Programmare in C su Amiga

di Dario de Judicibus

La potenza di un programma di utilità come LMK *e quella della libreria grafica di Amiga mettono il programmatore C in grado di sviluppare e mantenere programmi di elevata qualità in un computer da troppi erroneamente considerato una macchina per giocare*

Nella scorsa puntata abbiamo proposto un semplice esercizio di grafica non interattiva, cioè un programma che, una volta partito, si limita a disegnare oggetti grafici in una certa sequenza, senza che l'utente possa intervenire se non per chiudere il programma stesso. Quello che bisognava fare era di scrivere un programmino analogo a quelli che si trovavano nella directory *Demo* del dischetto del WorkBench 1.1. Ovviamente non esiste uno schema ben preciso per questo tipo di programmi. In figura 1 riportiamo una possibile soluzione a tale esercizio. Naturalmente il vostro programma può soddisfare i requisiti richiesti ed essere completamente diverso da quello riportato in figura.

Un certo numero di elementi, tuttavia, devono essere presenti in entrambi.

- Innanzi tutto un primo blocco dove sono definite e/o inizializzate le strutture che definiscono la finestra ed altre variabili, come ad esempio la maschera di controllo (cioè **unsigned char mask**). Nel nostro caso, è la zona marcata come *blocco 0*.
- Quindi un blocco che apre le librerie necessarie al programma e la finestra nello schermo del WorkBench. Nel nostro caso, è la zona marcata come *blocco 1*.
- A questo punto c'è il cuore vero e proprio del programma, e qui ognuno ci mette quello che vuole. Nell'esempio in figura, questo blocco contiene un ciclo da 1000 passi che disegna all'interno della finestra 1000 segmenti di lista, ognuno posizionato e colorato in modo casuale. Nel nostro caso, è la zona marcata come *blocco 2*.
- L'ultimo blocco, ovviamente, è quello di chiusura. Per dar modo all'utente di osservare il risultato del ciclo di cui sopra, questo blocco contiene una chiamata alla **Wait()**. Avremmo potuto usare anche **WaitPort()**, con lo stesso risultato, oppure **Delay()**, ed in tal caso non sarebbe stato necessario aggiungere il gadget di chiusura alla finestra.

Segnatevi la funzione **Random()** riportata in figura ed utilizzata per l'esercizio. Potrebbe tornarvi utile. In molti programmi di grafica, infatti, è spesso necessario avere una funzione che ritorni un numero casuale compreso tra 0 ed un intero **n**. Dato che invece la funzione interna **rand()** ritorna un numero compreso tra 0 ed il massimo intero (di tipo **int**) possibile, bisogna scalare il risultato di quest'ultima in modo da riportarlo al *range* richiesto.

Introduzione

Riprendiamo con questo numero la trattazione ad argomenti paralleli iniziata nella 12ª puntata. Da una parte parleremo delle *macro interne* di *LMK [built-in macros]*, dall'altra inizieremo a parlare di riempimenti *[filling]*.

LMK

Nella 12ª puntata abbiamo introdotto il programma di utilità *LMK*. In particolare, riassumendo brevemente per chi non lo ricordasse, avevamo detto che:

1. *LMK* è un programma di utilità che serve a mantenere ed eseguire processi più o meno complessi di produzione, quale è appunto la generazione di un programma eseguibile a partire dai file sorgente, i file di inclusione e le librerie di compilazione, offrendo una serie di vantaggi, di cui i più importanti sono:
a) l'ottimizzazione del processo con l'esclusione dallo stesso di quelle operazioni che non è necessario ripetere una seconda volta, basando il criterio di scelta sulla data di creazione (o dell'ultima modifica effettuata) di quelli che abbiamo chiamato *discendenti*;
b) la garanzia di offrire a chiunque tutte le informazioni necessarie a ripetere il processo in questione, fornendo per giunta un meccanismo automatico di produzione.
2. *LMK* è basato sulla descrizione del processo da eseguire contenuta in un file e riportata secondo un formalismo ben definito.
3. Il formalismo or ora menzionato è sufficientemente flessibile da offrire una sintassi semplice ma sufficiente a gestire un elevato numero di possibilità, grazie soprattutto alle *macro* che lo trasformano quasi in un vero e proprio linguaggio.

Come è detto, *LMK* si basa sul programma di utilità **make** ben conosciuto ai programmatori in ambiente UNIX.

Vediamo ora di approfondire il discorso delle macro *LMK*.

Macro

Abbiamo detto due numeri fa, che una macro è definita dall'istruzione.

```
MNAME = definizione_della_macro
```

Essa deve essere inoltre definita *prima* di essere utilizzata, altrimenti sarà espansa in una stringa nulla.

