Programmare in C su Amiga (26)

di Dario de Judicibus

Continuiamo la nostra carrellata sulle strutture elementari utilizzate da quadri e controlli. In questa puntata parleremo di bordi. Ritornano inoltre la Scheda Tecnica e la rubrica «Casella Postale»

Nella scorsa puntata avevamo proposto un semplice esercizio allo scopo di dimostrare come certi colori e modi grafici siano più adattati di altri qualora non si sappia con esattezza su che tipo di fondo si stia operando. In pratica si trattava di generare un fondo formato da rettangoli sovrapposti di varie forme e colori, e quindi di stampare su tale sfondo una frase qualunque utilizzando varie combinazioni di colori e modi grafici. Ovviamente questo esercizio serviva anche a farvi prendere confidenza con la struttura **IntuiText** presentata nella scorsa puntata.

In figura 1 viene mostrato un possibile esempio della soluzione all'esercizio proposto. Naturalmente il vostro programma può essere molto differente da quello presentato. L'importante è che soddisfi i requisiti richiesti.

La struttura del programma è molto semplice. Dopo le oramai classiche dichiarative iniziali, il programma principale non fa altro che chiamare in sequenza cinque sottoprogrammi. Il primo e

```
/*****************************************************************************
**  Programmare in C su Amiga  (c) 1989  Dario de Judicibus - Roma (I)     **
** ------------------------------------------------------------------------ **
**  Scheletro di un programma di visualizzazione dei testi                  **
**                                                                          **
*****************************************************************************/

 #include <exec/types.h>
 #include <intuition/intuitionbase.h>
 #include <graphics/gfxbase.h>
 #include <libraries/dos.h>

 #include <proto/all.h>
 #include <stdlib.h>
 #include <string.h>
 #include <stdio.h>
 #include <time.h>

/*
**  Prototipi delle funzioni interne al programma
*/
 void StartAll     ( void );
 void CloseAll     ( void );
 void DrawStuff    ( void );
 void DrawTexts    ( void );
 void LetsGo       ( void );

/*
**  Costanti
*/
 #define IREV 0
 #define GREV 0
 #define INAME "intuition.library"
 #define GNAME "graphics.library"
 #define DJ_COLS 400
 #define DJ_ROWS 150
 #define DJ_TITL "Esempio di gestione dei testi [DdJ]"
 #define COL0 0
 #define COL1 1
 #define COL2 2
 #define COL3 3
 #define COLS 4
 #define MODS 5
 #define LINE 10
 #define REPEAT 5

 /*
 **  Tipi
 */
 typedef struct IntuiMessage IMSG;
 typedef struct IntuiText    ITXT;
 typedef struct TextAttr     TXTA;

 /*
 **  Caratteristiche della finestra: gadget di CHIUSURA, di PROFONDITA',
 **  di SPOSTAMENTO, restauro automatico intelligente, tipo GZZ, attiva.
 */
  #define DJ_BASE  WINDOWCLOSE|WINDOWDEPTH|WINDOWDRAG
  #define DJ_SPEC  GIMMEZEROZERO|SMART_REFRESH|NOCAREREFRESH|ACTIVATE

 /*
 **  Puntatori alle principali strutture
 */
  struct IntuitionBase *IntuitionBase;
  struct GfxBase *GfxBase;
  struct Window *w;
  struct RastPort *rp;
  struct MsgPort *up;
  IMSG *imsg;

 /*
 **  Strutture di definizione della finestra e dei testi
 */
  struct NewWindow nw =
  {
    20, 20, DJ_COLS, DJ_ROWS,  /* posizione e dimensioni della finestra  */
    0, 1,                      /* colore delle penne di fondo e di segno */
    CLOSEWINDOW,               /* Segnalatori IDCMP: gadget di chiusura  */
    DJ_BASE|DJ_SPEC,           /* caratteristiche della finestra         */
    NULL, NULL, DJ_TITL,       /* gadget, checkmark, titolo              */
    NULL, NULL, 0, 0, 0, 0,    /* schermo, superbitmap, dimensioni min.  */
    WBENCHSCREEN               /* da aprire sullo schermo del WorkBench  */
  };

  TXTA topaz8 = { "topaz.font", 8, 0, 0 };
  TXTA topaz9 = { "topaz.font", 9, 0, 0 };

  ITXT base = { 1,0,NULL,20,0,&topaz9,NULL,NULL };

 /*
 **  Maschere di controllo
 */
  #define MSK_INT 0x0001
  #define MSK_GFX 0x0002
  #define MSK_WIN 0x0004
  UWORD  mask = 0x0000;
```

```
/*************************************************************************
**************************************************************************
** main: programma principale                                           **
**************************************************************************
*************************************************************************/
void main()
{
  StartAll ();          /* Effettuiamo le chiamete di partenza.          */
  DrawStuff();          /* Disegniamo un po' di roba                     */
