Programmare in C su Amiga (27)

di Dario de Judicibus (MC2120)

Con la descrizione della struttura utilizzata per definire immagini grafiche a mappa di bit, chiudiamo la rassegna sulle strutture grafiche referenziabili dai vari oggetti che compongono l'interfaccia a finestre del nostro Amiga, e cioè quadri, controlli e menu

In questa puntata concludiamo la carrellata sulle strutture grafiche elementari parlando di *immagini*. Le immagini, come dice la parola, sono oggetti grafici definiti punto per punto in un campo rettangolare. Ovviamente la forma non deve essere necessariamente rettangolare; tuttavia la matrice che definisce i singoli pixel delimita un'area di tale forma. Anche queste strutture, come già **IntuiText** e **Border**, sono strutture elementari utilizzabili direttamente o referenziabili da altri oggetti più complessi. Ed anche queste strutture, come quelle viste in precedenza, fanno sempre riferimento ad un *elemento contenitore* che rappresenta anche il sistema di coordinate nel quale vanno posizionate. Come già detto nelle scorse puntate, il punto del contenitore rispetto al quale sono definiti tali oggetti si chiama *origine*.

Image

La struttura **Image** permette di definire qualunque immagine grafica, di qualunque dimensione e forma, nei limiti della memoria di tipo *chip* disponibile. Ricordo infatti che i dati relativi alle immagini, così come qualunque altro dato che deve essere reso accessibile ai processori speciali, va posizionato appunto in questo tipo di memoria. Per far questo ci sono due possibilità: o si alloca dinamicamente memoria specificando **MEMF_CHIP**, oppure si definisce staticamente un'area di memoria nella zona in questione nella dichiarativa dell'area dati dell'immagine, utilizzando l'attributo **chip** che molti compilatori C per Amiga mettono a disposizione dei programmatori. Nel caso che si debba utilizzare un'immagine predefinita la seconda soluzione è la migliore, in quanto permette di specificare i dati dell'immagine una sola volta, staticamente. La prima soluzione infatti costringerebbe il programmatore a definire comunque l'immagine staticamente, che finirebbe poi nella memoria *fast* con il resto del programma, per poi allocare in *chip* un'uguale quantità di memoria e copiare il contenuto della prima nella seconda, come mostrato qualche anno fa in questa stessa rubrica. Questa soluzione, tuttavia, può risultare più conveniente nel caso l'immagine venga generata dinamicamente dallo stesso programma, specialmente se l'area da allocare è considerevole, in quanto riduce le dimensioni dell'eseguibile.

La struttura **Image** (vedi figura 6) è formata da nove campi:

LeftEdge
indica la distanza in pixel del lato sinistro dell'immagine dall'origine dell'elemento contenitore;

TopEdge
indica la distanza in pixel del lato superiore dell'immagine dall'origine dell'elemento contenitore;

Width
è la larghezza in pixel dell'immagine;

Height
è l'altezza in pixel dell'immagine;

Depth
è il numero di piani necessari a definire l'immagine;

ImageData
è il puntatore alla matrice che contiene la definizione dell'immagine, come spiegato più avanti;

PlanePick
è una maschera di bit che indica quali piani dell'elemento contenitore sono destinati a ricevere l'immagine;

PlaneOnOff
è una maschera di bit che indica come riempire quei piani dell'elemento contenitore che non sono destinati a ricevere l'immagine;

NextImage
punta ad un'altra struttura **Image**, permettendo così di collegare più immagini insieme a formare una lista.

Naturalmente anche le immagini, come già i bordi ed i testi, possono essere direttamente disegnati in un *raster* utilizzando una funzione apposita, oltre che essere indirettamente resi quando viene visualizzato un oggetto che li refe-

renzia, come ad esempio un quadro od un controllo. Questa funzione si chiama **DrawImage()** ed è riportata in figura 7.

Vediamo ora praticamente come costruire un'immagine.

