Ormai i linguaggi si assomigliano tutti, salvo naturalmente quelli non procedurali (Lisp, Prolog, ecc.) che però si assomigliano tra di loro. È quindi forse più un fatto di moda la scelta di un linguaggio piuttosto che un altro. Ecco così arrivare ondate di software tutto di un certo tipo; prima c'era il Basic poi è stata la volta del Pascal ora si nota un ritorno al Basic, grazie soprattutto ai vari Turbo e Quick, e contemporaneamente ad uno sviluppo notevole del C.
E infatti sono ambedue in C le procedure di questa volta; una si occupa di gestire meglio ... il C, l'altra di gestire meglio il buffer di tastiera

Inline in Turbo C

di Aldo Buson - Magenta (MI)

Il C è indubbiamente un linguaggio molto potente e, grazie alla Borland, anche pratico da usare. Il Turbo C con il suo ambiente integrato è quanto di meglio si possa desiderare. Ma c'è una limitazione. Chiunque abbia tentato di inserire del codice macchina in un programma in TC si sarà sicuramente accorto delle scomodità che si incontrano usando il compilatore separato. Sarebbe stato meglio implementare l'INLINE del Turbo Pascal, ma alla Borland questa strada è sembrata troppo complicata evidentemente. Un sistema per aggirare l'ostacolo comunque c'è. Il trucco sta nell'utilizzare i puntatori a funzioni che, come dice lo stesso nome, puntano all'indirizzo di una funzione. In pratica, quindi, si inseriscono i dati del codice macchina come se fosse una array di CHAR, si prepara un puntatore a funzione che punti proprio all'array e il gioco è fatto. Tramite il puntatore a funzione si può adesso eseguire il codice macchina contenuto nell'array come se fosse una funzione qualsiasi. Esempio.c (fig. 1) spiega, a grandi linee, come si debba procedere. Ovviamente non è tutto così semplice. Il primo problema che sorge è come fare a trovare i codici esadecimali che corrispondono ad un programma in Assembler. Ci sono due strade: usare il MASM o il DEBUG. Nel caso del MASM è sufficiente chiedere che venga prodotto il file .LST durante la compilazione.

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

```
char dati[]={
                              /* inizio  segment 'code'                    */
                              /* somma   proc far                          */
0X55,                         /*         push bp          ;salva bp        */
0X8B,0XEC,                    /*         mov bp,sp        ;aggiusta bp     */
0X8B,0X5E,0X06,               /*         mov bx,[bp+6]    ;bx=offset dati  */
0X8B,0X4E,0X08,               /*         mov cx,[bp+8]    ;cx=n. di dati   */
0X33,0XC0,                    /*         xor ax,ax        ;ax=0            */
0X33,0XD2,                    /*         xor dx,dx        ;dx=0            */
0X03,0X07,                    /* ciclo:  add ax,[bx]      ;ax=ax+dato      */
0X83,0XD2,0X00,               /*         adc dx,0         ;per il riporto... */
0X43,                         /*         inc bx                            */
0X43,                         /*         inc bx                            */
0XE2,0XF7,                    /*         loop ciclo                        */
0X5D,                         /*         pop bp           ;ripristina bp   */
0XCB,                         /*         ret              ;ritorno al C    */
                              /* somma   endp                              */
                              /* inizio  ends                              */
                              /* end                                       */
0x00);
                                   /*codice macchina preso da somma.lst*/

long int(far * sommanumeri)(unsigned int*,int); /*dichiarazione        */
unsigned int valori_da_sommare[5]={30000,       /*un pò di numeri da   */
            20000,12000,22,1000);               /*sommare tra loro     */

void main()
  {
  sommanumeri=(void far*)dati;                  /*inizializza          */
  printf("%ld",(*sommanumeri)(valori_da_sommare,5));
  }
```

Programma di esempio. Va compilato con il modello di memoria SMALL dato che, anche se prevede un puntatore a funzione di tipo far, il codice macchina si aspetta un DS già a posto. Il tutto si limita a sommare n. UNSIGNED INT. Il codice macchina si aspetta sullo stack un offset per i dati e un int che dica quanti sono.

Si prega il Sig. Donatello Norese di mettersi in contatto con la redazione.

