a cura di Andrea de Prisco

Questo mese il software dei lettori è davvero speciale. Non si tratta infatti del solito programma preconfezionato, ma di un insieme di routine, consigli e trucchetti assai preziosi per chi vorrà sfruttare maggiormente alcune capacità nascoste di Amiga. Sul prossimo numero, invece...., beh! meglio non fare anticipazioni...

Grafica evoluta con l'Amiga

di Maurizio Mangrella - Eboli (SA)

Premessa

Le possibilità grafiche che l'Amiga è in grado di offrire vanno ben oltre le prestazioni del Basic, che pure conosce, su questa macchina, una implementazione di buon livello. Ne è testimonianza il gran numero di utenti che, per programmare questo computer nel cuore delle sue feature, passa dal buon vecchio Basic ad un linguaggio di livello minore e, quindi, più vicino all'architettura della macchina: il C e il Linguaggio Macchina, tanto per fare un esempio.

Il mio lavoro

Il sottoscritto, in maniera inusuale, ha percorso la via inversa: ha cominciato con il Basic, è passato al C per disperazione ed è ritornato al Basic con dolori — spirituali, s'intende — ancora maggiori...

Mi spiego: tutto quello — e non è poco — che si può fare in Basic è ovviamente, possibile anche da C, ma con la necessità di molto lavoro in più e di molta documentazione: a questo punto il bivio — o comprare tutti i manuali dell'Amiga o tornare al Basic ed accontentarsi — era chiaro.

Sinceramente parlando, non avevo nemmeno la minima intenzione di investire il fior fiore del mio (poco) denaro e delle mie arrabbiature per acquistare all'estero i famigerati manuali della Bantam Books, dunque sarei stato ben presto costretto a scegliere la seconda via, cioè quella di tornare al Basic.

Ma c'era qualcosa che mi faceva sperare in un futuro migliore: la relativamente facile espandibilità dell'interprete, adoperando le famigerate «librerie esterne». Dunque, cominciai ad esaminare i vari files.bmap presenti su Extras e da lì cominciai a capire che quello che avevo intenzione di fare da C era possibile anche da Basic — cioè il viceversa di quanto avevo detto prima — naturalmente con molta più facilità. Una notte — insonne — mi venne il colpo di genio!

I nomi che avevo visto

... nei sorgenti in C di alcuni programmi public domain che mi ero faticosamente procurato li avevo già visti da qualche altra parte, e, precisamente, nei suddetti files.bmap.

Il giorno dopo, tra un verso e l'altro della Divina Commedia, buttai giù un dump di graphics.bmap, e trovai la maggior parte delle istruzioni grafiche che avevo scovato nei sorgenti. Nella figura 1 cito le più importanti, senza approfondire troppo il discorso altrimenti raggiungiamo la decina di pagine.

Due parole su come utilizzare queste routine. Innanzitutto, va precisato che ogni operazione di disegno viene effettuata considerando il foreground color e il background color: cioè, se ho settato il foreground a 2, ogni punto, linea o rettangolo che andrò a disegnare sarà disegnato con il colore 2. Analogo discorso vale per il background color, che equivale al secondo parametro dell'istruzione COLOR del Basic.

rp& è il puntatore al RastPort Record, quella struttura dati che definisce l'allocazione dei piani di bit nella memoria, la loro dimensione e parametri di altra natura. Esiste un RastPort Record per ogni finestra, che nel caso di una window aperta dal Basic, è sistemato a partire dall'indirizzo WINDOW(8) — detto tra di noi, questo indirizzo è ottenibile anche con

```
PEEKL (WINDOW(7)+50)
```

