Una manciata di utility per questa puntata di software MS-DOS complete dei listati sorgente. I primi tre programmini vengono da un programmatore professionista e servono a controllare meglio gli errori dei dischi. L'ultimo programma invece, scritto in Assembler, permette di sapere, da un file batch in esecuzione, la posizione attuale del cursore

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. 265.

Gestione errori critici da dischi

di Giovanni Viva, Roma

In questo articolo presentiamo alcuni programmi in grado di prevenire delle situazioni di emergenza per il DOS che vanno sotto il nome di «errori critici». Ogni volta che il DOS si trova in una di queste situazioni che gli impediscono di proseguire, chiama l'interrupt 36. La descrizione di ciò che accade è complessa ed esula dai nostri scopi. È utile soltanto sapere che vengono restituite informazioni sull'errore nei registri. In alcuni casi però, particolarmente nelle attività del disco, il DOS chiede all'utente di scegliere tra tre possibilità: ignorare l'errore, ritentare l'operazione o porre termine al programma. Ciò provoca la comparsa sul video della dicitura «Abort, Retry, Ignore» e l'esecuzione del programma viene momentaneamente sospesa.

In alcuni casi ciò comporta notevoli fastidi.

Ad esempio, quando si lavora in modalità grafica Hercules con il Turbo Graphix Toolbox, compaiono sul video dei puntini che disturbano l'immagine, con la sospensione del programma stesso.

Questi inconvenienti, oltretutto, rischiano di rendere il programma meno «professionale».

Non dimentichiamo che, se per errore si batte il tasto 'A' (Abort), si torna in ambiente DOS con la perdita di dati introdotti. Le situazioni che vogliamo controllare, attraverso l'uso del Turbo Pascal, sono:

1) individuazione del drive (hard, floppy, virtuale ecc.);
2) nel caso di un floppy, verificare la presenza del dischetto;
3) nel caso di un floppy, verificare la presenza della targhetta di protezione della scrittura.

Vogliamo comunque precisare che ta-

```
program VerificaProtezioneScritturaSuDisco;
var
 CodErr:byte;

function ProtezioneScrittura(Dsk:byte;  (* FUNZIONE CHE *)
 var CodErr:byte):boolean;  (* VERIFICA SE UN DISCO E' *)
type                        (* PROTETTO DALLA SCRITTURA *)
 BufferType=array[1..512] of byte;   (contiene un settore)
var
 Buffer:BufferType;
 Opzione,Lettore:char;
 Risultato:boolean;

function LetDisk(Lettore,LetScrit:char;Lato,Traccia,
Settore,NumSettori:integer;var CodErr:byte):boolean;
type                    (LETTORE=lettera del drive)
 ListaRegistri=record (LETSCRIT:'l'=lettura, 's'=scrittura)
   ax,bx,cx,dx,bp,si,di,ds,es,     ('v'=verifica)
   flags:integer
 end;
ListaLettori=set of char;
ListaOpzioni=set of char;
var
 Registro:ListaRegistri;
 LL:ListaLettori;
 LO:ListaOpzioni;
 Opzione,IndiceFor,Appoggio:integer;
begin
LL:=['a'..'e','A'..'E'];
LO:=['s','l','v','S','L','V'];
if (Lettore in LL) and (letscrit in LO) then
 begin
 case Lettore of
  'a','A':Appoggio:=$00;
  'b','B':Appoggio:=$01;
  'c','C':Appoggio:=$02;   (descritto già in precedenza)
  'd','D':Appoggio:=$03;
  'e','E':Appoggio:=$04
 end;
 case LetScrit of
  's','S':Opzione:=3;
  'l','L':Opzione:=2;  (funzione nel registro AH)
  'v','V':Opzione:=4
 end;
 for IndiceFor:=1 to 3 do
  begin
   Registro.ax:=$0000;   (chiamata al servizio 0 in modo da)
   intr($13,Registro)    (ripristinare il sistema di dischi)
  end;
 for IndiceFor:=1 to 3 do
  begin
  Registro.dx:=256*lato+Appoggio; (DH=lato, DL=lettore)
  Registro.cx:=Traccia*256+Settore;(CH=traccia, CL=settore)
  Registro.ax:=256*Opzione+NumSettori; (num. settori)
  Registro.es:=seg(Buffer);(posizione dell'area di memoria)
  Registro.bx:=ofs(Buffer);(dove i dati sono collocati)
  intr($13,Registro)
  end;
 Appoggio:=Registro.flags and 1; (risultato del CARRY FLAG)
 CodErr:=hi(Registro.ax);(codice errore nel registro AH)
 if Appoggio=1 then
  LetDisk:=false  (vi è un errore)
    else
  LetDisk:=true   (tutto OK)
 end
   else
 begin
 writeln('Lettore errato');
 CodErr:=255;
 LetDisk:=false
 end;
end;

begin  (* INIZIO PROTEZIONE SCRITTURA *)
Lettore:=chr(65+Dsk);(se Dsk=0 lettore='A', Dsk=1 'B', ecc)
Risultato:=LetDisk(Lettore,'l',0,0,1,1,CodErr); (lettura)
if not Risultato then  (operazione di lettura fallita)
 ProtezioneScrittura:=Risultato
      else  (operazione di lettura OK)
 begin
 Risultato:=LetDisk(Lettore,'s',0,0,1,1,CodErr);(scrittura)
 if not Risultato then  (operazione di scrittura fallita)
  ProtezioneScrittura:=Risultato
       else  (operazione di scrittura OK)
  begin
  Risultato:=LetDisk(Lettore,'v',0,0,1,1,CodErr);(verifica)
  ProtezioneScrittura:=Risultato
  end
 end
end;

begin  (* PROGRAMMA PRINCIPALE *)
writeln(ProtezioneScrittura(0,CodErr)); (TRUE=non protetto)
writeln(CodErr)                         (FALSE=protetto)
end.
```

