Programmare in C su Amiga (17)

di Dario de Iudicibus

Il nostro slalom parallelo continua con la gestione degli eventi via IDCMP e con un nuovo programma di utilità: GREP. *Torna la rubrica «Casella Postale»*

Ho potuto notare in questi ultimi mesi come il livello degli articoli di MCmicrocomputer dedicati all'Amiga abbiano raggiunto un notevole livello, sia in termini di qualità che di quantità. Livello che pone questa rivista ai vertici nel panorama italiano anche in questa area, quella Amiga cioè, troppo spesso ignorata dal mondo dell'editoria informatica italiana. In particolare, ho notato con piacere come molti argomenti da me trattati nelle prime puntate, e poi abbandonati, vuoi per mantenere il livello di questa rubrica accessibile anche a chi ha da poco iniziato a programmare in C su Amiga, vuoi per non favorire troppo un certo argomento a scapito degli altri, sono stati ripresi ed approfonditi da altri, sia come articoli veri e propri, che nell'area *Amiga software*.

Indubbiamente molto di tutto ciò si deve allo splendido lavoro compiuto da adp e collaboratori (*ADPnetwork*) e che, a quanto mi risulta, non ha eguali in altri paesi, Stati Uniti compresi. Anzi, devo dire che tutti coloro con cui ho parlato negli Stati Uniti ed in Europa di questo progetto, hanno dimostrato un notevole interesse, lamentandosi solo, per così dire, del fatto di non poter avere notizie di prima mano in inglese. Il che mi conferma ancora una volta che forse la scarsa visibilità che hanno molte idee sviluppate nel nostro paese, sia anche dovuta al fatto che pochi nel mondo conoscono l'italiano, riducendo parecchio la loro diffusione al di fuori del nostro paese.

Forte di questa situazione, ho deciso di continuare a sviluppare questa rubrica secondo lo schema già applicato in passato, cioè quello di cercare di toccare un po' tutti gli argomenti, approfondendoli quel tanto che permetta ad altri di riprenderli portando un contributo valido ed originale alla rivista ed ai suoi lettori. Il *C1-Text*, i vari articoli e programmi pubblicati su MC, le lettere che ho ricevuto in questi mesi (mi scuso se non riesco a rispondere a tutti in rubrica

```
struct MsgPort
{
  struct Node   mp_Node;    /* Nodo di aggancio                          */
  UBYTE         mp_Flags;   /* Azioni da prendere in caso di messaggio   */
  UBYTE         mp_SigBit;  /* Numero del segnale in caso di PA_SIGNAL   */
  struct Task  *mp_SigTask; /* Task da segnalare o struttura interruzione */
  struct List   mp_MsgList; /* Lista dei messaggi arrivati               */
}

struct Message
{
  struct Node     mn_Node;      /* Nodo di aggancio                      */
  struct MsgPort *mn_ReplyPort; /* Porta per la risposta                 */
  UWORD           mn_Length;    /* Lunghezza del messaggio in byte       */
}

struct IntuiMessage
{
  struct Message       ExecMessage;     /* Messaggio di tipo "classico"  */
  ULONG                Class;           /* Contiene i flag IDCMP         */
  USHORT               Code;            /* -- Vedi testo dell'articolo -- */
  USHORT               Qualifier;       /* -- Vedi testo dell'articolo -- */
  APTR                 IAddress;        /* Puntatore ad un oggetto       */
  SHORT                MouseX, MouseY;  /* Posizione del mouse           */
  ULONG                Seconds, Micros; /* Orologio dell'Amiga           */
  struct Window       *IDCMPWindow;     /* Finestra a cui msg appartiene */
  struct IntuiMessage *SpecialLink;     /* Riservato per il sistema      */
}
```

Figura 1
Strutture MsgPort, Message ed IntuiMessage.

in tempi brevi, ma lo spazio ed il tempo sono quelli che sono), i sempre più frequenti programmi di *public domain* scritti da italiani, dimostrano, se ce ne fosse bisogno, che non abbiamo nulla da invidiare a Stati Uniti, Inghilterra e Germania, per quello che riguarda l'Amiga.

D'altra parte gli argomenti da trattare sono ancora molti. A parte una buona fetta di Intuition (menu, quadri, aggeggi vari), dobbiamo affrontare ancora discorsi di base relativi ai *fonts* (e quindi gestione dei testi), animazione, musica e suono, *IFF files*, librerie, *device*, *handler*, per non parlare della programmazione multitasking. È quindi confortante per me sapere che alcuni discorsi più tecnici siano sviluppati in parallelo da altri per la gioia di quei lettori che abbiano già una certa esperienza.

Andiamo avanti quindi con IDCMP, come promesso nella scorsa puntata.

