di Sergio Polini

Gestione delle eccezioni in Turbo Pascal 3.0

MC vi ha proposto a luglio la prova del nuovissimo Turbo C++; in questo numero vi propone le prove delle ultime versioni del Turbo Debugger e del Turbo Assembler, e prossimamente vi proporrà quella del rivoluzionario Turbo Profiler. Con una politica coerente e coraggiosa, la Borland sta realizzando una crescente integrazione tra prodotti di diverso livello (assembler, linguaggi, applicativi), accompagnata dall'adesione alle metodologie più avanzate di analisi e programmazione (database relazionali con SQL, programmazione orientata all'oggetto). Potrà quindi sembrarvi strano un ritorno al vecchio e glorioso Turbo Pascal 3.0. In realtà questo è ancora piuttosto diffuso, e in esso vennero messe a punto tecniche di gestione degli errori che si ritrovano anche nelle più sofisticate versioni successive. Inoltre, come ricorderete, lo spunto per questa nostra chiacchierata ci è stato offerto proprio da un utente della 3.0. Ci sono quindi buoni motivi per un temporaneo ritorno all'antico

La volta scorsa abbiamo passato in rassegna le tecniche di gestione delle eccezioni adottate in diversi linguaggi: dal potente PL/1, con le sue soluzioni forse anche troppo flessibili, al rigoroso CLU, caratterizzato dalla adesione ad una metodologia di programmazione basata sui tipi di dati astratti. Abbiamo anche considerato che, a meno di insopportabili acrobazie, non è possibile adottare liberamente le scelte che potrebbero sembrarci più opportune: bisogna fare i conti con il Turbo Pascal così come è. Abbiamo quindi delineato le caratteristiche che può avere un gestore di eccezioni in Turbo Pascal, mediante un confronto con quelle dei gestori presenti in altri linguaggi: dopo il verificarsi di una eccezione il controllo, se non si vuole far terminare il programma, deve tornare alla funzione o procedura in cui questa si è verificata; il gestore deve poter essere installato o disinstallato più volte, e quindi associato in modo dinamico alle possibili eccezioni, le quali devono poter essere abilitate o disabilitate a piacere; si devono poter simulare eccezioni mediante una procedura analoga al SIGNAL del PL/1 o al raise di ADA.

Divisione intera per zero

Il nostro lettore era partito dalla divisione per zero. Se durante l'esecuzione di una istruzione DIV o IDIV accade che l'argomento di questa è zero, il microprocessore fa scattare un INT 0. Ciò equivale a passare il controllo ad una routine che, per default, provoca la fine immediata del programma in esecuzione e la visualizzazione di un messaggio del tipo «Oltre i limiti matematici consentiti». Il Turbo Pascal intercetta l'INT 0, ma solo per garantire una terminazione controllata del programma in esecuzione. Nulla vieta, peraltro, di associare all'INT 0 una routine che produca effetti diversi, come quella che vi propongo nella figura 1. La prima istruzione **inline** salva sullo stack i registri, l'ultima li ripristina, rimette a posto i registri SP e BP, quindi ritorna il controllo a dove l'esecuzione era stata interrotta con una istruzione IRET. La seconda istruzione agisce invece sul registro IP.

Quando scatta un INT 0, vengono salvati nello stack, oltre ai flag, il segmento e l'offset dell'istruzione che lo ha provocato; il successivo IRET rimette quel segmento nel registro CS e quell'offset nel registro IP, in modo che l'esecuzione possa ritornare lì dove era stata interrotta. Ciò vorrebbe dire che verrebbe rieseguita l'istruzione che aveva tentato la divisione per zero, con nuova chiamata dell'INT 0, e così via. Non possiamo ovviamente lasciare che le cose vadano in questo modo: occorre provocare un trasferimento del controllo alla istruzione *successiva* alla DIV o IDIV che aveva provocato l'interrupt, aggiungendo un opportuno valore alla copia del registro IP salvata nello stack, all'indirizzo BP+2. Valore «opportuno» vuol dire che si deve tener conto della «lunghezza» dell'istruzione, che può in teoria essere di 2 o 4 byte; possiamo tuttavia rimandare questo problema, in quanto (per quanto ho potuto constatare) il Turbo Pascal 3.0 pone sempre il divisore nel registro CX ed esegue sempre un IDIV CX, istruzione lunga 2 byte.