Per prima cosa bisogna decidere cosa si vuole disegnare, ed in quanti colori. Supponiamo ad esempio di voler disegnare una doppia freccia in avanti, come quella usata per lo scorrimento veloce in avanti nei registratori o nei VCR [*fast forward*], e di voler usare un solo colore più lo sfondo. Per prima cosa (fase zero od iniziale), disegniamo l'immagine su di un pezzo di carta, non troppo piccola. Quindi sovrapponiamole una griglia quadrettata di dimensioni opportune, a seconda di quanti pixel vogliamo usare in larghezza ed altezza. Ogni quadretto rappresenterà un pixel. A questo punto, come si può vedere in figura 1 (fase uno), facciamo una crocetta su un quadretto, se l'area sottesa è compresa nella zona colorata per più del 50%, lasciamolo vuoto in caso contrario. Naturalmente, in caso di più colori avremmo usato una lettera diversa per ciascun colore (X, Y, Z,...) e definito una regola un po' più complessa per le assegnazioni dei colori ai vari quadretti. Se l'immagine è complessa e sappiamo già che colori (reali, non registri) andremo ad usare, potremmo anche pensare a qualche tecnica di sfumatura dei bordi [*anti-aliasing*]. Per ora, comunque, limitiamoci ad un solo colore più sfondo.

La fase due (vedi figura 2) consiste nel separare i vari piani, preparando una griglia per piano, per poi assegnare a ciascun quadretto un uno oppure uno zero a seconda del registro di colore assegnatogli. Conviene dividere verticalmente la griglia in blocchi da quattro pixel, sia perché così risulta più facile poi calcolare i valori esadecimali equivalenti ai *nibble* (mezzo byte) formatisi, sia perché la matrice finale è formata da elementi di due byte l'uno, e bisogna quindi definire comunque un numero intero di questi elementi per ogni riga. Questo vuol dire che, a seconda di come si centrerà l'immagine sulla griglia, formata sempre da un numero di pixel multipli di 16 (due byte appunto), alcune colonne a destra e/o a sinistra dell'immagine saranno eventualmente riempite da zeri in tutti i piani. Nel nostro caso sono vuote la prima e l'ultima colonna (colore trasparente o di fondo).

Nella fase tre (vedi figura 3), infine, per ogni piano si calcolano le parole da due byte che formano ogni riga, in modo da generare il vettore di definizione dell'area dati dell'immagine. Se ci sono più piani (non è il caso mostrato nell'esempio), questi vanno riportati in ordine crescente, cioè: tutti i dati relativi all'intera immagine per il piano zero per primi, quindi tutti quelli relativi al piano uno, due, e così via.

A questo punto, tuttavia, bisogna de-

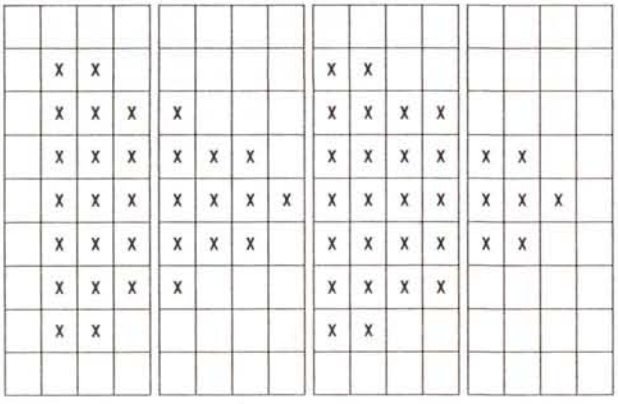

Figura 1 - Come costruire un'immagine: fase uno.

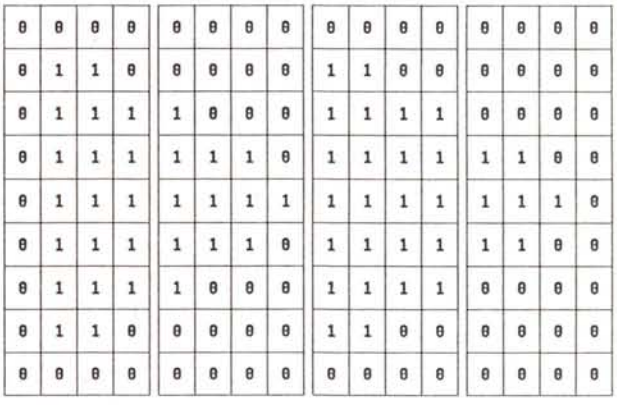

Figura 2 - Fase due.

```
0000 0000 0000 0000  |  0000  |
0110 0000 1100 0000  |  60C0  |
0111 1000 1111 0000  |  78F0  |
0111 1110 1111 1100  |  7EFC  |
0111 1111 1111 1110  |  7FFE  |
0111 1110 1111 1100  |  7EFC  |
0111 1000 1111 0000  |  78F0  |
0110 0000 1100 0000  |  60C0  |
0000 0000 0000 0000  |  0000  |
```

