SF-Search File

di Andrea Suatoni - Roma

L'aumento degli Amiga in circolazione ha comportato anche una maggiore diffusione delle espansioni disponibili per questa macchina. Fra queste, quella che indubbiamente sta riscuotendo maggiore successo è l'Hard Disk, soprattutto dopo l'uscita della versione 1.3 del sistema operativo che rende disponibile non solo un nuovo e veloce file system, il Fast File System (FFS), ma soprattutto la possibilità di effettuare il bootstrap da un device diverso dal solito DF0: quale può essere, per l'appunto, un HD. La maggiore disponibilità di memoria di massa è indubbiamente un grosso vantaggio ma, a volte, può capitare di non sapere più in quale directory un certo file si trovi. Chi ha usato un PC sa che esiste una serie di utility create da Peter Norton, il «Guru» del mondo dei PC. Tra queste utility una delle più interessanti è sicuramente FF (File Find) che permette di cercare, nell'ambito di un disco, un file o anche un insieme di file. Su Amiga sono stati sviluppati simili programmi di pubblico dominio, ma, per un verso o per l'altro, nessuno di questi possiede la flessibilità dell'utility di Norton. Per questo motivo ho ideato SF (Search File).

L'ARP.library

L'idea mi fu suggerita dalla lettura del manuale per il programmatore della «ARP.library», libreria sulla quale è basato in parte SF. Per coloro che ancora non lo sapessero, l'ARP (acronimo che sta per AmigaDOS Replacement Project o, per la versione 1.3 di ARP, AmigaDOS Resource Project) consiste in un insieme di comandi AmigaDOS che sostituiscono ed ampliano quelli già esistenti e in una serie di file, specifici per i linguaggi Basic, C e Modula 2, che permettono l'utilizzo delle funzioni contenute nella libreria ARP.library. Di questa libreria è stata rilasciata ultimamente la versione 1.3 che, rispetto alla precedente versione 1.1, corregge alcuni errori ed aggiunge nuove funzionalità. Per quanto riguarda SF, le differenze tra le due versioni sono minime e verrano spiegate mano a mano che si presenteranno.

Uno dei vantaggi di questa libreria è quello di essere «shared» (condivisa): questo vuol dire che, una volta caricata in memoria, la libreria rimarrà a disposizione di tutti quei programmi che ne richiederanno l'uso, senza per questo avere duplicazioni di codice, cosa che invece accade normalmente quando utilizziamo una libreria «linked» (agganciata, che brutta italianizzazione!) come lo è, ad esempio, la libreria standard del compilatore C.

Librerie «shared» e «linked»

Apro una piccola parentesi, a favore di chi non lo sapesse, per spiegare la differenza tra librerie «shared» e «linked». Tale differenza consiste nel fatto che, nel caso delle librerie «shared», le chiamate alle funzioni della libreria vengono normalmente risolte tramite degli indirizzamenti indiretti, ovvero tramite «offset» (spiazzamenti, altra terribile italianizzazione) rispetto all'indirizzo base della libreria residente in memoria. Chiunque abbia utilizzato le librerie di sistema di Amiga ha, probabilmente senza saperlo, implicitamente utilizzato delle librerie «shared». Per le librerie «linked», invece, il discorso è diverso: in questo caso, infatti, il linker accoda al nostro programma le routine di cui il programma stesso necessita. È importante sapere che, se la libreria è stata ben strutturata, solo i moduli contenenti le funzioni utilizzate saranno estratti dalla libreria e inclusi nel codice eseguibile. Si possono trarre, quindi, le seguenti conclusioni:

— se utilizziamo delle librerie «shared», otterremo dei file eseguibili più compatti, avremo un'unica copia in memoria delle librerie, ma saremo costretti a tenere disponibili le librerie sul disco (più precisamente nella directory identificata dal nome logico LIBS:) e dovremo avere abbastanza memoria per caricare l'intera libreria. Questo perché Exec non può sapere a priori di quali routine il nostro programma avrà bisogno;