Per utilizzare una macro, al contrario di quanto si fa in C, bisogna metterla tra parentesi (senza spazi di separazione), e precedere il tutto con il simbolo del dollaro, come segue:

```
Qui uso $(MNAME) definita prima.
```

Esistono poi cinque macro definite internamente. Due di esse vengono ridefinite dopo che *LMK* ha elaborato una istruzione che riporta una assegnazione di dipendenze (quelle cioè con il nome di un ascendente seguito dai due punti, e la lista dei suoi discendenti), ma prima che la relativa operazione venga effettuata. Le altre tre sono definite solo ogni qual volta viene usata una *regola di trasformazione*, come vedremo nella prossima puntata. Si tratta di regole che si utilizzano quando, sulla base dell'estensione del file, si desidera operare allo stesso modo su tutti i discendenti (ad esempio, compila con certe opzioni tutti i file con estensione **.c** e produci file con lo stesso nome base ed estensione **.o**).

Le prime due macro sono:

$@che assume come valore il nome del file bersaglio

$? che assume come valore la lista dei nomi di tutti quei discendenti che sono più recenti dell'attuale file bersaglio, se esiste.

Tutti i nomi dei file sono comprensivi di eventuali percorsi specificati nell'istruzione elaborata *[full directory pathname]*.

Le restanti tre sono:

$* che assume come valore il nome

del **primo** discendente ad eccezione dell'estensione (ad esempio **src/new/test** se il file si chiama **src/new/test.c**) ma incluso il percorso (vedi nota 1). In pratica percorso più nome base.
$< che assume come valore il nome di quel discendente che ha causato l'esecuzione dell'operazione associata ad un ascendente, cioè del primo discendente che è stato riconosciuto più recente del suo ascendente e che, come vedremo più avanti, verifica le *condizioni di trasformazione*, se fornite. Il percorso (vedi nota 1) non è incluso.
$>** che assume come valore il nome base (vedi nota 1) dello stesso file di **$<.

Questo almeno secondo il manuale. Ma dato che fidarsi è bene, ma non fidarsi è meglio, abbiamo voluto verificare di persona. In particolare abbiamo studiato le due macro **$<** e **$>**.

Le prove effettuate con *LMK* **1.05** hanno dato i seguenti risultati:
1. il programma utilizza comunque il *primo* file nella lista dei dipendenti, piuttosto che quello che ha causato l'azione, come riportato nelle due figure relative all'esempio più avanti;
2. la macro **$<** risulta essere comprensiva del percorso, contrariamente a quanto riportato dal manuale.

Riportiamo qui una delle tante prove effettuate. Consideriamo l'esempio in figura 2, e supponiamo di avere già eseguito una volta **plot.lmk**. Supponiamo di aver modificato **defs.h** che, come già detto nella 12ª puntata è un discendente che non appare esplicitamente nelle istruzioni di produzione, ma che è stato esplicitizzato in quanto contribuisce alle caratteristiche del prodotto finale.

Vediamo ora come si comporta *LMK*. Abbiamo numerato le righe per comodità nei riferimenti. Ovviamente la numerazione non fa parte del file. Per prima cosa vengono assegnate le macro **LIBS** e **LOPT**.

A questo punto tutte le macro interne hanno ancora un valore nullo. Quindi *LMK*, scorrendo il file **plot.lmk**, identifica tutti i discendenti terminali ed incomincia ad analizzare una per una le istruzioni che producono gli ascendenti immediatamente superiori (a loro volta, eventualmente, discendenti di un livello ulteriormente superiore). A questo punto il programma passa alla linea **5**, si accorge che **plinc/defs.h** è stato modificato, ed assegna le macro interne come riportato in figura 3 — passo 1. Quindi esegue l'istruzione alla linea **6** e passa all'ascendente successivo (linea

```
/*
** Esercizio 13 - Dario de Judicibus - Maggio 1989
*/

 #include "exec/types.h"
 #include "intuition/intuition.h"
 #include "graphics/gfxmacros.h"
 #include "proto/exec.h"
 #include "proto/intuition.h"
 #include "proto/graphics.h"
 #include "stdio.h"
 #include "stdlib.h"

 int Random(int);
 void CloseAll(void);

 #define IREV 0
 #define GREV 0
 #define INAME "intuition.library"
 #define GNAME "graphics.library"
 #define WFLAGS  WINDOWCLOSE|WINDOWDEPTH|WINDOWDRAG|WINDOWSIZING

 struct IntuitionBase *IntuitionBase;
 struct GfxBase *GfxBase;                                   /* : */
                                                            /* : */
 #define WWW 300
 #define HHH 150
 struct NewWindow nw =                                      /* | */
 {                                                          /* | */
   20, 20, WWW, HHH,                                        /* | */
   0, 1,                                                    /*   */
   CLOSEWINDOW,                                             /* 0 */
   WFLAGS|GIMMEZEROZERO|SMART_REFRESH|ACTIVATE,             /*   */
   NULL, NULL, "Questa è una prova",                        /* | */
   NULL, NULL, 0, 0, 0, 0,                                  /* | */
   WBENCHSCREEN                                             /* | */
 };
 struct Window *w;
 struct RastPort *rp;                                       /* : */
                                                            /* : */
 #define IMASK 0x01
 #define GMASK 0x02
 #define WMASK 0x04
 unsigned char mask = 0x00;
```

