di Sergio Polini

Gestione delle eccezioni (exception handling)

Ricorderete che a marzo, rispondendo a Cristian D'Aloisio di Liscia (CH), che non riusciva ad intercettare la divisione per zero, avevo promesso di affrontare su queste pagine il tema della "gestione delle eccezioni" in Turbo Pascal. Eccomi ora a rispettare l'impegno. Si tratta di un argomento di notevole interesse, sia teorico che pratico, che abbiamo già sfiorato quando ci siamo occupati della gestione degli errori critici. Ora cercheremo di vedere come ci si può salvare anche da altri tipi di situazioni "anomale"; ma cercheremo anche di evitare la pura e semplice ricerca di espedienti, tenendo presente che la divisione per zero, o gli errori critici, sono solo aspetti parziali di un problema più generale

Una procedura o una funzione, in generale un sottoprogramma, può terminare in diversi modi. Può giungere fino alla sua ultima istruzione, e di qui ritornare al punto in cui era stato chiamato, ma può anche saltare una o più delle sue istruzioni se interviene un **return** (assente in Turbo Pascal, dove però si dispone della chiamata della procedura *Exit*, che ha lo stesso effetto), o un **goto** ad una *label* posta subito dopo l'ultima istruzione. Ancora: il **goto** del Turbo Pascal consente solo salti a locazioni contenute nello stesso blocco, ma altri linguaggi, compreso il Pascal standard, permettono anche di saltare fuori di un blocco; un sottoprogramma può quindi terminare (in modo un po'... disordinato) anche con un **goto** "non locale" (abbiamo visto a suo tempo come implementare salti di questo tipo, a volte sicuramente utili, in Turbo Pascal).

Non è tutto; divisione per zero, errori critici, tentativi di aprire troppi file contemporaneamente sono tutti esempi di un'altra classe di situazioni: un sottoprogramma termina perché, durante la sua esecuzione, si è verificata una condizione eccezionale che ne impedisce la normale prosecuzione. Non è possibile dare una definizione assoluta, valida per tutti gli ambienti hardware/software, di "condizione eccezionale"; hanno ovvia importanza le caratteristiche della macchina, in particolare la sua capacità di generare automaticamente "interruzioni" (quali l'INT 0), ma molto dipende dal sistema operativo e dalle decisioni assunte da chi ha definito le caratteristiche del linguaggio di programmazione che stiamo usando.

Prima di entrare nel vivo della discussione, e soprattutto prima di cominciare a "smanettare", sarà bene vedere come il problema si è presentato e si presenta in altri linguaggi.

PL/1

Il PL/1 nacque come tentativo di creare un unico linguaggio di uso generale che incorporasse le caratteristiche principali di FORTRAN, Algol 60 e COBOL. Non si trattò di una semplice somma (o media) di quanto poteva già trovarsi altrove, ma vennero introdotte alcune importanti innovazioni, tra cui la gestione delle eccezioni (chiamate "condizioni"; vedi figura 1). Era la prima volta che si tentava di offrire al programmatore la possibilità di tenere sotto controllo anche eventi anomali, e ciò venne fatto in modo molto articolato e potente.

In alcuni casi, ad esempio, il programmatore può decidere quali "condizioni" abilitare e quali no, con riferimento ad una singola istruzione o ad un blocco BEGIN..END ("disabilitare" una condizione vuol dire rinunciare al controllo

```
- condizioni abilitate per default ma disattivabili:

  CONVERSION FIXEDOVERFLOW OVERFLOW UNDERFLOW ZERODOVIDE

- condizioni disabilitate per default ma attivabili:

  SIZE SUBSCRIPTRANGE STRINGRANGE CHECK AREA

- condizioni sempre abilitate:

  ATTENTION CONDITION ENDFILE ENDPAGE ERROR FINISH
  KEY NAME PENDING RECORD TRANSMIT UNDEFINEDFILE
```

Figura 1
Le "condizioni" che possono essere gestite in PL/1.

che questa possa verificarsi; pensate alla direttiva $R del Turbo Pascal). Quando si verifica una "condizione", se questa è abilitata viene di norma eseguita una azione standard, che può tuttavia essere sostituita da altre mediante una istruzione ON: ON ZERODIVIDE BEGIN; ... END. Possono essere previste diverse istruzioni ON per ogni condizione: ognuna di esse ha effetto dal momento in cui viene eseguita (e qui "esecuzione" vuol dire "associazione di un dato gestore alla condizione specificata"), fino a quando non ne viene eseguita un'altra per la stessa condizione, oppure fino a che non si esce dal blocco in cui l'istruzione si trova (si tratta, potremmo dire, di gestori "locali" di eccezioni).