Introduzione

In questa diciassettesima puntata, incominceremo a porre le basi per un programma di gestione degli eventi di cui Intuition ci informa tramite appunto la porta *IDCMP*. Per comodità in figura 1 sono riportate di nuovo le strutture principali di cui ci serviremo nel proseguio dell'articolo. In particolare, l'elemento base è la struttura **IntuiMessage**.

Parleremo inoltre del programma di utilità *GREP* e di quali vantaggi esso offre al programmatore ed, in generale, a chi utilizza spesso il CLI.

Torna infine la rubrica dedicata alle lettere dei lettori.

IDCMP

Nella scorsa puntata abbiamo detto che la porta, o meglio *le* porte IDCMP, servono per lo scambio di messaggi tra una applicazione ed Intuition. In particolare è possibile chiedere ad Intuition di avvertirci di tutta una serie di eventi che ci interessano specificandogli la classe degli eventi in questione. Una lista completa delle classi è riportata in figura 2. Vedremo in seguito il significato delle varie classi. Per ora limitiamoci ad accettare il fatto che esiste una serie di eventi a cui corrispondono delle classi e che, quando Intuition ci spedisce un messaggio relativo ad un certo evento, è possibile sapere a che classe appartiene andando a leggere il valore contenuto nel campo **Class** della struttura IntuiMessage.

Costruiamo ora una routine che chiameremo **HandleEvent()** e che ha le seguenti caratteristiche:

- riceve in ingresso il puntatore ad un messaggio di tipo **IntuiMessage**;
- elabora il messaggio, analizza la classe e, in funzione di questa, chiama o

```
****** Classi IDCMP ************************************************************
*                                                                              *
** Classe ******** Valore ****************** Significato ***************** Note *

 SIZEVERIFY      0x00000001  Una finestra sta per essere ridimensionata.
 NEWSIZE         0x00000002  Una finestra è stata ridimensionata.          [A]
 REFRESHWINDOW   0x00000004  La finestra ha bisogno di essere restaurata.  [B]
 MOUSEBUTTONS    0x00000008  Un tasto del mouse è stato premuto.           [C]
 MOUSEMOVE       0x00000010  Riporta tutti i movimenti del mouse.         [D]
 GADGETDOWN      0x00000020  L'utente ha rilasciato un gadget.            [E]
 GADGETUP        0x00000040  L'utente ha selezionato un gadget.           [F]
 REQSET          0x00000080  Un quadro è stato aperto.
 MENUPICK        0x00000100  Un elemento di un menù è stato selezionato.
 CLOSEWINDOW     0x00000200  Il gadget di chiusura finestra è stato selezionato.
 RAWKEY          0x00000400  Un tasto è stato premuto (codice non convertito).
 REQVERIFY       0x00000800  Un quadro sta per essere aperto.
 REQCLEAR        0x00001000  Un quadro è stato chiuso.
 MENUVERIFY      0x00002000  Un menù sta per essere tirato giù.
 NEWPREFS        0x00004000  La configurazione di sistema è stata modificata.
 DISKINSERTED    0x00008000  Un disco è stato inserito nell'unità.
 DISKREMOVED     0x00010000  Un disco è stato rimosso dall'unità.
 WBENCHMESSAGE   0x00020000  Messaggio dal Workbench
 ACTIVEWINDOW    0x00040000  La finestra è stata attivata.
 INACTIVEWINDOW  0x00080000  La finestra è stata disattivata.
 DELTAMOVE       0x00100000  Riporta spostamenti relativi, non assoluti.
 VANILLAKEY      0x00200000  Un tasto è stato premuto (codice convertito).
 INTUITICKS      0x00400000  Un evento contatore è arrivato.               [G]
 LONELYMESSAGE   0x80000000  *** USO RISERVATO ***                         [H]

********** Note ************************************************************
*                                                                              *
********************************************************************************

 [A] Dalla versione 1.2 questo evento è emesso anche se poi la finestra non
     viene effettivamente ridimensionata.
 [B] Ha senso solo per finestre SIMPLE_REFRESH e SMART_REFRESH.
 [C] Solo gli eventi relativi alla pressione od al rilascio di un tasto del
     mouse che NON Hanno altro significato per Intuition vengono segnalati.
 [D] Funziona solo se sono stati impostati anche REPORTMOUSE o FOLLOWMOUSE.
     Inoltre dalla versione 1.2 non lavora mentre i piani dello schermo sono
     bloccati (ridimensionamento od operazioni sui menù).
 [E] Il gadget va creato con GADGETIMMEDIATE impostato.
 [F] Il gadget va creato con RELVERIFY impostato.
 [G] Dalla versione 1.2 gli eventi contatore sono sospesi fintanto che non
     si è risposto a tutti quelli già arrivati.
 [H] Questo bit non è usato dalla IDCMP, ma viene posto a 0 quando Intuition
     manda un messaggio, ed ad 1 quando riceve indietro la risposta.
```