Con il DEBUG si può invece sfruttare la funzione <A>ssemble annotandosi i codici. A proposito del MASM, faccio notare che il file .LST prodotto è facilmente decodificabile e infatti ho preparato un programmino che trasforma direttamente il file .LST in un file ASCII, adatto al TC, che conserva anche le istruzioni Assembler sotto forma di commento (fig. 2). Abbastanza leggibile, no? Il TC si becca gli esadecimali mentre tutti i comuni mortali possono leggersi l'Assembler. Adesso resta un solo problema: come si fa a passare un dato dal codice macchina al C o viceversa? Il C passa gli argomenti alle funzioni tramite lo stack ed è dallo stack che, con poche istruzioni, andremo a riprenderli. Supponiamo di aver compilato un programma in modo TINY contenente una chiamata ad una funzione che è stata dichiarata con

```
void funzione(int a,char b,char* c,long
              int d,char far* e)
```

Qui abbiamo un esempio di tutti i tipi più comuni. Quelli non indicati sono ad essi riconducibili tranne i numeri in virgola mobile, che vengono trattati in modo diverso, di cui non parlerò dato che vengono difficilmente usati da LM. Nel momento in cui la funzione viene richiamata lo stack si presenta come in figura 3. A questo punto la cosa più naturale è far iniziare il codice macchina con un bel:

```
push bp         ;salva bp
mov bp,sp       ;punta bp sullo stack
```

Per accedere al primo elemento della lista sarà sufficiente un

```
mov ax,[bp+4]   ;legge a
```

Niente di particolare, quindi. Basta rispettare l'ordine in cui i dati sono messi nello stack e fare i conti giusti per il displacement da aggiungere a BP. Attenzione ai CHAR, che occupano lo stesso 2 byte, ed ai LONG INT o comunque a tutti i dati che occupano più di una word. Bisogna sempre rispettare l'ordine in cui le strutture più lunghe

```
#include <stdio.h>
#include <ctype.h>
void esci(char);

void main(int argc,char *argv[])
  {
  FILE *fh1,*fh2;
  register int a,b;
  char flag1=0,flag2=0,stringa[200],dati[200];
  if(argc==1) esci(0);                                  /*c'è il nome?  */
  if((fh1=fopen(argv[1],"r"))==NULL) esci(1);           /*c'è il file?  */
  if(argc==3) for(a=0;(stringa[a]=*(argv[2]+a));a++);
  else                                                  /*nome?         */
    {
    for(b=1,a=0;(b!='.')&&(b!=0);a++) stringa[a]=(b=*(argv[1]+a));
    strcpy(stringa+a-1,".c");                           /*se non c'è    */
    }                                                   /*lo ricavo     */
  if((fh2=fopen(stringa,"w"))==NULL) esci(2);
  fputs("char ",fh2);                                   /*crea inizio   */
  for(a=0;(stringa[a]!=0)&&(stringa[a]!='.');) fputc(stringa[a++],fh2);
  fputs("[]={\n",fh2);                                  /*con il nome   */
  for(;!feof(fh1);)
    {
    fgets(dati,200,fh1);
    for(a=0,b=0;dati[a];a++)                            /*elimina tab   */
      {
      if(dati[a]==9)
        {
        for(;b&7;) stringa[b++]=' ';                    /*e RET         */
        stringa[b++]=' ';
        }
      else if((dati[a]!=13)&&(dati[a]!=10)) stringa[b++]=dati[a];
          else stringa[b++]=' ';
      }
    for(;b<79;) stringa[b++]=' ';                       /*completa con  */
    stringa[79]=0;                                      /*commenti      */
    stringa[30]='/';
    stringa[31]='*';
    stringa[78]='/';
    stringa[77]='*';
    for(flag2=0,a=1;a<5;a++) if(!isxdigit(stringa[a]))  /*cerca dati    */
                               {
                               flag2=1;
                               break;
                               }
    if((flag2)&&(flag1))                                /*se non ci sono*/
      {                                                 /*ma ha scritto */
      fprintf(fh2,"%s\n",stringa);                      /*prima scrive  */
      flag1=0;                                          /*ancora        */
      }
    if(!flag2)
      {
      for(flag1=1,b=30,a=7;a<25;)                       /*la riga con-  */
        {                                               /*tiene byte?   */
        if((isxdigit(stringa[a]))&&(isxdigit(stringa[a+1])))
          {
          fprintf(fh2,"0X%c%c,",stringa[a],stringa[a+1]);
          b-=5;                                         /*scrive dati   */
          a+=2;                                         /*nel formato C */
          }
        else a++;
        }
      if(b!=0) for(;b;b--) fprintf(fh2," ");            /*completa con  */
      fprintf(fh2,"%s\n",&stringa[30]);                 /*spazi e istru-*/
      }                                                 /*zioni         */
    }
  fputs("0x00};\n",fh2);
  puts("Tutto ok");
  }

void esci(char n)                                       /*scrive messag-*/
  {                                                     /*gio e esce    */
  char *messaggi[]=
    {
    "Devi indicare il nome",
    "Non trovo il file",
    "Non posso aprire il file"
    };
  puts(messaggi[n]);
  exit();
 }
```