Il programmatore può anche simulare il verificarsi di una condizione con una istruzione SIGNAL. In qualsiasi modo, reale o simulato, si sia verificata una condizione, il gestore ad essa eventualmente associato si comporta come una procedura chiamata esplicitamente: farà quello che deve fare, per poi restituire il controllo alla istruzione che aveva fatto scattare la condizione o a quella successiva, secondo il tipo di condizione; è però possibile intervenire su tale meccanismo per ottenere effetti diversi, ad esempio con un: ON OVERFLOW GO TO ...

In generale, si tratta di strumenti (anche troppo) flessibili e potenti, che saranno suonati familiari a molti di voi, vista la somiglianza con gli ON ERROR del BASIC, dei quali anticipano alcuni problemi. Sono stati criticati, ad esempio, il carattere molto dinamico della associazione tra una condizione ed un gestore (una procedura chiamata da un'altra può trovarsi ad ereditare tutto un insieme di associazioni condizione-gestore potenzialmente non coerente con le caratteristiche della procedura stessa), o il fatto che alcuni gestori, quali il "ON OVERFLOW GO TO ..." visto prima, risultano equivalenti a quei GOTO non-locali che molti (autori del Turbo Pascal compresi) preferiscono tenere al bando.

CLU

Nella seconda metà degli anni '70 venne proposto il CLU, linguaggio progettato come supporto ad una metodologia di programmazione basata sui tipi di dati astratti, chiamati **cluster.**

Il CLU presenta un meccanismo di gestione delle eccezioni più rigoroso di quello del PL/1. Quando si verifica un'eccezione in una procedura, questa termina e ritorna il controllo a quella da cui era stata chiamata, la quale deve fornire il gestore; l'associazione tra un'eccezione e il suo gestore è in questo modo statica. Nella figura 2 potete vedere un esempio. Le procedure vengono dichiarate indicando, oltre ai parametri e al tipo del risultato, anche la lista delle eccezioni che ne possono provocare una terminazione "anomala" ma controllata. Da rilevare che le eccezioni possono avere parametri, la cui elencazione consente al compilatore controlli di coerenza impensabili in PL/1.

```
sum_stream = proc(s: stream) returns(int)
        signals(overflow, unrepresentable_integer(string), bad_format(string))
  sum : int := 0
  while true do
    sum := sum + get_number(s)
      resignal unrepresentable_integer, bad_format, overflow;
  end
  except
    when end_of_file: return(sum)
  end
end sum_stream

get_number = proc(s: stream) returns(int)
        signals(end_of_file, unrepresentable_integer(string),
                bad_format(string))
  field : string := get_field(s)
    resignal end_of_file;
  return (s2i(field))
    except
      when unrepresentable_integer(field), bad_format, invalid_character (*):
      signal bad_format(field)
    end
end get_number
```

Figura 2 - Esempio di gestione delle eccezioni in CLU (tratto da Ellis Horowitz, Fundamentals of Programming Languages, Computer Science Press, 1984).

Nell'esempio, la procedura *sum_stream* riceve i valori letti da *get_number* e ne ritorna la somma. Viene considerata eccezione la fine del file di input; quando la chiamata di *get_field* incorre in tale eccezione, l'istruzione **resignal** la propaga da *get_number* alla procedura chiamante, *sum_stream*; in essa si trova il gestore, nella forma di un **except when**, che provoca l'uscita da *sum_stream* con il risultato in *sum*. Se però un dato letto dal file non può essere convertito in un intero, operazione a cui provvede *s2i*, viene generata una eccezione corrispondente al tipo di problema (intero troppo grande, formato non corretto, carattere non valido), che viene gestita dal gestore di *get_number* mediante la generazione di una eccezione con un **signal**. *sum_stream*, infine, può terminare anche propagando alla procedura che l'ha chiamata sia l'eccezione passatale da *get_number* sia i problemi di overflow eventualmente sorti durante le ripetute somme.