Esiste anche un RastPort Record, del

```
SetDrMd (rp&,mode)          Setta il modo di scrittura e disegno :

                            0 = JAM1 (il background viene eliminato)
                            1 = JAM2
                            2 = COMPLEMENT (inverte l'area)
                            4 = INVERSVID (in reverse)
                            Per capire come funzionano, provate con
                            l'IconEd : li troverete tutti ...

Move (rp&,x,y)              Sposta il Pixel Cursor - antidiluviano ricor -
                            do ... - a (x,y)

Draw (rp&,x,y)              Disegna una linea dal Pixel Cursor a (x,y)

RectFill (rp&,x1,y1,x2,y2)  Disegna un rettangolo da (x1,y1) a (x2,y2)

WritePixel (rp&,x,y)        Disegna un pixel a (x,y)

ReadPixel (rp&,x,y)         Ritorna il colore logico - cioe' la rappre -
                            sentazione nella bitmap - del pixel a (x,y),
                            dunque va dichiarata con DECLARE FUNCTION prima
                            di essere usata

DrawEllipse (rp&,x,y,rx,ry) Disegna un'ellisse vuota con centro in (x,y)
                            e raggi rx (orizzontale) e ry (verticale)

AreaEllipse (rp&,x,y,rx,ry) Come sopra, ma riempie l'interno

SetAPen (rp&,c)             Setta il foreground color a c

SetBPen (rp&,c)             Setta il background color a c

SetRGB4 (vp&,c,r,g,b)       Setta il colore alle componenti r, g e b
                            (che vanno da 0 a 15)

LoadRGB4 (vp&,addr&,ncol)   Carica la ColorMap - le componenti dei colori
                            precostituiti - a partire dall'indirizzo
                            addr& nella ViewPort : il numero dei colori
                            e' ncol .
```

Figura 1

tutto analogo, per ogni schermo: per ottenerlo basta dare

```
rp& = 84 + scr&
```

dove scr& è l'indirizzo dello Screen Record, ottenibile con

```
scr& = PEEKL (WINDOW(7)+46)
```

che fornisce l'indirizzo dello Screen Record dello schermo che contiene la Current Output Window — che confusione!

Dunque, a seconda delle vostre esigenze, potrete lavorare all'interno di una finestra o a tutto schermo — ve lo assicuro, quest'ultimo modo di fare grafica è veramente un sottile piacere per chi, come me, viene dal mondo degli smanettomani ad 8 bit e, quando vede 320 o 640, pensa allo spazio realmente utilizzabile e non ad un indice della risoluzione adoperata.

Come dimostrazione dell'uso di alcune delle routine descritte a tutto schermo, troverete, sul dischetto allegato, DK, un programmino che lentamente vi distruggerà lo schermo del WorkBench...

Per chi non si accontenta...

... c'è un'altra, grande novità. Da Basic è possibile aprire uno schermo in HAM o in HalfBrite o in Dual Playfield, come più vi aggrada.

Per fare ciò, dovrete prima creare — se non lo avete già fatto — la intuition.library utilizzando il ConvertFD del dischetto Extras. Fatto ciò — dovrebbe risultare il file intuition.bmap — potete iniziare a lavorare.

Il listato di figura 1 — che ho dattiloscritto essendo al momento sprovvisto di stampante — apre uno schermo in Hold&Modify e vi disegna sopra alcuni rettangoli con sfumature di Rosso, Verde e Blu.

Per aprire uno schermo è necessario la routine OpenScreen& della intuitition-.library, che, a sua volta necessita di una piccola struttura dati la cui conformazione è rappresentata in figura 2.

L'ascissa e l'ordinata dell'angolo superiore sinistro specificano la posizione dello schermo rispetto al WorkBench Screen: specificando (0,0) si ha la completa coincidenza col suddetto schermo.

Il foreground color e il background color del titolo servono a specificare le nostre preferenze sul titolo dello schermo: il Basic, tanto per citare un esempio, specifica sempre, rispettivamente, 0 e 1.

Il ViewPort Mode dello schermo risponde alle specifiche della tabella di figura 3.

Formato	Contenuto
WORD	Ascissa dell'angolo superiore sinistro
WORD	Ordinata dell'angolo superiore sinistro
WORD	Larghezza dello schermo
WORD	Altezza dello schermo
WORD	Numero di bitplanes
BYTE	Foreground color del titolo deloo schermo
BYTE	Background color del titolo dello schermo
WORD	ViewPort Mode dello schermo
WORD	Tipo di schermo
LONG	Indirizzo della struttura TextAttr
LONG	Indirizzo del titolo dello schermo
LONG	Puntatore ad altri eventuali gadgets

Figura 2

Modo	ViewPort Mode (hex.)	ViewPort Mode (dec.)
LORES (320x200)	0000	0
HIRES (640x200)	C000	-16384
HAM	0800	2048
EXTRA_HALFBRITE	0080	128
DUALPF	0400	1024

Figura 3

Il «-16384» si spiega tenendo conto che in Basic tutti gli interi short hanno un segno, dunque, per la cronaca, -16384 corrisponde a 49152...

Il tipo di schermo indirizza Intuition sull'uso che di questo schermo dovrà fare: noi utenti specificheremo CUSTOMSCREEN, cioè 15 dec.