Figura 1 - Una possibile soluzione all'esercizio della scorsa puntata.

```
void main()
{
  int i;
  int x = 0, y = 0;

 /*
 **  Apriamo le varie librerie e la finestra sullo schermo del WorkBench
 */
  IntuitionBase = (struct IntuitionBase *)OpenLibrary(INAME,IREV);  /* | */
  if (IntuitionBase == NULL) CloseAll();                            /* | */
  mask |= IMASK;                                                    /* | */
  GfxBase = (struct GfxBase *)OpenLibrary(GNAME,GREV);              /* | */
  if (GfxBase == NULL) CloseAll();                                  /*   */
  mask |= GMASK;                                                    /* 1 */
  w = (struct Window *)OpenWindow(&nw);                             /*   */
  if (w == NULL) CloseAll();                                        /* | */
  mask |= WMASK;                                                    /* | */
  rp = w->RPort;                                                    /* | */

 /*
 ** --------------------------------------------------------------------- *
 **  In questo blocco mettiamo le istruzioni per disegnare una 1000 linee
 **  con coordinate e colori scelti casualmente.
 ** --------------------------------------------------------------------- *
 */
  SetDrMd(rp,JAM1);                                                 /* | */
  for (i=0;i<1000;i++)                                              /* | */
  {                                                                 /* | */
    x = Random(WWW);                                                /*   */
    y = Random(HHH);                                                /* 2 */
    SetAPen(rp,Random(4));                                          /*   */
    Draw(rp,x,y);                                                   /* | */
  }                                                                 /* | */
 /* ---------------------------------------------------------------------- */

  Wait(1<<w->UserPort->mp_SigBit);  /*  OK, aspettiamo ad uscire... */
  CloseAll();
}

void CloseAll()  /* ordine inverso!!! */                            /* | */
{                                                                   /* | */
  if (mask & WMASK) CloseWindow(w);                                 /*   */
  if (mask & GMASK) CloseLibrary(GfxBase);                          /* 3 */
  if (mask & IMASK) CloseLibrary(IntuitionBase);                    /*   */
}                                                                   /* | */

/*
**  Semplice procedura per convertire un numero casuale tra 0 ed il massimo
**  intero, in uno tra 0 ed un valore prefissato n.
*/
 int Random(n)
   int n;
 {
   return (rand()/13)%n;
 }
```

7). Anche qui è necessario eseguire l'istruzione successiva, dato che la lista dei discendenti contiene lo stesso file che aveva causato l'esecuzione del processo alla linea **6**. Di nuovo le macro interne vengono ridefinite (figura 3 — passo 2) ed il primo livello di produzione viene così completato.

Ora gli ascendenti così prodotti risultano essere discendenti del prodotto finale con una data di creazione più recente di quella dell'attuale file **plot**. Di conseguenza viene letta la linea **3**, ridefinite le macro interne (figura 3 — passo 3) ed eseguito il comando alla linea **4**. Notare il carattere di continuazione (barra diagonale inversa \ ). In questo caso, entrambi i discendenti sono *responsabili* della esecuzione del comando, e quindi vengono riportati in **$?**.

Notate che, benché il vero responsabile dell'esecuzione dei comandi è il discendente terminale **plinc/defs.h**, sia la macro **$<** che la macro **$>** fanno riferimento al primo discendente della lista (analogamente alla macro **$***). Per verificare che fosse realmente così, abbiamo invertito i discendenti in linea **5** ed abbiamo effettivamente ottenuto il risultato riportato in figura 4. Non sappiamo se tale risultato è voluto o meno (cioè è un baco di *LMK*), dato che non ne è fatta menzione sul manuale. Se qualche lettore ha delle buone conoscenze di UNIX e **make**, potrà confermare eventualmente se la stessa cosa succede anche in tale ambiente o no.