Figura 3

```
end.program VerificaDispositivoAmovibile;
var
 Appoggio:byte;  (variabile di appoggio)
 Lettore:char;   (A..E)

function Amovibile
 (Lettore:char):byte; (* PROCEDURA CHE VERIFICA SE IL          *)
type                  (* DISPOSITIVO E' A DISCHI FLESSIBILI, *)
                      (* A DISCO RIGIDO O ALTRO              *)
 ListaRegistri = record  (* UTILIZZABILE SOTTO VERSIONI DOS 3.0 *)
                         (* O SUCC.                            *)
  ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
  end;
var
 Lista:ListaRegistri; (registri per le chiamate al DOS)
 LL:set of char; (insieme dei lettori utili)
begin
LL:=['A'..'E','X'];     (A..E drive ordinari, X drive di default)
if Lettore in LL then  (se il lettore è utile, procedi)
 begin
 Lista.ax :=$4408;             (in AH valore della funzione 44 HEX)
 if Lettore='X' then           (in AL valore della sottofunzione 8 HEX)
  Lista.bx:=0                  (in BX numero del lettore)
    else                       (0=default, 1='A', 2='B', ecc.)
  Lista.bx :=ord(Lettore)-64;  ; (ASCII del lettore - 64)
 MsDos(Lista);                 (chiamata all'interrupt 33)
 Amovibile:=Lista.ax           (in AX vi è il risultato:    )
 end                           (AX=0  dispositivo amovibile )
    else                       (AX=1  disco fisso           )
 Amovibile:=15                 (AX=15 dispositivo non valido)
end;              (FINE FUNZIONE)

begin             (INIZIO PROGRAMMA PRINCIPALE)
Lettore:='X'; (scelto lettore di default)
Appoggio:=Amovibile(Lettore); (chiamata alla funzione)
if appoggio=0 then
 writeln('Il drive ',Lettore,' è munito di mezzi amovibili')
   else
 if Appoggio=1 then
  writeln('Il drive ',Lettore,' è fisso')
    else
  writeln('Il drive ',Lettore,' o non esiste o è diverso o è errato')
end.
```