La struct TextAttr è un'altra piccola struttura, così composta:

Formato	Contenuto
LONG	Indirizzo del nome del font
WORD	Altezza del font
WORD	Flags
WORD	Preferences

Il nome del font è, appunto, il nome del font che si vuole utilizzare nel titolo in testa allo schermo: per non avere fastidi, e se non avete particolari preferenze, settate «topaz.font», cioè il font di sistema.

L'altezza è il numero di pixel verticali di cui è composto ogni carattere del font: settate 8 per il topaz di sistema.

Flags e Preferences, infine, non so precisamente come vadano usati: suppongo riguardino in particolare il corsivo, il grassetto e il sottolineato. Se non avete problemi, settateli entrambi a 0.

Il puntatore ad altri gadget punta alla struct Gadget, che punta ad un'altra struct Gadget... e così via, fino alla fine dei gadget. Settate 0 se non specificate gadget aggiuntivi.

Due note alle strutture ora viste: 1) ogni stringa di caratteri, comunque sia composta, deve terminare con un CHR$(0), che è un carattere riservato dei compilatori C; 2) una WORD equivale a due byte, una LONG-word a quattro byte (32 bit).

Magicamente, passeremo l'indirizzo della struttura precedente — quella più lunga per intenderci — ad OpenScreen della intuition.library, ed otterremo il tanto agognato schermo...! In pratica, la sintassi è questa:

```
sScreen& = OpenScreen& (indirizzo della struttura, cioè LONG)
```

dove «indirizzo della struttura» è una LONG-word che punta al primo byte della struttura stessa. OpenScreen& ritorna un valore — che esamineremo dopo —, dunque va dichiarata prima di essere adoperata.

Il valore ritornato da OpenScreen non è altro che un puntatore allo Scrren Record dello schermo in questione: questa struttura dati ingloba al suo in-

terno altre due strutture, la ViewPort e la RastPort.

La ViewPort contiene dati sulla visualizzazione dello schermo: colori, modo grafico, centratura, etc. Il modo grafico è dato da

PEEKW (vp& + 32)

secondo il seguente codice hardware:

Modo	Codice (dec)
LORES	16384
HIRES	-16384
HAM	18432
EXTRA_HALFBRITE	16512
DUALPF	17408

Ah , dimenticavo, a tutti questi valori — come ai ViewPort Modes visti prima — dovete aggiungere 4 per avere l'Interlace.

vp& è l'indirizzo della ViewPort, ottenibile al secolo con

sScreen& + 44

Se vi interessa sapere dove sono situati i colori dello schermo, sappiate che a partire dall'indirizzo

ColorTab& = PEEKL (PEEKL (vp&+4)+4)

```
Formato          Contenuto

WORD             Ascissa dell'angolo superiore sinistro
WORD             Ordinata dell'angolo superiore sinistro
WORD             Larghezza della finestra
WORD             Altezza della finestra
BYTE             Foreground color del titolo
BYTE             Background color del titolo
LONG             IDCMP Flags
LONG             Intuition Flags
LONG             Puntatore al primo gadget addizionale
LONG             Puntatore al "first user checkmark" (???)
LONG             Puntatore al titolo della finestra
LONG             Puntatore allo Screen Record
LONG             Puntatore alla Super-Bitmap
WORD             Minima dimensione orizzontale
WORD             Minima dimensione verticale
WORD             Massima dimensione orizzontale
WORD             Massima dimensione verticale
WORD             Tipo di schermo
```

Figura 4

è sistemata una sfilza di WORD nel formato.

XXXXRRRRGGGGBBBB X = don't care

dove ogni lettera rappresenta un bit: ad esempio, GGGG sono 4 bit che indicano i 16 possibili livelli del verde.

Il RastPort Record, il cui indirizzo è

rp& = sScreen& + 84

contiene informazioni sull'allocazione dei bitplane in memoria ed altre cosucce del genere; ad esempio, gli indirizzi dei bitplane sono dati da

bPlane&(i) = PEEKL (rp&+4)+8+4*i)
0<=i<=5

Terribile, non è vero?

A questo punto, come abbiamo visto precedentemente, potete lavorare a tutto schermo, il che è una bella comodità. Può risultare utile anche la funzione Text — la metto ora, lo confesso, perché ho dimenticato di inserirla prima... — che vi consentirà di scrivere sullo schermo: la sintassi è

Text (rp&,addr&,nchar)
Scrive i primi nchar caratteri della stringa puntata da addr&

Per decidere dove scrivere, dovete utilizzare la Move, che posizionerà il tasto in un punto corrispondente alla prima colonna di pixel del primo carattere, alla riga — sempre del carattere — indicata dal FontEditor con BaseLine; in pratica, con il topaz 8, la seguente istruzione.