Abbiamo ora elementi su *LMK* sufficienti a permetterci di scrivere praticamente qualunque processo di generazione di moduli eseguibili a partire dal codice sorgente. Di fatto, non essendo *LMK* legato ad un linguaggio in particolare, nulla impedisce di sviluppare un programma in più linguaggi, compilare i singoli moduli e quindi legarli insieme con un *link-editor*. Ovviamente bisogna assicurarsi di rispettare le regole di chiamata e ritorno *[call/return]* sia del linguaggio usato nel programma *chiamante* che di quello usato nella procedura *chiamata*. A volte basta semplicemente passare *tutti* i parametri come puntatori, altre volte bisogna operare una serie di salvataggi e rispristini *[save/restore]* di registri, aree dati, ed informazioni varie richieste da uno, od entrambi i linguaggi usati. Ma la *programmazione multilinguaggio* va al di là degli scopi di questa rubrica.

Nella prossima puntata analizzeremo le funzioni *avanzate* di *LMK* e le opzioni di comando.

Note

1. D'ora in poi useremo i seguenti termini Italiani per identificare le varie parti che compongono il *nome completo* di un file:
volume: è il nome del volume (dischetto, disco rigido, disco ottico) che contiene il file in questione (salvo eccezioni, si considera comprensivo dei due punti finali — ad esempio **WorkBench:**);
unità: è l'identificativo dell'unità fisica su cui è montato il volume (ad esempio, **df2:**, **hd1:**, o **jh0:**);
percorso: è la stringa di caratteri che contiene il volume (o l'unità) e la lista di tutte le directory separate dal *carattere separatore* (nel caso dell'Amiga è "**/**", nel caso del PC-DOS è "**** "); il percorso è spesso considerato comprensivo dell'ultimo separatore;
nome: è il nome del file estensione inclusa (vedi più sotto);
nome base: è il nome del file estensione esclusa (vedi più sotto);
estensione: è l'estensione del file (vedi più sotto).

Dato che *LMK* deriva da una utilità del mondo UNIX, alcuni dei concetti che essa utilizza derivano da tale ambiente. In particolare, la struttura gerarchica del *filing system*, basata su file e directory. Tale struttura si ritrova anche nel PC-DOS e nell'AmigaDOS. In quest'ultimo, tuttavia, al contrario del primo, un file può avere come nome una qualunque stringa di 30 caratteri al massimo, esclusi il carattere separatore "**/**" ed i due punti "**:**". Il punto quindi ("**.**") è un carattere come un altro da utilizzare nel nome del file. Lo stesso accade nel mondo UNIX, dove il nome non è rigidamente diviso in due parti, ma può contenere qualunque carattere stampabile escluso il separatore "**/**", per un massimo che va dai 14 dell'UNIX System V ai 255 dell'UNIX BSD. La divisione in nome base ed estensione è quindi puramente applicativa, e non di sistema. In PC-DOS, invece, il nome del file è diviso in due parti: il nome base (8 caratteri al massimo) e l'estensione (3 caratteri al massimo). Quest'ultima viene sovente a rappresentare una indicazione del *tipo* di file da un punto di vista logico. Spesso alcune estensioni sono riservate o comunque per convenzione indicano un certo tipo di file e non dovrebbero essere usate impropriamente. É il caso di "**.c**", "**.o** , ed "**.h**" per il linguaggio C. Alcuni programmi di utilità, come appunto il compilatore del C, sono codificati in modo da riconoscere certe estensioni ed a trattare di conseguenza i file in un modo piuttosto che un altro. Quando si usano tali programmi in AmigaDOS, essi considerano come estensione l'ultima stringa di caratteri dopo l'ultimo punto contenuto nel nome del file, se esiste. Altrimenti l'estensione viene considerata nulla.
Ad esempio, in **E10.001.c** l'estensione è **.c** mentre il nome base è **E10.001**.
Spesso l'estensione si considera inclusiva del punto separatore, che comunque non è **mai** incluso nel nome base. Ricordo però ancora una volta che sia per AmigaDOS che per UNIX l'estensione non è una specifica di sistema.

Riempimenti di aree

Nella scorsa puntata abbiamo visto un certo numero di funzioni e macro grafiche per disegnare punti, linee, poligoni. Abbiamo inoltre visto come definire ed impostare i colori dei vari tipi di penne disponibili, i modi grafici e, cosa importantissima, le mascherine lineari e bidimensionali.

Questa volta parleremo di riempimenti di aree, ed in particolar modo delle funzioni riportate in figura 5.

La libreria grafica di Amiga mette a disposizione del programmatore due differenti tecniche di riempimento:
1. riempimento a macchia d'olio;
2. riempimento a definizione d'area.