Figura 3 - Fase tre.

```
USHORT chip AvantiVeloce[] =
{
   0x0000,   /* Larghezza =   */
   0x60C0,   /*   16 pixel    */
   0x78F0,   /*               */
   0x7EFC,   /* Altezza =     */
   0x7FFE,   /*    9 pixel    */
   0x7EFC,   /*               */
   0x78F0,   /* Profondità =  */
   0x60C0,   /*    1 piano    */
   0x0000    /*               */
};
```

Figura 4 - Struttura finale.

```
USHORT chip AvantiVeloce[] =
{
   0x0000, 0x60C0, 0x78F0,
   0x7EFC, 0x7FFE, 0x7EFC,
   0x78F0, 0x60C0, 0x0000
};

struct Image AV =
{
   0, 0,              /* La posizione la calcoleremo in seguito          */
   16, 9, 1,          /* Largo 16 pixel, alto 9 - Due soli colori        */
   AvantiVeloce,      /* Il puntatore all'area dati - la doppia freccia */
   0x01, 0x02,        /* blu su rosso - orribile, vero?                  */
   NULL               /* Nessun altra immagine                           */
};
```

Scheda tecnica

Anche questo mese ecco cinque nuovi comandi dell'*AmigaDos 1.3* a partire da **JOIN**.

LEGENDA	
<parametro>	parametro da specificare
[<opzione>]	parametro opzionale
{<opz-rip>}	parametro opzionale che può essere ripetuto n volte
...	serie che può essere continuata
|	separatore per una lista di opzioni di cui una almeno VA specificata
/A	indica che il parametro DEVE essere specificato
/K	indica che quella determinata parola chiave VA specificata se si vuole usare l'opzione ad essa associata
/S	indica una parola chiave da specificare per attivare l'operazione ad essa associata

Comando:	LIST
Formato:	LIST [<indirizzario|modello>] [P|PAT <modello>] [KEYS] [DATES] [NODATES] [TO <nome>] [SUB <stringa>] [SINCE <data>] [UPTO <data>] [QUICK] [BLOCK] [NOHEAD] [FILES] [DIRS] [LFORMAT <formato>]
Sintassi:	LIST "DIR,P=PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K,SUB/K, SINCE/K,UPTO/K,QUICK/S,BLOCK/S,NOHEAD/S,FILES/S, DIRS/S,LFORMAT/K"
Scopo:	Lista informazioni selezionabili relative ad indirizzari e file
Specifiche:	Ora è possibile specificare un modello di ricerca direttamente, senza bisogno di usare P o PAT. I nuovi attributi, cioè s (script), p (pure), a (archive), sono stati aggiunti a quelli già visualizzati dalla versione 1.2. FILES e DIRS limitano rispettivamente la lista ai soli file od ai soli indirizzari. QUICK non visualizza più gli spazi extra in fondo ai nomi. BLOCK fornisce le dimensioni dei file in blocchi piuttosto che in byte. NOHEAD non stampa la testata. QUICK e NOHEAD sono i defaults nel caso sia stata usata l'opzione LFORMAT. Quest'ultima è stata spiegata in dettaglio nella Scheda Tecnica della puntata N°20 di questa rubrica.
Esempio:	LIST devs: KEYS NOHEAD NODATES BLOCK produce JDisk.device [10638] 7 ---arwed serial.device [10640] 11 ---arw-d keymaps [10642] Dir ----rwed parallel.device [10645] 4 ---arw-d MountList [10647] 6 ---arw-d clipboard.device [10649] 14 ---arw-d printer.device [10651] 56 ---arw-d narrator.device [10886] 48 ---arw-d printers [10654] Dir ----rwed clipboards [10661] Dir ----rwed system-configuration [10664] 1 ----rwed

Il termine indirizzario è l'equivalente italiano di directory

Comando:	LOCK
Formato:	LOCK <unità>: [ON|OFF] [parola d'ordine]
Sintassi:	LOCK "DRIVE/A,ON/S,OFF/S,PASSKEY"
Scopo:	Imposta la protezione da scrittura del disco fisso (solo FFS)
Specifiche:	Utilizzando una proprietà delle sole partizioni FFS, questo comando permette di impostare o cancellare la protezione da scrittura di un disco fisso o di una sua partizione. Una ripartenza anche a caldo, tuttavia, cancella il blocco impostato. E' possibile specificare una parola d'ordine quando si richiede il blocco. In tal caso la stessa parola deve essere fornita alla richiesta di sprotezione. La parola d'ordine è facoltativa e non può essere formata da più di 4 caratteri.
Esempio:	LOCK dh1: ON rosa