In sintesi, nel CLU diviene chiaro che l'"eccezione", per quanto "anomala", è pur sempre un modo del tutto normale in cui può terminare una procedura; invece di un **return** (o dell'equivalente *Exit* che ci propone il Turbo Pascal), una procedura può terminare anche con un **signal**, segnalando cioè l'anomalia alla procedura chiamante; l'unica differenza è nel modo in cui si ritorna a questa: nel primo caso, l'esecuzione prosegue con l'istruzione immediatamente successiva a quella di chiamata, dopo un **signal**, invece, il controllo viene trasferito al gestore che, mediante la clausola **except**, è associato alla chiamata.

```
begin
  -- sequenza di istruzioni
exception
  when NUMERIC_ERROR => ...
end;
```

Figura 3 - Schema di gestore di eccezioni in Ada.

Ada

Voluto dal Dipartimento della Difesa degli Stati Uniti, soprattutto per la programmazione di calcolatori dedicati e in applicazioni militari, Ada offre una gestione delle eccezioni simile a quella del CLU con una principale differenza: se una procedura A chiama una procedura B, e durante l'esecuzione di questa si verifica un'eccezione, il gestore non deve necessariamente essere fornito dalla procedura chiamante (A), ma può anche

essere fornito da un blocco ancora più esterno, o addirittura dalla stessa procedura B (che è quindi in grado di effettuare operazioni di "pulizia" prima di restituire il controllo alla chiamante). Non è possibile, peraltro, passare parametri ai gestori.

Sia Ada che CLU, comunque, si differenziano dal PL/1 per un aspetto fondamentale: non consentono che l'esecuzione di un blocco possa proseguire dopo un'eccezione. In Ada un gestore di eccezioni può essere posto alla fine di una qualsiasi sequenza di istruzioni nell'ambito di un blocco **begin..end** (vedi figura 3); se una istruzione genera un'eccezione per la quale è stata prevista un'apposita clausola **when**, il controllo passa al gestore e, quando questo termina, si esce dal blocco; altrimenti, si esce subito dal blocco e si cerca un gestore nei blocchi più esterni.

Non è il caso di illustrare tutti gli aspetti della gestione delle eccezioni in Ada, ma è interessante notare che è consentito definire eccezioni convenzionali oltre a quelle predefinite, e che si può provocare un'eccezione con **raise**, analoga al SIGNAL del PL/1; infine, come il PL/1 e a differenza del CLU, Ada consente di disabilitare alcune eccezioni.

```
procedure COPYOVER is
begin
  OPEN(INFILE);
  OPEN(OUTFILE);
loop
  GET(INFILE, CHARACTER);
  PUT(OUTFILE, CHARACTER);
end loop;
exception
  when END_OF_FILE =>
    PUT(OUTFILE, EOF);
    CLOSE(OUTFILE);
    CLOSE(INFILE);
end COPYOVER;

program CopyFile;
var
  InFile, OutFile: file of byte;
  b: byte;
begin
  Assign(InFile, 'COPYOVER.PAS');
  Reset(InFile);
  Assign(OutFile, 'COPIA');
  Rewrite(OutFile);
  (*$I-*)
  repeat
    Read(InFile, b);
    if IOResult = 100 then begin
      Close(InFile);
      Close(OutFile);
      Halt;
    end;
    Write(OutFile, b);
  until FALSE;
end.
```

Figura 4 - Un esempio di gestione di eccezione di I/O in Ada (anch'esso tratto da Horowitz), seguito da un equivalente codice Turbo Pascal in cui l'eccezione di "fine file" è controllata mediante la funzione IOResult.

Turbo Pascal

Dire che in Turbo Pascal c'è una gestione delle eccezioni sarebbe poco meno che un'eresia. Eppure... qualcosa c'è.