Entrambe queste tecniche definiscono solo la zona ed il modo con cui questa deve essere riempita. Il colore, il modo grafico, un'eventuale mascherina bidimensionale ed il bordo esterno, se desiderato, sono quelli che al momento sono stati definiti per mezzo delle funzioni e delle macro C descritte nella scorsa puntata. È quindi possibile riempire un'area con un motivo definito precedentemente, utilizzare entrambi i modi grafici **JAM1** e **JAM2**, e ottenere un contorno intorno all'area specificata.

Riempimento a macchia d'olio

La prima tecnica è quella usata da **DPaint** quando si seleziona l'icona con il barattolo di vernice e si fa click in un certo punto dello schermo. La zona intorno a tale punto incomincia a riempirsi con il colore od il motivo *[pattern]* specificato precedentemente. Esistono due modi differenti di riempimento:
a confinamento *[outline mode]*
il riempimento avviene a partire dal punto specificato e continua in tutte le direzioni fermandosi solo in corrispondenza di quei pixel colorati con il colore di contorno, quello cioè definito con la macro **SetOPen()**. Tutti gli altri pixel quindi, indipendentemente dal colore che hanno, sono interessati dal riempimento. Ovviamente, nel caso che sia stato specificato sia un motivo che il modo grafico **JAM2**, il disegno sottostante sarà completamente ricoperto dal motivo suddetto, riportato in due colori. Nel caso invece che, insieme al motivo sia stato specificato **JAM1**, parte del disegno precedente sarà ancora visibile in corrispondenza degli zero nella matrice di definizione del motivo stesso. Se il contorno è chiuso, il riempimento sarà limitato ad una parte solamente dello schermo, altrimenti si estenderà per tutto lo schermo salvo le zone formate dal colore di contorno. Se due pixel sono adiacenti verticalmente

od orizzontalmente, il riempimento può passare da uno all'altro; se invece si toccano per un angolo, il riempimento si ferma.

A simpatia di colore *[color mode]*
il riempimento avviene a partire dal punto specificato e continua in tutte le direzioni solamente in corrispondenza di quei pixel che hanno lo stesso colore di quello di partenza. Ovviamente l'insieme deve essere connesso. Anche qui, cioè, due pixel che si toccano solo per un angolo non permettono il passaggio del colore o del motivo. Questo si espande solo su pixel adiacenti su un lato.

Questa tecnica di riempimento ha il vantaggio di operare su aree anche molto complesse disegnate in precedenza, ma, oltre ad essere più lenta di quella a definizione d'area, ha lo svantaggio di essere meno controllabile di quest'ultima, dato che non si può mai essere del tutto sicuri di aver definito con precisione l'area da riempire. Basta un solo *buco* nel confine od un *ponte* anche di un solo pixel nel modo a simpatia di colore, per far traboccare il motivo od il colore di riempimento fuori dalla zona desiderata.

La funzione grafica per effettuare il riempimento a macchia d'olio è la **Flood()** il cui prototipo è riportato in figura 5. Analogamente figura 6 descrive i parametri da passare e riporta un esempio d'uso.

Riempimento a definizione d'area

La seconda tecnica è quella che si usa in **DPaint** quando si seleziona, ad esempio, una poligonale piena. In pratica si definisce un'area chiusa disegnandola virtualmente sullo schermo con comandi analoghi a **Draw()** e **Move()**. Questa volta però, la poligonale chiusa non è realmente tracciata, ma solo impostata. Al comando di chiusura del perimetro, l'area è riempita e, nel caso sia stato specificato anche un colore di contorno, il perimetro viene evidenziato in tale colore. Dato che per spiegare questa tecnica è necessario spendere un po' più di due parole, la vedremo in dettaglio nella prossima puntata, insieme ad altre funzioni grafiche.

In figura 5 sono riportati i prototipi delle quattro funzioni grafiche che servono a definire il perimetro dell'area da riempire. Questa tecnica ha il vantaggio di garantire il riempimento solo dell'area specificata, oltre ad essere più veloce della prece-

```
      #
      # file: plot.lmk
      #
      # Esempio di makefile
      #

      #
      # Variabili
      #
1.    LIBS = LIB:lc.lib+LIB:lcm.lib+LIB:amiga.lib
2.    LOPT = NODEBUG SC SD

      #
      # Come ottenere plot da plot.o e graph.o
      #
3.    plot: ram:plot.o ram:graph.o
4.      LC:blink FROM LIBo+ram:plot.o+ram:graph.o  \
(cont)    TO plot LIB $(LIBS) $(LOPT)

      #
      # Come ottenere plot.o da plot.c
      #
5.    ram:plot.o: plot/plot.c plinc/plot.h plinc/defs.h
6.      LC:lc -b0 -oram: plot/plot.c

      #
      # Come ottenere graph.o da graph.c
      #
7.    ram:graph.o: plot/graph.c plinc/defs.h
8.      LC:lc -ad -oram: plot/graph.c
```

Figura 2 - Makefile (esempio).