Converte un file assembler.lst in un file ASCII con codici esa. Prima devi compilare il sorgente con MASM chiedendo la generazione del file .LST e poi...

LST2ASCII NOMESORGENTE [NOMEDESTINAZIONE]

obbligatorio **facoltativo**

Attenzione: il programma non è superintelligente e quindi qualche messaggio di troppo potrebbe rimanere. Ma gli editor li hanno inventati apposta....

sono memorizzate, ricordando che viene prima la parte meno significativa e poi la parte più significativa. Un esempio per tutti: -char far * e- è un puntatore far che contiene sia offset che segment. Se puntasse ad una locazione della memoria video (segment=b800,offset=1122) nello stack avremo in sequenza: 22,11,00,b8. A questo punto bisogna fare una distinzione riguardante i modelli di memoria. Per il modello TINY vale quanto detto. Per tutti gli altri bisogna ricordarsi che le funzioni sono richiamate con delle FAR CALL (e quindi DEVONO terminare con dei FAR RET) il che implica una piccola differenza. Dato che nello stack abbiamo un indirizzo di ritorno a 32 bit (e non più di soli 16) per accedere al primo elemento dovremo usare

```
mov ax,[bp+6]     ;legge a
```

e così via per tutti gli altri. Bisogna cioè prevedere 2 byte in più di displacement per BP. Lo stesso problema si presenta anche coi modelli SMALL e COMPACT che prevedono, per default, funzioni di tipo NEAR. Purtroppo il codice macchina finisce dentro il data segment e quindi non è possibile fare una CALL NEAR. Col TINY non ci sono problemi, dato che CS e DS coincidono, ma negli altri casi è necessario dichiarare FAR i puntatori a funzione. Già che ci siamo vediamo come si lavora con più di 64K di codice o dati. Con più di 64K di codice tutte le chiamate sono di tipo FAR e quindi anche la nostra dichiarazione del puntatore a funzione va modificata, a scanso di equivoci col compilatore (fig. 4). Con più di 64K di dati non ci sono problemi particolari se non che i puntatori sono tutti a 32 bit. Occhio quindi a DS. Vediamo adesso il problema inverso. Come si restituisce al C un valore? Al solito avremo a che fare con quantità a 8, 16, 32 bit. In figura 5 sono riassunti alcuni dei casi possibili. Quelli non riportati sono comunque ad essi riconducibili. Semplice vero? Basta mettere nei registri AX e DX il dato che di solito si mette nell'istruzione RETURN(..) del C. Ormai ho detto quasi tutto quello che può servire. Aggiungo solo qualche consiglio. Innanzi tutto non serve salvare tutti i registri all'interno della funzione in codice macchina. Il TC salva tutto quello che non deve essere alterato prima della CALL alla funzione. Bisogna lasciar stare solo i registri di segmento, tranne ES che è disponibile. Non bisogna preoccupparsi neppure di ripulire lo stack dai dati perché questo lavoro lo fa la funzione chiamante. Prima di augurarvi buon lavoro vi invito a dare un'occhiata al programmino d'esempio. Buon divertimento!

Fig.1:esempio.c

```
void (*punt_a_funzione)(void);                  /*dichiarazione
                                                puntatore/*
char codice_macchina[]={0xaa,0xbb,0xcc};        /*array del
                                                codice macchina*/
void main()
  {
  punt_a_funzione=codice_macchina;              /*inizializza
                                                il puntatore*/
  (*punt_a_funzione)();                         /*esegue il codice
  }                                             macchina*/
```