  DrawTexts();          /* Scriviamo  un po' di roba                     */
  LetsGo   ();          /* Va bene. E' tutto pronto. Andiamo!            */
  CloseAll ();          /* Finito. Chiudiamo tutto.                      */
}

/*************************************************************************
** StartAll: chiamate di partenza                                       **
*************************************************************************/
void StartAll()
{
  long t;

  /*
  **  Apre le librerie (Intuition & Graphics) e la finestra
  */
  IntuitionBase = (struct IntuitionBase *)OpenLibrary(INAME,IREV);
  if (IntuitionBase == NULL) CloseAll();
  mask |= MSK_INT;
  GfxBase = (struct GfxBase *)OpenLibrary(GNAME,GREV);
  if (GfxBase == NULL) CloseAll();
  mask |= MSK_GFX;
  w = (struct Window *)OpenWindow(&nw);
  if (w == NULL) CloseAll();
  mask |= MSK_WIN;
  rp = w->RPort;      /* RastPort per la grafica */
  up = w->UserPort;   /* Porta utente per IDCMP  */

  srand(time(&t));

}

/*************************************************************************
** CloseAll: chiamate di chiusura                                       **
*************************************************************************/
void CloseAll()       /* ordine inverso rispetto StartAll()!!! */
{
  if (mask & MSK_WIN) CloseWindow(w);
  if (mask & MSK_GFX) CloseLibrary((struct Library *)GfxBase);
  if (mask & MSK_INT) CloseLibrary((struct Library *)IntuitionBase);
  exit(0);
}


/*************************************************************************
** DrawStuff: disegni a caso                                            **
*************************************************************************/
void DrawStuff()
{
  int i,j;
  int xm,ym,xM,yM,xt,yt;

  for (i=0;i<REPEAT;i++)
  {
    SetAPen(rp,rand()%COLS);
    for (j=0;j<REPEAT;j++)
    {
      SetDrMd(rp,rand()%MODS);
      xm = rand()%(DJ_COLS-10) + 1;
      ym = rand()%(DJ_ROWS-10) + 1;
      xM = rand()%(DJ_COLS-10) + 1;
      yM = rand()%(DJ_ROWS-10) + 1;
      if (xm > xM) xt = xm, xm = xM, xM = xt;
      if (ym > yM) yt = ym, ym = yM, yM = yt;
      RectFill(rp,xm,ym,xM,yM);
    }
  }

}

/*************************************************************************
** DrawTexts: testi non a caso                                          **
*************************************************************************/
void DrawTexts()
{
  base.FrontPen = COL0;
  base.BackPen  = COL1;

  base.DrawMode = JAM1;
  base.TopEdge += LINE;
  base.IText    = "COL0 COL1 JAM1";
  PrintIText(rp,&base,0,0);

  base.DrawMode = JAM2;
  base.TopEdge += LINE;
  base.IText    = "COL0 COL1 JAM2";
  PrintIText(rp,&base,0,0);

  base.DrawMode = COMPLEMENT;
  base.TopEdge += LINE;
  base.IText    = "COL0 COL1 COMPLEMENT";
  PrintIText(rp,&base,0,0);

  base.DrawMode = COMPLEMENT|INVERSVID;
  base.TopEdge += LINE;
  base.IText    = "COL0 COL1 COMPLEMENT|INVERSVID";
  PrintIText(rp,&base,0,0);

  base.DrawMode = JAM1|INVERSVID;
  base.TopEdge += LINE;
  base.IText    = "COL0 COL1 JAM1|INVERSVID";
  PrintIText(rp,&base,0,0);

  base.DrawMode = JAM2|INVERSVID;
  base.TopEdge += LINE;
  base.IText    = "COL0 COL1 JAM2|INVERSVID";
  PrintIText(rp,&base,0,0);

  base.FrontPen = COL2;
  base.BackPen  = COL3;

  base.DrawMode = JAM1;
  base.TopEdge += LINE;
  base.IText    = "COL2 COL3 JAM1";
  PrintIText(rp,&base,0,0);

  base.DrawMode = JAM2;
  base.TopEdge += LINE;
  base.IText    = "COL2 COL3 JAM2";
  PrintIText(rp,&base,0,0);

  base.DrawMode = COMPLEMENT;
  base.TopEdge += LINE;
  base.IText    = "COL2 COL3 COMPLEMENT";
  PrintIText(rp,&base,0,0);

  base.DrawMode = COMPLEMENT|INVERSVID;
  base.TopEdge += LINE;
  base.IText    = "COL2 COL3 COMPLEMENT|INVERSVID";
  PrintIText(rp,&base,0,0);

  base.DrawMode = JAM1|INVERSVID;
  base.TopEdge += LINE;
  base.IText    = "COL2 COL3 JAM1|INVERSVID";
  PrintIText(rp,&base,0,0);

  base.DrawMode = JAM2|INVERSVID;
  base.TopEdge += LINE;
  base.IText    = "COL2 COL3 JAM2|INVERSVID";
  PrintIText(rp,&base,0,0);
}

/*************************************************************************
** LetsGo: OK. Blocco principale di controllo                           **
*************************************************************************/
void LetsGo(void)
{
  /*
  **  Svuotiamo la coda messaggi o mettiamoci in attesa del successivo
  */
  FOREVER  /* Ciclo infinito: si interrompe con "break" */
  {
    if ((imsg = (IMSG *)GetMsg(up)) == NULL)  WaitPort(up);
    else                                      break;
  }
}
```