La routine illustrata nella figura 1 quindi funziona, ma fa ben poco: consente solo la prosecuzione del programma senza che nessuno possa accorgersi di cosa è successo. Per ottenere effetti più utili occorrerebbe visualizzare un messaggio di avvertimento, assegnare un dato valore ad una qualche variabile, magari provocare un risultato convenzionale della divisione. Prima di andare oltre, tuttavia, occorre considerare che la divisione per zero è solo una delle eccezioni possibili, ed inoltre che solo la divisione tra due interi può provocare un INT 0: nel codice che esegue la divisione tra due *real*, infatti, non c'è traccia né di DIV né di IDIV. Abbiamo

```
procedure NuovoInt0;
begin
  inline($50/$53/$51/$52/$56/$57/$1E/$06);
  inline($83/$46/$02/$02);          (* ADD WORD PTR [BP+2],2 *)
  inline($07/$1f/$5F/$5E/$5A/$59/$5B/$58/$8B/$E5/$5D/$CF);
end;
```

Figura 1
Un primo approccio ad una routine che intercetti l'INT 0.

quindi bisogno di un meccanismo più generale.

Il file GESTECC.INC

Come abbiamo appena ricordato, anche il Turbo Pascal intercetta l'INT 0. La routine a ciò designata non fa altro che mettere un quattro nel registro DL, per proseguire poi «a cascata» nel codice che gestisce tutte le situazioni di fine programma, normali o anomale. Nello stesso codice finiscono anche le altre eccezioni e quindi, almeno in linea di principio, intervenendo in qualche modo su questo si potrebbe controllare tutto. Vale la pena di provare. Nella figura 2 vi propongo quindi un tentativo del genere. La prima procedura non è altro che una nuova versione di quella della figura 1: la differenza è rappresentata dal codice che, dopo aver ripristinato, per ogni evenienza, il segmento dati, pone un 5 nella variabile *ErroreRunTime;* si tratta di un codice d'errore diverso da quelli predefiniti, messo lì con l'intento di distinguere la divisione intera per zero da quella che si verificasse con numeri *real*. Segue la funzione *ErroreRT* che, analogamente alla *IOResult*, ritorna il codice dell'ultimo errore se ve ne è stato uno, oppure zero.

Il gestore di eccezioni viene installato o disinstallato con la procedura *InstallaGestoreEccezioni*. Questa vuole un parametro di tipo *integer* che può assumere due tipi di valori; se diverso da zero, deve essere l'indirizzo di una funzione con risultato di tipo *boolean* (il gestore vero e proprio); se uguale a zero, l'effetto della esecuzione della procedura sarà quello di disinstallare il gestore. Una variabile locale *PrimaVolta* consente di distinguere dalle successive la prima chiamata della procedura; in occasione di questa vengono inizializzate la costante tipizzata *DataSegment* e la variabile *ErroreRunTime*, e viene salvato l'indirizzo della routine standard di intercettazione dell'INT 0; soprattutto si controlla se... il vostro compilatore è uguale al mio. Mi spiego. Ho lavorato con la versione 3.01A del compilatore e ho dovuto fare riferimento a locazioni specifiche del codice da questo prodotto; in particolare, ho dovuto assumere che l'indirizzo dell'istruzione *successiva* a quella che provoca un'eccezione viene salvato nella locazione $0186 del segmento dati, assunzione ovviamente valida per quella versione, ma non necessariamente per altre. Per controllare che il file GESTECC.INC funzioni anche con il vostro compilatore, non avete che da provare incrociando le dita. Ho comunque previsto un qualche controllo automatico: all'offset $103C del segmento codice, perché tutto funzioni, ci devono essere le istruzioni MOV DH,02 e PUSH DX: se così non è il programma termina dopo aver mostrato il messaggio «Impossibile installare gestore eccezioni».