Fig.2:esempio dell'uso di LST2ASCI.COM

```
iniz segment 'code'    ¦  c  ¦  char nome[]={
prog proc  near        ¦  o  ¦              /*iniz segment 'code'*/
     mov ah,..         ¦  n  ¦              /*prog proc near     */
     mov bh,..         ¦  v  ¦  0x..,0x.., /*     mov ah,..     */
     ret               ¦> e >¦  0x..,0x.., /*     mov bh,..     */
prog endp              ¦  r  ¦  0x..,      /*     ret           */
iniz ends              ¦  s  ¦              /*prog endp          */
     end               ¦  i  ¦              /*iniz ends          */
                       ¦  o  ¦              /*     end           */
                       ¦  n  ¦  0x00};
                       ¦  e  ¦
```

Fig.3: Lo STAK come lo vede -funzione()-

```
                dw e1,e2              ;2 word per char far*
                dw d1,d2              ;2 word per long int
                dw c                  ;1 word per char*
                db 0,b                ;1 word per char
                dw a                  ;1 word per int
Stack Pointer>>>> dw indirizzo_per_RET
```

Fig.4:esempio2.c

```
void (far * punt_a_funzione)(void);             /*dichiarazione
                                                puntatore/*
char codice_macchina[]={0xaa,0xbb,0xcc};        /*array del
                                                codice macchina*/
void main()
  {
  punt_a_funzione=(void far*)codice_macchina;   /*inizializza
                                                il puntatore*/
  (*punt_a_funzione)();                         /*esegue il codice
  }                                             macchina*/
```

Fig.5: restituzione di valori da codice macchina

```
Dim. dati:        8bit           16bit              32bit
Esempi   :        char        int,char*,int*     long int,int far*
                    ¦              ¦                  ¦
Codice   :   mov al,valore   mov ax,valore    mov ax,parte_bassa
             ret             ret              mov dx,parte_alta
                                              ret
```

Svuota Keyboard Buffer

di Donatello Norese

Il problema nasce dalla caratteristica dell'MS-DOS di consentire un buffer di tastiera, utile in tutte le situazioni, tranne nel caso in cui si abbia, per esempio, un testo molto lungo stampato a video mediante l'istruzione «type».

Infatti, se prima del «Ctrl-S» si schiacciano inavvertitamente altri tasti, il buffer si riempie e non si riesce più a fermare lo scrolling, a meno che non ci si affidi al «Ctrl-C» che interrompe definitivamente l'operazione (ma è sempre noioso ricominciare da capo la visualizzazione del testo). Il programma presentato fornisce una soluzione al problema: non appena si schiaccia il tasto «Ctrl» (non necessariamente associato a qualche altro tasto) si provoca lo svuotamento del buffer della tastiera.

La locazione 0000:0417h contiene il byte con gli indicatori dello stato della tastiera, cioè ad ogni bit è associata un'informazione su quale o quali tasti senza un codice associato (Alt premuto, Ctrl premuto, e così via) sono stati digitati dall'utente, fornendo così una guida alla INT09h per la determinazione dei codici dei tasti premuti (Alt F, Ctrl-C, ecc.).

Il bit numero 2 della locazione contiene il flag di controllo per sapere se il tasto «Ctrl» è stato premuto: basta fare una operazione AND fra la locazione e il valore «4» (00000100 in binario) per sapere se il tasto «Ctrl» e stato schiacciato.

Le locazioni 0000:041Ah (insieme con la 0000:041Bh) e 0000:041Ch (insieme con la 0000:041Dh) puntano rispettivamente al successivo carattere da reperire nel buffer e alla prossima posizione inutilizzata del buffer. Se i due indirizzi sono uguali per il computer il buffer è vuoto. I due valore tra parentesi contengono sempre lo stesso valore (04h), mentre per le altre due l'uguaglianza dipende dal fatto che ci siano o non ci siano tasti nel buffer. Le locazioni utilizzate da tale buffer vanno da 0000:41Eh a 0000:043Ch.

Le funzioni utilizzate per allocare il programma e reindirizzare l'interrupt 09h sono proprie del Microsoft C e necessitano di una spiegazione qualora si volesse tradurre l'algoritmo in un linguaggio diverso:

_chain_intr (puntatore) = richiama l'interrupt il cui indirizzo iniziale è posto in «puntatore».

_dos_getvect (numero interrupt) = restituisce l'indirizzo iniziale dell'interrupt fra parentesi.