Figura 1

```
program VerificaDiPresenzaDiDisco;
var
 CodiceErrore:integer;
function ExistDisk(Lettore:char; (* FUNZIONE CHE CONTROLLA *)
var CodiceErrore:integer):boolean; (* CHE UN DISCO SIA *)
type                               (* INSERITO NEL DRIVE E CHE LO *)
 ListaRegistri=record              (* SPORTELLINO SIA CHIUSO *)
   ax,bx,cx,dx,bp,si,di,ds,es,
   flags:integer
 end;
ListaLettori=set of char; (insieme di lettori utili)
var
 Registro:ListaRegistri;
 LL:ListaLettori;
 IndiceFor,Appoggio:integer;
begin
LL:=['a'..'e','A'..'E'];
if Lettore in LL then
 begin
 case Lettore of
  'a','A':Appoggio:=$00;
  'b','B':Appoggio:=$01;
  'c','C':Appoggio:=$02;  (in APPOGGIO vi è il codice)
  'd','D':Appoggio:=$03;  (del drive)
  'e','E':Appoggio:=$04
 end;
 for IndiceFor:=1 to 3 do  (esegue tre volte affinchè il)
  begin  (motore del lettore raggiunga la giusta velocità)
  Registro.dx:=$0000+Appoggio;  (DL=numero del lettore)
        (0='A', 1='B', 2='C' ecc)  (DH=numero di lati)
  Registro.cx:=$0001; (CL=num. settore)(CH=numero traccia)
  Registro.ax:=$0401; (AH=selezione verifica)
                      (AL=numero di settori da leggere)
  intr($13,Registro)  (chiamata all'interrupt $13)
  end;
 Appoggio:=Registro.flags and 1; (se APPOGGIO=1 c'è errore)
 CodiceErrore:=hi(Registro.ax); (AH contiene bit di stato)
 if Appoggio=1 then
  ExistDisk:=false
    else
  ExistDisk:=true  (la funzione restituisce TRUE se vi)
 end               (è un dischetto)
   else
 begin
 writeln('Lettore errato');
 CodiceErrore:=255;
 ExistDisk:=false
 end
end;

begin
writeln(ExistDisk('A',CodiceErrore)); (chiamata alla)
writeln(CodiceErrore)                 (funzione)
end.
```

Figura 2

li routine possono essere convertite anche per altri linguaggi di programmazione che prevedano l'uso diretto degli interrupt.

Iniziamo da una funzione in grado di indicare se un dispositivo sia munito, o meno, di mezzi amovibili (dischi flessibili).

Questo è estremamente utile per i nostri scopi poiché permette ad un programma di controllare il cambio dei dischi oppure di poter contare sulla presenza costante dello stesso disco.

Questo programma utilizza l'interrupt 33 tramite la funzione del Turbo Pascal MS-DOS. I numeri delle singole funzioni si collocano nel registro AH. Nel nostro caso si è utilizzata la funzione 68 che, a sua volta, richiede che nel registro AL sia posta una sottofunzione, nel nostro caso la numero 8. Nel registro AX si ottiene il risultato commentato ampiamente nel listato di figura 1.

Questo programma necessita del DOS 3.0 o versioni successive.

Passiamo, ora, ad un programma in grado di «percepire» la presenza di un dischetto in un floppy disk. Diversamente dalla routine precedente, quella che segue sfrutta i servizi del ROM BIOS per i dischetti. Viene chiamato l'interrupt 19 con il servizio 4 nel registro AH.

Il servizio utilizzato consiste nella verifica di uno o più settori ed è normalmente utilizzato dopo la scrittura di un settore. Nel nostro caso serve soltanto a verificare la presenza del dischetto tramite il CARRY FLAG, che risulta 1 se non vi è dischetto, 0 in caso contrario. Il listato è quello di figura 2.

