Programmare in C su Amiga (31)

di Dario de Judicibus (MC2120)

In questa puntata vedremo come vanno utilizzate le funzioni e le macro presentate nella scorsa puntata. Vedremo inoltre cosa sono i programmi puri ed il significato dei termini riusabile, rientrante e residente. Con la Scheda Tecnica di questo mese termina la carrellata sui comandi della versione 1.3 dell'AmigaDOS, che ci ha accompagnati per diversi mesi. Torna finalmente la rubrica dedicata alle lettere dei lettori, per troppo tempo sacrificata a vantaggio degli argomenti trattati nelle ultime puntate

Introduzione

Adesso che la nostra scatola degli attrezzi [*toolbox*] ha incominciato a riempirsi, vediamo come possiamo utilizzare i primi strumenti che abbiamo creato. Ricordo che si tratta di un gruppo di funzioni e macro C intese a creare e gestire pulsanti a rilascio automatico e manuale.

Lo schema del programma utilizzatore è più o meno quello del programma scheletro presentato quando abbiamo parlato di menu. Utilizziamo dunque quello scheletro per scrivere un programma con menu del tipo riportato in figura 1.

La struttura del programma sarà quindi formata da un certo numero di file. Quello principale, che chiameremo **gdgmain.c**, conterrà il codice utilizzatore vero e proprio, analogamente al vecchio **sklmain.c**; un secondo file, **gdgprocs.c**, conterrà le procedure relative ai pulsanti; un terzo file, **gdgusrh.c** conterrà le dichiarative necessarie ai primi due, e servirà a generare la oramai consueta tabella dei simboli (**gdgusrh.sym**). A questi file si aggiungeranno gli stessi file utilizzati con il programma scheletro **sklmain.c**, ad esclusione proprio di quest'ultimo, dato che abbiamo comunque una struttura a menu da gestire.

Vediamo allora quali sono i prototipi delle funzioni utilizzate, riportati in figura 2. Innanzi tutto abbiamo due nuovi eventi da gestire, oltre a quelli relativi ai menu: **GADGETUP** e **GADGETDOWN**. Ricordo che il primo viene emesso

Figura 1
Menu del programma utilizzatore.

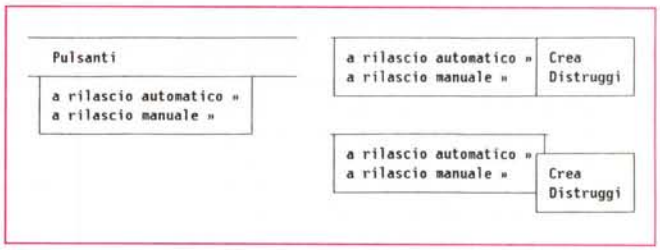


Figura 2 - Prototipi.

```
/*
**  Prototipi delle funzioni interne al programma
*/
 void StartAll     ( void );
 void CloseAll     ( int, ... );
 void BuildMenus   ( void );
 void LetsGo       ( void );
 void Buttons      ( BOOL );
 void Toggles      ( BOOL );
 void SwapTxts     ( ITXT *, ITXT *);
 int  HandleEvent  ( IMSG * );
/* ------------------------------------------------------------ STUBs --- */
 int  H_MenuVerify   ( IMSG * );
 int  H_MouseButtons ( IMSG * );
 int  H_RefreshWindow ( IMSG * );
/* ------------------------------------------------------------------------ */
 int  H_MenuPick      ( IMSG * );
 int  H_GadgetDown    ( IMSG * );
 int  H_GadgetUp      ( IMSG * );
 int  H_RefreshWindow ( IMSG * );
 int  H_CloseWindow   ( IMSG * );
```

quando viene selezionato un pulsante a rilascio automatico, dato che così abbiamo deciso nella scorsa puntata, mentre il secondo viene emesso alla selezione del pulsante a rilascio manuale, sempre per il modo in cui abbiamo definito questo pulsante nella **CreateButton()**. A tale scopo utilizzeremo quindi due nuove funzioni, **H_GadgetUp()** e **H_GadgetDown()**, appunto. Queste sono riportate rispettivamente in figura 3 ed in figura 4. Entrambe si limitano a ricavare dal messaggio spedito da Intuition l'identificativo del pulsante selezionato, e quindi a stampare un breve messaggio nella finestra CLI dalla quale si è fatto partire il programma. In aggiunta, la **H_GadgetDown()** inverte lo stato del pulsante a rilascio manuale, utilizzando una delle macro presentate nella scorsa puntata: la **ToggleButtonStatus()**.

Ma chi ha creato i pulsanti? Innanzi tutto diciamo subito che il programma qui presentato si limita a creare in modo dinamico un paio di pulsanti a rilascio automatico ed un paio a rilascio manuale sul fondo di una finestra. Non mostra quindi come utilizzare queste funzioni qualora i controlli vadano posizionati nel bordo di una finestra, e tantomeno se i pulsanti debbano far parte di un quadro, dei quali peraltro non abbiamo ancora parlato in dettaglio. A voi quindi il compito di verificare se le funzioni presentate siano adatte anche a tale scopo e se e quali verifiche vanno aggiunte al codice per evitare problemi nell'utilizzare tali funzioni in contesti differenti da quello presentato da questo mese. A tale riguardo, sono quasi sicuro che, così come sono, sarà necessario apportare un certo numero di modifiche per garantirne il corretto funzionamento in tutte le possibili situazioni. D'altro canto è giusto che facciate qualcosa anche voi, no?

Tornando al nostro programma, vediamo innanzi tutto le dichiarative relative ai vari pulsanti, riportate in figura 5. Vi risparmio quelle per i menu e le voci, che ricalcano quelle presentate nel vecchio programma scheletro. Per prima cosa abbiamo bisogno di due nuove maschere per tener traccia della creazione dei due pulsanti a rilascio automatico e dei due a rilascio manuale, dato che usiamo come di consueto la tecnica della maschera per la chiusura intelligente delle risorse acquisite. Quindi definiamo un vettore che può contenere fino a ventuno puntatori a strutture **Gadget**. Per ogni pulsante, automatico o manuale che sia, definiamo quattro costanti: l'identificativo del controllo stesso, il testo del pulsante, e la sua posizione nella finestra. Quest'ultima segue le regole riportate nella scorsa puntata, che differiscono da quelle classiche di Intuition in quanto il punto del pulsante da posizionare è differente a seconda del segno delle coordinate stesse, scelta a mio avviso più funzionale e più intuitiva (fate riferimento alla 30ª puntata per la descrizione in dettaglio di questa tecnica).