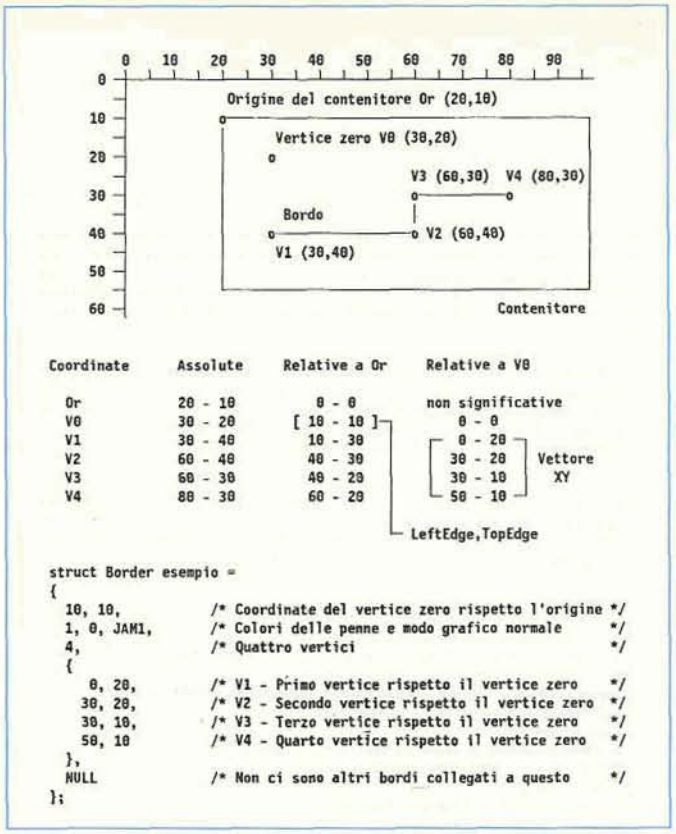
Figura 1 - Esercizio proposto nella 25ª puntata.

```
struct Border
{
    SHORT LeftEdge, TopEdge;    /* Posizione relativa all'origine          */
    UBYTE FrontPen, BackPen;    /* Penne per il tratto ed il fondo         */
    UBYTE DrawMode;             /* modo grafico                            */
    BYTE Count;                 /* numero di coordinate                    */
    SHORT *XY;                  /* coordinate relative a LeftEdge/TopEdge  */
    struct Border *NextBorder;  /* puntatore ad un'altra struttura Border  */
};
```