Il programma di figura 3 verifica la presenza della targhetta di protezione della scrittura. Ciò è possibile utilizzando il servizio di scrittura del ROM BIOS numero 3 nel registro AH tramite l'interrupt 19. Nel caso in cui il disco abbia la targhetta di protezione, dopo il tentativo di scrittura di un settore, viene restituito il codice di errore 3 nel registro AH con il CARRY FLAG settatto ad 1 altrimenti il settore viene riscritto. Per evitare che questa ultima operazione alteri parte di un file presente in tale settore, si è utilizzato un semplice accorgimento descritto in seguito:

A) leggere, mediante il servizio 2, il settore 1 della traccia 0 del lato 0 ed immagazzinare queste informazioni in un array lungo 512 byte di byte.

B) Scrivere sul settore precedente le stesse informazioni, se il disco risulta protetto accade ciò che è stato descritto in precedenza altrimenti l'operazione di scrittura avviene regolarmente.

C) Eseguire anche un'operazione di verifica per rivelare possibili altri errori.

POS-CUR

di Corrado Mayer, Roma

Scopo del programma

Il programma POS-CUR permette di sapere durante l'esecuzione di un file batch in quale riga del video si trova il cursore. È utile soprattutto per chi possiede un PC con molta memoria e utilizza un disco virtuale; il file autoexec del dischetto con cui solitamente si bootstrappa, contiene anche tanti comandi del tipo COPY A:WS.COM C: oppure COPY A:*.COM C:, ora sia in un caso che nell'altro, vengono scritte molte righe sul video e quindi si **perdono i messaggi del self test** che la macchina compie all'accensione. Considerando inoltre che normalmente non si rimane a guardare il video mentre il PC (soprattutto il vecchio IBM che ho io!) compie il self test ed esegue la batch è chiaro a cosa serve POS-CUR: senza di lei potrebbero passare dei mesi prima che uno si accorga che, per esempio, c'è un baco nella ROM.

```
TITLE   POS-CUR.ASM   ver. 1.01

PAGE 64,132

; Definizione delle costanti di programma

dosfun          equ     21h             ; DOS function request interrupt

;codifica delle funzioni DOS
pstring         equ     09h             ; Print STRING
exit_erl        equ     4ch             ; exit con errorlevel

;codifica delle funzioni BIOS
videoio         equ     10h             ; video i/o

;codifica delle sottofunzioni di videoio
read_cur_pos    equ     3               ;Read cursor position


CODE    SEGMENT PARA
        ASSUME  CS:CODE, DS:CODE

        org 100h
inizio:
jmp start

;  ZONA DATI

cr      equ 13
lf      equ 10

nome    db cr,lf
        db 'POS-CUR V. 1.01',cr,lf
        db '(C) Corrado Mayer 1988',cr,lf,lf
        db 'Corso Trieste, 128',cr,lf
        db '  00198  ROMA ',cr,lf,lf,'$',1ah

mess    db 'questa ',138,' la riga: '           ; 138 = e accentata
numrig  dw 0
        db '$'


start:
incode  proc    far

        mov     ax,cs                   ;inizializzo DS
        mov     ds,ax

        mov     ah,read_cur_pos         ; 3 = read cursor position del BIOS
        xor     bx,bx                   ; pagina attiva = 0
        int 10h                         ; output in dh,dl = riga,col
  push dx                               ; salvo per dopo il risultato.
        mov     al,dh                   ; Metto il risultato
        xor     ah,ah                   ;  in ax e lo
        call binasc                     ;  traduco in ascii.
        mov     numrig,ax               ; E lo stampo sul video
        mov     ah,pstring
        mov     dx,offset mess
        int dosfun
  pop dx                                ; recupero il risultato e lo
        mov     ah,exit_erl             ; rimando come errorlevel
        mov     al,dh
        int dosfun

incode  endp

;-BINASC-------------------------------------

binasc  proc near

  push cx
        xor     ah,ah
        mov     cl,10
        div     cl                      ;al=decine,ah=unita'
        add     al,48
        add     ah,48
  pop cx

  ret
binasc  endp


CODE    ENDS

        END inizio
```

