di Sergio Polini

Gestione delle eccezioni in Turbo Pascal 5.x

La volta scorsa abbiamo visto come, fin dalle prime versioni del Turbo Pascal, le routine di controllo degli errori di esecuzione sono tutt'uno con quelle che presiedono anche alle situazioni normali di fine programma. Siamo riusciti ad intervenire su tale meccanismo per ottenere che, una volta verificatasi l'«eccezione», sia possibile provocare un ritorno al punto che l'ha provocata piuttosto che al DOS. Abbiamo però dovuto rassegnarci ad una limitata portabilità della nostra soluzione (che potrebbe non funzionare su versioni anche leggermente diverse del compilatore), e fare i conti con un trattamento degli errori poco organico, con errori di esecuzione da una parte e errori di I/O dall'altra. Problemi in buona parte risolti nelle versioni successive

A partire dalla versione 4.0, gli errori di esecuzione vengono classificati in quattro categorie: errori DOS, errori I/O, errori critici e errori fatali. Il cambiamento non è solo terminologico. In primo luogo, infatti, fino alla versione 3 non c'era alcun controllo degli errori critici; al verificarsi di un errore, inoltre, ne veniva posto il codice numerico nel registro DL, mentre a DH veniva assegnato un valore destinato proprio a distinguere gli errori di esecuzione da quelli di I/O, considerati «altra razza». Ora le cose vanno diversamente: prima di passare il controllo alle routine che sovraintendono alla fine del programma (normale o anomala che sia), in AX viene posto il solo codice dell'errore, senza alcuna distinzione tra le quattro diverse categorie. Ciò ha contribuito a rendere possibile un trattamento più lineare di tali situazioni, al punto che risulta anche più facile adottare soluzioni valide per versioni diverse del compilatore. Il codice che vi sto per proporre funziona sia con la 5.0 che con la 5.5, ma potrebbe essere facilmente adattato anche alla 4.0; le modifiche riguarderebbero solo aspetti della sintassi del linguaggio, quali i parametri procedurali.

```
unit GestEcc;

interface

uses Dos;

type
  BoolFunc = function: boolean;        (* tipo del "gestore" *)

var
  ErroreRunTime: word;                 (* da usare solo nel "gestore" *)

procedure InstallaGestoreEccezioni(Func: BoolFunc);
procedure DisinstallaGestoreEccezioni;
function ErroreRT: word;
procedure Raise(Eccezione: word);

implementation

(*$L GESTECC.OBJ*)

var
  PrevInt0: pointer;
  Gestore: BoolFunc;

procedure NuovoInt0(Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP: word);
(* Codifica di DIV  = 1^ byte: 1111 011w; 2^ byte: mm11 0rrr; *)
(* codifica di IDIV = 1^ byte: 1111 011w; 2^ byte: mm11 1rrr. *)
interrupt;
var
  p: ^byte;          (* puntatore al secondo byte di DIV o IDIV *)
begin
  p := Ptr(CS, IP+1);
  case p^ shr 4 of  (* p^ shr 4 = primi quattro bit del secondo byte *)
    $3: if (p^ = $36) or (p^ = $3E) then Inc(IP, 4)
        else Inc(IP, 2);
    $7: Inc(IP, 3);
    $B: Inc(IP, 4);
    $F: Inc(IP, 2);
  end;
  ErroreRunTime := 199;
end;
```

```
procedure ErrRTime; external;

(*$F+*)
function HeapFunc(Size: word): integer;
begin
  ErroreRunTime := 203;
  HeapFunc := 1;
end;

function NoGest: boolean;
begin
end;
(*$F-*)

procedure InstallaGestoreEccezioni(Func: BoolFunc);
const
  PrimaVolta: boolean = TRUE;
  VecchioCodice: pointer = nil;
var
  bp: ^byte;
  wp: ^word;
  pp: ^pointer;
begin
  bp := Ptr(Seg(PrevInt0^), Ofs(PrevInt0^)+14);
  pp := Ptr(Seg(PrevInt0^), Ofs(PrevInt0^)+15);
  if PrimaVolta then begin
    PrimaVolta := FALSE;
    wp := Ptr(Seg(PrevInt0^), Ofs(PrevInt0^)+17);
    if (bp^ <> $BA) or (wp^ <> $DA8E) then begin
      (* se non c'e' MOV DX, SEG DATA / MOV DS, DX *)
      Writeln('Impossibile installare gestore eccezioni.');
      Halt(1);
    end;
    VecchioCodice := pp^;
  end;
  Gestore := Func;
  if Addr(Gestore) <> Addr(NoGest) then begin
    SetIntVec(0, Addr(NuovoInt0));
    bp^ := $9A;
    pp^ := Addr(ErrRTime);
  end
  else begin
    SetIntVec(0, PrevInt0);
    bp^ := $BA;
    pp^ := VecchioCodice;
  end;
end;

procedure DisinstallaGestoreEccezioni;
begin
  InstallaGestoreEccezioni(NoGest);
end;

procedure Raise(Eccezione: word);
const
  NonGestibili: set of byte = [0,1,6,199,201,202,203,204,207,208,209];
begin
  if (Eccezione > 255) or not (byte(Eccezione) in NonGestibili) then
    RunError(Eccezione);
end;

function ErroreRT: word;
var
  ERT: word;
begin
  ERT := IOResult;        (* per azzerare l'eventuale errore di I/O *)
  if ERT = 0 then
    ERT := ErroreRunTime;
  ErroreRunTime := 0;
  ErroreRT := ERT;
end;

begin
  GetIntVec(0, PrevInt0);
  HeapError := @HeapFunc;
  ErroreRunTime := 0;
end.
```

