Programmare in C su Amiga

di Dario de Judicibus

ottava puntata

Continuiamo con questa puntata il nostro viaggio nei segreti di Intuition, in modo da preparare, un passo alla volta, una solida base su cui costruire programmi in grado di sfruttare appieno uno dei più interessanti componenti del sistema operativo dell'Amiga

L'esercizio della scorsa puntata aveva principalmente due obiettivi: il primo era quello di familiarizzare il lettore con Intuition, essendo il primo esercizio di programmazione in C proposto ai lettori da quando abbiamo iniziato ad affrontare Intuition; il secondo, quello di dimostrare alcuni dei limiti imposti da Intuition per quello che riguarda le dimensioni degli schermi.

Se siete riusciti a scrivere il programma in questione (vedi figura 1), vi sarete certamente resi conto che, pur essendo di dimensioni relativamente ridotte, e di conseguenza offrendo funzionalità abbastanza scarse, una volta lanciatolo, è possibile spostare la finestra qua e là per tutto il tempo che si vuole, fino a che non si chiude il tutto selezionando il gadget di chiusura della finestra. Questa è di fatto la vera potenza di Intuition: pur non limitandovi nelle vostre scelte e nel grado di controllo che desiderate avere dell'ambiente creato (schermi, finestre, gadget, menu...), Intuition è in grado di gestire in modo automatico tutto ciò di cui non volete interessarvi personalmente, secondo determinati schemi in buona parte anch'essi definibili dal programmatore. È insomma un perfetto maggiordomo, che vi lascia fare tutto (o quasi) quello che volete, ma è pronto a sostituirvi nel fare gli onori di casa non appena gli restituite il controllo.

Tornando al nostro esercizio, come potete vedere in figura 1, si tratta proprio di un programmino elementare.

In pratica, una volta aperta la libreria di Intuition, si apre uno schermo di dimensioni 100×100 in alta risoluzione ed una finestrella 60×60. A questo punto ci si mette in attesa sulla porta utilizzata da Intuition per fornire informazioni e ricevere istruzioni relativamente a *quella specifica finestra*. Dato che l'unico messaggio che abbiamo richiesto a Intuition è quello relativo alla selezione del gadget di chiusura della finestra, potremo tranquillamente muovere la finestra per lo schermo senza per questo uscire dal programma. È importante capire a questo punto che, se avessimo informato Intuition che desideravamo anche essere avvisati di altre operazioni effettuate dall'utente, avremmo dovuto intercettare il contenuto del messaggio inviato (vedi quinta puntata — MCmicrocomputer n. 78), replicare al messaggio, effettuare eventuali operazioni previste a fronte del messaggio ricevuto e rimetterci in attesa di quello successivo a meno che non si fosse trattato della selezione del gadget di chiusura. In quest'ultimo caso, saremmo usciti dal programma dopo aver fatto le stesse operazioni di chiusura riportate anche nel nostro programmino.

In realtà il codice proposto non è molto educato, dato che, ricevuto il messaggio, se ne esce tranquillamente senza neanche un grazie ad Intuition. Chi ha letto con attenzione la quinta puntata, si ricorderà certamente che bisogna sempre replicare ad un messaggio ricevuto, per evitare di lasciare appese aree di memoria od addirittura interi task. Per fortuna Intuition è abbastanza intelligente da cavarsela anche da solo, in questo caso! Può anche succedere che tutto sembri funzionare lo stesso (non è sempre facile accorgersi di aver *perso* qualche byte!), ma, dato che non sempre sappiamo come si comporta internamente Intuition, non è detto che in una versione successiva le cose vadano altrettanto bene. Morale della favola: scrivete sempre programmi educati, anche se questo vi costa qualche riga in più.

Per quello che riguarda il funzionamento del programma, avrete notato come, pur avendo definito uno schermo molto piccolo, questi sembra ricoprire l'intero schermo fisico. Se tuttavia spostate la finestra, vi accorgerete di non poter andare più in là di un rettangolo ben definito, guarda caso 100×100 pixel.

Questo è il motivo perché, a meno di ragioni contingenti, si definiscono sempre schermi che assumino orizzontalmente la massima estensione. Analogamente, se la loro altezza è inferiore a quella disponibile verticalmente, il loro angolo superiore sinistro va posizionato in modo che il bordo inferiore dello schermo non sia al di sopra di quello del monitor. Questa, che è una limitazione imposta da Intuition 1.1, rimane comunque una regola pratica in Intuition 1.2.