Figura 2 - Classi IDCMP.

```
/*
**  Queste sono le definizioni iniziali ed il prototipo.
*/
 #define GOAHEAD 1
 #define CLOSEME 0
 typedef struct IntuiMessage IMSG;

 int HandleEvent ( IMSG * );

/*
**  Qui inizia la procedura per gestire i messaggi da Intuition.
*/
 HandleEvent(msg)
   IMSG *msg;
 {
   IMSG localmsg;      /* Questa è una fotocopia del messaggio ricevuto */
   int  result;        /* Questo è il valore da restituire al chiamante */
   CopyMem(msg,&localmsg,sizeof(IMSG)); /* Fotocopiamo il messaggio... */
   ReplyMsg(msg);      /* ...così possiamo rispondere subito.           */

   /* ------------------------------------------------------------------ *
    * ATTENZIONE: da questo punto in poi il messaggio puntato da "msg"   *
    *             non è più disponibile a questo task.                   *
    *             Useremo la copia locale salvata in "localmsg".         *
    * ------------------------------------------------------------------ */

   switch (localmsg.Class)
   {
      case GADGETDOWN: result = GadgetDown     (&localmsg);    break;
      case GADGETUP  : result = GadgetUp       (&localmsg);    break;
      case NEWPREFS  : result = NewPrefs       (&localmsg);    break;
      case ...       : result = ...            (   ...   );    break;
      case ...       : result = ...            (   ...   );    break;
      case ...       : result = ...            (   ...   );    break;
      case INTUITICKS: result = IntuiTicks     (&localmsg);    break;
      default        : result = GOAHEAD                  ;    break;
   };
   return (result);
 }

/*
**  Questo è un esempio di prototipo di una procedura specifica.
**  Sotto è riportata una procedura MUTA [dummy o stub] da usare
**  durante i test o per procedure NO-OP (no-operation).
*/
 int IntuiTicks  ( IMSG * );
 IntuiTicks(msg) IMSG *msg; { return(GOAHEAD); }
```

Figura 3 - HandleEvent().

meno una procedura che si occupa di gestire quel particolare evento;
- ritorna **GOAHEAD** per tutti gli eventi salvo che per **CLOSEWINDOW**, per il quale restituisce il valore **CLOSEME**.

Uno scheletro è riportato in figura 3. Come si può vedere in figura, sia la procedura generale che quelle specifiche per evento, accettano in ingresso il puntatore ad una struttura contenente il messaggio di Intuition, e ritornano un valore che indica se si può proseguire oppure è stato selezionato il gadget di chiusura della finestra.

La procedura generale non fa altro che fare una fotocopia del messaggio originale ricevuto da Intuition, in modo da poter poi replicare subito allo stesso come richiesto dalle regole di buona educazione spesso menzionate. Quindi, in base alla classe del messaggio viene chiamata una procedura specifica alla quale viene fornito il puntatore al messaggio duplicato, in modo da metter loro a disposizione tutte le informazioni necessarie ad effettuare il loro lavoro. Queste, a loro volta, ritornano un codice di fine procedura che viene passato dalla **HandleEvent()** al codice chiamante.

Da notare che l'assegnazione a **result** del valore **GOAHEAD** nel caso che una classe non sia tra quelle listate (clausola **default**), ha un duplice scopo:
- nel caso si intenda gestire solo una parte delle possibili classi, serve ad andare avanti qualora arrivi un messaggio non previsto; pur essendo questa una condizione anomala, dato che ci si aspetta di ricevere solo i messaggi esplicitamente richiesti ad Intuition, in generale non c'è motivo di bloccare il programma solo per questo;
- nel caso invece che si intendano gestire tutte le classi, esso è un utile espediente qualora, per un qualche motivo, arrivi un messaggio nuovo come quello che si può ottenere al rilascio di una nuova versione di Intuition.

Ovviamente, in entrambi i casi, si può definire un nuovo valore, diciamo
#define UNKNOWN 2
che può essere usato per avvertire il programma chiamante che è arrivato un messaggio non previsto. In tal caso il chiamante può lanciare un messaggio di *attenzione [warning]* a terminale e poi continuare. Anche se questo valore implica che si può andare comunque avanti, raccomando fortemente di utilizzare la coppia
UNKNOWN|GOAHEAD
e di verificare i singoli bit indipendentemente.

Vediamo ora il codice nel programma principale che si occupa di agganciare il messaggio e di passarlo alla **HandleEvent()**. Il listato è riportato in figura 4, e come si può vedere è particolarmente semplice. In pratica si acquisisce un messaggio dalla porta. Se questa è vuota ci si mette in attesa, altrimenti lo si passa alla procedura di gestione non specializzata. Questa chiamerà la procedura specifica di gestione del messaggio e quindi restituirà un valore al codice chiamante. Se si richiede di chiudere la finestra il ciclo termina, altrimenti si va avanti *ad iterim*.