CALL Move (rp&,0,6)
CALL Text (rp&,SADD(«Prova di testo»),14)

scriverà la stringa «Prova di testo» nella banda verticale compresa tra 0 e 7, dal momento che la BaseLine del topaz 8 è posta alla settima riga dall'alto e

7 – 1 = 6

Se non ci avete capito niente — è successo anche a me, a suo tempo — è colpa di chi ha inventato l'Amiga: fate qualche prova e tutto sarà chiaro.

Se volete utilizzare qualche altro font di caratteri, seguite le istruzioni del demo Library, disco Extras, subdirectory BasicDemos.

Finestre, care finestre...

Qualcuno di voi potrebbe desiderare di utilizzare anche la logica a finestre di

Grafica evoluta

```
CLS
DIM font%(4) , scr%(13)                    ' Strutture TextAttr e ScreenData
PRINT "Premere un tasto per uscire"
DECLARE FUNCTION OpenScreen& LIBRARY       ' OpenScreen ritorna un valore
LIBRARY "graphics.library"                 ' Le librerie utilizzate
LIBRARY "intuition.library"
DATA 0,0,320,200,6,0,2048,15,0,0,0,0,0,0
DATA 0,0,8,0,0
RESTORE
FOR k = 0 TO 13 : READ scr%(k) : NEXT      ' Inizializzazione delle strutture
FOR k = 0 TO 4 : READ font%(k) : NEXT
font$ = "topaz.font"+CHR$(0)               ' Nome del font
title$ = "HAM"+CHR$(0)                     ' Titolo dello schermo
font& = SADD(font$)                        ' Puntatore al nome del font
title& = SADD(title$)                      ' Puntatore al titolo dello schermo
font%(0) = PEEKW(VARPTR(font&))            ' Lo sistemiamo nella struttura
font%(1) = PEEKW(VARPTR(font&)+2)          ' TextAttr
font& = VARPTR(font%(0))                   ' Indirizzo di struct TextAttr
scr%(8) = PEEKW(VARPTR(font&))             ' Sistemiamo il puntatore alla strut
tu-
scr%(9) = PEEKW(VARPTR(font&)+2)           ' ra TextAttr
scr%(10) = PEEKW(VARPTR(title&))           ' dunque il puntatore al titolo dell
o
scr%(11) = PEEKW(VARPTR(title&)+2)         ' schermo
sScreen& = OpenScreen&(VARPTR(scr%(0)))    ' Ragazzi , si parte!
vp& = sScreen&+44                          ' La ViewPort
rp& = sScreen&+84                          ' La RastPort
CALL SetRGB4(vp&,0,0,0,0)                  ' Sfondo nero
FOR y = 0 TO 2                             ' Tre file di rettangoli
   y& = 70*y                               ' Posizione verticale
   cbase& = 16*(y+1)                       ' I primi due bits del colore logico
   FOR x = 0 TO 15                         ' 16 sfumature
      x& = 20*x                            ' Posizione orizzontale
      c& = cbase&+x
      CALL SetAPen(rp&,c&)                 ' Colore di disegno
      CALL RectFill(rp&,x&,y&,x&+19,y&+59) ' Il rettangolino
   NEXT x
NEXT y
WHILE INKEY$ = "" : WEND                   ' Prima o poi qualcuno premer* un ta
sto
CALL CloseScreen(sScreen&)                 ' Chiudiamo lo schermo
LIBRARY CLOSE                              ' E le librerie , per benino
CLS : END                                  ' Arrivederci!

' Per ulteriori informazioni , ctr. articolo .
```

Intuition: è possibile anche questo, con un po' di pazienza e la famigerata intuition.library.

Per aprire una finestra, dovete aprire una struttura di dati come appare in figura 4.

Le prime specifiche della finestra sono abbastanza ovvie, se non fosse per la possibilità che ci viene offerta — e che quasi nessuno sfrutta — di scegliere i colori del titolo e, volendo, di tutta la Title Bar della finestra.

