di Sergio Polini

Come usare... le istruzioni che mancano

Paragonato al Basic il Turbo Pascal sembra «povero», almeno se si conta il numero delle parole riservate: poche decine contro più di 200. La realtà è ovviamente ben diversa: il nucleo del Pascal è molto semplice solo per consentire la massima creatività e flessibilità, e attorno a questo nucleo è facile costruire quello che serve sotto forma di funzioni e procedure. Secondo il bisogno. La Borland fornisce infatti numerose funzioni e procedure pre-definite (pag. 38 del manuale), ma nulla vieta di usare il nome di una di queste per una propria routine o di aggiungerne altre. La comoda interfaccia con il sistema operativo fa poi il resto. In una parola, si può fare «tutto», non solo quello che offre un elenco — lungo quanto si vuole ma pur sempre limitato — di parole riservate.
Nel numero 69 di MC Maurizio Mauri ha offerto interessanti e utili esempi di questa flessibilità per la versione CP/M su sistemi MSX; in questo e nel prossimo numero vedremo anche qualcosa per l'ambiente PC-DOS

Potenze

Un operatore di elevazione a potenza non c'è; c'è solo una funzione Sqr che serve per calcolare il quadrato di un numero. Non è un problema: basta ricordare che x elevato a y è uguale a exp(y*ln(x)).

La funzione PotRR (listato n. 1) calcola appunto in questo modo le potenze con basi ed esponenti reali, prendendosi cura di tutti i possibili casi di errore (non si può calcolare il logaritmo di un numero negativo, 0 elevato a 0 è indefinito, elevare 0 ad un esponente negativo equivale a dividere per 0, ecc.). Una funzione PotIR (base intera e esponente reale) sarebbe del tutto analoga.

Se però l'esponente è intero possiamo costruirci una funzione PotRI più veloce (sempre nel listato n. 1) grazie ad un algoritmo molto antico, le cui prime tracce si trovano nella matematica indù e risalgono a circa il 200 a.C. (ne potete trovare una esposizione dettagliata nei *Seminumerical Algorithms* di Knuth, pp. 441-444). Si tratta in sintesi di scomporre l'esponente: x^5 è uguale a (x^4 * x), ovvero a ((x * x)^2 * x) e può quindi essere calcolata con sole tre moltiplicazioni.

Locate

L'istruzione LOCATE del Basic consente di posizionare il cursore come la procedura GotoXY del Turbo Pascal, ma anche di farlo scomparire o di determinarne la forma. C'è un interrupt del BIOS che provvede a queste cose ed è quindi facile farsi una procedura Cursore che accetti due argomenti, «Da» per la riga di scansione iniziale del cursore e «A» per quella finale, con l'intesa che se il primo parametro è maggiore del massimo numero valido per la riga di scansione finale (13 per scheda monocromatica, 7 per CGA) il cursore scompare (vedi figura A). Qui assumiamo che nel programma venga dichiarato un type Registri come illustrato a pagina 208 del manuale, comprendente sia i registri a 16 bit (AX, BX, ecc.) che i «mezzi registri» a 8 bit (AL e AH, BL e BH, ecc.). Con Cursore (INVISIBILE, 0) possiamo far sparire il cursore; per farlo riapparire dobbiamo distinguere tra video grafico e video monocromatico. Il tipo di video installato è memorizzato nell'area dati del BIOS, all'indirizzo 0:449H; tenendo presente che il codice per la scheda monocromatica è 7 possiamo fare così come in figura B.

Con gli array predefiniti Mem e MemW si ha accesso a qualsiasi locazione di memoria, con Port e PortW a qualsiasi porta, con le procedure Intr e MsDos si possono utilizzare tutti gli interrupt del BIOS e del DOS. Non è ovviamente possibile esaminare in questa sede tutte le risorse che il Turbo Pascal ci mette così a disposizione; è

Figura A

```
const
  INVISIBILE = $FF;
procedure Cursore(Da,A: integer);
var
  Reg: Registri;
begin
  Reg.AH := 1;    Reg.BX := 0;
  Reg.CH := Da;   Reg.CL := A;
  Intr($10,Reg)
end;
```

Figura B

```
if mem[$0000:$0449] = 7 then
  Cursore(12,13)
else
  Cursore(6,7);
```

tuttavia sufficiente consultare i «Technical Reference» dell'IBM o testi come quelli di Norton per averne un'idea. Tutto quello che si può fare con il BIOS e con il DOS si può fare in Turbo Pascal.

