegolatoreRGB:¶
r=MatriceColore(NumeroColore,1)¶
```

```
g=MatriceColore(NumeroColore,2)¶
b=MatriceColore(NumeroColore,3)¶
LOCATE 10,1: PRINT "Rosso (R): <7>=- <8>=+ ";
Riempitore$¶
LOCATE 10,24+r : PRINT CHR$(124);¶
LOCATE 11,1: PRINT "Verde (G): <4>=- <5>=+ ";
Riempitore$¶
LOCATE 11,24+g : PRINT CHR$(124);¶
LOCATE 12,1: PRINT "Blu   (B): <1>=- <2>=+ ";
Riempitore$¶
LOCATE 12,24+b : PRINT CHR$(124);¶
LOCATE 13,1: PRINT "           <0>=Colore o.k."¶
PALETTE NumeroColore,r/15,g/15,b/15¶
¶
PremiTasti:¶
Key$=INKEY$¶
IF Key$="" THEN PremiTasti¶
IF Key$="7" THEN r=r-1¶
IF Key$="8" THEN r=r+1¶
IF Key$="4" THEN g=g-1¶
IF Key$="5" THEN g=g+1¶
IF Key$="1" THEN b=b-1¶
IF Key$="2" THEN b=b+1¶
IF Key$="0" THEN CambioColore¶
¶
IF r<0 THEN r=0¶
IF r>15 THEN r=15¶
IF g<0 THEN g=0¶
IF g>15 THEN g=15¶
IF b<0 THEN b=0¶
IF b>15 THEN b=15¶
¶
MatriceColore(NumeroColore,1)=r¶
MatriceColore(NumeroColore,2)=g¶
MatriceColore(NumeroColore,3)=b¶
GOTO RegolatoreRGB¶
```

```
¶
AssegnaColore:¶
a=Sfondo : a$="Sfondo"¶
GOSUB ImmettiColore:Sfondo=a¶
¶
a=ColoreTesto : a$="Colore Testo"¶
GOSUB ImmettiColore:ColoreTesto=a¶
¶
a=SfondoTesto : a$="Sfondo del Testo"¶
GOSUB ImmettiColore:SfondoTesto=a¶
¶
COLOR ColoreTesto,Sfondo¶
CLS : GOTO Partenza¶
¶
¶
ImmettiColore:¶
LOCATE 14,1¶
PRINT a$" corrente: ";a¶

Ciclo3:¶
LOCATE 15,1¶
PRINT a$" prescelto: "; : INPUT Risposta$¶
Risposta=VAL(Risposta$)¶
IF Risposta$="" THEN Risposta=.5¶
IF Risposta<0 OR Risposta>MaxColori THEN BEEP :
GOTO Ciclo3¶
IF Risposta<>.5 THEN a=Risposta¶
RETURN¶
```

Se si incontrano grossi problemi nella comprensione dei comandi discussi finora, si consiglia di ricorrere alle appendici dove tutti i comandi vengono discussi in dettaglio. Se questo non fosse ancora sufficiente è opportuno rileggere attentamente le parti in cui questi comandi vengono trattati.

Ecco ora spiegati dettagliatamente i nuovi comandi contenuti nell'ultima sezione del programma.

STRING$
Il comando STRING$(16,"-") genera una stringa contenente 16 trattini ("-") utilizzata per controllare il colore. Questa funzione si rivela estremamente utile quando devono essere utilizzate lunghe stringhe formate dallo stesso carattere.

• Provare ad inserire in modo diretto nella finestra BASIC la seguente linea:

```
WIDTH 70:?STRING$(800,"!")¶
```

WIDTH
Il comando WIDTH viene utilizzato per specificare la larghezza di una linea. Amiga infatti non inizia automaticamente una nuova linea quando raggiunge il limite destro dello schermo, ma prosegue scrivendo oltre il margine visibile. Si può verificare questa ultima affermazione inserendo:

```
WIDTH 255:?STRING$(800,"!")¶
```

In base al numero di caratteri per linea definito mediante le Preferences (60 o 80), il valore massimo di WIDTH può essere 60 o 76.

COLOR
Il comando COLOR, utilizzato per modificare il colore del testo visualizzato, è seguito da due parametri che specificano rispettivamente i numeri di colore del testo e dello sfondo. In tal modo, per poter scrivere su sfondo bianco, è necessario inserire:

```
COLOR 0,1¶
```

UCASE$
Il termine UCASE$(...) è un'abbreviazione del vocabolo inglese UpperCASE (maiuscolo). Il comando UCASE$ consente di convertire in lettere maiuscole l'intero contenuto di una stringa.

LEFT$ e RIGHT$
La variabile stringa Risposta$ è utilizzata per memorizzare il nome fornito dall'utente. E' possibile che l'utente inserisca più caratteri di quelli consen-

titi; poichè nel programma vengono comunque utilizzati soltanto i primi due caratteri della stringa, è necessario far ricorso al comando LEFT$ che consente di estrarre un dato numero di caratteri da una stringa, partendo dal primo carattere a sinistra. Il comando RIGHT$ permette l'operazione analoga con l'estrazione dei caratteri dal primo carattere a destra. Il primo parametro di questi comandi specifica la stringa da cui estrarre i caratteri.

• Inserire:

```
?LEFT$("Ciao",3),RIGHT$("Amiga",2)¶
```

E' possibile ora utilizzare il comando LEFT$ nel programma di titolazione per migliorare ulteriormente il controllo sulla selezione delle opzioni del menù principale.

• Modificare le seguenti linee nella routine Richiesta:

```
..
..
INPUT a$¶
a$=LEFT$(a$,1)¶
IF a$<"1" OR a$> "5" THEN BEEP : GOTO Richiesta¶
..
..
```

GOSUB e RETURN

Nella routine AssegnaColore viene utilizzato il comando GOSUB che differisce leggermente dal comando GOTO. Il nome del comando rappresenta infatti l'abbreviazione di "GO to SUBroutine" (salta alla subroutine), dove il termine subroutine identifica una porzione di programma che effettua particolari operazioni. Quando Amiga trova un comando GOSUB, memorizza il punto in cui esso è stato riscontrato e richiama la subroutine specificata; a questo punto viene eseguita la subroutine; quando viene incontrato il comando RETURN viene effettuato il ritorno al punto di chiamata della subroutine e l'esecuzione riprende dalla linea successiva al comando GOSUB. I comandi RETURN e GOSUB operano quindi insieme, esattamente come i comandi IF e THEN o FOR e NEXT e compariranno quindi sempre insieme.

L'operatore di disuguaglianza
Tra le linee della routine ImmettiColore una deve essere ancora analizzata:

```
IF Risposta<>.5 THEN a=Risposta
```

In BASIC il simbolo <> identifica l'operatore di disuguaglianza ("non uguale a" o "diverso da"). L'istruzione presa in considerazione controlla quindi che il valore sia diverso da 0.5.

Il controllo del colore
La variabile MaxColori indica il massimo valore ammissibile per l'identificatore di un colore. La routine DefinisciColore visualizza piccoli rettangoli contenenti i numeri di colore validi. Viene richiesto poi il numero del colore che si intende modificare (che può essere inserito da tastiera). Per terminare la fase di modifica dei colori è necessario premere il tasto <F> al quale viene applicato il comando UCASE$ (per convertire in maiuscolo il carattere inserito). Dopo aver selezionato il colore, la routine RegolatoreRGB visualizza un messaggio che indica quali tasti utilizzare per apportare eventuali modifiche al colore prescelto. I tasti numerici <7> e <8> consentono la modifica delle tonalità del rosso, <4> e <5> permettono il controllo del verde, mentre <1> e <2> regolano il blu. Il tasto <0> registra le modifiche apportate ed il programma chiede se si intende modificare un altro colore. Le variabili r, g e b contengono un valore numerico fra 0 e 15 che specifica le intensità di rosso, verde e blu dei vari colori e che viene memorizzato nella matrice MatriceColore.

La routine per il controllo del colore è meno efficace delle Preferences. Si può riscontrare il fatto che, quando si cambia per la prima volta un colore, si ottiene il nero. Accade questo in quanto all'inizio tutti i valori della matrice MatriceColore sono uguali a zero e quindi AmigaBASIC non è in grado di conoscere il valore attuale del colore selezionato. Per questa ragione nel programma è stata inserita la routine Inizializzazione che assegna il colore bianco alla matrice MatriceColore in modo tale da associare al testo un colore diverso dal nero.

Alla fine delle regolazioni dei colori, il programma richiede altri tre valori numerici per specificare il colore del testo ed i colori di sfondo dello schermo e del testo (se non si vogliono modificare i valori normali di questi tre colori, premere per tre volte il tasto <Return>).

1.17 Il programma di visualizzazione

Il programma di titolazione elettronica è ora vicino al suo completamento. L'unica subroutine che manca è quella dedicata alla visualizzazione dei risultati ottenuti.

Come scrivere un programma di presentazione
Per scrivere un'adeguata routine di visualizzazione dei dati finali è innanzitutto necessario specificare ciò che il programma deve fare. Ecco ciò che il programma video dovrebbe eseguire:

innanzitutto ripulire lo schermo ed attendere che l'utente prema il tasto <Return> (in questo periodo d'attesa potrebbe ad esempio essere collegato il video-registratore ed avviata la registrazione). Dopo che il tasto <Return> è stato premuto, viene effettuato un conteggio alla rovescia (10 secondi) e successivamente è possibile inserire il testo nel colore selezionato. Dopo aver inserito il testo che dovrebbe comparire insieme alla stella creata, inizia lo spostamento della stella stessa, mentre il testo rimane fisso sullo schermo. Alla fine sarebbe bello veder scomparire il testo dallo schermo esattamente come avviene nel programma AmigaTutor contenuto nel dischetto Extras. Dopo questo effetto vi sono alcuni secondi di silenzio per concludere la visualizzazione dei risultati del programma.

Il prossimo passo nello sviluppo del programma consiste in una verifica di quanto già fatto finora e di ciò che deve ancora essere realizzato. Fino ad ora sono state scritte le routine che consentono di realizzare:

un oggetto grafico contenuto nel dischetto, caricato in memoria ed etichettato con il numero 1;

una matrice di nome Mossa(d) contenente fino a sette punti per lo spostamento dell'oggetto grafico. Mossa(0) contiene il numero dei punti definiti, seguito dalle coordinate dei punti;

una matrice di nome Velocita(d) realizzato in modo analogo alla matrice Mossa(d), in cui vengono memorizzate le velocità di spostamento orizzontale e verticale utilizzate per muovere l'oggetto fra i diversi punti;

una matrice di nome Testo$ che contiene fino a 15 linee di testo da visualizzare;

i colori del testo e dello sfondo dello schermo sono memorizzati nelle variabili Sfondo, ColoreTesto e SfondoTesto; inoltre è dimensionata una matrice di nome MatriceColore che per il momento non viene utilizzata;

infine vi sono altre variabili e dati per ora di minore importanza.

Per poter scrivere l'ultima parte del programma, è necessario analizzare alcuni nuovi comandi. Il primo di essi risulterà utile per il conto alla rovescia iniziale. Amiga effettua calcoli di intervalli di tempo utilizzando alcuni particolari comandi. Si provi, ad esempio, a battere la seguente linea nella finestra BASIC:

```
?date$¶
```

DATE$
Il comando Date$ consente di visualizzare mese, giorno ed anno della data di sistema. Nel caso del programma di videotitolazione, si devono tuttavia utilizzare (per il conto alla rovescia) variazioni di secondi e quindi il comando Date$ si rivela inadeguato.

• Inserire la linea:

```
?time$¶
```

TIME$
Questo comando visualizza ore, minuti e secondi (secondo la regolazione attuale delle Preferences). E' possibile estrarre soltanto i secondi ricorrendo alla formula (VAL(RIGHT$(TIME$,2)) che risulta però abbastanza complessa e difficile da tenere a mente. In alternativa si può allora utilizzare la seguente linea:

```
?timer¶
```

TIMER
Il comando TIMER fornisce il tempo in secondi e decimi di secondo indicando il numero di secondi trascorsi dalla mezzanotte (00:00:00). Anche il contenuto di questa variabile dipende dalle regolazioni attuali delle Preferences.

INT
Nei confronti fra numeri non è generalmente necessario avere a disposizione le cifre alla destra del punto decimale. Nel caso del conto alla rovescia che si vuole effettuare, i decimi di secondo risultano superflui. Il comando INT(x), abbreviazione di INTeger (intero), consente di trascurare i numeri alla destra del punto decimale.

• Inserire la linea:

```
?INT(23.42143)¶
```

Come è possibile vedere si ottiene come risultato 23. Il comando INT non effettua un arrotondamento vero e proprio del valore fornito poichè ad esempio INT(9.99999) produce come risultato 9 e non 10. In seguito verrà presentato un comando che consente di effettuare arrotondamenti.

Per far in modo che Amiga resti in attesa per un secondo è necessario procedere come specificato in seguito: il valore ottenuto dal comando INT(TIMER) deve essere assegnato ad una variabile (viene in pratica memorizzato il numero di secondi trascorsi dalla mezzanotte); è necessario utilizzare poi un ciclo per confrontare continuamente il valore di questa variabile con il valore del timer, ciclo che viene eseguito fino a quando i valori non sono uguali, cioè fino a quando è trascorso circa un secondo. Sotto forma di listato ciò appare in questo modo:

```
Tim=INT(TIMER)¶
Pausa2:¶
IF INT(TIMER)=Tim THEN  Pausa2¶
```

CINT
CINT (Conversione ad INTero) è il comando per effettuare le operazioni di arrotondamento di un numero. L'arrotondamento avviene secondo le seguenti regole: se la parte decimale è inferiore al valore .50000 (incluso)

il numero viene arrotondato per difetto (al numero intero precedente), se è superiore a .50000 il numero viene arrotondato per eccesso (al numero intero successivo). In tal modo 4.50000 diventa 4, mentre 4.50001 diventa 5. I due comandi riportati nella linea seguente:

```
?cint(4.6) : ?int(4.6)¶
```

producono risultati differenti.

ABS

Un altro comando molto utile è ABS, che consente di estrarre il *valore assoluto* di un numero, cioè il valore di un numero relativo senza segno. Per chiarire eventuali dubbi inserire la seguente linea:

```
?abs(-3.5) : ?abs(3.5)¶
```

Il comando ABS verrà utilizzato nel programma di videotitolazione per calcolare le velocità orizzontale e verticale dell'oggetto in movimento a partire dalla distanza fra due punti appartenenti al cammino percorso.

SCROLL

Dopo questa serie di comandi che consentono di eseguire particolari funzioni matematiche, è possibile passare ora all'analisi di un nuovo comando grafico. Per poter realizzare l'effetto grafico che consente di far scomparire il testo dallo schermo è necessario ricorrere ad un particolare comando che permette di "muovere" porzioni di schermo.

• Inserire la seguente linea nella finestra BASIC:

```
FOR x=1 To 50 : SCROLL(0,0)-(630,150),2,2: ¶
```

Dopo la visualizzazione del testo, una sezione dello schermo si abbassa lentamente verso l'angolo in basso a destra. Viene effettuato uno *scrolling*, cioè uno scorrimento del contenuto dello schermo.

Il comando SCROLL consente di effettuare lo scroll in ogni direzione. La sintassi del comando SCROLL è la seguente:

```
SCROLL(x1,y1)-(x2,y2),direzione-x,direzione-y¶
```

L'espressione (x1,y1)-(x2,y2) specifica gli angoli opposti della porzione di schermo di cui effettuare lo scroll; le coordinate (x1,y1) si riferiscono all'angolo in alto a sinistra ed (x2,y2) all'angolo in basso a destra. I valori direzione-x e direzione-y specificano il numero di punti dello schermo di cui deve spostarsi il rettangolo lungo le direzioni orizzontale e verticale. Un'unica chiamata al comando SCROLL consente di spostare il rettangolo specificato nella posizione indicata. Per creare un finto movimento è comunque preferibile utilizzare un ciclo FOR ... NEXT ed eseguire più volte scroll ridotti.

Sono stati ora esaminati tutti i comandi che verranno utilizzati nella prossima sezione del programma. Battendo le linee seguenti è quindi possibile immaginare cosa accadrà in seguito all'esecuzione dei vari comandi. Si consiglia ancora una volta di procedere al salvataggio del lavoro effettuato.

- Posizionare il cursore alla fine del programma ed inserire le seguenti linee:

```
MostraTitolo:¶
CLS¶
PRINT "Premere il tasto <RETURN> "¶
PRINT "per iniziare la visualizzazione del titolo."¶
AspettaUnTasto:¶
a$=INKEY$¶
IF a$=CHR$(13) THEN CLS : c=10 :
GOTO ContoAllaRovescia¶
GOTO AspettaUnTasto¶
¶
ContoAllaRovescia:¶
LOCATE 14,36 : PRINT c¶
c=c-1:IF c<0 THEN InizioVisualizzazione¶
Tim=INT(TIMER)¶
Pausa2:¶
IF INT(TIMER)=Tim THEN Pausa2¶
GOTO ContoAllaRovescia¶
¶
InizioVisualizzazione:¶
WIDTH 60 ¶
```

```
COLOR ColoreTesto,Sfondo : CLS¶
COLOR ColoreTesto,SfondoTesto¶
FOR x=1 TO NumLinee¶
Testo$=LEFT$(Testo$(x),70)¶
h=INT((70-LEN(Testo$))/2)+2¶
LOCATE x+17-NumLinee,h : PRINT Testo$¶
NEXT x¶
COLOR ColoreTesto,Sfondo¶
IF Mossa(0)=0 THEN MossaTesto¶
¶
OBJECT.X 1,Mossa(1)¶
OBJECT.Y 1,Mossa(2)¶
OBJECT.ON 1¶
FOR x=1 TO Mossa(0)-1¶
OBJECT.STOP 1¶
GOSUB CalcoloVelocita¶
OBJECT.X 1,Mossa(x*2-1)¶
OBJECT.Y 1,Mossa(x*2)¶
OBJECT.VX 1,Velocita(x*2-1)¶
OBJECT.VY 1,Velocita(x*2)¶
OBJECT.HIT 1,0,0¶
OBJECT.START 1¶
Tst=TIMER¶
Ciclo4:¶
px=ABS(Mossa(x*2+1)-OBJECT.X(1))¶
py=ABS(Mossa(x*2+2)-OBJECT.Y(1))¶
IF INT(TIMER-Tst)<18 AND (px>15 OR py>15)
THEN Ciclo4¶
NEXT x¶
OBJECT.OFF 1¶
¶
MossaTesto:¶
Tst=TIMER¶
IF Mossa(0)<>0 THEN Finito¶
Pausa3:¶
IF TIMER-Tst<(2*NumLinee+2) THEN Pausa3¶
```

```
Finito:¶
FOR x=1 TO 30 ¶
SCROLL (1,1)-(630,100),0,3¶
SCROLL (1,100)-(630,180),0,-3¶
NEXT x¶
COLOR ColoreTesto,Sfondo¶
CLS : GOTO Partenza¶
¶
CalcoloVelocita:¶
ox=OBJECT.X (1) : oy=OBJECT.Y (1)¶
Mossa(x*2-1)=ox : Mossa(x*2)=oy¶
zx=Mossa(x*2+1) : zy=Mossa(x*2+2)¶
FOR xx=1 TO 64 STEP .2¶
Velocita(x*2-1)=CINT((zx-ox)/xx)¶
Velocita(x*2)=CINT((zy-oy)/xx)¶
IF ABS(Velocita(x*2-1))<40
AND ABS(Velocita(x*2))<40 THEN xx=64¶
NEXT xx¶
RETURN¶
```

Il funzionamento del programma

Per comprendere il funzionamento del programma è utile esaminare singolarmente le varie routine (in modo da poter confrontare le proprie interpretazioni con quelle fornite nel testo).

La visualizzazione del titolo

La routine MostraTitolo si pone in attesa della pressione del tasto <Return> da parte dell'utente. Il valore CHR$(13) corrisponde alla codifica ASCII di tale tasto. Successivamente viene assegnato il valore 10 alla variabile c, utilizzata come contatore per il conto alla rovescia. In ogni istante il valore di c viene visualizzato e decrementato di uno fino a quando non raggiunge il valore 0. Il ciclo Pausa2 consente di far trascorrere circa un secondo prima della prossima visualizzazione del valore di c.

La configurazione dei colori
La routine InizioVisualizzazione:, dopo aver inizializzato la larghezza dello schermo a 70 caratteri, ripulisce lo schermo con un colore di sfondo predeterminato e fissa il colore per il testo.

Il posizionamento del testo
Il ciclo che segue consente di porre il testo al centro dello schermo. L'inserimento del testo viene limitato a 70 caratteri per linea attraverso il comando LEFT$. La variabile h calcola poi la posizione orizzontale da cui far partire la visualizzazione del testo, mentre lo spazio restante sulla linea (70-LEN(Testo$)) viene diviso per due in modo da calcolare lo spazio che deve essere lasciato alla destra ed alla sinistra del titolo. Il testo viene visualizzato nella posizione calcolata e quindi centrato sullo schermo. Se non è stato definito alcun movimento (controllo effettuato dall'istruzione IF Mossa(0)=0), il programma salta immediatamente alla routine MossaTesto.

Il movimento dell'oggetto
Nel caso in cui sia stato definito uno spostamento, l'oggetto viene posizionato nel punto iniziale e viene avviato il ciclo per il suo movimento. Il ciclo viene continuamente interrotto per calcolare le nuove velocità, mentre il comando OBJECT.HIT 1,0,0 consente di controllare eventuali collisioni. Il ciclo Ciclo4 viene eseguito finchè l'oggetto non raggiunge la sua posizione finale. Tuttavia, a causa dei vari arrotondamenti effettuati, è possibile che l'oggetto non raggiunga mai la posizione finale. Per ovviare a questo eventuale inconveniente è stata inserita una routine di controllo del tempo impiegato dall'oggetto per raggiungere tale posizione: se questo tempo supera i 18 secondi, lo spostamento viene interrotto. E' stato scelto come limite massimo di tempo 18 secondi poichè questo è il tempo necessario per effettuare il più lungo spostamento possibile sullo schermo (da un angolo a quello opposto).

Il calcolo della velocità
La routine CalcoloVelocita determina le coordinate dell'oggetto. Esse non corrispondono al 100% a quelle contenute nella matrice Mossa(x), per cui vengono poi memorizzate in tale matrice ed assegnate alle variabili ox ed oy divenendo così le coordinate di partenza per lo spostamento successivo. Le coordinate del punto terminale per lo spostamento sono contenute

negli elementi successivi della matrice Mossa(x). Il ciclo FOR ... NEXT seguente effettua divisioni della distanza fra le coordinate X ed Y, per numeri sempre maggiori, fino a quando i due valori diventano minori di 40 in modo che la stella si sposti abbastanza velocemente (i valori X ed Y vengono ovviamente divisi per lo stesso numero in modo tale da mantenere costante il loro rapporto). Quando la velocità assume l'esatto rapporto, xx diventa uguale a 64 e viene terminato il ciclo FOR ... NEXT. (FOR xx=1 TO 64 STEP .2 incrementa xx di 0.2 finchè xx non è uguale a 64, istante nel quale termina il ciclo). Terminato tale ciclo il comando RETURN produce il ritorno al programma principale.

Lo spostamento del testo

Resta ora da esaminare solamente la routine MossaTesto. Se non si effettuano spostamenti dell'oggetto, il testo scompare velocemente dallo schermo per cui l'utente non ha il tempo necessario per poter leggere il testo stesso; nel programma viene quindi inserita una pausa di due secondi dopo l'inserimento di ogni linea.

La fine del programma

Dopo questo ciclo di attesa viene eseguito il comando SCROLL che consente di spostare le sezioni superiori ed inferiori del testo verso il centro dello schermo facendo quindi scomparire il testo dallo schermo. A questo punto viene nuovamente presentato il menù principale.

Da ultimo è necessario inserire, nella routine Richiesta:, sotto la linea:

```
IF a$="4" THEN DefinizioneColore¶
```

la linea:

```
IF a$="5" THEN MostraTitolo¶
```

per realizzare la chiamata della nuova routine.

Esecuzione del programma

E' ora finalmente possibile eseguire il programma selezionando l'opzione 1 per inserire il testo che si intende visualizzare, l'opzione 2 per acquisire la stella dal dischetto e l'opzione 3 per definirne il movimento.

Utilizzando l'opzione 4, è possibile modificare i colori, tenendo però presente che ogni modifica viene memorizzata solo dopo l'inserimento del carattere <F>. Il primo numero fornito specifica il colore dello schermo, il secondo quello del testo, il terzo consente di evidenziare il testo con una barra colorata. Premendo il tasto <Return> si confermano i colori fissati.

L'opzione 5 visualizza il titolo creato.

Il salvataggio del programma

Dopo aver apportato le ultime modifiche è necessario memorizzare il programma per l'ultima volta. Il programma di videotitolazione è finalmente terminato.

2 L'universo non è solo bianco e nero: i colori e la risoluzione di Amiga

Per quanto è stato visto fino ad ora si potrebbe affermare: OK, ecco un computer che è in grado di visualizzare 32 colori scelti tra i 4096 disponibili; ma come è possibile visualizzare più dei quattro colori che sono stati utilizzati fino ad ora?

Si è visto come i quattro colori utilizzabili possano essere cambiati mediante il comando PALETTE. Non si è tuttavia ancora affrontato il problema della visualizzazione di un numero di colori superiore a quattro in AmigaBASIC.

2.1 Un assaggio delle possibilità di Amiga: uno spettro di colori

E' necessario precisare che l'esempio fornito all'inizio di ogni capitolo viene utilizzato per mostrare le capacità di Amiga senza richiedere un grosso sforzo di programmazione. Non è necessario comprendere pienamente il programma riportato di seguito, ma è sufficiente inserirlo esattamente come è presentato in modo da poterne poi apprezzare i risultati prodotti.

Se si possiede un modello Amiga 1000 con solamente 256k è necessario apportare a questo programma alcune piccole modifiche, riportate nelle pagine successive.

• Inserire le seguenti linee:

```
SCREEN 1,320,200,5,1¶
WINDOW 2,"Ecco i meravigliosi colori che si possono
ottenere",,23,1¶
FOR x=0 TO 7¶
FOR y=0 TO 3¶
co=x+y+8¶
LINE (x*38,y*45) - ((x+1)*38,(y+1)*45),co,bf¶
NEXT x,y¶
WHILE INKEY$=""¶
PALETTE INT(31*RND)+1,RND,RND,RND¶
WEND¶
WINDOW CLOSE 2¶
SCREEN CLOSE 1¶
WINDOW 1¶
```

Ci si potrebbe chiedere se non vi sia un errore di stampa nella seconda linea di questo programma.

Ecco la stessa linea con il possibile errore sottolineato:

```
WINDOW 2,"Ecco i meravigliosi colori che si possono ot-
tenere",,23,1
```

Non è necessario preoccuparsi, non si tratta di un errore: le due virgole poste fra i doppi apici ed il numero 23 costituiscono la corretta sintassi del comando in AmigaBASIC.

E' consigliabile memorizzare sul dischetto il programma appena inserito, in modo da poterlo riutilizzare quando saranno stati compresi chiaramente tutti i concetti che ne sono alla base.

Alcune considerazioni prima di eseguire il programma: è possibile cambiare le dimensioni della finestra o modificarne la posizione. E' possibile inoltre interrompere il programma premendo un tasto qualunque.

Modifiche da apportare al programma per un suo utilizzo con il modello Amiga 1000 con 256k di memoria

La mancanza di memoria aggiuntiva nel modello Amiga 1000 diventa un grosso problema quando si stanno utilizzando numerosi colori in alta risoluzione. Senza una espansione di memoria RAM è pressochè impossibile visualizzare più di 8 colori contemporaneamente sullo schermo. Se si possiede un Amiga 1000 con soli 256k di RAM, è necessario innanzitutto modificare le prime due linee del programma.

Prima:

```
SCREEN 1,320,200,5,1¶
WINDOW 2,"Ecco i meravigliosi colori che si possono
ottenere",,23,1¶
```

Dopo:

```
SCREEN 1,320,256,3,1¶
WINDOW 2,"Ecco i meravigliosi colori che si possono
ottenere",,0,1¶
```

E' poi necessario modificare la quinta linea.

Prima:

```
co=x+y*8¶
```

Dopo:

```
co=(x+y*4) AND 7¶
```

Se si eccettuano le piccole restrizioni segnalate, questa versione del programma dovrebbe produrre gli stessi effetti di quella originale.

2.2 Pixel, colori e memoria: le risoluzioni possibili con Amiga

Nel precedente programma dimostrativo si è visto come sia possibile selezionare e modificare casualmente i colori con Amiga.

AmigaBASIC include comandi che permettono di gestire un numero qualsiasi di finestre. Tali finestre possono essere visualizzate in uno qualsiasi dei cinque schermi contemporaneamente gestibili. In BASIC è possibile utilizzare diverse risoluzioni per gli schermi e le finestre.

Colori e memoria

Prima di procedere all'analisi di concetti ed esempi di programmazione, è utile fornire alcune informazioni di carattere generale sul colore e sulla risoluzione per comprendere al meglio quanto verrà spiegato in seguito.

La gestione della grafica e dei colori implica un grande impiego di memoria. Nel caso si stia utilizzando un modello Amiga 1000 con 256k di RAM, molti programmi grafici non potranno funzionare correttamente, a causa della limitata disponibilità di memoria. Questo problema è abbastanza comprensibile se si tiene conto del fatto che il computer deve continuamente caricare in memoria le informazioni per la gestione dello schermo, utilizzando quindi una grande quantità di memoria. E' questo il prezzo che è necessario pagare a fronte della facilità di utilizzo di Amiga con il controllo del mouse e della sua grafica sofisticata.

Si consideri ad esempio come veniva effettuata la gestione dei bob: essi dovevano essere innnanzitutto caricati nella memoria; successivamente venivano visualizzati all'interno dello schermo, copiandoveli; ed infine, prima del movimento successivo, venivano rimossi. Amiga non è infatti in grado di ricordare la posizione di un oggetto sullo schermo, sia esso un carattere od un qualsiasi disegno. Poichè il computer è in grado di ricordare la posizione di un dato elemento dello schermo solamente mantenendolo in memoria, ogni grafico, ogni carattere, e persino ogni singolo punto visua-

lizzato sullo schermo, occupa una parte della memoria RAM.

Pixel

Lo schermo di un computer è composto da migliaia di piccoli puntini chiamati *pixel*. Un'immagine sullo schermo è composta di pixel esattamente come un disegno su carta è composto di punti di inchiostro. Quando i punti sono abbastanza piccoli e chi osserva lo schermo si trova ad una certa distanza, tali punti sembrano formare un disegno continuo.

All'interno di uno schermo monocromatico un pixel può assumere solamente uno tra due valori, acceso o spento (cioè visibile oppure no). In questo modo ogni pixel può essere facilmente associato ad un elemento di memoria: ogni punto di uno schermo monocromatico corrisponde ad un singolo bit nella memoria del calcolatore.

Risoluzione

Amiga consente differenti livelli di risoluzione; ogni livello visualizza un numero differente di pixel sullo schermo. La risoluzione più bassa consiste di 320 pixel in orizzontale e 256 in verticale (standard PAL). Il programma di dimostrazione all'inizio di questo capitolo, ad esempio, è visualizzato in bassa risoluzione. Moltiplicando 320 per 256 si ottiene un totale di 81920 pixel visualizzati sullo schermo.

Dato che la quantità di memoria utilizzata viene normalmente misurata in byte, ed un byte consiste di 8 bit, per questa risoluzione sono richiesti 10240 byte cioè 10 Kilobytes (10K). Se si desiderano maggiori informazioni sui multipli del byte, consultare l'Appendice E alla voce byte.

Il secondo livello di risoluzione consente di visualizzare 640 pixel in orizzontale e 256 in verticale. Questa è la risoluzione più frequentemente utilizzata. Ad esempio, sia il Workbench che AmigaBASIC utilizzano questa risoluzione. In totale sono visualizzati 163840 pixel, per cui uno schermo in media risoluzione richiede una quantità di memoria di 20k.

Vi sono due ulteriori livelli di risoluzione, realizzati nel cosiddetto *modo interlacciato*, di cui si tratterà più approfonditamente in seguito. Le loro risoluzioni sono rispettivamente di 320x512 (163840 pixel, cioè 20k) e 640x512 (327680 pixel, cioè 40k).

Colori multipli

I precedenti esempi di risoluzione sono stati leggermente semplificati, in quanto si è assunto che il disegno fosse visualizzato in un solo colore (o meglio, per essere più precisi, in due colori: uno utilizzato per lo sfondo, ad esempio il blu, ed uno per il primo piano, ad esempio il bianco). Se un pixel è bianco il bit corrispondente è acceso, cioè ha valore uno, se il pixel è blu, il corrispondente bit risulta spento.

Per uno schermo monocromatico risulta quindi necessario un solo bit per ogni pixel. Ma come è possibile ottenere contemporaneamente 4, 8 oppure 32 colori sullo schermo? La risposta è molto semplice: quando un solo bit non è sufficiente si usano più bit. Utilizzando, ad esempio, due bit per ogni punto, risulta possibile distinguere quattro colori: se entrambi i bit sono spenti, potrebbe essere visualizzato il blu; se il primo bit è spento ed il secondo acceso, il punto potrebbe essere di colore bianco; viceversa, se il primo bit è acceso ed il secondo spento, il punto potrebbe essere nero; se entrambi i bit sono accesi, si potrebbe infine avere un quarto colore, ad esempio l'arancione. In questo modo è possibile gestire schermi a quattro colori, come ad esempio quello del Workbench, o quello contenente le finestre LIST e BASIC dell'AmigaBASIC.

Con quattro colori ogni schermo richiede però 40k invece di 20. Se è necessario un numero maggiore di colori, devono essere utilizzati ulteriori bit. Utilizzando, ad esempio, tre bit per pixel, possono essere visualizzati otto colori. Le combinazioni possibili risultano infatti essere le seguenti: spento -spento -spento, spento -spento -acceso, spento -acceso -spento, spento -acceso -acceso, acceso -spento -spento, acceso -spento -acceso, acceso -acceso -spento e acceso -acceso -acceso.

E' possibile rimanere storditi da quest'insieme di possibili combinazioni accento-speso .... oops, spento-acceso. Per ovviare a questo problema è allora conveniente utilizzare il metodo standard per tali rappresentazioni, cioè le *cifre binarie*. Questo metodo consiste nell'indicare un bit spento con uno 0, mentre un bit acceso viene indicato con un 1. In base a questo metodo le combinazioni precedentemente presentate risultano essere codificate come segue:

```
000 001 010 011 100 101 110 111
```

Ecco il primo incontro con questi misteriosi numeri binari di cui verranno fornite maggiori informazioni nella Nota d'uso 3.

Lo stesso giochetto può essere applicato, in maniera identica, a combinazioni di quattro, cinque bit e, teoricamente, anche ad un numero superiore di bit. In tal modo i 4096 colori di Amiga possono essere codificati con 12 bit; tuttavia i chip non sono dedicati solamente alla visualizzazione della grafica, per cui solitamente si hanno a disposizione solo 32 colori.

La seguente tabella illustra tutti i possibili livelli di risoluzione, il numero possibile di colori e la corrispondente quantità di memoria richiesta da Amiga per gestirli.

Risoluzione	Colori	Bit/pixel	Memoria
320 x 256	2	1	8 kilobyte
(bassa risoluzione,	4	2	16 kilobyte
modo normale)	8	3	24 kilobyte
	16	4	32 kilobyte
	32	5	40 kilobyte
640 x 256	2	1	16 kilobyte
(alta risoluzione,	4	2	32 kilobyte
modo normale)	8	3	48 kilobyte
	16	4	64 kilobyte
320 x 512	2	1	16 kilobyte
(bassa risoluzione,	4	2	32 kilobyte
modo interlacciato)	8	3	48 kilobyte
	16	4	64 kilobyte
	32	5	80 kilobyte
640 x512	2	1	32 kilobyte
(alta risoluzione,	4	2	64 kilobyte
modo interlacciato)	8	3	96 kilobyte
	16	4	128 kilobyte

Tabella 2 I livelli di risoluzione di Amiga.

Bitplane

Dovrebbe ora essere chiaro come mai vi sia una così grande competizione tra i produttori di computer per realizzare computer con la maggior quantità di memoria al minor prezzo possibile. La tabella vista precedentemente mostra infatti abbastanza chiaramente come sia possibile riempire completamente 512k di memoria con due sole immagini. Ecco ora un'analisi teorica del rapporto tra memoria del calcolatore e risoluzione.

I bit che controllano un particolare pixel non sono posti uno di seguito all'altro all'interno della memoria: essi sono raggruppati in *bitplane*, in quanto questo risulta essere il metodo più comodo per Amiga per memorizzarli. Per gestire due colori è necessario un solo bitplane, mentre per quattro colori i bitplane richiesti sono due. Il primo bit associato ad un qualsiasi pixel è contenuto nel primo bitplane, mentre il secondo bit è contenuto nel secondo bitplane. Se è necessario un terzo, un quarto o addirittura un quinto bit per un pixel, viene aggiunto, in modo corrispondente, un terzo, un quarto ed un quinto bitplane.

Per comprendere meglio il concetto di bitplane, si provi ad immaginare una rappresentazione come quella riportata nella figura seguente:

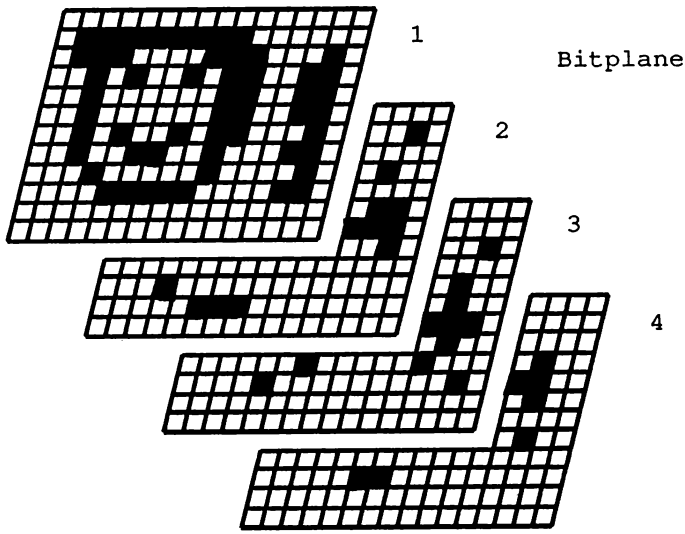


Figura 5 I vari livelli di bitplane producono, riuniti, una figura colorata.

I piani sono accatastati uno sopra l'altro; più piani sono presenti, più colori possono essere visualizzati.

Dal punto di vista tecnico la figura precedente non è del tutto corretta in quanto i singoli piani sono effettivamente allocati in memoria uno di seguito all'altro. In altre parole, Amiga memorizza i bit del primo bitplane, poi quelli del secondo, e così di seguito. Ciononostante, la figura 5 risulta fortemente esplicativa e visualizza abbastanza bene il concetto.

Se a questo punto si è compreso solamente metà di quanto spiegato, e neanche questa metà completamente bene, non è lo stesso il caso di preoccuparsi. Non è infatti essenziale conoscere completamente i dettagli della gestione della memoria per programmare i colori in AmigaBASIC. E' sufficiente sapere quanti bitplane sono richiesti per poter gestire un certo numero di colori e, approssimativamente, quanta memoria è richiesta per una certa immagine, in quanto è abbastanza facile utilizzare completamente 512k se si utilizzano in maniera sconsiderata i colori e gli schermi.

Nel prossimo paragrafo verranno fornite le informazioni relative ad uno schermo ed al suo uso.

2.3 Animazioni grafiche: gli schermi Amiga

Con molta probabilità si sara già provato ad utilizzare un programma grafico durante il quale l'intero schermo viene mandato in secondo piano (effettuando un click sull'apposito gadget) facendo in tal modo comparire uno schermo completamente nuovo (ad esempio la famosa demo della palla che rimbalza Bouncing Ball). Amiga può sovrapporre differenti schermi uno sopra l'altro ed il loro ordine può essere modificato utilizzando i gadget front e back. Nel caso si proceda alla chiusura di tutti gli schermi presenti, rimarrà sul monitor uno sfondo completamente blu.

Che cosa è uno schermo?
Non si devono confondere gli schermi con le finestre, in quanto rappresentano due diversi oggetti caratteristici di Amiga. In questo contesto il termine *schermo* si riferisce ad un intero quadro visualizzato sul monitor. Viceversa, le finestre e le icone sono oggetti visualizzati all'interno di uno schermo. Per comprendere pienamente questa affermazione si può provare ad eseguire il seguente esperimento:

- Rimpicciolire le dimensioni delle finestre LIST e BASIC in modo tale che risulti completamente visibile lo schermo del Workbench (per compiere questa operazione effettuare un click sul gadget di dimensionamento che si trova nell'angolo in basso a destra di queste finestre e, tenendo premuto il pulsante sinistro del mouse, spostare il puntatore fino ad ottenere la dimensione desiderata).

- Spostare il puntatore sulla riga Titolo bianca che si trova alla sommità dello schermo del Workbench.

- Premere il pulsante sinistro del mouse e, tenendolo schiacciato, muovere il puntatore verso il basso fino a quando l'intero schermo, incluse le finestre e le icone in esso contenute, scompare dal lato inferiore del monitor.

In questo modo si è spostato lo schermo del Workbench. Non ci dovrebbe essere nessun altro schermo dietro ad esso, a meno che non vi sia in esecuzione un altro programma oppure non sia rimasto aperto lo schermo di un programma BASIC provato precedentemente.

Uno schermo può, in teoria, contenere un numero illimitato di finestre o di icone. La risoluzione ed il numero massimo di colori associati ad un particolare schermo rimangono però immutati, in quanto questi valori devono essere specificati all'atto della generazione dello schermo stesso. Tali parametri rimangono costanti fino a quando lo schermo viene ridefinito oppure chiuso. Risulta perciò impossibile specificare, ad esempio, un numero differente di colori od una diversa risoluzione per una finestra particolare dello schermo, in quanto i parametri rimangono gli stessi per tutti gli oggetti dello schermo.

- Riportare lo schermo del Workbench nella sua posizione originale ed inserire la seguente linea nella finestra BASIC:

```
SCREEN 1,320,256,3,1¶
```

SCREEN
Non appena si preme il tasto <Return> viene visualizzato uno schermo vuoto che avrà una risoluzione di 320x256 pixel ed un numero massimo di otto colori. Gli schermi sono effettivamente aree grafiche indipendenti, se si astrae dal fatto che schermi diversi possono essere visualizzati sullo stesso monitor. E' possibile notare come il nuovo schermo presenti un gadget front ed un gadget back, esattamente come quello del Workbench. Effettuando un click sul gadget back ricompare in primo piano lo schermo del Workbench. Questo metodo di posizionamento in primo e secondo piano funziona nello stesso modo sia per gli schermi che per le finestre.

Dopo aver creato uno schermo, risulta possibile definire delle finestre all'interno di esso. A questo proposito, si provi ad inserire la linea seguente nella finestra BASIC:

```
WINDOW 1,"Ciao",,0,1¶
```

WINDOW
Nel nuovo schermo comparirà una finestra di nome "Ciao" che AmigaBA-

SIC assume come nuova finestra BASIC (tra poco si spiegherà il perchè di questo comportamento).

In ogni caso, Amiga sta ora visualizzando due schermi: il Workbench e quello appena creato. Per la maggior parte del tempo si lavora utilizzando lo schermo Workbench, come accade ad esempio quando si programma in AmigaBASIC. Gli altri schermi vengono solitamente utilizzati per eseguire in background altri programmi.

Spesso risulta difficile utilizzare i gadget front e back, in quanto essi sono nascosti dalle varie finestre contenute in uno schermo. Vi è perciò un altro metodo per portare in secondo piano lo schermo del Workbench. La combinazione di tasti <Logo Commodore> <N> (<Amiga sinistro> <N> nel caso di un Amiga 1000) posiziona lo schermo Workbench in primo piano rispetto a tutti gli altri. Per posizionare invece lo schermo Workbench dietro a tutti gli altri è possibile utilizzare la combinazione di tasti <Logo Commodore> <M> (o <Amiga sinistro> <M>).

SCREEN CLOSE

Per chiudere lo schermo appena creato, inserire la seguente linea nella finestra BASIC:

```
SCREEN CLOSE 1¶
```

Questa istruzione causa però anche la chiusura della finestra BASIC con cui si stava lavorando. Si rimane così senza una finestra BASIC. Non è il caso di preoccuparsi, in quanto è sufficiente selezionare l'opzione Show Output dal menù a tendina Windows per ottenere la visualizzazione di una finestra BASIC nello schermo del Workbench.

Ora che si conoscono i più importanti comandi AmigaBASIC necessari per gestire schermi e finestre, è opportuno capire il significato dei vari numeri inclusi nei precedenti comandi.

A questo proposito si prenda in considerazione un attimo l'esempio presentato:

```
SCREEN 1,320,256,3,1
```

Parametri del comando SCREEN
Il primo numero dopo il comando SCREEN costituisce il numero identificativo dello schermo (*screen ID*). AmigaBASIC è in grado di gestire fino a quattro schermi contemporaneamente, per cui lo screen ID può assumere un valore compreso tra 1 e 4.

Si dovrebbe intuire facilmente il significato dei due numeri successivi: il primo indica il numero dei pixel orizzontali, ed il secondo quello dei pixel verticali. Questi numeri non specificano, però, la risoluzione dello schermo, bensì la sua attuale dimensione nella risoluzione corrente. Si provi ad esempio ad eseguire la seguente linea:

```
SCREEN 1,200,256,3,1¶
```

Quando si inserisce un valore per i pixel verticali inferiore a 256, non si è in grado di spostare la freccia del puntatore sotto il lato inferiore di tale schermo. Se si muove questo schermo non è possibile superare l'altezza definita.

• Provare l'istruzione seguente:

```
SCREEN 1,200,100,3,1¶
```

Il quarto parametro del comando SCREEN specifica il numero di bitplane da utilizzare per lo schermo che si sta definendo. In questo modo si controlla il numero di colori che lo schermo è in grado di visualizzare.

L'ultimo parametro specifica la risoluzione dello schermo; sono possibili i valori riportati nella tabella seguente:

Valore	Livello di risoluzione	Pixel
1	Bassa risoluzione, normale	320 x 256
2	Alta risoluzione, normale	640 x 256
3	Bassa risoluzione, interlacciato	320 x 512
4	Alta risoluzione, interlacciato	640 x 512

Tabella 3: Risoluzioni possibili per il comando SCREEN.

I valori forniti per le dimensioni orizzontale e verticale dello schermo devono essere compatibili con la risoluzione specificata in quanto, in caso contrario, si ottengono strani risultati. Ad esempio, è consigliabile non selezionare una dimensione di 440x256 con una risoluzione di 320x256 (risoluzione 1). Anche se AmigaBASIC non segnala alcun messaggio di errore, è comunque meglio evitare di fornire valori che superino i limiti stabiliti dalla risoluzione adottata.

Modo interlacciato

Nella tabella mostrata sopra compare il termine *interlacciato.* E' ormai ora di spiegare il significato di questa parola. Per prima cosa si vedrà un esempio di dimostrazione degli effetti del modo interlacciato. Inserire a tale proposito la seguente linea nella finestra BASIC:

```
SCREEN 1,640,512,1,4: WINDOW 1,"Prova",
(0,0)-(400,300),15,1¶
```

Si possono ora notare distintamente due cose: i caratteri sullo schermo sono estremamente piccoli ed il monitor presenta uno sfarfallio (flickering). Il maggior vantaggio del modo interlacciato sta nel fatto che esso permette una risoluzione verticale maggiore, consentendo di produrre oggetti grafici di altezza fino a 512 pixel. Lo svantaggio di tale modalità è che essa causa uno sfarfallio dello schermo. E' necessario richiamare nuovamente qualche nozione di tecnologia televisiva per spiegare come è possibile ottenere la maggior risoluzione verticale e per quale motivo l'immagine subisce uno sfarfallio.

Si dovrebbe già essere a conoscenza del fatto che il movimento degli oggetti sullo schermo è artificioso piuttosto che reale. Le immagini sono semplicemente visualizzate una sopra l'altra a grande velocità (approssimativamente 30 volte al secondo). All'occhio umano questa animazione sembra continua e fluida. Tuttavia, con una velocità di visualizzazione di 30 fotogrammi (frame) al secondo, l'immagine subisce un forte sfarfallio.

Se non è richiesto l'utilizzo di più frame al secondo (per ottenere grafica in risoluzione più elevata, cosa che complicherebbe però la trasmissione), per stabilizzare l'immagine viene utilizzato un piccolo accorgimento. Nel primo sessantesimo di secondo vengono visualizzate sul monitor solamente le linee pari dell'immagine (viene in pratica visualizzata soltanto metà immagine). Nel secondo sessantesimo di secondo l'immagine viene

costruita nuovamente, visualizzando questa volta le linee lasciate inalterate durante la prima frazione di secondo. Con questo metodo ad occhio nudo sembra che si stiano visualizzando 60 frame al secondo, numero di frame sufficiente a stabilizzare l'immagine ed evitarne lo sfarfallio.

Stabilizzazione dell'immagine

Amiga utilizza lo stesso accorgimento, indipendentemente dal fatto che si stia utilizzando un monitor Amiga od un normale televisore. Normalmente viene utilizzata una risoluzione di 256 linee (risoluzione verticale); per raddoppiare questo numero in modo interlacciato Amiga invia una immagine differente quando il monitor sta disegnando il secondo insieme di linee, combinando poi le due parti grafiche per costituire un'immagine con una risoluzione verticale raddoppiata. Tuttavia, l'immagine risulta abbastanza instabile, in quanto vengono ora visualizzati solamente 30 frame al secondo.

Per concludere il discorso relativo al modo interlacciato, vi è da dire che lo sfarfallio appare più evidente nel caso si utilizzi un maggior contrasto fra i colori dello schermo. Oltre a questo, l'intensità dello sfarfallio dipende anche dalla forma dell'oggetto visualizzato. Se si decide quindi di utilizzare il modo interlacciato è necessario tener sempre presente queste ultime considerazioni.

Se si vuole ottenere un'immagine completamente stabile in modo interlacciato, è possibile utilizzare un'alternativa. Sono disponibili monitor speciali i quali, aumentando la persistenza, minimizzano il flickering. Comunque, se si pensa che il fenomeno del flickering sia molto poco professionale per un computer come Amiga, per ricredersi è sufficiente osservare attentamente ciò che viene proiettato alle spalle dei lettori di un notiziario televisivo. L'effetto che si può notare è praticamente lo stesso.

2.4 Creazione delle finestre Amiga

Dopo aver visto come creare gli schermi, è ora interessante creare finestre all'interno degli stessi. Come si è potuto notare nell'esempio precedente, questo compito viene svolto dal comando WINDOW.

WINDOW
Se lo schermo interlacciato creato nell'esempio precedente è ancora attivo (come si può rilevare dal debole sfarfallio degli altri schermi), è necessario chiuderlo con il comando:

```
SCREEN CLOSE 1¶
```

- Ritornare alla finestra BASIC e selezionare l'opzione Show Output dal menù a tendina Windows. Inserire poi la seguente linea:

```
WINDOW 2,"Prova", (320,20)-(615,150),31,-1¶
```

I possessori di un modello Amiga 1000 con soli 256k di RAM devono apportare una piccola modifica alla linea sopra riportata, inserendo al suo posto la linea seguente:

```
WINDOW 2,"Prova", (320,20)-(580,150),31,-1¶
```

Viene visualizzata una finestra di nome Prova.

- Provare ora ad inserire in tale finestra la linea seguente:

```
FOR x=1 TO 100: ? x: NEXT x¶
```

Come è possibile vedere, nella nuova finestra vengono visualizzati i numeri da 1 a 100. Il comando:

```
WINDOW OUTPUT 1¶
```

consente di visualizzare l'output nella finestra BASIC.

E' ora giunto il momento di compiere queste azioni in un modo più sistematico. Per prima cosa verranno spiegati i vari parametri del comando WINDOW:

```
WINDOW 2,"Prova", (320,20)-(615,150),31,-1¶
```

Per il modello Amiga 1000 con soli 256k di RAM :

```
WINDOW 2,"Prova", (320,20)-(580,150),31,-1¶
```

Parametri del comando WINDOW

Il primo numero dopo il nome del comando specifica il numero identificativo (Window ID) della finestra. AmigaBASIC fornisce la possibilità di aprire un numero qualsiasi di finestre, nei limiti delle capacità della memoria disponibile. Il numero 1 è riservato alla finestra BASIC per cui è possibile utilizzare per le proprie finestre numeri identificativi superiori a 2. Utilizzando il numero 1 come identificatore è possibile modificare dimensione, risoluzione e colori della finestra BASIC. Tuttavia, se si desidera visualizzare più di quattro colori, oppure una risoluzione maggiore di 640x256 pixel, la finestra deve essere spostata su un nuovo schermo, in quanto le inizializzazioni dello schermo del Workbench non possono essere modificate. La finestra BASIC attuale risulta quindi limitata ad una risoluzione di 640x256 e ad un massimo di due bitplane (4 colori).

Il secondo parametro del comando WINDOW è il nome della finestra. Esso deve essere specificato come una stringa di testo all'interno dei doppi apici, oppure sotto forma di una variabile stringa.

Il terzo insieme di parametri del comando WINDOW specifica la posizione e la dimensione della finestra. Se questi valori non sono specificati, la finestra ricopre l'intero schermo. Per visualizzare una finestra più piccola è necessario specificare le coordinate dell'angolo superiore sinistro e dell'angolo inferiore destro della finestra, secondo il seguente formato:

```
(x1,y1)-(x2,y2)
```

Ad esempio:

```
(100,10)-(500,140)
```

Il quarto valore determina le caratteristiche della finestra. Ad ogni proprie-

tà che una finestra può possedere viene associato un valore di riferimento. I singoli valori vengono poi sommati tra loro per determinare un unico numero che specifica quali proprietà la finestra deve avere. I valori disponibili sono riportati nella tabella seguente:

1	Le dimensioni della finestra possono essere modificate con il gadget di dimensionamento.
2	La finestra può essere spostata a schermo con il mouse.
4	La finestra può essere posizionata in primo e secondo piano con i gadget back e front.
8	La finestra può essere chiusa con il gadget di chiusura.
16	I contenuti della finestra vengono memorizzati quando essa viene modificata o posta in secondo piano.

Tabella 4: Valori per i parametri del comando WINDOW

Ad esempio, se si desidera una finestra che possa essere mossa (2) e chiusa effettuando un click sul gadget apposito (8) è necessario inserire come quarto valore 10 (8+2). Una finestra le cui dimensioni possano essere modificate (1), che possa essere spostata (2), e che possa essere portata in secondo piano (4), richiede il valore 7 (1+2+4) come quarto parametro. Se la finestra deve avere tutte le possibili opzioni, il quarto parametro deve essere 31 (1+2+4+8+16). Si noti, comunque, come l'ultima opzione richieda una grossa quantità di memoria, in quanto Amiga deve mantenere in memoria l'intero contenuto della finestra ed al tempo stesso gestirne un'altra. Il modello Amiga 1000 con 256k di RAM non è realmente in grado di utilizzare questa opzione, avendo un'insufficiente quantità di memoria a disposizione.

I contenuti di una finestra che non sono mantenuti in memoria (in quanto non è stato specificato il valore 16) vengono cancellati non appena la finestra viene spostata oppure quando un'altra finestra viene portata in primo piano. Per questo, quando nel calcolo del quarto parametro non si utilizza il valore 16, è consigliabile assicurarsi che la finestra definita rimanga sempre in primo piano e non venga mai portata in secondo piano. Per essere sicuri di questo fatto si può utilizzare il valore 0 (nessuna opzione).

I gadget (di chiusura, di dimensionamento, etc.) associati ad una finestra compariranno o meno nella stessa in funzione dal valore specificato come quarto parametro.

L'ultimo valore nel comando WINDOW specifica lo schermo in cui deve essere visualizzata la finestra che si sta definendo. Se non viene specificato alcun valore, oppure viene specificato il valore -1, la finestra viene visualizzata nello schermo del Workbench.

WINDOW OUTPUT

Dall'attimo in cui viene creata con un comando WINDOW, la finestra diventa automaticamente la finestra di output. Questo significa che ogni output di un programma BASIC in esecuzione verrà visualizzato su tale finestra. Per ridirigere l'output su una qualsiasi altra finestra è necessario utilizzare il comando seguente:

```
WINDOW OUTPUT numero_finestra
```

WINDOW OUTPUT ridirige semplicemente l'output su un'altra finestra, senza modificarne le caratteristiche. AmigaBASIC consente di dirigere l'output anche su una finestra chiusa o non visibile. Per portare la nuova finestra di output in primo piano è necessario utilizzare il comando:

```
WINDOW numero_finestra
```

WINDOW CLOSE

Il comando WINDOW CLOSE consente di chiudere la finestra specificata; esso non richiede altri parametri. Ovviamente, questo comando funziona solamente con finestre precedentemente definite.

```
WINDOW CLOSE numero_finestra
```

Pulizia delle finestre

Il comando precedente pulisce i contenuti di una finestra specificata, senza però cancellarla. AmigaBASIC non fornisce un comando che consenta di cancellare una finestra dalla memoria dopo che è stata aperta. Anche il comando SCREEN CLOSE non cancella le finestre dallo schermo. Solamente un nuovo comando RUN oppure una cancellazione dalla memoria del programma attraverso la corrispondente opzione di uno dei menù a ten-

dina, porta alla cancellazione delle finestre e degli schermi.

Per questo motivo non si dovrebbero utilizzare troppe finestre in quanto questo aspetto poco funzionale dell'AmigaBASIC causa spesso un errore in corrispondenza di un comando WINDOW che precedentemente funzionava correttamente con la conseguente visualizzazione del messaggio d'errore Illegal Function Call (messaggio che può comparire anche in seguito al verificarsi di altri errori). Se dovesse sorgere questo problema, utilizzare un altro numero identificativo per la finestra oppure cancellare tutte le finestre rilanciando il programma.

E' possibile a questo punto provare ad utilizzare il comando WINDOW, realizzando esempi di propria iniziativa.

Pur avendo visto ed utilizzato diversi esempi (compresi quelli di propria invenzione) è possibile che questo paragrafo sia sembrato un poco noioso e pedante. Per ovviare a questo eventuale inconveniente, si cercherà ora di sfruttare le conoscenze maturate sulla gestione delle finestre e degli schermi in AmigaBASIC per creare qualcosa di utile e divertente.

Nel programma di videotitolazione precedente sono stati finora utilizzati solamente quattro colori; si vedrà ora come superare questa limitazione.

- Richiamare il programma di videotitolazione e modificare la routine Inizializzazione nel modo seguente:

```
Inizializzazione:¶
Colori=2¶
d=15: MaxColori=(2^Colori)-1
ColoreTesto=1¶
SCREEN CLOSE 2
IF Colori>2 THEN SCREEN 2,640,256,Colori,1:
WINDOW 2,"Videotitolatrice",,28,2¶
DIM Testo$(d),MatriceColore(d,3),Mossa(d),
Velocita(d)¶
Riempitore$=STRING$(16,"-")¶
MatriceColore(1,1)=15¶
MatriceColore(1,2)=15¶
MatriceColore(1,3)=15
```

E' consigliabile salvare immediatamente la nuova versione del programma utilizzando, se non si vuole perdere la precedente versione, un nuovo nome.

La variabile Colori contiene il numero di bitplane che vengono utilizzati. Essa viene inizializzata con il valore 2, ma se si desidera utilizzare un numero superiore di colori (utilizzando un Amiga con almeno 512k di memoria) è possibile specificare il valore 3 (8 colori) oppure 4 (16 colori). Se il valore di Colori è maggiore di 2, il programma procederà a definire una finestra ed uno schermo corrispondente. E' possibile verificare questo fatto selezionando l'opzione 4 del menù principale (Definizione Colore).

Come si può notare le cose cominciano a farsi divertenti ... Ora che si è appreso come gestire gli schermi e le finestre, è possibile analizzare i comandi AmigaBASIC che utilizzano le finestre per l'output grafico.

2.5 Versatilità: i primi comandi grafici

Questo paragrafo descrive dettagliatamente i comandi grafici di Amiga. E' possibile notare come ogni comando grafico presenti un insieme abbastanza completo di opzioni che lo rendono estremamente versatile.

- Portare in primo piano la finestra BASIC ed attivarla (effettuando un click al suo interno) in modo da poter eseguire il prossimo esperimento.

- Si comincerà facendo qualcosa di molto semplice.

- Provare ad eseguire la seguente linea:

```
PSET (400,100)¶
```

PSET
Osservando attentamente lo schermo si dovrebbe notare un piccolo puntino bianco visualizzato approssimativamente al centro dello schermo: si tratta approssimativamente della posizione da cui solitamente inizia la finestra LIST, circa a metà schermo. Nel caso non si riesca a vedere tale puntino, è possibile renderlo un pò più grande.

- Inserire la linea seguente:

```
PSET (401,100)¶
```

Questo dovrebbe provocare la visualizzazione di un altro puntino affiancato al primo. Si dovrebbe ora essere in grado di vederli abbastanza chiaramente. Il comando BASIC PSET viene utilizzato per generare i singoli punti di un'immagine grafica. Poichè due punti su uno schermo non rappresentano niente di speciale, si continuerà creando una serie di punti tra loro adiacenti:

```
FOR x=1 TO 240: PSET (x,x),3: NEXT x¶
```

Questa istruzione consente di visualizzare una linea diagonale arancione sullo schermo.

Il comando PSET specifica dove deve essere posizionato il punto sullo schermo utilizzando l'insieme delle coordinate X e Y. Queste coordinate sono racchiuse tra parentesi (x,y) esattamente come accadeva per il comando WINDOW. Il concetto è il medesimo del famoso gioco della battaglia navale: un insieme di due coordinate specifica il bersaglio; allo stesso modo Amiga disegna un puntino in corrispondenza della posizione specificata come "bersaglio".

In una finestra posta all'interno di uno schermo da 640x256 pixel, il valore della y può variare tra 0 e 240 (parte dello schermo è utilizzata dal riquadro e dai simboli associati alla finestra). Il colore del punto può essere specificato subito dopo le sue coordinate. La sintassi risulta essere la seguente:

```
PSET (x,y),colore
```

L'istruzione BASIC precedente, che disegna una linea diagonale arancione sullo schermo, è abbastanza semplice. La variabile x viene incrementata da 1 a 184 con passo 1. I valori delle coordinate X ed Y sono sempre coincidenti tra loro, in quanto assumono sempre il valore della variabile x. In tal modo viene disegnato un punto in corrispondenza delle coordinate (1,1), (2,2), (3,3) e così via fino a (240,240) ottenendo pertanto una linea lungo lo schermo con una inclinazione di 45 gradi.

AmigaBASIC fornisce però un metodo più semplice per disegnare linee: il comando LINE. Utilizzando tale comando la linea precedente può essere disegnata con la seguente istruzione:

```
LINE (1,1)-(240,240)¶
```

LINE

I parametri del comando LINE sono costituiti dalle coordinate del punto di partenza e del punto finale della linea separate fra loro da un trattino. Di seguito a queste può essere specificato il colore da utilizzare per disegnare la linea. E' possibile omettere la prima coppia di coordinate; in questo caso la linea comincerà dal punto finale dell'ultima linea tracciata (si tenga presente che il trattino va comunque sempre inserito):

```
LINE -(500,10),2¶
```

RND

Un metodo molto semplice per creare grafica è quello di utilizzare valori random (casuali). Ciò può essere effettuato utilizzando la variabile RND (RaNDom) che contiene dei numeri casuali sempre compresi tra 0 e 1. Come verifica si può provare l'istruzione seguente ed osservare i valori generati:

```
FOR x=1 TO 10: ? RND: NEXT x¶
```

Digitare ora la seguente routine dimostrativa nella finestra LIST e provvedere immediatamente a salvarla su dischetto. Si cercherà di acquisire confidenza con RND modificando poco per volta tale routine. Si consiglia inoltre di continuare ad effettuare esperimenti per proprio conto. Se si riesce ad ottenere effetti di particolare gradimento, salvare la versione che li genera con un nome differente, in modo da mantenerla.

La prima versione consente di disegnare puntini colorati in posizioni casuali.

```
CLS¶
Ciclo:¶
PSET (617*RND,184*RND),3*RND¶
GOTO Ciclo¶
```

E' necessario specificare una cosa relativamente ai numeri casuali: moltiplicando RND per un numero (ad esempio 617) si ottiene un valore casuale compreso tra 0 ed il numero specificato (617). Nell'esempio precedente vengono in questo modo determinate sia le coordinate che il colore da utilizzare. Ben presto lo schermo risulta coperto da un notevole numero di puntini colorati. E' possibile interrompere il programma utilizzando il menù a tendina, oppure premendo contemporaneamente i tasti <Amiga> e <.>.

Se si ha a disposizione una grande quantità di memoria, è possibile utilizzare una risoluzione maggiore per lo schermo ed un numero superiore di colori.

```
Colori=4¶
MaxColori=2^Colori-15¶
SCREEN 1,640,256,Colori,2¶
WINDOW 2,"Punti",,31,1¶
CLS¶
Ciclo:¶
PSET (617*RND,240*RND),3*RND¶
GOTO Ciclo¶
```

E' possibile rendere quanto si vuole casuale la composizione RGB dei colori. A tale scopo è sufficiente inserire queste linee subito prima del comando CLS:

```
FOR x=1 TO (2^Colori)-1¶
PALETTE x,RND,RND,RND¶
NEXT x¶
```

La formula (2^Colori)-1 verrà spesso utilizzata. Essa consente di calcolare, a partire dal numero di bitplane disponibili (contenuto nella variabile Colori), il numero massimo di colori utilizzabili. Il simbolo ^ identifica l'elevamento a potenza, per cui 2 elevato alla 4 diventa 2^4 in AmigaBASIC.

RND diviene molto facile da utilizzare in combinazione con il comando PALETTE, in quanto il comando PALETTE richiede un numero compreso tra 0 ed 1, esattamente il numero fornito da RND.

- Provare ora a sostituire la linea contenente il comando PSET con la linea seguente:

```
LINE -(617*RND,184*RND),MaxColori*RND¶
```

Questa versione genera un numero molto elevato di linee colorate. Il comando LINE è comunque più potente di quanto si possa intuire.

- Per rendersene conto provare ad inserire la linea seguente nella finestra BASIC:

```
LINE (10,10)-(300,100),1,b¶
```

Come detto precedentemente, i comandi grafici AmigaBASIC risultano essere ancor più potenti e versatili quando sono utilizzati con i parametri addizionali. Il parametro b incluso nella linea precedente significa Box (cioè scatola o rettangolo). Come fa Amiga a sapere dove e quanto grande disegnare il rettangolo? E' molto semplice, le coordinate di partenza determinano l'angolo in alto a sinistra, mentre quelle finali determinano l'angolo in baso a destra, analogamente al formato utilizzato per specificare le dimensioni di una finestra. AmigaBASIC utilizza quindi lo stesso formato per determinare le coordinate di punti, linee, finestre, rettangoli ed altri oggeti grafici.

- Si può ora provare ad aggiungere l'opzione b al comando LINE del programma dimostrativo:

```
LINE -(617*RND,240*RND),MaxColori*RND,b¶
```

I risultati che si ottengono sono nuovamente abbastanza interessanti. Ma non è ancora finita: è possibile addirittura colorare questi rettangoli. Per far questo è necessario sostituire il parametro b con bf (Block Fill).

- Inserire nel programma dimostrativo.

```
LINE -(617*RND,240*RND),MaxColori*RND,bf¶
```

Non male!

A questo punto è possibile pensare di non realizzare grafica in maniera casuale, ma di riprendere il controllo e calcolare nei minimi dettagli cosa ci si accinge a fare. Un altro metodo per produrre grafica è quello di utilizzare delle formule matematiche. Anche questo metodo trova facile realizzazione in AmigaBASIC.

E' ora consigliabile salvare il programma di dimostrazione sviluppato, digitando poi NEW per cancellare la memoria. Si procederà ora ad un piccolo ripasso di alcune formule matematiche viste nelle scuole superiori, visualizzando le curve relative alle funzioni trigonometriche.

- Inserire la seguente routine:

```
FOR x=0 TO 617¶
y=90+80*SIN(x/40)¶
PSET (x,y)¶
NEXT x¶
```

SIN

La funzione SIN (*seno*) produce valori compresi tra -1 ed 1. La formula contenuta nella seconda linea consente di trasformare i valori prodotti dalla funzione seno in coordinate sullo schermo. Il valore della coordinata X viene diviso per 40 in modo tale che la curva ottenuta non sia troppo stretta. Il risultato viene moltiplicato per 80 (producendo in tal modo numeri compresi tra -80 ed 80) e viene successivamente incrementato di 90. Si ottengono così coordinate Y comprese tra 10 e 170.

Se tutto questo sembra troppo complesso, si può provare a modificare alcuni valori della formula (40, 80, 90), per poi verificare visivamente cosa accade.

Per generare una curva continua, invece di una sequenza di punti come quella attualmente prodotta, è possibile utilizzare nuovamente il comando LINE. Per prima cosa è necessario specificare il punto di partenza, in modo che la linea iniziale non parta da un punto casuale sullo schermo. Per questo motivo, nel programma viene inserito un comando PSET di inizializzazione:

```
PSET (0,90)¶
FOR x=0 TO 617¶
y=90+80*SIN(x/40)¶
LINE -(x,y)¶
NEXT x¶
```

COS

Se si utilizza la funzione COS (*coseno*) la curva risulta ovviamente diversa. SIN e COS consentono di realizzare alcuni effetti grafici interessanti. Di seguito sono riportati una serie di semplici programmi dimostrativi che illustrano cosa è possibile ottenere utilizzando tali funzioni:

```
FOR x=0 TO 617¶
y=90+80*SIN(x/40)¶
LINE (0,90)-(x,y)¶
NEXT x¶
```

Ogni punto della sinusoide viene connesso con un punto fissato ottenendo in tal modo una intricata disposizione delle linee. Ecco ora un disegno un po' più colorato:

```
Colori=4¶
SCREEN 1,640,256.Colori,2¶
WINDOW 2,"Sinusoide",,15,1¶
FOR x=0 TO 617¶
co=co+1: IF co>7 THEN co=0¶
y=90+80*SIN(x/40)¶
LINE (0,90)-(x,y),co¶
NEXT x¶
```

E' possibile effettuare degli esperimenti modificando ad esempio i valori dell'istruzione LINE:

```
LINE (320,90)-(x,y),co¶
```

oppure:

```
LINE (320,240)-(x,y),co¶
```

Si possono utilizzare anche le opzioni per realizzare box e box pieni.

```
LINE (320,240)-(x,y),co,b¶
```

L'unico limite a quello che si può fare è posto dalla propria fantasia. Si tenga presente che questi sono solamente programmi dimostrativi e che quindi è possibile modificarli a proprio piacimento senza alcun problema. Se si utilizzano formule più complicate, è possibile ottenere disegni ancor più intricati.

• Provare ad esempio le seguenti routine:

```
Colori=4¶
SCREEN 1,640,256,Colori,2¶
```

```
WINDOW 2,"Sinusoide",,15,1¶
FOR x=0 TO 617¶
co=co+1: IF co>7 THEN co=0¶
y1=90+80*SIN(x/40)¶
y2=90+70*SIN(x/60)¶
LINE (x,y1)-(617-x,y2),co¶
NEXT x¶
```

oppure:

```
Colori=4¶
SCREEN 1,640,256,Colori,2¶
WINDOW 2,"Sinusoide",,15,1¶
FOR x=0 TO 617¶
co=co+.5: IF co>7 THEN co=1¶
y1=90+80*SIN(x/40)¶
y2=90+70*SIN(x/60)¶
x2=320-300*SIN(x/50)¶
LINE (x,y1)-(x2,y2),co¶
NEXT x¶
```

Si spera che questi esempi siano abbastanza esplicativi e forniscano un buon incentivo per iniziare a sperimentare autonomamente i comandi descritti. Ciò che si rischia, nel peggiore dei casi, è di ottenere un messaggio di errore, oppure una grafica non proprio bellissima. Pian piano si apprenderà come realizzare grafica di maggior effetto.

Si può ora cercare di arricchire con queste routine il programma di video titolazione.

Nota d'uso 2: Riunire programmi differenti

Divertendosi a produrre immagini grafiche in cui vengono utilizzate sinusoidi, altre formule matematiche ed il comando LINE, si può notare come si riescano a realizzare effetti che potrebbero essere definiti stupefacenti. Non sarebbe gradevole poter utilizzare questi effetti come sfondo per il programma di videotitolazione precedentemente sviluppato? Dopo tutto, anche se gli oggetti volanti ed i testi colorati sono di grande impressione, al programma manca ancora qualcosa sullo sfondo. In questo paragrafo si vedrà come fare per sopperire a questa mancanza.

Concatenazione di programmi

Amiga presenta differenti metodi per visualizzare uno sfondo. Inizialmente verrà utilizzato un metodo semplicissimo: la routine che disegna la sinusoide verrà inserita in fondo al programma di videotitolazione. In questo modo si otterrà, durante l'esecuzione del programma, il salto a questa routine subito prima della visualizzazione del testo e degli oggetti in movimento. I grafici sinusoidali colorati prodotti in precedenza risultano abbastanza interessanti per essere utilizzati come sfondo per una sequenza titolata. Se il programma dovesse venire utilizzato per creare il titolo di un filmato su videocassetta, è comunque consigliabile cronometrare la sequenza titolata, per determinarne la durata. Dopo tutto, sarebbe abbastanza spiacevole cancellare parte del contenuto della cassetta per registrarvi sopra il titolo.

Ritornando al problema relativo all'inserimento della routine che realizza lo sfondo, appare inutile salvare e stampare il programma che realizza la sinusoide per poi ribatterlo alla fine del programma di videotitolazione. Per facilitare questo tipo di operazioni AmigaBASIC consente di collegare direttamente tra loro due programmi.

- Caricare in memoria il programma per la realizzazione della sinusoide (è probabile che si trovi già in memoria, dato che è stato l'ultimo ad essere

inserito; in caso contrario procedere al caricamento da dischetto).

E' ora necessario salvare nuovamente sul dischetto tale programma, questa volta con un formato speciale.

• Battere la seguente linea nella finestra BASIC:

```
SAVE "Videografica",a¶
```

ASCII

Il parametro posto alla fine della linea precedente specifica la richiesta di salvataggio in formato ASCII. Si è già avuto modo di spiegare questo termine: i codici ASCII sono i numeri utilizzati con il comando CHR$. La "a" che segue il comando SAVE indica ad Amiga di utilizzare i valori ASCII per salvare il programma. Viene da chiedersi: se questo è un parametro speciale per il salvataggio dei programmi, quale è il formato standard?

Normalmente, per il salvataggio dei programmi viene utilizzato il comando SAVE senza opzioni, oppure viene selezionata l'opzione Save dal menù a tendina Project. Il programma viene in questo caso salvato su dischetto in un formato speciale per la conservazione in memoria (se si desiderano informazioni relativamente a questo formato consultare l'appendice E alla voce Tokens). Tuttavia, se un programma viene salvato in questo formato, esso non può poi essere concatenato ad un altro programma. Il collegamento può essere effettuato solamente se il programma è stato salvato in formato ASCII.

Ecco un'altra possibile forma per il salvataggio di un file:

```
SAVE "Testo",p¶
```

Protezione dei file

La p viene utilizzata per identificare Protect (protetto). Tale parametro viene utilizzato per salvare un programma in modo tale che questo possa essere eseguito, ma non listato o modificato. Ad esempio, se si desidera vendere i propri programmi BASIC, l'eventuale compratore dovrebbe essere messo in condizione di eseguire ed utilizzare il programma, ma non dovrebbe essere messo in grado di esaminare i listati. E' possibile ottenere questo risultato salvando i programmi in modo protetto; quando viene

memorizzato su dischetto il programma viene codificato ed AmigaBASIC non è più in grado di decodificarlo per visualizzare nuovamente il listato. Quando tale programma viene caricato o mandato in esecuzione la finestra LIST risulta disabilitata ed è possibile solamente lanciare il programma.

Questo tipo di protezione è tuttavia facilmente superabile da un esperto che intende *sproteggere* il programma. Se un programma commerciale venisse protetto in questo modo, sarebbe molto semplice scrivere un programma di decodifica in grado di superare la protezione. Questo formato di protezione risulta quindi sufficientemente adeguato solamente per un utilizzo hobbistico del programma.

MERGE

Dopo aver brevemente accennato alla controversia relativa alla protezione o meno dei programmi, è il caso di ritornare ad applicazioni più innocenti. Una volta che il programma di video grafica è stato salvato in formato ASCII, è necessario provvedere al caricamento del programma di videotitolazione. Quando due programmi devono essere concatenati, il primo programma deve essere presente in memoria mentre il secondo viene caricato da dischetto. Il comando che realizza la concatenazione di un programma salvato in formato ASCII è molto semplice:

```
MERGE "NomeProgramma¶
```

dove la stringa NomeProgramma deve essere sostituita con il nome del programma che si desidera concatenare.

Naturalmente è ora necessario assicurarsi che la routine che disegna la sinusoide venga anche eseguita. Per prima cosa è necessario fornire a tale routine un'etichetta, ad esempio Grafica:. Se il nuovo programma contiene ancora comandi SCREEN e WINDOW è necessario provvedere a cancellarli. Il massimo numero di colori dovrebbe essere contenuto nella variabile Colori.

E' necessario altresì assicurarsi che nella routine non venga utilizzata nessuna variabile già utilizzata nel programma di videotitolazione per non incorrere in eventuali errori.

Alla fine del programma diventa ora necessario inserire un comando RETURN. Nel caso in esame la routine aggiunta dovrebbe quindi risultare la

seguente:

```
Grafica:¶
FOR x=0 TO 617¶
co=co+.5: IF co>Colori THEN co=1¶
y1=90+80*SIN(x/40)¶
y2=90+70*SIN(x/60)¶
x2=320-300*SIN(x/50)¶
LINE (x,y1)-(x2,y2),co¶
NEXT x¶
RETURN¶
```

L'unica cosa ancora mancante è il comando che effettua la chiamata di questa routine.

- Inserire quindi le linee seguenti nella routine InizioVisualizzazione del programma di videotitolazione:

```
InizioVisualizzazione:¶
WIDTH 60¶
COLOR ColoreTesto,Sfondo: CLS¶
GOSUB Grafica¶
COLOR ColoreTesto,SfondoTesto: CLS¶
```

Ora è tutto a posto: il programma può essere eseguito per poterne osservare i risultati. Se lo si desidera è ora consigliabile effettuare il salvataggio della versione aggiornata del programma con un diverso nome, in modo tale da conservare anche la versione precedente.

Con il metodo introdotto in questo paragrafo è possibile concatenare un numero qualsiasi di programmi secondo l'ordine desiderato, cosa che si rivela molto utile quando si stanno realizzando programmi molto grandi.

E' possibile inserire numerosi disegni differenti come sfondo per il proprio programma di videotitolazione. Ad esempio, è possibile consentire all'utente di scegliere tra differenti sfondi cui corrispondono diverse routine grafiche.

2.6 Tutto sui cerchi: altri comandi grafici

Il prossimo comando che si analizzerà consente di disegnare cerchi. Ecco una illustrazione preliminare delle capacità di tale comando:

```
CIRCLE (320,90),160¶
CIRCLE (260,50),15,,,,.5¶
CIRCLE (380,50),15,,,,.5¶
CIRCLE (260,53),8¶
CIRCLE (380,53),8¶
CIRCLE (380,50),45,,.2,2.2¶
CIRCLE (260,50),45,,.8,2.8¶
CIRCLE (320,80),20,,,,.7¶
CIRCLE (320,110),80,,3.1,0,.2¶
CIRCLE (320,10),10,,4.7,1.5,.9¶
CIRCLE (320,15),5,,5,1.8,.7¶
```

CIRCLE

Questo esempio non è semplice da inserire quanto i precedenti: è infatti necessario assicurarsi di includere il numero esatto di virgole. Si spera che il risultato venga apprezzato in modo tale da giustificare lo sforzo compiuto. Dopo tutto, questo programma ha un aspetto abbastanza piacevole quando viene eseguito. Se lo si desidera, si provveda a salvarlo sul dischetto per successivi utilizzi.

Fra tutti i comandi AmigaBASIC, CIRCLE è quello che dispone del maggior numero di parametri. Gli oggetti che sono stati disegnati in questo semplice esempio sono composti da cerchi, ellissi ed archi. Tutti questi oggetti grafici sono stati creati utilizzando opportunamente i parametri del comando CIRCLE.

La linea seguente mostra la forma più semplice del comando CIRCLE:

```
CIRCLE (x,y),Raggio
```

Questa istruzione consente di disegnare un cerchio con il raggio specificato avente centro nel punto di coordinate (x,y). Il valore del raggio viene fornito come numero di pixel. Tale numero, tuttavia, è valido solamente nella direzione delle X; per quanto riguarda le Y il problema sarà affrontato più avanti.

E' naturalmente possibile utilizzare anche colori differenti:

```
CIRCLE (x,y),Raggio,Colore
```

Archi ed ellissi

Con il comando CIRCLE non solo è possibile creare cerchi completi, ma anche parti di cerchio, cioè archi. Due parametri posti dopo il colore specificano l'angolo di partenza e quello finale dell'arco.

```
CIRCLE (x,y),Raggio,Colore,Angolo1,Angolo2
```

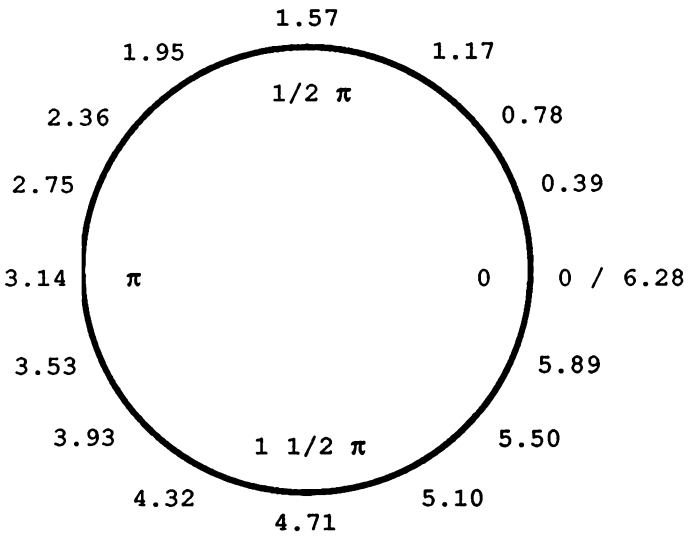


Figura 6: Rappresentazione degli angoli in radianti, cioe' come coefficienti di π.

La precedente figura è di valido aiuto nella determinazione dei valori corretti per un particolare angolo. I valori degli angoli devono essere compresi tra 0 e 2π, in quanto gli angoli sono misurati in radianti. Un quarto di cerchio parte da 0 e termina a 1,57, metà cerchio va da 0 a 3,14, tre quarti da 0 a 4,51, mentre un cerchio intero va da 0 a 6,28 (anche se per il cerchio completo questi parametri possono essere omessi).

AmigaBASIC fornisce anche un metodo per disegnare settori circolari, molto utili ad esempio per il disegno di grafici a torta. Se viene specificato un valore negativo per un angolo, il punto corrispondente viene connesso con una linea al centro del cerchio.

• Provare ad inserire la linea seguente nella finestra BASIC:

```
CIRCLE (320,100),200,3,-1.57,-3.14¶
```

Il settimo ed ultimo parametro determina se il cerchio debba essere compresso in un'ellisse. Esso specifica infatti la relazione che intercorre tra il raggio lungo la direzione x e lungo la direzione y.

```
CIRCLE (x,y),Raggio,Colore,Angolo1,Angolo2,Rapp. x/y
```

Se si desidera disegnare un cerchio perfettamente rotondo il rapporto x/y dipenderà dalla regolazione del proprio monitor. Una delle manopoline del monitor di Amiga determina infatti l'altezza dell'immagine. Se la regolazione è quella standard, il cerchio dovrebbe essere visualizzato come un ovale. Un valore di 0.5 per il rapporto x/y dovrebbe essere quello corretto nel caso di regolazioni normali. Tale valore risulta essere anche quello utilizzato da AmigaBASIC nel caso in cui non venga specificato questo parametro. Valori minori di 0.5 causano una compressione orizzontale del cerchio in un'ellisse. Valori maggiori di 0.5 producono una compressione verticale del cerchio.

Dopo aver assimilato queste nuove nozioni, è giunto il momento di addentrarsi nel listato del programma di dimostrazione.

Realizzare una piccola faccia stilizzata
La prima linea contenente il comando CIRCLE:

```
CIRCLE (320,90),160
```

disegna un cerchio con centro nel punto (320,90) ed avente un raggio di 160 pixel.

Le due linee seguenti:

```
CIRCLE (260,50),15,,,,.5
CIRCLE (380,50),15,,,,.5
```

disegnano i due occhi: vengono infatti visualizzate due ellissi leggermente a forma d'uovo, in quanto il valore del rapporto x/y utilizzato è uguale al valore normale.

La quarta e la quinta istruzione:

```
CIRCLE (260,53),8
CIRCLE (380,53),8
```

disegnano le pupille all'interno dei due occhi.

Le due linee seguenti disegnano invece le sopracciglia:

```
CIRCLE (380,50),45,,.2,2.2
CIRCLE (260,50),45,,.8,2.8
```

In questo ultimo caso è stato utilizzato un cerchio con un raggio di 45 pixel. Tuttavia sono stati utilizzati solamente segmenti dei due cerchi (archi) con angoli di partenza di 0.2 e 0.8 e angoli finali di 2.2 e 2.8. Per valutare la posizione degli angoli riferirsi alla figura 6.

Ecco ora la descrizione del procedimento seguito per disegnare il naso; l'istruzione:

```
CIRCLE (320,80),20,,,,.7
```

realizza un cerchio posizionato esattamente al centro della faccia, di forma leggermente ellittica.

```
CIRCLE (320,110),80,,3.1,0,.2
```

disegna un semicerchio da π a 0 (dove zero è equivalente a 2π) per realizzare il sorriso.

Gli ultimi due comandi disegnano una specie di capigliatura:

```
CIRCLE (320,10),10,,4.7,1.5,.9
CIRCLE (320,15),5,,5,1.8,.7
```

Questi due archi hanno la forma di un'ellisse verticale. Si consiglia nuovamente di riferirsi alla figura 6 per determinare la posizione degli angoli.

Potrebbe ora nascere il desiderio di seguire passo per passo la costruzione della faccia sullo schermo. Sfortunatamente la velocità di Amiga rende questo desiderio molto difficile da realizzare. In questo caso neanche la modalità Trace si rivela di grande aiuto. Vi è però un'altra opzione del menù a tendina Run che consente di semplificare la fase di test di un programma. L'opzione Step del menù Run specifica infatti ad Amiga di eseguire solamente una istruzione per volta effettuando una pausa prima di passare all'esecuzione del comando successivo. Amiga aspetta fino a quando non viene nuovamente selezionata l'opzione Step, o fino a quando non viene premuta la combinazione di tasti <Amiga destro> <T>.

L'opzione Step del menù Run non ha un corrispondente comando in BASIC. Si può provare ad utilizzare tale opzione con il programma che disegna la faccetta. Ogni volta che viene selezionata l'opzione Step, o premuta la corrispondente combinazione di tasti, viene disegnato un nuovo pezzetto di faccia. In questo modo risulta possibile vedere come il disegno viene costruito passo per passo ed osservare l'ordine in cui vengono eseguiti i vari comandi. Se si guarda attentamente la finestra LIST, è possibile notare come ogni comando venga evidenziato nel momento in cui viene eseguito, esattamente come nella modalità Trace. La combinazione di tasti <Amiga> <.> consente invece di disabilitare la modalità Step.

PAINT

Sia che venga eseguito velocemente che lentamente, il disegno realizzato da questo programma, ricorda più che altro uno scarabocchio su una lavagna scolastica. Per renderlo più consono alle capacità grafiche di Amiga è necessario utilizzare un altro comando.

Il comando PAINT consente di colorare una zona racchiusa tra linee. E' possibile modificare il listato precedente, in modo tale che esso risulti identico a quello che segue. Ricordarsi di salvare la nuova versione subito dopo aver finito di effettuare le modifiche.

```
CIRCLE (320,190),200,1,,,.6¶
PAINT (320,170),1¶
CIRCLE (320,90),160,3¶
PAINT (320,90),3¶
CIRCLE (260,50),15,1,,,.5¶
CIRCLE (380,50),15,1,,,.5¶
PAINT (260,50),1¶
PAINT (380,50),1¶
CIRCLE (260,53),8,2¶
CIRCLE (380,53),8,2¶
PAINT (260,53),2¶
PAINT (380,53),2¶
CIRCLE (380,50),45,2,.2,2.2¶
CIRCLE (260,50),45,2,.8,2.8¶
CIRCLE (320,80),20,2,,,.7¶
CIRCLE (320,110),80,2,3.1,0,.2¶
CIRCLE (320,10),10,2,4.7,1.5,.9¶
CIRCLE (320,15),5,2,5,1.8,.7¶
CIRCLE (160,70),40,3,,,.6¶
CIRCLE (480,70),40,3,,,.6¶
PAINT (160,70),3¶
PAINT (480,70),3¶
CIRCLE (160,60),30,2,1,3,.3¶
CIRCLE (480,60),30,2,.3,2.2,.3¶
CIRCLE (150,65),25,2,4,1,.7¶
CIRCLE (490,65),25,2,2.2,5.4,.7¶
```

Colorare la faccia

Il pupazzetto, oltre ad avere due orecchie e le spalle larghe, ha ora anche una faccia più colorita.

Se si esamina attentamente la costruzione di questo disegno, si ricava un'idea della modalità di azione del comando PAINT: specificato un punto all'interno di un'area completamente chiusa ed un colore, AmigaBASIC provvede a riempire l'intera area compresa nei contorni del disegno con quel colore. Il chip Blitter, dedicato in particolare a muovere, riempire o

modificare oggetti grafici, consente di ottenere una velocità molto elevata nell'esecuzione del comando PAINT.

Si noti una caratteristica importante di questo comando: PAINT riconosce la linea di contorno solamente se essa è disegnata con un colore particolare. Nel semplice programma precedente è possibile notare come venga innanzitutto disegnato e colorato il corpo, seguito dal disegno e dalla colorazione della testa che ricopre parte del corpo. E' possibile definire il colore da considerare come contorno per l'area da pitturare con la seguente sintassi:

```
PAINT (x,y),colore,colore_bordo
```

Se non viene specificato un colore di contorno, il comando assume che questo coincida con il colore utilizzato per l'area interna. Il comando:

```
PAINT (320,90),3
```

realizza quindi la colorazione dell'area in cui è contenuto il punto (320,90) racchiusa tra linee di contorno arancione. Nel caso si desideri utilizzare il bianco come colore di contorno (colore 1) ed il nero come colore per l'area interna (colore 2), il comando assume la forma seguente:

```
PAINT (x,y),2,1
```

E' opportuno tenere sempre presente questo aspetto del comando PAINT quando lo si utilizza, per non ottenere risultati indesiderati.

Alcune considerazioni sul comando PAINT

Le linee di contorno per l'area da colorare utilizzando PAINT possono essere disegnate con un qualsiasi comando grafico. Non è necessario che la linea di contorno costituisca una particolare figura geometrica: è fondamentale, invece, che non vi siano buchi lungo il contorno. In caso contrario il colore fuoriesce dall'area desiderata, espandendosi nelle aree contigue. Come si può ricordare, lo stesso problema era già stato riscontrato durante l'utilizzo dell'Object Editor. Questo è abbastanza normale, dato che l'Object Editor altro non è che un programma scritto in BASIC che utilizza il comando PAINT.

Il programma che segue mostra un metodo alternativo per disegnare i contorni. Esso utilizza il comando LINE per disegnare una griglia le cui parti

vengono riempite casualmente con colori differenti.

Ecco il listato:

```
FOR x=0 TO 615 STEP 15¶
LINE (x,0)-(x,185)¶
NEXT x¶
FOR Y=0 TO 180 STEP 10¶
LINE (0,y)-(635,y)¶
NEXT y¶
Riempimento:¶
PAINT (635*RND,180*RND)¶
GOTO Riempimento¶
```

In questa porzione di listato compare un comando nuovo: il comando STEP. Utilizzando questo comando è possibile modificare il valore utilizzato per il passo (incremento) di un ciclo. L'istruzione:

```
FOR x=0 TO 615 STEP 15
```

realizza incrementi di 15 in 15 partendo da zero fino a raggiungere 615. (0, 15, 30, 45, etc.) Nel programma STEP viene utilizzato per assicurare la spaziatura corretta tra le linee della griglia. Se lo si desidera si può provare a modificare il valore per il passo ed osservare cosa accade.

Piramidi in BASIC

Ora che si conoscono i comandi BASIC con cui disegnare punti, linee, rettangoli, cerchi, ellissi ed archi ed il comando per colorare aree chiuse, è possibile addentrarsi ulteriormente nell'analisi dei comandi grafici. AmigaBASIC fornisce infatti altri strumenti per realizzare grafica al calcolatore. Come detto in precedenza, il chip Blitter è in grado di riempire aree di schermo in tempi incredibilmente rapidi. Il comando BASIC che ci si accinge ad analizzare utilizza questa caratteristica per trarne il massimo vantaggio. Ecco un altro programma dimostrativo da provare:

```
COLOR 3,0¶
AREA (100,150)¶
AREA (400,150)¶
AREA (250,20)¶
AREAFILL¶
```

```
COLOR 2,0¶
AREA (100,150)¶
AREA (400,150)¶
AREA (200,180)¶
AREAFILL¶
COLOR 1,0¶
AREA (250,20)¶
AREA (400,150)¶
AREA (450,100)¶
AREAFILL¶
```

Come è possibile vedere, si può realizzare l'illusione ottica di una piramide disegnando tre piccoli triangoli di colore diverso.

AREA e AREAFILL

Con il comando AREA non è possibile creare solamente triangoli: utilizzando più comandi AREA possono infatti essere specificati i vertici di qualsiasi poligono. Amiga mantiene le informazioni relative ai punti specificati fino a quando non incontra un comando AREAFILL. In seguito a questo comando l'oggetto specificato viene disegnato sullo schermo pressochè istantaneamente. Vi è tuttavia una restrizione: AmigaBASIC non è in grado di ricordare più di 20 punti, per cui ogni punto specificato dopo i primi 20 a partire dall'ultimo comando AREAFILL viene ignorato. Un poligono avente 20 vertici costituisce comunque un oggetto grafico già abbastanza complesso che può essere disegnato in un sol colpo.

Se si stabiliscono i vertici del poligono utilizzando la funzione RND è possibile ottenere, come già per gli esempi precedenti, una grafica abbastanza interessante. Prima di addentrarsi in altri esempi è comunque opportuno salvare il programma che disegna le piramidi.

Il programma seguente, che presenta una struttura abbastanza semplice, costituisce un valido esempio:

```
Colori =4¶
SCREEN 1,640,256,Colori,2¶
WINDOW 1,"Aree",,15,1¶
Inizio:¶
COLOR ((2^Colori)-1)*RND,0¶
```

```
FOR x=1 TO 3+17*RND¶
AREA (617*RND,240*RND)¶
NEXT x¶
AREAFILL¶
GOTO Inizio¶
```

La prima azione compiuta è quella di selezionare in modo casuale un colore dall'insieme dei colori disponibili. Il ciclo FOR ... NEXT viene eseguito un numero di volte compreso tra 3 e 20 (il numero minimo di esecuzioni è 3, in quanto questo è il numero di vertici minimo per creare un'area visibile, dato che due punti consentono di generare solamente una linea). Dopo aver disegnato il poligono, il programma ritorna ad eseguire la routine etichettata Inizio:, ripetendo così la stessa procedura.

Dato che tutti i vertici sono scelti casualmente, è possibile ottenere effetti abbastanza piacevoli ed originali. Chiaramente i punti per il comando AREA possono essere determinati anche da equazioni matematiche, come nel seguente esempio:

```
Colori =4¶
SCREEN 1,640,256,Colori,2¶
WINDOW 1,"Aree",,15,1¶
FOR x=0 TO 617¶
co=co+1: IF co>(2^Colori)-1 THEN co=0¶
COLOR co,0¶
y1=90+80*SIN(x/49)¶
y2=90+70*COS((617-x)/25)¶
x2=ABS(320-x)¶
AREA (x,y1)¶
AREA (x2,y2)¶
AREA (x,x/4)¶
AREA (320,90)¶
AREAFILL¶
NEXT x¶
```

Con l'illustrazione dei comandi AREA ed AREAFILL si conclude l'analisi dei principali comandi grafici. Nel paragrafo seguente tutti i comandi visti verranno utilizzati per realizzare un programma più complesso. In questo modo, insieme agli esempi esplicativi necessari per la completa illustrazio-

ne dei comandi, verrà fornita anche una possibile applicazione pratica di quanto spiegato. Per prima cosa ci si addentrerà un pò nella statistica, realizzando un programma abbastanza utile per chi si occupa di affari e finanza.

2.7 Un programma per la visualizzazione di istogrammi e grafici a torta

Istogrammi e grafici a torta vengono solitamente utilizzati per la presentazione di risultati elettorali, per illustrare la ripartizione di uno stanziamento tra le varie voci di spesa, oppure per mostrare visivamente dati percentuali. Qualunque sia l'applicazione a cui sono rivolti essi costituiscono un buon modo per visualizzare informazioni statistiche.

Grafici a torta
Come suggerito dal nome stesso, i *grafici a torta* sono realizzati suddividendo un cerchio (e quindi schematicamente una torta) in settori (fette). La dimensione di ogni singola fetta corrisponde ad una certa percentuale sul totale. Qualunque sia il dato che deve essere mostrato (percentuali nella bilancia dei pagamenti, composizione del Parlamento per membri dei vari partiti, profitti realizzati, etc.) i grafici a torta aiutano a visualizzare i valori numerici consentendo inoltre una migliore comprensione globale delle informazioni.

Istogrammi (grafici a barre)
Un altro metodo per visualizzare graficamente dati statistici è quello di utilizzare *grafici a barre.* Un istogramma è composto da barre verticali di altezza diversa con cui vengono rappresentati i differenti valori. I grafici a barra si rivelano molto utili nella visualizzazione delle relazioni che intercorrono tra varie informazioni numeriche. E' possibile utilizzarli per mostrare l'ammontare delle vendite mensili, oppure l'incremento del tasso di inflazione, etc.

Il programma
E' possibile creare grafici a torta ed istogrammi con Amiga utilizzando i comandi grafici BASIC precedentemente illustrati. Il programma seguente

accetta una serie di dati statistici e li visualizza in forma grafica:

```
Grafici:¶
  IF MatriceNum(0)=0 THEN RETURN ¶
  IF UCASE$(MatriceStr$(0))="B" THEN
     GOSUB GraficoBarre¶
  IF UCASE$(MatriceStr$(0))="T" THEN
     GOSUB GraficoTorta¶
RETURN¶
¶
GraficoTorta:¶
  Totale=0¶
  FOR x=1 TO MatriceNum(0)¶
    Totale=Totale+MatriceNum(x)¶
  NEXT x¶
  Divisione=Totale/6.283 : Angolo1=.0001 : ColoreB=1¶
  FOR x=1 TO MatriceNum(0)¶
    ColoreL=ColoreB¶
    IF ColoreL>(2^Colori)-1 THEN ColoreL=1¶
    ColoreB=ColoreL+1¶
    IF ColoreB>(2^Colori)-1 THEN ColoreB=1¶
    Angolo2=Angolo1+MatriceNum(x)/Divisione¶
    CIRCLE (320,98),154,ColoreB ¶
    CIRCLE (320,98),148,ColoreB,-Angolo2,-Angolo1¶
    PAINT (320,31),ColoreL,ColoreB¶
    CIRCLE (320,98),148,ColoreB¶
    PAINT (320,32),0,ColoreB¶
    CIRCLE (320,98),148,ColoreB,-Angolo1,-Angolo2¶
    AngoloMediano=(Angolo1+Angolo2)/2¶
    px=320+165*COS(AngoloMediano)¶
    py=100-80*SIN(AngoloMediano)¶
    Distanza=0¶
    IF AngoloMediano>1.57 AND AngoloMediano<4.72 THEN
      Distanza=LEN(MatriceStr$(x))¶
    IF Distanza>15 THEN Distanza=15¶
    COLOR ColoreL,0¶
    LOCATE (py/8.4)+1,(px/8.05)+1-Distanza¶
```

```
    PRINT MatriceStr$(x);¶
    Angolo1=Angolo2¶
  NEXT x¶
  ¶
  CIRCLE (320,98),154,0¶
RETURN¶
¶
GraficoBarre:¶
  Max=.0001 : ColoreL=0¶
  FOR x=1 TO MatriceNum(0)¶
    IF MatriceNum(x)>Max THEN Max=MatriceNum(x)¶
  NEXT x¶
  LarghezzaBarra=INT(550/(MatriceNum(0)))¶
  IF LarghezzaBarra>100 THEN LarghezzaBarra=100¶
  Fattore=160/Max¶
  LOCATE 1,1 : PRINT Max;¶
  LOCATE 10,1 : PRINT Max/2;¶
  FOR x=0 TO 10¶
    LINE (1,170-x*16)-(5,170-x*16)¶
  NEXT x ¶
  FOR x=1 TO MatriceNum(0)¶
    ColoreL=ColoreL+1 : IF ColoreL>(2^Colori)-1
      THEN ColoreL=1¶
    LINE (30+(x-1)*LarghezzaBarra,170-MatriceNum(x)*
      Fattore)-(25+x*LarghezzaBarra,170),ColoreL,bf¶
    COLOR ColoreL,0¶
    LOCATE 23,(4+(x-1)*(LarghezzaBarra/7.7))¶
    PRINT MatriceStr$(x);¶
  NEXT x¶
RETURN¶
```

Dopo aver inserito il testo del programma si consiglia di provvedere a salvarlo su dischetto.

Strutturazione del codice

E' possibile notare come il listato appena presentato differisca da quelli

precedentemente visti. Ad alcune linee sono infatti stati premessi alcuni spazi per evidenziare anche visivamente la struttura del programma. Tale procedura viene detta indentazione. La maggior parte dei programmi BASIC risulta infatti molto difficile da leggere e comprendere, spesso anche da parte dello stesso autore del programma; e questo non è altro che il risultato di una cattiva *strutturazione del programma.*

Anche se AmigaBASIC fornisce tutti gli strumenti necessari per strutturare i programmi (etichette, subroutine, etc.) è talvolta necessario strutturare anche visivamente i programmi. Se si desidera comprendere la sequenza di esecuzione delle istruzioni di un programma direttamente dal listato, una forma come quella mostrata sopra risulta certamente più utile e comprensibile rispetto a quelle viste fino ad ora. Queste considerazioni non sono state esposte precedentemente, in quanto questo è il primo programma completo inserito in una sola volta.

La regola di base per strutturare i programmi è molto semplice: le linee che compongono una subroutine, un ciclo od una qualsiasi parte logicamente a sè stante del programma, vengono indentate di uno o due spazi a destra. L'indentazione termina dove finisce ogni singola unità.

Se si desidera aumentare ulteriormente la strutturazione è possibile inserire linee bianche in corrispondenza dell'inizio e della fine delle varie strutture.

Utilizzo del programma

Allo stato attuale il programma non è utilizzabile, in quanto esso rappresenta solamente una subroutine. Per effettuare i primi esperimenti è necessario inserire, all'inizio del programma battuto precedentemente, le linee che seguono.

• Spostare il cursore all'inizio del programma ed inserire:

```
Inizializzazione:¶
  Colori=3¶
  SCREEN 4,640,200,Colori,2¶
  WINDOW 99,"Grafici",,8,4¶
  PALETTE 7,.8,.2,.1¶
  DIM MatriceNum(50),MatriceStr$(50)¶
```

```
ControlloImmissione:¶
  WINDOW 1¶
  INPUT "B=barre, T=torta: ",MatriceStr$(0)¶
Immissione:¶
  MatriceNum(0)=MatriceNum(0)+1¶
  PRINT "Numero del valore "MatriceNum(0)":";¶
  INPUT MatriceStr$MatriceNum(0))¶
  PRINT "Titolo:";¶
  INPUT MatriceStr$(MatriceNum(0))¶
  IF MatriceStr(MatriceNum(0))<>"" THEN Immissione¶
Salto:¶
  MatriceNum(0)=MatriceNum(0)-1¶
  WINDOW 99¶
  GOSUB Grafici¶
END¶
```

Si consiglia di salvare questa nuova versione del programma con un nome differente da quella originale, in modo tale da avere entrambe le versioni su dischetto.

Una prima esecuzione

Se si desidera testare il programma, è innanzitutto necessario decidere se si vogliono visualizzare i dati statistici con grafico a torta oppure a barre.

- Effettuare un click nella finestra e battere <B> se si desidera un grafico a barre, <T> se si vuole ottenere un grafico a torta, seguiti dal tasto <Return>.

Il programma richiede di specificare un primo valore (ad esempio 850). Dopo aver fornito questo valore viene richiesto un nome per la sua identificazione (ad esempio Amiga). Questo nome verrà successivamente visualizzato sotto la barra o vicino al settore di torta relativo al valore corrispondente. La procedura viene ripetuta per acquisire gli altri dati da rappresentare. La seconda volta si può inserire ad esempio il valore 400 e, come nome identificativo, Atari; la terza volta si può pensare di inserire 1200 ed Altri. Per terminare l'acquisizione è sufficiente premere due volte consecutivamente il tasto <Return>. In questo caso i dati inseriti (visualizzati nel grafico prescelto) potrebbero riferirsi alle vendite mensili dei vari

computer. Ovviamente i numeri inseriti sono semplicemente simbolici e non hanno validità statistica nè corrispondenza con l'effettiva realtà.

Utilizzo del programma

La subroutine necessita di tre informazioni deducibili dal programma principale: per prima cosa richiede informazioni relative al numero di bitplane, valore contenuto nella variabile Colori. I numeri che devono essere visualizzati nell'immagine sono contenuti nel vettore MatriceNum(x). Questo vettore può avere un qualsiasi numero di elementi, ma se si desidera comprendere al meglio il grafico, è consigliabile limitare a 50 il massimo numero possibile di elementi. Il numero di valori è memorizzato nella variabile MatriceNum(0). Se tale valore è 5, vengono utilizzati i dati contenuti in MatriceNum(1), MatriceNum(2), MatriceNum(3), MatriceNum(4) e MatriceNum(5). E' possibile introdurre un qualsiasi valore per i dati che devono essere rappresentati nel grafico, in quanto la subroutine converte automaticamente i valori forniti in valori percentuali. Questo significa che non è necessario assicurarsi che la somma dei valori forniti risulti essere 100.

I nomi identificativi dei vari valori sono contenuti nel vettore MatriceStr$(x). Per ogni valore contenuto in MatriceNum(x), esiste il corrispondente identificatore in MatriceStr$(x). L'identificatore per MatriceNum(1) è quindi contenuto in MatriceStr$(1), etc. La variabile MatriceStr$(0) contiene il carattere identificativo del tipo di grafico scelto (nel caso si sia selezionato un grafico a barre tale carattere risulterà essere una B; nel caso di un diagramma a torta si avrà invece una T). Se viene inserito un qualsiasi altro carattere, il programma termina senza effettuare alcuna visualizzazione.

La routine non necessita di alcun altro valore oltre a quelli mostrati. Se si desidera scrivere un proprio programma di gestione dati, è necessario definire lo schermo e la finestra in cui si desidera che compaia il grafico. Prima che venga chiamata la subroutine, mediante il comando seguente viene definita come finestra di output la finestra desiderata:

```
WINDOW numero_finestra
```

La subroutine viene poi richiamata dalla linea:

```
GOSUB Grafici
```

che può essere seguita da ulteriori routine.

Per fornire un ulteriore chiarimento riguardo al significato delle variabili, nella tabella seguente viene riportata una lista completa delle variabili e del valore ad esse assegnato nell'esempio discusso.

Nome	Contenuti	Note
Colori	4	numero di bitplane per lo schermo Workbench
Matrice(0)	3	numero di elementi in MatriceNum e MatriceStr$
MatriceNum(1)	850	primo numero
MatriceNum(2)	400	secondo numero
MatriceNum(3)	1200	terzo numero
MatriceStr$(0)	B	T = torta B=barre
MatriceStr$(1)	Amiga	etichetta primo numero
MatriceStr$(2)	Atari	etichetta secondo numero
MatriceStr$(3)	altri	etichetta terzo numero

Tabella 5: variabili di trasferimento nel programma per la realizzazione di grafici a barre ed a torta.

Come opera il programma

La descrizione precedente ha evidenziato la modalità di interazione con il programma. Verrà ora discusso come funziona il nucleo del programma. Se in qualche punto le cose sembrano essere un pò troppo complesse, è necessario ricordare che il programma non deve essere pienamente compreso per essere utilizzato.

Inizializzazione

Nella routine Inizializzazione, viene per prima cosa fissato il numero di bitplane; viene poi aperto uno schermo 640x200 con il numero di colori

desiderato, e viene infine aperta una finestra cui viene assegnato il numero 99 come identificatore. E' stato scelto un numero così alto in modo tale da evitare problemi nel caso si vogliano apportare delle modifiche al programma.

Colori
Il comando PALETTE viene utilizzato per specificare il colore: è possibile modificare tutti i colori, se lo si desidera. Come si può ricordare il comando PALETTE lavora solamente con lo schermo in cui è posizionata la finestra corrente di output.

Nel listato mostrato viene ridefinito il colore 7 in quanto, all'accensione di Amiga questo colore presenta gli stessi valori del colore 1 (cioè del bianco). Se il grafico a torta presenta sette settori (e quindi richiede sette colori) e, se il settore visualizzato con il colore 1 si trova vicino a quello di colore 7, risulterà impossibile distinguere i due settori. Per questo motivo il colore 7 viene modificato per ottenere un rosso. Successivamente vengono dimensionati i vettori MatriceNum e MatriceStr$ a 50 elementi.

Controllo dell'input
In questi due vettori vengono inseriti i valori acquisiti all'interno della routine ControlloImmissione fino a quando non viene ricevuto un input vuoto con cui si stabilisce la fine degli input.

La routine Salto analizza il contenuto di MatriceNum(0), cioè il numero di elementi inizializzati; la finestra 99 viene poi designata come finestra di output e viene infine richiamata la subroutine grafica. Al termine di questa subroutine viene terminato il programma.

Il disegno dei grafici
La subroutine comincia con la routine Grafici e controlla, prima di tutto, se è stato fornito un valore di input. Se non è presente alcun valore, si ha il ritorno all'inizio del programma. Il valore di MatriceStr$(0) consente di stabilire se deve essere realizzato un diagramma a torta o un grafico a barre. L'utilizzo della funzione UCASE$ rende assolutamente irrilevante il fatto che tale valore sia una lettera maiuscola o minuscola. Dall'interno della subroutine, viene poi richiamata un'altra subroutine che consente di dise-

gnare il grafico del tipo specificato.

Disegnare una torta

Analizzando la subroutine GraficoTorta è possibile notare come le prime quattro linee sommino tra loro tutti i valori forniti per determinare il totale (cioè il valore corrispondente a 100%). Ognuno dei valori immesso viene poi diviso per una costante allo scopo di calcolare le dimensioni degli archi corrispondenti. Il numero così calcolato viene assegnato alla variabile Divisione. Il numero 6.283 corrisponde a 2*π cioè al massimo angolo possibile. Il valore di partenza di Angolo1 viene posto leggermente sopra a 0, in quanto il comando CIRCLE non può riconoscere la differenza tra +0 e -0, per cui se ci fosse un angolo di 0, il primo settore non verrebbe connesso correttamente con il centro. Come colore per il bordo (ColoreB) viene utilizzato il colore 1 (bianco).

Colorazione del grafico

Il ciclo FOR ... NEXT che segue contiene le istruzioni per la realizzazione del disegno e viene eseguito per tutti i valori da presentare nel grafico.

All'inizio del ciclo il colore per le linee di separazione (ColoreL) assume il vecchio valore di ColoreB. Se il numero di colori supera il massimo consentito, tale valore riparte da uno. Non viene utilizzato il colore 0 in quanto esso corrisponde al colore di sfondo. Il colore di bordatura viene poi incrementato di una unità. Quale è il significato di questi due colori? Come è possibile ricordare, il comando PAINT riconosce come linee di contorno solamente quelle disegnate in un colore particolare. Dato che si desidera colorare le varie parti di torta in colori differenti è necessario utilizzare un colore per le linee ed uno per i bordi.

Il metodo utilizzato per colorare le diverse aree non è particolarmente elegante, ma è comunque abbastanza pratico. Talvolta la praticità deve avere la precedenza. Per colorare un'area, PAINT richiede che venga specificato un punto interno ad essa. Dato che i settori sono di dimensioni differenti, non è semplice determinare tale punto.

Per risolvere questo problema vengono utilizzati, nel disegno del grafico a torta, due cerchi concentrici, il primo con un raggio di 154 pixel, il secondo con un raggio leggermente inferiore (148 pixel). Il colore di bordo

protegge quelle parti di cerchio che sono già state completate. L'area tra i due cerchi viene colorata utilizzando il comando PAINT e, durante tale processo, il colore riempie anche il cerchio più interno. Ecco il motivo per cui viene utilizzato come colore per il bordo un colore superiore di una unità al colore delle aree interne. Alla fine della routine il cerchio più esterno viene cancellato. Se si osserva molto attentamente l'esecuzione del programma, si dovrebbe essere in grado di notare questo processo mentre viene visualizzato il grafico.

Calcolo degli angoli

Si può ora vedere come il programma metta in pratica i concetti in precedenza evidenziati dal punto di vista teorico.

La quinta linea del ciclo calcola il valore che deve essere assunto dalla variabile Angolo2 (il maggiore fra i valori per la bordatura). Tale valore risulta essere la somma tra l'angolo di partenza e la dimensione dell'arco corrispondente allo specifico valore. Per prima cosa viene disegnato il cerchio esterno, con il colore di bordo corrente; viene poi tracciata la linea di contorno per proteggere la parte di grafico già realizzata. Il comando PAINT della linea successiva effettua la colorazione vera e propria (con colore ColoreL) utilizzando come bordo linee disegnate in ColoreB. Il punto di partenza si trova esattamente nell'area compresa tra i due cerchi concentrici. Dopo che le aree sono state colorate, l'intero cerchio interno viene contornato con il colore di bordo e l'area tra i due cerchi viene cancellata con il colore di sfondo. Per concludere, i vari settori vengono separati tra loro utilizzando il colore di bordo.

Aggiungere testo

I restanti comandi hanno principalmente uno scopo: visualizzare i nomi appropriati vicino ai settori corrispondenti. E' necessario utilizzare nuovamente un piccolo trucchetto per fare questo.

Per prima cosa viene calcolato l'angolo che divide esattamente in due parti uguali il settore circolare, angolo chiamato AngoloMediano. Ma come è possibile determinare un punto avendo a disposizione solamente un raggio ed un angolo? La figura 7 sintetizza abbastanza bene il problema da risolvere:

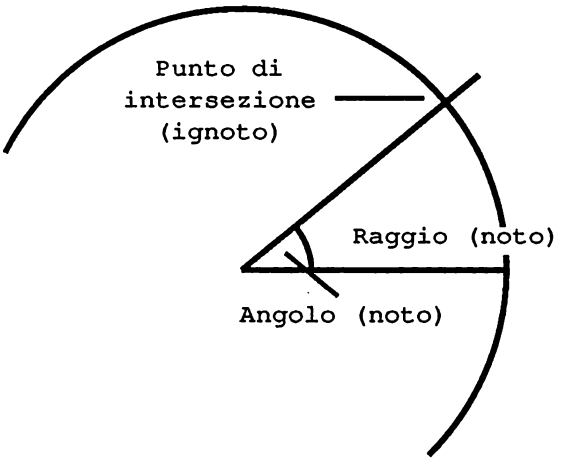


Figura 7: Determinazione del punto di intersezione tra angolo ed arco

La soluzione di tale problema è possibile ricorrendo ad un'equazione trigonometrica.

Le funzioni SIN e COS possono essere utilizzate non solo per realizzare simpatici effetti grafici al calcolatore, ma anche per effettuare utili calcoli. Il raggio, moltiplicato per il coseno dell'angolo, fornisce la coordinata x del punto cercato, mentre moltiplicando il raggio per il seno si ottiene la coordinata y. E' ora necessario convertire queste coordinate in relazione allo schermo attualmente utilizzato ed alla posizione dei vari oggetti sullo schermo (il centro del cerchio è (320,100)), per ottenere px e py.

Ecco un ulteriore problema: se il testo è posizionato a destra del grafico, può essere lungo quanto si vuole: nella peggiore delle ipotesi, sborderà dal lato destro dello schermo. Per quanto riguarda invece gli identificatori posti a sinistra, si ha che essi si sovrappongono ai settori dopo pochi caratteri, cosa chiaramente indesiderata. La linea:

```
Distanza=0
```

consente di lasciare lo spazio necessario per il testo.

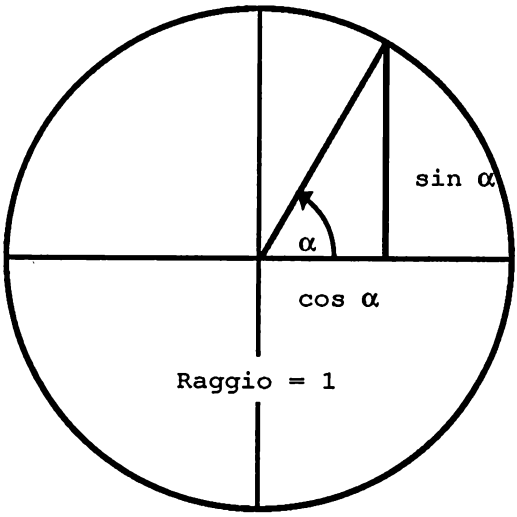


Figura 8: Definizione delle funzioni trigonometriche Seno e Coseno

LEN
Quando un angolo è compreso tra 1.57 (1/2*π) e 4.72 (3/2*π) il nome viene visualizzato a sinistra del grafico. In questo caso viene calcolata la lunghezza della stringa ed il testo viene spostato a sinistra in dipendenza di questa lunghezza. L'ultima lettera viene visualizzata immediatamente a sinistra del cerchio. Ecco un nuovo comando AmigaBASIC: LEN(a$). Questo comando consente di contare il numero di caratteri contenuti in una stringa.

Ad esempio:

```
LEN ("Ciao")
```

risulta essere 4, e

```
LEN ("anticiclonico")
```

risulta 13.

Se la distanza dal cerchio del primo carattere della stringa da scrivere ri-

sulta essere maggiore di 15, essa viene automaticamente ridotta a 15, in quanto non vi è ulteriore spazio sullo schermo.

Da ultimo le coordinate px e py vengono convertite in righe e colonne, per poter essere utilizzate dal comando LOCATE. Il testo viene infine visualizzato sullo schermo.

Questo programma è realizzato per funzionare a 80 caratteri per linea, per cui è necessario selezionare questo valore nelle Preferences.

Per l'esecuzione successiva del ciclo, all'angolo di partenza (Angolo1) è stato assegnato il valore dell'attuale angolo finale (Angolo2). L'intera procedura viene poi ripetuta in riferimento al nuovo settore. Come già ricordato, non appena sono stati disegnati tutti i settori, il cerchio più esterno viene cancellato. dopo di che l'esecuzione prosegue a partire dalla linea successiva a quella di chiamata della subroutine.

Se non si è riusciti a comprendere completamente questa procedura, non è il caso di preoccuparsi, non è assolutamente importante.

Le restanti equazioni

La suboutine GraficoBarre non è difficile da comprendere. Le prime quattro linee determinano il valore massimo fra quelli introdotti. Questo valore viene memorizzato nella variabile Max. Il più piccolo valore possibile per Max è 0.0001, in quanto la divisione per zero non è permessa e nella settima linea il numero 160 viene diviso per Max.

• Battere:

```
? 1/0¶
```

Come si può vedere Amiga visualizza il messaggio d'errore "Division By Zero".

La variabile LarghezzaBarra contiene la larghezza delle singole barre. Tale valore dipende dal numero totale di barre che devono essere visualizzate nel grafico. LarghezzaBarra, comunque, non può essere più grande di 100. Quello che si sta realizzando, infatti, è un grafico a barre, ed una barra che riempia tutto uno schermo non ha alcun senso.

La variabile Fattore contiene il numero per il quale vengono moltiplicati i vari valori per ottenere le altezze corrette. La barra del massimo risulta alta 160 pixel; tutte le altre barre risultano più basse.

Visualizzazione della scala
In corrispondenza del bordo sinistro dello schermo viene mostrata una scala per l'altezza delle barre. Il valore massimo ed i valori intermedi sono visualizzati in corrispondenza di 11 tratti di incremento. Le ultime sette linee della subroutine contengono il ciclo che consente di disegnare le singole barre. Esattamente come nel caso del grafico a torta, tale ciclo viene eseguito fino a quando non sono visualizzati tutti i valori. Il comando LINE viene utilizzato per costruire la barra sullo schermo. Ogni barra assume un colore differente tra quelli disponibili. Ancora una volta non viene utilizzato il colore di sfondo.

Dopo le barre viene visualizzato il nome identificativo di ogni valore, utilizzando i comandi LOCATE e PRINT. La posizione del testo viene determinata in dipendenza delle dimensioni delle barre. L'intero processo viene ripetuto per ogni barra, dopo di che il programma ritorna al punto di chiamata della subroutine.

Ecco finalmente conclusa questa sezione in cui vi erano numerosi concetti matematici. E' possibile effettuare un pò di esperimenti con il programma appena realizzato.

Nel corso di questo libro la routine di input verrà ulteriormente perfezionata, in modo tale da poter eventualmente caricare dati salvati su dischetto per visualizzarli in forma grafica.

2.8 Illusione o realtà: il mouse ed i menù in BASIC

I programmi realizzati fino ad ora non differiscono molto da programmi professionali come ad esempio Graphicraft o Textcraft. Sicuramente questi pacchetti sono, senza ombra di dubbio, più potenti e più veloci. Tuttavia, la caratteristica che più di ogni altra li differenzia, consiste nel fatto che i programmi professionali sono quasi completamente gestibili mediante mouse. Il programma è interamente controllato attraverso l'uso del puntatore del mouse e dei menù a tendina. Si vedranno ora ulteriori segreti di programmazione in AmigaBASIC che consentiranno di scrivere programmi più professionali.

• Per iniziare è possibile provare ad inserire le seguenti linee:

```
MENU 1,0,1 "Selezione"¶
FOR x=1 TO 9¶
  MENU 1,x,1,"Opzione "+CHR$(48+x)¶
NEXT x¶
FOR x=2 TO 4¶
  MENU x,0,0""¶
NEXT x¶
ON MENU GOSUB Selezione¶
MENU ON¶
Pausa:¶
  GOTO Pausa¶
Selezione:¶
  LOCATE 10,20¶
  PRINT "Hai scelto l'opzione";MENU(1)¶
  RETURN¶
```

MENU
Provando ad eseguire questo programma, si può pensare che esso non faccia proprio niente. Tuttavia, premendo il pulsante destro del mouse si nota che, invece degli usuali titoli di menù, compare un solo nuovo titolo.

• Selezionare tale menù ed osservare attentamente le sue opzioni.

Come si può vedere sono disponibili 9 differenti opzioni, selezionando le quali viene visualizzato sullo schermo il numero corrispondente.

MENU RESET
Questo programma, pur non essendo di grande utilità costituisce comunque un buon esempio di come sia facile programmare i menù in BASIC e di come i risultati ottenibili siano di livello professionale. Nel programma non è presente una opzione per terminare l'esecuzione e, come si può notare, non è disponibile neppure il menù Run con l'opzione Stop. Tuttavia è comunque possibile forzare la terminazione del programma. Digitando la combinazione di tasti <CTRL> <C>, si ottiene la terminazione istantanea di un qualsiasi programma. Come è possibile vedere i menù usuali non ritornano sullo schermo nemmeno dopo che il programma è stato interrotto. Dato che essi sono necessari per programmare, vi deve essere un qualche modo per farli riapparire.

• Battere il seguente comando nella finestra BASIC:

```
MENU RESET¶
```

I menù a tendina ritorneranno disponibili. E' possibile analizzare un po' più attentamente questi comandi e il loro funzionamento.

MENU richiede tre valori numerici ed il nome che deve essere assegnato al menù. Il comando iniziale del semplice programma dimostrativo è infatti:

```
MENU 1,0,1,"Selezione"¶
```

Specifica del menù
Il primo parametro del comando specifica il numero identificativo del me-

nù. Normalmente, il numero per il menù Project è 1, per il menù Edit 2, per il menù Run 3 e per il menù Windows 4. In una finestra BASIC può essere definito un numero massimo di 10 menù a tendina.

Il secondo parametro determina il numero identificativo di opzione nel menù. Il valore 0 corrisponde al titolo del menù. Sono possibili, per le opzioni, numeri tra 1 e 19.

Il terzo parametro può essere uno dei numeri seguenti:

* 0 consente di *disabilitare* l'opzione specificata. In tal modo l'opzione non risulta disponibile e viene mostrata come opzione *mascherata* (cioè in caratteri non perfettamente visibili). Nel caso il secondo parametro sia 0 il menù è completamente disabilitato.

* 1 produce l'effetto opposto a 0: consente infatti di abilitare le opzioni od il menù intero, rendendoli disponibili.

* 2 ha un effetto simile a 1, ad ecezione del fatto che l'opzione selezionata viene marcata con un contrassegno. Questo consente di segnalare all'utente che si tratta dell'opzione attualmente attiva.

Come nome per un'opzione di menù può essere utilizzata una qualsiasi stringa racchiusa tra doppi apici, oppure come contenuto di una variabile stringa. Se viene omesso il parametro stringa, il comando MENU assume la forma generalmente utilizzata per modificare le caratteristiche di una opzione già esistente (solitamente viene utilizzata per effettuare modifiche del terzo parametro). Il comando CHR$ consente di specificare un singolo carattere ed eventualmente, come nel caso in questione, aggiungerlo ad una stringa base (se si desidera sapere quale sia il carattere visualizzato dal comando CHR$(48), consultare l'Appendice B nella quale, alla voce CHR$, è riportata una tabella ASCII).

Attenzione

- Battere la linea che segue, senza però selezionare il menù a tendina che essa produce:

```
MENU 1,0,1,"Prova"¶
```

Se viene selezionato un menù per il quale non sono definite opzioni, si potrebbe verificare un crash di AmigaBASIC. Se non si è ancora provveduto a salvare il programma precedentemente inserito, esso verrebbe irrimediabilmente perso.

La linea appena inserita definisce un menù con numero identificativo 1 e nome Prova. Come è stato precedentemente detto, è meglio non provare a selezionare tale menù prima di aver provveduto a definire almeno una opzione per esso.

• Inserire la linea seguente:

```
MENU 1,1,1,"  Numero 1"¶
```

lasciando due spazi vuoti prima della parola Numero.

E' ora possibile utilizzare il menù a tendina senza correre alcun rischio. La linea appena inserita crea una opzione del menù. AmigaBASIC, tuttavia, non consente di fare molto con questa opzione, in quanto non vi è alcuna routine di supporto da eseguire in seguito alla sua selezione.

Per disabilitare completamente il menù è necessario inserire la seguente linea:

```
MENU 1,0,0¶
```

Per riabilitarlo utilizzare la linea:

```
MENU 1,0,1¶
```

La linea seguente marca la prima opzione con un contrassegno:

```
MENU 1,1,2¶
```

Come è possibile notare, se si definiscono meno di quattro menù, i menù definiti in precedenza devono essere cancellati. Questa operazione viene realizzata dalla linea seguente:

```
FOR x=1 TO 4: MENU x,0,0,"": NEXT x¶
```

Questa linea consente di cancellare dallo schermo i menù già esistenti, operazione che può rivelarsi molto utile in numerosi programmi. In ogni

caso, è possibile realizzare l'operazione inversa semplicemente con il comando:

```
MENU RESET¶
```

Intercettazione e gestione degli eventi (Event trapping)

Il comando MENU consente di ordinare i menù nel modo desiderato. Fino ad ora, però, non si è ancora esaminato il controllo e la valutazione dei menù. Il metodo utilizzato per il controllo dei menù costituisce una delle caratteristiche più affascinanti di AmigaBASIC. Se si analizza attentamente il programma di dimostrazione appena scritto, oppure lo si esegue in modalità TRACE, è possibile notare come il programma consista per il momento solamente del ciclo di ritardo Pausa: il quale continua a richiamare se stesso.

Questo funzionamento è basato sul principio dell'intercettazione e gestione degli eventi (*Event Trapping*). Se avviene un determinato evento che Amiga sta attendendo, il programma viene immediatamente sospeso, e viene eseguita una particolare subroutine. Dopo avere eseguito questa subroutine, AmigaBASIC riprende l'esecuzione del programma dal punto in cui era stato precedentemente interrotto. Questo modo di programmare non è solamente pratico, ma anche elegante. E' possibile realizzare un gran numero di subroutine che gestiscono vari eventi come collisione di oggetti in movimento, messaggi di errore, spostamenti del mouse e selezione di menù oppure intervalli di tempo, mentre il programma principale sta compiendo qualcosa di completamente differente.

Il comando seguente specifica ad AmigaBASIC quale subroutine richiamare quando viene selezionata un'opzione di menù:

```
ON MENU GOSUB ...¶
```

Questo comando consente di specificare tanto un numero quanto un'etichetta. Per ogni evento gestibile esiste un dispositivo di attivazione (o *trigger*) che attiva la funzione per l'intercettazione. Nel caso dei menù l'event trapping viene attivato dal comando:

```
MENU ON¶
```

MENU OFF, MENU ON, MENU STOP

Se si desidera interrompere l'intercettazione delle selezioni, è possibile utilizzare il comando MENU STOP. Per disabilitare completamente tale intercettazione è necessario invece utilizzare il comando MENU OFF. La differenza tra questi due comandi consiste nel riconoscimento dell'evento: con l'opzione MENU STOP Amiga continua a registrare il verificarsi dell'evento, ma non reagisce ad esso, aspettando semplicemente il momento in cui viene nuovamente riattivato l'event trapping. Dopo aver riattivato l'event trapping, Amiga ricorderà le selezioni che erano state effettuate mentre la gestione era stata disattivata. Il comando MENU OFF, viceversa, segnala ad Amiga di interrompere completamente la gestione degli eventi e nessuna selezione effettuata durante la sospensione dell'intercettazione viene ricordata quando l'event trapping viene riattivato.

Questi principi legati ai comandi ON, OFF e STOP sono identici per tutti i possibili eventi intercettabili. Nel corso del libro si farà sovente riferimento ad essi.

ON MENU GOSUB

L'ultima domanda a cui rispondere è la seguente: cosa può fare la subroutine richiamata con il comando ON MENU GOSUB? Come già rilevato, ogni volta che un utente seleziona un'opzione, viene richiamata la subroutine corrispondente. Prima è però necessario esaminare quale opzione sia stata selezionata. Per effettuare questa operazione si hanno a disposizione le funzioni MENU(0) e MENU(1).

MENU(0), MENU(1)

Il comando MENU(0) consente di determinare il numero del menù a tendina selezionato (fornisce cioè il valore del primo parametro), mentre con il comando MENU(1) viene determinata l'opzione scelta (fornendo il valore del secondo parametro).

La routine di dimostrazione che segue consente di espandere il semplice programma dimostrativo precedente aggiungendo al menù realizzato un'ulteriore opzione. E' necessario inserire questa linea tra i due cicli FOR ... NEXT della prima parte del programma originale:

```
MENU 1,10,1,"Fine"¶
```

- Modificare la routine Selezione:, in modo tale da porla nella forma seguente:

```
Selezione:¶
IF MENU(1)=10 THEN MENU RESET: END¶
LOCATE 10,20¶
PRINT "Hai scelto l'opzione";MENU(1)¶
RETURN¶
```

Tale routine funziona nel modo seguente: per prima cosa viene controllato se è stata selezionata l'opzione numero 10 (cioè l'opzione per la terminazione del programma), nel qual caso viene utilizzato il comando MENU RESET per richiamare i menù di partenza ed il programma viene terminato con il comando END. In tutti gli altri casi il programma funziona esattamente come in precedenza.

Dato che questo esempio presenta un solo menù a tendina, non è necessario preoccuparsi di sapere quale menù sia stato selezionato. All'interno di programmi in cui compaiono più menù, è necessario, prima di determinare l'opzione scelta (utilizzando il comando MENU(1)), determinare quale menù sia stato selezionato (utilizzando il comando MENU(0)).

MOUSE

Fino ad ora dovrebbe essere tutto abbastanza chiaro. Comunque, per realizzare una dimostrazione un pò più sostanziosa, verrà ora presentato un programma di utilità più lungo e potente. Prima è però necessario discutere alcuni concetti di base riguardo alla gestione dell'input da mouse ed alla programmazione del controllo del mouse. Il programma riportato di seguito costituisce probabilmente il più corto programma di disegno mai scritto per Amiga:

```
ON MUOSE GOSUB Disegna¶
MOUSE ON¶
Pausa:¶
  GOTO Pausa¶
Disegna:¶
  WHILE MOUSE(0)<>0¶
     PSET (MOUSE(1),MOUSE(2))¶
  WEND¶
```

```
RETURN¶
```

Considerando la brevità di questo programma, è abbastanza impressionante ciò che realizza. Come è possibile vedere l'event trapping per il mouse è analogo a quello per i menù. Il comando:

```
ON MOUSE GOSUB ...¶
```

specifica la subroutine che risponde ad un evento del mouse.

Il comando:

```
MOUSE ON¶
```

consente di attivare il processo di intercettazione. Il ciclo infinito costituisce il programma in esecuzione.

WHILE ... WEND

Solamente la routine Disegna: contiene qualcosa di veramente nuovo e non ancora spiegato: il ciclo WHILE ... WEND. Questo tipo di ciclo viene utilizzato per ripetere un'operazione fino a quando è verificata una particolare condizione. Un semplice ciclo che si pone in attesa di input da tastiera è il seguente:

```
WHILE INKEY$="": WEND¶
```

E' possibile provare ad eseguire questo esempio nella finestra BASIC. Il ciclo WHILE ... WEND è spesso più pratico ed elegante del ciclo FOR ... NEXT o IF ... THEN, benchè i tre siano abbastanza facilmente intercambiabili. La condizione controllata per l'esecuzione del ciclo WHILE ... WEND nel programma dimostrativo scritto è la seguente:

```
MOUSE(0)<>0¶
```

Eseguendo il programma si può notare che si è in grado di disegnare solamente quando viene premuto il pulsante sinistro del mouse. La funzione MOUSE(0) fornisce informazioni relative allo stato dei pulsante sinistro del mouse. Se non viene premuto tale pulsante, il valore ritornato da MOUSE(0) risulta essere 0; se invece viene premuto vi sono diverse possibilità che saranno discusse più avanti (nel caso si desiderino subito ulteriori informazioni sui valori ritornati da MOUSE(0) consultare l'appendice B dove

viene descritto dettagliatamente tale comando). Nell'esempio mostrato sono utilizzati anche MOUSE(1) e MOUSE(2).

MOUSE(1) contiene il valore della coordinata X a cui si trova il mouse.

MOUSE(2) contiene il valore della coordinata Y del mouse.

Questi due valori sono entrambi in pixel ed AmigaBASIC li calcola in modo automatico nello schermo corrispondente rendendo così semplice disegnare con il mouse.

Nel paragrafo seguente verrà realizzato un programma di disegno che utilizza quasi tutti i comandi AmigaBASIC, facile da utilizzare e che consente di realizzare disegni abbastanza interessanti.

2.9 Un programma di disegno in AmigaBASIC

In questo paragrafo viene presentato un programma che consente di realizzare grafica eccellente e di disegnare utilizzando Amiga. Naturalmente il programma è contenuto nel dischetto a corredo del libro. Nel caso non si volesse digitarlo linea per linea è possibile caricarlo direttamente dal dischetto. Il programma presente su dischetto è stato accuratamente provato ed è privo di eventuali errori tipografici involontariamente presenti nel libro.

Compatibilità tra programmi

Se si è provveduto ad inserire e salvare i programmi precedentemente esposti nel libro, si dovrebbe essere ora in possesso di due programmi di utilità: il programma di videotitolazione ed il programma per la realizzazione di grafici a barra e di diagrammi a torta. Con questi esempi si sta tentando di raggiungere uno scopo principale: fornire una serie di programmi che consentano, pur essendo abbastanza divertenti, di svolgere un buon processo di apprendimento e che si rivelino anche di una certa utilità. Vi è una importante caratteristica che tutti questi programmi dovrebbero avere: quella di essere *compatibili* tra loro. Questo significa che dovrebbe essere possibile realizzare una figura con il programma di disegno e successivamente utilizzare questa immagine come sfondo per il programma di videotitolazione. Dopo tutto, disegni realizzati con pacchetti grafici come Graphicraft o DeluxePaint dovrebbero essere utilizzabili in AmigaBASIC (come infatti risulta essere).

Il programma di disegno che ci si accinge a realizzare costituisce un importante anello in questa catena di collegamento. Esso risulterà essere un programma abbastanza sofisticato e potente, dotato di alcune caratteristiche riscontrabili in pacchetti grafici commerciali. Naturalmente, un programma così versatile non può essere corto come quello di disegno mostrato precedentemente: i prossimi due paragrafi conterranno quindi alcune pagine di listati da inserire.

Durante l'inserimento del programma si consiglia di mantenere le indentature e le divisioni mostrate; esse consentono di dare una forma più leggibile al programma nella versione finale, potendo così correggere più velocemente eventuali errori.

```
Inizializzazione:¶
  Colori=5 : MaxColori=2^Colori-1¶
  DIM Puntatore(4,1),AltroColore(4),Colori%(31,2)¶
  DIM ModoRiempi%(7),OgniModo%(8,7),Compatto%(1)¶
  TipoDisegno=1 : ColoreDisegno=1 : ColoreRiempi=2 :
      Modo=1¶
¶
  Compatto%(0)=&HFFFF : Compatto%(1)=&HFFFF¶
  FOR x=0 TO 7¶
    ModoRiempi%(x)=&HFFFF¶
    OgniModo%(0,x)=&HFFFF¶
  NEXT x¶
  FOR x=1 TO 8¶
    FOR y=0 TO 7¶
      READ OgniModo%(x,y)¶
    NEXT y¶
  NEXT x¶
¶
  DATA 24672,1542,24672,1542,24672,1542,24672,1542¶
  DATA -13108,13107,-13108,13107,-13108,13107,
      -13108,13107¶
  DATA 26214,13107,-26215,-13108,26214,13107,-26215,
      -13108¶
  DATA -13108,-26215,13107,26214,-13108,
      -26215,13107,26214¶
  DATA -258,-258,-258,0,-4113,-4113,-4113,0¶
  DATA -8185,-8197,-18019,-20491,-20467,-8197,-8185,
      -1¶
  DATA 0,0,1632,4080,4080,2016,384,0¶
  DATA 960,1984,3520,6592,16320,25024,-3104,0¶
¶
  SCREEN 1,320,256,Colori,1¶
```

```
WINDOW 2,"Programma di disegno AmigaBASIC",,16,1¶
WINDOW CLOSE 3¶
WINDOW CLOSE 4¶
WINDOW 2¶
```

Il programma comicia con una routine di inizializzazione in cui tutte le variabili vengono inizializzate con valori di partenza; i vettori vengono dimensionati e riempiti e le finestre e gli schermi vengono attivati.

La variabile Colori contiene, ancora una volta, il numero dei bitplane attivi. Essa può assumere i valori 2 (4 colori), 3 (8 colori), 4 (16 colori) e 5 (32 colori). Normalmente, dato che il programma lavora in bassa risoluzione, il modo operativo sarà a 32 colori. Non dovrebbero esservi dunque problemi di memoria per un qualsiasi modello Amiga con almeno 512k di RAM.

Vettori di interi

E' possibile notare, nell'istruzione utilizzata per il dimensionamento dei vettori, il simbolo % di seguito al nome del vettore Colori%, simbolo non ancora incontrato fino ad ora. Non si tratta di un errore di battitura. AmigaBASIC permette l'utilizzo di tipi differenti per le variabili. Due tipi distinti sono già stati visti: le variabili numeriche (a, b, etc.), e le *variabili stringa* (a$, b$, etc). Le variabili di tipo stringa sono identificate dal simbolo $ di seguito al loro nome. Analogamente, il simbolo % viene utilizzato per identificare le variabili intere. Sono già state incontrate le funzioni BASIC INT e CINT e si dovrebbe quindi sapere che i numeri interi sono quei numeri che non presentano nessuna cifra a destra del punto decimale.

E' possibile dichiarare alcune variabili come variabili intere prima che esse vengano utilizzate. Dato che esse non presentano cifre decimali, utilizzeranno una quantità minore di memoria. Accade questo in seguito al modo in cui Amiga gestisce i numeri all'interno della RAM. Le variabili intere possono assumere solamente valori compresi tra -32768 e 32767 (nella nota d'uso 3 verrà analizzato dettagliatamente questo fatto, mentre nei paragrafi seguenti verranno introdotti nuovi tipi di variabili).

• Inserire la linea seguente:

```
a=7/3 : a%=7/3 : ? a,a%¶
```

Come si può notare la variabile a% non ha cifre decimali. Dato che nel programma di disegno verranno utilizzati solamente numeri interi, i vettori in esso utilizzati verranno definiti come variabili intere. Il vettore Colori% viene dimensionato come una matrice (31,2) in modo da poter contenere 32 x 3 elementi (si ricordi quanto detto a suo tempo per i vettori a più dimensioni: è possibile rappresentarli facilmente come una tabella in cui ogni singola posizione rappresenta un particolare elemento del vettore). E' possibile utilizzare fino a 32 colori nel programma, per ognuno dei quali deve essere specificata la componente nelle tre bande principali (R, G e B).

FILL PATTERNS

Nella routine di inizializzazione vengono definiti anche i tipi possibili per il riempimento delle aree. Nel prossimo paragrafo verranno discussi in dettaglio i Fill Pattern di AmigaBASIC. Per il momento è necessario ricordare una sola cosa: il processo di riempimento secondo vari schemi richiede alcuni vettori interi in cui vengono memorizzati gli schemi. Nel programma dimostrativo sono utilizzati, per questo scopo, i vettori ModoRiempi%, OgniModo% e Compatto%. Il programma offre una scelta tra nove differenti stili di riempimento. Le definizioni di questi nove stili sono contenute nel vettore OgniModo%. Compatto% contiene uno schema particolare per realizzare riempimenti con aree solide in un'unica tinta.

Numeri esadecimali

Questi vettori vengono inizializzati con valori di partenza nella routine di inizializzazione. Il comando utilizzato in questa routine ha però una forma particolare: compare infatti come elemento destro di un assegnamento uno strano numero (&HFFFF). Tale numero è un *valore esadecimale.* Numeri di questo tipo si rivelano molto utili per i programmatori (è possibile trovare ulteriori informazioni sui numeri esadecimali nella nota d'uso 3). AmigaBASIC è in grado di lavorare con i numeri esadecimali tanto bene quanto con i normali numeri decimali.

Il valore &HFFFF realizza una definizione di schema di riempimento in cui tutti i punti vengono posti ad un valore nullo. Questo produce un'area solida, in tinta unita. Questo schema di riempimento viene assegnato ai vettori Compatto% e ModoRiempi% come il primo dei nove schemi selezionabili. Prima di iniziare il programma è necessario scegliere uno schema tra i 9 disponibili.

READ - DATA

I valori richiesti vengono caricati utilizzando i comandi READ e DATA. I comandi READ e DATA costituiscono una coppia molto affiatata che consente di inserire un valore in un programma, caricarlo e successivamente utilizzarlo. I valori forniti possono essere sia numeri che parole o una qualsiasi combinazione di caratteri. I dati devono essere scritti su una linea che comincia con il comando DATA, separati da virgole. Il comando READ carica poi i dati nelle variabili. La combinazione DATA - READ può essere utilizzata solamente all'interno di un listato e non può quindi essere utilizzata in modo diretto.

Le seguenti cinque linee non fanno parte del programma di disegno. Se lo si desidera è possibile eseguirle per comprendere meglio il funzionamento dei comandi DATA e READ. Prima di farlo, è però necessario provvedere a salvare il programma di disegno.

```
READ a$,b$,c$,d¶
PRINT "Ho comprato un ";a$¶
PRINT "E' un grande ";b$;"che mi procura un grande
    ";c$;"."¶
PRINT "con esso posso fare ";d;"cose."¶
DATA Amiga,Computer,divertimento,1000¶
```

I valori presenti nella linea con il comando DATA vengono assegnati alle variabili corrispondenti nella linea con il comando READ. In un programma, la lettura comincia sempre dalla prima linea DATA, indipendentemente da dove questa si trovi nel programma. I valori vengono poi letti consecutivamente. E' necessario fare attenzione alla esatta corrispondenza tra i tipi delle variabili ed i valori delle linee DATA. Infatti, se si tenta di caricare un carattere in una variabile numerica, si ottiene il messaggio di errore Type Mismatch. Se tutti i dati sono stati caricati e viene incontrato un ulteriore comando READ, Amiga visualizza il messaggio di errore Out of DATA.

Il primo blocco di istruzioni DATA contiene le strutture dei nove schemi di riempimento. Questi valori vengono caricati nel vettore OgniModo%. Dopo aver effettuato questa operazione, vengono definiti lo schermo e la finestra di output.

Bassa risoluzione e ....

Come segnalato precedentemente, il programma funziona in bassa risoluzione (320x256) principalmente perchè si desidera che i grafici realizzati siano compatibili con programmi commerciali come Graphicraft. Quasi tutti i programmi di disegno commerciali utilizzano la bassa risoluzione, in quanto essa consente di avere 32 colori disponibili. Se si è avuto occasione di vedere della semplice grafica di Amiga, si sarà sicuramente notato come la quantità di colori sia più varia per la risoluzione minore. Le due linee seguenti del programma disattivano le finestre 3 e 4, nel caso esse siano già presenti sullo schermo (cosa che può accadere se il programma viene interrotto e poi rilanciato).

- Posizionarsi alla fine del codice introdotto ed inserire le parti restanti del programma di disegno:

```
    FOR x=0 TO 31¶
      READ r,g,b¶
      PALETTE x,r/16,g/16,b/16¶
      Colori%(x,0)=r : Colori%(x,1)=g : Colori%(x,2)=b¶
    NEXT x¶
¶
    DATA 0,0,3, 15,15,15, 0,3,12, 15,0,0¶
    DATA 0,14,15, 15,0,15, 3,10,1, 15,14,0¶
    DATA 15,8,0, 10,0,14 ,8,5,0, 11,8,3¶
    DATA 2,11,0, 15,10,15, 0,0,9, 7,15,0¶
    DATA 14,12,0, 15,2,3, 0,0,0, 15,11,10¶
    DATA 0,6,8, 3,3,3, 4,4,4, 5,5,5¶
    DATA 6,6,6, 7,7,7, 8,8,8, 9,9,9¶
    DATA 11,11,11, 13,13,13, 0,0,15, 12,15,12 ¶
```

In questa parte di programma vengono assegnati i valori iniziali al vettore Colori%. Durante l'esecuzione del programma questo vettore conterrà i valori per le componenti di Rosso, Verde e Blu dei vari colori disponibili. I valori per le componenti sono numeri interi compresi tra 0 e 15. E' comunque necessario sottolineare un problema: come già fatto notare durante la realizzazione del programma di videotitolazione, non esiste un comando AmigaBASIC che consenta di leggere i valori correnti dei registri colore. Questi registri possono essere modificati utilizzando il comando PALETTE, ma non vi è modo, in BASIC, di poterne leggere i contenuti.

Se si modificano i colori nel programma per la prima volta, i valori di base vengono modificati in Nero (R=0, G=0, B=0). Questo rende il programma un po' difficile da utilizzare, in quanto risulta difficoltoso vedere linee disegnate in nero su sfondo nero. Per questo motivo vengono memorizzate le componenti RGB dei 32 colori nel vettore Colori%, in modo da essere sicuri di averli disponibili fino a quando non vengono modificati durante l'esecuzione del programma.

- Posizionarsi alla fine del listato del programma dimostrativo ed inserire le linee seguenti:

```
MenuTendina:¶
  MENU 3,0,0,""¶
  MENU 4,0,0,""¶
  MENU 1,0,1,"Programma"¶
  MENU 1,1,1,"Disegno          "¶
  MENU 1,2,1,"Palette Colori   "¶
  MENU 1,3,1,"Modo Riempimento "¶
  MENU 1,4,1,"Carica Schermo   "¶
  MENU 1,5,1,"Salva Schermo    "¶
  MENU 1,6,1,"Cancella Schermo "¶
  MENU 1,7,1,"Fine             "¶
  MENU 2,0,1,"Strumenti"¶
  MENU 2,1,2,"  Tratto Sottile "¶
  MENU 2,2,1,"  Tratto Marcato "¶
  MENU 2,3,1,"  Punti          "¶
  MENU 2,4,1,"  Spray          "¶
  MENU 2,5,1,"  Linee          "¶
  MENU 2,6,1,"  Rettangoli Vuoti"¶
  MENU 2,7,1,"  Rettangoli Pieni"¶
  MENU 2,8,1,"  Linee Connesse "¶
  MENU 2,9,1,"  Cerchi - Ovali "¶
  MENU 2,10,1,"  Riempimento   "¶
  MENU 2,11,1,"  Cancellino    "¶
  MENU 2,12,1,"  Testo         "¶
```

In questa sezione vengono definiti i menù a tendina. Prima di tutto vengono cancellati i menù 3 e 4, in quanto nel programma ne sono utilizzati

due soli. Il menù numero 1 contiene le opzioni per il controllo del programma: è possibile selezionare le opzioni per la gestione dei file, per il controllo dei colori, dello schermo e degli schemi di riempimento. Il menù 2 contiene invece i singoli strumenti utilizzati nel processo di disegno.

Lo strumento attualmente attivo è indicato mediante la visualizzazione di un contrassegno alla sinistra del suo nome nel menù. E' necessario assicurarsi di avere lasciato due spazi vuoti quando sono stati inseriti i nomi delle opzioni, in modo tale che il contrassegno possa essere visualizzato senza sovrapporsi al nome dell'opzione.

Dopo aver sbrigato tutti i preliminari, è possibile iniziare ad inserire le routine necessarie per il disegno, iniziando da quella contenente il ciclo principale:

```
CicloPrincipale:¶
  ON MENU GOSUB SelezioneMenu¶
  ON MOUSE GOSUB TestaMouse¶
  MENU ON¶
  MOUSE ON¶
¶
  WHILE -1¶
  WEND¶
```

Utilizzo dell'Event Trapping

Questa routine attiva l'intercettazione di eventi legati al mouse ed ai menù e definisce quali sottoprogrammi debbano essere richiamati in seguito al verificarsi di un evento. Nella routine è compreso anche un ciclo WHILE ... WEND. Il numero -1 posto di seguito a WHILE non è un errore di stampa; esso è infatti utilizzato, in AmigaBASIC, per rappresentare una condizione sempre vera (nella prossima nota d'uso verrà analizzata più dettagliatamente questa costruzione sintattica). Questa configurazione produce un ciclo infinito. Fino a quando non viene utilizzato il mouse, selezionando un menù oppure premendo un pulsante, il programma continua semplicemente ad eseguire questo ciclo. Tutte le funzioni del programma di disegno sono gestite in base all'evento intercettato. E' possibile che questo fatto non risulti completamente chiaro; esso rende tuttavia possibile la realizzazione di programmi molto potenti e flessibili. L'event trapping può essere utilizzato anche durante la programmazione "norma-

le". Il modo in cui strutturare un programma dipende da cosa questo deve fare.

Il programma principale è seguito dai sottoprogrammi necessari per il controllo dei menù:

```
SelezioneMenu:¶
  Men=MENU(0)¶
  SceltaMenu=MENU(1)¶
  ON Men GOTO Progetto,StrumentiDisegno¶
¶
TestaMouse:¶
  IF Modo=1 THEN ON TipoDisegno GOSUB
      TrattoLeggero,TrattoMarcato,Punti,Spray,
      TracciaLinee,RettVuoto,RettPieno,
      LineeCongiunte,Cerchi,Riempi,Cancella,Testo¶
  IF Modo=2 THEN GOSUB PaletteColori : IF OkFine=1 THEN
      GOSUB OkColori¶
  IF Modo=3 THEN GOSUB DefinisciModo : IF OkFine=2 THEN
      GOSUB OkModo¶
  IF Modo=4 THEN GOSUB RGBDef : IF OkFine=3 THEN
      Modo=2 : GOSUB SelezioneColore¶
RETURN¶
¶
Progetto:¶
  IF SceltaMenu=1 THEN GOSUB OkColori : GOSUB OkModo¶
  IF SceltaMenu=2 THEN GOSUB OkModo : MENU 2,0,0 :
      Modo=2 : GOSUB SelezioneColore¶
  IF SceltaMenu=3 THEN GOSUB OkColori : MENU 2,0,0 :
      Modo=3 : GOSUB EditorModi¶
  IF SceltaMenu=4 THEN GOSUB OkColori : GOSUB OkModo :
      GOSUB CaricaDisegno¶
  IF SceltaMenu=5 THEN GOSUB OkColori : GOSUB OkModo :
      GOSUB RegistraDisegno¶
  IF SceltaMenu=6 AND Modo=1 THEN OK=0 : GOSUB
      Richiesta : IF OK=1 THEN Adef=0 : AREAFILL: CLS¶
  IF SceltaMenu=7 THEN GOSUB OkColori : GOSUB OkModo :
```

```
    OK=0 :GOSUB Richiesta :IF OK=1 THEN FinisciTutto¶
RETURN¶
¶
StrumentiDisegno:¶
  MENU 2,TipoDisegno,1¶
  TipoDisegno = MENU (1)¶
  MENU 2,TipoDisegno,2¶
RETURN¶
```

ON ... GOTO

La routine SelezioneMenu: consente di determinare quale menù sia stato selezionato. La variabile Men contiene il numero identificatore del menù, mentre la variabile SceltaMenu contiene il numero dell'opzione selezionata. In base al menù selezionato, il programma esegue la corrispondente routine, rispettivamente Progetto: e Strumenti:. Per gestire il controllo viene utilizzata una particolare modifica del comando GOTO. Il comando ON ... GOTO consente di richiamare la routine etichettata con il nome specificato in dipendenza dal valore di una variabile.

La linea seguente, che non fa parte del programma di disegno, mostra il funzionamento di questo comando:

```
ON x GOTO aperitivo,antipasto,primo,secondo,dolce¶
```

Se x vale 1 viene eseguita la routine aperitivo, se x vale 2 la routine antipasto, e così via. Se x ha un valore minore di 1 o maggiore di 5 il programma continua con l'istruzione successiva.

ON ... GOSUB

La routine TestaMouse: è responsabile del controllo del mouse. All'inizio di questa routine vi è una linea, contenente un comando ON ... GOSUB, abbastanza lunga. Questo comando funziona in modo simile al comando ON ... GOTO appena visto. L'unica differenza consiste nel fatto che, in seguito al successivo comando RETURN, il programma ritorna alla linea contenente il dato ON ... GOSUB.

Se si preme il pulsante sinistro del mouse mentre il programma è in esecuzione, viene richiamata questa routine. Premere il pulsante sinistro del

mouse rappresenta un'azione abbastanza abituale il cui effetto dipenderà dalla modalità in cui si trova il programma. Per valutare la modalità corrente viene utilizzata la variabile Modo. I vari sottoprogrammi richiamati modificheranno opportunamente il valore di questa variabile. Se Modo contiene il valore 1, il programma è in modo disegno. Per portarsi in questa modalità è sufficiente selezionare la prima opzione del primo menù a tendina.

La variabile TipoDisegno contiene le informazioni relative alla modalità di disegno corrente. Quando viene selezionata un'opzione del secondo menù a tendina, viene specificata la modalità di disegno. Vi è un differente sottoprogramma per ogni diverso tipo di modalità di disegno. Il programma richiama ed esegue questo sottoprogramma non appena viene premuto il pulsante del mouse. Se Modo vale 2, risulta attiva la routine per la selezione dei colori. In questo caso, il controllo del programma viene spostato nella routine PaletteColori. La variabile OkFine informa che le modifiche dei colori sono state completate (operazione specificata effettuando un click sul gadget OK). Se OkFine vale 1, AmigaBASIC esegue la routine OkColori.

Se Modo vale 3, la procedura risulta essere la stessa, ad eccezione del fatto che la routine richiamata è la routine per la definizione dei modelli di riempimento. Viene perciò richiamato il sottoprogramma DefinisciModo. Terminata la definizione, la variabile OkFine deve avere valore 2.

Modo 4 non si differenzia di molto dagli altri tre casi: si tratta infatti di una particolare funzione del sottoprogramma per la gestione del colore. Non viene semplicemente selezionato un colore, ma sono attivi i tre cursori per il controllo delle componenti di Rosso, Verde e Blu del colore. Vi è infatti una modalità separata per distinguere tra la semplice selezione e la definizione di un colore. Terminata questa modalità (OkFine vale 3) il programma ritorna in modalità 2 (la modalità relativa alla selezione di un colore).

Dato che la routine TestaMouse risulta essere un sottoprogramma, deve essere terminata dal comando RETURN. In questo modo il controllo ritorna al programma principale che riprende l'esecuzione dal punto in cui era stata chiamata la subroutine.

All'interno dei menù
La routine Progetto è responsabile delle operazioni richieste con il primo menù a tendina. La variabile SceltaMenu contiene le informazioni relative alla scelta effettuata, informazioni catturate dalla routine SelezionaMenu. A seconda del valore di SceltaMenu vengono richiamate ed eseguite le routine necessarie per svolgere l'azione richiesta. L'opzione numero 1 del menù consente di ritornare alla modalità di disegno; tale modalità viene sempre automaticamente attivata all'inizio del programma (nella routine di inizializzazione viene infatti eseguito l'assegnamento Modo=1). Per compiere l'azione richiesta selezionando la prima opzione devono quindi essere richiamate solamente le parti di programma che portano a termine le routine per la definizione dei Pattern e per la gestione dei colori (devono cioè essere richiamate le routine OkColori e OkModo).

L'opzione 2 richiama il sottoprogramma per la selezione dei colori. Nel caso il sottoprogramma per la definizione dei pattern sia ancora attivo, viene per prima cosa terminato richiamando la routine OkModo (GOSUB OkModo). Il menù a tendina che consente di selezionare la modalità di disegno risulta disabilitato durante l'esecuzione della routine per la gestione del colore (MENU 2,0,0). Alla variabile Modo viene assegnato il valore 2 e viene richiamata la subroutine SelezioneColore.

La terza opzione del primo menù consente di attivare l'editor di pattern, con il quale è possibile definire modelli di riempimento (fill pattern). Per prima cosa viene terminato, nel caso fosse attivo, il sottoprogramma per la gestione del colore; vengono poi disattivati i vari strumenti di disegno (disabilitando il secondo menù) e viene assegnato il valore 3 alla variabile Modo. Dopo questi preparativi viene richiamata la routine EditorModi.

Le opzioni 4 e 5, che consentono di salvare e caricare dati da dischetto, non sono per ora disponibili. Verranno aggiunte al programma in un successivo capitolo, nel quale si troveranno anche le informazioni relative ai dispositivi periferici ed alla gestione dei dati su dischetto.

La sesta opzione è abilitata solamente in modalità 1; essa produce la cancellazione dell'immagine presente in memoria. Prima di cancellare il disegno viene eseguita la routine Richiesta, la quale visualizza un requester per assicurarsi che si desideri effettivamente effettuare la cancellazione. Effettuando un click sul gadget OK si avvia la cancellazione. L'assegnamento Adef=0 ed il comando AREAFILL procedono a rimuovere ogni possibile area in fase di costruzione (questo risulta essere l'unico procedi-

mento di disegno non visualizzato immediatamente, ma costruito passo passo). Il processo di costruzione deve essere terminato prima che lo schermo venga ripulito. Si procede poi a vuotare lo schermo utilizzando il comando CLS.

La settima, ed ultima, opzione del menù consente di terminare l'esecuzione del programma di disegno. Per prima cosa vengono terminati gli eventuali sottoprogrammi in esecuzione, dopo di che viene visualizzato un requester simile a quello precedente, per ottenere conferma della volontà di abbandonare il programma. Se si effettua un click sul gadget OK, viene eseguita la parte di programma etichettata FinisciTutto che causa la terminazione del programma.

La subroutine Progetto termina con un comando RETURN.

La routine StrumentiDisegno si occupa di gestire il secondo menù. All'opzione precedentemente selezionata viene tolto il contrassegno, la variabile TipoDisegno viene aggiornata e l'opzione attualmente selezionata viene marcata.

Ecco ora, ad uno ad uno, le subroutine che gestiscono le differenti modalità possibili di disegno.

Prima di tutto la routine per il disegno a mano libera con penna sottile:

```
TrattoLeggero:¶
  Test= MOUSE(0) : x=MOUSE(1) : y=MOUSE(2)¶
  WHILE MOUSE(0)<>0¶
    LINE (x,y)-(MOUSE(1),MOUSE(2)),ColoreDisegno¶
    x=MOUSE(1) : y=MOUSE(2)¶
  WEND¶
RETURN¶
```

Disegnare una linea sottile

Ci si potrebbe chiedere come mai era stato assegnato il valore di MOUSE(0) alla variabile Test, che non è stata ancora utilizzata. Come si può ricordare, è possibile dedurre se sia stato premuto o venga tenuto premuto il pulsante sinistro del mouse utilizzando MOUSE(0). MOUSE(1) fornisce invece le coordinate X relative alla posizione del puntatore, mentre MOU-

SE(2) fornisce le coordinate Y. MOUSE(1) e MOUSE(2) forniscono quindi le coordinate X ed Y relative al punto in cui è stato premuto, l'ultima volta, il pulsante. Per ottenere tale punto è quindi necessario controllare il valore di MOUSE(0) prima di leggere i valori delle coordinate. Il ciclo WHILE ... WEND consente di disegnare nella posizione del puntatore finchè viene tenuto premuto il pulsante sinistro del mouse. Come risultato, utilizzando il mouse, viene disegnata una linea a mano libera.

La subroutine seguente consente di disegnare a mano libera con una linea più marcata ed è praticamente identica alla precedente.

```
TrattoMarcato:¶
  Test=MOUSE(0)¶
  WHILE MOUSE(0)<>0¶
    x=MOUSE(1) : y=MOUSE(2)¶
    LINE (x,y)-(x+5,y+5),ColoreDisegno,bf¶
  WEND¶
RETURN¶
```

Disegnare una linea marcata

Per disegnare una linea marcata viene visualizzato un quadrato nella posizione attuale del puntatore. Quando viene spostato il mouse, non molto velocemente, viene disegnata una linea grossa. Se però il mouse viene spostato velocemente, i singoli quadratelli risultano staccati tra loro e la linea sembra quindi essere tratteggiata.

Se si desidera disegnare effettivamente una linea tratteggiata, è necessario inserire la routine seguente:

```
Punti:¶
  Test=MOUSE(0)¶
  WHILE MOUSE(0)<>0¶
    PSET (MOUSE(1),MOUSE(2)),ColoreDisegno¶
  WEND¶
RETURN¶
```

Questa routine consente di disegnare un punto nella posizione corrente del puntatore; tenendo premuto il pulsante sinistro e muovendo il mouse lentamente, si ottiene una linea sfumata; se si mouve il mouse velocemen-

te la linea risulta tratteggiata.

E' possibile ottenere facilmente l'effetto Spray:

```
Spray:¶
  Test=MOUSE(0)¶
  WHILE MOUSE(0)<>0¶
    x=MOUSE(1)+14*RND : y=MOUSE(2)+7*RND¶
    LINE (x,y)-(x,y),ColoreDisegno,bf¶
  WEND¶
RETURN¶
```

Effetto Spray
Lo spruzzatore disegna alcuni piccoli puntini all'interno di un'area specificata. Le coordinate dei singoli puntini sono casuali; i punti possono distare fino a 14 pixel in orizzontale e sette in verticale dalla posizione del puntatore. Ci si potrebbe chiedere come mai venga utilizzata un'istruzione abbastanza complessa contenente il comando LINE (oltretutto con il parametro per il riempimento) per realizzare l'effetto Spray. Dopo tutto deve solamente essere disegnato un singolo punto. Il motivo è molto semplice: il metodo adottato presenta un grosso vantaggio rispetto al comando PSET. Utilizzando il parametro bf viene infatti usato il fill pattern corrente. Nel caso sia selezionato un fill pattern e venga utilizzato lo spruzzatore per un certo periodo di tempo sullo stesso punto, verrà pian piano ricostruito il pattern in quell'area. Con questa tecnica è possibile ottenere effetti molto interessanti.

```
TracciaLinee:¶
  Test=MOUSE(0)¶
  x1=MOUSE(3) : y1=MOUSE(4)¶
  PSET (x1,y1),ColoreDisegno¶
  WHILE MOUSE(0)<>0¶
  WEND¶
  LINE (x1,y1)-(MOUSE(5),MOUSE(6)),ColoreDisegno¶
RETURN¶
```

Ulteriori informazioni sul tracciamento delle linee
La routine precedente consente di tracciare una linea tra due punti. La pro-

cedura utilizzata è molto semplice:

- Posizionare il puntatore nel punto iniziale della linea.
- Premere il pulsante sinistro del mouse e, tenendolo premuto, spostare il puntatore all'altra estremità della linea.
- Rilasciare il pulsante del mouse.

In questo modo è stata disegnata una linea.

Come si sarà potuto notare in questa routine vengono utilizzati alcuni nuovi parametri della funzione MOUSE: MOUSE(3), MOUSE(4), MOUSE(5) e MOUSE(6).

Quando viene premuto e poi rilasciato il pulsante sinistro del mouse tali parametri assumono i valori mostrati di seguito:

MOUSE(3) contiene il valore della coordinata X di partenza.

MOUSE(4) contiene il valore dellla coordinata Y di partenza.

MOUSE(5) il valore della coordinata X di arrivo.

MOUSE(6) il valore della coordinata Y di arrivo.

Controllo delle coordinate del mouse

Ancora una volta MOUSE (1) e MOUSE(2) contengono i valori delle coordinate del puntatore nel momento in cui viene premuto il pulsante sinistro. I parametri successivi, da MOUSE(3) a MOUSE(6), forniscono invece informazioni relative alle coordinate iniziali e finali di un movimento. Tale movimento comincia con la pressione del pulsante e termina con il suo rilascio. Nel programma, le coordinate di partenza sono memorizzate in x1 e y1; il punto viene marcato sullo schermo utilizzando il comando PSET, in modo che risulti possibile vederlo. Il ciclo vuoto WHILE ... WEND produce un ciclo di attesa fino a quando il pulsante rimane premuto. Una volta che questo viene rilasciato, la linea viene disegnata.

MOUSE(3), MOUSE(4), MOUSE(5) e MOUSE(6) si rivelano utili specialmente quando si stanno disegnando linee e poligoni (per i quali sono richiesti più punti). Essi possono essere utilizzati per realizzare il disegno di rettangoli, come si può vedere nella routine che segue.

- Posizionarsi alla fine del programma inserito fino ad ora e battere le linee seguenti:

```
RettVuoto:¶
  Test=MOUSE(0)¶
  x1=MOUSE(3) : y1=MOUSE(4)¶
  Puntatore(0,0)=x1 : Puntatore(0,1)=y1¶
  Puntatore(1,0)=x1 : Puntatore(2,1)=y1¶
  Value=4¶
  WHILE MOUSE(0)<>0¶
    Puntatore(3,0)=MOUSE(5)¶
    Puntatore(3,1)=MOUSE(6)¶
    Puntatore(1,1)=Puntatore(3,1)¶
    Puntatore(2,0)=Puntatore(3,0)¶
    GOSUB PiazzaPunto¶
  WEND¶
  LINE (x1,y1)-(Puntatore(3,0),Puntatore(3,1)),
      ColoreDisegno,b¶
RETURN¶
```

Rettangoli

Gran parte delle istruzioni di questa routine sono utilizzate per assegnare i valori ai vari elementi della matrice Puntatore. Solamente nell'ultima linea vi è un comando LINE. In alcune funzioni, come in quella relativa alla costruzione di un rettangolo, è possibile specificare le dimensioni del rettangolo prima di procedere al disegno finale. E' comunque opportuno avere un modo per individuare le attuali dimensioni dell'oggetto. In alcuni programmi, viene visualizzato l'oggetto in fase di costruzione; nel programma presentato non viene utilizzata questa soluzione per non appesantire eccessivamente il tempo di esecuzione ed aumentare la quantità di memoria necessaria al programma. Si è invece deciso di visualizzare solamente i punti relativi ai vertici del rettangolo.

Tali vertici sono contenuti nella matrice Puntatore secondo lo schema seguente:

Puntatore(0,0) contiene la coordinata X del primo punto.

Puntatore(0,1) contiene la coordinata Y del primo punto.

Puntatore(1,0) contiene la coordinata X del secondo punto.

Puntatore(1,1) contiene la coordinata Y del secondo punto.

La variabile Puntatore(2,0) contiene il terzo valore per la X, Puntatore(2,1) il terzo valore per la Y, e così via.

La variabile Value contiene il numero totale di punti. La subroutine PiazzaPunto converte poi i valori contenuti in Puntatore in punti in movimento sullo schermo.

Disegnare rettangoli

Per disegnare un rettangolo è necessario premere il pulsante sinistro del mouse, definendo in tal modo il primo vertice; tenendo ora premuto il pulsante, è possibile spostare il puntatore fino al vertice opposto e completare il rettangolo rilasciando il pulsante. Durante questo procedimento i vertici vengono visualizzati con puntini sullo schermo.

E' possibile notare una caratteristica peculiare: durante il ciclo WHILE ... WEND vengono continuamente acquisiti i valori MOUSE(5) e MOUSE(6), anche se il pulsante non è ancora stato rilasciato. I valori forniti da tali parametri sono in questo caso identici a quelli contenuti in MOUSE(1) e MOUSE(2). Questo spiega come l'ultima chiamata a MOUSE(0) si riveli importantissima.

Blocchi

La routine per la realizzazione di rettangoli pieni risulta identica a quella per i rettangoli, ad eccezione del fatto che il parametro b risulta ora sostituito dal parametro bf (block fill).

Per ridurre i tempi di inserimento è possibile duplicare la precedente routine utilizzando insieme le opzioni Copy e Paste del menù a tendina Edit apportando poi le modifiche necessarie:

```
RettPieno:¶
  Test=MOUSE(0)¶
  x1=MOUSE(3) : y1=MOUSE(4)¶
  Puntatore(0,0)=x1 : Puntatore(0,1)=y1¶
  Puntatore(1,0)=x1 : Puntatore(2,1)=y1¶
  Value=4¶
  WHILE MOUSE(0)<>0¶
    Puntatore(3,0)=MOUSE(5)¶
    Puntatore(3,1)=MOUSE(6)¶
    Puntatore(1,1)=Puntatore(3,1)¶
    Puntatore(2,0)=Puntatore(3,0)¶
    GOSUB PiazzaPunto¶
  WEND¶
  LINE (x1,y1)-(Puntatore(3,0),Puntatore(3,1)),
    ColoreDisegno,bf¶
RETURN¶
```

L'opzione RettPieno consente di disegnare rettangoli nella stessa modalità vista in precedenza; i rettangoli risulteranno però ora riempiti con il fill pattern corrente. Per riempire altre aree (oggetti invece di rettangoli) è possibile utilizzare il comando AREA:

```
LineeCongiunte:¶
   Test=MOUSE(0)¶
  x1=MOUSE(3) : y1=MOUSE(4)¶
  IF y1>242 THEN y1=242¶
  IF x1>311 THEN x1=311¶
  AREA (x1,y1)¶
  IF Adef=0 THEN Adef=1 : xa=x1 : ya=y1¶
  IF Adef<>1 AND x1=xa AND y1=ya THEN IniziaRiempi¶
  Adef=Adef+1 : IF Adef=20 THEN IniziaRiempi¶
  LINE (xa,ya)-(x1,y1),ColoreDisegno¶
  xa=x1 : ya=y1¶
RETURN¶
```

```
¶
IniziaRiempi:¶
  Adef=0 : COLOR ColoreDisegno,0 : AREAFILL ¶
RETURN¶
```

Disegnare altri tipi di poligoni
Dopo aver selezionato l'opzione Linee Connesse del menù a tendina Strumenti, è necessario posizionare il puntatore nel punto in cui si desidera collocare il primo vertice del poligono.

• Effettuare un click e spostare il puntatore fino al secondo vertice.

• Ripetere la procedura fino ad massimo di venti punti.

I punti precedentemente definiti verranno connessi da linee, in modo che risulti possibile vedere i contorni del poligono. Dopo il ventesimo punto il poligono viene automaticamente visualizzato sullo schermo, dato che il comando AREA non è più in grado di accettare altri punti. Per disegnare poligoni con meno di 20 lati è sufficiente effettuare un doppio-click sullo stesso punto: il poligono viene immediatamente visualizzato.

Mantenersi nell'area consentita
Il comando AREA visualizza un messaggio di errore se si tenta di selezionare un punto esterno all'area specificata. La massima area disponibile nella finestra è di 240 x 311 punti, automaticamente determinata dalla routine LineeCongiunte (i due blocchi IF ... THEN ... ELSE controllano questo). Il programma fornisce poi ad Amiga i punti in cui è stato effettuato il click relativo al presente comando AREA.

La variabile Adef memorizza il numero di punti precedentemente definiti. Se Adef vale 0 viene incrementata di uno ed AmigaBASIC utilizza le coordinate relative alla posizione del mouse nell'istante in cui è stato effettuato il click. Tali coordinate vengono memorizzate nelle variabili xa e ya. Queste due coordinate sono necessarie per tracciare la linea dal punto precedente a quello corrente. La linea di programma successiva controlla se le coordinate X e Y sono state modificate rispetto all'ultimo click. Nel caso non vi sia stata alcuna modifica, la definizione del poligono deve essere terminata (viene richiamata la sezione IniziaRiempi). Ogni volta che

viene definito un nuovo punto AmigaBASIC incrementa Adef di uno. Quando Adef raggiunge il valore 20 la definizione del poligono viene terminata eseguendo la routine IniziaRiempi. Al termine, ad xa e ya vengono assegnati i vecchi valori. La routine IniziaRiempi riporta Adef a zero, seleziona (utilizzando il comando COLOR) il colore scelto per il riempimento e disegna l'oggetto definito.

Cerchi ed ellissi

Fino ad ora si è lavorato semplicemente con forme rettangolari o comunque poligonali; è ora giunto il momento di realizzare forme curvilinee. La routine seguente consente di disegnare cerchi ed ellissi:

```
Cerchi:¶
  Test=MOUSE(0)¶
  x1=MOUSE(3) : y1=MOUSE(4)¶
  Puntatore(0,0)=x1 : Puntatore(0,1)=y1¶
  Puntatore(1,0)=x1 : Puntatore(2,1)=y1¶
  Puntatore(3,0)=x1 : Puntatore(4,1)=y1¶
  Value=5¶
  WHILE MOUSE(0)<>0¶
    r1= ABS(x1-MOUSE(5))¶
    r2= ABS(y1-MOUSE(6))¶
    Puntatore(1,1)=y1-r2 : Puntatore(2,0)=x1+r1¶
    Puntatore(3,1)=y1+r2 : Puntatore(4,0)=x1-r1¶
    GOSUB PiazzaPunto¶
  WEND¶
  IF r1=0 THEN r1=.1¶
  IF r1<r2 THEN Fattore=(r2/r1) : r1=r1*Fattore :
     r2=r2*Fattore¶
  CIRCLE (x1,y1),r1,ColoreDisegno,,,(r2/r1)¶
RETURN¶
```

Per creare cerchi ed ellissi con il programma di disegno, è necessario posizionare il puntatore nel punto centrale della figura circolare che si desidera disegnare.

- Premere il pulsante sinistro del mouse e spostare il mouse stesso.

La distanza orizzontale dal centro fornisce il raggio lungo le X, mentre la distanza verticale fornisce il raggio lungo le Y. In corrispondenza dei punti che risultano essere i più lontani dal centro è possibile vedere quattro puntatori. Un quinto puntatore è posizionato in corrispondenza del centro. La posizione dei puntatori è mostrata nella figura 9.

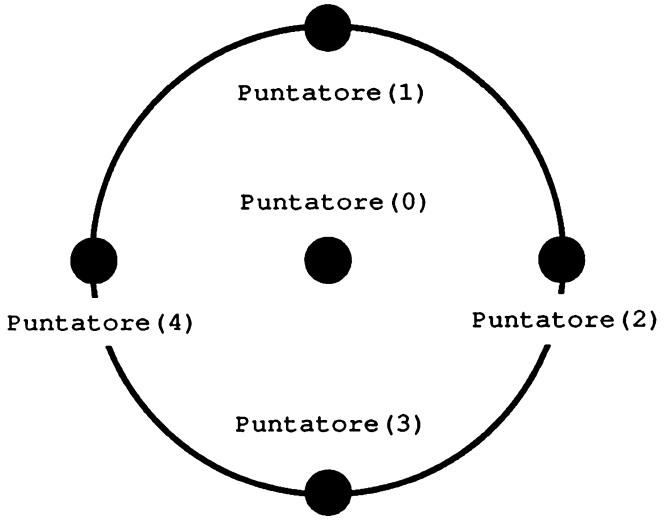


Figura 9: Identificazione delle posizioni dei puntatori sul cerchio

Il cerchio viene disegnato quando il pulsante del mouse viene rilasciato. Le coordinate del punto centrale sono memorizzate nella variabili x1 e y1. Confrontando la figura 9 con i valori memorizzati nel vettore Puntatore, risulta molto semplice comprendere il funzionamento della routine: ogni punto possiede una delle due coordinate identica al punto centrale. L'altro valore viene determinato durante il ciclo WHILE ... WEND. La variabile r1 contiene il valore assoluto (quindi sempre positivo) del raggio lungo le X, mentre la variabile r2 compie lo stesso per il raggio lungo le Y. Fino a quando il pulsante del mouse rimane premuto vengono visualizzati solamente i puntatori. Nel caso il raggio lungo le X sia nullo (=0), viene automaticamente posto a 0,01, per evitare di incorrere nell'errore "Division By Zero".

E' ora necessaro determinare il rapporto tra i due raggi da utilizzare per il comando CIRCLE. Normalmente tutte le operazioni vengono effettuate se

il valore per le X risulta essere maggiore od uguale a quello per le Y. Nel caso questo non sia vero (IF r1<r2 ...) è necessario calcolare il rapporto inverso. Dopo aver effettuato questa operazione viene utilizzato il comando CIRCLE per disegnare il cerchio (o l'ellisse).

Ecco ora la routine che realizza la colorazione delle aree contornate. Il comando chiave risulta essere il comando PAINT:

```
Riempi:¶
Test=MOUSE(0)¶
IF Click=0 THEN¶
  Click=1¶
  SOUND 440,6,200¶
  x=MOUSE(1) : y=MOUSE(2)¶
  RETURN¶
ELSE¶
  Click=0¶
  IF ABS(x-MOUSE(1))<11 AND ABS(y-MOUSE(2))<6 THEN¶
    PAINT (x,y),ColoreRiempi,ColoreDisegno¶
  ELSE¶
    SOUND 440,6,200¶
  END IF¶
END IF¶
RETURN¶
```

Riempimento di oggetti

Si sarà probabilmente notata la combinazione insolita dei blocchi IF ... THEN ... ELSE ... ENDIF. Fino ad ora si è visto solamente il blocco IF ... THEN, che consente di prendere decisioni in seguito al verificarsi di una condizione. ELSE dopo IF ... THEN consente di specificare il da farsi nei casi in cui la condizione non è verificata.

Spesso è necessario eseguire più di un comando all'interno di un blocco IF ... THEN. Fino ad ora tali comandi sono stati scritti tutti sulla stessa linea separati dal carattere : (due punti). Questo può andar bene fino a quando i comandi racchiusi nel blocco non diventano troppi, dato che la lunghezza massima ammessa per una linea è di 255 caratteri. Oltretutto troppi comandi sulla stessa linea possono generare confusione.

Per risolvere questo problema AmigaBASIC fornisce un modo per aumentare la dimensione di un blocco IF ... THEN ... ELSE ed inserire parti di programma lunghe quanto si vuole in tale blocco. E' però importante non inserire comandi sulla stessa linea dei comandi IF ... THEN ed ELSE. I comandi vengono eseguiti partendo dalla linea successiva. L'intero blocco viene completato con un comando END IF. La struttura risulta essere la seguente:

```
IF (condizione) THEN¶
   (sezione di programma¶
    da eseguire se la¶
    condizione è vera)¶
ELSE¶
    (sezione di programma ¶
    da eseguire se la¶
    condizione è falsa)¶
END IF¶
```

Come si può vedere nel programma di disegno, è possibile integrare tra loro strutture IF ... THEN/ELSE/END IF. Si consiglia però di indentare le varie strutture in modo da poterne valutare più facilmente il senso.

Ulteriori informazioni relative al riempimento

Per quale motivo è stata realizzata una routine tanto complicata per un semplice comando PAINT? E' utile ricordare quanto detto a proposito del programma Object Editor. Anche esso è un programma di disegno scritto in AmigaBASIC. Quando si colora un'area utilizzando l'Object Editor è abbastanza semplice distruggere l'intero contenuto della finestra: è sufficiente selezionare il colore di contorno errato, oppure dimenticare che la funzione PAINT è attiva. Dopo aver passato esperienze simili a questa, è abbastanza logico cercare di salvaguardare i propri programmi da ogni tipo di utilizzo errato. Ecco una buona idea per cercare di ovviare a questo inconveniente:

Il Workbench utilizza un doppio click per completare una selezione. Nel presente programma viene utilizzata la stessa modalità. Di conseguenza, quando si desidera colorare un'area, è necessario posizionare il puntatore all'interno dell'area stessa ed efettuare un doppio click. Dopo il primo click viene emesso un suono di avvertimento (si tratterà il comando

SOUND più avanti nel libro). Nel caso ci si fosse scordati che la modalità Fill è attiva, questo suono lo ricorda. A questo punto è possibile scegliere un'altra modalità di disegno, nel qual caso non succede proprio niente. Solamente se viene effettuato un secondo click nella stessa posizione l'area viene colorata.

Controllo dello stato del mouse

Per determinare quante volte è stato premuto il pulsante del mouse, il numero di click viene memorizzato nella variabile Click. La routine funziona nel modo seguente.

Alla prima esecuzione si ha Click=0. Se viene effettuato un click la variabile viene incrementata, viene prodotto il suono di avvertimento, il programma memorizza le coordinate della posizione del puntatore e viene eseguito un RETURN. Il colore di riempimento RiempiColore ed il colore del bordo ColoreDisegno possono essere specificati nella routine di selezione dei colori.

Quando viene effettuato un secondo click, viene richiamata nuovamente la routine Riempi. Questa volta, dato che Click vale 1, viene eseguito il blocco ELSE. Click viene nuovamente posta a zero; se la posizione del puntatore non è mutata (cosa determinabile comparando i valori attuali con quelli memorizzati nelle variabili x ed y), l'area viene colorata. Se invece le coordinate sono cambiate, si ha un altro suono di avvertimento ed il programma effettua un RETURN. Se si desidera ancora colorare l'area è necessario effettuare un doppio-click nella nuova posizione. Questa operazione richiede una certa pratica per tenere il mouse perfettamente fermo mentre si sta premendo il pulsante, ma si tratta di un piccolo prezzo da pagare tenendo conto della sicurezza offerta.

Se si desidera consentire un piccolo movimento del mouse è sufficiente modificare la seconda linea IF ... THEN nella routine Riempi come segue:

```
IF ABS(x-MOUSE(1))<11 AND ABS(y-MOUSE(2))<6 THEN¶
```

Tale linea consente un movimento orizzontale di 10 pixel ed una deviazione verticale di 5 pixel. Ovviamente questo accresce però le possibilità di errore.

Cancellazione degli errori
Dato che si è ora acquistata una certa familiarità con tutti i metodi possibili per generare un grafico, è il momento di fornire la possibilità di cancellare eventuali errori o tratti imprecisi. A questo scopo viene utilizzata la routine Cancella:

```
Cancella:¶
  Test=MOUSE(0)¶
  WHILE MOUSE(0)<>0¶
    x=MOUSE(1):y=MOUSE(2)¶
    PATTERN ,Compatto%¶
    LINE (x,y)-(x+10,y+5),0,bf¶
    PATTERN ,ModoRiempi%¶
  WEND¶
RETURN¶
```

E' necessario utilizzare tale routine per rimuovere porzioni indesiderate del disegno. Il programma funziona nello stesso modo della routine per il disegno di linee grasse a mano libera, ad eccezione del fatto che ora viene utilizzato il colore 0 (cioè il colore di sfondo). Come risultato si ottiene la cancellazione di parti di disegno che, venendo colorate con il colore di sfondo, diventano parti di sfondo.

PATTERN
In questa routine compare un nuovo comando: PATTERN. Come dice il nome stesso PATTERN si riferisce ai fill pattern. E' necessario però sapere che PATTERN ,Compatto% comunica al programma di utilizzare lo schema solido. D'altro canto PATTERN ,ModoRiempi% implica l'utilizzo del pattern attualmente attivo.

Prima del comando che visualizza la gomma sullo schermo, è necessario selezionare il pattern desiderato, altrimenti la cancellazione verrà effettuata utilizzando il pattern attualmente attivo producendo come risultato l'esatto contrario del pattern stesso. Se però si desidera che tutto quello che vi è in un particolare punto scompaia, è necessario utilizzare due comandi PATTERN.

Non è il caso di preoccuparsi se questo ha creato un po' di confusione.

Il sottoprogramma che segue, il quale gestisce tre differenti modalità di disegno, costituisce la routine PiazzaPunto. Essa consente di creare il punto lampeggiante utilizzato dalle routine RettVuoto, RettPieno e Cerchi, e di visualizzarlo sullo schermo. La routine PiazzaPunto riceve le coordinate dagli altri sottoprogrammi mediante la matrice Puntatore.

```
PiazzaPunto:¶
  FOR x=0 TO Value-1¶
    xz=Puntatore(x,0):yz=Puntatore(x,1)¶
    IF xz<0 THEN xz=0 : Puntatore(x,0)=0¶
    IF xz>311 THEN xz=311 : Puntatore(x,0)=311¶
    IF yz<0 THEN yz=0 : Puntatore(x,1)=0¶
    IF yz>242 THEN yz=242 : Puntatore(x,1)=242¶
    AltroColore(x)=POINT(xz,yz)¶
  NEXT x¶
  FOR x=0 TO Value-1¶
    PSET (Puntatore(x,0),Puntatore(x,1)),
      - (AltroColore(x)=0)¶
  NEXT x¶
  FOR x=0 TO Value-1¶
    PSET (Puntatore(x,0),Puntatore(x,1)),
        AltroColore(x)¶
  NEXT x¶
RETURN¶
```

L'intera routine consiste di tre cicli FOR ... NEXT. Il primo ciclo calcola i punti e li memorizza. Il secondo ciclo visualizza i punti, il terzo li cancella nuovamente. La routine cicla tante volte quanti sono i punti. Dato che il contatore parte da zero, l'ultimo punto viene visualizzato quando il contatore raggiunge il valore Value-1. Le quattro linee IF ... THEN del primo ciclo controllano che le coordinate dei vari punti non fuoriescano dall'area consentita. Se il valore della X risulta minore di zero o maggiore di 311, oppure se il valore della Y risulta minore di zero o maggiore di 242, il valore eccedente viene posto uguale al limite consentito.

Successivamente il programma memorizza, nel vettore AltroColore, il colore del punto che si trovava originariamente nella posizione in cui viene visualizzato il puntatore. La funzione BASIC POINT(x,y) fornisce informazioni relative al colore del pixel specificato, oppure il valore -1 nel caso il

pixel si trovi al di fuori dell'area consentita.

1 uguale 1 .... o no

Il compito del secondo ciclo è quello di visualizzare il puntatore sullo schermo. La sola cosa difficile da comprendere è relativa al colore utilizzato che viene determinato con l'espressione -(AltroColore(x)=0). La ragione di questa espressione è molto semplice: si desidera infatti che il puntatore contrasti chiaramente con lo sfondo. Se il punto in cui viene posizionato il puntatore ha lo stesso colore dello sfondo (colore numero 0, cioè il nero), viene utilizzato per il puntatore il colore 1 (bianco). In caso contrario viene utilizzato per il puntatore il colore dello sfondo.

Si è ora spiegato cosa si vuole realizzare con tale formula, ma non ancora come essa viene utilizzata. L'intero processo deve essere ricondotto a come AmigaBASIC calcola le espressioni logiche. Un'espressione falsa assume valore zero. Una espressione vera ha valore -1. Si discuterà maggiormente di questo nella nota d'uso 3. Se lo si desidera è possibile inserire, per esercizio, la seguente linea nella finestra BASIC:

```
? (0=1)¶
```

Quello che si sta facendo è domandare ad Amiga se zero ed uno sono uguali. Amiga risponderà: no, non lo sono. Nel suo linguaggio questa affermazione risulta:

```
0
```

Se la domanda fosse:

```
? (1=1)¶
```

la risposta sarebbe:

```
-1
```

Questa risposta sancisce il fatto che effettivamente uno è uguale ad uno. Una espressione logica vera assume quindi il valore -1.

La formula -(AltroColore(x)=0) fornisce i seguenti risultati:

0 Se AltroColore non è uguale a zero, dato che -(espressione falsa) risulta -0, cioè 0.

1 Se AltroColore è uguale a zero, dato che -(espressione vera) risulta -(-1), cioè 1.

Come si può vedere si tratta di un concetto abbastanza complicato che può però essere risolto con una semplice formula.

L'ultimo ciclo FOR ... NEXT riporta il pixel al suo colore originario, facendo scomparire il puntatore dallo schermo.

Ecco ora l'ultima subroutine riguardante le differenti modalità di disegno che, a dir la verità, non definisce una modalità di disegno in quanto consente l'inserimento di testo nel disegno. Questa routine dovrebbe essere inserita immediatamente prima della routine PiazzaPunto, ma, dato che il programma è strutturato, non ha molta importanza il punto esatto di inserimento.

```
Testo:¶
  Test=MOUSE(0)¶
  x=MOUSE(1) : y=MOUSE(2)¶
  MENU OFF : MOUSE OFF¶
  MENU 1,0,0 : MENU 2,0,0¶
  WINDOW 5,"Immetti il testo:",(0,177)-
     (311,185),18,1¶
  CLS¶
  LINE INPUT Testo$¶
  WINDOW CLOSE 5¶
  WINDOW 2¶
  MENU 1,0,1 : MENU 2,0,1¶
  MENU ON : MOUSE ON¶
   LOCATE INT(y/8.86)+1,INT(x/10)+1 :
      COLOR ColoreDisegno,ColoreRiempi¶
  PRINT Testo$;¶
  COLOR ColoreDisegno,0¶
RETURN¶
```

Questa routine può essere utilizzata per etichettare parte del disegno realizzato.

• Effettuare un click nel punto in cui si desidera visualizzare testo.

Vicino al fondo dello schermo comparirà una finestra che è possibile spostare in qualsiasi punto.

• Inserire il testo che si desidera visualizzare e premere il tasto <Return>.

La finestra si chiuderà ed il testo verrà visualizzato nella posizione specificata.

Immissione del testo

La subroutine Testo ricava innanzitutto le coordinate del puntatore. La gestione dell'event trapping ed i menù a tendina vengono momentaneamente disabilitati, mentre sullo schermo viene visualizzata la finestra per l'immissione del testo. Il testo viene acquisito utilizzando il comando LINE INPUT, dato che tutti i caratteri inseriti devono essere utilizzati. Dopo che è stato inserito il testo, viene cancellata la finestra 5 e si riprende ad interagire con la finestra 2. Vengono inoltre riattivati i menù e l'event trapping. Prima che il testo venga visualizzato, è necessario trasformare le coordinate del punto in righe e colonne per il comando LOCATE. Per rendere più flessibile la gestione del colore del testo, il colore attuale viene memorizzato nella variabile ColoreDisegno, mentre il colore di sfondo è memorizzato in ColoreRiempi. In questo modo è possibile utilizzare testo colorato evidenziato con barre colorate. Se si desidera utilizzare il colore di sfondo per il proprio testo, è sufficiente scegliere il colore 0 come colore di riempimento. Prima di uscire dalla subroutine è necessario ristabilire la situazione standard per i colori utilizzando il comando COLOR ColoreDisegno,0. In caso contrario, se si tenta di ripulire lo schermo, questo verrà riempito completamente con il colore di riempimento.

Con questa routine si conclude la parte relativa al disegno. E' necessario ricordarsi di procedere al salvataggio del programma.

Controllo del colore

E' ora necessario realizzare la routine per la gestione del colore. Questa routine verrà richiamata in seguito alla selezione dell'opzione Palette Colori del menù a tendina Programma. Essa si compone di due sezioni distinte. La prima parte viene utilizzata per selezionare il colore, mentre la

seconda sezione consente di definire le componenti RGB di ogni singolo colore.

• Inserire il listato seguente:

```
SelezioneColore:¶
  ColorePrescelto=0 : OkFine=0¶
  MOUSE OFF : MENU OFF¶
  WINDOW 3,"Palette Colori",(4,20)-(245,160),18,1¶
  PATTERN ,Compatto%¶
  FOR x= 1 TO (MaxColori+1)/8¶
    FOR y= 0 TO 7   ¶
      LINE (y*30,(x-1)*16)-((y+1)*30,x*16),
           (x-1)*8+y,bf¶
    NEXT y¶
  NEXT x¶
  LINE (10,70)-(50,93),ColoreDisegno,b¶
  LINE (15,73)-(45,90),ColoreDisegno,bf¶
  LOCATE 13,2 : COLOR 0,1 : PRINT "Disegno";¶
  LINE (70,70)-(110,93),ColoreRiempi,b¶
  LINE (75,73)-(105,90),ColoreRiempi,bf¶
  LOCATE 13,10 : COLOR 1,0 : PRINT "Riempimento";¶
  LINE (140,70)-(235,93),1,b¶
  LOCATE 11,21: PRINT "Palette";¶
  LINE (199,114)-(230,132),1,b¶
  LOCATE 16,27 : PRINT "OK";¶
  PATTERN ,ModoRiempi%¶
  MOUSE ON : MENU ON¶
RETURN¶
¶
¶
PaletteColori:¶
  Test=MOUSE(0)¶
  x=MOUSE(3) : y=MOUSE(4)¶
¶
  GOSUB ScegliColore¶
¶
```

```
  PATTERN ,Compatto%¶
  LINE (10,70)-(50,93),ColoreDisegno,b¶
  LINE (15,73)-(45,90),ColoreDisegno,bf¶
  LINE (70,70)-(110,93),ColoreRiempi,b¶
  LINE (75,73)-(105,90),ColoreRiempi,bf¶
  PATTERN ,ModoRiempi%¶
¶
  IF WINDOW(0)=3 AND 69<y AND y<94 THEN¶
    IF 69<x AND x<111 THEN ColorePrescelto=1¶
    IF 9<x AND x<51 THEN ColorePrescelto=0 ¶
    IF 139<x AND x<236 THEN¶
      PATTERN ,Compatto%¶
      PAINT (142,72),3,1¶
      PATTERN ,ModoRiempi%¶
      GOSUB DefPalette¶
      RETURN¶
    END IF¶
  END IF¶
  GOSUB TestaOk¶
¶
  IF ColorePrescelto=0 THEN¶
    LOCATE 13,2 : COLOR 0,1 : PRINT "Disegno";¶
    LOCATE 13,10 : COLOR 1,0 : PRINT "Riempimento";¶
  ELSE¶
    LOCATE 13,2 : COLOR 1,0 : PRINT "Disegno";¶
    LOCATE 13,10 : COLOR 0,1 : PRINT "Riempimento";¶
  END IF¶
RETURN¶
¶
ScegliColore:¶
  IF WINDOW(0)=3 AND x<240 AND y<(2^(Colori+1)) THEN¶
    fx=INT(x/30) : fy = INT(y/16)¶
  IF ColorePrescelto=0 THEN
      ColoreDisegno=fy*8+fx¶
    ELSE¶
      ColoreRiempi=fy*8+fx¶
```

```
    END IF¶
  END IF¶
RETURN¶
¶
TestaOk:¶
  IF x>198 AND x<231 AND y>113 AND y<133 THEN¶
    PATTERN ,Compatto%¶
    PAINT (201,116),3,1 : OkFine=1¶
    PATTERN ,ModoRiempi%¶
  END IF¶
RETURN¶
¶
OkColori:¶
  MENU 2,0,1 : Modo=1¶
  WINDOW CLOSE 3¶
  WINDOW OUTPUT 2¶
RETURN¶
```

Dopo aver richiamato questa routine viene visualizzata una nuova finestra, di dimensioni minori rispetto a quella utilizzata per il disegno, contenente alcuni riquadri colorati, uno per ogni colore disponibile. Effettuando un click in uno di questi riquadri si seleziona il colore corrispondente.

Sotto questi riquadri vi sono altri quattro riquadri. I primi due vengono utilizzati per indicare i valori correnti per i colori di disegno e di riempimento. Il colore di riempimento risulta importante solamente per il comando PAINT, dato che, in questo caso, devono essere specificati un colore per i bordi e uno di riempimento.

Effettuando un click su uno di questi due riquadri è possibile specificare quale dei due colori si desidera modificare. L'etichetta relativa al riquadro selezionato viene visualizzata in negativo. I riquadri sono essi stessi visualizzati con il colore attivo.

Il gadget Palette consente di attivare la modalità di definizione dei colori. Dato che questa parte di programma non è ancora stata inserita, è necessario non selezionare tale modalità. Effettuando un click sul gadget OK è possibile uscire dalla subroutine.

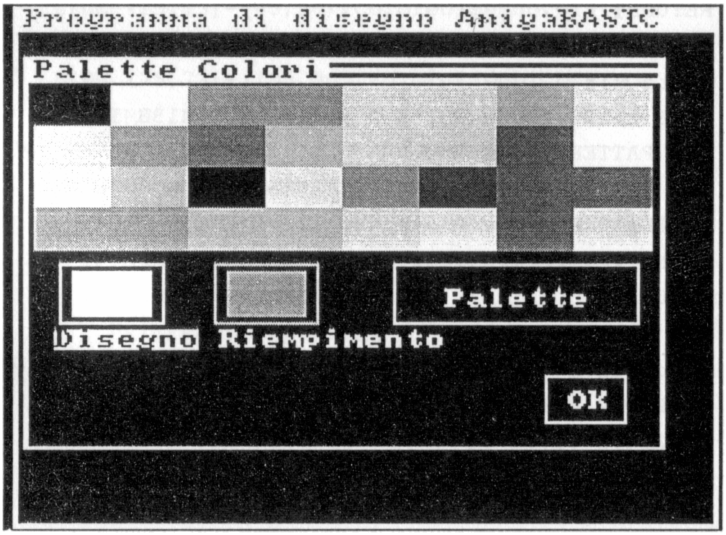


Figura 10: La finestra Palette Colori utilizzata nel programma di disegno.

Funzionamento della routine per la gestione del colore
La routine SelezioneColori genera la finestra, vi posiziona tutti gli accessori richiesti e realizza tutti i preparativi necessari. Le variabili ColorePrescelto e OkFine vengono poste a zero. La variabile ColorePrescelto consente di stabilire se è stato modificato il colore di disegno (ColorePrescelto=0) oppure il colore di riempimento (ColorePrescelto=1). OkFine=0 consente di disabilitare l'event trapping mentre viene costruita la finestra. E' necessario agire in questo modo in quanto viene cambiata la finestra di output e si potrebbero quindi ottenere risultati indesiderati (porzioni di testo divise tra le due finestre). Per questo motivo vengono utilizzati i comandi MENU OFF e MOUSE OFF. La nuova finestra viene poi visualizzata sullo schermo.

Per costruire i riquadri colorati è necessario utilizzare il pattern solido e si fa quindi ricorso al comando PATTERN ,Compatto%. I due cicli nidificati FOR ... NEXT che seguono consentono di inserire i colori nei riquadri. Se sono disponibili otto colori (3 bitplane), viene realizzata una riga con otto riquadri; se i colori disponibili sono 16, si avranno due righe da otto

riquadri; nel caso di 32 colori le righe saranno quattro. L'espressione (MaxColori+1)/8 consente di calcolare il numero di righe, fornendo il valore 1 nel caso di tre bitplane, 2 nel caso di 4 e 4 nel caso di 5 bitplane. Anche i gadget Palette e OK vengono costruiti in questa routine. Prima di eseguire il comando RETURN, il fill pattern viene riportato al valore originario e viene riattivato l'event trapping.

Il mouse e la routine PaletteColori

Quando è attiva la modalità 2, la routine PaletteColori è responsabile della gestione di eventuali click. Prima di tutto, vengono determinate le coordinate del mouse. Viene poi richiamata la routine ScegliColore che controlla se è stato selezionato uno dei riquadri e, in caso affermativo, il colore selezionato diviene l'attuale colore di disegno o di riempimento (a seconda del valore di ColorePrescelto).

Quando il programma ritorna dalla routine ScegliColore, vengono ridisegnati i riquadri relativi ai colori attivi, nel caso questi siano stati modificati. Per fare questo viene utilizzato il pattern contenuto nella variabile Compatto%.

Nel caso questo non sembri, per il momento, avere molto senso, non è il caso di preoccuparsi: tra poco verrà descritta la routine ModoRiempi nella quale saranno chiariti tutti gli eventuali dubbi.

E' già stata descritta la funzione svolta dalla routine ScegliColore: essa controlla se è stato selezionato uno dei riquadri colorati e, in caso affermativo, modifica il colore di disegno o quello di riempimento. Il blocco IF ... THEN contiene la funzione WINDOW(0). Come già per il comando MOUSE, anche il comando WINDOW può essere utilizzato per ottenere informazioni. WINDOW(0) fornisce il numero della finestra selezionata, cioè della finestra all'interno della quale è stato premuto il pulsante del mouse. Dopo tutto, è necessario sapere se l'utente ha effettuato un click nella finestra corretta prima di valutare la posizione del puntatore, in quanto è possibile che sia stata selezionata un'altra finestra.

Una delle principali regole nell'utilizzo del calcolatore stabilisce che l'utente può commettere ogni possibile tipo di errore; è compito del programmatore intervenire per evitare che questi errori abbiano conseguenze disastrose per il programma.

La subroutine viene eseguita solamente se viene effettuato un click all'interno della finestra 3 e all'interno di un riquadro colorato.

Controllo dei bitplane

Vengono poi confrontate le coordinate Y con la formula (2^(Colori+1)), il cui risultato dipende dal numero di colori disponibili. Il limite inferiore è differente in dipendenza dal numero di righe di riquadri. Se vi è una sola riga, y deve essere minore di 16; nel caso di due righe il limite è 32, mentre nel caso di 4 righe, il bordo è a 64. I riquadri colorati sono infatti alti esattamente 16 pixel. La variabile Colori contiene il numero di bitplane. Se non si comprende esattamente come viene calcolata questa formula si consiglia di inserire direttamente in essa i valori assunti dalla variabile Colori (cioè 3, 4 e 5) ed osservare i risultati ottenuti.

La variabile fx contiene la posizione orizzontale del riquadro selezionato, cioè il numero di riquadri a sinistra di quello selezionato. La variabile fy specifica la riga in cui si trova il riquadro. Da questi due valori è possibile ricavare il numero del colore scelto utilizzando la formula: fy*8+fx.

A seconda del valore di ColorePrescelto tale colore viene poi assegnato alla variabile ColoreDisegno oppure a ColoreRiempi. L'etichetta TestaOK è abbastanza evocativa: essa controlla infatti se è stato effettuato un click sul gadget OK, nel qual caso viene assegnato il valore 1 alla variabile OkFine e si ha il ritorno dalla routine. Viene poi eseguita la routine OkColori che termina il sottoprogramma per la gestione del colore, il menù a tendina Strumenti viene riabilitato e Modo viene posto a 1. La finestra per la selezione del colore (Window 3) scompare dallo schermo e la finestra 2 (contenente il disegno in fase di realizzazione) ritorna ad essere la finestra di output.

Selezione dei colori

Dopo avere inserito queste linee di programma è possibile effettuare la scelta tra 8, 16 o 32 colori. Potrebbe tuttavia capitare che questi colori non siano sufficienti per il disegno che si sta realizzando. Si supponga infatti di avere bisogno di numerose sfumature di verde per realizzare un paesaggio di campagna e di non avere assolutamente bisogno di colori come l'arancione, il blu scuro oppure il rosa. In questo caso è possibile utilizzare la seguente sezione di programma per modificare le componenti RGB

di ogni singolo colore.

• Inserire la routine seguente tra le subroutine PaletteColori e ScegliColore.

```
DefPalette:¶
  IF ColorePrescelto=0 THEN NuovoColore=ColoreDisegno
     ELSE NuovoColore=ColoreRiempi¶
  PATTERN ,Compatto%¶
  LINE (0,69)-(240,107),0,bf¶
  COLOR 1,0¶
  LOCATE 10,2 : PRINT "R";¶
  LOCATE 11,2 : PRINT "G";¶
  LOCATE 12,2 : PRINT "B";¶
  LINE (24,72)-(218,78),1,b¶
  LINE (24,80)-(218,86),1,b¶
  LINE (24,88)-(218,94),1,b¶
  LINE (222,72)-(238,94),NuovoColore,bf¶
  Modo=4¶
  PATTERN ,ModoRiempi%¶
RETURN ¶
¶
RGBDef:¶
  Test=MOUSE(0)¶
  x=MOUSE(3) : y=MOUSE(4)¶
  GOSUB ScegliColore¶
  IF ColorePrescelto=0 THEN NuovoColore=ColoreDisegno
     ELSE NuovoColore=ColoreRiempi¶
  GOSUB RegolatoreRGB¶
  GOSUB TestaOk : IF OkFine=1 THEN OkFine=3¶
  WHILE MOUSE(0)<>0¶
    x=MOUSE(1) : y=MOUSE(2)¶
    IF WINDOW(0)=3 AND x>23 AND x<219 AND y>71 AND
         y<95 THEN¶
      Colori%(NuovoColore,INT((y-71)/8.7))=
          INT((x-26)/12)¶
      GOSUB RegolatoreRGB¶
    END IF¶
```

```
  WEND¶
RETURN¶
¶
RegolatoreRGB:¶
  PATTERN ,Compatto%¶
  LINE (25+r*12,73)-(37+r*12,77),0,bf¶
  LINE (25+g*12,81)-(37+g*12,85),0,bf¶
  LINE (25+b*12,89)-(37+b*12,93),0,bf¶
  r=Colori%(NuovoColore,0)¶
  g=Colori%(NuovoColore,1)¶
  b=Colori%(NuovoColore,2)    ¶
  LINE (25+r*12,73)-(37+r*12,77),1,bf¶
  LINE (25+g*12,81)-(37+g*12,85),1,bf¶
  LINE (25+b*12,89)-(37+b*12,93),1,bf¶
  PALETTE NuovoColore,r/16,g/16,b/16¶
  LINE (222,70)-(238,98),NuovoColore,bf¶
  PATTERN ,ModoRiempi%¶
RETURN¶
```

La prima di queste tre subroutine, DefPalette, è responsabile di tutti i preparativi e della costruzione del nuovo schermo. A seconda del valore della variabile ColorePrescelto, alla variabile NuovoColore viene assegnato il numero del colore di disegno o di riempimento. Questo colore può essere modificato nelle sue componenti nelle tre bande fino a quando non viene selezionato un altro colore. I cursori per il rosso, il verde ed il blu vengono visualizzati nella finestra Colori.

Dato che è necessario avere a disposizione una certa quantità di spazio per poter visualizzare tali cursori, è necessario cancellare dalla finestra i riquadri relativi ai colori di disegno e riempimento correnti. Per effettuare questa operazione viene sovrapposto a tali riquadri un blocco con il colore dello sfondo (colore numero 0) e vengono poi disegnati i tre cursori etichettati R, G e B nello spazio così liberato. Infine viene disegnato un piccolo riquadro per contenere il colore selezionato, in modo da mostrare quale sia il colore in fase di modifica.

Modo vale in questo momento 4.

Controllo dell'input da mouse
La routine RGBDef valuta l'input del mouse fino a quando rimane attiva la routine di definizione dei colori. Dopo aver determinato le coordinate del puntatore, la routine ScegliColore controlla se è stato selezionato un colore. In questo modo è possibile determinare quale sia il colore da modificare. Il nuovo colore viene immediatamente assegnato alla variabile NuovoColore.

La subroutine RegolatoreRGB posiziona poi correttamente i cursori in base alle definizioni effettuate.

Per controllare se viene premuto il gadget per la conferma viene utilizzata la routine TestaOK. In caso affermativo OkFine viene posta a uno. Tale valore viene poi incrementato a tre (per realizzare il segnale relativo alla terminazione del programma per la definizione del colore). Il programma torna quindi in modalità due, cioè nella modalità relativa alla selezione del colore. Effettuando un altro click sul gadget OK è possibile ritornare nella modalità di disegno.

E' possibile sveltire il ritorno alla modalità di disegno selezionando direttamente l'opzione Disegno dal menù a tendina Programma.

Spostamento dei cursori
Il ciclo WHILE ... WEND nella routine RGBDef realizza lo spostamento automatico dei cursori. Mentre il pulsante del mouse rimane premuto, una delle tre componenti (R, G o B) viene costantemente modificata. La finestra attiva risulta essere la numero 3 ed il puntatore rimane all'interno delle barre di controllo. Quale sia la componente in regolazione dipende dalla posizione del puntatore lungo le Y. Il valore della componente viene invece determinato dalla posizione lungo le X. Prima che venga completato il blocco IF ... THEN ed il ciclo WHILE ... WEND, viene richiamata la routine RegulatoreRGB che utilizza i contenuti delle variabili r, g e b per posizionare correttamente i cursori.

Per prima cosa la routine provvede a cancellare i cursori dalle vecchie posizioni, ridisegnandoli con il colore dello sfondo. Successivamente carica o esamina i contenuti di r, g, b nel vettore Colori%, ottenendo numeri compresi tra 0 e 15. Zero indica l'assenza di componente di quel colore, mentre 15 indica la saturazione. I cursori vengono poi ridisegnati nelle nuove po-

sizioni. Utilizzando il comando PALETTE viene comunicata ad AmigaBASIC la necessità di procedere alla ridefinizione del colore. Il colore appena definito viene poi visualizzato nel riquadro relativo al colore attivo. Viene poi eseguito il comando RETURN che consente di ritornare al punto di chiamata della routine.

E' possibile a questo punto ridefinire altri colori. La gestione dei cursori risulta, come è possibile notare, simile a quella delle Preferences e di molti programmi professionali. Dopo avere modificato i colori secondo il proprio gradimento, è possibile uscire dalla routine per la definizione dei colori effettuando un click sul gadget OK.

Come si potrà notare risulta sicuramente più semplice utilizzare il programma che seguire semplicemente le spiegazioni contenute nel libro. Dopo tutto nella vita è sempre così; si provi infatti a pensare a quanta documentazione si potrebbe scrivere relativamente ad una bicicletta, e come sia invece semplice utilizzarla.

Nota d'uso 3: bit, byte ed altri misteri

E' giunto ancora una volta il momento di interrompere un attimo il processo di apprendimento; è il momento giusto per prendersi una piccola pausa, sgranchirsi un attimo le dita per poi continuare a leggere, ma prima di questo è necessario procedere a salvare su dischetto tutto il lavoro precedentemente svolto.

La parte successiva del programma di disegno riguarda la definizione dei modelli di riempimento (Fill Pattern). Per consentire di comprendere pienamente la programmazione e la gestione dei vari pattern, è necessario esporre alcuni principi matematici fondamentali. In questa nota d'uso verranno analizzati i numeri binari, i numeri esadecimali e tutto ciò che è possibile fare con essi.

Numeri binari

E' gia stato fatto notare come un calcolatore sia in grado di distinguere solamente tra due differenti condizioni: 0 e 1, corrispondenti alla presenza od assenza di tensione e, dal punto di vista logico, alla condizione di verità o falsità. Tutte le operazioni svolte da un calcolatore sono realizzate gestendo le combinazioni di queste due possibili condizioni.

Ci si potrebbe chiedere a questo punto come Amiga sia in grado di ricordare un numero diverso da 0 o da 1. Il numero 2, ad esempio, fornisce il primo problema, dato che non può essere immediatamente associato alla condizione di presenza o assenza di tensione. Per quanto detto nei capitoli precedenti la soluzione al problema non dovrebbe essere troppo difficile. Come infatti era già stato fatto per poter utilizzare più di due colori, Amiga utilizza più bit per trattare numeri più grandi di 1. Con un solo bit è possibile gestire solamente due numeri: 0 e 1. Con due bit si possono distinguere quattro diversi numeri: 0, 1, 2 e 3. I numeri da 0 a 7 vengono rappresentati da Amiga come riportato nella tabella seguente:

Numero	Cod. binaria
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Tabella 6: Numeri decimali e binari

Il sistema di numerazione utilizzato da Amiga è il *sistema di numerazione binario.* In tale sistema vengono utilizzate solamente le due cifre 0 e 1, ma è comunque possibile rappresentare tutti i numeri rappresentabili con il *sistema decimale.*

Utilizzo dei numeri binari

Il sistema di numerazione binario risulta molto usato in campo informatico. Il sistema decimale standard utilizza 10 cifre (0, 1, 2, ..., 9). Quando è necessario rappresentare un numero più grande viene aggiunta una cifra davanti alle altre. Una sola cifra è sufficiente per rappresentare i numeri da 0 a 9, ma per i numeri superiori a 10 sono necessarie due cifre. I valori di ogni singola cifra risultano essere sempre potenze della base del sistema di numerazione (1=10^0, 10=10^1, 100=10^2, etc.).

Ci si potrebbe chiedere a questo punto come mai sia stato scelto proprio il numero 10 come base per il sistema di numerazione. A pensarci bene non vi è alcuna ragione particolare. Probabilmente è stata fatta questa scelta in quanto, dato che si ha sempre bisogno di qualche aiuto in campo matematico, e visto che le dita delle mani sono 10, questo poteva essere un buon aiuto. Comunque è possibile costruire sistemi di numerazione su

una qualsiasi base: 3, 8, oppure, se si vuole, 127.

Come è semplice contare in binario: 1, 10, 11
I calcolatori conoscono solamente due numeri (0 e 1) e per queso motivo devono lavorare con il sistema di numerazione binario. Per rappresentare il numero decimale 2 in binario è necessaria una seconda cifra. Lo stesso si verifica per ogni altro esponente di due: 2^0=1, 2^1=2, 2^2=4, 2^3=8, 2^4=16, etc. Il sistema di numerazione binario richiede l'aggiunta di una nuova cifra per ognuno di questi numeri. Lo svantaggio sta nel fatto che un numero relativamente piccolo richiede un numero elevato di cifre. Ma, se si eccettua questo effetto collaterale, tutto funziona come nel caso del sistema decimale. Se si desidera calcolare il valore di un numero binario, è sufficiente calcolare il valore relativo ad ogni singola cifra e sommare i valori calcolati tra loro.

Ad esempio, il numero binario 10011011 può essere facilmente interpretato: esso corrisponde al valore decimale 128+16+8+2+1=155.

Byte, parole (Word) e parole lunghe (Longword)
Per ua ragione tecnica otto bit vengono normalmente raggruppati in una unità chiamata *byte*. Un byte risulta composto di otto bit, per cui il valore massimo che può assumere è 255. Per controllare questa affermazione è sufficiente sommare tutti i valori decimali nell'esempio precedente.

Ognuna delle celle della memoria di Amiga è in grado di memorizzare un byte. Questa caratteristica discende dai tempi in cui i microprocessori erano in grado di gestire solamente otto bit alla volta (il Commodore 64 ed il 128, ad esempio, utilizzavano processori ad otto bit). Il processore centrale di Amiga, il 68000, può elaborare 16 bit per volta. Un valore a 16 bit viene denominato anche parola (*Word*). Una parola consiste di due byte memorizzati in due celle consecutive di memoria. Utilizzando 16 bit è possibile rappresentare numeri fino a 65535. Il processore 68000 può gestire anche numeri a 32 bit. Valori di questo tipo vengono definiti Parole Lunghe (*Longword*) e consentono di rappresentare numeri fino a 4292967294.

I termini Word e Longword non verranno più incontrati nel resto del libro: sono stati segnalati semplicemente in modo che si possa riconoscerli se vengono incontrati in qualsiasi altra occasione.

Numeri esadecimali

Spesso vengono utilizzati solamente 15 o 31 bit per rappresentare un numero, mentre l'ultimo bit viene utilizzato per il segno del numero. In questi casi 0 indica un numero positivo, 1 un numero negativo. AmigaBASIC lavora nel modo seguente: i numeri a 16 bit risultano compresi tra -32768 e 32767, mentre quelli a 32 bit sono compresi tra -2147483647 e 2147485646. Questi numeri decimali risultano piuttosto difficili da interpretare direttamente in binario, per cui i programmatori preferiscono utilizzare un ulteriore sistema di numerazione collegato al sistema binario, cioè il *sistema esadecimale*. Questo sistema utilizza come base il numero 16. Le lettere da A a F vengono utilizzate per rappresentare i numeri decimali da 10 a 15 ed il numero 16 è rappresentato come 10.

Decimale	Binario	Esadecimale
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Tabella 7: Un confronto tra i tre principali sistemi di numerazione utilizzati in informatica.

Vantaggio del sistema esadecimale
Il sistema esadecimale ha alcuni piccoli vantaggi rispetto agli altri sistemi per quanto riguarda la sua applicazione ai calcolatori: per prima cosa è possibile rappresentare numeri più grandi con meno cifre. Inoltre, 16 risulta essere una potenza di due. Questo significa che è possibile facilmente passare dalla rappresentazione esadecimale a quella binaria e viceversa. Gruppi di quattro cifre di un numero binario coincidono con una cifra esadecimale, come mostrato nella tabella precedente.

Ad esempio, risulta molto più semplice determinare quanti bit sono utilizzati dal numero esadecimale FFFF piuttosto che dalla sua rappresentazione decimale 65535.

Notazione esadecimale
Se si desidera utilizzare i numeri in notazione esadecimale in AmigaBASIC, è necessario specificare tale richiesta in qualche modo. E' necessario perciò far precedere il numero da una & (e commerciale) e da una h:

```
? &h7fff
```

Notazione ottale
Dato che si sta parlando di sistemi di numerazione, è il caso di segnalare che AmigaBASIC è in grado di riconoscere un'altra notazione: la *notazione ottale*. In questo sistema di numerazione la base è 8 e le cifre sono quindi comprese tra 0 e 7. Ad esempio:

```
? &o7777
```

I numeri ottali non verranno quasi mai utilizzati. D'altro canto verranno utilizzati abbastanza spesso i numeri esadecimali ed i numeri binari. Sfortunatamente AmigaBASIC non possiede una modalità che consenta la visualizzazione diretta dei numeri binari. Per questo motivo verranno utilizzati principalmente numeri decimali ed esadecimali all'interno dei programmi AmigaBASIC con i quali è possibile eseguire ogni funzione matematica:

```
? &hf + &hao2
```

Questa linea corrisponde a 15+2562 ed il risultato è 2577. La notazione utilizzata nella linea non ha alcuna importanza per AmigaBASIC, l'unica cosa

che conta sono i valori. E' quindi possibile utilizzare anche entrambi i sistemi all'interno della stessa linea:

```
? 15 + &ha02
```

Operatori logici

Vi sono molte cose she possono essere effettuate con questi tipi di numeri; ad esempio, possono essere utilizzati gli *operatori logici.* Due esempi di operatori logici sono gia stati incontrati ed utilizzati all'interno di un blocco IF ... THEN: l'operatore AND e l'operatore OR.

Un AND logico significa "sia uno che l'altro", cioè entrambe le condizioni devono essere vere perchè sia vera l'espressione.

Un OR logico implica "almeno una delle due"; affinchè sia vera l'espressione è sufficiente che sia vera una delle condizioni.

Operatori come AND e OR sono utilizzati per calcolare operazioni logiche. In questo processo di calcolo vengono confrontati tra loro i singoli bit di due valori. Come si è già avuto modo di vedere AmigaBASIC esprime il valore delle espressioni logiche utilizzando 0 e -1. Si vedrà ora di spiegare come viene calcolato il risultato di tali espressioni.

AND

Si inizierà con l'operatore AND.

• Inserire la linea seguente nella finestra BASIC:

```
? 59 AND 93
```

Amiga visualizza come risultato il numero 25. Ecco come viene svolto il calcolo:

```
          00111011 (59)
AND       01011101 (93)
          --------
          00011001 (25)
```

Nel risultato, un bit è posto a 1 solamente se i corrispondenti bit di entrambi gli operandi sono posti a 1. Lo schema seguente illustra i risultati delle combinazioni che si possono verificare:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Quando vengono valutate espressioni logiche con numeri interi, AmigaBASIC compara uno ad uno i singoli bit di tali numeri e li pone nelle posizioni corrispondenti per il risultato, come si può facilmente notare nell'esempio mostrato in precedenza per l'operatore AND.

OR

L'operatore logico OR pone a uno un bit se il bit corrispondente di almeno uno dei due operandi è 1. L'esempio seguente:

```
? 77 OR 132
```

fornisce come risultato 205. Il calcolo viene svolto nel modo seguente:

```
          01001101 (77)
OR        10000100 (132)
          --------
          11001101 (205)
```

Le combinazioni possibili con l'operatore OR sono le seguenti:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

XOR

Vi sono altri operatori logici; uno di questi è l'operatore XOR, il cui nome deriva da "Exclusive or". Il funzionamento è uguale a quello dell'operatore OR ad eccezione del fatto che il risultato è 0 nel caso in cui entrambi i bit siano 1 (la corrispondenza nel funzionamento può essere fatta con aut .. aut in latino).

```
90 XOR 213
```

fornisce come risultato 143.

```
          01011010 (90)
XOR       11010101 (213)
          ----------
          10001111 (143)
```

Ecco le combinazioni per l'operatore XOR:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

EQV

L'operatore logico EQV (il nome è una abbreviazione di EQuiValent) consente di valutare l'uguaglianza tra bit ponendo a 1 il risultato se i bit comparati hanno lo stesso valore. Questo operatore non viene utilizzato molto di frequente e non è nemmeno semplice come i precedenti, dato che in questo caso il carattere principale del numero (positivo o negativo) deve essere preso in considerazione:

```
? 106 EQV -42
```

fornisce come risultato 67:

```
                0001101010 (106)
EQV             1111010110 (-42)
                ----------
                0001000011 (67)
```

Le possibili combinazioni sono:

0 EQV 0 = 1
0 EQV 1 = 0
1 EQV 0 = 0
1 EQV 1 = 1

IMP

Questo operatore risulta essere un pò complicato. Il nome IMP costituisce una troncatura del termine IMPlied utilizzato nelle espressioni matematiche. Comparando i valori tale operatore pone il secondo bit in accordo con il primo. Praticamente questo operatore non verrà mai utilizzato; per cercare di spiegarne il funzionamento ecco comunque un esempio e le tabelle delle combinazioni possibili:

```
? 370 IMP -474
```

fornisce come risultato -373.