Comando:	JOIN
Formato:	JOIN <nome> <nome> ... AS|TO <nome>
Sintassi:	"JOIN NAME(S),TO=AS/K"
Scopo:	Concatenazione di files
Specifiche:	E' in grado di concatenare, cioè di aggiungere sequenzialmente in modo da formare un solo file, fino a 15 file. Ovviamente questo limite può essere superato come mostrato nell'esempio.
Esempio:	JOIN f1a f2a f3a f4a f5a f6a f7a f8a f9a f10a AS ja JOIN f1b f2b f3b f4b f5b f6b f7b f8b f9b f10b AS jb JOIN ja jb AS fj

Per esercizio, provate a scrivere una semplice macro (un file SCRIPT od anche ARexx, se lo conoscete), per concatenare un numero qualunque di file.

Comando:	LOADWB
Formato:	LOADWB [DELAY] [-DEBUG]
Sintassi:	LOADWB "DELAY/S,-DEBUG/S"
Scopo:	Fa partire il WorkBench
Specifiche:	In genere questo comando va piazzato nella startup-sequence per far partire il WorkBench. Quando il comando viene eseguito il WB fotografa la situazione al momento dei cammini [path] definiti, e la usa per tutte le finestre CLI o Shell che l'utente aprirà in seguito. DELAY permette un ritardo di tre secondi tra la richiesta di esecuzione e l'esecuzione vera e propria del comando. Se si usa l'opzione -DEBUG, viene aggiunto al menù di sistema un nuovo menù, contenente due voci: Debug e FlushLib. Il primo attiva ROMWACK, il debugger di Amiga residente in ROM. Per utilizzarlo, cosa peraltro non per principianti, è necessario collegare all'Amiga un terminale seriale da 9600 baud. FlushLib, invece, fa sì che WB provi ad allocare quanta più memoria possibile, liberando tutta quella allocata per quelle librerie, device, font, ed altro codice residente non in uso al momento della richiesta. Le due opzioni NON possono essere usate contemporaneamente.

Comando:	MOUNT
Formato:	MOUNT <unità> [FROM <file>]
Sintassi:	MOUNT "DEVICE/A,FROM/K]
Scopo:	Monta una nuova unità
Specifiche:	Serve a creare un nuovo nodo nella lista dei device del sistema, cercando nel file MountLists nel'indirizzario DEVS:, o nel file specificato con l'opzione FROM, i parametri relativi alla nuova unità. A questo comando dedicheremo una puntata della Scheda Tecnica.
Esempio:	MOUNT ms1: FROM devs:msdos_device

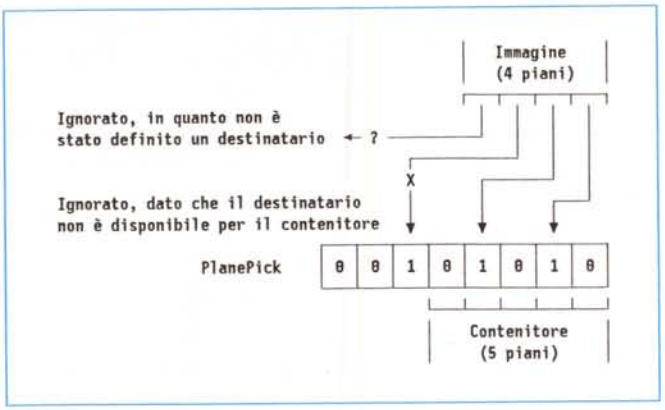

Figura 5 Assegnazione dei piani.

finire gli altri campi della struttura **Image**, alcuni dei quali sono fondamentali ai fini del risultato finale. Non basta infatti definire i colori dei vari piani dell'immagine. I colori finali, e quindi il modo in cui l'immagine sarà effettivamente visualizzata, dipendono anche dalle caratteristiche del *raster* che riceverà tale immagine, ed in particolare da quali piani dovranno essere utilizzati e cosa fare di quelli non utilizzati. Ma andiamo per ordine.

Vediamo prima i campi «più semplici» da definire. Innanzi tutto la posizione dell'immagine relativamente all'origine del contenitore, cioè i due campi **LeftEdge** e **TopEdge**. Questi campi danno le coordinate dell'angolo in alto a sinistra del rettangolo di pixel che abbiamo appena definito, utilizzando la tecnica della griglia quadrettata, nel sistema di riferimento del contenitore. Se già sapete in quale oggetto volete piazzare l'immagine, potete impostare questi valori opportunamente, sempre tenendo presente che il risultato sarà differente a seconda di come avete centrato l'immagine vera e propria nel rettangolo utilizzato. In molti casi, tuttavia, conviene calcolare automaticamente tale posizione. Vediamo perché.