Figura 1 - Il listato di GESTECC.PAS, unit per la gestione di eccezioni in Turbo Pascal 5.0 e 5.5.

Installazione del gestore

Il meccanismo generale ricorda molto quello visto la volta scorsa, anche se può essere ora meglio organizzato in una unit (vedi figura 1). La interface ci propone un tipo BoolFunc (funzione che ritorna un boolean), una variabile ErroreRunTime, le procedure InstallaGestoreEccezioni, DisinstallaGestoreEccezioni e Raise, ed una funzione ErroreRT. La procedura InstallaGestoreEccezioni accetta come parametro una funzione di tipo BoolFunc, che chiameremo d'ora in avanti «il gestore», a cui verrà passato il controllo al verificarsi di un'eccezione «gestibile» (vedremo tra breve cosa questo voglia dire). Il meccanismo è analogo a quello usato la volta scorsa: si interviene sul codice della routine di libreria che presiede alle terminazioni, normali o anomale, di qualsiasi programma Turbo Pascal, per sostituirne i primi cinque byte con una CALL ad una procedura ErrRTime. Questa, dopo aver riprodotto le istruzioni contenute nei cinque byte sostituiti e dopo i controlli del caso, chiamerà a sua volta il gestore; se questo ritornerà TRUE l'esecuzione del programma proseguirà normalmente (ma dopo aver assegnato un opportuno valore alla variabile ErroreRunTime), altrimenti si tornerà alla routine di libreria, con conseguente interruzione del programma.