— se utilizziamo delle librerie «linked», otterremo dei file eseguibili meno compatti, avremo delle duplicazioni di codice quando più programmi chiamano la stessa funzione (printf(), per esempio) ma non dovremo più avere disponibili le librerie a «run-time» (esecuzione), e,

*Alla vostra sinistra potete ammirare un esempio di output del programma SF. Si tratta della interminabile lista di file contenuti in tutte le directory e subdirectory dell'HD di AdP (più di 2000 file per circa 10 metri di stampato) ottenuto semplicemente con SF.**

È disponibile, presso la redazione, il disco con il programma pubblicato in questa rubrica. Le istruzioni per l'acquisto e l'elenco degli altri programmi disponibili sono a pag. 263.

```
/*************************************************************
 *                                                           *
 *                      S F - Search File                    *
 *                                                           *
 *                    (c)1989 Andrea Suatoni                 *
 *                                                           *
 *************************************************************/

#include <exec/types.h>
#include <exec/lists.h>
#include <exec/memory.h>
#include <arpfunctions.h>
#include <string.h>
#include <libraries/arpbase.h>
#include <libraries/dosextens.h>

#ifdef   LATTICE
#include <proto/dos.h>
#include <proto/arp.h>
#include <proto/exec.h>
#endif

#define  NO_ERROR              0
#define  ERROR_NO_MEM          -101

typedef
  struct
    {
    struct MinNode  dir_Node;
    LONG            PathLen;
    STRPTR          PathName;
    }
  DIR_ENTRY;

GLOBAL VOID (*_ONBREAK)();

struct MinList  DirList;

VOID Usage()
{
  Puts("\nUso: SF PAT <PathName Pattern> FILES/S DIRS/S QUICK/S NOROOT/S\n");
  Puts("  PAT <PathName Pattern> = pattern di ricerca (ARP o AmigaDOS)");
  Puts("  FILES                  = cerca solo tra i file");
  Puts("  DIRS                   = cerca solo tra le directory");
  Puts("  QUICK                  = visualizza solo i nomi");
  Puts("  NOROOT                 = inizia la ricerca a partire dalla");
  Puts("                           directory corrente anziche' dalla root");
}

STRPTR StrUpper(Str)
REGISTER STRPTR  Str;
{
  REGISTER STRPTR  r = Str;

  do
    {
    *Str = Toupper(*Str);
    }

  while (*(++Str));
  return(r);
}

LONG AddDirEntry(PathNode, Dir)
REGISTER STRPTR  PathNode,
                 Dir;
{
  REGISTER DIR_ENTRY  *Entry;

  if ((Entry =
       (DIR_ENTRY *) AllocMem(sizeof(DIR_ENTRY), MEMF_CLEAR)) == NULL)
    return(ERROR_NO_MEM);
  Entry->PathLen = strlen(PathNode) + strlen(Dir) + 2;
  if ((Entry->PathName =
       (STRPTR) AllocMem(Entry->PathLen, MEMF_CLEAR)) == NULL)
    {
    FreeMem(Entry, sizeof(DIR_ENTRY));
    return(ERROR_NO_MEM);
    }
  strcpy(Entry->PathName, PathNode);

/*************************************************************
 *
 * Se si utilizza la versione 1.3 (release 39) della ARP.library,
 * si possono sostituire le linee seguenti con la seguente
 * funzione ARP:
 *
 *   TackOn(Entry->PathName, Dir);
 *
 * Le linee da sostituire sono quelle comprese fra questo
 * commento e il prossimo.
 *
 *************************************************************/

  if (*Dir)
    {
    if (PathNode[strlen(PathNode) - 1] != ':')
      strcat(Entry->PathName, "/");
    strcat(Entry->PathName, Dir);
    }

/********************  Fine sostituzione  ********************/

  AddTail((struct List *) &DirList, (struct Node *) Entry);
  return(0);
}

VOID RemDirEntry()
{
  REGISTER DIR_ENTRY  *Entry;

  Entry = (DIR_ENTRY *) RemHead((struct List *) &DirList);
```

(continua a pagina 240)

inoltre, non dovremo includere nel nostro programma tutta la libreria (a meno che non ne utilizziamo tutte le funzioni, caso particolarmente raro).