La funzione **HandleEvent()** è simile a quella del programma scheletro. Ci sono solo due istruzioni in più, intese a intercettare i due nuovi eventi relativi alle due tipologie di pulsanti in questione (vedi figura 6).

Due nuove funzioni sono invece necessarie per creare e distruggere i pulsanti. Esse hanno più o meno la stessa struttura, sebbene facciano riferimento ai due differenti gruppi di pulsanti. Si tratta della **Buttons()** e della **Toggles()**, la prima (in figura 7) per i pulsanti a rilascio automatico, la seconda (in figura 8) per quelli a rilascio manuale. Per quello che riguarda la logica di queste funzioni, fate riferimento allo schema riportato in figura 9, e che utilizza uno spe-

```
/*****************************************************************************
** H_GadgetUp: gestisce l'evento GADGETUP                                   **
*****************************************************************************/
int H_GadgetUp(msg)
  IMSG *msg;
{
  IGDG *gdg;
  USHORT gid;

  /*
  **  E' stata fatta effettivamente una selezione? Se si, allora
  **  determiniamo a quale controllo si riferisce.
  */
  gdg = (IGDG *)msg->IAddress;       /* Puntatore al controllo          */
  gid = gdg->GadgetID;               /* Identificativo selezione        */

  /*
  **  Alcune definizioni per la stampa
  */
  #define PRT_BUT(g) printf("Controllo [%s]\n",(g))

  /*
  **  BLOCCO PER LA GESTIONE DEI CODICI
  */
  switch (gid)
    {
      case ABUTTON: PRT_BUT((gdg->GadgetText)->IText); break;
      case BBUTTON: PRT_BUT((gdg->GadgetText)->IText); break;
      defaults    : break;
    }

  return(GOAHEAD);
}
```

Figura 3 - GadgetUp().

```
/*****************************************************************************
** H_GadgetDown: gestisce l'evento GADGETDOWN                               **
*****************************************************************************/
int H_GadgetDown(msg)
  IMSG *msg;
{
  IGDG *gdg;
  USHORT gid;

  /*
  **  E' stata fatta effettivamente una selezione? Se si, allora
  **  determiniamo a quale controllo si riferisce.
  */
  gdg = (IGDG *)msg->IAddress;       /* Puntatore al controllo          */
  gid = gdg->GadgetID;               /* Identificativo selezione        */

  /*
  **  Alcune definizioni per la stampa
  */
  #define PRT_TGL(g,st) printf("Controllo [%s] : %s\n",(g),(st)?"ON":"OFF")

  /*
  **  BLOCCO PER LA GESTIONE DEI CODICI
  */
  switch (gid)
    {
      case ATOGGLE:
           PRT_TGL((gdg->GadgetText)->IText,ToggleButtonStatus(gdg)) ;
           break;
      case BTOGGLE:
           PRT_TGL((gdg->GadgetText)->IText,ToggleButtonStatus(gdg)) ;
           break;
      defaults:
           break;
    }

  return(GOAHEAD);
}
```

Figura 4 - GadgetDown(). ▶

```
/*
**  Maschere di controllo
*/

  :

 #define MSK_MST 0x0010
 #define MSK_BUT 0x0020
 #define MSK_TGL 0x0040
 UWORD  mask = 0x0000;

  :

/*
**  Pulsanti
*/
 #define NUMBUTT  21
 IGDG *buttons[NUMBUTT];         /* Vettore dei puntatori ai pulsanti     */

/*
**  Definizioni per i pulsanti a rilascio automatico
*/
 #define ABUTTON  0
 #define ABUTEXT  "Premimi"
 #define ABUTCOL  10
 #define ABUTROW  10
 #define BBUTTON  1
 #define BBUTEXT  "Pure a me!"
 #define BBUTCOL  -10
 #define BBUTROW  -10

/*
**  Definizioni per i pulsanti a rilascio manuale
*/
 #define ATOGGLE  2
 #define ATGTEXT  "Premimi"
 #define ATGLCOL  10
 #define ATGLROW  30
 #define BTOGGLE  3
 #define BTGTEXT  "Pure a me!"
 #define BTGLCOL  -10
 #define BTGLROW  -30
```

Figura 5 - Dichiarative relative ai pulsanti.

```
/*************************************************************************
** HandleEvent: gestione dei messaggi da Intuition                      **
*************************************************************************/
 int HandleEvent(msg)
   IMSG *msg;
 {
   IMSG localmsg;      /* Questa è una fotocopia del messaggio ricevuto */
   int  result;        /* Questo è il valore da restituire al chiamante */
                       /* Fotocopiamo il messaggio originale            */
   CopyMem((char *)msg,(char *)&localmsg,sizeof(IMSG));

  /* ------------------------------------------------------------------ *
   * ATTENZIONE: i controlli di tipo VERIFY vanno messi PRIMA di        *
   *             rispondere al messaggio, altrimenti non servono a      *
   *             niente! Potevamo anche usare msg->Class, ovviamente.   *
   * ------------------------------------------------------------------ */
   switch (localmsg.Class)
   {
     case MENUVERIFY  : result = H_MenuVerify   (&localmsg);   break;
   }

   ReplyMsg((struct Message *)msg); /* OK. Adesso possiamo rispondere. */

  /* ------------------------------------------------------------------ *
   * 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".         *
   *             MENUVERIFY è ripetuto per evitare che l'assegnazione   *
   *             di "result" in "default:" si sovrapponga a quella      *
   *             precedente. Questo è uno dei tanto modi per evitarlo.  *
   * ------------------------------------------------------------------ */

   switch (localmsg.Class)
   {
     case CLOSEWINDOW  : result = H_CloseWindow   (&localmsg);   break;
     case MENUPICK     : result = H_MenuPick      (&localmsg);   break;
     case MOUSEBUTTONS : result = H_MouseButtons  (&localmsg);   break;
     case REFRESHWINDOW: result = H_RefreshWindow (&localmsg);   break;
     case GADGETDOWN   : result = H_GadgetDown    (&localmsg);   break;
     case GADGETUP     : result = H_GadgetUp      (&localmsg);   break;
     case MENUVERIFY   : /* # già trattato in precedenza # */    break;
     default           : result = GOAHEAD                   ;   break;
   };
   return (result);
 }
```

Figura 6 - HandleEvent().