Normalmente il programma verrà richiamato nel file autoexec e sarà il primo comando. Vediamo come inizia un tipico autoexec che lo contiene:

```
POS-CUR
IF ERRORLEVEL 10 PAUSE - C'È QUALCOSA
CHE NON VA
...
...
```

Come si può capire dall'esempio il programma POS-CUR assegna alla variabile ERRORLEVEL del sistema operativo il numero della riga attuale (la successiva a quella in cui è richiamato POS-CUR). Nell'esempio è stato testato se ERRORLEVEL è maggiore o uguale a 10 perché normalmente POS-CUR viene eseguito con il cursore alla riga 9; se il self test dà uno o più messaggi di errore, tutto il seguito si svolge spostato di una o più righe in basso e quindi l'esecuzione della batch viene sospesa dal PAUSE (così possiamo leggere il messaggio del self test). Ovviamente tutto ciò ha senso se POS-CUR viene richiamato prima della riga 23 (quando tutto va bene) perché altrimenti per effetto dello scroll il risultato sarà sempre e comunque 24.

Uso del programma

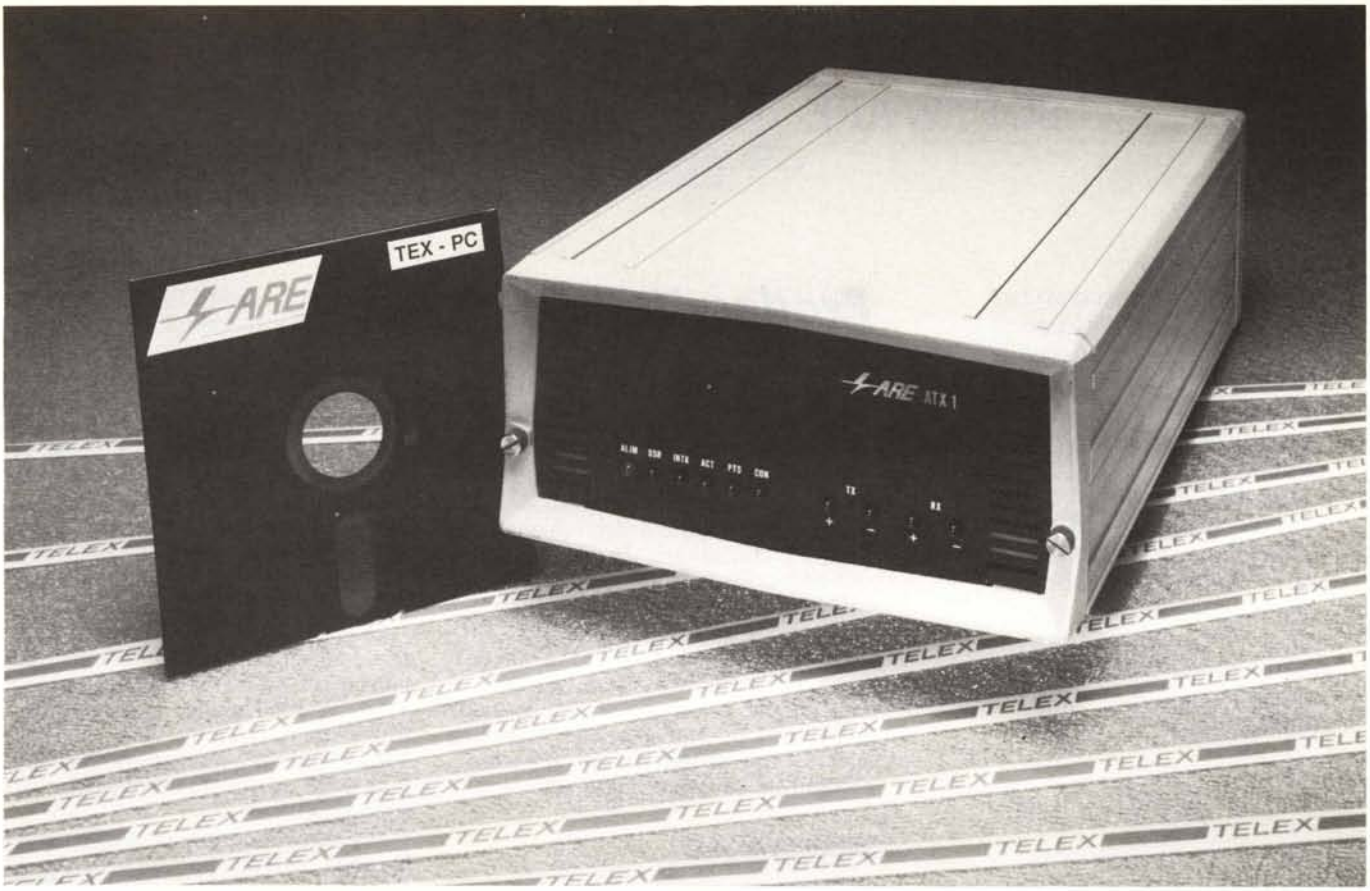
Il programma va semplicemente richiamato dal file autoexec. Consigliamo di metterlo come primo comando, ma non è essenziale. Procedendo per tentativi, due o tre di solito bastano, bisogna determinare qual è la riga «normale» in cui viene eseguito il programma per poter poi effettuare il test corretto su ERRORLEVEL. Per facilitare l'individuazione del valore corretto il programma scrive anche sul video la riga a cui si trova il cursore (scrive «questa è la riga: ??»). Le righe sono numerate da 0 a 24 a partire dall'alto.

