Gli argomenti trattati questo mese nella rubrica Software Amiga riguardano lo standard Ansi della console e un programma esercizio, preparato dal sottoscritto per mostrare alcune delle feature della programmazione in C trattate nella apposita rubrica da Dario de Judicibus. Ciò (anche) per lamentare la carenza di elaborati pubblicabili ad opera dei lettori, che nonostante i nostri preavvisi, si ostinano a inviare materiale o troppo scadente o, valido, ma «troppo ingombrante». Cito, uno per tutti, il lavoro di Michele Jacobellis: una splendida interfaccia mouse disco che, tra programma e include, ha il piccolo svantaggio di essere lungo quasi un migliaio di linee. Sarebbe stato meglio (nonché pubblicato) un articolo che spiegava, con piccoli esempi, come procedere per realizzare una interfaccia simile.
Del Mangrella, di contro, ne riparleremo dal prossimo mese.
È già arrivato in redazione un altro plico...

Sequenze Ansi

di Riccardo Torrini - Firenze

In seguito all'uscita della versione 1.3 abbiamo potuto apprezzare i nuovi comandi e le modifiche a quelli esistenti. Molto interessante è la possibilità di creare dei file BATCH (file testo che contengono una lista di comandi eseguibili) che possono essere eseguiti senza l'ausilio del comando EXECUTE, grazie al flag «S» (script file) da abilitare purtroppo dopo ogni modifica al file.

Poiché fin dalla precedente versione il comando ECHO permetteva l'inserimento di seguenze di Escape Ansi per la gestione della console, vediamo come con tali codici si possa cancellare lo schermo e ottenere una finestra CLI senza bordo da 80 caratteri veri (ottima per visualizzare file impaginati da altri).

Nella tabella A i codici più utili per la gestione dello schermo.

Note

La sequenza *e[viene interpretata dal comando ECHO come il codice 9B esadecimale (esc=1B(h), 80(h)+1B(h) =9B(h)).

Le parentesi indicano un valore numerico e non devono essere inserite nel comano ECHO.

Le lettere «H» e «J» devono essere impostate maiuscole, tutte le altre minuscole.

Quindi per cancellare lo schermo basta creare un file di nome CLS che posizioni a riga 1 colonna 1 e cancelli fino alla fine della finestra.

```
CLS : Echo "*e [1;1H*e[J" noline.
```

Per ottenere una finestra senza bordo avremo bisogno di due file, il primo che crea una finestra di dimensioni note ed il secondo che toglie il bordo, questo perché non possiamo determinare la dimensione di una finestra modificata manualmente.

```
Off : NewCLI «NEWCON:0/77/640/179/Finestra senza bordo» from S:Off-Startup
```

Questo crea una finestra CLI alta 20 righe col bordo (diventeranno 21 senza bordo) posizionata in fondo allo schermo di Workbench

```
Off-Startup:
C:Echo "*e[0x*e[10y*e[80u*e[21t*e[1;1H*e[J*e[11y*e[J" noline.
```

Con questo invece modifico l'offset sinistro portandolo da 4 a 0, aumento il numero di caratteri per riga a 80 e cancello lo schermo ottenendo anche la cancellazione dei bordi creati aprendo la nuova finestra.

Coloro che ancora non avessero la versione 1.3 dovranno omettere la dichiarazione «NOLINE» dopo il comando ECHO, modificare NEWCON: in CON: e eseguire il tutto con EXECUTE OFF.

Poiché CLS è un comando di una riga possiamo, per ottenere una maggiore velocità di esecuzione, utilizzare il comando Alias

```
Alias Cls Echo "*e(1;1H*e[J" noline
```

Tabella A

```
CH : Cursor Home      : *e[1;1H
CP : Cursor Position  : *e[(x);(y)H   (x) colonna, (y) riga
CB : Clear to Bottom  : *e[J
CS : Clear Screen     : *e[1;1H*e[J
LO : Left Offset      : *e[(n)x       (n) in pixels, defaul 4
TO : Top Offset       : *e[(n)y       (n) in pixels, default 11
LL : Line Length      : *e[(n)u       (n) in caratteri
PL : Page Length      : *e[(n)t       (n) in linee
```

e anche

Alias Off Echo NewCLI ... eccetera

e magari rendere anche ECHO residente. Più veloce di così...

Sempre restando in tema di Alias ecco un test velocissimo per sapere se un disco ha un Boot-Block standard:

Alias? Install DF []: Check

La sintassi per un corretto utilizzo è:

? 0 <RETURN> : Controlla il disco del drive DF0:
? 1 <RETURN> : Controlla il disco del drive DF1:
? 2 <RETURN> : Come sopra x DF2:
? 3 <RETURN> : Come sopra x DF3:

Nota

Le parentesi quadre [] permettono il passaggio di parametri nei comandi creati con alias, non credo si possa assegnare un valore di default.