Supponiamo di aver definito un'immagine da utilizzare come voce di un menu. Inoltre l'immagine va posizionata dopo un testo, sempre nello stesso menu. I due campi in questione vanno quindi calcolati in funzione delle dimensioni del rettangolo che contiene la voce e della lunghezza del testo utilizzato. Ma che succede se decidiamo di cambiare il testo, od anche semplicemente di utilizzare un font più largo o più stretto per quella voce? Ci tocca ricalcolare tutto da capo, ovviamente. Ed è qui il punto cruciale: *quando si costruisce un oggetto formato da vari componenti, se si codificano tutte le caratteristiche dei vari componenti in modo fisso [hard-code], ogni qual volta si deve modificare un componente, bisogna variare anche gli altri.*

Un buon programmatore, tuttavia, ha sempre un occhio verso i problemi di manutenzione del programma. Pensa cioè sempre al futuro. In questo caso, la cosa migliore da fare è impostare i due valori **LeftEdge** e **TopEdge** a zero, e lasciare che sia il programma a calcolare tali valori sulla base delle dimensioni del contenitore, a fronte di una logica di posizionamento, piuttosto che di valori fissi che ci si potrebbe anche dimenticare di aggiornare qualora si debbano modificare le dimensioni del contenitore. La programmazione *di alto livello* insegna che bisogna ragionare in termini astratti, isolando la proiezione dei concetti dentro scatole nere. È da queste basi che si arriva alla programmazione per oggetti, o *OOP (object-oriented programming)*.

Torniamo comunque alla nostra struttura, lasciando da parte un discorso che ci porta troppo lontano ed avrebbe bisogno di un proprio spazio. L'impostazione dei due campi **LeftEdge** e **TopEdge** a zero può risultare più pratica anche quando si usa la **DrawImage()**. Infatti questa funzione permette già di specificare la posizione dell'immagine, grazie ai parametri **LeftOffset** e **TopOffset**, i cui valori andrebbero a sommarsi a quelli dei primi due campi della struttura **Image**. Se la stessa immagine è usata più volte in posizioni differenti, può essere conveniente lasciare che siano i parametri della funzione a definirne la posizione, piuttosto che la somma di questi e di quelli specificati nella struttura stessa. Più immediata è l'identificazione dei dati responsabili di determinati effetti, e più semplice è modificare il programma od individuare eventuali errori. Ovviamente queste sono raccomandazioni generali, non applicabili sempre e comunque. Tuttavia, in molti casi, permettono di scrivere programmi più flessibili e di ridurre i tempi di ricerca di eventuali difetti.

Impostiamo quindi i due valori a zero. Passiamo ora ai tre campi successivi. Avendo centrato l'immagine sia orizzontalmente che verticalmente, comprenderemo nel calcolo anche la cornice da un pixel che circonda la doppia freccia. Risultato: 16 pixel in larghezza e 9 in altezza. Se avessimo deciso di restringere il rettangolo al solo simbolo di *Avanti Veloce*, avremmo avuto bisogno di soli 14 pixel per 7. Da notare che questo ci avrebbe permesso di usare solo sette elementi nel vettore **AvantiVeloce** ma, dato che ogni elemento deve essere di due byte, avremmo avuto comunque da definire 16 bit per riga (gli ultimi due nulli), anche se avremmo impostato

```
struct Image
{
  SHORT         LeftEdge, TopEdge; /* Posizione relativa all'origine        */
  SHORT         Width, Height;     /* Dimensioni dell'immagine              */
  SHORT         Depth;             /* Numero di piani per l'immagine        */
  USHORT       *ImageData;         /* Puntatore ai dati dell'immagine       */
  UBYTE         PlanePick;         /* Quali piani ricevono l'immagine       */
  UBYTE         PlaneOnOff;        /* Che fare con gli altri piani          */
  struct Image *NextImage;         /* Puntatore ad un'altra struttura Image */
};
```