Abbiamo anche ora il problema di cercare il punto esatto in cui andare a inserire l'istruzione «CALL ErRRTime». Ci è però qui di grande aiuto la disponibilità dei sorgenti della libreria del compilatore (Turbo Pascal 5.5 Runtime Library, L. 249.000+IVA), che non solo svela i nomi simbolici e il significato di quelle sequenze di byte che possiamo spiare con un debugger, ma consente anche di verificare che non vi sono differenze tra l'una e l'altra delle ultime versioni del Turbo Pascal (almeno non per quello che ora ci interessa). Come potete vedere nella figura 2, è facile riscontrare che l'inizio della routine Terminate è posto esattamente 14 byte dopo Int00Handler, che altro non è che l'indirizzo che il Turbo Pascal attribuisce all'INT 0. Il codice di inizializzazione della nostra unit GESTECC salva quindi subito quest'indirizzo nella variabile PrevInt0, per consentire a InstallaGestoreEccezioni di verificare, la prima volta che viene eseguita, che 14 byte dopo vi siano proprio quei MOV DX,SEG DATA e MOV DS,DX. Solo dopo tale verifica queste istruzioni vengono sostituite da CALL ErrRTime e l'indirizzo del gestore viene salvato nella variabile Gestore.

```
; Divide by zero interrupt handler. Control arrives here upon
; executing a DIV or IDIV instruction with a zero divisor.

Int00Handler:

        MOV     AX,200

; RunError standard procedure

HaltError:

        POP     CX
        POP     BX
        JMP     SHORT Terminate

; Control-C interrupt handler. Control arrives here when DOS
; detects a Ctrl-C or Ctrl-Break.

Int23Handler:

        MOV     AX,255

; Halt standard procedure

HaltTurbo:

        XOR     CX,CX
        XOR     BX,BX

; Terminate program and return to DOS
; In    AX    = Exit code
;       BX:CX = Error address (or NIL)

Terminate:

        MOV     DX,SEG DATA             ;Reset DS
        MOV     DS,DX
        ecc.
```

Figura 2 - Il codice che sovraintende alla fine di un programma (normale o anomala che sia), come illustrato nella Turbo Pascal 5.5 Runtime Library. L'inizio della routine Terminate si trova 14 byte dopo Int00Handler.

Diciamo subito che questo meccanismo consente di installare come gestore in ogni momento la funzione boolean che si preferisce (una che ritorni sempre TRUE, una che ritorni TRUE solo in alcuni casi, una che non faccia altro, una che proponga all'utente di scegliere il da farsi, ecc.), ma anche di tornare al funzionamento normale, ovvero alla interruzione del programma con il messaggio «Runtime error XXX at XXXX:XXXX»: ciò si ottiene semplicemente rimettendo al loro posto le istruzioni originarie. Il mese scorso ottenevamo tale risultato passando uno zero alla procedura di installazione, ma ora non è più possibile, in quanto occorre passare il nome di una funzione; per liberare il programmatore dalla necessità di dichiararne una, la interface della unit propone la procedura DisinstallaGestoreEccezioni, che altro non fa che chiamare InstallaGestoreEccezioni con il nome di una funzione NoGest dichiarata nella implementation.

Può sembrare un po' macchinoso, ma in realtà si tratta solo di approfittare della disponibilità dei parametri procedurali, dei quali il Turbo Pascal 3 era completamente ignaro: avevamo dovuto simularli ricorrendo a qualche trucco. L'uso di tali parametri, ora che è possibile, non è solo più elegante, ma anche più sicuro: il compilatore si farà infatti carico di ricordarci, ad esempio, che i «gestori» dovranno essere dichiarati nel modo previsto (funzione senza argomenti e con risultato di tipo boolean) e compilati con la direttiva $F+.

Intercettazione dell'INT 0

La locazione Int00Handler va cercata non solo per localizzare il punto in cui intervenire come detto, ma anche per sostituire una nostra routine a quella associata dal compilatore all'INT 0. Guardate la figura 2: le istruzioni POP CX e POP BX mettono in questi due registri il segmento e l'offset della istruzione successiva a quella che ha provocato l'errore, e possono farlo perché trovano nello stack quanto vi ha posto la CALL alla routine in cui quella istruzione si trova. Nel caso di divisione intera per zero, invece, scatta un INT 0 che pone nello stack l'indirizzo della istruzione DIV o IDIV che l'ha provocato e il valore dei flag; ne segue che, se si facesse conto su BX:CX, si ritornerebbe sempre alla stessa istruzione, ma ogni volta lasciando nello stack una word con i valori dei flag, per finire con un errore di stack overflow. Essendo uguali i codici d'errore per la divisione intera e quella reale per zero, non sarebbe possibile aggiustare le cose nell'ambito della procedura ErrRTime, che non avrebbe modo di distinguere tra i due casi. Ecco perché dobbiamo avvalerci di una procedura NuovoInt0 (anch'essa nella figura 1).