Per il resto, il comportamento della procedura dipende dal valore del suo parametro, assegnato alla variabile globale *OfsGestore*: se questo è diverso da zero, viene associata all'INT 0 la procedura *NuovoInt0* e quelle istruzioni all'offset $103C vengono sostituite da un JMP al codice della procedura *ErrRTime*; in caso contrario, vengono ripristinati sia l'originario INT 0 che le istruzioni all'offset $103C come prodotte dal compilatore. La procedura *InstallaGestoreEccezioni* può essere chiamata più volte; ciò consente di installare volta per volta il gestore che si ritenga più opportuno, come pure di ripristinare, anche temporaneamente, il comportamento normale di un programma Turbo Pascal.

La procedura ErrRTime

Abbiamo osservato a luglio che il Turbo Pascal già offre una gestione di alcune eccezioni, mediante la direttiva $I e la funzione predefinita *IOResult*; ci occupiamo quindi dei soli errori di ese-

```
(* GESTECC.INC - Gestione delle eccezioni in Turbo Pascal 3.0 *)

const
  DataSegment: integer = 0;

var
  ErroreRunTime: integer;
  OfsGestore: integer;

procedure NuovoInt0;
begin
  inline($50/$53/$51/$52/$56/$57/$1E/$06);
  inline($2E/$A1/DataSegment/      (* MOV AX,CS:DataSegment *)
         $8E/$D8);                 (* MOV DS,AX *)
  inline($B8/$05/$00/              (* MOV AX,0005h *)
         $A3/ErroreRunTime);       (* MOV ErroreRunTime,AX *)
  inline($83/$46/$02/$02);         (* ADD WORD PTR [BP+2],2 *)
  inline($07/$1f/$5F/$5E/$5A/$59/$5B/$58/$8B/$E5/$5D/$CF);
end;

function ErroreRT: integer;
begin
  ErroreRT := ErroreRunTime;
  ErroreRunTime := 0;
end;

procedure ErrRTime;
const
  OfsRitorno: integer = 0;
begin
  inline($8B/$E5/$5D);             (* MOV SP,BP / POP BP *)
  inline($81/$E2/$FF/$00/          (* AND DX,00FFh *)
         $89/$16/ErroreRunTime);   (* MOV ErroreRunTime,DX *)
  inline($A1/$86/$01/              (* MOV AX,[0186] *)
         $2E/$A3/OfsRitorno);      (* MOV CS:OfsRitorno,AX *)
  if (ErroreRunTime < $F0) and (ErroreRunTime <> $11) then begin
    inline($BB/OfsGestore/         (* MOV BX,OfsGestore *)
           $8B/$16/ErroreRunTime/  (* MOV DX,ErroreRunTime *)
           $52/                    (* PUSH DX *)
           $4C/                    (* DEC SP ; per call di funz. boolean *)
           $FF/$17/                (* CALL [BX] *)
           $5A/                    (* POP DX *)
           $75/$06/                (* JNZ Prosegui *)
           $B6/$02/                (* MOV DH,02 *)
           $52/                    (* PUSH DX *)
           $E9/$103F-*-2/          (* JMP alla locazione $103F *)
      (* Prosegui: *)
           $80/$FA/$04/            (* CMP DL,04 *)
           $77/$03/                (* JA NonReal *)
           $50/$53/$52/            (* PUSH AX/PUSH BX/PUSH DX *)
      (* NonReal: *)
           $2E/$FF/$36/OfsRitorno/(* PUSH CS:OfsRitorno *)
           $C3);                   (* RET *)
  end
  else
    inline($8B/$16/ErroreRunTime/  (* MOV DX,ErroreRunTime *)
           $B6/$02/$52/            (* MOV DH,02 / PUSH DX *)
           $E9/$103F-*-2);         (* JMP alla locazione $103F *)
end;
```

(continua a pag. 268)

