Appunti sul PASCAL[1]

"A Strange Site"

http://astrangesite.altervista.org

powered by

dott. Alessandro Strano

FUNZIONI MATEMATICHE

```
ABS(v) rest. il valore assoluto (:v)
ARCTAN(v) rest. l'arcotangente (angoli in radianti) (:real)
COS(v) rest. il coseno (angoli in radianti) (:real)
a DIV b esegue la divisione con troncamento di a/b (:longint)
EXP(v) rest. e^v (:real)
FRAC(v) rest. la rappresentazione real di v (abs(v)<=1)
HI(v) rest./imposta Most Significant Byte di v
INT(v) rest. la rappresentazione real di v (abs(v)>=1)
LN(v) rest. il logaritmo naturale (:real)
LO(v) rest./imposta Least Significant Byte di v
a MOD b rest. il resto della divisione di a/b (:longint)
ODD(v) rest. FALSE se numero è pari (:boolean)
RANDOM rest. un numero casuale tra 0 e 1 (:real)
RANDOMIZE; modifica il generatore di numeri casuali (RANDSEED)
ROUND(v); arrotonda valore da 0.5 in su per eccesso (:integer,longint)
a SHL n rotazione di n bit a sinistra
a SHR n rotazione di n bit a destra
SIN(v) rest. il seno (angoli in radianti) (:real)
SQR(v) rest. il quadrato (:real)
SQRT(v) rest. la radice quadrata (:real)
SWAP(v)  scambia MSB e LSB (v:word,integer)
TRUNC(v) rest. la parte intera di v (:longint)
$hexnum numero in formato esadecimale
a/b divisione di a/b (:real)

costanti: maxint=32767,maxlongint=2147483647,pi=3.1415926536
```

PRIORITA'

operatore	priorità
not	1
and,div,mod,shl,shr,*,/	2
xor,or,+,-	3
<>,<=,>=,=,<,>	4

TIPI DI DATI NUMERICI

tipo	byte	range
byte	1	0;255
shortint	1	-128;127
word	2	0;65535
integer	2	-32768;32767
longint	4	-2147483648;2147483647

[1] **Si fa riferimento al TURBO PASCAL V 4.0 ® Borland International Inc.**

```
real      6
single    4       formato IEEE
double    8       formato IEEE
comp      8       formato IEEE
extended  10      formato IEEE
```

FUNZIONI STRINGA

```
BYTE(car) rest. il codice ascii di car (:byte)
CHR(cod_ascii) rest. il carattere corrispondente a cod_ascii (:char)
CONCAT(s1,..,sn) rest. una stringa formata dalle n stringhe (:string)
COPY(s,inizio,n_car) rest. n_car della stringa a partire da inizio (:string)
DELETE(var s:string;inizio,n_car); cancella n_car da s a partire da inizio
INSERT(sorg,var dest:string;inizio); inserisce sorg in dest a partire da inizio
LENGTH(s) rest. la lunghezza di s (:byte)
POS(ricer,s) rest. la posiz. di ricerc in s (:byte)
STR(valore,var s:string); pone s uguale a valore, allinea a sinistra
UPCASE(car) rest. car in maiuscole (:char)
VAL(s,valore,errpos); rest. in valore il valore di s e in errpos la pos.
   dell'errore
#cod_ascii carattere corrispondente a cod_ascii
^car carattere generato dalla combinazione di ctrl carattere
```