Si tratta forse del codice più «oscuro» della unit. L'obiettivo è quello di ottenere un ritorno alla istruzione succesiva alla DIV o IDIV, ma, a differenza di quanto abbiamo visto il mese scorso, ora si devono fare i conti con la maggiore efficienza del codice prodotto dal compilatore. Il Turbo Pascal 3 usava sempre una IDIV CX, ma ora si possono avere sia DIV che IDIV, e soprattutto il divisore non è sempre un registro, ma può anche essere un indirizzo di memoria. Anche questa volta si tratta di andare ad aggiungere un opportuno valore al registro IP come salvato nello stack, questo valore però non è più sempre 2, ma tanti byte (2, 3, o 4) quanti è lunga l'istruzione. Per scoprirlo occorre andare ad esaminare i singoli bit del secondo byte della codifica dell'istruzione. Un po' macchinoso, ma funziona.

NuovoInt0 assegna a ErroreRunTime il valore 199, diverso da quello previsto dal compilatore per la divisione per zero, al fine di consentire di distinguere tra l'operazione su interi e quella su numeri in virgola mobile. Viene per il resto lasciato indefinito (come per tutti gli altri casi) il risultato dell'operazione.

Va rilevato che, così come è ora la unit, si può installare il gestore che più piace ma l'intercettazione dell'INT 0 sarà sempre la stessa; chi lo volesse, tuttavia, potrebbe facilmente modificare l'intestazione della procedura InstallaGestoreEccezioni aggiungendo un secondo parametro.

La procedura ErrRTime

Una volta installato il gestore, ogni eccezione provoca la chiamata della procedura ErrRTime. Questa (figura 3) per prima cosa riproduce le istruzioni sostituite dalla procedura di installazione e assegna ad ErroreRunTime il codice dell'errore; provvede quindi ad esaminare il valore di questo per rifiutare la «gestione» di situazioni «non gestibili». Vediamole una per una, anche per motivare una «non gestibilità» che discende solo da mie considerazioni di opportunità.

Gli errori DOS sono quasi tutti analoghi agli errori di I/O, al punto che possono come questi essere normalmente

```
        .MODEL TPASCAL
        .DATA
EXTRN   ErroreRunTime : WORD
EXTRN   Gestore : DWORD
SegErr  DW      ?               ; Indirizzo della istruzione successiva
OfsErr  DW      ?               ; a quella che ha causato l'errore.

        .CODE

PUBLIC ErrRTime

ErrRTime PROC FAR
        push    ds              ; Salva DS.
        mov     dx, SEG DATA    ; Istruzioni sostituite da CALL ErrRTime
        mov     ds, dx          ; (ripristinano il data segment).
        mov     ErroreRunTime, ax ; Codice d'errore in ErroreRunTime.
        cmp     ax, 1           ; Salta gli errori non gestibili:
        jbe     NonGestibili    ; 0-1
        cmp     ax, 6
        je      NonGestibili    ; 6
        cmp     ax, 199
        je      NonGestibili    ; 199
        cmp     ax, 201
        jb      Gestibili
        cmp     ax, 204
        jbe     NonGestibili    ; 201-204
        cmp     ax, 207
        jb      Gestibili
        cmp     ax, 209
        jbe     NonGestibili    ; 207-209
Gestibili:
        mov     SegErr, bx      ; In SegErr e OfsErr l'indirizzo della
        mov     OfsErr, cx      ; istruzione SUCCESSIVA a quella che
        mov     ax, bx          ; ha causato l'errore.
        or      ax, cx          ; Se BX e CX sono nulli, non e' errore.
        jz      NonGestibili
        call    [Gestore]       ; Chiamata del gestore installato.
        or      ax, ax          ; Se ritorna FALSE,
        jz      NonGestibili
        pop     dx              ; Originario DS in DX
        add     sp, 4           ; Toglie dallo stack l'indirizzo cui
        mov     bx, SegErr      ; tornare dopo il RET, per mettervi
        mov     cx, OfsErr      ; quello dell'istruzione successiva a
        push    bx              ; quella che ha causato l'errore.
        push    cx
        mov     ds, dx          ; Ripristina DS
        ret                     ; Normale prosecuzione del programma.
NonGestibili:
        mov     ax, ErroreRunTime ; Rimetti il codice d'errore in AX.
        pop     ds              ; Ripristina DS.
        ret                     ; Torna alla routine di fine-programma.
ErrRTime ENDP
        END
```