```
                0101110010 (370)
IMP             1000100110 (-474)
                ----------
                0001000011 (-337)
```

Le combinazioni dell'operatore IMP sono:

0 IMP 0 = 1
0 IMP 1 = 1
1 IMP 0 = 0
1 IMP 1 = 1

NOT

L'ultimo operatore logico che rimane da analizzare è l'operatore NOT. Tale operatore commuta tra loro valori logici: il valore 0 diventa -1, e -1 diventa 0 (è necessario ricordare che il valore di verità viene espresso dal numero -1. Questo non è molto consistente e sembrerebbe più logico utilizzare il valore 1 come controparte dello zero; purtroppo, però non è possibile intervenire in tal senso). NOT calcola il nuovo valore utilizzando la formula seguente:

```
nuovo_valore = - vecchio_valore - 1
```

In questo modo 0 viene cambiato in 1 e viceversa. Questa operazione non ha molto senso quando viene utilizzata con altri numeri. Perchè, del resto, NOT 2 dovrebbe essere -3? NOT viene perciò utilizzato quasi esclusivamente per la valutazione delle condizioni per i blocchi IF ... THEN o in condizioni simili. Le due linee BASIC che seguono hanno entrambe il medesimo risultato:

```
IF NOT (a$="Ciao") THEN ...
```

e

```
IF a$<>"Ciao" THEN ...
```

A questo punto si conosce tutto ciò che è necessario riguardo a bit, byte, sistemi di numerazione ed operatori logici. Tutte queste conoscenze verranno utilizzate più avanti nel libro, quando si comincerà a programmare per proprio conto.

Se questa nota d'uso è sembrata un pò noiosa, nel prossimo paragrafo si inizieranno a mettere in pratica alcune delle cose discusse utilizzandole per completare il programma di disegno.

2.10 Il Blitter ed il programma di disegno: la definizione dei vari Fill Pattern

I fill pattern, come del resto molte altre operazioni grafiche di Amiga, sono gestiti direttamente dal chip Blitter. Oltre a poter copiare e colorare aree, il Blitter è anche in grado di eseguire operazioni più complicate. Esso utilizza gli stessi operatori logici discussi nel paragrafo precedente per svolgere queste operazioni. Per il Blitter risulta abbastanza semplice l'operazione di riempimento di un blocco o di un poligono creato con il comando AREA, utilizzando qualsiasi tipo di simbolo grafico, come ad esempio piccoli cuori, invece che tinte unite. E' possibile disegnare ogni tipo di pattern; tutto quello che il Blitter richiede è che vengano specificati i dati che descrivono questo pattern.

Disegnare modelli utilizzando il comando PATTERN
Per gestire le operazioni relative ai pattern, AmigaBASIC mette a disposizione il comando PATTERN, già incontrato nel listato precedente. I dati che descrivono il modello devono essere specificati di seguito al comando PATTERN:

```
PATTERN (valore per le linee),(vettore per l'area)
```

E' possibile specificare un pattern in prima posizione che può poi essere utilizzato per disegnare le singole linee. Poichè questo pattern viene specificato come un intero a 16 bit, si hanno a disposizione 16 punti contigui per la descrizione del proprio modello per la singola linea. Ad esempio, se si desidera realizzare linee secondo lo schema un punto sì un punto no, è necessario specificare i valori seguenti:

```
Binario:            01010101 01010101
Decimale :          21845
Esadecimale :       &H5555
```

• **Inserire la linea seguente nella finestra BASIC:**

```
pattern &h5555¶
```

Line Pattern
Dopo avere inserito questa istruzione, il nuovo pattern fornito viene utilizzato per disegnare le linee. Un bit con valore 1 rappresenta un pixel visibile, mentre un bit con valore 0 corrisponde ad un pixel invisibile.

- Provare ad inserire nella finestra BASIC i comandi per disegnare alcune linee:

```
FOR x=1 TO 1000 : LINE (635*RND,185*RND) -
    (635*RND,185*RND) : NEXT¶
```

E' necessario osservare molto attentamente queste ultime linee per riuscire a notare la separazione tra i pixel. Utilizzando un differente pattern:

```
PATTERN &hcccc
```

si dovrebbe essere in grado di notare maggiormente la separazione.

Il comando PATTERN consente di inserire uno schema di bit e di utilizzarlo successivamente nelle operazioni grafiche con i normali comandi AmigaBASIC. Il resto è svolto direttamente da Amiga.

Si è appena appreso qualcosa di nuovo: un fill pattern, o un line pattern, rimane attivo fino a quando non viene sostituito da uno nuovo. Il pattern base di Amiga corrisponde al valore esadecimale &HFFFF, cioè "tutti i punti visibili".

Line pattern e fill pattern
E' necessario porre molta attenzione nell'operazione di colorazione di aree: se si è infatti effettuato il disegno utilizzando il pattern definito precedentemente, il colore sborderà dalle linee invadendo le aree circostanti. Per questo motivo non è stata inserita nel programma di disegno la possibilità di definizione dei pattern per le linee, in quanto le linee vengono per lo più utilizzate per disegnare i contorni delle varie aree da colorare. E' invece possibile definire Fill pattern per la colorazione delle aree utilizzando quindi il secondo parametro del comando PATTERN. La sola differenza,

rispetto al caso precedente, riguarda il fatto che, per la definizione di pattern piani, è necessario specificare un vettore di interi invece di un singolo valore:

```
PATTERN ,(nome_vettore)
```

Come si può vedere, nel comando PATTERN della linea sopra presentata vi è una virgola prima del parametro specificato in quanto il primo parametro, quello relativo al fill pattern, è stato omesso. All'interno del programma di disegno era già stata effettuata la medesima cosa per i pattern Compatto% e ModoRiempi%.

Gli elementi del vettore devono essere numeri interi, in quanto la conversione di numeri frazionari o reali in bit risulterebbe abbastanza complicata. Inoltre i fill pattern devono avere una larghezza di 16 pixel (quindi ogni elemento deve essere un intero a 16 bit); per quanto riguarda l'altezza in pixel del pattern non vi sono limitazioni, ad eccezione del fatto che deve comunque essere una potenza di due (cioè 2, 4, 8, 16, 32, etc.). Nel programma di disegno è stata utilizzata un'altezza di 8 pixel per i pattern. Le aree che possono essere colorate risultano naturalmente più larghe, in quanto il pattern viene semplicemente ripetuto fino al riempimento completo dell'area.

Nel programma di disegno è possibile scegliere tra nove differenti pattern, oppure modificarne uno per produrne uno personalizzato.

• Inserire l'ultima parte del listato del programma di disegno:

```
EditorModi:¶
  MOUSE OFF : MENU OFF¶
  OkFine=0¶
  WINDOW 4,"Modi di riempimento",(54,30) -
    (300,130),18,1¶
  LINE (0,0)-(132,66),3,b¶
¶
  FOR x=0 TO 2 ¶
    FOR y=0 TO 2 ¶
      FOR i=0 TO 7: ModoRiempi%(i)=OgniModo%(y*3+x,i)
        : NEXT i¶
      PATTERN ,ModoRiempi%¶
```

```
      LINE (144+x*34,y*25)-(175+x*34,23+y*25),1,bf¶
    NEXT y¶
  NEXT x¶
  GOSUB MarcaModo¶
¶
  LINE (5,68)-(74,80),1,b¶
  LOCATE 10,2 : PRINT "Cancella";¶
  LINE (77,68)-(140,80),1,b¶
  LOCATE 10,11: PRINT "Inverti";¶
  LINE (5,85)-(74,97),1,b¶
  LOCATE 12,3 : PRINT "Carica";¶
  LINE (77,85)-(140,97),1,b¶
  LOCATE 12,12: PRINT "Salva";¶
  LINE (162,77)-(207,90),1,b¶
  LOCATE 11,23 : PRINT "OK";¶
¶
  FOR i=0 TO 7 : ModoRiempi%(i)=OgniModo%(NumModo,i) :
      NEXT i¶
  GOSUB DisegnaModo¶
¶
  MENU ON : MOUSE ON¶
RETURN¶
¶
DefinisciModo:¶
  Test=MOUSE(0)¶
  x=MOUSE(3) : y=MOUSE(4)¶
  IF WINDOW(0)=4 AND x<132 AND y<66 THEN¶
    px=INT(x/8.25) : py=INT(y/8.25)¶
    Bit=ModoRiempi%(py) AND 2^(15-px)¶
    IF Bit=0 THEN¶
      ModoRiempi=ModoRiempi%(py) OR 2^(15-px)¶
    ELSE¶
      ModoRiempi=ModoRiempi%(py) AND (65535& -
          2^(15-px))¶
    END IF¶
    IF ModoRiempi>32767 THEN ModoRiempi=
```

```
          ModoRiempi-65536&¶
      ModoRiempi%(py)=ModoRiempi¶
      PATTERN ,Compatto%¶
      LINE (px*8+4,py*8+2)-(px*8+9,py*8+8),-(Bit=0),bf¶
      PATTERN ,ModoRiempi%¶
      y1=INT(NumModo/3) : x1=NumModo-y1*3¶
      LINE (144+x1*34,y1*25)-(175+x1*34,23+y1*25),1,bf¶
     FOR i=0 TO 7 : OgniModo%(NumModo,i)=ModoRiempi%(i)
         : NEXT i¶
      RETURN¶
    END IF ¶
    IF WINDOW(0)=4 AND x>142 AND x<244 AND y<75 THEN¶
      px=INT((x-143)/34) : py=INT(y/25)¶
      IF px+py*3=NumModo THEN RETURN¶
      NumModo=px+py*3¶
     FOR i=0 TO 7 : ModoRiempi%(i)=OgniModo%(NumModo,i)
         : NEXT i¶
      GOSUB MarcaModo¶
      GOSUB DisegnaModo¶
      PATTERN ,ModoRiempi%¶
      RETURN¶
    END IF ¶
¶
    IF WINDOW(0)=4 AND x<222 AND x>162 AND y<93 AND
          y>76 THEN¶
      PATTERN ,Compatto%¶
      PAINT (164,78),2,1¶
      PATTERN ,ModoRiempi%¶
      OkFine=2 : RETURN¶
    END IF ¶
¶
    IF WINDOW(0)=4 AND x<135 AND y>68 AND y<100 THEN¶
      PATTERN ,Compatto%¶
      IF x<75 AND x>4 AND y<81 THEN¶
        PAINT (7,70),2,1¶
        LINE (1,1)-(131,65),0,bf¶
```

```
      FOR i=0 TO 7 : OgniModo%(NumModo,i)=0 :
           ModoRiempi%(i)=0 : NEXT¶
      PAINT (7,70),0,1¶
      PATTERN ,ModoRiempi%¶
      y1=INT(NumModo/3) : x1=NumModo-y1*3¶
      LINE (144+x1*34,y1*25)-(175+x1*34,
          23+y1*25),1,bf¶
    END IF ¶
    IF x<141 AND x>76 AND y<81 THEN¶
      PAINT (79,70),2,1¶
      FOR i=0 TO 7¶
        ModoRiempi%(i)=ModoRiempi%(i) XOR &HFFFF¶
        OgniModo%(NumModo,i)=ModoRiempi%(i)¶
      NEXT i¶
      GOSUB DisegnaModo¶
      PAINT (79,70),0,1¶
      PATTERN ,ModoRiempi%¶
      y1=INT(NumModo/3) : x1=NumModo-y1*3¶
      LINE (144+x1*34,y1*25)-(175+x1*34,
          23+y1*25),1,bf¶
    END IF¶
    IF x<75 AND x>4 AND y>84 THEN GOSUB CaricaModo¶
    IF x<141 AND x>76 AND y>84 THEN GOSUB SalvaModo¶
  END IF¶
RETURN¶
¶
MarcaModo:¶
  y1=INT(AltPattern/3) : x1=AltPattern-y1*3¶
  LINE (143+x1*34,y1*25-1)-(176+x1*34,24+y1*25),0,b¶
  y1=INT(NumModo/3) : x1=NumModo-y1*3¶
  LINE (143+x1*34,y1*25-1)-(176+x1*34,24+y1*25),3,b¶
  AltPattern=x1+y1*3¶
RETURN¶
¶
DisegnaModo:¶
  MOUSE OFF : MENU OFF¶
```

```
  PATTERN ,Compatto%¶
  LINE (1,1)-(131,65),0,bf ¶
  FOR y=0 TO 7¶
    FOR x=0 TO 15¶
      Bit=ModoRiempi%(y) AND 2^(15-x)¶
      IF Bit<>0 THEN LINE (x*8+4,y*8+2)-(x*8+9,
           y*8+8),1,bf¶
    NEXT x¶
  NEXT y¶
  PATTERN ,ModoRiempi%¶
  MOUSE ON : MENU ON¶
RETURN¶
¶
CaricaModo:¶
  MOUSE OFF : MENU OFF¶
  PAINT (7,87),2,1¶
  GOSUB ImmettiNome¶
  IF Nome$="" THEN FineCaricaModo¶
  OPEN Nome$ FOR INPUT AS 1¶
    FOR x=0 TO 8¶
      FOR y=0 TO 7¶
        OgniModo%(x,y)=CVI(INPUT$(2,1))¶
      NEXT y¶
    NEXT x¶
  CLOSE 1¶
¶
FineCaricaModo:¶
  WINDOW CLOSE 5 : WINDOW 4¶
  PAINT (7,87),0,1¶
  MOUSE ON : MENU ON¶
  FOR x=0 TO 8¶
    FOR y=0 TO 7¶
      ModoRiempi%(y)=OgniModo%(x,y)¶
      PATTERN ,ModoRiempi%¶
      y1=INT(x/3) : x1=x-y1*3¶
     LINE (144+x1*34,y1*25)-(175+x1*34,
```

```
          23+y1*25),1,bf¶
      NEXT y¶
    NEXT x¶
    FOR i=0 TO 7 : ModoRiempi%(i)=OgniModo%(NumModo,i):
      NEXT i¶
    GOSUB DisegnaModo¶
RETURN¶
¶
SalvaModo:¶
    MOUSE OFF : MENU OFF¶
    PAINT (79,87),2,1¶
    GOSUB ImmettiNome¶
    IF Nome$="" THEN FineCaricaModo¶
    OPEN Nome$ FOR OUTPUT AS 1¶
      FOR x=0 TO 8¶
        FOR y=0 TO 7¶
          PRINT #1,MKI$(OgniModo%(x,y));¶
        NEXT y¶
      NEXT x¶
    CLOSE 1¶
¶
FineSalvaModo:¶
    WINDOW CLOSE 5 : WINDOW 4¶
    PAINT (79,87),0,1¶
    MOUSE ON : MENU ON¶
RETURN¶
¶
OkModo:¶
    MENU 2,0,1 : Modo=1¶
    WINDOW CLOSE 4¶
    WINDOW OUTPUT 2¶
    PATTERN ,ModoRiempi%¶
RETURN¶
¶
ImmettiNome:¶
    AltroNome$=Nome$¶
```

```
    WINDOW 5,"Immetti il nome:",(0,80)-(311,88),0,1¶
    CLS¶
    LINE INPUT Nome$¶
    IF Nome$= "=" OR Nome$="*" THEN Nome$=AltroNome$¶
RETURN¶
```

A questo punto si dovrebbe avere abbastanza familiarità con i comandi grafici dell'AmigaBASIC per poter facilmente comprendere il listato inserito.

Come funziona il Pattern Editor

Le prime linee realizzano una finestra e posizionano i gadget in essa contenuti. La routine EditorModi è responsabile di questa procedura. Durante questi preparativi viene disabilitato l'event trapping. Risulta ora possibile vedere i nove fill pattern disponibili all'interno della finestra appena aperta. All'interno dei cicli nidificati essi vengono memorizzati nel vettore OgniModo% e visualizzati sullo schermo. Il simbolo % indica che gli elementi del vettore sono interi. Gli otto schemi vengono prima copiati dal vettore OgniModo% nel vettore ModoRiempi%, il quale contiene il pattern attualmente attivo. La subroutine disegna poi un riquadro per ogni singolo pattern.

Viene poi richiamata la subroutine MarcaModo che disegna un contorno arancione attorno al riquadro del pattern attualmente attivo, in modo che questo risulti riconoscibile facilmente. Le linee successive costruiscono i restanti gadget: Cancella, Inverti, Carica, Salva ed OK. Per attivare una delle corrispondenti funzioni è sufficiente effettuare un click su tali gadget.

Per finire viene richiamata la routine DisegnaModo che disegna l'immagine del pattern attivo nel riquadro grande, all'interno del quale è possibile intervenire per apportare modifiche al pattern.

Viene poi riattivato l'event trapping e la subroutine viene terminata. La figura seguente mostra come appare la finestra per la definizione dei Fill Pattern nel programma di disegno.

Figura 11: La finestra per la definizione dei fill pattern nel programma di disegno.

Definizione dei Pattern

La routine DefinisciModo risponde al mouse fino a quando è attiva la modalità 3. Per prima cosa vengono determinate le coordinate relative alla posizione del puntatore che vengono poi memorizzate nelle variabili x ed y. Viene successivamente controllato passo per passo, il campo in cui è stato effettuato un click e l'operazione che deve essere realizzata dal programma.

Nel riquadro grande viene mostrato un ingrandimento del pattern attivo. E' possibile modificare ogni singolo pixel di tale pattern effettuando un click su di esso. Se la finestra attiva è la numero 4, e viene effettuato un click nel riquadro grande, è necessario determinare il particolare pixel selezionato. Le variabili px e py convertono le coordinate della posizione del muose in colonne e righe per specificare la posizione del quadrato selezionato.

La variabile Bit consente di isolare il bit del vettore ModoRiempi% corrispondente al pixel selezionato. Per effettuare questa operazione viene utilizzato l'operatore logico AND.

La variabile ModoRiempi contiene il nuovo valore assegnato al vettore ModoRiempi%. Per assegnare il nuovo valore ad un vettore di interi è necessario sottrarre il numero 65535 a tutti i numeri superiori a 32767. In questo modo si ottengono valori negativi che possono poi essere assegnati al vettore ModoRiempi%.

Questo completa la spiegazione matematica della definizione dei vari pattern.

Il risultato di questa formula viene utilizzato per disegnare il colore.

Per fare in modo che l'utente possa vedere come il pattern compare nella sua dimensione naturale, viene riprodotto il pattern nel riquadro originale. Modificando semplicemente il contenuto del vettore ModoRiempi% non viene modificata anche l'immagine del pattern sullo schermo: è quindi necessario provvedere a ridisegnare il riquadro utilizzando un comando LINE con parametro bf.

Quando viene selezionato un nuovo pattern, o caricato un nuovo insieme di pattern da dischetto, i pattern sullo schermo rimangono immutati; questo si verifica in quanto il Blitter utilizza i pattern locali all'immagine quando disegna lo schermo. E' possibile ottenere lo stesso risultato modificando a mano ogni singolo pixel del pattern, ma questo richiede una grande quantità di tempo.

La routine DefinisciModo posiziona anche il contenuto modificato del vettore ModoRiempi% nell'appropriato elemento del vettore OgniModo%.

Cambiare il pattern attivo

La sezione successiva del programma consente di ridefinire il pattern corrente. E' sufficiente effettuare un click nel riquadro del pattern che si desidera utilizzare oppure modificare. Il riquadro verrà allora evidenziato con la contornatura arancione e il suo contenuto verrà ricopiato nell griglia di lavoro (il riquadro grande). La variabile NumModo contiene il numero del pattern attualmente selezionato. Se viene riselezionato lo stes-

so pattern il programma esegue direttamente il comando RETURN risparmiando così tempo.

Gli otto bit di dati vengono copiati all'interno del vettore ModoRiempi% dal vettore OgniModo% e vengono richiamate le routine MarcaModo e DisegnaModo. Attorno al riquadro contenente il pattern selezionato viene disegnata la contornatura arancione ed il pattern viene ricopiato nella griglia di lavoro. Viene poi attivato il nuovo pattern utilizzando il comando PATTERN.

La sezione successiva richiede di effettuare un click sul gadget OK. Risulta molto semplice: la variabile OkFine viene posta a 2 e la routine termina. La routine TestaMouse che richiama tale subroutine si preoccupa di gestire il valore ritornato.

Controllo dell'interazione con i gadget

Tutto quello che resta da discutere riguarda la gestione dei vari gadget. Il blocco IF ... THEN più esterno controlla se le coordinate del puntatore, quando viene effettuato un click, sono all'interno di uno dei gadget. Se questo non si verifica, il programma continua la sua esecuzione normalmente; il blocco IF ... THEN più interno controlla le coordinate di ogni gadget, uno di seguito all'altro.

Effettuare un click sul gadget Cancella. Il riquadro attivo viene ripulito. Come al solito viene utilizzato il pattern Compatto% ed il comando LINE con il parametro bf ed il colore di sfondo per realizzare la cancellazione del contenuto del riquadro. Cancellare parte di un disegno significa infatti semplicemente ricoprire tale parte con il colore dello sfondo. Gli otto valori interi del pattern cancellato vengono azzerati nei vettori ModoRiempi% ed OgniModo%. Il gadget Cancella, che era stato evidenziato, viene poi riportato in condizioni normali. Per finire, il nuovo pattern (vuoto) viene disegnato nel riquadro.

Invertire un pattern

La routine associata al gadget Inverti funziona nella stessa maniera. Viene utilizzata la funzione XOR per ricevere il valore negativo di un pattern. Utilizzando XOR &HFFFF i valori 0 ed 1 del pattern vengono scambiati tra loro. I nuovi valori vengono assegnati ai vettori ModoRiempi% ed Ogni-

Modo%, esattamente come nel caso precedente. Viene poi eseguita la routine DisegnaModo per visualizzare nella griglia di lavoro il pattern in negativo.

Se vengono selezionati i gadget Carica e Salva, vengono richiamate le routine CaricaModo e SalvaModo. Eseguite queste routine si ha il ritorno al programma principale.

La routine MarcaModo disegna per prima cosa un riquadro (con il colore dello sfondo) attorno al pattern precedentemente selezionato, e successivamente un riquadro arancione attorno al pattern attivo. Il numero del nuovo pattern è fornito dalla variabile NumModo. Per memorizzare il numero del pattern precedente viene utilizzata la variabile AltPattern.

Non dovrebbe esservi alcuna difficoltà nella comprensione della routine DisegnaModo: essa consente di visualizzare pixel per pixel un fill pattern all'interno della griglia. L'event trapping viene disabilitato in modo che la subroutine non venga interrotta durante questo processo. Il contenuto precedente della griglia viene cancellato disegnando un riquadro pieno con il colore dello sfondo. Per realizzare la griglia vengono controllati i singoli valori di ModoRiempi%: per ogni bit acceso viene disegnato un quadrato. Viene infine riattivato l'event trapping.

Salvare e caricare pattern

Rimane ancora un ultimo dettaglio: chiaramente non si desidera che tutto il lavoro effettuato vada perso e quindi nel programma è stata inserita la possibilità di salvare su dischetto e recuperare successivamente i pattern definiti.

Caricare Fill Pattern

La routine successiva del programma di disegno è responsabile del caricamento dei pattern. La routine CaricaModo disabilita innanzitutto l'event trapping, in quanto non devono avvenire interruzioni durante l'esecuzione di tale routine. Il comando PAINT nella seconda linea riempie il gadget Carica in modo da evidenziarlo per fornire una conferma visiva dell'avvenuta selezione. Quando viene chiamata la routine ImmettiNome, Amiga richiede l'immissione del nome del file da cui caricare i pattern, nome che viene memorizzato nella variabile Nome$. Se viene premuto il tasto <Re-

turn> senza inserire alcun nome, Nome$ risulta vuoto e si ha quindi l'immediato richiamo della routine FineCaricaModo.

La routine FineCaricaModo si preoccupa di terminare correttamente la routine precedente. Se viene inserito correttamente il nome di un file, questo viene aperto in lettura. Ulteriori informazioni sul comando OPEN verranno fornite nel Capitolo 3 e nel Capitolo 5. Per ora tutto quello che è necessario sapere è che i valori per i pattern vengono caricati nel vettore OgniModo%. Questo viene effettuato nei due cicli FOR ... NEXT nidificati. Il comando CLOSE è ovviamente il compagno naturale del comando OPEN. La routine FineCaricaModo termina correttamente la procedura.

Per l'inserimento del nome viene aperta la finestra 5, che viene poi richiusa in modo tale che la finestra 4 ritorni ad essere la finestra corrente. Viene poi riattivato nuovamente l'event trapping, per poter visualizzare i nuovi pattern sullo schermo. Fino ad ora si era proceduto solamente a memorizzare i relativi valori all'interno del vettore OgniModo%. Per effettuare la visualizzazione, due cicli nidificati copiano, ad uno ad uno, i valori da questo vettore nel vettore ModoRiempi%. Contemporaneamente vengono visualizzati sullo schermo i pattern all'interno di nove riquadri. Dopo questi due cicli al vettore ModoRiempi% viene assegnato il valore del pattern attivo.

Salvare Fill Pattern

Le due subroutine per il salvataggio dei fill pattern risultano abbastanza semplici e facili da comprendere. La routine SalvaModo disabilita per prima cosa l'event trapping, evidenzia poi il gadget Salva e utilizza infine la routine ImmettiNome per determinare il nome del file in cui salvare i pattern. Il file specificato viene poi aperto in scrittura ed al suo interno vengono scritti, valore per valore, gli elementi del vettore OgniModo%. Il file viene poi richiuso e viene eseguita la routine FineSalvaModo per terminare correttamente la procedura. La finestra 5 deve essere chiusa e deve essere riattivata la finestra 4. Viene poi riportato in condizioni normali il gadget Save e viene riattivato l'event trapping. Alcuni dei comandi contenuti in tale routine verranno discussi dettagliatamente nai capitoli 3 e 5.

Salvataggio di pattern

Tutti e nove i pattern definiti vengono salvati in un file su dischetto. Effettuando un click sul gadget Salva si ottiene la visualizzazione di un requester

(avente la larghezza dello schermo ed alto solamente una linea) utilizzato per specificare il nome del file in cui si desidera salvare il pattern.

Si consiglia di utilizzare nomi corti ed abbastanza significativi per i propri file. Premendo il tasto <Return> si termina l'inserimento del nome, il disk drive si mette in funzione ed il file viene salvato su dischetto

Attenzione
Ricordarsi di non estrarre il dischetto fino a quando non si è spenta la luce del drive.

Terminato il salvataggio il programma ritorna alla routine di definizione dei pattern.

Caricare Pattern
Per caricare un file contenente i nove pattern è sufficiente effettuare un click sul gadget Carica causando la visualizzazione di un requester per l'inserimento del nome analogo al precedente. Il programma carica poi i pattern dal file specificato e li utilizza come pattern base per il programma di disegno.

Il listato del programma continua con la routine OkModo, molto simile alla routine OkColori nella sezione per la gestione del colore. Tale routine, che termina correttamente la procedura di definizione dei vari pattern, viene eseguita quando viene effettuato un click sul gadget OK.

OkModo riattiva il secondo menù a tendina, pone uguale ad 1 la modalità, chiude la finestra aperta dalla subroutine per la definizione dei pattern e rende attiva la finestra numero 2. Attiva infine l'ultimo pattern selezionato e ritorna al programma principale. Si è ora in grado di continuare il disegno o selezionare un nuovo strumento.

Deve ancora essere analizzata la routine ImmettiNome che consente di specificare il file da aprire. Utilizzando il comando CLS viene per prima cosa attivato il requester di inserimento al cui interno viene visualizzato il cursore. Il programma richiede ora l'inserimento di un nome. Inserendo il carattere = o il carattere * si conferma come nome il nome precedentemente memorizzato in AltroNome$.

Ecco ora l'ultima parte del programma:

```
Richiesta:¶
  MENU 1,0,0 : MENU 2,0,0¶
  MENU OFF : MOUSE OFF¶
  WINDOW 5,"ATTENZIONE!",(43,70)-(270,125),0,1¶
  COLOR 0,1 : CLS : LOCATE 2,2¶
  PRINT "Vuoi proseguire senza"¶
  PRINT " registrare il lavoro ?"¶
  PATTERN ,Compatto%¶
  LOCATE 6,13 : PRINT "Sì";¶
  LOCATE 6,21 : PRINT "No";¶
  LINE (80,36)-(127,50),0,b¶
  LINE (145,36)-(192,50),0,b¶
  SOUND 880,6,100¶
Attesa:¶
  Test=MOUSE(0)¶
  WHILE MOUSE(0)=0¶
    x=MOUSE(1) : y=MOUSE(2)¶
  WEND¶
  IF (y<51 AND y>35) THEN¶
    IF (x<128 AND x>79) THEN PAINT (82,38),3,0 : OK=1:
         GOTO FineRichiesta ¶
   IF (x<193 AND x>144) THEN PAINT (147,38),3,0 : OK=0:
         GOTO FineRichiesta¶
  END IF¶
  GOTO Attesa ¶
FineRichiesta:¶
  MENU ON : MOUSE ON : MENU 1,0,1 : MENU 2,0,1¶
  WINDOW CLOSE 5 : WINDOW 2¶
RETURN¶
¶
FinisciTutto:¶
  MENU RESET¶
  SCREEN CLOSE 1¶
END¶
```

Conferma di eventuali cancellazioni
Questa routine esegue i controlli per la conferma della decisione di cancellare una figura, oppure di uscire dal programma. Come al solito viene utilizzato un requester, viene disattivato l'event trapping ed i menù a tendina, in modo che l'utente non possa selezionare alcuna opzione.

Il titolo della finestra è "ATTENZIONE!" per assicurarsi che l'utente sappia che questa è l'ultima possibilità a disposizione per evitare errori irreparabili. Nella finestra viene visualizzato il messaggio "Vuoi proseguire senza registrare il lavoro?" e due gadget per la specifica della risposta. Mentre viene visualizzata la finestra viene prodotto un breve suono di avvertimento. Il ciclo Attesa viene eseguito fino a quando non viene effettuato un click su uno dei due gadget. A seconda del gadget selezionato, alla variabile OK viene assegnato il valore 0 oppure 1. Il sottoprogramma reagisce di conseguenza.

La routine FineRichiesta risulta abbastanza semplice: essa termina la routine Richiesta abilitando nuovamente l'event trapping ed i menù a tendina.

Fine del Programma
Rimane ancora una piccolissima routine: la routine FinisciTutto. Il comando MENU RESET consente di ristabilire i menù originali, mentre il comando SCREEN CLOSE consente di chiudere lo schermo con cui si stava lavorando. Il comando END termina poi il programma.

Il comando END porta alla terminazione corretta del programma.

Nel caso del programma in esame, tale comando avrebbe anche potuto essere evitato dato che, se il programma dovesse raggiungere la linea seguente, in cui non vi è nessun comando GOTO o GOSUB, l'effetto sarebbe identico. Vi sono comunque alcuni buoni motivi per utilizzare END per terminare il programma: innanzitutto risulta in tal modo possibile identificare immediatamente la fine del programma sul listato; inoltre, nel caso vi fossero ulteriori linee dopo la fine del programma, queste non verrebbero eseguite evitando così possibili errori.

Debug del programma
Durante l'utilizzo del programma si potrebbero verificare degli errori; il

programma potrebbe fermarsi e potrebbe ricomparire il listato nella finestra LIST con la linea in cui si è verificato l'errore posta in evidenza, all'interno di un riquadro arancione.

E' possibile ricontrollare sul libro tale linea e correggere l'errore.

Se invece il programma funziona. ma non come dovrebbe, diventa più difficile risolvere il problema.

Trucchetti vari nel programma di disegno
Per concludere ecco alcuni piccoli trucchi per l'utilizzo del programma. Dopo aver lanciato il programma sono necessari alcuni secondi prima che esso diventi operativo. Durante questi istanti vengono svolte tutte le operazioni di preparazione, dopo di che il programma è effettivamente pronto per essere utilizzato.

Informazioni sull'utilizzo di testo
Per finire alcune informazioni sull'utilizzo del testo all'interno del disegno: questa opzione dovrebbe essere utilizzata molto attentamente, dal momento che risulta possibile rovinare tutta la figura. Per la visualizzazione del testo viene utilizzato il colore di disegno, mentre per lo sfondo viene utilizzato il colore di riempimento. In questo modo è possibile sovrapporre testo ad aree piene. L'opzione testo non può comunque utilizzare fill pattern. Il testo ricopre qualsiasi cosa sottostante in quanto è prioritario rispetto alla grafica. Se si desidera scrivere sullo schermo vuoto è necessario scegliere come colore di riempimento il colore dello sfondo.

3 Organizzazione dei dati: gestione dei file e dei dischetti

Dopo avere imparato ad utilizzare le possibilità grafiche e di animazione dell'AmigaBASIC, è utile analizzare una serie di comandi che si rivelano molto importanti quando si sta lavorando con grosse quantità di dati.

I dati sono normalmente memorizzati su *dischetti* chiamati floppy disk. Amiga può utilizzare differenti tipi di dischetti. Si analizzerà questo punto più avanti utilizzando alcuni utili programmi.

Come prima cosa si procederà a creare un dischetto che possa essere usato esclusivamente con AmigaBASIC ed i suoi programmi. Fino ad ora è stato utilizzato il dischetto Extras, ma non è possibile continuare in questo modo ancora per molto, per non esaurire lo spazio a disposizione sul dischetto per la memorizzazione.

3.1 Salvare il lavoro per il futuro: realizzare un proprio dischetto BASIC

Fino ad ora, dopo aver aperto il dischetto Extras, sono stati memorizzati solamente pochi programmi. E' stato creato un cassetto nel quale sono stati salvati i programmi realizzati, senza preoccuparsi però del processo che veniva eseguito.

Nel frattempo, il dischetto Extras ha cominciato a riempirsi. Esso era già abbastanza pieno prima di iniziare a salvare i programmi realizzati, dato che AmigaBASIC, il cassetto BasicDemos ed il programma AmigaTutor in esso contenuti occupano una grande quantità di spazio.

Requester

Non vi dovrebbero essere stati problemi fino a questo punto; tuttavia, è possibile che, durante la fase di salvataggio di un programma, il dischetto Extras sia stato trovato pieno, nel qual caso è stato visualizzato un appropriato requester.

I requester sono prodotti dal Workbench. Nel caso in questione all'interno del requester dovrebbe essere visualizzato il messaggio: Volume Extras 1.2 is full, indicante che sul dischetto non vi è spazio sufficiente per completare il salvataggio del programma.

Effettuando un click sul gadget Cancel si conferma al Workbench di aver ricevuto il messaggio. Se dopo aver effettuato questa operazione viene nuovamente visualizzato il requester, probabilmente la versione di Workbench che si sta utilizzando presenta un errore. Se si dovesse verificare questo, effettuare nuovamente un click sul gadget Cancel.

Come se non bastasse l'avvertimento ricevuto, anche AmigaBASIC mostra un messaggio di errore: Disk Full, in risposta al quale è necessario effettuare un click sul gadget OK.

Preparare un dischetto di lavoro
Nel caso si sia verificato quanto sopra discusso, sul dischetto Extras non vi è più spazio per memorizzare programmi. Anche se non è ancora capitato, probabilmente questa situazione si verificherà tra poco.

La soluzione al problema è abbastanza semplice: è possibile creare un dischetto separato contenente AmigaBASIC ed i programmi sviluppati. Per fare questo, è necessario avere a disposizione un dischetto vuoto, oppure un dischetto il cui contenuto possa essere cancellato senza che si verifichi la perdita di qualcosa di importante.

Non è possibile creare un dischetto direttamente da AmigaBasic; è quindi necessario uscire dal BASIC e ritornare al Workbench. Se non vi è alcun programma in memoria, è sufficiente selezionare l'opzione Quit dal menù a tendina Project oppure inserire il comando SYSTEM nella finestra BASIC.

Se vi è un programma in memoria e si è ricevuto il messaggio "Disk Full" mentre si stava salvando il programma, abbandonare il BASIC significa inevitabilmente perdere tale programma. E' quindi necessario mantenere in memoria tale programma.

Grazie alla possibilità di multitasking offerta da Amiga, è possibile aggirare questo problema.

- Effettuare un click sul gadget di chiusura della finestra LIST (per chiuderla) ed utilizzare il gadget di dimensionamento per rendere la finestra BASIC il più piccolo possibile.

- Spostare questa finestra in una posizione che consenta di vedere completamente la finestra del Workbench.

Mentre si sta utilizzando il Workbench, AmigaBASIC continuerà la sua esecuzione in background.

Per attivare il Workbench, effettuare un click in un punto qualsiasi al di fuori della finestra BASIC. Nella riga titolo verrà visualizzato il messaggio: "Workbench release 1.2" (o un diverso numero di identificazione della versione di cui si è in possesso), seguito dall'informazione sulla quantità di memoria libera nel sistema.

• Rimuovere il dischetto Extras dal drive ed inserire un dischetto vuoto.

Formattazione del dischetto

Per prima cossa è necessario *formattare* il dischetto. Questa operazione è necessaria per consentire ad Amiga di poter utilizzare il dischetto. Un dischetto è essenzialmente un pezzo di plastica magnetizzata contenuto in un involucro di protezione. Amiga utilizza dischetti da 3.5". Questo significa che il supporto di plastica magnetizzata del floppy ha un diametro di circa tre pollici e mezzo (7-8 cm.). Quasi tutti i computer più recenti utilizzano questo formato. Fino a pochi anni fa i personal computer utilizzavano dischetti da 5.25", i quali erano però più facilmente danneggiabili, più scomodi da maneggiare, e potevano contenere una minore quantità di dati rispetto a quelli da 3.5".

Il pezzo di plastica contenuto nell'involucro protettivo è magnetizzato. Una testina magnetica contenuta nel drive, *la testina di lettura e scrittura (read/write Head)*, si sposta lungo il dischetto mentre questo gira ad alta velocità nel drive. La testina di lettura/scrittura effettua la scrittura o la lettura dei bit dal dischetto. Una magnetizzazione positiva corrisponde ad un bit acceso; una magnetizzazione negativa ad un bit spento. Gruppi contigui di bit formano byte, i quali, raggruppati a loro volta, formano file (contenenti dati o programmi).

I bit vengono scritti su *tracce* concentriche (concentriche significa che le tracce circolari sono posizionate una di fianco all'altra). Si potrebbe pensare ad un dischetto per computer come ad un qualcosa di intermedio tra un disco LP ed un nastro magnetico per registratore.

Le tracce devono essere create sul dischetto prima che questo venga utilizzato, di modo che la testina di lettura/scrittura possa lavorare correttamente. Queste tracce devono essere create secondo un preciso formato. Per questo motivo è necessario *formattare* , cioè inizializzare, il dischetto, creando su di esso il formato specifico, prima che esso possa essere utilizzato.

Formati differenti

Ci si potrebbe a questo punto domandare come mai le tracce non vengano create direttamente. Accade questo in quanto i dischetti da 3.5" vengono

utilizzati da computer differenti che utilizzano formati diversi tra loro. Anche se i dischetti hanno le medesime dimensioni e sono realizzati con lo stesso materiale, le informazioni sul dischetto sono organizzate in modo diverso. Il numero di tracce, la distanza tra esse, la codifica delle informazioni variano notevolmente da un sistema all'altro. Un Atari St, ad esempio, non è in grado di utilizzare un dischetto Amiga, e viceversa.

AmigaDOS

Amiga è in grado di utilizzare diversi tipi di formati per i dischetti. Normalmente l'accesso ai dischi è realizzato utilizzando l'AmigaDOS (DOS è una abbreviazione di *Disk Operating System*). AmigaDOS è un programma responsabile dell'input, dell'output e della completa gestione dei dischetti. Se Amiga funziona utilizzando un altro sistema operativo è in grado di leggere i dischetti in un formato differente. Amiga 1000, ad esempio, è in grado di leggere dischetti dei PC IBM utilizzando un adattatore MS-DOS, esattamente come può essere fatto con un Amiga 2000 adeguatamente equipaggiato. Per lavorare in AmigaBASIC, non è comunque il caso di preoccuparsi degli altri formati dei dischetti.

Informazioni tecniche

Ecco una serie di informazioni tecniche per chiunque fosse interessato. Il formato utilizzato dall'AmigaDOS per i dischetti utilizza 80 tracce per ogni lato del dischetto (160 tracce in totale) consentendo di memorizzare 880K per ogni dischetto. Dato che vengono utilizzati, per la lettura e la scrittura, entrambi i lati del dischetto, il disk drive di Amiga ha due testine (per questo motivo, quando si acquistano dischetti nuovi, è necessario assicurarsi che si tratti di *dischetti a doppia faccia*, cioè dischetti per i quali sia stato eseguito il test di controllo degli eventuali difetti su entrambe le facce).

Quando viene inserito nel drive un dischetto non formattato, viene visualizzata, sullo schermo del Workbench, l'icona di un dischetto con il nome DF0:BAD. AmigaDOS non è in grado di leggere il dischetto e quindi lo classifica come BAD (cattivo). Un dischetto identificato in questo modo potrebbe essere un disco non formattato, oppure non in formato AmigaDOS e quindi non utilizzabile da AmigaDOS.

NOTA:
E' necessario assicurarsi che il dischetto che ci si accinge a formattare non contenga programmi o dati importanti. L'operazione di inizializzazione, infatti, distrugge le vecchie informazioni eventualmente presenti. E' consigliabile etichettare il dischetto con un nome in modo da poterlo riconoscere e da poterne identificare il contenuto. E' almeno opportuno scrivere il nome del dischetto sulle etichette di carta autoadesive fornite insieme ai dischetti.

- Se si è assolutamente sicuri che si desidera formattare il dischetto, attivare l'icona DF0:BAD, effettuando un click su di essa in modo da renderla nera.

- Selezionare l'opzione Initialize dal menù a tendina Disk.

I termini inizializzare e formattare hanno lo stesso significato, identificando la procedura di creazione delle tracce su un dischetto.

Amiga, in risposta alla selezione effettuata, visualizza una finestra contenente un messaggio per richiedere l'inserimento del dischetto del Workbench. Nel caso si possiedano due drive, è possibile inserire il Workbench nel secondo drive. Se si possiede un solo drive, estrarre il dischetto vuoto ed inserire il dischetto del Workbench.

Dopo pochi attimi Amiga chiede di inserire il dischetto da formattare nel drive.

Viene poi visualizzato un requester per chiedere conferma per la continuazione della procedura selezionata. Nel requester è visualizzato il messaggio "OK to initialize disk in drive DF0 (all data will be erased)?". Nel caso si tratti di un dischetto già precedentemente formattato, al quale era quindi già stato attribuito un nome, il messaggio presentato sarà: "OK to initialize disk .... (nome del dischetto)"

- E' necessario confermare, effettuando un click sul gadget relativo, l'intenzione di procedere nell'operazione.

AmigaDOS identifica il drive interno come DF0, il primo drive esterno, se presente, come DF1, etc.

Il processo di formattazione inizia non appena viene effettuato un click sul gadget Continue; una volta avviato tale processo non può essere interrotto.

• Quando si è pronti a procedere effettuare un click sul gadget Continue.

Le 80 tracce vengono create sul dischetto e viene effettuato il controllo della correttezza della procedura. Sullo schermo compare una finestra di nome Initialize in cui viene visualizzato il numero della traccia in fase di formattazione ed il numero delle tracce ancora da formattare. La prima traccia è la numero 0, l'ultima è la numero 79. Compare il messaggio "Formatting" mentre le tracce sono in formattazione, il messaggio "Verifying" mentre Amiga sta effettuando il controllo.

Errori in fase di formattazione
Nel caso Amiga rilevi un errore mentre sta controllando la correttezza della formattazione, viene visualizzato un messaggio di errore. Le cause alla base di esso possono essere molte. Un primo rimedio può essere quello di ritentare ancora la procedura. Se non si riesce nuovamente a concluderla correttamente, è possibile che il dischetto presenti un difetto di fabbricazione (una qualche imperfezione della plastica magnetizzata). E' allora opportuno provare a formattare un altro dischetto; se il problema si ripresenta nuovamente, è consigliabile contattare il proprio venditore di fiducia, oppure un centro di assistenza Commodore.

La procedura non è terminata quando tutte le 80 tracce sono state create sul dischetto: Amiga infatti segnala con un apposito messaggio nella finestra Initialize, di attendere ancora un attimo:

```
Warning: Initialize Still in Progress.
DO NOT REMOVE DISK.
```

Attendere!!!
La luce del drive si spegne per un paio di secondi, ma non è ancora giunto il momento di rimuovere il dischetto. Su di esso devono infatti essere memorizzate ancora alcune importanti informazioni di controllo. Solamente dopo aver effettuato anche questo è terminata la procedura di inizializzazione.

Al dischetto formattato viene automaticamente attribuito il nome Empty.

• Effettuare un doppio click sull'icona del dichetto Empty.

Sullo schermo viene visualizzata una finestra contenente solamente l'icona del Trashcan. *L'indicatore del disco* sul lato sinistro della finestra segnala che il dischetto è completamente vuoto.

Rename
Il nome Empty non è sicuramente un buon nome per identificare il contenuto di un dischetto. Dato che il dischetto dovrà contenere programmi BASIC, appare più opportuno assegnare un nome che evochi in qualche modo questa funzione.

• Attivare l'icona del dischetto Empty (se non è gia attiva), effettuando un click su di essa.

• Selezionare l'opzione Rename dal menù a tendina Workbench.

In seguito a questa azione si ottiene la visualizzazione di una finestra di input al centro dello schermo. Per cancellare il nome Empty è sufficiente premere il tasto <DEL> fino a quando tale finestra (composta da una sola linea) non è completamente vuota. A questo punto è possibile inserire il nuovo nome che si intende dare al dischetto. Un buon nome potrebbe essere DiscoAmigaBasic.

• Inserire il nome scelto e premere il tasto <Return>.

Il disco Empty viene rinominato con il nome specificato; come si può notare, nello stesso istante in cui viene cambiato il nome del dischetto, cambia anche il titolo della finestra corrispondente.

• Non chiudere la finestra DiscoAmigaBasic

Copiare programmi
Si procederà ora alla copia di alcuni programmi sul nuovo dischetto. Per prima cosa è necessario copiare AmigaBasic.

- Inserire il dischetto Extras nel drive.

- Effettuare le copie con due drive invece che con uno solo è molto più semplice. Se si possiede un solo dischetto è necessario togliere ed inserire diverse volte i dischetti dal drive durante la procedura di copia.

Attenzione
Ricordarsi sempre di non estrarre il dischetto fino a quando non si è spenta la luce del drive.

Per essere più sicuri è conveniente attendere fino a quando il drive non smette di funzionare (cioè fino a quando non termina il rumore). Nel caso non si segua questa raccomandazione, è possibile perdere tutti i dati contenuti nel dischetto.

- Effettuare un click sull'icona del dischetto Extras.

Viene aperta e visualizzata sullo schermo del Workbench la finestra corrispondente.

- Spostare il puntatore del mouse sopra l'icona AmigaBASIC e, tenendo schiacciato il pulsante sinistro del mouse, spostare tale icona nella finestra DiscoAmigaBasic.

Per copiare oggetti da una finestra all'altra, è sufficiente spostare l'icona dalla finestra di origine a quella di destinazione, nel caso in esame dalla finestra Extras alla finestra DiscoAmigaBasic.

Nel caso si abbiano a disposizione due drive, la procedura di copia inizia immediatamente ed il puntatore si trasforma nel puntatore di attesa (la nuvoletta contenente la stringa Zzz).

Effettuare una copia avendo a disposizione un solo drive
Nel caso si possieda un solo disk drive, è necessario cambiare diverse volte il dischetto in esso contenuto, estraendo ed inserendo, rispettivamente, il dischetto Extras e DiscoAmigaBasic. Il Workbench informa quando è necessario effettuare tale operazione visualizzando un opportuo requester.

Quando la copia è terminata, nella finestra DiscoAmigaBasic compare l'i-

cona dell'AmigaBASIC.

Copia di cassetti
E' ora necessario copiare sul dischetto i programmi BASIC precedentemente realizzati e salvati. E' possibile copiare un intero cassetto contenente più programmi nello stesso modo in cui viene copiato un programma singolo. Nel caso di un solo drive il numero di cambi di dischetto da effettuare dipende dalla quntità di memoria disponibile, dal numero di file da copiare e dalla loro dimensione.

- Copiare il cassetto Programmi dal dischetto Extras al dischetto DiscoAmigaBasic.

I programmi vengono copiati da un dischetto all'altro, e non solamente mossi. Vi è ora, infatti, una copia dei programmi su entrambi i due dischetti, a differenza di quanto si verifica quando i programmi vengono spostati all'interno dello stesso dischetto. Quando si sposta un programma BASIC da un cassetto ad un altro, all'interno dello stesso dischetto, i programmi non vengono copiati, ma realmente spostati.

Dopo aver copiato sul dischetto i programmi, è necessario provvedere a riordinare il dischetto.

- Chiudere la finestra Extras ed aprire la finestra BASIC, mantenendo sempre aperta la finestra DiscoAmigaBasic.

Naturalmente, nel caso si possieda un solo drive, viene richiesto di immettere nel drive il dischetto del Workbench. Non verrà spiegata nouvamente la procedura in quanto si dovrebbe ora essere in grado di comprendere cosa è necessario fare in risposta alle richieste di Amiga.

Copiare all'interno di un cassetto
Durante una delle precedenti note d'uso, era stato realizzato un cassetto per contenere i programmi realizzati. Probabilmente non ci si ricorda più come era stata effettuata questa operazione. Nella finestra Workbench si può trovare un cassetto di nome Empty. Ogni volta che è necessario un nuovo cassetto vuoto, è sufficiente realizzare una copia di tale cassetto.

- Spostare l'icona del cassetto dalla finestra del Workbench nella nuova finestra, tenendo premuto il pulsante sinistro del mouse e rilasciare poi il pulsante stesso.

Sono già state effettuate un numero sufficiente di operazioni di copia dal Workbench, per cui non è il caso di soffermarsi ulteriormente sulla descrizione della procedura da effettuare.

- Effettuare un click sul gadget di chiusura della finestra Workbench.

Ordinamento del dischetto e dei cassetti in esso contenuti
E' necessario ora procedere ad ordinare i cassetti nel dischetto DiscoAmigaBasic. Dato che, da qui alla fine del libro, verrano scritti numerosi programmi, è opportuno cominciare a creare i cassetti che li contengano e che consentano di recuperarli facilmente.

In base al tipo di programmi che verranno realizzati, è possibile distinguere le seguenti categorie:

- Video
- Disegno
- Grafica
- Archivi
- Suono
- Voce
- Vari

Sono quindi necessari in tutto sette cassetti.

Duplicate
Per creare i cassetti è necessario eseguire la procedura seguente:

- Effettuare un click sull'icona del cassetto Empty e selezionare l'opzione Duplicate del menu a tendina Workbench.

Si ottiene in questo modo una copia del cassetto Empty.

Per evitare in seguito confusioni, è opportuno procedere immediatamente a rinominare il cassetto appena creato.

- Effettuare un click sull'icona del nuovo cassetto e selezionare l'opzione Rename del menu a tendina Workbench.

E' possibile utilizzare un piccolo trucchetto per cancellare il nome che compare nella finesta visualizzata.

Invece di cancellare carattere per carattere, utilizzando il tasto <Del>, è possibile effettuare la cancellazione in un colpo solo utilizzando la combinazione di tasti <Amiga destro> <x>.

- Dopo aver rinominato il cassetto, spostare la sua icona in una posizione tale da consentire di avere spazio sufficiente per crearne altri.

Si deve ora procedere nello stesso modo per creare altri cinque cassetti, cercando di posizionarli nella finestra in posizioni tali da risultare abbastanza ordinati.

Snapshot
Come già accennato nella nota d'uso 1, e come si può facilmente constatare dopo un po' di esperienza con il Workbench, Amiga non è in grado di ricordare automaticamente il posizionamento dato ai cassetti all'interno di una finestra.

- Tenendo premuto il tasto <Shift>, effettuare un click su tutte le icone contenute nella finestra DiscoAmigaBasic.

- Selezionare l'opzione Snapshot dal menù a tendina Special.

In questo modo si è indicato ad Amiga come disporre le varie icone all'interno della finestra.

Nella figura seguente è mostrata la disposizione delle icone nel dischetto allegato al libro.

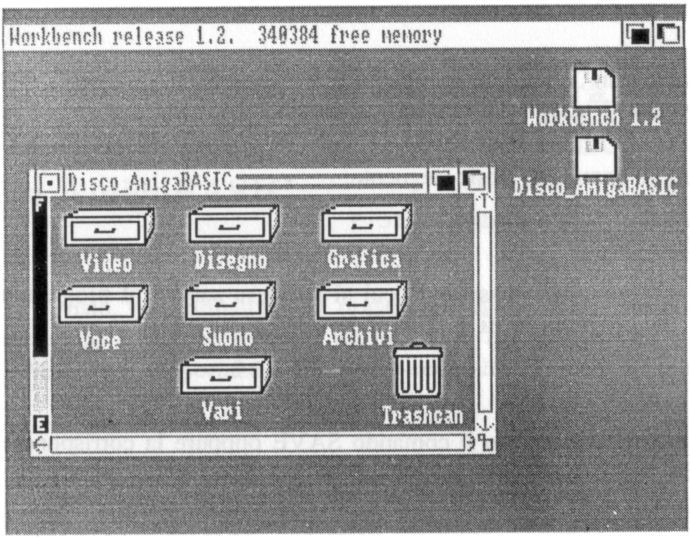


Figura 12: La finestra DiscoAmigaBasic come appare dopo l'ordinamento

Riordinamento dei cassetti.
Ora che tutti i cassetti sono stati creati correttamente, rinominati e posizionati, è necessario spostare i programmi contenuti nel vecchio cassetto all'interno di quelli appena creati.

• Aprire il cassetto contenente questi programmi.

Probabilmente il cassetto è abbastanza disordinato. Quando effettua i salvataggi, Amiga non è molto ordinato, per cui potrebbe capitare di avere icone parzialmente sovrapposte, ma non è il caso di preoccuparsi, dato che il cassetto verrà ora svuotato, spostando ad uno ad uno i programmi all'interno dei cassetti specifici, organizzandoli in base alle categorie precedentemente evidenziate. Nel cassetto Grafica potrebbe, ad esempio, essere spostato il programma "Sfere volanti". Il Workbench si preoccupa di realizzare la copia dei programmi da un cassetto all'altro del dischetto.

Nel caso vi sia in memoria un programma BASIC, ad esempio quello realizzato nel capitolo precedente, non ancora salvato, è possibile procedere

ora al salvataggio. Per effettuare questa operazione è necessario utilizzare un comando BASIC.

- Effettuare un click all'interno della finestra BASIC ed allargarla, utilizzando il gadget di dimensionamento.

- Inserire il comando Basic seguente:

```
chdir "DiscoAmigaBasic:¶
```

seguito dal nome del cassetto in cui si desidera salvare il programma. Nel caso di un programma di grafica il comando dovrebbe essere:

```
chdir "DiscoAmigaBasic:Grafica"¶
```

E' poi possibile utilizzare il comando SAVE opppure la corrispondente opzione del menù a tendina per salvare il programma.

- Chiudere la finestra BASIC effettuando un click sul gadget di chiusura.

Un posto per ogni cosa

Posizionare ora nei cassetti desiderati tutti i programmi realizzati. Spostare il programma di videotitolazione nel cassetto Video, il programma di disegno nel cassetto Programmi di disegno, il programma per la realizzazione di grafici a torta ed istogrammi nel cassetto Grafica, etc.

Dopo aver eseguito queste operazioni, è necessario selezionare, per ogni singolo cassetto, l'opzione Snapshot, in modo da istruire Amiga su come disporre le icone all'interno delle varie finestre.

- Chiudere tutte le finestre aperte, con la sola eccezione di quella DiscoAmigaBasic, ed effettuare un click sull'icona AmigaBASIC.

Nel prossimo paragrafo verranno mostrati alcuni dei comandi BASIC disponibili per la gestione del dischetto.

3.2 Directory, alberi ed altro ancora: comandi per la gestione dei dischi in AmigaBASIC

Come si è visto, per formattare ed organizzare il dischetto, è stato necessario uscire dal BASIC e ritornare ad interagire col Workbench, dato che AmigaBASIC non supporta tali funzioni.

SAVE e LOAD

AmigaBASIC, tuttavia, fornisce alcuni utili comandi per la gestione del dischetto. Due esempi sono già stati visti: per salvare programmi è possibile utilizzare il comando SAVE, mentre per richiamarli si ha a disposizione il comando LOAD. Per effettuare queste operazioni è anche possibile ricorrere all'utilizzo dei menu a tendina. Le opzoni in essi contenute sono a tutti gli effetti comandi BASIC, non ha alcuna importanza se all'opzione Open del menù Project corrisponde il comando LOAD, cioè se la corrispondenza dei nomi non è totale: quello che si ottiene, utilizzando una o l'altra modalità, è il medesimo risultato.

Il primo comando che si analizzerà assume una notevole importanza se si desidera visualizzare la lista dei file e dei programmi contenuti in un dischetto. Nel caso non si ricordi il nome di un programma, è possibile non dover ricorrere al Workbench utilizzando un comando BASIC.

• Inserire nella finestra BASIC il comando seguente:

```
Files
```

FILES

Il comando FILES visualizza il nome di tutti i file contenuti nella *directory corrente.* Prima di procedere è opportuno chiarire il significato del termine directory corrente. Un file è costituito da qualsiasi cosa memoriz-

zabile su dischetto: un programma, dei dati, un'immagine, etc. Una *directory* è invece l'indice dei contenuti di un dischetto.

Osservando l'output visualizzato dal comando precedente si nota che la prima linea riporta il seguente messaggio:

```
Directory of DiscoAmigaBasic¶
```

AmigaBASIC fornisce informazioni sulla directory che viene visualizzata. Se si osserva dal Workbench il contenuto di una finestra, si potrebbe pensare che in essa siano contenuti solamente i file associati alle icone mostrate.

File .Info

Se, tuttavia, si osserva attentamente quanto visualizzato dal comando FILES, e lo si confronta con gli oggetti contenuti nella finestra DiscoAmigaBasic, si nota come nel primo caso compaiano alcuni nomi non presenti nella finestra. Un'attenta analisi consente di notare che la maggior parte dei nomi sono accoppiati tra loro, nel senso che oltre al file AmigaBASIC, è presente un file con identico nome ed un'estensione .Info (AmigaBASIC.Info). Per comprendere che tipo di file siano quelli con *estensione .Info*, è necessario analizzare un attimo il funzionamento del Workbench.

Il Workbench assegna ad ogni file su dischetto un'icona; le icone non sono tutte uguali, in quanto generalmente ogni programma possiede un proprio tipo di icona; è possibile, inoltre, costruire delle icone personalizzate utilizzando il programma IconEd. I file con estensione .Info sono appunto quelli relativi all'icona associata al file.

Nell'output del comando FILES compaiono anche alcuni nomi racchiusi tra parentesi quadre. Si tratta dei nomi delle *sottodirectory* (cioè dei *cassetti*) contenute nella directory corrente. In una finestra esse vengono visualizzate con icone a forma di cassetto. Per ogni cassetto si ha un file .Info relativo all'icona associata. E' possibile memorizzare ogni tipo di file in un cassetto, ad esempio anche file di icone personalizzate.

Sottodirectory
Il concetto di sottodirectory è abbastanza semplice da comprendere. Programmi e file sono memorizzati all'interno di sottodirectory, suddivisi in base alla propria categoria di appartenenza, analogamente a come era stata precedentemente realizzato per effettuare la riorganizzazione del DiscoAmigaBasic. E' possibile nidificare cassetti in cassetti, in cassetti, etc, fino a riempire, al limite, tutta la memoria.

E' possibile chiarire meglio il concetto precedente utilizzando una struttura astratta ad albero. La struttura ad albero della figura seguente mostra le sottodirectory esistenti, la loro organizzazione e come sia possibile muoversi al loro interno.

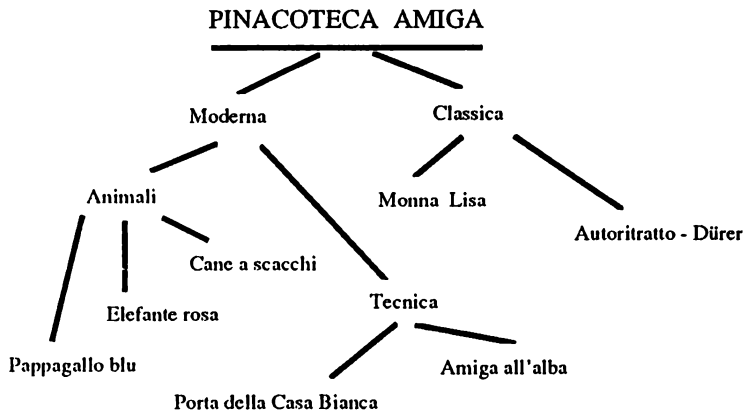


Figura 13: Struttura ad albero delle directory su dischetto.

Si supponga di essere in possesso di un dischetto di nome Galleria Amiga, utilizzato per memorizzare disegni e grafica. Il nome del disco è posto alla radice dell'albero. Nel dischetto sono contenute due directory: Classici e Moderni (l'albero ha due grandi rami). Se si desidera caricare il disegno Elefante Rosa, è necessario specificare il cammino completo per arrivare a tale disegno, cioè passando attraverso Moderni ad Animali.

In BASIC è possibile caricare programmi e dati esclusivamente dalla directory corrente. Se il programma si trova in un'altra directory del dischetto, AmigaBASIC non è in grado di trovarlo. I programmi possono essere identificati in modo univoco dal cammino necessario per arrivare a loro, cioè dai rami della struttura ad albero su cui è necessario arrampicarsi per portarsi fino al programma specifico. Questo significa che è possibile avere sullo stesso dischetto più programmi con lo stesso nome, a patto che essi siano contenuti in directory differenti; il cammino per raggiungere un file, che costituisce il nome completo del file, è quindi univoco.

CHDIR

Per spostarsi lungo la struttura ad albero, è possibile utilizzare il comando CHDIR (CHange DIRectory - Cambia directory) visto nel pargrafo precedente.

Nel caso si desideri conoscere i nomi dei programmi contenuti nella directory Grafica del dischetto DiscoAmigaBasic, è necessario utilizzare il comando:

```
CHDIR "Grafica"
```

In questo modo si effettua uno spostamento dalla directory principale del dischetto alla directory Grafica. Utilizzando ora il comando FILES per ottenere la lista del contenuto di tale directory, viene visualizzata, nella prima linea di output, la seguente scritta:

```
Directory of: [grafica]
```

Di seguito è visualizzata la lista dei nomi di tutti i file contenuti in tale directory. Se tale lista contiene il nome di una sottodirectory, è possibile utilizzare nuovamente il comando CHDIR per arrampicarsi sull'albero.

Ritorno alla radice

Se il programma che si sta cercando non si trova sul ramo per il quale ci si è incamminati, è possibile utilizzare una delle due soluzioni seguenti: ripercorrere al contrario il cammino precedentemente intrapreso, oppure effettuare un salto unico all'indietro.

Vi sono quindi due differenti comandi che possono essere utilizzati.

• Inserire la linea:

```
CHDIR "/"
```

CHDIR"/"
Questo comando consente di spostarsi all'indietro di una directory all'interno dell'albero. Nel caso della figura precedente, ad esempio, si potrebbe passare dalla directory Animali a Moderni. Inserendo nuovamente il comando ci si sposterebbe nella radice.

CHDIR":"
Se non si desidera ripercorrere completamente a ritroso il cammino, è possibile utilizzare il comando:

```
CHDIR ":"
```

che consente di ritornare direttamente alla radice. Da qui sarà poi possibile intraprendere un altro cammino sull'albero, per muoversi alla ricerca del programma.

Utilizzo di CHDIR con più drive
E' necessario analizzare ora come ci si deve comportare nel caso si sia in possesso di più di un drive. Fino ad ora sono stati effettuati movimenti attraverso le directory utilizzando un solo dischetto (DiscoAmigaBasic); si vedrà ora cosa è necessario fare se si desidera esaminare il contenuto del dischetto inserito nel drive esterno (DF1).

• Inserire la seguente linea:

```
CHDIR "df1:"
```

AmigaBASIC fa riferimento al drive interno come a DF0:; il primo drive esterno è invece DF1:. La lettera D viene utilizzata come abbreviazione di Drive, la lettera F sta per Floppy, mentre il numero specifica il numero progressivo del drive. Un secondo drive esterno assume il nome DF2:. Un eventuale disco rigido verrà riferito come DH0: (Drive HardDisk 0).

Scelta del drive
E' possibile precedere il nome di una directory con un identificatore di drive. Il comando:

```
CHDIR "DF1:Testo"
```

specifica come directory corrente la directory Testo del dischetto contenuto nel drive 1. Ovviamente questo comando ha senso solamente se il drive 1 è effettivamente connesso ad Amiga. Se AmigaBASIC non è in grado di trovare il drive, richiederà l'inserimento di un dischetto di nome DF1 (Please insert volume Df1 in any drive).

E' anche possibile utilizzare il nome del dischetto invece dell'identificatore del drive:

```
CHDIR "DiscoAmigaBasic:Grafica"
```

Questa linea consente di definire come corrente la directory Grafica, sempre che il disco DiscoAmigaBasic sia inserito nel drive. In caso contrario viene presentato un messaggio per richiedere l'inserimento del disco nel drive.

Muoversi velocemente lungo le sottodirectory
Vi è un metodo veloce per raggiungere la directory contenente il file Elefante Rosa. La linea seguente:

```
CHDIR "DF1:Biblioteca_Amiga/Moderni/Animali
```

consente infatti di posizionarsi immediatamente nella directory voluta. E' quindi possibile inserire i nomi delle varie directory poste sul cammino per giungere al file, separati da uno "slash" (il carattere /). Nel caso venga specificato un cammino inesistente, AmigaBASIC visualizza il messaggio di errore "File not Found". Un po' di pratica con questi comandi consente di muoversi in modo abbastanza rapido attraverso le directory del dischetto.

AmigaBASIC però, oltre ad offrire la possibilità di cambiare la directory corrente, di visualizzarne il contenuto, di caricare e salvare file, consente anche di rinominare e cancellare programmi BASIC.

Ecco un piccolo esempio di come sia necessario procedere per realizzare questo.

- Inserire un paio di linee di programma nella finestra List (non ha alcuna importanza il testo di queste linee, visto che verranno immediatamente cancellate.

- Salvare il programma selezionando l'opzione Save as del menù a tendina Project.

- Inserire come nome per il file Testo, e premere il tasto <Return>.

Amiga salverà il file su dischetto, come è possibile verificare utilizzando il comando FILES. Nella lista visualizzata dal comando compaiono infatti i nomi Testo e Testo.Info.

E' possibile ora cambiare il nome del programma, per assegnargli un nome che identifichi meglio cosa effettivamente esso realizza.

NAME

Per modificare il nome di un programma, AmigaBASIC mette a disposizione il comando NAME.

- Inserire nella finestra BASIC la linea seguente:

```
NAME "testo" AS "Fittizio"
```

Il drive si mette in funzione per un istante e registra sul dischetto la modifica apportata. Ovviamente è a questo punto necessario modificare anche il nome del file Testo.Info, in modo da associare correttamente l'icona al programma. Teoricamente AmigaBASIC dovrebbe provvedere automaticamente alla modifica; nel caso in cui questo non si verifichi, cioè nel caso si sia in possesso di una versione non recente di AmigaBASIC, è necessario inserire la seguente linea:

```
NAME "Testo.Info" AS "Fittizio.Info"
```

A questo punto tutto dovrebbe essere a posto (nel caso non si sia in possesso dell'ultima versione di AmigaBASIC, è opportuno procurarsela al più

presto richiedendola al proprio venditore di fiducia).

KILL

Si analizzerà ora un comando molto utile per sbarazzarsi di file non desiderati (nel caso specifico si provvederà a cancellare il file Fittizio appena creato). Per rimuovere programmi dal Workbench è necessario buttarli nel cestino e selezionare successivamente l'opzione Empty Trash. AmigaBASIC mette invece a disposizione un comando che consente di cancellare direttamente un file, il comando KILL. Una procedura di questo tipo, pur rivelandosi più veloce e vantaggiosa, può talvolta essere pericolosa. Una volta utilizzato il comando KILL per cancellare un file, infatti, non è più possibile recuperarlo; è quindi necessario essere assolutamente sicuri di voler effettuare la cancellazione.

• Inserire la linea seguente:

```
KILL "Fittizio"
```

AmigaBASIC realizza immediatamente la cancellazione del file.

E' possibile cancellare solamente il file .Info associato ad un programma; per realizzare questo, è sufficiente utilizzare il comando nella forma seguente:

```
KILL "Fittizio.Info"
```

In questo modo il programma Fittizio, essendo stato privato della sua icona, non sarà più visibile da Workbench, ma risulterà ugualmente accessibile da AmigaBASIC. La sua esistenza può infatti essere rilevata utilizzando il comando FILES (ecco un modo per proteggere i propri programmi dagli sguardi indiscreti dei soliti curiosi.)

L'analisi dei comandi per la gestione del dischetto, almeno per quanto riguarda i file, è terminata; resta ancora da vedere come accedere direttamente al contenuto dei file per poter memorizzare e richiamare dati, gestire archivi di informazioni, etc.

3.3 Collezioni di dati: gestione di una agenda in BASIC

Fino a questo punto si è semplicemente provveduto a salvare programmi AmigaBASIC su dischetto. Sono già stati mostrati alcuni esempi di salvataggio di file non contenenti programmi (ad esempio il file realizzato con l'editor di oggetti), ma tali esempi non sono stati discussi, rimandandone la spiegazione.

Memorizzazione di dati
Oltre ai programmi, AmigaBASIC consente di salvare su dischetto anche file contenenti dati, cioè numeri, testo, immagini, etc. Anche la maggior parte dei programmi commerciali consente di effettuare il salvataggio su dischetto dei file contenenti dati, che altrimenti verrebbero perduti irrimediabilmente allo spegnimento del computer. Un *file di dati* può contenere informazioni di vario tipo; ad esempio, informazioni relativi al testo creato con un programma di videoscrittura, in modo tale da poter successivamente modificare tale testo.

Il programma seguente, che crea su dischetto un file contenente indirizzi, consente di dare un piccolo sguardo a come AmigaBASIC salva i dati ed a come li recupera in seguito.

```
OPEN "FileIndirizzi" FOR OUTPUT AS 1¶
¶
Immissione:¶
  PRINT¶
  INPUT "Nome";Nome$¶
  INPUT"Indirizzo";Indirizzo$¶
  INPUT"Citta";Citta$¶
  INPUT"Telefono";Telefono$¶
  PRINT#1,Nome$¶
  PRINT#1,Indirizzo$¶
```

```
    PRINT#1,Citta$¶
    PRINT#1,Telefono$¶
    x=x+1¶
    PRINT "Record"x" ("Nome$") memorizzato."¶
    PRINT "Altri Record?"¶
    INPUT "S/N:";Ans$¶
    IF UCASE$(Ans$)="S" THEN Immissione¶
¶
CLOSE¶
PRINT "File chiuso; Programma terminato."¶
```

Tutti i programmi per la gestione dei dati dovrebbero essere memorizzati nel cassetto Dati.

- Eseguire il seguente comando:

```
CHDIR "DiscoAmigaBasic:Dati"
```

- Salvare il programma con il nome Gestione Agenda (all'interno del nome di un programma possono essere compresi eventualmente anche degli spazi bianchi).

Modalità di funzionamento del programma
Il comando OPEN realizza l'apertura del file FileIndirizzi. Un file deve essere aperto prima che un programma BASIC possa utilizzarlo, esattamente come un libro deve essere aperto per poter essere letto.

OPEN
Il comando OPEN può anche essere utilizzato per dichiarare se si intende usare il file in scrittura (OPEN ... FOR OUTPUT) o in lettura (OPEN ... FOR INPUT). Un file può essere aperto tanto in lettura quanto in scrittura, ma non contemporaneamente in entrambe le modalità. E' tuttavia possibile aprire più di un file per volta, alcuni utilizzati per l'input, altri per l'output.

Per questo motivo ad ogni file aperto viene associato un numero (nell'esempio riportato sopra il numero 1), in modo tale che i comandi di lettura e scrittura siano in grado di identificare su quale file operare.

Quando un file viene aperto per la prima volta (come nell'esempio mostrato), il comando OPEN crea il file sul dischetto nella directory corrente. La sintassi del comando è la seguente:

```
OPEN "Nome_File" FOR (Modalità) AS (Numerofile)
```

Questo comando dispone anche di una seconda forma sintattica, contenente le stesse informazioni disposte in ordine diverso:

```
OPEN "(Modalità)",#(Numerofile),"Nome_File"
```

Nel programma esplicativo presentato, la prima linea:

```
OPEN "O",#1,"File Agenda"
```

utilizza appunto la seconda forma sintattica.

E' possibile utilizzare indifferentemente una delle due forme; a seconda delle varie situazioni, una può rivelarsi più comoda e comprensibile dell'altra. Si ritornerà comunque su questo punto più avanti.

La prima linea crea un file di nome File Agenda, con numero identificativo 1, all'interno del quale è possibile inserire dati. Si procederà ora a riempire questo file.

PRINT#

La subroutine Immissione$: richiede all'utente i valori da assegnare alle variabili Nome$, Indirizzo$, Citta$ e Telefono$. Le stringhe fornite in risposta contengono le informazioni che devono essere incluse nell'agenda elettronica. Per scrivere questi dati sul file è necessario utilizzare il comando PRINT#, seguito dal numero di identificazione del file su cui si desidera scrivere.

Il comando:

```
PRINT#1,Nome$
```

scrive sul file numero 1, in questo caso File Agenda, il contenuto della variabile Nam$. Il comando PRINT# funziona esattamente come il comando PRINT; l'output viene però inviato ad un file e non allo schermo.

File sequenzali
I dati dell'agenda elettronica vengono salvati in un *file sequenziale*, cioè in un file in cui i dati sono memorizzati uno di seguito all'altro (in sequenza per l'appunto).

Nel programma in esame viene salvato il primo nome, seguito da indirizzo, città e numero telefonico; poi il secondo nome, e così via.

I file sequenziali costituiscono uno dei tipi di file gestiti da AmigaBASIC. Tra gli altri tipi gestibili vi sono, come si analizzerà in seguito, i file ad accesso casuale.

Le parti restanti di programma dovrebbero essere abbastanza semplici da comprendere. La variabile x viene incrementata in modo tale da poter visualizzare il numero progressivo degli indirizzi inseriti nell'agenda. Il programma chiede poi all'utente se desidera inserire ulteriori indirizzi; premendo il tasto <s> viene eseguita nuovamente la routine Immissione$:.

Informazioni aggiuntive riguardo al comando INPUT
La linea contenente il comando INPUT potrebbe sembrare non corretta. Si osservi attentamente la differenza tra le due linee seguenti:

```
INPUT "(Testo)"; Variabile
' viene utilizzato il punto e virgola '


INPUT "(Testo)", Variabile
' viene utilizzata la virgola '
```

Nel programma di esempio vengono utilizzate entrambe le forme; esse sono entrambe corrette.

Nel caso in cui compare il punto e virgola tra il testo e la variabile, AmigaBASIC visualizza un punto di domanda subito dopo il testo; nel caso della virgola, il punto di domanda non viene visualizzato.

CLOSE
Il comando CLOSE consente di chiudere un file precedentemente aperto

con un comando OPEN.

La gestione di un file su dischetto è realizzata nel modo seguente:

quando viene eseguito un comando per scrivere informazioni su disco, i dati non vengono scritti carattere per carattere. Se la procedura venisse realizzata in questo modo, essa richiederebbe troppo tempo e non consentirebbe ad altri programmi di accedere al dischetto per parecchio tempo. Amiga funziona in multitasking ed è quindi probabile che più di un programma richieda l'accesso al disco durante un certo periodo di tempo. Se AmigaBASIC riservasse per sè il canale di comunicazione con il dischetto durante tutto il tempo in cui un file è aperto, ogni altro programma, per poter accedere al disco, dovrebbe fermarsi ad attendere la chiusura di tutti i file aperti da AmigaBASIC.

Per superare questo problema, viene inizializzato un *buffer* (canale) per ogni file aperto. Un buffer è una zona di memoria, generalmente delle dimensioni di 128 byte, in cui vengono memorizzati caratteri. Quando il buffer è completamente pieno i caratteri vengono trasferiti e memorizzati su dischetto. Il comando CLOSE assicura che il contenuto del buffer venga salvato su dischetto anche se il buffer non è ancora completamente pieno.

Il comando CLOSE è necessario anche prima di utilizzare per la seconda volta alcuni comandi AmigaBASIC. Non è possibile, ad esempio, aprire nuovamente un file in lettura o in scrittura prima di averlo richiuso; non è neppure possibile cancellare un file mentre questo è ancora aperto. Il comando CLOSE consente infine di liberare il numero di identificazione del file, numero che può così essere riutilizzato nuovamente per un altro file.

Se si tenta di leggere o scrivere un file chiuso, si ottiene il messaggio di errore "Bad File Number".

Come interagire con il programma
Utilizzare il programma dimostrativo è piuttosto semplice. Amiga richiede le informazioni relative all'indirizzo ad una ad una. Dopo aver inserito un indirizzo, è possibile proseguire con altri premendo il tasto <s> in risposta alla richiesta specifica.

• Inserire un paio di indirizzi per provare il programma.

Non è tuttavia sufficiente, per completare la gestione di un'agenda, avere realizzato un programma che consente di memorizzare su dischetto i vari indirizzi; è infatti necessario disporre anche di un programma che consenta di leggerli e recuperarli. Prima di inserire la parte di programma riportata di seguito, si consiglia di salvare il codice precedentemente realizzato.

• Ripulire la memoria utilizzando il comando NEW.

• Inserire le linee seguenti:

```
OPEN "File Agenda" FOR INPUT AS 1¶
¶
LetturaDati:¶
   INPUT#1,Nomedue$¶
   INPUT#1,Indirizzo$¶
   INPUT#1,Citta$¶
   INPUT#1,Telefono$¶
   PRINT¶
   PRINT "Nome: ";Nomedue$¶
   PRINT "Indirizzo: ";Indirizzo$¶
   PRINT "Città: ";Citta$¶
   PRINT "Telefono: ";Telefono$¶
¶
IF EOF(1) =0 THEN LetturaDati¶
CLOSE 1¶
```

Quando si esegue il programma, gli indirizzi memorizzati sul file vengono letti ad uno ad uno e visualizzati sullo schermo.

• Provvedere a salvare il programma su dischetto.

Il comando OPEN effettua l'apertura del file File Agenda in modalità di lettura. Il comando non dovrebbe apparire incomprensibile, in quanto è praticamente identico a quello utilizzato per aprire un file in scrittura, già utilizzato, in questa stessa forma, in un programma precedente per realizzare il recupero di un oggetto grafico.

Per poter acquisire i dati, AmigaBASIC deve prima essere in grado di trovare il file. In caso contrario, viene visualizzato il messaggio di errore "File Not Found". Se si ottiene un errore di questo tipo, accertarsi che il nome inserito sia corretto e che la directory corrente sia quella in cui è contenuto il file cercato (per portarsi nella directory desiderata utilizzare il comando CHDIR).

INPUT#

E' possibile utilizzare il comando INPUT# per leggere da un file informazioni scritte precedentemente utilizzando il comando PRINT#. Naturalmente queste informazioni devono essere lette nella stessa sequenza in cui sono state scritte, per non ottenere dati senza senso.

Il comando INPUT# consente di leggere una variabile esattamente come il comando INPUT. La differenza tra i due comandi è che INPUT# legge le informazioni da un file e non dalla tastiera visualizzando poi sullo schermo i dati acquisiti.

EOF

La funzione EOF (End Of File, Fine del file) fornisce informazioni sulla presenza di ulteriori dati, da leggere nel file. Il numero del file per cui viene richiesta l'informazione, compreso tra parentesi, è riportato di seguito al nome della funzione. Nel caso vi siano ancora dati da leggere, EOF ritorna il valore 0, nel caso sia già stato letto l'ultimo carattere del file, viene ritornato il valore -1. In questo modo è possibile sapere se sono presenti ancora informazioni da leggere o se, viceversa, è possibile chiudere il file.

Nel caso si tenti di leggere un file senza che in esso vi siano più dati, si ottiene in risposta il messaggio di errore "Input Past End", che indica un tentativo di lettura oltre la fine del file.

Lettura dei file sequenziali

La lettura di un file sequenziale viene effettuata con una modalità molto simile a quella utilizzata per leggere le informazioni contenute all'interno di linee DATA: un puntatore viene spostato da una posizione alla successiva, in modo tale da acquisire l'ultimo valore non ancora letto. E'

nuovamente opportuno ricordare che le informazioni sono memorizzate in un file sequenziale una di seguito all'altra e che devono essere caricate nello stesso ordine in cui sono state salvate sul file per avere la corretta corrispondenza.

Questa è la caratteristica principale di un file sequenziale: i suoi record devono essere letti uno di seguito all'altro. Se si desidera leggere solamente il decimo record, è necessario leggere sempre e comunque anche i primi nove, per poter poi accedere al decimo.

E' impossibile saltare direttamente al record desiderato. I file sequenziali si rivelano molto utili quando si sta lavorando con dati che devono essere gestiti sempre nello stesso ordine.

FOR OUTPUT/ INPUT/ APPEND

Se si desidera incrementare il contenuto di un file sequenziale, risulta chiaramente troppo dispendioso dover acquisire tutti i dati del file e memorizzarli nuovamente per poter poi aggiungere le nuove informazioni. Vi è tuttavia un modo per evitare di dover eseguire questa procedura: è infatti possibile aprire un file in modalità FOR APPEND.

- Caricare il programma per la gestione dell'agenda elettronica e modificare la prima linea:

```
OPEN "File Agenda" FOR APPEND AS 1
```

E' ora possibile eseguire il programma per accodare un paio di indirizzi. Non è il caso di preoccuparsi del fatto che il programma faccia riferimento a questi come al numero 1, numero 2, etc. Quelli che si stanno inserendo sono l'indirizzo numero 1, numero 2, etc. da aggiungere al file. Terminata l'immissione è possibile utilizzare il programma di lettura del file sequenziale per controllare che gli indirizzi siano stati effettivamente aggiunti in fondo al file, dopo tutti quelli già presenti.

NOTA:

Utilizzando la modalità APPEND è possibile aggiungere dati in fondo ad un file sequenziale, ma non è possibile aggiungerli all'inizio od in mezzo allo stesso.

Se lo si desidera è possibile utilizzare la modalità APPEND ogni volta che si scrive il file. In questo modo si è sicuri di non distruggere alcuna delle informazioni precedenti. Se il file non esiste ancora AmigaBASIC agisce come nella modalità OUTPUT creandolo.

Ora che si hanno sufficienti informazioni sui file sequenziali, è possibile applicare queste conoscenze in un programma più complesso.

3.4 Informazioni per diagrammi a torta e a barre: gestione di dati statistici

Per generare grafici a torta e diagrammi a barre è necessario disporre di una certa quantità di dati. Nell'esempio precedentemente mostrato tali dati erano stati inseriti direttamente da tastiera; si trattava però di una soluzione momentanea. Si supponga infatti di voler visualizzare un grafico relativo all'andamento del mercato dei calcolatori; ovviamente, se tutte le volte si devono inserire nuovamente i dati da tastiera per ottenere il grafico, il lavoro diventa troppo pesante al punto da sconsigliare l'utilizzo del calcolatore. E' però possibile pensare di scrivere un programma che memorizzi tali dati, in modo da poterli facilmente riottenere ed aggiornare intervenendo solamente su quelli modificati.

Il programma per la generazione di grafici a torta e diagrammi a barre è già stato scritto e dovrebbe essere stato memorizzato nel cassetto Grafica del dischetto DiscoAmigaBASIC.

- Liberare l'area di memoria BASIC utilizzando il comando NEW.

- Caricare il programma per la generazione di diagrammi a barre e grafici a torta (utilizzare il comando CHDIR "DiscoAmigaBASIC/Grafica" per accedere al cassetto desiderato ed il comando FILES per ritrovare il nome del file).

- Posizionarsi nella directory Dati (utilizzando il comando CHDIR).

Si hanno ora a disposizione due programmi che devono essere uniti tra loro; è possibile a questo punto concatenare i due programmi, oppure ribattere completamente il testo di uno inserendolo nell'altro. Se non si desidera ribattere completamente uno dei programmi è opportuno procedere all'operazione di concatenazione tra i due. Come è stato illustrato in una precedente nota d'uso, nella quale si è discusso come concatenare

due programmi tra loro, è necessario a questo punto salvare il programma come file ASCII.

- Inserire, nella finestra BASIC, la linea:

```
SAVE "NewBarPie",A¶
```

- Ripulire ora la memoria utilizzando il comando NEW.

- Inserire la prima parte del nuovo programma:

```
Inizializzazione:¶
  DIM Numero$(58),Descrizione$(58),Valore$(58)¶
  DIM MatriceStr$(50),MatriceNum(50)¶
  FOR x=1 TO 58¶
    IF x>4 AND x<55 THEN Numero$(x)=STR$(x-4)¶
  NEXT x¶
  LineaSup=1¶
¶
  Colori=3¶
  SCREEN 4,640,200,Colori,2¶
  WINDOW 99,"Grafici",,20,4¶
  PALETTE 7,.8,.2,.1¶
¶
  WINDOW 1,"Gestione Dati Statistici",(0,12)-
      (631,111),22,-1¶
¶
  MENU 1,0,1,"Dati  "  ¶
  MENU 1,1,1,"Recupera  "¶
  MENU 1,2,1,"Registra  "¶
  MENU 1,3,1,"Cancella"¶
  MENU 1,4,1,"Fine  "¶
  MENU 2,0,1,"Grafici"¶
  MENU 2,1,1,"A Barre"¶
  MENU 2,2,1,"A Torta"¶
  MENU 3,0,0,""¶
  MENU 4,0,0,""¶
¶
```

```
    ON MENU GOSUB ControlloMenu¶
    MENU ON¶
¶
GOTO CicloPrincipale¶
```

Vettori

Il programma effettua, come prima cosa, il dimensionamento dei vettori utilizzati. Numero$ è il numero delle linee di dati, Descrizione$ è la descrizione di un valore, Valore$ è il valore necessario per le elaborazioni statistiche, ad esempio una percentuale. Viene utilizzato un vettore di stringhe per i numeri in quanto questa scelta rende la routine di input più semplice consentendo di inserire un qualsiasi tipo di dato. Il contenuto del vettore viene convertito in numero prima di richiamare la routine grafica.

I vettori MatriceStr$ e MatriceNum sono utilizzati per passare le informazioni alla subroutine.

Il vettore Numero$ ha sempre lo stesso contenuto, cioè i numeri delle linee dati. E' possibile avere fino ad un massimo di 50 valori; il vettore è però dimensionato a 58 elementi. Il motivo di questa scelta di dimensionamento non è legato solamente a questioni di sicurezza (garantendo al vettore un numero di elementi superiore a quello poi realmente utilizzato si è sicuri di non uscire dalla memoria assegnata al vettore).

L'idea di base deriva dall'impossibilità di poter mostrare sullo schermo tutte e 50 le linee di dati, ma solamente un numero molto minore. Una linea di dati contiene tre diversi elementi, nell'ordine seguente: il numero progressivo della linea, il nome ed il valore corrispondente.

Posizionamento del cursore

Sullo schermo vengono visualizzate contemporaneamente 9 linee. La linea con cui si sta lavorando (per effettuare input, modifiche, inserimenti, cancellazioni, etc.) risulta essere sempre la quinta. Le prime quattro e le ultime quattro linee sullo schermo sono mostrate solamente per orientamento, in modo da poter vedere le linee immediatamente precedenti e seguenti la linea in fase di modifica. E' possibile far scorrere in avanti e indietro le linee, ma il cursore rimane sempre posizionato sulla linea centrale dello schermo.

Dato che probabilmente si dovrà operare anche con le prime e con le ultime linee del file, vengono inseriti alcuni campi vuoti per precedere il primo elemento e seguire l'ultimo. Il primo vero dato risulta essere il quinto elemento del vettore. Si ha quindi 4+50+4=58.

STR$

Il ciclo FOR ... NEXT inizializza i valori del vettore Numero$. I primi quattro elementi risultano vuoti. Agli elementi successivi vengono assegnati in successione numeri interi da 1 a 50. Vi sono poi altri quattro elementi vuoti. Per convertire numeri in stringhe alfanumeriche, è possibile utilizzare il comando STR$, che risulta essere l'opposto del comando VAL.

VAL

La linea seguente:

```
? VAL (STR$(5))¶
```

produce come output il numero 5 (converte infatti il numero 5 prima in stringa e poi nuovamente in numero). Le conversioni dei dati da un formato all'altro rivestono una notevole importanza nella stesura di un programma, per cui questi comandi saranno analizzati dettagliatamente più avanti.

La variabile LineaSup è inizializzata a uno. La linea indicata da LineaSup-1 risulta essere l'ultima linea contenente dati. Dato che i dati non sono ancora stati immessi, la linea 1 risulta essere la prima che non contiene alcun dato.

E' possibile selezionare da uno a quattro bitplane, in dipendenza dal numero di colori che si intende utilizzare per i grafici.

E' ora necessario inizializzare lo schermo e la finestra utilizzati dal sottoprogramma di visualizzazione dei grafici. Il colore numero 7 viene ridefinito (da bianco in rosso), esattamente come già fatto precedentemente nel programma per la generazione degli istogrammi.

La finestra numero 1 (cioè la finestra BASIC), viene ridimensionata e rinominata come Gestione Dati Statistici. Il parametro per lo schermo di questo

comando WINDOW è -1. Tale valore identifica lo schermo del Workbench, mentre i valori da 1 a 4 identificano gli schermi definiti dall'utente.

Definizione dei menù a tendina

Dato che si desidera che questo programma risulti completamente guidato da menù, è necessario procedere a questo punto alla definizione dei menù a tendina. Per il menù Dati vengono definite quattro opzioni: Recupera, Registra, Cancella (per la cancellazione dei dati in memoria) e Fine. Al menù Grafici sono associate due opzioni: Grafico a Barre e Diagramma a Torta.

I menù 3 e 4 vengono disabilitati.

Viene poi attivato l'event trapping per il controllo del menù ed il programma richiama il ciclo principale.

A differenza di quanto effettuato nel programma di disegno, questo programma non è solamente guidato da eventi. Le routine richiamate dalle varie opzioni si inseriscono infatti nell'esecuzione del ciclo principale e della routine ImmettiTesto. La gestione degli eventi viene qui utilizzata per azioni supplementari a quelle svolte nel programma principale.

Ecco la subroutine che ha il compito di gestire le scelte da menù:

```
ControlloMenu:¶
  Men=MENU(0) : SceltaMenu=MENU(1)¶
  IF Men=1 THEN¶
    IF SceltaMenu=1 THEN GOSUB RecuperaDati¶
    IF SceltaMenu=2 THEN GOSUB RegistraDati¶
    IF SceltaMenu=3 THEN GOSUB CancellaDati¶
    IF SceltaMenu=4 THEN FineProgramma¶
  END IF¶
  IF Men=2 THEN¶
    IF SceltaMenu=1 THEN MatriceStr$(0)="B"¶
    IF SceltaMenu=2 THEN MatriceStr$(0)="T"¶
    MatriceNum(0)=LineaSup¶
    IF Valore$(MatriceNum(0)+4)="" THEN
          MatriceNum(0)=MatriceNum(0)-1¶
```

```
    FOR x=1 TO MatriceNum(0)¶
      MatriceStr$(x)=Descrizione$(x+4)¶
      MatriceNum(x)=VAL(Valore$(x+4))¶
      IF MatriceNum(x)=0 THEN MatriceNum(x)=.01¶
    NEXT x¶
    MENU OFF¶
    MENU 1,0,0 : MENU 2,0,0¶
    WINDOW 99 : CLS¶
    ¶
    GOSUB Grafici¶
    ¶
     WINDOW 2,"Premere un tasto!",(350,0)-
          (631,0),20,4¶
    COLOR 0,1 : CLS¶
    WHILE INKEY$=""¶
    WEND¶
    WINDOW CLOSE 2¶
    WINDOW 1¶
    MENU ON¶
    MENU 1,0,1 : MENU 2,0,1¶
  END IF¶
RETURN¶
```

Il numero relativo al menù selezionato viene memorizzato nella variabile Men. Il numero dell'opzione scelta viene invece assegnato alla variabile SceltaMenu, come nel programma di disegno. Come si può vedere, spesso si può ricorrere a vecchie soluzioni per risolvere nuovi problemi.

Se viene selezionata un'opzione del primo menù, viene richiamata direttamente la routine responsabile della gestione dell'evento verificatosi; nel caso venga invece selezionato il secondo menù, viene semplicemente scelto uno tra i due tipi possibili di grafico, inizializzando opportunamente il primo elemento del vettore MatriceStr$ (cioè MatriceStr$(0)).

Riallineamento

Il primo elemento del vettore MatriceNum deve contenere, quando viene richiamata la subroutine Grafici, il numero di dati da graficare. Tale infor-

mazione è memorizzata nella variabile LineaSup che viene modificata usando un'espressione di assegnamento. Talvolta però, LineaSup contiene un numero leggermente superiore a quello necessario. Se viene premuto il tasto <Return>, dopo avere inserito l'ultimo dato, il cursore si sposta di una linea. In questo caso nel grafico apparirà un elemento con valore zero e non identificato da alcun nome.

Per evitare che si verifichi questa situazione, abbastanza inutile, il programma controlla se vi è un valore nella colonna numerica dell'ultima linea. Nel caso non vi sia alcun valore, tale linea viene trattata come linea vuota, e MatriceNum(0) viene decrementato di uno.

Il successivo ciclo FOR .. NEXT copia negli elementi dei vettori MatriceNum e MatriceStr$ le appropriate informazioni. E' necessario decrementare il contatore di 4 in quanto i primi quattro elementi dei vettori Descrizione$ e Valore$ sono deliberatamente lasciati vuoti. Il ciclo viene eseguito fino all'esaurimento dei dati.

La funzione VAL viene utilizzata per convertire in valori numerici, da assegnare al vettore MatriceNum, il contenuto del vettore Valore$. Nel caso si abbia un elemento con valore nullo (cioè 0), esso viene incrementato a 0,01, in modo da evitare un eventuale errore di "Division By Zero".

Menù

Utilizzando il comando MENU OFF vengono disabilitati, durante la generazione del grafico, i due menù a tendina rendendoli mascherati per indicare come nessuna delle opzioni sia attualmente selezionabile.

Viene poi attivata la finestra in cui verrà visualizzato il grafico: WINDOW 99. Essa viene prima ripulita utilizzando il comando CLS, in modo da cancellare ogni eventuale grafico precedentemente visualizzato.

La visualizzazione viene effettuata dalla subrotine Grafici che viene richiamata utilizzando il comando GOSUB.

Creare un requester

Dopo aver visualizzato il grafico viene creato un requester con il messaggio "Premere un tasto!". La dimensione della finestra risulta estremamente

ridotta, in quanto il titolo risulta autoesplicativo.

Il ciclo WHILE ... WEND pone il programma in attesa di input. Quando viene premuto un tasto, il requester viene chiuso, viene attivata la finestra 1 e viene poi riattivato l'event trapping relativo al menù.

Sfortunatamente non è ancora possibile utilizzare questo programma, in quanto non è ancora stata inserita la subroutine Grafici. Inoltre il programma non è ancora utilizzabile con qualsiasi tipo di dato.

• Inserire ora una nuova parte di programma:

```
CicloPrincipale:¶
  CLS¶
  IF LineaSup>50 THEN LineaSup=50¶
  IF LineaUno>LineaSup THEN LineaUno=LineaSup : BEEP¶
  IF LineaUno<1 THEN LineaUno=1 : BEEP¶
  PRINT "Numero";TAB(10);"Descrizione";TAB(45);
      "Valore"¶
  FOR x=LineaUno TO LineaUno+8¶
    COLOR 1,0¶
   PRINT Numero$(x);TAB(10);Descrizione$(x);TAB(45);
       Valore$(x)¶
  NEXT x¶
  IF DatiDesc=0 THEN ParteIniz=10 : ParteFin=40¶
  IF DatiDesc=1 THEN ParteIniz=45 : ParteFin=55¶
  xp=ParteIniz¶
¶
  GOSUB ImmettiTesto¶
  in$=""¶
¶
GOTO CicloPrincipale¶
¶
¶
ImmettiTesto:¶
  IF xp<ParteIniz THEN xp=ParteIniz¶
  LOCATE 6,xp¶
```

```
  COLOR 0,3 : PRINT " "; : COLOR 1,0¶
  i$=INKEY$¶
  IF i$="" THEN ImmettiTesto¶
  IF i$=CHR$(2) THEN LineaUno=1 : RETURN¶
  IF i$=CHR$(5) THEN LineaUno=LineaSup : RETURN¶
  IF i$=CHR$(4) THEN CancellaLinea : RETURN¶
  IF i$=CHR$(14) THEN InserisciLinea : RETURN¶
  IF i$=CHR$(28) THEN GOSUB AccettaTesto :
       xp=ParteIniz : LineaUno=LineaUno-1: RETURN¶
  IF i$=CHR$(29) THEN GOSUB AccettaTesto :
       xp=ParteIniz : LineaUno=LineaUno+1: RETURN¶
  ¶
  PosTesto=xp-ParteIniz+1¶
  IF DatiDesc=0 THEN Testo$=Descrizione$(LineaUno+4)¶
  IF DatiDesc=1 THEN Testo$=Valore$(LineaUno+4)¶
¶
  IF i$=CHR$(30) THEN¶
    IF PosTesto<=LEN(Testo$) THEN i$=MID$(Testo$,
         PosTesto,1)¶
  END IF ¶
¶
  IF i$=CHR$(13) OR i$=CHR$(9) THEN¶
    GOSUB AccettaTesto¶
    DatiDesc=1-DatiDesc¶
    IF DatiDesc=0 THEN LineaUno=LineaUno+1¶
    xp=ParteIniz¶
    IF LineaSup<LineaUno THEN LineaSup=LineaUno¶
    RETURN¶
  END IF¶
  IF i$=CHR$(8) OR i$=CHR$(31) THEN¶
    LOCATE 6,xp¶
    IF PosTesto<=LEN(Testo$) THEN¶
      PRINT RIGHT$(Testo$,LEN(Testo$)-PosTesto+1);¶
    ELSE¶
      PRINT " ";¶
    END IF¶
```

```
      xp=xp-1 : IF xp<ParteIniz THEN xp=ParteIniz :
          BEEP : GOTO ImmettiTesto¶
      in$=LEFT$(in$,(LEN(in$)-1))¶
      GOTO ImmettiTesto¶
    END IF¶
    IF i$=CHR$(34) THEN i$=CHR$(39)¶
    IF i$ > CHR$(31) AND i$ < CHR$(127) THEN¶
      IF xp>=ParteFin THEN xp=ParteFin : BEEP :
          GOTO ImmettiTesto¶
      LOCATE 6,xp¶
      PRINT i$;¶
      in$=in$+i$¶
      xp=xp+1¶
    END IF¶
  GOTO ImmettiTesto¶
  ¶
  AccettaTesto:¶
    IF in$<>"" THEN¶
      IF DatiDesc=0 THEN Descrizione$(LineaUno+4)=in$¶
      IF DatiDesc=1 THEN Valore$(LineaUno+4)=in$¶
      in$=""¶
      AltriDati=1¶
    END IF¶
  RETURN¶
  ¶
  CancellaLinea:¶
    FOR x=LineaUno+4 TO 54¶
      Descrizione$(x)=Descrizione$(x+1)¶
      Valore$(x)=Valore$(x+1)¶
    NEXT x¶
    LineaSup=LineaSup-1¶
    IF LineaSup<1 THEN LineaSup=1¶
  RETURN¶
  ¶
  InserisciLinea:¶
    IF LineaSup>=50 THEN BEEP : RETURN¶
```

```
    FOR x=LineaSup+4 TO LineaUno+4 STEP -1¶
      Descrizione$(x+1)=Descrizione$(x)¶
      Valore$(x+1)=Valore$(x)¶
    NEXT x¶
    Descrizione$(LineaUno+4)=""¶
    Valore$(LineaUno+4)=""¶
    LineaSup=LineaSup+1¶
  RETURN¶
```

La routine è divisa in due sezioni principali: ImmettiTesto, responsabile della gestione dell'input ricevuto, e CicloPrincipale, utilizzata per realizzare lo scorrimento delle linee sullo schermo. Durante lo scorrimento alcune informazioni scompaiono dal basso dello schermo e nuovi dati compaiono in alto, o viceversa, consentendo in tal modo all'utente di vedere, porzione per porzione, tutto il file.

Ogni volta che viene effettuato uno scroll, CicloPrincipale: rivisualizza il testo sullo schermo. Innanzittutto viene utilizzato il comando CLS per ripulire lo schermo, dopo di che viene visualizzata la porzione di testo appropriata. Il valore massimo che può essere assunto dalla variabile LineaSup è 50. La variabile LineaUno contiene il numero relativo alla prima linea da visualizzare. Più precisamente, LineaUno contiene la posizione dell'elemento del vettore che deve essere visualizzato nella prima delle nove linee di schermo; infine vengono visualizzati gli otto elementi successivi.

Un esempio aiuta a chiarire meglio questo concetto. Se LineaUno vale 1, la prima linea visualizzata conterrà i valori di Numero$(1), Descrizione$(1) e Valore$(1). Dato che, come spiegato precedentemente, tutti questi elementi sono vuoti, la prima linea risulta essere bianca. La quinta linea è la prima linea raggiungibile con il cursore e risulta essere la linea di input numero 1. Dato che Numero$(5) contiene il numero 1, la variabile LineaUno contiene il numero della attuale linea di input.

Quanto detto potrebbe sembrare un po' confuso; non è il caso di preoccuparsi: quasi tutta la teoria risulta confusa prima di metterla in pratica.

E' ora possibile provare ad eseguire il programma, per osservarne il funzionamento; prima è però consigliabile salvare su dischetto il listato inserito. Se durante l'esecuzione dovessero verificarsi degli errori, si con-

siglia di controllare il testo battuto con quello del libro, procedendo eventualmente a modificare le parti che dovessero risultare differenti.

Utilizzo del programma

Quando tutto funziona correttamente è possibile provare ad inserire alcuni dati di input. Come si può notare, dopo aver inserito un dato e premuto il tasto <Return> il cursore si sposta lungo i diversi campi (cioè dalla colonna Descrizione a Valore).

- Inserire un certo numero di dati, in modo da portare il cursore alla linea di input numero 5.

Ora è possibile provare ad utilizzare i tasti <Cursore in alto> e <Cursore in basso> per gestire lo scroll del testo. Come si può notare le linee si spostano sullo schermo. Quando il cursore viene posizionato all'inizio del file, sopra alla linea di input si trovano quattro linee bianche. In questa situazione la variabile LineaUno vale 1, in quanto si sta lavorando con la prima linea di input, e la prima riga visualizzata sullo schermo contiene i valori dei primi elementi dei vettori Numero$, Descrizione$, Valore$ (che sono riempiti con spazi vuoti).

Se il cursore si trova sulla linea 5, LineaUno vale 5. Compreso questo esempio il resto del programma dovrebbe risultare abbastanza semplice da capire.

Le due linee successive controllano che lo scroll rimanga nei limiti del file: il cursore non può infatti essere posizionato su una linea precedente la linea numero uno, oppure su una linea successiva all'ultima (identificata dal valore di LineaSup). Quando l'utente tenta di fuoriuscire da tali limiti, in segno di avvertimento viene prodotto un suono utilizzando il comando BEEP.

TAB

Il programma visualizza una linea contenente gli identificatori dei vari campi (riga titolo) e le nove linee contenenti i dati. Per realizzare correttamente l'incolonnamento dei dati viene utilizzato un nuovo comando BASIC: il comando TAB (il termine TAB è un'abbreviazione di Tabulator). Il comando TAB viene utilizzato, esattamente come l'omonimo tasto nelle macchine

da scrivere o nei Word Processor, per posizionarsi in una colonna specificata (ovviamente, nel caso in esame, sullo schermo).

La linea:

```
PRINT TAB(10);"ciao"¶
```

visualizza la parola ciao a partire dalla decima colonna dello schermo. TAB può essere utilizzato per generare facilmente le spaziature necessarie per tabelle o liste.

La variabile DatiDesc viene utilizzata alla fine del CicloPrincipale: consentendo al programma di sapere se sta per essere inserita un'etichetta (nella colonna per il testo), oppure un valore (nella colonna numerica).

Vengono utilizzate le variabili ParteIniz e ParteFin per specificare i margini relativi ai campi per i due differenti tipi di input. Tali margini saranno poi utilizzati nella routine ImmettiTesto per determinare la lunghezza massima di input. Il valore della variabile xp, anch'esso utilizzato nella routine ImmettiTesto:, indica la posizione orizzontale del cursore e, quando comincia l'input, corrisponde alla colonna di partenza.

Dopo questi preparativi, viene richiamata la routine ImmettiTesto, al ritorno dalla quale viene cancellato il contenuto della variabile in$, in modo da poter riutilizzare successivamente la variabile stessa. Per finire CicloPrincipale: richiama nuovamente se stesso, per cui, se non vi sono interruzioni, il programma esegue il ciclo all'infinito.

Modifica dell'input da tastiera

Verrà ora analizzata la routine ImmettiTesto. In essa è inserita una routine specifica per l'acquisizione dell'input da tastiera. Normalmente, per acquisire input, vengono utilizzati i comandi INPUT e LINE INPUT. Durante l'esecuzione di questi comandi AmigaBASIC non è però in grado di controllare quali tasti vengano premuti. Dato che in questo caso si desidera avere il pieno controllo sull'input fornito, tali comandi non si rivelano utili; per questo motivo è stato utilizzato il comando INKEY$ che consente di acquisire ogni singolo carattere separatamente.

Il programma effettua per prima cosa un controllo sul valore di xp, valo-

re che deve essere sempre superiore al contenuto della variabile ParteIniz. In teoria questa eventualità non dovrebbe mai verificarsi, in quanto inizialmente il valore di xp viene posto uguale a quello di ParteIniz. Tuttavia, durante l'immissione dell'input, potrebbe verificarsi che, nell'eventualità di tentativo di correzione di errori, il cursore venga spostato a sinistra del margine di inizio del campo. La prima linea della routine ImmettiTesto: consente di evitare il verificarsi di tale situazione.

LOCATE

Il cursore rimane sulla stessa linea di schermo e non può essere spostato verso l'alto o verso il basso. La terza linea della routine utilizza il comando LOCATE 6,xp per posizionare il cursore correttamente.

Il cursore appare sotto forma di rettangolo pieno di color arancione. Viene utilizzato il comando COLOR 0,3 per definire l'arancione come colore di sfondo ed il blu come colore di primo piano. A questo punto, inserendo uno spazio, si ottiene il cursore della forma desiderata. Dopo che questo è stato realizzato, i colori vengono riportati alla situazione normale.

INKEY$

Segue poi il comando fondamentale di questa routine, cioè l'espressione i$=INKEY$. La variabile i$ contiene il carattere corrente di input. Se i$ è vuoto, significa che non è ancora stato premuto un tasto, per cui è necessario ricontrollare il contenuto di i$ fino a quando non viene premuto un tasto.

Buffer di tastiera

Se vengono premuti i tasti tanto velocemente che Amiga non riesce ad elaborare il carattere ricevuto prima del sopraggiungere di un nuovo carattere, nessuno dei caratteri in sovrappiù andrà perso. AmigaBASIC utilizza infatti un buffer di tastiera di 15 caratteri, una specie di contenitore, per memorizzare i caratteri che devono essere esaminati. Il comando INKEY$ analizza i contenuti di questo buffer. Se si batte un carattere dopo che il buffer è stato riempito, un piccolo suono segnala che ogni ulteriore carattere inviato verrà ignorato. Il segnale è lo stesso utilizzato da AmigaBASIC per comunicare l'indisponibilità di una finestra di input.

La parte rimanente della routine ImmettiTesto viene utilizzata per valuta-

re i caratteri di input ricevuti. Per effettuare tale operazione con INKEY$ è necessario utilizzare il comando CHR$. Il comando CHR$ converte una codifica ASCII nel corrispondente carattere. E' quindi possibile confrontare i caratteri ricevuti da INKEY$ (o la copia di questi memorizzata nella variabile in$) con CHR$(x).

Caratteri di controllo

CHR$(2) corrispone alla combinazione di tasti <Control> <B>, ottenuta premendo contemporaneamente il tasto <CTRL> ed il tasto <B>. Come la maggior parte dei caratteri di controllo <Control> <B> non viene visualizzato sullo schermo, ma questo non significa che non abbia effetto. Gran parte di questi caratteri possono avere effetti deleteri sull'esecuzione di un programma. E' possibile notare come, premendo la combinazione di tasti <Control> <G> nella finestra BASIC si ottenga un suono; la combinazione <Control> <C> provoca invece la terminazione del programma BASIC attualmente in esecuzione.

Inizio del File

<Control> <B> viene utilizzato nel programma per spostarsi all'inizio del file. La variabile LineaUno viene posta a 1, dopo di che la routine CicloPrincipale: si preoccupa di visualizzare le linee iniziali del file.

Fine del File

<Control> <E> (CHR$(5)) viene utilizzzato per spostarsi alla fine del file, cioè alla linea che risulta essere l'ultima contenente un dato di input. Potrebbero eventualmente esservi poi altre linee non contenenti alcun input.

Cancellazione di linee

<Control> <D> (CHR$(4)) viene utilizzato per effettuare la cancellazione di una linea. E' possibile utilizzare questa combinazione di tasti per effettuare la cancellazione della linea su cui è attualmante posizionato il cursore. La routine CancellaLinea gestisce questa procedura. Dopo aver eseguito questa routine il programma ritorna ad eseguire la routine CicloPrincipale:.

Inserimento di linee
<Control> <N> (CHR$(14)) viene utilizzato per l'inserimento di una nuova linea nella posizione corrente del cursore, operazione effettuata dalla routine InserisciLinea:.

Ecco un breve resoconto di queste quattro funzioni:

<Control> <B>	Inizio del File
<Control> <E>	Fine del File
<Control> <D>	Cancellazione linea corrente
<Control> <N>	Inserimento linea

Assegnazione dei codici di controllo
Quando vengono utilizzati all'interno di un programma dei codici di controllo, è necessario aver cura di assegnare a tali codici una diretta corrispondenza con l'azione svolta. Nel caso sia possibile, è preferibile sostituire tutte le sequenze <CTRL> con opzioni selezionabili da menù, in modo da poter inserire un nome esplicativo. Inoltre è spesso più semplice lavorare utilizzando il mouse che non la tastiera.

Le due linee seguenti di programma gestiscono i tasti <Cursore in alto> e <Cursore in basso>.

Spostamento in alto di una linea
La codifica ASCII del tasto <Cursore in alto> è 28. Quando viene premuto questo tasto, viene abbandonato il campo corrente di input e l'input dovrebbe essere inserito nella lista. La routine AccettaTesto: si preoccupa di gestire questo. Il cursore viene poi posizionato (modificando opportunamente il valore della variabile xp) nella colonna iniziale, in modo che esso venga a trovarsi all'inizio di un nuovo campo di input. La variabile LineaUno viene decrementata di uno e si verifica il ritorno alla routine CicloPrincipale:, in modo da effettuare lo scorrimento di una linea verso l'alto.

Spostamento in basso di una linea
La codifica del tasto <Cursore in basso> è 29. La procedura eseguita è la

stessa del caso precedente, con l'unica differenza che lo scorrimento viene effettuato verso il basso.

La variabile PosTesto determina la posizione del cursore relativamente all'input corrente. Se, ad esempio, il cursore è posizionato in colonna 14 ed il valore di ParteIniz è 10, significa che si è attualmente nella posizione numero 5 (14-10+1) relativamente all'input attuale.

La variabile Testo$ contiene il testo inserito (o modificato). Il valore viene acquisito dal campo Descrizione$ oppure dal campo Valore$ (a seconda del valore di DatiDesc).

La codifica del tasto <Cursore a destra> è 30. Se il cursore è posizionato in un campo in cui era già stato precedentemente inserito del testo, e viene premuto il tasto <Cursore a destra>, alla variabile i$ viene assegnato il carattere su cui era posizionato il cursore.

Return e Tab

La codifica del tasto <Return> è 13, mentre al tasto <Tab> è associato il valore 9. Nel programma in esame entrambi questi tasti sono utilizzati per la stessa funzione: terminare l'input del campo corrente. Viene richiamata la routine AccettaTesto:, in cui la variabile DatiDesc determina di quale campo si tratta. La formula (variabile)=1-(variabile) viene utilizzata per passare dal valore 0 a 1 e viceversa.

Se DatiDesc vale 0 il cursore passa dalla colonna Valore a quella Descrizione e viene eseguito un salto alla successiva linea. La variabile xp viene reinizializzata al valore della prima colonna e viene poi incrementata LineaSup (che contiene il numero relativo all'ultima linea contenente un input). Se il valore di LineaUno risulta maggiore di quello di LineaSup, tale valore viene assegnato a LineaSup. Ora che la routine ImmettiTesto: ha svolto il suo lavoro, il programma ritorna ad eseguire la routine CicloPrincipale:.

Backspace e movimenti all'indietro

La sezione seguente di programma intercetta i tasti <Backspace> e <Cursore a sinistra>. Entrambi questi tasti producono lo stesso effetto nel programma: cancellano il carattere a sinistra del cursore.

Vi sono infine due casi da analizzare: se il cursore è posizionato in un punto in cui vi è del testo precedente, il vecchio testo viene mostrato quando viene cancellato quanto appena inserito; se invece non vi è alcun testo dopo il nuovo input, viene semplicemente visualizzato uno spazio vuoto.

L'espressione seguente:

```
RIGHT$(Testo$,LEN(Testo$)-PosTesto+1)¶
```

ritorna la parte di testo che segue la posizione corrente del cursore.

Inserimento di input

Se viene compiuto un errore in fase di inserimento è necessario utilizzare il tasto <Backspace> fino a quando non viene cancellato tutto il testo errato.

Se si sta utilizzando un campo in cui è già contenuto testo, è possibile mouversi avanti e indietro lungo il testo. Quando viene inserito un nuovo carattere questo sostituisce quello posto nella stessa posizione. Spostando nuovamente in quella posizione il cursore ricompare il vecchio carattere. Quando viene premuto il tasto <Return>, viene accettato il testo fino alla posizione corrente del cursore. Il testo che segue tale posizione viene perso.

Cancellazione di caratteri

E' possibile cancellare i caratteri contenuti in un campo agendo nel modo seguente: la posizione del cursore (xp) viene decrementata di uno; quando viene raggiunto il valore di ParteIniz, viene prodotto un suono per avvisare che il cursore non può essere spostato ulteriormente. La stringa di input risulta essere lunga tanto quanto il segmento di stringa a sinistra della posizione del cursore, per cui, se il cursore viene portato all'inizio del campo, si ottiene la cancellazione completa del contenuto del medesimo. Dopo avere fatto questo, il programma ritorna all'inizio della routine ImmettiTesto:, per consentire l'inserimento di nuovi caratteri.

Come si può notare non è possibile inserire doppi apici nella stringa fornita in input. Tutti gli altri caratteri sono invece permessi. I doppi apici vengono invece sostituiti con gli apostrofi. I doppi apici in un testo pos-

sono generare problemi quando si procede alla memorizzazione di dati.

Limitazioni in input
Le ultime linee del ciclo ImmettiTesto: selezionano l'input fornito; esse agiscono in modo da controllare che non vengano visualizzati caratteri di controllo o tasti funzione, e quindi permettono l'inserimento solamente dei codici ASCII compresi tra 32 e 126. Se lo si desidera è possibile consultare la tabella riportata nelle Appendici per avere una corrispondenza tra questi codici ed i relativi caratteri. Se i$ risulta essere uno dei caratteri accettabili e xp è minore od uguale a ParteFin (in caso di uguaglianza Amiga produce un suono di avvertimento e rifiuta ogni ulteriore carattere inserito), il carattere fornito viene visualizzato sullo schermo ed alla stringa in$ verrà accodato un altro carattere. La variabile xp per il posizionamento del cursore viene incrementata di uno. Il programma esegue poi nuovamente il ciclo ImmettiTesto:. E' ora necessario dare uno sguardo alle subroutine che vengono richiamate dalla routine ImmettiTesto:, iniziando dalla routine AccettaTesto:.

Assegnamento di testo ad un vettore
La routine AccettaTesto: viene richiamata solamente quando l'input deve essere assegnato al vettore corrispondente: essa viene quindi eseguita solamente quando la stringa contenuta in in$ non è vuota, in quanto, nel caso in essa non vi sia nulla, non vi è ovviamente niente da assegnare. Se il cursore è posizionato nella prima posizione di un campo di input (punto in cui si trova quando viene visualizzato per la prima volta il cursore) e viene premuto il tasto <Return>, il tasto <Tab>, oppure uno dei tasti cursore, il contenuto del campo non viene modificato. Questo rende possibile la ricerca attraverso i listati.

Se in$ contiene una stringa, il suo contenuto viene assegnato al vettore Descrizione$ oppure al vettore Valore$, a seconda del valore di DatiDesc. Viene poi cancellato il contenuto di in$ in modo che tale variabile sia utilizzabile per contenere un nuovo input. La variabile AltriDati viene posta ad uno.

Modifiche del testo salvato
I dati possono essere modificati solamente nella routine AccettaTesto:. Se

l'utente abbandona il programma senza aver salvato i dati modificati, viene visualizzata una finestra di avvertimento. All'inizio del programma la variabile AltriDati vale zero. Se AltriDati vale uno sono state effettuate alcune modifiche. Una volta che AccettaTesto: ha terminato di svolgere il suo compito, il controllo ritorna alla linea da cui era stata effettuata la sua chiamata.

La subroutine successiva, CancellaLinea:, consente di cancellare una linea. Ogni linea successiva a quella cancellata, viene ricopiata sulla precedente, in modo da scalare tutte le linee verso l'alto. Agli elementi dei vettori Descrizione$ e Valore$ vengono assegnati i contenuti degli elementi successivi del vettore.

La linea in cui è posizionato il cursore scompare dallo schermo e viene sostituita dalla linea immediatamente successiva. La variabile LineaSup viene decrementata di una unità. Nel caso ci si trovi in fondo al file (IF LineaSup<1 THEN ...), LineaSup viene posto ad uno. Dopo avere effettuato queste operazioni la subroutine restituisce il controllo alla routine da cui era stata chiamata.

Inserimento di linee

La routine complementare a CancellaLinea: è la routine InserisciLinea:, utilizzata per l'inserimento di nuove linee. Prima di inserire effettivamente una linea, tale routine controlla che vi sia a disposizione spazio sufficiente. Nel caso LineaSup valga 50, la lista risulta essere completa e non possono quindi essere inserite altre linee. Per indicare questa situazione viene prodotto un suono di avvertiento.

Se LineaSup risulta essere minore di 50, la routine InserisciLinea: effettua lo spostamento di una posizione verso il basso di tutte le linee che seguono quella corrente, partendo dalla fine del file (LineaSup + 40) fino alla linea corrente. Viene utilizzato il passo -1 (STEP -1) per far in modo che il ciclo FOR ... NEXT agisca effettuando decrementi. Dopo aver terminato questo ciclo, l'ultima linea spostata in avanti viene cancellata, in modo da poter poi realizzare l'inserimento dei nuovi dati. La variabile LineaSup viene incrementata di una unità e viene infine eseguito il comando RETURN per restituire il controllo alla routine chiamante.

Ecco ora la routine che consente di leggere e scrivere su dischetto.