FUNZIONI MEMORIA/PORTE

```
ADDR(var) rest. l'indirizzo della variabile in formato pointer
CSEG rest. il segmento del codice (:word)
DSEG rest. il segmento dei dati (:word)
ERRORADDR rest./imposta l'indirizzo dell'istruz. successiva a quella che ha
   generato l'errore, il segmento è lo scostamento da CSEG (:pointer)
EXITPROC rest./imposta l'indirizzo della routine di gest. errori. Questa routine
   viene eseguita anche quando il programma termina normalmente con EXIT|END.
   Per uscire dalla routine usare HALT o retf, in quest'ultimo caso solo dopo
   aver impostato l'indirizzo di ritorno nello stack e aver ripristinato i
   registri bp e sp. La routine originaria esce con HALT(EXITCODE)  (:pointer)
FILLCHAR(var p^:pointer;n,ordinale); riempie l'area di memoria puntata da p con
   n byte referenziati da ordinale (numerico,char)
FREEMEM(var p:pointer;dim); rilascia un'area di memoria riservata con GETMEM
FREEMIN variabile disponibile  (:word)
FREEPTR imposta/rest. l'indirizzo ultimo segmento disponibile (:pointer)
HEAPERROR imposta/rest. l'indirizzo routine gestione errori heap (:pointer)
GETMEM(var p:pointer;dim); riserva un'area di mem. e ne assegna l'indir. a p
HEAPORG imposta/rest. l'indirizzo in formato pointer dell'heap
HEAPPTR imposta/rest. l'indirizzo disponibile dell'heap (:pointer)
MARK(var p:pointer); restituisce p=ultimo indirizzo occupato dal programma
MAXAVAIL rest. la dim. dell'heap contigua libera (:longint)
MEM[seg:ofs] lettura/scrittura di 1 byte
MEMAVAIL rest. la dim. dell'heap libera (:longint);
MEML[seg:ofs] lettura/scrittura di 4 byte
MEMW[seg:ofs] lettura/scrittura di 2 byte
MOVE(var sorg^,dest^:pointer;n); muove n byte dall'area di mem. puntata da sorg
   all'area puntata da dest
OFS(var) rest. l'ofs della variabile (:word)
PARAMCOUNT rest. il numero di parametri scritti nella linea comando, separati
   dal carattere spazio (:word)
PARAMSTR(n) rest. il parametro ennesimo (:string)
PORT[porta:word] lettura/scrittura di byte dalla porta di comunicazione (:byte)
PREFIXSEG rest. il segmento PSP (:word)
var_pointer:=PTR(seg,ofs) imposta una variabile di tipo pointer, oppure
   MEMW[DSEG:OFS(var_pointer)+2] rest./imposta il segmento
   MEMW[DSEG:OFS(var_pointer)] rest./imposta l'offset
```

RELEASE(var p:pointer); rilascia un'area riservata con GETMEM dall'indirizzo referenziato da p sino alla fine dell'heap
SAVEINT00 imposta/rest. l'ind. originario dell'int 0 (:pointer)
SAVEINT02 imposta/rest. l'ind. originario dell'int 2 (:pointer)
SAVEINT23 imposta/rest. l'ind. originario dell'int 23 (:pointer)
SAVEINT24 imposta/rest. l'ind. originario dell'int 24 (:pointer)
SAVEINT75 imposta/rest. l'ind. originario dell'int 75 (:pointer)
SEG(var) rest. il segmento della variabile (:word)
SIZEOF(var) rest. la lunghezza in byte della variabile (:word)
SPTR rest. l'offset dello stack (:word)
SSEG rest. il segmento dello stack (:word)
@var|procedure|function rest. l'indirizzo della variabile|funzione|procedura in formato pointer

NB:in luogo di var_pointer^ è possibile utilizzare una variabile qualsiasi della quale verrà referenziato l'indirizzo

FUNZIONI ARCHIVI/DISCO

APPEND(var f); apre in append (sola scrittura) l'archivio sempre che esista (f:text)
ASSIGN(var f;path); assegna un pathname ad f(:text,file,file of)
BLOCKREAD(var f:file;var p^:pointer;n:word); legge n blocchi dal file memorizzandoli nell'area di memoria puntata da p
BLOCKWRITE(var f:file;var p^:pointer;n:word); scrive n blocchi sul file leggendoli dall'area di memoria puntata da p
CHDIR(path); cambia il drive e/o la directory corrente
CLOSE(var f); chiude l'archivio. N.B. Esegue la chiamata dell'int 21h anche se il buffer è vuoto allorquando l'archivio è stato aperto in scrittura o in scrittura lettura
EOF{(var f)} rileva l'eof (:boolean)
EOLN{(var f)} rileva l'eoln (:boolean)
ERASE(var f); cancella l'archivio
FILEMODE imposta/rest. il tipo di accesso al file (0=sola lettura o scrittura (a seconda del tipo di apertura); 1=sola scrittura; 2=lettura e scrittura)default=2 (:byte)
FILEPOS(var f) rest. la posiz. sul file in blocchi (f:file,file of)(:longint) [0-]
FILESIZE(var f) rest. la dim. del file in blocchi (f:file,file of)(:longint)
FLUSH(var f:text); trasferisce su file il buffer
GETDIR(num_drive;var path:string); rest. in path la directory corrente sul drive num_drive(:byte 0=drive corrente, 1=a, ecc.)
INPUT var di tipo text assegnata all'input standard ma riassegnabile per riferimenti impliciti per READ|READLN|EOF|EOLN
IORESULT rest. il codice di errore in una operazione su file (solo se I/O check è OFF) (:byte)
MKDIR(path); crea una directory
OUTPUT var di tipo text assegnata all'output standard ma riassegnabile per riferimenti impliciti per WRITE|WRITELN
READ(var f:FILE OF tipo;var t:tipo); lettura da file, la lung. del blocco è data dalla lunghezza di t
READ(var f:text;var1,..,varn); lettura di variabili da file
READLN(var f:text;var1,..,varn); lettura di variabili da file
RENAME(var f;NuovoNome); rinonima l'archivio
RESET(var f;{blocco:word}); apre l'archivio in input (f:text) apre l'archivio secondo la modalità impostata con FILEMODE (f:file of,file) {blocco} lunghezza blocco (standard 128) (f:file)
REWRITE(var f;{blocco:word}); crea ed apre l'archivio in output (f:text) crea ed apre l'archivio in I/O (f:file of,text); {blocco} lunghezza blocco (standard 128) (f:file)
RMDIR(path); cancella una directory