La scelta dell'ARP.library è dovuta al fatto che tale libreria è già normalmente utilizzata da qualche programma commerciale (T×ED Plus, per esempio), oltre al fatto che la libreria è di pubblico dominio, compresi i file per i vari compilatori e la relativa documentazione (per coloro che fossero utenti di MC-Link è disponibile in area «Programmi» una serie di file ARCati contenenti tutto il necessario). Il compilatore che ho utilizzato è il Lattice C 5.02, ma ho cercato di scrivere il codice in maniera portatile e non utilizzando funzioni specifiche della versione 5.02, per cui i possessori di versioni precedenti del compilatore oppure in possesso dell'Aztec C non dovrebbero incontrare particolari problemi nella compilazione del programma.

Utilizzo di SF

La sintassi di SF è la seguente:

SF PAT <PathName Pattern> FILES/S DIRS/S QUICK/S NOROOT/S.

In pratica l'unico parametro necessario è il «pattern» (modello, sigh!) di ricerca, mentre gli altri parametri costituiscono degli «switch» (deviatori, argh!!) opzionali che influiscono sul funzionamento del programma. Esaminiamo ora i parametri uno per uno.

Il parametro «PAT <PathName Pattern>» è l'unico parametro obbligatorio: è attraverso esso, infatti, che comunichiamo al programma che cosa vogliamo cercare. La parola chiave «PAT» è opzionale e può non essere specificata. Dato che il «parsing» (analisi grammaticale e semantica) dei parametri viene effettuato da una funzione della ARP.library (GADS), oltre alla normale richiesta di help, tipica dell'AmigaDOS (SF ?) è possibile chiedere un help più esteso digitando un ulteriore «?» quando è visualizzato la riga di help standard del programma. Sempre grazie alle funzioni dell'ARP.library, è possibile specificare dei pattern di ricerca utilizzando anche la «wildcard» (carattere jolly) «*», tipica dei sistemi MS-DOS e UNIX. Ciò vuol dire che possiamo specificare dei pattern di ricerca complessi quale:

```
SF (*.c|#?.h|p*|b#?)
```

che, unendo le potenzialità delle wildcard riconosciute dall'AmigaDOS e dall'ARP.library, ci permetterà di cercare

nell'ambito del drive (o volume) correntemente selezionato tutti i file con estensione «.c» e «.h» nonché tutti i file che iniziano per «p» e «b». Infine, è possibile specificare un drive (o volume), nonché eventuali directory e sottodirectory da cui far partire la ricerca dei file. Se questi ultimi non vengono specificati, la ricerca inizia dalla directory «root» (radice) del drive (o volume) corrente, indipendentemente dalla directory su cui siamo posizionati. Se vogliamo iniziare la ricerca a partire dalla directory corrente, allora occorre specificare l'opzione «NOROOT». Per coloro che utilizzeranno la versione 1.3 della ARP.library sarà possibile specificare anche delle «classi di carattere», termine per il quale rimando alla documentazione ARP per una più ampia discussione. A titolo di esempio, il comando:

SF [b-dh-l]*

cercherà tutti i file che iniziano con le lettere comprese tra «b» e «d» e tra «h» e «l». Le opzioni «FILES» e «DIRS» informano il programma di effettuare la ricerca del pattern specificato rispettivamente solo per i file o per le directory. L'opzione «QUICK», infine, viene utilizzata quando si vuole solo la visualizzazione del nome del file (o directory). Infatti, SF normalmente visualizza, per ogni file trovato, il nome, la lunghezza (oppure «(dir)», nel caso di directory) e la data e l'ora dell'ultima modifica.

Se si vuole interrompere il programma, è possibile digitare, in qualsiasi momento, Control-C o Control-D.