Intuition, l'interfaccia utente di Amiga, comunica con i task attivi in un dato momento tramite dei sistemi di trasferimento dati, detti «ports». E così Intuition, che è uno dei tanti processi che l'Amiga può gestire contemporaneamente, riesce ad informare tutti i task attivi sullo stato del sistema, in particolare — è quello che ci interessa — sullo stato delle finestre: infatti ogni finestra contiene, nel suo Window Record — che vedremo fra poco — un puntatore ad uno speciale port, gestito da Intuition. In Basic è assai difficile maneggiare i dati dei port, visto che ciò richiederebbe la maggior parte del tempo macchina disponibile: in C, però, dove la velocità intrinseca dei processi è più elevata, è possibile, ad intervalli regolari, dedicarsi ai port. Gli IDCMP Flag informano Intuition su ciò che Intuition stessa dovrà mettere a disposizione di un dato processo sul port di una data finestra (Vedi figura 5).

Per selezionare una finestra si intende clickare all'interno di essa in modo che la Title Bar sia perfettamente leggibile; al contrario, inattivare significa clickare in un'altra finestra e deselezionare tutte le altre.

Il port di Intuition fornisce due dati, Class e Code: Class vi restituisce il codice dell'IDCMP Flag relativo all'evento accaduto, mentre Code vi informa ancora più efficacemente sulla tipologia dell'evento stesso.

Per specificare più di un flag, è sufficiente sommare i relativi codici.

Gli Intuition Flag (che pubblichiamo in figura 6) sono sicuramente più importanti: essi definiscono il comportamento della finestra.

Per attivare un refresh che si rispetti dovete allocare dei piani di bit e creare una piccola struttura dati che ne contenga l'indirizzo, analoga alla Bitmap degli schermi — non so come questa sia composta: vedete di scoprirlo voi, spulciando in qualche demo... —: dico: «dovete» perché, se non lo fate, una Guru Meditation è sempre in agguato...

Per «cambiare priorità rispetto alle altre finestre» compariranno, nell'angolo superiore destro della finestra, i Back e i Front Gadget: sempre se voi specificate il WINDOWDEPTH.

Al solito, per specificare più di una caratteristica, potete sommare i codici.

Il titolo della finestra, come al solito, deve terminare con un CHR$(0).

Il puntatore allo Screen Record non è altro che lo sScreen& visto prima, cioè il parametro ritornato dalla OpenScreen. Solo in un caso potete non specificarlo: quando la finestra è sul WorkBench Screen; in tal caso, al posto di 15 per lo Screen Type, dovrete specificare 1 (WBENCHSCREEN).

A questo punto, signore e signori, passate l'indirizzo di questa struttura alla routine OpenWindow della intuition.library, e vedrete la tanto desiderata finestra: la sintassi è

sWindow& = OpenWindow&
(indirizzo della struttura, cioè LONG)

sWindow& è analogo alla funzione WINDOW(7) del Basic, e rappresenta il puntatore alla struttura denominata come Window Record, che contiene tutte le informazioni sulla finestra: il puntatore allo schermo è

sScreen& = PEEKL (sWindow&+46)

mentre la RastPort della finestra è data da

rp& =PEEKL (sWindow&+50)

passando quest'ultimo valore alle routine grafiche potremo lavorare nelle finestre come ci pare e piace...

Flag	Codice (hex.)	Intuition vi informa nel caso che ...
SIZEVERIFY	0	voi chiediate le attuali dimensioni della finestra
NEWSIZE	2	le dimensioni della finestra siano cambiate
MOUSEBUTTONS	8	siano stati premuti i bottoni del mouse
MOUSEMOVE	10	il mouse sia stato mosso
CLOSEWINDOW	200	la finestra sia stata chiusa
ACTIVEWINDOW	40000	la finestra sia stata selezionata
INACTIVEWINDOW	80000	la finestra sia stata inattivata

Figura 5

Un momento...

... sento già le grida dei lettori: «Ehi, questo ci ha lasciato con uno schermo e una finestra aperti: come facciamo a chiuderli?».

È molto semplice: per chiudere uno schermo si esegue l'istruzione

CALL CloseScreen (sScreen&)

mentre, per chiudere una finestra, si lancia la

CALL CloseWindow (sWindow&)

Abbiate sempre l'accortezza di chiudere le finestre PRIMA di chiudere gli schermi: la migliore cosa che vi può capitare è quella di avere un bel po' di memoria occupata senza sapere bene perché.

Uso del mouse

Un consiglio: non chiudete ancora le finestre che avete creato, ma lavorateci un po' sopra. Sarà il modo migliore per comprendere quanto detto sinora, e, magari, per scoprire tutto ciò che io, nel mio pur frenetico lavoro, non sono riuscito ad appurare completamente.