Tanto per esercitarvi, provate a riscrivere un pezzo di codice equivalente più corretto da un punto di vista della *programmazione strutturata*, evitando cioè il ciclo infinito con istruzione brutale di interruzione. E visto che ci siete, provate anche a scriverne uno il più compatto possibile, come uso dei fanatici del C...

Vediamo infine, come promesso nella 16ª puntata, come si può associare a più finestre la stessa porta utente IDCMP. Chiameremo *porta utente* la **UserPort** e *porta Intuition* la **WindowPort**.

L'allocazione e l'inizializzazione delle porte IDCMP segue le seguenti regole:
- se **OpenWindow()** viene chiamata con un valore nullo per **IDCMPFlags** non viene creata né la porta utente, né quella Intuition;
- se **OpenWindow()** viene chiamata con un valore non nullo per **IDCMPFlags** vengono create entrambe le porte;
- se **ModifyIDCMP()** viene chiamata con un valore nullo per **IDCMPFlags** vengono chiuse entrambe le porte, a meno che i relativi puntatori non siano già nulli;
- se **ModifyIDCMP()** viene chiamta con un valore non nullo per **IDCMPFlags** vengono aperte entrambe le porte a meno che una o tutte e due non siano già aperte.

A questo punto la tecnica per associare a più finestre una stessa porta utente è semplice. Supponiamo di avere già il puntatore alla porta da utilizzare. Questa può essere la porta associata ad un'altra finestra od una porta creata *ad hoc* tramite EXEC. Quello che vogliamo fare è aprire un'altra finestra ed associarvi la stessa porta utente. Per far questo basta:
- aprire la finestra con un valore nullo per **IDCMPFlags**, di modo che Intuition non apra alcuna porta;
- assegnare al campo **UserPort** della

Figura 4
Blocco di gestione messaggi.

```
/*
**  Questo blocco di codice serve a gestire i messaggi che arrivano da
**  Intuition, tenendo presente che questi può spedire più di un messaggio
**  alla volta mentre il programma è in attesa, e quindi bisogna svuotare la
**  coda prima di ritornare a dormire. Ovviamente "w" è il puntatore ad una
**  finestra, ed "imsg" quello ad una struttura IntuiMessage.
*/
 for(;;)
 {
   if   ((imsg = (IMSG *)GetMsg(w->UserPort)) == NULL) WaitPort(w->UserPort);
   else if (HandleEvent(imsg) == CLOSEME) break;
 }
```

Figura 5
Porta utente associata a più finestre.

```
/*
**  w_Color->UserPort è il puntatore alla porta comune
*/
 SharedPort = w_Color->UserPort;

 nw_Black->IDCMPFlags = NULL; /* Chiediamo ad Intuition di non aprire le    */
 nw_White->IDCMPFlags = NULL; /* porte IDCMP per le due finestre secondarie */

 w_Black = (struct Window *)OpenWindow(nw_Black); /* Apriamo ora entrambe   */
 w_White = (struct Window *)OpenWindow(nw_White); /* le finestre secondarie */

 w_Black->UserPort = SharedPort;  /* Copiamo il puntatore alla porta comune */
 w_White->UserPort = SharedPort;  /* nei campi relativi alle nuove finestre */

 ModifyIDCMP(w_Black,BLACKCLASSES); /* Chiediamo ad Intuition di aprire la  */
 ModifyIDCMP(w_White,WHITECLASSES); /* sua porta e formiamo le classi       */

/*
**  A questo punto la stessa porta è condivisa da tre finestre:
**  w_Color, w_Black, w_White. Supponiamo ora di voler chiudere w_Black.
*/
 w_Black->UserPort = NULL; /* Annulliamo il puntatore alla porta utente     */
 CloseWindow(w_Black);     /* Chiudiamo la porta NERA                       */

/*
**  Adesso decidiamo che non vogliamo più ricevere messaggi relativi alla
**  porta BIANCA, ma intendiamo lasciarla aperta.
*/
 w_White->UserPort = NULL;  /* Annulliamo il puntatore alla porta utente    */
 ModifyIDCMP(w_White,NULL); /* Chiudiamo la comunicazione con Intuition     */

/*
**  Da qui in poi ci rimane solo da chiudere le due finestre ancora aperte
**  come sappiamo già fare.
*/
```

```
Questi sono i tre file su cui effettuare la ricerca:

 Linea                                                    Nome

   1     10:00 Ricordarsi di scrivere a Francesca        12Lug89.memo
   2     12:30 Pranzo di lavoro col capo
   3     15:50 Partita Italia-Olanda
   4     18:00 Ti telefona Alfredo
   5     22:30 Stasera c'è Indiana Jones

   1     10:30 Ritira il pacco alla Posta                13Lug89.memo
   2     16:00 Compra il regalo per Francesca
   3     18:20 Stasera viene a cena Alfredo

   1      9:00 Riporta a casa le chiavi dell'ufficio     14Lug89.memo
   2     10:00 Fai lavare la macchina
   3     11:20 Spedisci il pacco a Francesca via corriere
   4     14:30 Prove di Formula 1
   5     21:00 Prepara le valigie per domani

Questa è la prima richiesta effettuata ed il corrispondente risultato della
ricerca effettuata da GREP:

[1] grep Francesca ram:??Lug89.memo

ram:13lug89.memo:
    2:16:00 Compra il regalo per Francesca
ram:14lug89.memo:
    3:11:20 Spedisci il pacco a Francesca via corriere
ram:12lug89.memo:
    1:10:00 Ricordarsi di scrivere a Francesca

Questa è la seconda richiesta effettuata ed il corrispondente risultato della
ricerca effettuata da GREP:

[1] grep "il pacco" ram:??Lug89.memo

ram:13lug89.memo:
    1:10:30 Ritira il pacco alla Posta
ram:14lug89.memo:
    3:11:20 Spedisci il pacco a Francesca via corriere
```