_dos_setvect (numero interrupt, puntatore) = rimpiazza nel vettore delle interruzioni il vecchio indirizzo dell'interrupt specificato con quello posto in «puntatore»

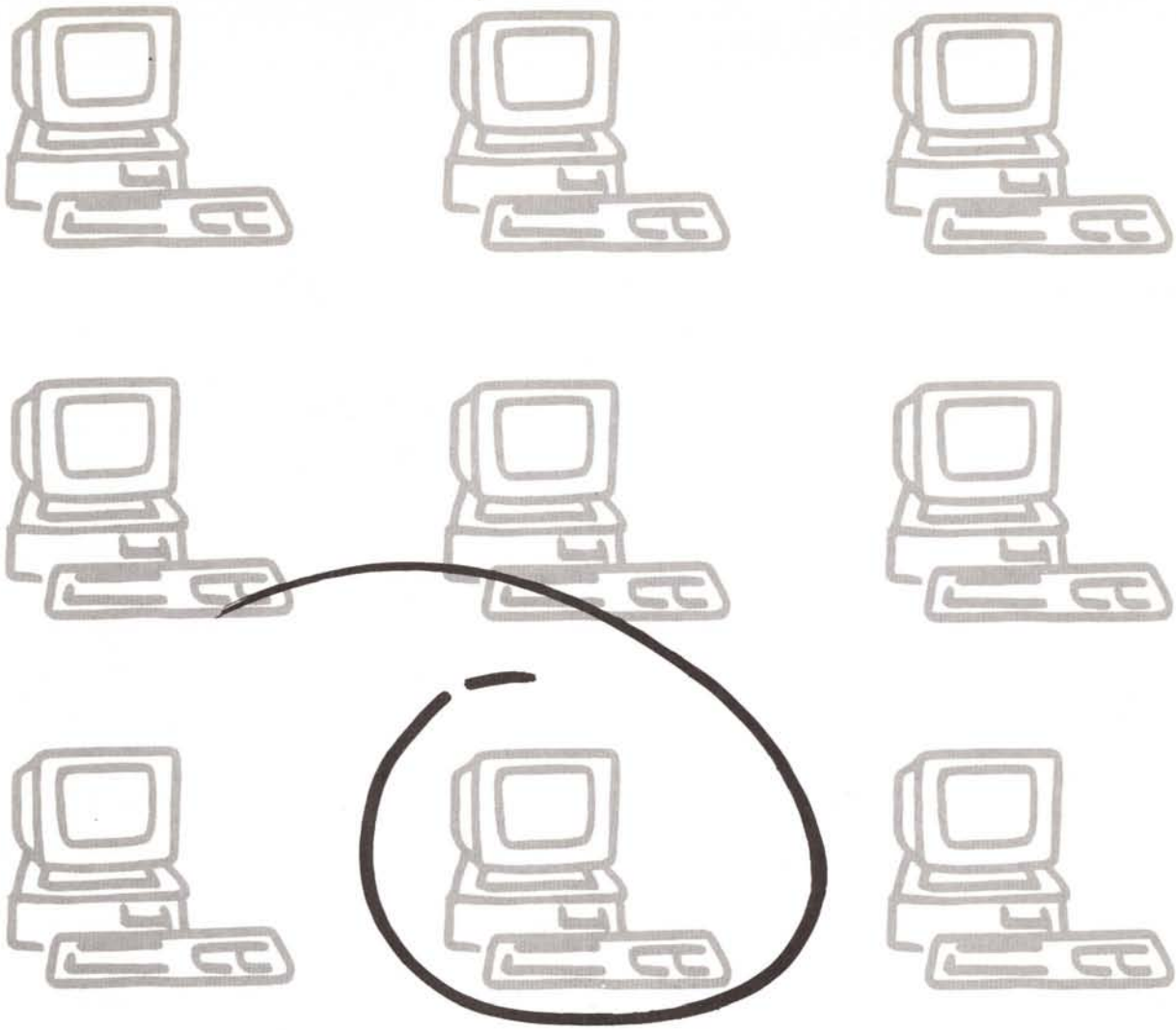
_dos_keep (codice di ritorno, numero paragrafi) = rende permanente un programma in memoria. Il codice di ritorno è un codice che può venire usato da un programma chiamante per effettuare controlli sulla riuscita dell'operazione, mentre il numero di paragrafi è la lunghezza totale del programma in memoria (compresi il program-header e il PSP) divisa per sedici.