Figura 2 - Border.

```
/*
**  NOTA: La funzione DrawBorder() configura il RastPort nel modo
**        appropriato, e quindi sposta il cursore sul primo vertice,
**        disegnando successivamente i segmenti ad i vertici seguenti.
**        Una volta terminato, se il campo NextBorder non è nullo,
**        DrawBorder() chiama se stessa ricorsivamente.
*/
 void DrawBorder    /* Stampa il bordo nel raster in una certa posizione */
 (
   struct RastPort  *RastPort  , /* Destinazione del bordo                */
   struct Border    *Border    , /* Bordo da stampare (anche una catena) */
   SHORT             TopOffset , /* Posizione dall'alto                  */
   SHORT             LeftOffset  /* Posizione da sinistra                */
 );
```

Figura 4 - Draw Border().




```
struct Border esempio =
{
  10, 10,           /* Coordinate del vertice zero rispetto l'origine */
  1, 0, JAM1,       /* Colori delle penne e modo grafico normale      */
  4,                /* Quattro vertici                                */
  {
    0, 20,          /* V1 - Primo vertice rispetto il vertice zero    */
   30, 20,          /* V2 - Secondo vertice rispetto il vertice zero  */
   30, 10,          /* V3 - Terzo vertice rispetto il vertice zero    */
   50, 10           /* V4 - Quarto vertice rispetto il vertice zero   */
  },
  NULL              /* Non ci sono altri bordi collegati a questo     */
};
```

Figura 3 - Definizione di un bordo.

l'ultimo sono i soliti **StartAll()** e **CloseAll()** che sfruttano la tecnica delle maschere di controllo per l'apertura e la chiusura delle varie risorse utilizzate dal programma.

La seconda procedura serve a disegnare sul fondo della finestra. Come si vede in figura, è stata utilizzata la funzione **rand()** del Lattice **C** per calcolare le dimensioni ed i colori dei vari rettangoli che faranno da sfondo al testo. Il valore ottenuto da **rand()** è rinormalizzato nella gamma richiesta di valori utilizzando l'operatore *modulo%*. Inoltre, per quello che riguarda le coordinate dei vertici dei rettangoli, si è scelto di prendersi 10 pixel di bordo e di evitare valori nulli sommando uno al risultato dell'operazione **rand()%latomax.** Questo non è in effetti necessario, dato che stiamo utilizzando una finestra GZZ, ma non fa certo male. Ricordo che, nel caso avessimo deciso di non usare una *GimmeZeroZero*, avremmo dovuto variare tutte le coordinate in modo da tener conto delle dimensioni del bordo della finestra.

Per finire bisogna assicurarsi che le coordinate del secondo vertice del rettangolo siano maggiori di quelle del primo vertice. A questo scopo si utilizzano due semplici istruzioni di scambio *[swap]*. Se uno vuole può anche aggiungere un paio di istruzioni per evitare che il rettangolo degeneri in una linea.

La terza procedura serve a rendere i testi. Per comodità, ogni testo riporta i parametri relativi agli attributi del testo stesso (colori e modo grafico). Inoltre, al fine di ottimizzare la memoria, si è deciso di utilizzare sempre la stessa struttura **IntuiText.**

La penultima procedura altro non è che il ciclo di attesa dell'evento che indica che l'utente ha deciso di chiudere la finestra e terminare il programma. Dato che questo è l'unico evento che può arrivare alla porta IDCMP, come specificato nella struttura **NewWindow,** non c'è bisogno di una procedura tipo **HandleEvents().**

Non entrerò ulteriormente nel dettaglio del codice anche perché ritengo sia sufficientemente chiaro a chi mi abbia seguito nelle ultime puntate od abbia comunque una certa conoscenza di Intuition e del C. Chi vuole può comunque divertirsi a modificarlo aggiungendo nuove combinazioni od ottimizzando il codice, che è stato volutamente scritto seguendo uno stile lineare, specialmente nella **DrawTexts(),** per renderlo più semplice e chiaro.

Introduzione

In questa puntata parleremo di *bordi*. Le strutture che servono a definire tali oggetti sono, come nel caso di **IntuiText,** strutture elementari utilizzabili direttamente o referenziabili da altri oggetti più complessi. In entrambi i casi, comunque, esse fanno sempre riferimento ad un *elemento contenitore* che rappresenta anche il sistema di coordinate nel quale vanno posizionate. Come già detto nella scorsa puntata, il punto del contenitore rispetto al quale sono definiti tali oggetti si chiama *origine*.

Border

La struttura **Border** non serve solo, come farebbe pensare il nome, a disegnare cornici, ma è la base per rendere qualunque figura formata da segmenti connessi di qualunque inclinazione. Inoltre, legando fra di loro più strutture **Border** a formare l'oramai classica catena di strutture, è possibile di fatto disegnare qualunque gruppo di figure anche non connesse fra loro.