(segue da pag. 267)

```
procedure Raise(Codice: byte);
begin
  if Codice <= 4 then Codice := $F1;
  inline($8A/$56/<Codice);          (* MOV DL,BYTE PTR Codice *)
  inline($8B/$E5/                   (* MOV SP,BP *)
         $5D);                      (* POP BP *)
  inline($8F/$06/$86/$01/           (* POP WORD PTR [0186] *)
         $58/                       (* POP AX *)
         $E9/$103C-*-2);            (* JMP alla locazione $103C *)
end;

procedure InstallaGestoreEccezioni(Proc: integer);
const
  PrimaVolta: boolean = TRUE;
  SegPrevInt0: integer = 0;
  OfsPrevInt0: integer = 0;
var
  Reg: 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;
  pb: ^byte;
  pi: ^integer;
begin
  if PrimaVolta then begin
    PrimaVolta := FALSE;
    pi := Ptr(CSeg, $103C);
    pb := Ptr(CSeg, $103E);
    if (pi^ <> $02B6)           (* se in CS:103C non c'e' MOV DH,02 *)
      or (pb^ <> $52)           (* se in CS:103E non c'e' PUSH DX *)
        then begin
          Writeln('Impossibile installare gestore eccezioni');
          Halt(1);
        end;
    DataSegment := DSeg;
    ErroreRunTime := 0;
    Reg.AX := $3500;
    MsDos(Reg);
    SegPrevInt0 := Reg.ES;
    OfsPrevInt0 := Reg.BX;
  end;
  OfsGestore := Proc;
  if OfsGestore <> 0 then begin
    Reg.DS := CSeg;
    Reg.DX := Ofs(NuovoInt0);
    Reg.AX := $2500;
    MsDos(Reg);
    pb := Ptr(CSeg,$103C);
    pi := Ptr(CSeg,$103D);
    pb^ := $E9;
    pi^ := Ofs(ErrRTime) - $103C - 3;
  end

    else begin
      Reg.DS := SegPrevInt0;
      Reg.DX := OfsPrevInt0;
      Reg.AX := $2500;
      MsDos(Reg);
      pi := Ptr(CSeg, $103C);
      pb := Ptr(CSeg, $103E);
      pi^ := $02B6;
      pb^ := $52;
    end;
  end;
```

Figura 2 — Il file GESTECC.INC contiene il codice per la gestione degli errori di esecuzione, esclusi quelli con codice $11 (indice stringa non valido), $F0 (non trovato il file di overlay) e $FF (collisione heap/stack).

cuzione (breve nota terminologica: nelle prime versioni del compilatore, gli errori di I/O e quelli di esecuzione erano considerati due distinte categorie; a partire dalla versione 4, invece, gli errori di I/O sono una delle quattro categorie degli errori di esecuzione, accanto a errori DOS, errori critici e errori fatali; questi ultimi corrispondono agli errori di esecuzione delle prime versioni).

Ogni volta che si verifica un errore di esecuzione, il Turbo Pascal 3.0 (o almeno il 3.01A) provoca un salto all'offset $103C; se mettiamo qui un salto ad una nostra procedura, questa potrà gestire l'errore con una certa libertà. Occorrono tuttavia alcune cautele e un po' di preliminari; ecco quindi perché la procedura *InstallaGestoreEccezioni* non pone un salto diretto al gestore, ma ad una procedura intermedia che ho chiamato *ErrRTime*. Questa per prima cosa annulla le istruzioni automaticamente generate dal compilatore al suo inizio (PUSH BP / MOV BP, SP / PUSH BP), rimettendo subito a posto lo stack e i registri SP e BP; provvede poi a salvare il codice dell'errore, contenuto in DL, nella variabile *ErroreRunTime* e l'indirizzo della istruzione successiva a quella che lo ha provocato nella costante tipizzata *OfsRitorno*. Si controlla poi che il codice d'errore sia minore di $F0 e diverso da $11; ciò equivale ad escludere da possibili tentativi di gestione tre tipi di errore: un valore superiore a 255 per il parametro Pos delle procedure *Copy, Delete* e *Insert* (in quanto in questi casi il codice prodotto dal compilatore fa un uso dello stack che costringerebbe ad acrobazie forse eccessive), un file di overlay non trovato (situazione che ritengo obbligatorio gestire in modo «normale»), e le collisioni tra heap e stack (in quanto provocate da due situazioni troppo diverse tra loro: memoria dinamica insufficiente e overflow dello stack).