Come funziona SF

Il funzionamento di SF si basa principalmente sulla funzione «PatternMatch()» della ARP.library. È grazie ad essa che possiamo utilizzare quei complessi pattern di ricerca appena visti. Tale funzione ritorna un valore booleano (che in C equivale a dire un valore pari a 0 per FALSE e diverso da 0 per TRUE) in base al confronto tra i suoi due operandi. Il primo dei due operandi è il pattern di confronto, mentre il secondo costituisce la stringa che vogliamo confrontare. Per quanto riguarda il pattern di confronto, c'è da dire che esso non può essere direttamente il pattern specificato sulla linea di comando, ma deve essere opportunamente processato dalla funzione ARP «PreParse()» che si incarica di tradurre in «token» (letteralmente gettone, ma va inteso come «codice chiave») la stringa contenente il pattern di ricerca. Ma procediamo con ordine.

(segue da pagina 239)

```
    FreeMem(Entry->PathName, Entry->PathLen);
    FreeMem((STRPTR) Entry, sizeof(DIR_ENTRY));
}

VOID FreeDirList()
{
  while (DirList.mlh_Head->mln_Succ)
    RemDirEntry();
}

VOID FindPat(ArgV)

STRPTR ArgV[];

#define     PATTERN       ArgV[0]
#define     FILES         ArgV[1]
#define     DIRS          ArgV[2]
#define     QUICK         ArgV[3]
#define     NOROOT        ArgV[4]

{
  REGISTER struct FileLock         *DirLock;
  REGISTER struct FileInfoBlock    *FileInfo;
  REGISTER STRPTR                   CurrentPath;
  REGISTER LONG                     Error;
  REGISTER BOOL                     Found = FALSE,
                                    First,
                                    Dir;
           TEXT                     Day       [12],
                                    Time      [10],
                                    Date      [10],
                                    StrToken [120],
                                    Path     [120];
           WORD                     Tabs = 0;
           struct DateTime          DateTime;

  if ((FileInfo = (struct FileInfoBlock *)
       AllocMem(sizeof(struct FileInfoBlock), MEMF_CLEAR)) != NULL)
    {
    DateTime.dat_Format = FORMAT_DOS;
    DateTime.dat_Flags = DTF_FUTURE;
    DateTime.dat_StrDay = Day;
    DateTime.dat_StrDate = Date;
    DateTime.dat_StrTime = Time;
    NewList((struct List *) &DirList);
    *Path = '\0';
    if (NOROOT == NULL)
      if (strchr(PATTERN, ':') == NULL)
        strcpy(Path, ":");
    strcat(Path, PATTERN);
    PreParse((CurrentPath = StrUpper(BaseName(Path))), StrToken);
    *CurrentPath = '\0';
    if ((DirLock = (struct FileLock *) Lock(Path, ACCESS_READ)) == NULL)
      Error = ERROR_INVALID_LOCK;
    else
      {
      PathName((BPTR) DirLock, Path, 10);
      UnLock((BPTR) DirLock);
      Error = AddDirEntry(Path, "");
      }
    while (DirList.mlh_Head->mln_Succ)
      {
      CurrentPath = ((DIR_ENTRY *) DirList.mlh_Head)->PathName;
      if ((DirLock = (struct FileLock *) Lock(CurrentPath, ACCESS_READ)) ==
                     (struct FileLock *) 0)
        {
        Error = ERROR_INVALID_LOCK;
        break;
        }
      First = TRUE;
      if (Examine((BPTR) DirLock, FileInfo))
        {
        while (ExNext((BPTR) DirLock, FileInfo))
          {
          if (Dir = (FileInfo->fib_DirEntryType > 0))
            if ((Error = AddDirEntry(CurrentPath,
                                     FileInfo->fib_FileName)) != NO_ERROR)
              break;
          if (PatternMatch(StrToken,
                StrUpper(strcpy(Path, FileInfo->fib_FileName))) == TRUE)
            if (!(DIRS ^ FILES) ||
                (DIRS && Dir) ||
                (FILES && !Dir))
              {
              Found = TRUE;
              if (First)
                {
                if (Tabs)
                  {
                  Puts("");