Detto questo, non rimane che spiegare come funziona il programma: la funzione «Main» richiama la routine di installazione e controlla se il programma è già stato allocato; se lo è già il programma termina con un messaggio. La routine «assume_control» ha invece il compito di salvare l'indirizzo della vecchia INT 09h e di inserire al suo posto nel vettore delle interruzioni l'indirizzo di partenza della mia routine. La routine «ctrl_control» è infine il cuore del sistema; è una interrupt che controlla se è stato schiacciato il tasto «Ctrl» e qualora la condizione si verifichi azzera il buffer uguagliando il valore della locazione 0000:041Ah con quello della 0000:041Ch; il controllo viene poi ritornato alla vecchia INT 09h.

```
/*
   ┌──────────────────────────┐
   │   KEYBOARD BUFFER CONTROL   │
   │                          │
   │      DONATELLO NORESE    │
   │        10/03/1989        │
   └──────────────────────────┘*/

# include <dos.h>                          /*Libreria contenente le funzioni*/
                                           /*_chain_intr, _dos_getvect, FP_OFF */
                                           /*_dos_setvect, _dos_keep.        */
                                           /*                                */
# include <stdio.h>                        /*Libreria contenente la funzione*/
                                           /*printf.                         */
                                           /*                                */
void far *function;                        /*Puntatore alla prima  locazione*/
                                           /*della routine ctrl_control.     */
                                           /*                                */
void far *old_int9_value;                  /*Puntatore alla prima  locazione*/
                                           /*della vecchia INT 09h.          */
                                           /*                                */
void far *answer;                          /*Puntatori  di  appoggio  per*/
void far *enquiry;                         /*determinare se il programma è  già*/
                                           /*stato installato.               */
                                           /*                                */
char far *kb_ctrl=(char far *)0x00000417L; /*Puntatore     alla     locazione*/
                                           /*0000:0417h.                     */
                                           /*                                */
char far *buffer_head=(char far *)0x0000041AL;  /*Le   locazioni   0000:041Ah   e*/
                                           /*0000:041Bh puntano  al  successivo*/
                                           /*carattere da reperire  nel  buffer*/
                                           /*circolare della tastiera.       */
char far *buffer_tail=(char far *)0x0000041CL;  /*Le   locazioni   0000:041Ch   e*/
                                           /*0000:041Dh puntano  alla  prossima*/
                                           /*locazione inutilizzata nel  buffer*/
                                           /*circolare della tastiera.       */
                                           /*                                */
void interrupt far ctrl_control (){        /*Se  il  bit  n°2  della  locazione*/
   if (*kb_ctrl&0x04)                      /*0000:0417h  è  acceso  (1  logico)*/
      *buffer_tail=*buffer_head;           /*viene  azzerato  il  buffer  della*/
   _chain_intr (old_int9_value);           /*tastiera.                       */
}                                          /*                                */
void assume_control (){                    /*Salva  l'indirizzo  della  vecchia*/
   old_int9_value=_dos_getvect (0x09);     /*INT  09h  e  scrive  quello  della*/
   function=ctrl_control;                  /*routine 'ctrl_control' nel vettore*/
   _dos_setvect (0x09,function);           /*delle interruzioni.             */
}                                          /*                                */
void main (){                              /*Controlla se il  programma  è  già*/
   answer=ctrl_control;                    /*stato  installato  e  richiama  la*/
   enquiry=_dos_getvect (0x09);            /*routine di installazione.       */
   if (FP_OFF (answer)==FP_OFF (enquiry))  /*Se   l'offset   degli   indirizzi*/
                                           /*della  routine  'ctrl_control'   e*/
                                           /*della  INT  09h  sono  uguali,  il*/
                                           /*programma è già stato installato. */

      printf ("\nIl programma KB_CTRL.EXE è già stato installato!\n\0");
   else {
      assume_control ();
      function=ctrl_control;
      if (function==_dos_getvect (0x09))
         printf ("\nIl programma KB_CTRL.EXE è stato installato con successo!\n\0");
      _dos_keep (1,600);
   }
}
```



DISCOM

Da sempre Discom è preparata per correre e vincere. La sua professionalità e il suo dinamismo fanno della Discom una società di distribuzione tra le più trainanti: le proposte più adeguate e i prezzi più competitivi per i prodotti vincenti, cioè i migliori, per Voi.

00128 Roma - Via Marcello Garosi, 23
Telef. (06) 52.07.839-52.07.917-52.02.293 - Telex 620238 - Telefax (06) 52.05.433

SOLO I MIGLIORI. PER VOI.