Errata corrige

Nella 6ª puntata c'è una inesattezza. L'ultimo paragrafo prima della conclusione avvertiva che il programmino d'esempio non doveva essere lanciato da CLI in quanto non verificava l'esistenza dello schermo del WorkBench. In realtà questo non è esatto, in quanto una finestra CLI «poggia» comunque su uno schermo WorkBench, anche se non è stata lanciata l'applicazione associata che gestisce menu ed icone attraverso il comando **LoadWB**. Inoltre, essendo il Workbench uno schermo standard, come spiegato nel presente articolo, esso viene automaticamente aperto da Intuition se vi si apre una finestra. Ci scusiamo con i lettori dell'imprecisione.

```
/* ------------------------------------------------------------------------ */
/* - Esercizio #7 -- Dario de Judicibus -- 15 Ottobre 1988 -- Esercizio #7 - */
/* ------------------------------------------------------------------------ */

#include "exec/types.h"
#include "intuition/intuition.h"
#include "proto/intuition.h"

struct IntuitionBase *IntuitionBase;
struct NewScreen ns = /* struttura di definizione dello schermo 100x100x2p. */
{
  0,0,100,100,2,0,1,HIRES,CUSTOMSCREEN,NULL,"Prova",NULL,NULL
};

struct NewWindow nw = /* struttura di definizione della finestra 60x60.     */
{
  20,20,60,60,0,1,CLOSEWINDOW,
  WINDOWCLOSE|SMART_REFRESH|ACTIVATE|WINDOWDRAG|NOCAREREFRESH,
  NULL,NULL,"Finestra",NULL,NULL,0,0,0,0,CUSTOMSCREEN
};

#define INTLIB "intuition.library"
#define INTVER 33L

main()
{
  struct Screen *s;
  struct Window *w;

 /* --- apri la libreria di Intuition ------------------------------------- */
  IntuitionBase = (struct IntuitionBase *)OpenLibrary(INTLIB,INTVER);
  if (IntuitionBase == NULL) exit(FALSE);

 /* --- apri lo schermo 100x100 (2 piani) --------------------------------- */
  if ((s = (struct Screen *)OpenScreen(&ns)) == NULL) exit(FALSE);
  nw.Screen = s;

 /* --- apri la finestra 60x60 con l'angolo sup. sin. a (20,20) ----------- */
  if ((w = (struct Window *)OpenWindow(&nw)) == NULL) exit(FALSE);

 /* --- aspetta un qualunque segnale attraverso IDCMP --------------------- */
  Wait(1<<w->UserPort->mp_SigBit);

 /* --- chiudi tutto ed esci ---------------------------------------------- */
  CloseWindow(w);
  CloseScreen(s);
  exit(TRUE);
}
```

Figura 1 - Soluzione all'esercizio della 6ª puntata.

Una eccezione (solo 1.2) potrebbe essere quella in cui si hanno tre o più schermi che dividano a fasce la dimensione verticale.

Introduzione

Nelle due puntate precedenti abbiamo parlato di schermi e finestre. Abbiamo visto le strutture che ci permettono di definire uno schermo od una finestra, spiegando in dettaglio alcuni campi o rimandando in seguito la spiegazione di quelli più complessi. Abbiamo altresì visto come aprire e chiudere sia schermi che finestre, anche se, per il momento, non sappiamo ancora come gestire l'interfaccia con l'utente (IDCMP), le operazioni di scrittura e lettura, e quelle di tipo grafico. Abbiamo inoltre accennato come Intuition 1.2 metta a disposizione alcuni campi per una gestione più coerente delle dimensioni di schermi e finestre nei due standard PAL ed NTSC. In questa puntata analizzeremo:

— schermi standard ed utente;
— le differenze tra Intuition 1.1 ed 1.2 per quello che riguarda la gestione di schermi e finestre;
— alcune utili funzioni di Intuition.

Schermi standard ed utente

Esistono fondamentalmente due tipi di schermi:

1. quelli standard *[standard screens]*, e
2. quelli utente *[custom screens]*.

Gli schermi standard hanno quattro caratteristiche che li differenziano da quelli utente.

1. Uno schermo standard è condivisibile da più applicazioni che, nei limiti pratici imposti dal sistema (memoria), possono aprirvi quante finestre vogliono. Intuition gestisce direttamente tali schermi che occupano sempre le dimensioni massime disponibili sullo schermo fisico.

2. Se tutte le finestre di uno schermo standard vengono chiuse, allora anche lo schermo viene *automaticamente* chiuso. L'unico schermo standard che si comporta diversamente, a riguardo, è proprio quello del WorkBench, il quale rimane aperto anche se tutte le finestre sono chiuse. Tale schermo va considerato quindi come uno schermo base, che viene automaticamente aperto quando tutti gli altri schermi sono chiusi, se non lo è già. Esso infatti deve fornire un *piano di lavoro* per l'utente del sistema.