Per prima cosa dobbiamo chiederci cosa intendere per "eccezioni". La risposta che vi propongo per una tale domanda è semplice: consideriamo eccezioni tutte le condizioni che provocano un errore di esecuzione (un *Run-time Error*). In prima approssimazione, quindi, possiamo dire che la gestione delle eccezioni in Turbo Pascal consiste nella terminazione del programma in esecuzione, in modo tale da ripristinare gli interrupt sostituiti dal codice di *start-up* (i vari *SaveIntXX*), e da tornare al DOS con un codice d'errore descrittivo del problema e con l'indicazione del segmento e dell'offset dell'istruzione che lo ha provocato.

Se non ci fosse altro saremmo davvero nell'eresia. E tuttavia, a ben vedere, sia pure in modo frammentario, il Turbo Pascal offre alcuni strumenti che consentono di evitare qualche volta il brutale ritorno al DOS. Notiamo innanzitutto che gli errori di *Run-time* vengono divisi in quattro categorie: errori DOS, errori di I/O, errori critici, errori fatali. Prendiamo gli errori di I/O. C'è qui un "gestore" che, se "abilitato" disattivando la direttiva $I, sostituisce alla terminazione del programma un'azione completamente indolore: viene assegnato un codice ad una variabile interna, il cui valore può essere letto da una funzione predefinita *IOResult* (in figura 4 si confrontano codici Ada e Turbo Pascal sostanzialmente equivalenti; l'effetto ottenuto in Ada con la gestione della eccezione END_OF_FILE viene replicato in Turbo Pascal con *IOResult*).

Provando ad interpretare un tale "gestore" alla luce delle considerazioni sin qui svolte, potremmo dire che è un gestore "globale", nel senso che non ha la "località" e la "dinamicità" proprie di quelli del PL/1, ma può essere abilitato o disabilitato anche localmente; l'errore di I/O non può essere simulato, ma la sua gestione (anche qui come nel PL/1) consente il ritorno alla procedura che l'aveva generato. Non si tratta di considerazioni banali, ma certamente ovvie: CLU e Ada offrono meccanismi più rigorosi grazie ad aspetti (procedure che terminano con un **signal**, gestori racchiusi dentro un blocco **begin..end**) che richiedono apposite estensioni della sintassi di un linguaggio. E il Turbo Pascal è pur sempre un Pascal.

In compenso non è tutto qui. Gli errori critici, ad esempio, vengono trattati come normali errori di I/O. Ancora: tra gli errori fatali vi è pure l'overflow dello heap, provocato da una chiamata di *New* o *GetMem* in un momento in cui non vi è memoria disponibile per l'allocazione dinamica. Abbiamo qui una variabile *HeapError* alla quale possiamo assegnare l'indirizzo di una funzione *HeapFunc* che ritorni 0, 1 o 2. Utile in particolare il ritorno di un 1, che fa sì che *New* e *GetMem* ritornino un **nil** invece di provocare la fine del programma. Abbiamo cioè la possibilità di gestire una eccezione installando un gestore; analogamente a quanto avviene in PL/1, qui abbiamo perfino la possibilità di una associazione dinamica tra l'eccezione e il suo gestore: in ogni momento, infatti, l'effetto di un tentativo di allocare memoria non disponibile può causare la terminazione del programma o il ritorno al blocco in cui si era verificata l'eccezione, secondo il valore della variabile *HeapError*, che può essere **nil** o l'indirizzo di una funzione, o secondo il valore ritornato da questa funzione.

È possibile fare in modo che una eccezione provochi la terminazione della procedura in cui si è verificata e il ritorno immediato alla procedura chiamante? È possibile, in altri termini, una gestione delle eccezioni analoga a quella del CLU o di Ada? Potremmo dire che non sarebbe del tutto impossibile, ma certo complicato in misura quasi insopportabile. È relativamente facile, infatti, provocare l'uscita immediata dalla procedura o funzione in cui si è verificata l'eccezione, ma il ritorno ad un gestore contenuto nella chiamante (per limitarci al più semplice caso del CLU) richiederebbe l'implementazione di "tabelle di gestori" associate ad ogni procedura.