Figura 6 - Image

```
void DrawImage   /* Stampa l'immagine nel raster in una certa posizione */
(
  struct RastPort  *RastPort  , /* Destinazione dell'immagine           */
  struct Image     *Image     , /* Immagine da stampare (od una catena) */
  SHORT             LeftOffset, /* Posizione da sinistra                */
  SHORT             TopOffset   /* Posizione dall'alto                  */
);

NOTA: nelle scorse puntate, per errore, ho invertito l'ordine di LeftOffset
      e TopOffset, sia per DrawBorder che per PrintIText. L'ordine corretto
      è lo stesso riportato qui sopra. Me ne scuso con i lettori.
```

Figura 7 - DrawImage().

```
        MOVE.L   $4,A6
        CMP.B    #$32,$0212(A6)     ;Controlla il PowerFrequency
        BEQ.S    FINE
        MOVE.L   #$FC0002,$000000B4
        TRAP     #13
FINE:   CLR.L    D0
        RTS
```

Figura 8 - Casella Postale: routine assembler.

Width a 14 comunque. Intuition semplicemente avrebbe ignorato le due colonne di bit a destra nell'area dati dell'immagine. In quanto al numero di piani dell'immagine, dato che abbiamo solo due colori, cioè tratto e fondo, il valore di **Depth** sarà uno.

Ma che succede se il contenitore ha un numero di piani maggiore? Facciamo un esempio. Supponiamo di essere nello schermo del WorkBench e che quindi il contenitore, diciamo una finestra, sia formata da due piani, in modo da avere quattro colori. Prendiamo un pixel. Se il punto ad esso relativo in entrambi i piani è a zero, il registro di colore è **0** (in genere in fondo). Se quello stesso punto è a uno solo nel piano zero, il registro è **1**, e così via fino a **3** (uno in entrambi i piani).

Se adesso diciamo a Intuition di caricare l'immagine nel contenitore, il colore della doppia freccia sarà differente a seconda che essa sia resa nel piano zero o nel piano uno. Supponiamo che i colori siano i seguenti: grigio (0), nero (1), rosso scuro (2) e blu scuro (3). Se l'immagine viene caricata nel primo piano, la freccia sarà nera su fondo grigio. Se essa invece viene caricata nel secondo piano, sarà rossa su fondo grigio.

E cosa succede se tutti i bit del piano non interessato dall'immagine vengono impostati a uno? In questo caso la freccia sarà blu su fondo rosso nel primo caso, e blu su fondo nero nel secondo. Verificatelo con un programmino.

I campi **PlanePick** e **PlaneOnOff** servono appunto a specificare quale di queste combinazioni va utilizzata. Sono entrambi due maschere da 8 bit ciascuna — per potenziali otto piani, quindi — in cui ogni bit rappresenta un piano. Il bit meno significativo — quello a destra, tanto per intenderci — rappresenta il piano zero, gli altri a seguire. **PlanePick** indica quali piani debbano ricevere i piani dell'immagine. Nel nostro caso scegliamo come combinazione la doppia freccia in blu su fondo rosso. Questo vuol dire che a ricevere l'unico piano definito per l'immagine è il piano zero. Di conseguenza la nostra maschera sarà:

0 0 0 0 0 0 0 1 cioè 0x01

dato che il significato di un uno in questa maschera è appunto quello di indicare che il piano corrispondente sarà interessato dall'area dati dell'immagine. **PlaneOnOff**, invece, dice cosa fare dei piani non interessati dal caricamento dell'immagine, quelli cioè impostati a zero in **PlanePick**. In questo caso, il bit corrispondente al piano indica se esso debba essere riempito di **0** (bit a zero) o di **1** (bit a uno) per la *sola* parte sovrapposta al rettangolo dell'immagine, ovviamente. Nel nostro esempio, il piano uno va riempito di **1**, e di conseguenza la maschera sarà:

0 0 0 0 0 0 1 0 cioè 0x02

Da notare che il bit zero, quello cioè corrispondente al piano zero, avrebbe anche potuto essere uno, dato che quel piano riceve l'immagine e quindi **PlaneOnOff** non ha alcun effetto su di esso.

La struttura finale è riportata in figura 4.

Naturalmente, qualora sia l'immagine che il contenitore siano formati da più piani, il primo piano dell'immagine va nel primo piano ad avere un bit ad uno in **PlanePick**, il secondo nel secondo piano ad avere un bit non nullo in tale maschera, e così via, come mostrato in figura 5. Nel caso poi che il numero di piani del contenitore destinati a ricevere l'immagine sia inferiore al numero di piani dell'immagine, vuoi perché il contenitore non ha abbastanza piani, vuoi perché il numero di piani marcati in **PlanePick** sono troppo pochi o sono marcati piani non utilizzabili dal contenitore (nell'esempio in figura, il sesto), Intuition si limita ad ignorare i piani dell'immagine in eccesso, visualizzandone così solo una sezione parallela al piano dello schermo, e perdendo così una parte dell'informazione sui colori, se non addirittura alcuni aspetti dell'immagine.