Ogni *bordo* ha inoltre una sua propria origine, cioè un punto rispetto al quale vanno date le coordinate di tutti i vertici. Per distinguerla dall'origine del contenitore la chiameremo *vertice zero*. In pratica, per definire un *bordo* basta fornire il vettore delle coordinate relative al vertice zero di tutti i vertici della figura da disegnare.

La struttura **Border** (vedi figura 2) è

La scheda tecnica

Anche questa volta presenteremo cinque comandi dell'*AmigaDOS 1.3* a partire da **GETENV.** Ricordo ancora una volta che le informazioni qui riportate sono specifiche della versione 1.3 e che, sebbene la sintassi ed il formato di ogni comando siano riportati nella forma completa, le specifiche *non* includono anche quelle informazioni che possono essere trovate nel manuale dell'*AmigaDOS 1.2*.

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

Comando:	GETENV
Formato:	GETENV <nome>
Sintassi:	GETENV "NAME/A"
Scopo:	Ritorna il valore di una variabile di sistema
Specifiche:	Serve per visualizzare il valore di una variabile di sistema, memorizzata tramite l'handler ENV:. Questo è attualmente simulato via memoria RAM:, ma è descinato a diventare un "vero" handler nella versione 2.0 del sistema operativo.

Comando:	ICONX
Formato:	ICONX
Sintassi:	ICONX
Scopo:	Esegue una macro AmigaDOS da Workbench
Specifiche:	Permette di eseguire macro [script] AmigaDOS da Workbench. Per usare ICONX si deve procedere come segue: - si crea una macro di sistema utilizzando un editore - si associa a questo file una icona di tipo "progetto" - si chiama da WorkBench la voce INFO e si assegna al parametro DEFAULT TOOL il valore C:ICONX - se lo si desidera si possono specificare i parametri di tipo (TOOL TYPE) WINDOW= per specificare le dimensioni e la posizione della finestra tipo CLI che ICONX apre all'esecuzione della macro DELAY= per specificare un eventuale intervallo di tempo tra la fine dell'esecuzione della macro e la chiusura della finestra, in modo da permettere all'utente di leggere l'eventuale risultato visualizzato dalla macro. DELAY=0 vuol dire: chiudi la finestra quando l'utente preme Ctrl-C. Utilizzando la tecnica della SELEZIONE ESTESA, tramite il tasto SHIFT mentre si seleziona un file, è possibile passare alla macro il contenuto del file come parametri della macro. Per far ciò, questa deve avere in testa l'istruzione .key "" oppure .<spazio>

Comando:	IF
Formato:	IF [NOT] [WARN] [ERROR] [FAIL] [stringa EQ|GT|GE stringa] [VAL] [EXISTS <file>]
Sintassi:	IF "NOT/S,WARN/S,ERROR/S,FAIL/S,EQ/K,GT/K,GE/K,VAL/S,EXISTS/K"
Scopo:	Permette l'esecuzione condizionata di istruzioni nelle macro
Specifiche:	Esegue un blocco di istruzioni fino ad un comando ELSE od ENDIF, qualora la condizione specificata sia vera. GT e GE non erano presenti nella versione 1.2. In genere la comparazione viene effettuata tra stringhe, tuttavia, se si specifica VAL, essa diviene di tipo numerico.
Esempio:	IF EXISTS <filename> IF <filesize> GT 2000 VAL DELETE <filename> QUIET ENDIF ENDIF

Comando:	INFO
Formato:	INFO <unità>
Sintassi:	INFO "DEVICE"
Scopo:	Visualizza informazioni sull'organizzazione dei file
Specifiche:	Visualizza informazioni sui volumi montati e sulla ripartizione dello spazio nelle memorie di massa. La versione 1.3 permette una corretta gestione dei volumi con nomi molto lunghi e permette di visualizzare le informazioni relative anche ad un singolo volume.
Esempio:	INFO df1: produce Mounted disks: Unit Size Used Free Full Errs Status Name DF1: 880K 1219 539 69% 0 Read Only Boot Disk