Figura 3 - Il file GESTECC.ASM, contenente la procedura ErrRTime.

trattati con la direttiva $I e la funzione predefinita IOResult. La unit GESTECC, approfittando di tali parentele (anche gli errori critici possono essere trattati in questo modo), tratta tutte le eccezioni allo stesso modo, al punto che la funzione ErroreRT va considerata come una estensione di IOResult alla quale virtualmente si sostituisce, senza bisogno di agire sulla direttiva $I. Meglio: la direttiva dovrebbe essere sempre attivata, proprio per consentire la generazione dell'errore e quindi la chiamata del gestore (così si fa negli esempi che vedremo). Se però risultasse chiamata una funzione del DOS che non esiste (errore 1), non potrebbe trattarsi che o di un bug bello e buono o di una misteriosa corruzione del codice: ben altro che, ad esempio, un tentativo di aprire un file che non esiste. Analogo il caso di una chiamata del DOS con un numero di file (handle) non valido.

L'errore 199 viene escluso in quanto riservato alla divisione intera per zero, trattata a parte; analogamente viene esclusa la situazione di spazio insufficiente nello heap (errore 203), in quanto, come consiglia il manuale e come si cura di fare il codice di inizializzazione della unit, viene assegnato alla variabile predefinita HeapError l'indirizzo di una funzione HeapFunc che si limita ad assegnare 203 a ErroreRunTime e a tornare 1 (uno): in tal modo, una chiamata a New o GetMem in situazioni di memoria insufficiente ci ritorna un puntatore nullo invece che far terminare il programma.

Le situazioni di stack overflow (errore 202) non consentono in nessun caso la normale prosecuzione del programma, mentre l'errore di range (codice 201, possibile solo se viene attivata la direttiva $R) viene escluso in quanto non generato solo dall'uso di un indice improprio per l'accesso ad un array o simili, ma, nel Turbo Pascal 5.5, anche per bloccare programmi in cui si chiamino metodi virtuali di istanze di oggetti non inizializzate mediante la chiamata del relativo constructor.

Le operazioni non valide sui puntatori (errore 204) non vengono intercettate in quanto sarebbe ben arduo continuare ad usare puntatori «impazziti», e tali eventualità vanno quindi obbligatoriamente escluse mediante un accurato debugging. Analogamente, non mi sembra possibile gestire gli errori 208 e 209 (overlay manager non installato e errore di lettura del file di overlay). Siete naturalmente liberi di operare scelte diverse dalle mie; ricordate solo che, se cambiate la procedura ErrRTime, dovreste apportare coerenti variazioni anche alla costante NonGestibili nella procedura Raise (figura 1).