Corollario: se state scrivendo un programma in cui l'utente può definire il numero di colori da usare nello schermo, e quindi il numero di piani dello stesso, limitatevi a disegnare controlli [*gadget*] che possano essere visualizzati comun-

```
;Assemblate con il DevPec specificando LINKABLE anziché EXECUTABLE
;Salvate con ram:esempio
;Caricate il PREZIOSO C-MONITOR V2.02 (o similare) e digitate:
;L ram:esempio 50000 (mi raccomando la elle maiuscola)
;adesso per finire salvate su disco il BootBlock digitando:
;>t 50028 00 01 (la t sempre minuscola)
;Adesso avete su boot il programma.
;La routine del bootblock standard è presa dall'articolo comparso sul nr 91
Parto: dc.b    'DOS',0                    ;Identifica un disco DOS
       dc.l    $91a3502d                  ;Checksum del bootblock
       dc.l    $370                       ;vedere MC nr 91.(He he he! )
       move.l  $4,e6                      ;Questo è lo stesso
       cmp.b   #$32,$0212(a6)
       beq.s   fine                       ;programma del listato
       move.l  #$fc0002,$000000b4
       trap    #13                        ;precedente
fine:  lea     dos(pc),a1                 ;Questo è l'aspetto
       jsr     -96(a6)
       tst.l   d0
       beq.s   err                        ;di un bootblock
       movea.l d0,a0
       movea.l $16(a0),a0                 ;standard
       moveq   #0,d0
usc:   rts
err:   moveq   #-1,d0                     ;errore ci fu
       bra.s   usc
dos    dc.b    'dos.library',0
;Le linee che seguono servono ad identificare il bootblock come INNOCUO
       dc.b    "QUESTO NON E' UN VIRUS MA SOLO UN BOOBLOCK CHE CONTROLLA SE LO"
       dc.b    "SCHERMO E' PAL"
       dc.b    'CREATO DA DAVIDE FICANO.'
arrivo: dc.b   'SE FOSSE UN VIRUS NON METTEREI IL MIO NOME!!'
       dcb.b   1024-(arrivo-parto),0      ;In questo modo il bootblock
                                          ;contiene memoria sporca
```

Figura 9 - Casella Postale: caricamento nel boot-block.

que nel numero minimo di piani che l'utente è autorizzato a definire, altrimenti il vostro disegnino potrebbe diventare incomprensibile.

Conclusione

Adesso abbiamo le basi per affrontare il lungo e complesso discorso dei controlli e dei quadri. L'Amiga permette una grande flessibilità nella definizione di questi oggetti, ma purtroppo non fornisce funzioni di alto livello per la costruzione di oggetti standard completi, come invece è possibile avere nel Macintosh. Per fortuna le cose sono cambiate con la versione 2.0 del sistema operativo, dove, ad esempio, è possibile avere un *file requester* già pronto per l'uso, e vengono messe a disposizione del programmatore molte nuove funzioni per i controlli ed i quadri. Dato che però il parco macchine attuale è ancora formato prevalentemente da Amiga che girano la versione 1.3 del sistema operativo, per il momento dovremo inventarci noi qualcosa per rendere la costruzione dei controlli un po' più semplice. Al momento ci sto lavorando sopra. Vedremo un po' che ne esce fuori. MC

Casella Postale

Ancora una lettera, questa volta da Palermo. In essa il Sig. Ficano riprende il discorso relativo alla mancata apertura di uno schermo PAL su macchine di questo tipo, dovuta ad un errore in una procedura della **graphics.library**, presentato nella *Scheda Tecnica* del numero 99 di MC (novembre 1989).

Comando FF

Egr. Dott. de Judicibus,
ritengo la sua rubrica sul C per Amiga la cosa più interessante che un programmatore possa trovare, e per questo motivo le scrivo per dare a lei e a i suoi lettori qualche dritta in più.