3. Uno schermo standard non è mai aperto esplicitamente. Basta semplicemente indicarlo nell'apposito campo della finestra da aprire (**NewWindow .Type**) e, quando viene chiamata la funzione **OpenWindow()**, Intuition verifica se lo schermo già esiste e, in caso contrario, lo apre. Viceversa uno schermo utente va aperto esplicitamente utilizzando la funzione **OpenScreen()**.

4. In linea di massima, mentre è sempre possibile modificare anche in modo dinamico le caratteristiche di uno schermo utente (colori, risoluzione, e via dicendo), non si dovrebbero mai cambiare quelle di uno schermo standard, dato che è condivisibile da più applicazioni.

Per il momento l'unico schermo standard disponibile è quello del WorkBench, a cui, come abbiamo già visto, è associato il tipo **WBENCHSCREEN**. Si tratta, come detto sopra, di uno schermo standard privilegiato, che si discosta un po' dalle regole enunciate. In teoria, tuttavia, è possibile che in futuro altri schermi standard facciano la loro comparsa.

Un'altra caratteristica che distingue lo schermo di WorkBench dagli altri schermi standard è quella di poter essere chiuso e poi riaperto da un programma. Tale tecnica è spesso usata da programmi grafici che, necessitando di molta memoria, chiudono il WorkBench per mezzo della funzione **CloseWork Bench()**, in modo da poter riutilizzare la memoria rilasciata dal sistema, salvo poi, prima di terminare, riaprirlo con la funzione **OpenWorkBench()**. Intuition si assume l'onere di «ricordare» eventuali finestre aperte e di far ritrovare all'utente lo schermo di WorkBench così come lo aveva lasciato. Dato inoltre che il WorkBench è qualcosa di più di un normale schermo standard, come avrete già capito, ma è una vera e propria applicazione, con icone, menu, e così via, esso ha altri privilegi, come

```
struct GfxBase
{
    struct Library  LibNode;
    struct View     *ActiView;
    struct copinit  *copinit;
    long    *cia;
    long    *blitter;
    UWORD   *LOFlist;
    UWORD   *SHFlist;
    struct bltnode *blthd,*bltl;
    struct bltnode *bsblthd,*bsbltl;
    struct Interrupt vbsrv,timsrv,bltsrv;
    struct List     TextFonts;
    struct TextFont *DefaultFont;
    UWORD  Modes;
    BYTE   VBlank;
    BYTE   Debug;
    SHORT  BeamSync;
    SHORT  system_bplcon0;
    UBYTE  SpriteReserved;
    UBYTE  bytereserved;
    USHORT Flags;
    SHORT  BlitLock;
    short  BlitNest;
    struct List    BlitWaitQ;
    struct Task    *BlitOwner;
    struct List    TOF_WaitQ;

 /* ------------------------------------------------------------------ *\
 -*  - 1 - Questo è un campo che ci interessa                          *-
 \* ------------------------------------------------------------------ */
    UWORD   DisplayFlags;
 /* ------------------------------------------------------------------ */

    struct SimpleSprite **SimpleSprites;
    UWORD   MaxDisplayRow;          /* NON USARE -- informazioni interne */
    UWORD   MaxDisplayColumn;       /* NON USARE -- informazioni interne */

 /* ------------------------------------------------------------------ *\
 -*  - 2 - Questi sono altri due campi che ci interessano              *-
 \* ------------------------------------------------------------------ */
    UWORD   NormalDisplayRows;
    UWORD   NormalDisplayColumns;
 /* ------------------------------------------------------------------ */

    UWORD   NormalDPMX;
    UWORD   NormalDPMY;
    struct  SignalSemaphore *LastChanceMemory;
    UWORD   *LCMptr;
    UWORD   MicrosPerLine;
    UWORD   MinDisplayColumn;
    ULONG   reserved[2];
    ULONG   hedley[4];
    ULONG   hedley2[4];
    SHORT   hedley_count;
    USHORT  hedley_flags;
    SHORT   hedley_count1;
    SHORT   hedley_tmp;
    ULONG   hedley3[3];
    ULONG   hedley_sprites[4] ;    /* Puntatore allo sprite del mouse */
    ULONG   hedley_sprites1[4] ;   /* Puntatore allo sprite del mouse */
};

 /* ------------------------------------------------------------------ *\
 -*  - 3 - Queste sono tre costanti che ci interessano                 *-
 \* ------------------------------------------------------------------ */
    #define NTSC    1
    #define GENLOC  2
    #define PAL     4
 /* ------------------------------------------------------------------ */
```