Figura 6 - Esempio di uso di GREP.

Modello di ricerca	Esempi di stringa corrispondente
pippo.?	pippo.c pippo.h pippo.a
pippo.???	pippo.fil pippo.obj pippo.lnk
pippo.#?	pippo. pippo.c pippo.asm pippo.memo
pippo(%|.?|.???)	pippo pippo.h pippo.lnk
pippo.'#0?	pippo.#01 pippo.#0a pippo.#0F
pippo(%|.)(#_)	pippo pippo___ pippo._
pippo(%)|(.#x)	pippo pippo.x pippo.xxx pippo.xxxxxx

Figura 7 - Modelli di ricerca dell AmigaDOS.

Espressione regolare	Esempi di stringa corrispondente
P.zza	Pazza Pezza Puzza Pizza P.zza P4zza
etc\.\.\.	etc...
12\.78	12.78
12.78	12a78 12Q78 12:78 12/78 12+78
[CcPp]ane	Cane cane Pane pane
et[àa]	età eta
AZ[0-9][0-9][0-9]	AZ412 AZ540 AZ432 AZ212
AZ[0-2][5-9][5-9]	AZ067 AZ155 AZ298 AZ159
12[!1-9]78	12A78 12078 12o78 12ù78 12§78
(*27	27 (27 ((27 (((((27
(+27	(27 ((27 (((27 ((((27
Mo.*re	More Morire Mostrare Moderare
Ce.+to	Cento Celebrato Ce l'ho portato
Spezza\x03	Spezza<Ctrl-C>
12\tMario\tRossi	12<tab>Mario<tab>Rossi

Figura 8 - Espressioni regolari di GREP.

nuova finestra il puntatore della porta *comune*;

- chiamare **ModifyIDCMP()** passandogli le classi di eventi a cui siamo interessati.

A questo punto Intuition si accorgerà che la porta utente è già assegnata e si limiterà a creare ed inizializzare solo la sua porta.

È importante tener presente che la porta comune appartiene a più finestre, ora, e quindi bisogna fare attenzione a non chiuderla fintanto che anche una sola delle finestre che la utilizza possa avere ancora bisogno di essa. Intuition non tiene traccia del fatto che tale porta è condivisa da più finestre, infatti. La gestione del relativo puntatore è quindi ora affidata solo a voi.

Questo vuol dire che prima di chiamare la **CloseWindow()** o la **ModifyIDCMP(w,NULL)** per una certa finestra **w**, è necessario annullare *a mano* il puntatore **UserPort** relativo, altrimenti si rischia di andare in GURU!

Note

1. *GREP* è fornito insieme al Lattice C 5.xx ma programmi analoghi sono disponibili anche tra quelli PD su vari BBS.

La figura 5 riporta uno scheletro di programma relativo alle operazioni or ora descritte. Ovviamente sono riportate solo le istruzioni chiave. Il resto lo si lascia alla fantasia del lettore.

GREP

Vediamo ora un nuovo programma di utilità per chi sviluppa programmi e, in generale, particolarmente utile per chi lavora con testi da CLI: *GREP* (vedi nota 1).

GREP sta per *Global Regular Expression Search and Print*, cioè ricerca e stampa di espressioni regolari globali. Questo altisonante e non certo breve nome, in realtà nasconde una funzionalità alquanto semplice ma in effetti molto utile, e cioè la capacità di cercare e mostrare a video tutte le linee di uno o più file specificati, che contengano una certa stringa di caratteri. Ad esempio, tutte le righe in **grafica.c** che contengono **gfx**.