Se l'errore appartiene ad un tipo «gestibile», viene chiamato il gestore che risulta in quel momento installato, cioè quello il cui indirizzo è stato posto nella variabile *OfsGestore*. Deve trattarsi di una funzione di tipo boolean: se ritorna FALSE, ciò vuol dire che si deve tornare alle routine standard; vengono quindi replicate le istruzioni sostituite dal JMP a *ErrRTime* e si provoca un salto alla istruzione successiva a queste (all'offset $103F; analogo il comportamento nel caso l'errore non sia di tipo «gestibile»). Se quella funzione ritorna invece TRUE, viene provocato un ritorno alla istruzione successiva a quella che aveva provocato l'errore. Ciò si ottiene con un RET preceduto da un PUSH che pone nello stack il valore prima assegnato ad *OfsRitorno*. Alcuni errori richiedono tuttavia un trattamento particolare. I primi quattro codici corrispondono tutti ad operazioni in virgola mobile: overflow, divisione per zero, argomento negativo per *Sqrt*, argomento nullo o negativo per *Ln* (curiosamente, in quest'ultimo caso si genera un errore con codice 4, lo stesso attribuito alla divisione intera per zero: è questo il motivo per cui tale situazione deve essere gestita a parte, intercettando l'INT 0, invece che mediante un unico meccanismo di portata più generale); le operazioni in virgola mobile vengono implementate mediante due o tre chiamate a routine di libreria, e l'ultima chiamata presuppone che nello stack siano stati spinti tre registri contenenti la rappresentazione di un numero *real*. Prima di restituire il controllo al sottoprogramma interrotto occorre quindi mettere qualcosa nello stack; ecco il motivo di quei tre PUSH che trovate nel listato. Ciò comporta, ovviamente, una rinuncia a provocare risultati convenzionali, ma se ne guadagna in generalità.

```
Program GestDemo;
var
  x, y: real;
  m, n: integer;
(*$I GESTECC.INC *)
function GestoreEccezioni: boolean;
var
  Codice: integer;
  Prosecuzione: boolean;
begin
  Prosecuzione := TRUE;
  Codice := ErroreRunTime;
  case Codice of
    1: Writeln('Overflow su real');
    2: Writeln('Divisione real per zero');
    3,4: Writeln('Argomento non valido');
    else Prosecuzione := FALSE;
  end;
  GestoreEccezioni := Prosecuzione;
end;
function GestoreNullo: boolean;
begin
  if ErroreRunTime <= 4 then
    GestoreNullo := TRUE
  else
    GestoreNullo := FALSE;
end;
begin
  InstallaGestoreEccezioni(Ofs(GestoreEccezioni));
  x := 1e37;              (* Overflow su real *)
  y := 1e37;
  x := x * y;

  x := 3.0;               (* Divisione real per zero *)
  y := 0.0;
  x := x / y;

  m := 3;
  n := 0;
  m := m div n;
  if ErroreRT = 5 then Writeln('Divisione intera per zero');

  x := -3;                (* Sqrt di un numero negativo *)
  y := Sqrt(x);

  InstallaGestoreEccezioni(Ofs(GestoreNullo));
  x := 0.0;               (* Logaritmo di zero *)
  y := Ln(x);
  if ErroreRT = 4 then Writeln('Argomento nullo o negativo');

  InstallaGestoreEccezioni(0);
  x := 3.0;
  y := 0.0;
  x := x / y;             (* Provoca un Run-time error *)
end.
```

Figura 3 — Un esempio di uso dei meccanismi offerti dalle routine del file GESTECC.INC.