```
RegistraDati:¶
  MENU 1,0,0 : MENU 2,0,0 ¶
  MENU OFF¶
  GOSUB ImmettiNome¶
  WINDOW 1¶
  IF Nome$="" THEN FineRegistrazione¶
  OPEN Nome$ FOR OUTPUT AS 1¶
    PRINT #1,LineaSup+4¶
    FOR x=1 TO LineaSup+4¶
      WRITE #1,Descrizione$(x)¶
      WRITE #1,Valore$(x)¶
    NEXT x¶
  CLOSE 1¶
  ¶
FineRegistrazione:¶
  MENU 1,0,1 : MENU 2,0,1¶
  MENU ON¶
  AltriDati=0¶
RETURN¶
¶
RecuperaDati:¶
  IF AltriDati=1 THEN GOSUB Richiesta¶
  MENU 1,0,0 : MENU 2,0,0¶
  MENU OFF¶
  GOSUB ImmettiNome¶
  WINDOW 1¶
  IF Nome$="" THEN FineRecupero¶
  FOR x=1 TO 58¶
    Descrizione$(x)=""¶
    Valore$(x)=""¶
  NEXT x¶
  OPEN Nome$ FOR INPUT AS 1¶
    INPUT #1,NumeroDati¶
    LineaSup=NumeroDati-4¶
    FOR x=1 TO NumeroDati¶
      INPUT #1,Descrizione$(x)¶
```

```
        INPUT #1,Valore$(x)¶
      NEXT x¶
      LineaUno=LineaSup¶
    CLOSE 1¶
    ¶
FineRecupero:¶
    WINDOW 1¶
    COLOR 1,0¶
    CLS¶
    PRINT "Numero";TAB(10);"Descrizione";
        TAB(45);"Matrice"¶
    FOR x=LineaUno TO LineaUno+8¶
      PRINT Numero$(x);TAB(10);Descrizione$(x);
          TAB(45);Valore$(x)¶
    NEXT x¶
    MENU 1,0,1 : MENU 2,0,1¶
    MENU ON¶
    AltriDati=0¶
RETURN¶
        ¶
ImmettiNome:¶
    AltroNome$=Nome$¶
    WINDOW 2,"Immettere il nome del file:",(50,80)-
       (580,88),0,-1¶
    CLS¶
    LINE INPUT Nome$¶
    IF Nome$= "=" OR Nome$="*" THEN Nome$=AltroNome$¶
    WINDOW CLOSE 2¶
RETURN¶
```

Quando viene scelta l'opzione Registra dal menù a tendina Dati, il programma esegue la subroutine RegistraDati:. La gestione dell'event trapping rende possibile l'effettuazione di salvataggi in ogni istante. E' necessario ricordare che le modifiche effettuate sulla linea corrente non vengono registrate nel file fino a quando non viene premuto il tasto <Return> o il tasto <Tab>. Le modifiche non vengono trasferite su dischetto fino a quando non viene salvato il file.

La routine RegistraDati: disabilita innanzitutto i menù. Tutti i menù a tendina vengono visualizzati come menù mascherati. Viene poi richiamata la routine InserisciNome: utilizzata per inserire il nome del file. Se il contenuto di Nome$ risulta essere vuoto, non viene salvato niente ed il programma esegue direttamente le istruzioni che seguono l'etichetta Fine-Registrazione.

WRITE

Nel caso venga specificato il nome di un file, questo viene aperto in output e su di esso viene per prima cosa scritto il numero delle linee di dati (LineaSup + 4). Tale numero rappresenta il contatore utilizzato dal successivo ciclo FOR ... NEXT: che consente di memorizzare il contenuto di Descrizione$ e Valore$ nel file. Per realizzare l'operazione di scrittura viene utilizzato il comando WRITE.

Il comando WRITE funziona, sotto certi aspetti, come il comando PRINT. Anche esso consente infatti di scrivere dati (numeri o stringhe) sullo schermo, oppure in un file. La differenza consiste solamente in un paio di dettagli. Per evidenziarla è utile ricorrere ad un piccolo esempio esplicativo.

• Inserire nella finestra BASIC le linee seguenti:

```
PRINT 1,2,3;4;5;6¶
```

e

```
WRITE 1,2,3;4;5;6¶
```

WRITE e PRINT a confronto

Utilizzando le virgole come separatori all'interno di un comando PRINT, è possibile ottenere una stampa incolonnata, in cui ogni colonna risulta larga 15 caratteri. Il comando WRITE presenta invece semplicemente come output le virgole.

Quando viene utilizzato il punto e virgola con il comando PRINT, come carattere di separazione, i vari dati vengono riportati uno di seguito all'altro (i numeri positivi sono preceduti da uno spazio in sostituzione del segno +). Il comando WRITE trasforma i punti e virgola in virgole.

• Eseguire i seguenti due esempi:

```
PRINT "Ciao, ";"Come stai?"¶
```

e

```
WRITE "Ciao, ";"Come stai?"¶
```

Questi due comandi producono differenti risultati. Il comando WRITE aggiunge i doppi apici attorno alle stringhe; PRINT invece no.

E' opportuno essere a conoscenza di queste informazioni per poter gestire correttamente i file sequenziali in AmigaBASIC. Un file sequenziale è essenzialmente una lista di caratteri che si susseguono uno dietro l'altro. Per fare in modo che AmigaBASIC sia in grado di riconoscere dove termina un elemento ed inizia il successivo, è necessario utilizzare caratteri di separazione.

Carriage Return e Line Feed

Il codice del carattere per il Carriage Return è CHR$(13). Il tasto <Return> riceve il suo nome dal tasto della macchina da scrivere che esegue il ritorno del carrello (Carriage Return) e l'avanzamento di linea (Line Feed). I progettisti di calcolatori hanno utilizzato un'idea simile per il carattere di terminazione di una linea: il codice di Carriage Return (CHR$(13)). Il carattere con codifica CHR$(10) ha una funzione simile. Si tratta del carattere di avanzamento linea (Line Feed) che consente di avanzare di una linea sullo schermo o sulla stampante senza ritornare ad inizio linea. Per tale motivo è necessario che queste due codifiche vengano utilizzate insieme.

Il comando PRINT pone automaticamente tali codici dopo un'espressione che non termina con una virgola o con un punto e virgola, indipendentemente dal fatto che si stia visualizzando l'output sullo schermo o si stia stampando con una stampante.

Scrivendo in un file sequenziale con i comandi seguenti:

```
PRINT #1, "Ciao"¶
PRINT #1, 1,2,3¶
PRINT 4;5;6¶
```

si ottiene il seguente risultato:

```
Ciao <Line Feed> 1 <13 Spazi> 2 <13 Spazi> 3 <Line Fe-
ed> 4 5 6 <Line Feed>¶
```

Come si può vedere direttamente in questo esempio, quando un comando PRINT non viene terminato con un carattere di separazione, viene automaticamente aggiunto un Line Feed. Quando viene utilizzato come separatore la virgola, AmigaBASIC inserisce automaticamente gli spazi necessari per realizzare il posizionameto sulla colonna successiva. Quando viene invece utilizzato il punto e virgola i dati vengono presentati uno di seguito all'altro, ad eccezione del fatto che viene inserito uno spazio bianco in corrispondenza del segno + per i numeri positivi.

Problemi con INPUT#

Si possono verificare alcuni problemi durante la lettura di valori da un file sequenziale utilizzando il comando INPUT#, dato che questo comando legge fino ad incontrare il separatore successivo (le virgole vengono trattate come separatori).

E' possibile provare ad utilizzare il seguente esempio per creare un file sequenziale nella finestra BASIC.

```
OPEN "FileTesto" FOR OUTPUT AS 1¶
a$+"Ciao, come stai?"¶
b$="testo"¶
PRINT #1, a$¶
PRINT #1, b$¶
CLOSE 1¶
```

In questo modo sono state scritte due stringhe in un file sequenziale. Per leggerle è possibile utilizzare il programma seguente:

```
OPEN "FileTesto" FOR INPUT AS 1¶
INPUT #1,a$¶
INPUT #1,b$¶
?a$¶
?b$¶
```

Come si può notare si verificano alcuni errori. La stringa a$ contiene Ciao,

mentre la stringa b$ contiene Come stai. La virgola nel testo viene interpretata come separatore e quindi non compare in alcun punto la stringa testo. E' comunque possibile risolvere questo problema in modo molto semplice:

```
INPUT #1,c$¶
?c$¶
CLOSE 1¶
```

In tal modo si è potuto leggere tutti i dati del file sequenziale. La virgola genera comunque una serie di problemi durante la fase di lettura del file. Questo potrebbe causare una certa confusione all'interno del programma, dato che i contenuti dei campi vengono letti e scritti in maniera diversa.

Ulteriori informazioni relative al comando WRITE

Per neutralizzare gli eventuali separatori all'interno di stringhe il BASIC pone attorno alle stesse i doppi apici. Per questo si rileva di grande utilità il comando WRITE. Quando i dati vengono scritti utilizzando il comando WRITE si può essere sicuri di poter leggere i dati scritti sul file correttamente e nello stesso formato.

Il comando WRITE consente di inserire all'interno di una stringa qualsiasi carattere ad eccezione dei doppi apici. Nel caso si utilizzino i doppi apici le cose non funzionano correttamente. Per questo motivo nella routine TypeText, viene effettuato un controllo per impedire l'inserimento di doppi apici nella stringa; tale routine effettua la conversione automatica dei doppi apici in apostrofi.

La spiegazione di due sole linee di programma è stata abbastanza lunga; ha però consentito di chiarire alcune caratteristiche fondamentali dei file sequenziali, e di esporre alcune considerazioni relative alla gestione di tali file che si riveleranno sicuramente molto utili quando si inizieranno a scrivere propri programmi.

Chiusura del file

La routine RegistraDati: non contiene altre cose nuove. Il file viene chiuso utilizzando il comando CLOSE. La sezione FineRegistrazione: riattiva l'event trapping per i menù, e pone a zero la variabile AltriDati. Tale va-

riabile deve valere 0 in quanto i dati presenti in memoria coincidono con i dati memorizzati nel file; il suo valore dovrà essere aggiornato nel caso vengano apportate modifiche ai dati in memoria. E' ora necessario analizzare il modo in cui possono venire letti i dati memorizzati; la routine che si occupa di questa azione è la routine RecuperaDati:.

Cancellazione del vecchio file

Caricare dati nuovi significa, implicitamente, cancellare i dati relativi al file precedentemente in memoria. Per evitare di perdere dati modificati e non ancora salvati, prima di procedere alla cancellazione viene controllato il valore di AltriDati. Nel caso siano state effettuate modifiche, viene fornita all'utente la possibilità di effettuare il salvataggio di tali dati e viene eseguita la routine Richiesta:. Per prima cosa vengono disattivati i menù in modo da evitare che venga perso qualcosa o che venga interrotta la procedura.

La subroutine ImmettiNome: consente di inserire il nome di un file. Se non viene fornito alcun nome viene interrotto il caricamento e viene eseguita la routine FineRecupero:.

Il successivo ciclo FOR ... NEXT consente di cancellare i dati attualmente presenti nei vettori. Se non venisse eseguito tale ciclo e venisse caricato un file più corto di quello attualmente presente in memoria, ai dati del nuovo file potrebbero essere accodati alcuni dati del vecchio file. Dato che non si desidera che si verifichi questo fatto, viene effettuata la cancellazione di tutti i dati prima di procedere al caricamento del nuovo file.

Viene poi aperto in lettura il file da cui procedere al caricamento. Per prima cosa viene letto il numero di dati presenti nel file. Dato che i quattro elementi vuoti sono memorizzati all'inizio del file, alla variabile LineaSup viene assegnato il valore NumeroDati-4. Tutti i dati vengono poi caricati nei vettori Descrizione$ e Valore$, utilizzando il ciclo FOR ... NEXT.

Effettuato il caricamento, il cursore viene posizionato sulll'ultima linea del file, in modo da consentire di aggiungere nuovi dati. Questo si rivela molto utile nel caso di elenchi che devono essere costantemente aggiornati con nuovi elementi, come nel caso di un listino. Per effettuare questo, a LineaUno viene assegnato il valore della variabile LineaSup.

La routine FineRecupero: contiene un paio di linee presenti anche all'interno del CicloPrincipale:. La lista corrente viene visualizzata sullo schermo. Questo si rivela necessario in quanto il ciclo principale viene attivato solamente premendo un tasto. L'utente può però vedere ugualmente i nuovi dati prima che si verifichi questo. Vengono riattivati i menù e il valore di AltriDati viene posto a zero, in quanto i dati presenti in memoria appena caricati risultano identici a quelli memorizzati su dischetto. Il controllo viene poi restituito alla routine chiamante.

WildCard

La subroutine ImmettiNome: è pressocche identica a quella contenuta nel programma di disegno. Quando viene inserito uno dei caratteri speciali (ad esempio = oppure *), viene utilizzato nuovamente il nome dell'ultimo file a cui si è avuto accesso. Dopo aver inserito questo nome la finestra scompare dallo schermo.

Dopo aver inserito tutte le routine necessarie per il caricamento ed il salvataggio dei dati, deve ora essere inserita la routine Richiesta: che avverte l'utente che i dati correnti non sono ancora stati salvati.

```
Richiesta:¶
  WINDOW 2,"Attenzione !",(155,50)-(475,135),0,-1¶
  COLOR 0,1¶
  CLS¶
  LOCATE 3,4¶
  PRINT  "     I dati immessi non sono"¶
  PRINT  "        stati ancora registrati"¶
  PRINT : PRINT  "            Vuoi farlo ora?"¶
  LOCATE 9,15 : PRINT "Sì"¶
  LOCATE 9,26 : PRINT "No"¶
  LINE (95,58)-(146,76),0,b¶
  LINE (181,58)-(232,76),0,b¶
  BEEP¶
AspettaMouse:¶
  Test=MOUSE(0)¶
  WHILE MOUSE(0)=0¶
  WEND¶
  x=MOUSE(1) : y=MOUSE(2)¶
```

```
    IF 95<x AND x<146 AND 58<y AND y<76 THEN¶
      PAINT (97,59),3,0¶
      GOSUB RegistraDati¶
      PAINT (97,59),1,0¶
      WINDOW CLOSE 2¶
      RETURN¶
    END IF¶
    IF 181<x AND x<232 AND 58<y AND y<76 THEN¶
      PAINT (183,59),3,0¶
      WINDOW CLOSE 2¶
      RETURN¶
    END IF¶
    GOTO AspettaMouse¶
¶
CancellaDati:¶
    IF AltriDati=1 THEN GOSUB Richiesta¶
    RUN¶
    ¶
FineProgramma:¶
    IF AltriDati=1 THEN GOSUB Richiesta¶
    COLOR 1,0¶
    MENU RESET¶
    CLS¶
END¶
```

La routine Richiesta: è abbastanza semplice da comprendere. Essa visualizza un requester con sfondo bianco e testo blu. Nel requester vengono visualizzati il messaggio "I dati immessi non sono ancora stati registrati. Vuoi farlo ora?" ed i gadget per la conferma e la cancellazione. Viene prodotto inoltre un suono di avvertimento.

La routine AspettaMouse: si pone in attesa di un click su uno dei due gadget. Se l'utente seleziona il gadget Si, questo viene colorato di arancione e viene richiamata la subroutine RegistraDati:.

Attenzione:
Se viene fornito come nome di file una stringa vuota, i dati verranno persi irrimediabilmente.

Dopo che il programma ha terminato la procedura di salvataggio, il gadget viene riportato al suo colore originale, viene chiusa la finestra e viene restituito il controllo alla routine chiamante. Nel caso venga invece selezionato il gadget No, esso viene colorato in arancione, viene chiusa la finestra e viene restituito il controllo alla routine chiamante. Nel caso venga effettuato un click in un qualsiasi altro punto delle finestra, viene richiamata nuovamente la subroutine AspettaMouse.

Vengono infine utilizzate due routine abbastanza corte. La routine CancellaDati effettua la cancellazione dei dati presenti in memoria. Il metodo più rapido e veloce per riportare tutte le variabili ai loro valori originali è quello di far ripartire il programma utilizzando il comando RUN.

Utilizzando il comando RUN all'interno di un programma, è possibile far ripartire il programma. Quando viene incontrato tale comando vengono infatti azzerati tutti i valori delle variabili e degli elementi dei vettori ed il programma ricomincia completamente l'esecuzione. Questa azione si rivela molto utile in diverse circostanze, ma certamente non sempre. Normalmente viene utilizzato il comando GOTO per saltare all'inizio del programma e ricominciare l'esecuzione del programma. Lo stesso discorso vale per la routine Quit: che viene utilizzata per terminare il programma. In essa vengono riportati al loro valore originale i colori per lo sfondo e per il testo, vengono riattivati i menù originali del BASIC e viene chiuso lo schermo. Il comando END consente poi di ritornare in modo diretto.

A questo punto è opportuno salvare la versione attuale del programma su dischetto. Per poter poi aggiungere alla parte di gestione dei dati, la parte di visualizzazione grafica degli stessi, è necessario richiamare la routine grafica precedentemente salvata in formato ASCII e, utilizzando il comando MERGE, appendere tale routine al programma appena inserito.

Se sono state eseguite tutte le istruzioni indicate in questo paragrafo, vi dovrebbe essere, all'interno del cassetto DATA, una versione in formato ASCII del programma per la realizzazione di istogrammi e grafici a barra. Per controllare che questo sia effettivamente vero, è possibile utilizzare il comando FILES.

• Inserire la linea seguente nella finestra BASIC:

```
MERGE "New BarPie"¶
```

Dopo un paio di secondi viene aggiunta in fondo al programma la routine grafica.

• Effettuare immediatamente le operazioni di salvataggio del nuovo programma (utilizzando, se lo si desidera, lo stesso nome della versione senza parte grafica).

E' possibile a questo punto eseguire il programma per rilevare l'eventuale presenza di errori. Come al solito, quando si verifica un errore, viene visualizzato il relativo messaggio, compare la finestra contenente il listato del programma e la linea in cui si è verificato l'errore èevidenziata da un riquadro arancione. Per interpretare il tipo di errore verificatosi, è possibile fare riferimento alle descrizioni riportate nelle Appendici.

Esecuzione del programma

Il nuovo programma visualizza una finestra che riempie circa metà schermo. All'interno di questa finestra è possibile notare la presenza di un cursore arancione posizionato in corrispondenza dell'inizio della colonna descrizione sulla prima linea. A partire da questa posizione devono essere inseriti i dati da graficare. Il testo inserito in questa colonna verrà poi presentato sotto la barra relativa all'istogramma, oppure vicino alla corrispondente fetta della torta. E' possibile confermare l'inserimento effettuato premendo il tasto <Return>, il tasto <Tab> oppure uno dei tasti per lo spostamento verso l'alto o verso il basso. Il valore numerico corrispondente deve essere inserito nel campo Valore. Se il cursore è posizionato sul primo carattere inserito, tale input non può essere modificato. In caso contrario il programma accetta il testo fino alla posizione corrente del cursore, mentre il testo che segue il cursore viene cancellato. Per effettuare la cancellazione di caratteri è necessario utilizzare il tasto <Backspace>, oppure il tasto <Cursor Left>. Il testo precedentemente sovrascritto riappare.

Lo schermo contiene sempre nove linee di dati. L'intera lista può contenere un numero massimo di cinquanta elementi. E' possibile utilizzare la combinazione di tasti <Control> <B> per spostarsi all'inizio della lista, <Control> <E> per spostarsi alla fine, <Control> <D> per cancellare una linea e <Control> <N> per inserire una nuova linea.

Salvataggio di una lista
Per salvare un elenco di dati è necessario selezionare l'opzione Save del menù a tendina Registra. L'opzione Recupera carica i dati memorizzati in precedenza. E' possibile utilizzare Cancella per effettuare la cancellazione dell'area di lavoro e Fine per terminare il programma.

Selezione del tipo di grafico
Per scegliere il tipo di grafico da visualizzare, è necessario effettuare la scelta utilizzando il menù a tendina Grafici. Viene costruito il grafico e successivamente un piccolo requester richiede che venga premuto un tasto qualsiasi. E' possibile porre in secondo piano questo requester effettuando un click sull'apposito gadget. Premere poi un tasto qualsiasi per ritornare allo schermo dedicato all'inserimento.

Tricks & Tips
Per concludere ecco alcuni piccoli trucchetti per riuscire ad ottenere i migliori risultati con il programma:

- Aggiungere una ulteriore linea ad un file inserendo un numero molto più grande di quelli presenti per utilizzarlo poi come valore di confronto. Ad esempio, se il valore più grande contenuto nel file è 898, inserire in un'altra linea il valore 1000. Questo rende la scala molto più chiara.

- Se si hanno molti numeri piccoli, alternarli con quelli più grandi, in quanto, se tutti i valori molto piccoli sono in fondo al grafico, le scritte relative possono sovrapporsi.

- E' opportuno utilizzare etichette abbastanza corte quando si intende visualizzare un grafico a torta, per evitare che esse fuoriescano dallo schermo o si sovrappongano al grafico. E' inoltre consigliabile tenere abbastanza corte anche le etichette per gli istogrammi, per non sprecare spazio per le barre.

- Se si desiderano utilizzare colori particolari, è necessario inserire un ulteriore comando PALETTE all'interno della routine Inizializzazione: del programma.

3.5 Amiga e i suoi amici: i device periferici

Questo paragrafo introduce l'argomento relativo ai device periferici. Tali device sono componenti hardware connessi ad Amiga, solitamente attraverso una serie di cavi, ma che non fanno direttamente parte del computer; si tratta infatti di accessori opzionali. E' possibile collegare differenti tipi di device periferici ad Amiga. Anche se non si possiede una stampante, e non sono state utilizzate le varie interfacce hardware di Amiga, le pagine seguenti si rivelano comunque interessanti, in quanto forniscono informazioni utili riguardo alle possibilità di utilizzo di Amiga in connessione con device aggiuntivi.

Il primo tipo di periferica che verrà analizzato è il disco RAM di Amiga.

Pro e contro del disco RAM

Sia AmigaBASIC che AmigaDOS consentono di utilizzare parte della memoria di Amiga come disk drive virtuale, cioè RAM disk. RAM, come già fatto rilevare, costituisce la sigla per Random Access Memory (memoria ad accesso casuale).

Utilizzando un RAM disk le informazioni sono memorizzate nella memoria centrale del calcolatore invece che su un dischetto. Questo significa che è possibile leggere e scrivere dati e programmi in un RAM disk esattamente come con un normale floppy disk. Un disco RAM può contenere anche directory e cassetti.

L'efficienza di un RAM disk dipende dalla quantità di memoria disponibile. Il disco RAM alloca la memoria dinamicamente: in altre parole utilizza la minima quantità di memoria necessaria e solamente quando realmente necessario. La memoria utilizzata dal RAM disk non può essere utilizzata, nel medesimo istante, da AmigaBASIC o da altri programmi. Se viene cancellato un programma memorizzato nel disco RAM, la memoria occupata

da quel programma viene immediatamente liberata per altri usi.

- Per passare dalla teoria alla pratica provare ad inserire la seguente linea nella finestra BASIC:

```
CHDIR "RAM:"¶
```

RAM:
Quando viene inserito per la prima volta questo comando AmigaBASIC richiede l'inserimento del dischetto Workbench per accedere al programma di gestione del RAM disk.

E' possibile utilizzare il comando FILES per esaminare il contenuto del disco RAM. Non vi è alcun nome per il disco, come è possibile rilevare dall'esame della scritta Directory of [ ]. La directory del disco RAM potrebbe contenere il file BasicClip, in quanto, utilizzando le opzioni Cut, Copy e Paste del menù a tendina Edit, AmigaBASIC inserisce il contenuto della Clipboard nel disco RAM.

- Inserire alcune linee di programma nella finestra LIST e salvarle all'interno del disco RAM.

- Inserire la seguente linea nella finestra BASIC:

```
SAVE "Prova"¶
```

Pressochè contemporaneamente alla pressione del tasto <Return>, viene visualizzato sullo schermo il messaggio di prompt OK. Una delle caratteristiche principali del disco RAM è infatti quella di essere estremamente veloce. Per le operazioni di lettura e scrittura su disco RAM è infatti richiesta una quantità di tempo ridottissima, in quanto i file devono essere semplicemente spostati all'interno della memoria centrale.

Ulteriori informazioni sui file .Info
Dopo aver effettuato il salvataggio, la directory del disco RAM contiene i due file Prova e Prova.Info. AmigaBASIC, durante l'operazione di salvataggio, crea automaticamente un file con estensione .Info associato al file salvato.

NOTA:
La versione 1.1 del Workbench non è in grado di aprire una finestra per il disco RAM.

- Inserire la linea seguente:

```
KILL "Prova.Info"¶
```

Nel caso si possieda la versione 1.1, è opportuno contattare il proprio rivenditore di fiducia, per ottenere al più presto la nuova versione del Workbench.

- Cancellare le informazioni contenute in memoria utilizzando il comando NEW e caricare poi il programma Prova dal disco RAM:

```
LOAD "Prova"¶
```

Il disco RAM non è eterno...
La velocità di lavoro del disco RAM è veramente impressionante; programmi di dimensioni rilevanti possono essere caricati in frazioni di secondo, in tempi pressochè minimi rispetto a quelli necessari per il caricamento da floppy. Un utilizzo vantaggioso del disco RAM è quindi quello di utilizzarlo per i salvataggi intermedi di un programma che può essere trasferito su floppy al termine della sua scrittura. In questo modo il lavoro risulta essere fortemente velocizzato.

Vi sono numerosi casi in cui risulta vantaggioso utilizzare un disco RAM. Tuttavia, è necessario ricordare che fino a quando non si è provveduto a salvare i dati su floppy, tutte le informazioni memorizzate su un disco RAM sono solo temporanee. Una caduta di tensione, una mancanza di corrente, oppure un crash di sistema sono sufficienti per causare la perdita totale di tali dati.

NOTA:
Alcuni programmi Amiga non funzionano quando è presente il disco RAM. Nel caso si possieda un Amiga 500 privo di espansione è necessario disabilitare il disco RAM. La procedura da effettuare è già stata descritta, ma è forse opportuno presentarla nuovamente.

- Accendere Amiga ed inserire, in seguito alla richiesta del computer, il dischetto Workbench.

- Quando compare lo schermo blu, premere la combinazione di tasti <CTRL> <D> che porta alla comparsa del simbolo 1> per indicare che si sta interagendo con il CLI.

- Battere LoadWB ed attendere fino a quando non compare nuovamente sullo schermo il simbolo 1>.

- Battere, a questo punto, EndCLI.

Stampanti

E' ora necessario analizzare un ulteriore accessorio che può essere connesso ad Amiga: la stampante. Anche nel caso non si possieda un componente hardware di questo tipo, è consigliabile leggere le pagine seguenti per due buoni motivi: innanzitutto, è possibile che quanto prima si entri in possesso di un tale dispositivo; secondariamente, non è sicuramente un danno conoscere i comandi per la gestione delle stampe.

Prima di iniziare a lavorare con una stampante, è necessario effettuare alcuni preparativi. Per prima cosa è necessario acquistare un cavo per il collegamento della stampante ad Amiga. Tale cavo è reperibile presso ogni rivenditore Amiga.

NOTA:

Vi è una grande differenza tra il cavo di collegamento necessario per il modello Amiga 1000 e quello per i modelli 500 e 2000. E' necessario assicurarsi di essere in possesso del cavo di connessione appropriato, in quanto un cavo diverso potrebbe procurare danni al computer.

E' necessario selezionare alcune opzioni delle Preferences del Workbench per informare Amiga del tipo di programma di controllo della stampante (printer driver) da utilizzare. Programmi di questo tipo sono programmi di utilità che controllano il formato in cui i dati sono inviati al particolare modello di stampante collegata. Il Workbench contiene programmi di controllo per tutte le stampanti più diffuse ed usate. Gli sviluppatori della Commodore e di altre case lavorano attivamente per realizzare printer dri-

ver per altri tipi di stampanti (nel caso la stampante di cui si è in possesso non fosse compresa nell'elenco interno alle Preferences, è opportuno consultare il manuale fornito con Amiga). Dopo aver selezionato il driver corretto per la propria stampante, quello che rimane da fare è accendere la stampante stessa.

LPRINT

Il comando più semplice per inviare dati alla stampante è il comando LPRINT. La lettera L identifica la parola Line. Il comando LPRINT funziona esattamente come il comando PRINT: l'unica differenza è che esso invia l'output alla stampante e non allo schermo.

```
LPRINT "Ciao, come stai?"¶
```

Prima di iniziare la stampa, Amiga deve effettuare il caricamento del printer driver dal dischetto del Workbench. Dopo questa operazione verrà stampata la linea "Ciao, come stai?". AmigaBASIC, infatti, ha ora tutte le informazioni necessarie riguardo alla stampante.

LPRINT consente di utilizzare gli stessi caratteri separatori utilizzati normalmente dal comando PRINT:

```
LPRINT "Ciao",5,"Amiga";"Prova"¶
```

Il comando LPRINT ed i separatori

Nel paragrafo precedente sono stati discussi i codici di controllo ed i vari separatori. Come è possibile ricordare, quando la linea viene terminata con un punto e virgola, i dati in output non vengono stampati immediatamente in quanto essi vengono memorizzati in un buffer fino a quando non viene incontrato un Line Feed (codice CHR$(10)). La linea seguente produce due Line Feed:

```
LPRINT CHR$(10)¶
```

Il primo Line Feed viene realizzato direttamente dal codice CHR$(10), mentre il secondo viene prodotto automaticamente dal comando LPRINT. La stampante potrebbe non interpretare correttamente il codice del Line Feed e stampare invece qualcosa sulla stessa linea. Nel caso si verifichi questo, è opportuno consultare il manuale della stampante e controllare la selezione di stampante effettuata nelle Preferences: potrebbe infatti essere

necessario modificare la configurazione degli interruttori DIP della stampante. Nel caso non si riesca a risolvere il problema, è consigliabile rivolgersi al proprio negoziante di fiducia.

LLIST

E' possibile utilizzare il comando LLIST per stampare il listato di un programma su stampante durante la fase di sviluppo o di test del programma stesso. Come esempio, si può provare ad inserire un breve programma, e battere poi la linea seguente:

```
LLIST¶
```

La stampante dovrebbe stampare il listato del programma. AmigaBASIC risulta disabilitato durante la stampa. La procedura di stampa richiede una gran quantità di tempo nel caso di un programma piuttosto lungo (come ad esempio il programma di disegno). In tali casi il multitasking si rivela di grande aiuto fornendo la possibilità di proseguire il lavoro, senza dover aspettare la fine della stampa. Ad esempio, è possibile porre in secondo piano AmigaBASIC ed interagire nel frattempo con un altro programma. La possibilità di utilizzo del multitasking dipende ovviamente dalla quantità di memoria disponibile.

La stampante potrebbe avere un buffer di memoria che consente di memorizzare i dati da stampare all'interno della sua RAM e segnalare ad Amiga che la stampa è già stata effettuata, in modo tale che il computer possa dedicarsi ad altri compiti, anche se la stampante deve continuare il suo lavoro per altri 15 minuti. E' possibile forzare, in ogni momento, l'interruzione della stampa, utilizzando la combinazione di tasti <Control> <C>.

I comandi esposti sono i comandi principali di stampa dell'AmigaBASIC: AmigaBASIC prevede però molte altre facilità per la gestione delle stampe.

PRT:

E' stato predisposto un device, concettualmente simile al RAM disk, di nome PRT (printer), che fornisce un altro modo per effettuare la stampa di un listato:

```
LIST ,"PRT:"¶
```

LIST
E' possibile indicare al comando LIST dove ridirigere il listato prodotto, aggiungendo una serie di informazioni specifiche. E' possibile anche scrivere un file su dischetto utilizzando questo metodo. Ad esempio, per salvare un file su RAM disk, è possibile inserire la linea seguente:

```
LIST ,"RAM:Prova"¶
```

Questo comando produce un file di nome Prova in cui il programma viene memorizzato in formato ASCII. E' possibile utilizzare il comando LIST come alternativa al comando SAVE ,A.

SCRN:
Un altro device interessante è il device SRC: (Screen). Inserendo la linea:

```
LIST ,"SCRN:"¶
```

è possibile visualizzare l'intero listato del programma nella finestra BASIC.

LPT1:
Ritornando al problema relativo all'output della stampante, è possibile utilizzare tanto il device PRN: quanto il device LPT1: come nomi identificativi della stampante; essi produono infatti gli stessi identici risultati. La Microsoft ha utilizzato entrambi questi due nomi per rendere AmigaBASIC il più possibile compatibile con le altre sue versioni di BASIC. Il BASIC Microsoft per PC IBM utilizza il nome LPT1: (Line printer 1) per identificare la prima stampante collegata al calcolatore. Per facilitare il compito di trasferire un programma realizzato con un PC IBM su Amiga, la Microsoft ha progettato AmigaBASIC in modo tale da essere in grado di riconoscere quel particolare nome per il device, e poter inviare l'output al device standard di stampa PRT:.

File di output
E' possibile utilizzare il nome di un device per aprire un file in output, in modo tale da poter inviare direttamente dati ai vari device:

```
OPEN "PRT:" FOR OUTPUT AS 1¶
```

Questa linea invia tutto l'output fornito per il file 1 alla stampante. Se AmigaBASIC risponde con il messaggio di errore "File Already Open", significa che il device PRT: è già stato aperto in questa modalità. AmigaBASIC potrebbe, talvolta, rifiutarsi di effettuare una seconda connessione. Nel caso si verifichino alcuni problemi di questo tipo, dato che solo ora si sta cominciando a sperimentare queste cose, è opportuno cancellare preventivamente la memoria utilizzando il comando NEW. Si può poi provare ad inserire la linea seguente:

```
PRINT#1, "Ciao"¶
```

Non si deve rimanere sorpresi nel caso in cui il testo non venga stampato immediatamente: AmigaBASIC inizializza sempre un buffer per i file per evitare di dover trasferire immediatamente ogni carattere.

Il contenuto del buffer viene trasferito quando viene utilizzato il comando CLOSE o quando il buffer è pieno. E' questa la sola differenza tra LPT1: e PRT:. LPT1: stampa immediatamente, mentre PRT: utilizza un buffer di trasferimento. Inserendo la linea seguente si avvia la stampa:

```
CLOSE 1¶
```

I/O

Utilizzando questi comandi è possibile aprire file per ogni tipo di device, scegliendo ovviamente la modalità corretta. Aprire la stampante in modalità FOR INPUT provoca il messaggio di errore "Bad File Mode" o "Device I/O Error", dove il termine I/O costituisce una abbreviazione di Input/Output.

Nel caso si utilizzino i nomi dei vari device senza porre molta attenzione, può verificarsi che AmigaBASIC si rifiuti di aprire un qualche device. E' possibile, in questo caso, ricaricare il Workbench, per ripristinare le condizioni standard.

KYBD:

E' possibile utilizzare la tastiera (keyboard KYBD:) FOR INPUT verso un programma. Il listato che segue rappresenta un semplice caso di programma che accetta i caratteri battuti a tastiera, li visualizza sullo schermo e li presenta in output sulla stampante.

```
OPEN "KYBD:" FOR INPUT AS 1¶
OPEN "SCRN:" FOR OUTPUT AS 2¶
OPEN "KYBD:" FOR OUTPUT AS 3¶
WHILE i$<>CHR$(138)¶
  i$=INPUT$(1,1)¶
  PRINT#2 ,i$:¶
  IF i$=CHR$(13) THEN i$=CHR$(10)¶
  IF i$+CHR$(8) THEN i$=CHR$(127)¶
  PRINT#3 ,i$¶
WEND¶
CLOSE 1,2,3¶
```

Vengono aperti in lettura la tastiera ed in scrittura lo schermo e la stampante (del resto non potrebbe essere altrimenti).

Premendo <F10> viene forzata la terminazione del programma.

Il comando INPUT$ viene utilizzato per caricare file oggetto, come verrà spiegato tra breve.

Prima di inviare l'output alla stampante vengono effettuate alcune conversioni. Se l'utente preme il tasto <Return> (codifica CHR$(13)), esso viene convertito in CHR$(10), in modo che la stampante esegua un Line Feed. Se l'utente preme il tasto <Backspace>, il carattere viene convertito nel carattere con codifica CHR$(127), codifica che identifica, per la quasi totalità delle stampanti, il carattere per la cancellazione nel buffer. Ogni carattere viene inviato alla stampante non appena viene premuto il relativo tasto; tuttavia, fino a quando esso non viene stampato, è possibile intervenire per cancellare eventuali errori annullando l'ultimo carattere del buffer.

Se questo non dovesse funzionare sulla stampante di cui si è in possesso, è opportuno consultare attentamente il manuale in dotazione. Gran parte dei manuali contengono una tabella relativa all'insieme di caratteri utilizzati che fornisce informazioni relativamente ai caratteri di controllo che consentono il cambiamento dello stile di stampa, la spaziatura tra le linee ed altre cose varie.

Codici di Escape
Vi sono numerosi codici di Escape (<ESC> Code), che possono essere utilizzati sulla propria stampante, nel caso si tratti di una stampante Epson oppure Epson-compatibile. Si sarà potuto notare il tasto <ESC> sulla tastiera di Amiga. Tale tasto corrisponde alla codifica CHR$(27) e può essere utilizzato per produrre alcuni effetti interessanti. Prima di affrontare questi argomenti è comunque opportuno analizzare un attimo la storia e gli sviluppi subiti nel tempo dalle stampanti.

Printer Story
Quando, molti anni fa, venne sviluppata la codifica standard ASCII, non esistevano virtualmente stampanti potenti per i piccoli calcolatori. I vari utilizzatori di calcolatori non professionali potevano dirsi contenti se le loro stampanti erano in grado di stampare in lettere maiuscole e minuscole. Nessuno pensava che avrebbero mai potuto stampare in corsivo, grassetto od in qualità lettera (letter quality).

Aggiungere codici per il controllo della stampante
Di conseguenza le posizioni della tabella ASCII non utilizzate per codificare lettere, numeri o caratteri particolari (in totale circa 30 codici inutilizzati) erano sufficienti per attivare i primitivi effetti speciali di tali stampanti. Ad esempio, CHR$(14) era utilizzato, su numerose stampanti, per attivare la stampa con caratteri molto larghi. Al giorno d'oggi molte codifiche, come ad esempio CHR$(10) e CHR$(13), sono state utilizzate per ulteriori compiti e non vi è più un numero sufficiente di codifiche inutilizzate per abilitare gli effetti speciali.

Funzionamento dei codici di Escape
Gli specialisti di calcolatori avevano la necessità di espandere il numero dei codici di controllo. Per questa ragione i progettisti decisero di utilizzare il carattere 27 per realizzare sequenze di Escape. Il carattere 27 informa la stampante di non stampare il carattere che lo segue immediatamente, ma di interpretarlo come carattere di controllo. Ad esempio, quando la stampante incontra un carattere con codifica CHR$(69), stampa una E. Se invece incontra la codifica CHR$(27) seguita dal carattere con codifica CHR$(69), in altre parole la sequenza <ESC> <E>, la stampante interpreta 69 come carattere di controllo e determina la funzione associata alla sequenza di Escape <ESC> <E>.

La casa costruttrice di stampanti Epson ha sviluppato una codifica standard molto estesa per quanto riguarda i caratteri di controllo utilizzata con le stampanti a matrice di punti. In tale codifica standard per tutte le stampanti Epson ed Epson compatibili, la sequenza di Escape <ESC> <E> abilita la stampa in grassetto. Gran parte delle stampanti utilizzano questa codifica standard per gestire i caratteri di controllo.

Altre sequenze di Escape

Per numerose buone ragioni molte stampanti non sono tuttavia Epson compatibili. Alcuni produttori di stampanti hanno infatti pensato di utilizzare un metodo diverso per la gestione dei codici di Escape, a volte più efficace e semplice rispetto allo standard Epson, oppure di seguire un altro standard per quanto riguarda le stampanti, come ad esempio lo standard IBM.

Normalmente un programma di controllo della stampante funziona correttamente solamente con il tipo di stampante specifica per il quale è stato realizzato. Modificare il printer driver per renderlo utilizzabile anche con una stampante di tipo diverso, richiede a volte uno sforzo abbastanza rilevante. Quando si acquista una nuova stampante, sorge generalmente il problema della compatibilità con la vecchia stampante. Questo significa che diventa necessario cambiare il particolare printer driver per gestire gli eventuali caratteri di controllo differenti od aggiuntivi.

Printer driver

Gli sviluppatori di Amiga hanno ritenuto opportuno risolvere il problema in un modo particolarmente brillante. Essi hanno infatti stabilito una tabella standard di codici di controllo per la stampante da inserire in ogni programma Amiga. Questi codici di controllo vengono poi convertiti da ognuno dei particolari printer driver in caratteri speciali riconoscibili dalla stampante. In questo modo la codifica standard Amiga per le stampanti può essere facilmente adattata ad una gran numero di stampanti, ognuna delle quali supportata dal suo particolare driver.

Prima di effettuare la prima stampa, è necessario selezionare il driver corrispondente alla stampante di cui si è in possesso, interagendo opportunamente con le Preferences del Workbench e salvando su dischetto la scelta effettuata per non dover successivamente ripetere tale operazione. Nel caso la stampante di cui si è in possesso non fosse com-

presa nella lista, non è il caso di farsi prendere dal panico. Molte stampanti sono state infatti realizzate rispettando e copiando le caratteristiche di modelli molto popolari e potrebbe quindi verificarsi il caso che si riesca ad utilizzare la propria stampante utilizzando il driver specifico di un altro modello.

Il driver Epson funziona, ad esempio, con la maggior parte delle stampanti a matrice di punti. La cosa migliore è comunque quella di fare riferimento al proprio rivenditore di fiducia per ogni tipo di consiglio. Tuttavia, nel caso si possieda una stampante a margherita, è possibile contattare direttamente la Commodore o la fabbrica produttrice della stampante (fortunatamente molti costruttori di stampanti utilizzano le codifiche standard Amiga riportate nelle pagine seguenti).

Ogni singolo printer driver elencato nelle Preferences agisce come una specie di interprete: esso riceve infatti informazioni in un linguaggio particolare (la codifica standard dei caratteri di controllo Amiga) e li traduce immediatamente in informazioni in un altro linguaggio (caratteri di controllo che la stampante specifica è in grado di comprendere). Se il driver specifico può essere trovato nelle Preferences, Amiga è in grado di compiere la traduzione.

Le tabelle 8a, 8b e 8c mostrano un panorama completo dei codici di controllo Amiga. E' opportuno consultare il manuale specifico della stampante per ottenere ulteriori informazioni sugli effetti di ogni singolo codice sulla propria stampante.

Ecco un semplice esempio di utilizzo di questi codici: per specificare alla propria stampante di stampare in grassetto, è sufficiente inserire la linea seguente:

```
LPRINT CHR$(27);"[1m";"Ciao!"¶
```

Naturalmente alcune stampanti possono realizzare un numero di effetti superiori rispetto ad altre.

Quando si utilizza un carattere di controllo che la stampante non è in grado di comprendere, quel carattere viene semplicemente ignorato.

Carattere di controllo	Effetto
CHR$(27)"c"	Inizializzazione stampante
CHR$(27)"#1"	Disattiva altri modi
CHR$(27)"D"	Line Feed
CHR$(27)"E"	Line Feed + Carriage Return
CHR$(27)"M"	Una linea verso l'alto
CHR$(27)" [0m"	Stampa normale
CHR$(27)"[1m"	Attiva Grassetto
CHR$(27)"[22m	Disattiva Grassetto
CHR$(27)"[3m"	Attiva Corsivo
CHR$(27)"[23m"	Disattiva Corsivo
CHR$(27)"[4m"	Attiva Sottolineatura
CHR$(27)"[24m"	Disattiva Sottolineatura
CHR$(27);x;"m"	Clr.: Sf. x=30-39 P.Piano x=40-49
CHR$(27)"[0w"	Dimensione Font Normale
CHR$(27)"[2w"	Attiva Stampa Elite
CHR$(27)"[1w"	Disattiva Stampa Elite
CHR$(27)"[4w"	Attiva Stampa Condensed
CHR$(27)"[3w"	Disattiva Stampa Condensed
CHR$(27)"[6w"	Attiva Stampa Larga
CHR$(27)"[5w"	Disattiva Stampa Larga
CHR$(27)"[2"CHR$(34)"z"	Attiva Letter Quality
CHR$(27)"[1"CHR$(34)"z"	Disattiva Letter Quality
CHR$(27)"[4"CHR$(34)"z"	Attiva doppia passata
CHR$(27)"[3"CHR$(34)"z"	Disattiva doppia passata
CHR$(27)"[6"CHR$(34)"z"	Attiva stampa sfumata
CHR$(27)"[5"CHR$(34)"z"	Disattiva stampa sfumata

Tabella 8a: caratteri di controllo standard per le stampanti.

Carattere di controllo	Effetto
CHR$(27)"[2v"	Attiva esponente
CHR$(27)"[1v"	Disattiva esponente
CHR$(27)"[4v"	Attiva pedice
CHR$(27)"[3v"	Disattiva pedice
CHR$(27)"L"	Esponente a passo ridotto
CHR$(27)"K"	Pedice a passo ridotto
CHR$(27)"[0v"	Modalità normale
CHR$(27)"[2p"	Attiva stampa proporzionale
CHR$(27)"[1p"	Disattiva stampa proporzionale
CHR$(27)"[0p"	Cancella spaziatura proporzionale
CHR$(27)"[";x;"E"	Spaziatura proporzionale = x
CHR$(27)"[5F"	Spostamento a sinistra
CHR$(27)"[7F"	Spostamento a destra
CHR$(27)"[6F"	Attiva block character
CHR$(27)"[0F"	Disattiva block character
CHR$(27)"[3F"	Regolazione larghezza carattere
CHR$(27)"[1F"	Centratura
CHR$(27)"[0z"	Spaziatura linee 1/8"
CHR$(27)"[1z"	Spaziatura linee 1/6"
CHR$(27)"[";x;"t"	Lunghezza pagina = x linee
CHR$(27)"[";x;"q"	Salto pagina a linea x
CHR$(27)"[0q"	Salto pagina
CHR$(27)"(B"	Selezione caratteri Americani
CHR$(27)"(R"	Selezione caratteri Francesi
CHR$(27)"(K"	Selezione caratteri Tedeschi
CHR$(27)"(A"	Selezione caratteri Inglesi

Tabella 8b: caratteri di controllo standard per le stampanti.

Carattere di controllo	Effetto
CHR$(27)"(E"	Selezione caratteri Danesi (n. 1)
CHR$(27)"(H"	Selezione caratteri Svedesi
CHR$(27)"(Y"	Selezione caratteri Italiani
CHR$(27)"(Z"	Selezione caratteri Spagnoli
CHR$(27)"(J"	Selezione caratteri Giapponesi
CHR$(27)"(6"	Selezione caratteri Norvegesi
CHR$(27)"(C"	Selezione caratteri Danesi (n. 2)
CHR$(27)"#9"	Posizionamento margine sinistro
CHR$(27)"#0"	Posizionamento margine destro
CHR$(27)"#8"	Posizionamento testata pagina
CHR$(27)"#2"	Posizionamento piede pagina
CHR$(27)"#3"	Annullamento margini
CHR$(27)"[";x;y;"r"	Posizionamento relativi testata e piede pagina
CHR$(27)"[";x;y;"s"	Posizionamento relativo margine destro e sinistro
CHR$(27)"H"	Inizializzazione tabulatore or.
CHR$(27)"J"	Inizializzazione tabulatore vr.
CHR$(27)"[0g"	Annullamento Tab. orizzontale
CHR$(27)"[3g"	Annullamento tutti Tab. orizz.
CHR$(27)"[1g"	Annullamento Tab. verticale
CHR$(27)"[4g"	Annullamento tutti Tab. vert.-
CHR$(27)"#4"	Annullamento tutti Tab.
CHR$(27)"#5"	Reinizializzazione standard Tab.

Tabella 8c: caratteri di controllo standard per le stampanti.

Informazioni sulle interfacce
E' possibile creare un printer driver personalizzato che includa i propri codici di controllo; tuttavia la conversione dei codici di controllo richiede un grande sforzo di programmazione. Vi è comunque una soluzione abbastanza semplice: è possibile filtrare i caratteri di controllo utilizzando una interfaccia connessa con la propria stampante.

Non è ancora stato detto nulla riguardo alle interfacce. Amiga possiede un certo numero di "porte" attraverso le quali i dati vengono trasferiti da ed ai device periferici. Il nome identificativo dell'interfaccia parallela è PAR:, mentre il nome identificativo dell'interfaccia seriale è SER:.

PAR:
La maggior parte delle stampanti viene connesso ad Amiga attraverso la porta parallela, ma una stampante potrebbe essere collegata anche attraverso la porta seriale. Una interfaccia parallela presenta un numero maggiore di linee per i dati, in quanto i bit relativi ai caratteri vengono trasferiti in un colpo solo (cioè in parallelo). Una interfaccia seriale prevede invece una sola linea di trasmissione dei dati ed i singoli bit vengono passati uno alla volta (cioè in serie). Questa è la caratteristica principale che differenzia questi due tipi di interfaccia. Nel resto del paragrafo si supporrà di utilizzare una stampante connessa ad Amiga attraverso la porta parallela.

Per inviare caratteri di controllo è sufficiente aprire un file verso il device PAR:

```
OPEN "PAR:" FOR OUTPUT AS 1¶
```

AmigaBASIC potrebbe rifiutare l'apertura di tale file e presentare il messaggio "File Already Open" in quanto sia i dati per il device LPT1: che per il device PRT: vengono entrambi inviati attraverso la porta parallela. AmigaBASIC suppone che se una stampante è connessa alla porta parallela, non è allora possibile che a tale interfaccia sia connesso un secondo device, per cui PAR: risulta non disponibile.

La sola cosa che è possibile fare è ricaricare il Workbench, premendo contemporaneamente i tasti <Ctrl> <Amiga Sinistro> <Amiga Destro>. Quando si utilizza AmigaBASIC è necessario scegliere preventivamente la connessione desiderata.

- Dopo aver effettuato il caricamento del Workbench, inserire nuovamente il disco AmigaBASIC ed effettuare un click sull'icona AmigaBASIC.

- Inserire poi la linea di comando precedente.

A questo punto è possibile inviare dati alla stampante utilizzando, ad esempio, il comando seguente:

```
PRINT #1,CHR$(27);CHR$(69);"Stampa in grassetto"¶
CLOSE 1¶
```

Nel caso non venga effettuata la stampa in grassetto, significa che la stampante di cui si è in possesso non prevede tale opzione.

COM1:

Fino ad ora sono stati analizzati i device RAM:, PRT:, LPT1:, PAR:, SER:, SCRN: E KYBD:. Vi è tuttavia ancora un device da analizzare: COM1:, che viene utilizzato per attivare in modo standard la porta seriale RS-232. Dato che con ogni probabilità questo device non verrà quasi mai utilizzato, si è preferito racchiudere le informazioni relative ad esso nell'Appendice B (a riguardo è possibile consultare la voce OPEN).

Joystick

Non è ancora finita. Vi è un altro tipo di device periferico di cui non si è ancora fatta menzione: il Joystick. Un Joystick può muoversi in quattro diverse direzioni ed è provvisto di uno o più pulsanti per far fuoco. Un tale dispositivo, come è possibile intuire facilmente, si rivela più utile per i VideoGame che per i programmi professionali. E' possibile utilizzare un Joystick anche con AmigaBASIC, per cui, nel caso se ne possieda uno, può essere interessante provare gli esperimenti riportati di seguito.

Sul lato destro (o nella parte posteriore) Amiga presenta due connettori di nome JOY1 e JOY2. Al primo di questi due connettori (JOY1) dovrebbe essere già connesso il mouse.

- Procedere a collegare il Joystick al connettore JOY2 (lasciando collegato il mouse che dovrà essere utilizzato anche in seguito).

Muovendo il Joystick, oppure premendo il pulsante per far fuoco, non accade niente. AmigaBASIC non è in grado di gestire il nuovo dispositivo di input. E' possibile modificare questa situazione inserendo il programma seguente:

```
OPEN "Extras:BASICDemos/Ball" FOR INPUT AS 1¶
  OBJECT.SHAPE 1,INPUT$(LOF(1),1)¶
CLOSE 1¶
x=320 : y=100¶
OBJECT.ON 1¶
WHILE 1¶
  OBJECT.X 1,x¶
  OBJECT.Y 1,y¶
  x=x+STICK(2)¶
  y=y+STICK(3)¶
WEND¶
```

• Salvare il programma su disco e poi provare ad eseguirlo.

Utilizzo del Joystick

E' possibile utilizzare il Joystick per muovere oggetti grafici sullo schermo. Per realizzare questo, caricare il programma Ball dalla directory BASICDemos del dischetto Extras. E' possibile utilizzare, ad esempio, anche lo sprite realizzato nel programma di videotitolazione.

Il programma per la gestione del Joystick è abbastanza semplice da comprendere; le variabili x ed y contengono le posizioni sullo schermo dell'oggetto grafico. Tali variabili vengono inizializzate all'inizio del programma in modo da porre l'oggetto approsimativamente nel mezzo dello schermo. Per rendere visibile l'oggetto viene utilizzata una istruzione OBJECT.ON

STICK

Il ciclo infinito WHILE ... WEND viene utilizzato per effettuare gli spostamenti dei vari oggetti. E' possibile notare la presenza di un nuovo comando: il comando STICK(x), utilizzato per interpretare le informazioni provenienti dal Joystick.

Ad Amiga possono essere connessi fino a due Joystick, uno collegato al primo connettore (JOY1) ed uno al secondo (JOY2).

Ogni Joystick può essere spostato sia lungo le x che lungo le y. Il numero tra parentesi di seguito al comando STICK identifica quale deve essere la direzione in cui deve essere controllato lo spostamento.

STICK(0) controlla i movimenti lungo le X del Joystick numero 1.
STICK(1) controlla i movimenti lungo le Y del Joystick numero 1.
STICK(2) controlla i movimenti lungo le X del Joystick numero 2.
STICK(3) controlla i movimenti lungo le Y del Joystick numero 2.

STICK(x) è una funzione BASIC che ritorna uno dei valori esposti nella tabella seguente.

Valore	Significato
0	Nessun movimento nella direzione controllata
1	Movimento in altro oppure a destra
-1	Movimento in basso oppure a sinistra

Se il Joystick numero due viene mosso in alto, la funzione STICK(3) ritorna il valore 1. Nel programma precedente i valori ritornati vengono sommati ai valori delle coordinate X ed Y relative alla posizione dell'oggetto grafico in modo che questo possa essere spostato in accordo ai movimenti del Joystick. Effettuando un movimento contemporaneo lungo le X e lungo le Y, si produce un movimento diagonale. Il funzionamento dei comandi per lo spostamento dovrebbe essere facilmente comprensibile facendo riferimento alla modalità di azione del comando OBJECT illustrata precedentemente.

E' possibile a questo punto sperimentare un po' questo programma per comprendere come il Joystick gestisca i movimenti degli oggetti grafici.

Il pulsante per far fuoco (fire button)
E' necessario ora analizzare la modalità di utilizzo del pulsante per far fuoco. Il programma precedente potrebbe essere espanso inserendo, all'interno del ciclo WHILE ... WEND, la linea seguente:

```
IF STRIG(3) =-1 THEN BEEP¶
```

STRIG
D'ora in poi, tutte le volte che viene premuto il pulsante per far fuoco, Amiga prouce un suono. Il nome del comando rappresenta una abbreviazione per Stick Trigger. Naturalmente è necessario indicare al comando STRIG l'evento da esaminare. STRIG(1) ritorna il valore -1 se viene premuto il Fire Button del JOY1, mentre STRIG(3) ritorna il valore -1 quando viene premuto il pulsante di JOY2.

Rimangono da descrivere le funzioni STRIG(0) e STRIG(2) il cui funzionamento è simile a quello della funzione MOUSE(0). Esse ritornano infatti il valore -1 quando il pulsante è stato premuto dopo l'ultimo controllo effettuato. In questo modo è possibile controllare se il pulsante è stato premuto mentre il programma era impegnato nello svolgimento di altri compiti.

A questo punto si potrebbe pensare di saperne abbastanza per accingersi a realizzare da soli un Video-Game. Prima però è meglio sottolineare una caratteristica: se viene ralizzata una connessione per il Joystick sulla porta JOY1, AmigaBASIC si aspetta da quella connessione un segnale da Joystick e non è più in grado di riconoscere segnali dal mouse. Il puntatore del mouse rimane bloccato nella sua posizione e può essere spostato solamente utilizzando le combinazioni di tasti cursore e tasti Amiga.

Per comprendere come sia difficile lavorare senza il mouse è possibile provare, dopo avere salvato il proprio lavoro, ad inserire la linea seguente:

```
? STICK(0)¶
```

A questo punto il mouse diventa inutilizzabile. L'unico modo per risolvere il problema è quello di utilizzare il comando NEW; tuttavia talvolta anche questo comando non è sufficiente, nel qual caso è necessario ricaricare il Workbench.

3.6 Ottenere una stampa: una routine di stampa per il programmma di statistica

E' ora possible utilizzare tutte le conoscenze acquisite relativamente alla stampante per ampliare il programma di statistica. Si procederà infatti a realizzare ora una routine che consenta di stampare i file statistici in modo da ottenere una copia comparata dei dati su carta.

Per prima cosa è necessario caricare il programma di statistica. Non è necessario apportare numerose modifiche al suo interno. Per realizzare l'opzione di stampa è necessario introdurre, come prima cosa, una nuova opzione di menù. Per questo motivo è necessario modificare la routine Inizializzazione:

```
MENU 1,0,1,"Dati   "¶
MENU 1,1,1,"Recupera  "¶
MENU 1,2,1,"Registra  "¶
MENU 1,3,1,"Stampa  "¶
MENU 1,4,1,"Cancella"¶
MENU 1,5,1,"Fine  "¶
```

E' necessario introdurre anche una modifica nella routine ControlloMenu:

```
ControlloMenu:¶
  Men=MENU(0) : SceltaMenu=MENU(1)¶
  IF Men=1 THEN¶
    IF SceltaMenu=1 THEN GOSUB RecuperaDati¶
    IF SceltaMenu=2 THEN GOSUB RegistraDati¶
    IF SceltaMenu=3 THEN GOSUB StampaDati¶
    IF SceltaMenu=4 THEN GOSUB CancellaDati¶
    IF SceltaMenu=5 THEN FineProgramma¶
```

E' necessario ora scrivere la routine che realizza la stampa. E' possibile in-

serire tale routine dopo la routine ImmettiNome:

```
StampaDati:¶
  MENU 1,0,0 : MENU 2,0,0¶
  MENU OFF¶
  OPEN "PRT:" FOR OUTPUT AS 1¶
    PRINT #1,"File:";AltroNome$;CHR$(10)¶
    PRINT #1,"Numero";TAB(10);"Descrizione";
TAB(45);Valore"¶
    FOR x=4 TO LineaSup+4¶
      PRINT #1,Numero$(x);TAB(10);Descrizione$(x);
TAB(45);Valore"¶
    NEXT x¶
  CLOSE 1¶
  MENU 1,0,1 : MENU 2,0,1¶
  MENU ON¶
RETURN¶
```

Queta routine risulta abbastanza semplice da comprendere. Per prima cosa viene disabilitato l'event trapping; viene poi aperto un file di stampa, stampato il nome del file caricato (la variabile AltroNome$ contiene il nome dell'ultimo file utilizzato), stampata la linea titolo e successivamente le linee relative ai dati. Viene poi chiuso il file di stampa, vengono riabilitati i menù ed il controllo ritorna alla linea dove era stata precedentemente sospesa l'esecuzione prima della scelta dell'opzione PRINT.

E' possibile ottenere i dati stampati in differenti stili, a patto che la stampante li supporti..

Quando si implementano e si ampliano i vari programmi, è facile intravedere i grandi vantaggi forniti da AmigaBASIC; la sua costruzione modulare consente di inserire facilmente nuove parti di programma per arricchirne le caratteristiche supportate.

Hardcopy

Per stampare su carta il grafico a barre oppure quello a torta, è necessario utilizzare un programma per la realizzazione di hardcopy. Non è possibile affrontare il problema all'interno di questo libro, dato che le differenze

fra le varie stampanti, per quanto riguarda la gestione della grafica sono molteplici e non risulta possibile scrivere un programma che funzioni per tutte le stampanti esistenti. Inoltre AmigaBASIC si rivela troppo lento per eseguire i calcoli necessari per le stampe grafiche. Una routine normale di stampa in BASIC impiegherebbe circa una quarantina di minuti per stampare un'immagine.

Tuttavia è possibile realizzare un'hardcopy utilizzando il programma GraphicDump presente nel dischetto dell'ultima versione del Workbench. Per informazioni sull'utilizzo di tale programma si consiglia di consultare il manuale fornito con Amiga, oppure il libro Amiga! Primi Passi edito dalla F.T.E.

Nel capitolo successivo verranno illustrate alcune possibilità per il salvataggio dei propri lavori grafici. Con un paio di trucchetti risulterà poi possibile realizzare hardcopy grafiche.

4 Caricare e salvare la grafica

Si sarà notato che fino ad ora è stato omesso qualcosa di vitale nei programmi di disegno: non si è fatta menzione di come salvare la grafica.

Non è assolutamente divertente dedicare tantissimo tempo per realizzare un disegno destinato a scomparire nel momento stesso in cui viene spento il calcolatore. Si vedrà ora come salvare su dischetto la grafica e come poterla poi recuperare.

4.1 Il comando GET ed il comando PUT

Si comincerà con l'introduzione di due nuovi comandi BASIC: GET e PUT. Il metodo migliore per illustrarli è quello di analizzare un programma dimostrativo che ne faccia uso:

```
DIM Vettore%(563)¶
¶
CIRCLE (50,20),40¶
CIRCLE (35,12),5¶
CIRCLE (65,12),5¶
CIRCLE (50,20),8¶
CIRCLE (50,18),30,,3.5,5.8¶
PAINT (50,20),3,1¶
GET (0,0)-(99,39),Vettore%¶
¶
CLS¶
FOR x=0 TO 5¶
  FOR y=0 TO 4¶
    PUT (x*100,y*40),Vettore%¶
  NEXT y¶
NEXT x¶
```

Sullo schermo vengono visualizzate numerose piccole facce, disegnate con i comandi CIRCLE. Ma la parte veramente importante di questo programma è costituita dai due comandi appena citati.

GET

Il comando GET consente di salvare porzioni di disegno in un vettore. In questo esempio è utilizzato il vettore di interi di nome Vettore%. Il comando PUT consente di posizionare il segmento grafico salvato in un punto qualsiasi dello schermo. GET legge semplicemente i bit dai differenti insiemi di bit, li compatta in byte e memorizza tali byte nel vettore indicato.

PUT copia invece i byte dal vettore specificato nei bitplane.

La sintassi del comando GET prevede l'indicazione delle coordinate dell'angolo in alto a sinistra dell'area selezionata e dell'angolo in basso a destra. L'area di schermo deve essere rettangolare. Di seguito alla coordinate deve essere specificato il nome del vettore in cui devono essere inseriti i dati:

```
GET (xInizio,yInizio)-(xFine,yFine),Vettore¶
```

Prima di utilizzare il comando GET, è necessario dimensionare il vettore. A tale scopo si utilizza una semplice formula per calcolare quanti elementi sono necessari per una determinata area di schermo. Tra poco verrà fornita questa formula.

PUT

Il disegno salvato può essere visualizzato in qualsiasi punto dello schermo utilizzando il comando PUT in cui devono essere specificate le coordinate dell'angolo in alto a sinistra ed il nome del vettore da cui leggere i dati.

```
PUT (xInizio,yInizio),Vettore_dati¶
```

Occupazione di memoria dei dati grafici

La formula per determinare la quantità di memoria necessaria per una particolare area grafica risulta essere la seguente:

bitplane * altezza * 2 * INT((larghezza+16)/16) + 6

I valori per altezza e larghezza devono essere forniti in pixel. L'ultima parte della formula consente di calcolare il numero di byte richiesti per un segmento grafico di una data dimensione. Questo numero deve essere moltiplicato per il numero di bitplane, in quanto un punto colorato è composto da più bit, ognuno dei quali costituisce un bitplane. Infine, è necessario aggiungere 6 byte per le informazioni di controllo. Come risultato si ottiene l'occupazione richiesta in byte. Questo numero non è ancora identico al numero di elementi del vettore, in quanto un vettore con elementi interi utilizza due byte per ogni singolo elemento. Bisogna infatti ricordare che un intero può assumere un valore compreso tra -32768 e

32767. Amiga utilizza 16 bit (2 byte) per memorizzare un intero.

Calcolo della memoria richiesta
Nel caso dell'esempio esposto in precedenza, l'applicazione della formula presentata utilizza i seguenti valori: l'altezza dell'area da salvare è 40 pixel, la larghezza è 100 pixel. Si ottiene quindi:

```
(INT (100+16)/16) = INT (116/16) = 7
```

Moltiplicando per 2 si ottiene 7 * 2 = 14. Una linea di schermo larga 100 pixel richiede quindi 14 byte di memoria. Poichè 14 * 40 (altezza del disegno) = 560, si ottiene che il piccolo disegno costruito dal programma precedente richiede una occupazione di memoria di 560 byte.

Poichè lo schermo Workbench lavora con uno standard di due bitplane, è necessario moltiplicare 560 per due ottenendo in tal modo 1120 a cui vanno aggiunti i 6 byte necessari per le informazioni di controllo. In totale il disegno precedente richiede quindi 1126 byte di memoria.

Per dimensionare Vettore% in modo tale che possa contenere 1126 byte, è necessario utilizzare un numero di elementi pari a 563, valore ottenuto dall'operazione 1126/2.

Come è possibile vedere, la grafica richiede una grande quantità di memoria, molto più di quanto fosse richiesto dal programma di statistica precedentemente sviluppato per la memorizzazione dei file.

Ci si potrebbe chiedere in quali elementi del vettore siano contenute le informazioni di controllo: esse vengono inserite nei primi tre elementi. E' quindi possibile esaminarle utilizzando l'istruzione:

```
? Vettore%(0); Vettore%(1); Vettore%(2)¶
```

Il primo elemento del vettore contiene la larghezza dell'oggetto grafico, il secondo elemento contiene l'altezza mentre il terzo elemento contiene il numero di bitplane. Non è possibile modificare l'ordine di questi dati, in quanto AmigaBASIC legge i dati grafici interni in questa precisa sequenza.

PUT e gli operatori logici

E' possibile inserire un terzo parametro in un comando PUT, subito dopo il nome del vettore; questo parametro rappresenta il risultato di un confronto logico tra il grafico memorizzato nel vettore e lo sfondo dello schermo. Normalmente, AmigaBASIC effettua una operazione logica XOR tra il disegno e lo sfondo. Se non vi sono pixel accesi sullo sfondo, un qualsiasi pixel del disegno viene visualizzato normalmente. Se un pixel dello sfondo è già acceso, esso viene invertito. Si può provare a controllare direttamente cosa accade inserendo le seguenti linee:

```
CLS¶
LINE (100,120)-(199,139),1,bf¶
PUT (100,100),Vettore%¶
```

In questo modo si ottiene l'inversione di parte della faccia in cui è posizionato il blocco selezionato. Un effetto interessante si ottiene quando viene utilizzato nuovamente il comando PUT:

```
PUT (100,100),Vettore%¶
```

Divertente, vero? Si ottiene la cancellazione completa della faccia. L'operatore XOR consente, attraverso l'uso di un secondo comando PUT di rimuovere un disegno da un'area dello schermo senza disturbare lo sfondo. Questo trucco può essere utilizzato per ottenere effetti di animazione abbastanza buoni.

Si può provare qualcosa di diverso:

```
CLS¶
PUT (100,100),Vettore%,preset¶
```

PRESET

Utilizzando PRESET, si ottiene una comparazione tra il disegno e lo schermo che produce come risultato l'inversione dei parametri grafici definiti con la conseguente visualizzazione dei colori invertiti.

I quattro colori standard del Workbench vengono invertiti rispetto al loro stato normale: il blu viene trasformato in arancione; il bianco in nero; il nero in bianco e l'arancione in blu.

E' possibile vedere lo stesso effetto attivando un'icona nel Workbench. Anche l'etichetta di testo è visualizzata con colore opposto rispetto allo sfondo.

```
COLOR 0,1¶
CLS¶
PUT (100,100),Vettore%,pset¶
```

PSET

Il comando di confronto PSET consente di visualizzare l'immagine sullo schermo esattamente come è stata salvata, senza produrre alcuna alterazione dello sfondo.

```
CLS¶
LINE (100,120)-(199,139),1,bf¶
PUT (100,100),Vettore%,and¶
```

Questo esempio consente di combinare i punti dell'immagine con quelli dello sfondo utilizzando l'operatore relazionale AND. In questo modo, ovunque vi sia un punto già acceso, appare un punto del disegno.

Quanto vale per l'operatore AND, si applica anche all'operatore OR. Si provi ad eseguire nuovamente il programma, inserendo le linee seguenti in modo diretto:

```
CLS¶
LINE (100,120)-(199,139),1,bf¶
PUT (100,100),Vettore%,or¶
```

L'operatore OR copia il grafico sullo schermo senza invertire i punti dello sfondo. Si ottiene ora un divertente naso all'interno delle faccette.

Questi sono tutti gli operatori relazionali utilizzabili per effettuare confronti tra grafica e sfondo. Essi possono essere utilizzati per produrre effetti speciali.

Salvataggio di dati grafici: un primo metodo

Si è finora visto come copiare un'area grafica e posizionarla in un punto qualsiasi dello schermo. Tuttavia non si è ancora affrontato il problema

basilare, non è cioè ancora stato descritto un metodo per salvare i dati su dischetto. Una buona idea per la risoluzione di tale problema potrebbe essere quella di salvare gli elementi del vettore su dischetto.

• Provare, in proposito, il seguente programma dimostrativo:

```
DIM Vettore%(563)¶
¶
CIRCLE (50,20),40¶
CIRCLE (35,12),5¶
CIRCLE (65,12),5¶
CIRCLE (50,20),8¶
CIRCLE (50,18),30,,3.5,5.8¶
PAINT (50,20),3,1¶
GET (0,0)-(99,39),Vettore%¶
¶
OPEN "Facce" FOR OUTPUT AS 1¶
  FOR x=0 TO 563¶
     PRINT#1,MKI$(Vettore%(x));¶
  NEXT x¶
CLOSE 1¶
```

• Salvare questa versione del programma con un nome distintivo.

MKI$

In questo programma compare un nuovo comando: la funzione MKI$. Questa funzione verrà utilizzata abbastanza frequentemente. Essa infatti si rivela di grande aiuto per il salvataggio di numeri su dischetto. La regola essenziale è che quando un numero come -32768 viene salvato su dischetto occupa 7 byte: uno per il segno meno, cinque per le cifre ed infine uno per separarlo dal numero successivo. E' comunque anche possibile rappresentare il numero -32768 come un intero a 16 bit (cioè 2 byte). Sfruttando questo fatto è possibile utilizzare solamente un terzo di memoria, riducendo in tal modo i tempi di lettura e scrittura e risparmiando una grande quantità di spazio sul dischetto.

E' possibile utilizzare il comando MKI$ (MaKe Integer $tring) per poter convertire un intero a 16 bit in due byte. Tale funzione consente di crea-

re una stringa lunga due caratteri che può essere salvata su dischetto.

NOTA:
La funzione CHR$ consente un'operazione analoga su un numero a 8-bit, consentendo di convertire un numero minore di 255 in un carattere (es.: CHR$(255)).

ASC
Per avere nuovamente a disposizione il valore salvato utilizzando MKI$ oppure CHR$ è necessario utilizzare una funzione che converta i caratteri in un numero intero. La funzione inversa di CHR$ è la funzione ASC, che fornisce il codice ASCII del carattere.

• Provare, ad esempio, ad eseguire la linea seguente:

```
a$=CHR$(10): ? ASC(a$)¶
```

Se a$ risulta essere più lunga di un singolo carattere, ASC considera solamente il primo carattere della stringa.

La funzione ASC si rivela utile anche quando si desidera ottenere il codice ASCII di un dato carattere:

```
? ASC("H")¶
```

Si ottiene in questo modo il numero 72 che rappresenta il codice ASCII del carattere H maiuscolo.

CVI
La funzione inversa di MKI$ è chiamata CVI (ConVert to Integer). CVI consente di convertire una stringa in un numero intero:

```
a$=MKI$(32000): ? CVI(a$)¶
```

La stringa specificata deve essere lunga almeno due caratteri. Nel caso di stringhe più lunghe vengono convertiti solamente i primi due caratteri, mentre quelli restanti vengono ignorati.

E' necessario a questo punto effettuare un confronto tra due comandi che sembrano effettuare la stessa azione.

Il comando VAL consente di convertire un numero scritto come una stringa in un numero intero.

```
? VAL ("32768")¶
```

CVI combina due byte contenenti una stringa in un numero a 16 bit e ritorna il risultato:

```
a$=CHR$(0) + CHR$(254):? CVI(a$)¶
```

I due byte sono prodotti dai due comandi CHR$. Il risultato è 254, dato che i nove bit più significativi hanno tutti valore 0. Questo potrebbe sembrare a prima vista piuttosto complicato, ma approfondendo la conoscenza della programmazione BASIC, diventerà abbastanza semplice e comprensibile. Dopo aver pienamente compreso la differenza tra i vari comandi e funzioni appena descritti, si può procedere al caricamento dei dati grafici da dischetto. Si cercherà ora di risparmiare un pò di tempo nella fase di immissione dei programmi, caricando il programma originale di dimostrazione ed effettuando le modifiche viste precedentemente.

```
DIM Vettore%(563)¶
¶
OPEN "Facce" FOR INPUT AS 1¶
  FOR x=0 TO 563¶
    Vettore%(x)=CVI(INPUT$(2,1))¶
  NEXT x¶
CLOSE 1¶
¶
¶
CLS¶
FOR x=0 TO 5¶
  FOR y=0 TO 4¶
    PUT (x*100,y*40),Vettore%¶
  NEXT y¶
NEXT x¶
```

Controllando attentamente questo programma si nota che gli omini ven-

gono visualizzati senza utilizzare nessuna istruzione CIRCLE; essi vengono infatti caricati dal vettore memorizzato su dischetto.

INPUT$

Il comando CVI è già stato descritto, mentre è necessario fornire alcune delucidazioni relativamente al comando INPUT$. Questo comando è stato utilizzato abbastanza spesso all'interno del libro, ma sempre senza alcuna spiegazione. INPUT$ viene utilizzato per acquisire da un file su dischetto un numero fissato di caratteri. INPUT$(2,1) estrae una stringa di due caratteri da un file identificato come 1. La sintassi del comando è:

```
INPUT$ (n_caratteri, n_file)¶
```

E' necessario assicurarsi di non acquisire più caratteri di quanti siano effettivamente disponibili nel file. In caso contrario AmigaBASIC visualizza il messaggio di errore "Input Past End" per specificare che si è tentato di estrarre dal file più dati di quelli disponibili. Vi sono due metodi per risolvere questo problema.

LOF

La prima soluzione possibile è l'utilizzo della funzione LOF(x) (Length Of File, lunghezza del file). Il numero tra parentesi indica il numero di riferimento del file. Se il file è aperto, LOF restituisce la lunghezza del file in byte. E' possibile acquisire da un file un numero di caratteri non superiore a quanto indicato da LOF. Per acquisire un oggetto grafico, la sintassi dovrebbe essere la seguente:

```
INPUT$(LOF(1),1)¶
```

Tutti i caratteri contenuti nel file identificato con il numero 1 vengono acquisiti come una stringa unica. Se si desidera estrarre un solo carattere per volta è necessario procedere con un ciclo:

```
FOR x=1 TO LOF(1)¶
```

EOF

Il secondo metodo possibile comporta l'utilizzo della funzione EOF. Si è già visto qualcosa a proposito di questa funzione: essa restituisce il valore -1 quando viene raggiunta la fine del file. In caso contrario ritorna il

valore 0. Si può pensare di utilizzare un ciclo di questo tipo:

```
WHILE EOF(1)=0¶
```

oppure, in modo più elegante:

```
WHILE NOT EOF(1)¶
```

Non si dovrebbero incontrare altri problemi, visto che si è gia determinato il numero dei caratteri contenuti nel file.

Ulteriori informazioni sul comando INPUT$

INPUT$ può essere utilizzato in una modalità abbastanza interessante. Se non viene specificato nessun numero di riferimento del file, INPUT$ acquisisce i caratteri direttamente dalla tastiera.

• Provare in proposito ad inserire la linea seguente:

```
? INPUT$(10)¶
```

Dopo aver premuto il tasto <Return>, il cursore BASIC riappare sullo schermo. Premendo alcuni tasti, è possibile notare come i corrispondenti caratteri non vengano visualizzati, anche se Amiga li conserva ugualmente. Dopo aver inserito il decimo carattere, viene visualizzata sullo schermo tutta la sequenza battuta.

INPUT$ si rivela utile anche per effettuare il trasferimento di dati tra i dispositivi periferici e nel campo delle telecomunicazioni. Esso risulta quindi essere uno dei più importanti comandi di utilità AmigaBASIC per l'acquisizione dei dati.

Si è quindi appreso un metodo per salvare i dati grafici su dischetto. Questo metodo si rivela molto utile per memorizzare parti di schermo; tuttavia, quando si tenta di utilizzarlo per salvare interi schermi, si incontrano alcune limitazioni. Un disegno come quello realizzato dal programma di grafica precedentemente visto richiede infatti 300 x 200 pixel e 5 bitplane e richiede quindi, per la memorizzazione, un vettore di circa 21000 elementi. AmigaBASIC non è in grado di gestire un vettore di queste dimensioni e risulta quindi necessario caricare separatamente le varie parti dello schermo (utilizzando GET) e provvedere poi a salvarle.

Vi è comunque un metodo migliore per memorizzare dati grafici su dischetto.

4.2 Interchange File Format (IFF)

Acquistando ed utilizzando programmi quali Deluxe Video o Aegis Animator, è possibile notare come essi generino file che risultano compatibili tra loro. Ad esempio, un disegno realizzato con Graphicraft può essere modificato con Deluxe Paint e può essere utilizzato come sfondo o come elemento di primo piano in un programma di animazione.

Lo standard IFF e le sue conseguenze

Questo fatto non è casuale. Nel 1985, la casa produttrice di software Electronic Arts propose il concetto della memorizzazione dei dati secondo un formato standard universale, in modo tale che ogni programma potesse utilizzare qualunque file. Il risultato di questa geniale trovata prese il nome di IFF (Interchange File Format). Molti altri produttori software raccolsero questo suggerimento trovando molto interessanti i vantaggi offerti da tale formato standard: esso infatti migliorava l'immagine delle compagnie che vendevano software e che avevano deciso di adottarlo. Inoltre, l'usabilità di un programma veniva notevolmente aumentata dalla compatibilità esistente fra i diversi file: 100 $ per un programma che consente solamente di disegnare è un prezzo abbastanza alto. Ma 100 $ per un programma che offre la possibilità di disegnare, realizzare titoli video, animazioni e grafica di ogni genere, è senza dubbio un investimento più equo e completo.

Lo standard IFF rende possibile modificare le applicazioni sviluppate senza preoccuparsi della compatibilità. Senza IFF, i vecchi file risultavano essere inutilizzabili con programmi differenti da quelli con cui erano stati realizzati. I dati IFF sono virtualmente riconosciuti da qualsiasi programma; in caso contrario essi vengono semplicemente ignorati.

Molti possessori di altri computer non sono stati ugualmente forunati. Parecchi programmi di disegno per personal computer piuttosto diffusi e popolari non consentono l'interscambio dei file, cosicchè un file creato con il programma A non può essere utilizzato con il programma B. Il Com-

modore 64, ad esempio, è una vittima di questo tipo di incompatibilità.

Cosa altro consente di fare IFF

Lo standard IFF non è stato realizzato esclusivamente per la grafica. Questo formato consente infatti di salvare anche note musicali, codice, testo, fonti di caratteri, etc. anche se, in questo contesto, ci si concentrerà, per ora, esclusivamente sui dati grafici. Se si realizzano su Amiga programmi grafici in AmigaBASIC che rispettano la compatibilità IFF è poi possibile caricare le immagini da essi prodotte all'interno dei programmi commerciali. Analogamente risulta possibile recuperare dai programmi di disegno realizzati in AmigaBASIC immagini realizzate con programmi professionali. Ad esempio è possibile caricare uno dei propri disegni "fatti in casa" all'interno di un programma commerciale. Inoltre, risulta possibile scambiare file grafici con chiunque possieda un programma di disegno che utilizzi lo standard IFF.

Prima di immergersi nella programmazione, si darà uno sguardo agli aspetti fondamentali e funzionali dell'IFF.

Chunk e Form

Nello standard IFF tutti i dati che svolgono la stessa funzione sono raggruppati in un unico blocco. Questo gruppo di dati viene definito *Chunk*.

Un file IFF è composto da più chunk. Tutti i chunk dello stesso tipo sono radunati in un'unica struttura (*form*). Un programma Amiga verifica il tipo di ogni chunk controllando l'identificatore di ogni form, il quale contiene le informazioni riguardanti la lunghezza ed il tipo del file.

Bitmap

Un file grafico IFF è composto come minimo da tre chunk. Il primo contiene dati di controllo, ad esempio la larghezza e l'altezza del disegno, il numero di bitplane. Il secondo chunk contiene i valori RGB per i colori. Il terzo chunk contiene il disegno, o, più precisamente, in esso sono memorizzati i bit da cui è costituito il disegno. Un insieme di bit che costituisce un disegno è chiamato bitmap. Questo nome si accorda abbastanza bene con i bitplane, in quanto questi ultimi costituiscono un *bitmap*. Il bitmap può utilizzare più chunk come, ad esempio, per i dati che realizzano una

finta animazione con i colori (probabilmente si è avuto modo di vedere uno di questi giochi basati sul cambio ciclico di colore che forniscono l'illusione del movimento), oppure per le modalità operative del video, oppure per le coordinate associate al punto in cui un segmento grafico viene visualizzato sullo schermo.

Identificatori

Gli identificatori di chunk e di form sono costituiti da quattro byte. Questi quattro byte, che compongono una parola di codice riconosciuta dalla routine di lettura IFF, sono seguiti da altri quattro byte che mantengono le informazioni relative alla quantità di dati del disegno. La figura seguente mostra il formato di un tipico file grafico IFF:

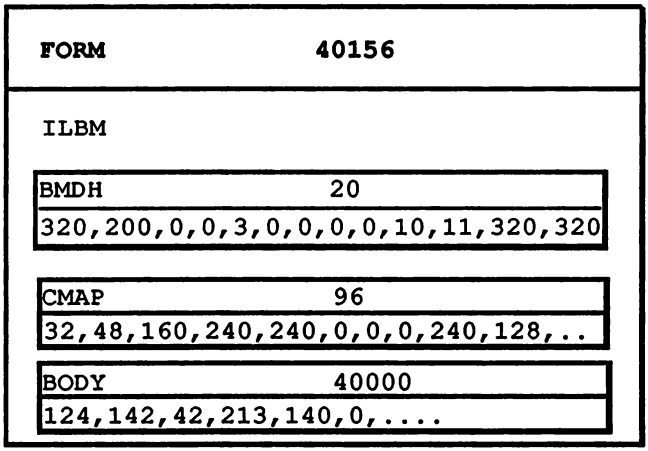


Figura 14: Formato IFF (standard ideato dalla Electronics Arts) per i file.

FORM e ILBM

I primi quattro caratteri del file contengono la parola FORM. I successivi quattro byte contengono la lunghezza della form (40156 byte) come numero a 32 bit. Il programma di acquisizione riconosce che è stato incontrato l'inizio di una form la quale è lunga in tutto 40156 byte. Di seguito vi è un

identificatore che specifica il tipo di file. Nell'esempio questo identificatore è ILBM, una abbreviazione per InterLeaved BitMap, che indica come il bitmap sia combinato con altri dati in una unica form. Il programma di acquisizione riconosce che i restanti dati di questa form sono all'interno di chunk ed hanno tutti un preciso ruolo nella composizione del disegno.

BMHD e CMAP

Nel primo chunk seguono queste informazioni: quattro byte che contengono l'identificatore BMHD (BitMap Header, contenente i dati di controllo); quattro byte che precisano la dimensione del chunk BMHD (20 byte in questo caso). Il programma di acquisizione può ora saltare alla routine di caricamento ed interpretazione dei dati di controllo. Proseguendo nell'analisi del formato IFF si trova l'identificatore CMAP (Color MAP, mappa dei colori) seguito dalla sua dimensione (96 byte). Questo chunk contiene i dati per i valori RGB dei vari colori utilizzati nella palette. Le tre tonalità red, green e blue usano fino ad un byte ciascuna, valore che, moltiplicato per 32 colori, fornisce un totale di 96 byte.

BODY

Dopo altri 96 byte si trova l'identificatore del chunk BODY, che risulta essere lungo 40.000 byte. Il chunk BODY costituisce il bitmap del disegno. I dati contenuti al suo interno seguono lo standard IFF: le linee che costituiscono i differenti bitplane sono accatastate sequenzialmente nel file (la prima linea del primo bitplane, seguita dalla prima linea del secondo bitplane, etc.). Dopo queste linee vi sono la seconda linea del primo bitplane, la seconda linea del secondo bitplane e così via.

Quando il programma di acquisizione è arrivato all'ultima linea dell'ultimo bitplane, sono stati letti un totale di 40156 byte. Il file IFF è stato completamente caricato ed interpretato.

Come si è visto il chunk BMHD viene per primo, seguito dal BODY e dal CMAP. BMHD deve essere il primo chunk presente nel file, in quanto i suoi dati risultano essenziali per determinare le dimensioni del bitmap contenuto nel chunk BODY. Per prima cosa il programma acquisisce i dati di controllo, successivamente carica i colori. Per i restanti chunk non esiste un ordine prefissato.

NOTA:

Un chunk deve essere composto da un numero pari di byte. Se, ad esempio, il chunk CMAP contiene solamente i valori per 7 registri colore (21 byte), deve essere inserito un byte di riempimento per incrementare da 21 a 22 il numero di byte, producendo quindi un riallineamento. Questo byte di riempimento è solitamente posto a 0. Il byte per il reallineamento non viene considerato come informazione di controllo; esso è inserito solamente per effettuare il necessario riempimento. Ogni numero dispari deve quindi essere incrementato di una unità. In figura 15 viene mostrata la funzione del byte di riempimento.

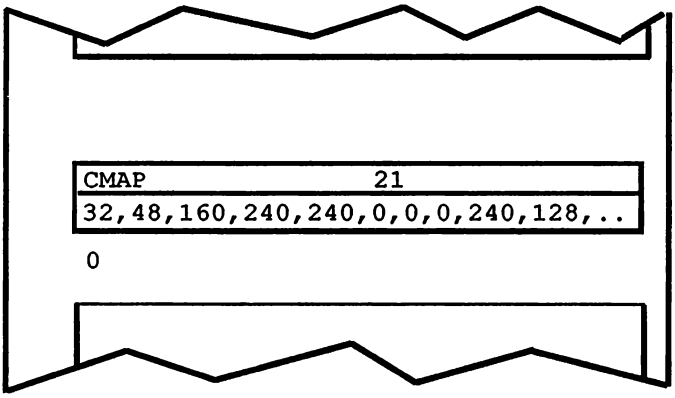


Figura 15: A tutti i chunk composti da un numero dispari di byte viene accodato un byte di riempimento.

4.3 La routine di acquisizione IFF

Si vedrà ora come mettere in pratica quanto illustrato teoricamente nel paragrafo precedente, in modo da poter interpretare i dati IFF.

Prima di iniziare è necessario segnalare una cosa: recentemente la Commodore ha rilasciato una versione del dischetto Extras che contiene, nel cassetto Demos, tre programmi addizionali. Questi prgrammi consentono di leggere e salvare dati grafici in formato IFF. Queste routines sono descritte nell'appendice D insieme all'analisi delle librerie. Esse risultano essere leggermente più veloci di quelle contenute in questo capitolo. Tuttavia, i programmi sul dischetto Extras risultano essere considerabilmente più difficili da comprendere e da utilizzare. Per questa ragione viene qui incluso un programma AmigaBASIC per la gestione dei file IFF, completamente commentato e che non utilizza nessuna libreria. Ma prima è necessario esaminare tutti i comandi AmigaBASIC.

I successivi programmi non si pongono come obiettivo quello di spiegare ogni cosa riguardo alla gestione dell'input e dell'output da disco in AmigaBASIC. Lo scopo è qui quello di imparare le tecniche di gestione dei file IFF. In questo paragrafo si troverà ogni spiegazione riguardo ai programmi LoadACBM, LoadILBM, SaveACBM, e SaveILBM riportati nell'appendice D.

```
INPUT "Inserire il nome del file";Nomedue$¶
OPEN Nomedue$ FOR INPUT AS 1¶
    Form$=INPUT$(4,1))¶
    Lunghezza=CVL(INPUT$(4,1))¶
    IF INPUT(4,1)<>"ILBM" THEN
       PRINT "Errore nei dati"¶
       END¶
    ENDIF¶
LetturaDati:¶
    IF EOF(1) THEN END¶
       Chunk$=INPUT$(4,1)¶
```

```
        Lunghezza=CVL(INPUT$(4,1))¶
        IF INT(Lunghezza/2)<>(Lunghezza/2) THEN
           Lunghezza=Lunghezza + 1¶
        IF Chunk$="BMHD" THEN BMHeader¶
        IF Chunk$="CMAP" THEN ColorMap¶
        IF Chunk$="BODY" THEN BodyMap¶
        Fasullo$=INPUT$(Lunghezza,1)¶
  GOTO LetturaDati¶
¶
BMHeader:¶
     xd=CVI(INPUT$(2,1))¶
     yd=CVI(INPUT$(2,1))¶
     Fasullo$=INPUT(4,1)¶
     Bitplane=ASC(INPUT$(1,1)¶
     Fasullo$=INPUT$(11,1)¶
     IF xd=320 THEN Tipo=1¶
     IF xd=640 THEN Tipo=2¶
     IF yd>256 THEN Tipo=Tipo+2¶
     SCREEN 1,xd,yd,Bitplane,Tipo¶
     WINDOW 2,NomeDue$,,0,1¶
     Locazione=PEEKL(WINDOW(8)+4)+8¶
     FOR x=0 TO Bitplane -1
           PianoLocazione=PEEKL(Locazione+4*x)¶
     NEXT x¶
     Fasullo$=INPUT$(Lunghezza-20,1)¶
  GOTO LetturaDati¶
¶
ColorMap¶
    FOR x=0 TO Lunghezza/3¶
      r=(ASC(INPUT$(1,1) AND 240)/16¶
      g=(ASC(INPUT$(1,1) AND 240)/16¶
      b=(ASC(INPUT$(1,1) AND 240)/16¶
      PALETTE (x-1),r/16,g/16,b/16¶
   NEXT x¶
       IF INT(Lunghezza/3)<>(Lunghezza/3) THEN
         Fasullo$=INPUT$(1,1)¶
```

```
    GOTO LetturaDati¶
    ¶
BodyMap:
    LineaByte=xd/8¶
    FOR y1=0 TO yd-1¶
       FOR b=0 TO BitPlane-1¶
         FOR x1=0 TO LineaByte/4-1¶
             POKEL (PianoLocazione(b)+4*x1+40*y1,
                    CVL(INPUT$(4,1))¶
         NEXT x1¶
      NEXT b¶
   NEXT y1¶
  GOTO LetturaDati¶
```

Questo programma è in grado di caricare grafica IFF in AmigaBASIC. E' necessario salvare questo programma non appena si è finito di inserire il codice e prima di eseguirlo, in quanto vi sono alcuni punti critici dove un errore di battitura può generare, in fase di esecuzione, un crash di sistema, cosa che, se non si era provveduto a salvare il programma, porta alla sua inevitabile perdita.

Utilizzare il programma

- Dopo aver lanciato il programma, battere il nome del file grafico da cui deve essere acquisito il disegno ed inserire il dischetto relativo.

Nel caso non si possieda un dischetto contenente file grafici utilizzare il dischetto Extras e digitare, come nome di file:

```
Extras1.2:BasicDemos/Heart.ILBM¶
```

Il programma determina la risoluzione dello schermo, seleziona i colori (si noti che il colore del puntatore risulta modificato) e visualizza il disegno linea dopo linea. Si osservi cosa succede ai bordi della finestra: essi vengono sovrascritti dai dati grafici. Quando viene premuto per la prima volta il pulsante sinistro del muose, i bordi ricompaiono. Si noti anche che non è possibile accedere ai menù a tendina mentre è in fase di caricamento la parte superiore del disegno. Accade questo per evitare che vi sia sovrapposizione tra i dati grafici in fase di caricamento e le opzioni dei menù.

Modifiche al disegno
Si ha ora una finestra BASIC in cui è visualizzato un disegno IFF. Vi sono a questo punto numerose possibilità: si può sovrappore del testo al disegno, oppure copiare parti del disegno utilizzando i comandi GET e PUT, oppure muovere oggetti grafici con il disegno come sfondo. Ma, prima di iniziare questa serie di esperimenti, è necessario avere maggiori informazioni riguardo al programma.

INPUT consente di acquisire il nome del file da caricare. Il primo INPUT$(4,1) esamina l'identificatore FORM, ed assegna la lunghezza della form alla variabile Lunghezza.

CVL
Il comando CVL (ConVert Long) nella quarta linea non è un errore tipografico. Analogamente a come il comando CVI converte 2 byte in un intero a 16 bit, CVL converte 4 byte in un numero a 32 bit.

MKL$
Il comando di scrittura corrispondente a CVL è il comando MKL$. Pur non essendo chiaro il motivo di tale scelta, lo standard IFF stabilisce come lunghezza massima per una form 4.294.967.295 byte e quindi il valore della lunghezza di una form deve essere contenuto in una variabile a 32 bit. In altre parole, possono essere riempiti con dati grafici fino a 32 Gigabyte.

I quattro byte successivi del file contengono l'identificatore ILBM. In caso contrario, poichè non si sta utilizzando un file ILBM, il programma termina.

Nelle variabili Chunk$ e Lunghezza vengono posti il valore dell'identificatore di chunk e la lunghezza del chunk stesso.

Ricerca di eventuali Chunk dispari
La linea di programma:

```
IF INT(Lunghezza/2) <> (Lunghezza/2)¶
```

controlla che la lunghezza sia un numero pari. Lunghezza viene divisa per

2, e successivamente viene rimosso ogni valore decimale dal risultato di questa operazione. Se i valori delle due operazioni risultano uguali, si può concludere che il numero è divisibile per due. Il carattere <> identifica l'operatore di disuguaglianza "Diverso da". I numeri dispari vengono poi incrementati di uno per ottenere un numero pari.

Memorizzare i resti
Il programma, in funzione del contenuto di Chunk$, effettua il richiamo di una particolare subroutine. Se il chunk non è riconoscibile, l'intero contenuto viene posto in Fasullo$, un nomignolo per un identificatore. Fasullo$ contiene i dati che il programma non è in grado di interpretare o comprendere.

Il ciclo LetturaDati: viene ripetuto fino a che non vi sono più dati nel file.

La routine BMHeader: carica i dati dal chunk avente lo stesso nome. Questo chunk è solitamente lungo 20 byte: i primi due byte contengono numeri a 16 bit specificanti la larghezza del disegno, mentre i due byte successivi specificano l'altezza. Questi valori vengono assegnati alle variabili xd e yd.

I quattro byte successivi vengono inseriti in Fasullo$. Nello standard IFF, questi forniscono le coordinate x ed y dell'angolo in alto a sinistra dei grafici che non utilizzano l'intero schermo. Poichè in questo momento si sta lavorando con disegni a tutto schermo, questi quattro byte non vengono presi in considerazione.

Il byte seguente contiene il numero di bitplane ed il suo valore viene assegnato alla variabile Bitplane.

I successivi 11 byte mantengono i dati utilizzati dalla routine per il salvataggio in formato IFF e quindi, per il momento, non vengono considerati.

xd può assumere un valore di 320 oppure 640, mentre yd può assumere un valore di 256 o 512 (512 nel caso del modo interlacciato). Il valore assegnato a Tipo rappresenta il valore di controllo della risoluzione.

Dopo aver creato lo schermo, viene visualizzata una finestra in cui com-

pare il nome del file che deve essere utilizzato.

Gestione della memoria

Vi è ora la trattazione di una parte un pò più complicata. Come ogni altro computer, anche Amiga controlla la sua memoria. Nessun bitmap o nessun dato può essere posizionato in aree di memoria utilizzate da altri programmi (ad esempio dal Workbench o dall'AmigaBASIC). La caratteristica di multitasking dell'Amiga rende complessa la suddivisione della memoria. In altre parole non è possibile stabilire in quale zona debba essere caricato l'AmigaBASIC in quanto è Amiga che si fa carico di queste decisioni, ponendo i vari programmi in differenti aree di memoria.

Ogni cella di memoria può contenere 1 byte; la memoria RAM di Amiga contiene almeno 262.000 o 524.000 celle che possono essere utilizzate in lettura e scrittura. In aggiunta alla memoria leggibile e scrivibile (RAM), Amiga 500 ed Amiga 2000 possiedono 256 k di *ROM* (Read Only Memory) stabile, dove risiedono i dati del Kickstart. Amiga 1000 utilizza invece 256 k di RAM in cui viene caricato il contenuto del dischetto Kickstart; dopo il caricamento tale porzione di memoria viene protetta elettronicamente in modo tale da farla sembrare memoria ROM.

Ogni cella di memoria è identificata da un numero chiamato *indirizzo* . Il primo indirizzo di memoria assoluto in Amiga è identificato dal numero 0, il secondo dal numero 1, e così via.

Intuition

Il sistema operativo di Amiga contiene un programma che controlla quali porzioni di memoria sono riservate ad un particolare programma. Questo programma per la gestione della memoria è chiamato Intuition. Intuition stesso è una parte integrante del sistema operativo. Esso conosce, ad esempio, che la parte di memoria che va dall'indirizzo 0 all'indirizzo 40.000 deve essere riservata per il Workbench, dall'indirizzo 40.001 all'indirizzo 72.000 è utilizzata dallo schermo, dall'indirizzo 72.001 all'indirizzo 160.000 viene dedicata ad AmigaBASIC, etc. (ovviamente questi non sono i numeri reali, ma sono stati utilizzati solamente per fornire un esempio).

PEEK, POKE

Normalmente, non è necessario cambiare direttamente il contenuto di una particolare locazione di memoria, in quanto molte modifiche possono produrre un conflitto tra i dati. AmigaBASIC, tuttavia, mette a disposizione una serie di comandi che permettono questo tipo di azione: il comando PEEK consente di esaminare il contenuto di una locazione di memoria, mentre il comando POKE consente di scrivere un valore in una data locazione di memoria. Questi comandi risultano essere estremamente importanti per molti computer, in quanto essi permettono all'utente di utilizzare routine sostitutive di eventuali comandi BASIC mancanti. Ad esempio, per il Commodore 64, i comandi PEEK e POKE assumevano una tale importanza e si rivelavano di tale utilità da giustificare la stesura di libri su di essi. AmigaBASIC utilizza PEEK e POKE molto meno, in quanto in tale linguaggio è praticamente disponibile ogni comando. AmigaBASIC non fornisce tuttavia routine dirette per il salvataggio od il caricamento di dati grafici, per cui è necessario utilizzare PEEK e POKE per realizzare tali routine.

• Battere la seguente linea di programma nella finestra BASIC:

```
? PEEK(30000)¶
```

Si ottiene in risposta un numero compreso tra 0 e 255. Tale numero corrisponde al valore memorizzato nella cella che si trova all'indirizzo 30000. In pratica è stato visualizzato il numero ad 8 bit dell'indirizzo 30000.

```
POKE 30000, (Valore)¶
```

Questa linea scrive invece un valore all'indirizzo di memoria 30000 che potrebbe poi essere letto con il comando PEEK. In questo modo si è inviato un valore all'indirizzo 30000 per comunicare il nuovo numero da conservare.

PEEKW, POKEW

Se si desidera scrivere un numero di 16 bit invece di uno a 8 bit, è necessario utilizzare i comandi PEEKW e POKEW. La lettera W significa Word (parola), espressione utilizzata per indicare 16 bit.

I comandi PEEKW e POKEW lavorano con due byte che risiedono uno di seguito all'altro nella memoria. I residenti di entrambi gli indirizzi gestiscono complessivamente un totale di 16 bit. Ognuno contiene 8 bit.

• Battere la seguente linea:

```
POKEW 30001, (Valore)¶
```

Compare il messaggio di errore "Illegal function call".

PEEKL, POKEL

L'intero gioco può essere applicato anche ai numeri a 32 bit. I comandi da utilizzare in questo caso sono PEEKL e POKEL (L = Longword, parola lunga). Gli indirizzi devono essere anche qui numeri pari. Quattro locazioni di memoria residenti una di seguito all'altra forniscono quattro valori, ognuno di 8 bit, per un totale di 32 bit.

WINDOW(x)

Il comando WINDOW(0), già trattato in precedenza, ritorna il numero di identificazione della finestra attualmente attiva. Ma vi sono altre modalità di funzionamento di tale comando con differenti funzionalità. WINDOW(8) ritorna l'indirizzo di una lista (i programmatori C ed i progettisti di computer la definiscono *structure*), in cui Intuition annota i dati relativi alla finestra corrente. Il secondo insieme di quattro byte della lista, indicata da WINDOW(8), contiene l'indirizzo di inizio della lista successiva. Vi sono parecchi valori a 32 bit in questo ottavo bit della lista. Questi sono gli indirizzi di partenza di ogni singolo bitplane. Se non si conosce abbastanza bene la disposizione della memoria di Amiga è difficile comprendere completamente quanto viene spiegato in questa parte, anzi è possibile che non si comprenda nulla di tutto questo. Tuttavia, analizzando il programma, si dovrebbe comprendere come il valore dell'indirizzo venga assegnato alla variabile PianoLocazione(x) che riporta l'indirizzo a cui ogni eventuale bitplane della finestra grafica è posizionato in memoria. Successivamente verranno fornite ulteriori informazioni su quest'ultima variabile.

Da ultimo, Fasullo$ conserva i byte in eccesso.

Interpretazione del chunk CMAP

Meno complicata risulta l'interpretazione del chunk CMAP. Vengono riservati tre byte per i vari colori: uno per la componente di rosso, uno per la componente blu, uno per la componente di verde. Si è già detto che le routine standard IFF consentono questa operazione. Lo stesso vale anche

per quanto riguarda la risoluzione. Amiga può riconoscere 16 livelli di rosso, verde e blu. IFF è in grado di fornire 256 sfumature di colore.

Per utilizzare disegni gestiti da calcolatori con capacità grafiche superiori a quelle di Amiga, gli sviluppatori dell'IFF hanno pensato ad un elegante trucco: per memorizzare le intensità dei colori da 0 a 15 vengono utilizzati i quattro byte più significativi. Questo significa che, per ogni passo di colore, Amiga ha la possibilità di utilizzare 15 numeri per raffinamenti nella risoluzione dei colori. Le percentuali di rosso, verde e blu, conserveranno le giuste proporzioni.

Utilizzando AND 240 e dividendo per 16, sono stati isolati i quattro bit più significativi . In tal modo si ottiene il valore corretto per il comando PALETTE. Ogni byte di riempimento, precedente o seguente, viene posto in Fasullo$.

Interpretazione del chunk BODY

L'ultima parte del programma intepreta il chunk BODY, cioè il bitmap. La variabile LineaByte determina il numero di byte per linea di schermo, utilizzando il valore di xd. L'ultima parte specifica che le linee vengono memorizzate secondo la sequenza di bitplane. L'acquisizione dei dati verrà quindi effettuata nello stesso identico modo. E' ora possibile utilizzare PianoLocazione per individuare l'indirizzo di partenza di ogni singolo bitplane, e porlo nella giusta posizione in memoria linea per linea. In questo caso viene utilizzato il comando POKEL in quanto si desiderano ottenere i risultati più rapidi: ogni esecuzione del ciclo trasferisce in memoria 32 bit.

La variabile y1 memorizza il numero delle linee del disegno, b i bitplane, x1 il totale di numeri a 32 bit per linea. Dopo aver trasferito in memoria tutti i dati del bitmap, il sistema ritorna alla routine LetturaDati:.

Se vi sono altri chunk all'interno del file, essi verranno interpretati ed utilizzati.

Forma compressa

Vi è ancora un problema da risolvere che potrebbe essere stato incontrato in questo capitolo, specialmente se si è caricato un file da DeluxePaint.

Questo pacchetto e molti altri pacchetti di grafica presentano uno svantaggio: il bitmap contenuto nel chunk BODY è salvato in *forma compressa.* Che cosa significa questo?

Abbastanza spesso alcune aree dello schermo sono riempite con il colore dello sfondo e quindi molti altri dettagli dello schermo non risultano visibili. Non ha molto senso salvare lo stesso valore 2000 volte sul dischetto. Per risparmiare spazio, i programmatori hanno sviluppato un sistema mediante il quale un insieme di byte identici può essere compattato. Un disegno che occupa 40 k di RAM può essere compresso in questo modo in modo che esso occupi solamente circa 25 k sul dischetto. Le immagini così compresse creano però dei problemi qualora si tenti di caricarle in AmigaBASIC. Una routine di compensazione risulta abbastanza complicata da scrivere e piuttosto lenta in esecuzione. Questo significa che un disegno salvato da DeluxePaint non deve essere letto direttamente in memoria, altrimenti esso risulterà completamente diverso dall'originale. Se si ha un po' di tempo si può provare a verificare questa affermazione.

I disegni in bassa risoluzione (320 x 256) non presentano un effetto così disastroso, se si utilizza Graphicraft. Graphicraft, infatti, non compatta i dati durante il loro salvataggio. E' possibile in tal modo caricare in Graphicraft un'immagine compattata e poi salvarla. L'immagine potrà così essere letta in AmigaBASIC. Tuttavia Graphicraft non è in grado di lavorare con immagini in altre risoluzioni, ad esempio 640 x 512. Tutti i programmi grafici attualmente disponibili che gestiscono l'alta risoluzione salvano le immagini in forma compattata, forma che non può venire gestita da AmigaBASIC.

Un'ultima considerazione: lo standard IFF consente di trasformare in qualunque tipo di risoluzione le immagini realizzate in BASIC. Maggiori informazioni verranno fornite in seguito.

4.4 Passare la mano: caricare e salvare immagini realizzate con programmi di disegno

Ritornando alla questione principale si deve ancora vedere come caricare immagini realizzate con Deluxe Paint nel programma di disegno realizzato in precedenza, oppure immagini realizzate con questo programma all'interno di Graphicraft, etc.

Dopo il lavoro di base analizzato nei paragrafi precedenti, non dovrebbe essere difficile scrivere una routine che consenta la gestione dei file IFF.

- Salvare il programma realizzato nel paragrafo precedente, se non si è ancora provveduto a farlo, e caricare poi il programma di disegno.

Ogni passo necessario per arrivare al risultato voluto sarà guidato ed adeguatamente analizzato. Per prima cosa, è necessario inserire le linee seguenti all'interno della routine Progetto: del programma.