Prova eseguita con LMK 1.05 facente parte del Lattice C 5.02 il 16/5/89

Passo	$@	$?
1	ram:plot.o	plinc/defs.h
2	ram:graph.o	plinc/defs.h
3	plot	ram:plot.o ram:graph.o

Passo	$*	$<	$>
1	plot/plot	plot/plot.c	plot
2	plot/graph	plot/graph.c	graph
3	ram:plot	ram:plot.o	plot

Figura 3 - Assegnazione delle macro interne (esempio).

VARIAZIONE EFFETTUATA

```
      #
      # file: plot.lmk
      :
      :
      #
      # Come ottenere plot.o da plot.c
      #
5.    ram:plot.o: plinc/defs.h plot/plot.c plinc/plot.h
6.      LC:lc -b0 -oram: plot/plot.c
      :
      :
```

Prova eseguita con LMK 1.05 facente parte del Lattice C 5.02 il 16/5/89

Passo	$@	$?
1	ram:plot.o	plinc/defs.h
:	:	:

Passo	$*	$<	$>
1	plinc/defs	plinc/defs.h	defs
:	:	:	:

Figura 4 - Variazione del Makefile e risultato conseguente (esempio).

```
/*
** *** AREE ***
*/
 long  AreaDraw   (struct RastPort *, long, long);
 void  AreaEnd    (struct RastPort *);
 long  AreaMove   (struct RastPort *, long, long);
 void  InitArea   (struct AreaInfo *, short *, long);
/*
** *** RIEMPIMENTI ***
*/
 void  Flood      (struct RastPort *, long, long, long);
```

Figura 5 - Le funzioni grafiche dell'Amiga trattate in questo articolo.

dente, a parità di area (forma e dimensioni), ma ha lo svantaggio di richiedere l'allocazione di un *raster* temporaneo e quindi un maggiore utilizzo di memoria.

Conclusione

Anche per questa volta è tutto. Utilizzate le funzioni apprese per scrivere programmi grafici un po' più complessi, e tenetevi pronti, perché nella prossima puntata sarà proposto un esercizio alquanto interessante ma che richiede un po' di abilità ed una buona dimestichezza con i concetti fin qui analizzati. E scrivete, naturalmente...

```
/*
**  Flood ( rp, modo, x, y);
**
**  dove    rp    è il puntatore alla struttura RastPort corrispondente
**                al raster su cui si sta operando
**          mode  è la modalità di riempimento: 0 per quella a contorno
**                fisso ed 1 per quella a simpatia di colore
**          x     è l'ascissa del pixel da cui deve partire il riempimento
**          y     è l'ordinata del pixel da cui deve partire il riempimento
*/

SetAPen(rp,1);     /* Usa il colore nel registro 1 come colore primario   */
SetOPen(rp,3);     /* Usa il colore nel registro 3 come colore di contorno */
SetDrMd(rp,JAM1);  /* Seleziona il modo grafico JAM1                      */

Move(rp,25,34);    /* Disegna un rombo          /\ (47,55)               */
Draw(rp,52,12);    /* irregolare con           /  \                      */
Draw(rp,74,32);    /* vertici:        (25,34) \    \ (74,32)             */
Draw(rp,47,55);    /*                          \  /                      */
Draw(rp,25,34);    /*                   (52,12) \/                       */

Flood(rp,0,50,30); /* e riempilo a partire da (50,30)                     */
```

Figura 6
Riempimento a macchia d'olio.

Casella Postale

Inauguriamo con questa puntata una nuova sezione, dedicata alle lettere dei lettori. Ricordo che potete scrivere direttamente alla Technimedia, oppure mandarmi un messaggio su MC-Link (identificativo **MC2120**). *Nel primo caso risponderò solo attraverso la rivista, e quindi, dato che gli articoli vanno consegnati con un paio di mesi di anticipo, se avete urgenza di ricevere una risposta, vi consiglio di contattarmi via modem alla mia casella postale in MC-Link.*
In ogni caso le lettere od i messaggi più significativi saranno comunque riportati in questa sezione. Naturalmente chi, leggendo di un determinato problema avuto da un lettore ed avendolo risolto, volesse contribuire a questa rubrica con risposte, trucchi, idee, sarà sempre il benvenuto. Cercate però sempre di verificare di persona quanto affermato o, se non è possibile, riportate sempre la fonte delle vostre informazioni

Mouse Pointer e Screen

A proposito di Amiga, nella decima puntata del tuo «Programmare in C su Amiga» (MC-marzo pag. 180, seconda colonna), ho riscontrato un'inesattezza. Tu dici che «... se il programmatore non specifica altrimenti, il puntatore [sprite] associato ad una finestra è lo stesso dello schermo a cui quella finestra appartiene...».