Bsave e Bload

Le istruzioni BSAVE e BLOAD del Basic copiano su disco e poi caricano nuovamente in RAM delle porzioni di memoria; vengono usate per caricare e salvare routine in linguaggio macchina (cosa inutile in Turbo Pascal, data la disponibilità degli «inline statement» o della dichiarazione «external», decisamente più comodi), ma consentono anche di salvare su disco una schermata per poi richiamarla al momento opportuno; la tecnica è utile soprattutto nel caso di schermate di help o grafiche.

Per fare la stessa cosa in Turbo Pascal dobbiamo per prima cosa definire le nostre strutture di dati (v. listato 2): una variabile S appartenente ad un tipo Schermo, e quindi un array di 25 righe per 80 colonne di coppie carattere/attributo, e un puntatore ad una struttura dello stesso tipo. La variabile cui punta SPtr non viene dichiarata in quanto la procedura Prepara, oltre ad aprire il file su disco, assegna a questo puntatore l'indirizzo della memoria-video del PC in funzione della scheda video installata (monocromatica o CGA); in questo modo la memoria-video diventa la variabile cui punta SPtr (parleremo più diffusamente dei puntatori in una prossima puntata).

A questo punto basterebbe scrivere «write(FS,SPtr^)» per il BSAVE, «read (FS,SPtr^)» per il BLOAD. Si può anche fare, ma se abbiamo una CGA in modo testo la lettura/scrittura della memoria-video produce un fastidioso sfarfallio sullo schermo. Per evitarlo usiamo la stessa tecnica che usa il BIOS per lo scrolling: in BSave «spegnamo» il video agendo sulla porta 03D8H (e quindi sul Mode-Select Register del CRT Controller), copiamo la memoria-video in S con l'istruzione Move, «riaccendiamo» il video, scriviamo S sul disco. Per riaccendere bisogna mandare alla porta 03D8H il valore che questa aveva prima dello spegnimento, letto dall'area dati del BIOS invece che dal Mode-Select Register in quanto questo può essere solo scritto. Analoga la tecnica per il BLoad (chi volesse maggiori dettagli può trovarli, oltre che nel «Technical Reference», anche in David J. Bradley, *Assembly Language Programming for the IBM Personal Computer*, Prentice-Hall, 1984). Con schermate grafiche le cose sono ancora più semplici: «Schermo» è un «array[0..16383] of byte», non dobbiamo tener conto della presenza di una scheda mono, non vi sono problemi di sfarfallio (bisogna magari ricordare che nei modi grafici la procedura ClrScr non può essere utilizzata; v. pag. 163 del manuale). Notate che in ogni caso (testo o grafica) il file su disco può contenere anche più di una schermata: ognuna in un diverso record.

Listato 1

```
program Potenze;
type
  stringa = string[255];
var
  b,e: real; i: integer;
procedure Error(s: stringa);
begin
  writeln; writeln('Errore: ',s)
end;
function PotRR(x,y: real): real;
  begin
    if x > 0.0 then PotRR := exp(y*ln(x))
    else begin
      PotRR := 0.0; { valido se x = 0 e y > 0 }
      if x < 0.0 then Error('base < 0')
      else if (x = 0.0) and (y = 0.0) then Error('0 a 0 indefinito')
      else if (x = 0.0) and (y < 0.0) then Error('0 a esp. < 0')
    end
  end;
function PotRI(x: real; n: integer): real;
  var y: real;
  begin
    if (x <> 0.0) and (n < 0) then begin
      x := 1.0/x; n := -n
    end;
    if n > 0 then begin
      y := 1.0;
      while n > 0 do begin
        if odd(n) then y := y * x;
        n := n shr 1;
        x := sqr(x)
      end;
      PotRI := y
    end
    else if (n = 0) and (x <> 0.0) then PotRI := 1.0
    else begin
      PotRI := 0.0;
      if (n = 0) and (x = 0.0) then Error('0 a 0 indefinito')
      else if (n < 0) and (x = 0.0) then Error('0 a esp. < 0')
    end
  end;
begin
  write('Base reale, esponente reale: '); readln(b,e);
  writeln(PotRR(b,e):10:10);
  write('Base reale, esponente intero:'); readln(b,i);
  writeln(PotRI(b,i):10:10)
end.
```