```
IF SceltaMenu=4 THEN GOSUB OKColori: GOSUB OKModo:
      GOSUB CaricaDisegno¶
IF  SceltaMenu=5 THEN GOSUB OKColori: GOSUB OKModo:
      GOSUB RegistraDisegno¶
```

Scorrere ora il listato del programma fino alla routine OkModo: dopo la quale inserire quanto segue:

```
CaricaDisegno:¶
  MENU 2,0,0: MENU 1,0,0¶
  MENU OFF: MOUSE OFF¶
  GOSUB ImmettiNome¶
  WINDOW CLOSE 5¶
  WINDOW 2¶
  IF Nome$="" THEN FineCarica¶
```

```
  OPEN Nome$ FOR INPUT AS 1¶
  Form$=INPUT(4,1)¶
  Lunghezza$=CVL(INPUT$(4,1))¶
  IF INPUT$(4,1)<>"ILBM" THEN BEEP: GOTO FineCarica¶
¶
LetturaDati:
  IF EOF(1) THEN FineCarica¶
  Chunk$=INPUT$(4,1)¶
  Lunghezza$=CVL(INPUT$(4,1))¶
  IF INT(Lunghezza/2)<>(Lunghezza/2) THEN
       Lunghezza=Lunghezza+1¶
  IF Chunk$="BMHD" THEN BMHeader¶
  IF Chunk$="CMAP" THEN ColorMap¶
  IF Chunk$="BODY" THEN BodyMap¶
  Fasullo$=INPUT$(Lunghezza,1)¶
GOTO LetturaDati¶
¶
BMHeader:  ¶
    xd=CVI(INPUT$(2,1))¶
    IF xd>320 THEN FineCarica¶
    yd=CVI(INPUT$(2,1))¶
    IF yd>256 THEN FineCarica¶
    Fasullo$=INPUT$(4,1)¶
    BitPlane=ASC(INPUT$(1,1))¶
    Fasullo$=INPUT$(11,1)¶
    Locazione=PEEKL(WINDOW(8)+4)+8¶
    FOR x=0 TO BitPlane-1¶
      LocazionePiano(x)=PEEKL(Locazione+4*x)¶
    NEXT x¶
  GOTO LetturaDati¶
¶
ColorMap:¶
  FOR x=0 TO (Lunghezza/3)-1¶
  r=(ASC(INPUT$(1,1)) AND 240)/16¶
  g=(ASC(INPUT$(1,1)) AND 240)/16¶
  b=(ASC(INPUT$(1,1)) AND 240)/16¶
```

```
    PALETTE x,r/16,g/16,b/16¶
    Colori%(x,0)=r : Colori%(x,1)=g : Colori%(x,2)=b¶
  NEXT x¶
  IF INT(Lunghezza/3)<>(Lunghezza/3) THEN
      Fasullo$=INPUT$(1,1)¶
  GOTO LetturaDati¶
¶
BodyMap:¶
    FOR y1=0 TO yd-1¶
      FOR b=0 TO BitPlane-1¶
        IF b<Colori THEN¶
          FOR x1=0 TO 9¶
           POKEL LocazionePiano(b)+4*x1+40*y1,
                 CVL(INPUT$(4,1))¶
          NEXT x1¶
        ELSE¶
          Fasullo$=INPUT$(40,1)¶
        END IF¶
      NEXT b¶
    NEXT y1¶
  GOTO LetturaDati      ¶
¶
FineCarica:¶
  CLOSE 1¶
  MENU ON : MOUSE ON¶
  MENU 1,0,1 : MENU 2,0,1¶
  RETURN¶
¶
RegistraDisegno:¶
  MENU 2,0,0 : MENU 1,0,0¶
  MENU OFF : MOUSE OFF¶
  GOSUB ImmettiNome¶
  WINDOW CLOSE 5¶
  WINDOW 2¶
  IF Nome$="" THEN FineRegistra¶
  OPEN Nome$ FOR OUTPUT AS 1 LEN=FRE(0)-500¶
```

```
      PRINT #1,"FORM";¶
      PRINT #1,MKL$(156+8000*Colori);¶
      PRINT #1,"ILBM";¶
      PRINT #1,"BMHD";MKL$(20);¶
      PRINT #1,MKI$(320);MKI$(256);¶
      PRINT #1,MKL$(0);¶
      PRINT #1,CHR$(Colori);¶
      PRINT #1,CHR$(0);MKI$(0);MKI$(0);¶
      PRINT #1,CHR$(10);CHR$(11);¶
      PRINT #1,MKI$(320);MKI$(256);¶
¶
      PRINT #1,"CMAP";MKL$(96); ¶
      FOR x=0 TO 31¶
        PRINT #1,CHR$(Colori%(x,0)*16);¶
        PRINT #1,CHR$(Colori%(x,1)*16);¶
        PRINT #1,CHR$(Colori%(x,2)*16);¶
      NEXT x¶
      ¶
      PRINT #1,"BODY";MKL$(8000*Colori);¶
      Locazione=PEEKL(WINDOW(8)+4)+8¶
      FOR x=0 TO Colori-1¶
        LocazionePiano(x)=PEEKL(Locazione+4*x)¶
      NEXT x¶
      FOR y1=0 TO 255¶
        FOR b=0 TO Colori-1¶
          FOR x1=0 TO 9 ¶
                  PRINT#1,MKL$(PEEKL(LocazionePiano
                    (b)+4*x1+40*y1));¶
          NEXT x1¶
        NEXT b¶
        PLocazione=LocazionePiano(0)+40*y1¶
        POKE PLocazione,PEEK(PLocazione) AND 63¶
        POKE PLocazione+39,PEEK(PLocazione+39) AND 252¶
      NEXT y1¶
      ¶
      PRINT #1,"CAMG";MKL$(4);¶
```

```
    PRINT #1,MKL$(16384);¶
  CLOSE 1¶
¶
FineRegistra:¶
  MENU ON : MOUSE ON¶
  MENU 1,0,1 : MENU 2,0,1¶
RETURN¶
```

Funzionamento delle routine
Si può facilmente notare come queste routine differiscano di poco da quelle utilizzate nel programma di disegno. La routine BMHeader:, che effettua la lettura del chunk BHMD, controlla le dimensioni del disegno: se esso risulta più largo di 320 pixel o più alto di 256 pixel, la routine di caricamento termina in quanto il grafico non può essere visualizzato sullo schermo.

I valori RGB, acquisiti dalla routine ColorMap:, vengono assegnati agli elementi del vettore Colori%, in modo che il programma possa utilizzare normalmente i colori.

Il ciclo all'interno della routine BodyMap: inizializza i valori per un disegno di 320 x 256 pixel; il programma non funziona con disegni in formati differenti.

Adattamento dei Bitplane
E' ora necessario adattare il numero dei bitplane. Quando il disegno che viene caricato utilizza uno schermo con un numero di bitplane superiore a quello utilizzato nel programma di disegno, i dati verranno impacchettati ed inseriti nel vettore Fasullo$. Il programma carica solamente il numero di bitplane stabilito dalla variabile Colori. Conseguentemente, possono essere specificati differenti colori in ogni istante.

File Buffer
All'interno della routine RegistraDisegno vi sono altre nuove istruzioni che consentono di memorizzare sul dischetto il disegno in formato IFF. Questa porzione di programma comincia allo stesso modo della routine

CaricaDisegno: i menù a tendina vengono disabilitati e viene richiesto il nome del file in cui salvare il disegno. Tale file viene poi aperto in scrittura. Che significato ha la linea LEN=FRE(0) - 500 ? Come noto, AmigaBASIC crea un buffer per ogni file, in cui i dati vengono radunati, prima di essere effettivamente scritti su dischetto. Questo consente di risparmiare una grande quantità di tempo, in quanto il salvataggio su dischetto risulta essere la più lenta delle operazioni svolte dal computer. Normalmente tale buffer è di 128 byte. Più grande è il buffer, più veloce risulta il trasferimento dei dati. La dimensione del buffer può essere specificata nel comando OPEN accodando LEN=(Dimensione Buffer) al comando stesso OPEN. Ad esempio, l'istruzione:

```
OPEN Nam$ FOR OUTPUT AS 1 LEN=1000¶
```

assegna al file 1 un buffer di 1000 byte.

Lo svantaggio di questo metodo è che non si dispone di una quantità di memoria illimitata, in quanto l'area di memoria in cui vengono allocate le variabili, le matrici ed i file buffer, è di soli 25 k in un Amiga con 512 k di RAM. Il messaggio visualizzato all'atto del caricamento di AmigaBASIC fornisce informazioni sulla quantità di memoria disponibile.

FRE(0)
AmigaBASIC possiede anche una funzione che consente di determinare la quantità di memoria libera: FRE(X).

• Inserire la linea seguente nella finestra BASIC:

```
? FRE(0)¶
```

FRE(-1)
Il numero che viene visualizzato riporta la quantità di byte liberi nella memoria BASIC. Per un Amiga con 512 k di RAM e con il programma di disegno caricato, tale quantità è di circa 9000 byte. Se FRE è seguito dal valore zero racchiuso tra parentesi, vengono fornite informazioni sulla memoria disponibile in BASIC. Sostituendo tale valore con -1 si ottengono informazioni sulla intera memoria di sistema a disposizione.

```
? FRE(-1)¶
```

Informazioni sulla memoria di sistema
Il numero visualizzato risulta notevolmente più grande di quello ottenuto con FRE(0). Non è possibile dire esattamente quale sia il valore corrente di questo numero, in quanto dipende da cosa è in esecuzione in background, da quante finestre sono aperte e da quanti programmi sono presenti in memoria. Si potrebbe pensare di reperire ulteriore memoria per il bitmap grafico dalla memoria di sistema; più avanti verrà discusso come fare per ottenere in AmigaBASIC una quantità maggiore di memoria.

Dopo quanto detto, assume significato l'espressione LEN=FRE(0)-500. La porzione di memoria BASIC disponibile viene utilizzata come memoria buffer, con 500 byte riservati alle variabili. Se viene superata anche di un solo byte la quantità di memoria disponibile, AmigaBASIC visualizza il messaggio di errore "Out Of Memory".

Funzionamento della routine Save
La routine di salvataggio consente di effettuare, nel modo più veloce possibile, registrazioni su dischetto; dopo questa operazione viene chiuso il file e liberata la memoria di buffer.

La procedura di invio dei dati comincia con l'apertura del file cui segue l'invio dei dati verso la loro destinazione. Per prima cosa viene scritto nel file l'identificatore di Form, seguito dalla lunghezza della Form stessa. Tale lunghezza dipende dal numero di bitplane; dato che ogni bitplane utilizza 8000 byte e che per le informazioni di controllo dei dati e dei colori sono necessari 156 byte, la richiesta totale di memoria è di 156+8000 x (n. di bitplane) Byte.

Di seguito a queste informazioni viene scritto l'identificatore ILBM (Interleaved Bitmap).

Viene poi inserito il Chunk BMHD, la cui lunghezza è solitamente di 20 byte. Per prima cosa sono memorizzati i numeri relativi alla larghezza (320) ed all'altezza (256) del disegno, entrambi a 16 bit. Seguono poi quattro byte nulli (contenenti le coordinate di partenza del disegno sullo schermo). Il byte successivo contiene il numero di bitplane ed è seguito da un altro byte nullo.

I due byte seguenti consentono di determinare se i dati grafici memoriz-

zati nel Chunk BODY sono in forma compressa oppure no. Nel caso di dati non in forma compressa in tali byte viene posto il valore 0. Questa informazione viene utilizzata dai programmi che comprendono una routine di compensazione della compressione, cioè una routine di decompressione. Vi sono poi nuovamente due byte nulli che non sono attualmente utilizzati dallo standard IFF, ma sono riservati per ulteriori implementazioni. Si hanno poi due valori, 10 e 11, che indicano il grado della larghezza di un pixel in proporzione alla sua altezza, in modo tale che un'immagine IFF possa essere trasferita su un altro computer. Le proporzioni dello schermo possono essere radicalmente diverse da un computer all'altro. Amiga utilizza un rapporto di 10:11.

Si hanno infine i valori relativi alla risoluzione corrente (nel caso in esame 320x256). Il Chunk BMHD viene completamente riempito inserendo un ultimo ventesimo byte.

Vi sono poi le informazioni relative ai colori, Nel caso in esame sono stati salvati 32 diversi colori, cioè il numero massimo di colori visualizzabili. Devono quindi essere trasferiti un totale di 96 (32x3) valori RGB dal vettore Colori% al Chunk CMAP.

Determinazione dell'indirizzo dei bitplane

Ecco ora la routine più importante, quella relativa al salvataggio del Chunk BODY. La lunghezza di questo Chunk dipende dal numero di bitplane che si utilizzano (ogni singolo bitplane occupa 8000 byte). Nelle routine CaricaDisegno: e RegistraDisegno: vengono determinati gli indirizzi relativi ad ogni singolo bitplane. Pur essendo irrilevante quale routine venga eseguita per prima, è necessario esaminare il dato contenuto all'interno del vettore LocazionePiano.

Il ciclo legge direttamente dalla memoria utilizzando il comando PEEKL scrivendo poi i dati sul file. Le tre linee successive di programma sono state inserite per mostrare ciò che viene salvato.

Ad ogni ciclo di esecuzione la variabile LocazionePiano determina il primo byte della linea corrente del disegno relativamente al primo bitplane. Questo byte viene scritto (POKE) nell'indirizzo relativo dopo aver azzerato i bit più significativi (utilizzando AND 63). L'ultimo byte della linea grafica viene poi caricato nei bit meno significativi, dopo averli azzerati

(con AND 252). Si ottiene il seguente risultato: i bordi della finestra scompaiono da cima a fondo e viene salvata l'altezza della linea. In questo modo è possibile sapere quando il programma ha terminato il salvataggio. I bordi riappaiono non appena viene premuto il pulsante sinistro del mouse.

Salvataggio del Chunk "Spare"
Da ultimo viene salvato un ulteriore Chunk, non importante per il programma attualmente in esame, ma necessario se si desiderano riutilizzare i dati salvati con Graphicraft, o con altri programmi commerciali. Il Chunk CAMG (Commodore AMiGa) ha una dimensione di soli quattro byte e contiene un numero a 32 bit che fornisce ad Intuition informazioni relative al modo grafico di funzionamento. Dopo aver inviato al dischetto l'ultimo Chunk, il file viene chiuso.

Ora il programma di disegno è IFF-compatibile. Risulta quindi possibile caricare e salvare disegni ed utilizzarli in altri programmi di grafica. I pacchetti grafici commerciali permettono di tagliare od inserire porzioni di disegno, ingrandire singole aree, effettuare stampe del lavoro realizzato ed altro ancora. Il programma di disegno realizzato presenta una modalità di riempimento delle aree che potrebbe rivelarsi molto utile come supporto per Graphicraft.

Come si può quindi vedere, Amiga manifesta maggiori caratteristiche di compatibilità che non di incompatibilità. Tenendo presente questa considerazione è quindi possibile rendere IFF-compatibile il programma di videotitolazione precedentemente sviluppato.

Nota d'uso 4: la chiave di volta è la compatibilità; implementazione del programma di videotitolazione

A questo punto si può pensare di realizzare un disegno con Graphicraft, o con qualsiasi altro pacchetto grafico, inserire alcuni tratti realizzati con il programma di grafica precedentemente sviluppato, ed utilizzare poi l'immagine realizzata come sfondo per il programma di videotitolazione animata. Per realizzare questo è necessario modificare opportunamente il programma in modo che sia in grado di caricare file IFF.

Questione di risoluzione

Vi è tuttavia un problema da discutere immediatamente, quello relativo alla risoluzione utilizzata. Tutti i disegni realizzati con Graphicraft, e la maggior parte dei sistemi grafici IFF, gestiscono la bassa risoluzione (320x256). Anche il programma di disegno sviluppato in precedenza consente di gestire questa modalità; il programma di videotitolazione, tuttavia, effettua le visualizzazioni nella risoluzione standard del BASIC, cioè 640x256. I grafici IFF risultano così troppo piccoli, in quanto ricoprono solamente metà schermo.

La soluzione più semplice e più immediata è quella di apportare modifiche al programma di videotitolazione in modo che esso funzioni in bassa risoluzione. Non è necessaro effettuare una grande quantità di cambi, dato che la sua prima linea inizializza il testo visualizzato sullo schermo. Gli altri cambi devono essere effettuati per consentire di visualizzare caratteri più larghi e quindi più facili da leggere.

• Caricare il programma di videotitolazione da modificare.

Se si desidera mantenere sia la vecchia versione che la nuova, è necessario procedere al salvataggio, dopo aver inserito i cambi, della nuova

versione con un nome differente; in questo modo si conservano entrambe le versioni del programma di videotitolazione.

Cambi

Ecco le linee di programma che devono essere modificate: nella routine Inizializzazione: è necessario cancellare il ciclo IF ... THEN che precede il comando SCREEN. Nella prima versione del programma viene aperto un nuovo schermo solamente se devono essere utilizzati più di due bitplane; in caso contrario viene utilizzato il normale schermo BASIC. E' necessario ora aprire sempre e comunque un nuovo schermo, in quanto le caratteristiche che esso deve avere differiscono da quelle dello schermo BASIC. I parametri del comando SCREEN devono essere modificati ed il valore 640 deve essere sosituito con 320. Dopo la modifica la situazione della quinta linea del programma dovrebbe essere quella mostrata di seguito:

[prima]

```
IF Colori>2 THEN SCREEN 2,640,256,Colori,2: WINDOW
     2,"Videotitolatrice",,28,2¶
```

[dopo]

```
SCREEN 2,320,256,Colori,1: WINDOW 2,"Video
        titolatrice",,28,2¶
```

La linea successiva contiene alcune istruzioni di dimensionamento. MatriceColori dovrebbe essere dimensionata al maggior numero possibile di colori; è necessario perciò modificare MatriceColori (d,3) e trasformarlo in MatriceColori (31,3):

[prima]

```
DIM Testo$(d), MatriceColore(d,3), Muovi(d),
      Velocita(d)¶
```

[dopo]

```
DIM Testo$(d), MatriceColore(31,3), Muovi(d),
      Velocita(d)¶
```

Dopo aver modificato la routine Inizializzazione: è necessario intervenire sulla routine Partenza:

[prima]

```
PRINT "Videotitolatrice ";¶
```

[dopo]

```
PRINT "Videotitolatrice"¶
```

La modifica è consistita nella eliminazione del punto e virgola. In questo modo viene forzato un ritorno a capo tra il testo Videotitolatrice ed il nome, che altrimenti avrebbe oltrepassato il bordo destro dello schermo.

Si deve ora effettuare una serie ingente di modifiche, iniziando dalla routine LeggiOggetto:

[prima]

```
PRINT "Immetti il nome dell'oggetto da caricare."¶
```

[dopo]

```
PRINT "Immetti il nome dell'oggetto": PRINT
     "da caricare."¶
```

Ecco ora le modifiche alla routine Mossiere:

[prima]

```
PRINT "Muovere l'oggetto verso il suo punto
     di partenza"¶
PRINT "usando il tasto freccia."¶
```

[dopo]

```
PRINT "Muovere l'oggetto verso"¶
PRINT "il suo punto di partenza"¶
PRINT "usando i tasti freccia."¶
```

Deve ora essere modificata la routine Colori:

[prima]

```
Colori:¶
  FOR x=0 TO MaxColori¶
     COLOR -(x=0),x¶
     LOCATE 5, (X*4) + 1¶
     PRINT x;CHR$(32);CHR$(32)¶
   NEXT x¶
```

[dopo]

```
Colori:¶
  FOR x=0 TO MaxColori¶
     IF (x/8)=INT(x/8) THEN PRINT¶
     COLOR -(x=0),x¶
     PRINT x¶
     IF x<10 THEN PRINT CHR$(32)¶
   NEXT x¶
```

La routine risulta abbastanza semplice da comprendere.

```
IF (x/8)= INT (x/8)
```

significa: se x risulta essere un multiplo di 8 (cioè 8, 16, 24, ...) deve essere inserito un ritorno a capo in modo tale da ottenere la visualizzazione in maniera corretta del vettore dei colori.

```
IF x<10 THEN
```

inserisce uno spazio bianco per i numeri compresi tra 0 e 9, dato che questi sono numeri a singola cifra, mentre tutti quelli superiori a 10 sono a doppia cifra, garantendo in questo modo il corretto incolonnamento.

Deve essere ora alterata la routine CambioColore:

[prima]

```
PRINT "Immetti il numero del colore da cambiare."¶
```

[dopo]

```
PRINT "Immetti il numero del colore":PRINT
      "da cambiare."¶
```

I cambi da effettuare nella routine RegolatoreRGB: comprendono le compensazioni necessarie per la dimensione inferiore del video.

I cursori R, G, e B devono essere spezzati su due linee:

[prima]

```
LOCATE 10,1: PRINT "Rosso (R): <7>=- <8>=+ ";
   Riempitore$¶
LOCATE 10,24+r : PRINT CHR$(124);¶
LOCATE 11,1: PRINT "Verde (G): <4>=- <5>=+ ";
   Riempitore$¶
LOCATE 11,24+g : PRINT CHR$(124);¶
LOCATE 12,1: PRINT "Blu   (B): <1>=- <2>=+ ";
   Riempitore$¶
LOCATE 12,24+b : PRINT CHR$(124);¶
LOCATE 13,1:PRINT "            <0>=Colore o.k."¶
```

[dopo]

```
LOCATE 11,1: PRINT "Rosso (R): <7>=- <8>=+ ":
   PRINT Riempitore$¶
LOCATE 12,r+1 : PRINT CHR$(124);¶
LOCATE 13,1: PRINT "Verde (G): <4>=- <5>=+ ":
   PRINT Riempitore$¶
LOCATE 14,r+1 : PRINT CHR$(124);¶
LOCATE 15,1 : PRINT "Blu   (B): <1>=- <2>=+ ":
   PRINT Riempitore$¶
LOCATE 16,1+b : PRINT CHR$(124);¶
LOCATE 17,1: PRINT "           <0>=Colore o.k."¶
```

Le posizioni identificate dai comandi LOCATE vengono modificate: le linee sono incrementate di 1, mentre, per quanto riguarda le colonne, al posto del valore 24+r compare r+1. Inoltre Riempitore$ è ogni volta pre-

ceduto dal comando PRINT.

La modifica successiva relativa alle istruzioni LOCATE deve essere effettuata nella routine ImmettiColore: ed in Ciclo3:

[prima]

```
LOCATE 14,1¶
```

[dopo]

```
LOCATE 19,1¶
```

L'ultima modifica deve essere effettuata in altri due punti.

Nella routine ContoAllaRovescia deve essere modificato il numero della colonna:

[prima]

```
LOCATE 14,36: PRINT c¶
```

[dopo]

```
LOCATE 14,18: PRINT c¶
```

Deve essere ora modificata la lunghezza della linea specificata dal comando WIDTH nella routine InizioVisualizzazione:

[prima]

```
WIDTH 60¶
```

[dopo]

```
WIDTH 33¶
```

La modifica effettuata richiede l'intervento anche su altre linee della stessa routine, quelle in cui viene suddiviso il testo in base alla lunghezza della linea. Devono quindi essere modificate la sesta e la settima linea della routine InizioVisualizzazione:

[prima]

```
Testo$=LEFT$(Testo(x),60)¶
h=Int((60-LEN(Testo$))/2)+2¶
```

[dopo]

```
Testo$=LEFT$(Testo(x),32)¶
h=Int((32-LEN(Testo$))/2)+2¶
```

Sono a questo punto terminate le modifiche da apportare al programma per consentirne il funzionamento in bassa risoluzione. E' consigliabile provare ad eseguire il programma alcune volte, in modo tale da comprenderne il funzionamento e da individuare eventuali errori di battitura compiuti.

E' consigliabile procedere a salvare il programma con un nuovo nome, in modo da mantenere anche la vecchia versione.

4.5 Luci, Camera, Azione: caricare disegni nel programma di videotitolazione

Il programma di videotitolazione necessita ora di una routine per il caricamento di file IFF. Non è il caso di inserire completamente di nuovo questa routine: è infatti consigliabile utilizzare alcuni accorgimenti per riutilizzare i testi di programma già battuti, apportando eventualmente leggere modifiche.

- Salvare il programma di videotitolazione, cambiare directory corrente e caricare il programma di grafica.

Inserire routine all'interno di un programma
Il programma di grafica contiene una routine che consente di caricare un'immagine in bassa risoluzione. E' possibile inserire questa routine nel programma di videotitolazione con un piccolissimo sforzo.

- Inserire la seguente linea nella finestra BASIC:

```
LIST CaricaDisegno-FineCarica,"Ram:IFF.Load"
```

Questa linea di comando consente di salvare la routine di caricamento del programma di grafica nel disco virtuale Ram: come file ASCII. Tutte le linee di programma comprese tra le etichette CaricaDisegno e FineCarica vengono memorizzate nel file IFF.Load.

- Cancellare ora dalla memoria il programma di grafica utilizzando il comando NEW e caricare nuovamente il programma di videotitolazione.

- Effettuare l'inserimento della nuova routine utilizzando il comando:

```
MERGE "Ram:IFF.Load"
```

Raffinamenti
E' ora necessario effettuare alcuni raffinamenti per integrare la nuova routine con il resto del programma.

Per prima cosa è necessario aggiungere una opzione per richiamare questa routine da menù.

• Inserire, dopo la quinta linea della routine Selezione:, la linea seguente:

```
PRINT "6 Caricamento immagine sfondo"
```

E' ora necessario effettuare alcuni cambi derivanti dall'inserimento di questa nuova opzione; la routine Richiesta: deve essere modificata come mostrato di seguito:

[prima]

```
IF a$<"1" OR a$>"5" THEN BEEP: GOTO Richiesta
```

[dopo]

```
IF a$<"1" OR a$>"6" THEN BEEP: GOTO Richiesta
```

Infatti il massimo input accettabile è ora 6 e non più 5. Poche linee dopo, nella serie di istruzioni IF ... THEN, deve essere inserita la chiamata alla nuova subroutine:

```
IF a$=6 THEN  PreparaSchermo
```

Per posizionare adeguatamente la nuova linea all'interno del menù, deve essere opportunamente modificato il parametro del comando LOCATE nella routine Richiesta:

[prima]

```
LOCATE 11,1
```

[dopo]

```
LOCATE 16,1
```

- Inserire le linee di programma che seguono, subito dopo la routine CalcoloVelocita:

```
PreparaSchermo:¶
  CLS¶
  PRINT "Desideri caricare"¶
  PRINT "una immagine come sfondo?"¶
   PRINT "Risposta: (S/N)"¶
¶
Ciclo5:¶
  LOCATE 3,17 : INPUT RispostaDue$¶
  IF UCASE$(RispostaDue$)="N" THEN IFF=0 : CLS :
      GOTO Partenza¶
  IF UCASE$(RispostaDue$)="S" THEN IFF=1 :
      GOTO ImmettiNome ¶
GOTO Ciclo5¶
¶
ImmettiNome:¶
  PRINT¶
  PRINT "Indica il nome:"¶
  INPUT NomeDue$¶
  PRINT ¶
  PRINT "Vuoi usare la tavolozza"¶
  PRINT "dei colori di "NomeDue$" ?"¶
  PRINT "Rispondi (S/N)";¶
Ciclo6:¶
  LOCATE 10,16 : INPUT RispostaDue$¶
  IF UCASE$(RispostaDue$)="N" THEN IFFTab=0 : CLS :
      GOTO Partenza¶
  IF UCASE$(RispostaDue$)="S" THEN IFFTab=1 : CLS :
      GOTO Partenza¶
GOTO Ciclo6¶
```

La routine PreparaSchermo: chiede innanzitutto se si desidera caricare un'immagine per lo sfondo oppure no. Gli input accettati, in risposta alla richiesta, sono <S> ed <N>. Ogni altro input fornito causa la riesecuzione del ciclo di input. La variabile IFF stabilisce se deve essere caricato un disegno (IFF=1) oppure no (IFF=0).

La routine ImmettiNome, chiede il nome dell'immagine da caricare. Il nome fornito viene memorizzato nella variabile NomeDue$. Viene poi chiesto se si desidera caricare (dal Chunk CMAP) la tavolozza relativa a tale immagine oppure no.

NOTA:
Il testo ed il disegno devono avere gli stessi colori; nel caso si siano utilizzati colori differenti per il testo e lo sfondo da quelli utilizzati nel disegno, è necessario rispondere <N> alla richiesta precedente. Il programma utilizza allora la tavolozza inizializzata nel programma di videotitolazione. La variabile IFFTab viene utilizzata per memorizzare il comportamento richiesto.

E' ora necessario modificare la routine LetturaDati:, inserendo alcune opzioni per il nuovo programma. Durante la fase di modifica deve essere posta grande attenzione quando si inseriscono comandi GOTO o GOSUB che richiamano una nuova routine.

Stesse variabili, utilizzi differenti
La routine IFF creata utilizza numerose di variabili necessarie al programma principale. Ad esempio, le variabili x ed y vengono utilizzate per gestire il corretto dimensionamento. Ad un certo punto x assume il valore 3000. Dopo aver caricato l'ultima immagine di sfondo il programma ritorna alla routine di caricamento. Alla variabile x, utilizzata poi per contare le variabili in ColorMap:, viene assegnato il valore 31. In un solo colpo sono così andati persi 2969 pixel. Ma quale dei due valori della x è più importante?

Come si può vedere, utilizzare la stessa variabile sia nel programma principale che in una subroutine provoca risultati indesiderati. Una possibile soluzione è quella di utilizzare nomi diversi per le variabili all'interno delle subroutine, ma anche questo non garantisce la sicurezza delle variabili.

I sottoprogrammi
Il comando SUB ... STATIC è stato realizzato proprio per risolvere questo problema. Esso consente infatti di definire un tipo di subroutine speciale, chiamato *sottoprogramma.* I sottoprogrammi si differenziano dalle subroutine per la diversa gestione delle variabili: essi usano infatti *variabili locali.*

Questo significa che tutte le variabili all'interno di un sottoprogramma mantengono i loro valori staccati dalle variabili del programma principale. Se si utilizza la variabile x sia nel programma principale che nel sottoprogramma, i valori sono gestiti separatamente l'uno dall'altro.

SUB ... STATIC

E' così possibile modificare il valore della x nel programma principale senza che venga contemporaneamente modificato il valore della x nel sottoprogramma. E' possibile assegnare alla variabile della SUB un proprio registro in memoria. Se AmigaBASIC ritorna da un sottoprogramma al programma principale, e poi nuovamente al sottoprogramma, all'interno del registro si trova lo stesso valore assegnato precedentemente. La parola chiave STATIC deve seguire il comando SUB. Il nome della routine deve essere compreso tra SUB e STATIC. Ad esempio:

```
SUB convertitore STATIC¶
```

Questa linea non ha niente a che fare con il programma di videotitolazione, è stata semplicemente utilizzata come esempio.

END SUB

Ogni sottoprogramma deve essere terminato dal comando END SUB. E' possibile inserire un sottoprogramma in un punto qualsiasi del programma. Dato che si tratta di una sezione chiusa (racchiusa tra SUB ... STATIC e END SUB), AmigaBASIC è in grado di riconoscere che essa non costituisce parte del programma principale. Una routine non viene eseguita solamente per il fatto che è inserita nel mezzo del programma: ad esempio, nel programma seguente vengono eseguite solamente la prima e l'utlima linea.

```
x-=100¶
¶
SUB conversione STATIC¶
x=40¶
x=x*100+3¶
PRINT "eccomi"¶
END SUB¶
¶
PRINT x¶
```

Anche queste linee sono solamente di esempio e non fanno parte del programma di videotitolazione.

Errori nei sottoprogrammi

Vi sono alcune regole che devono essere tenute ben presenti quando ci si accinge a scrivere un sottoprogramma. All'interno di un sottoprogramma non possono ad esempio essere definiti altri sottoprogrammi, nè possono essere definiti sottoprogrammi all'interno di una struttura SUB/END.

La struttura seguente è quindi proibita:

```
SUB Testo STATIC¶
    SUB Testo2 STATIC¶
        PRINT "Testo 2"¶
     END SUB¶
     PRINT "Testo"¶
END SUB¶
```

Amiga visualizzerà pertanto il messaggio d'errore "Tried to declare SUB within SUB".

Se vengono definiti due sottoprogrammi con lo stesso nome, si ottiene il messaggio di errore "SUB already defined".

Se si dimentica la parola chiave STATIC Amiga visualizza il messaggio "Missing STATIC in SUB statement".

Se viene tralasciato il comando END SUB viene segnalato un errore: "SUB without END SUB".

CALL

Per richiamare un sottoprogramma si utilizza il comando CALL. Ad esempio:

```
CALL convertitore¶
```

A volte è possibile omettere CALL e richiamare il sottoprogramma specifi-

candone direttamente il nome.

```
convertitore¶
```

CALL abbreviato

Alcuni esempi di utilizzo corretto e scorretto di chiamata abbreviata sono mostrati di seguito (nessuno di questi esempi ha a che fare con il programma di videotitolazione, ma è necessario apprendere alcune cose per poi poterle applicare all'interno di questo programma).

```
FOR x=1 TO 10: convertitore: NEXT x¶
```

La linea è abbastanza semplice da comprendere; il sottoprogramma convertitore viene richiamato all'interno del ciclo FOR.

Il caso successivo è un poco più difficile da discutere:

```
convertitore: PRINT x¶
```

Il problema che sorge è relativo al significato di convertitore: si tratta di una chiamata di SUB seguita da un separatore di istruzioni, oppure di una etichetta per identificare una sezione di programma che comincia con un comando PRINT? In questo caso la parola convertitore viene comunque interpretata come un'etichetta. Se si vuole realizzare una chiamata di sottoprogramma è quindi necessario specificarlo utilizzando CALL.

```
CALL convertitore: PRINT x¶
```

Ecco un ulteriore esempio più o meno simile:

```
IF x=10 THEN convertitore¶
```

Il termine convertitore viene qui interpretato come etichetta da cui il programma deve poi continuare l'esecuzione. Per realizzare la chiamata di sottoprogramma è ancora una volta necessario utilizzare CALL:

```
IF x=10 THEN CALL convertitore¶
```

Dopo aver esposto queste importanti questioni riguardo ai sottoprogrammi, è opportuno ritornare a considerare il programma di videotitolazione. La routine CaricaDisegno verrà trasformata in un sottoprogramma, in modo da ovviare ai problemi di sovrapposizione dei nomi delle variabili.

- Aggiungere all'etichetta CaricaDisegno quanto segue:

```
SUB  CaricaDisegno STATIC¶
```

- Ritornare ora al programma principale e cancellare la parte di programma che precede la linea:

```
IF NomeDue$="" THEN FineCarica¶
```

Dovrebbero in tal modo essere state cancellate le prime cinque linee del sottoprogramma CaricaDisegno. Si inserirà ora la parte nuova, in sostituzione di queste linee. Ci si potrebbe chiedere ora: se un sottoprogramma utilizza variabili proprie che non modificano i valori delle variabili del programma principale, come è possibile utilizzare in un sottoprogramma valori provenienti dal programma principale e viceversa?

Se, ad esempio, si desidera comunicare al sottoprogramma il numero di bitplane, trasferendolo dalla variabile Colori al vettore MatriceColore, che cosa è necessario fare?

SHARED

Fortunatamente gli sviluppatori di AmigaBASIC hanno dedicato un comando proprio a questo: il comando SHARED.

- Inserire nel sottoprogramma, di seguito alla linea SUB ... STATIC, la linea seguente:

```
SHARED Colori, MatriceColori(), IFFTab, NomeDue$¶
```

Il programma principale ed il sottoprogramma condividono le varabili specificate. Questo significa che si tratta di variabili comuni. E' possibile specificare tanto variabili quanto vettori in un comando SHARED. Come è possibile notare, dopo il nome del vettore è necessario inserire una coppia di parentesi (). In questo modo risulta possibile distinguere tra nomi di vettori e di variabili.

Il prossimo cambio da effettuare nel programma è all'interno della routine ColorMap.

Ecco il listato originale:

```
ColorMap:¶
  FOR x=0 TO (Lunghezza/3)-1¶
    r=(ASC(INPUT$(1,1)) AND 240)/16¶
    g=(ASC(INPUT$(1,1)) AND 240)/16¶
    b=(ASC(INPUT$(1,1)) AND 240)/16¶
    ¶
    PALETTE x,r/16,g/16,b/16¶
    Colori(x,1)=r :Colori(x,2)=g : Colori(x,3)=b¶
    ¶
  NEXT x¶
  IF INT(Lunghezza/3)<>(Lunghezza/3) THEN
      Fasullo$=INPUT$(1,1)¶
GOTO LetturaDati¶
```

Ed ecco quello modificato:

```
ColorMap:¶
  FOR x=0 TO (Lunghezza/3)-1¶
    r=(ASC(INPUT$(1,1)) AND 240)/16¶
    g=(ASC(INPUT$(1,1)) AND 240)/16¶
    b=(ASC(INPUT$(1,1)) AND 240)/16¶
    IF IFFTab=1 THEN¶
       PALETTE x,r/16,g/16,b/16¶
       MatriceColore(x,1)=r : MatriceColore(x,2)=g :
            MatriceColore(x,3)=b¶
    END IF¶
  NEXT x¶
  IF INT(Lunghezza/3)<>(Lunghezza/3) THEN
      Fasullo$=INPUT$(1,1)¶
GOTO LetturaDati¶
```

Per prima cosa deve essere inserito un blocco IF ... THEN. I colori letti devono essere passati solamente quando IFFTab vale 1. Il vettore Colori del programma grafico viene poi modificato nel vettore MatriceColore. La seconda dimensione per il vettore risulta incrementata di una unità. Il valore per la banda rossa del colore x non è contenuto in MatriceColore(x,0), ma in MatriceColore(x,1). Lo stesso deve essere effettuato per i valori del verde e del blu.

E' ora necessario modificare la routine FineCarica.

```
FineCarica:¶
CLOSE 1¶
END SUB¶
```

A questo punto la routine di caricamento dei file IFF è stata modificata per il programma di videotitolazione. E' ora necessario inserire la chiamata della subroutine.

- Posizionare il cursore nella sezione InizioVisualizzazione: del programma ed inserire il comando CALL:

```
InizioVisualizzazione:¶
  WIDTH 33 ¶
  COLOR ColoreTesto,Sfondo : CLS¶
  COLOR ColoreTesto,SfondoTesto¶
  IF IFF=1 THEN CALL CaricaDisegno¶
  FOR x=1 TO NumLinee¶
```

Ora la lettura dell'immagine per lo sfondo è completa.

Utilizzo della routine

Gli effetti delle modifiche apportate al programma non sono difficili da comprendere. Per utilizzare un'immagine di sfondo è sufficiente effettuare un click sull'opzione 6 del menù. E' possibile inserire il nome del file e determinare se debba essere utilizzata la tavolozza corrente oppure quella del file IFF. Dopo il conto alla rovescia e prima che il testo e gli oggetti in movimento vengano mostrati, vengono caricati i parametri per lo sfondo.

NOTA:

il caricamento dell'immagine di sfondo cancella i bordi della finestra. Questo non interferisce con il titolo, ma, se si desiderano avere i bordi, è necessario effettuare un click all'interno della finestra stessa.

Se si desidera avere sullo schermo un secondo oggetto in movimento, è possibile utilizzare le Preferences per modificare a piacimento il puntato-

re; in questo modo si avrà a disposizione un secondo oggetto grafico i cui movimenti saranno completamente controllati dal mouse.

Nel prossimo paragrafo si vedranno ulteriori miglioramenti al programma di videotitolazione.

4.6 Un'altra idea: caricare e salvare sequenze titolate

Dopo aver progettato un titolo, ci si potrebbe sentire frustrati perchè non è possibile salvarne i parametri. In questo paragrafo si mostrerà come salvare e successivamente caricare sequenze titolate.

• Inserire la seguente routine alla fine del programma di videotitolazione:

```
RegistraTitolo:¶
  CLS : PRINT "Registri con il nome di:"¶
  INPUT NomeDati$¶
  OPEN NomeDati$ FOR OUTPUT AS 1¶
    PRINT #1,NumLinee      : REM Numero delle linee di
       testo¶
    FOR x=1 TO NumLinee¶
      WRITE #1,Testo$(x)¶
    NEXT x¶
    ¶
    PRINT #1,IndicatoreOgg      ' E' stato caricato un
       oggetto?¶
    WRITE #1,NomeOgg$           ' nome del file ¶
    ¶
    PRINT #1,Mossa(0)    ' Numero di movimenti¶
    FOR x=1 TO Mossa(0)¶
      PRINT #1,Mossa(x)¶
    NEXT x¶
    ¶
    PRINT #1,Colori      ' Numero di Bitplane¶
    FOR x=0 TO 31        ' 32 Colori in IFF¶
      PRINT #1,CHR$(MatriceColore(x,1)*16);¶
      PRINT #1,CHR$(MatriceColore(x,2)*16);¶
      PRINT #1,CHR$(MatriceColore(x,3)*16);¶
```

```
    NEXT x    ¶
    PRINT #1,Sfondo      ' Colore del testo etc.¶
    PRINT #1,ColoreTesto¶
    PRINT #1,SfondoTesto¶
    ¶
    PRINT #1,IFF         ' C'è uno sfondo per lo
        schermo?¶
    PRINT #1,IFFTab      ' C'è un cambio di colori?¶
    WRITE #1,NomeDue$    ' Nome file ¶
  CLOSE 1¶
  CLS¶
GOTO Inizio¶
¶
RecuperaTitolo:¶
  CLS : PRINT "Nome del file da recuperare:"¶
  INPUT NomeDati$¶
  OPEN NomeDati$ FOR INPUT AS 1¶
    INPUT #1,NumLinee¶
    FOR x=1 TO NumLinee¶
      INPUT #1,Testo$(x)¶
    NEXT x¶
    ¶
    INPUT #1,IndicatoreOgg¶
    INPUT #1,NomeOgg$¶
    ¶
    IF IndicatoreOgg=1 THEN¶
      OPEN NomeOgg$ FOR INPUT AS 2¶
        OBJECT.SHAPE 1,INPUT$(LOF(2),2)¶
      CLOSE 2¶
    END IF¶
    ¶
    INPUT #1,Mossa(0)¶
    FOR x=1 TO Mossa(0)¶
      INPUT #1,Mossa(x)¶
    NEXT x¶
    ¶
```

```
      INPUT #1,Colore1¶
      IF Colore1<=Colori THEN Colori=Colore1¶
      MaxColori=(2^Colori)-1¶
      FOR x=0 TO 31¶
        r=(ASC(INPUT$(1,1)) AND 240)/16¶
        g=(ASC(INPUT$(1,1)) AND 240)/16¶
        b=(ASC(INPUT$(1,1)) AND 240)/16¶
        PALETTE x,r/16,g/16,b/16¶
        MatriceColore(x,1)=r : MatriceColore(x,2)=g :
              MatriceColore(x,3)=b¶
      NEXT x¶
      INPUT #1,Sfondo¶
      INPUT #1,ColoreTesto¶
      INPUT #1,SfondoTesto¶
      ¶
      INPUT #1,IFF¶
      INPUT #1,IFFTab¶
      INPUT #1,NomeDue$¶
    CLOSE 1¶
    CLS¶
  GOTO Inizio¶
```

Si dovrebbe già avere una idea del funzionamento di questa routine per cui non è il caso di soffermarsi su di essa più di tanto. Di seguito sono riportate alcune considerazioni e spiegazioni sulle eventuali sezioni di programma che potrebbero non risultare immediatamente comprensibili.

E' stata usata la stessa sintassi precedentemente utilizzata per il salvataggio dei colori in formato IFF, in quanto tale sintassi consente di risparmiare una certa quantità di memoria. Comunque, il file creato non è IFF-compatibile.

Il comando WRITE consente di evitare i problemi derivanti dalla presenza di virgole all'interno delle varie stringhe di testo.

Se il file specificato esiste, i dati relativi agli oggetti grafici vengono caricati direttamente in memoria. OBJECT.SHAPE viene etichettato come

oggetto numero 1.

Se il file contenente il titolo ha più bitplane di quelli permessi, il programma di videotitolazione utilizza il numero massimo permesso. La variabile Colore1 legge i valori dal file. Se Colore1 è minore od uguale al valore contenuto in Colori, la routine di lettura memorizza tale valore in Colori.

REM

Il comando REM viene utilizzato per inserire commenti all'interno dei listati BASIC. Un commento non ha niente a che fare con l'esecuzione del programma, ma costituisce semplicemente una descrizione o una nota per consentire al programmatore di comprendere il significato di parti di programma. Osservando, ad esempio, le due linee seguenti, estratte dalla routine RegistraTitolo:

```
OPEN NomeDati$ FOR OUTPUT AS 1¶
PRINT #1,NumLinee : REM Numero delle linee di testo¶
```

si può notare come il testo che segue REM sia utilizzato per spiegare cosa succede nel programma. Ogni testo che segue un comando REM viene ignorato dal programma.

• Provare ad inserire nella finestra BASIC la linea seguente:

```
FOR x=1 TO 10: REM commento: NEXT x¶
```

Questo ciclo non viene eseguito correttamente, dato che AmigaBASIC ignora il comando NEXT x e visualizza il messaggio di errore "FOR without NEXT". Per ottenere lo stesso identico effetto del comando REM è possibile utilizzare un apostrofo (al posto di REM), come viene ad esempio effettuato nella linea seguente, estratta dalla routine RegistraTitolo:

```
PRINT #1,IndicatoreOgg ' E' stato caricato un
      oggetto?¶
```

Il commento che segue l'apostrofo spiega lo scopo della variabile IndicatoreOgg. I commenti si rivelano molto utili per inserire spiegazioni in alcune parti di programma, o per fornire consigli all'utente. Nella routine RegistraTitolo i commenti sono utilizzati per spiegare il significato delle varie variabili.

Il testo dei commenti non viene eseguito da AmigaBASIC, ma esso deve essere letto e gestito, cosa che richiede una certa quantità di tempo. E' consigliabile lasciare i commenti fuori dalle parti di programma che vengono eseguite un elevato numero di volte. Benchè il tempo per analizzare un commento corrisponda ad una piccolissima frazione di secondo, un REM in un ciclo FOR .. NEXT che viene eseguito 8000 volte appesantirà il tempo di esecuzione per più di un minuto.

Aggiungere i menù

Devono ancora essere inserite le istruzioni per la selezione delle nuove opzioni nella routine Selezione:

• Inserire le linee seguenti, dopo la linea relativa all'opzione numero 6:

```
PRINT "7 Recupero Sequenza Titolo"¶
PRINT "8 Registrazione Sequenza Titolo"¶
```

La routine Richiesta deve essere modificata inserendo le nuove opzioni.

• Aggiungere le linee seguenti a tale routine:

```
Richiesta:¶
  LOCATE 16,1¶
  PRINT "Immettere un numero:";¶
  INPUT a$¶
  a$=LEFT$(a$,1)¶
  IF a$<"1" OR a$>"9" THEN BEEP: GOTO Richiesta¶
  IF a$="1" THEN ImmettiTesto¶
  IF a$="2" THEN LeggiOggetto¶
  IF a$="3" THEN MuoviOggetto¶
  IF a$="4" THEN DefinisciColore¶
  IF a$="5" THEN MostraTitolo¶
  IF a$="6" THEN PreparaSchermo¶
  IF a$="7" THEN RecuperaTitolo¶
  IF a$="8" THEN RegistraTitolo¶
  GOTO Richiesta¶
```

L'effetto è abbastanza semplice: quando è stato completato un titolo e si

desidera salvarlo, è necessario selezionare l'opzione numero 8. La subroutine richiede di specificare il nome del file in cui salvare la sequenza. I dati vengono poi scritti nel file specificato. Quando si desidera richiamare il titolo, è necessario selezionare l'opzione numero 7 ed inserire il nome del file relativo. Dopo che il contenuto del file è stato caricato, è possibile selezionare l'opzione MostraTitolo.

Il programma corto di videotitolazione

Concatenando tra loro solamente le routine Inizializzazione, MostraTitolo, CalcoloVelocita, LeggiTitolo e CaricaDisegno tralasciando tutte le altre, è possibile ottenere una routine per la titolazione da utilizzare eventualmente in altri programmi.

4.7 Un piccolo sovrappiù: aggiungere comandi propri

A questo punto si conosce la modalità in cui opera la routine di caricamento di un programma commerciale. Certamente sarebbe stato molto più comodo se la Microsoft avesse inserito direttamente nell'AmigaBASIC comandi per caricare e salvare immagini. Benchè AmigaBASIC sia un BASIC abbastanza buono, a volte è necessario costruire alcuni comandi che non esistono di per se stessi.

Comandi definiti dall'utente
Gli sviluppatori di AmigaBASIC hanno realizzato il linguaggio in modo da consentire la definizione di comandi da parte dell'utente con la possibilità di utilizzare poi tali comandi come se fossero comandi propri del linguaggio. La definizione di un sottoprogramma rappresenta un semplice modo per definire comandi propri. E' possibile creare sottoprogrammi che realizzino comandi non appartenenti al linguaggio, o comandi che estendono e modificano quelli già esistenti.

Modifica della funzione Date$
Il comando Date$ fornisce la data di sistema. La sintassi AmigaBASIC per la presentazione della data prevede che venga visualizzato prima il mese, poi il giorno ed infine l'anno. Nel caso si preferisse visualizzare la data in un altro formato è possibile realizzare un sottoprogramma come il seguente:

```
SUB data STATIC¶
    PRINT MID$(DATE$,4,2)"."LEFT$(DATE$,2)
        "."RIGHT$(DATE$,4)¶
END SUB¶
```

Grazie a questo sottoprogramma è possibile a questo punto richiamare direttamente data, invece di utilizzare PRINT DATE$, per visualizzare la data

di sistema con il nuovo formato.

E' possibile richiamare il sottoprogramma in modo diretto, inserendo la chiamata nella finestra BASIC.

Quando un sottoprogramma si trova all'interno del programma attualmente in memoria può essere richiamato in modo diretto.

Si dovrebbe essere ora in grado di comprendere il motivo per cui AmigaBASIC visualizza il messaggio di errore "Undefined Subprogram" quando non riesce a comprendere l'input fornito. Molte volte questo messaggio è la conseguenza di un errore di battitura. Se ad esempio si inserisce la linea:

```
PINT a$¶
```

invece di

```
PRINT a$¶
```

AmigaBASIC ricerca un sottoprogramma di nome PINT, e, non trovandolo, visualizza il suddetto messaggio di errore.

MID$

Il comando MID$ analogo ai comandi RIGHT$ e LEFT$, consente di posizionarsi logicamente all'interno di una stringa indicando il nome della stringa, la posizione in cui collocarsi e la porzione di stringa che si desidera estrarre:

```
PRINT MID$(nome-stringa, posizione, lunghezza)
```

Per isolare due caratteri della stringa a$ a partire dal quarto, è necessario utilizzare il comando MID$(a$,4,2). Nel sottoprogramma precedente, ad esempio, tale comando è stato utilizzato per isolare i caratteri relativi al giorno, all'interno della stringa DATE$.

Parametri multipli

Gran parte dei comandi sono seguiti da numerosi parametri. E' possibile specificare parametri anche per un sottoprogramma.

Si supponga di voler scrivere un programma che converta con il cambio corrente valori in Dollari in Lire. Supponendo che il cambio attuale sia di 1350 Lire per un dollaro, il sottoprogramma risulterà essere il seguente:

```
SUB Dollari (valore) STATIC¶
    PRINT "$"Valore " = "Valore*1350" Lire"¶
END SUB¶
```

I parametri specificati per un sottoprogramma sono racchiusi tra parentesi di seguito al nome del sottoprogramma stesso. Nell'esempio mostrato il nome del parametro è Valore.

• Provare ad inserire la linea seguente nella finestra BASIC:

```
Dollari 34.00¶
```

Amiga presenta come risultato:

```
$ 34 = 45900 Lire¶
```

Variabili locali

L'esempio mostrato consente di determinare il corrispondente in Lire di 34 dollari. Il sottoprogramma riceve il valore fornito per il parametro e calcola da esso l'ammontare in lire. Valore risulta essere una *variabile locale* in quanto non è possibile accedere al suo contenuto dal programma principale.

Nel caso dell'esempio precedente questo non è necessario, ma a volte ciò potrebbe rivelarsi molto utile.

Nel caso si desideri conservare il valore calcolato nel sottoprogramma, per poter utilizzare nel programma principale il contenuto della variabile locale, è sufficiente procedere come mostrato di seguito:

• Modificare il sottoprogramma nel modo seguente:

```
SUB Dollari (Valore) STATIC¶
    Valore = Valore * 1350¶
END SUB¶
¶
```

• Inserire nella finestra BASIC:

```
Valore=34: Dollari Valore :? Valore¶
```

Alla variabile Valore viene assegnato 43; viene poi richiamato il sottoprogramma Dollari con il parametro Valore. Quando viene visualizzata la variabile Valore il suo contenuto è stato modificato in base all'operazione svolta nel sottoprogramma. E' possibile richiamare il sottoprogramma anche con una qualsiasi altra variabile:

```
Pippo = 6: Dollari Pippo: ? Pippo¶
```

Non vi è niente di magico in tutto questo: si tratta solamente di una connessione con la sintassi del sottoprogramma. La variabile locale Valore agisce semplicemente come marcatore per il valore da passare al sottoprogramma memorizzando il valore per la computazione. Viene poi passato il valore di Pippo. La computazione viene poi effettuata con Valore. Una volta completato il calcolo, il contenuto di Valore viene trasferito in Pippo ed il sottoprogramma termina.

Ecco un altro esempio chiarificatore, un semplicissimo programma per raddoppiare un numero:

```
SUB Raddoppia(numero) STATIC¶
     Numero= Numero * 2¶
END SUB¶
```

• Richiamare il sottoprogramma in modo diretto utilizzando la seguente linea:

```
a=2 : Raddoppia a : PRINT a¶
```

Viene richiamato il sottoprogramma e viene fornita come parametro la variabile a. Numero viene elaborato dalla routine che trasferisce poi il contenuto della variabile Numero in a. In questo modo il programma principale riceve il risultato. Non è necessario avere precedentemente alcuna cognizione dell'esistenza della variabile Numero.

Normalmente vi sono anche situazioni in cui non si desidera che il valore della variabile del programma principale venga modificato pur essendo necessario passare tale variabile al sottoprogramma. Esiste una specie di meccanismo di salvataggio per preservare il valore precedente della varia-

bile del programma principale; tale meccanismo consiste nell'utilizzo di parentesi in cui racchiudere il nome della variabile.

Richiamando la routine Raddoppia con la linea seguente:

```
a=2: Raddoppia(a):PRINT a¶
```

il valore di a non viene modificato: a mantiene il valore 2 anche dopo la chiamata del sottoprogramma, il quale viene eseguito normalmente, ma il cui risultato non viene riportato in a, in quanto tale variabile è protetta dalla parentesi.

Ovviamente l'esempio è abbastanza inutile, ma se si considera nuovamente l'esempio precedente:

```
SUB Dollari (Valore) STATIC¶
    PRINT VALORE "Dollari corrispondono a "¶
     Valore = Valore * 1350¶
     PRINT Valore" Lire"¶
END SUB¶
```

Quando questo sottoprogramma viene richiamato con l'istruzione:

```
Doll=34:Dollari Doll¶
```

viene effettuata la conversione da dollari in lire, ma viene anche modificato il valore di Doll che contiene ora il corrispondente valore in lire. All'interno del programma il vecchio valore di Doll non esiste più.

Per evitare il verificarsi di questo fatto è possibile eseguire la precedente chiamata nel modo seguente:

```
Doll=34:Dollari (Doll)¶
```

Il risultato della conversione è lo stesso, ma Doll mantiene il suo vecchio valore.

Nota d'uso 5: i sistemi di numerazione di Amiga

Numeri Decimali
Prima di iniziare a scrivere i propri programmi è necessario prendere confidenza con alcune funzioni interne di Amiga. Negli esempi di conversione precedente sono stati utilizzati sempre numeri decimali. In una chiamata di sottoprogramma è comparso il numero 34. Ci si potrebbe domandare se esista qualche differenza tra il numero 34 ed il numero 34.00.

Si consideri un sottoprogramma come il seguente:

```
SUB Dollari (Valore) STATIC¶
    PRINT VALORE "Dollari corrispondono a "¶
     Valore = Valore * 13.50¶
     PRINT Valore" Lire"¶
END SUB¶
```

Se si richiama tale sottoprogramma con la seguente istruzione:

```
Dollari 34¶
```

Amiga visualizza il messaggio d'errore "Type Mismatch". Il tipo di numero utilizzato nella chiamata non è infatti dello stesso tipo di quello elaborato dal sottoprogramma.

Numeri in virgola mobile (Floating Point)
Il problema è il seguente: il numero 34 è un numero intero, ma Valore non è un numero intero, è una variabile in *virgola mobile.*

Per comprendere il significato di questa frase è necessario analizzare un attimo come Amiga gestisce i numeri. Si è già potuto vedere che esistono differenti tipi di numeri, ad esempio gli interi a 16 ed a 32 bit.

Le variabili in virgola mobile, utilizzate abbastanza frequentemente, sono quelle i cui nomi non presentano un carattere identificativo speciale ($,%) alla fine del loro nome. Ad esempio a, Ciao, Colori sono variabili in virgola mobile. Il termine virgola mobile specifica il fatto che il punto decimale non si trova in una posizione fissata.

Ecco alcuni esempi di numeri in virgola mobile:

100.23
3.1415593
1.3
.143
4.1165

In essi il punto decimale non è in una posizione rigidamente definita. I numeri in virgola mobile possono avere fino a sette cifre decimali. Se le cifre decimali sono in numero maggiore, si ha una diminuzione della precisione.

• Provare ad inserire la seguente linea:

```
a=0.2345475776443: ? a¶
```

Il risultato porterà alla visualizzazione del numero .2345476. Le eventuali cifre dopo la settima vengono infatti troncate ed il numero viene arrotondato. Questo si verifica per numeri piccoli. Ci si potrebbe ora chiedere come si comporti AmigaBASIC con numeri più grandi. Ad esempio nel caso seguente:

```
a=34264782364873674 89: ? a¶
```

Il risultato è:

```
3.426478E+18¶
```

Notazione scientifica

Si dovrebbe avere una certa familiarità con il tipo di *notazione scientifica* utilizzata per il numero precedente: si tratta infatti della notazione esponenziale. Il suo nome deriva dal fatto che gli scienziati, specialmente matematici e fisici, utilizzano numeri molto grandi. Di conseguenza essi

hanno sviluppato un metodo alternativo per rappresentarli:

```
3.426478E+18 = 3.426478 * 10 ^ 18
(dove ^ indica l'elevamento a potenza)
```

Ulteriori note sul sistema binario

Nella nota d'uso 3 si è visto come il sistema di numerazione binario utilizzi come base per la rappresentazione dei numeri il numero 2. Ogni esponente di 2 porta ad un aumento del numero delle cifre utilizzate: 2^0=1 (in binario 1), 2^1=2 (in binario 10), 2^2=4 (in binario 100), etc. Il sistema decimale funziona nel medesimo modo, ad eccezione del fatto che la base per la rappresentazione è ora il numero 10: 10^0=1, 10^1=10, 10^2=100, 10^3=1000, etc. Ogni esponente di 10 porta ad un aumento del numero delle cifre. Il numero 10^18 equivale ad un trilione, cioè alla cifra 1 seguita da 18 zeri. Il numero relativo all'esponente stabilisce il numero di zeri che seguono le cifre non nulle.

```
1.0E+3 è 1.0*10^3
3.4E*2 è 3.4*10^2
```

AmigaBASIC è in grado di utilizzare la notazione esponenziale anche per numeri piccoli. Ad esempio nel caso seguente:

```
a=1/202: ? a¶
```

il risultato è 4.950495E-03 (cioè 4.950495*10^(-3). 10^(-3) corrisponde a 1/10^3. In notazione normale il numero risulta essere: 0.004950495.

Computer e numeri in virgola mobile

Il risultato dell'esempio precedente non potrebbe essere gestito correttamente da Amiga. Tutti i computer hanno qualche difficoltà nella gestione dei numeri in virgola mobile e se si vuole sapere come vengono gestiti i numeri in virgola mobile consultare l'appendice D del manuale BASIC fornito con Amiga.

Dato che Amiga deve convertire i numeri in virgola mobile in numeri binari, potrebbero verificarsi alcuni errori matematici gestiti dalla maggior parte dei computer come errori di arrotondamento.

• Provare ad inserire la linea seguente:

```
? 100.1 - 100¶
```

Invece del valore corretto 0.1, viene visualizzato in risposta il numero 0.99. Non è molto diverso, ma non è corretto.

Precisione

Gli esempi mostrati fino ad ora sono stati realizzati con numeri in *singola precisione*. La precisione dipende dalla gestione interna del numero: i numeri in singola precisione hanno 7 cifre significative per i decimali. Il loro valore risulta compreso tra 10E-38 e 10E+37. E' possibile rappresentare i numeri anche in *doppia precisione*: si hanno a disposizione 16 cifre significative ed ogni valore possibile risulta compreso tra 10E-308 e 10E+307.

Normalmente AmigaBASIC utilizza numeri in singola precisione per i numeri in virgola mobile. L'identificatore per i numeri in doppia precisione è il carattere # posto di seguito al nome della variabile.

Nell'esempio seguente:

```
a=1/202: ? a¶    (risultato 4.950495E-03)

a#=1/202 : ? a#¶   (risultato 4.950494971126338D-03)
```

Il secondo caso risulta più preciso presentando un numero di decimali superiore. Il carattere D che rimpiazza il carattere E specifica il fatto che il numero è stato trattato in doppia precisione. 10E-03 e 10D-03 hanno lo stesso identico significato, e cioè 10^(-3).

Necessità di una maggior precisione

Ci si potrebbe chiedere il motivo per cui sia necessario utilizzare numeri in doppia precisione. Il grado di precisione richiesto dipende dal tipo di calcolo effettuato. Quando risulta possibile si dovrebbe evitare l'utilizzo di numeri in doppia precisione, in quanto si hanno alcuni svantaggi: il calcolo richiede un tempo maggiore e la quantità di memoria necessaria per memorizzare il numero è superiore.

Interi

I numeri *interi* sono già stati presentati: variabili come Ciao%, a%, Colori% possono avere valori compresi tra -32768 e 32767; in altre parole sono interi a 16 bit con segno. L'intervallo possibile per questi valori si rivela più che adeguato nella maggior parte dei casi (coordinate, colori, velocità, etc.). Quando un numero non risulta compreso in questo intervallo AmigaBASIC utilizza un numero a 32 bit. In questo caso possono essere gestiti valori tra -2147483648 e 2147483647 (anche se numeri di questo tipo verranno utilizzati di rado in BASIC).

Quando una variabile deve essere un intero a 32 bit, è necessario specificare questa richiesta utilizzando il carattere & di seguito al suo nome.

Si considerino i due esempi seguenti:

```
a=100000¶

a&=100000¶
```

Nel primo caso si ottiene il messaggio di errore "Overflow Error", in quanto il numero è troppo grande per un intero a 16 bit. Nel secondo caso non vi è invece alcun problema.

Si sarà probabilmente notato come AmigaBASIC visualizzi i numeri a 32 bit nella finestra LIST seguiti dal carattere &. Accade ad esempio questo con il numero 65535 che compare nel programma di disegno e che non può essere rappresentato da un intero a 16 bit (in quanto il primo bit è utilizzato per il segno ed il suo valore non è rappresentabile con i restanti 15 bit). AmigaBASIC utilizza quindi un numero a 32 bit, evidenziando tale scelta nella finestra LIST con il carattere &.

Stringhe

L'ultimo tipo di variabile che è necessario trattare è la *variabile* di tipo *stringa*. Queste variabili, che possono contenere stringhe di testo con lunghezza massima di 32767 caratteri, sono identificate dal suffisso $.

La tabella seguente mostra un riassunto di quanto presentato in questa nota.

Tipo Variabile	Identificatore	Memoria	Esempio
Floating Point (Singola Prec.)	Nessuno o !	4 byte	.3245643 2.43E+09
Floating Point (Doppia Prec.)	#	8 byte	4.9019690957795 382D-03
Interi Short	%	2 byte= 16 bit	32767
Interi Long	&	4 byte= 32 bit	-2147483648
Stringhe	$	5 byte + lung. Str	"Enrico"

Tabella 9: Tipi di variabile in AmigaBASIC.

I vari identificatori consentono di utilizzare variabili con lo stesso nome in modo completamente indipendente. A e A$, infatti, non sono assolutamente correlate tra loro. E' necessario però ricordare una cosa: i numeri in virgola mobile in singola precisione non hanno nessun tipo di carattere identificatore (sono infatti le variabili di base utilizzate da AmigaBASIC), per cui, se casualmente si aggiunge al nome di una di esse un carattere come !, si ha uno spiacevole inconveniente: Ciao e Ciao! sono la stessa identica variabile.

Dopo tutto questo discorso, rimane ancora da spiegare il motivo per cui Amiga non accetta il valore 34 nella chiamata SUB Dollari: Amiga si aspetta un numero in virgola mobile, ma 34 non è un valore di questo tipo.

L'assegnamento Valore=34 risolve il problema. Quando AmigaBASIC incontra un numero intero in input, lo tratta come numero intero. Tale numero intero potrebbe poi dover essere trattato come numero in virgola

mobile all'interno di un sottoprogramma. E' allora necessario forzare una conversione di tipo.

Un'altra possibile soluzione del problema è quella di richiamare il sottoprogramma aggiungendo un punto decimale dopo il valore:

```
Dollari 34.¶
```

Amiga suppone che dopo il punto decimale si trovino solamente degli zeri, ma il valore viene trattato come numero in virgola mobile. Altra possibile soluzione è quella di aggiungere un identificatore per forzare una particolare precisione:

```
Dollari 34!¶
```

In questo caso AmigaBASIC interpreta il numero come valore in virgola mobile in singola precisione.

Ecco alcuni ulteriori esempi:

3.4% risulta essere 3 (data la presenza dell'identificatore %). Questo numero, che normalmente verrebbe memorizzato come un valore in virgola mobile, viene forzato ad essere un intero a 16 bit. Viene quindi perso il valore dopo il punto decimale.

0.4& risulta essere 0. La gestione è la stessa del caso precedente, anche se questa volta viene trattato come intero a 32 bit (data la presenza dell'identificatore &).

Passando da singola a doppia precisione il numero non subisce modifiche, ma cambia comunque la sua rappresentazione interna.

Da questo punto di vista dovrebbe apparire chiara la differenza tra:

```
? 1/3¶
```

e

```
? 1# /3¶
```

Ecco un esempio di utilizzo di nozioni riguardanti i sistemi di numerazio-

ne tratto dal programma di videotitolazione:

```
SUB Pausa (Sec%) STATIC¶
Pausa:¶
   TIM=INT(TIMER)¶
   WHILE INT(TIMER)=TIM : WEND¶
   Sec%=Sec%-1¶
   IF Sec%>0 THEN Pausa¶
END SUB
```

Questo sottoprogramma consente di effettuare una pausa di alcuni secondi. La variabile Sec% aspetta l'inserimento di un valore intero. In tal modo, richiamando il sottoprogramma con l'istruzione seguente:

```
Pausa 10¶
```

si ottiene una pausa di 10 secondi. Il sottoprogramma deve essere richiamato passando come parametri valori interi; in caso contrario si ottiene il messaggio di errore "Type Mismatch".

Nel paragrafo seguente le informazioni esposte in questa nota verranno utilizzate per scrivere la routine per il salvataggio di immagini IFF.

4.8 Salvare le immagini: il sottoprogramma PicSave

Il seguente sottoprogramma definisce un comando per il salvataggio delle immagini su dischetto.

```
SUB PicSave (Nome$,FinestraNum%,MatriceSN%) STATIC¶
  IF MatriceSN%=1 THEN SHARED Colori%()¶
  IF MatriceSN%=0 THEN¶
    IF Colori%(0,0)<>2 THEN ERASE Colori% :
         DIM Colori%(31,2)¶
    RESTORE TabellaColore¶
    FOR x=0 TO 31¶
      READ Colori%(x,0),Colori%(x,1),Colori%(x,2)¶
    NEXT x¶
  TabellaColore:  ¶
    DATA 2,3,10, 15,15,15, 0,0,0, 15,8,0¶
    DATA 0,0,15, 15,0,15, 0,15,15, 15,15,15¶
    DATA 6,1,1, 14,5,0, 8,15,0, 14,11,0¶
    DATA 5,5,15, 9,0,15, 0,15,9, 12,12,12¶
    DATA 0,0,0, 13,0,0, 0,0,0, 15,12,10¶
    DATA 4,4,4, 5,5,5, 6,6,6, 7,7,7¶
    DATA 8,8,8, 9,9,9, 10,10,10, 11,11,11¶
    DATA 12,12,12, 13,13,13, 14,14,14, 15,15,15¶
  END IF¶
  IF Nome$="" THEN EXIT SUB¶
  AltraFinestraNum=WINDOW(1)¶
  WINDOW FinestraNum%¶
  Larghezza=WINDOW(2)¶
    IF Larghezza>320 THEN¶
      Larghezza=640¶
      Risoluzione=2¶
```

```
      Piani=16000¶
    ELSE¶
      Larghezza=320¶
      Risoluzione=1¶
      Piani=8000¶
    END IF¶
  Altezza=WINDOW(3)¶
    IF Altezza>200 THEN¶
      Altezza=400¶
      Piani=Piani*2¶
      Risoluzione=Risoluzione+2¶
    ELSE¶
      Altezza=200¶
    END IF¶
  Colori=LOG(WINDOW(6)+1)/LOG(2)¶
¶
  OPEN Nome$ FOR OUTPUT AS 1 LEN=FRE(0)-500¶
    PRINT #1,"FORM";¶
    PRINT #1,MKL$(156+Piani*Colori);¶
    PRINT #1,"ILBM";¶
    PRINT #1,"BMHD";MKL$(20);¶
    PRINT #1,MKI$(Larghezza);MKI$(Altezza);¶
    PRINT #1,MKL$(0);¶
    PRINT #1,CHR$(Colori);¶
    PRINT #1,CHR$(0);MKI$(0);MKI$(0);¶
    PRINT #1,CHR$(10);CHR$(11);¶
    PRINT #1,MKI$(Larghezza);MKI$(Altezza);¶
    ¶
    PRINT #1,"CMAP";MKL$(96); ¶
    FOR x=0 TO 31¶
      PRINT #1,CHR$(Colori%(x,0)*16);¶
      PRINT #1,CHR$(Colori%(x,1)*16);¶
      PRINT #1,CHR$(Colori%(x,2)*16);¶
    NEXT x¶
    ¶
    PRINT #1,"BODY";MKL$(Piani*Colori);¶
```

```
    Locazione=PEEKL(WINDOW(8)+4)+8¶
    FOR x=0 TO Colori-1¶
      LocazionePiano(x)=PEEKL(Locazione+4*x)¶
    NEXT x¶
    FOR y1=0 TO Altezza-1¶
      FOR b=0 TO Colori-1¶
        FOR x1=0 TO (Larghezza/32)-1 ¶
          PRINT#1,MKL$(PEEKL(LocazionePiano(b)+
                4*x1+(Larghezza/8)*y1));¶
        NEXT x1¶
      NEXT b¶
      PLocazione=LocazionePiano(0)+(Larghezza/8)*y1¶
      POKE PLocazione,PEEK(PLocazione) AND 63¶
      POKE PLocazione+Larghezza/8-1,PEEK(PLocazione+
           Larghezza/8-1)AND 252¶
    NEXT y1¶
    ¶
    PRINT #1,"CAMG";MKL$(4);¶
    PRINT #1,MKL$(16384);¶
  CLOSE 1¶
  WINDOW AltraFinestraNum ¶
END SUB¶
```

- Effettuare il salvataggio del programma in formato ASCII, utilizzando l'istruzione:

```
SAVE "PicSave",a¶
```

- Utilizzare il comando MERGE per includere il sottoprogramma in un qualsiasi altro programma, dopo essersi assicurati che il programma a cui si sta concatenando tale sottoprogramma corrisponde ad un programma salvato in formato ASCII.

Vi sono alcune cose all'interno di questo sottoprogramma di comprensione non immediata.

Il sottoprogramma, il cui nome è riportato tra le parole chiave SUB ... STATIC, è stato chiamato PICSAVE come abbreviazione di PICture SAVE. I

parametri sono specificati subito di seguito al nome, racchiusi tra parentesi. Come si può notare, è possibile passare più di un parametro ad un sottoprogramma. La sintassi per richiamare PICSAVE risulta essere la seguente:

```
PICSAVE "(nome_file)", (numero_finestra),
     (vettore_per_i parametri_dei_colori)¶
```

Per quanto riguarda il nome del file, è possibile specificare un nome qualsiasi.

Il secondo parametro è relativo alla finestra in cui è contenuta l'immagine da salvare.

L'ultimo parametro può valere 0 oppure 1. Nel caso valga 1, significa che si desidera utilizzare il vettore per i colori definito nel sottoprogramma (oppure nel programma di disegno) contenente le componenti per le tre bande; nel caso venga invece passato il valore 0 significa che si intendono utilizzare i colori standard del Workbench.

L'ultimo dei tre parametri, MatriceSN%; viene utilizzato nel medesimo modo nelle due successive linee: se vale 1, la matrice per i colori Colori% viene definita in comune tra il programma ed il sottoprogramma: se nel programma principale non esiste la matrice Colori%, si otterrà il messaggio d'errore "Undefined Array".

Se MatriceSN% vale 0 si devono effettuare alcune azioni. Per prima cosa il sottoprogramma controlla il contenuto del primo elemento della matrice Colori%: se Colori%(0,0) vale 2, non è necessario procedere al ridimensionamento della matrice.

Autogenerazione di vettori

Se invece si desidera ridimensionare la matrice, che cosa succede? Quando viene utilizzato un vettore senza che questo sia stato precedentemente dimensionato, AmigaBASIC stabilisce automaticamente una dimensione standard per tale vettore: 10 elementi.

Ad esempio, inserendo nella finestra BASIC la linea:

```
? VettoreProva(1)¶
```

AmigaBASIC riconosce che tale vettore non è ancora stato dimensionato ed inserisce automaticamente un'istruzione di dimensionamento: DIM VettoreProva(10). E' ora possibile utilizzare gli elementi di VettoreProva fino a VettoreProva(10). Se si prova ad utilizzare:

```
VettoreProva(11)¶
```

si ottiene il messaggio d'errore "Subscript out of Range" in quanto VettoreProva è stato dimensionato a 10 elementi. La stessa cosa si verifica per i vettori bidimensionali.

```
? AltroVettoreProva(1,1)¶
```

dimensiona automaticamente AltroVettoreProva come matrice 10 x 10. La stessa cosa accade per la matrice Colori%(0,0): se tale matrice non è stata ancora dimensionata, viene automaticamente dimensionata a 10 x 10.

ERASE

Il comando ERASE consente di eliminare un vettore specificandone il nome di seguito al comando. Il vettore ed il suo contenuto vengono cancellati ed è successivamente possibile utilizzare il comando DIM per ricreare e ridimensionare tale vettore.

Come precedentemente rilevato non esiste un comando che consenta di acquisire direttamente tutti i valori per i colori; è comunque possibile utilizzare dei comandi DATA per mantenere all'interno del sottoprogramma dei valori base per le varie componenti RGB.

RESTORE

Dopo aver caricato la matrice dei dati, viene utilizzato il comando RESTORE che consente di posizionare il puntatore per il prossimo comando DATA da leggere. In questo modo è possibile assicurarsi che il sottoprogramma legga i dati correttamente.

Il ciclo FOR ... NEXT che segue, carica i valori per le componenti RGB nella matrice Colori%.

EXIT SUB
Se Nome$ risulta essere una stringa vuota, la routine deve essere interrotta. Il comando EXIT SUB abilita il ritorno da un sottoprogramma. AltraFinestraNum contiene il numero della finestra attualmente attiva, ottenuto con il comando WINDOW(1). Dopo il ritorno al programma principale, è necessario riportare ogni cosa nella condizione originale. Per richiamare alcuni dati importanti è necessario rendere attiva la finestra AltraFinestraNum.

E' necessario ora discutere alcuni nuovi comandi riguardanti le finestre: WINDOW(2) fornisce la larghezza della finestra corrente, valore che viene poi assegnato alla variabile Larghezza. Dato che si vuole che la finestra occupi tutto lo schermo, viene utilizzato un blocco IF/THEN/ELSE/END IF per determinare se la dimensione dello schermo sia di 320 o 640 pixel in larghezza. Tale dimensione dipende dalla risoluzione scelta (1 per 320, 2 per 640) e dalla memoria richiesta per ogni singolo bitplane. Il comando WINDOW(3) viene utilizzato per determinare l'altezza della finestra.

LOG
Per finire, viene determinato il numero dei bitplane. Il comando WINDOW(6) fornisce il numero dei colori disponibili all'interno della finestra. Dato che, noto il numero di bitplane, è possibile determinare il numero di colori utilizzando la formula Colori=(2^Bitplane), è possibile determinare il numero di bitplane, noto il numero dei colori, utilizzando la formula inversa.

A questo scopo viene utilizzata la funzione LOG. Il nome LOG deriva dal termine logaritmo. Come forse si può ricordare dalla matematica studiata a scuola, se 2^x=8, è possibile determinare il valore di x ricorrendo alla formula x=LOG(8)/LOG(2).

A questo punto tutti i preparativi sono stati completati e la routine per il salvataggio di immagini IFF è stata scritta e spiegata completamente. Un piccolo commento: per il Chunk FORM sono necessari 156+Piani x colori byte. 56 è la lunghezza del file senza BODY, mentre BODY ha lunghezza (byte per Piani) x (numero di Piani).

Dopo il comando CLOSE viene riattivata la finestra il cui numero è memorizzato in AltraFinestraNum ed il sottoprogramma termina.

Richiamare PICSAVE

Si è in questo modo realizzato il comando PICSAVE che può ora esere richiamato da un qualsiasi programma. Nella routine sono gestiti il livello di risoluzione, la dimensione dell'immagine ed il numero di bitplane. Per salvare il contenuto della finestra BASIC è possibile richiamare il sottoprogramma nel modo seguente:

```
PISCAVE "BASICwindow",1,0¶
```

Il file IFF creato risulterà poi leggibile con la routine di lettura sviluppata nel paragrafo 4.3.

E' possibile utilizzare questo sottoprogramma anche con il programma per la realizzazione di istogrammi e grafici a torta.

- Caricare il programma per la realizzazione di istogrammi.

- Aggiungere la routine PICSAVE

- Visualizzare la subroutine Inizializzazione ed inserire la seguente linea:

```
MENU 2,3,1,"Salva Figura"¶
```

- Inserire, sotto la linea:

```
IF Men=2 THEN¶
```

della routine ControlloMenu, le seguenti linee:

```
IF SceltaMenu=3 THEN¶
 MENU 1,0,0: MENU 2,0,0¶
 MENU OFF¶
   GOSUB ImmettiNome¶
 WINDOW 99¶
   PicSave Nome$,99,0¶
 WINDOW 1¶
 MENU ON¶
 MENU 1,0,1 : MENU 2,0,1¶
END IF¶
```

A questo punto l'opzione SalvaFigura compare nel menù a tendina. Quan-

do viene selezionata, il contenuto corrente della finestra Grafici viene salvato su dischetto. E' ovviamente necessario creare prima un grafico, per non salvare inutilmente una finestra vuota.

5 Tutte le informazioni riguardo ai dati

In questo capitolo non si illustrerà tutto quanto si può desiderare di sapere riguardo ai dati. Sono già stati analizzati alcuni dei comandi BASIC utilizzati per la gestione dei file, in particolare dei file sequenziali. Finora, non si sono tuttavia ancora presi in considerazione i file ad accesso casuale (Random access file). Le pagine seguenti sono focalizzate proprio su questo argomento.

Dopo aver letto questo capitolo ed aver inserito le linee di programma presentate, si sarà entrati in possesso di un versatile programma per la archiviazione dei dati.

5.1 Tutto è relativo: utilizzo dei file ad accesso casuale

A questo punto si dovrebbe aver acquisito una certa familiarità nella gestione dei file sequenziali. Tuttavia, nel caso lo si fosse dimenticato, sequenziale significa in sequenza e questo implica che i dati vengono memorizzati uno di seguito all'altro. I file sequenziali risultano ottimali per alcuni specifici campi di applicazione, ma essi si dimostrano carenti per applicazioni più avanzate.

File sequenziali

Tutti i file inviati ad una stampante sono file sequenziali. Questo ha senso poichè una stampante stampa un carattere di seguito all'altro. I file ricevuti in input dalla tastiera od inviati in output allo schermo sono sequenziali, come pure i file IFF.

I file sequenziali rappresentano dunque la scelta di programmazione più appropriata nella gestione di compiti di questo tipo. Tuttavia, quando si desidera leggere o scrivere una lista di indirizzi, una collezione di record, oppure altre informazioni, lo svantaggio maggiore dei file sequenziali è che essi memorizzano i dati in sequenza sul dischetto. E lo stesso metodo deve essere utilizzato anche per caricare i dati. Questo va bene fino a quando si desidera ritrovare, da un file contenente 1000 numeri di telefono alfabeticamente ordinati, l'indirizzo di Mrs. Andover; ma se è necessario cercare il numero di Mr. Tiemstra, si deve ogni volta esaminare tutto il file, considerando anche i numeri di Mr. Fisher, di Mrs. Lloyd, del Dr. Taber, etc. E questo implica un certo tempo di attesa durante il quale il sistema è impegnato nell'analisi del file.

File Random

Di conseguenza, AmigaBASIC fornisce un secondo metodo per la gestione dei file utilizzando i *file random*, cioè i *file ad accesso casuale.* Accesso casuale significa che ogni singolo elemento del file può essere reperito ve-

locemente e facilmente. Un'altra valida definizione di random file è quella di file in cui ogni singolo accesso può essere effettuato in relazione al primo accesso. Questo è il motivo per cui si definisce un file random anche come file relativo.

Nome, via, città, telefono, stato e CAP, sono visti come *campi* di un record che, in questo caso, formano un indirizzo. Si ha un indirizzo per ognuno dei nominativi contenuti nel file. Ogni *record* è costituito da una collezione di dati connessi tra loro.

Un file ad accesso casuale assegna ad ogni record un numero di identificazione, consentendo quindi un facile e veloce accesso ad ogni elemento, tanto in lettura quanto in scrittura. E' possibile accedere al file indipendentemente dalla sua posizione sul disco o dalla sua dimensione.

La creazione di un random file richiede semplicemente qualche comando in più rispetto a quella di un normale file sequenziale. E' veramente abbastanza semplice.

Si inizierà ad analizzare la questione con un semplice esempio.

• Inserire le seguenti linee di seguito al programma per agenda:

```
OPEN "r",#1, "FileIndirizzi.rel",92
FIELD #1,30 AS Nome$,30 AS Indirizzo$,
   20 AS Citta$,12 As Telefono$¶
¶
Immissione:¶
    PRINT¶
    INPUT "Nome";Nomeimmesso$¶
    INPUT "Indirizzo";Indirizzoimmesso$¶
    INPUT "Citta";Cittaimmessa$¶
    INPUT "Telefono";Telefonoimmesso$¶
    LSET Nome$=Nomeimmesso$¶
    LSET Indirizzo$=Indirizzoimmesso$¶
    LSET Citta$=Cittaimmessa$¶
    LSET Telefono$=Telefonoimmesso$¶
     x=x+1¶
```

```
        PUT #1,x¶
        PRINT "Record "x" ("Nome$") Memorizzato."¶
        PRINT "Altri Record?"¶
        INPUT "Y/N:";Ans$¶
        IF UCASE$(Ans$)="Y" THEN Immissione¶
¶
CLOSE 1¶
PRINT "File Chiuso, Programma Finito."¶
```

• Salvare il programma prima di eseguirlo.

Il comando OPEN per i File Random

Esaminando con attenzione il programma si nota come per il comando OPEN non sia stata utilizzata la stessa sintassi usata precedentemente. OPEN possiede una seconda sintassi, utilizzata per i file ad accesso casuale. Essa prevede una unica modalità per le operazioni con i file (non è prevista una sintassi separata per lettura, scrittura e concatenamento). Il carattere r minuscolo identifica il fatto che il file sia Random. Seguono il numero ed il nome del file. L'ultimo parametro fornisce la lunghezza del singolo record del file ad accesso casuale. Il numero nel programma riportato sopra stabilisce come lunghezza totale per il record contenente i vari indirizzi 92 byte.

I file sequenziali consentono di specificare la lunghezza del file buffer. Non è necessario effettuare un'operazione analoga per gestire un file random. AmigaBASIC dimensiona opportunamente il buffer in funzione della dimensione dei record.

FIELD

Il comando FIELD è il primo nuovo comando contenuto in questo programma. Esso consente di dividere in singoli campi il record. Ad ognuno di questi campi viene assegnato un parametro variabile.

```
FIELD «#» (filenumber), (number of byte)  AS
(variable name), ....
```

Questo esempio utilizza la variabile Nome$ per identificare i 30 byte disponibili. Il contenuto del campo Nome$ non deve superare i 30 byte per

ogni record del file, ma esso può anche essere più corto. Con i valori specificati nel comando Field si stabiliscono la massima lunghezza di una variabile esterna al file.

Questo vale anche per le restanti variabili: Indirizzo$ può essere lunga fino a 30 caratteri, Citta$ non può superare i 20, mentre la lunghezza massima per Telefono$ è fissata in 12 caratteri. Ogni carattere in più che viene inserito risulterà perso.

Gli indirizzi vengono letti con la routine che segue Immissione:. In essa sono utilizzate altre variabili per l'input, e precisamente le variabili NomeImmesso$, IndirizzoImmesso$, CittaImmessa$ e TelefonoImmesso$. Come si può vedere, è possibile utilizzare i nomi interi, ma ad una condizione: essi devono essere nomi differenti da quelli utilizzati nella istruzione in cui compare il comando FIELD, in modo tale che il successivo comando LSET possa funzionare correttamente.

LSET

Le variabili definite con FIELD rappresentano variabili parametro. Nome$ è il nome identificativo di un'area lunga 30 byte nel buffer dati. Il comando LSET scarica i dati in questa area buffer. Nell'esempio considerato, i dati sono quelli contenuti nella variabile NomeImmesso$. Nell'area identificata da Indirizzo$ vengono trasferiti i dati contenuti in IndirizzoImmesso$. E lo stesso meccanismo vale per Citta$ e Telefono$.

Se il contenuto della variabile assegnata risulta essere più lungo dell'area di buffer corrispondente, la parte in eccesso viene troncata; se risulta più corto della dimensione del buffer, la parte rimanente viene riempita con degli spazi bianchi.

Le variabili definite con il comando FIELD hanno una funzione ulteriore: è possibile assegnare loro una quantità che deve essere trasferita nell'area buffer utilizzando LSET. Leggendo il contenuto di una variabile parametro, si accede al contenuto dell'area di buffer identificata da quel nome, secondo quanto stabilito dal comando FIELD.

• Si osservi attentamente la linea seguente:

```
PRINT Memorizzare Record"x" ; "Nome$
```

Nome$ contiene i primi 30 byte dell'area buffer. Nel momento in cui viene assegnato un valore a Nome$ senza utilizzare il comando LSET (utilizzando il comando INPUT oppure mediante il normale assegnamento Nome$=), l'assegnamento del buffer risulta modificato. I 30 byte del buffer non sono più accessibili. Tuttavia, il dato in input viene trasferito nella variabile NomeImmesso$.

Quanto detto vale ovviamente anche per tutte le altre variabili parametro del buffer dati.

La sintassi del comando LSET risulta essere la seguente:

```
LSET (variabile parametro)=(variabile)
```

Il comando PUT per i File Random

I dati sono stati così trasferiti nel buffer di memoria. E' necessario utilizzare un comando PUT per trasferire i dati sul dischetto. Si è gia visto il comando PUT durante lo sviluppo del progetto grafico analizzato nel capitolo 4, ma quello usato in quell'occasione era il comando PUT per lo *schermo*. Quella forma rappresentava una delle due possibil forme del comando PUT; l'altra forma possibile è quella per la gestione dei file ad accesso casuale.

Nella sintassi corretta per questa forma, il numero di identificazione del file deve seguire il comando PUT e successivamente deve essere specificato il numero del record in cui scrivere i valori contenuti nel buffer dati. Poichè, nell'esempio in questione, ad ogni ciclo di scrittura, il numero del record viene incrementato di una unità, nessun record risulta essere inutilizzato. Teoricamente, è possibile utilizzare, per identificare un record, un numero compreso tra 1 e 16777215. Ma prima di provare con un numero così alto, è bene fare una piccola precisazione: 16777215 è teoricamente il numero massimo di record che AmigaBASIC può gestire. Il numero massimo di record realmente utilizzabile dipende dalla quantità di memoria disponibile e dalla dimensione dei singoli record. In ogni caso, un file così grande riempirebbe quasi completamente un disco rigido da 20 Mbyte, per cui ..... Un dischetto completamente vuoto da 3" 1/2 per Amiga consente di mantenere circa 10000 record per il file di indirizzi.

Si è a questo punto conclusa l'analisi dei nuovi comandi nella parte di scrittura del file. La parte restante del programma non presenta alcuna novità.

Esaminare il file

- Provare a salvare un decina di indirizzi utilizzando il programma visto. Successivamente inserire le seguenti linee di programma per apprendere alcune nozioni relative all'analisi dei file random.

```
OPEN "r",#1,"File indirizzi.rel",92¶
FIELD  #1,30 AS Nome$,30 AS Indirizzo$,
            20 AS Citta, 20 AS Telefono$¶
¶
LetturaDati:¶
    INPUT "Indirizzo numero",Num¶
    GET #1,Num¶
    IF EOF(1) THEN PRINT "Record non appartenente al
          file.": GOTO LetturaDati¶
    PRINT Nome$¶
    PRINT Indirizzo$¶
    PRINT Citta$¶
    PRINT Telefono$¶
    INPUT "Nuovo record (Y/N)";ans$¶
    IF UCASE$(ans$)<>"N" THEN LetturaDati¶
CLOSE 1¶
PRINT "File chiuso, Programma finito"¶
```

Le linee di programma contenenti i comandi sono identiche a quelle viste nel programma per la scrittura dei file random. Viene aperto il file e viene definito il buffer di trasferimento. Il numero dei buffer e la lunghezza dei record utilizzati per il caricamento devono essere identici ai valori usati durante la scrittura del file.

Programmi che gestiscono file di grosse dimensioni utilizzano altrettanto bene file sequenziali e file random. I file sequenziali contengono le necessarie informazioni relative alla lunghezza dei record per l'utilizzo dei file random.

Poichè non è necessario fornire la modalità (lettura o scrittura) di accesso al file quando viene aperto un file random, è possibile passare dalla lettura alla scrittura quando lo si desidera. Questo è un altro vantaggio dei file ad accesso casuale.

Esecuzione del programma
Il programma per il caricamento del file di indirizzi richiede per prima cosa che venga specificato il numero del record che deve essere caricato. Supponendo che siano stati inseriti nel file di indirizzi 10 indirizzi, deve essere introdotto un numero compreso tra 1 e 10. Se sono stati inseriti nel file meno di dieci indirizzi, allora il numero massimo per il record sarà minore, il numero deve cioè corrispondere a quello degli indirizzi contenuti nel file.

Il comando GET per i File Random
Ecco ora il comando complementare del comando PUT: GET. Anche questo comando ha due possibili forme: quella per i file random e quella per lo schermo.

Il comando GET per i file random consente di acquisire un determinato record dal dischetto trasferendo nel buffer i suoi dati. E' necessario specificare il numero del file e del record desiderato. E' poi possibile leggere i contenuti del buffer utilizzando le variabili parametro giungendo in tal modo ai dati relativi agli indirizzi.

EOF
Il comando EOF svolge per i file ad accesso casuale lo stesso servizio svolto per i file sequenziali. Se si tenta di leggere un record non esistente (identificato da un numero superiore al numero dei record effettivamente esistenti), EOF(1) ritorna il valore -1. Questo fornisce un'idea di quanti record esistano. Se tale comando non viene incluso in un programma che utilizza i file random, è possibile ottenere strani dati nei buffer: sia frammenti che combinazioni casuali di diversi record, oppure un assortimento di caratteri casuali, per la maggior parte caratteri di controllo.

Visualizzazione dei dati
Il comando PRINT consente di visualizzare l'indirizzo sullo schermo. Come è già stato spiegato, è possibile leggere i dati di ogni singolo campo attraverso le variabili parametro utilizzando il comando GET:. Nome$, Indirizzo$, Citta$ e Telefono$ forniscono i contenuti del buffer utilizzato.

Il programma richiede poi se si desidera leggere altri dati oppure no. Pre-

mendo il tasto <N>, viene chiuso il file e terminato il programma. Nel caso venga premuto un qualsiasi altro tasto, il programma leggerà un nuovo record.

Un breve utilizzo di questo programma dimostrativo consente di notare i vantaggi dei file random. Si può, ad esempio, provare a leggere il primo indirizzo, poi il decimo, poi l'ottavo, il secondo e così via. Amiga ritrova i dati pressochè immediatamente.

Dopo aver analizzato completamente questo programma, verrà ora illustrato qualcosa di nuovo, ed allo stesso tempo verrà creato un utile programma basato sui file ad accesso casuale. Dopo queste operazioni si sarà veramente in grado di catalogare ed organizzare qualsiasi cosa, dalla propria collezione di francobolli al proprio portafogli finanziario.

5.2 Memorizzazione: un programma per la gestione di un DataBase

E' già stato scritto precedentemente un programma per la gestione di un DataBase. Il programma di analisi statistica realizzato consentiva di raccogliere in un file un certo insieme di dati. L'approccio utilizzato per la memorizzazione, di tipo sequenziale, si rivela più indicato quando si sta lavorando con grafici a torta ed istogrammi piuttosto che con indirizzi e collezioni di dischi. Inoltre, il programma di statistica realizzato non è in grado di gestire due o più insiemi di dati. Per questi motivi verrà sviluppato in questo paragrafo un programma più completo ed efficiente.

Efficiente significa che consenta le operazioni di salvataggio e modifica dei dati velocemente, facilmente, ed in ogni istante. Il DataBase realizzato potrà poi essere utilizzato per archiviare una collezione di dischi di musica classica oppure catalogare i compact disk inserendo i propri commenti sul disco e così via.

Ecco ora la descrizione del programma, iniziando dalle routine di preparazione al progetto.

```
Inizializzazione:¶
  PALETTE 0,0,.1,.4¶
  PALETTE 2,0,1,0¶
  ¶
Partenza:¶
  CLS : LOCATE 1,1 : PRINT "Selezionare"¶
  LOCATE 1,25 : COLOR 3,0 : PRINT "Nome File:"; :
   COLOR 1,0¶
  IF AltroNome$<>"" THEN PRINT AltroNome$ ELSE
     PRINT "(nessun file)"¶
  PRINT¶
  COLOR 0,3 : PRINT SPACE$(21)"AmigaBASIC DataBase"
    SPACE$(21)¶
```

```
    LOCATE 5,22 : COLOR 3,0¶
    PRINT "Opzioni disponibili:"¶
    LOCATE 7,22¶
    COLOR 0,1 : PRINT " 1 "; : COLOR 1,0 : PRINT
      " Creare un File"¶
    LOCATE 9,22¶
    COLOR 0,1 : PRINT " 2 "; : COLOR 1,0 : PRINT
      " Immettere Dati"¶
    LOCATE 11,22¶
    COLOR 0,1 : PRINT " 3 "; : COLOR 1,0 : PRINT
       "Leggere un File"¶
    LOCATE 13,22¶
    COLOR 0,1 : PRINT " 4 "; : COLOR 1,0 : PRINT
      " Effettuare Ricerche"¶
    LOCATE 15,22¶
    COLOR 0,1 :PRINT " 5 "; : COLOR 1,0 : PRINT " Fine"¶
  Scelta:¶
    LOCATE 18,1 : PRINT SPACE$(60)¶
    LOCATE 18,22 : COLOR 3,0 : PRINT
     "Immettere un Numero:";¶
    COLOR 1,0 : LINE INPUT Numero$¶
    Numero$=LEFT$(Numero$,1)¶
    IF Numero$<"1" OR Numero$>"5" THEN Scelta¶
    IF Numero$="1" THEN CreazioneFile¶
    IF Numero$="2" THEN ImmissioneDati¶
    IF Numero$="3" THEN RicercaDati=0 : GOTO
      LetturaDati¶
    IF Numero$="4" THEN RicercaDati=1 : GOTO
      LetturaDati¶
    PRINT "Programma concluso."¶
  END¶
```

Inizializzare il programma

Non vi sono molte nuove nozioni nella routine di Inizializzazione:. Per prima cosa vengono eseguiti due comandi PALETTE che scuriscono

leggermente lo sfondo blu, fornendo al programma un aspetto leggermente più professionale. Lo schermo risulta più semplice da leggere se non si utilizzano i colori standard del Workbench. Il secondo comando PALETTE sostituisce al nero del Workbench il verde, colore che contrasta maggiormente con il nuovo sfondo.

La routine Partenza: pulisce lo schermo e visualizza la parola Selezione nell'angolo in alto a sinistra dello schermo. In quest'angolo verra visualizzato il modo corrente di funzionamento del programma. In questo modo, l'utente è in grado di sapere, in ogni istante, dove questo messaggio viene visualizzato.

Nel mezzo della prima linea viene listato il file corrente. La prima volta che viene eseguito il programma non è selezionato alcun file, per cui risulta visualizzata il messaggio "(nessun file)".

Space$(x)
La linea seguente visualizza l'intestazione. In questa linea compare un nuovo comando, il comando Space$(x), il cui nome è in relazione con la barra di spaziatura (definita anche <SPACE>). SPACE(x) genera x spazi. Utilizzando questo comando è possibile visualizzare barre colorate, come per la barra titolo, semplicemente utilizzando un colore per il testo uguale a quello di sfondo, ad esempio, COLOR 0,3. Per la barra titolo è stato utilizzato il blu scuro come colore per il testo e l'arancione come colore di sfondo. SPACE$ può essere utilizzato anche per cancellare parti di testo, come sarà spiegato più avanti.

Successivamente viene presentato un menù dal quale è possibile selezionare l'opzione desiderata.

Creare un File, selezionata per creare un nuovo file, consente di specificare il nome del nuovo file e la lunghezza di ogni singolo campo.

Immettere Dati consente di inserire i dati.

Leggere un File fornisce la possibilità di spostarsi all'interno di un file e cambiare i dati dei singoli record.

Effettuare Ricerche consente di cercare i dati all'interno di un record secondo il criterio specificato.

Fine provoca la terminazione del programma.

Le linee di programma successive sono utilizzate per effettuare la selezione. In queste linee il comando SPACE$ viene utilizzato per cancellare parti di testo: ogni volta che l'utente inserisce qualcosa durante una selezione, la vecchia selezione viene cancellata e sostituita con la nuova.

Le singole routine del programma sono chiamate in dipendenza dal dato di input. Le opzioni Leggere un File ed Effettuare Ricerche utilizzano entrambe la routine LetturaDati. La variabile RicercaDati viene utilizzata per stabilire quando devono essere letti tutti i record e quando solamente uno.

```
CreazioneFile:¶
  CLS : LOCATE 1,1 : COLOR 1,0 : PRINT  "Creazione di
    un File"¶
  LOCATE 1,25 : COLOR 3,0 : PRINT "Nome del File:";¶
  COLOR 1,0 : PRINT "(nessun file)" ¶
  COLOR 3,0 : LOCATE 3,1¶
  PRINT "Immettere il nome del campo e le sue
    dimensioni."¶
  COLOR 1,0¶
  FOR x=0 TO 9¶
    NomeCampo$="" : Lunghezza(x)=0¶
  NEXT x¶
  LOCATE 4,1 : PRINT "Nome" : LOCATE 4,26 : PRINT
    "Lunghezza (<40)"¶
  FOR x=0 TO 9¶
    NumeroCampi=x¶
    LOCATE x+6,1 : LINE INPUT NomeCampo$(x)¶
    IF NomeCampo$(x)="" THEN x=10 : NumeroCampi=
       NumeroCampi-1¶
    NomeCampo$(x)=LEFT$(NomeCampo$(x),25)¶
    LOCATE x+6,26 : PRINT SPACE$(40);¶
    LOCATE x+6,26 : LINE INPUT Lunghezza$¶
    IF Lunghezza$="" OR ABS(VAL(Lunghezza$))>40 THEN
```

```
        Lunghezza$="40"¶
      Lunghezza(x)=INT(ABS(VAL(Lunghezza$)))¶
      IF Lunghezza(x)=0 THEN Lunghezza(x)=40¶
    NEXT x¶
¶
Correzioni:¶
    GOSUB ImmissioneOK¶
    IF Corr=0 THEN AperturaFile¶
    IF Corr=1 THEN CorrezioneErrore¶
    GOTO Correzioni¶
¶
CorrezioneErrore:¶
    FOR x=0 TO NumeroCampi¶
      LOCATE x+6,1 : PRINT SPACE$(60)¶
      LOCATE x+6,25 : PRINT Lunghezza(x)¶
      LOCATE x+6,1 : PRINT NomeCampo$(x)¶
    NEXT x¶
    FOR x=0 TO NumeroCampi¶
      LOCATE x+6,1 : LINE INPUT NomeCampo$¶
      IF NomeCampo$<>"" THEN NomeCampo$(x)=
         LEFT$(NomeCampo$,25)¶
      LOCATE x+6,26 : LINE INPUT Lunghezza$¶
      IF ABS(VAL(Lunghezza$))>40 THEN Lunghezza$="40"¶
        IF Lunghezza$<>"" THEN Lunghezza(x)=
          INT(ABS(VAL(Lunghezza$)))¶
      IF Lunghezza(x)=0 THEN Lunghezza(x)=40¶
    NEXT x¶
    GOTO Correzioni¶
¶
AperturaFile:¶
    LOCATE 19,1 : PRINT SPACE$(60);¶
    LOCATE 19,1 : COLOR 3,0 : PRINT "Immettere
       Nome File:";¶
    COLOR 1,0 : LINE INPUT Nome$¶
    LunghezzaRecord=0¶
    FOR x=0 TO NumeroCampi¶
```

```
    LunghezzaRecord=LunghezzaRecord+Lunghezza(x)¶
  NEXT x¶
  IF Nome$="" OR LunghezzaRecord=0 THEN BEEP :
     GOTO Partenza¶
  OPEN "R",#1,Nome$,LunghezzaRecord¶
     FIELD #1,Lunghezza(0) AS Dat$(0),Lunghezza(1) AS
Dat$(1),Lunghezza(2)  AS  Dat$(2),Lunghezza(3)  AS
Dat$(3),Lunghezza(4)  AS  Dat$(4),Lunghezza(5)  AS
Dat$(5),Lunghezza(6)  AS  Dat$(6),Lunghezza(7)  AS
Dat$(7),Lunghezza(8)  AS  Dat$(8),Lunghezza(9)  AS
Dat$(9)¶
```

Lunghezza della linea

E' necessario a questo punto discutere un attimo sul formato delle linee di programma accettate dall'AmigaBASIC. La lunghezza massima accettata è di 255 caratteri, ma un buon stile di programmazione sconsiglia di utilizzare linee di tale lunghezza. Tuttavia in casi come quello precedente è inevitabile utilizzare linee così lunghe. Infatti è necessario definire completamente il record in un'unica linea FIELD. Utilizzando in un'altra linea il comando FIELD si provoca la cancellazione dei parametri precedentemente definiti per il buffer. In tal modo, se si desidera utilizzare 10 variabili, è necessario inserire una linea come quella vista, fornendo completamente le definizioni di seguito al comando FIELD e premendo il tasto <Return> solamente quando si è finito di battere l'intera linea.

La parte successiva deve essere inserita come al solito:

```
    FOR x=1 TO NumeroCampi¶
      LSET Dat$(x)=" " ¶
    NEXT x¶
  CLOSE 1¶
  OPEN Nome$+".Campi" FOR OUTPUT AS 2¶
    PRINT #2,NumeroCampi¶
    PRINT #2,LunghezzaRecord¶
    PRINT #2,0¶
    FOR x=0 TO NumeroCampi¶
      WRITE #2,NomeCampo$(x)¶
      PRINT #2,Lunghezza(x)¶
```

```
      NEXT x¶
    CLOSE 2¶
    AltroNome$=Nome$¶
  GOTO Partenza¶
```

Questa parte di programma consente di creare un nuovo file. Per prima cosa viene inserita nell'angolo in alto a sinistra, dove viene specificato il modo corrente, la scritta Creazione File. Il file su cui si sta agendo in questo momento ha come nome "(nessun nome)", in quanto non è stato ancora richiamato alcun file e quindi non vi è alcun nome.

Utilizzo dei colori

Ci si potrebbe chiedere il perchè di questo utilizzo elaborato del comando COLOR nel listato. Si sta cercando di visualizzare il testo con colori facilmente visibili e che non disturbino gli occhi. Si può notare come le parole Note, Commenti, Avvertimenti e Spiegazioni siano visualizzate in arancione. I dati, la barra titolo e gli input forniti dall'utente sono invece di colore bianco. I nomi dei campi dei record sono in verde. COLOR 3,0 consente di selezionare l'arancione, COLOR 2,0 il verde, COLOR 1,0 il bianco.

La routine CreazioneFile: è quella in cui l'utente può definire il nome e la lunghezza dei vari campi. Questa informazione viene inserita come precedentemente specificato.

Ogni record di dati può contenere fino a dieci elementi (come definito dal ciclo FOR x=0 TO 9).

Il programma conserva il numero dei campi per record nella variabile NumeroCampi, valore incrementato di una unità per ogni nuovo campo inserito.

Nomi dei campi

Per prima cosa è necessario specificare i nomi dei vari campi. Si tratta di identificatori come Nome, Via, Citta, Titolo, etc. La lunghezza massima prevista per il nome di un campo è di 25 caratteri. Successivamente è necessario riservare una certa quantità di spazio per i dati stessi.

Se viene inserito uno spazio bianco durante questa azione, il programma riconosce che tutti i campi sono stati specificati. Quando sono stati definiti 10 campi, il ciclo termina.

Dopo aver inserito i nomi identificativi dei vari campi, è necessario specificarne la lunghezza. La lunghezza massima per un campo dati è di 40 caratteri.

Per prima cosa il comando SPACE$(40) consente di cancellare il nome del campo corrente, causando quindi la cancellazione di 25 caratteri. Deve ora essere inserito un valore per la variabile Lunghezza$. Se non viene specificato alcun valore, oppure viene specificato un numero superiore a 40, la lunghezza del campo viene fissata a 40 caratteri. I comandi nidificati INT(ABS(VAL(Lungezza$))) filtrano tutti gli errori casuali che si verificano nella fase di input. VAL consente di convertire una stringa in un numero. Se nella stringa vengono inseriti caratteri alfabetici o di controllo, essi vengono scartati. ABS effettua la conversione di un numero in un numero positivo: in altre parole, se l'input comincia con il segno meno, viene cambiato di segno. INT converte un numero in intero. La lunghezza di un campo deve essere un numero intero; se l'utente specifica 0, viene utilizzato il valore 40. In questo modo si spera di prevenire eventuali input erronei.

Il ciclo viene eseguito 10 volte. Si ottengono così 10 nomi di campi, il massimo numero possibile.

Correzione dell'input

La routine Correzioni: richiama la subroutine ImmissioneOK: che richiede di specificare se l'input fornito è corretto. In caso di risposta affermativa, la variabile Corr accetta l'input, in caso contrario viene chiamata la routine CorrezioneErrori:.

A questo punto vengono visualizzati i valori precedentemente definiti per i nomi e le lunghezze dei campi, incolonnati uno sotto all'altro. Un ciclo FOR ... NEXT conta da 1 a NumeroCampi. Ogni dato di specifica di un campo può essere corretto.

Mediante il comando LINE INPUT per ogni campo viene visualizzato il cursore come segnale di richiesta di un nuovo input. Se si risponde con una

stringa vuota, il precedente valore rimane invariato, Ogni altro input rimpiazza il vecchio input. Le limitazioni nella lettura della lunghezza dei campi sono identiche a quelle effettuate dalle linee di programma che svolgono la stessa funzione nella routine CreateFile:.

Dopo aver effettuato tutte le correzioni, per consentire di effettuare ulteriori modifiche, il programma ritorna alla linea etichettata con Correzioni:. La routine viene eseguita finchè si desidera apportare modifiche.

Apertura del File

Viene quindi eseguita la ruotine AperturaFile:, la quale richiede il nome del file. Un ciclo FOR ... NEXT consente di calcolare, a partire dai contenuti del vettore Lungezza(x), la lunghezza totale del record. Se tale lunghezza risulta essere zero, Amiga produrrà un BEEP, e il programma ritornerà alla linea etichettata con Partenza:. Altrimenti, viene aperto il file ad accesso casuale con il nome fornito in Nome$. Si giunge ora alla linea FIELD; è necessario inserire questa linea attentamente e su una singola linea.

Come si può notare, risulta possibile utilizzare come variabili parametro, variabili del tipo Dat$(x). La soluzione è differente rispetto a quella precedentemente utilizzata per la quantità di dati associati ad ogni record, in quanto i nomi e le lunghezze dei campi sono variabili. Ogni elemento del campo dati Dat$(x) viene trasferito nel buffer con lunghezza Lungezza(x). Se questa lunghezza è resa nulla da elementi inutilizzati dell'array, le variabili in soprannumero non sprecano memoria del buffer.

Infine, per cancellare ogni dato precedente, vengono riempiti con spazi bianchi i campi dati del buffer.

Con questa ultima operazione si concludono le routine preliminari per la gestione di un data base con un file ad accesso casuale.

Viene creato un file sequenziale in cui vengono mantenute alcune informazioni riguardanti il numero dei campi (NumeroCampi), la lunghezza (LunghezzaRecord), il numero di record, il nome e la lunghezza dei vari campi. Tale file sequenziale viene creato con il nome del file ad accesso casuale, ma con l'estensione .Campi. Nel caso del file di indirizzi, il nome risulta quindi Indirizzi.Campi.

Se questo file è stato scritto, utilizza come ultimo nome di file Altronome$, ritornando poi a Partenza:.

Ora che si ha a disposizione un file ad accesso casuale aperto ed in attesa di input, si può iniziare a riempirlo con dati.

```
ImmissioneDati:¶
  CLS : LOCATE 1,1 : PRINT "Immettere Dati"¶
¶
  IF Nome$="" THEN¶
    LOCATE 3,1 : COLOR 3,0 : PRINT "Immettere
       Nome File:"¶
    COLOR 1,0 : LINE INPUT Nome$¶
    IF Nome$="=" OR Nome$="*" THEN Nome$=AltroNome$¶
    IF Nome$="" THEN Partenza¶
    AltroNome$=Nome$¶
  END IF¶
  GOSUB CampoFilePresenteSN¶
  IF FilePresente=0 THEN¶
    COLOR 3,0 : PRINT ¶
    PRINT "Premere un tasto."¶
    WHILE INKEY$="" : WEND : COLOR 1,0¶
    GOTO Partenza¶
  END IF¶
  GOSUB LeggiCampoFile¶
  NumeroRecord=NumeroDeiRecord+1¶
  ¶
  OPEN "R",#1,Nome$,LunghezzaRecord¶
```

Ecco nuovamente la lunga linea FIELD. E' possibile utilizzare le opzioni Copy e Paste per duplicare quella scritta precedentemente.