SEEK(var f;n:longint); si posiziona sul blocco n (f:file,file of) [0-]
SEEKEOF{(var f)} ¿rileva l'eof/eoln su file text quando viene letto con read,
SEEKEOLN{(var f)} Ùnella lettura ci si posiziona sul primo carattere dal codice ascii maggiore di 32 (:boolean)
SETTEXTBUF(var f:text;var{,ncar:word}); definisce il buffer per f referenziato da var se ncar non viene specificato si assume la lunghezza del tipo di var
SETTEXTBUF(var f:text;var p^:pointer;ncar:word);definisce il buffer per f refe renziato da p e di dimensione ncar
WRITE(var f:FILE OF tipo;var t:tipo); scrittura su file
WRITE(var f:text;var1,..,varn); scrittura su file senza ritorno a capo
WRITELN(var f:text,var1,..,varn); scrittura su file con ritorno a capo

VARIE

AND,OR,NOT,XOR operatori logici
DISPOSE(var punt1,..,puntn); libera lo spazio occupato da uno o più puntatori
DEC(var ord);rest. ord:=predecessore di ord [word,byte,integer,longint,shortint, char,boolean]
EXITCODE rest./imposta il codice di ritorno (:byte)
INC(var ord); rest. ord:=successore di ord
INLINE(cod_op); inserisce nel .EXE l'istruz. assembler che corrisponde a cod_op(:word scambia msb con lsb, oppure :byte)
NEW(var punt); crea una nuova variab. referenziata e ne assegna l'indir. a punt
NIL costante indicante l'ultimo elemento di una lista
ORD(ord) rest. l'ordinale associato a ord
PRED(ord) rest. il predecessore di ord
SUCC(ord) rest. il successore di ord
costanti booleane: TRUE=1,FALSE=0