Environ$ e Shell

L'environment è un'area di memoria attraverso la quale il DOS passa parametri ai programmi che mette in esecuzione. Il principale è COMSPEC, il cui valore è il pathname del file COMMAND.COM, che deve talvolta essere ricaricato da disco quando il programma termina. Altri parametri predefiniti dal DOS sono PATH e PROMPT, ma se ne possono aggiungere anche alcuni definiti dall'utente. Si può ad esempio volere che un programma cerchi il suo file di dati in una subdirectory il cui path costituisca il valore di un parametro DATI dell'environment, e ottenere così il corretto funzionamento del programma quali che siano le subdirectory in cui risiedono lui e il suo file di dati.

Per far ciò occorre usare il comando SET del DOS (ad es.: SET DATI=C:\PIPPO\PLUTO), ma occorre soprattutto che il programma sia in grado di leggere l'environment. In Basic si può usare ENVIRON$, in Turbo Pascal... la funzione GetEnv (listato 3). L'indirizzo dell'environment di un programma è contenuto nel Program Segment Prefix (illustrato da Pierluigi Panunzi nel numero 69 di MC), ogni parametro è seguito da un «=», dal suo valore e da uno zero, alla fine c'è un altro zero; tutti i caratteri sono maiuscoli. GetEnv ritorna il valore di un parametro se questo è definito, altrimenti una stringa nulla.

Il listato 4 propone un esempio piuttosto ghiotto di uso di GetEnv: mostra infatti come si possono usare i comandi del DOS o chiamare altri programmi da un programma scritto in Turbo Pascal (quello che nelle versioni più recenti del Basic si fa con SHELL). Nel numero 68

Listato 2

```
program BSaveBLoad;
const
  MONO = 7;       { se e' installata la scheda monocromatica IBM }
type
  Schermo = array[1..25,1..80] of record        (* per testo *)
    ch: char; attr: byte
  end;
{ Schermo = array[0..16383] of byte;           (* per grafica *) }
var
  S: schermo;
  SPtr: ^Schermo;
  FS: file of Schermo;
  TipoVideo: byte absolute $0000:$0449;
  ModoVideo: byte absolute $0000:$0465;
procedure Prepara;
  begin
    assign(FS,'VIDEO.RAM'); rewrite(FS);
    if TipoVideo = MONO then SPtr := ptr($B000,$0000)
    else SPtr := ptr($B800,$0000)
  end;
procedure BSave(rec: integer);
  begin
    seek(FS,rec);
    if TipoVideo in [0..3] then begin    { modi testo della CGA }
      PortW[$3D8] := $25;
      move(SPtr^,S,sizeof(S));
      PortW[$3D8] := ModoVideo;
      write(FS,S)
    end
    else write(FS,SPtr^)
  end;
procedure BLoad(rec: integer);
  begin
    seek(FS,rec);
    if TipoVideo in [0..3] then begin    { modi testo della CGA }
      read(FS,S);
      PortW[$3D8] := $25;
      move(S,SPtr^,sizeof(S));
      PortW[$3D8] := ModoVideo;
    end
    else read(FS,SPtr^)
  end;
begin
{ GraphMode;                                   (* per grafica *) }
{ Draw(0,0,319,199,1);                                            }
  Prepara;
  BSave(0); clrscr; delay(1000);                (* per testo *)
{ BSave(0); GraphMode; delay(1000);            (* per grafica *) }
  BLoad(0); close(FS); delay(1000);
end.
```