```
      FIELD #1,Lunghezza(0) AS Dat$(0),Lunghezza(1) AS
Dat$(1),Lunghezza(2)  AS  Dat$(2),Lunghezza(3)  AS
Dat$(3),Lunghezza(4)  AS  Dat$(4),Lunghezza(5)  AS
Dat$(5),Lunghezza(6)  AS  Dat$(6),Lunghezza(7)  AS
Dat$(7),Lunghezza(8)  AS  Dat$(8),Lunghezza(9)  AS
Dat$(9)
```

Ecco ora altre linee di programma:

```
CicloImmissione:¶
      CLS : LOCATE 1,1 : COLOR 1,0 : PRINT
           "Immettere nuovi dati"¶
      LOCATE 1,25 : COLOR 3,0 : PRINT "File:";¶
      COLOR 1,0 : PRINT Nome$¶
¶
      Inpt=0¶
      LOCATE 1,50 : PRINT "Record:";NumeroRecord¶
      PRINT : COLOR 3,0¶
      PRINT "Immettere nuovi dati:" : COLOR 1,0¶
    FOR x=0 TO NumeroCampi¶
      LOCATE 5+x,1 : COLOR 2,0 : PRINT
           NomeCampo$(x)": "¶
    NEXT x : COLOR 1,0¶
    FOR x=0 TO NumeroCampi¶
      LOCATE 5+x,LEN(NomeCampo$(x))+3¶
      LINE INPUT Immissione$¶
      IF Immissione$<>"" THEN Imm=1¶
      Immissione$(x)=LEFT$(Immissione$,Lunghezza(x))¶
      LSET Dat$(x) = Immissione$(x)¶
    NEXT x¶
    Correzioni2:¶
    GOSUB ImmissioneOK¶
    IF Corr=0 THEN ScritturaRecord¶
    IF Corr=1 THEN ImmettiCorrezioni¶
  GOTO Correzioni2¶
¶
  ImmettiCorrezioni:¶
    CLS : LOCATE 1,1 : COLOR 1,0 : ¶
      PRINT "Aggiungere Dati"¶
    LOCATE 1,25 : COLOR 3,0 : PRINT "File:";¶
    COLOR 1,0 : PRINT  Nome$¶
¶
      LOCATE 1,50 : PRINT "Record:";NumeroRecord¶
      PRINT : PRINT ¶
```

```
    FOR x=0 TO NumeroCampi¶
      LOCATE 5+x,1 : COLOR 2,0 : PRINT
           NomeCampo$(x)": ";¶
      COLOR 1,0 : PRINT Immissione$(x)¶
 ¶
    NEXT x¶
    FOR x=0 TO NumeroCampi¶
      LOCATE 5+x,LEN(NomeCampo$(x))+2¶
      LINE INPUT Immissione$¶
      IF Immissione$<>"" THEN¶
        Imm=1¶
            Immissione$(x)=LEFT$(Immissione$,
              Lunghezza(x))¶
        LSET Dat$(x) = Immissione$(x)¶
      END IF¶
    NEXT x¶
    GOTO Correzioni2¶
 ¶
ScritturaRecord:¶
  IF Imm=1 THEN¶
    PUT #1,NumeroRecord¶
    IF SegnaleDati=1 THEN SegnaleDati=0 : GOTO
         CicloLettura¶
    NumeroRecord=NumeroRecord+1¶
  END IF¶
  IF SegnaleDati=1 THEN SegnaleDati=0 : GOTO
      CicloLettura¶
SeguenteSN:¶
  LOCATE 19,1 : PRINT SPACE$(60) : COLOR 3,0¶
  LOCATE 19,1 : PRINT "Record seguente (S/N)";¶
  COLOR 1,0 : LINE INPUT a$¶
  IF UCASE$(a$)="S" OR a$="" THEN CicloImmissione¶
  IF UCASE$(a$)="N" THEN ChiusuraFile¶
GOTO SeguenteSN¶
¶
ChiusuraFile:¶
```

```
    CLOSE 1¶
    OPEN Nome$+".Campi" FOR OUTPUT AS 2¶
      PRINT #2,NumeroCampi¶
      PRINT #2,LunghezzaRecord¶
      PRINT #2,NumeroRecord-1¶
      FOR x=0 TO NumeroCampi¶
        WRITE #2,NomeCampo$(x)¶
        PRINT #2,Lunghezza(x)¶
      NEXT x¶
    CLOSE 2¶
    Nome$=""¶
  GOTO Partenza¶
```

Inserimento dei dati

La routine per l'inserimento dei dati visualizza il suo nome nell'angolo in alto a sinistra dello schermo. Se la variabile Nome$ non contiene alcun valore, deve essere specificato un nuovo nome di file. E' possibile utilizzare anche i caratteri = e * per specificare che si desidera riutilizzare l'ultimo file indicato.

Viene poi richiamata la subroutine CampoFilePresenteSN: che consente di determinare se esiste un file con lo stesso nome e con estensione .Campi assegnando il risultato di questo controllo alla variabile FilePresente. Se tale variabile vale 0 significa che il file non esiste e quindi non viene letto; viceversa, se vale 1 il file esiste e viene letto. Se il file con estensione .Campi esiste, viene allora anche letto un corrispondente file ad accesso casuale. La routine per il controllo dell'esistenza del file è stata inserita nel programma per evitare di ottenere la segnalazione di un messaggio d'errore dall'AmigaBASIC nel caso il file non esista.

A questo scopo è stato utilizzato un piccolo trucchetto: se il file non esiste il programma ritorna all'etichetta Partenza: quando viene premuto un tasto qualsiasi.

La subroutine LeggiCampoFile: legge il file .Campi corrispondente al file ad accesso casuale specificato. Di conseguenza, vengono acquisite le informazioni relative ai valori che devono essere assuenti dalle variabili NumeroCampi, LunghezzaRecord, e NumeroDeiRecord e vengono letti i

nomi dei vari campi e le relative lunghezze.

Aggiungere dati
Alla variabile NumeroRecord, utilizzata come contatore per i record, viene assegnato il valore della variabile NumeroDeiRecord, incrementato di una unità, in modo da consentire l'inserimento di nuovi record. Viene poi aperto il file ad accesso casuale.

La routine di input CicloImmissione: visualizza il nome del file corrente al centro della prima linea dello schermo e presenta il messaggio "Immettere Nuovi Dati". Viene anche inizializzata a zero una variabile Inpt, che verrà successivamente utilizzata per verificare se sia stato fornito o meno un nuovo input.

Input ai record
Nell'angolo destro dello schermo viene visualizzato il numero del record con cui l'utente sta attualmente lavorando. Possono ora essere inseriti i nuovi dati. Il programma favorisce questa procedura visualizzando opportuni caratteri di sollecitazione e mostrando ad uno ad uno i nomi dei vari campi, dal primo all'ultimo. Il cursore viene posizionato a fianco di ogni nome, cominciando dal primo, ed il programma si pone in attesa di un input (in seguito al comando LINE INPUT). Il primo input non nullo (cioè il primo input che differisce da uno spazio) porta all'assegnazione del valore 1 alla variabile Inpt. Questo segnala che è stato incontrato un input corretto. Nel relativo campo viene allora inserito il valore della variabile Immissione$ che, dopo essere stata opportunamente dimensionata, viene scritta nel buffer utilizzando il comando LSET. Il vettore Immissione$(x) conserva tutti i dati in input al record corrente, indipendentemente dallo stato del buffer.

Correzione degli errori
Durante l'esecuzione del ciclo l'utente ha a disposizione l'opportunità di correggere eventuali errori. Per consentire questo viene utilizzata la routine ImmissioneOK: che gestisce la variabile Corr. Se Corr vale 1 viene richiamata la routine ImmettiCorrezioni:. Questa routine visualizza, per prima cosa, la scritta Aggiungere Dati all'estrema sinistra della prima linea dello schermo e successivamente il nome del file ed il numero di record.

Il ciclo FOR ... NEXT immediatamente successivo visualizza il nome dei vari campi ed il contenuto di ogni singolo campo. All'interno del ciclo è contenuta una routine di input che consente di effettuare le correzioni. Dopo aver effettuato tutte le modifiche, il record viene scritto sul dischetto utilizzando la routine ScritturaRecord: (chiaramente solo se la variabile Inpt vale 1).

Se la variabile SegnaleDati vale 1, alla fine della fase di inserimento viene richiamata la routine CicloImmissione:. In caso contrario, NumeroRecord viene incrementata di uno e può essere effettuato un nuovo input. La routine SeguenteSN: richiede all'utente se desidera inserire un nuovo record; in caso affermativo il file viene preparato a ricevere un nuovo input, viceversa il file viene chiuso e viene scritto un nuovo file .Campi contenente le informazioni aggiornate. E' possibile notare che ora Nome$ viene svuotato prima di ritornare alla linea etichettata Partenza:. Dovrà quindi essere specificato un nuovo nome prima del successivo accesso al file.

```
LetturaDati:¶
    CLS : LOCATE 1,1 : PRINT "Read Data"¶
    IF RicercaDati=1 THEN LOCATE 1,1 : PRINT
        "Ricerca Dati"¶
    LOCATE 3,1 : COLOR 3,0 : PRINT "Immettere
        Nome File:"¶
    COLOR 1,0 : LINE INPUT Nome$¶
    IF Nome$="=" OR Nome$="*" THEN Nome$=AltroNome$¶
    IF Nome$="" THEN Partenza¶
     AltroNome$=Nome$¶
¶
    GOSUB CampoFilePresenteSN¶
    IF FilePresente=0 THEN¶
      PRINT : COLOR 3,0¶
      PRINT "Premere un tasto."¶
      COLOR 1,0¶
      WHILE INKEY$="" : WEND¶
      GOTO Partenza¶
    END IF¶
    GOSUB LeggiCampoFile¶
    IF NumeroDeiRecord=0 THEN¶
      PRINT : BEEP¶
```

```
        COLOR 1,0¶
        PRINT "Non esistono record nel file!"¶
        PRINT : COLOR 3,0¶
        PRINT "Premere un tasto."¶
        COLOR 1,0¶
        WHILE INKEY$="" : WEND¶
      END IF¶
    IF RicercaDati=1 THEN GOSUB CercareDati¶
    OPEN "R",#1,Nome$,LunghezzaRecord¶
       FIELD #1,Lunghezza(0) AS Dat$(0),Lunghezza(1) AS
Dat$(1),Lunghezza(2)  AS  Dat$(2),Lunghezza(3)  AS
Dat$(3),Lunghezza(4)  AS  Dat$(4),Lunghezza(5)  AS
Dat$(5),Lunghezza(6)  AS  Dat$(6),Lunghezza(7)  AS
Dat$(7),Lunghezza(8)  AS  Dat$(8),Lunghezza(9)  AS
Dat$(9)¶
       NumeroRecord=1¶
CicloLettura:¶
     CLS : LOCATE 1,1 : COLOR 1,0 : PRINT
         "Lettura Dati"¶
     LOCATE 1,25 : COLOR 3,0 : PRINT "File:";¶
     COLOR 1,0 : PRINT Nome$¶
     COLOR 3,0¶
     LOCATE 17,1 : PRINT "[Freccia SU] = Record
         Precedente"¶
     LOCATE 17,37 : PRINT "[F1] = Primo Record"¶
     PRINT "[Freccia GIU'] = Record Seguente"¶
     LOCATE 18,37 : PRINT "[F2] = Ultimo Record"¶
     PRINT "[CTRL]-[P] = Stampa Record"¶
     LOCATE 19,37 : PRINT "[HELP] = Modifica Record"¶
     PRINT "[F10] = Menu Principale";¶
 LeggiRecord:¶
     COLOR 1,0¶
     IF NumeroRecord>NumeroDeiRecord THEN BEEP :
         NumeroRecord=NumeroDeiRecord¶
     IF NumeroRecord<1 THEN BEEP : NumeroRecord=1¶
     LOCATE 1,50 : PRINT "Record:";NumeroRecord¶
     GET #1,NumeroRecord¶
```

```
      IF RicercaDati=1 THEN LOCATE 1,1 : PRINT
          "Ricerca Dati" : GOSUB EsameRicercaDati¶
      IF RicercaDati=1 AND Trovato=0 THEN¶
         IF NumeroRecord=NumeroDeiRecord THEN
             Direzione=-1¶
         IF NumeroRecord=NumeroDeiRecord AND
             TrovaRecord=0 THEN¶
            CLS¶
           LOCATE 5,10 : PRINT "Nessun record trovato!"¶
            LOCATE 7,10 : COLOR 3,0¶
            PRINT "Premere un tasto."¶
            COLOR 1,0 : BEEP¶
            WHILE INKEY$="" : WEND : CLOSE 1 :
                 GOTO Partenza¶
         END IF¶
         IF NumeroRecord=1 THEN Direzione=1¶
         NumeroRecord=NumeroRecord+Direzione¶
        GOTO LeggiRecord¶
      END IF¶
      TrovaRecord=1¶
      FOR x=0 TO NumeroCampi¶
        LOCATE 5+x,1 : COLOR 2,0 : PRINT NomeCampo$(x)":
"¶
      NEXT x : COLOR 1,0¶
      FOR x=0 TO NumeroCampi¶
        LOCATE 5+x,LEN(NomeCampo$(x))+3¶
        PRINT Dat$(x)¶
        Immissione$(x)=Dat$(x)¶
      NEXT x¶
      Tasto$=""¶
      WHILE Tasto$="" : Tasto$=INKEY$ : WEND¶
      IF Tasto$=CHR$(28) THEN NumeroRecord=
          NumeroRecord-1 : Direzione=-1¶
      IF Tasto$=CHR$(29) THEN NumeroRecord=
          NumeroRecord+1 : Direzione=1¶
      IF Tasto$=CHR$(129) THEN NumeroRecord=1¶
```

```
    IF Tasto$=CHR$(130) THEN NumeroRecord=
         NumeroDeiRecord¶
    IF Tasto$=CHR$(138) THEN FineCarica¶
    IF Tasto$=CHR$(16) THEN¶
       FOR x=0 TO NumeroCampi¶
         LPRINT NomeCampo$(x)":"Dat$(x)¶
       NEXT x¶
       LPRINT¶
    END IF¶

    IF Tasto$=CHR$(139) THEN SegnaleDati=1 :
        GOTO ImmettiCorrezioni¶
GOTO CicloLettura¶
¶
FineCarica:¶
    CLOSE 1¶
    Nome$=""¶
GOTO Partenza¶
```

Lettura e ricerca

La routine LetturaDati: presenta due differenti modalità di funzionamento: lettura dei dati e ricerca specifica. Nella modalità di ricerca, vengono visualizzati solamente i record contenenti la chiave di ricerca specificata. La variabile RicercaDati viene utilizzata per stabilire quale modalità è stata attivata. Se tale variabile vale 1, viene attivata la modalità di ricerca, in caso contrario quella di visualizzazione globale dei dati. La modalità attiva viene visualizzata nell'angolo in alto a sinistra dello schermo.

Per entrambe le modalità è necessario specificare il nome del file di cui si vogliono esaminare i dati. Anche in questo caso è possibile utilizzare i caratteri = o * per riferirsi all'ultimo nome specificato.

La subroutine CampoFilePresente: viene richiamata per determinare se il file richiesto esista. In caso negativo si ottiene una segnalazione appropriata, altrimenti viene eseguita la routine LeggiCampoFile:.

Nel caso non vengano trovati record viene visualizzato il messaggio "Non

esistono Record nel File!". Si tenga sempre presente che, prima di poter leggere dei dati, questi devono venire inseriti nel file utilizzando la modalità di inserimento, selezionabile con l'opzione numero 2 del menù.

Nel caso sia attiva la ricerca (cioè nel caso in cui la variabile RicercaDati valga 1), viene chiamata la subroutine CercareData: in cui l'utente può specificare la chiave di ricerca. Viene quindi aperto il file ad accesso casuale.

Visualizzazione dei dati
Il programma visualizza i record cominciando dal primo. La variabile NumeroRecord, inizializzata ad 1, viene incrementata man mano che viene mostrato un record successivo.

Viene ora eseguita la routine CicloLettura: che consente di leggere il record e visualizzare i dati. Questa parte di programma comincia con un identificatore in modo tale da stabilire il file da leggere. Viene poi effettuata l'inizializzazione dello schermo. All'esterno dell'area in cui vengono visualizzati i dati vengono mostrate le funzioni svolte da alcuni tasti e da alcune combinazioni di tasti durante l'interazione con il programma.

Spostarsi tra i record
I tasti cursore consentono di passare da un record all'altro: <Cursore in alto> consente di passare al record successivo, mentre <Cursore in basso> al record precedente. Il tasto <F1> consente di posizionarsi immediatamente sul primo record, mentre il tasto <F2> consente di spostarsi in fondo al file. Quando ci si trova nella modalità di ricerca si ha un funzionamento analogo, ad eccezione del fatto che quanto sopra mostrato vale solamente per i record che contengono la chiave di ricerca, per cui i salti verranno effettuati solamente tra questi record (<Cursore in alto> consentirà di passare al record successivo contenente la chiave cercata e così via). La combinazione di tasti <CTRL> <P> consente di stampare il record attualmente visualizzato; il tasto <HELP> consente di modificare il record presentato, sia per eventuali modifiche che aggiornamenti; il tasto <F10> consente di chiudere il file, dopo di che il programma ritorna alla linea etichettata Partenza:.

E' possibile accedere ai dati inseriti utilizzando la routine LetturaDati:. Prima dello spostamento tra i record con i tasti cursore, il programma controlla

se il record a cui ci si intende spostare esista e se il suo numero non sia troppo grande o troppo piccolo. Il numero del record viene visualizzato nell'angolo in alto a destra dello schermo. Il comando GET consente di caricare dal dischetto il record specificato nel buffer.

Nel caso sia attiva la modalità di ricerca, viene visualizzato il relativo identificatore di modalità (RicercaDati al posto di LetturaDati). Viene quindi richiamata la routine EsameRicercaDati: per controllare se in qualche record è contenuta la chiave di ricerca specificata. In caso positivo, alla variabile Trovato viene assegnato il valore 1, altrimenti il valore 0. Viene in seguito eseguito il blocco IF / ENDIF per decidere che tipo di azione intraprendere se il record non contiene quanto cercato. Nel caso venga raggiunto il limite superiore del numero dei record, viene invertita la direzione ed il valore diventa -1. La ricerca viene eseguita dal fondo al principio e continua fino a quando non viene esaminato l'ultimo record.

NOTA:
Vi è un problema all'interno della routine di ricerca: nel caso in cui il file non contenga la chiave cercata all'interno di nessun record, il programma prosegue nella ricerca all'infinito. Inoltre, se il record corrente è l'ultimo record del file, non sono disponibili record per la ricerca (la variabile TrovaRecord vale 0), per cui viene terminata la procedura di ricerca e viene presentato il messaggio "Nessun Record Trovato! - Premi un tasto" che consente di ritornare al menù principale premendo un tasto qualsiasi.

Nel caso in cui il programma trovi almeno un record contenente la chiave, l'esecuzione continua correttamente. Partendo dal valore di record inferiore, cioè dal primo record del file (NumeroRecord=1), la ricerca per localizzare l'occorrenza della stringa ricercata viene effettuata in avanti (Direzione=1). A NumeroRecord viene sommato Direzione e viene poi controllato un nuovo record. Dopo ENDIF alla variabile TrovaRecord viene assegnato 1, assumendo ovviamente che sia stato trovato almeno un record contenente la chiave cercata. A questo punto vengono visualizzati il nome del campo ed il contenuto del file. Il contenuto del record corrente viene marcato nello stesso modo descritto per il vettore Immissione$, in modo tale che i dati possano essere eventualmente corretti.

Il ciclo WHILE .. WEND che segue pone il programma in attesa di un input da tastiera, assegnando successivamente l'input ricevuto alla variabile Tasto$. Se Tasto$ assume il valore CHR$(28), corrispondente al tasto <Cur-

sore in alto>, il numero del record corrente viene decrementato di uno. Se è attiva la modalità di ricerca, la ricerca viene effettuata all'indietro. Nel caso venga ricevuto CHR$(29), codice del tasto <Cursore in basso>, si ottiene il funzionamento contrario. F1 invia il codice CHR$(129) che informa il calcolatore della richiesta di posizionamento all'inizio del file, cioè in corrispondenza del record numero 1.

Al tasto F2 corrisponde il codice CHR$(130) che porta al posizionamento sull'ultimo record. La variabile NumeroDeiRecord conterrà allora il numero dell'ultimo record. CHR$(138), corrispondente al tasto F10, produce il ritorno al menù principale e la chiusura del file.

Ottenere una stampa

Quando si desidera effettuare la stampa del record corrente, è sufficiente premere la combinazione di tasti <CTRL> <C>. Il codice relativo a tale combinazione è CHR$(16). I nomi dei campi ed i loro contenuti sono inviati alla stampante utilizzando il comando LPRINT. I comandi LPRINT con spazi bianchi sono utilizzati per mantenere la necessaria spaziatura tra i vari record.

La pressione del tasto <HELP>, che produce il codice CHR$(139), consente di realizzare correzioni od inserimenti di nuovi dati. La variabile SegnaleDati viene posta ad 1 e viene richiamata la routine ImmettiCorrezioni:. Queste variabili consentono al programma di sapere quale routine utilizzare per la correzione. La routine CicloLettura: termina poi richiamando nuovamente se stessa.

La routine FineCarica: riporta il sistema al menù principale, chiudendo il file ad accesso casuale.

Devono essere ora realizzate le routine per la ricerca dei dati, per il controllo dell'esistenza del file contenente la informazioni relative ai vari campi, e per la lettura di questo file.

- Posizionarsi alla fine del listato battuto precedentemente, ed inserire le linee seguenti:

```
CercareDati:¶
  CLS : LOCATE 1,1 : COLOR 1,0 : PRINT "Ricerca Dati"¶
```

```
    LOCATE 1,25 : COLOR 3,0 : PRINT "File:";¶
    COLOR 1,0 : PRINT Nome$¶
    FOR x=0 TO NumeroCampi¶
      LOCATE 5+x,1 : PRINT NomeCampo$(x)":"¶
    NEXT x¶
    COLOR 3,0 : LOCATE 4,1¶
    PRINT "Immettere la stringa da cercare."¶
    COLOR 1,0¶
      FOR x=0 TO NumeroCampi¶
        LOCATE 5+x,LEN(NomeCampo$(x))+2¶
        LINE INPUT Immissione$¶
        IF Immissione$<>"" THEN¶
          Ricerca$=LEFT$(Immissione$,Lunghezza(x))¶
          RicercaNum=x : x=10¶
        ELSE¶
          Ricerca$=""¶
        END IF¶
      NEXT x¶
    Correzioni3:¶
       GOSUB ImmissioneOK¶
      IF Corr=0 THEN FineRicerca¶
      IF Corr=1 THEN RicercaCorrente¶
    GOTO Correzioni3¶
¶
    RicercaCorrente:¶
      LOCATE 5+RicercaNum,1 : PRINT
          NomeCampo$(RicercaNum)":"Ricerca$¶
       LOCATE 5+RicercaNum,LEN(NomeCampo$
          (RicercaNum))+2¶
      LINE INPUT Immissione$¶
      IF Immissione$<>"" THEN Ricerca$=LEFT$
          (Immissione$,Lunghezza(RicercaNum))¶
    GOTO Correzioni3¶
    ¶
  FineRicerca:¶
    IF Ricerca$="" THEN RicercaNum=0 : RicercaDati=0¶
```

```
  TrovaRecord=0¶
RETURN¶
¶
EsameRicercaDati:¶
    x=0¶
CicloRicerca:¶
    x=x+1¶
    IF x>LEN(Dat$(RicercaNum))-LEN(Ricerca$) THEN
       Trovato=0 : RETURN¶
    IF MID$(Dat$(RicercaNum),x,LEN(Ricerca$))=
       Ricerca$ THEN Trovato=1 : RETURN¶
GOTO CicloRicerca¶
¶
ImmissioneOK:¶
  LOCATE 19,1 : COLOR 3,0¶
  PRINT "Immissione OKAY? (S/N)";
  COLOR 1,0 : INPUT "",a$¶
  IF UCASE$(a$)="S" OR a$="" THEN Corr=0 : RETURN¶
  IF UCASE$(a$)="N" THEN Corr=1 : RETURN¶
GOTO ImmissioneOK¶
¶
CampoFilePresenteSN:¶
  OPEN Nome$+".Campi" FOR APPEND AS 1¶
    IF LOF(1)<=0 THEN FilePresente=0 ELSE
        FilePresente=1¶
  CLOSE 1¶
  IF FilePresente=0 THEN¶
    LOCATE 3,1 : PRINT SPACE$(60) : BEEP¶
    LOCATE 3,1 : COLOR 1,0 : PRINT "File ";Nome$¶
    PRINT "Non Trovato!"¶
    KILL Nome$+".Flds"¶
    Nome$="" : COLOR 3,0¶
  END IF¶
RETURN¶
¶
LeggiCampoFile:¶
```

```
    FOR x=1 TO 10¶
      NomeCampo$(x)="" : Lunghezza(x)=0¶
    NEXT x¶
    OPEN Nome$+".Campi" FOR INPUT AS 2¶
      INPUT #2,NumeroCampi¶
      INPUT #2,LunghezzaRecord¶
      INPUT #2,NumeroDeiRecord¶
      FOR x=0 TO NumeroCampi¶
        INPUT #2,NomeCampo$(x)¶
        INPUT #2,Lunghezza(x)¶
      NEXT x¶
    CLOSE 2¶
  RETURN
```

La routine CercareDati: contiene tutte le inizializzazioni necessarie per la modalità di ricerca. Essa identifica le varie parti del programma e, dopo aver determinato il nome del file corrente, visualizza tale nome, seguito dal messaggio "Immettere la stringa da cercare". Si ha così la possibilità di inserire la stringa di cui verrà ricercata la presenza in tutti i record del file.

Consigli per la ricerca di dati

Se si desidera ricercare tra i dischi della propria collezione tutti i dischi di Elton John, è sufficiente specificare come stringa di ricerca per il campo Artisti la stringa Elton John. Il programma controlla tutti i record per vedere se contengono, nel campo Artisti, la stringa specificata. Se la stessa stringa venisse specificata per il campo Titolo, il programma cercherebbe invece in tale campo, in cui si potrebbe ad esempio trovare: Elton John's greatest hits. Questo semplice esempio mostra abbastanza bene il funzionamento della modalità di ricerca. Insieme alla stringa viene mostrato il testo completo contenuto nel campo per cui, se per il campo Artisti venisse specificato solamente il nome John, verrebbero considerati anche i record relativi ad Olivia Newton-John.

NOTA:

E' importante inserire correttamente nella stringa di ricerca lettere maiuscole e minuscole, in quanto la stringa viene confrontata esattamente come

è stata inserita.

Modalità di funzionamento della routine di ricerca
La routine CercareDati esegue il ciclo di lettura. E' possibile spostarsi tra i vari campi premendo semplicemente il tasto <Return>, fino a quando non si raggiunge il campo per il quale si desidera specificare la stringa di ricerca. L'input fornito viene memorizzato nella variabile Ricerca$. RicercaNum contiene il numero di campi che devono essere confrontati. La routine Correzioni3: offre la possibilità di correggere l'input fornito. E' possibile solamente modificare la stringa di ricerca, mentre il campo selezionato viene comunque mantenuto come campo di confronto.

La routine FineRicerca: consente di ritornare alla linea da cui era stata chiamata la subroutine. Se il contenuto della variabile Ricerca$ è nullo, viene annullata la ricerca: in questo caso RicercaNum e RicercaDati hanno entrambi valore 0. Se anche la variabile TrovaRecord vale 0, si potrà successivamente determinare se sia stato realmente trovato un Record in base agli ultimi criteri di ricerca utilizzati.

Nella sezione EsameRicercaDati: viene controllato se il record corrente contenga la chiave di ricerca. Il ciclo CicloRicerca: viene eseguito fino a quando tutte le possibili posizioni sono state confrontate con le variabili Dat$ e Ricerca$. La variabile Dat$ contiene una stringa parziale della stessa lunghezza della stringa di ricerca. Quando x diventa più grande della lunghezza di Dat$ meno la lunghezza di Ricerca$, viene dedotto che non è stata trovata la stringa ricercata, la variabile Trovato viene posta a zero, e si ha il ritorno dalla subroutine. Se viene invece trovata una stringa parziale, la variabile Trovato viene posta ad 1. Anche in questo caso si ha il ritorno dalla subroutine.

Correzioni
La sezione di programma successiva, cioè la subroutine ImmissioneOK:, viene richiamata molte volte da punti differenti del programma. Essa consente di determinare se l'utente desidera, o meno, correggere una qualsiasi informazione di input. Viene visualizzata la domanda "Immissione OKAY? (S/N)", ed il programma si pone in attesa della risposta. L'istruzione

```
INPUT "",a$
```

consente di eliminare il punto di domanda. Premendo il tasto "n" o la barra di spaziatura, seguiti da <Return>, viene posta a 0 la variabile Corr, cioè l'indicazione che non è stata apportata nessuna correzione. Viene accettato anche l'input nullo (Spazio bianco), al fine di sveltire il più possibile la procedura di selezione. Nel caso venga premuto il tasto <S> Corr viene posta a 1. Il programma continua poi la sua esecuzione.

La subroutine CampoFilePresente: controlla l'esistenza del file specificato con estensione .Campi. Nel caso si richieda l'apertura di un file non esistente, si riceve in risposta il messaggio di errore File Non Trovato. Il programma termina e i vecchi contenuti vengono cancellati, cosa non proprio ideale. E' possibile risolvere questo problema, utilizzando la modalità APPEND. Quando viene aperto un file sequenziale in modalità APPEND, i nuovi record vengono aggiunti ai vecchi, senza cancellarli. Se il file specificato non esiste, ne viene creato uno nuovo.

Per questo motivo viene aperto il file .Campi in modalità APPEND e ne viene determinata la lunghezza utilizzando il comando LOF(1): se tale lunghezza risulta essere 0 oppure -1, vuol dire che non vi sono dati nel file. Deve essere utilizzata la modalità APPEND per aprire inizialmente il file dato che esso non esiste. Se la variabile FilePresente ha valore 0, viene mostrato un messaggio indicante che il file specificato non è stato trovato.

Da ultima, ma non meno importante, la subroutine LeggiCampoFile: che consente di leggere ed esaminare il contenuto del file .Campi. Viene innanzitutto eseguito un ciclo FOR ... NEXT per cancellare il contenuto dei vettori NomeCampo$(x) e Lunghezza(x), in modo che non si verifichi il mantenimento dei nomi di campi in eccesso, ancora memorizzati da file precedenti. Vengono poi caricati il numero di campi per record (NumerodeiCampi), la lunghezza totale del record (LunghezzaRecord) ed il numero di record contenuti nel file. Il programma può utilizzare queste informazioni per aprire ed accedere al file.

Se non si è ancora provveduto a salvare questo programma, effettuare immediatamente il salvataggio. Se si dovesse verificare un guasto all'impianto elettrico, oppure se dovesse capitare che qualcuno per errore tolga la spina, potrebbero essere guai.

5.3 Un data base è un archivio organizzato: utilizzo del DataBase

Tutte le principali informazioni necessarie per utilizzare il programma per la gestione del DataBase sono state fornite nel paragrafo precedente.

In questo paragrafo vengono invece presentati suggerimenti per utilizzare al meglio il DataBase appena realizzato.

L'utilizzo più comune di un calcolatore riguarda l'analisi e l'elaborazione di dati. L'elaborazione di dati può essere utilizzata non solamente per affari, ma anche per divertimento.

E' possibile memorizzare racconti, poemi, elenchi di videocassette, di libri, istruzioni per il giardinaggio, qualsiasi altra cosa si desideri.

Tutto quello che è necessario fare, è memorizzare queste informazioni nel DataBase gestibile con il programma appena realizzato.

Il primo passo da compiere è quello di creare un nuovo file, selezionando l'opzione numero 1 del menù a tendina.

E' possibile specificare fino ad un massimo di 10 nomi di campi con la relativa lunghezza.

La massima lunghezza possibile per un campo è di 40 caratteri, valore assunto come valore di base se non viene specificato un parametro relativo alla sua lunghezza (cioè se viene semplicemente premuto il tasto <Return> in corrispondenza della relativa richiesta).

Nella figura 16 è mostrato un esempio di come avviene l'interazione in fase di input.

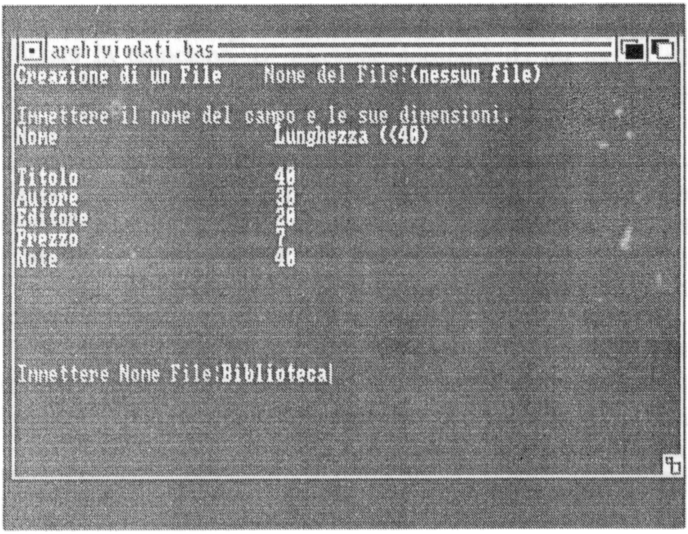


Figura 16: Il programma per la gestione di Data Base nella modalità per la Creazione del File.

Per segnalare che si è terminato di definire i campi relativi al proprio data base, è sufficiente premere due volte consecutivamente il tasto <Return>. Si ha ora la possibilità di correggere un qualunque errore di battitura. Nel caso sia stato commesso un errore, premere <N> e poi <Return> in risposta alla domanda "Immissione OKAY (S/N)". Il cursore verrà visualizzato in corrispondenza del primo campo dati. A questo punto è possibile reinserire il nuovo input, oppure premere <Return>. Quando ogni cosa sembra essere corretta, ed è stato premuto il tasto <Return> in corrispondenza di ogni campo dati, il programma visualizza nuovamente la domanda "Immissione OKAY? (S/N)". Dopo aver effettuato tutte le correzioni, battere <S> ed il tasto <Return>. Viene ora richiesto l'inserimento del nome che si desidera attribuire al file in cui verranno memorizzati i dati, mediante la domanda "Immettere Nome File". Inserire il nome desiderato e premere il tasto <Return>. Il programma apre due file su dischetto e ritorna al menù principale.

Scegliendo l'opzione 2 si entra direttamente in modalità Input (Immissione Dati). Se non è stato creato un file immediatamente prima della

selezione di questa opzione, è necessario inserire il nome di un file. Il nome dell'ultimo file utilizzato può essere abbreviato utilizzando il carattere * o il carattere =. Qusto fatto si rivela molto pratico quando si desidera passare direttamente dalla fase di inserimento a quella di ricerca dei dati.

Sullo schermo compare una "maschera" raffigurante i nomi precedentemente definiti per i vari campi. Il cursore viene posizionato nel primo campo dati del record corrente. E' possibile a questo punto inserire il dato. Un record completamente vuoto non verrà salvato, verrà invece riproposta, in questo caso, la procedura di input. Rispondendo alla domanda "Record Seguente(S/N)" con <S> o <Return>, è possibile spostarsi sul record successivo per un eventuale nuovo input. Nel caso si risponda invece con il tasto <N>, viene chiuso il file ed il programma visualizza nuovamente il programma principale.

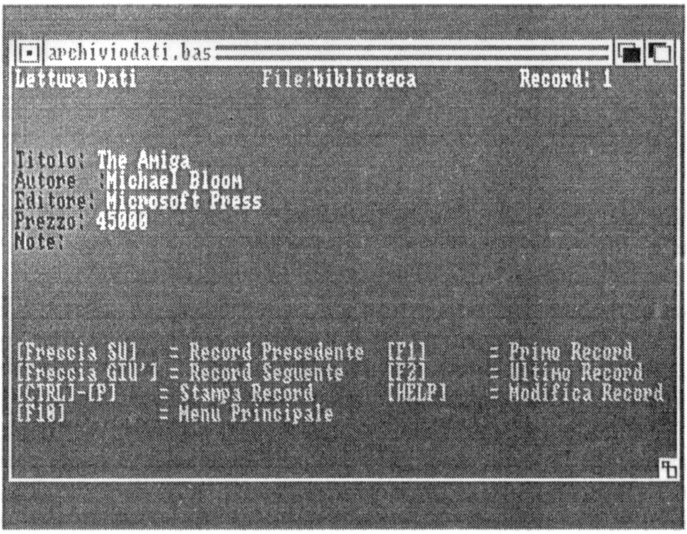


Figura 17: Il Programma per la gestione di DataBase in modalità Lettura-Dati.

L'opzione numero 3 del menù principale consente di visualizzare il contenuto di un file. E' necessario specificare il nome del file, oppure l'opportuna abbreviazione con il carattere * o = per indicare l'eventuale ri-

petizione del nome precedente. Se non viene fornito il nome di un file, il programma lo richiede nuovamente. All'interno dell'area dati è possibile vedere tutte le combinazioni di tasti disponibili e le corrispondenti opzioni, visualizzate in arancione. E' possibile spostarsi da un record all'altro, sia in avanti che indietro, oppure saltare direttamente al primo o all'ultimo record, stampare il contenuto di un record, oppure modificare lo schermo. Il tasto <F10> consente di ritornare al menù principale.

L'opzione numero 4 consente di selezionare la modalità Ricerca Dati che fornisce la possibilità di ricercare tutti i record identificabili secondo particolari criteri. Solamente i record in cui viene trovata la chiave di ricerca vengono visualizzati.

Dopo aver selezionato Effettuare Ricerche, è necessario specificare il nome del file in cui deve essere effettuata la ricerca. Il programma visualizza il nome dei vari campi sullo schermo e richiede di inserire la stringa di ricerca in uno dei campi. L'occorrenza di tale stringa viene poi ricercata in tutti i record del file. Dopo aver specificato la correttezza dell'input fornito, il programma apre il file ed inizia la ricerca. Se viene trovato almeno un record contenente la stringa di ricerca, il programma visualizza tale record. Nel caso non venga trovato alcun record, viene visualizzato il messaggio "Nessun Record Trovato". Il record eventualmente trovato può essere gestito esattamente come se si fosse in modalità Lettura Dati. Il programma esce dalla routine dopo aver raggiunto l'ultimo record.

Per concludere, un consiglio

I dati dovrebbero essere trattati con notevole cautela. Il pericolo è sempre in agguato: un'azione maldestra potrebbe causare la cancellazione di dati e la conseguente perdita di gran parte del contenuto di un Data Base.

6 AmigaParlante: la sintesi vocale in BASIC

Amiga possiede tutto l'hardware e il software necessari per una sintesi vocale perfettamente comprensibile. E' probabile che si sia già sentito "parlare" Amiga. I programmi dimostrativi del Workbench comprendono anche routine per la sintesi vocale ed esempi riguardanti l'eloquenza di Amiga possono essere trovati anche nel cassetto BASICDemos del dischetto Extras.

Da questo punto in avanti viene dato per scontato che l'utente abbia effettuato il collegamento tra il proprio Amiga ed un qualunque altoparlante esterno. L'altoparlante presente nel monitor funziona benissimo, ma niente vieta di collegare le uscite audio di Amiga ad un impianto stereo da 200 Watt. Qualunque altoparlante venga utilizzato, è necessario comunque regolare il volume ad un livello appropriato. In seguito verranno realizzati alcuni rumori ed effetti sonori speciali, e non tutti potrebbero gradire, ad esempio, il suono di una sirena d'allarme.

6.1 Un nuovo esempio: Amiga ora parla

Nei capitoli dedicati alla gestione dei file non comparivano programmi di piccole dimensioni. Infatti è quasi impossibile fornire una dimostrazione semplice e allo stesso tempo significativa di come Amiga gestisca i file. Si sarebbe dovuto spiegare ogni dettaglio prima di poter fare qualcosa con i file. Per potersi divertire un pò con Amiga è ora sufficiente introdurre questo semplice programma:

```
Testo$="Amiga"¶
 Partenza:¶
  Campo%(0)=90+180*RND¶
  Campo%(1)=RND¶
  Campo%(2)=50+200*RND¶
  Campo%(3)=RND¶
  Campo%(4)=17000+10000*RND¶
  Campo%(5)=40+24*RND¶
  Campo%(6)=11*RND¶
  Campo%(7)=0¶
  Campo%(8)=0¶
  ¶
  SAY TRANSLATE$(Testo$),Campo%¶
 GOTO Partenza¶
```

E' possibile utilizzare il cassetto Voce in cui salvare questo programma per la prima volta immettendo il comando CHDIR"Voce" prima di registrare il programma.

Quando inizia l'esecuzione, AmigaBASIC richiama il Workbench. Se si possiede un sistema con un solo drive, o se il dischetto del Workbench non è presente in uno dei drive, comparirà una finestra di dialogo che ne richiederà l'introduzione. Dopo una breve pausa necessaria per il recupero delle opportune informazioni, Amiga comincerà a pronunciare il proprio nome utilizzando voci, velocità, toni e volumi differenti.

Amiga ha una personalità con molte facce, non è vero? Se non si vuole che esso parli esclusivamente di sè, è possibile immettere altri valori nella variabile Testo$.

Per imparare a trasformare Amiga in un perfetto annunciatore è sufficiente leggere il paragrafo seguente.

6.2 Ditelo con i fonemi: SAY e TRANSLATE$

Nell'ultimo esempio sono stati impiegati due dei comandi più importanti per la sintesi vocale: SAY e TRANSLATE$. Il comando SAY dà origine alla voce artificiale; esso però non può funzionare da solo. Infatti, se si tenta di far parlare Amiga immettendo nella finestra BASIC un comando del tipo:

```
SAY "Amiga"¶
```

si otterrà il messaggio d'errore "Illegal function call". Accade questo poichè il comando SAY non è in grado di pronunciare direttamente il contenuto di una sequenza di caratteri.

Tradurre il testo per Amiga

Il testo deve per prima cosa essere tradotto in una forma che possa essere compresa da Amiga. Questo è il punto in cui agisce il comando TRANSLATE$:

```
Testo$=TRANSLATE$("Amiga")¶
SAY Testo$¶
```

Questo comando traduce il testo in una forma che può essere maneggiata da SAY, forma chiamata *codice fonetico* che verrà trattata in seguito. Il prossimo tentativo sarà quello di far pronunciare una intera frase ad Amiga.

Si consiglia, per prima cosa, di allargare l'area disponibile sullo schermo:

```
WIDTH 70
```

in caso contrario alcune parole potrebbero scomparire al di là del margine destro dello schermo.

• Dopo aver provveduto a ciò immettere le linee seguenti:

```
Testo$="This is your Amiga speaking. I feel good.
How are you ?" ¶
Testo$=TRANSLATE$(Testo$)¶
SAY Testo$¶
```

Se non si conosce in anticipo il testo della frase, è necessario porre particolare attenzione per capire quanto sta dicendo Amiga, ma ciò diventa più facile man mano che procede il lavoro.

Creare un Amiga bilingue

Se si desidera far parlare Amiga in un'altra lingua rispetto all'originale inglese-americano, ad esempio l'italiano, si possono incontrare dei problemi, dato che il calcolatore non conosce molto bene i linguaggi stranieri. Per rendersene conto, è sufficiente tradurre in italiano il testo della frase inglese sopra riportata:

```
Testo$="Pronto, parla Amiga. Io sto bene, e tu ?"¶
Testo$=TRANSLATE$(Testo$)¶
SAY Testo$¶
```

E' molto probabile che la frase pronunciata sia ben lontana da quanto ci si aspettava.

Codice fonetico

Come ricordato in precedenza, l'istruzione SAY comprende solamente il testo che si presenta codificato in un modo particolare. Questa codifica è conosciuta col nome di codice fonetico. Per vedere come viene tradotta in codice fonetico la frase "Io sto bene, e tu ?", è sufficiente immettere:

```
Testo$="Io sto bene, e tu ?"¶
Testo$=TRANSLATE$(Testo$)¶
SAY Testo$¶
PRINT Testo$¶
```

Sullo schermo apparirà:

```
IH4OW STUW BIXN, IY4 TUW4
```

che rappresenta la stringa di fonemi ottenuta dalla traduzione del testo originale per mezzo di TRANSLATE$. Si può dunque comprendere perchè non bisogna prendersela con SAY se la pronuncia italiana risulta imperfetta. Il colpevole, se di colpevole si può parlare, è TRANSLATE$ che tratta il testo come se fosse sempre in inglese.

Riscrivere il linguaggio

Vi sono due modi per far sì che AmigaBASIC possa pronunciare in modo corretto frasi in italiano. Il primo consiste nel riscrivere il comando TRANSLATE$ per far sì che lavori in modo appropriato anche con lingue diverse dall'inglese-americano. Il secondo modo consiste nel creare i propri fonemi in italiano. Si può anche cercare di ingannare il comando TRANSLATE$, riscrivendo la frase in un "americano spaghetti" così che il risultato si avvicini più o meno all'italiano:

```
Testo$=TRANSLATE$("Ioh stoh beneh, eh thow ?)"¶
SAY Testo$¶
```

Comunque per fare questo correttamente è necessario avere molta familiarità sia con l'inglese che con la lingua che si sta cercando di emulare, altrimenti si dovrà procedere per tentativi. Nel caso sopra riportato, la traduzione in codice fonetico è:

```
IH4AA STAA4 BIXNEH4, EH4 THOW4
```

L'ortografia non vale in questo caso

Il fonema, un'idea mutuata dalla linguistica, indica un "suono". Il codice fonetico è una forma di scrittura che ha molto poco in comune con la corretta ortografia della parola, ma è strettamente legato alla giusta pronuncia della parola stessa. Ad esempio, IY è il fonema che corrisponde ad una "e" lunga in inglese, come nelle parole "bee" (ape) o "meat" (carne); è così evidente che due diversi insiemi di lettere (ee, ea) possono essere rappresentate dallo stesso fonema. Il numero 4 rappresenta un indicatore di quanto vada calcato l'*accento* su una determinata sillaba: il valore di tale indicatore può variare da 0 a 9.

Fonemi e lettere maiuscole

Una cosa molto importante da ricordare quando si creano fonemi perso-

nalizzati riguarda il fatto che essi debbano essere immessi da tastiera facendo uso di lettere maiuscole; una sola lettera minuscola dà origine al messaggio d'errore "Illegal function call".

Si provi a modificare la stringa fonetica prima ottenuta:

```
Testo$="IH9AA STAA3 BIXNEH4, EH4 THOW6¶
SAY Testo$¶
```

Questa versione di "Io sto bene, e tu ?" produce un accento più forte sulla prima sillaba ed un rinforzo di quello presente sull'ultima parola.

Il formato del codice fonetico segue regole ben precise; non è possibile sostituire semplicemente lettere con fonemi quando ciò pare soddisfacente. Se si desidera lavorare con i fonemi in modo più approfondito, il consiglio migliore è quello di leggere l'Appendice H del manuale BASIC fornito con Amiga. In esso sono spiegate in modo esteso le modalità da seguire per costruire appropriati codici fonetici (almeno per quanto riguarda la lingua inglese).

Il narrator.device

Le routine SAY e TRANSLATE$ di AmigaBASIC fanno riferimento al narrator.device presente nel dischetto del Workbench. Questo viene caricato quando si incontrano per la prima volta in un programma AmigaBASIC i comandi SAY e TRANSLATE$. Se si fa ripartire lo stesso programma, Amiga accede immediatamente al Workbench potendo in tal modo utilizzare le routine SAY e TRANSLATE$ non appena vengono chiamate. Il narrator.device occupa circa 30 k di RAM; immettendo un comando NEW o RUN, tale device viene cancellato dalla memoria, venendo successivamente ricaricato ogni volta che si utilizzano i comandi per la sintesi vocale.

NOTA:

Vi sono casi in cui la routine di traduzione dovrebbe essere stata cancellata dalla memoria, ma, poichè Amiga non ha in realtà eseguito tale operazione, essa rimane ancora in memoria. La cooperazione tra AmigaBASIC e Workbench non è così amichevole come qualcuno potrebbe pensare. Un conflitto sorge quando viene eliminata la routine TRANSLATE$ ed ogni comando SAY dà origine all'errore "Illegal function call". Non

esistono rimedi a questo problema, a parte far ripartire il Workbench, nel qual caso è però necessario, prima di far ripartire il sistema, ricordarsi di memorizzare il lavoro svolto.

Il problema sopra accennato si manifesta solitamente quando si utilizzano in modo diretto i comandi per la sintesi vocale dopo aver cancellato un programma che conteneva tali istruzioni.

6.3 Solo parole, niente musica: le opzioni del comando SAY

La cosa veramente interessante nell'esempio riportato nella Sezione 6.1 era la quantità di variazioni possibili della voce emessa da Amiga. Essa può essere alta o bassa, veloce o lenta, accentata o monocorde, maschile o femminile. Se si è fatto eseguire il programma per un periodo di tempo sufficientemente lungo, è possibile aver ascoltato alcuni esempi impressionanti, insieme ad altri molto buffi.

La domanda che viene ora spontanea è come poter sfruttare queste caratteristiche nei propri programmi. Tale operazione non è comunque difficile come potrebbe sembrare. Il numero di opzioni e l'elevato grado delle capacità di Amiga sono tali da poter generare un pò di confusione al primo impatto. Prima di proseguire, è bene cancellare il programma presente nella finestra LIST. Si inizierà a scrivere un programma di prova procedendo sezione per sezione.

Definire una matrice per SAY

Innanzitutto, un principio importante: è possibile specificare una matrice di numeri interi, composta da un massimo di nove elementi, da richiamare dopo il testo nel comando SAY. Gli elementi dopo il nono sono inutili in quanto SAY utilizza solo i primi nove ignorando i restanti. La prima cosa da immettere nella finestra LIST dovrebbe essere:

```
DIM parola%(8)
```

Dovrebbe essere già chiaro come DIM parola%(8) indichi una matrice di nove elementi, dato che il vettore parte in questo caso da 0 e non da 1. Non è una grave dimenticanza tralasciare l'istruzione DIM in questo programma per la sintesi vocale.

Ognuno dei nove elementi è un numero che rappresenta una certa proprietà della voce risultante. In questo programma, tali numeri verranno

acquisiti da linee comprendenti il comando DATA:

```
FOR x=0 TO 8¶
  READ Parola%(x)¶
NEXT x¶
```

Ora si deve indicare un modo per immettere il testo che verrà in seguito pronunciato:

```
ImmettiTesto:¶
  LINE INPUT "Testo:", Immissione$¶
  IF Immissione$<>"=" THEN Testo$=Immissione$¶
  Leggimi$=TRANSLATE$(Testo$)¶
  CLS : PRINT Testo$¶
  SAY Leggimi$, Parola%¶
GOTO ImmettiTesto¶
```

Il programma e' realizzato in modo da consentire la ripetizione della frase precedente inserendo il carattere = e premendo il tasto <Return>.

Con la seguente linea, Amiga avrà una nuova voce:

```
DATA 140,0,160,0,22000,64,11,1,0¶
```

Parametri per la sintesi vocale

Se si prova questo programma, si nota che la voce emessa da Amiga è realmente cambiata; sembra più alta e più dolce della voce originale. I cambiamenti sono stati introdotti dai parametri contenuti nella linea DATA: è quindi ora di esaminare il significato di questi parametri.

Frequenza di base

Il primo valore della matrice specifica la *frequenza di base* della voce. Questo parametro influenza l'acutezza della voce, cioè la sua caratteristica più importante. I valori ammessi per la frequenza di base possono variare tra 65 (voce molto bassa) e 320 (voce molto acuta, stridente); il valore di partenza è 110. Nell'esempio sopra riportato è stato scelto 140, che corrisponde ad una voce poco più acuta del normale. E' possibile specificare qualunque frequenza: è consigliabile effettuare qualche esperimento con i valori

compresi all'interno dell'intervallo citato ed ascoltare i risultati ottenuti.

Se ad esempio si cambia il listato appena presentato, utilizzando come frequenza di base il valore 80, si ottiene una voce molto profonda. Si possono anche impiegare i valori estremi dell'intervallo, cioè 65 e 320.

• Ricercare la frequenza preferita da utilizzare d'ora in avanti.

Inflessione
Il secondo valore della matrice può essere pari a 0 o a 1. Esso determina l'*inflessione* della voce: 0 indica l'inflessione della voce normale, mentre 1 dà come risultato una voce del tutto monocorde o monotona (come quella che si ottiene con i sintetizzatori elettronici). E' preferibile utilizzare la voce umana, ma se si vuole che Amiga possieda una pronuncia da robot, oppure se si vuole ascoltare la differenza fra la voce "normale" di Amiga e quella monocorde, è necessario sapere come fare. Il valore di base di AmigaBASIC per ciò che riguarda questo parametro è 0 (inflessione normale).

Velocità
Con il terzo parametro si possono fare cose molto simpatiche. Esso rappresenta la *velocità di pronuncia* (cioè il numero di parole pronunciate per minuto). Il valore di base utilizzato da AmigaBASIC è pari a 150; nell'esempio sopra riportato il valore 160 fa sì che Amiga parli un po' più velocemente del normale. Si puo' ora provare a cambiare tale valore con 350, immettendo la frase "The big black bug bled black blood" (la grande pulce nera perde sangue nero) o "Unique New York" (New York, l'unica), ascoltando il risultato e tentare poi di pronunciare la frase con la stessa velocità. I valori permessi per la velocità variano tra 40 e 400: alle velocità più elevate è possibile che Amiga si "mangi" le sillabe; analogamente, velocità molto basse (sotto i 70) possono conciliare il sonno.

Tipo di voce
Il quarto valore permette di scegliere il genere della voce emessa, che può essere maschile o femminile. La voce femminile assomiglia molto a quella emessa dal computer di bordo delle navi stellari di alcuni telefilm. Il valore 0 (quello assunto di base) corrisponde ad una voce maschile, men-

tre con il valore 1 viene generata una voce femminile. Per creare una voce che si avvicini a quella femminile reale, è necessario utilizzare valori di frequenza più elevati (attorno a 240).

Tono

Anche il successivo valore deve essere modificato se si vuole sintetizzare una voce femminile (ad esempio, si provi a specificare 23000). Questo valore indica il *tono* della voce, ovvero la frequenza di campionamento. Questi termini possono ora risultare sconosciuti ai più; se ne riparlerà diffusamente nel Capitolo 7: Suono e Musica. Per ora, è sufficiente sapere che questo parametro influenza sia il tono della voce che la frequenza di base; i suoi valori possono essere compresi tra 5000 e 28000. Il valore di base è pari a 22200, e quello dell'esempio è di poco superiore. E' necessario porre particolare attenzione quando si modifica per la prima volta questo valore, perchè anche piccole variazioni possono significare una grossa differenza nella qualità della voce.

Volume

Il sesto valore è molto più semplice da trattare: esso determina infatti il *volume* della voce, che deve essere indicato con un numero compreso tra 0 e 64. Nell'esempio si utilizza il valore più alto, usato anche da AmigaBASIC. Questo parametro è utile qualora si desideri realizzare un programma che contenga conversazioni o voci multiple, fatto che richiede la presenza di volumi differenti. Va infine ricordato che anche il controllo del volume del monitor o dello stereo cui è collegato Amiga sono attivi, per cui il volume della voce sintetizzata dipende, oltre che da questo parametro, anche dalle regolazioni dei dispositivi collegati al calcolatore.

Canale

Il settimo valore imposta l'uscita verso uno o più *canali audio*. E' necessario porre attenzione a questo parametro solo quando si collega Amiga a qualche dispositivo stereofonico; il monitor ha generalmente un altoparlante con uscita monofonica, e tutto viene inviato a quell'unica uscita. Amiga può però avere una uscita audio stereofonica: agli altoparlanti destro e sinistro possono essere assegnati un totale di quattro canali. I canali 0 e 3 (il più basso ed il più alto) rappresentano l'uscita sinistra; quelli 1 e 2 vengono utilizzati per l'uscita destra. E' possibile ottenere effetti stereofonici cambiando in modo corretto l'assegnazione dei canali, oppure

realizzare una "conversazione" tra altoparlante destro e sinistro.

Vi sono alcune limitazioni alle capacità stereofoniche di Amiga: in un sistema stereo, ad esempio, è presente un effetto eco (cioè anche quando viene trasmessa una singola voce, ad ogni altoparlante viene inviato un segnale leggermente differente). Amiga non dispone di questa caratteristica. Inoltre, un normale sistema stereofonico dispone di una traccia di ogni segnale inviato ad ogni altoparlante, cosa di cui Amiga è sprovvisto. Questo può dar luogo ad effetti poco naturali, specialmente se si ascolta la sintesi vocale utilizzando una cuffia: se il suono viene inviato al solo canale sinistro, quello destro rimane completamente muto; nel caso di altoparlanti contenuti nel monitor o in casse acustiche questo effetto è mitigato dall'eco che si genera nella stanza.

Vi sono 12 combinazioni possibili per i canali, come illustrato nella Tabella 10:

Valore settimo elemento matrice	Canale/i	Note
0	Canale 0	sinistra
1	Canale 1	destra
2	Canale 2	destra
3	Canale 3	sinistra
4	Canali 0 e 1	stereo
5	Canali 0 e 2	stereo
6	Canali 3 e 1	stereo
7	Canali 3 e 2	stereo
8	Tutti i canali sinistri liberi	0 e/o 3
9	Tutti i canali destri liberi	1 e/o 2
10	Ogni coppia disponibile	Valore di base
11	Ogni canale libero	-

Tabella 10: Inizializzazione dei canali per il comando SAY.

A proposito di "canali liberi"
Un canale audio può essere già occupato da un programma o comando BASIC utilizzato in precedenza. Accade questo solo se un suono è già in fase di sintesi: un canale è libero quando non c'è alcun suono che sta per essere inviato su di esso.

E' possibile utilizzare un particolare valore fra quelli indicati in tabella per inviare la sintesi vocale verso qualunque combinazione di canali; quando altri comandi SAY o altre istruzioni connesse con la generazione di suoni sono attive, è dunque possibile scegliere una delle opzioni che utilizzano i canali eventualmente liberi.

AmigaBASIC utilizza come valore standard 10 (ogni coppia sinistra/destra libera); nell'esempio sopra riportato si è indicato un valore di 11 (ogni canale libero) nella linea DATA. Si tenga però presente che questi valori hanno un significato solo se Amiga è collegato ad un sistema stereofonico.

Gli ultimi due valori presenti nella matrice servono per indicare come deve comportarsi Amiga quando altri comandi seguono una istruzione SAY.

Modo
L'ottavo valore è il *modo di sincronizzazione*: se il suo valore è 0, AmigaBASIC genera una *uscita vocale sincrona.* Ciò significa che il programma aspetta che venga completato del tutto il comando SAY prima di passare ad eseguire qualunque altro comando dopo tale istruzione. Il valore 1 permette invece di avere una *uscita vocale asincrona*: in questo caso si lavora in modo simile a quello già visto con i comandi OBJECT: il programma inizia semplicemente la sintesi vocale ed esegue i comandi seguenti mentre la voce viene emessa. La sintesi vocale può avvenire "in sottofondo" mentre AmigaBASIC svolge altri compiti. Il modo di base di AmigaBASIC per ciò che riguarda la sincronizzazione è 0.

Controllo
Il nono ed ultimo valore corrisponde al *controllo* (controllo del narrator.device): esso è attivo solo quando il penultimo parametro è posto a 1 (uscita vocale asincrona) per permettere la sintesi vocale in sottofondo. Questo parametro determina il modo in cui Amiga deve comportarsi quando in-

contra un comando SAY mentre è già impegnato in una sintesi vocale: 0 consente di iniziare l'esecuzione di ciascun SAY solo quando è terminato quello precedente; 2 interrompe l'uscita corrente per eseguire subito il nuovo comando SAY. Il valore 1 non ha invece niente a che fare con i valori 0 e 2: tale valore disattiva infatti SAY rendendolo completamente indipendente da quanto sta accadendo nel programma. Tale valore può essere utilizzato in programmi di grandi dimensioni: il comando SAY viene abilitato o disabilitato tramite una variabile il cui valore è 0 per la sintesi vocale e 1 per il silenzio.

Termina qui l'esame delle numerose opzioni disponibili con il comando SAY. L'esperienza dovrebbe insegnare che la sintesi di una voce piacevole viene raggiunta solo attraverso numerosi esperimenti. Comunque, per non sprecare tempo e lavoro (ogni volta che il programma esemplificativo sopra riportato viene eseguito, si crea un nuovo file di dati da inviare al narrator.device), può essere d'aiuto il programma di utilità per la sintesi vocale presentato nella prossima sezione.

6.4 Un programma di utilità per la sintesi vocale

Il programma che viene illustrato in questo paragrafo fornisce un aiuto nell'effettuare esperimenti con le varie opzioni a disposizione nella sintesi vocale. Quando si è trovato un tipo di voce che si pensa possa essere utilizzato in successive occasioni, è consigliabile salvare i parametri definiti in modo tale da poterli poi inserire in altri programmi tramite un comando MERGE.

Amiga deve per prima cosa essere predisposto per visualizzare 80 caratteri per riga, opzione attivabile utilizzando le Preferences.

```
PreparaSchermo:¶
  CLS¶
  PALETTE 0,.1,.1,.4¶
  LOCATE 2,2 : PRINT "Testo:"¶
  LINE (60,5)-(612,18),1,b¶
  LOCATE 22,3¶
PRINT "Frequenza  Velocità   Tono       Volume"¶
  LINE (40,30)-(65,160),1,b¶
  LINE (120,30)-(145,160),1,b¶
  LINE (205,30)-(230,160),1,b¶
  LINE (285,30)-(310,160),1,b¶
  LOCATE 6,50 : PRINT "Maschile   Femminile"¶
  LINE (382,34)-(462,52),1,b¶
  LINE (475,34)-(555,52),1,b¶
  LOCATE 9,50 : PRINT " Umana     Computer"¶
  LINE (382,58)-(462,76),1,b¶
  LINE (475,58)-(555,76),1,b¶
  LOCATE 13,57 : PRINT "Pronuncia"¶
  LINE (430,90)-(535,108),1,b¶
  LOCATE 16,57 : PRINT "Registra"¶
```

```
LINE (430,115)-(535,133),1,b¶
```

Preparazione dello schermo

La routine PreparaSchermo ha come compito quello di realizzare una serie di legende e di accessori. Nella riga posta in alto sullo schermo è possibile inserire il testo che dovrà essere pronunciato da Amiga. Sotto ad essa sono presenti quattro cursori per impostare la frequenza di base, la velocità di pronuncia, il tono ed il volume. Tutti questi parametri possono essere variati sfruttando interamente l'intervallo dei valori consentiti.

Di fianco ai quattro cursori sono presenti un paio di "bottoni" per selezionare una voce maschile o femminile ed uno per determinare il tipo di voce da sintetizzare, umana (con inflessione) o elettronica (senza accenti). Sotto questi bottoni è presente un altro bottone che dà inizio alla sintesi vocale utilizzando i valori correnti mostrati su schermo. L'ultimo bottone in basso va selezionato quando si desidera memorizzare i valori correnti per un uso futuro. La figura 18 mostra come lo schermo dovrebbe apparire, in modo da poter vedere se il programma è stato correttamente inserito:

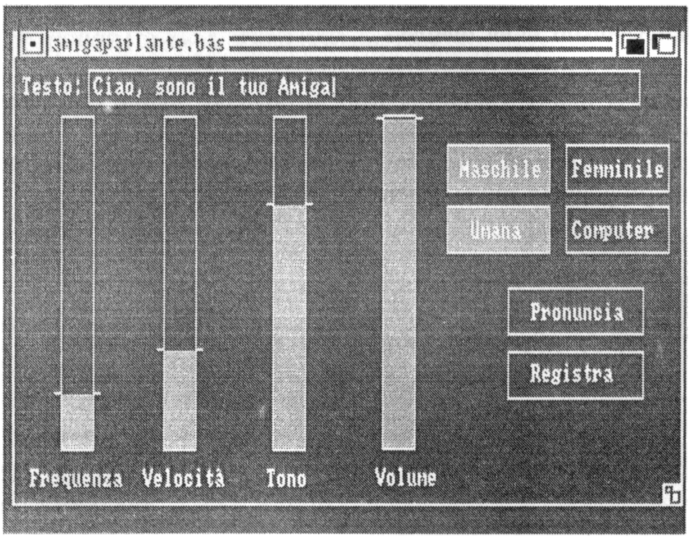


Figura 18: Programma di utilità per la sintesi vocale - schermo principale.

```
ValorePartenza:¶
 FOR x=0 TO 8¶
    READ Parola%(x)¶
 NEXT x¶
 DATA 110,0,150,0,22200,64,10,0,0¶
 GOSUB MostraValore¶
 ¶
CicloPrincipale:¶
 ON MOUSE GOSUB LeggiMouse¶
 MOUSE ON¶
 WHILE 1 : WEND¶
```

ValorePartenza: inserisce nella matrice Parola% i valori base di AmigaBASIC per ciò che concerne il comando SAY. E' possibile cambiare tali valori mentre il programma è in esecuzione, ma è necessario specificare in qualche modo i valori iniziali: è questa la ragione per cui viene usata la riga DATA. Se si desiderano utilizzare valori diversi per l'inizializzazione, è sufficiente modificare la linea DATA senza che ciò causi alcun problema. E' comunque fondamentale che nel programma venga inclusa una linea DATA. Alla fine di questa sezione di programma viene chiamata la subroutine MostraValore: che visualizza i cursori ed i bottoni in relazione ai valori correnti degli elementi della matrice.

Analizzando la routine CicloPrincipale: si dovrebbe intuire che questo programma fa uso dell'intercettamento di eventi. Dato che per l'immissione dei dati sono sufficienti il mouse e la tastiera, non è necessario includere in questo programma menù a tendina.

```
LeggiMouse:¶
 Test=MOUSE(0)¶
 x=MOUSE(3) : y=MOUSE(4)¶
 IF x>39 AND x<311 AND y>29 AND y<161 THEN¶
  IF x<66 THEN¶
   Frequenza:¶
   Parola%(0)=(255-(y-30)*(255/130))+65¶
   ValoreFrequenza=((320-Parola%(0))/255)*130¶
   LINE (41,31)-(64,31+ValoreFrequenza),0,bf¶
   LINE (41,32+ValoreFrequenza)-(64,159),3,bf¶
   y=MOUSE(6)¶
```

```
  IF y<31 THEN y=31¶
  IF y>159 THEN y=159¶
  IF MOUSE(0)<=-1 THEN Frequenza¶
END IF¶
```

Le linee di programma che seguono la linea

```
IF x<66
```

dovrebbero essere ricopiate per tre volte (usando le opzioni Copy e Paste del menù Edit) in quanto queste linee di programma sono usate quattro volte con piccolissimi cambiamenti.

```
IF x>119 AND x<146 THEN¶
  Velocita:¶
  Parola%(2)=(360-(y-30)*(360/130))+40¶
  ValoreVelocita=((400-Parola%(2))/360)*130¶
  LINE (121,31)-(144,31+ValoreVelocita),0,bf¶
  LINE (121,32+ValoreVelocita)-(144,159),3,bf¶
  y=MOUSE(6)¶
  IF y<31 THEN y=31¶
  IF y>159 THEN y=159¶
  IF MOUSE(0)<=-1 THEN Velocita¶
END IF¶
IF x>204 AND x<231 THEN¶
  Tono:¶
  Parola%(4)=(23000-(y-30)*(23000/130))+5000¶
  ValoreTono=((28000-Parola%(4))/23000)*130¶
  LINE (206,31)-(229,31+ValoreTono),0,bf¶
  LINE (206,32+ValoreTono)-(229,159),3,bf¶
  y=MOUSE(6)¶
  IF y<31 THEN y=31¶
  IF y>159 THEN y=159¶
  IF MOUSE(0)<=-1 THEN Tono¶
END IF¶
IF x>284 AND x<311 THEN¶
  Volume:¶
  Parola%(5)=(64-(y-30)*(64/130))¶
```

```
    ValoreVolume=((64-Parola%(5))/64)*130¶
    LINE (286,31)-(309,31+ValoreVolume),0,bf¶
    LINE (286,32+ValoreVolume)-(309,159),3,bf¶
    y=MOUSE(6)¶
    IF y<31 THEN y=31¶
    IF y>159 THEN y=159¶
    IF MOUSE(0)<=-1 THEN Volume¶
  END IF¶
END IF¶
IF x>381 AND x<463 AND y>33 AND y<53 THEN¶
  Parola%(3)=0¶
  PAINT (384,36),3,1 : PAINT (477,36),0,1¶
END IF¶
IF x>474 AND x<556 AND y>33 AND y<53 THEN¶
  Parola%(3)=1¶
  PAINT (477,36),0,1 : PAINT (384,36),3,1¶
END IF¶
IF x>381 AND x<463 AND y>57 AND y<77 THEN¶
  Parola%(1)=0¶
  PAINT (384,60),3,1 : PAINT (477,60),0,1¶
END IF¶
IF x>474 AND x<556 AND y>57 AND y<77 THEN¶
  Parola%(1)=1¶
  PAINT (384,60),0,1 : PAINT (477,60),3,1¶
END IF¶
IF x>59 AND x<613 AND y>4 AND y<19 THEN¶
  LOCATE 2,9 : PRINT  SPACE$(67)¶
  LOCATE 2,9 : LINE INPUT Testo$¶
END IF¶
IF x>429 AND x<536 AND y>89 AND y<109 THEN¶
  PAINT (432,92),3,1¶
  SAY TRANSLATE$(Testo$),Parola%¶
  PAINT (432,92),0,1¶
END IF¶
IF x>429 AND x<536 AND y>114 AND y<134 THEN¶
  PAINT (432,117),3,1¶
```

```
    LOCATE 2,9 : PRINT  SPACE$(67)¶
    LOCATE 2,9 : COLOR 0,3 : PRINT "Nome File:";¶
    COLOR 1,0 : LINE INPUT Nome$¶
    IF Nome$<>"" THEN¶
     IF Nome$="=" OR Nome$="*" AND AltroNome$<>"" THEN
     Nome$=AltroNome$¶
     OPEN Nome$ FOR OUTPUT AS 1¶
      PRINT #1, "REM DATA creati con AmigaParlante in
      AmigaBASIC"¶
      PRINT #1, "DATA ";¶
      FOR x=0 TO 7¶
       PRINT #1,Parola%(x)",";¶
      NEXT x¶
      PRINT #1,Parola%(8)¶
     CLOSE 1¶
     AltroNome$=Nome$¶
    END IF¶
    LOCATE 2,9 : PRINT SPACE$(67)¶
    LOCATE 2,9 : COLOR 1,0 : PRINT  Testo$¶
    PAINT (432,117),0,1¶
   END IF¶
   ¶
RETURN¶
```

Verificare la posizione del mouse

La maggior parte del tempo di esecuzione è dedicata alla routine Leggi-Mouse:. Dapprima viene chiamato MOUSE(0) ed il risultato di tale funzione viene assegnato alla variabile Test; lo scopo è quello di ottenere i valori aggiornati sulla posizione del mouse.

Le funzioni MOUSE(3) e MOUSE(4) sono utilizzate per determinare le coordinate x ed y della posizione del mouse (se non si ricorda il significato di queste funzioni, consultare la Sezione 2.9 o l'Appendice B.4). MOUSE(3) e MOUSE(4) forniscono le coordinate del mouse quando viene premuto il pulsante sinistro; ciò viene utilizzato per avviare una subroutine. Nel programma vengono impiegati questi valori dato che la posizione del mouse è interessante solo quando viene premuto il pulsante sinistro.

A questo punto viene verificata la posizione del puntatore sullo schermo, in riferimento agli accessori presenti. Se il pulsante sinistro del mouse è stato premuto in un punto all'interno di una delle aree associate ad un cursore, il programma richiama la subroutine relativa a quel cursore. Le quattro subroutine Frequenza:, Velocita:, Tono: e Volume: sono molto simili tra loro ad eccezione dei valori, dati e nomi utilizzati. E' importante inserire a questo punto una piccola nota: AmigaBASIC non permette l'uso di vocali accentate nei nomi delle variabili e delle routine; l'uso dell'apostrofo <'> viene infatti interpretato come una istruzione REM: da qui l'uso di Velocita: e non Velocità:

Dato che i quattro cursori lavorano in modo analogo, si eviterà di spiegare il funzionamento di tutti e quattro esaminando solamente il cursore della Frequenza.

Cursori

Se il puntatore del mouse è posizionato sul cursore associato alla Frequenza, il programma calcola innanzitutto il nuovo valore della frequenza di base, inserendolo poi nella relativa posizione all'interno della matrice Parola%, cioè Parola%(0). La variabile ValoreFrequenza contiene il valore della frequenza calcolato in base alle coordinate del mouse. Il primo dei due comandi LINE utilizza l'opzione di riempimento per colorare l'area al di sopra del puntatore con il colore di sfondo (blu scuro), mentre il secondo comando LINE esegue la stessa operazione, riempiendo di arancione l'area al di sotto del puntatore. Come risultato si ottiene una barra verticale che aumenta o diminuisce in base alla frequenza.

y riporta la posizione verticale del mouse quando viene premuto il pulsante sinistro; se y cade al di fuori dell'intervallo consentito (da 31 a 159) il valore corrispondente della frequenza sarà pari al minimo o al massimo, a seconda della posizione del puntatore. Se MOUSE(0) è minore od uguale a -1, significa che il pulsante sinistro è ancora premuto: in tal caso l'indicatore verrà di nuovo riposizionato, in quanto il programma di controllo torna nuovamente all'etichetta Frequenza:.

Accessori

Le routine relative agli altri cursori manipolano i propri valori nello stesso modo, usando ovviamente i propri dati. Per quanto riguarda il "bottone" Maschile, posizionando il puntatore su di esso e schiacciando il pulsante

sinistro, si assegna al parametro Parola%(3) il valore corrispondente, cioè 0. Il bottone diventa arancione, mentre il "bottone" Femminile viene riempito con il colore di sfondo. Il blocco di istruzioni successivo verifica se vi è stato un click del pulsante sinistro del mouse quando il puntatore si trovava sul "bottone" Femminile; se così fosse, Parola%(3) assumerebbe il valore 1, mentre il "bottone" Maschile verrebbe colorato con il colore di sfondo e quello Femminile in arancione.

Selezionando il "bottone" Umano si ottiene una voce con inflessione: Parola%(1) viene posta uguale a 0 ed il pulsante cambia il proprio colore da blu in arancione, mentre Computer risulta "spento". Il "bottone" Computer svolge la funzione opposta: quando viene selezionato la voce viene resa monocorde, Parola%(1) è posto a 1, il "bottone" Umano viene colorato col colore di sfondo e quello Computer diventa arancione.

La routine successiva serve per accettare la sequenza di caratteri immessa e per dare origine alla sintesi vocale. Va notato che l'immissione della stringa va terminata con <RETURN> per poter poi ripristinare la normale intercettazione degli eventi. Se le coordinate del mouse corrispondono ad un punto all'interno del "bottone" Pronuncia, un comando SAY (utilizzando i dati presenti) dà luogo alla sintesi vocale desiderata.

Registrazione dei dati

La routine successiva del programma controlla la memorizzazione del file di dati. Per indicare il nome da associare al file viene utilizzato lo spazio dedicato all'immissione del testo da pronunciare: i caratteri eventualmente presenti vengono cancellati con il comando SPACE$. Il programma legge quindi il nome immesso in questo spazio (= e * possono essere usati come abbreviazioni per indicare l'ultimo nome utilizzato) e, se il nome non è una stringa vuota, apre il file corrispondente.

Dato che il calcolatore non può riconoscere la differenza tra un file sequenziale ed alcune linee di programma BASIC in formato ASCII, è necessario a questo punto ricorrere ad un piccolo trucco. Lo scopo è quello di inserire una serie di comandi in un file che potrà essere in seguito accodato ad un altro programma tramite il comando MERGE.

La linea REM riporta il nome del programma dal quale sono stati generati i dati, mentre la linea successiva contiene l'istruzione DATA seguita da 8

numeri, separati da virgole, estratti da Parola%(x). Il nono valore è scritto direttamente dalla routine, in quanto l'ultimo dato dopo una istruzione DATA non può essere seguito da una virgola. Con questa istruzione si conclude la routine e viene chiuso il file.

AltroNome$ contiene il nome assegnato al file, di modo che la volta successiva possano essere utilizzate le abbreviazioni = e *. E' necessario a questo punto ristabilire le condizioni iniziali: una istruzione SPACE$ cancella il nome del file ed al suo posto viene rimesso il vecchio Testo$; infine PAINT riempie di blu il "bottone" Registra (che durante le operazioni di registrazione dei dati appariva arancione).

Il comando RETURN alla fine della routine LeggiMouse: fa tornare il controllo alla sezione CicloPrincipale:.

Per finire, ecco l'ultima routine che consente di regolare i vari accessori all'inizio del programma, passando loro i valori di inizializzazione dei parametri:

```
MostraValore:¶
 LOCATE 2,9 : PRINT  SPACE$(67)¶
 LOCATE 2,9 : PRINT  Testo$¶
 IF Parola%(3)=0 THEN¶
  PAINT (384,36),3,1 : PAINT (477,36),0,1¶
 ELSE¶
  PAINT (384,36),0,1 : PAINT (477,36),3,1¶
 END IF¶
 IF Parola%(1)=0 THEN¶
  PAINT (384,60),3,1 : PAINT (477,60),0,1¶
 ELSE¶
  PAINT (484,60),0,1 : PAINT (477,60),3,1¶
 END IF¶
¶
 ValoreFrequenza=((320-Parola%(0))/255)*130¶
 LINE (35,31+ValoreFrequenza)-
        (70,31+ValoreFrequenza)¶
 LINE (41,31)-(64,31+ValoreFrequenza),0,bf¶
 LINE (41,32+ValoreFrequenza)-(64,159),3,bf¶
```

```
¶
ValoreVelocita=((400-Parola%(2))/360)*130¶
LINE (115,31+ValoreVelocita)-
      (150,31+ValoreVelocita)¶
LINE (121,31)-(144,31+ValoreVelocita),0,bf¶
LINE (121,32+ValoreVelocita)-(144,159),3,bf¶
¶
ValoreTono=((28000-Parola%(4))/23000)*130¶
LINE (200,31+ValoreTono)-(235,31+ValoreTono)¶
LINE (206,31)-(229,31+ValoreTono),0,bf¶
LINE (206,32+ValoreTono)-(229,159),3,bf¶
¶
ValoreVolume=((64-Parola%(5))/64)*130¶
LINE (280,31+ValoreVolume)-(315,31+ValoreVolume)¶
LINE (286,31)-(309,31+ValoreVolume),0,bf¶
LINE (286,32+ValoreVolume)-(309,159),3,bf¶
¶
RETURN¶
```

Alcune sezioni di questa routine dovrebbero essere ormai familiari, perchè sono già state presentate all'interno della routine LeggiMouse: in una forma molto simile. Per concludere la costruzione della schermata di partenza è necessario riscriverla di nuovo all'interno di un'altra subroutine. Se la subroutine MostraValore: fosse lasciata fuori dal programma, i cursori non apparirebbero a video fino al loro primo impiego. Ogni cursore dispone di un indicatore in corrispondenza della sua posizione originale rendendo in tal modo più semplice ritrovare i valori di partenza.

Se si impiegano valori diversi da quelli qui proposti nella linea DATA, agli indicatori corrisponderanno nuovi valori: il programma considera i valori presenti nella linea DATA come quelli di inizializzazione.

Problemi?
Se Amiga visualizza il messaggio d'errore "Out of memory", è necessario memorizzare il programma, ricaricare il Workbench, far ripartire AmigaBASIC e chiudere tutte le finestre eventualmente aperte (la finestra del disco BASIC, le finestre del Workbench e la finestra LIST); effettuata que-

sta operazione si avrà a disposizione una sufficiente quantità di memoria per eseguire il programma senza problemi.

Uso del programma

Il programma per la sintesi vocale è molto semplice da utilizzare: il mouse viene utilizzato per alzare o abbassare i cursori verticali; dopo aver premuto il pulsante sinistro del mouse quando il puntatore è in corrispondenza del campo Testo: si può immettere da tastiera la sequenza di caratteri che verrà pronunciata da Amiga (ricordando di premere <RETURN> per poter di nuovo utilizzare il mouse). L'intercettazione degli eventi tiene infatti conto della pressione del pulsante sinistro del mouse, ma non può fare assolutamente nulla mentre viene eseguito il comando INPUT.

Si possono selezionare gli opportuni "bottoni" per decidere se la voce emessa dovrà essere maschile o femminile, con inflessione o senza; un click del pulsante sinistro sul "bottone" Pronuncia ordina ad Amiga di "recitare" il testo immesso impiegando i parametri correnti della sintesi vocale. Il programma deve accedere al Workbench prima che Amiga possa emettere la voce.

Se la voce emessa risulta di proprio gradimento, è consigliabile registrare i valori dei parametri correnti selezionando il "bottone" Registra creando in tal modo un file ASCII che potrà essere accodato ad un altro programma tramite un comando MERGE.

E' necessario salvare il programma per la sintesi vocale con le ultime correzioni. Per una dimostrazione vale la pena creare una voce piacevole e salvarne i parametri su dischetto con il nome DatiVoce, ad esempio. Cancellare infine la finestra LIST (impartendo un comando NEW) e battere questo piccolo esempio:

```
FOR x=0 TO 8¶
  READ a%(x)¶
NEXT x¶
SAY TRANSLATE$("Salve Amiga"),a%¶
```

In seguito battere il comando MERGE nella finestra BASIC:

```
MERGE "DatiVoce"¶
```

Ciò aggiungerà al programma due nuove linee:

```
REM DATA creati con AmigaParlante in AmigaBASIC¶
DATA ...¶
```

La linea DATA contiene tutti i valori dei parametri della voce selezionata.

Un suggerimento finale: se si impiega il programma di utilità per la sintesi vocale nello sviluppo di un programma più ampio, vale la pena di impiegare il disco RAM:. E' sufficiente utilizzare un nome del tipo RAM:DatiVoce ed effettuare in seguito il MERGE dei dati dal disco RAM:. In questo modo la velocità è più elevata e non si consuma spazio su dischetto. In ogni caso, se si desidera conservare i propri valori di DATA, è necessario memorizzarli su dischetto o su disco rigido; si può eventualmente collezionare una serie di voci, per poi scegliere quella più adatta con cui far parlare Amiga.

7 Da un F/X speciale alle sinfonie: il suono e la musica

Dopo aver visto le possibilità di Amiga in campo vocale, è il momento di passare in rassegna le sue capacità musicali. L'hardware di Amiga dedicato alla sintesi vocale è così potente e raffinato che può essere facilmente utilizzato anche nella creazione di complessi effetti speciali e di musica.

Vi è tuttavia un piccolo problema a cui è necessario prestare attenzione: AmigaBASIC non sfrutta al 100% le capacità dell'hardware di Amiga.

Tutto quanto messo a disposizione da Amiga fino a questo punto è accessibile mediante comandi BASIC complessi ma potenti; la produzione di suoni e musica è invece sotto il controllo di due comandi apparentemente limitati.

7.1 Il quarto assaggio: la colonna sonora di Guerre Stellari

Tutte le informazioni contenute nei successivi paragrafi sono indirizzate alla produzione di suoni con AmigaBASIC. Chiaramente non si riuscirà a realizzare il tipo di musica ottenibile con programmi scritti esclusivamente per questo scopo, o quella presente nei programmi dimostrativi del Workbench; il prossimo programma mostra comunque cosa è possibile fare in AmigaBASIC senza essere costretti ad un lavoro complesso:

```
TrovaInformazioni:¶
  READ Frequenza,Durata¶
  IF Frequenza=-1 THEN SOUND RESUME : END¶
  SOUND WAIT¶
  SOUND Frequenza,Durata,21,0¶
  SOUND Frequenza/2,Durata,127,1¶
  SOUND Frequenza *2,Durata,21,2¶
GOTO TrovaInformazioni¶
¶
DATA 523.25,15,784,15,698.48,6¶
DATA 659.28,6,587.28,6,1046.52,15¶
DATA 784,17,-1,-1¶
```

Un suono familiare

Il programma realizza la colonna sonora di uno dei film più noti degli ultimi anni: esso riproduce infatti le prime battute del tema musicale di Guerre Stellari. Se per caso la musica non dovesse sembrare fluente e melodica come nel film originale, è utile controllare attentamente le linee DATA: è possibile che sia stato commesso un errore di battitura durante l'inserimento dei dati.

7.2 Musica, maestro!: il comando SOUND

Il comando SOUND gioca un ruolo particolarmente importante nella realizzazione dei suoni. Tale comando è già stato utilizzato nei programmi mostrati in precedenza: nel programma Disegno, ad esempio, SOUND è stato utilizzato per creare un messagio sonoro di avvertimento.

BEEP
Per la produzione di avvertimenti sonori AmigaBASIC mette a disposizione anche il comando BEEP: esso produce il classico beep accompagnato da un lampeggio dello schermo quando l'utente commette qualche errore. BEEP produce un suono con frequenza e durata fissa.

SOUND
Il comando SOUND, viceversa, è molto flessibile sia per quanto riguarda la durata che la frequenza; come si vedrà più avanti, esso offre numerose modalità d'azione.

La musica: un fenomeno fisico
La fisica insegna che la musica e i suoni non sono altro che vibrazioni nell'aria, rilevate e poi elaborate dall'orecchio umano. Per frequenza di un suono si intende il numero di cicli (vibrazioni) che l'onda sonora compie in un secondo; quanto più alto è il numero di cicli per secondo, tanto più acuto è il suono. L'unità di misura della frequenza si chiama hertz, in onore del fisico tedesco Rudolf Hertz a cui si deve la scoperta delle onde elettromagnetiche ed il cui lavoro ha contribuito alla nascita della radio e della televisione.

All'interno di radio, registratori o componenti di un impianto stereofonico i suoni sono codificati come una serie di impulsi elettrici che vengono in-

viati ad uno o più altoparlanti; all'interno di questi ultimi, un magnete consente di far vibrare un cono di carta o di plastica. Questo produce a sua volta vibrazioni nell'aria circostante che l'orecchio percepisce come suoni o musica.

La sintesi sonora su Amiga è realizzata nel modo seguente. Il chip responsabile della produzione dei suoni genera un segnale elettrico corrispondente ad una data frequenza e lo invia ad una o ad entrambe le uscite sonore. Proprio come in un complesso stereofonico, tale segnale deve essere amplificato prima di essere inviato agli altoparlanti. Il monitor di Amiga contiene al suo interno sia un piccolo amplificatore che uno o più altoparlanti.

Il controllo della frequenza

La teoria esposta fino ad ora è sufficiente per fornire una prima infarinatura; val ora la pena di tornare ad Amiga. E' possibile usare il comando SOUND per immettere direttamente la frequenza di un suono. Amiga può produrre suoni con frequenza compresa tra 20 e 1500 hertz: qualunque impianto stereofonico di discreta qualità è in grado di trattare suoni compresi in questo intervallo.

Il controllo della durata

Altro parametro che caratterizza un suono è la sua durata; i valori permessi variano da 0 a 77. 0 non produce alcun suono, mentre con i valori successivi si ottiene un suono tanto più lungo quanto più elevato è il valore: 77 produce un suono lungo circa quattro secondi. Più avanti si vedrà come sia posibile controllare in modo più preciso la durata di un suono. Il comando:

```
SOUND 440,18¶
```

produce un suono con una frequenza di 440 hertz e della durata di circa un secondo. Tale frequenza corrisponde allo standard internazionale per la nota "La", che ogni musicista dovrebbe impiegare per accordare il proprio strumento.

Il comando:

```
SOUND 500,10¶
```

produce un suono un po' più acuto e breve. Se si desidera trovare i valori che corrispondono al BEEP di Amiga, val la pena di provare il seguente comando:

```
SOUND 880,3¶
```

880 hertz è la frequenza per un "La" una ottava sopra al "La" udito in precedenza. Per ascoltare tutto l'intervallo sonoro messo a disposizione da Amiga è sufficiente immettere:

```
FOR x=20 to 1500 STEP 10 : SOUND x,1 : NEXT x¶
```

E' consigliabile abbassare un po' il volume dell'altoparlante prima di premere il tasto <Return>. L'orecchio umano percepisce i suoni con frequenza intermedia come se avessero un volume più alto di quelli con frequenza molto alta o molto bassa; suoni molto acuti a volumi molto elevati possono essere molto sgradevoli da udire.

STEP

Nel ciclo FOR è stato utilizzato un passo di 10 (STEP 10), allo scopo di evitare una lunga attesa prima del termine del programma. Se lo si desidera, si può anche eliminare tale istruzione, per valutare il tempo necessario ad Amiga per concludere il programma.

Con quanto finora presentato, è possibile produrre ogni tipo di beep e di suono; per il momento l'unico modo per trovare il suono desiderato è quello di provare e riprovare.

Si può forse pensare che non sia conveniente modificare tutte le volte il volume dell'altoparlante: in effetti ciò è abbastanza semplice per quanto riguarda il monitor, ma potrebbe non esserlo se l'impianto stereo a cui è collegato Amiga è situato dall'altra parte della stanza (in questo caso si dovrebbe continuamente andare avanti ed indietro per effettuare la regolazione).

Volume

Per risolvere questo problema AmigaBASIC utilizza il terzo parametro del comando SOUND. E' possibile inserire un valore per il volume proprio come nel caso del comando SAY. Questa volta i valori ammessi variano da

0 (nessun suono) a 255 (altezza massima del volume). Se non viene indicato alcun valore, AmigaBASIC impiega quello di base, 127, valore che cade proprio nel mezzo dell'intervallo ammissibile.

Si confronti il comando:

```
SOUND 880,10,48
```

con quest'altro:

```
SOUND 880,10,255
```

Vale la pena di ascoltare lo stesso suono ad un volume sempre più alto:

```
FOR x=0 TO 255 : SOUND 880,1,x : NEXT x
```

Il controllo del volume è molto utile per i beep di avvertimento: è infatti possibile renderli più o meno avvertibili. Per quanto riguarda la musica, il volume è indispensabile perchè la dinamica (cioè l'altezza relativa dei brani musicali nel passaggio tra l'uno e l'altro) è un aspetto importante.

Voce

Il quarto ed ultimo parametro che può essere incluso nel comando SOUND determina quale dei quattro canali audio verrà impiegato dal segnale emesso. A differenza di quanto accade con SAY, SOUND può utilizzare solo un canale alla volta (0, 1, 2, o 3). E' facile ricordarsi che 0 e 3 corrispondono ai canali audio di sinistra, 1 e 2 a quelli di destra; il canale 0 è quello impiegato in partenza.

Per percepire un'uscita sonora che proviene dal canale destro o sinistro è necessario utilizzare un impianto stereofonico, ma è possibile impiegare quattro canali anche in un sistema monofonico. Si possono infatti utilizzare i canali per produrre contemporaneamente quattro note differenti; il programma dimostrativo utilizza, ad esempio, tre note nello stesso istante. Per assicurarsi che tutte le note vengano suonate contemporaneamente è necessario utilizzare due nuovi comandi che verranno esposti in seguito.

In stereofonia
Gestendo sia il controllo del volume che l'assegnazione dei canali, si possono realizzare effetti molto interessanti. Un suono che esce più smorzato dal canale sinistro e più deciso da quello destro fornisce l'impressione di viaggiare da sinistra a destra. Per verificare questo effetto, è sufficiente inserire nella finestra LIST il breve programma che segue:

```
FOR x=0 TO 255¶
  SOUND 440,1,x,0¶
  SOUND 440,1,255-x,1¶
NEXT x¶
```

Aggiungendo poi due istruzioni analoghe, nelle quali l'ordine dei due canali risulta scambiato, è possibile ricreare l'effetto acustico di un incontro di tennis.

A questo punto si dovrebbe avere una certa familiarità con il comando SOUND. Gli esempi finora presentati possono però essere ulteriormente migliorati utilizzando informazioni sulle frequenze che corrispondono alle note musicali. La nota base di riferimento è il "La" con frequenza di 440 hertz. Esistono sette note appartenenti a differenti ottave (una ottava è l'intervallo tra un "Do" ed il "Do" successivo). Il sistema che viene più usato si basa sulla nota "La" con frequenza di 440 hertz. La Tabella 11 riassume alcune informazioni necessarie per comporre musica:

Nome della nota	Frequenza (hertz)
Do (C)	261.63
Re (D)	293.66
Mi (E)	329.63
Fa (F)	349.23
Sol (G)	392.00
La (A)	440.00
Si (B)	493.88
Do (C)	523.25

Tabella 11: Frequenze dell'ottava base

Tra parentesi sono riportati i nomi delle note secondo la notazione anglosassone. Il principio alla base del calcolo delle frequenze è molto semplice. Ad ogni ottava la frequenza raddoppia: nell'ottava immediatamente superiore a quella base, il "La" corrisponde ad 880 hertz, il "Fa" a 698.46 hertz e così via; dividendo i valori per due, si ottengono le note dell'ottava inferiore: il "La" corrisponde a 220 hertz. Note al di sotto del "Do" di questa ottava (130.82 hertz) sono del tutto inutilizzabili per creare musica, perchè sotto i 100 hertz le note producono vibrazioni indistinguibili. La tabella 12 contiene le frequenze musicali delle prime quattro ottave:

Nome della nota	Ottava	Frequenza
Do	1	130.82
Re	1	146.83
Mi	1	164.82
Fa	1	174.62
Sol	1	196.00
La	1	220.00
Si	1	246.94
Do	2	261.63
Re	2	293.66
Mi	2	329.63
Fa	2	349.23
Sol	2	292.00
La	2	440.00
Si	2	493.88
Do	3	523.28
Re	3	587.28
Mi	3	659.28
Fa	3	698.48
Sol	3	784.00
La	3	880.00
Si	3	987.76

Nome della nota	Ottava	Frequenza
Do	4	1046.52
Re	4	1174.52
Mi	4	1318.52
Fa	4	1396.92
Sol	4	1568.00
La	4	1760.00
Si	4	1675.52
Do	5	2093.00

Tabella 12: Comando SOUND: note e rispettive frequenze

Il "Do" da 2093 hertz costituisce già un suono un po' troppo acuto; frequenze ancora più elevate difficilmente possono essere considerate come "musicali".

E' ora possibile scoprire quali note sono state impiegate nell'esempio per realizzare il tema di Guerre Stellari: è sufficiente guardare i valori presenti nelle linee DATA e confrontarli con quelli presenti nella tabella.

Quando si compone musica, si deve procedere nel modo opposto: partendo dalle note del brano musicale si dovrà ricavare dalla tabella le frequenze corrispondenti.

A questo punto sono necessarie le informazioni riguardanti la durata del singolo suono o nota.

Questo parametro non viene misurato in secondi, ma in unità più piccole, (18.2 unità corrispondono ad un secondo).

La Tabella 13 mostra alcuni valori:

Durata in secondi	Unita' di SOUND
0.1	1.8
0.2	3.6
0.4	7.3
0.5	9.1
0.6	10.9
0.8	14.6
1.0	18.2
2.0	36.4
3.0	54.6
4.0	72.8

Tabella 13: Lunghezza dei suoni in secondi e misure equivalenti per il comando SOUND

Questa tabella può essere utilizzata per calcolare qualunque lunghezza: se un suono deve durare 3.5 secondi, basta sommare il valore di 3 secondi a quello di 0.5 (cioè 54.6 + 9.1 = 63.7). Pertanto un valore di durata di 63.7 produrrà un suono lungo tre secondi e mezzo.

Nel programma dimostrativo le durate sono state assegnate ad orecchio: vale la pena di provare ad effettuare modifiche del tempo assegnato a ciascuna nota per vedere l'effetto risultante.

7.3 Scatti e pause: SOUND WAIT e SOUND RESUME

Il comando SOUND permette di suonare più note in sequenza, una dopo l'altra; normalmente un comando SOUND non inizia la sua esecuzione fino a che non sono terminati i suoni prodotti da un altro comando SOUND precedentemente eseguito. Il resto del programma prosegue tranquillamente nella sua esecuzione mentre Amiga sta producendo musica (AmigaBASIC può dedicarsi ad altri compiti). Ciò significa che tutto quello che si deve fare affinchè un motivo venga suonato costantemente è inserire le note nelle posizioni desiderate all'interno del programma, ponendo comunque particolare attenzione al numero di note che si inseriscono. Ogni nota occupa infatti spazio in memoria, e se si tenta di codificare una intera sinfonia si rischia di andare incontro ad un errore di "Out of memory" (dovuto all'occupazione completa della memoria disponibile).

SOUND WAIT

In precedenza è stato affermato che è possibile suonare fino a quattro note contemporaneamente; perchè ciò avvenga è necessario usare un comando che impedisca l'esecuzione immediata delle singole note:

```
SOUND WAIT.
```

SOUND RESUME

Tutti comandi SOUND che seguono una istruzione SOUND WAIT vengono immagazzinati in sequenza senza essere eseguiti. Il comando SOUND RESUME consente di eseguire la sequenza di note immagazzinata dopo un SOUND WAIT in base alla seguente regola: le note che sono assegnate ad un identico canale vengono suonate una di seguito all'altra in sequenza, mentre note che sono assegnate a canali diversi vengono eseguite contemporaneamente.

Ad esempio:

```
SOUND WAIT¶
SOUND 440,10¶
SOUND 880,10¶
SOUND RESUME¶
```

produce come risultato l'esecuzione in sequenza delle note indicate in quanto esse sono entrambe assegnate allo stesso canale (lo 0, il canale di base). Al contrario:

```
SOUND WAIT¶
SOUND 261.63,36,,0¶
SOUND 329.63,36,,1¶
SOUND 392,36,,2¶
SOUND RESUME¶
```

genera tre note suonate simultaneamente, dato che ciascuna è assegnata ad un canale diverso. Se si ha intenzione di comporre pezzi musicali in cui note assegnate a canali diversi possiedono una durata variabile, è necessario riempire i canali non impiegati con note a volume 0, per fare in modo che le note giuste vengano riunite assieme. Una piccola tabella dovrebbe aiutare a chiarire meglio questo concetto:

Canali/ Note	0	1	2	3
1° Nota	523.25	659.28	784.00	-
2° Nota	784	987.76	1174.52	-
3° Nota	698.48	0	0	-
4° Nota	659.28	0	0	-
5° Nota	587.28	0	0	-
6° Nota	1046.52	1318.52	1568	-
7° Nota	784	987.76	1174.52	-

Tabella 14: Note e canali

Se i canali 1 e 2 non fossero riempiti con note "nulle" tra la terza e la quinta, le relative note nella sesta riga verrebbero eseguite assieme alla terza nota del canale 0. E' sufficiente consultare attentamente la tabella per rendersi conto di quanto affermato.

• Per ascoltare il risultato inserire nella finestra LIST il programma seguente:

```
SOUND WAIT¶
¶
TrovaInformazioni:¶
 READ Frequenza,Durata,Volume,Canale¶
 IF Frequenza=-1 THEN Suona:¶
 SOUND Frequenza,Durata,Volume,Canale¶
GOTO TrovaInformazioni:¶
¶
Suona:¶
 SOUND RESUME¶
¶
DATA 523.23,15,127,0, 659.28,15,96,1, 784,15,96,2¶
DATA 784.15,15,222,0, 987.76,15,180,1,
1174.52,15,180,2¶
DATA 698.48,6,127,0, 0,6,0,1, 0,6,0,2¶
DATA 587.28,6,127,0, 0,6,0,1, 0,6,0,2¶
DATA 1046.52,15,225,0, 1318.52,15,180,1,
1568,15,180,2¶
DATA 784,24,180,0, 987.76,24,160,1, 1174.52,24,160,2¶
DATA -1,-1,-1,-1¶
¶
```

Le linee DATA corrispondono esattamente alla tabella riportata sopra. Il ciclo TrovaInformazioni: legge tutti i valori per il comando SOUND immagazzinandoli in attesa della loro esecuzione. Quando si incontra il valore -1 per la Frequenza, il programma si rende conto che non esistono altri dati, ed esce dal ciclo.

Se la melodia risultasse troppo acuta, è sufficiente dimezzare i valori della frequenza dopo averli acquisiti:

```
SOUND Frequenza/2,Durata,Volume,Canale
```

In tal modo il motivo viene trasferito nell'ottava inferiore. E' consigliabile non utilizzare frequenze molto alte.

I comandi SOUND e BEEP consentono di potersi esercitare nella composizione di musica; è comunque possibile realizzare qualcosa di ancor più interessante.

7.4 Tendere l'orecchio: un poco di teoria acustica

E' ora utile prendere in considerazione ancora un poco di teoria e di nozioni fondamentali in modo da poter meglio comprendere quello che verrà spiegato in seguito.

E' già stato rilevato come un suono (o una nota) costituisca fisicamente una vibrazione nell'aria: pertanto ogni dispositivo che generi un qualunque tipo di suono deve essere in grado di produrre le giuste vibrazioni per poterlo realizzare.

Vibrazioni

In uno strumento musicale la vibrazione viene prodotta da una corda (chitarra, violino, pianoforte), da una membrana od una superficie dura (strumenti a percussione), oppure da una colonna d'aria (organo, tromba o altri strumenti a fiato). Un impianto stereofonico, gli strumenti elettronici e i calcolatori impiegano un oscillatore per produrre vibrazioni elettroniche. Un amplificatore invia quindi all'altoparlante un segnale elettrico, più forte o più debole a seconda della vibrazione, basato sul segnale dell'equipaggiamento collegato. Le caratteristiche del suono dipendono dalle qualità della vibrazione.

Per comprendere meglio le caratteristiche delle vibrazioni dell'aria, è possibile sfruttare il paragone con i cerchi che si creano in uno specchio d'acqua quando si getta un sasso. Il cambiamento nella densità dell'aria ha un comportamento simile a quello delle increspature sulla superficie dell'acqua.

La forma d'onda tipica di un suono è l'onda sinusoidale, simile a quella descritta nel Capitolo 2.

Onde sinusoidali

La figura 19 riporta un'onda sinusoidale. La curva inizia a livello della linea mediana, si innalza, ritorna alla linea mediana, continua a cadere, quindi risale nuovamente. Quando raggiunge di nuovo la linea di mezzo, un ciclo è stato completato. Una vibrazione del genere avviene 440 volte in un secondo per realizzare il "La" dell'ottava di base. Più vibrazioni si hanno in un secondo, più acuto risulta il suono.

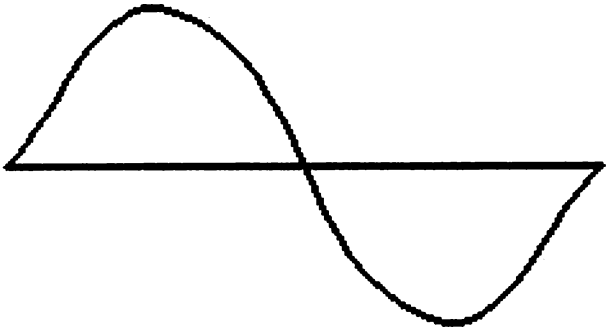


Figura 19: Rappresentazione su assi cartesiani di un'onda sinusoidale

Le vibrazioni sotto una certa frequenza, attorno ai 20 hertz, non vengono percepite come suoni. L'altezza di un suono dipende dalla frequenza delle vibrazioni. E' ancora possibile avvertire vibrazioni a bassa frequenza quando sono molto ampie, come nel caso di una porta sbattuta.

Ampiezza

Anche il volume dipende dalla vibrazione. Più alta è l'onda, più distintamente viene percepito il suono. L'altezza di una onda, cioè la distanza tra il punto più alto dell'onda stessa e la linea di mezzo, viene chiamata ampiezza. Il volume di un suono dipende dall'ampiezza dell'onda.

Timbro
Altre proprietà dell'onda influenzano il tipo di suono prodotto. La forma d'onda determina anche il timbro, o colore, del suono di un particolare strumento musicale. Maggiori dettagli in proposito sono contenuti nella prossima sezione.

Musica analogica e digitale
Fino a poco tempo fa veniva impiegato solamente il metodo analogico per registrare e riprodurre la musica. Quando si ascolta un disco, la puntina del giradischi vibra, e l'amplificatore traduce tali vibrazioni in segnali elettrici che vengono inviati in uscita agli altoparlanti. I registratori a nastro e a cassetta funzionano con lo stesso principio, se si esclude il fatto che i nastri hanno un segnale magnetico variabile che la testina del registratore è in grado di leggere.

Recentemente si è invece affermata la tecnica della registrazione digitale, che su Amiga viene utilizzata per la gestione dei suoni. Amiga è stato costruito per immagazzinare ed elaborare numeri: la registrazione, l'immagazzinamento ed il riascolto dei suoni vengono effettuati rappresentando le vibrazioni analogiche sotto forma di valori numerici (digitali).

Principi di base della registrazione digitale
Il principio alla base della registrazione digitale è molto semplice: è sufficiente osservare come sono registrati ed immagazzinati i suoni su un CD. Il valore associato ad una vibrazione viene registrato e convertito in un numero migliaia di volte al secondo. Effettuando frequentemente una registrazione di tale valore, si ottiene una serie di valori numerici che rappresenta accuratamente la forma d'onda. Questi numeri possono quindi venire utilizzati per riprodurre n ovamente il suono. La frequenza (tono), il volume (altezza) e la forma dell'onda possono quindi essere suddivisi in frammenti e successivamente ricomposti.

Il processo di rappresentazione delle vibrazioni utilizzando numeri viene detto campionamento (sampling); il realismo di un suono campionato dipende soprattutto dalla frequenza di campionamento, ovvero dal numero di letture dell'ampiezza d'onda effettuate in una unità di tempo. La frequenza di campionamento su Amiga può arrivare fino a 30000 valori al secondo, mentre un lettore di CD effettua più di 40000 campionamenti al

secondo dei dati su disco.

Paula

Il processo di riascolto, o di generazione, della musica viene semplicemente realizzato in modo inverso rispetto al processo di registrazione. Il chip di Amiga responsabile della sintesi sonora (chiamato Paula) converte i valori numerici presenti in memoria in segnali elettrici analogici. A questo scopo Paula è provvista di una serie di convertitori analogico/digitale (A/D). Questo circuito elettronico consente di generare da un numero digitale (memorizzato ad esempio in un byte) un segnale elettrico la cui intensità corrisponde al numero stesso. Il segnale elettrico generato viene inviato alla porta d'uscita audio di Amiga e da qui all'altoparlante. In questo modo viene realizzato un suono digitale. Dato che Paula dispone di quattro di questi convertitori, si hanno a propria disposizione quattro canali stereofonici su cui lavorare.

7.5 Splish, splash: le forme d'onda

Come detto in precedenza la profondità (o altezza) ed il volume di un suono dipendono dalle caratteristiche dell'onda sonora: la frequenza determina l'altezza, mentre l'ampiezza influisce sul volume. La forma dell'onda è però talvolta ancora più importante: essa infatti determina le caratteristiche tonali di una nota. Se si è campionato il suono originato da uno strumento musicale o da qualunque altro generatore di note e si rappresenta graficamente quanto campionato, è possibile vedere che ogni strumento dispone di una forma d'onda caratteristica. Un'onda sinusoidale viene percepita come dolce ed armonica; in natura, ad ogni buon conto, non esistono onde sinusoidali pure. Strumenti a percussione come le batterie producono invece forme d'onda irregolari, con picchi acuminati.

Più è alta la frequenza di campionamento, migliore risulta la riproduzione della forma d'onda originale. Una frequenza di campionamento molto elevata richiede però una notevole quantità di memoria. Un Compact Disk, che può contenere fino a 70 minuti di musica di alta qualità acustica, ha una capacità di immagazzinamento dati pari a 550 Megabyte (Mb).

WAVE

AmigaBASIC mette a disposizione dell'utente solo due comandi per la sintesi sonora: SOUND, che è già stato ampiamente descritto, e WAVE, che consente di modificare la forma dell'onda sonora.

Normalmente AmigaBASIC utilizza un'onda sinusoidale. Se si vuole impiegare un'altra forma d'onda, è necessario creare, per prima cosa, una matrice di numeri interi contenente almeno 256 elementi. I valori dei numeri della matrice possono variare da -128 a +127; ciascun elemento rappresenta un valore di conversione analogico/digitale a 8 bit. I valori digitali ottenuti dalla conversione sono ovviamente numeri a 8 bit.

Il valore -128 all'interno della matrice indica il punto di minimo dell'onda; viceversa +127 costituisce il valore massimo

Onda triangolare

Esistono altre forme d'onda oltre a quella sinusoidale, ognuna delle quali fornisce un risultato tonale differente. La prima forma da esaminare è quella triangolare, simile a quella sinusoidale ma più appuntita (la vibrazione cambia direzione bruscamente, anzichè gradualmente come accade invece nell'onda sinusoidale). La Figura 20 riporta un esempio di onda triangolare.

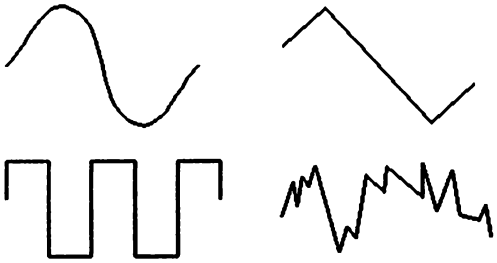


Figura 20: Rappresentazione di alcune differenti forme d'onda

Il programma seguente impiega una forma d'onda triangolare con i dati per la riproduzione a tre voci del tema di Guerre Stellari.

- Inserire le linee riportate di seguito prima di quelle precedentemente inserite:

```
DIM OndaTriangolare%(255)¶
 FOR x=0 to 62¶
  a=a+2¶
  OndaTriangolare%(x)=a¶
 NEXT x¶
 FOR x=63 TO 188¶
```

```
  a=a-2¶
  OndaTriangolare%(x)=a¶
NEXT x¶
FOR x=189 TO 255¶
  a=a+2¶
  OndaTriangolare%(x)=a¶
NEXT x¶
¶
WAVE 0,OndaTriangolare%¶
WAVE 1,OndaTriangolare%¶
WAVE 2,OndaTriangolare%¶
¶
ERASE OndaTriangolare%¶
```

Per prima cosa viene definita la matrice intera OndaTriangolare%; il primo ciclo FOR...NEXT calcola i valori degli elementi che fanno parte di questa matrice dal punto di partenza a quello più alto della curva, il secondo ciclo computa i valori per gli elementi che man mano scendono verso il punto più basso, ed infine il terzo si prende cura del ritorno verso il punto di partenza.

Cambio di forma d'onda

I comandi WAVE fanno sì che le tre voci impiegate nel programma corrispondano alla matrice OndaTriangolare%. Se lo si desidera, è possibile impiegare una forma d'onda diversa per ciascun canale audio di cui dispone Amiga. Ogni istruzione WAVE assegna al proprio canale una sezione di memoria, per l'immagazzinamento dei valori di campionamento. Quando questa memoria è stata allocata, non è più necessario mantenere i valori della matrice OndaTriangolare%, a meno che non si intenda utilizzarli in seguito. Dato che nel programma per la creazione del tema di Guerre Stellari la matrice non viene più riutilizzata, essa viene eliminata con un comando ERASE. Nel caso si stia scrivendo un programma molto lungo, oppure un programma che utilizza numerose forme d'onda, è consigliabile procedere alla cancellazione delle matrici dopo averle assegnate ai canali audio; in caso contrario verrebbe inutilmente sprecata memoria.

Eseguendo il programma è possibile notare come i suoni siano leggermente cambiati rispetto a quelli del precedente programma. Va ricordato che

il nuovo tipo di forma d'onda rimane attivo anche quando si torna al modo di immissione diretta dei comandi: ciò consente di effettuare alcuni semplici e rapidi esperimenti sfruttando la finestra BASIC. Il suono emesso da un comando BEEP risulta ora diverso.

• Per tornare al vecchio suono BEEP, inserire la linea:

```
WAVE 0, SIN¶
```

Questo comando consente di associare una forma d'onda sinusoidale al canale 0. SIN non è una matrice di numeri interi, ma un'opzione del comando WAVE che permette di ritornare rapidamente all'onda sinusoidale di base.

Onde quadrate

Un altro tipo di forma d'onda è l'onda quadrata (o quadra) che consiste in una serie di salti da una ampiezza ad un'altra, e ciò dà origine ad un suono secco, "legnoso". In Figura 20 è riportato il grafico di un'onda quadra. Se si desidera utilizzare tale tipo d'onda per ascoltare il tema di Guerre Stellari, è sufficiente memorizzare la versione precedente del programma (contenente un'onda triangolare) e successivamente modificare le linee della routine iniziale in questo modo:

```
DIM OndaQuadra%(255)¶
FOR x=0 TO 127¶
  OndaQuadra%(x)=127¶
NEXT x¶
FOR x=128 TO 255¶
  OndaQuadra%(x)=-128¶
NEXT x¶
¶
WAVE 0,OndaQuadra%¶
WAVE 1,OndaQuadra%¶
WAVE 2,OndaQuadra%¶
¶
ERASE OndaQuadra%¶
```

Il resto del programma non necessita di modifiche. Un'onda quadra è facile da realizzare: nella prima metà degli elementi è inserito il valore +127

(il più alto possibile), nella seconda metà è presente invece -128 (il valore più basso ammesso).

Rumore

Il quarto tipo di forma d'onda è quello associato ad un rumore. L'onda non segue alcun tipo di regola: il suono generato è completamente "casuale". Il risultato assomiglia talora ad un campanello elettrico o all'allarme di una radio sveglia. Utilizzando frequenze più basse è anche possibile imitare esplosioni od altri effetti sonori. Nella Figura 20 è riportata la forma d'onda tipica di un rumore.