ciale formalismo detto *pseudocodifica*, di cui parlerò in una delle prossime schede tecniche. Per rendere lo pseudocodice più chiaro a chi non l'abbia mai visto, ho deciso di evidenziare i vari blocchi con una tecnica grafica di incorniciamento, raramente utilizzata quando si scrive pseudocodice. A questo punto non resta che aggiungere un'ulteriore considerazione, e cioè che l'aggiornamento della *maschera traccia* va fatta dopo aver creato il primo pulsante, non entrambi. Se non si facesse così, nel momento in cui il programma non fosse riuscito a creare il secondo pulsante dopo avere creato con successo il primo, la maschera non verrebbe aggiornata, ed il programma non avrebbe modo di deallocare la memoria relativa a quest'ultimo. In questo modo, invece, qualora un qualunque pulsante dopo il primo non potesse essere allocato, dato che la maschera è stata comunque attivata, il programma saprà che deve in ogni caso distruggere un certo numero di pulsanti prima di terminare. Sarà allora un ulteriore controllo sul valore del puntatore ai vari pulsanti, nullo per quelli non creati, a discriminare quelli da distruggere.
Ma chi chiama queste due funzioni? Semplice. Dato che i pulsanti sono creati e distrutti selezionando opportune voci del menu **Pulsanti**, il codice che chiama la **Buttons()** e la **Toggles()** si trova nella procedura di gestione dei menu, e cioè la **H_MenuPick()**, come riportato in figura 10. E questo è tutto...

CreateButton()

Vi ricordate che vi dissi, nella scorsa puntata, che c'era un errore nella **CreateButton()**? Lo avete trovato? Beh, per quelli che stanno ancora arrovellandosi il cervello per capire qual è, se volete divertirvi ancora per un po', saltate le prossime righe. Se invece vi siete miseramente arresi, o siete curiosi di sapere se avete vinto il primo premio, ecco la soluzione:

la funzione in questione considera nel calcolo della posizione del pulsante nel contenitore i bordi della finestra, anche se questo va collocato in un quadro!

Avete indovinato? Sì? Bravi! Ecco il premio: dovete modificare la **CreateButton()** in modo che tenga conto anche di questa possibilità. Come sarebbe a dire «che razza di premio è?». Cosa potrebbero desiderare di più degli appassionati di programmazione su Amiga quali sicuramente siete voi? Ingrati!

Programmi puri

Finalmente, dopo tanto tempo, sono riuscito a trovare un po' di spazio per parlarvi di comandi puri e comandi residenti. Si tratta di un concetto molto importante, non solo per chi programma, ma anche per l'utente finale. Questi infatti ne trae beneficio sia in termini di utilizzo delle risorse del sistema, sia in quelli di maggiore sfruttamento dei vantaggi del *multitasking*. Se un coman-

do è utilizzato molto spesso, il fatto che debba venire caricato in memoria ogni volta che viene invocato, rappresenta un fattore di riduzione delle prestazioni del sistema [*performance*].

AmigaDOS dà la possibilità di mantenere tali comandi in memoria, anche quando non sono utilizzati, in modo da evitare i tempi di caricamento da disco. Inoltre, in tal modo, viene mantenuta in memoria una sola copia di tali comandi, detti appunto «residenti» [*resident*], anche se essi vengono invocati da più sessioni contemporaneamente (cioè, in *multitasking*).

Questo tuttavia è possibile solo se il comando è invocato da SHELL, in quanto è necessario che il sistema possa mantenere traccia dei comandi residenti, in modo da evitare di ricaricare un comando che è già in memoria. La SHELL infatti mantiene una lista di tutti i comandi residenti. Quando l'utente chiede al sistema di eseguire un comando, la SHELL verifica prima se non è già tra quelli caricati, e quindi, se è questo il caso, lo manda in esecuzione senza ricaricarlo da disco.

È importante notare che questa tecnica è differente da quella che utilizza parte della memoria per creare un disco virtuale di tipo ricoverabile o meno. Un programma memorizzato su RAM:, ad esempio, è sì già in memoria, ma solo come file. La memoria è cioè utilizzata in questo caso come archivio, non per eseguire il codice. Quando tale programma viene lanciato, esso viene infatti caricato *una seconda volta* in memoria, dove questa volta viene effettivamente eseguito, per poi essere cancellato a fine esecuzione.

Non tutti i comandi (o programmi) possono essere caricati come residenti, tuttavia. Come certamente saprete, un programma è fatto di istruzioni e di dati. Mentre le istruzioni *non dovrebbero* cambiare durante la fase di esecuzione [*run*], a meno che non si tratti appunto di programmi automodificanti, i dati, od almeno una parte di essi, vengono ad assumere valori differenti man mano che il sistema esegue le varie istruzioni. Ad esempio, una variabile indice in un ciclo [*loop*] assume valori crescenti o decrescenti ad ogni riesecuzione del blocco di istruzioni che lo compone. Consideriamo adesso il modo con il quale il sistema gestisce i programmi di tipo residente:

- una sola copia del codice è mantenuta in memoria per ogni programma residente;
- uno stesso programma può andare in esecuzione contemporaneamente a fronte di più lavori [*task*];
- le istruzioni che il sistema esegue dello stesso programma per conto dei vari lavori che ne hanno richiesto l'esecuzione, non sono necessariamente le stesse nello stesso momento (*sfasamento*);
- i dati su cui il programma deve operare per conto dei vari lavori, possono essere differenti per ogni lavoro (*parametri di ingresso, aree dati, file*).

Questo pone delle serie limitazioni alla struttura del programma candidato alla posizione di *residente*, e cioè, esso deve essere scritto in modo da essere:

- riusabile [*re-executable*];
- rientrante [*reentrant*].

Vediamo cosa significa.

Pensiamo ad un programma come ad una serie di istruzioni su un foglio di carta. Il sistema (cioè noi) legge una istruzione alla volta e la esegue. Che succede se una di queste operazioni consiste nel modificare una o più istruzioni tra quelle già eseguite? Semplice. La volta successiva che la lista viene letta a partire dalla prima istruzione, essa sarà differente, e quindi porterà ad un risultato differente. Il programma in questione si dice *non riusabile*. Ora, se il programma viene caricato ogni volta da disco, dato che le modifiche sono avvenute in memoria, la cosa può non dare alcuna conseguenza visibile, ma se ad essere rieseguita è la copia in memoria, come nel caso appunto dei programmi residenti, i risultati possono essere del tutto imprevedibili.