È possibile ottenere le coordinate del mouse nell'ambito di una data finestra in questo modo:

x = PEEKW (sWindow&+14)
y = PEEKW (sWindow&+12)

dove x e y possono assumere anche valori negativi — quando la freccina del mouse è fuori della finestra. Le coordinate del mouse, anche se viene specificato il GIMMEZEROZERO, sono riferite SEMPRE all'angolo superiore sinistro della Title Bar: un trucchetto, questo, che sulle prime potrà creare più di un fastidio, ma che poi si rivelerà di un'utilità impagabile.

I modi grafici

In questo paragrafo — il penultimo dell'articolo, spero... — mi occuperò in maniera concisa dei modi grafici speciali dell'Amiga: in fondo, è per ottenerli che abbiamo fatto tutte queste elucubrazioni, non è vero?

HAM, ovvero Hold & Modify

In questo modo grafico — uno dei più famosi e più utili — avremo a disposizione, praticamente senza limitazioni, 16 colori precostituiti, e, con limitazioni di una certa entità, tutti i 4096 colori che l'Amiga è in grado di generare.

In pratica, dato un pixel sullo scher-

Flag	Codice (hex.)	Caratteristica
WINDOWSIZING	1	Cambio delle dimensioni (type 1 del BASIC)
WINDOWDRAG	2	Cambio della posizione (typl 2 del BASIC)
WINDOWDEPTH	4	Cambio della priorita' rispetto alle altre finestre (type 4 del BASIC)
WINDOWCLOSE	8	Chiusura della finestra (type 8 del BASIC)
GIMMEZEROZERO	400	Pone l'origine delle coordinate non nell'angolo superiore sinistro della Title Bar, ma nell'angolo superiore sinistro dello spazio interno alla finestra stessa
BORDERLESS	800	Niente bordi della finestra (solo la Title Bar)
ACTIVATE	1000	Attiva la finestra (specificatelo SEMPRE)
NOCAREREFRESH	20000	Non si preoccupa del refresh della finestra
WINDOWREFRESH	10000	Rinfresca la finestra (type 16 del BASIC)
SUPERBITMAP	80	Specifica l'esistenza di piani di bit riservati alla finestra

Figura 6

mo, il suo colore sarà dettato dalle specifiche riportate in tabella A.

Il colore logico di un pixel è la sua rappresentazione numerica nella bitmap mentre il colore fisico è il reale colore che quel pixel assumerà sullo schermo.

Ad ogni fine di riga, il registro interno di colore viene settato a 0, dunque si riparte dal nero.

EXTRA_HALFBRITE

In questo modo grafico avremo a disposizione 32 colori precostituiti e i loro corrispondenti a luminosità dimezzata (si faccia riferimento alla tabella B).

DUALPF, cioè dual Playfield

Con questo modo grafico avremo due schermi separati, ognuno da 3 bitplane (se abbiamo specificato 6 bitplane). Poco posso dirvi sul comportamento logico dei colori in quanto tutte le sperimentazioni che ho condotto non hanno dato i risultati sperati — leggi: non si capisce niente... Secondo me è proprio ora di comprare qualche manuale della Bantam Books...

LoadILBM

Ed eccoci alla fine dell'articolo (quasi mi dispiace...): voglio parlarvi del più importante tra i programmi del dischetto che ho allegato: LoadIlbm.

Questo programma in Basic è una rielaborazione del famoso LoadILBM — SaveACBM di Carolyn Scheppner: non è più capace di salvare in formato ACBM — ma non è che si perda molto, in fin dei conti — ma può adattarsi a caricare qualunque schermata, in qualunque modo grafico. Quindi, se avete qualche immagine un po' strana, magari presa col Grabbit, potete visualizzarla con questo software.

IL caricamento delle immagini compattate, specie se con molti bitplane, risulta un po' lento: sto lavorando ad una seconda versione, con un decompattatore scritto in Linguaggio Macchina. Sul salvataggio in ACBM non vi prometto nulla.

Ironia della sorte, sul dischetto troverete un programma in C compilato, ViewILBM — è di pubblico dominio — che svolge esattamente la stessa cosa, solo molto più velocemente...

Tabella A

Colore logico	Colore fisico
%00xxxx	Il pixel avrà il colore n. xxxx della palette
%01xxxx	Il pixel eredita rosso e verde dal pixel adiacente alla sinistra e avrà un valore di blu pari a xxxx
%10xxxx	Il pixel eredita verde e blu dal pixel adiacente alla sinistra e setta il rosso a xxxx
%11xxxx	Il pixel eredita rosso e blu dal pixel precedente e setta il verde a xxxx

Tabella B

Colore logico	Colore fisico
%0xxxxx	Il colore xxxxx della palette a luminosità piena
%1xxxxx	Il colore xxxxx della palette a luminosità dimezzata

Conclusioni

Se non ci mettevo un altro paragrafo, non mi sentivo bene...