```
DIM Rumore%(255)¶
FOR x=0 TO 255¶
  Rumore%(x)=RND*255-128¶
NEXT x¶
¶
WAVE 0,Rumore%¶
WAVE 1,Rumore%¶
WAVE 2,Rumore%¶
¶
ERASE Rumore%¶
```

La formula RND*255-128 permette di generare un numero casuale compreso tra -128 e +127, cioè nell'intervallo concesso per i valori di campionamento.

Sono state descritte le varie forme d'onda riproducibili con il calcolatore: l'onda triangolare fornisce un suono più acuto e chiaro dell'onda sinusoidale; l'onda quadra sembra legnosa, ma chiara; il rumore di solito non produce risultati particolarmente brillanti.

E' interessante notare come le note più basse si sentano meglio con un'onda quadra o con un rumore rispetto a quanto accade con un'onda sinusoidale o triangolare.

7.6 Un gran finale: SinthAmiga, un programma di utilità per la sintesi sonora

Il programma per la sintesi sonora che verrà presentato di seguito non deve essere considerato un programma di sostituzione di programmi commerciali quali Instant Music, DeLuxe Music Construction Set o Sonix. Con l'impiego del solo AmigaBASIC non è possibile produrre musica paragonabile a quella che si può realizzare con l'uso dei pacchetti sopra citati o di altri programmi dedicati.

Questo programma di utilità è dedicato alla produzione di vari tipi di forme d'onda con cui effettuare esperimenti. Dopo aver individuato una forma di proprio gradimento, è possibile memorizzare su memoria di massa (dischetto o disco rigido) la matrice degli elementi che definiscono l'onda, per utilizzarla successivamente all'interno dei propri programmi con il comando WAVE. Il programma risulta essere il seguente:

```
Inizializzazione:¶
   DIM FormaOnda%(255)¶
        DEF FNYFormaOnda(a)=ABS(FormaOnda%(a)-128)¶
¶
        SCREEN 1,320,200,2,1¶
        WINDOW 2,"Forma d'onda",(0,0)-(256,63),22,1¶
¶
        FOR x=0 TO 256¶
           FormaOnda%(x)=127*SIN(x/20)¶
        NEXT x¶
¶
¶
        WINDOW 3,"Funzione",(175,80)-(310,185),22,1¶
        WINDOW OUTPUT 3¶
        LINE (5,5)-(65,30),1,b¶
```

```
        PSET (5,17)¶
        FOR x=0 TO 58¶
           LINE -((x+5),17-10*SIN(x/3.8))¶
        NEXT x¶
        LINE (69,5)-(130,30),1,b¶
        LINE (71,18)-(79,7) : LINE -(95,27)¶
        LINE -(111,7) : LINE -(127,27)¶
        LINE (5,40)-(65,65),1,b¶
        LINE (7,52)-(7,42)¶
        LINE -(18,42) : LINE -(18,62)¶
        LINE -(30,62) : LINE -(30,42)¶
        LINE -(41,42) : LINE -(41,62)¶
        LINE -(52,62) : LINE -(52,42)¶
        LINE -(63,42) : LINE -(63,52)¶
        LINE (69,40)-(130,65),1,b¶
        LOCATE 7,10 : PRINT  "Annulla"¶
        LINE (5,75)-(65,100),1,b¶
        LOCATE 11,3 : PRINT "Salva"¶
        LINE (69,75)-(130,100),1,b¶
        LOCATE 11,10 : PRINT "Carica"¶
  ¶
        WINDOW OUTPUT 2¶
        GOSUB MostraOnda¶
¶
¶
        ON MOUSE GOSUB ControlloMouse¶
        MOUSE ON¶
¶
        WINDOW 3¶
¶
    PressioneTasto:¶
        a$=INKEY$¶
        F=0¶
        IF a$="" THEN F=0 : GOTO PressioneTasto¶
        IF a$=CHR$(9) THEN F=261.63¶
        IF a$="1" THEN F=277.18¶
```

```
        IF a$="q" THEN F=293.66¶
        IF a$="2" THEN F=311.13¶
        IF a$="w" THEN F=329.63¶
        IF a$="e" THEN F=349.23¶
        IF a$="4" THEN F=369.99¶
        IF a$="r" THEN F=392!¶
        IF a$="5" THEN F=415.3¶
        IF a$="t" THEN F=440!¶
        IF a$="6" THEN F=466.16¶
        IF a$="y" THEN F=493.88¶
        IF a$="u" THEN F=523.25¶
        IF a$="8" THEN F=554.37¶
        IF a$="i" THEN F=587.58¶
        IF a$="9" THEN F=622.25¶
        IF a$="o" THEN F=659.28¶
        IF a$="p" THEN F=698.48¶
        IF a$="-" THEN F=739.99¶
        IF a$="[" THEN F=784!¶
        IF a$="=" THEN F=830.61¶
        IF a$="]" THEN F=880¶
        IF a$=CHR$(93) THEN F=932.33¶
        IF a$=CHR$(13) THEN F=987.76¶
        IF a$=CHR$(139) THEN F=1046.52¶
        IF a$=CHR$(27) THEN SCREEN CLOSE 1 :
             CLEAR : END¶
        IF F=0 THEN PressioneTasto¶
¶
    Suona:¶
        Vol=127 : IF F=0 THEN l=0¶
        SOUND WAIT¶
        SOUND F,3,Vol,0¶
        SOUND F,3,Vol,1¶
        SOUND RESUME¶
    GOTO PressioneTasto¶
¶
    ControlloMouse:¶
```

```
      IF WINDOW(0)=2 THEN CambiaFormaOnda¶
      IF WINDOW(0)=3 THEN CambiaFunzione¶
   RETURN¶
¶
   CambiaFormaOnda:¶
      WINDOW 2¶
      WHILE MOUSE(0)<0¶
         x=MOUSE(5)¶
         IF x>256 THEN GOSUB MostraOnda : RETURN¶
         IF x<1 THEN x=1¶
         y=MOUSE(6)¶
         IF y> 63 THEN GOSUB MostraOnda :RETURN¶
          LINE (x-1,FNYFormaOnda(x-1)/4) -
                (x,FNYFormaOnda(x)/4),0¶
          LINE (x-1,FNYFormaOnda(x-1)/4)-(x,y),1¶
          FormaOnda%(x)=127-(y*4)¶
      WEND¶
      GOSUB MostraOnda¶
   RETURN¶
¶
   CambiaFunzione:¶
      Test=MOUSE(0)¶
      x=MOUSE(3)¶
      y=MOUSE(4)¶
      IF x>4 AND x<66 AND y>4 AND y<31 THEN¶
         WINDOW 3 : PAINT (7,7),3,1¶
         FOR x=0 TO 256¶
            FormaOnda%(x)=127*SIN(x/20)¶
         NEXT x¶
         GOSUB MostraOnda¶
         WINDOW 3 : PAINT (7,7),0,1¶
      END IF¶
      IF x>68 AND x<131 AND y>4 AND y<31 THEN¶
         WINDOW 3 : PAINT (71,7),3,1¶
         FOR x=0 TO 256¶
            IF x<41 THEN FormaOnda%(x)=x*3 : a=x*3¶
```

```
        IF (x>=41 AND x<126) OR (x>=210) THEN
                a=a-2.57 : FormaOnda%(x)=a¶
        IF x>=126 AND x<210 THEN a=a+2.57 :
                FormaOnda%(x)=a¶
    NEXT x¶
    GOSUB MostraOnda¶
    WINDOW 3 : PAINT (71,7),0,1¶
END IF¶
IF x>4 AND x<66 AND y>39 AND y<66 THEN¶
    WINDOW 3 : PAINT (6,41),3,1¶
    FOR x=0 TO 256¶
        IF x<64 OR (x>=128 AND x<191) THEN
                FormaOnda%(x)=127¶
        IF (x>=64 AND x<128) OR x>192 THEN
                FormaOnda%(x)=-128¶
    NEXT x¶
    GOSUB MostraOnda¶
    WINDOW 3 : PAINT (6,41),0,1  ¶
END IF¶
IF x>68 AND x<131 AND y>39 AND y<66 THEN¶
    WINDOW 3¶
    PAINT (71,42),3,1¶
    FOR x=0 TO 256¶
       FormaOnda%(x)=0¶
    NEXT x¶
    GOSUB MostraOnda¶
    WINDOW 3 : PAINT (71,42),0,1¶
END IF¶
IF x>4 AND x<66 AND y>74 AND y<101 THEN¶
    WINDOW 3¶
    PAINT (7,77),3,1¶
    GOSUB ImmettiNome¶
   IF Nome$="" THEN PAINT (7,77),0,1 : RETURN¶
    OPEN Nome$ FOR OUTPUT AS 1¶
    FOR x=0 TO 256¶
       PRINT #1,CHR$(127-FormaOnda%(x));¶
```

```
            NEXT x¶
            CLOSE 1¶
            WINDOW 3 : PAINT (7,77),0,1¶
         END IF¶
         IF x>68 AND x<131 AND y>74 AND y<101 THEN¶
            WINDOW 3¶
            PAINT (71,77),3,1¶
            GOSUB ImmettiNome¶
           IF Nome$="" THEN PAINT (71,77),0,1 : RETURN¶
            OPEN Nome$ FOR INPUT AS 1¶
            FOR x=0 TO 256¶
               FormaOnda%(x)=127-ASC(INPUT$(1,1))¶
            NEXT x¶
            CLOSE 1¶
            WINDOW 3 : PAINT (71,77),0,1¶
            GOSUB MostraOnda¶
         END IF¶
      RETURN¶
¶
      MostraOnda:¶
         WINDOW 2 : CLS¶
         FOR x=1 TO 256¶
            LINE (x-1,FNYFormaOnda(x-1)/4)-
                  (x,FNYFormaOnda(x)/4),1¶
         NEXT x¶
         WINDOW 3¶
         WAVE 0,FormaOnda%¶
         WAVE 1,FormaOnda%¶
      RETURN¶
¶
      ImmettiNome:¶
         WINDOW 4,"Immetti il nome del file:",(5,100)-
                (300,110),0,1¶
         CLS : LINE INPUT Nome$ ¶
         IF Nome$="=" OR Nome$="*" THEN
               Nome$=AltroNome$¶
```

```
        IF Nome$<>"" THEN AltroNome$=Nome$¶
        WINDOW CLOSE 4 : WINDOW 3¶
    RETURN¶
```

- Dopo aver terminato l'immissione, procedere immediatamente, alla memorizzazione su dischetto, per evitare di incorrere in deprecabili incidenti ...

Nella routine Inizializzazione: vengono innanzitutto definite le dimensioni della matrice FormaOnda%, in modo da avere a disposizione 256 elementi per la forma d'onda da provare.

DEF FN

Il comando successivo, DEF FN, non è stato finora utilizzato nel libro, in quanto non era ancora emersa la necessità di un suo utilizzo. E' un comando utile nei programmi dove devono essere eseguiti molti calcoli. DEF FN è l'abbreviazione di DEFine FunctioN (Definisci la funzione): esso permette la definizione di funzioni personalizzate all'interno dei programmi BASIC.

y=SIN(x) è una funzione già esistente in AmigaBASIC. Tuttavia, se si desidera impiegare all'interno di un programma la funzione:

```
y=SIN(x)*COS(x)+0.5*(SIN(x)-COS(x))¶
```

un certo numero di volte, risulta molto scomodo dover ribattere l'intera linea ogni volta. E' invece possibile risparmiare tempo e lavoro, rendendo inoltre il programma molto più comprensibile, inserendo una linea come la seguente:

```
DEF FNCalcolo(x)=SIN(x)*COS(x)+0.5*(SIN(x)-COS(x))¶
```

all'inizio del programma, in modo tale da poter specificare FNCalcolo anzichè immettere quella lunga formula. Il nome della funzione è Calcolo; il prefisso FN permette di distinguerla dalle normali variabili. Se si immette PRINT FNCalcolo(1), AmigaBASIC calcola il risultato della funzione con x=1 e lo visualizza.

Il nome della funzione è seguito dall'elenco di uno o più parametri e dalla formula espressa in termini di questi parametri. I nomi delle variabili

presenti nella linea di definizione della funzione possono essere usati senza alcun problema nel resto del programma. La definizione di una funzione non produce alcuna modifica sulle variabili utilizzate come parametri, in quanto essi vengono utilizzati solo come mezzo per esprimere la procedura da realizzare.

Per comprendere meglio il funzionamento di quanto esposto può essere utilizzato questo semplice esempio:

```
DEF FNDoppio(x)=2*x.¶
```

In qualunque punto del programma compaia la linea FNDoppio(2), AmigaBASIC calcola l'espressione definita nella funzione Doppio sostituendo alla x il valore 2 presente nella linea di chiamata della funzione. Se si richiama una funzione che non è stata definita tramite una DEF FN, si ottiene il messaggio di errore "Undefined user function".

DEF FN nel programma SinthAmiga

Nel programma che è stato appena presentato la funzione FNYFormaOnda viene utilizzata per convertire i valori impiegati per la definizione degli elementi della matrice in numeri interi compresi tra 0 e 255. Infatti, ogni elemento della matrice che identifica una forma d'onda deve avere un valore compreso tra -128 e +127; se si sottrae 128 da ognuno di questi valori e si elimina l'eventuale segno negativo, si possono ottenere numeri compresi tra 0 e 255. In BASIC, tutto ciò viene tradotto nella linea DEF FNYFormaOnda(a)=ABS(FormaOnda%(a)-128).

Il programma sfrutta uno schermo a bassa risoluzione (320 x 200); in esso viene creata una finestra con lo scopo di visualizzare il tipo di forma d'onda corrente.

Il programma per la sintesi sonora impiega un'onda sinusoidale come forma d'onda iniziale. Quest'onda è un po' diversa da quella che AmigaBASIC considera come onda base.

Il ciclo FOR ... NEXT nella sezione Inizializzazione: del programma consente di calcolare i valori iniziali dell'onda sinusoidale.

Forme d'onda a scelta
In una seconda finestra vengono presentati una serie di pulsanti da selezionare per specificare determinate opzioni. Un primo pulsante consente di scegliere un'onda sinusoidale, un secondo un'onda triangolare, un terzo una di tipo quadro. Per realizzare questi pulsanti si è fatto uso dell'istruzione LINE; all'interno del primo è tracciato il profilo di un'onda sinusoidale, mentre nel secondo e nel terzo sono raffigurate rispettivamente l'onda triangolare e quella quadra. Sono poi presenti altri pulsanti: uno per cancellare completamente l'onda corrente, uno per registrare su memoria di massa la matrice utilizzata da WAVE ed infine uno per recuperare tali dati memorizzati.

Se la finestra 3 (quella contenente i pulsanti) è inattiva, il programma chiama la subroutine MostraOnda: per visualizzare la forma dell'onda corrente (sinusoidale quando il programma inizia l'esecuzione) nella finestra 2 (quella superiore, più estesa). La finestra di output per tutte le altre funzioni è la numero 3: essa deve essere resa attiva quando si desiderano produrre suoni (è sufficiente selezionare tale finestra dopo aver modificato la forma dell'onda presente nella finestra 2). AmigaBASIC può ricevere dati da tastiera solamente da una finestra, e la numero 3 è adibita a questo scopo; l'altra serve solo per l'intercettazione degli eventi riguardanti posizione e stato del mouse.

La tastiera
La routine PressioneTasto: consente di verificare se è stato premuto un tasto e assegna l'eventuale immissione alla variabile a$. Nel programma la tastiera di Amiga simula quella di un pianoforte: i tasti <TAB>, <Q>, <W>, <E>, ... <HELP> rappresentano i tasti bianchi, mentre i tasti numerici soprastanti corrispondono ai tasti neri del pianoforte. La pressione del tasto <ESC> consente di chiudere lo schermo del programma e le sue finestre (SCREEN CLOSE 1), di cancellare tutte le variabili e tutti gli elementi della matrice FormaOnda% tramite il comando CLEAR e quindi terminare il programma (END).

Le opzioni disponibili sono:

- Suona: ha il compito di sintetizzare la nota specificata dal tasto premuto; le note così create vengono inviate una al canale sinistro, l'altra al destro: entrambe iniziano nello stesso momento grazie ad una istruzione SOUND RESUME. Ciò significa che, se Amiga è collegato ad un impianto stereofo-

nico, la musica verrà diffusa da entrambi gli altoparlanti.

- ControlloMouse: impiega la funzione WINDOW(0) per determinare in quale finestra è localizzato il puntatore quando viene premuto il pulsante sinistro del mouse; a seconda del risultato viene chiamata la routine CambiaFormaOnda: o quella CambiaFunzione:.

- CambiaFormaOnda: definisce come finestra di output la numero 2; è possibile utilizzare il mouse all'interno di tale finestra per immettere una forma d'onda a piacere disegnandola come in un programma grafico; l'onda presente in precedenza viene cancellata e sostituita da quella tracciata a mano.

Forme d'onda combinate

Se si sposta il puntatore fuori dalla finestra 2 (IF x>256 o IF y>63) la subroutine MostraOnda: combina in modo appropriato la parte di curva appena disegnata con quella presente in precedenza che non è stata cancellata, rimuovendo dallo schermo tutto quanto non è più significativo. Il programma torna alla sezione CambiaFormaOnda: dopo il comando RETURN.

I comandi LINE cancellano i vecchi segmenti dell'onda (utilizzando il colore di sfondo, cioè 0) e tracciano quelli nuovi con il colore 1 (quello di primo piano). Infine, all'elemento di FormaOnda% che corrisponde alla coordinata orizzontale del puntatore alla pressione del pulsante sinistro del mouse viene assegnato il valore corretto. La variabile y (la coordinata verticale del puntatore) ha un valore compreso tra 0 e 63, corrispondenti agli estremi della finestra; per ottenere valori compresi tra -128 e +127 viene impiegata la formula 127-(y*4).

Per finire, la subroutine MostraOnda: viene richiamata al termine della sezione CambiaFormaOnda:. Alcune parti della vecchia forma d'onda, pur non essendo più comprese nella matrice FormaOnda%, possono rimanere sullo schermo anche dopo il tracciamento della nuova onda, soprattutto se il mouse viene mosso rapidamente nella fase di disegno della nuova onda. Per tale ragione, il programma cancella lo schermo e traccia una nuova onda dopo ogni modifica. Oltre a ciò MostraOnda: consente di assegnare la matrice degli elementi ai due canali audio di uscita tramite WAVE.

Cambiare funzione

La sezione successiva è quella etichettata come CambiaFunzione:, il cui compito è quello di verificare se è stato premuto il pulsante sinistro del mouse all'interno della finestra 3 (il cui nome è Funzione) e in quale punto si trovava il puntatore quando tale evento si è verificato, per poter agire di conseguenza. Infatti le variabili x e y contengono la posizione del puntatore in occasione dell'ultima pressione del tasto sinistro del mouse; la posizione del puntatore in altri istanti non è importante per cui non viene registrata.

Il primo blocco IF...END IF controlla se il pulsante sinistro del mouse è stato premuto in corrispondenza del pulsante dell'onda sinusoidale; se ciò risulta vero, il programma assegna agli elementi di FormaOnda% i valori dell'onda sinusoidale, quindi passa alla subroutine MostraOnda:, ed infine ritorna al punto di partenza. Tutte le volte che si seleziona un pulsante in questa finestra, il suo colore cambia prima che venga iniziata l'esecuzione delle istruzioni relative alla scelta effettuata, per ritornare al colore di partenza quando tali istruzioni sono terminate. La routine che verifica se è stato selezionato il pulsante dell'onda triangolare esegue un compito simile a quello già descritto nel caso dell'onda sinusoidale: la variabile a in questo caso viene incrementata e poi diminuita in modo che i valori degli elementi della matrice FormaOnda% siano quelli propri di un'onda triangolare. Anche la selezione del pulsante dell'onda quadra fornisce un risultato similare: in tal caso però gli elementi della matrice FormaOnda% assumono dapprima il valore +127, poi -128, quindi +127 ancora ecc.

Il pulsante Annulla ha come risultato quello di azzerare tutti gli elementi della matrice FormaOnda%, cancellando dunque la forma d'onda corrente.

Registrazione e recupero dei dati

Rimangono ora da esaminare i pulsanti Salva e Carica. Se si seleziona il primo, il programma apre un file sequenziale per la scrittura dei dati; se invece si sceglie il secondo, il file sequenziale viene aperto in lettura. La routine ImmettiNome: è uguale a quella già incontrata nei programmi esposti precedentemente ed il suo compito è quello di prendersi cura dell'immissione del nome del file (da registrare o recuperare).

La routine MostraOnda: consente di visualizzare la forma dell'onda corrente all'interno della finestra 2. Anche in questo caso la funzione

FNYFormaOnda viene impiegata per calcolare le coordinate di ciascun punto dell'onda. Il programma calcola le coordinate verticali dividendo per 4 i valori che spaziano tra 0 e 256. I comandi WAVE posti alla fine della subroutine fanno sì che la forma d'onda corrente (mostrata nella finestra) venga utilizzata per la sintesi delle note.

La routine ImmettiNome: richiede all'utente di immettere il nome di un file. Si possono utilizzare i caratteri = o * come abbreviazioni per indicare di utilizzare l'ultimo nome immesso. Per l'immissione del nome del file viene impiegata per breve tempo un'altra finestra, che viene successivamente rimossa.

Uso del programma

L'utilizzo del programma è molto semplice: è sufficiente prendere il mouse, portare il puntatore sulla finestra 2 (Forma d'onda), premere il tasto sinistro e tracciare, sempre col tasto premuto, la forma d'onda desiderata. Dopo un po' di esperimenti è possibile comprendere come differenti forme d'onda possano fornire risultati diversi.

Per udire le note sintetizzate da Amiga, basta utilizzare le prime due file di tasti al di sotto dei tasti funzione; sfortunatamente è possibile ascoltare solo una nota per volta, in quanto il calcolatore può accettare la pressione di un singolo tasto alla volta.

Se si desidera richiamare una forma d'onda "standard" da impiegare per la sintesi sonora o da modificare a piacimento, è sufficiente selezionare uno dei pulsanti destinati a queste operazioni presenti nella finestra 3 (Funzioni); è anche possibile iniziare da zero, dopo aver selezionato il pulsante Annulla: dopo aver compiuto tale scelta non è più possibile avere alcuna sintesi di note fino a quando non si specifica una nuova forma d'onda.

Quando si è individuata una forma d'onda di proprio gradimento, è consigliabile salvarla su memoria di massa (dischetto o disco rigido) selezionando il pulsante Salva; in questo caso bisogna anche indicare il nome da assegnare al file da registrare. L'operazione inversa può essere effettuata selezionando Carica: dopo aver indicato il nome del file da recuperare, in modo da poter riutilizzare i valori precedentemente registrati.

E' possibile recuperare tali valori all'interno dei propri programmi tramite

una routine come la seguente:

```
OPEN <NomeFile> FOR INPUT AS 1¶
   FOR x=0 TO 256¶
          FormaOnda%(x)=127-ASC(INPUT$(1,1))¶
      NEXT x¶
    CLOSE 1¶
```

In seguito è sufficiente ricordare di assegnare la matrice che descrive la forma d'onda ai canali audio che devono impiegarla. Si possono assegnare forme d'onda differenti a canali audio diversi; ogni comando SOUND impiega la forma d'onda del canale audio che utilizza.

Per concludere il programma e ritornare alla finestra BASIC è sufficiente premere il tasto <ESC>.

Appendici

A Messaggi di errore e di aiuto

Questo capitolo contiene un elenco alfabetico di tutti i messaggi d'errore che si possono incontrare quando si lavora con AmigaBASIC. Quando si incontra un numero di errore mentre si sta programmando o lavorando con questo libro, è consigliabile cercare in questa appendice per scoprire le possibili cause d'errore e avere suggerimenti per superare il problema.

AmigaBASIC è in grado di scoprire alcuni errori prima della partenza del programma; altri li può trovare solo quando il programma è in esecuzione. In entrambi i casi, l'errore verrà visualizzato nella finestra LIST, e una finestra di dialogo apparirà per avvertire della presenza di un errore. Prima di poter correggere l'errore in questione bisognerà selezionare il bottone OK nella finestra di dialogo. I numeri di errore elencati in questo capitolo sono utilizzati con il comando ON ERROR e la variabile di sistema ERR. L'appendice B.4 (comandi per il controllo del programma) contiene maggiori informazioni a questo proposito.

Gli errori che AmigaBASIC scopre mentre un programma è in esecuzione sono riportati in questa appendice in lettere maiuscole associate ai rispettivi numeri d'errore. Gli errori trovati prima che il programma venga eseguito sono mostrati in lettere sia maiuscole che minuscole con un trattino (-) al posto del numero di errore.

ADVANCED FEATURE ***Numero errore: 73***

Questo messaggio appare solo se si utilizza un comando che non è utilizzato in questo libro. Esso compare quando AmigaBASIC riconosce come legittimo un comando, ma scopre che non è stato ancora implementato nella versione in uso di AmigaBASIC. Non si è a conoscenza di comandi di tal genere. Questo errore è una eredità delle versioni precedenti del BASIC Microsoft.

ARGUMENT COUNT MISMATCH *Numero errore: 37*

Questo messaggio viene visualizzato quando si chiama un sottoprogramma con troppi o troppo pochi argomenti/valori/testo. E' allora necessario esaminare la prima linea del sottoprogramma per trovare quanti e quali tipi di argomenti sono necessari. Quando si richiama il sottoprogramma in questione vanno forniti tanti parametri quanti sono richiesti.

BAD FILE MODE *Numero errore: 54*

Questo messaggio di errore viene visualizzato quando si tenta di usare alcuni comandi che non possono essere impiegati per quel determinato tipo di file. Di seguito vengono riportate alcune possibili cause di questo errore:

Tentativo di utilizzare MERGE per accodare un programma ad un altro che non era stato registrato in formato ASCII. E' necessario richiamare il programma in questione e registrarlo utilizzando SAVE "NomeProgramma",A per essere in grado di fondere i due file insieme.

Tentativo di caricare in memoria un file ad accesso casuale con il comando LOAD.

Tentativo di utilizzare comandi per la gestione di file ad accesso casuale (ad esempio, GET o PUT) su un file sequenziale.

Immissione di un modo che non sia "I", "O", "A" o "R" quando si usa il comando OPEN.

BAD FILE NAME *Numero errore: 64*

Si è indicato il nome di un file in un formato non corretto (nome troppo lungo o non accettabile per AmigaBASIC). La sezione 3.2 descrive il formato dei nomi associati ai file. Il formato di base è:

"(nome del dispositivo o del drive):(directory)/(directory)/ .../(nomefile)"

BAD FILE NUMBER *Numero errore: 52*

Solitamente questo errore indica che si è fornito un numero di file non ancora aperto a comandi come PRINT# o INPUT#. E' necessario riguardare i comandi OPEN contenuti nel programma per ritrovare i numeri dei file effettivamente aperti.

BAD RECORD NUMBER *Numero errore: 63*

Si è tentato di accedere con un comando PUT o GET ad un record utilizzando un numero non accettabile; l'intervallo dei numeri riconosciuti per i record è compreso tra 0 e 16777215.

BLOCK ELSE/END IF
must be the first statement on the line *Numero errore: -*

I due comandi ELSE ed END IF devono essere i primi ed unici comandi di una linea. In caso contrario AmigaBASIC non è in grado di riconoscere la struttura IF/ELSE/END IF. Il problema può essere di solito evitato scrivendo il codice di un programma in modo strutturato (stile Pascal). Questo errore è riconosciuto prima che il programma parta.

CAN'T CONTINUE *Numero errore: 17*

Questo messaggio compare dopo aver impartito il comando CONT, oppure quando si sceglie CONTINUE dal menù RUN. Il programma può continuare la sua esecuzione solo se non sono stati rilevati errori e se il programma non è stato modificato dopo l'interruzione. Per far ripartire un programma qualora si sia verificata una delle due evenienze sopra riportate è necessario utilizzare il comando RUN.

DEADLOCK *Numero errore: 77*

Non sono disponibili dati sicuri riguardanti questo errore: pur esistendo il messaggio d'errore, non è stato possibile recuperare la documentazione

specifica. E' probabilmente un messaggio del sistema operativo e non di AmigaBASIC. Potrebbe trattarsi di una specie di errore interno.

DEVICE I/O ERROR *Numero errore: 57*

Il sistema operativo di Amiga non è stato in grado di completare una operazione di input/output. La causa potrebbe essere un problema tecnico, ad esempio con un disco. Solitamente è necessario ricaricare il Workbench quando si incontra questo errore.

DEVICE UNAVAILABLE *Numero errore: 68*

Il dispositivo in questione non è collegato oppure non è acceso. AmigaBASIC non può pertanto accedervi. Si può essere costretti a ricaricare il Workbench.

DISK FULL *Numero errore: 61*

Si incontra questo errore quando si tenta di registrare programmi o dati su un dischetto dove non vi è abbastanza spazio a disposizione. E' necessario allora cancellare qualcosa che non serve più (copiando eventualmente altrove i programmi o i dati prima di cancellarli), oppure utilizzare un altro dischetto.

Se si incorre in questo errore quando si sta lavorando con il dischetto Extras è consigliabile, come indicato nel Capitolo 1 di questo libro, dare un'occhiata alla Sezione 3.1

DIVISION BY ZERO *Numero errore: 11*

Non è ammessa la divisione di un numero per 0. Si incorre in questo errore quando si esegue un comando come PRINT 1/0, oppure utilizzando una variabile cui non è stato ancora assegnato un valore, oppure per un errore in una formula. E' anche possibile che sia dovuto ad un errore di battitura di una variabile.

DUPLICATE DEFINITION *Numero errore: 10*

Questo errore si manifesta quando si tenta di ridimensionare un vettore (o matrice) che è già stato dimensionato in precedenza. La matrice in questione non può essere ridimensionata in quanto si è già utilizzato il comando DIM per quel nome di variabile, oppure perchè si è già utilizzata la matrice prima del comando DIM. In questo caso è possibile cancellare la matrice utilizzando il comando ERASE. E' possibile ricavare i limiti superiore ed inferiore della matrice utilizzando UBOUND e LBOUND. Se si utilizza il comando OPTION BASE subito dopo il primo DIM di un programma, si avrà questo errore.

DUPLICATE LABEL *Numero errore: 33*

E' stata assegnata la medesima etichetta (label) a due distinte sezioni dello stesso programma. E' possibile che AmigaBASIC consideri come etichetta qualcosa che l'utente non ritiene tale: in tal caso è sufficiente eliminare i due punti alla fine del testo incriminato. E' anche possibile che l'utente abbia confuso un sottoprogramma con una routine (subroutine) o viceversa. E' possibile assegnare ad un sottoprogramma un nome già utilizzato per una etichetta (anche se è una cattiva abitudine, perchè rende il listato di difficile comprensione), ma in questo caso è necessario ricorrere al comando CALL per evitare che il programma faccia confusione.

ELSE/ELSE IF/END IF without IF *Numero errore: -*

Uno di questi comandi è stato inserito nel programma anche se non esiste un corrispondente IF che inizi in modo corretto un blocco IF/END IF. Forse questo IF è stato inserito dopo THEN sulla linea che contiene un altro IF; non è possibile fare ciò all'interno di un blocco IF/END IF, altrimenti il programma non sarà in grado di distinguere tra i due possibili usi del comando IF. AmigaBASIC individua questo errore prima che il programma vada in esecuzione.

EXIT SUB outside of a subprogram *Numero errore: -*

Il comando EXIT SUB può essere contenuto solo all'interno di un blocco

di istruzioni che inizia con SUB e termina con END SUB. END SUB può essere stato inserito troppo presto nel blocco, oppure l'utente può aver scambiato i comandi EXIT SUB ed END SUB. AmigaBASIC individua questo errore prima che il programma vada in esecuzione.

FIELD OVERFLOW *Numero errore: 50*

Questo errore è dovuto al fatto che, all'interno di un file ad accesso casuale, si è assegnato, tramite l'istruzione FIELD, un numero di byte di memoria buffer superiore a quello stabilito in partenza con il comando OPEN per la lunghezza del file. La struttura del file dovrebbe essere attentamente progettata prima della sua realizzazione, in modo tale che l'utente sappia sempre quanti byte avrà a disposizione.

FILE ALREADY EXISTS *Numero errore: 58*

L'utente ha immesso, con i comandi SAVE o NAME, un nome di file già presente. E' necessario introdurre un altro nome, oppure, se si desidera mantenere quello in questione, utilizzare CHDIR per spostarsi in un'altra directory.

FILE ALREADY OPEN *Numero errore: 55*

Questo messaggio appare quando si tenta di aprire un file che è già stato aperto. AmigaBASIC visualizza questo messaggio anche quando si cerca di eliminare con il comando KILL un file che è attualmente aperto. Prima di eliminare un file con KILL, è necessario chiuderlo.

FILE NOT FOUND *Numero errore: 53*

Il file che si sta cercando non è presente. L'utente può aver immesso un nome errato, oppure ci si trova nella directory errata; usare FILES per visualizzare il contenuto della directory, CHDIR per cambiare directory.

FOR WITHOUT NEXT ***Numero errore: 26***

AmigaBASIC ha trovato un FOR senza un corrispondente NEXT. Si può aver dimenticato NEXT, oppure aver commesso un errore quando si è inserita la variabile da conteggiare. Risultati migliori si ottengono se i cicli FOR ... NEXT sono ben allineati e si evita di porre il nome della variabile dopo NEXT.

IF without END IF ***Numero errore: -***

Se si utilizza un blocco IF/END IF, è necessario inserire alla fine del blocco un END IF. Questo errore avviene quando l'utente dimentica di inserire END IF alla fine del blocco in questione, oppure quando, pur non volendo utilizzare il blocco IF/END IF, l'utente non ha messo niente dopo il THEN sulla linea che contiene IF. In questo caso, AmigaBASIC interpreta la linea come se facesse parte di un blocco IF/END IF. Se non si desidera mettere alcun comando dopo THEN, ma non si vuole utilizzare il blocco IF/END IF, è sufficiente inserire un comando REM.

Poichè questo errore viene individuato prima che il programma vada in esecuzione, può accadere mentre l'utente sta sviluppando un programma e non ha ancora terminato di finire una linea.

ILLEGAL DIRECT ***Numero errore: 12***

Alcuni comandi possono essere utilizzati solo all'interno di programmi e non possono essere usati in modo diretto. Si ottiene questo errore quando si immettono comandi come DEF FN, COMMON, SUB all'interno della finestra BASIC.

ILLEGAL FUNCTION CALL ***Numero errore: 5***

Si incorre in questo errore quando si forniscono valori non corretti ad un comando AmigaBASIC, come valori negativi per indicare elementi di una matrice, oppure associati a comandi come GET e PUT.

Questo errore capita anche quando l'utente tenta di fornire ad un comando AmigaBASIC parametri troppo grandi, troppo piccoli, o del tipo errato. Se il messaggio di errore compare dopo un LIST, il programma è stato salvato in forma protetta (SAVE "nomeprogramma",P).

INPUT PAST END *Numero errore: 62*

Questo messaggio appare quando si tenta di leggere qualcosa dopo la fine di un file, oppure si tenta di leggere un file aperto solo in output. Usare i comandi EOF e LOF per evitare di incorrere in tale errore.

INTERNAL ERROR *Numero errore: 51*

Non si può fare nulla in questo caso, in quanto AmigaBASIC ha incontrato un errore interno: è necessario caricare nuovamente il programma AmigaBASIC.

LINE BUFFER OVERFLOW *Numero errore: 23*

Questo messaggio viene visualizzato quando l'editor di schermo nella finestra LIST ha qualche problema. L'utente può aver tentato di inserire troppi caratteri su una singola linea, che ne può contenere al massimo 255.

La causa può anche essere la presenza di un errore nell'editor di schermo; è consigliabile allora provare ad usare CUT e PASTE (opzioni del menù EDIT): spesso l'errore scompare da sè.

MISSING OPERAND *Numero errore: 22*

Questa segnalazione compare quando manca un valore dopo un comando come AND, OR, XOR o dopo un segno di calcolo (+, -, *, /, ^ ecc.).

Missing STATIC in SUB statement *Numero errore: -*

Ogni SUB necessita di un comando STATIC. L'utente deve immettere STATIC, in quanto altrimenti AmigaBASIC visualizzerà questo messaggio prima che il programma vada in esecuzione.

NEXT WITHOUT FOR *Numero errore: 1*

AmigaBASIC ha incontrato una istruzione NEXT che non era stata preceduta dalla corrispondente istruzione FOR. Questo messaggio è il logico errore complementare al messaggio FOR WITHOUT NEXT (cui si rimanda per ulteriori informazioni).

NO RESUME *Numero errore: 19*

AmigaBASIC non è in grado di trovare un comando RESUME all'interno di una routine ON ERROR (dedicata al trattamento degli errori). RESUME è richiesto per chiudere la routine e ritornare al luogo dove si è verificato l'errore.

OUT OF DATA *Numero errore: 4*

Si è raggiunta la fine delle istruzioni DATA mentre se ne stanno leggendo i valori con READ DATA. Non vi sono più altri valori DATA disponibili. Il programma dovrebbe essere in grado di riconoscere da sè questa situazione. L'utente può, ad esempio, utilizzare il valore -1 come ultimo valore dei DATA, e confrontare i dati letti con questo valore. Non va dimenticato che si può cambiare il puntatore ai DATA tramite il comando RESTORE.

OUT OF HEAP SPACE *Numero errore: 14*

La prima volta che compare questo messaggio l'utente può spaventarsi un po', in quanto viene mostrato in rosso sullo schermo. Il messaggio compare quando non vi è abbastanza memoria per permettere al programma di svolgere i compiti assegnati. E' necessario premere il pulsante sinistro

del mouse per poter continuare. L'uso del comando CLEAR può essere d'aiuto: se la situazione non muta, è necessario registrare il programma, far ripartire il Workbench e selezionare di nuovo AmigaBASIC. E' a questo punto indispensabile interrompere tutti i programmi e chiudere tutte le finestre che non sono in relazione con il programma in esecuzione in quanto, quando non vi sono altri programmi in memoria, si ha a disposizione uno spazio maggiore per il proprio programma.

OUT OF MEMORY *Numero errore: 7*

Questo messaggio avverte che è stata utilizzata tutta l'area di memoria riservata ad AmigaBASIC. Ciò non significa necessariamente che il sistema non ha più memoria a sua disposizione, ma solo che il programma e i suoi dati occupano troppo spazio. Il comando CLEAR può spesso essere una soluzione in questi casi (vedere anche OUT OF HEAP SPACE più sopra).

OVERFLOW *Numero errore: 6*

Questo errore indica che un numero è troppo grande per essere rappresentato nel formato numerico prescelto (vedere la Nota d'uso 5 per la spiegazione dei formati numerici). L'utente può solitamente evitare errori di questo tipo utilizzando variabili in doppia precisione.

PERMISSION DENIED *Numero errore: 70*

Questo errore accade solitamente quando si tenta di scrivere su un dischetto che è protetto da scrittura. Se si è sicuri di voler scrivere su di esso, è sufficiente estrarre il dischetto dal drive quando la luce del led è spenta, togliere la protezione da scrittura (facendo scivolare l'apposita protezione) e quindi reinserire il dischetto nel drive.

RENAME ACROSS DISKS *Numero errore: 74*

L'utente ha tentato di inserire il nome di un dischetto anzichè quello di un file in risposta ad un comando NAME. Non è possibile cambiare il nome del dischetto, in quanto il risultato sarebbe una operazione di copia e non

di cambiamento del nome di un file.

RESUME WITHOUT ERROR *Numero errore: 20*

AmigaBASIC ha incontrato un comando RESUME anche se non vi è stata alcuna segnalazione d'errore. Il programma è probabilmente entrato accidentalmente in un ciclo ON ERROR. Il programma principale deve essere chiuso prima di cicli e subroutine ERROR, oppure deve avere dei salti prima dei blocchi ON ERROR/RESUME. Nè le subroutine, nè i blocchi ON ERROR/RESUME vengono saltati automaticamente da AmigaBASIC.

RETURN WITHOUT GOSUB *Numero errore: 3*

AmigaBASIC ha trovato un comando RETURN senza che in precedenza sia stato eseguito un GOSUB. Accade questo quando un programma entra accidentalmente in una subroutine. Va ricordato che le subroutine non vengono eseguite automaticamente, al contrario di quanto accade per i sottoprogrammi. Il programma dovrebbe prevedere un salto in un'altra area sull'ultima linea di istruzioni prima della subroutine.

SHARED outside of a subprogram *Numero errore: -*

Il comando SHARED può essere utilizzato solo all'interno di un sottoprogramma. L'istruzione END SUB potrebbe essere stata inserita troppo presto, oppure sono stati scambiati i comandi EXIT SUB ed END SUB.

Statement illegal within subprogram *Numero errore: -*

Alcuni comandi non possono essere impiegati all'interno di un sottoprogramma: ad esempio, non si possono utilizzare DEF FN, COMMON, o CLEAR. L'utente non deve utilizzare tali comandi all'interno di un blocco delimitato dalle istruzioni SUB e END SUB.

STRING FORMULA TOO COMPLEX *Numero errore: 16*

Si ottiene questo messaggio se i comandi per la manipolazione delle sequenze di caratteri (stringhe) come MID$, LEFT$, RIGHT$ ecc. sono nidificati l'uno dentro l'altro in modo eccessivo. Conviene suddividere le proprie formule in linee diverse.

STRING TOO LONG *Numero errore: 15*

AmigaBASIC supporta sequenze di caratteri (stringhe) lunghe fino a 32767 caratteri; si va incontro a questo errore se una stringa supera tale limite, nel qual caso è necessario suddividere la stringa in una serie di stringhe di minori dimensioni.

SUB already defined *Numero errore: -*

Questo errore avviene quando l'utente ha definito due sottoprogrammi con lo stesso nome; in tal caso è necessario cambiare il nome ad uno dei due sottoprogrammi.

SUBPROGRAM ALREADY IN USE *Numero errore: 36*

Un sottoprogramma può richiamare un altro sottoprogramma, ma non autochiamarsi.

Ci si imbatte in questo errore anche quando si esce prematuramente da un sottoprogramma che viene in seguito nuovamente richiamato. In tal caso, è necessario terminare il vecchio sottoprogramma con CONT, oppure far ripartire il programma con RUN.

SUBSCRIPT OUT OF RANGE *Numero errore: 9*

L'utente incontra questo errore quando tenta di accedere ad un elemento di una matrice che è situato oltre l'intervallo stabilito con il dimensionamento automatico. L'errore può anche essere dovuto al fatto di introdurre

dimensioni superiori a quelle dichiarate precedentemente con l'istruzione DIM.

Tale errore è frequentemente dovuto ad errori di battitura nel nome di una matrice. Può anche darsi che l'utente stia accedendo ad un elemento di una matrice usando una variabile con valore errato.

SUB without END SUB *Numero errore: -*

I comandi SUB ed END SUB vanno a braccetto: si manifesta questo errore quando ad un SUB non fa seguito un END SUB. La programmazione strutturata aiuta ad evitare problemi di questo genere.

SYNTAX ERROR *Numero errore: 2*

AmigaBASIC ha interpretato una parte del comando, ma la restante non è corretta: l'ordine degli argomenti può essere sbagliato, oppure essi sono utilizzati in un contesto errato. Può esservi un numero disuguale di parentesi destre e sinistre. Si ottiene un errore di questo tipo anche se si tenta di leggere tramite READ una stringa da una linea DATA e associarla ad una normale variabile. Consultare l'appendice B se non si è sicuri della sintassi di un determinato comando.

TOO MANY FILES *Numero errore: 67*

In altre versioni di BASIC realizzate dalla Microsoft, questo messaggio indica che ci sono troppi file su dischetto. Questo non accade con Amiga, in quanto il dischetto ha una organizzazione diversa rispetto ad altri calcolatori. Se non vi è più spazio su disco compare il messaggio DISK FULL.

AmigaBASIC consente di aprire contemporaneamente un massimo di 256 file. Poichè è raro aver bisogno di tanti file, è probabile che non si incorrerà mai in questo errore.

Tried to declare SUB within a SUB *Numero errore: -*

Non è possibile dichiarare un sottoprogramma all'interno di un altro sottoprogramma. E' però possibile dichiararne uno di seguito all'altro.

TYPE MISMATCH *Numero errore: 13*

L'utente si imbatte in questo errore se tenta di assegnare qualcosa in formato numerico a qualcos'altro in altro formato senza aver prima eseguito una conversione di formato (vedere la Nota d'uso 5 [I sistemi di numerazione di Amiga] per una descrizione dettagliata). Le variabili devono essere dello stesso tipo anche per il comando SWAP.

UNDEFINED ARRAY *Numero errore: 38*

Si è introdotto, dopo il comando SHARED, il nome di una matrice che non esiste nel programma. Probabilmente non si intendeva utilizzare il nome di una matrice; se così fosse, è necessario eliminare le parentesi dopo il nome della variabile.

UNDEFINED LABEL *Numero errore: 8*

L'utente ha utilizzato il nome di un'etichetta che non esiste nel programma (solitamente dopo un GOTO o un GOSUB). Controllare che non vi siano stati errori di battitura. E' facile dimenticare i due punti (:) dopo le etichette; i due punti sono comunque necessari quando una etichetta identifica una linea o una sezione del programma.

UNDEFINED SUBPROGRAM *Numero errore: 35*

Si ottiene questo errore quando si chiama un sottoprogramma inesistente. L'utente può non accorgersi di aver chiamato un sottoprogramma perchè AmigaBASIC può non riconoscere un comando che è stato battuto in modo erroneo, andando pertanto a cercare un sottoprogramma con lo stesso nome e visualizzando, nel caso in cui non lo trovi, questo messaggio.

UNDEFINED USER FUNCTION *Numero errore: 18*

Questo messaggio compare se l'utente cerca di usare una funzione (FN...) che non è stata definita con il comando DEF FN. Può darsi che il nome della funzione sia stato immesso in modo errato. Forse l'utente ha fornito ad una variabile un nome che inizia con le lettere FN (cosa non consentita). Si tenga presente che CLEAR annulla la definizione delle funzioni.

UNKNOWN VOLUME *Numero errore: 49*

L'utente ha immesso il nome di un dischetto che AmigaBASIC non riesce a trovare. Potrebbe trattarsi di un errore di battitura. Se si introduce il nome di un dischetto che AmigaBASIC non riesce a trovare, compare la tipica finestra di dialogo di AmigaDOS che ne richiede l'introduzione in un drive qualsiasi. Selezionando Cancel si indica ad AmigaBASIC che il nome del dischetto era errato.

UNPRINTABLE ERROR *Numeri errore: 21, 24, 25, 27, 28, 31, 32, 39-48, 56, 59, 60, 65, 69, 71, 72, 75, 76, 78-255*

Si è incappati in un errore, ma AmigaBASIC non dispone di un messaggio per identificarlo. Ciò solitamente avviene quando l'utente definisce delle routine personali per la gestione degli errori usando i comandi ON ERROR ed ERROR. Se si definiscono numeri d'errore personalizzati, questi devono essere richiesti anche da ON ERROR.

WEND WITHOUT WHILE *Numero errore: 30*

Ad ogni istruzione WEND deve essere associato un corrispondente WHILE. Se si incappa in questo errore significa che AmigaBASIC ha trovato almeno un WEND di troppo. Se l'utente scrive un programma in modo strutturato, è meno probabile che dimentichi qualcosa. Poichè WHILE e WEND vanno sempre in coppia, è sufficiente cercare il punto in cui ne è stato dimenticato uno.

WHILE WITHOUT WEND *Numero errore: 29*

Questo è il contrario di quanto appena visto sopra (WEND WITHOUT WHILE): AmigaBASIC ha trovato almeno un WHILE di troppo. Vedere quanto riportato a proposito del messaggio WEND WITHOUT WHILE.

B Sezione di riferimento AmigaBASIC

Questa appendice contiene la descrizione dei comandi disponibili in AmigaBASIC. Essi sono stati raggruppati a seconda della funzione svolta e sono quindi elencati in ordine alfabetico all'interno del gruppo di appartenenza. E' possibile ricavare nozioni su comandi poco noti, come imparare qualcosa di più su quelli che già si conoscono.

E' necessario fare molti esercizi impiegando i comandi che non risultano familiari, in quanto essi hanno spesso molte opzioni ed argomenti. Le opzioni sono spiegate in modo completo, ma, per carenza di spazio, non in modo dettagliato come nel resto del volume.

Per ogni comando (o istruzione) vengono fornite le seguenti informazioni:

Comando : [nome del comando] **Sezione** : [la sezione del libro dove viene incontrato la prima volta il comando]

Sintassi : `dove vengono enumerate tutte le opzioni del comando [i parametri tra parentesi quadre sono opzionali].`

Questa è infine la zona dove è contenuta la descrizione di ciascun comando e qualche esempio di programmazione che utilizza il comando in questione.

B.1 Input ed output su schermo

BEEP *Sezione 1.6*

```
BEEP
```

Questo comando dà origine ad un segnale sonoro e ad un breve lampeggio sullo schermo. Lo si può utilizzare per attirare l'attenzione dell'utente. Il suono non può essere udito a meno che almeno un canale audio (generalmente lo 0) sia connesso con un altoparlante.

Si può utilizzare PRINT CHR$(7) al posto di BEEP, in quanto entrambi producono lo stesso effetto.

CLS *Sezione 1.3*

```
CLS
```

CLS cancella quanto visualizzato nella finestra di output corrente. Il cursore di questa finestra verrà posizionato nel suo angolo superiore sinistro. Questo significa che una seguente uscita di dati sullo schermo, ottenuta tramite PRINT o comandi simili, verrà visualizzata a partire da quella posizione.

COLOR *Sezione 1.16*

```
COLOR [colore primo piano] [,colore sfondo]
```

COLOR serve per stabilire il colore di primo piano (con cui verranno scritti i testi o tracciati disegni) e quello di sfondo. Entrambi i colori sono identificati da un numero, il cui valore dipende dal numero di bit per pixel che vengono usati dallo schermo corrente.

CSRLIN

```
CRSLIN
```

CRSLIN è una variabile di sistema. Come DATE$ e FRE(0), è una variabile cui AmigaBASIC assegna un valore che l'utente non può modificare.

La variabile CRSLIN, contenente il numero di linea su cui è posizionato il cursore, viene utilizzata per eseguire calcoli per il comando LOCATE. Confrontare questa istruzione con il comando POS.

INKEY$ *Sezione 1.13*

```
INKEY$
```

Questo comando legge un carattere dal buffer assegnato alla tastiera trasferendolo come stringa lunga un carattere. Se non è stato premuto alcun tasto, INKEY$ produce una stringa nulla (vuota).

Il comando INKEY$ non mostra il carattere sullo schermo, ma si limita a metterlo a disposizione del programma.

INPUT *Sezione 1.6*

```
INPUT ["(Testo esplicativo)"] [, o ;]
Variabile [,Variabile, Variabile,...]
```

INPUT consente di immettere dati da tastiera mentre il programma è in esecuzione. Tali dati sono quindi assegnati alle variabili specificate.

Se l'utente lo desidera, INPUT può visualizzare su schermo un testo che spiega ciò che deve essere immesso da tastiera. Se al testo esplicativo si fa seguire un punto e virgola (;), AmigaBASIC visualizza un punto interrogativo dopo la spiegazione; se si fa seguire il testo da una virgola (,) il punto interrogativo non sarà mostrato.

Si possono avere diverse variabili cui assegnare i valori immessi da tastiera; i singoli valori devono essere separati tramite una virgola. Se il numero

di valori immesso è diverso da quello atteso, comparirà il messaggio d'errore "?Redo from start" (ricomincio da capo ?). Ciò significa che l'immissione deve essere ripetuta.

Inserendo un punto e virgola prima del testo esplicativo, il cursore si fermerà dopo l'ultimo carattere introdotto tramite INPUT ed un successivo output del programma verrà visualizzato a partire da quel punto.

LINE INPUT *Sezione 1.7*

```
LINE INPUT ["(Testo esplicativo)"] [, o ;]
   Variabile stringa
```

Il comando LINE INPUT legge una variabile stringa (cioè una sequenza di caratteri) immessa da tastiera. La stringa può includere qualunque tipo di carattere incluse virgole, punti e virgola ecc.; l'immissione va terminata con <RETURN>.

Anche qui è possibile visualizzare un testo esplicativo. A differenza di INPUT, LINE INPUT può accettare una sola variabile per volta, che deve inoltre essere del tipo stringa; qualunque altro tipo darà origine ad un errore di "Type mismatch".

LINE INPUT non visualizza un punto interrogativo dopo il testo esplicativo e non vi è quindi alcuna differenza nell'inserimento di un punto e virgola o di una virgola tra il testo e la variabile.

Inserendo un punto e virgola tra LINE INPUT e testo esplicativo, un successivo output del programma verrà visualizzato a partire dalla locazione di schermo subito a destra del termine dell'immissione.

LOCATE *Sezione 1.7*

```
LOCATE [Riga] [,Colonna]
```

Questo comando sposta il cursore della finestra di output del programma in una determinata posizione sullo schermo. E' necessario fornire valori positivi per Riga e Colonna, in quanto la posizione (1,1) corrisponde al-

l'angolo superiore sinistro dello schermo. Se lo schermo visualizza 60 caratteri per riga, il valore massimo di Riga è 21, e 62 è il valore massimo di Colonna. Se invece si lavora nel modo a 80 caratteri per linea, il valore massimo per Riga è 23, e quello di Colonna è pari a 77.

POS

```
POS(x)
```

L'utente può immettere un valore qualsiasi per x, in quanto essa è una variabile fittizia. (molte funzioni del BASIC richiedono che si specifichi un argomento, cioè un valore tra parentesi). POS(x) produce come risultato il numero della colonna in cui il cursore è attualmente posizionato (ovviamente il cursore della finestra di output), vale a dire la posizione sullo schermo da cui inizierà la visualizzione della prossima comunicazione del programma. Per conoscere la posizione corrente del cursore, si può utilizzare la seguente linea:

```
Riga=CRSLIN : Colonna=POS(0)
```

PRINT *Sezione 1.3*

```
PRINT [Variabile o Valore] [Separatore] [
Variabile o     Valore] . . .
```

Si può indicare PRINT con l'abbreviazione ? (punto interrogativo). Questo comando è utilizzato per visualizzare su schermo variabili o valori, incluse variabili stringa e testo. Il testo deve essere incluso tra doppi apici (ad esempio "Ciao").

Se non si indica alcuna variabile o valore, verrà inserita una linea vuota.

Diversi caratteri possono essere utilizzati come separatori: il punto e virgola (;) fa sì che i dati vengano visualizzati uno dietro l'altro; con la virgola (,) i dati verranno invece distanziati l'un l'altro dello spazio corrispondente ad una tabulazione (di solito pari a 15 spazi, valore che può comunque essere variato con il comando WIDTH).

L'utente può anche utilizzare i simboli #, &, % per separare i nomi delle

variabili: essi hanno lo stesso significato del punto e virgola.

Inserendo un punto e virgola alla fine di una linea, la successiva comunicazione del programma verrà visualizzata subito dopo l'ultimo valore mostrato. In caso contrario, verrà aggiunto un "a capo" (linefeed).

PRINT USING

```
PRINT USING [stringa di formattazione] ;
[Variabile o Valore] [;]
```

PRINT USING viene utilizzato per formattare su schermo l'output del programma. Tale comando si rivela utile per stampare tabelle numeriche allineate correttamente, ma può essere utilizzato anche per variabili stringa (sequenze di caratteri). La stringa di formattazione serve per indicare il formato prescelto. Esistono una serie di opzioni possibili: il carattere cancelletto (#) indica ad esempio una cifra. I numeri con cifre decimali che eccedono i decimali visualizzabili vengono arrotondati.

```
PRINT USING "###.##";32.4 dà origine a     32.40
PRINT USING "###.##";17                    17.00
PRINT USING "###.##";324.124              324.12
PRINT USING "###.##";128.489              128.49
PRINT USING "###.##";129.9984             130.00
```

Se un numero è composto da più cifre prima del punto decimale di quelle specificate dalla stringa di formattazione, verrà visualizzato preceduto dall'identificatore %:

```
PRINT USING "####.###";43259.3253        %43259.325
```

Un segno più (+) all'inizio della stringa fa sì che il segno (positivo o negativo) venga stampato prima del numero stesso. Il + può essere posto anche alla fine della stringa di formattazione ed in tal caso il segno del numero viene visualizzato dopo il numero stesso. Se si pone un meno (-) alla fine della stringa di formattazione, il segno negativo verrà stampato solo dopo i numeri negativi.

```
PRINT USING "+###.##";324.234          +324.23
PRINT USING "+###.##";-518.284         -518.28
PRINT USING "###.##+";-518.294          518.29-
PRINT USING "###.##-";324.234           324.23
PRINT USING "###.##-";-518.294          518.29-
```

Inserendo due asterischi (**) prima dei caratteri (#), eventuali spazi vuoti verranno riempiti con asterischi. I due asterischi valgono quanto la posizione di una cifra.

```
PRINT USING "**###.##";42.2             **42.20
PRINT USING "*******.##";43.1         *****43.10
PRINT USING "******.##";-523.456        *-523.46
```

Due simboli dollaro ($$) fanno in modo che venga visualizzato il carattere $ davanti al numero. Il primo simbolo $ rappresenta un segnale per il carattere che verrà stampato in seguito; il secondo $ indica la posizione di una cifra. Se si desidera visualizzare in ogni caso il segno +, esso deve essere posto al termine della stringa di formattazione.

```
PRINT USING "$$###.##";43.54            $43.54
PRINT USING "$$###.##";-43.54           -$43.54
PRINT USING "$$###.##";-53245.345     %-$53245.35
PRINT USING "$$###.##+";45.3            $45.30+
PRINT USING "$$+##.##";45.3              %$45+
```

L'ultima linea di esempio causa l'errore in cui si incorre quando si tenta di utilizzare + e $$ contemporaneamente. Il comando PRINT USING non visualizza in tal caso le cifre dopo il punto decimale, in quanto la sequenza $$+ confonde il calcolatore. E' però possibile combinare gli effetti delle combinazioni ** e $$. **$ rappresenta una posizione di tre caratteri: una per il segno $, e due per cifre od asterischi. Funziona in questo modo:

```
PRINT USING "**$##.##";45.3             **$45.30
PRINT USING "**$##.##";345.3            *$345.30
```

Una virgola a sinistra del punto decimale fa si che i grossi numeri vengano formattati in modo più chiaro:

```
PRINT USING "##########,.##";3444233.5  3,444,233.50
```

La virgola vale quanto la posizione di un carattere. La nota d'uso 5 ha introdotto anche la notazione esponenziale dei numeri. PRINT USING può lavorare anche in forma esponenziale: è sufficiente inserire quattro segni di esponente (^^^^) al termine della stringa di formattazione.

```
PRINT USING "##.##^^^^";34.533.45E+01
PRINT USING "###.##^^^^";.0099.00E-03
```

Il carattere sottolineato (_) fa sì che il carattere successivo venga stampato, anzichè essere considerato come carattere di formattazione.

```
PRINT USING "_####.##";34.95# 34.95
```

E' possibile anche inserire del testo nelle stringhe di formattazione. Tale testo può essere lungo quanto si desidera; non è comunque possibile definire una stringa di formattazione per numeri con più di 24 cifre. Se si tenta di utilizzare un numero con più di 24 cifre, si avrà il messaggio d'errore "Illegal Function Call".

```
PRINT USING "Questo è il numero: ##.##";3.3 Questo è
il numero: 3.30
```

Se PRINT USING non comprende il testo allegato alla stringa di formattazione, è sufficiente inserire un _ davanti al testo stesso.

Con PRINT USING è possibile formattare anche sequenze di caratteri. Un punto esclamativo (!) indica al calcolatore di visualizzare solo il primo carattere di una stringa:

```
PRINT USING "!";"Amiga"  A
```

Se si desidera impostare una grandezza massima di più di un carattere, è importante inserire degli spazi tra due sbarre rovesciate (\) (carattere backslash, il cui corrispondente tasto è posizionato alla sinistra del tasto <BACKSPACE>). Il numero di spazi tra i due caratteri aumentato di due

fornisce la lunghezza totale della stringa.

```
PRINT USING "\ \";"Mario" Mari
```

Nell'ultimo esempio vi sono due spazi tra le due sbarre rovesciate. Se la stringa è più lunga del numero di spazi a disposizione, essa viene troncata, mentre se è più corta le vengono aggiunti degli spazi. Infine, il simbolo di formattazione & consente di visualizzare su schermo stringhe di lunghezza qualunque.

```
PRINT USING "\\";"Amiga" Am
PRINT USING "&";"Amiga" Amiga
```

Naturalmente si può ottenere lo stesso risultato senza utilizzare PRINT USING.

Il comando PRINT USING non è molto semplice da usare. Se lo si utilizza sfruttando le sue opzioni più importanti è possibile creare molto più facilmente liste e tabelle formattate in modo corretto. Questo comando lavora meglio di LOCATE e delle funzioni per la manipolazione di stringhe.

E' consigliabile fare esperimenti con PRINT USING, in quanto ne vale proprio la pena.

SPC

```
SPC(x)
```

Questo comando stampa a video un numero x di spazi, con x compreso tra 0 e 255. SPC è spesso utile nella creazione di maschere video.

TAB *Sezione 3.4*

```
TAB(x);
```

Si utilizza questo comando per formattare i dati inviati alla stampante. AmigaBASIC riempie la distanza tra la posizione corrente di stampa e la colonna

indicata con x con spazi bianchi. La stampa su carta avviene a partire dalla colonna x. Se la posizione corrente corrisponde a una colonna più avanzata rispetto al valore indicato da x, la linea di stampa seguente inizierà alla colonna x.

L'utente può utilizzare TAB solo in combinazione con i comandi PRINT, PRINT#, e LPRINT.

WIDTH *Sezione 1.16*

```
WIDTH [lunghezza linea] [,intervallo tabulazione]
WIDTH [LPRINT] [lunghezza linea] [,intervallo
tabulazione]
WIDTH #NumeroFile [,lunghezza linea] [,intervallo
tabulazione]
WIDTH NomeDispositivo [,lunghezza linea] [,intervallo
tabulazione]
```

Il comando WIDTH presenta quattro possibili sintassi, alcune delle quali utilizzate per la manipolazione dei dati.

La funzione base di WIDTH è quella di stabilire la lunghezza della linea che accoglierà i dati di output dal programma. Dato che Amiga, mancando di una funzione di ritorno a capo automatico, può tranquillamente scrivere oltre il bordo destro dello schermo, è necessario stabilire la larghezza massima della riga; nel caso di Preferences regolate a 60 caratteri per riga, la lunghezza massima della linea visualizzabile sarà pari a WIDTH 61, mentre se lo schermo visualizza 80 caratteri per riga, sarà pari a WIDTH 76.

Il secondo valore dopo WIDTH indica l'intervallo di tabulazione corrispondente all'utilizzo di una virgola (,) nell'istruzione PRINT. E' anche possibile determinare la lunghezza massima della linea e l'intervallo di tabulazione per la stampante, utilizzando l'opzione LPRINT, oppure tramite la terza sintassi. In questo caso, l'utente deve aprire un file che identifica la stampante ed impostare il numero di tale file. Il valore specificato con WIDTH viene utilizzato solo dal file in questione; un'altra uscita su stampante che utilizza LPRINT non viene influenzata.

La quarta sintassi permette di inviare i valori direttamente ad un dispositivo (SCR, LPT1 o COM1). La lunghezza della linea e l'intervallo di tabulazione specificati verranno utilizzati solo dopo l'apertura del successivo file. Se un file è già aperto, i suoi valori non verranno modificati. Il valore base della lunghezza della linea per una stampante è pari a 80 caratteri.

WRITE ***Sezione 3.4***

```
WRITE [Variabile o Valore] [Separatore] [Variabile o
Valore] ...
```

Il comando WRITE stampa i caratteri a partire dalla posizione corrente del cursore come il comando PRINT. I separatori possono essere solo virgole (,) o punti e virgola (;). La differenza tra i due comandi consiste nel fatto che WRITE inserisce le virgole tra due valori adiacenti e pone le virgolette alle stringhe. Ciò rende WRITE assai utile per inviare dati a file.

B.2 Animazione di oggetti

COLLISION

```
Valore=COLLISION(NumeroOggetto)
```

COLLISION è una funzione che può essere utilizzata in modi diversi per trattare le collisioni tra oggetti. Ogni collisione viene memorizzata in un'area di "parcheggio", affinchè il programma possa verificarla e operare opportunamente. AmigaBASIC non può trattare contemporaneamente più di 16 collisioni.

Il numero che identifica il primo oggetto coinvolto in una collisione viene posto fra le parentesi. La funzione ritorna come risultato il numero del secondo oggetto coinvolto in una collisione. E' anche possibile che l'oggetto indicato si scontri contro i bordi della finestra; in tal caso la funzione darà come risultato un valore compreso tra -1 e -4:

-1 = bordo superiore
-2 = bordo sinistro
-3 = bordo inferiore
-4 = bordo destro

L'utente può immettere anche 0 o -1 al posto di NumeroOggetto.

La funzione COLLISION(0) produce come risultato il numero dell'oggetto che è stato coinvolto nella collisione più recente, senza però rimuoverlo dall'area di parcheggio. Ciò è utile per scoprire quali collisioni sono avvenute senza doverle trattare subito.

COLLISION(-1) indica in quale finestra è avvenuta la collisione. Questo può risultare utile se si hanno oggetti che si muovono all'interno di finestre diverse.

COLLISION ON
COLLISION OFF
COLLISION STOP

```
COLLISION ON
COLLISION OFF
COLLISION STOP
```

Questi comandi vengono utilizzati per l'intercettazione degli eventi (event trapping) collegati alle collisioni. COLLISION ON fa sì che il calcolatore inizi a rilevare la presenza di eventuali collisioni. COLLISION OFF interrompe la rilevazione, mentre COLLISION STOP sospende la gestione degli eventi collegati alle collisioni, pur continuando il loro intercettamento, fino al successivo COLLISION ON. L'utente può utilizzare ON COLLISION GOSUB per richiamare l'apposita subroutine quando si verifica una collisione. E' poi possibile utilizzare la funzione COLLISION per ottenere maggiori informazioni sugli oggetti coinvolti nella collisione.

OBJECT.AX
OBJECT.AY *Sezione 1.15*

```
OBJECT.AX NumeroOggetto,Accelerazione
NumeroOggetto,Accelerazione
```

Con questi due comandi l'utente stabilisce l'accelerazione di cui deve essere dotato un oggetto; OBJECT.AX definisce l'accelerazione orizzontale: valori positivi muovono l'oggetto da sinistra a destra, valori negativi da destra a sinistra. OBJECT.AY imposta l'accelerazione verticale: valori positivi muovono l'oggetto verso il basso, valori negativi verso l'alto. Fare riferimento alla Figura 4 nella Sezione 1.15 per ulteriori dettagli. L'accelerazione viene espressa in pixel al secondo; il movimento non inizia fin quando non si impartisce il comando OBJECT.START.

OBJECT.CLIP

```
OBJECT.CLIP (x1,y1)-(x2,y2)
```

Tramite OBJECT.CLIP l'utente indica un rettangolo di visualizzazione entro i cui confini AmigaBASIC muoverà gli oggetti. In altre parole, l'oggetto

sarà visibile solo quando le sue coordinate saranno comprese all'interno dell'area specificata.

AmigaBASIC considera come valore base l'intera area della finestra. Se questa viene allargata o ridotta, il rettangolo di visualizzazione dovrà essere modificato in modo appropriato (l'aggiornamento non è automatico !).

OBJECT.CLOSE

```
OBJECT.CLOSE [NumeroOggetto] [,NumeroOggetto...]
```

Ogni oggetto che rimane in memoria occupa spazio. Se non si deve più utilizzare un oggetto, è possibile recuperare lo spazio che esso occupa tramite OBJECT.CLOSE

Se non si introduce un numero di identificazione di un oggetto, tutti gli oggetti che appartengono alla finestra corrente di output verranno cancellati recuperando lo spazio da essi occupato.

OBJECT.HIT

```
OBJECT.HIT NumeroOggetto, [Valore1] [,Valore2]
```

Questo comando viene utilizzato per indicare quali oggetti (o bordi dello schermo) possono entrare in collisione con altri oggetti. L'utente può evitare che determinati oggetti vadano incontro a collisioni (ad esempio, potrebbe non essere importante il fatto che gli oggetti vadano oltre i confini dello schermo, oppure non si desidera che nel corso dell'esecuzione del programma una nave spaziale vada a sbattere contro una stella).

Se avviene una collisione, essa può essere rilevata e trattata con una routine di gestione delle collisioni. Maggiori dettagli a questo riguardo sono contenuti nelle voci ON COLLISION GOSUB e COLLISION.

I valori base consentono a tutti gli oggetti presenti di avere collisioni fra loro e con i bordi dello schermo. E' possibile fornire un numero decimale a 16-bit per Valore1 e Valore2. Se il bit più basso di Valore2 è posto a 1 (cioè se Valore2 è dispari), l'oggetto può entrare in collisione con i bordi

dello schermo. Se è posto a 0, esso può invece uscire dall'area visibile.

I rimanenti bit sono usati nel modo seguente: Valore1 indica l'oggetto in questione; Valore2 indica gli altri oggetti con cui il primo può entrare in contatto. Se si manifesta una collisione, AmigaBASIC esegue l'operazione booleana AND tra Valore2 dell'oggetto estraneo e Valore1 dell'oggetto in questione. Se nessuno dei bit impostati si accoppia (se cioè l'operazione di AND dà 0 come risultato), la collisione non viene rilevata. Se invece qualche bit corrisponde, la collisione viene rilevata. Ad esempio:

Oggetto	Valore1	Valore2
Nave spaziale	00000000 00000010	00000000 00000011
Razzo	00000000 00000010	00000000 00000010
Stella	00000000 00000100	00000000 00000100

I numeri sono stati scritti in forma binaria per agevolare i confronti. In realtà il comando OBJECT.HIT richiede che i valori vengano introdotti sotto forma di numeri decimali: ad esempio, la sintassi per la nave spaziale è

```
OBJECT.HIT 1,2,3
```

L'analisi dei bit mostra che il bit basso di Valore2 per la nave spaziale è posto a 1, per cui l'oggetto non può oltrepassare i confini dello schermo; invece sia il razzo che la stella possono lasciare lo schermo in una qualunque direzione.

Valore1 della nave spaziale e Valore2 del razzo sono specificati in modo tale che i due oggetti possano entrare in collisione; in altre parole, se il razzo tocca la nave spaziale, il programma rileva una collisione.

Il secondo bit di Valore2 della nave spaziale e quello di Valore1 del razzo sono definiti in modo tale che venga rilevata una collisione anche se accade l'opposto, se cioè la nave spaziale tocca il razzo. Dato che la stella ha solo il terzo bit posto a 1 sia per Valore1 che per Valore2, essa non può essere oggetto di collisioni rilevabili (anche se la nave spaziale o il

razzo la toccano il programma non rileva niente).

OBJECT.ON
OBJECT.OFF — Sezione 1.15

```
OBJECT.ON [NumeroOggetto] [,NumeroOggetto...]
OBJECT.OFF [NumeroOggetto] [,NumeroOggetto...]
```

OBJECT.ON rende visibile l'oggetto identificato tramite NumeroOggetto, mentre OBJECT.OFF lo rende non visibile. Se non si indica alcun numero, il comando si applica a tutti gli oggetti presenti nella finestra corrente di output.

OBJECT.OFF interrompe inoltre i movimenti degli oggetti, di modo che non si rischia di entrare in collisione con oggetti non più visibili.

OBJECT.PLANES

```
OBJECT.PLANES NumeroOggetto [,BitPlane] [,ValorePiano]
```

E' probabile che non si debba usare molto spesso questo comando, in quanto esso è coinvolto con le routine interne di Amiga. BitPlane è un valore ad 8 bit rappresentato con un numero decimale: l'insieme dei suoi bit determina in quale bitplane apparirà l'oggetto identificato con NumeroOggetto. Per esempio, se si hanno a disposizione 4 bitplane e si desidera che un oggetto sia mostrato sul primo e terzo bitplane (livello 0 e livello 2), è necessario associare a BitPlane il valore 2^0 + 2^2 = 1 + 4 = 5.

ValorePiano, anch'esso un numero ad 8 bit, indica cosa dovrebbe essere inserito nei bitplane rimanenti.

OBJECT.PRIORITY

```
OBJECT.PRIORITY NumeroOggetto,NumeroPriorità
```

OBJECT.PRIORITY indica la priorità dei singoli oggetti: lo si può impiegare per determinare quale oggetto, tra quelli che occupano una medesima

locazione sullo schermo, sarà visualizzato.

NumeroPriorità è un numero il cui valore è compreso tra -32768 e +32767. Quando due oggetti occupano una medesima locazione, sarà mostrato in primo piano quello con priorità più elevata. Se entrambi possiedono le stesso valore di priorità, l'uno o l'altro potrebbe essere in primo piano.

OBJECT.SHAPE **Sezione 1.13**

```
OBJECT.SHAPE NumeroOggetto,StringaDefinizione
OBJECT.SHAPE NumeroOggetto,NumeroOggetto ulteriore
```

Questo comando permette di definire forma, dimensione e colore di un oggetto. Tutte queste informazioni sono contenute in una stringa di definizione che può essere letta da un file sequenziale.

```
OPEN "(File Oggetto)" FOR INPUT AS 1
OBJECT.SHAPE NumeroOggetto, INPUT$(LOF(1),1)
CLOSE 1
```

L'utente può creare i dati dell'oggetto contenuti nel file sequenziale utilizzando il programma ObjectEditor contenuto nel cassetto BASICDemos del dischetto Extras. Ulteriori informazioni a questo riguardo sono riportate nella Sezione 1.11 (l'object editor).

La seconda sintassi consente di copiare la definizione di un oggetto già presente in memoria per creare un altro oggetto. E' così possibile avere in memoria molti oggetti identici.

OBJECT.START
OBJECT.STOP **Sezione 1.15**

```
OBJECT.START [NumeroOggetto] [,NumeroOggetto,...]
OBJECT.STOP [NumeroOggetto] [,NumeroOggetto,...]
```

OBJECT.START fa sì che un oggetto inizi a muoversi, mentre OBJECT.STOP ne interrompe il movimento. E' possibile far iniziare a muovere o bloccare diversi oggetti contemporaneamente.

OBJECT.VX e OBJECT.VY stabiliscono la velocità e/o accelerazione dell'oggetto. OBJECT.X e OBJECT.Y possono essere usati per indicare la posizione iniziale dell'oggetto sullo schermo, mosso in seguito tramite il comando OBJECT.START.

Da qui in avanti, l'oggetto si muoverà da solo e il programma potrà passare ad occuparsi d'altro.

AmigaBASIC attua automaticamente un OBJECT.STOP per gli oggetti coinvolti in una collisione.

OBJECT.VX
OBJECT.VY **Sezione 1.15**

```
OBJECT.VX NumeroOggetto,Velocità
OBJECT.VY NumeroOggetto,Velocità
```

Questi due comandi servono per impostare la velocità degli oggetti movibili. OBJECT.VX determina la velocità orizzontale, OBJECT.VY quella verticale.

Valori positivi muovono gli oggetti verso sinistra o verso il basso, valori negativi verso destra o l'alto.

Per maggiori informazioni consultare la Figura 4 a pag. 82. La velocità è indicata in pixel al secondo. Gli oggetti iniziano a muoversi (dal luogo indicato con OBJECT.X e OBJECT.Y) tramite il comando OBJECT.START.

OBJECT.VX
OBJECT.VY **Sezione 1.15**

```
Valore = OBJECT.VX(NumeroOggetto)
Valore = OBJECT.VY(NumeroOggetto)
```

Questa funzione fornisce dati sulla velocità orizzontale o verticale di un oggetto in un determinato istante. Queste informazioni possono essere utilizzate nel caso si intenda variare la velocità stessa dell'oggetto.

OBJECT.X
OBJECT.Y **Sezione 1.15**

```
OBJECT.X NumeroOggetto,Coordinata X
OBJECT.Y NumeroOggetto,Coordinata Y
```

OBJECT.X ed OBJECT.Y stabiliscono la posizione e/o le coordinate di partenza del movimento di un oggetto. Coordinata X e Coordinata Y sono riferite all'angolo superiore sinistro dell'oggetto. I valori che rendono visibile sullo schermo un determinato oggetto dipendono direttamente dal tipo di risoluzione video adottata.

Non è necessario che gli oggetti siano all'interno della regione visibile, ma le loro coordinate devono essere comunque comprese in un intervallo di valori che varia da -32768 a +32767. Si tenga presente che gli oggetti non saranno visibili fino a quando non viene impartito il comando OBJECT.ON.

OBJECT.X
OBJECT.Y **Sezione 1.15**

```
Valore = OBJECT.X(NumeroOggetto)
Valore = OBJECT.Y(NumeroOggetto)
```

In questo caso OBJECT.X ed OBJECT.Y assumono il significato di funzioni BASIC. L'utente, tramite queste istruzioni, può ricavare i dati sulla locazione attuale di un oggetto, decidendo quindi cosa fare con l'oggetto in questione. Il valore ottenuto si riferisce alle coordinate dell'angolo superiore sinistro dell'oggetto.

ON COLLISION GOSUB

```
ON COLLISION GOSUB Etichetta
```

Questo comando viene utilizzato per la gestione delle collisioni: l'utente indica quale subroutine richiamare quando viene rilevata una collisione. La subroutine appropriata viene identificata con il parametro Etichetta.

Se si immette 0 al posto di una etichetta o di un numero di linea, la gestione delle collisioni viene interrotta. Ciò accade anche se "0" è una

etichetta o un numero di linea presente nel programma. Per ripristinare la gestione delle collisioni è necessario utilizzare COLLISION ON.

B.3 Comandi grafici

AREA **Sezione 2.6**

```
AREA [STEP] (x,y)
```

E' possibile utilizzare AREA per definire un vertice di un poligono. Il comando AREAFILL disegna il poligono sullo schermo (vedi AREAFILL per maggiori informazioni).

Utilizzando l'opzione STEP del comando, x ed y verranno aggiunti ai valori delle ultime coordinate.

AREAFILL **Sezione 2.6**

```
AREAFILL [Modalità]
```

Con AREA si possono definire fino a venti vertici per poligono, limite massimo permesso da AmigaBASIC. AREAFILL consente di visualizzare il poligono specificato da AREA.

Il processo è molto rapido in quanto si avvale dell'uso del blitter.

Modalità può assumere due valori: se l'utente immette 0, il poligono sarà riempito con il modello di riempimento stabilito tramite il comando PATTERN; se invece viene immesso 1, il poligono verrà invertito, cioè, se i colori sono regolati normalmente, si ha che:

il blu	si trasforma in	arancione
il bianco		nero
il nero		bianco
l'arancione		blu

CIRCLE **Sezione 2.6**

```
CIRCLE [STEP] (x,y),Raggio [,Colore] [,AngoloIniziale]
[,AngoloFinale] [,Rapporto x/y]
```

Questo comando disegna un cerchio od un'ellisse con centro posizionato in (x,y). Raggio indica la lunghezza del raggio in pixel. Se STEP precede (x,y), AmigaBASIC somma i valori di x ed y alle ultime coordinate utilizzate.

Colore indica il numero del colore da utilizzare: il valore consentito per questo parametro dipende dal numero di bitplane utilizzati dallo schermo corrente. AngoloIniziale ed AngoloFinale rendono più semplice disegnare sezioni di cerchio, cioè archi; i valori dei punti di partenza e d'arrivo (multipli di pi-greco) vanno introdotti in radianti; angoli negativi fanno sì che i punti di partenza e d'arrivo vengano collegati al centro del cerchio. La Figura 6 a pag. 140 può aiutare nella determinazione dell'angolo.

Rapporto x/y determina la relazione intercorrente tra il raggio orizzontale e quello verticale: variando tale valore si possono disegnare ellissi di tutti i tipi. Valori minori di 0.5 danno origine a raggi verticali piccoli, valori superiori a 0.5 permettono di ottenere ellissi con raggio verticale maggiore del raggio orizzontale. Il valore 0.5 corrisponde ad un cerchio perfetto, anche se ciò dipende dal monitor e dalle sue regolazioni.

GET (screen GET) **Sezione 4.1**

```
GET (x1,y1)-(x2,y2),NomeMatrice [(indice,...)]
```

L'utente può impiegare questo comando per salvare una sezione dello schermo in una matrice di dati. La matrice deve essere numerica; l'utente può utilizzare una matrice numerica intera, a singola oppure a doppia precisione. Le dimensioni della matrice in byte sono calcolate con la seguente formula:

```
6 + Numero BitPlane * Altezza * 2 * INT((Larghezza +
16) / 16)
```

Il risultato viene poi diviso per il numero di byte per ciascun elemento della matrice secondo i seguenti valori:

2 byte per matrice numerica intera (Matrice%(x))
4 byte per matrice numerica a singola precisione (Matrice(x))
8 byte per matrice numerica a doppia precisione (Matrice#(x))

L'elemento 0 della matrice contiene la larghezza, quello 1 l'altezza e quello 2 il numero di bitplane del segmento di schermo salvato.

Se si ha una matrice multidimensionale, si possono immagazzinare diversi segmenti di schermo, oppure differenti viste della stessa immagine in modo da passare da una all'altra in modo rapido.

Per visualizzare di nuovo l'area salvata sullo schermo, è necessario usare il comando PUT.

LINE **Sezione 2.5**

```
LINE [[STEP] (x1,y1)]-[[STEP] (x2,y2) [,Colore] [,B
oppure ,BF]
```

Questo comando permette di tracciare una linea od un rettangolo (vuoto o pieno). L'uso più semplice consiste nell'indicare un punto di partenza ed uno d'arrivo, ottenendo così una linea retta:

```
LINE (x1,y1)-(x2,y2)
```

Se si tralascia il primo punto, AmigaBASIC traccia una linea dall'ultimo punto disegnato sullo schermo al punto (x2,y2).

```
LINE -(x2,y2)
```

Entrambe le modalità possono essere impiegate in combinazione con STEP: in tal caso i valori di x ed y verranno aggiunti a quelli dell'ultimo punto disegnato. L'utente può specificare il colore con cui sarà tracciata la linea immettendone l'appropriato numero di identificazione. Se alla fine del comando si immette ,B, LINE darà origine ad un rettangolo anzichè ad una linea: le coordinate (x1,y1) e (x2,y2) corrisponderanno allora all'angolo superiore sinistro ed inferiore destro. Se si utilizza ,BF, si otterrà un rettangolo riempito con il colore specificato.

PAINT **Sezione 2.6**

```
PAINT [STEP] (x,y) [,ColoreRiempimento] [,ColoreBordo]
```

Con PAINT l'utente può riempire una superficie racchiusa da linee con il colore indicato in ColoreRiempimento. Se non si specifica altrimenti, per ColoreBordo verrà utilizzato ColoreRiempimento; se non viene indicato nessun colore, AmigaBASIC impiegherà il colore corrente sia come ColoreRiempimento che come ColoreSfondo. Per utilizzare PAINT, è necessario riferirsi ad una finestra il cui valore di tipo è compreso tra 16 e 31 (il contenuto della finestra deve poter accedere ad un buffer di memoria).

PALETTE **Sezione 1.16**

```
PALETTE NumeroColore,Rosso,Verde,Blu
```

E' possibile utilizzare colori personalizzati nei propri programmi, ottenuti tramite il comando PALETTE. Il numero massimo di colori specificabili dipende dal numero di bitplane utilizzato per lo schermo. Ogni colore possiede un particolare NumeroColore di identificazione (compreso tra 0 e 31).

L'utente può immettere un valore tra 0 (= 0%) e 1 (= 100%) per ciascun componente Rosso, Verde, Blu. I nuovi parametri rimpiazzano quelli precedenti. I quattro colori dello schermo del Workbench (i colori con numero da 0 a 3 sono utilizzati dal Workbench e possono essere modificati con le Preferences) vengono visualizzati da Amiga all'inizio ed alla fine di un programma.

PATTERN **Sezione 2.10**

```
PATTERN [valore a 16 bit per le linee] [,matrice per
le  superfici]
```

E' possibile usare questo comando per stabilire un modello con cui tracciare linee o riempire aree; se lo si utilizza per le linee, è necessario immettere un valore a 16 bit che permette di definire una linea campione lunga 16 pixel. Ogni bit posto a 1 corrisponde ad un punto acceso sullo schermo; il numero a 16 bit può essere immesso in forma decimale (65535),

esadecimale (&HFFFF) oppure ottale (&O177777).

Per il riempimento di aree è necessario introdurre una matrice numerica intera. I numeri nella matrice definiscono un modello che può essere utilizzato per il riempimento e la colorazione di aree (con i comandi PAINT, AREAFILL o LINE...,BF). Anche in questo caso i bit posti a 1 corrispondono ai punti accesi sullo schermo.

Il numero di elementi in una matrice deve essere pari ad una potenza di due (cioè 1, 2, 4, 8, 16, o 32).

POINT

```
POINT(x,y)
```

Questa funzione BASIC fornisce il numero di colore del punto alle coordinate (x,y) della finestra corrente di output del programma. Se il punto non appartiene alla finestra, il valore di ritorno è -1.

PRESET
PSET **Sezione 2.5**

```
PRESET [STEP] (x,y) [NumeroColore]
PSET [STEP] (x,y) [,Colore]
```

Questi comandi tracciano un punto nella finestra corrente di output alle coordinate x,y. Si può utilizzare STEP per disegnare punti aggiuntivi le cui coordinate abbiano valori non assoluti, ma relativi a quelli dell'ultimo punto tracciato. Il colore del punto è specificato dal valore di NumeroColore. L'unica differenza tra PRESET e PSET consiste nel fatto che, quando il colore non viene specificato, PRESET disegna il punto utilizzando il colore di sfondo, mentre PSET lo traccia usando il colore di disegno corrente.

PTAB

```
PRINT PTAB(x)
```

Questa funzione lavora in modo simile a TAB, ad eccezione del fatto che

la posizione viene in questo caso indicata in pixel anzichè in caratteri. L'utente può dunque posizionare il testo all'interno della linea corrente in modo molto preciso. Ad esempio:

```
FOR x=0 TO 100
PRINT PTAB(x);"Ciao !"
NEXT x
```

PUT (screen PUT) **Sezione 4.5**

```
PUT [STEP] (x,y),NomeMatrice [(indice,...)] [,Modo di
Azione]
```

Si usa questo comando per visualizzare nuovamente sullo schermo le sezioni di disegno memorizzate in una matrice con l'istruzione GET. I parametri sono gli stessi di GET a cui si rimanda per eventuali ulteriori informazioni.

L'unica cosa nuova è il parametro Modo di Azione; l'utente ha cioè la possibilità di scegliere tra diverse opzioni per quanto riguarda la modalità di visualizzazione su schermo:

PSET: i dati in arrivo dalla matrice si sovrappongono, sostituendosi, a quelli preesistenti: la porzione di schermo in questione apparirà uguale a quella che era stata registrata in precedenza.

PRESET: la figura viene invertita.

AND: vengono visualizzati sullo schermo solo i punti che rimangono dopo una operazione logica di AND tra la sezione che deve essere letta e lo sfondo.

OR: il contenuto della matrice e lo sfondo subiscono una operazione logica di OR. La sezione verrà copiata per intero sullo sfondo.

XOR: è il valore di base per Modo di Azione; la sezione viene copiata sullo schermo, mentre la figura sottostante viene visualizzata invertita.

SCREEN **Sezione 2.3**

```
SCREEN NumSchermo,Larghezza,Altezza,Profondità,Modo
```

L'utente può usare questo comando per creare un nuovo schermo; AmigaBASIC consente di avere contemporaneamente 4 schermi (numerati da 1 a 4) oltre a quello del Workbench (NumeroSchermo 0). Larghezza ed Altezza sono misurati in pixel; Profondità indica il numero di bitplane usati dallo schermo.

Profondità (numero di bitplane)	**Colori disponibili**
1	2
2	4
3	8
4	16
5	32

Modo stabilisce la risoluzione e l'attivazione o meno dell'interlacciamento: è un numero compreso tra 1 e 4.

Mode	**Descrizione**	**Memoria richiesta per bitplane**
1	320*256 Pixels	10240 byte
2	640*256 Pixels	20480 byte
3	320*512 Pixels	20480 byte
4	640*512 Pixels	40960 byte

SCREEN CLOSE

```
SCREEN CLOSE NumeroSchermo
```

Dato che gli schermi utilizzano preziosa memoria (CHIP RAM), vanno chiusi quando non servono più. SCREEN CLOSE è il comando per effettuare tale operazione; esso cancella lo schermo indicato con NumeroSchermo restituendo al sistema la memoria utilizzata da questo schermo.

SCROLL Sezione 1.17

```
SCROLL (x1,y1)-(x2,y2),delta-x,delta-y
```

Con questo comando è possibile spostare una porzione di schermo in una direzione qualunque. (x1,y1) - (x2,y2) indica un'area rettangolare: il contenuto di questa area viene spostato. delta-x determina di quanti punti in direzione orizzontale deve essere spostato il rettangolo specificato: valori positivi danno luogo ad uno spostamento verso destra, valori negativi verso sinistra. delta-y ha lo stesso significato per quanto riguarda la direzione verticale: valori positivi muovono la porzione di schermo verso il basso, valori negativi verso l'alto.

WINDOW Sezione 2.4

```
WINDOW NumeroFinestra [,Titolo] [,(x1,y1)-(x2,y2)]
[,Tipo] [,NumeroSchermo]
```

Con questo comando si possono creare ed usare nuove finestre. Ogni finestra dispone di un NumeroFinestra che la identifica e che ne permette un facile accesso. La finestra BASIC ha come numero di identificazione 1, mentre le finestre create dall'utente hanno numeri che vanno dal 2 in su.

Quando si crea una finestra, le si può assegnare un titolo e specificarne le dimensioni, indicando un rettangolo con angolo sinistro superiore avente coordinate (x1,y1) e angolo inferiore destro con coordinate (x2,y2). Se non si stabilisce diversamente, AmigaBASIC assegna alla finestra le dimensioni dell'intero schermo cui appartiene.

Tipo è un valore ottenuto a partire da uno dei numeri seguenti o dalla

somma di più di uno degli stessi:

1 - L'utente può ridimensionare la finestra utilizzando il mouse e l'apposito gadget di ridimensionamento.

2 - La finestra può essere spostata con il mouse, utilizzando la riga titolo.

4 - La finestra è dotata del gadget di profondità e può essere posta in secondo piano od in primo piano.

8 - La finestra è dotata del gadget di chiusura.

16 - Il contenuto della finestra viene ridisegnato dopo che è avvenuto uno spostamento, un ridimensionamento o la sovrapposizione di un'altra finestra. Se si desidera utilizzare il comando PAINT nella finestra è necessario ricorrere a questa opzione.

L'utente può anche indicare a quale schermo appartiene la finestra; se si omettono tutte le opzioni e si utilizza il comando WINDOW nel modo seguente:

```
WINDOW NumeroFinestra
```

la finestra in questione verrà posta in primo piano e diventerà la finestra corrente in cui verranno mostrati i dati di output del programma.

WINDOW **Sezione 2.4**

```
WINDOW(x)
```

In questo caso WINDOW è una funzione BASIC: l'utente può ricavare informazioni sulla finestra corrente di output. x è un valore compreso tra 0 ed 8.

WINDOW(0) fornisce il numero di identificazione della finestra di output selezionata. Può anche non essere la finestra corrente di output: ad esempio, potrebbe essere la finestra selezionata per ultima tramite il mouse (il cui titolo è scritto in lettere piene).

WINDOW(1) fornisce il numero di identificazione della finestra di output corrente a cui AmigaBASIC invia i dati di output.

WINDOW(2) dà come risultato la larghezza in pixel della finestra di output corrente.

WINDOW(4) fornisce la coordinata orizzontale (in pixel) della locazione in cui sarà visualizzato il prossimo carattere nella finestra corrente di output. In pratica fornisce la locazione orizzontale corrente del cursore.

WINDOW(5) fornisce la coordinata verticale (in pixel) della locazione in cui verrà visualizzato il prossimo carattere.

WINDOW(6) indica il numero massimo di colori disponibili nella finestra di output corrente.

WINDOW(7) è un puntatore al record della finestra Intuition che corrisponde alla finestra di output corrente.

WINDOW(8) fornisce l'indirizzo dell'area dati Intuition riguardante la struttura RASTPORT. E' consigliabile utilizzare questo valore solo se si conosce molto bene il sistema operativo di Amiga.

WINDOW CLOSE **Sezione 2.4**

```
WINDOW CLOSE NumeroFinestra
```

Questo comando rimuove la finestra indicata dallo schermo senza cancellarla. E' possibile inviare ancora dati dal programma a quella finestra anche se essa non è più visibile.

WINDOW OUTPUT **Sezione 2.4**

```
WINDOW OUTPUT NumeroFinestra
```

La finestra indicata in questo comando diventa la finestra di output corrente, ma questo non significa che venga messa in primo piano. Questa è l'unica differenza con il comando WINDOW NumeroFinestra. In questo

modo, infatti, l'utente può inviare i dati di output del programma a una finestra non visibile sullo schermo.

B.4 Comandi per il controllo del programma

BREAK ON
BREAK OFF
BREAK STOP

```
BREAK ON
BREAK OFF
BREAK STOP
```

Questi comandi sono connessi all'intercettazione degli eventi di interruzione del programma. Normalmente si può interrompere lo svolgimento di un programma premendo la combinazione di tasti <CTRL> <C>, oppure selezionando l'opzione STOP dal menù RUN. Se si desidera che nessuno possa interrompere un programma, od una sua parte, è necessario fare ricorso a questi comandi. E' necessario indicare con il comando ON BREAK GOSUB l'apposita subroutine che il programma deve richiamare se riceve un segnale di interruzione. BREAK ON abilita l'intercettazione delle interruzioni, BREAK OFF la disattiva fino ad un successivo BREAK ON. BREAK STOP sospende l'intercettazione: il programma continua a registrare segnali di interruzione, ma non esegue l'apposita subroutine fino a quando non viene impartito un nuovo comando BREAK ON.

CALL *Sezione 4.5*

```
CALL Nome [(Lista-Argomenti,...)
CALL Variabile [(Valore,...)
```

La prima forma del comando è utilizzata per chiamare un sottoprogramma; ulteriori notizie sui sottoprogrammi si possono trovare sotto la voce SUB ... STATIC. Solitamente, in questo caso si tralascia CALL che può essere utilizzato per evitare l'insorgere di confusione qualora si abbia

un'etichetta con lo stesso nome di un sottoprogramma.

Tramite CALL è possibile richiamare anche subroutine in linguaggio macchina. Il linguaggio nativo di Amiga non è il BASIC, che è un linguaggio interpretato; se si desidera comunicare col calcolatore nel suo linguaggio nativo è necessario conoscere l'Assembler del 68000. Chi non fosse particolarmente interessato a questo argomento, può saltare i prossimi paragrafi.

La prima cosa da fare è caricare in memoria il programma in linguaggio macchina che può essere caricato da dischetto o letto tramite READ da una matrice di dati. E' poi necessario conoscere l'indirizzo di partenza della routine in linguaggio macchina: a questo provvedono le funzioni VARPTR o SADD. Un esempio:

```
DIM ProgrammaLM%(20)
FOR x=0 to 20
    READ ProgrammaLM%(x)
NEXT x
IndirizzoPartenza&=VARPTR(ProgrammaLM%(0))
CALL IndirizzoPartenza&(10,20)
DATA ....
```

L'intero programma in linguaggio macchina è contenuto nella matrice ProgrammaLM%. L'indirizzo di partenza di tale matrice lo si ricava con:

```
VARPTR(ProgrammaLM%(0))
```

E' anche possibile passare parametri alla routine in linguaggio macchina, sfruttando l'istruzione CALL (nel caso visto sopra, sono stati passati 10 e 20). E' pure possibile memorizzare un programma in linguaggio macchina in una stringa:

```
Ma$=""
FOR x=0 to 20
    READ Valore
    Ma$=Ma$+MKI$(Valore)
NEXT x
IndirizzoPartenza&=SADD(Ma$)
CALL IndirizzoPartenza&(10,20)
DATA ....
```

La funzione SADD fornisce come risultato l'indirizzo di partenza della stringa. Tramite MKI$ è possibile costruire una stringa a partire dai singoli valori contenuti nelle linee di DATA.

L'utente può anche utilizzare routine in linguaggio macchina contenute in una raccolta di routine (library). Vedere la voce LIBRARY per maggiori informazioni.

CHAIN

```
CHAIN [MERGE] NomeFile [,Espressione] [,ALL]
[,DELETE  Intervallo]
```

Con questo comando è possibile, all'interno di un programma BASIC, caricare un altro programma BASIC a cui passare il controllo dell'esecuzione. La differenza tra questo comando e MERGE sta nel fatto che in questo caso il programma chiamante viene parzialmente o totalmente cancellato dalla memoria. L'opzione MERGE è una delle tante possibili per il comando CHAIN; se la si utilizza, il programma chiamante verrà rimpiazzato ad iniziare dal numero di linea indicato da Espressione. Il programma da caricare deve essere stato registrato in formato ASCII su dischetto.

Se non si utilizza MERGE, Espressione indica il numero di linea del programma richiamato da cui dovrebbe iniziare l'esecuzione. Non è possibile usare un'etichetta al posto del numero di linea: in questo caso è necessario ricorrere ad un numero di linea. L'opzione ALL consente il passaggio di tutte le variabili del programma chiamante verso quello chiamato; se invece si desidera passarne solo alcune, queste vanno indicate con il comando COMMON. Il comando COMMON e l'opzione ALL sono mutualmente esclusivi.

L'opzione DELETE elimina un intervallo di linee dal programma chiamato; l'intervallo può essere specificato sia tramite numeri di linea che per mezzo di etichette.

Il comando CHAIN inizializza il puntatore agli elementi DATA: esso agisce come il comando RESTORE. Dopo l'esecuzione del comando CHAIN, tutti i file aperti rimangono aperti, come pure i parametri come OPTION BASE. CHAIN disabilita però l'intercettazione degli eventi, che, se neces-

sario, va quindi riattivata nel programma chiamato. Vengono perse anche le dichiarazioni dei tipi di variabile effettuate tramite DEFINT, DEFLNG, ecc.; se il programma chiamato non contiene un comando DEF FN, AmigaBASIC cancellerà la definizione della funzione in oggetto, presente nel programma chiamante.

CLEAR

```
CLEAR [,Area riservata al BASIC] [,Stack]
```

CLEAR può essere utilizzato per cancellare tutte le variabili, stringhe e matrici; questo comando chiude anche tutti i file aperti, oltre ad avere altre importanti funzioni.

Con CLEAR è possibile variare la quantità di memoria usata da AmigaBASIC: l'utente può allargare o ridurre l'allocazione di memoria per i programmi e dei dati BASIC. L'area di memoria minima per il BASIC è di 1024 bytes (1 Kb); il limite superiore è la quantità di memoria RAM presente nel sistema. Dato che Amiga necessita di molto spazio di memoria per i programmi grafici, è consigliabile lasciare a disposizione del sistema una quantità sufficiente di RAM. Se Amiga esaurisce la memoria disponibile, si avrà un crash di sistema (vedere la Sezione 1.6 per una discussione sulle Guru Meditation).

Se invece un programma lavora principalmente con dati numerici e non necessita di molta memoria per le funzioni grafiche, è possibile avere un'area di memoria riservata al BASIC molto grande. Il valore base dell'area riservata al BASIC è di 25000 byte in un Amiga con 512 k di memoria. Ad esempio:

```
CLEAR,40000
```

rende disponibile per il BASIC un'area di memoria pari a 40000 byte.

Stack rappresenta l'area di memoria che AmigaBASIC utilizza per la gestione interna; in quest'area viene, ad esempio, tenuta traccia del contatore del ciclo FOR ... NEXT, come pure della linea cui ritornare dopo il comando RETURN alla fine di una subroutine. Il valore base di quest'area è pari a 4789 byte; essa non può essere minore di 1024 bytes. Il comando FRE(x) fornisce informazioni sulle dimensioni attuali delle due aree di memoria.

COMMON

```
COMMON Lista-Variabili
```

Questo comando passa singole variabili ad un programma caricato tramite il comando CHAIN. Se si desidera passare matrici, è necessario indicarle con parentesi vuote:

```
CHAIN "NomeProgramma"
COMMON a,B$,Ciao%,Colori(),Testi$()
```

E' possibile utilizzare più d'una istruzione COMMON, ma ciascuna variabile deve comparire una sola volta.

CONT

```
CONT
```

CONT permette di far ripartire il programma dal punto in cui era stato interrotto mediante la combinazione di tasti <CTRL> <C>, tramite la selezione dell'opzione STOP del menù RUN, o da un comando STOP o END all'interno del programma. Il programma non deve aver subito modifiche dopo l'interruzione, in quanto, in caso contrario, verrà visualizzato il messaggio d'errore "Can't continue".

DATA *Sezione 2.9*

```
DATA Lista-Costanti
```

L'utente può immettere valori (numeri o stringhe) su una linea preceduta da DATA; tali valori saranno in seguito letti dal programma per mezzo dell'istruzione READ. I singoli valori devono essere separati tra loro per mezzo di virgole; se una stringa contiene virgole o punti e virgola, deve essere posta tra virgolette (" ").

Le linee DATA possono essere sistemate in qualunque punto del programma; i valori verranno letti uno di seguito all'altro.

DECLARE FUNCTION ... LIBRARY

```
DECLARE FUNCTION NomeFunzione [Lista-Param] LIBRARY
```

Se si desidera utilizzare un programma in linguaggio macchina che costituisce una funzione (dà cioè origine ad un valore), e questa funzione è contenuta in una libreria, è necessario dichiarare la funzione con il comando DECLARE FUNCTION ... LIBRARY (vedere anche CALL e LIBRARY). Se il valore prodotto è di un certo tipo, è sufficiente inserire l'identificatore di tipo al termine di NomeFunzione:

```
DECLARE FUNCTION Test%(Valore_Test) LIBRARY
```

La funzione in linguaggio macchina Test%(Valore_Test) produce come risultato un intero a 16 bit. Immettere le variabili che il programma in linguaggio macchina si aspetta non è necessario in quanto AmigaBASIC non viene coinvolto in questo passaggio. Se tuttavia l'utente le specifica, è necessario sapere quali valori devono essere in seguito passati alla funzione chiamata.

DEF FN *Sezione 7.6*

```
DEF FNNome [(Lista-Param)] = Definizione Funzione
```

Tramite l'uso di DEF FN è possibile creare proprie funzioni BASIC. Il nome della funzione va scritto subito dopo FN. Se si vuole utilizzare una funzione all'interno di un programma, è sufficiente immettere qualcosa come PRINT FNa (100) o Test=FNTesto (1,2,4). I valori opzionali posti tra parentesi sono utilizzati nella formula allo stesso modo di quelli fra parentesi nella definizione (DEF FNa(x) = 2*x). Il programma può anche usare variabili che non sono tra parentesi per calcolare il valore della funzione:

```
DEF FNTest(x) = 2*x*a
a=100 : PRINT FNTest (25)
a=12 : PRINT FNTest (25)
```

Questi due esempi danno come risultato rispettivamente 5000 e 600. E' anche possibile definire una funzione facendo uso delle funzioni di stringa:

```
DEF FNPrimaLettera$(a$) = LEFT$(a$,1)
PRINT FNPrimaLettera$("Amiga")
```

Come risultato si otterrà la visualizzazione di A.

DEFDBL
DEFINT
DEFLNG
DEFSNG
DEFSTR

```
DEFDBL Intervallo (da lettera a lettera)¶
DEFINT Intervallo (da lettera a lettera)¶
DEFLNG Intervallo (da lettera a lettera)¶
DEFSNG Intervallo (da lettera a lettera)¶
DEFSTR Intervallo (da lettera a lettera)¶
```

Si possono utilizzare questi comandi per associare alle variabili i cui nomi iniziano con una certa lettera un preciso tipo. Il primo carattere di ogni nome di variabile deve essere compreso in un certo intervallo; ad esempio, DEFINT a-c significa che, da questo punto in poi, tutte le variabili i cui nomi con le lettere a, b, c (Amiga, bimbo, ciao, ...) sono considerate del tipo intero a 16 bit, per cui non è più necessario utilizzare il segno % dopo il nome della variabile. Questo vale anche per le matrici. Comunque, se si specifica un indicatore di tipo, questo ottiene la priorità: Aiuto$ verrà quindi trattato come una stringa, mentre Aiuto verrà considerato un intero a 16 bit.

Sono possibili le seguenti definizioni:

Comando	Tipo variabile	Esempio
DEFINT	Intero a 16 Bit	Ciao%
DEFLNG	Intero a 32 Bit	Ciao&
DEFSNG	Decimale singola precisione	Ciao, ciao!
DEFDBL	Decimale doppia precisione	Ciao#
DEFSTR	Stringa	Ciao$

E' necessario specificare il tipo di variabile prima di poterlo utilizzare.

DELETE

```
DELETE [Etichetta o Numero di linea] [-] [Etichetta o
Numero di linea]
```

DELETE può essere utilizzato tanto all'interno di programmi quanto in modo diretto (immettendolo nella finestra BASIC) per eliminare un gruppo di linee di codice. E' necessario specificare l'intervallo da eliminare:

```
DELETE Inizio - DefinizioneColori
```

fa sì che venga cancellato quanto compreso tra le due etichette.

```
DELETE Inizio -
```

cancella tutte le linee comprese tra l'etichetta specificata e la fine del programma.

```
DELETE - DefinizioneColori
```

elimina tutte le linee di programma comprese tra il suo inizio e l'etichetta indicata.

DIM *Sezione 1.7*

```
DIM [SHARED] NomeMatrice (Valore [,Valore,...])
[,NomeMatrice (...),...]
```

DIM viene utilizzato per specificare le dimensioni delle matrici all'interno del programma. Se non si procede al dimensionamento, AmigaBASIC pone automaticamente la dimensione di una matrice pari a 10 elementi. Il dimensionamento impedisce che una matrice occupi più spazio in memoria di quanto sia necessario.

L'opzione SHARED permette che la matrice in questione possa essere utilizzata tanto dal programma principale quanto dai sottoprogrammi ad esso collegati. DIM SHARED può comunque essere utilizzato solo all'interno del programma principale. Se si desidera impiegare un certo numero di variabili in tutti i sottoprogrammi, è possibile utilizzare il comando DIM SHARED:

```
DIM SHARED Colori%(31,2), Colori
```

La variabile Colori è una variabile globale, in contrasto con le variabili locali che un sottoprogramma può utilizzare.

Tramite il comando OPTION BASE, è possibile stabilire se il primo elemento di ogni matrice deve essere indicato con zero od uno.

Si possono dimensionare matrici multidimensionali (ad esempio DIM Matrice (2,2,2) indica una matrice tridimensionale). Il numero massimo ammesso per il dimensionamento è di 255. Il numero più alto assegnabile ad un elemento contenuto in una matrice è pari a 32767: è praticamente quasi impossibile raggiungerlo, in quanto si esaurirà la memoria a disposizione molto prima di giungere a tale limite.

END *Sezione 2.10*

```
END
```

Questo comando fa terminare un programma BASIC; tutti i file aperti vengono automaticamente chiusi.

END SUB *Sezione 4.5*

```
END SUB
```

Con END SUB si indica la fine di un sottoprogramma; per ulteriori informazioni vedere la voce SUB.

ERASE

```
ERASE NomeMatrice [,NomeMatrice,...]
```

Il comando ERASE cancella dalla memoria una o più matrici; lo spazio occupato viene liberato perdendo ovviamente il contenuto della matrice specificata. Il ridimensionamento di una matrice tramite DIM è possibile solo dopo averla cancellata con ERASE.

ERL

```
ERL
```

Questa variabile di sistema contiene il numero dell'ultima linea eseguita prima che intercorresse un errore quando è attiva l'intercettazione degli eventi. Non è possibile ricavare il nome dell'etichetta che precedeva il punto in cui si è verificato l'errore. Vedere ON ERROR GOSUB per ulteriori informazioni.

ERR

```
ERR
```

Anche questa, come ERL, è una variabile di sistema impiegata nella intercettazione degli errori. ERR fornisce il numero d'errore: per sapere a quale errore corrisponde un determinato numero è sufficiente consultare l'Appendice A (Messaggi d'errore e d'aiuto). Per ulteriori informazioni, vedere la voce ON ERROR GOSUB.

ERROR

```
ERROR NumeroErrore
```

Questa è la terza istruzione utilizzabile in caso di errore. Quando si impiega ON ERROR GOSUB per la gestione degli errori, è possibile creare propri messaggi d'errore. Per realizzare un errore personalizzato è sufficiente prendere un numero che non viene utilizzato dal sistema ed impiegare il comando ERROR. Qualora si verifichi tale errore, la gestione degli errori provoca il richiamo della subroutine corrispondente, in cui si può utilizzare ERR per ricavare il numero corrispondente all'errore in questione. La finestra Error di AmigaBASIC non può visualizzare il messaggio d'errore che si riferisce ad un errore personalizzato; è comunque possibile creare una propria finestra Error in cui far comparire l'apposito messaggio.

Se si utilizza un numero d'errore impiegato da AmigaBASIC e non si attiva l'intercettazione degli errori, si crea un errore che non è in realtà avvenuto. Questo può essere utilizzato per proteggere i propri programmi da utenti non autorizzati. Ad esempio:

```
ERROR 2
```

darà origine ad un "Syntax error".

EXIT SUB *Sezione 4.5*

```
EXIT SUB
```

Questo comando rende possibile interrompere un sottoprogramma prima che abbia terminato l'esecuzione. Ulteriori informazioni sono disponibili sotto la voce SUB ... STATIC.

FOR ... NEXT *Sezione 1.7*

```
FOR NomeVariabile=x TO y [STEP z] NEXT [NomeVariabile]
[NomeVariabile,...]
```

Questo comando viene utilizzato per realizzare cicli (loop): tutto quello che è compreso tra FOR e NEXT viene eseguito un determinato numero di volte. La prima volta che viene percorso il ciclo, il contatore NomeVariabile possiede il valore x; ad ogni giro, tale contatore viene incrementato di un valore z (se si omette STEP z, x viene incrementato di una unità tutte le volte che viene eseguito il ciclo). Tale procedura continua fino a che NomeVariabile raggiunge un valore superiore ad y.

Non è obbligatorio ripetere NomeVariabile dopo NEXT, nel qual caso viene incrementato il contatore del ciclo FOR ... NEXT più vicino. E' possibile riunire diversi cicli FOR ... NEXT in un singolo NEXT:

```
FOR x=1 TO 100
FOR y=1 TO 20
FOR z=1 TO 30
NEXT z,y,x
```

L'esempio mostra che si possono nidificare (cioè inserire uno dentro l'altro) diversi cicli FOR ... NEXT; il ciclo più interno va chiuso prima di chiudere quelli più esterni.

FRE(x) *Sezione 4.4*

```
FRE (x)
```

Questa variabile di sistema fornisce informazioni riguardanti l'estensione di porzioni di memoria; l'immissione di un valore al posto di x specifica su quale porzione si desidera ricevere informazioni.

FRE(0) mostra quanta memoria libera è a disposizione nell'area riservata al BASIC, vale a dire il numero di byte che non sono attualmente utilizzati dal programma e dalle sue variabili.

FRE(-1) indica quanto spazio è a disposizione nell'intera memoria di sistema.

FRE(-2) specifica quanti byte di stack sono lasciati liberi da AmigaBASIC.

GOSUB ... RETURN *Sezione 1.16*

```
GOSUB Etichetta o Numero di linea
RETURN [Numero di linea]
```

E' necessario porre molta attenzione al fatto che GOSUB non si trovi all'interno di un ciclo (FOR ... NEXT, WHILE ... WEND), od all'interno di un'altra subroutine. In caso contrario è possibile che il ciclo o la subroutine non vengano terminati correttamente o addirittura che non finiscano del tutto. Ciò può portare ad errori tipo "FOR WITHOUT NEXT" o "WHILE WITHOUT WEND".

E' possibile specificare un numero di linea dopo il comando RETURN: in tal caso AmigaBASIC effettua un salto all'etichetta che segue il comando RETURN. E' necessario porre molta attenzione al fatto che GOSUB non si trovi all'interno di un ciclo (FOR ... NEXT, WHILE ... WEND), o all'interno di un'altra subroutine. In caso contrario è possibile che il ciclo o la subroutine non vengano terminati correttamente o addirittura che non finiscano del tutto. Ciò può portare ad errori tipo "FOR WITHOUT NEXT" o "WHILE WITHOUT WEND".

GOTO *Sezione 1.6*

```
GOTO Etichetta o Numero di linea
```

Questo comando obbliga il programma a saltare alla linea specificata, continuando l'esecuzione da quel punto.

IF ... THEN ... ELSE *Sezione 1.6*

```
IF espressione THEN clausola1 [ELSE clausola2]
IF espressione GOTO Numero di linea [ELSE clausola2]
IF espressione THEN Blocco di istruzioni
[ELSEIF espressione THEN Blocco di istruzioni]
[ELSE Blocco di istruzioni]
END IF
```

Il comando IF ... THEN rende possibile verificare l'esistenza di determinate condizioni in base alle quali eseguire comandi o fare salti in altre parti del programma a seconda dei risultati.

Vi sono diverse possibili sintassi per questo comando; la più semplice è la seguente:

```
IF espressione THEN clausola
```

La condizione da verificare è sempre rappresentata da un confronto che riporta un valore logico (ad esempio IF a<10 o IF Ciao=0...). Se la condizione risulta vera, AmigaBASIC dà come risultato il valore -1 (vero, true). In caso contrario il risultato è 0 (falso, false). Nell'esempio sopra riportato, la clausola che segue THEN verrà eseguita se il risultato di IF sarà vero. E' anche possibile espandere il comando con un ELSE, che indica al programma cosa fare se la condizione risulta non verificata (falsa).