L'errore 207 merita un'attenzione particolare: si verifica quando l'argomento di un Trunc o Round non «entra» in un longint, quando si tenta la radice quadrata di un numero negativo o il logaritmo di zero o di un numero negativo, in caso di overflow nello stack di un eventuale coprocessore numerico (di quest'ultimo parleremo la prossima volta). Potreste pensare che si tratta di casi analoghi alla divisione per zero, e sarei d'accordo con voi. Il problema è che c'è un bug nella routine di libreria che calcola la radice quadrata, Sqrt. Come potete verificare (figura 4), la routine decrementa il registro SP per far posto ad alcune variabili locali, e quasi subito dopo verifica che non le sia stato passato un argomento negativo; in tal caso salta ad un POP BP che, da solo, non ha alcun senso: dovrebbe essere preceduto da un MOV SP,BP con il quale si libererebbe lo stack di quelle variabili locali (proprio come avviene nel caso di ritorno normale, con la macro EXIT). Poiché così non è, non succede nulla nel caso di errore «non gestito» (il programma termina con un errore e basta), ma un gestore non riuscirebbe a trovare nello stack il corretto indirizzo da cui far proseguire l'esecuzione.

Sottoporrò quanto prima il problema alla Borland; intanto non resta altro che rinunciare ad intercettare l'errore 207 (o magari intercettarlo ma solo in programmi in cui Sqrt non ci sia o i suoi argomenti vengano sempre scrupolosamente controllati; fate voi).

```
; Sqrt standard function

RSqrt:

        LOC     Expo,BYTE,2     ; Le tre macro producono:
        LOC     Temp,BYTE,6     ;       PUSH    BP
                                ;       MOV     BP,SP
        ENTRY   FAR             ;       SUB     SP,8
        MOV     CX,AX
        MOV     SI,BX
        MOV     DI,DX
        OR      AL,AL
        JZ      @@2
        TEST    DH,80H          ; Se argomento negativo
        JNZ     @@3             ; salta a @@3
        ...
        ...
        EXIT                    ; Produce: MOV SP,BP / POP BP / RETF
@@3:    POP     BP
        MOV     AX,207
        JMP     HaltError
```

Figura 4 - L'inizio e la fine della routine di libreria che calcola la radice quadrata, tratti dalla Turbo Pascal Runtime Library.

```
Program ErrIO;
(*$I+*)
uses GestEcc;
var
  f: text;
(*$F+*)
function Gestore: boolean;
var
  Errore: word;
begin
  Errore := ErroreRT; (* si usa ErroreRT per azzerare InOutRes *)
  Gestore := TRUE;
  case Errore of
    1..17   : Writeln('Errore DOS ', Errore);
    100..106: Writeln('Errore I/O ', Errore);
    150..162: Writeln('Errore critico ', Errore);
    else      Gestore := FALSE;
  end;
end;
(*$F-*)
begin
  InstallaGestoreEccezioni(Gestore);
  Assign(f, 'PIPPO');
  Writeln(f, 'Scrittura su file non aperto');
end.
```

```
Program ErrDos;
(*$I+*)
uses Gestecc;
var
  f: text;
  Errore: word;
(*$F+*)
function Gestore: boolean;
begin
  if ErroreRunTime <= 17 then
    Gestore := TRUE     (* si tratta di errore DOS *)
  else
    Gestore := FALSE;   (* altro tipo di errore -> halt *)
end;
(*$F-*)
begin
  InstallaGestoreEccezioni(Gestore);
  Assign(f, 'NONESIST');
  Reset(f);                 (* Apertura file inesistente *)
  Errore := ErroreRT;
  Writeln('Errore: ', Errore);
  DisinstallaGestoreEccezioni;
end.
```

Figura 5 - Esempio di intercettazione di un errore DOS.

```
Program ErrCrit;
(*$I+*)
uses Gestecc;
var
  f: text;
  Errore: word;
(*$F+*)
function Gestore: boolean;
begin
  if (ErroreRunTime >= 150) and (ErroreRunTime <= 162) then
    Gestore := TRUE     (* si tratta di errore critico *)
  else
    Gestore := FALSE    (* altro tipo di errore -> halt *)
end;
(*$F-*)
begin
  InstallaGestoreEccezioni(Gestore);
  Writeln('Apri il drive A:, poi premi Enter');
  Readln;
  Assign(f, 'A:PIPPO');
  Rewrite(f);
  Errore := ErroreRT;
  Writeln('Errore: ', Errore);
end.
```

Figura 7 - Esempio di intercettazione di un errore critico.