Il mio lavoro è principalmente indirizzato a quanti intendono lavorare con i modi grafici speciali dell'Amiga nella maniera più semplice possibile, cioè da Basic — come sono belli i sinonimi...

Qualcuno obietterà che una tale potenza è degna solo del C: non ha tutti i torti, specie per quanto riguarda la velocità. Ma bisogna anche sapersi accontentare: non tutti hanno la voglia e la possibilità di comperare un compilatore C e tanti bei manualoni in inglese.

Quando avrò delle buone nuove sull'Amiga, mi risentirete. Ciao.

Font

di Maurizio Lotauro - Bolzano

Il programma che presento vuole essere un esempio sull'uso delle librerie disponibili all'interno dell'Amiga, e contemporaneamente far vedere come usare i font col Basic.

Le librerie dell'Amiga contengono una grande quantità di funzioni con le quali è possibile fare di tutto, come ad esempio chiamate del dos o funzioni grafiche. Per usare una libreria occorre che sia presente un file nomelibreria.bmap nella directory corrente oppure nella SYS:LIBS (SYS: rappresenta il disco da cui si è fatto il bootstrap). Questi file si ottengono partendo dal file nomelibreria_lib.fd che si possono trovare nei dischi dell'Assembler, Lattice e Atzec C. La conversione dal formato .fd a .bmap si ottiene tramite il programma CONVERTIFD che si trova nella directory BASICDEMOS del disco EXTRAS fornito insieme al calcolatore (v. in seguito le modifiche da apportare a tale programma). Qui si trovano già i file dos.bmap e graphics.bmap che vengono utilizzati dai demo.

Per accedere alle funzioni contenute all'interno di una libreria bisogna aprire la libreria con la funzione LIBRARY:

```
LIBRARY "nomelibreria"
```

Ora quando l'interprete Basic troverà una funzione non compresa tra quelle del linguaggio o tra quelle dichiarate nel programma, la cercherà all'interno delle librerie aperte. I parametri da passare devono sempre essere di tipo LONG INTEGER. Per le funzioni che hanno come output un valore bisogna anche dichiararle come funzioni:

Font

```
'  Questo programma e' un esempio di
'  come usare i vari font disponibili
'
DECLARE  FUNCTION  OpenFont&  LIBRARY
DECLARE  FUNCTION  OpenDiskFont&  LIBRARY
LIBRARY  "graphics.library"
LIBRARY  "diskfont.library"
'
'
WINDOW 2,"OUTPUT FONT",(0,44)-(617,186),15
WINDOW 3,"INPUT FONT",(0,0)-(617,30),15
'
  INPUT  "Nome del font ";NomeFont$
'
  WHILE  NomeFont$ <> ""
    INPUT  "Altezza in pixel "; AltFont
'
    ta&(0) = SADD(NomeFont$+".font"+CHR$(0)) ' prepara gli attributi del font
    ta&(1) = AltFont*65536&
    pf& = OpenFont&(VARPTR(ta&(0)))
    IF  pf& = 0  THEN  pf& = OpenDiskFont&(VARPTR(ta&(0))) ' se il font indicato
 non e'
    IF  pf& = 0  THEN                                  ' nel sistema lo cerca nel
disco
        BEEP
        PRINT
        PRINT  "Il font ";NomeFont$;" non esiste !!!"
        PRINT
       ELSE
        WINDOW OUTPUT 2
        SetFont WINDOW(8),pf&
        PRINT
        PRINT  "Questo e' il font ";NomeFont$;" alto ";AltFont;" pixel";
'
'             il ; serve se il font successivo e' piu' alto        '
'             altrimenti coprirebbe in parte la riga precedente    '
'
     END IF
    INPUT "Nome del font ";NomeFont$
  WEND
'
'
  WINDOW CLOSE 2
  WINDOW CLOSE 3
'
END
```