Supponiamo ora che il programma sia riusabile. Supponiamo ora che in fondo alla lista, dopo cioè le istruzioni [*code hunk*], ci sia una zona che contiene alcuni dati per eseguire le operazioni richieste [*data hunk*], ed una che serve per eseguire conti, scrivere appunti e via dicendo [*bss hunk*], all'inizio bianca. Fintanto che ad eseguire il programma è solo Carlo (cioè un singolo lavoro), non ci sono grossi problemi. Supponiamo ora però che la lista non sia su di un foglio, ma su una lavagna, e che un paio di minuti dopo che Carlo ha incomincia-

Figura 7 - Buttons().

```
/*****************************************************************************
** Buttons: crea o cancella i pulsanti a rilascio automatico                **
*****************************************************************************/
void Buttons (onoff)
  BOOL onoff;
{
  if (onoff)
  {
    if (mask & MSK_BUT)
      ShowMsgReq(w,"Pulsanti già visibili");
    else
    {
     /*
     **  Crea i pulsanti a rilascio automatico
     */
      buttons[ABUTTON] =
                    CreateAutoButton(ABUTTON,ABUTEXT,&c,ABUTCOL,ABUTROW);
      if (buttons[ABUTTON] == NULL) CloseAll(EMSG_NOMEMORY) ;

     /*
     **  Attenzione: basta un solo pulsante per attivare la maschera
     */
      mask |= MSK_BUT;

      buttons[BBUTTON] =
                    CreateAutoButton(BBUTTON,BBUTEXT,&c,BBUTCOL,BBUTROW);
      if (buttons[BBUTTON] == NULL) CloseAll(EMSG_NOMEMORY) ;

     /*
     **  Visualizza i pulsanti a rilascio automatico
     */
      DisplayGadget(buttons[ABUTTON],&c);
      DisplayGadget(buttons[BBUTTON],&c);
    }
  }
  else
  {
    if (!(mask & MSK_BUT))
      ShowMsgReq(w,"Pulsanti già cancellati");
    else
    {
      if (buttons[ABUTTON]) DeleteAutoButton(buttons[ABUTTON],&c);
      if (buttons[BBUTTON]) DeleteAutoButton(buttons[BBUTTON],&c);

      mask &= ~MSK_BUT;

    }
  }
}
```

to a lavorare sul programma, entra Francesca nella stanza e si mette anche lei ad eseguire le varie istruzioni, partendo ovviamente dalla prima. Abbiamo così due lavori che eseguono lo stesso programma sfasati l'un l'altro. Il problema è che entrambi i lavori utilizzano la stessa lavagna anche per i calcoli e gli appunti. Quando Francesca inizia a lavorare, la parte della lavagna che contiene i dati è già stata modificata da Carlo. Inoltre man mano che Francesca utilizza lo spazio di lavoro per gli appunti ed i calcoli intermedi, essa va ad alterare i dati che Carlo si era temporaneamente salvato per riutilizzare più tardi. Risultato. Entrambi i lavori finiscono per trovare risultati inattendibili, col rischio di mandare a carte quarantotto tutto il sistema (leggi GURU). E allora? Anche qui la soluzione è semplice, in apparenza. Basta semplicemente che sia Carlo che Francesca si copino i dati sulla lavagna su un proprio quaderno, ed utilizzino lo stesso anche per gli appunti ed i calcoli intermedi. Un programma rientrante è ap-

La scheda tecnica

Con questa puntata termina la nostra carrellata sui comandi della versione 1.3 che hanno subito modifiche rispetto alla versione precedente del sistema operativo, o che non esisteva nella versione 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:	STATUS
Formato:	STATUS <processo> [FULL] [TCB] [CLI|ALL] [COMMAND]
Sintassi:	STATUS "PROCESS,FULL/S,TCB/S,CLI=ALL/S,COMMAND/K"
Scopo:	Visualizza informazioni sui processi (CLI e SHELL)
Specifiche:	In aggiunta alle caratteristiche della versione 1.2, STATUS ha ora una nuova opzione (COMMAND) e supporta anche processi con priorità negative. L'opzione COMMAND permette di ricercare il processo fatto partire con uno specifico comando. Se trovato, il numero del processo è visualizzato, e STATUS termina con il codice di ritorno 0, altrimenti il codice di ritorno è 5 (WARN)
Esempio:	STATUS >ram:status.rc COMMAND=Monitor IF NOT WARN ; Trovato il processo partito con Monitor CHANGETASKPRI 5 ? ; Cambia la priorità a 5 (modo interattivo) TYPE ram:status.rc ; Fornisci il numero del processo ENDIF

Comando:	TYPE
Formato:	TYPE <origine> [TO <bersaglio>] [OPT H=HEX|OPT N=NUMBER]
Sintassi:	TYPE "FROM/A,TO/S,OPT/K,HEX/S,NUMBER/S"
Scopo:	Visualizza un file
Specifiche:	Le due opzioni OPT H e OPT N possono ora essere sostituite dalle opzioni HEX e NUMBER rispettivamente. Inoltre, se si usa "TO" ed il file bersaglio già esiste, questo non viene ricoperto dal nuovo file. TYPE si comporta in tal modo come una sorta di comando COPY di tipo "sicuro", che non ricopre vecchi file.
Esempio:	TYPE c:dir TO ram:dir.dump HEX ; Salva il contenuto hex di DIR

Comando:	VERSION
Formato:	VERSION <libreria|device> [<versione>] [<revisione>] [<unità>]
Sintassi:	VERSION "NAME,VERSION,REVISION,UNIT"
Scopo:	Visualizza la versione di librerie, device, o dischi WorkBench
Specifiche:	VERSION permette di visualizzare o verificare il numero di versione e/o revisione di una libreria, di una device, o di un disco di sistema (cioè "bootable" di tipo WorkBench). Se non è fornito alcun parametro, vengono visualizzati i numeri di versione e revisione del KickStart e del WorkBench. Se viene fornito il nome di una libreria o di una device, visualizza la versione e la revione di questa. VERSION permette inoltre di verificare se uno dei suddetti oggetti ha un numero di versione e/o revisione maggiore od uguale di quello specificato in tal caso come parametro del comando. A seconda che il risultato sia positivo o negativo, viene impostato il codice di ritorno a 0 od a 5 (WARN). Non è possibile specificare il nome di una device in un indirizzario diverso da DEVS:. Il parametro "unità" serve nel caso il sistema abbia più di una unità dischetto.
Esempio:	; Cerca su DF1: una libreria Intuition con versione >= 34.0
Esempio:	VERSION intuition.library 34 0 1 IF WARN ; Non c'è? RUN >NIL: OldStuff ; Lancia il programma vecchio ELSE ; Trovata? Sì? RUN >NIL: NewStuff ; Lancia il programma nuovo ENDIF