Figura 2 - Struttura GfxBase.

```
#include "exec/types.h"
#include "graphics/gfx.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"

struct GfxBase *GfxBase;

#define GFXLIB "graphics.library"
#define GFXVER 33L

main()
{

  if ((GfxBase = (struct GfxBase *)OpenLibrary(GFXLIB,GFXVER)) == NULL)
    exit(FALSE);

  if (GfxBase->DisplayFlags & PAL)      ;   /* Amiga con standard PAL        */
  else                                  ;   /* Amiga con standard NTSC       */

  if (GfxBase->DisplayFlags & GENLOC)   ;   /* Amiga con attaccato un GenLock */
  else                                  ;   /* Amiga senza GenLock           */

  linee   = GfxBase->NormalDisplayRows;     /* Numero massimo di linee       */
  colonne = GfxBase->NormalDisplayColumns; /* Numero massimo di colonne      */

  CloseLibrary(GfxBase);

}
```

Figura 3 - Esempio di utilizzo di alcuni campi di GfxBase.

```
Prototipo: void ShowTitle(struct Screen *,long)

Utilizzo:  ShowTitle(schermo,mostra);

           schermo      puntatore ad uno schermo
           mostra       se TRUE il titolo va sopra le finestre di fondo
                        se FALSE il titolo va sotto a tutte le finestre
-------------------------------------------------------------------------
Prototipo: void MoveScreen(struct Screen *,long,long)

Utilizzo:  MoveScreen(schermo,DeltaX,DeltaY);

           schermo      puntatore ad uno schermo
           DeltaX       ignorato
           DeltaY       numero di linee di cui abbassare od alzare lo schermo
-------------------------------------------------------------------------
Prototipo: void ScreenToBack(struct Screen *)

Utilizzo:  ScreenToBack(schermo);

           schermo      puntatore ad uno schermo
-------------------------------------------------------------------------
Prototipo: void ScreenToFront(struct Screen *)

Utilizzo:  ScreenToFront(schermo);

           schermo      puntatore ad uno schermo
-------------------------------------------------------------------------
Prototipo: long WBenchToBack(void)

Utilizzo:  fatto = WBenchToBack();

           fatto        TRUE se lo schermo del WorkBench era aperto
                        FALSE se lo schermo del WorkBench era chiuso
-------------------------------------------------------------------------
Prototipo: long WBenchToFront(struct Screen *)

Utilizzo:  fatto = WBenchToFront(schermo);

           fatto        TRUE se lo schermo del WorkBench era aperto
                        FALSE se lo schermo del WorkBench era chiuso
-------------------------------------------------------------------------
```

Figura 4 - Funzioni che operano sugli schermi.

quello di poter essere personalizzato dall'utente tramite il comando **Preferences**.

Per quello che riguarda gli schermi utente, invece, è opportuno spendere due parole su come vengono gestite le aree grafiche in memoria.

Innanzitutto, come già accennato nella scorsa puntata, gli schermi sono basati su una struttura chiamata **ViewPort**. Una o più **ViewPort** formano una **View**, che quindi descrive in modo esaustivo la schermata che compare sul vostro monitor. Non entreremo nei dettagli tecnici dato che nella maggior parte dei casi i puntatori a queste ed altre strutture saranno utilizzati solo come identificatori delle aree grafiche su cui si vuole operare, senza peraltro sapere cosa realmente contengono (come abbiamo fatto con i File Handle dell'AmigaDOS). Ad ogni **ViewPort** è associata un'area grafica detta *raster*, rappresentata da una struttura chiamata **BitMap**. Tale area può raggiungere al massimo le dimensioni di 1024x1024 pixel. Mentre la struttura **BitMap** descrive l'area grafica su cui si vuole operare, la struttura **RastPort** contiene informazioni relative al modo con cui le routine grafi-

che dovranno operare su tale area. Vedremo infatti che, ogni qual volta chiameremo una routine grafica per disegnare una figura, cambiare il colore della penna, scrivere del testo, e così via, dovremo anche specificare il puntatore alla struttura **RastPort** associata all'area su cui stiamo lavorando.

Sia gli schermi che le finestre hanno quindi associata una struttura **RastPort**. Questo vuol dire che possiamo scrivere o disegnare sia in una finestra, sia direttamente in uno schermo. Quest'ultima possibilità richiede tuttavia una buona conoscenza sia di Intuition che delle strutture grafiche menzionate. Mentre infatti Intuition è in grado, se richiesto, di gestire tutti i possibili problemi che possono sorgere dal sovrapporsi di più finestre, dal fatto che una finestra può essere spostata o se ne possono cambiare le dimensioni, e dalla sovrapposizione di menu e finestre, le aree grafiche relative allo schermo stesso devono sempre essere gestite dall'utente. Di fatto si preferisce utilizzare un particolare tipo di finestra, che vedremo in seguito, chiamata **BackDrop**, qualora si desideri avere un'area grafica a tutto schermo.