Comando:	INSTALL
Formato:	INSTALL DRIVE <DF0|DF1|DF2|DF3>: [NOBOOT] [CHECK]
Sintassi:	INSTALL "DRIVE/A,NOBOOT/S,CHECK/S"
Scopo:	Rende il disco un disco DOS
Specifiche:	Questa versione permette di trasformare un disco non-DOS in un disco DOS scrivendo nel blocco di caricamento [boot block]. Nel blocco può essere caricato o meno il codice di caricamento del sistema a seconda che l'opzione NOBOOT sia omessa o specificata. CHECK serve a verificare se il blocco di caricamento è di tipo valido o meno. In quest'ultimo caso la cosa potrebbe essere dovuta alla presenza di un virus. Ci sono tuttavia alcuni prodotti che usano blocchi anomali a scopo di protezione o per altri motivi; la maggior parte sono giochi. Se il disco contiene un blocco di caricamento valido ed è stata specificata CHECK, INSTALL ritorna 0, altrimenti ritorna 5 (WARN).

```
/* ************************************************************ **
**                                                              **
**    Programmare in C su Amiga  -  Dario de Judicibus  1990    **
** ------------------------------------------------------------ **
**    Esempio di scrittura e lettura in finestre CON: e RAW:    **
**    utilizzando solo le funzioni standard del Lattice C 5.xx  **
**                                                              **
** ************************************************************ */

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

 void StartAll(void);
 void CloseAll(void);

 #define IREV 0
 #define INAME "intuition.library"

 struct IntuitionBase *IntuitionBase;
 FILE *confp, *rawfp;

 #define IMASK 0x01
 unsigned char mask = 0x00;


 void main()
 {

   int c;

   StartAll();

   confp = fopen("CON:20/20/320/170/CON:","w+");
   if (confp == NULL) CloseAll();
   do
   {
     rewind(confp);
     fprintf(confp,"Premi uno o più caratteri ed INVIO\nESC per uscire\n");
     rewind(confp);
     c = fgetc(confp);
     rewind(confp);
     fprintf(confp,"Hex equivalente: 0x%2X\n",c);
   }
   while (c != 0x1B);
   fclose(confp);

   rawfp = fopen("RAW:20/20/320/170/RAW:","w+");
   if (rawfp == NULL) CloseAll();
   do
   {
     rewind(rawfp);
     fprintf(confp,"Premi un carattere senza INVIO\nESC per uscire\n");
     rewind(rawfp);
     c = fgetc(rawfp);
     rewind(confp);
     fprintf(confp,"Hex equivalente: 0x%2X\n",c);
   }
   while (c != 0x1B);
   fclose(rawfp);

   CloseAll();
 }

 void CloseAll()  /* ordine inverso!!! */
 {
   if (mask & IMASK) CloseLibrary(IntuitionBase);
   exit(0);
 }

 void StartAll()
 {
  /*
  ** Apri le librerie
  */
   IntuitionBase = (struct IntuitionBase *)OpenLibrary(INAME,IREV);
   if (IntuitionBase == NULL) CloseAll();
   mask |= IMASK;
 }
```

Figura 5 - I/O in finestre CON: e RAW:.

formata da otto campi:

LeftEdge: indica la posizione del vertice zero del *bordo*, in pixel, rispetto al lato sinistro dell'elemento che lo contiene;

TopEdge: indica la posizione del vertice zero del *bordo*, in pixel, rispetto al lato superiore dell'elemento che lo contiene;

FrontPen: è il numero del registro del colore utilizzato per tracciare i segmenti che compongono il *bordo*;

BackPen: non è utilizzato per i *bordi*;

DrawMode: è il modo grafico di tracciamento del *bordo* (vedi più avanti);

Count: specifica il numero di vertici forniti nel vettore **XY;**

XY: è il vettore che contiene le coordinate di tutti i vertici che compongono il *bordo*, relative al vertice zero (vedi più avanti);

NextBorder: punta ad un'altra struttura **Border,** permettendo così di collegare più *bordi* insieme a formare una lista; ogni *bordo* può avere ovviamente caratteristiche completamente differenti.

Spendiamo qualche parola in più su un paio di questi campi. Innanzi tutto **DrawMode**. Questo campo, differentemente da quanto succede per **IntuiText**, può assumere solo due valori:

JAM1: che indica che va usata FRONTPEN per il tracciamento dei segmenti;

COMPLEMENT: con il quale il segmento viene tracciato cambiando lo sfondo con il suo complemento binario (in termini di numero del registro di colore).

Ecco perché il campo **BackPen** viene di fatto ignorato. Esso non ha alcuna utilità se si usa uno di questi due modi grafici.