Replace

di Andrea de Prisco

Prima di commentare il listato, è necessario definire con precisione cosa «combina» il programma Replace. Innanzitutto si tratta di un comando CLI che potremo aggiungere alla nostra directory «c:» per renderlo disponibile come vero e proprio comando di sistema. È stato compilato col Lattice C 4.0 direttamente col comando LC seguito dall'opzione −L che, come noto, provvede anche alla chiamata del Linker. Più semplice di così...

Come i più attenti avranno capito, Replace permette la ricerca e sostituzione di stringhe all'interno di un file, creandone uno nuovo con le modifiche attuate. La sintassi è molto semplice:

Replace SourceFile DestFile str1 str2.

Dove SourceFile è il file da modificare (quindi già esistente), DestFile è un nome per il file modificato (va bene anche lo stesso nome, ma, attenzione, così perdiamo l'originale!!!) mentre str1 e str2 sono la stringa da ricercare e la stringa da sostituire per ogni occorrenza della prima. Ovviamente le due stringhe possono avere anche lunghezze diverse nonché contenere anche codici esadecimali per individuare caratteri più ostici. È anche possibile inserire una stringa vuota in modo da eliminare stringhe dal testo senza sostituirle con alcunché. Il carattere di controllo per queste selezioni «strane» è il simbolo del dollaro «$». Un dollaro seguito da una coppia numerica esadecimale individua il carattere dal codice indicato. Se invece intendiamo cercare o sostituire proprio il carattere dollaro basterà usare la coppia $$. Facciamo qualche esempio: sul nostro disco abbiamo il file denominato «testo». Decidiamo, ad esempio, di sostituire tutte le occorrenze della parola «però» con occorrenze di «ma». Il nuovo file, come al solito, lo chiameremo «pippo». Scriveremo:

Replace testo pippo però ma

```
/************************************
*                                   *
*                                   *
*           R E P L A C E           *
*                                   *
*                                   *
*       ----------------------      *
*       (c) 1989  ADP  SOFTWARE     *
*       ----------------------      *
*                                   *
*                                   *
************************************/


#include "exec/types.h"
#include "exec/memory.h"
#include "libraries/dosextens.h"
#include "stdio.h"

void                        Abort(char *, int);
long                        len;
extern struct FileHandle   *Open();
char                       *sorgente;
struct FileHandle          *infh;

VOID main(argc,argv)
int argc;
char **argv;
{

int                   i,j,k,nchr,x,y;
int                   da=0,a=0;
char                  buffin[BUFSIZ],buffout[BUFSIZ];
struct FileInfoBlock  *info;
struct FileLock       *lock;
struct FileHandle     *outfh;

if (argc<2 || argv[1][0] == '?')
Abort("Uso: Replace Source Dest str1 str2\n\n** (c) 1989  ADP SOFTWARE **",0);

for (i=3;i<=4;i++)
for (j=0;j<strlen(argv[i]);j++)

  if (argv[i][j] == '$')
     {
     if (argv[i][j-1] == '$' || argv[i][j-1] == '\0')
         for (k=j;k<strlen(argv[i]);k++) argv[i][k] = argv[i][k+1];
     else
         {
         x = argv[i][j-1];
         y = argv[i][j-2];
         argv[i][j] = (x-((x<58) ? 48 : 55))*16+(y-((y<58) ? 48 : 55));
         for (k=j+1;k<strlen(argv[i])-1;k++) argv[i][k] = argv[i][k+2];
         }
     }

info = (struct FileInfoBlock *)AllocMem(
                sizeof(struct FileInfoBlock),MEMF_CLEAR);

k=j=0;
lock = (struct FileLock *)Lock(argv[1],ACCESS_READ);
Examine(lock,info);

len = info->fib_Size;

FreeMem(info,sizeof(struct FileInfoBlock));

UnLock(lock);


infh = Open(argv[1],MODE_OLDFILE);
if (infh == 0) Abort("Source File Not Found",0);
```

Come vi avevo detto il tutto è molto intuitivo. Proviamo ora a sostituire tutti i Carriage Return ($0D) con più amichevoli New Line ($0A):

```
Replace testo pippo $0D $OA
```

oppure a togliere tutte le occorrenze della parola «buongiorno»:

```
Replace testo pippo buongiorno $.
```