A questo punto è chiaro cosa è uno schermo «CustomBitMap». È uno schermo la cui area di lavoro è fornita dall'utente, e può quindi contenere delle figure predefinite. Ad esempio, utilizzando questa tecnica, è possibile avere uno schermo il cui sfondo contiene una tassellatura *[pattern]* od addirittura delle vere e proprie immagini (anche una vostra fotografia, per i più megalomani!).

Una tecnica molto utilizzata è quella del doppio buffer che permette, grazie a due **BitMap** che si possono alternare sul monitor, di effettuare operazioni grafiche sull'area nascosta per poi fare apparire il risultato finale sullo schermo. In questo modo il disegno non viene «costruito» di fronte all'utente, ma appare all'improvviso già completato. Si tratta indubbiamente di una tecnica avanzata che richiede una certa abilità nella programmazione in C, ed una buona conoscenza delle strutture grafiche base.

Dopo questa breve puntata in profondità, torniamo ad Intuition per evidenziare alcune possibilità fornite dalla versione 1.2.

Intuition 1.2

Chi di voi non ha maledetto almeno una volta il fatto che, essendo la maggior parte dei programmi che girano su Amiga basati sullo standard americano NTSC, ed avendo questi un numero di linee inferiore a quello PAL usato in Europa ed Australia, ci tocca spesso convivere con quella benedetta fascia nera, in fondo allo schermo? Se nelle prime versioni del sistema operativo questa era una caratteristica a cui, volenti o nolenti, bisognava adattarsi, Intuition 1.2 (e seguenti) mette a disposizione del programmatore il modo di scrivere programmi capaci *loro* di adattarsi all'utente americano od europeo, e non viceversa! Che poi molte *software house* continuino ad ignorare tutto ciò è un altro discorso... Sta di fatto che se volete scrivere programmi che utilizzino l'intero schermo fisico in entrambi i sistemi (PAL ed NTSC), ora ne avete la possibilità.

Vediamo dunque come sia possibile dimensionare l'altezza degli schermi in base allo standard usato, sia nel caso che utilizziate una propria BitMap, sia in quello che lasciate che sia Intuition ad occuparsi degli aspetti relativi alla memorizzazione delle aree grafiche.

Se aprite lo schermo normale, basterà specificare l'altezza dello schermo come segue:

```
NewScreen.Height = STDSCREENHEIGHT;
```

come già accennato nella scorsa puntata. Lo schermo si adatterà automaticamente allo standard usato dal vostro Amiga. Se viceversa fornite una vostra BitMap (selezionando quindi **CUSTOM BITMAP**), allora dovrete sapere *a priori* la massima estensione verticale. Per far ciò dovete prima aprire la libreria grafica e poi utilizzare alcune informazioni contenute nella struttura **GfxBase** (vedi figura 2).

Queste sono:

NormalDisplayRows

che contiene il numero massimo di linee utilizzabili in modo non interlacciato (200 o 256);

```
-------------------------------------------------------------------------------
Prototipo: long = WindowLimits(struct Window *,long,long,long,long);

Utilizzo:  fatto = WindowLimits(finestra,MinLarg,MinAlt,MaxLarg,MaxAlt);

           fatto        TRUE se tutto è andato bene, altrimenti FALSE
           finestra     puntatore ad una finestra
           MinLarg      nuova larghezza minima (zero se come quella iniziale)
           MinAlt       nuova altezza minima (zero se come quella iniziale)
           MaxLarg      nuova larghezza massima (zero se come quella iniziale)
           MaxAlt       nuova altezza massima (zero se come quella iniziale)
-------------------------------------------------------------------------------
Prototipo: void SetWindowTitles(struct Window *,char *,char *);

Utilizzo:  SetWindowTitles(finestra,TitFinestra,TitSchermo);

           finestra     puntatore ad una finestra
           TitFinestra  puntatore ad una stringa, 0 (nessun titolo) o -1
                        (nessuna modifica)
           TitSchermo   puntatore ad una stringa, 0 (nessun titolo) o -1
                        (nessuna modifica)
-------------------------------------------------------------------------------
Prototipo: void MoveWindow(struct Window *,long,long);

Utilizzo:  MoveWindow(finestra,DeltaX,DeltaY);

           finestra     puntatore ad una finestra
           DeltaX       spostamento orizzontale in pixel
           DeltaY       spostamento verticale in pixel
-------------------------------------------------------------------------------
Prototipo: void SizeWindow(struct Window *,long,long);

Utilizzo:  SizeWindow(finestra,DeltaX,DeltaY);

           finestra     puntatore ad una finestra
           DeltaX       incremento orizzontale in pixel
           DeltaY       incremento verticale in pixel
-------------------------------------------------------------------------------
Prototipo: void WindowToBack(struct Window *);

Utilizzo:  WindowToBack(finestra);

           finestra     puntatore ad una finestra
-------------------------------------------------------------------------------
Prototipo: void WindowToFront(struct Window *);

Utilizzo:  WindowToFront(finestra);

           finestra     puntatore ad una finestra
-------------------------------------------------------------------------------
```

Figura 5 - Funzioni che operano sulle finestre.