Veniamo ora al vettore **XY.** Esso è formato dalle coordinate (coppia X, Y) di tutti i vertici che compongono il *bordo*, misurate a partire dal vertice zero. Quest'ultimo non fa necessariamente parte del *bordo*, a meno che una delle coppie fornite non sia appunto (0,0). La prima coppia rappresenta quindi le coordinate del primo vertice. Ogni coppia successiva rappresenta il punto di arrivo di un segmento ed, eventualmente, quello di partenza del successivo, a meno che non sia l'ultima coppia (vedi figura 3).

Una caratteristica interessante di questo vettore è quella di rappresentare di fatto la forma del *bordo*, indipendentemente da dove esso sia posizionato nel contenitore. Questo permette di riutilizzare lo stesso vettore per definire più *bordi* che abbiano la stessa forma ma che si trovano in punti diversi di un elemento contenitore o che appartengano addirittura a contenitori diversi. Questo è il motivo per il quale i vertici del *bordo* vanno misurati relativamente al vertice zero, piuttosto che rispetto all'origine del contenitore. Se fosse stata scelta quest'ultima soluzione, infatti, avremmo dovuto usare vettori differenti solo per un fattore costante pur essendo i *bordi* da disegnare esattamente identici, sprecando così memoria e complicando inutilmente il programma.

È importante capire le ragioni che stanno alla base delle scelte operate dagli sviluppatori di un sistema operativo o di un programma, piuttosto che limitarsi ad accettarle ed utilizzarle, perché prima o poi vi capiterà sicuramente di dover fare scelte analoghe. L'espe-

rienza di un programmatore non sta infatti tanto nella sua conoscenza di un linguaggio o di un sistema operativo, quanto nell'insieme di tutte quelle tecniche logiche (chiamate a volte, scherzosamente, trucchi) che sono in effetti il vero motore di un programma. Molti ottimi programmatori infatti, conoscono così tanti linguaggi da aver bisogno comunque della guida di riferimento mentre programmano, sia per evitare errori di sintassi, sia per trovare le funzioni ed i servizi messi a disposizione da quello specifico linguaggio. Non è quindi tanto importante conoscere a menadito un linguaggio od il formato delle funzioni di un sistema, quanto avere la capacità di sviluppare una logica di programmazione semplice, flessibile e potente.

Naturalmente anche i bordi, come già i testi, possono essere direttamente disegnati in un *raster* utilizzando una funzione apposita, oltre che essere indirettamente resi quando viene visualizzato un oggetto che li referenzia, come ad esempio un quadro od un controllo. Questa funzione si chiama **DrawBorder()** ed è riportata in figura 4.

Conclusione

Nella prossima puntata parleremo della struttura **Image**. Purtroppo lo spazio limitato non mi ha consentito di farlo in questa puntata. Ad ogni modo vi lascio un semplice esercizio per la prossima volta. Modificate nel programma che avete scritto a soluzione dell'esercizio che vi avevo proposto nella 24ª puntata, la parte relativa alla generazione del fondo su cui stampare una frase in differenti modi grafici. Invece di disegnare sul fondo tanti rettangoli di differenti dimensioni e colori, tracciate un certo numero di *bordi* scegliendo a caso vuoi le coordinate dei vertici, vuoi i colori ed il modo grafico.

Buon lavoro!

Casella Postale

Questo mese rispondo ad una lettera ricevuta da Campagna, in provincia di Salerno, il 30 luglio 1990.

Oggetto:
Chiarimenti sulle funzioni **getchar()** e simili, programmando in C su Amiga 500 - v. 1.3.

Gentilissimo Signor Dario de Judicibus,
sono un lettore di MC dal lontano 1985. Da allora non ho perso un numero del suddetto mensile ed ho avuto così l'occasione di seguire i vostri articoli, soprattutto quelli legati all'Amiga ed al Linguaggio C. Come avete già capito, posseggo un Amiga 500 e programmando in C su di esso ho riscontrato l'evidente difficoltà di usare correttamente la funzione getchar() e le sue simili nei programmi tipo i classici di K&C.