DECLARE FUNCTION nomefunzione LIBRARY

Veniamo ora al programma proposto. Nella libreria graphics.library si trovano le funzioni per gestire i font. La libreria diskfont.library serve per aprire i font presenti nel disco. Ogni finestra che viene aperta può avere un proprio font indipendentemente da quello che c'è nelle altre. La funzione Setfont è quella che impone un determinato font nella finestra di output. I parametri che bisogna passare sono due. Il primo fornisce delle indicazioni sulla finestra di output (si ottiene con WINDOW(8)), mentre il secondo è un puntatore del font. Questo secondo parametro si ottiene con la funzione OpenFont, o con OpenDiskFont, a seconda che il font che si vuole usare sia già nel sistema oppure debba essere prelevato dal disco (un font è nel sistema se è stato caricato con Openfont, e vi resta fintanto che non è stato eliminato con closefont (puntatorefont). Il font topaz è l'unico sempre residente perché si trova in ROM). Se il valore di questo puntatore è 0 allora il font non è stato trovato. Il test su pf& è messo appositamente per evitare un immancabile Guru Meditation. Il parametro da passare invece alla funzione OpenFont (o OpenDiskfont) è il puntatore di una locazione di memoria contenente il nome del font ed i suoi attributi. Questi ultimi sono separati dal nome da un carattere nullo. In questo esempio l'unico attributo è quello relativo all'altezza del font. Altri attributi che il font potrebbe avere sono ad esempio lo stile proprio del font (da non confondere con lo stile determinato in modo algoritmico), che però fino ad ora non ho ancora visto applicato. Dalle prove che ho fatto è scaturito che un'altezza sbagliata causa l'uso del font con altezza più piccola rispetto a quella impostata, oppure l'unica esistente.

Per coloro che hanno intenzione di usare i font voglio rammentare alcune cose. Quando si cambia font conviene modificare con l'istruzione WIDTH anche il numero di colonne stampabili per evitare che parte del testo scompaia al di fuori della finestra. Se si vogliono usare font che sono in una directory differente dall'SYS:FONTS, basta eseguire da dos ASSIGN FONTS: nomedirectory.

Infine un ultimo consiglio. Se, come mi è capitato, volete posizionare una scritta in una determinata posizione della finestra e non riuscite a farlo con LOCATE, potete usare la funzione Move&(WINDOW(8),x&,y&) contenuta nella libreria graphics.library, dove x& e y& sono le coordinate del pixel relativo all'angolo basso sinistro del carattere col quale inizierà la stampa successiva.

Modifica del programma Convertfd

Nel numero di settembre-ottobre della rivista Amiga World sono riportate le modifiche da apportare al programma Convertfd in modo da eliminare degli inconvenienti. Il problema risiedeva nel fatto che alcune funzioni di libreria hanno lo stesso nome delle istruzioni Basic, come ad esempio READ, INPUT ecc. Questo ha portato a una nuova versione denominata NEWCONVERTFD. Le modifiche da apportare sono le seguenti. La prima linea del programma CONVERTFD risulta essere:

```
DEFINT a-Z 'by default, all variables are integer
```

Scrivere subito dopo questa linea le seguenti istruzioni:

```
READ Cnt
DIM Con$(Cnt)
FOR K=0 TO Cnt-1
     READ Con$(K)
NEXT K
```

In questo modo i dati che verranno aggiunti alla fine del programma verranno caricati nel vettore Con$. La prossima modifica è nella soubroutine chiamata GotFunction che si trova 30 linee al di sotto di quelle appena aggiunte. La prima linea risulta:

```
   GetToken 'Token$=FUNCTION'S NAME
Subito dopo questa linea, aggiungere:
   K$=Token$
   FOR K=0 TO Cnt-1
       I K$=CON$(K) THEN TOKEN$="X"+TOKEN$
   NEXT K
```

La prossima modifica è nella subroutine Bad file format, in cui bisogna cambiare la parola STOP in CLOSE e RETURN in STOP.

Infine aggiungere le seguenti righe di data alla fine del programma (cioè dopo la sobroutine GETCHAR):

```
DATA 11
DATA abs,Close,Exit,Input,Open,Output
DATA Read,tan,Translate,Wait,Write
```

Si raccomanda nei DATA di trascrivere letteralmente, rispettando le maiuscole e le minuscole.

Ora durante la conversione il programma aggiunge una x davanti al nome se questo coincide con uno presente nei DATA. Quando richiamate queste routine da programma è necessario usare questo nome nuovo, come ad esempio XREAD o XOPEN.

Un'ultima osservazione. Tutte queste modifiche sono necessarie solo se il programma chiama le funzioni di libreria indicate nei DATA.

Penso che sia tutto. Buon Lavoro!

Bibliografia per l'uso dei font: Amiga ROM Kernel References Manual: Libraries and Devices - Addison Wesley.