PROPOSIZIONI

AND,OR,NOT,XOR operatori logici [if (a=1) AND (b='a') ..]
BEGIN;..;END; definisce un blocco di proposizioni
CASE var1 OF lista1:proposizione1;..;listan:proposizionen;{else}...;END; esegue una delle proposizioni a seconda dell'appartenenza di var1 alle liste
CONST const1=valore,..,constn=valore; definizione di costanti
EXIT; uscita da programma/procedura/funzione
FOR var:=inival TO|DOWNTO finval DO proposizione; esegue proposizione per valori di var da inival a finval
FUNCTION nome {({var} v1,v2:tipo;..;)}:tipo;{TYPE;}{VAR.;}{..;}BEGIN;..;END; definisce una funzione per richiamarla: nome{(v1,..,v2)}) se si specifica var vengono passati gli indirizzi e non i valori
GOTO etichetta; salto alla istruz. etichetta: proposizione
HALT{(errorlevel)}; esce dal programma senza eseguire la routine di gestione errori ed imposta errorlevel (codice di ritorno) se questo è stato specificato
IF var= dato|IN [dato1,..,daton] THEN proposizione {ELSE}; esegue proposizione se la condizione è vera. N.B. Se dopo then vi è un blocco di proposizioni la end che le delimita non deve essere seguita da <;> se dopo vi è un else;
LABEL etic1,..,eticn; definisce le etichette utilizzabili con GOTO
PROCEDURE nome {({var} v1,v2:tipo;..;)};{TYPE.;}{VAR.;}{..;}BEGIN;..;END; definisce una procedura per richiamarla: nome{(v1,..,v2)}), se si specifica var vengono passati gli indirizzi e non i valori
PROCEDURE|FUNCTION nome..;FORWARD; la definiz. della procedura/funzione può essere fatta dopo con PROCEDURE|FUNCTION nome,{..};BEGIN;..;END
PROCEDURE|FUNCTION nome..;EXTERNAL; procedura/funzione esterna (.OBJ)
PROGRAM nome {(input,output)};{USES.;}{TYPE.;}{CONST.;}{LABEL.;}{PROCEDURE.;} { FUNCTION.;}BEGIN;..;END. definisce un programma (input e output var text assegnati all'i/o standard ma riassegnabili per riferimenti impliciti per read, write ecc.
READ{(var1,..varn)}; lettura di variabili

```
READLN{(var1,..varn)}; lettura di variabili
REPEAT;..;UNTIL condizione; esegue il ciclo per condizione falsa
TYPE nome=tipo,..,nome=tipo; definisce un tipo di dato
USES interfaccia,..,interfaccia; specifica le interfacce da usare
VAR var1,..,varn:tipo;var1,..;varn:tipo; definisce le variabili da utilizzare
WRITE{(var1,..,'prompt')}; scrittura senza ritorno a capo
WRITELN{(var1,..,'prompt')}; scrittura con ritorno a capo
WRITE|WRITELN(var1:n_car{:n_cifredecimali}); scrittura formattata, se il
   parametro da scrivere è più corto di n_car vengono inseriti degli spazi vuoti
   a sinistra, se è più lungo n_car viene ignorato
var_record.campo (1) riferimento al campo di var_record(:record)
WHILE condizione DO proposizione; esegue il ciclo per condizione vera
WITH var_record DO proposizione; esegue proposizione permettendo il riferimento
   diretto al campo senza ricorrere alla (1)
interfaccia.funzione|procedura|costante; indica che ci si riferisce a una data
   interfaccia; se vi sono più unità aventi nomi di cost|procedure|funzioni
   uguali si assumono implicitamente quelle dell'unità caricata per ultima
var1:=numero|var|'stringa'; assegnazione
   l'assegnaz. è possibile tra variabili dello stesso tipo (eccetto
   text,file,file of) ed inoltre nei seguenti casi:
   var_real:=tipo_numerico;
   var_string:=tipo_char;
   var_numerica:=tipo_numerico_non_real;
   array_char:=constante_stessa_lunghezza;
   var_string:=array_char;
   Per i tipi ARRAY,RECORD solo se le variabili sono state definite sulla stessa
   riga o se si è utilizzata l'istruzione type
(*commento*)|{commento}; commento
```

TIPI DI DATI

```
Numerici: byte,shortint,integer,word,longint,real,single,double,comp,extended
Vettori: {PACKED} ARRAY [n..n1{n,..n1}] OF tipo_di_dato
Alfanumerici: char,string,string[n]
Archivi: text,file,file of
Vari: boolean,pointer,set of,case,record
```

LISTE - variabile referenziata

```
TYPE p=^lista;
     lista=RECORD uno:CHAR; {campo informazione}
                  due:p;    {campo puntatore}
           END;
VAR punt,prec:p;
per il riferimento punt^.due=prec; {punta all'elemento precedente, per il primo
prec:=NIL per i successivi NEW(punt); prec:=punt;}
punt^.uno:=informazione;
```

SET OF - tipo scalare

```
TYPE colore=(rosso,verde); {tipo scalare}
     misto=SET OF colore; {valori possibili [],[rosso],[verde],[rosso,verde]}
     codice=SET OF 1..2; {valori possibili [],[1],[2],[1,2]}
     xcod=SET OF 'a'..'b';

Siano A e B operandi di tipo set;
      A + B produce l'insieme di tutti i valori che sono in A o B
      A * B produce l'insieme dei valori che sono contemporaneamente in A e B
      A - B produce l'insieme dei valori che sono in A ma non in B
      gli operatori <= >= verificano l'inclusione: A <= B [true se A è contenuto
      in B] A >= B [true se B è contenuto in A]
```

FILE OF - RECORD - CASE

```
TYPE statocivile=(sposato,divorziato,vedovo,nonsposato);
     dati=RECORD
          nome:string;
          cognome:string;
          CASE st:statocivile of
               sposato: (coniuge:string; figli:integer);
               divorziato,vedovo: (figli:integer);
               nonsposato: ();
          END;

VAR archivio:FILE OF dati; {nell'archivio i dati numerici vengono scritti in
forma impaccata e al tipo scalare viene assegnato l'ordinale corrispondente}
```