Ebbene non mi risulta che sia possibile associare un «mouse-pointer» ad uno schermo (non esiste chiamata alla ROM, né a librerie che abbiano tal scopo, né esiste campo della struttura «Screen» che possa far pensare ciò). Inoltre, secondo quanto dice la documentazione aggiornata Commodore («Autodocs 1.3») a proposito di ClearPointer(struct Window):*
(.....)
After calling ClearPointer(), every time this Window is the active one the default Intuition pointer will be the pointer displayed to the user. (....)

Il «default Intuition pointer» a cui si fa qui riferimento è quello presente nella struttura Preferences (struttura di Intuition globale, non associabile a uno schermo).

Oscar Sillani

Esatto Oscar. Non esiste la possibilità (peccato) di associare un puntatore ad uno schermo. Il puntatore base è quindi quello delle **Preferences**. L'unico sistema per *simulare* un puntatore di schermo, è quello di aprire una finestra di fondo sullo stesso ed associarvi il puntatore desiderato. Comunque non so fino a che punto ne valga la pena, dato che aprire una finestra del genere solo per il gusto di assegnarle un puntatore, a meno che non la si utilizzi anche per altro, mi sembra un inutile spreco di memoria.

Opzione di compilazione -b0

Nell'undicesima parte di «Programmare in C su Amiga» (MC-aprile '89) tu affermi che l'opzione di compilazione (Lattice) -b0 assicura che tutti i dati definiti come statici, esterni e le stringhe di caratteri siano indirizzate per mezzo di un campo di spostamento [offset] rispetto al registro base A4 da 32 bit piuttosto che 16 bit come è in genere. Tale tua affermazione risulta essere parzialmente inesatta: non esiste nel set di istruzioni del 68000 una modalità di indirizzamento con offset a 32 bit (nel 68020, forse...). L'opzione -b0 causa l'adozione di istruzioni ad indirizzamento fisico, senza ALCUN uso di offset, per quel che riguarda i dati di un modulo.

Ti porto un esempio tratto da un mio programma in cui accedo direttamente al blitter per fare «grafica veloce»; la linea di codice in questione è:

```
*BLTSIZE = bs;
```

posto che in precedenza la variabile «bs» è stata dichiarata come statica non inizializzata e che viene fatta la definizione di BLTSIZE con:

```
#define BLTSIZE ((UWORD*)0×DFF058)
```

la riga in questione, con l'opzione -b(opzional) viene tradotta in:

```
MOVEA.L #00DFF058,A0
MOVE.W 02.00000000(A4),(A0)
```

(nota l'offset di 0 [primo dato] nella sezione 2 [dati statici non inizializzati] rispetto ad A4) mentre con l'opzione -b0 avremo:

```
MOVEA.L #00DFF058.A0
MOVE.W 02.00000000,(A0)
```

dove noterai la totale assenza di offset (il registro A4 è assente) e l'adozione di un riferimento fisico (che nell'eseguibile sarà un RELOC32, indirizzo a 32 bit rilocabile, più ingombrante, più lento e implicante maggior overhead da parte di «LoadSegment()» del dos.library).

Oscar Sillani

Colpito di nuovo. In realtà l'opzione **-b0** causa l'adozione di quello che in Inglese è chiamato *full 32 bit addressing*, cioè l'indirizzamento assoluto (almeno così penso che dovrebbe essere tradotto *full* qui, dato che è in opposizione a quello relativo ad un offset). E questo è quanto intendevo scrivere, almeno a quanto risulta dai miei appunti. Un'altra volta imparo a scrivere un articolo fino all'una di notte. Grazie comunque del test in Assembler che io non avrei saputo fare dato che non conosco l'Assembler del 68000.

INTERNATIONAL SHOW ROOM

★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★

COMMUNICATION

3Com

3Plus Share
3Plus Open

Software operativi di rete in ambiente MS/DOS e OS/2. In entrambi gli ambienti operativi sono disponibili moduli per la gestione di posta elettronica e collegamenti remoti.

3S400
3Station

Server di rete basato su CPU 386, Workstation diskless

Etherlink
Tokenlink

Adattori Ethernet e Token Ring

ALCOM

LAN FAX/10
Easy Gate

Gateway di comunicazione per il collegamento di reti locali verso Telex, Fax e sistemi pubblici di posta elettronica.