Comando:	WAIT
Formato:	WAIT <n> [SEC|SECS] [MIN|MINS] [UNTIL <ora>]
Sintassi:	WAIT "N,SEC=SECS/S,MIN=MINS/S,UNTIL/K"
Scopo:	Aspetta per un tempo specificato.
Specifiche:	Rispetto la versione precedente, è possibile ora mettere uno zero in testa al valore relativo al tempo di sospensione.
Esempio:	WAIT 5 ; Il default è in secondi

Comando:	WHICH
Formato:	WHICH <comando> [NORES] [RES]
Sintassi:	WHICH "FILE/A,NORES/S,RES/S"
Scopo:	Cerca il cammino relativo al comando specificato
Specifiche:	WHICH permette di trovare dove un certo comando si trova. In realtà esso NON cerca dapertutto, ma solo - nella lista dei comandi residenti - nell'indirizzario corrente - nel cammino di ricerca dei comandi - nell'indirizzario C: Se un comando non si trova in un indirizzario tra quelli della lista suddetta, anche se esso esiste da qualche parte in un dischetto montato od in una partizione del disco fisso, WHICH imposta il codice di ritorno a 5 (WARN). NORES serve a saltare nella ricerca la lista dei comandi residenti. RES, viceversa, effettua la ricerca SOLO fra i comandi residenti.
Esempio:	WHICH ls RES ; Il comando "ls" è stato caricato come residente?

```
/*************************************************************************
** Toggles: crea o cancella i pulsanti a rilascio manuale               **
*************************************************************************/
void Toggles (onoff)
  BOOL onoff;
{
  if (onoff)
  {
    if (mask & MSK_TGL)
      ShowMsgReq(w,"Pulsanti già visibili");
    else
    {
     /*
     **  Crea i pulsanti a rilascio manuale, uno già selezionato,
     **  l'altro no.
     */
      buttons[ATOGGLE] =
         CreateToggleButton(ATOGGLE,ATGTEXT,&c,ATGLCOL,ATGLROW,TOGGLEON);
      if (buttons[ATOGGLE] == NULL) CloseAll(EMSG_NOMEMORY) ;

     /*
     **  Attenzione: basta un solo pulsante per attivare la maschera
     */
      mask |= MSK_TGL;

      buttons[BTOGGLE] =
         CreateToggleButton(BTOGGLE,BTGTEXT,&c,BTGLCOL,BTGLROW,TOGGLEOFF);
      if (buttons[BTOGGLE] == NULL) CloseAll(EMSG_NOMEMORY) ;

     /*
     **  Visualizza i pulsanti a rilascio automatico
     */
      DisplayGadget(buttons[ATOGGLE],&c);
      DisplayGadget(buttons[BTOGGLE],&c);
    }
  }
  else
  {
    if (!(mask & MSK_TGL))
      ShowMsgReq(w,"Pulsanti già cancellati");
    else
    {
      if (buttons[ATOGGLE]) DeleteToggleButton(buttons[ATOGGLE],&c);
      if (buttons[BTOGGLE]) DeleteToggleButton(buttons[BTOGGLE],&c);

      mask &= ~MSK_TGL;
    }
  }
}
```

Figura 8 - Toggles().

punto un programma che, alla partenza, copia l'area dati su un pezzo di memoria a parte e si alloca un altro pezzo per gli «appunti». Tutto ciò *ogni volta* che viene invocato, per cui alla fine avremo in memoria una sola copia del codice riusabile, e tante copie delle aree dati inizializzate [*data*] e non [*bss*], quanti sono i lavori che lo stanno eseguendo allo stesso tempo.

A dirsi sembra facile, ma a farsi? Beh, per quanto riguarda la possibilità di rieseguire il programma in memoria, se usate un qualunque linguaggio di alto livello e non vi mettete a giocare con gli indirizzi assoluti di memoria, il risultato sarà sempre un programma riusabile. Se invece usate l'*Assembler* dovete fare un pochino più di attenzione a quello che scrivete, dato che una istruzione può operare su un'altra istruzione come se si trattasse di dati, alterandola o cancellandola del tutto. Il discorso si fa un po' più complicato quando parliamo di rientranza. In un qualunque programma ci capita spesso di modificare direttamente od indirettamente il valore delle variabili. Per evitare problemi dovremmo allocare tutte le variabili dinamicamente in un'area di memoria a parte.

```
SE si richiede di creare i pulsanti
  ALLORA
  SE questi sono già stati creati
    ALLORA
      visualizza un messaggio di errore
    ALTRIMENTI
      crea il primo pulsante
      SE fallito
        ALLORA
          esci in modo pulito
      mantieni traccia della creazione
      crea il secondo pulsante
      SE fallito
        ALLORA
          esci in modo pulito
      visualizza i pulsanti così creati
  ALTRIMENTI
  SE questi sono già stati distrutti
    ALLORA
      visualizza un messaggio di errore
    ALTRIMENTI
      SE c'è il primo pulsante
        ALLORA
          distruggilo
      SE c'è il secondo pulsante
        ALLORA
          distruggilo
      cancella la traccia della creazione
```

Figura 9 Pseudocodifica. ▼

Un giochino che non è alla portata di tutti. Per fortuna il *Lattice C 5.x* ci mette a disposizione una serie di *profili di partenza* [*startup file*] da sostituire a quello solitamente usato, e cioè **c.o**, che gestiscono automaticamente il problema. Si tratta di **cres.o**, **catchres.o** e **catchresnr.o**. Senza entrare nel dettaglio, che comunque sarebbe utile solo ai programmatori che hanno questa marca di compilatore, mentre il discorso può essere applicato oramai alla maggior parte dei compilatori C per Amiga (quantomeno le ultime versioni), basti sapere che, se avete scritto un programma riusabile e vi siete assicurati di aver seguito un paio di regolette che riporto più avanti, sostituendo **c.o** con **cres.o** nella fase di assemblaggio [*link*], il vostro programma risulterà automaticamente residente, con addirittura già il bit **pure** impostato. Gli accorgimenti da tener presente sono i seguenti:

- non usare mai l'opzione **-bθ** quando compilate il programma;
- tutti i blocchi di data [*data hunk*] devono poter essere ricomponibili in un unico blocco [*merged block*];
- non devono esistere posizioni *assolute* per i dati del programma;
- è necessario usare l'istruzione del preprocessore **#pragma** invece di assemblare il tutto alla libreria **amiga.lib**.