                  Tabs = 0;
                  }
                Printf("\n\033[3m\033[33mDirectory %s\033[0m\033[31m\n",
                       CurrentPath);
                First = FALSE;
                }
              if (QUICK)
                {
                Printf("%s%-18s", (Tabs) ? "" : "  ", FileInfo->fib_FileName);
                if (++Tabs == 4)
                  {
                  Tabs = 0;
                  Puts("");
                  }
                }
              else
```

```
                    }
                    DateTime.dat_Stamp.ds_Days = FileInfo->fib_Date.ds_Days;
                    DateTime.dat_Stamp.ds_Minute = FileInfo->fib_Date.ds_Minute;
                    DateTime.dat_Stamp.ds_Tick = FileInfo->fib_Date.ds_Tick;
                    StamptoStr(&DateTime);
                    Printf("    %-30s ", FileInfo->fib_FileName);
                    if (Dir)
                      Printf("  (dir)");
                    else
                      Printf("%7ld", FileInfo->fib_Size);
                    Printf(" %-9s %9s %8s\n", Day, Date, Time);
                    }
                  }
                if (CheckBreak(SIGBREAKF_CTRL_C | SIGBREAKF_CTRL_D, NULL))
                  {
                  Error = ERROR_BREAK;
                  break;
                  }
                }
              if (Error == NO_ERROR)
                Error = IoErr();
              }
            RemDirEntry();
            UnLock((BPTR) DirLock);
            if (Error != ERROR_NO_MORE_ENTRIES)
              break;
            }
          if (Tabs)
            Puts("");
          if (Error == ERROR_NO_MORE_ENTRIES && Found == TRUE)
            Error = NO_ERROR;
          FreeDirList();
          FreeMem((STRPTR) FileInfo, sizeof(struct FileInfoBlock));
          }
        else
          Error = ERROR_NO_MEM;
        switch(Error)
          {
          case NO_ERROR:
            break;
          case ERROR_NO_MEM:
            Puts("Non c'e' abbastanza memoria");
            break;
          case ERROR_INVALID_LOCK:
            Puts("Pattern (o nome di path) non valido");
            break;
          case ERROR_NO_MORE_ENTRIES:
            Puts("Ricerca fallita");
            break;
          case ERROR_OBJECT_NOT_FOUND:
            Puts("Oggetto non trovato");
            break;
          case ERROR_BREAK:
            Puts("*** BREAK");
            break;
          default:
            Printf("Errore %ld (errore interno)\n", Error);
          }
        }

        VOID MemCleanup()  /* evita la chiamata alla libreria della Lattice */
        {
        }

        VOID Break()
        {
          Puts("*** BREAK");
          ArpExit(RETURN_WARN, 0);
        }

        VOID _main(Line)
        REGISTER STRPTR Line;

        #define    MAX_ARG      5

        {
          REGISTER LONG  ArgC;
          STRPTR         Arg[MAX_ARG];

          _ONBREAK = Break;
          Puts("\n\033[32mSF-Search File v.1.0 - \2511989 by Andrea Suatoni\033[31m\n")
        ;

          for (ArgC = 0; ArgC < MAX_ARG; ++ArgC)
            Arg[ArgC] = (STRPTR) NULL;
          while (*Line > ' ')
            ++Line;
          ArgC = GADS(++Line,
                      strlen(Line),
                      "\nUso: SF PAT <PathName Pattern> FILES/S DIRS/S QUICK/S\
         NOROOT/S\n  SF senza parametri per maggiore aiuto\n",
                      Arg,
                      "PAT/A,FILES/S,DIRS/S,QUICK/S,NOROOT/S");
          if (ArgC < 0)
            {
            Printf("Parametri non corretti per SF: %s\n\n", Arg[0]);
            ArpExit(RETURN_FAIL, ERROR_LINE_TOO_LONG);
            }
          if (ArgC > 0)


            FindFile(Arg);
          else
            Usage();
          Puts("");
        }
```

main() o _main()?