Vedremo tra breve come *GREP* permetta molte opzioni di ricerca ed accetti specifiche di stringa alquanto complesse, dette appunto *espressioni regolari [regular expression]*.

Tanto per cominciare vediamo in tabella A la sintassi.

Prima di entrare nel dettaglio vediamo un esempio di come si comporta *GREP* in un caso semplice. Fate riferimento alla figura 6. Come si può notare, è possibile effettuare la ricerca su più

```
GREP [opzioni] espressione file1..filen

dove  opzioni       servono a specificare a GREP il comportamento da
                    seguire durante la ricerca oppure alla visualizzazione
                    delle linee identificate

      espressione   serve a specificare a GREP le caratteristiche della
                    stringa di caratteri da ricercare

      file1..filen  è la lista dei file in cui cercare la stringa specificata
```

Tabella A

file anche utilizzando i caratteri speciali dell'AmigaDOS per la definizione dei modelli di ricerca *[pattern]*. È quanto abbiamo riportato in tabella B.

Alcuni esempi sono riportati in figura 7. Inoltre, come già accade per altri comandi a cui siamo abituati, se l'espressione regolare contiene degli spazi bianchi o altri caratteri interpretabili dallo *Shell* come i simboli di reindirizzamento, è necessario includerla fra virgolette (doppi apici). Esistono tuttavia alcuni caratteri speciali che, se incontrati da *GREP* all'interno di una espressione regolare, acquistano un significato particolare. Tale comportamento non è inibito dalle virgolette, ma può essere sospeso per mezzo di un particolare carattere di controllo detto disabilitatore *[escape character]*. Un esempio di disabilitatore lo abbiamo or ora visto nella tabella precedente nel caso dei modelli di ricerca dell'AmigaDos, e cioè *l'apostrofo*.

I caratteri speciali

Vediamo quindi quali sono questi caratteri speciali riconosciuti da *GREP*. Attenzione: non confondete questi caratteri con quelli usati nei modelli AmigaDOS per i nomi dei file. Sono due cose completamente distinte.

\
è il disabilitatore. Qualunque carattere speciale può essere riconvertito ad un carattere normale facente parte della stringa da ricercare semplicemente facendolo precedere da una barra diagonale inversa *[backslash]*. In particolare, se la stringa deve contenere tale barra basterà scrivere \\. Questo carattere, se seguito da alcuni specifici caratteri, forma alcune sequenze speciali *[escape sequences]* che vedremo più avanti.

.
ha la stessa funzione del punto interrogativo nei modelli di ricerca dell'AmigaDOS, e cioè rappresenta un singolo carattere di un qualunque tipo.

[]
servono a definire una classe di caratteri *[character class]*, cioè un insieme di caratteri ognuno dei quali può trovarsi in quella specifica posizione nella stringa. Allora **[Ss]ole** rappresenta sia la parola **Sole**, sia la parola **sole**. Vedremo più in dettaglio l'utilizzo delle classi di caratteri nella prossima puntata.

*
vuol dire *zero o più ripetizioni di*, un po' come # nell'AmigaDOS. Attenzione però: questo carattere va posto **dopo** il carattere da ripetere, non prima come in AmigaDOS. Quindi **0*** in *GREP* si comporta come **#0** in AmigaDOS.

```
In questa tabella il simbolo <p> rappesenta un modello valido mentre
<c> rappresenta un carattere

?       rappresenta un singolo carattere di qualsiasi tipo
%       rappresenta la stringa nulla
#<p>    rappresenta zero o più ripetizioni del modello successivo
'<c>    trasforma un carattere speciale in un carattere semplice
<p><p>  rappesenta la concatenazione di due modelli
<p>|<p> rappresenta la somma logica (OR) di due modelli
(....)  serve a raggruppare una sequenza di caratteri semplici e speciali
        a formare un modello
```

Tabella B

+
vuol dire *una o più ripetizione di*. Non ha equivalente nell'AmigaDOS. Attenzione: anche questo carattere va posto **dopo** il carattere da ripetere: **0+** in *GREP* si comporta come **0#0** in AmigaDOS.

se usata come *primo* carattere della specifica di ricerca, significa che la stringa che segue deve partire a *inizio riga*, cioè da colonna zero (od uno, a seconda di come si conta).

Casella Postale

Questo mese risponderò ad una lettera che ho ricevuto da Milano alla fine di settembre e che mi è stata spedita il 27 luglio di quest'anno. Le vacanze estive e l'efficienza del nostro servizio postale hanno rappresentanto un'accoppiata *perdente* per Matteo Olivieri. A questo si aggiunga il fatto che un articolo va consegnato due mesi prima della pubblicazione...

Beh, penso che oramai fosse stata data per dispersa dall'autore. Meglio tardi che mai, giusto?