La seconda raccomandazione vale soprattutto per i programmatori in *Assembler* o che usano frammistare *Assembler* e *C*. L'ultima non è strettamente necessaria a condizione che **blink** non generi alcun avvertimento sull'esistenza di riferimenti assoluti nell'eseguibile.

L'opzione **pure** del comando **resident**, permette di caricare anche comandi solo riusabili ma non rientranti. Attenzione però. Se avete scritto un programma puro o riusabile, e volete verificarne il funzionamento come residente, assicuratevi sempre:

- di aver fatto partire il sistema da disco WorkBench standard con la protezione da scrittura attivata;
- di non operare le verifiche mentre si è in un indirizzario del disco rigido;
- di non assegnare *alcun* indirizzario del disco rigido;

per evitare di danneggiare la struttura dei file su disco. Operate sempre da RAM: o RAD:. Installate il comando come residente da una SHELL, posizionatevi su un indirizzario qualunque del disco virtuale, e da lì eseguite prima i controlli di riusabilità, e poi, se pensate che il vostro programma sia anche rientrante, aprite un'altra SHELL e lanciate il vostro comando da entrambe. La riusabilità si verifica semplicemente lanciando il programma più volte di seguito dalla stessa finestra in modo sincrono, cioè

```
/*************************************************************************
** H_MenuPick: gestisce l'evento MENUPICK                               **
*************************************************************************/
int H_MenuPick(msg)
  IMSG *msg;
{

  :

   /*
   **  BLOCCO PER LA GESTIONE DEI CODICI
   */
    switch (menunum)
    {
      case M_GADGET:         /* Menù PULSANTI                          */
           switch (itemnum)
           {
             case I_GDGBUT:  /* Voce A RILASCIO AUTOMATICO             */
                  switch (subnum)
                  {
                    case  S_BUTON : Buttons(TRUE);  break;  /* CREA      */
                    case  S_BUTOFF: Buttons(FALSE); break;  /* DISTRUGGI */
                  }
                  break;
             case I_GDGTGL:  /* Voce A RILASCIO MANUALE                */
                  switch (subnum)
                  {
                    case  S_TGLON : Toggles(TRUE);  break;  /* CREA      */
                    case  S_TGLOFF: Toggles(FALSE); break;  /* DISTRUGGI */
                  }
                  break;
           }
    }

  :

  return(GOAHEAD);
}
```

Figura 10
H_MenuPick().

aspettando ogni volta che l'esecuzione precedente sia terminata prima di lanciarlo di nuovo (senza utilizzare **run**, quindi). Tra una esecuzione e la seguente, verificate lo stato del sistema, le variabili globali di ambiente [*ENV variable*] usate eventualmente dal programma, e qualunque altra cosa il programma potrebbe «ricordare». Se il programma ha dimenticato quanto successo nelle sue «vite precedenti», allora molto probabilmente è riusabile. Nei controlli di rientranza, invece, fate molte prove incrociate, per verificare se l'esecuzione in un processo stia o meno provocando effetti collaterali in quella nel processo parallelo. In particolare, fate attenzione nel caso il programma operi su file, dato che una non perfetta rientranza può danneggiare seriamente i dati su disco acceduti in scrittura dal comando in questione.

Conclusioni

E con questo terminiamo anche questa puntata. Se continua così, a furia di provare programmi e programmini per questa rubrica, finirà che non avrò più tempo per fare qualsiasi altra cosa con il mio Amiga.

Tuttavia sono molto contento delle vostre relazioni a questa rubrica (se non altro non violente!), e delle molte lettere che ho ricevuto, tutte estremamente positive. Spero sinceramente che in breve tempo diveniate tutti molto più bravi di me nel programmare con l'Amiga (cosa peraltro non particolarmente difficile), e che possiate sempre ricavare la massima soddisfazione dai vostri programmi. AG

Casella Postale

Schede di espansione per il 1000 e Sidecar

Ho deciso... di tenermi il mio buon 1000 con relativo SIDECAR ed accontentarmi di espandere la memoria di quest'ultimo... Non so quante telefonate ho fatto per cercare di avere notizie su una espansione che non desse noia al Sidecar... MC di aprile di quest'anno... menzionava la famigerata scheda INSIDER della Michigan Software... vorrei soltanto un indirizzo, un numero telefonico per poter contattare la Michigan Software ed avere informazioni per poter acquistare la loro scheda Insider...

Gianluca Agus - 50131 Firenze

Innanzitutto la scheda Insider serve ad espandere la memoria dell'Amiga 1000, non del Sidecar. La prima affermazione quindi non è del tutto corretta. Assumendo quindi che ciò che a Lei interessa è proprio l'espansione del 1000, senza peraltro interferire con il Sidecar, penso le sarà utile sapere quanto segue.

Ho acquistato l'Insider più di tre anni fa, oramai, portando così la memoria del mio A1000 a 1.5Mb. Fino ad oggi non ho avuto alcun problema con essa, né per quello che riguarda il Sidecar, né per quello che riguarda l'orologio installato sull'Insider. La scheda è arrivata completa di garanzia ed una ventina di pagine di manuale. L'installazione è molto semplice, in teoria. Basta togliere il 68000 dalla sua base. Per far ciò è bene usare le dita od una apposita pinzetta per l'estrazione dei chip. Non faccia leva tra il processore e la base, perché rischia di danneggiare le tracce sulla scheda. A questo punto si innesta la scheda sulla base vuota, si raddrizzano i piedini del processore se si sono allargati nell'estrarre il chip, facendo delicatamente pressione sul bordo del tavolo, e si reinstalla il 68000 nella base apposita sulla scheda dell'Insider. L'Insider è inoltre provvisto di due cavetti

dotati di gancetti a pressione, che devono essere collegati in due punti specifici dell'Amiga: uno sulla scheda madre, ed uno sulla scheda secondaria. Quest'ultime devono infine essere collegate fra loro ed a terra.