Possiamo quindi delineare alcune delle caratteristiche che può e deve avere un gestore di eccezioni in Turbo Pascal: deve consentire il ritorno al blocco in cui si era verificata l'eccezione, può essere installato o disinstallato più volte e può quindi essere associato in modo dinamico alle eccezioni, queste possono essere abilitate o disabilitate; l'effetto di una eccezione sarà la terminazione del programma o, se "gestita", l'assegnazione di opportuni valori ad alcune variabili (e magari la visualizzazione di messaggi di avvertimento). Si potrebbero anche prevedere eccezioni convenzionali (definite dal programmatore) e qualche meccanismo analogo al SIGNAL del PL/1 o al **raise** di Ada. Il mese prossimo ci metteremo all'opera. **MC**

NEWEL

home e personal computer

Via Mac Mahon, 75 - 20155 MILANO
Tel. (02) 33000036/323492 tutto il giorno - (02) 3270226 al mattino - Fax (02) 33000035
Chiuso il lunedì - Aperto il sabato

PC o AMIGA?? TUTTI E DUE!!!
Oggi è possibile con NEWEL
e la sua fantastica offerta speciale:

AMIGA 2000 SERIE 6.02. 1Mb chip Ram, nuovo Fat Agnus, con tastiera completo di mouse, Manuali basic e dischetti in italiano, Garanzia valida 12 mesi in tutto il territorio, + hard disk 20Mb, MS DOS AMIGA DOS originale 2092, + scheda di compatibilità MS DOS completa di drive 5 1/4, Manuali, MS DOS GW BASIC in italiano.

Garanzia ufficiale Commodore Italiana Spa.

Configurazione con Emulator Janus 8086 XT 512K + drive 5 1/4 360K L. 2.600.000
Configurazione con scheda AT Janus 80286 1Mb + drive 5 1/4 1,2Mb L. 3.200.000
AMIGA 2000 come sopra versione base L. 1.600.000

Scheda Janus XT base L. 499.000
Scheda Janus AT base L. 999.000
Hard disk 20Mb MS DOS L. 450.000

OPPURE
Se preferisci un PC scegli sempre
nella famiglia Commodore ai prezzi scontati NEWEL

PC30-3. Microprocessore 80286 6-8-12MHz, Memoria 640Kb, 1 FDD da 3 1/2", 1 HD da 20Mb, Tastiera avanzata, Monitor 1404 14" fosfori ambra, MS-DOS 4.01 L. 2.250.000

PC30V 3-2. Microprocessore 80286 6-8-12MHz, Memoria 640Kb, 1 FDD da 3 1/2", 1 HD da 20Mb, Tastiera avanzata, Scheda Super VGA, Monitor 1403 14" fosfori bianco carta, MS-DOS 4.01 L. 2.500.000
Con monitor Super VGA colori L. 3.200.000

PC40/40-3. Microprocessore 80286 6-8-12MHz, Memoria 1Mb, 1 FDD da 5 1/4", 1 HD da 40Mb (19ms), Tastiera avanzata, VGA, Monitor VGA 1403 14" fosfori bianco carta, MS-DOS 4.01 L. 2.990.000
Con monitor Super VGA colori L. 3.700.000

TUTTI I COMPUTER SONO CORREDATI DI MANUALI IN ITALIANO, GARANZIA COMMODORE ITALIA 12 MESI.

TUTTI I PREZZI SI INTENDONO IVA COMPRESA

Le schede Janus sono acquistabili solo in combinazione.

DATI TECNICI PC 30-3

CPU. CPU 80286, 16 bit; Coprocessore matematico 80286 (opzionale); Frequenza di clock 6/8/12MHz, commutabile.
Memoria. RAM da 640Kb (standard); Unità a dischetti da 3 1/2, 1,44Mb; Unità a dischetti da 5 1/2", 1,2Mb (opzionale); BIOS AutoConfiguration™ Commodore con funzione di ripristino incorporata (ROM da 32K); Unità a disco fisso da 20Mb, con interfaccia per bus AT.
Interfacce. Interfaccia seriale RS232C; Interfaccia parallela Centronics; Interfaccia Mouse (Bus-Mouse Microsoft ™ compatibile, per collegamento con Mouse 1352 Commodore)
Adattatore video. Adattatore video EGA, MDA, CGA; Hercules compatibile.
Monitor. Monitor monocromatico da 14" con possibilità di inclinazione e rotazione.
Orologio. Orologio in tempo reale con data (alimentazione a batteria).
Sistema operativo. MS-DOS 4.1, GW BASIC Video.
Espansione. 3 slot di espansione compatibili con lo standard industriale, completamente disponibili per schede di espansione, schede di rete, ecc.