NormalDisplayColumns
che contiene il numero massimo di colonne utilizzabili in bassa risoluzione (320);
DisplayFlags
che contiene varie informazioni tra cui lo standard usato. Due costanti predefinite in **graphics/gfxbase.h** possono essere utilizzate per determinare quale: **PAL** ed **NTSC**.

Uno scheletro è riportato in figura 3.

Dimensionare correttamente lo schermo, tuttavia, non è sufficiente al fine di ottenere una completa portabilità dei nostri programmi tra Amiga di nazioni diverse. Senza per ora entrare in considerazioni come quelle relative alle varie tastiere nazionali, ma limitandoci ai problemi con Intuition, dobbiamo dimensionare correttamente anche finestre, gadget, menu.

Per quello che riguarda le finestre, ce ne sono di due tipi:
1. quelle a dimensione fissa, e
2. quelle a dimensione variabile.

Il primo tipo è relativamente semplice da trattare. Se la dimensione verticale della finestra deve essere quella massima dello schermo, od un certo delta rispetto a quest'ultima, basterà usare il valore in **NormalDisplayRows**. Se viceversa deve essere un valore fisso indipendentemente dalle dimensioni massime dello schermo, basterà utilizzare un valore inferiore a 200. In quest'ultimo caso, infatti, non si può pretendere di riuscire *comunque* a visualizzare una finestra alta, diciamo, 245 pixel su uno schermo NTSC.

Nel secondo caso, oltre alle dimensioni iniziali della finestra (**Height**), bisognerà tener conto anche di quelle massime (**MaxHeight**), dato che l'utente può modificare le dimensioni della finestra per mezzo del gadget di ridimensionamento *[sizing gadget]*. Se l'altezza della finestra è già quella massima, basterà porre a zero **MaxHeight**, altrimenti bisognerà impostarlo allo stesso valore assunto da **NormalDisplayRows**.

Nel caso che la finestra sia stata aperta nello schermo del WorkBench, si possono utilizzare anche le informazioni fornite da una nuova funzione [1.2], cioè **GetScreenData()**.

Vedremo in seguito come comportarci anche per altri oggetti come ad esempio i gadget.

Altre differenze rispetto Intuition 1.1 per quello che riguarda schermi e finestre sono le seguenti:

- Due nuove costanti da usare in **NewScreen.Flags:**

SCREENBEHIND
avverte Intuition che questo schermo va aperto dietro a tutti gli altri; sarà cura del programma decidere quando portarlo di fronte. Questa tecnica è particolarmente utile quando il programma deve preparare alcune immagini sullo schermo, dopo che questo è già stato aperto, per poi visualizzarlo quando presentabile.
SCREENQUIET
avverte Intuition che non deve assolutamente disegnare la barra del titolo e i gadget di sistema quando lo schermo è aperto. Questo serve ad evitare che Intuition copra con questi elementi la zona superiore dello schermo, interferendo così con eventuali operazioni grafiche del programma. I gadget di scorrimento e di fronte-retro in ogni caso, pur invisibili, continuano a funzionare regolarmente. Questo segnalatore non disabilita viceversa la presentazione dei menu a discesa, che devono comunque essere intercettati dal programma (vedremo come quando parleremo di menu) per evitare che interferiscano con le sue operazioni grafiche. Questo genere di problemi, che analizzeremo in dettaglio in seguito, non riguardano soltanto la possibile cancellazione di aree già disegnate dal programma, ma anche possibili interferenze con operazioni grafiche *in corso*.

Di fatto i rischi sono due: il primo è quello di coprire, e quindi cancellare, immagini disegnate dal programma *direttamente* sullo schermo; il secondo è quello di disegnare sopra gli elementi gestiti da Intuition, cioè i menu, la barra del titolo ed i gadget di sistema.

Nel caso delle finestre, invece, il problema può essere risolto scegliendo una opportuna tecnica di restauro *[refresh]* del *raster* della finestra.

- Una nuova funzione per attivare una finestra, particolarmente utile per quelle applicazioni che aprono più di una finestra per operazioni di ingresso ed uscita *[I/O]*.