Ovviamente qualsiasi codice esadecimale può trovarsi anche in mezzo a caratteri più «umani», ad esempio possiamo fare la sostituzione:

```
Replace testo pippo ciao$20cara bye$20bye
```

dove, come è noto, $20 rappresenta il codice ASCII di uno spazio. Attenzione ad usare sempre maiuscole per le cifre alfabetiche dei caratteri esadecimali: nessun controllo è eseguito per questi input.

```
sorgente = (char *)AllocMem(len,MEMF_CLEAR);
if (sorgente == 0) Abort("AmigaDOS Memory Fault... Sorry.",1);

while ((nchr=Read(infh,buffin,BUFSIZ))!=0)
        for (i=0; i<nchr; i++) sorgente[j++] = buffin[i];
Close(infh);
da=j=0;
outfh = Open(argv[2],MODE_NEWFILE);
if (outfh == 0) Abort("Dest File Error",2);

do
  {
  a=Search(sorgente,argv[3],da,len);

  if (a!=da) for (i=da;i<a;i++)
     {
     buffout[j++]=sorgente[i];
     if (j == BUFSIZ)
              {
              Write(outfh,buffout,BUFSIZ);
              j=0;
              }
     }
  if (a!=len+1)
     {
     printf("N. Sostituzioni: %d\r",++k);
     for (i=0;i<strlen(argv[4]);i++)
        {
        buffout[j++]=argv[4][i];
        if (j == BUFSIZ)
              {
              Write(outfh,buffout,BUFSIZ);
              j=0;
              }
        }
     }

  da = a+strlen(argv[3]);

  } while (a!=len+1);

if (j!=0) Write(outfh,buffout,j-1);
Close(outfh);
FreeMem(sorgente,len);
printf("\n");

} /* End of Program */


Search(array,testo,start,stop)
char    array[],testo[];
long    start,stop;
{
int     len;
long    i,k;
BOOL    cont=TRUE;

len = strlen(testo);

for (i=start;i<stop-len;i++)
   {
   for (k=0; k<len; k++) cont = cont && (array[i+k]==testo[k]);
   if (cont) return(i);
   else cont = TRUE;
   }
return(stop+1);
}



void Abort(ErrorString,QuitCode)
char ErrorString[];
int  QuitCode;
{
printf("\n\nReplace_Error: %s\n\n",ErrorString);
if (QuitCode == 2) FreeMem(sorgente,len);
if (QuitCode == 1) Close(infh);
Exit(QuitCode);
```

Descrizione del programma

Le prime linee del programma C listato in queste pagine, sono classici include che ci permettono di utilizzare tipi e definizioni all'interno del programma. La prima serie di dichiarazioni sono poste fuori dal corpo del programma essendo globali tanto al main quanto alle due subroutine utilizzate. Dopo il controllo e la traduzione dell'input il file sorgente viene caricato in una zona di memoria allocata col comando AllocMen e analizzato e scaricato nel file destinazione man mano che avvengono le sostituzioni. Con «traduzione» degli input si intende la trasformazione di eventuali codici preceduti dal simbolo «$» negli effettivi valori ASCII.

Per allocare una quantità di memoria necessaria e sufficiente a contenere l'intero file sorgente, la cosa più semplice da fare è vedere quanto è lungo il file in questione ed allocare una pari quantità di memoria. Per fare questo utilizziamo la struttura FileInfoBlock relativa al file, dapprima utilizzando la funzione Lock (per... acchiapparla) e poi la funzione Examine per saperne di più.

Segue l'apertura del file sorgente e l'allocazione della giusta quantità di memoria. Con le linee successive (ma state seguendo sul listato?) non facciamo altro che trasferire quanto ottenuto dal file nella memoria all'uopo allocata. Il rimanente listato, che non commenteremo ulteriormente, non fa altro che effettuare le sostituzioni avvalendosi della subroutine Search (definita di seguito) la quale accetta quattro parametri in ingresso e restituisce un valore numerico. I parametri sono: array nel quale cercare, stringa da cercare, posizione iniziale e posizione finale in cui effettuare la ricerca. Il valore ritornato è naturalmente la posizione in cui la stringa è stata trovata oppure la posizione finale più uno se non viene trovata la stringa nel range indicato.

L'ultima funzione Abort, serve per uscire dal programma in caso di fallimento e chiudere le cose lasciate aperte o occupate. Nella fattispecie memoria allocata e file d'ingresso. I parametri passati a questa funzione sono due, il messaggio d'errore che sarà stampato su video e codice d'errore che sarà restituito dal programma e serve per sapere cosa è rimasto «aperto». Credo di aver detto tutto: alla prossima... MC