È banalmente evidente che potrei fare a meno di scrivere proprio questi programmi, dedicandomi a cose più serie. Ma due motivi mi spingono ad insistere nella risoluzione di questo problema:

- *il primo è che sono metodico e testardo, perciò devo mettere anche questo tassello al posto giusto, se voglio andare avanti serenamente.*
- *Il secondo motivo è legato a voi (ecco il perché della lettera!?) e precisamente ad un vostro «vecchiotto» articolo: «Programmare in C su Amiga (2)» del giugno 1988. In esso (scusate se l'ho ripescato!?) mi confermate i problemi I/O che si riscontrano nel CLI di Amiga — mi riferisco alle finestre aperte come CON: o come RAW: — ma nello stesso tempo rimandate ad un chiarimento prossimo, che però non mi risulta abbiate fatto ancora. Avete sì toccato molti temi interessanti, quanto piuttosto ostici per me, ma non più quello.*

Sono alquanto imbarazzato nel parlarvi proprio di questo problema, perché sono convinto che non lo riteniate più molto interessante; tuttavia vi pregherei lo stesso di darmi una risposta, anche minima, anche solo come il titolo di un libro (facilmente reperibile...) sul quale possa studiare da solo questa soluzione.

Credo di aver abusato già abbastanza della vostra cortese attenzione per finire vi elenco il sistema su cui studio:
Amiga 500 con espansione ed accessori vari + Lattice C 5.04 (semi-originale).

Se anche non meritassi la vostra risposta, vi ringrazio per avermi seguito fin qua e vi saluto cordialmente.

Ciao Liberato - Campagna (SA)

Innanzi tutto non penso che provare programmini *classici alla K&R* sia da considerare stupido o banale. Il linguaggio C, benché indubbiamente potente, non può certo considerarsi uno dei più lineari e semplici per quello che riguarda leggibilità e sintassi. Spesso questi piccoli programmi possono insegnare più cose di quante se ne può imparare affrontando programmi «più seri». E se qualcuno ha qualche dubbio a riguardo, vada a leggersi lo splendido *The C Puzzle Book* di Alan R. Feuer, pubblicato dalla Prentice-Hall (ISBN 0-13-109926-4).

Veniamo ora al problema in questione. La funzione **getchar(),** implementata in realtà nel Lattice C 5.xx come macro, dovrebbe leggere un carattere per volta da console. Se apriamo una finestra come CON:, tuttavia, i dati immessi da tastiera vengono filtrati in modo da lasciar passare solo i caratteri da 0x20 a 0x7E e da 0x80 a 0xFF. La maggior parte dei caratteri di controllo, i tasti cursore ed i tasti funzionali sono intercettati ed utilizzati dal gestore di CON:. Ad esempio, se si usa NEWCON:, il tasto di cancellazione inversa può essere utilizzato per cancellare un carattere mentre i tasti cursore permettono di spostarsi su e giù per il testo. Se poi si utilizza **ConMan**, anche alcuni tasti funzione sono attivi per varie operazioni. Inoltre, il testo non viene presentato al programma fintanto che non viene premuto il tasto di *invio*. Ecco perché la **getchar()** non si comporta come ci si potrebbe aspettare in una console pura. Per risolvere il problema, è necessario aprire la finestra come RAW:, ma in questo caso bisogna aspettarsi di dover gestire anche un certo numero di altri caratteri oltre a quelli stampabili, e precisamente:

- i tasti funzione,
- il tasto di aiuto,
- i tasti cursore.

In aggiunta a tali sequenze (non si tratta infatti di singoli caratteri quanto di sequenze di due o più byte), è possibile chiedere alla console di ricevere informazioni più dettagliate relativamente al codice del tasto premuto, il tempo di durata della pressione, i tasti qualificatori premuti contemporaneamente al tasto in questione e così via. Usando RAW: quindi, la **getchar()** può funzionare a condizione di ridefinire **stdin,** ma la gestione dei dati in ingresso è più complessa.

Un esempio di quanto detto, che utilizza la **fgetc()** invece dalla **getchar()** in quanto, piuttosto che ridefinire **stdin,** si è scelto di usare la **fopen()** per ottenere il puntatore al flusso I/O, è riportato in figura 5. Provi a compilare questo semplice programmino con **1c -L** ed a farlo girare.

Una tecnica più completa per le operazioni di I/O coinvolge la **console.device**. A questo riguardo Le consiglio di leggere l'articolo *I/O serial device* a pag. 220 di MC88 del settembre '89, e la lettera *La Input.Device* pubblicata nella rubrica *Software Amiga* a pag. 244 di MC89 dell'ottobre '89. Si tratta di due letture molto interessanti per comprendere il concetto di *device* e quindi affrontare successivamente la **console.device**, di cui spero di parlare quanto prima.