Listato 3

```
function GetEnv(P: stringa): stringa;  { file GETENV.INC }
  type
    Env = array[0..32767] of char;
  var
    EPtr: ^Env;  EStr: stringa;  FineEnv: boolean;
    i,l : Integer;
  begin
    EPtr := ptr(MemW[CSeg:$002C],0);
    i := 0;  FineEnv := FALSE;  EStr := '';
    P := P + '=';  l := length(P);
    repeat
      if EPtr^[i] = #0 then begin
        if EPtr^[i+1] = #0 then FineEnv := TRUE;
        if copy(EStr,1,l) = P then begin
          GetEnv := copy(EStr,l+1,255); Exit
        end;
        EStr := '';
      end
      else
        EStr := EStr + EPtr^[i];
      i := i + 1
    until FineEnv;
    GetEnv := ''
  end;

program GetEnv;  { file GETENV.PAS }
type
  stringa = string[255];
var
  Parametro: stringa;
{$I GETENV.INC}
begin
  write('Parametro (in maiuscolo): ');
  readln(Parametro);
  writeln(GetEnv(Parametro))
end.
```

di MC è stato pubblicato (pag. 229) un programma di un lettore che fa proprio questo, ma lo stesso autore rilevava alcuni malfunzionamenti e chiedeva suggerimenti. Il programma SpawnDemo che vi propongo ora funziona bene: si tratta di software di pubblico dominio elaborato dagli utenti di Compuserve e diffuso dal Technical Support della Borland e dal Turbo User Group a beneficio di chi non abbia accesso a Compuserve.

Noterete che non è proprio brevissimo: per spiegarne tutti i dettagli non basterebbero certo queste poche righe; vi rimando quindi all'*Advanced MSDOS* di Ray Duncan, edito dalla Microsoft Press (cap. 10), o alla rubrica «I trucchi dell'MS-DOS» di Pierluigi Panunzi, che sta trattando con la dovuta gradualità e completezza l'argomento. Qui ci limiteremo ad alcuni punti essenziali, nella speranza di chiarire almeno le differenze con il programma pubblicato sul numero 68.

1) Il comando passato alla funzione Spawn può essere o il path-name completo di un file eseguibile (compresa l'estensione COM, EXE o BAT), o «COMMAND.COM» da solo per richiamare momentaneamente il DOS (dando poi «Exit» per tornare al nostro programma), o «COMMAND.COM/C» seguito dal nome (anche senza estensione) dal file o da un comando «interno» (come DIR). Si può digitare un asterisco al posto di «COMMAND.COM/C» (ad es.: «DIR»); in questo caso viene chiamata GetEnv per determinare il pathname del processore di comandi.

2) Il programma va compilato come file COM assegnando valori quanto più possibile bassi (ad es. 100) alla «mInimum» e alla «mAximum free dynamic memory» nel menu «compiler Options». Un programma compilato con il Turbo Pascal si appropria infatti, se non si fa così, di tutta la memoria disponibile. Non è prudente «liberare» con la funzione 4AH del DOS l'area compresa tra lo heap e lo stack, in quanto si rischia sia di sconvolgere la gestione della memoria del DOS sia di andare a scrivere sullo stack del programma chiamate. Bisogna liberare l'area che si trova DOPO questo stack.

3) Non basta passare alla funzione 4BH (EXEC) l'indirizzo di stringhe che contengano il nome del file o comando e dei suoi argomenti; occorrono anche gli indirizzi dell'environment e di due File controll Block, questi ultimi da approntare usando l'apposita funzione 29H.

4) La funzione EXEC distrugge tutti i registri tranne CS e IP; è quindi necessario salvare in costanti tipizzate (che risiedono appunto nel «Code Segment») i registri SS e SP, per poter loro riassegnare il valore corretto dopo l'EXEC.