Questa la teoria. La realtà è stata un po' differente. Innanzi tutto gli Amiga europei hanno solo la scheda madre, e l'elettronica è alquanto differente, almeno per quello che riguarda la disposizione dei punti di aggancio. Devo dire che quelli della *Michigan Software* sono stati abbastanza gentili. Infatti, quando hanno visto che la richiesta veniva dall'Italia, hanno aggiunto al manuale un foglio disegnato a mano, che riporta i collegamenti da effettuare su un Amiga europeo. Il disegno è molto chiaro, ma non è accompagnato da alcun testo, per cui le istruzioni di installazione si riferiscono comunque alla versione americana. Poco male. Con un po' di pazienza è possibile comunque arrivare a capire come va effettuato il montaggio in un Amiga europeo. Il problema grosso tuttavia si è rivelato un altro. Dopo aver aperto il sistema (operazione estremamente laboriosa in un Amiga 1000, a causa delle molte viti, vitine, e lamelle che fissano il coperchio e lo schermo RFI) ed estratto il 68000 dalla sua base, mi sono accorto che, nell'installare la scheda, un bordo di questa toccava lo schermo metallico di protezione dell'unità per i dischetti. In pratica non era possibile incastrare la scheda al posto suo. Per fortuna era solo lo schermo ad essere troppo lungo. L'unità dischetti era ovviamente delle stesse dimensioni di quella americana. Per cui ho dovuto staccare lo schermo, tagliare con un seghetto a ferro uno degli angoli, richiudere il foro con lo stesso lamierino per schermare il driver (all'inizio non lo avevo fatto con conseguenti problemi di interferenza), reinstallare il tutto, e finalmente incastrare la scheda sopra la scheda madre dell'Amiga. A questo punto ho fatto i collegamenti secondo lo schema «europeo», che tra l'altro non richiede più il terzo collegamento tra le due schede dell'Amiga, essendocene solo una in quelli europei. Ho chiuso il tutto, ed ho fatto partire il programma che prova la memoria, fornito con la scheda in un dischetto. L'unico altro problema che ho avuto, è stato un leggero prodursi di linee diagonali sullo schermo, probabilmente dovute ad interferenze tra il tubo catodico e la scheda di espansione, forse non sufficientemente schermata dallo schermo RFI dell'Amiga, oppure tra il tubo catodico e l'unità dischetti interna, a causa delle modifiche effettuate alla scatola di protezione.

A questo punto è però importante dire un paio di cose. Innanzitutto questo è successo tre anni fa, forse più, quando questa scheda era disponibile solo negli Stati Uniti e solo per il mercato americano. È possibile, anzi probabile, che oggi esista una versione della scheda anche per gli Amiga 1000 europei, a meno che la *Michigan Software* non abbia discontinuato il supporto per la serie 1000. Secondo, nonostante i problemi iniziali, la scheda ha sempre funzionato egregiamente. L'orologio dovrebbe funzionare per altri sette anni, ma spero, per quella data, di aver cambiato sistema con uno più avanzato, sempre della serie Amiga, ovviamente.

Per sicurezza comunque, se e quando ordinerà la scheda, faccia presente il fatto che va installata su un Amiga europeo, ed accenni al problema delle dimensioni dello schermo del driver, nel caso poco probabile che non ne siano ancora a conoscenza.

L'indirizzo è il seguente:
Michigan Software
43345 Grand River
Novi, MI 48050, USA.
Tel. 001-313-348-4477 e 4478
BBS 001-313-348-4479 24h
(Novi Download).

Problemi con il programma del Sig. Ficano

...La routine che il signor Davide Ficano propone non funziona assolutamente. Cioè, fa il suo dovere e fa ripartire il sistema se necessario, ma lo fa ripartire in modo assolutamente SBAGLIATO, saltando nella ROM con una TRAP, una cosa che avevo visto solo nei peggiori giochi protetti, giuro! A parte la mia ovvia avversione a tali diseducative tecniche di programmazione, in questo modo, mancando la necessaria istruzione RESET, il sistema al riavvio non riconosce né memoria aggiuntiva né altre espansioni come l'hard disk. Nulla di male, ho visto tanti programmi (ad es. Guardian) che avevano problemi di questo tipo, ma senz'altro sarebbe opportuna una segnalazione dell'errore. Non dico la corretta routine di reset fornita dalla Commodore perché si uscirebbe dagli scopi dei tuoi articoli, ma perlomeno menzionare lo sbaglio.
Nicola Salmoria (MC6489)

Detto e fatto. Non se ne risenta comunque il Sig. Ficano, anzi! Quando si scrive un programma e lo si distribuisce ad altri, bisogna mettere sempre in conto il fatto di poter ricevere segnalazioni di problemi da parte di utenti, come ben sanno gli autori di programmi PD o gli sviluppatori professionisti. Questo perché non esiste programma che non contenga errori, per quante prove e controprove si possano fare. Non solo. Spesso i problemi nascono dal fatto che un programma può funzionare perfettamente su un sistema, e tuttavia andare in tilt su un altro. Un esempio classico sono i primi programmi per Amiga, che non erano capaci di funzionare su macchine dotate di espansioni di memoria. Ecco perché la Commodore, e di fatto tutte le case produttrici di sistemi operativi, raccomandano fortemente di evitare trucchetti e di seguire certe regole [*guidelines*] anche quando non sembra esservi alcuna ragione apparente. Perché allora pubblicare procedure «non perfette» come quella del Sig. Ficano? Semplice. Innanzitutto perché, in un paese in cui il software è troppo spesso copiato e raramente prodotto, gli sforzi di chi affronta tematiche interessanti e non banali come quella relativa all'apertura di uno schermo su macchine PAL, vanno indubbiamente premiati. Secondo, e questo è il motivo principale, questa rubrica non ha lo scopo di presentare programmini interessanti (per questo c'è l'area *Amiga Software* gestita da AdP). Essa viceversa ha lo scopo di stimolare l'interesse, la curiosità, e soprattutto le capacità della vasta utenza Amiga e di indirizzarla nella non facile arte della programmazione di questa splendida e troppo spesso sottovalutata macchina. Per questo un programma come quello del Sig. Ficano rappresenta un interessante spunto di riflessione e di dibattito che, spero, porterà qualcun altro a scrivere una procedura più efficiente e sicura.

Ricordo comunque che questa rubrica riguarda principalmente la programmazione in C, e quindi in futuro pubblicherò codice Assembler solo se avrà una qualche attinenza con argomenti già trattati (come era il caso del Sig. Ficano), o qualora tale codice vada a coprire tecniche non gestibili tramite C.

Disponibilità dei sorgenti in C

Andrea Simioni (Ostia Lido) ed altri, mi hanno chiesto i sorgenti dei programmi e degli esempi riportati nella mia rubrica.

A questo riguardo sto lavorando per trovare una soluzione che vada bene a tutti, anche a quelli che non possono collegarsi con MC-Link e ricevere il codice via modem. Un po' di pazienza.

MC

"20 ANNI DI ESPERIENZA NELL'INFORMATICA GARANTISCONO PRESTAZIONI E AFFIDABILITÀ"

DATASTAR s.r.l.