Guasto al Blitter?

Caro Dario, credo di aver trovato una imprecisione in quanto tu scrivi nella puntata di «Programmare in C su Amiga» pubblicata su MC 87 (luglio/agosto). Riguardo alle tecniche di riempimento, tu affermi che la tecnica a definizione d'area è «più veloce della precedente [quella a macchia d'olio], ma ha lo svantaggio di richiedere l'allocazione di un raster temporaneo».

Ora il fatto è che, a quanto mi risulta dalle prove effettuate sul mio Amiga 2000, anche la tecnica a macchia d'olio, cioè quella che utilizza la funzione **Flood()***, richiede l'allocazione di un raster temporaneo, pena un rallentamento pazzesco delle prestazioni o, addirittura, un non funzionamento totale. Provate, infatti, a scrivere un semplicissimo programma che apre uno schermo 320×256, una finestra a tutto schermo, preleva l'indirizzo della* **RastPort** *della finestra e poi prosegue con queste poche linee:*

```
/* il colore di bordo è uguale al colore di disegno */
SetAPen(rp, 1);
SetOPen(rp, 1);
/* disegna un cerchio di centro (160,128) e raggio 50:
   è una macro, includere <graphics/gfxmacros.h> */
DrawCircle(rp, 0, 160, 128, 50);
/* riempe il cerchio con il modo a confinamento */
Flood(rp, 0, 160, 128);
```

Compilatelo e fatelo partire: se vedrete quello che ho visto io (lo spero, altrimenti vuol dire che è guasto il mio blitter...!), probabilmente rimpiangerete come me il buon vecchio C64: sissignori, il **Flood()** *ha impiegato ben 6 (sei!!!) secondi per riempire un cerchio di 50 pixel di raggio. Ma scherziamo, è roba da Amiga questa? E tutte le storie sulla velocità del blitter? No, non è possibile, proviamo il modo a simpatia di colore, cioè sostituiamo lo* **0***del* **Flood()** *con un* **1** *risultato: il cerchio non si riempie affatto. Il mistero continua. Avendo letto da qualche parte come funziona la tecnica a definizione d'area, provo con quella, e uso l'****AreaCircle()*** *al posto del* **DrawCircle()** *e del* **Flood()** *(no, non provate con una semplice sostituzione, bisogna fare un sacco di altre cose): stavolta il riempimento è istantaneo: Dario, avevi ragione, la tecnica a definizione d'area è più veloce... Insomma, prova e riprova ho scoperto che se si mantiene lo stesso raster temporaneo che si usa per l'***AreaCircle()** *anche per il* **Flood()***, quest'ultimo funziona a dovere, sia con* **0** *che con l'***1***, ed è ragionevolmente veloce (qualche decimo di secondo), anche se, come dici tu, più lento dei vari* **AreaDraw()** *e compagnia.*

Dunque, provate a scrivere:

```
BYTE *rast;
struct TmpRas tmp_ras;
...
rast = AllocRaster(320, 256);
rp->TmpRas = InitTmpRas(&tmp_ras, rast, RASSIZE(320, 256));
/* qui le stesse righe di prima */
FreeRaster(rast, 320, 256);
```

$

se usato come *ultimo* carattere della specifica di ricerca, significa che la stringa che segue deve trovarsi a *fine riga*.

In figura 8 sono riportati alcuni esempi di utilizzo dei caratteri speciali di *GREP*. Nella prossima puntata entreremo nel dettaglio per quello che riguarda le espressioni regolari.

Le sequenze speciali

Come detto in precedenza, il carattere disabilitatore può anche essere usato per formare delle sequenze speciali che permettono di specificare caratteri non presenti sulla tastiera o di controllo:

\n

rappresenta il carattere di *a capo [newline]*.

\s

rappresenta un *singolo* spazio bianco.

\b

rappresenta il carattere di retrocancellazione (o spazio indietro) *[backspace]*.

\t

rappresenta il carattere di tabulazione *[tab]*.

\x_

rappresenta il carattere corrispondente al valore esadecimale di due cifre che segue la **x** e qui rappresentato da due trattini di sottolineatura.

La sequenza che inizia con **\x**, è particolarmente utile in due situazioni:
- quando la stringa da cercare contiene dei caratteri non disponibili da tastiera;
- quando la stringa si trova in un file generato da un elaboratore di testi *[word processor]* e stiamo cercando una specifica sequenza di controllo per la stampante.

Conclusione

Nella prossima puntata incominceremo a parlare di menu, prima da un punto di vista generale e poi, nelle puntate successive, cercheremo di renderne semplice l'utilizzo da C nei casi più classici, per mezzo di una particolare tecnica basata su strutture modello.

Continueremo anche a parlare di *GREP* e, se ci sarà sufficiente spazio, vedremo la seconda scheda tecnica. A presto e buon fine anno.