Next

La funzione Spawn usa un «inline statement». Non sempre infatti è opportuno usare le procedure Intr e MsDos per accedere agli interrupt del DOS. La prossima volta vedremo con maggiore calma come preparare una routine in linguaggio macchina e come incorporarla in un programma Turbo Pascal. A presto. MC

Listato 4

```
program SpawnDemo;
type
  stringa = string[255];
function Spawn(Comando: stringa): integer;
  const
    regSS: integer = 0; regSP: integer = 0;
  var
    Registri: record case integer Of
      1: (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags: integer);
      2: (AL,AH,BL,BH,CL,CH,DL,DH: byte)
    end;
    FCB1,FCB2: array[0..36] of byte;
    PathName: stringa;
    Argomenti: stringa;
    ParmTable: record
      EnvSeg: integer;   ArgPtr: ^integer;
      FCB1Pr: ^integer;  FCB2Pr: ^integer
    end;
    i,regFlags: integer;
  begin
    if pos(' ',Comando) = 0 then begin
      PathName := Comando + #0; Argomenti := ^M;
    end
    else begin
      PathName := copy(Comando,1,pos(' ',Comando)-1)+#0;
      Argomenti := copy(Comando,pos(' ',Comando),255)+^M;
    end;
    Argomenti[0] := pred(Argomenti[0]);
    with Registri do begin
      FillChar(FCB1,Sizeof(FCB1),0);
      AX := $2901;
      DS := Seg(Argomenti[1]); SI := Ofs(Argomenti[1]);
      ES := Seg(FCB1);         DI := Ofs(FCB1);
      MsDos(Registri); { Crea il primo File Control Block }
      FillChar(FCB2,Sizeof(FCB2),0);
      AX := $2901;
      ES := Seg(FCB2);         DI := Ofs(FCB2);
      MsDos(Registri); { Crea il secondo FCB }
      ES := CSeg;
      BX := SSeg-CSeg+MemW[CSeg:MemW[CSeg:$0101]+$112];
      AH := $4A;
      MsDos(Registri); { Rilascia la memoria non utilizzata }
      with ParmTable do begin
        EnvSeg := MemW[CSeg:$002C]; FCB1Pr := addr(FCB1);
        ArgPtr := addr(Argomenti);  FCB2Pr := addr(FCB2)
      end;
      inline($8D/$96/ PathName /$42/  { DX:=Ofs(PathName[1]) }
             $8D/$9E/ ParmTable /     { BX:=Ofs(ParmTable) }
             $B8/$00/$4B/             { AX:=$4B00 }
             $1E/$55/                 { PUSH DS e BP }
             $16/$1F/                 { DS:=Seg(PathName[1]) }
             $16/$07/                 { ES:=Seg(ParmTable) }
             $2E/$8C/$16/ regSS /     { Salva SS in regSS }
             $2E/$89/$26/ regSP /     { Salva SP in regSP }
             $FA/                     { CLI }
             $CD/$21/                 { INT 21H }
             $FA/                     { CLI }
             $2E/$8B/$26/ regSP /     { Ripristina SP }
             $2E/$8E/$16/ regSS /     { Ripristina SS }
             $FB/                     { STI }
             $5D/$1F/                 { POP BP e DS }
             $9C/$8F/$86/ regFlags /  { regFlags:= Flags }
             $89/$86/ Registri );     { Registri.AX:=AX }
      if (regFlags and 1) <> 0 then Spawn := AX
      else Spawn := 0
    end
  end;
{$I GETENV.INC }
var
  Cmd: stringa;
  i: integer;
begin
  writeln('Esempio di chiamata di comandi DOS e di programmi.');
  writeln('Potete usare "*" al posto di COMMAND.COM /C.');
  writeln('Digitate "*" da solo per finire.');
  repeat
    write('=->'); readLn(Cmd);
    if Cmd = '*' then Halt;
    if Cmd <> '' then begin
      if Cmd[1] = '*' then
        Cmd := GetEnv('COMSPEC')+' /C '+Copy(Cmd,2,255);
      i := Spawn(Cmd);
      if i <> 0 then writeLn('Errore ',i)
    end
  until FALSE { si esce dal programma se Cmd = '*' }
end.
```