Qualcuno avrà notato una piccola «stranezza» nel sorgente del programma: esiste una funzione «_main()» e non una funzione «main()». Non entrerò troppo nei particolari, poiché ci vorrebbero almeno un paio di pagine di MC per illustrare l'argomento, ma darò ugualmente una breve spiegazione. Quello di cui ci si lamenta spesso, quando si programma in C sull'Amiga, è che per una misera «printf()» vengano generati dei file eseguibili di lunghezza decisamente spropositata. Il motivo principale è che spesso vengono utilizzati i file di «startup» (inizializzazione) standard del C che, a loro volta, chiamano delle funzioni di libreria «linked» a nostra insaputa. In generale, la composizione finale di un programma C è così fatta:

c.o	(modulo di startup)
_main	(funzione chiamata da c.o e presente in libreria)
main	(funzione principale utente chiamata da _main)
....	(altre funzioni utente)
....	(funzioni di libreria aggiunte dal linker).

Brevemente, il modulo di startup inizializza le varie aree di memoria, apre le librerie «shared» Exec e Dos e quindi chiama la funzione _main(). Lo scopo di quest'ultima è quello di inizializzare il vettore argc[] in base ai parametri passati sulla linea di comando, aprire una finestra nel caso di chiamata del programma da Workbench, impostare i canali standard di input e output (stdin, stdout e stderr) nonché l'area di lavoro per la gestione dei file tramite le funzioni UNIX e di chiamare, finalmente, la funzione utente main(). Quello che appare chiaro è che se non si intende lanciare il programma da Workbench e non si intendono utilizzare le funzioni UNIX per i file, si può tranquillamente definire la funzione _main() al posto di main(), in quanto, in questo modo, il linker non estrarrà il relativo modulo dalla libreria. L'unico svantaggio che si ottiene è quello di dover «spezzettare» a mano la riga di comando che l'AmigaDOS ci passa ma, nel nostro caso, ciò non rappresenta un problema in quanto a questo provvede la funzione ARP «GADS()». Questa funzione, tra l'altro, è quella che esegue anche l'analisi dei parametri specificati in base al «template» (modello) passatogli come parametro. Ecco che allora il lavoro della nostra funzione _main() si riduce alla chiamata della funzione GADS() e, in base al

valore di ritorno di quest'ultima, alla chiamata o meno della funzione vera e propria di ricerca dei file (FindFile). È importante notare che, sempre nella funzione _main(), viene definita la funzione che deve essere chiamata quando vogliamo interrompere il programma tramite lo «statement» (istruzione):

_ONBREAK = Break;

dove «_ONBREAK» rappresenta il puntatore alla funzione di interruzione e «Break» l'indirizzo della funzione che vogliamo far eseguire. (Nota: quanto

```
typedef
  struct
    {
    struct MinNode  dir_Node;
    LONG            PathLen;
    STRPTR          PathName;
    }
  DIR_ENTRY;
```

Figura 1

detto è sicuramente vero per il Lattice C, mentre per l'Aztec C occorre trovare una modalità analoga).

Algoritmo di ricerca

Dovendo cercare un file disperso in una qualsiasi directory, non esiste un metodo migliore di un altro per scandire il disco, quindi ho scelto la soluzione che mi è sembrata più semplice. In pratica, attraverso le funzioni messe a disposizione da Exec, ho creato una lista gestita a FIFO (First In First Out). Ogni volta che viene incontrata una directory viene inserito il suo nome in coda alla lista tramite la funzione «Add DirEntry()», mentre ogni volta che la scansione di una directory è terminata, quest'ultima viene eliminata dalla lista tramite la funzione «RemDirEntry()» e, dall'inizio della stessa lista, viene letto il nome di una nuova directory da esaminare. Il tutto si ripete fino allo svuotamento della lista. La struttura che costituisce l'elemento base della lista è quello che si vede in figura 1. Si può notare, nella funzione AddDirEntry(), come venga sfruttata la funzione di Exec «AllocMem()» per allocare, oltre alla struttura DIR_ENTRY, anche lo spazio di memoria necessario per contenere il nome della directory. Se da un lato questo comporta un codice leggermente più lungo, dall'altro si ha l'innegabile vantaggio di non allocare spazio inutilmente, come sarebbe successo se si fosse utilizzato un array di caratteri. L'unica accortezza, ovviamente, è quella di deallocare tutte le aree di memoria eventualmente occupate e questo va fatto non soltanto quando il programma ha termine, ma anche quando l'utente ne richiede l'interruzione. A questo scopo provvede la funzione «FreeDirList()», che scandisce tutti gli (eventuali) elementi della lista deallocandoli uno per uno. Tale funzione viene chiamata all'uscita del loop di scansione del disco in FindFile(), uscita che può anche essere forzata dall'utente tramite la pressione dei tasti Control-C o Control-D: questo controllo viene effettuato dalla funzione ARP «CheckBreak()».

```
#
# Make file per SF (SearchFile)
# (Versione per Lattice C 5.02)
#
LIBS = LIB:Arp.lib LIB:Lc.lib LIB:Amiga.lib
OPTC = -cus -v -O
OPTL = SC SD ND

SF:    SF.o
       BLink FROM LIB:ArpC.o SF.o TO SF LIB $(LIBS) $(OPTL)

SF.o: SF.c
       Lc $(OPTC) SF
```