Il prototipo (vedi nota 1) è il seguente:

```
void ActivateWindow( struct Window * );
```

Questa funzione è utilizzata, ad esempio, dal WorkBench quando richiamate l'operazione Rename da menu. Quella lunga stringa alta un carattere e larga quanto lo schermo, altro non è che una finestra contenente un gadget di stringa, ed attivata automaticamente dalla funzione in questione.

- Una nuova funzione per restaurare i bordi di una finestra nel caso che il vostro programma vi abbia disegnato sopra.

Il prototipo è:

```
void RefreshWindowFrame (struct Window
*);
```

Altre novità della versione 1.2 saranno analizzate in seguito, essendo legate ad argomenti non ancora trattati.

Spesso useremo la notazione "[1.2]" per indicare funzionalità tipiche di questa versione.

Analogamente, conto al più presto di introdurre la stessa notazione anche per quello che riguarda la più recente versione 1.3.

Note

1. Ricordo che il prototipo di una funzione descrive la funzione stessa relativamente al tipo di parametri passati e a quello di un eventuale valore di ritorno. Se quest'ultimo non è previsto, la funzione è definita di tipo *void*.

In genere i prototipi sono posizionati all'inizio del codice sorgente, od in un eventuale file di inclusione posto nella directory **proto**.

2. In seguito useremo la seguente terminologia italiana per definire le operazioni che un utente può effettuare su di una finestra o su uno schermo:

spostamento - operazione effettuata per mezzo della barra di spostamento *[drag bar]* e che consente di spostare una finestra in uno schermo, o di muovere verticalmente lo stesso schermo (scorrimento);

ridimensionamento - operazione effettuata per mezzo del gadget di ridimensionamento *[sizing gadget]* e che permette di variare le dimensioni di una finestra;

retro-fronte - operazione effettuata per mezzo del gadget di retro-fronte *[front gadget]* e che permette di spostare una finestra normale (od uno schermo) di fronte a tutte le altre;

fronte-retro - operazione effettuata per mezzo di fronte-retro *[back gadget]* e che permette di spostare una finestra normale (od uno schermo) dietro a tutte le altre;

chiusura - operazione effettuata per mezzo del gadget di chiusura *[close gadget]* e che permette di chiudere una finestra.

Le due operazioni di fronte-retro e retro-fronte saranno anche dette «operazioni di profondità».

3. Ricordo che uno schermo può avere due titoli: uno è quello associato allo schermo stesso *[default title]*, l'altro dipende da quale finestra è attiva *[current title]*. Quando uno schermo viene aperto, i due titoli coincidono. Quando una finestra è attivata, tuttavia, Intuition controlla se per caso il programmatore ha associato a quella finestra un titolo corrente per lo schermo a cui essa appartiene (per mezzo della funzione **SetWindowTitles()** appunto). Se così è, il titolo di default viene sostituito da quello corrente fintanto che quella finestra resta attiva. La prossima volta che lanciate una applicazione che apre finestre sullo schermo del WorkBench, osservate con attenzione la barra che di solito contiene il numero della versione del WorkBench e la memoria disponibile. Prima o poi vi capiterà di osservare il titolo cambiare all'attivazione di una delle finestre menzionate (è il caso di FACC, ad esempio).

```
/*  ------------------------------------------------------------------- *
 *  Scheletro per l'utilizzo delle funzioni di Intuition descritte nel testo *
 *  >>>  Notare che non è stato riportato tutto il codice dell'esempio!  <<< *
 *  ------------------------------------------------------------------- */

.
/*
 *  Apri una finestra specificando che si vogliono ricevere SOLO eventi
 *  corrispondenti alla pressione su di uno qualunque dei tasti del mouse.
 */
NewWindow.Flags = RMBTRAP|SMART_REFRESH|ACTIVATE|NOCAREREFRESH;
NewWindow.IDCMPFlags = MOUSEBUTTONS;
if ((w = (struct Window *)OpenWindow(&nw)) == NULL) exit(FALSE);

.
/*
 *  Mettiti in attesa sulla porta IDCMP, in modo da svegliarti ogni volta
 *  che l'utente preme un tasto del mouse. Appena svegliato esegui una
 *  qualunque delle funzioni descritte (con un minimo di criterio, tuttavia!)
 */
Wait(1<<w->UserPort->mp_SigBit);
Funzione(parametri);

/*
 *  Ripeti il blocco di sopra quante volte vuoi, con funzioni differenti.
 */
Wait(1<<w->UserPort->mp_SigBit);
Funzione(parametri);

.
Wait(1<<w->UserPort->mp_SigBit);
Funzione(parametri);

/*
 *  OK, questa è l'ultima volta, poi chiudi. Non è necessario quindi il
 *  gadget di chiusura.
 */
Wait(1<<w->UserPort->mp_SigBit);
CloseWindow(w);

.
```

Figura 6 - Scheletro per gli esercizi.