VENDITA ANCHE PER CORRISPONDENZA

Prato
Via Guicciardini, 29 - Tel. (0574) 38065/7 - Fax (0574)38068

EPSON

24 aghi
80 colori
150 CPS L. 680.000

NEC

MULTISYNK
3D L. 1.154.000
4D L. 1.838.000

SCHEDA VIDEO: 1024x768

TSENG	512K	256	colori	L. 175.000
TSENG	1M	256	colori	L. 245.000
TRIDENT	1M	256	colori	L. 206.000

LaserJet III Hewlett-Packard
CON POSTSCTIPT
DA L. 2.456.000

I PC CENTRAL UNITS SONO GARANTITI 2 ANNI

TVM®

MONITOR TVM MULTISYNC 14"

DP 0,28 colore 1024x768:

3A 14" 31,5 - 35KHz	L. 560.000
4A 14" 31,5 - 38KHz	L. 680.000
3A + 14" 15-38KHz	L. 765.000
5A 15" 35 - 48KHz	L. 895.000

GARANZIA 1 ANNO ESTENDIBILE A 3 - PREZZI IVA 19% ESCLUSA
SU RICHIESTA INSTALLAZIONE E ASSISTENZA A DOMICILIO IN TUTTA ITALIA.
TRASPORTO ECONOMICO TRAMITE CORRIERE.

PREZZI INCLUSIVI DI SCHEDA MADRE, CONTROLLER AT-BUS, MEMORIA RAM, TASTIERA ESTESA microswitch CHERRY, CABINET MINI TORRE o DESKTOP* con DISPLAY. FDD 1,2Mb o 1,4Mb

CPU	MHz	LANDM.	EXP. on board fino a	memor CACHE	RAM	LIRE
486DX	25 EISA	114 MHz	64M	128K	4Mb	6.279.000
486DX	25 ISA	114 MHz	8M		4Mb	5.319.000
386DX	33 EISA	58 MHz	32M		4Mb	2.402.000
386DX	33 ISA	58 MHz	16M	64K	4Mb	2.180.000
386DX	25	42 MHz	16M	64K	1Mb	1.580.000
386DX	25	42 MHz	16M		1Mb	1.460.000
386SX	20	27 MHz	17M		1Mb	984.000
*286	16	21 MHz	5M		1Mb	652.000
*286	12	16 MHz	5M		1Mb	628.000
V20NEC	12	12 MHz	1M		512K	495.000

I nostri COMPUTERS sono AFFIDABILISSIMI, COSTANO MENO DI UN CLONE. E queste sono alcune delle caratteristiche che li rendono VELOCISSIMI e di QUALITÀ ALTAMENTE SUPERIORE:

SHADOW RAM - EEMS VER 3.2. - LIM-EMS 4.0 - SHADOW RAM - VIRTUAL MODE 386 PER MULTITASKING - DISK CACHING - PIPELINE MODE - MEMORY FETCHING - BUS INTELLIGENT - ADVANCE NETWARE - VIRTUAL MODE 386 PER MULTITASKING - DATA BASE MANAGEMENT - M.M.U (MEMORY MANAGEMENT UNIT).

TUTTI I COMPONENTI ORIGINALI DI PRIMA SCELTA INTEL etc.
GARANTITI 100% COMPATIBILI - UNIX - XENIX - LAN NOVELL.

LAPTOP (3 KG)

BATTERIE RICARICABILI
286-16 MHz
RAM 1MB EXP. 4MB
FD 1,44M
V. EGA HD 40M **L. 2.360.000**
V. VGA HD 20M **L. 2.850.000**

con CPU 386SX - 16 MHz
RAM 1MB EXP. 4MB
FD 1,44M - HDD 20MB
V. VGA HD 20M **L. 3.360.000**

Tutti con due porte ser. 1 par.
per m. multisynk e tastiera

LAPTOP (7 KG)

CON 2 SLOTS D'ESPANSIONE
RAM 1/4Mb - HDD 40M
FDD 1,44M -
con Video PLASMA:
CPU 80286-20MHz
EGA 640x400 L. 2.929.000
VGA 640x480 L. 3.160.000
CPU 80386-25MHz RAM 2/8Mb
VGA 640x480 L. 4.250.000

DISCHI RIGIDI Seagate

42Mb	28ms	At-Bus	L.	385.000
90Mb	18ms	At-Bus	L.	726.000
124Mb	18ms	At-Bus	L.	846.000
211Mb	15ms	At-Bus	L.	1.336.000
337Mb	17ms	ESDI	L.	2.130.000

GARANZIA 1 ANNO ESTENDIBILE A 3

COPROCESSORI INTEL

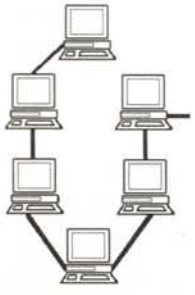
287XL 6 -20MHz	L. 316.000
387SX - 16MHz	L. 475.000
387SX - 20MHz	L. 565.000
387DX - 25MHz	L. 775.000
387DX - 33MHz	L. 945.000

Garanzia 5 anni

INVIATECI IL VOSTRO COMPUTER.
Il costo per l'installazione di un DISCO RIGIDO
e di una SCHEDA VIDEO è di L. 98.000
(Trasporti compresi)

office automation

LAN



NOVELL RETE LOCALE
Affidabilissima e Velocissima (Leader Mondiale)
Il software NOVELL può collegare ogni PC, in qualunque linguaggio siano i programmi-utilizzatore.

Il prezzo comprensivo dell'Installazione, di Software NOVELL e della Scheda ETHERNET 1Mbit/s per ogni posto di lavoro varia da L. 600.000 a L. 900.000.

SISTEMI OPERATIVI XENIX - UNIX
Il prezzo comprensivo dell'Installazione, di Software e Scheda seriale per ogni posto di lavoro varia da L. 450.000 a L.800.000

SUPER FAX 14 9600 baud
Tempo di trasmmissione 15sec.
142 memorie programmabili
16 livelli di grigio
Segreteria telefonica
Orario programmabile L 950.000
SCHEDA FAX 2400-9600 baud L 560.000
Invio/ricezione. Confortevolissima.

RIVENDITORI

Informatevi sui vantaggi di una AFFILIAZIONE GRATUITA e del PROGRAMMA COMUNE DI PUBBLICITÀ

PER PREZZI PARTI STACCATE E PERIFERICHE SI PREGA CONSULTARE LE ULTIME PAGINE