Figura 6 - Esempio di intercettazione di un errore di I/O.

Un po' di esempi

Le figure 5, 6 e 7 vi propongono esempi di intercettazione di errori DOS, di I/O e critici, con qualche variazione sul tema. Notiamo in primo luogo che, per i motivi sopra illustrati, per prima cosa si assicura che sia attiva la direttiva $I. Per il resto, la cosa più importante è l'uso della variabile ErroreRunTime e della funzione ErroreRT. La funzione dovrebbe essere usata con le stesse modalità che il manuale raccomanda per IOResult: subito dopo le istruzioni che potrebbero generare un'eccezione, e per assegnare l'eventuale codice di errore ad una variabile; ErroreRT infatti non si limita a ritornare il codice di errore, ma chiama IOResult per reinizializzare a zero la variabile predefinita InOutRes (in caso contrario verrebbero ignorate tutte le richieste di I/O successive ad una che avesse provocato errore) e azzera anche ErroreRunTime. Il valore di questa non dovrebbe essere quindi letto direttamente, se non nell'ambito del gestore. Il gestore, infatti, ha bisogno di sapere che tipo di errore si è verificato, ma se si servisse di ErroreRT provocherebbe un azzeramento che vanificherebbe le chiamate alla funzione che occorressero nel codice eseguito dopo l'istruzione «colpevole» (vedi figura 5). D'altra parte, se si vuole instaurare un dialogo con l'utente nell'ambito del gestore (come in figura 6), è opportuno chiamare ErroreRT per consentire ulteriori operazioni di I/O nel caso che sia stata una di queste a generare l'eccezione.

Lascio alla vostra fantasia ulteriori esempi relativi ad errori di tipo aritmetico (potreste anche trarre ispirazione dall'esempio del mese scorso), e mi limito ad un breve cenno alla procedura Raise: consente di simulare l'insorgere di un eccezione, e lo fa nel modo più semplice, chiamando la procedura predefinita RunError. Vi prego però di lavorare sempre senza coprocessore (reale o emulato che sia). Le eccezioni segnalate da questo seguono tutt'altra impostazione e richiedono quindi accorgimenti particolari. Ne riparleremo tra un mese. MC

L'informatica su misura

&Compaq

SPARTA Informatica

Compaq LTE. Un computer da usare ovunque: in treno, in aereo e durante un meeting.
Misura 21,6 x 27,9 x 4,8 e pesa solo 2 chili e 800 grammi. E' il compagno insostituibile del manager, del consulente, del dirigente d'azienda e del professionista: cioè di chi ha bisogno di lavorare in viaggio ed in luoghi diversi, con la funzionalità e la veloce capacità di elaborazione necessarie, in versione 80C86 e 80C286 a 10 e 12 MhZ.
Dispone di una unità a disco fisso ad alta capacità da 20 a 40 Mb, una unità dischetti di 3,5" da 1,44 Mb ed un Modem interno opzionale da 2400 baud, che permette di colloquiare con altri computer. La batteria ricaricabile interna garantisce tre ore e mezza di autonomia, e l'adattatore, oltre a ricaricare in appena un'ora e mezza, permette anche il funzionamento del computer in rete. Il video retroilluminato supertwist a cristalli liquidi è di chiara visione in qualsiasi condizione di luce.
Compaq LTE, un contributo significativo al mondo del Personal Computer.

SPARTA INFORMATICA Via delle Sette Chiese, 142 - 00145 Roma
Tel.06/5141652-5141653-5137104-5137901 • Fax:06/5126489 • Teleassistenza:06/5126752

Concessionario Autorizzato

COMPAQ