Alcune utili funzioni di Intuition

Vediamo ora alcune delle molte funzioni che Intuition ci mette a disposizione.

Funzioni per gli schermi

È possibile operare su uno schermo e su alcune sue caratteristiche per mezzo delle seguenti funzioni di Intuition i cui prototipi sono riportati in figura 4:

ShowTitle()
Questa funzione serve a specificare se la barra del titolo di uno schermo debba essere posizionata sopra o sotto ad un'eventuale finestra di fondo *[backdrop window]*. Come spiegheremo nella prossima puntata, una finestra di fondo ha la caratteristica di essere *agganciata* allo schermo a cui appartiene, in modo tale che qualunque operazione di profondità (vedi nota 2) sulle finestre normali non modifichi comunque questa posizione privilegiata. Mentre la barra di scorrimento di uno schermo viene sempre occultata da una normale finestra che vi si sovrapponga, una finestra di fondo si posiziona sempre dietro ad essa. Per evitare che operazioni grafiche su tale finestra cancellino anche parzialmente il titolo dello schermo, è quindi necessario usare **ShowTitle()** specificando **FALSE** come secondo parametro.

MoveScreen()
Questa funzione serve a spostare verticalmente uno schermo di un certo numero di linee specificate. Il secondo parametro (vedi figura 4) non è usato, non potendosi spostare gli schermi orizzontalmente.

ScreenToBack()
Sposta uno schermo dietro a tutti gli altri (fronte-retro).

ScreenToFront()
Sposta uno schermo di fronte a tutti gli altri (retro-fronte).

WBenchToBack()
Sposta lo schermo del WorkBench dietro a tutti gli altri (fronte-retro). Funziona solo se lo schermo del WorkBench è aperto, cioè non apre automaticamente tale schermo qualora sia stato chiuso.

WBenchToFront()
Sposta lo schermo del WorkBench di fronte a tutti gli altri (retro-fronte). Funziona solo se lo schermo del WorkBench è aperto, cioè non apre automaticamente tale schermo qualora sia stato chiuso.

Esistono altre funzioni che richiedono una maggiore conoscenza degli elementi grafici che stanno alla base di Intuition, e vengono quindi utilizzate solo da programmatori esperti. Dato che c'è ancora molto da dire sulle funzionalità più elementari sia di Intuition che dell'Amiga in generale, non entreremo ulteriormente nel dettaglio.

Un'ultima cosa su **CloseScreen()**: se chiudete uno schermo senza aver prima chiuso tutte le sue finestre, e poi andate a chiuderne una, il sistema cade *[crash]*. Attenti quindi a rispettare il giusto ordine di chiusura.

Funzioni per le finestre

È possibile operare su una finestra e su alcune sue caratteristiche per mezzo delle seguenti funzioni di Intuition i cui prototipi sono riportati in figura 5:

WindowLimits()
Questa funzione permette di modificare sia i limiti massimi che quelli minimi di una finestra. Qualora per uno o più limiti si intenda lasciare il valore originale, basterà porre a zero il campo relativo.

SetWindowTitles()
Questa funzione cambia il titolo di una finestra, il titolo che ha lo schermo quando questa finestra è attiva (vedi nota 3), oppure entrambi.

MoveWindow()
Questa funzione muove una finestra dell'ammontare specificato, sia verticalmente che orizzontalmente.

SizeWindow()
Questa funzione modifica le dimensioni di una finestra.

WindowToBack()
Sposta una finestra dietro a tutte le altre nello stesso schermo (fronte-retro).

WindowToFront()
Sposta una finestra davanti a tutte le altre nello stesso schermo (retro-fronte).

Esistono altre funzioni che operano sulle finestre o su oggetti collegati ad esse (menu, gadget, requester). Alcune le vedremo quando tratteremo esplicitamente tali oggetti, altre richiedono una profonda conoscenza degli elementi grafici che stanno alla base di Intuition. Vale per quest'ultime lo stesso discorso già fatto per le funzioni che operano sugli schermi.

Conclusione

Bene, anche per questa volta è tutto. Avete abbastanza materiale per sbizzarrirvi. Vi suggerisco di provare a creare, aprire e chiudere schermi e finestre di vari colori, dimensioni e risoluzione, in modo da acquisire una certa dimestichezza con i parametri descritti in questa e nelle due puntate precedenti. Provate anche alcune delle funzioni descritte poco fa usando la stessa tecnica usata nel programmino di esempio in figura 1 eseguendo lo scheletro riportato in figura 6.

La prossima volta entreremo più nel dettaglio nella gestione delle finestre. Alla prossima puntata allora, e buon divertimento! MC