PACKED

Una stringa di n caratteri è considerata una costante di tipo PACKED ARRAY[1..n] OF CHAR;

DISPONIBILI CON USES CRT

ASSIGNCRT(var f:text); assegna il video alla variabile f
CHECKBREAK abilita(:=TRUE) o disabilita (:=FALSE) la funzione di CTRL BREAK
CHECKEOF abilita(:=TRUE) o disabilita (:=FALSE) la rilevaz. di CTRL Z
CHECKSNOW abilita(:=TRUE) o disabilita (:=FALSE) l'elim. dell'effetto neve
CLREOL; cancella dalla pos. del cursore a fine riga (non modifica la posiz.)
CLRSCR; cancella lo schermo e posiz. il cursore il alto a sinistra
DELAY(ritardo:word); ciclo di ritardo
DELLINE; cancella la riga corrente (non modifica la pos. del cursore)
DIRECTVIDEO abilita(:=TRUE) o disabilita (:=FALSE) il video primario
GOTOXY(colonna,riga:byte); posiziona il cursore
HIGHVIDEO; luminosità… intensa
INSLINE; inserisce una riga vuota (non modifica la pos. del cursore)
KEYPRESSED rest. TRUE se vi sono caratteri nel buffer di tastiera.
LASTMODE rest./imposta la modalità attuale (:word)
LOWVIDEO; luminosità bassa
NORMVIDEO; luminosità normale
NOSOUND; blocca la generazione del suono
READKEY lettura di un carattere (anche tasti funzione e controllo) senza
   visualizzazione e con attesa dell'input (:char)
SAVEINT1B rest./imposta l'ind. di ritorno dell'int. 1B (:pointer)
SOUND(freq:word); genera in continuo il suono specificato
TEXTATTR rest./imposta il colore dei caratteri (:byte)
TEXTBACKGROUND(colore:byte); imposta il colore per lo sfondo
TEXTCOLOR(colore:byte); imposta il colore per i caratteri
TEXTMODE(modo:word); imposta la modalità testo desiderata
WINDOW(col1,rig1,col2,rig2:byte); imposta il porto di testo
WINDMAX rest. in LO(windmax) n colonne-1 e in HI(windmax) n righe-1 (:word)
WINDMIN rest. in LO(windmin) n min.col-1 e in HI(windmin) n min.righe-1 (:word)
WHEREX rest. la colonna corrente (:byte)
WHEREY rest. la riga corrente (:byte)
Costanti colori: black, brown, blue, green, cyan, red, lightgray, magenta,
lightgreen, lightcyan, lightred, lightmagenta, yellow, white, blink, darkgray
costanti modo: bw40, bw80, co40, c40, c80, mono,font8x8

DISPONIBILI CON USES PRINTER

var LST di tipo text assegnata ed aperta (LPT1)