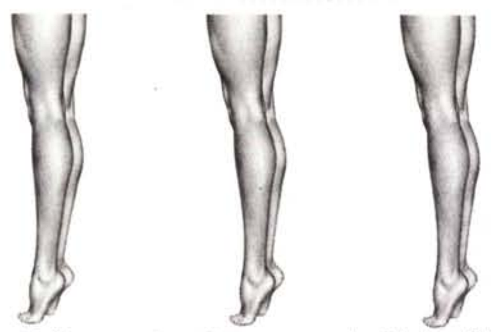
Rabbit SOFTWARE

Rabbit Station
Rabbit Gate

Emulatori IBM 3270 per PC stand alone o collegati in rete locale. Disponibili in versione Remote, DFT, X.25.

☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

CAD/CAM



Cadkey

Software di progettazione CAD 3D con modellazione geometrica di tipo wireframe.
Quotatura automatica standard ANSI-ISO.

CADKEY 3

Mastercam

Modulo CAD/CAM integrato per controllo Frese, Torni, EDM fino a tre assi. Post-processor standard ISO.

CNC
Software Inc.

- Schede grafiche ad alta risoluzione 1280x1024 16 o 256 colori
- Compatibili con i software di grafica e CAD più usati: Cadkey, Autocad, Personal Designer, Dr. Halo.

IMA*graph*

★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★

ADD·ON
E PERIFERICHE

LOGITECH

Mouse in versione seriale, bus e per PS/2 risoluzione standard 320 dpi.

ScanMan

Hand scanner in versione standard bus, MicroChannel e Macintosh, risoluzione fino a 400 dpi.

Finesse

Pacchetto software di Desk Top Publishing.

Jasmine

DirectDrive - DirectPrint
DirectServer

Hard disk esterni e removibili per Macintosh PLUS/SE/II.
Laser printer 300 dpi, 3Mb RAM, compatibile Post-script.
AppleShare file server per reti Macintosh Local Talk compatibile AFP.

SOTA
STATE OF THE ART TECHNOLOGY

Schede Super VGA, risoluzione massima 1024x768 16 colori

Schede accelleratrici compatibili con IBM PS/2 mod. 30, PC, XT Compaq DeskPro, Olivetti M24 e con la maggior parte delle macchine 8088 e 8086 con clock fino a 10 Mhz.

D A T A B A S E
S O F T W A R E

Fox Software

Fox Base +
Fox Base + Mac

Pacchetto di Data-Base relazionale compatibile con dBase III Plus. Disponibile sia in versione MS/DOS che Macintosh. Esiste in Versione Single User e Multiuser, 386 MS/DOS, nonché il modulo di Run Time per un numero illimitato di installazioni.

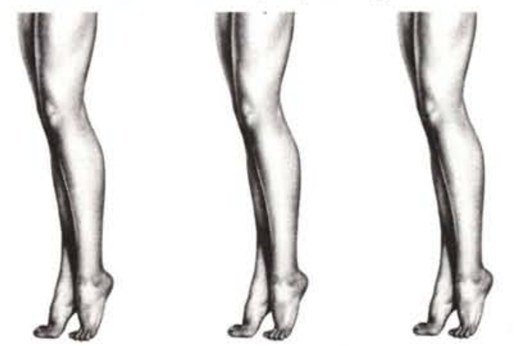
Nantucket

Clipper

Il compilatore per dBase III Plus. Un ambiente di sviluppo completo, aggiunge più di 30 funzioni al linguaggio dBase III, comprende un efficiente debugger, permette di agganciare routine scritte in C o in Assembler.

Sycero

Generatore di programmi in linguaggio dBase III/Clipper.



★★★★★★★★★★★★★★

P R O T E Z I O N I
S O F T W A R E

LOGIKEY

Dispositivo per la protezione del software. Si applica sulla porta seriale, non inibendone comunque l'utilizzo. Viene personalizzato per ogni cliente.

Fermatevi un attimo davanti a una vetrina di prodotti che non conosce frontiere geografiche. Apprezzerete hardware e software selezionati fra le migliori firme internazionali. Novità esclusive assolutamente in linea con le esigenze del mercato italiano. Una collezione di prodotti che abbina tecnologia e prezzo all'internazionalità dell'esperienza Algol. Troverete professionalità, competitività e risparmio. Fermatevi ancora un attimo: il nostro servizio di telemarketing è a vostra disposizione per parlarvi di soluzioni ma anche di prezzi, avviamento e assistenza. Un consiglio: tuffatevi.

MILANO: Via Feltre 28/6 - Tel. 02/26411411 (r.a.) - Fax 02/2154629
FILIALI: Roma Tel. 06/5913971-5919479 - Fax 06/5918745 - Ivrea Tel. 0125/424541-2
CENTRO SVILUPPO PROGETTI E SELEZIONE PRODOTTI: Palo Alto (California) USA Tel. 001 415 493-3800

ALGOL