---

e vedrete che tutto funzionerà a meraviglia, e, soprattutto, velocemente.

Ah, dimenticavo. Naturalmente il programmino d'esempio in figura 6 (sempre MC 87) non funziona, perché disegna il perimetro del rombo con il colore 1, ma usa il colore 3 come colore di contorno a cui fermarsi. Risultato, la **Flood()** *riempie tutta la finestra, tra l'altro con la famosa velocità da lumaca di cui è capace quando manca il raster temporaneo.*

Non so se si tratti di un bug nella funzione **Flood()** *o più semplicemente questa sia la normale procedura da seguire e, in questo caso, non so neanche se sia l'unica, sta di fatto che, anche se non ho potuto verificare la cosa su altre macchine (siamo in estate, e tutti gli amighi, gli amici con l'Amiga, sono in vacanza), non credo proprio che si tratti di un guasto al mio blitter...*

Complimenti, nonostante tutto, per la rubrica, e un cordiale saluto.

Matteo Olivieri, Milano

Appena letta questa lettera mi sono messo davanti all'Amiga ed ho effettuato una serie di prove sulla **Flood()** e su altre funzioni grafiche collegate.

Innanzitutto confermo quanto riportato da Matteo, e cioè:
1. i tempi di risposta della **Flood()** con e senza *raster* temporaneo;
2. il non funzionamento della stessa nel modo 1 senza *raster* temporaneo.

Per quello che riguarda il secondo punto, non si tratta tuttavia di una mia imprecisione, quanto di un vero e proprio baco della **Flood()** stessa, a quanto pare mai rilevato prima (il che, sinceramente, mi sembra alquanto strano arrivati alla versione 1.3 del sistema operativo).

In quanto al primo punto, ribadisco che la **Flood()** non prevede, almeno ufficialmente, e comunque di sicuro non quella della versione 1.1, l'utilizzo del *raster* temporaneo. Nella documentazione ufficiale della versione 1.2 si accenna ad una modifica effettuata sulla **Flood()** per raddoppiarne la velocità, ma non si fa alcun riferimento all'uso di un *raster* temporaneo, e comunque un fattore due rispetto alla versione 1.1 non può aver alcuna relazione con l'enorme differenza riscontrata nelle prove effettuate da Matteo e me. Il fatto che in effetti il suo comportamento cambi sostanzialmente con l'aggiunta del **TmpRas** mi fa pensare ad una modifica non documentata (e quindi in teoria non supportata in futuro, ma non credo sia questo il caso), e tra l'altro forse poco conosciuta anche a molti programmatori commerciali, i quali, almeno negli Stati Uniti, sono in contatto con il CATS (il gruppo di supporto tecnico della CBM) e dovrebbero quindi ricevere informazioni non disponibili al programmatore dilettante. Il dubbio mi è venuto facendo un paio di prove sul riempimento di un rettangolo di 100×100 pixel.

Il risultato curioso non è tanto che il tempo di riempimento sia di circa 2 secondi e mezzo senza **TmpRas** in modo **0**, e meno di un decimo di secondo se si alloca il *raster* temporaneo, ma che *DPaintIII* nelle stesse condizioni, ci metta tra uno e due secondi, sia cioè più veloce del caso senza *raster*, ma enormemente più lento di quello con il *raster* allocato. Può essere, Matteo, che tu abbia scoperto qualcosa di nuovo, o magari disponibile solo a pochi addetti ai lavori.

In quanto ai sei secondi da te riportati, il cerchio sembra essere una delle figure *più lente* da riempire, tra quelle elementari.

I rettangoli sono molto più rapidi, ma in tal caso si può usare la velocissima **RectFill()**. Ti consiglio comunque di ridifferenziare la **OPen** e la **APen** prima di iniziare il riempimento.

Nel fare le prove sulla **Flood()** ho anche scoperto un altro problema, e cioè che se l'area da riempire non è chiusa su un lato, la funzione sembra continuare il riempimento all'infinito nel *limbo* oltre i bordi. La finestra in questione era tra l'altro una GZZ, una cioè delle più «sicure» da usare. Ho paura che sia necessario bloccare a mano i piani (con le funzioni fornite dalla **layers.library**) per evitare ciò. Peccato, speravo che le funzioni grafiche fossero più *intelligenti*.

Per l'esercizio in figura 6 ovviamente hai ragione, si tratta di un errore nel riportare il pezzo di programma. Il tutto va modificato così: ▼

```
...
oldpen = rp->FgPen;     /* salva la vecchia penna primaria              */
SetAPen(rp,rp->AOlPen); /* usa la penna di contorno come penna primaria */
Move( ...
...
Flood( ...
SetAPen(rp,oldpen);     /* reimposta la penna primaria come era prima   */
...
```