Per finire, GESTECC.INC vi propone anche una procedura *Raise*. Questa prevede un parametro a cui assegnare un codice di errore qualsiasi, che si voglia poi trattare con un gestore di eccezioni. Si controlla però che il codice non sia di quelli che, come abbiamo appena visto, richiedono un trattamento particolare dello stack: se così fosse, il codice viene convertito in un valore «non gestibile» in modo da provocare la terminazione del programma; altrimenti si pulisce lo stack analogamente a quanto avviene in *ErrRTime*, si pone nella locazione $0186 del segmento dati l'indirizzo della istruzione successiva alla chiamata di *Raise*, si toglie dallo stack anche il parametro, si salta infine all'indirizzo $103C. L'effetto è del tutto analogo al verificarsi di una situazione di errore, ma con un codice scelto dall'utente; l'effetto è cioè analogo a quello prodotto dalla procedura *RunError* del Turbo Pascal 5.x.

Un esempio

La figura 3 contiene un esempio di uso dei meccanismi messi a punto in GESTECC.INC. Vengono generate diverse eccezioni in tre diverse situazioni: installazione di un gestore che «fa qualcosa» e di uno che non fa nulla (o quasi), gestore disinstallato.

Nel listato che vi propongo il gestore che «fa qualcosa» si limita a mostrare un messaggio di avvertimento. È chiaro che ciò, in un programma vero, dovrebbe essere fatto in modo coerente con l'interfaccia utente che si sta adottando; potreste provare, ad esempio, a includere GESTECC nel MicroCalc in modo da implementare un controllo della elaborazione delle formule tale che, nel caso di calcoli che portino a risultati inaccettabili, la terminazione del programma venga sostituita dalla visualizzazione nelle caselle di messaggi analoghi a quelli di un vero *spreadsheet*. Le cose da tenere presenti sono due: la funzione che viene installata come gestore deve ritornare TRUE se si vuole la prosecuzione dell'esecuzione o FALSE se si vuole far terminare il programma; il codice d'errore deve essere esaminato cercandolo nella variabile *ErroreRunTime* all'interno del gestore, chiamando invece la funzione *ErroreRT* fuori di questo. Il motivo dovrebbe essere ovvio: normalmente va chiamata la funzione, in quanto questa azzera la variabile *ErroreRunTime* dopo averne letto il valore, e può quindi essere usata in modo analogo a quello raccomandato dal manuale per la funzione *IOResult*; non dovrebbe essere chiamata all'interno del gestore, proprio per consentire tale uso nelle normali funzioni e procedure.

Il gestore che «non fa nulla» in realtà deve comunque ritornare TRUE o FALSE e non è detto che debba sempre ritornare TRUE. Dico che «non fa nulla» nel senso che non avverte l'utente che qualcosa è andato storto, e quindi lascia al programmatore l'onere di verificare l'esito delle sue istruzioni mediante chiamate della funzione *ErroreRT* (al solito, analogamente a quanto si fa con *IOResult*). La funzione *GestoreNullo* del listato, per il resto, fa qualcosa di non trascurabile: seleziona gli errori di esecuzione che devono essere intercettati senza provocare l'interruzione del programma; in altri termini, mediante diversi «gestori nulli» diventa possibile abilitare o disabilitare a piacere le eccezioni.

Conclusioni

Possiamo concludere che, pure con qualche limitazione, siamo riusciti a conseguire gli obiettivi che ci eravamo posti: includendo il file GESTECC.INC nei vostri programmi, potrete tenere sotto controllo anche eccezioni diverse da quelle catalogate come errori di I/O e con tecniche analoghe (una procedura di installazione al posto della direttiva $I, una funzione *ErroreRT* al posto di *IOResult*).

Siamo però stati costretti a qualche acrobazia, soprattutto a fare pesanti assunzioni sul codice prodotto dal compilatore, al punto che non è sicuro che i «trucchi» adottati funzionino con tutte le edizioni del Turbo Pascal 3.0. Vedremo il mese prossimo come le versioni successive consentano un trattamento più pulito, insieme ad ulteriori possibilità. Incontreremo in compenso altri problemi, ma... cercheremo di non perderci d'animo.

A presto.