Figura 2

Le altre funzioni

Rimangono da fare alcune osservazioni di carattere generale. La funzione nulla «MemCleanUp()» serve ad evitare che il linker estragga dalla libreria del compilatore C la funzione omonima, il cui scopo è quello di liberare le aree di memoria allocate tramite le funzioni standard del C (come la malloc(), per esempio). Dato che nel programma queste ultime non vengono utilizzate, la funzione non è necessaria e quindi la sua dichiarazione impedisce al linker di estrarre il relativo modulo dalla libreria. Gli utenti di un compilatore diverso dal Lattice quasi sicuramente non avranno bisogno di dichiarare questa funzione, dato che il loro modulo di startup iniziale probabilmente non la richiama.

Si può inoltre notare che ho utilizzato le funzioni «Printf()» e «Puts()» al posto di «printf()» e «puts()». Il motivo è che tali funzioni sono già contenute nella ARP.library, per cui, visto che la libreria viene caricata in memoria in ogni caso, mi è sembrato logico preferirle alle loro «quasi omonime» funzioni presenti nella libreria standard del C, ottenendo, tra l'altro, un codice decisamente più compatto.

Il «bug» della ARP.library

A causa di un salto ad un indirizzo sbagliato, coloro che adotteranno una ARP.library versione 1.1 (release 34) non potranno fare uso della funzione ARP «TackOn()», il cui scopo è quello di aggiungere (tack on, per l'appunto) un nome di file a un nome di directory o volume interponendo o meno il carattere «/». Per questo motivo, nella funzione AddDirEntry(), sono state inserite delle linee di C che sostituiscono tale funzione (vedi listato). Chi invece utilizzerà la versione 1.3 (release 39) non avrà questo problema e potrà quindi utilizzare la funzione TackOn() eliminando le linee di C nominate in precedenza.

Compilazione di SF

La compilazione di SF non richiede particolari accorgimenti. Ovviamente occorre aver a disposizione la libreria di aggancio alla ARP.library (ARP.lib) nonché tutti i file di «include» necessari. Se volete generare del codice compatto, vi consiglio di disabilitare il controllo sullo stack (opzione -v della Lattice): infatti, come si può vedere dal listato del programma, lo stack viene utilizzato in minima parte, per cui la sua dimensione di default (4K) è più che sufficiente. Inoltre è opportuno utilizzare il modulo di startup «Arp.o» in luogo del consueto «c.o», in quanto questo modulo, oltre ad eseguire le stesse funzioni del modulo «c.o», verifica anche l'esistenza della ARP.library ed eventualmente ne esegue l'apertura.