Per prima cosa devo dirle che nella scheda tecnica del nr. 99 lei giustamente dubita sul funzionamento del comando FF, ma esso funziona perfettamente con la sintassi corretta nel modo seguente:

FF siesta.font

cioè al nome del font bisogna aggiungere il suffisso **.font.** *Attenzione però, il font non deve essere proporzionale. Se è proporzionale lo si può aggiustare lavorando con il FED dell'EXTRAS: si carica il font desiderato con il FED, si sceglie FIXEDWIDTH dal submenu FONT TYPE del menu ATTRIBUTES, si salva e il gioco è fatto.*

Altra cosa, qualche tempo fa lei ha spiegato come risolvere il bug per il quale si apriva uno schermo NTSC circa ogni 30 reset, però lei lo risolve su un 1000 (a buon intenditor poche parole!), io allego una piccola routine in assembler che piacerà ai 2000 & 500 con l'1.3 (vedi figura 8). Basta assemblarla con il devpac e metterla in cima alla startup-sequence. Ma si può metterla anche su boot-block, come mostrato in figura 9.

Ho cercato di essere molto corto ma forse non ci sono risuscito! In bocca al lupo a tutti, e ancora complimenti per la rubrica e la rivista. Scusi per la pessima qualità di questo dattiloscritto.

Davide Ficano

Ci sono due cose che mi fanno molto piacere e mi stimolano a continuare a proporre in questa rubrica sempre qualcosa di più del semplice riportare informazioni che, con un po' di pazienza, qualunque programmatore con un minimo di esperienza riuscirebbe a mettere insieme a partire dall'ormai sostanziosa mole di dati reperibile sui vari manuali della Commodore e sulle decine di libri scritti sull'Amiga. La prima è che sempre più persone dimostrano di voler affrontare argomenti e linguaggi che vanno al di là del programmino in Basic per stampare la schedina o disegnare una sinusoide.

Questo spirito da *hacker*, utilizzando il termine nella sua asserzione positiva, cioè di chi vuole andare a fondo nelle cose in modo non solo di capirne il funzionamento, ma di arrivare addirittura a sfruttare questa conoscenza per ottenere nuovi risultati ed esplorare nuove ed interessanti possibilità che spesso vanno al di là delle intenzioni originali degli ideatori del sistema che si sta analizzando, è la base di quella cultura informatica ancora così poco sviluppata nel nostro paese, e comunque ristretta ad una cerchia di appassionati e di conseguenza poco applicata nella vita di tutti i giorni.

Basti pensare al classico «*fatto dal computer*» piuttosto che «*fatto con il computer*», che i nostri telegiornali spesso ci propongono e che rivela come queste macchine siano ancora viste come entità quasi magiche, intelligenze indipendenti dagli uomini che le usano, piuttosto che come semplici strumenti di lavoro con ben precise capacità e limitazioni.

La seconda cosa che mi dà molta soddisfazione è vedere come questo sforzo di approfondimento e di ricerca non sia limitato a poche regioni, ma è ormai diffuso in tutto il paese, come dimostrano le molte lettere che ho ricevuto da ogni parte di Italia, Nord, Centro, Sud ed Isole. Purtroppo lo stesso equilibrio non l'ho riscontrato per quello che riguarda il rapporto tra uomini e donne.

Quest'ultime sembrano essere ancora un'esigua minoranza, non tanto nel mondo dell'informatica, quanto in quello dei «dilettanti esperti», quali sono ad esempio gli autori dei molti programmi di pubblico dominio che si possono trovare in circolazione. Molte donne lavorano nel campo dell'informatica, poche però sembrano dimostrare quella passione che spinge ad affrontare questo mondo da un punto di vista hobbistico. Spero vivamente che tutto ciò cambi in futuro, grazie anche al contributo di riviste come MC.

Tornando alla lettera del Sig. Ficano, raccomando ancora una volta a tutti i lettori che si vogliono confrontare con il problema riportato, di non partire in quarta se non si ha una buona conoscenza dell'argomento trattato, peraltro decisamente *avanzato*. Non ho avuto modo di provare la routine, avendo un A1000, ma penso che non dovrebbe essere di difficile verifica a chi conosce l'Assembler del 68000. In quanto al comando FF, pur avendo il sistema originale 1.3 comprato negli USA appena uscì, continua a non funzionarmi, con o senza il suffisso **.font**. Forse la mia versione contiene un errore fissato in seguito. Non è comunque un problema, dato che uso il programma **PD setfont**.

A proposito di lettere, una cortesia. Non spedite lettere scritte a mano o stampate con caratteri proporzionali. L'ideale è la buona vecchia macchina da scrivere od un font sottile e chiaro non proporzionale, così da permettermi di caricare il testo con lo *ScanMan* ed interpretarlo con un OCR, evitandomi così di ricopiarlo a mano.

Grazie.

MC