DATI TECNICI PC 40-3

CPU. CPU 80286, 16 bit; Clock commutabile 6/8/12MHz; Coprocessore matematico 80287 (opzionale).
Memoria. RAM standard da 1Mb, di cui 640Kb per MS-DOS e 340Kb come disco RAM; Unità a dischetti da 5 1/4", 1,2Mb; Unità a disco da 3 1/2", 760Kb (opzionale); Unità a disco fisso da 40Mb, 20 ms.
Interfacce. Interfaccia seriale RS232C; Interfaccia parallela Centronics; Interfaccia per mouse, (Microsoft Bus-Mouse compatibile per il collegamento con il mouse 1352 Commodore); Interfaccia video.
Espansione. 4 slot di espansione conformi allo standard industriale, 2 disponibili per schede di espansione con capacità di memoria 8 o 16Mb, o schede per il collegamento in rete.
Adattatore video. Adattatore grafico VGA integrato con funzioni VGA ottimizzate; 256 colori emulati contemporaneamente da una tavolozza da 262144, MDA, CGA, Hercules, EGA; Frequenza max. di quadro 70 Hz; Risoluzione max. 924 x 400 punti, (modo monocromatico).
Monitor. Monitor monocromatico da 14", visualizzazione analogica, con possibilità di rotazione e inclinazione.
Tastiera. Tastiera DIN, 102 tasti; Tasto di Reset.
Orologio. Orologio/data in tempo reale (alimentazione a batteria).
Sistema operativo. MS-DOS 4.1 GW Basic
Interruttore a chiavetta

Le specifiche tecniche sono suscettibili di modifiche senza preavviso

Inoltre vi proponiamo le nostre offerte speciali promozione luglio-settembre per apertura filiali

Per AMIGA 500

Hard disk 20Mb autobooting FFS - DMA GVP espandibile internamente 4Mb RAM L. 1.800.000
+ tavola grafica Easel valore di listino L. 2.050.000

Come sopra ma hard disk 40Mb L. 1.990.000
valore di listino L. 2.250.000

Digitalizzatore audio o video per AMIGA - Easy Wiew - Easy Sound L. 99.000

OFFERTE VARIE DA NON PERDERE

Modem 300/1200 Videotel per Commodore 64	L. 49.000
Espansione di memoria 512K per AMIGA	L. 130.000
Come sopra con clock	L. 160.000
Portatili PC XT 8086 Bondwell 512/640K, doppio drive 3 1/2 720K con modem integrato 300/1200/Baud, schermo al plasma	L. 1.590.000
Portatili 286 1Mb, HD 40Mb, schermo al plasma da	L. 3.990.000
Schede GVP 68030 + 6882 28MHz a partire da	L. 1.300.000

Sono in offerta speciale di svendita una grossa quantità e tipi di cartucce per Commodore 64

Smart Card 32K ROM, in tampone	L. 99.000
Stard OS velocizzatore per drive 64	L. 20.000
ROM 801 caratteri discendenti	L. 10.000
ROM 802 caratteri grafici	L. 15.000
ROM 803 caratteri vari	L. 25.000
Interfaccia MIDI 64	L. 70.000
CPM Emulator 64 con software	L. 29.000
Modem 300/1200 accoppiatore acustico	L. 30.000

E tante, tante altre offerte!!
Passate a trovarci a Milano in Via Mac Mahon, 75
Oppure ordinate direttamente a:
3270226 Servizio Tecnico Ricezione Ordini
93580086 Servizio Automatico Ricezione Ordini
0377..... Prossima Installazione Ski Line

COMPUMAIL®
GRUPPO NEWEL MI

- 20020 ARESE (MI) - VIA MATTEOTTI, 21 -

VENDITA PER CORRISPONDENZA SU RETE NAZIONALE

NON SI VENDE AL PUBBLICO. I PRIVATI DEVONO RIVOLGERSI IN VIA MAC MAHON, 75 - MILANO
TEL/FAX 93580086, RICEZIONE AUTOMATICA ORDINI