Nel caso che qualcuno abbia personalizzato una particolare versione del modulo di startup, è possibile il suo utilizzo a patto che venga aperta, all'inizio del programma, la libreria ARP e venga assegnato l'indirizzo di ritorno della funzione OpenLibrary() alla variabile globale «ArpBase» (è importante che si chiami esattamente così). In quest'ultimo caso, al posto della libreria di aggancio ARP.lib deve essere utilizzata la libreria A.lib (per maggiori dettagli, vi consiglio di leggere il manuale per il programmatore della ARP.library). Ricordatevi, ovviamente, di chiudere la ARP.library all'uscita del programma. In figura 2 si può vedere il «makefile» che ho usato per la compilazione di SF. Ricordo che è necesario essere in possesso di una utility «make» (per esempio la LMK della Lattice) per poter utilizzare questo file. Infine, vorrei far notare che la lunghezza del codice eseguibile, per quanto possa sembrare incredibile, non dovrebbe risultare superiore ai 4 Kbyte!

MC

☞COMPUTER

AMIGA 500	720.000
AMIGA 2000	1.555.000
ATARI 520 STFM	670.000
ATARI 1040 STF	799.000
ATARI 1040 STFM	850.000
BONDWELL T8	2.300.000
BONDWELL T8H	2.900.000
PHILIPS NMSTC100+mon.	840.000
PHILIPS NMS 9110+mon.	1.170.000
PHILIPS NMS 9115+mon.	1.765.000
PHILIPS NMSAT25+mon.	2.599.000
EASY XT 256K 1 DRIVE	840.000
EASY XT PRO20 HD 20M	1.500.000
EASY AT PRO20 HD 20M	1.900.000
Z88 COMPLETO	700.000

☞MONITOR

PHILIPS 12" MON. CVBS	135.000
PHILIPS 12" MON. TTL	140.000
PHILIPS 14" MON. TTL	199.000
PHILIPS 8802 14" COLORE	360.000
PHILIPS 8833 14" COLORE	430.000
PHILIPS 9CM053 14" EGA	610.000
PHILIPS 9CM073 14" EGA	699.000
PHILIPS 9CM082 14" VGA	740.000
PHILIPS 9CM875 14" MULTI	999.000
COMMODORE 1084S	479.000
MITSUBISHI EUM1481A	999.000
VISA 14" DUAL L.V.	225.000
NEC II MULTISYNC	1.100.000

SOFTWARE ORIGINALE

GENIUS : Contabilita' / Fatturazione e Magazzino.

JSOFT : Lotus, Symphony, Word, Work, Windows, Excel, Quattro, Paradox, Concorde, Ventura, Mida, Turbo Pascal, Reflex, Sidekick,ecc.

CTO : Originali per AMIGA e MS/DOS

EASYDATA
Via A.Omodeo 29/21
00179 Roma

☎ 06/7858020

9.30/13.00 – 15.00/19.00
SABATO APERTO – LUNEDI' MATTINA CHIUSO

Spedizioni in tutta italia in contrassegno postale urgente.
I PREZZI SONO **IVA ESCLUSA**

ACCESSORI VARI

Dischi vergini, contenitori , Digitalizzatori Audio e Video, Genlock, Int. MIDI, Espansioni Memoria A500 e A2000, Drive Esterni, ecc.

Si realizzano video e diapositive con il sistema AMIGA

☞STAMPANTI

CITIZEN 120D	299.000
CITIZEN 180E	350.000
CITIZEN MPS15E	570.000
CITIZEN MPS40	599.000
CITIZEN MPS45	747.000
CITIZEN MPS50	830.000
CITIZEN MPS55	965.000
CITIZEN HQP40	915.000
CITIZEN HQP45	1.197.000
STAR LC10	370.000
STAR LC10 COLOR	460.000
EPSON LX800	460.000
EPSON LQ500	650.000
NEC2200	650.000
NEC P6 PLUS	1.300.000

☞SCHEDE

S.MADRE XT 10 MHZ	130.000
S.MADRE AT 16 MHZ	399.000
CGA	99.000
EGA	299.000
VGA	450.000
SERIALE	39.000
KIT 2a SERIALE	29.000
MILTI IO	99.000
PARALLELA	29.000
JOYSTIK	35.000
CONTROLLER HD XT	99.000
CONTROLLER HD AT	199.000
DRIVE 360K	110.000
DRIVE 1.2 MEGA	159.000
DRIVE 720/1.44	199.000