```
IF a<10 THEN PRINT "a è minore di 10"
ELSE PRINT "a è maggiore di 10"
```

L'utente può immettere un GOTO seguito da un'etichetta o da un numero di linea dopo la condizione da verificare, al posto di un THEN seguito da un comando.

Nel caso in cui diverse linee di programma vadano eseguite quando la condizione risulta essere vera, si dovrebbe usare la struttura IF/ELSEIF/ELSE/END IF. In questo caso, la linea IF ... THEN imposta la condizione da verificare; non vi possono essere altri comandi dopo il THEN sulla medesima linea. I comandi presenti nelle successive linee che precedono un END IF, ELSE o ELSEIF verranno eseguiti se la condizione risulta vera. La linea opzionale contenente un ELSEIF ... THEN fornisce una seconda condizione da verificare se la prima è risultata falsa; anche in questo caso non possono essere posti sulla stessa linea altri comandi dopo il THEN. I comandi da eseguire qualora la seconda condizione risulti vera sono posti nelle linee seguenti.

Un eventuale ELSE specifica cosa il programma dovrebbe fare nel caso in cui le condizioni non risultassero verificate.

Il comando END IF conclude la struttura IF/ELSEIF/ELSE. E' possibile annidare queste strutture una dentro l'altra. Ad esempio:

```
IF...THEN
   IF...THEN
         ...
      ELSE
         ...
      END IF
   ELSEIF...THEN
      IF...THEN
         ...
      END IF
   ELSE
      ...
   END IF
```

In questo caso è indispensabile ricorrere alla programmazione strutturata.

LET ***Sezione 1.3***

```
LET NomeVariabile=valore
```

LET permette di assegnare un valore alla variabile specificata. LET è op-

zionale e può essere omesso, in quanto è sufficiente utilizzare un semplice assegnamento come a=10.

LIBRARY

```
LIBRARY NomeFile
```

Se l'utente desidera utilizzare subroutine in linguaggio macchina all'interno dei suoi programmi, non necessariamente deve scriverle da sè. E' infatti possibile usufruire delle librerie, ossia di un insieme di programmi o chiamate in linguaggio macchina che assolvono determinati compiti. L'utente necessita di informazioni sui parametri che le routine delle librerie si aspettano. Per ottenere ciò, dovrà utilizzare la documentazione esistente rivolta ai programmatori in linguaggio macchina e agli sviluppatori di software: Amiga ROM Kernel Manual e Intuition: The Amiga User Interface. In questi volumi sono riportate tutte le routine in linguaggio macchina presenti nel sistema operativo e nelle librerie.

Senza queste informazioni è praticamente impossibile fare qualcosa; per tale ragione, non è stato incluso in questo libro alcun esempio di utilizzo del comando LIBRARY. Il programma Library, presente nel cassetto BASICDemos sul dischetto Extras darà alcune spiegazioni sull'uso di questa istruzione. Per utilizzare una routine in linguaggio macchina, l'utente deve chiamarla con il comando CALL; ad esempio, nel sottoprogramma Font: contenuto nel programma Library, esiste la linea CALL CloseFont(pFont&). La routine CloseFont: fa parte della libreria graphics.library cui si accede all'inizio del programma. Tali librerie devono essere in formato .BMAP; leggere l'Appendice D per trovare spiegazioni riguardanti i programmi Library e ConvertFD. In un programma AmigaBASIC è possibile utilizzare fino a 5 librerie contemporaneamente.

Non solo è possibile chiamare con CALL programmi in linguaggio macchina, ma è anche consentito utilizzare funzioni in tale linguaggio. A queste funzioni (che, al contrario di quanto avviene per la chiamata di routine, danno come risultato un valore) si accede con il comando DECLARE FUNCTION ... LIBRARY. Ad esempio, sempre il programma Library contiene le funzioni AskSoftStyle&, OpenFont&, ed Execute&.

LIST ***Sezione 1.5***

```
LIST [Etichetta o Numero di linea] [-] [Etichetta o
Numero di linea] [,"NomeFile"]
```

Questo comando visualizza le linee del programma contenuto in memoria. Se si immette LIST senza altre opzioni, verrà listato interamente il programma, dall'inizio alla fine; è però possibile visualizzare le linee a partire da una certa etichetta o numero di linea fino ad un'altra etichetta o numero di linea.

A seconda dell'intervallo di linee da visualizzare è possibile avere:

LIST lista tutto il programma dall'inizio alla fine.

LIST Etichetta o Numero di linea visualizza la linea specificata.

LIST Inizio-DefinizioneColore lista tutte le linee comprese tra l'etichetta Inizio e quella DefinizioneColore.

LIST Inizio- visualizza tutte le linee tra l'etichetta Inizio e la fine del programma.

LIST -DefinizioneColore mostra tutte le linee di programma comprese tra l'inizio e l'etichetta DefinizioneColore.

LISTvisualizza le linee del programma all'interno della finestra con lo stesso nome; è però possibile, aggiungendo NomeFile al comando LIST, inviare il listato verso un file o un dispositivo qualunque:

LIST,"SCRN:"mostra il listato nella finestra Basic

LIST,"PRT:"invia il listato alla stampante

LIST,"DF0:TEST" manda il listato sul dischetto inserito nel drive interno in forma ASCII.

MENU *Sezione 2.8*

```
MENU NumeroMenù,NumeroOpzione,Stato [,TitoloMenù]
```

E' possibile utilizzare il comando MENU per creare menù personalizzati in AmigaBASIC. Per accedere alle opzioni contenute nei menù viene utilizzata l'intercettazione degli eventi collegati alla selezione dei menù.

NumeroMenù costituisce il numero di identificazione del menù: al primo menù a sinistra viene assegnato il numero 1; il numero massimo utilizzabile è 10.

NumeroOpzione identifica una delle opzioni disponibili nel menù in questione: all'interno di un menù possono esservi al massimo 19 opzioni. Il valore 0 corrisponde al titolo del menù stesso.

Stato può assumere tre possibili valori:

0 disabilita l'opzione indicata da NumeroOpzione che non sarà quindi disponibile. Se si specifica 0 per NumeroOpzione, l'intero menù non sarà accessibile.

1 rende accessibile l'opzione specificata da NumeroOpzione (o l'intero menù se NumeroOpzione è 0).

2 attiva un'opzione del menù ponendo un piccolo segno vicino ad essa. L'utente dovrebbe inserire due spazi nel testo di modo che il segnetto non si sovrapponga al nome dell'opzione. Non è possibile porre un segnetto a fianco del titolo del menù.

Se si utilizza il comando MENU per attivare o disattivare le singole opzioni, è necessario utilizzare questi valori. Quando si crea un nuovo menù, è necessario specificare il testo corrispondente a ciascuna opzione.

MENU *Sezione 2.8*

```
MENU(x)
```

Questa funzione BASIC fornisce informazioni sull'ultima opzione scelta in

un menù.

MENU(0) fornisce il numero del menù che contiene l'opzione selezionata per ultima.

MENU(1) riporta il numero dell'opzione prescelta in un determinato menù.

MENU ON
MENU OFF
MENU STOP

```
MENU ON
MENU OFF
MENU STOP
```

Questi comandi attivano, disattivano o sospendono l'intercettazione degli eventi correlati con la scelta dei menù. MENU ON comunica al programma di attivare l'intercettazione degli eventi; MENU OFF la disattiva (e il programma non registra quindi più simili eventi) fino al successivo MENU ON; MENU STOP sospende l'intercettazione (il programma continua a registrare la scelta di un menù o di una sua opzione, ma non effettua la chiamata all'apposita subroutine) fino ad un nuovo MENU ON.

MOUSE *Sezione 2.8*

```
MOUSE (x)
```

La funzione MOUSE fornisce informazioni riguardanti la pressione dei pulsanti ed il movimento del mouse.

Il valore che x può assumere varia da 0 a 6. MOUSE(0) ritorna lo stato del pulsante sinistro del mouse. Se MOUSE(0) viene eseguito nel corso del programma, AmigaBASIC prende nota della condizione attuale del mouse; MOUSE(0) dovrebbe essere chiamato prima di ogni altro riferimento a questa funzione (MOUSE(1) ... MOUSE(6)). Se non si intende utilizzare questo valore, è sufficiente assegnarlo ad una variabile fasulla. MOUSE(0) può assumere uno dei seguenti valori:

0 il pulsante sinistro del mouse non è premuto.

1 il pulsante sinistro del mouse non è attualmente premuto, ma lo è stato dall'ultima volta in cui è stato richiamato MOUSE(0).

2,3 il pulsante sinistro del mouse non è attualmente premuto, ma lo è stato diverse volte dall'ultima chiamata di MOUSE(0); 2 significa che è stato premuto due volte, 3 che è stato premuto tre o più volte.

-1 il pulsante sinistro è attualmente premuto dopo essere stato schiacciato una sola volta.

-2,-3 il pulsante sinistro del mouse è attualmente premuto dopo essere stato schiacciato due (-2) o tre (-3) volte o più.

MOUSE(1) ritorna la coordinata orizzontale del puntatore quando è stato chiamato MOUSE(0) l'ultima volta.

MOUSE(2) restituisce la coordinata verticale del puntatore in corrispondenza dell'ultima chiamata MOUSE(0).

MOUSE(3) dà la coordinata orizzontale di partenza del puntatore in coincidenza con l'ultima pressione del tasto sinistro del mouse prima della chiamata a MOUSE(0).

MOUSE(4) ritorna la coordinata verticale di partenza del puntatore quando è stato premuto per l'ultima volta il pulsante sinistro del mouse prima di chiamare MOUSE(0).

MOUSE(5) dà la coordinata orizzontale di arrivo del puntatore rispetto all'ultima pressione del tasto sinistro del mouse prima della chiamata a MOUSE(0).

MOUSE(6) restituisce la coordinata verticale di arrivo del puntatore in coincidenza con l'ultima pressione del tasto sinistro del mouse prima di una chiamata a MOUSE(0).

MOUSE ON
MOUSE OFF
MOUSE STOP *Sezione 2.8*

```
MOUSE ON
MOUSE OFF
MOUSE STOP
```

Questi comandi attivano, disattivano o sospendono l'intercettazione degli eventi correlati con la pressione dei pulsanti o il movimento del mouse. MOUSE ON attiva l'intercettazione degli eventi, che MOUSE OFF termina (il programma non registra più questi eventi fino a quando non viene impartito un ulteriore comando MOUSE ON); MOUSE STOP disattiva l'intercettazione fino ad un successivo MOUSE ON (il programma registra gli eventi, ma non richiama le apposite subroutine per la loro gestione).

Le subroutine che gestiscono gli eventi correlati al mouse sono indicate dal comando ON MOUSE GOSUB.

NEW *Sezione 1.9*

```
NEW
```

Il comando NEW cancella il programma che attualmente risiede in memoria, provvedendo alla chiusura dei file aperti; una eventuale modalità di funzionamento del programma a singoli passi (trace) viene disattivata dal comando TROFF.

Se l'utente non ha ancora registrato il programma, una finestra di dialogo lo informa di ciò, di modo che egli abbia la possibilità di non perdere il lavoro compiuto.

ON BREAK GOSUB

```
ON BREAK GOSUB Etichetta o Numero di linea
```

Con questo comando si indica ad AmigaBASIC quale subroutine richiamare nel caso si incontri una condizione di interruzione del programma e sia attivata l'intercettazione degli eventi.

Il punto in cui saltare può essere indicato tanto con un'etichetta quanto con un numero di linea. Se si immette 0 si disabilita l'intercettazione delle interruzioni; per riattivarla è necessario impartire il comando BREAK ON.

La subroutine dovrebbe informare l'utente che il programma non può essere interrotto durante il suo funzionamento e come uscirne in ogni caso (sarebbe bello veder comparire una richiesta del tipo "Sei sicuro di voler uscire ?", con bottoni selezionabili per il Sì o per il No).

AmigaBASIC blocca l'intercettazione delle interruzioni mentre è all'interno della subroutine, di modo che un secondo evento non disturbi la gestione del primo.

La subroutine in questione deve essere conclusa da un RETURN.

ON ERROR GOSUB

```
ON ERROR GOSUB Etichetta o Numero di linea
```

Con questo comando si danno informazioni ad AmigaBASIC riguardanti la subroutine da richiamare in caso di errori e di attivazione dell'intercettazione degli eventi ad essi correlati. L'utente può specificare un'etichetta od un numero di linea; immettendo 0, viene disabilitata l'interce azione degli errori.

E' possibile utilizzare le variabili ERR per scoprire in quale errore si è incorsi, e ERL per risalire all'ultima linea eseguita prima di quella in cui si è verificato l'errore.

Con il comando ERROR è anche possibile utilizzare numeri d'errore personalizzati all'interno di un programma. Se nella subroutine richiamata da ON ERROR GOSUB si verifica un altro errore, AmigaBASIC blocca del tutto il programma. Per ritornare al programma principale, è necessario immettere il comando RESUME al termine della subroutine per la gestione degli errori (vedere RESUME per ulteriori informazioni).

ON ... GOSUB *Sezione 2.9*

```
ON x GOSUB Etichetta o Numero di linea [,Etichetta]
[,Etichetta]...
```

Con questo comando è possibile richiamare diverse subroutine a seconda del valore di x. Se x=1, il programma salta alla prima etichetta, se x=2 alla seconda e così via. Se x è un numero decimale a singola precisione, il suo valore verrà arrotondato. Se x contiene un valore superiore al numero delle etichette che seguono GOSUB, il programma continua la sua esecuzione senza effettuare nessun richiamo di subroutine. Se x ha valore negativo, viene visualizzato il messaggio d'errore "Illegal function call".

ON ... GOTO *Sezione 2.9*

```
ON x GOTO Etichetta o Numero di linea [,Etichetta]
[,Etichetta]...
```

ON ... GOTO agisce nello stesso modo di ON ... GOSUB, cui si rimanda per ulteriori informazioni. L'unica differenza consiste nel fatto che il programma non chiama una subroutine, ma salta ad un'altra sezione del corpo principale.

ON MENU GOSUB *Sezione 2.8*

```
ON MENU GOSUB Etichetta o Numero di linea
```

Questo comando indica ad AmigaBASIC quale subroutine chiamare quando l'utente effettua la selezione di un'opzione da un menù, a patto che, ovviamente, sia stata attivata l'intercettazione degli eventi correlati alla scelta dei menù. Anche in questo caso è possibile indicare la subroutine con un'etichetta od un numero di linea. Inserendo 0, si disabilita la gestione dei menù, che può essere riattivata dal comando MENU ON. La funzione MENU(x) permette di rilevare quale opzione è stata selezionata. AmigaBASIC blocca la gestione dei menù mentre è attiva la subroutine, di modo che un secondo evento non può sovrapporsi al primo. La subroutine deve concludersi con un comando RETURN.

ON MOUSE GOSUB *Sezione 2.8*

```
ON MOUSE GOSUB Etichetta o Numero di linea
```

Questo comando informa AmigaBASIC su quale subroutine richiamare quando l'utente schiaccia il pulsante sinistro del mouse ed è attiva l'intercettazione di tale evento. Il punto a cui saltare può essere indicato con un'etichetta o con un numero di linea. Se viene immesso 0 come valore di Etichetta, la gestione del mouse viene disabilitata. Per riattivarla è necessario impartire il comando MOUSE ON. La funzione MOUSE(x) fornisce informazioni riguardanti il mouse ed il suo stato. AmigaBASIC blocca l'intercettazione degli eventi correlati al mouse quando è in esecuzione la subroutine incaricata di trattare tali eventi, per evitare che un secondo evento possa interferire col primo.

La subroutine deve essere terminata da un RETURN.

ON TIMER GOSUB

```
ON TIMER (x) GOSUB Etichetta o Numero di Linea
```

Questo comando indica ad AmigaBASIC quale subroutine richiamare se sono trascorsi x secondi dall'ultimo evento temporale, nel caso in cui sia attiva l'intercettazione di tali eventi. In questo modo è possibile chiamare una subroutine od un sottoprogramma ad intervalli regolari. x deve avere un valore compreso tra 1 (un secondo) e 86400 (60*60*24 secondi = 24 ore). Il punto cui saltare può essere indicato con una etichetta od un numero di linea. Se si immette 0 come etichetta, si disabilita l'intercettazione degli eventi collegati allo scorrere degli intervalli di tempo, che è poi possibile riattivare con TIMER ON.

AmigaBASIC blocca la gestione degli intervalli di tempo quando la subroutine è in esecuzione, per impedire che un secondo evento possa interferire con il primo. La subroutine deve essere terminata con un RETURN.

OPTION BASE

```
OPTION BASE x
```

Le matrici hanno solitamente il primo elemento identificato dall'indice 0 (ad esempio Ciao(0), Tinta(0,0)). Se si desidera che il primo elemento della matrice abbia indice 1 (Ciao(1), Tinta(1,1)), è necessario ricorrere al comando OPTION BASE 1. Questa istruzione deve essere eseguita prima di ogni comando DIM e prima di accedere agli elementi di una matrice. In caso contrario apparirà il messaggio d'errore "Duplicate definition".

RANDOMIZE

```
RANDOMIZE [x]
RANDOMIZE TIMER
```

Se si utilizzano numeri casuali generati dall'istruzione RND, si otterranno gli stessi valori ogni volta che il programma verrà eseguito. Il comando RANDOMIZE permette di evitare questo inconveniente fornendo una serie di numeri casuali.

Inserendo al posto di x un valore compreso tra -32768 e 32767, tale numero sarà usato per il calcolo di numeri casuali. Se in una seconda occasione si immette lo stesso valore, si otterrà la stessa sequenza di numeri casuali; se non si fornisce alcun valore, AmigaBASIC ne richiede l'immissione con il messaggio:

```
Random Number Seed (-32678 to 32767)?
```

Se non si desidera immettere alcun valore manualmente, è necessario utilizzare RANDOMIZE TIMER, per cui AmigaBASIC utilizza il valore dell'ora corrente per eseguire i calcoli; in questo modo è possibile generare numeri effettivamente casuali: il programma calcolerà numeri diversi ogni volta che verrà eseguito.

READ *Sezione 2.9*

```
READ Lista-variabili
```

Con questo comando si legge un singolo valore da una linea di DATA e lo si attribuisce alla variabile indicata dopo READ. La variabile deve essere dello stesso tipo del valore letto (stringa per stringa, numero per numero).

Ogni comando READ incrementa un puntatore interno di DATA. Se tutte le istruzioni DATA sono state lette prima dell'esaurimento dei comandi READ, si va incontro all'errore "Out of DATA".

REM *Sezione 4.6*

```
REM [Testo di commento]
```

L'utente può utilizzare REM per inserire un commento alla fine di una linea BASIC. E' anche possibile introdurre una intera linea di testo. REM deve essere l'ultimo comando della linea, oppure l'unico contenuto in una riga di programma. Ogni cosa che lo segue su quella linea verrà ignorato da AmigaBASIC. E' possibile utilizzare l'apostrofo (') al posto di REM, se lo si desidera.

RESTORE *Sezione 4.8*

```
RESTORE [Etichetta o Numero di linea]
```

Come evidenziato nelle voci DATA e READ, dopo ogni lettura READ incrementa un puntatore che punta al prossimo elemento DATA da leggere.

E' possibile utilizzare RESTORE per spostare tale puntatore ad una linea od etichetta specificata (sia in avanti che indietro). Se non si immette una etichetta od un numero di linea dopo il comando, il puntatore di DATA viene azzerato come all'inizio del programma.

RESUME

```
RESUME [0]
RESUME NEXT
RESUME NumeroLinea
```

Il comando RESUME viene utilizzato per concludere le subroutine che gestiscono gli errori rilevati tramite l'intercettazione di eventi (vedere ON ERROR GOSUB). RESUME o RESUME 0 riesegue il comando che ha causato l'errore. RESUME NEXT ritorna alla linea successiva a quella in cui si è verificato l'errore: in questo caso il comando che ha causato l'errore vie-

ne ignorato. RESUME NumeroLinea permette di saltare a qualunque etichetta, e può quindi essere visto come una particolare versione del comando RETURN.

RUN

```
RUN [Etichetta]
RUN NomeFile [,R]
```

Il comando RUN consente di avviare l'esecuzione del programma attualmente presente in memoria. Lo schermo viene cancellato, la finestra LIST viene posta in secondo piano e ogni file aperto viene chiuso. RUN può essere richiamato sia all'interno della finestra BASIC, che all'interno di un programma. Inserendo un'etichetta dopo RUN, il programma inizia la sua esecuzione partendo dal punto specificato. Se si indica un particolare programma come NomeFile, questo viene caricato in memoria ed eseguito; se dopo il nome del file si inserisce ,R, tutti i file aperti rimarranno tali. In questo modo è possibile caricare ed eseguire un programma che eseguirà operazioni su file precedentemente aperti.

SHARED *Sezione 4.5*

```
SHARED Lista-variabili
```

Questo comando, che va inserito all'interno di un sottoprogramma, specifica le variabili del programma principale che possono essere utilizzate anche nel sottoprogramma. I nomi delle variabili vanno immessi dopo il comando SHARED. Le matrici sono identificate da una coppia di parentesi tonde:

```
SUB Test STATIC
SHARED NumeroTest, MatriceTest(), Test$
```

E' possibile usare diverse istruzioni SHARED, ma solo all'interno di un sottoprogramma. Confrontare questo con il comando DIM che dispone anch'esso di un'opzione SHARED.

SLEEP

```
SLEEP
```

Questo comando risulta utile quando si desidera gestire eventi come pressioni del pulsante sinistro del mouse o di tasti, collisioni fra oggetti, selezione di menù od eventi temporali; infatti esso sospende l'esecuzione del programma fino a quando non accade una condizione che possa essere rilevata dall'apposita ruotine di gestione. Non è necessario che la routine di rilevazione sia attiva.

Le seguenti condizioni causano la terminazione di questo comando:

Pressione di un tasto della tastiera

Pressione del pulsante sinistro del mouse

Selezione di un'opzione da un menù

Collisione tra oggetti grafici

Conclusione di un evento legato a TIMER

SLEEP risulta utile anche quando si desidera che l'utente prema un tasto qualunque sulla tastiera; infatti, diversamente da INKEY$, SLEEP riconosce anche tasti come <SHIFT>, <ALT>, <CTRL>, <AMIGA>.

STOP

```
STOP
```

Con STOP viene interrotta l'esecuzione di un programma BASIC; per farne riprendere il funzionamento è necessario utilizzare CONT. STOP non chiude i file aperti: è importante tener presente questo fatto quando si ritorna ad operare in modo diretto.

SUB ... STATIC

```
SUB NomeSottoprogramma [(Lista-parametri,...)] STATIC
```

Per iniziare un sottoprogramma è necessario utilizzare questo comando. Un sottoprogramma è una subroutine indipendente dal resto del programma: esso infatti usa variabili proprie, ma può utilizzare anche quelle del programma principale tramite il comando SHARED.

Ogni sottoprogramma è identificato da un nome utilizzato per richiamarlo; la sua chiamata può essere effettuata con il comando CALL, ma è anche possibile omettere tale istruzione. NomeSottoprogramma può essere lungo al massimo 40 caratteri. E' anche possibile passare variabili ad un sottoprogramma: è però necessario assicurarsi che le variabili passate siano dello stesso tipo di quelle dichiarate nella linea SUB ... STATIC. Le matrici vanno indicate con parentesi tonde:

```
SUB Ciao (Test, Colori(), Alpha) STATIC
```

Un sottoprogramma viene terminato da un comando END SUB; il codice contenuto tra le linee SUB ed END SUB viene eseguito quando il programma principale chiama la routine in questione.

Se un sottoprogramma si trova all'interno del programma principale, AmigaBASIC lo salta, continuando l'esecuzione dalla linea successiva a END SUB (se il sottoprogramma non viene chiamato).

Tramite il comando EXIT SUB è possibile uscire da un sottoprogramma prima di aver raggiunto il comando END SUB.

SWAP

```
SWAP Variabile1,Variabile2
```

SWAP permette di scambiare tra loro i valori assunti da due variabili; dopo questo comando, Variabile1 ha come valore quello contenuto in precedenza da Variabile2 e viceversa. Le due variabili devono essere dello stesso tipo.

SYSTEM *Sezione 1.9*

```
SYSTEM
```

Questo comando consente di uscire da AmigaBASIC e tornare in ambiente Workbench o CLI.

Se il programma correntemente attivo non è stato ancora registrato, AmigaBASIC avverte di ciò consentendo di salvare il lavoro svolto.

TIMER ON
TIMER OFF
TIMER STOP

```
TIMER ON
TIMER OFF
TIMER STOP
```

Questi comandi permettono di controllare l'intercettazione degli eventi collegati allo scorrere del tempo; TIMER ON attiva tale intercettazione, TIMER OFF la disabilita completamente, TIMER STOP la sospende fino al successivo TIMER ON (anche se gli eventi temporali vengono registrati, il programma non richiama le subroutine per la loro gestione).

Con il comando ON TIMER GOSUB è possibile indicare la subroutine od il sottoprogramma da richiamare quando si è in presenza di un evento temporale.

TRON
TROFF *Sezione 1.14*

```
TRON
TROFF
```

Con questi due comandi è possibile controllare la funzione Trace: TRON la attiva, TROFF la disattiva.

Quando Trace è attivo, il comando che il programma sta eseguendo vie-

ne mostrato nella finestra LIST circondato da un bordo arancione. Ciò può rivelarsi molto utile quando si sta verificando il funzionamento di un programma.

WHILE ... WEND *Sezione 2.8*

```
WHILE Condizione
[Blocco di istruzioni] WEND
```

Il ciclo WHILE ... WEND viene eseguito fino a quando la condizione indicata rimane vera; quando questo non accade più, il programma prosegue con l'esecuzione della linea successiva al comando WEND.

Un tipico esempio può essere l'attesa di un'immissione da tastiera:

```
WHILE INKEY$="" : WEND
```

E' possibile nidificare uno dentro l'altro i cicli WHILE ... WEND.

B.5 Funzioni del BASIC per il calcolo

ABS *Sezione 1.17*

```
ABS(Valore)
```

La funzione ABS restituisce il valore assoluto di un numero, cioè il numero stesso senza l'eventuale segno positivo o negativo. Ad esempio, ABS(-2) vale 2; ABS(0) vale 0.

AND *Nota d'uso 3*

```
Valore=Valore1 AND Valore2
```

L'operazione logica AND restituisce un valore combinando i singoli bit di Valore1 e Valore2 nel modo seguente:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

ASC *Sezione 4.3*

```
Valore=ASC(Stringa)
```

La funzione ASC restituisce il valore del codice ASCII (American Standard Code for Information Interchange) del primo carattere della sequenza inclusa tra parentesi. Se la stringa è vuota, si otterrà il messaggio d'errore "Illegal function call". Una tabella dei codici ASCII è compresa sotto la voce CHR$.

ATN

```
Valore=ATN(Valore)
```

La funzione ATN calcola l'arcotangente del valore tra parentesi fornendo il risultato in radianti. Il risultato è compreso tra -1/2*π e +1/2*π. Se si desidera convertire i radianti in gradi, è possibile utilizzare una delle seguenti formule:

```
Gradi = (180 * Radianti) / p
Gradi = 57.296 * Radianti
```

CDBL
CINT
CLNG
CSNG *Sezione 1.17*

```
CDBL(Valore)
CINT(Valore)
CLNG(Valore)
CSNG(Valore)
```

Queste funzioni permettono di convertire un numero da un formato in un altro.

CDBL converte valore producendo come risultato un numero decimale in doppia precisione.

CINT dà origine ad un numero intero a 16 bit che verrà arrotondato nel consueto modo (es., CINT(3.4)=3 e CINT(3.5)=4); il valore tra parentesi deve essere compreso tra -32768 e +32767.

CLNG produce un intero a 32 bit, eventualmente arrotondando la parte decimale come visto sopra. Il valore tra parentesi deve essere compreso tra -2147483648 e +2147483647.

CSNG produce un numero decimale a singola precisione.

Se Valore è troppo grande per essere compreso in una variabile del tipo

prescelto, si ottiene il messaggio d'errore "Overflow". Vedere la Nota d'uso 5 per ulteriori informazioni sui diversi tipi numerici.

CHR$ *Sezione 1.15*

```
CHR$(Valore)
```

L'istruzione CHR$ fornisce il carattere corrispondente al numero avente la codifica ASCII (American Standard Code for Interchanging Information) specificata tra parentesi.

Valore deve essere un numero intero compreso tra 0 e 255.

Il codice ASCII assegna ad ogni carattere stampabile o di controllo un numero di codice tra 0 e 255.

I numeri con codifica compresa tra 128 e 255 sono non-standard, ma Amiga usa tali codici per i tasti funzione e i caratteri internazionali.

La figura 21 nelle pagine seguenti riporta la tabella delle codifiche ASCII utilizzate da Amiga; ai numeri aventi a fianco un rettangolino non è stato assegnato alcun carattere.

COS *Sezione 2.5*

```
Valore=COS(Valore)
```

La funzione COS calcola il coseno del valore tra parentesi, che deve essere espresso in radianti.

Se si desidera effettuare la conversione gradi -> radianti, è possibile utilizzare una delle seguenti formule:

```
Radianti = (π * Gradi) / 180
Radianti = 0.0175 * Gradi
```

0 ⟨CTRL⟩ ⟨@⟩	33 !	66 B	99 c
1 ⟨CTRL⟩ ⟨A⟩	34 "	67 C	100 d
2 ⟨CTRL⟩ ⟨B⟩	35 #	68 D	101 e
3 ⟨CTRL⟩ ⟨C⟩ ⟨Break⟩	36 $	69 E	102 f
4 ⟨CTRL⟩ ⟨D⟩	37 %	70 F	103 g
5 ⟨CTRL⟩ ⟨E⟩	38 &	71 G	104 h
6 ⟨CTRL⟩ ⟨F⟩	39 '	72 H	105 i
7 ⟨CTRL⟩ ⟨G⟩ ⟨Beep⟩	40 (	73 I	106 j
8 ⟨CTRL⟩ ⟨H⟩ ⟨Backspace⟩	41)	74 J	107 k
9 ⟨CTRL⟩ ⟨I⟩ ⟨Tab⟩	42 *	75 K	108 l
10 ⟨CTRL⟩ ⟨J⟩ ⟨Line feed⟩	43 +	76 L	109 m
11 ⟨CTRL⟩ ⟨K⟩	44 ,	77 M	110 n
12 ⟨CTRL⟩ ⟨L⟩ ⟨Delete⟩	45 -	78 N	111 o
13 ⟨CTRL⟩ ⟨M⟩ ⟨Return⟩	46 .	79 O	112 p
14 ⟨CTRL⟩ ⟨N⟩	47 /	80 P	113 q
15 ⟨CTRL⟩ ⟨O⟩	48 0	81 Q	114 r
16 ⟨CTRL⟩ ⟨P⟩	49 1	82 R	115 s
17 ⟨CTRL⟩ ⟨Q⟩	50 2	83 S	116 t
18 ⟨CTRL⟩ ⟨R⟩	51 3	84 T	117 u
19 ⟨CTRL⟩ ⟨S⟩	52 4	85 U	118 v
20 ⟨CTRL⟩ ⟨T⟩	53 5	86 V	119 w
21 ⟨CTRL⟩ ⟨U⟩	54 6	87 W	120 x
22 ⟨CTRL⟩ ⟨V⟩	55 7	88 X	121 y
23 ⟨CTRL⟩ ⟨W⟩	56 8	89 Y	122 z
24 ⟨CTRL⟩ ⟨X⟩	57 9	90 Z	123 {
25 ⟨CTRL⟩ ⟨Y⟩	58 :	91 [	124 |
26 ⟨CTRL⟩ ⟨Z⟩	59 ;	92 \	125 }
27 ⟨CTRL⟩ ⟨[⟩ ⟨Escape⟩	60 <	93]	126 ~
28 ⟨CTRL⟩ ⟨\⟩ ⟨Up⟩	61 =	94 ^	127
29 ⟨CTRL⟩ ⟨]⟩ ⟨Down⟩	62 >	95 _	128 □
30 ⟨CTRL⟩ ⟨^⟩ ⟨Right⟩	63 ?	96 `	129 □ ⟨F1⟩
31 ⟨CTRL⟩ ⟨_⟩ ⟨Left⟩	64 @	97 a	130 □ ⟨F2⟩
32 ⟨Space bar⟩	65 A	98 b	131 □ ⟨F3⟩

132 □ (F4)
133 □ (F5)
134 □ (F6)
135 □ (F7)
136 □ (F8)
137 □ (F9)
138 □ (F10)
139 □ (Help)
140 □
141 □
142 □
143 □
144 □
145 □
146 □
147 □
148 □
149 □
150 □
151 □
152 □
153 □
154 □
155 □
156 □
157 □
158 □
159 □
160 Non-break space
161 ¡
162 ¢
163 £
164 ¤
165 ¥
166 ¦
167 §
168 ¨
169 ©
170 ª
171 «
172 ¬
173 -
174 ®
175 ¯
176 °
177 ±
178 ²
179 ³
180 ´
181 µ
182 ¶
183 ·
184 ¸
185 ¹
186 º
187 »
188 ¼
189 ½
190 ¾
191 ¿
192 À
193 Á
194 Â
195 Ã
196 Ä
197 Å
198 Æ
199 Ç
200 È
201 É
202 Ê
203 Ë
204 Ì
205 Í
206 Î
207 Ï
208 Ð
209 Ñ
210 Ò
211 Ó
212 Ô
213 Õ
214 Ö
215 □
216 Ø
217 Ù
218 Ú
219 Û
220 Ü
221 Ý
222 Þ
223 ß
224 à
225 á
226 â
227 ã
228 ä
229 å
230 æ
231 ç
232 è
233 é
234 ê
235 ë
236 ì
237 í
238 î
239 ï
240 ð
241 ñ
242 ò
243 ó
244 ô
245 õ
246 ö
247 □
248 ø
249 ù
250 ú
251 û
252 ü
253 ý
254 þ
255 ÿ

CVD
CVI
CVL
CVS *Sezione 4.1*

```
Valore=CVD(Stringa da 8 byte)
Valore=CVI(Stringa da 2 byte)
Valore=CVL(Stringa da 4 byte)
Valore=CVS(Stringa da 4 byte)
```

La lettura e la scrittura nei file di numeri sotto forma di stringhe è più veloce ed occupa meno spazio di quanto richiederebbe la scrittura o la lettura del numero sotto forma di cifre. La funzione MK*$ aiuta l'utente in questa operazione di conversione. Per riconvertire in numeri le stringhe così create, sono disponibili queste quattro funzioni.

CVD converte una stringa lunga 8 byte in un numero decimale a doppia precisione.

CVI trasforma una stringa di 2 byte in un intero a 16 bit.

CVL converte una stringa lunga 4 byte in un intero a 32 bit.

CVS trasforma una stringa di 4 byte in un numero decimale a singola precisione.

Vedere la Nota d'uso 5 per avere ulteriori informazioni sui tipi numerici a disposizione.

DATE$ *Sezione 1.17*

```
Stringa=DATE$
```

Questa funzione fornisce la data attuale di sistema sotto forma di una stringa lunga 10 caratteri. La data attuale è quella specificata tramite Preferences o quella ricavata dall'eventuale orologio interno. Il formato della data è MM-GG-AAAA (mese, giorno, anno).

EQV *Nota d'uso 3*

```
Valore=Valore1 EQV Valore2
```

L'operazione logica EQV combina i singoli bit di Valore1 e Valore2 per fornire i seguenti risultati:

0 EQV 0 = 1
0 EQV 1 = 0
1 EQV 0 = 0
1 EQV 1 = 1

EXP

```
Valore=EXP(Valore)
```

La funzione EXP calcola l'esponente di e (e=2.7182818284590) che costituisce la base dei logaritmi naturali.

Vedere la voce LOG per ulteriori informazioni.

FIX

```
Valore=FIX(Valore)
```

Questa funzione ha lo scopo di rimuovere le cifre decimali presenti in un numero. Ad esempio, l'istruzione FIX(3.235325) fornisce come risultato 3 e FIX (-2.3532325) è pari a -2. Al contrario di quanto avviene per INT e CINT, FIX non arrotonda il numero, ma restituisce solamente le cifre rimaste dopo aver cancellato quelle decimali.

HEX$

```
Stringa=HEX$(Valore)
```

Questa funzione produce una stringa che rappresenta il numero esadecimale corrispondente ad un dato numero decimale; Valore è un numero

compreso tra -32768 e +65535.

Ad esempio:

```
PRINT HEX$(60037)
```

produce come risultato EA85. Ulteriori informazioni sui numeri esadecimali sono contenute nella Nota d'uso 3.

IMP *Nota d'uso 3*

```
Valore=Valore1 IMP Valore2
```

L'operatore logico IMP fornisce come risultato Valore utilizzando i singoli bit di Valore1 e Valore2 nel modo seguente:

0 IMP 0 = 1
0 IMP 1 = 1
1 IMP 0 = 0
1 IMP 1 = 1

INSTR

```
Valore=INSTR( [Valore,] Stringa, Stringa da cercare)
```

Questa funzione viene utilizzata per ricercare una particolare sequenza di caratteri all'interno di una stringa. Se tale ricerca ha successo, la funzione fornisce la posizione da cui inizia la sequenza di caratteri all'interno della stringa.

INSTR ritorna il valore 0 qualora la sequenza ricercata non sia contenuta nella stringa in esame, oppure sia una sequenza nulla (cioè non composta da alcun carattere), oppure sia più lunga della stringa in cui va ricercata.

E' possibile anche specificare la posizione da cui far partire la ricerca all'interno della stringa; se tale posizione è un valore superiore alla lunghezza totale della stringa, la funzione INSTR darà 0 come risultato.

```
INSTR("Amiga","iga")      risultato      3
INSTR("Ciao","iga")           "          0
INSTR(3,"Ciao","Ci")          "          0
```

INT *Sezione 1.17*

```
Valore=INT(Valore)
```

Questa funzione converte una numero decimale in un numero intero. Rispetto a CINT e FIX, possiede queste caratteristiche: elimina ciò che si trova dopo il punto decimale senza arrotondare il numero se questo è positivo, arrotondandolo se invece è negativo; INT restituisce l'intero minore o uguale più vicino al numero indicato.

```
INT(3.4)        risultato      3
INT(3.8)             "         3
INT(-2.2)            "        -3
```

La funzione INT può operare su qualunque tipo di numero.

LBOUND
UBOUND

```
Valore=LBOUND(NomeMatrice [,Dimensione]
Valore=UBOUND(NomeMatrice [,Dimensione])
```

Queste funzioni ritornano l'indice del primo (Lower BOUNDary) e dell'ultimo (Upper BOUNDary) elemento contenuto in una matrice. In tal modo si possono recuperare i valori definiti tramite i comandi OPTION BASE e DIM.

Ad esempio, nel caso di OPTION BASE 0 e DIM a(200), LBOUND(a) equivale a 0 e UBOUND(a) ritorna 200.

Se si indica anche una dimensione della matrice, è possibile ottenere le informazioni suddette riguardanti singole dimensioni.

Ad esempio, nel caso di DIM a(3,4,56), UBOUND (a,1) fornisce il valore

della prima dimensione della matrice, cioè 3; UBOUND (a,2) dà 4 e UBOUND (a,3) dà 56.

Se non si specifica una dimensione particolare, il risultato che si ottiene è relativo alla prima dimensione della matrice.

LEFT$ *Sezione 1.16*

```
Stringa=LEFT$(Stringa,Numero)
```

LEFT$ produce come risultato una sequenza di caratteri composta dalla porzione sinistra della stringa specificata. Numero equivale alla lunghezza della sequenza da ottenere (cioè il numero di caratteri da estrarre dalla stringa specificata tra parentesi). Poichè AmigaBASIC può maneggiare stringhe non più lunghe di 32767 caratteri, Numero deve avere un valore compreso tra 0 e 32767.

```
LEFT$("Amiga sei grande !",5)    risultato    "Amiga"
```

LEN *Sezione 1.17*

```
Valore=LEN(Stringa)
```

La funzione LEN fornisce come risultato il numero di caratteri contenuto nella stringa specificata tra parentesi. Vengono contati anche gli spazi e i caratteri di controllo eventualmente presenti.

```
LEN("Amiga sei grande !")      risultato      18
```

LOG *Sezione 4.8*

```
Valore=LOG(Valore)
```

La funzione LOG calcola il logaritmo naturale di un numero, cioè il logaritmo di e (e = 2.7182818284590).

Il logaritmo di a in base b è il numero c, dove b^c=a. LOG(a) in base c è c.

E' possibile che, per utilizzare i logaritmi nei calcoli di un programma BASIC (come nella routine Salvaimmagine presentata in questo libro), sia necessario avere a disposizione una base diversa per il calcolo del logaritmo.

La formula per trovare il logaritmo di un numero in un'altra base è:

```
LOG(a) in base b = LOG(a) / LOG(b)
```

MID$ *Sezione 4.7*

```
Stringa=MID$(Stringa,Posizione [,Lunghezza])
```

Con MID$ è possibile estrarre una porzione di stringa dal contesto in cui è compresa.

E' sufficiente indicare la stringa su cui eseguire l'operazione e la posizione del primo carattere da estrarre; è inoltre possibile specificare la lunghezza della parte da estrarre.

Se non si indica tale lunghezza, il risultato sarà una stringa che conterrà tutti i caratteri dal punto specificato in poi.

```
MID$("Amiga sei grande !",11,6)   risultato   "grande"
```

MID$

```
MID$(Stringa,Posizione [,Lunghezza])=Rimpiazzo
```

Questa versione dell'istruzione MID$ viene utilizzata per sostituire parte della stringa specificata con una diversa sequenza di caratteri.

Il parametro Posizione indica il punto da cui inizia la sostituzione della vecchia parte con la nuova; se viene specificata Lunghezza, verrà sostituito solo il numero di caratteri indicato.

```
a$="Amiga sei super !" : MID$(a$,11)="grande"
PRINT a$            risultato      "Amiga sei grande!"
```

MKD$
MKI$
MKL$
MKS$ *Sezione 4.1*

```
MKD$(Numero decimale a doppia precisione)
MKI$(Numero intero a 16 bit)
MKL$(Numero intero a 32 bit)
MKS$(Numero decimale a singola precisione)
```

E' possibile risparmiare tempo e memoria registrando valori numerici sotto forma di stringhe composte da tanti caratteri quanti sono i byte che compongono i numeri:

MKD$ converte un numero decimale in doppia precisione in una stringa lunga 8 byte.

MKI$ trasforma un numero intero a 16 bit in una stringa di 2 byte.

MKL$ effettua la conversione da numero intero a 32 bit a stringa di 4 byte.

MKS$ trasforma un numero decimale a singola precisione in una stringa lunga 4 byte.

E' possibile riconvertire le stringhe ottenute nei valori numerici di partenza utilizzando le istruzioni CV* (vedere le voci corrispondenti).

NOT *Nota d'uso 3*

```
Valore=NOT Valore
```

L'operatore logico NOT converte 0 in -1 e -1 in 0 invertendo, ad esempio, il valore di una condizione:

```
IF NOT (a<1) THEN...
```

In questo esempio le istruzioni che seguono THEN verranno eseguite se a non è minore di 1, cioè se è maggiore od uguale a 1. Questo si può ot-

tenere con la seguente formula:

```
NOT x = -(x+1)
```

Ciò non ha molto senso per valori che non siano -1 e 0.

OCT$

```
Stringa=OCT$(Valore)
```

Questa funzione ha come risultato quello di dare origine ad una stringa che contiene la rappresentazione ottale del numero indicato tra parentesi. Per altre informazioni sui numeri ottali, vedere la Nota d'uso 3.

```
OCT$(60037)          risultato       "165205"
```

OR **Nota d'uso 3**

```
Valore=Valore1 OR Valore2
```

L'operatore logico OR produce il suo risultato combinando nel modo seguente i singoli bit di Valore1 e Valore2:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

PEEK **Sezione 4.3**

```
PEEK(Locazione di memoria)
```

PEEK visualizza il contenuto della locazione di memoria specificata, ritornando un numero compreso tra 0 e 255, cioè un byte. La locazione di memoria più alta in Amiga è quella identificata dal valore 16777215. Molte delle locazioni in questo ambito non avranno niente al loro interno, a meno che il sistema non abbia 8 M di RAM.

PEEKL *Sezione 4.3*

```
PEEKL(Locazione di memoria)
```

PEEKL ritorna il contenuto di 4 byte consecutivi con partenza dalla locazione specificata, che deve essere un numero pari. Il valore del risultato è un numero compreso tra -2147483648 e +2147483647, cioè un intero a 32 bit (= 4 byte). Vedere anche la voce PEEK per altre informazioni.

PEEKW *Sezione 4.3*

```
PEEKW(Locazione di memoria)
```

PEEKW mostra il contenuto di due byte di memoria a partire dalla locazione specificata, che deve essere un numero pari. Il risultato è un numero con valore compreso tra -32768 e +32767, un intero a 16 bit (= 2 byte). Vedere anche la voce PEEK per altre informazioni.

POKE *Sezione 4.3*

```
POKE Locazione di memoria,Valore
```

Con POKE è possibile inserire un valore ad 8 bit (= 1 byte) all'interno della locazione di memoria specificata. Altre informazioni sugli indirizzi di memoria sono sotto la voce PEEK.

E' necessario porre molta attenzione quando si scrivono valori direttamente in memoria. Se non si pianifica il proprio lavoro correttamente e non si conosce a sufficienza il sistema operativo di Amiga, è facile arrecare più danno che bene.

POKEL *Sezione 4.3*

```
POKEL Locazione di memoria,Valore
```

L'istruzione POKEL scrive in uno spazio di memoria ampio 4 byte, a partire dalla locazione di memoria specificata, un valore a 32 bit (= 4 byte). La locazione di partenza deve avere un valore pari. Altre informazioni re-

lative a questo comando possono essere trovate sotto le voci PEEKL e POKE.

POKEW *Sezione 4.3*

```
POKEW Locazione di memoria,Valore
```

POKEW immette un valore a 16 bit (= 2 byte) in uno spazio di memoria ampio 2 byte, a partire dalla locazione indicata, che deve essere un valore pari. Altre informazioni riguardanti gli accessi sono riportate sotto la voce PEEK, ed ulteriori nozioni possono essere dedotte dalle voci PEEKW e POKE.

RIGHT$ *Sezione 1.16*

```
Stringa=RIGHT$(Stringa,Lunghezza)
```

Questa funzione permette di creare una sequenza di caratteri lunga quanto specificato dal parametro Lunghezza, estraendola dalla parte terminale della stringa indicata.

```
RIGHT$("Amiga sei grande !",8)  risultato  "grande !"
```

RND *Sezione 2.5*

```
Valore=RND [(x)]
```

La funzione RND produce come risultato un numero casuale compreso tra 0 ed 1.

Inserendo un numero tra parentesi, è possibile controllare la generazione dei numeri casuali: 1 e tutti i numeri positivi producono una sequenza fissa di numeri casuali. Ogni volta che il programma verrà eseguito, verrà sempre creata la stessa sequenza di numeri casuali: questo è il modo di funzionamento di base.

0 ritorna l'ultimo numero casuale generato. Se si lavora con RND(0) si ottiene sempre lo stesso numero. -1 e tutti i numeri negativi passano un

nuovo valore di partenza per la generazione di numeri casuali. Se l'utente immette gli stessi valori, otterrà la medesima sequenza di numeri casuali. Ulteriori notizie possono essere reperite sotto la voce RANDOMIZE.

Spesso è necessario ottenere un numero casuale che si trovi all'interno di un preciso intervallo; la formula per calcolarlo è:

```
NumeroCasuale=InizioIntervallo+(FineIntervallo)*RND)
```

Se il numero deve essere intero, la formula è:

```
NumeroCasuale=INT(InizioIntervallo+
(FineIntervallo)*RND)
```

SADD

```
Valore=SADD(Stringa)
```

Questa funzione viene utilizzata soprattutto nei programmi in linguaggio macchina. Essa fornisce la locazione di memoria di partenza della stringa specificata.

Se si utilizzano altre stringhe all'interno del programma, è necessario chiedere di nuovo l'indirizzo di partenza, in quanto AmigaBASIC cambia spesso la locazione delle varie stringhe utilizzate.

SGN

```
Valore=SGN(Valore)
```

La funzione SGN viene usata per ricavare il segno di una variabile; essa ritorna come risultato:

-1 per i numeri negativi
0 per il numero 0
1 per i numeri positivi

SIN *Sezione 2.5*

```
Valore=SIN(Valore)
```

La funzione SIN calcola il seno di un angolo indicato in radianti. Se si preferisce lavorare in gradi, è sufficiente andare alla voce COS per reperire la formula di conversione dei valori da radianti a gradi.

SPACE$ *Sezione 5.2*

```
Stringa=SPACE$(Lunghezza)
```

Questa funzione produce una stringa contenente un numero di spazi uguali a quello specificato da Lunghezza; il valore di quest'ultimo parametro deve essere un intero compreso tra 0 e 32767.

Stringhe come quelle create con questa istruzione sono utili per organizzare il formato dello schermo e per l'output su stampante. Possono anche essere utilizzate per cancellare linee di testo sullo schermo.

SQR

```
Valore=SQR(Valore)
```

Con questa funzione è possibile calcolare la radice quadrata di un numero; SQR(4) dà 2, SQR(2) è pari a 1.414214.

STR$ *Sezione 3.4*

```
Stringa=STR$(Valore)
```

STR$ converte un numero in una stringa: non produce byte per la memorizzazione, ma una stringa contenente tutte le cifre che compongono il numero (che sono viste come caratteri).

```
STR$(4095)          risultato       "4095"
```

STRING$ *Sezione 1.6*

```
STRING$(Lunghezza,Codice ASCII)
STRING$(Lunghezza,Stringa)
```

La funzione STRING$ produce una stringa di lunghezza specificata contenente il carattere che corrisponde al codice ASCII indicato oppure il primo carattere della stringa specificata.

```
STRING$(5,118)       risultato       "vvvvv"
STRING$(4,"Amiga")      "            "AAAA"
```

TAN

```
Valore=TAN(Valore)
```

TAN è la funzione che permette di calcolare il valore della tangente dell'angolo specificato in radianti. Informazioni per la conversione da gradi a radianti possono essere ricavate esaminando la voce COS.

TIME$ *Sezione 1.17*

```
Stringa=TIME$
```

La funzione TIME$ fornisce come risultato l'ora corrente di sistema. Se non si ha a disposizione un orologio interno, l'ora va regolata tutte le volte che si accende il calcolatore utilizzando le Preferences. TIME$ ritorna una stringa lunga 8 caratteri, con formato OO:MM:SS (Ore, Minuti, Secondi)

TIMER *Sezione 1.17*

```
Valore=TIMER
```

TIMER fornisce il tempo corrente di sistema in secondi: il risultato è il numero di secondi che sono passati dalla mezzanotte (00:00:00). Il valore viene ricavato dall'orologio interno di Amiga; qualora non lo si possedesse, per avere un risultato corretto è necessario regolare l'ora attuale delle Preferences.

UCASE$ *Sezione 1.16*

```
UCASE$(Stringa)
```

L'istruzione UCASE$ converte tutti i caratteri minuscoli compresi nella stringa specificata in caratteri maiuscoli.

```
UCASE$("Amiga")      risultato      'AMIGA"
```

Questo può rivelarsi molto utile per effettuare ordinamenti alfabetici o per controllare l'immissione di dati da parte dell'utente.

VAL *Sezione 1.7*

```
VAL(Stringa)
```

La funzione VAL è l'inverso di STR$: una stringa contenente cifre viene convertita in un numero.

La conversione prosegue fino a quando non si incontra il primo carattere:

```
VAL("1234Ciao")      risultato      1234
VAL("1234Ciao567")       "          1234
VAL("Ciao1234")          "             0
```

VARPTR

```
VARPTR(NomeVariabile)
```

VARPTR svolge una funzione simile a quella che SADD svolge per le stringhe in quanto ritorna la locazione di partenza in memoria della variabile specificata.

Se si utilizzano subroutine in linguaggio macchina all'interno dei programmi AmigaBASIC, si farà ampio ricorso a VARPTR; altre informazioni possono essere trovate sotto la voce CALL.

XOR **Nota d'uso 3**

```
Valore=Valore1 XOR Valore2
```

L'operatore logico XOR produce un risultato operando sui singoli bit di Valore1 e Valore2 nel modo seguente:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

B.6 Comandi per gestire file, memorie di massa e dati di input/output

CHDIR *Sezione 3.2*

```
CHDIR "[Dispositivo o disco:] [Indirizzario
[/Indirizzario...]"
```

Il comando CHDIR permette di cambiare l'indirizzario (directory) corrente. E' possibile andare più a fondo nelle sottodirectory oppure scegliere una directory totalmente nuova. La figura 13 schematizza come è organizzato un sistema di sottodirectory.

Se si immette CHDIR "/", il risultato ottenuto è quello di risalire di un livello nel sistema di sottodirectory (tornando all'indirizzario di livello superiore, o parent directory). Se si è già al livello più alto possibile (indirizzario radice, o root directory), si incorre nel messaggio di errore "File not found". Lo stesso errore si manifesta quando si specifica una directory che non esiste. Per raggiungere direttamente l'indirizzario radice, è sufficiente immettere CHDIR ":".

CLOSE *Sezione 3.3*

```
CLOSE [ [#] NumeroFile] [, [#] NumeroFile]
```

Il comando CLOSE può essere utilizzato per chiudere uno o più file.

Esistono diverse ragioni per cui è buona norma chiudere i file dopo averli aperti per manipolarli.

Innanzitutto, AmigaBASIC utilizza un'area di memoria buffer per ogni file: gli ultimi dati di tale buffer non verranno registrati su memoria di massa fino a che il file in questione non verrà chiuso. In secondo luogo, AmigaDOS registra su memoria di massa alcune informazioni riguardanti lo stato

corrente del file (ad esempio, la sua lunghezza). Infine, il numero di identificazione del file chiuso diviene nuovamente disponibile per l'utilizzo da parte di un altro comando OPEN.

Ulteriori informazioni sui file sono contenute nella descrizione del comando OPEN.

EOF *Sezione 3.3*

```
EOF(NumeroFile)
```

La funzione EOF verifica se vi sono ulteriori dati da leggere in un file. EOF(NumeroFile) fornisce come risultato 0 quando sono ancora presenti dati, -1 se si è giunti alla fine del file.

E' possibile usare questa funzione per accertarsi se si possono leggere o meno altri dati all'interno di un file. Cercare di leggere altri dati dopo essere arrivati al termine del file provoca il messaggio d'errore "Input past end".

FIELD *Sezione 5.1*

```
FIELD [#] NumeroFile,Lunghezza AS Stringa [,Lunghezza
AS Stringa,...]
```

Questo comando viene utilizzato per agire su file ad accesso casuale (random). E' possibile utilizzare FIELD per definire un'area buffer per la registrazione dei dati ed indicare variabili stringa diverse come variabili di trasferimento per l'area buffer. Lunghezza corrisponde al numero di caratteri di un campo. Stringa indica il nome della variabile in oggetto. Se si desidera definire un'area buffer di 10 byte per il nome e di 20 byte per il cognome, è possibile procedere come indicato qui sotto:

```
FIELD #1, 10 AS Nome$, 20 AS Cognome$
```

Solo i comandi LSET e RSET possono essere utilizzati per assegnare nuovi valori a queste variabili, al fine di non perdere le relazioni tra variabile e buffer. La lunghezza totale di tutti i campi non può essere superiore a quella specificata per i record del file (definita con il comando OPEN). In

caso contrario si avrà il messaggio d'errore "Field overflow". Immettendo una nuova linea FIELD per lo stesso file, l'area buffer verrà completamente ricreata.

FILES *Sezione 3.2*

```
FILES [Indirizzario]
```

Il comando Files visualizza il contenuto della directory corrente; è possibile usare l'istruzione CHDIR per cambiare l'indirizzario nel quale ci si trova. Se si specifica una directory particolare dopo FILES, il comando visualizza quanto si trova al suo interno senza però farla divenire directory corrente. Questo risulta utile quando si intende esaminare una directory specifica.

GET (Random file GET) *Sezione 5.1*

```
GET [#] NumeroFile [,NumeroRegistrazione]
```

Il comando GET legge il record specificato da NumeroRegistrazione nell'area buffer del file quando si sta lavorando con file ad accesso casuale. Se non si specifica alcun numero di registrazione, verrà utilizzato il record successivo a quello corrente. NumeroRegistrazione deve essere compreso teoricamente tra 0 e 16777215. In pratica è la capacità di immagazzinamento dati delle memorie di massa (floppy disk, hard disk o RAM disk) che stabilisce il numero di registrazione più alto possibile che, in ogni caso, è ben inferiore a 16777215.

Dopo aver usato il comando GET, è possibile leggere il contenuto dell'area buffer. Si possono utilizzare tanto le variabili di trasferimento definite con il comando FIELD quanto le istruzioni INPUT# o LINE INPUT#.

INPUT$ *Sezione 4.1*

```
Stringa=INPUT$(Lunghezza [, [#] NumeroFile])
```

Questo comando si rivela molto utile quando si utilizzano file sequenziali: esso legge una stringa della lunghezza specificata dal file indicato.

Se si tralascia NumeroFile, INPUT$ acquisisce il numero di caratteri desiderato dalla tastiera. Il cursore per il testo è presente quando si immettono caratteri da tastiera, che non vengono comunque visualizzati sullo schermo mentre vengono inseriti. Non è possibile interrompere la lettura da tastiera tramite <RETURN>; l'unico modo per farlo è premere la combinazione di tasti <CTRL> <C>. Mentre si stanno caricando byte per mezzo di INPUT$, è possibile riconvertirli in numeri utilizzando le funzioni CV*.

INPUT# *Sezione 3.3*

```
INPUT# NumeroFile,Variabile [,Variabile,...]
```

INPUT# opera come il comando INPUT dello schermo, leggendo però variabili o stringhe da un file specificato piutosto che da tastiera. Questo file deve essere stato precedentemente aperto. I dati che INPUT# legge, sono stati in precedenza scritti nel file tramite i comandi PRINT# o WRITE#.

INPUT# considera i dati che legge come appartenenti ad una variabile fino a quando non incontra un carattere di separazione all'interno del file. Questo comando riconosce come separatori spazi, virgole, <RETURN> (CHR$(10)) e Line Feed (CHR$(13)). Gli spazi non sono considerati separatori nel caso di stringhe.

Se si desidera leggere correttamente stringhe che contengono virgole, è necessario scrivere nel file tali stringhe tra doppi apici.

KILL *Sezione 3.2*

```
KILL NomeFile
```

KILL viene usato per cancellare file da dischetti, dischi rigidi o RAM disk. I file che sono aperti nella sessione di lavoro non possono essere cancellati. La versione attuale di AmigaBASIC cancella automaticamente anche i file .info associati ai file cancellati.

LINE INPUT#

```
LINE INPUT# NumeroFile,Variabile
```

Questo comando svolge le stesse funzioni di LINE INPUT, ad eccezione del fatto che legge i caratteri non da tastiera ma dal file specificato e riconosce come separatori solo i <RETURN> (CHR$(10)) ed i Line Feed (CHR$(13)). Tutti gli altri caratteri, compresi spazi e virgole, vengono inclusi nella stringa.

LLIST *Sezione 3.5*

```
LLIST [Etichetta o Numero di linea] [-] [Etichetta o
Numero di linea]
```

LLIST svolge lo stesso lavoro di LIST, ma invia il listato in questione ad una stampante collegata ad Amiga. Nelle Preferences è possibile indicare quale stampante ed interfaccia si sta utilizzando. Vedere LIST per ulteriori informazioni.

LOAD *Sezione 1.13*

```
LOAD [NomeFile] [,R]
```

Il comando LOAD carica in memoria un programma contenuto nella memoria di massa (dischetto, disco rigido o RAM disk). Se si aggiunge l'opzione ,R, tale programma verrà subito mandato in esecuzione dopo il caricamento. Se non si specifica il nome del file da caricare, AmigaBASIC fa comparire una finestra di dialogo che ne richiede l'immissione.

LOC

```
Valore=LOC(NumeroFile)
```

La funzione LOC fornisce come risultato il numero del blocco dati che è stato letto o scritto per ultimo su dischetto, disco rigido o RAM disk. Anche se tale numero dipende solo dalla suddivisione della memoria fatta da AmigaDOS, quando si sta lavorando con file ad accesso casuale, LOC ritorna il numero dell'ultima registrazione letta.

La cosa è più complicata per i file sequenziali: in questo caso LOC restituisce come risultato il numero del blocco letto o scritto in un modo che

dipende dalle dimensioni dell'area buffer. Le dimensioni base per l'area buffer sono di 128 byte, ma le si possono modificare utilizzando il comando OPEN.

LOF *Sezione 4.1*

```
LOF(NumeroFile)
```

La funzione LOF restituisce la lunghezza del file specificato in byte; questa istruzione viene utilizzata per leggere informazioni quando viene riconosciuto tale valore, oppure per leggere un intero file in una sola volta (es. per OBJECT.SHAPE).

LPOS

```
LPOS(x)
```

LPOS svolge una funzione simile a POS, fornendo però come risultato il valore dell'ultimo carattere stampato presente nel buffer di stampa. Lo si può utilizzare per trovare il numero di caratteri della linea corrente già inviati alla stampante. Il valore x qui indicato rappresenta solo un parametro di comodo (vedere anche la voce POS).

LPRINT *Sezione 3.5*

```
LPRINT [Variabile o Valore] [separatore] [Variabile o
Valore...]
```

Questa istruzione svolge compiti simili a PRINT; i dati vengono tuttavia indirizzati non sullo schermo, ma verso una stampante collegata ad Amiga. I caratteri sono immessi in un'area buffer e stampati quando viene incontrato un <RETURN> (CHR$(10)). Se non si mette un separatore dopo l'ultima variabile, AmigaBASIC invia automaticamente un CHR$(10) alla stampante.

Vedere la voce PRINT per ulteriori informazioni.

LPRINT USING

```
LPRINT USING (Stringa di formattazione);(Variabile o
Valore) [;]
```

Rappresenta il corrispettivo per la stampante del comando PRINT USING. Vedere le voci LPRINT e PRINT USING per maggiori informazioni.

LSET *Sezione 5.1*

```
LSET Stringa da trasferire=Stringa
```

Il comando LSET, usato quando si utilizzano file ad accesso casuale, trasferisce i dati all'interno dell'area buffer associata ad un record. Tale comando prende la stringa da trasferire (definita dal comando FIELD) e la assegna ad una stringa trasferita poi nell'area buffer. La L di LSET significa Left-justified (giustificato a sinistra). Se i dati sono più corti della lunghezza stabilita dal comando FIELD, LSET li collocherà all'interno del campo allineandoli a sinistra (aggiungendo eventualmente spazi alla destra del testo). Vedere anche il comando RSET. Per trasferire informazioni numeriche all'area buffer, è necessario utilizzare funzioni di conversione come MK* o STR$. Per muovere i dati dall'area buffer alla memoria di massa è necessario utilizzare il comando PUT.

MERGE *Nota d'uso 2*

```
MERGE NomeFile
```

MERGE viene usato per leggere un programma o sottoprogramma da memoria di massa ed accodarlo al programma corrente. Il programma da leggere deve essere stato registrato in formato ASCII (vedere il comando SAVE).

E' possibile inserire MERGE all'interno di un programma; dopo aver eseguito questa istruzione AmigaBASIC torna però in modo diretto. Il comando CHAIN fornisce risultati migliori per il concatenamento e caricamento dei programmi. MERGE si rivela invece migliore durante lo sviluppo di programmi.

NAME *Sezione 3.2*

```
NAME (Vecchio NomeFile) AS (Nuovo NomeFile)
```

Viene utilizzato per cambiare il nome di un file presente su dischetto, disco rigido o RAM disk, nome nel quale non è possibile utilizzare il nome di un dispositivo, volume o indirizzario che non sia presente nel vecchio nome, per non incorrere nel messaggio d'errore "Rename across disk".

La versione attuale di AmigaBASIC cambia automaticamente anche il nome del file .info.

OPEN *Sezione 3.3*

```
OPEN NomeFile [FOR Modalità] AS [#] NumeroFile
[LEN=Lunghezza]
OPEN Simbolo Modalità, [#] NumeroFile, NomeFile
[,Lunghezza]
```

OPEN viene usato per aprire file; è possibile usare l'una o l'altra delle sintassi sopra esposte, in quanto forniscono gli stessi risultati. Un file può essere aperto per l'immissione di dati (FOR INPUT), per il recupero di dati (FOR OUTPUT) o per l'aggiunta di dati (FOR APPEND). Tutti questi modi agiscono su file sequenziali. Se si tralascia FOR e Modalità nella prima sintassi, AmigaBASIC apre un file ad accesso casuale (random file).

Al termine della linea contenente OPEN è possibile immettere opzionalmente la dimensione in byte dell'area buffer, opzione molto importante soprattutto per i file ad accesso casuale. Il valore che viene immesso è uguale alla somma delle lunghezze dei campi specificati dal comando FIELD. La seconda sintassi sembra un po' differente, ma opera esattamente nello stesso modo; Simbolo Modalità è un carattere che indica il modo di apertura del file: I per immissione dati in file sequenziale, O per recupero dati da file sequenziale, A per aggiunta dati in file sequenziale, R per file ad accesso casuale. I restanti parametri hanno lo stesso significato di quelli già visti nella prima sintassi.

E' possibile aprire file sequenziali per tutti i dispositivi di input/output (schermo -SCRN:-, tastiera -KYBD:-, stampante -PRT:-, porta parallela -PAR:- e seriale -SER:-), come anche leggere dati in arrivo da essi o inviarne

verso gli stessi. Un file sequenziale può essere aperto o solo in scrittura o solo in lettura: non è possibile effettuare contemporaneamente le due operazioni.

OPEN"COM1:"

```
OPEN "COM1: [Baud] [,Parità] [,Lunghezza] [,Bit di
stop]" [FOR Modalità] AS [#] [NumeroFile].
```

E' possibile utilizzare questa variante del comando OPEN per aprire un file che comunichi con un'interfaccia RS-232 (un'interfaccia seriale per il trasferimento dei dati). Se l'utente non intende utilizzare tale interfaccia, le informazioni seguenti non saranno di molto interesse.

Baud indica la velocità di trasferimento dei dati (baudrate), il cui valore può essere pari a 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600 o 19200. Il valore base è 9600. Il valore da impiegare dipende dal tipo di dispositivo con cui si intendono scambiare dati.

Parità stabilisce il modo di controllare i dati inviati o ricevuti. Il mittente e il destinatario devono utilizzare i medesimi dati. Sono possibili tre scelte: E (pari), O (dispari) e N (nessun controllo di parità). La scelta base è E.

Lunghezza viene utilizzata per indicare quanti bit all'interno di un byte (un byte = 8 bit) contengono informazioni. I valori possibili sono 5, 6, 7 e 8; il valore standard è pari a 7.

Bit di stop stabilisce il numero di tali bit. Questo valore deve essere controllato anche per la trasmissione seriale. I valori disponibili sono 1 e 2; solitamente 2 viene impiegato per trasmettere dati a 110 baud, mentre tutte le altre velocità di trasferimento utilizzano 1 bit di stop.

Le modalità tra cui scegliere sono FOR INPUT o FOR OUTPUT; se non si specifica la modalità ci si trova ad operare in un modo di azione in cui è possibile sia mandare che ricevere dati.

PRINT#
PRINT USING# *Sezione 3.3*

```
PRINT# NumeroFile [USING (Stringa di formattazione);]
[Variabile o valore] [Separatore] [Variabile o Valore]
```

Questi due comandi agiscono in modo analogo ai comandi PRINT e PRINT USING, ad eccezione del fatto che essi inviano i dati per la scrittura ad un file, anzichè allo schermo. Altre informazioni possono essere trovate alle voci PRINT, PRINT USING e INPUT#.

PUT (random file PUT) *Sezione 5.1*

```
PUT [#] NumeroFile [,NumeroRegistrazione]
```

Il comando PUT scrive su disco i dati contenuti nell'area buffer di un record appartenente ad un file ad accesso casuale. Solo dopo aver utilizato questo comando i dati vengono a far parte realmente del file in questione. E' possibile specificare o meno il numero del record; nel secondo caso, AmigaBASIC scrive i dati nel record avente il numero successivo a quello corrente. NumeroRegistrazione deve essere compreso tra 0 e 16777215. Ulteriori informazioni sono disponibili sotto la voce GET (random file GET).

RSET

```
RSET Stringa da trasferire=Stringa
```

I comandi RSET e LSET vengono utilizzati per scrivere dati all'interno dell'area buffer di un record quando si stanno usando file ad accesso casuale. Si utilizza una variabile da trasferire, definita tramite un comando FIELD, e la si assegna ad una stringa che verrà quindi immessa nel buffer. La R di RSET significa che la stringa verrà giustificata a destra (Right-justified), vale a dire che, nel caso la stringa in questione fosse più corta dello spazio riservatole all'interno di un campo, il testo verrà spostato verso destra, affinchè il suo termine coincida con il limite destro del campo, aggiungendo l'opportuno numero di spazi alla sinistra della stringa, fino a raggiungere il limite sinistro del campo.

Questo risulta molto utile per valori numerici stampati in un formato standard. Per stabilire le dimensioni dell'area buffer, è necessario usare il

comando FIELD; per spostare i dati dall'area buffer verso la memoria di massa, è necessario impiegare il comando PUT. RSET e LSET possono essere utilizzati anche su stringhe normali per ottenere del testo allineato a destra o a sinistra. Ad esempio:

```
Forme$=SPACES$(20)
RSET Forme$="Amiga"
WRITE Forme$.
```

SAVE *Sezione 3.5*

```
SAVE [(NomeFile)] [,A] [,B] [,P]
```

Con questo comando si possono registrare su memoria di massa (dischetti, disco rigido o RAM disk) i programmi sviluppati. Se non si specifica il nome del file, AmigaBASIC fa comparire una finestra di dialogo che ne richiede l'immissione. Inserendo ,A dopo il nome assegnato, il programma viene registrato in formato ASCII, formato indispensabile se si vogliono fondere due programmi tramite il comando MERGE, oppure se si vuole modificare il listato di un programma tramite un qualunque text editor.

,B dopo il nome del file indica che il programma verrà salvato in forma binaria, vale a dire nel modo base (per cui è possibile omettere ,B); i comandi che costituiscono il programma verranno registrati in forma "tokenizzata". E' anche possibile registrare un programma in forma protetta specificando ,P dopo il nome assegnato al file: in tal modo il programma può essere caricato ed eseguito, ma non può essere listato o modificato.

STICK *Sezione 3.5*

```
STICK(x)
```

Questo comando fornisce informazioni riguardanti i joystick connessi ad Amiga.

STICK(x) fornisce come risultato 1 se il joystick è stato mosso verso l'alto o a destra.

STICK(x) ritorna 0 se il joystick specificato non è stato mosso.

STICK(x) dà come risultato -1 se il joystick in questione è stato mosso in basso o a sinistra.

x specifica le seguenti informazioni:

Valore	Informazioni
0	Movimento orizzontale Joystick 1
1	Movimento verticale Joystick 1
2	Movimento orizzontale Joystick 2
3	Movimento verticale Joystick 2

E' necessario porre attenzione al joystick 1: se al suo posto è inserito un mouse, utilizzando STICK(0) o STICK(1), viene bloccata la registrazione dei suoi movimenti.

STRIG *Sezione 3.5*

```
STRIG(x)
```

STRIG viene utilizzato per verificare se è stato premuto il pulsante di fuoco di un joystick. La x stabilisce il tipo di informazione da ricercare.

STRIG(0) va a verificare se il pulsante di fuoco del joystick 1 è stato premuto dopo l'ultimo comando STRIG(0); se la risposta è positiva, il risultato ritornato è -1, altrimenti si ottiene 0.

STRIG(1) verifica se il pulsante di fuoco del joystick 1 è attualmente premuto: il risultato è -1 per la risposta positiva, 0 per quella negativa.

STRIG(2) controlla se il pulsante di fuoco del joystick 2 è stato premuto successivamente all'ultimo comando STRIG(2), fornendo -1 in caso positivo e 0 in caso negativo.

STRIG(3) verifica se il pulsante di fuoco del joystick 2 è attualmente premuto ritornando -1 in caso affermativo, 0 in caso negativo.

Altre informazioni possono essere trovate sotto la voce STICK.

WRITE# ***Sezione 3.4***

```
WRITE# NumeroFile, [Variabile o Valore] [Separatore]
[Variabile o Valore...]
```

Questo comando assolve le stesse funzioni di WRITE, scrivendo però i dati non su schermo ma in un file.

Questo comando si rivela molto utile soprattutto per scrivere sequenze di caratteri in file sequenziali, dato che esse devono essere inviate al file all'interno delle virgolette. Quando si recuperano tali dati, si ottiene l'intera stringa, compresi i caratteri separatori. Non è possibile inserire virgolette all'interno della stringa da scrivere.

B.7 Comandi per la sintesi vocale e sonora

SAY *Sezione 6.2*

```
SAY Stringa di fonemi [,Matrice%(0), Matrice%(1),
Matrice%(2), Matrice%(3), Matrice%(4), Matrice%(5),
Matrice%(6), Matrice%(7), Matrice%(8)]
```

Il comando SAY fa si che Amiga "reciti" una stringa scritta utilizzando codici fonetici. Per tradurre un normale testo in una sequenza di codici fonetici è necessario utilizzare il comando TRANSLATE$. E' anche possibile immettere direttamente i codici fonetici che corrispondono ad una determinata sequenza di caratteri; l'Appendice H del manuale BASIC fornito con Amiga spiega come eseguire tale operazione. Dopo la stringa di fonemi è possibile inserire il nome di una matrice costituita da numeri interi, che abbia almeno nove elementi. Tali valori influiscono sul modo con cui Amiga produce la sintesi vocale.

Il primo elemento, Matrice%(0), rappresenta il tono: è così possibile indicare quanto acuta deve essere la voce. I valori devono essere compresi tra 65 (basso) e 320 (acuto). Il valore base, cioè quello che Amiga impiega se non si specifica una matrice di valori dopo la stringa fonetica, è 110.

Il secondo elemento, Matrice%(1), indica se la voce deve avere inflessioni (0) o essere monotona (1), come quella sintetizzata da componenti elettronici; il valore base è pari a 0.

Il terzo elemento, Matrice%(2), imposta la velocità della sintesi vocale, espressa in parole al minuto. I valori possono variare tra 40 (lento) e 400 (veloce); quello di base è 150.

Il quarto elemento, Matrice%(3), determina se la sintesi vocale deve possedere un timbro maschile (0) o femminile (1).

Il quinto elemento, Matrice%(4), viene usato per stabilire la frequenza di

campionamento, fattore che più di ogni altro influenza la profondità della voce. I valori possono variare da 5000 (profondo) a 28000 (acuto), con quello base pari a 22200. Valori troppo vicini alle due estremità possono portare ad una incomprensione della voce sintetizzata.

Il sesto elemento, Matrice%(5), indica il volume della voce. Il valore base è 65, mentre i limiti sono 0 (silenzio) e 65.

Il settimo elemento, Matrice%(6), determina il canale audio che verrà utilizzato per la sintesi vocale (vedere la Tabella 10 nella Sezione 6.3). Il valore base è 10 (corrispondente ad una coppia qualunque di canali destro e sinistro liberi).

Il penultimo elemento, Matrice%(7), stabilisce se AmigaBASIC debba interrompere il programma quando il calcolatore sta parlando (0) o se la sintesi vocale debba essere effettuata da un processo accessorio mentre il programma continua la sua esecuzione (1).

Il nono ed ultimo elemento, Matrice%(8), specifica cosa dovrebbe accadere se due comandi SAY si susseguono (a condizione che Matrice%(7) sia pari a 1). 0 indica che il primo SAY deve essere completato prima che venga eseguito il secondo; 2 stabilisce invece che il secondo SAY venga eseguito immediatamente, mentre viene bloccata la prima sintesi vocale; infine il valore 1 significa che il corrente comando SAY non deve essere eseguito, indipendentemente dal valore di Matrice%(7). Fino a quando questo parametro rimane pari a 1, il comando SAY non viene eseguito.

SOUND *Sezione 7.2*

```
SOUND Frequenza, Durata [,Volume] [,Voce]
```

Questo comando produce un suono. Frequenza, che deve essere espressa in Hertz(Hz), può assumere valori compresi tra 20 e 15000 Hz. La Tabella 11 nella Sezione 7.2 mostra quali frequenze corrispondono a particolari note musicali. La durata di un suono può avere un valore compreso tra 0 e 77; se si desidera che un suono duri un secondo circa, è necessario immettere un valore pari a 18. Vedere la tabella 13 nella Sezione 7.2 per ulteriori dettagli. Volume può avere un valore tra 0 e 255; se non si specifica altrimenti, AmigaBASIC utilizza il valore base 127. E' infine possibile controllare il canale audio che emetterà il suono specificato per il

quale si hanno a disposizione quattro valori:

0 e 3 = canale sinistro

1 e 2 = canale destro

Il valore base del canale audio da utilizzare è 0 tanto per il comando SOUND quanto per il comando BEEP.

SOUND RESUME
SOUND WAIT *Sezione 7.3*

```
SOUND RESUME
SOUND WAIT
```

Per far sì che diversi suoni vengano prodotti contemporaneamente, oppure che un suono venga eseguito in sottofondo, è necessario utilizzare un ciclo di attesa.

Il comando SOUND WAIT produce come risultato quello di immagazzinare i comandi SOUND che non vengono quindi eseguiti direttamente.

Per brani musicali multivocali, è necessario specificare i canali audio non utilizzati; la migliore cosa da fare è usare un suono con livello volume 0.

SOUND RESUME permette di terminare il ciclo di attesa e di iniziare la sintesi sonora.

TRANSLATE$ *Sezione 6.2*

```
Stringa di fonemi=TRANSLATE$(Stringa)
```

TRANSLATE$ traduce la stringa specificata in una stringa fonetica, che verrà in seguito utilizzata da SAY per la sintesi vocale. Si possono trovare altre informazioni sotto la voce SAY.

WAVE ***Sezione 7.5***

```
WAVE NumeroCanale, Matrice di interi%
WAVE NumeroCanale, SIN
```

E' possibile impiegare WAVE per modificare la forma d'onda che verrà utilizzata dal comando SOUND. SIN permette di ottenere un'onda sinusoidale, corrispondendo tra l'altro al valore base dell'onda utilizzato quando non si specifica alcun comando WAVE.

E' anche possibile utilizzare una matrice di numeri interi, che contenga almeno 256 elementi; tale matrice contiene una forma d'onda definita dall'utente. Gli elementi possono avere un valore compreso tra -128 e +127: essi forniscono l'ampiezza dell'onda al tempo di campionamento. Se non si hanno le idee chiare, è consigliabile leggere la Sezione 7.4 (Un po' di teoria acustica) e quella 7.5 (Forme d'onda).

La forma d'onda verrà assegnata ad uno o più dei 4 canali audio (0 e 3 = canale sinistro; 1 e 2 = canale destro); dopo tale assegnamento, si dovrebbe cancellare la matrice in questione (tramite il comando ERASE) per recuperare preziosa memoria.

C Listati senza errori

Nel testo originale questa sezione delle Appendici era stata inclusa per contenere i listati dei programmi privi di errore, al fine di fornire una sezione di riferimento cui il lettore potesse accedere per controllare l'effettiva correttezza dei programmi descritti nel libro.

Nella versione italiana si è deciso, invece, di allegare al volume un dischetto, al quale si rimanda per il controllo dei listati, contenente tutti i programmi BASIC inseriti e spiegati nel testo.

D I programmi contenuti nel cassetto BASICDemos

Sul dischetto EXTRAS fornito dalla Commodore assieme all'Amiga sono contenuti, all'interno del cassetto BASICDemos, diversi programmi dimostrativi che possono essere utilizzati con AmigaBASIC.

LIST-ME descrive i programmi presenti e come essi lavorano. Le informazioni contenute sono però incomplete, ed inoltre l'utente non deve essere costretto a caricare tale programma ogni volta che desidera imparare ad utilizzare un altro programma. Lo scopo di questa Appendice è di facilitare le cose, descrivendo brevemente tali programmi dimostrativi.

LIST-ME
Come ricordato sopra, questo file costituisce un'introduzione agli altri programmi; per esaminarlo è sufficiente seguire il suggerimento fornito dal suo nome e visualizzarne il listato. Il programma è composto da una serie di linee REM: tutte le linee che lo compongono iniziano infatti con l'apostrofo ('). Il testo può pertanto essere letto solo all'interno della finestra LIST; il programma è organizzato in modo tale da visualizzare 80 caratteri per linea, di modo che lo si possa leggere completamente all'interno della finestra LIST, dopo averla ovviamente allargata a pieno schermo. E' anche possibile stamparne una copia, per avere sempre sottomano il suo contenuto. Questa appendice dovrebbe comunque fornire sufficienti informazioni sui programmi dimostrativi.

Music
Tramite questo programma, AmigaBASIC sintetizza musica. Una composizione a tre voci è stata convertita in una serie di istruzioni DATA ed è stata anche inclusa una sezione grafica dedicata al tracciamento di linee che accompagnano la sintesi musicale.

La sezione PlaySong: del programma ha il compito di leggere i valori contenuti nelle linee DATA e di calcolarne le frequenze; le istruzioni DATA contengono le note scritte sotto forma di lettere (c, d, e, f, g, ecc.) ed informazioni sulla durata delle note, cambi di ottava ed altro.

Per calcolare la forma d'onda è stato utilizzato un piccolo trucco: invece di calcolare direttamente i valori dell'onda sinusoidale desiderata, chi ha realizzato il programma ha inserito i valori calcolati all'interno delle linee DATA, in quanto l'operazione di lettura è più veloce del calcolo. La formula utilizzata per il calcolo dei valori è riportata davanti alle linee DATA nel listato.

Per il disegno delle linee viene adottata l'intercettazione degli eventi collegati al passare del tempo: ogni due secondi l'istruzione ON TIMER richiama la subroutine TimeSlice:. Le dimensioni del disegno grafico son automaticamente aggiustate alle dimensioni correnti della finestra.

Library
In questo programma è possibile trovare spiegazioni sull'utilizzo della funzione LIBRARY per ottenere, ad esempio, differenti stili di caratteri. A tale scopo vengono utilizzate le librerie graphics.bmap e dos.bmap. Per individuare l'esatta procedura da seguire, è sufficiente esaminare l'Appendice B alle voci DECLARE FUNCTION ... LIBRARY e LIBRARY.

Il sottoprogramma DosLibDemo: mostra come richiamare da AmigaBASIC le funzioni AmigaDOS. Per poter impiegare tale sezione, è necessario far partire AmigaBASIC da CLI, e non dal Workbench.

BitPlanes
ScreenPrint
Utilizzare le librerie nella fase di programmazione rende possibile raggiungere risultati altrimenti non avvicinabili con l'uso del solo AmigaBASIC. Questi due programmi sono un buon esempio di quanto affermato. Il listato di BitPlanes dovrebbe essere familiare dopo aver letto la Sezione 4.3; è possibile conoscere la locazione di memoria di partenza riguardante schermi (screen) e finestre (window), o i colori correntemente utilizzati.

ScreenPrint è una routine per la stampa di quanto appare sullo schermo

che non è stata comunque molto semplice realizzare sotto il controllo delle librerie di AmigaBASIC; questo programma è stato appositamente realizzato per chi intende studiare a fondo l'utilizzo delle librerie nell'ambito della programmazione con AmigaBASIC. Il programma BitPlanes può essere interrotto premendo un pulsante del mouse, mentre ScreenPrint si ferma non appena si preme il tasto <Q>.

Screen
Tramite l'istruzione AREA vengono disegnati rettangoli colorati su uno schermo di 320 x 200 pixel utilizzando 32 colori (5 bitplane). Il programma utilizza formule contenenti SIN e COS per calcolare la posizione degli angoli; l'istruzione PATTERN viene utilizzata per includere qualche semplice modello di riempimento. Se l'utente allarga o riduce le dimensioni della finestra di output del programma, vengono modificate anche le dimensioni del rettangolo. I colori vengono scelti in modo casuale (random).

Demo
Questo programma impiega quattro finestre e mostra come il BASIC può gestire diversi processi contemporaneamente (multitasking). La prima finestra mostra due sfere che rimbalzano da una parte all'altra; la seconda contiene linee che si muovono, simili a quelle che compaiono nel programma Music; la terza viene impiegata per visualizzare i rettangoli già visti nel programma Screen, mentre la quarta contiene alcuni cerchi dai colori brillanti.

E' possibile ingrandire la prima finestra consentendo alle sfere di utilizzare maggior spazio per i propri movimenti. Le restanti finestre possono essere chiuse con il gadget di chiusura, nel qual caso i programmi rimanenti mostreranno una velocità di esecuzione maggiore. La finestra che contiene le sfere (identificata dal nome Animation), risulta essere una finestra BASIC in formato ridotto. I sottoprogrammi NextLine:, NextPoly: e NextCircle:, richiamati uno dopo l'altro, all'interno di un ciclo WHILE ... WEND, si prendono cura di quanto accade nelle finestre 2, 3 e 4, mentre l'animazione di oggetti nella finestra 1 è sotto il controllo dell'intercettazione degli eventi riguardanti le collisioni tra sfere oppure tra una sfera ed un bordo. Il programma Demo recupera le informazioni riguardanti le sfere dal file Ball.

Picture
Picture2
Queste sono due copie dello stesso programma. Il manuale di AmigaBASIC fornito dalla Commodore utilizza questo programma per aiutare l'utente a comprendere il funzionamento di AmigaBASIC. Vengono utilizzate le istruzioni che riguardano il controllo del mouse ed i comandi PUT e GET: tramite il mouse è possibile disegnare su schermo un cerchio. Dato che il manuale appena citato indica di modificare il programma, sono presenti sul dischetto due versioni dello stesso per poter far fronte ad ogni eventualità.

ObjEdit
Questo programma viene utilizzato per realizzare oggetti grafici per le animazioni; la Sezione 1.11 spiega come impiegarlo.

Un avvertimento: se si possiedono 512 k di RAM e lo sfarfallio degli oggetti creati non crea particolari problemi, è possibile modificare il numero dei colori utilizzabili nel listato del programma. All'inizio del file, dopo alcuni apostrofi (REM), si trova un file già realizzato riguardante un oggetto. In seguito, vi è un'istruzione DEF FN e un comando DIM.

Nella successiva sezione del programma si trovano, al termine delle linee, alcuni comandi che non vengono eseguiti perchè preceduti da apostrofi ('). Togliendo tali apostrofi, è possibile assegnare alla variabile Depth il numero dei bitplane, il cui valore può essere solo 3 o 4, dato che il programma lavora solo su uno schermo di 640 x 200 pixel. E' importante ricordare che i file che descrivono gli oggetti non comprendono le informazioni riguardanti i colori: è necessario avere un programma che permetta di aggiungere tale informazione agli oggetti.

Ogni bitplane aggiunto aumenta lo sfarfallio presente nella animazione dei bob, almeno per quanto riguarda la versione attuale di AmigaBASIC. Per comprendere a fondo il programma ObjEdit è necessario possedere una notevole esperienza di AmigaBASIC: avendo tempo e voglia, sarebbe utile esaminare il modo con cui è stato realizzato l'editor di oggetti. Il programma impiega l'intercettazione degli eventi allo stesso modo del nostro programma di grafica.

Terminal
Questo programma mostra come impiegare l'interfaccia seriale. Se a tale interfaccia è connesso un dispositivo esterno con una velocità di trasferimento dati di 9600 baud, è possibile inviare e ricevere caratteri. Per ulteriori informazioni su come aprire un canale seriale, vedere l'Appendice B alla voce OPEN "COM1:".

ConvertFd
Questo è un programma molto utile per chi intende lavorare spesso con le routine in linguaggio macchina richiamate tramite l'istruzione LIBRARY. Affinchè AmigaBASIC possa impiegare tali routine, è necessario che esse vengano registrate su dischetto in uno speciale formato (.BMAP). Il cassetto BasicDemos contiene quattro librerie di questo tipo, exec.bmap, diskfont.bmap, dos.bmap e graphics.bmap, utilizzate da programmi come Library, lib2 ed altri. Se si hanno a disposizione altre librerie che terminano con il suffisso .FD, è possibile utilizzare ConvertFd per ottenere un file di tipo .BMAP. I file .FD si trovano nel cassetto FD1.2 presente sul dischetto Extras. Non si dovrebbe utilizzare questo programma fin quando non si possiede una sufficiente esperienza di lavoro con le librerie.

Speech
Questo programma agisce in modo simile all'analogo programma presentato in questo libro. Per prima cosa AmigaBASIC richiede il dischetto del Workbench: se si dispone di un solo drive, è necessario estrarre quello presente ed inserire quello richiesto. In seguito è possibile immettere una riga di testo che verrà "letta" dopo la pressione del tasto <RETURN>.

Sei parametri influenzano il tipo di voce: Pitch rappresenta la profondità della voce; Inflection, che può assumere due soli valori, determina se la voce sintetizzata avrà una inflessione umana o sarà monocorde; Rate influenza la velocità di pronuncia; Voice decide se la voce sarà maschile o femminile; Tune regola la frequenza di campionamento che influenza la profondità della voce; Volume indica quanto alto sarà il volume.

Il programma ha solo lo scopo di presentare le possibilità offerte dalla sintesi vocale; non permette di registrare i valori ritenuti ottimali.

LoadACBM
LoadILBM-SaveACBM
SaveILBM

Per rimediare alla mancanza di comandi che rendano possibile da AmigaBASIC il recupero e la registrazione di file grafici IFF, la Commodore ha inserito questi programmi di utilità nel dischetto Extras. Questi tre programmi svolgono principalmente il ruolo di librerie: essi sono infatti incapaci di funzionare se impiegati da soli. Nella Sezione 4.2 si è già discusso il formato ILBM, o InterLeaved BitMap; questo è il modo in cui vengono registrate immagini IFF: la figura è salvata una linea per volta, registrando tutti i bitplane della prima linea, poi quelli della seconda e così via. Tale sistema presenta due vantaggi: innanzitutto, l'utente vede i colori corretti dell'immagine dal momento in cui inizia il recupero della stessa; in secondo luogo, il formato della memoria viene facilmente gestito quando si passa da un grado di risoluzione all'altro. Per tale ragione, tutti i programmi IFF compatibili a nostra conoscenza registrano i dati in formato ILBM.

Il formato ILBM presenta però anche uno svantaggio: alcuni calcoli devono aver luogo durante il recupero dei dati rallentando quindi l'intera procedura, soprattutto in BASIC. Per rendersene conto, è sufficiente esaminare le routine presentate in questo libro per il recupero di Immagini IFF: esse sono molto lente, se confrontate con quelle presenti nei pacchetti grafici più noti.

Per tale ragione la Commodore ha inventato un nuovo tipo chunk per i suoi programmi di utilità destinati a maneggiare le informazioni bitmap di nome ACBM, ovvero Amiga Contiguous BitMap. Questo formato salva i singoli bitplane uno dopo l'altro, e nello stesso modo li richiama in memoria. I dati sono letti molto rapidamente e il processo non implica alcun calcolo. Il fattore velocità ha però un prezzo: le immagini in formato ACBM possono essere registrate o recuperate solo nel grado di risoluzione in cui sono state create originalmente. E' necessario valutare bene se i vantaggi del formato ILBM sopravanzano gli svantaggi di quello ACBM.

Vi è ancora un ultimo problema: che aiuto può fornire la velocità quando non si vuole "leggere" un normale file IFF? Tutti i programmi di grafica, da GraphiCraft a DeluxePaint ed emuli, registrano e recuperano i dati solo in formato ILBM.

I tre programmi forniti dalla Commodore eseguono questi compiti: LoadILBM-SaveACBM legge file ILBM, esegue gli opportuni calcoli e quindi

registra tali file in formato ACBM, in modo che possano venir letti tramite il programma LoadACBM. SaveILBM completa il trio creando una immagine grafica a linea mobile e salvandola in formato ILBM. Un importante vantaggio offerto da queste tre routine è rappresentato dal fatto che esse mettono in grado di realizzare gli effetti di animazione che si basano sulla rotazione dei colori. I dati necessari si trovano in un chunk all'interno del file IFF, nella sezione chiamata CCRT.

NOTA:

Se si desidera utilizzare queste tre routine all'interno dei propri programmi, è necessario assicurarsi che il file ILBM venga indicato alla fine della stringa.

E Un breve glossario tecnico

Le parole riportate in corsivo nel libro sono state impiegate nel contesto senza essere spiegate in dettaglio. L'utente può trovare in questa appendice informazioni riguardanti il loro significato.

AmigaDOS
DOS è l'abbreviazione di Disk Operating System (Sistema Operativo del Disco). Questo è il programma (o meglio, l'insieme di routine) che si occupa del trasferimento dati tra Amiga e le memorie di massa (dischetti, disco rigido ecc.); esso è inoltre responsabile dell'organizzazione dei dati sulle memorie di massa. L'utente può impartire comandi AmigaDOS utilizzando la finestra del CLI, l'interfaccia a linee di comandi contenuto nel dischetto Workbench.

ASCII
Abbreviazione di American Standard Code for Information Interchange. E' un codice standard che assegna un particolare valore ad ogni carattere. Nella Appendice B.5 (Funzioni del BASIC per il calcolo), alla voce CHR$, si trova la tabella dei valori ASCII utilizzata da Amiga.

BASIC
Abbreviazione di Beginners All-purpose Symbolic Instruction Code. Non è comunque una descrizione del tutto accurata del BASIC, dato che esso non è un linguaggio adatto solamente a chi si avvicina per la prima volta alla programmazione, nè un manipolatore particolamente potente di espressioni simboliche. Comunque il BASIC è stato sviluppato quando la maggior parte dei programmatori utilizzava linguaggi particolarmente complicati come l'Assembly, il COBOL e il FORTRAN, e ciò spiega il motivo del suo nome.

Bit
Abbreviazione di Binary Digit (cifra binaria). Un bit è l'unità di informazione più piccola; può essere "acceso" o "spento", "vero" o "falso", 0 o 1. Un insieme di otto bit forma un byte.

Bitplane

Ciascun punto di uno schermo monocromatico corrisponde ad un bit quando l'immagine viene registrata. Per rendere possibile l'uso di più colori, si impiegano un certo numero di bit per ogni punto dello schermo (pixel). Le immagini sono costituite da più bitplane. I bit che giacciono uno sotto l'altro determinano il colore di un particolare punto dello schermo. Per una illustrazione dei bitplane, rifarsi alla Figura 5 nella Sezione 2.2 sulla risoluzione.

Byte
Ogni locazione di memoria di Amiga corrisponde ad un byte. Un byte è formato da otto bit; questa suddivisione è una eredità dei microprocessori ad 8 bit. Un byte può immagazzinare un valore compreso tra 0 (binario 00000000) e 255 (binario 11111111). Solitamente si impiega un byte per immagazzinare un carattere o una cifra.

Poichè ogni Kilobyte (k) contiene 2^10 bytes, un k non contiene 1000, ma 1024 byte; 256 k corrispondono a 262144 byte mentre 512 k contengono 524288 byte.

Compatibilità
Se due dispositivi sono compatibili per quanto riguarda il software, essi possono utilizzare gli stessi programmi. Il modello Amiga 1000 può essere reso parzialmente compatibile ad un PC IBM tramite un programma particolare (Transformer) o per mezzo di un dispositivo hardware (Sidecar); Amiga 2000 possiede degli slot di espansione in cui può essere inserito un emulatore hardware del PC IBM (fra breve anche di un AT). Se due programmi sono compatibili per quanto riguarda i dati, essi possono utilizzare gli stessi dati: è quanto capita nel mondo Amiga per ciò che riguarda lo standard IFF.

Convertitore D/A
D/A è l'abbreviazione per Digitale/Analogico. E' un dispositivo che converte (traduce) un segnale digitale (cioè una lista di valori numerici) in una corrente elettrica analogica: più grande è il numero, più forte è il voltaggio. Dispositivi di questo genere vengono utilizzati da Amiga per produrre suoni e musica. Il coprocessore che sintetizza suoni, chiamato Paula, contiene al suo interno un convertitore digitale/analogico.

Per digitalizzare un suono (cioè convertirlo in una serie di valori numerici) è necessario avere a disposizione un dispositivo che agisce in senso contrario, cioè un convertitore analogico/digitale (A/D).

Cursore
E' un piccolo segnalino che indica all'utente il punto sullo schermo in cui sta scrivendo o lavorando. Vi sono diversi tipi di cursore utilizzati da Amiga: il puntatore, una piccola freccia che si muove quando l'utente sposta il mouse; il cursore a sbarra che si può trovare nelle finestre LIST e BASIC di AmigaBASIC: i dati immessi da tastiera vengono visualizzati su schermo nella locazione del cursore. Altri programmi impiegano cursori di altre fogge.

Dispositivi periferici
Conosciuti anche con il nome di "periferiche", sono dispositivi che possono essere collegati ad Amiga. Vengono quindi classificate come periferiche stampanti, drive esterni per dischetti, dischi rigidi, tavolette grafiche, joystick, scanner ecc; tutte le periferiche sono considerate hardware.

Editor
Un editor è la parte di un programma o di un sistema operativo che si occupa del controllo del cursore prendendosi cura dell'inserimento, della cancellazione e della correzione del testo che compare su schermo. Vedere anche le descrizioni di Editor di schermo ed Editor di linea. In poche parole, il programma permette all'utente di effettuare variazioni, sia esso un editor per il testo, per le icone o per gli oggetti.

Editor di linea
E' un editor che permette di lavorare solo su una singola linea di testo alla volta; non è possibile quindi sfruttare tutto lo schermo come nel caso di un editor di schermo. Il cursore può essere spostato a destra o a sinistra sulla linea, ma non sulle righe superiori ed inferiori. AmigaBASIC dispone di un editor di linea solo all'interno della finestra BASIC.

Editor di schermo o a tutto schermo
Un editor permette di controllare il cursore; un editor di schermo consente all'utente di spostare il cursore in ogni punto dello schermo per immettere o correggere dati. La finestra LIST di AmigaBASIC dispone di un editor di schermo. Questo tipo di editor è solitamente più utile di un editor di linea.

Hardcopy
Con questo termine si indica l'operazione di stampa su carta del contenuto dello schermo. AmigaBASIC ha difficoltà a portare a termine questo compito, per cui è necessario ricorrere ad un programma di utilità.

Hardware
Con hardware si indica l'insieme dei dispositivi e dei componenti di un calcolatore (monitor, stampante, tastiera ecc.). Hardware identifica quindi gli oggetti che si possono effettivamente toccare con mano (vedi Software).

IFF
Abbreviazione di Interchange File Format; è un formato per i file dati di Amiga sviluppato dalla Electronic Arts. Ulteriori informazioni sono raccolte nella Sezione 4.2 (Interchange File Format).

Indirizzario (directory)
Questo termine si riferisce ad un indirizzario o directory presente su dischetto o disco rigido. Fornisce informazioni sui file disponibili su quel particolare supporto di memoria di massa.

Interfaccia
E' una connessione che si utilizza per collegare Amiga a dispositivi esterni oppure a strumenti per il trasferimento dei dati. Amiga dispone di diverse interfacce: una parallela, una seriale, una per drive esterni ed alcune per il collegamento di monitor.

Interprete
E' ciò che rappresenta AmigaBASIC: Amiga non comprende infatti quanto viene scritto direttamente in un programma BASIC, dato che il calcolatore opera solo in termini di "acceso" "spento". L'interprete BASIC traduce in linguaggio macchina i comandi BASIC, in modo che Amiga e i suoi microprocessori possano interpretarli ed eseguirli.

Intuition
Costituisce una parte del sistema operativo di Amiga; Intuition è il responsabile dell'interfaccia iconica e a finestre e si prende cura dei compiti a ciò connessi. Molte delle routine che fanno parte di Intuition possono essere utilizzate come subroutine all'interno di altri programmi.

Joystick
Un dispositivo che permette di registrare la direzione impressa dall'utente ad una leva (usato principalmente nei giochi). Ad Amiga possono essere collegati due di questi dispositivi.

Kernel
E' la parte del sistema operativo sulla quale si basano tutte le altre. Si occupa del movimento di dati in input/output dal sistema e di compiti similari. Viene caricato da dischetto (Amiga 1000) o dalla ROM a ciò destinata (Amiga 500, 2000); non può essere modificato quando è in memoria.

Linguaggio macchina
Questo è il linguaggio che il microprocessore 68000 di Amiga comprende in modo diretto. Esso consiste di 0 ed 1, ma può essere reso più comprensibile per chi programma utilizzando codici mnemonici come MOVEA, MOVEC, CMPI, e SBCD.

Microprocessore
E' il "cervello" del calcolatore che controlla le più importanti funzioni dell'elaborazione. Un microprocessore può eseguire assai rapidamente comandi in linguaggio macchina. Amiga dispone di un processore Motorola 68000 che può eseguire fino ad 8 milioni di questi comandi al secondo. Esso può concentrarsi sulle cose realmente importanti dato che molti compiti di routine, come scambio di dati, visualizzazione degli stessi su monitor e così via, sono effettuati dai coprocessori Agnus, Denise e Paula.

Modo interlacciato
E' un modo di visualizzazione sullo schermo che permette di tracciare una linea di scansione nello spazio compreso tra due altre linee. Ciò permette di raddoppiare le linee di scansione del monitor, ottenendo una risoluzione video più elevata. Per ulteriori informazioni, vedere la Sezione 2.2 sulla risoluzione.

Multitasking
Con questo termine si indica la possibilità di far eseguire al calcolatore diversi programmi allo stesso tempo in modo che essi siano indipendenti l'uno dall'altro. Mentre l'utente interagisce con uno di essi, gli altri continuano la loro esecuzione sullo sfondo (background). In tal modo è possibile risparmiare tempo e avere dati in uscita da diverse fonti nello stesso tempo. Dato che il microprocessore deve dividere il suo tempo tra compiti diversi, ciascun programma verrà eseguito più lentamente di quanto lo sarebbe se venisse eseguito da solo.

Near letter quality (NLQ)
Le stampanti ad aghi forniscono spesso risultati di stampa difficili da leggere e di brutto aspetto. Tramite alcuni accorgimenti è possibile fare in modo che la stampa dia risultati simili a quelli ottenibili con stampanti a margherita o con macchine da scrivere.

Pixel
Un pixel (Picture Element) corrisponde ad un singolo punto dello schermo. Amiga utilizza i bitplane per rappresentare in memoria un pixel sfruttando da uno a cinque bit. Una immagine realizzata dal calcolatore è

composta da pixel: tanto maggiori sono i pixel presenti sullo schermo, tanto più elevata risulterà la risoluzione, e tanto più realistica apparirà l'immagine. In questo caso sono anche molto importanti i colori; ulteriori informazioni sono riportate nella Sezione 2.2 sulla risoluzione.

RAM
Abbreviazione per Random Access Memory (memoria ad accesso casuale). Si possono scrivere dati in questo tipo di memoria e successivamente recuperarli; l'unico problema consiste nel fatto che quando si spegne il calcolatore il contenuto di questa memoria viene irrimediabilmente perso. E' necessario pertanto registrare i dati che si trovano in RAM su memoria di massa (dischetti o disco rigido) prima di procedere allo spegnimento del calcolatore.

ROM
Abbreviazione per Read Only Memory (memoria a sola lettura). In contrasto con la RAM, la ROM non perde le informazioni che contiene quando il calcolatore viene spento. L'utente non può inserire propri dati nella ROM ed è difficile sbarazzarsi di quanto è contenuto in essa.

Questa caratteristica la rende adatta per contenere il sistema operativo e cose simili. Amiga 500 e 2000 hanno la maggior parte del sistema operativo immagazzinato in ROM.

Amiga 1000 dispone invece di poca ROM; quella presente contiene il programma che permette di caricare i dati contenuti nel dischetto del Kickstart e che mostra la "mano che tiene un dischetto". La parte di RAM in cui vengono posizionati tali dati viene bloccata elettronicamente e si comporta quasi come una ROM; quando però il calcolatore viene spento, essa perde le informazioni che conteneva.

Schermo
Quanto compare sul monitor viene detto schermo. Ulteriori informazioni sono alla Sezione 2.3 sugli schermi.

Scrolling
Questo termine indica che i dati possono scomparire oltre uno dei bordi dello schermo in modo tale che l'utente possa visualizzare quanto era in precedenza nascosto. In tal modo si possono, ad esempio, osservare le varie sezioni che compongono un'immagine troppo grande per essere visualizzata per intero sullo schermo. L'esempio è quello di una antica pergamena: quando si avvolge il margine superiore, si svolge quello inferiore e ciò dà modo di leggere quanto prima non era visibile.

Sistema numerico binario
E' il sistema numerico conosciuto anche come sistema in base due. I numeri hanno una cifra per ogni potenza di due; le cifre utilizzate sono 0 e 1. I numeri decimali tra 1 ed 11 hanno i seguenti valori nel sistema binario:

Decimale	Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110

Sistema numerico decimale

Ne fanno parte i normali numeri in base 10 rappresentati con le cifre 0, 1, 2, 3, 4, 5, 6, 7, 8, e 9. E' il sistema che viene considerato più naturale, ma i calcolatori non riescono a gestirlo nel modo migliore come invece accade per gli esseri umani. Ulteriori informazioni sono contenute nella Nota d'uso 3.

Sistema numerico esadecimale

E' un sistema numerico molto utilizzato nei calcolatori. Viene anche chiamato sistema in base 16. Comprende le cifre da 0 a 9 e le lettere A, B, C, D, E ed F. Per identificare tali numeri si premette il simbolo del dollaro ($) alla loro sinistra. I primi 36 numeri esadecimali sono:

Decimale	Esadecimale	Decimale	Esadecimale
0	$0	18	$12
1	$1	19	$13
2	$2	20	$14
3	$3	21	$15
4	$4	22	$16
5	$5	23	$17
6	$6	24	$18
7	$7	25	$19
8	$8	26	$1A
9	$9	27	$1B
10	$A	28	$1C
11	$B	29	$1D
12	$C	30	$1E
13	$D	31	$1F
14	$E	32	$20
15	$F	33	$21
16	$10	34	$22
17	$11	35	$23

Sistema operativo
Il sistema operativo è un programma, o meglio un insieme di routine, che viene sempre eseguito dal calcolatore. Esso si occupa delle funzioni di base come l'immissione dei dati da tastiera, la visualizzazione su monitor, la comunicazione tramite le interfacce, l'esecuzione di altri compiti. Il sistema operativo di Amiga è composto da diverse parti: vedere le voci Intuition, Kernel ed AmigaDOS.

Software
Con questo termine si indicano il programma ed i dati contenuti nella memoria del calcolatore. Non è possibile toccare effettivamente con mano tali cose, anche se è possibile toccare il dischetto in cui esse sono contenute. L'hardware può essere usato correttamente solo quando si dispone del software adatto.

Stampante
Dispositivo periferico per trasferire su carta dati, testi ed immagini. Con Amiga si possono utilizzare diversi tipi di stampanti; le relazioni tra Amiga e stampante vengono regolate mediante il programma Preferences contenuto nel dischetto Workbench. La maggior parte delle stampanti si connettono ad Amiga tramite la porta parallela, ma alcune utilizzano l'interfaccia seriale (vedere la Sezione 3.5 per maggiori informazioni). Esistono diversi tipi di stampanti: a matrice d'aghi (le più comuni), a getto d'inchiostro (le più silenziose), laser (le migliori ma anche le più costose), a margherita (che forniscono la miglior stampa del testo dopo quelle laser). La differenza tra i vari tipi risiede nelle modalità utilizzate per trasferire su carta i dati (testi od immagini); le stampanti a margherita non possono stampare immagini, in quanto esse sono costruite come le macchine da scrivere. In modo ad alta qualità (NLQ) le stampanti ad aghi possono produrre risultati che sembrano quasi buoni quanto quelli ottenibili con stampanti a margherita.

Token
Un token è un codice di un byte che identifica un comando BASIC. AmigaBASIC usa i token per risparmiare spazio quando registra i programmi su memorie di massa (dischetto, disco rigido). Ad esempio, anzichè salvare l'istruzione PRINT come sequenza dei caratteri P, R, I, N e T, AmigaBASIC registra 172 (il token corrispondente). Quando AmigaBASIC recupera i da-

ti immagazzinati, rimpiazza il codice 172 con il comando PRINT.

Utility

Viene indicato con questo termine un programma che aiuta l'utente a svolgere determinati compiti. E' possibile considerare utility (o programma di utilità) un qualsiasi programma che si ritenga utile. Questo libro contiene numerosi utili programmi e quindi contiene un gran numero di utility (almeno si spera!).

Indice Analitico

AmigaBASIC

Un volume completo per chi intende sfruttare le grandi potenzialità di Amiga usando il BASIC. AmigaBASIC rappresenta una combinazione di un tutorial per principianti, una guida avanzata ed un manuale di riferimento, tutto in un unico libro, insegnando il BASIC con un approccio applicativo e spiegando ogni dettaglio del linguaggio in termini chiari e di facile comprensione. Ogni utente Amiga apprezzerà il grande valore della sezione di riferimento: in essa si trovano infatti un completo glossario, una guida di rapida consultazione e dettagliate descrizioni dei messaggi d'errore con opportuni suggerimenti.

AmigaBASIC tratta argomenti quali:

- finestre
- funzioni del mouse
- animazione di oggetti
- risoluzione video
- definizione di menú a tendina
- fill pattern
- comandi per la gestione del disco
- formati dei file
- creazione di comandi personalizzati

Vengono presentati, descritti e spiegati numerosi programmi di pronto utilizzo: interessanti animazioni grafiche; un programma di videotitolazione; un potente database; un ottimo programma di disegno; applicazioni di visualizzazione dati per la creazione di istogrammi e grafici a torta; un programma di sintetizzazione per realizzare effetti musicali personalizzati.

AmigaBASIC costituisce un indispensabile tutorial e guida di riferimento per tutti gli utenti Amiga.

FTE • FREE TIME EDITIONS s.r.l.
Sede: Via Sassoferrato, 1 • 20135 Milano • Telefoni: 02/5459785•5452756

PREZZO L. 60.000 COD. 104
ISBN 88-85111-03-3