Se si vuole utilizzare il programma per altri scopi, ad esempio in una batch complessa se si vuole eseguire un CLS solo se mancano meno di 10 righe alla fine del video, o se la scritta effettuata dal programma dà fastidio, si possono togliere (ovviamente chi possiede un Assembler e conosce l'Assembly) le parti relative alla scritta sul video che non sono essenziali per il suo funzionamento. In particolare vanno tolte:

— le tre righe ad iniziare dall'etichetta «mess»;
— le nove righe tra il «push dx» e il «pop dx» compresi, nella procedura di base;
— tutta la subroutine BINASC.

Ultimo commento: le righe dell'etichetta «nome» non servono al programma ma servono, chiedendo un type di POS-CUR.COM (!), a sapere che programma è (se gli abbiamo cambiato nome) e chi l'ha fatto. Notate che l'ultimo carattere è 1A in esadecimale che equivale a un EOF e che ferma il type.

MC



RETE TELEX

ARE
Applicazioni Radio Elettroniche S.p.A.

La gestione ottimale della rete telex si realizza con gli adattatori **ATX 1** e **UTA 1,** due soluzioni **ARE** per rispondere sia ad esigenze semplici che sofisticate.

ATX 1 è un adattatore molto versatile, pensato per collegare qualsiasi area aziendale alla rete telex attraverso un terminale o un personal computer. Un programma specifico, di facile uso, (**PC-TEX**) agevola la gestione dei testi, l'invio automatico o in differita dei messaggi, la memorizzazione dei testi in arrivo.

UTA 1 è un adattatore intelligente che effettua in modo automatico le conversioni di codice (ASCII-BAUDOT) e di velocità per utilizzazioni riservate a sistemi più potenti.

UTA 1 esegue automaticamente le procedure di controllo e disconnessione della centrale telex. Inoltre esegue più tentativi di richiamata automaticamente, quando la prima chiamata non è andata a buon fine e si avvale di pacchetti software per sistemi IBM 3X e 43XX.

Entrambi gli adattatori hanno un punto di connessione per la telescrivente e gestiscono le selezioni in due tempi delle chiamate internazionali.

IBM è un marchio registrato dalla International Business Machine

APPLICAZIONI RADIO ELETTRONICHE - Castellanza (VA) - Tel. (0331) 59.41.35 - Telex 331223 ARECAS I - Telefax (0331) 59.40.30