DISPONIBILI CON USES DOS

```
DISKFREE(n_drive:byte); rest. lo spazio disponibile su disco (n_drive
   0=corrente, 1=a, ecc.) (:longint)
DISKSIZE(n_drive:byte); rest. la dim. del disco (n_drive 0=corrente, 1=a, ecc.)
   (:longint)
DOSERROR rest. il [valore del registro AX] codice di errore DOS; usarla con le
   operazioni di I/O su disco di questa sezione
DOSEXITCODE rest. il codice di ritorno (usare con EXEC) (:word)
EXEC(path_comando,cmdline:string); esegue il programma
FINDFISTR(path:string;attr:word;var a:searchrec); rest. il primo descrittore
   file
FINDNEXT(var a:searchrec); rest. il successivo descrittore file
GETINTVEC(n_int:byte;var p:pointer); rest. l'ind. dell'interrupt
GETDATE(var anno,mese,giorno,giornosettimana:word); lettura data. Giornosettima-
 na=0 se domenica, 1 se lunedì, ecc.
GETFATTR(var f;var attr:word); rest. in attr l'attributo del file
GETFTIME(var f;var time:longint); rest. in time la data del file
GETTIME(var ore,minuti,secondi,cent:word); lettura orario
INTR(n_int:byte;var registri:registers); chiamata di interrupt
KEEP(n_blocchi_da_16_byte:word); rende il programma residente
MSDOS(var registri:registers); chiamata dell'int. 21h
PACKTIME(var time:datetime;var packtime:longint); rest. in packtime la data
   time in formato impaccato
SETDATE(anno,mese,giorno:word); imposta la data
SETINTVEC(n_int:byte;var p:pointer); imposta l'ind. dell'interrupt
SETFATTR(var f;attr:byte); imposta l'attributo del file che non deve essere
   aperto
SETFTIME(var f;data:longint); imposta la data del file che deve essere stato
 aperto con RESET se file TEXT, con RESET o REWRITE per FILE, FILE OF
SETTIME(ore,minuti,secondi:word); imposta l'orario
UNPACKTIME(var packtime:longint;var time:datetime); rest. in time la data in
   formato datetime
tipi di dati:
SEARCHREC=record
 FILL:21 byte;
 ATTR:byte;
 TIME,SIZE:longint;
 NAME:string[13];
 end;
DATETIME=record
 YEAR,MONTH,DAY,HOUR,MIN,SEC:word;
 end;
FILEREC=record
 HANDLE,MODE,RECSIZE:word;
 PRIVATE:26 byte;
 USERDATA:16 byte;
 NAME:string[80];
 end;
TEXTREC=record
 MODE,HANDLE,BUFSIZE,PRIVATE,BUFPOS,BUFEND:word;
 BUFFER:array [0..127] of char;
 BUFPTR,OPENFUNC,INOUTFUNC,FLUSHFUNC,CLOSEFUNC:pointer;
 OPENFUNC,INOUTFUNC,CLOSEFUNC:pointer;
 USERDATA:16 byte;
 NAME:string[80];
 BUFFER:string[128];
 end;
REGISTERS=record
 AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS:word;
```

```
 AL,AH,BL,BH,CL,CH,DL,DH:byte;
 end;
TEXTBUF:string[128];
costanti: fcarry=1, fparity=4, fauxiliary=16, fzero=64, fsign=128,
   foverflow=2048, fmclosed=55216, fminput=55217, fmoutput=55218, fminout=55219
costanti attributi: readonly=1, sysfile=4, volumeid=8, directory=16, archive=32,
   hidden=2,anyfile=63
```

ESEMPIO: CREAZIONE DI UNITA' DA UTILIZZARE CON USES

```
UNIT nomeunità;
INTERFACE
VAR..;    (* variabili, costanti, funzioni e procedure referenziabili *)
CONST..;  (* dal programma *)
PROCEDURE nome(var:tipo);
FUNCTION nome:tipo;
..
IMPLEMENTATION
PROCEDURE nome;BEGIN;..;END;
FUNCTION nome;BEGIN;..;END;
[BEGIN]
..
eventuali istruzioni che verranno eseguite la prima volta che nel programma ci
si riferirà alla UNIT
..
END.
```

ESEMPIO: GESTIONE ERRORI

```
USES CRT;
VAR ERR: BOOLEAN;
    I, R: REAL;
PROCEDURE ERRROUTINE;
BEGIN;
      IF EXITCODE > 0 THEN BEGIN;
            EXITCODE := 0;
            EXITPROC := @ERRROUTINE;
            ERR:=TRUE;
            MEMW[SSEG:SPTR + 4] := MEMW[DSEG:OFS(ERRORADDR)];(1)
            MEMW[SSEG:SPTR + 6] := CSEG;                     (1)
            INLINE($EC89); (* MOV SP,BP *)   (2)
            INLINE($5D);   (* POP BP *)      (2)
            INLINE($CD);   (* RETF *)        (2)
      END;
      HALT; (* termina se la procedure non è stata richiamata da un errore *)
END;
BEGIN;
      CLRSCR;
      EXITPROC := @ERROROUTINE; (* imposta l'indirizzo della routine *)
      I:=1;                     (* di gestione degli errori *)
      REPEAT;
            WRITE('I=',I,' LN(I)=');
            R:=LN(I); (* istruzione che genera l'errore *)
            IF ERR THEN BEGIN; (* istruzione puntata da erroraddr *)
                  ERR := FALSE;
                  WRITELN('Non esiste');
            END ELSE WRITELN(R);
            INC(I);
      UNTIL I > 5;
```

```
END. (* richiama erroroutine e termina *)
```

(1) impostazione dell'indirizzo di ritorno al valore referenziato da erroraddr.
(2) istruzioni in assembler inserite per il corretto funzionamento della routine.