Università di Napoli Federico II - Facoltà di Ingegneria - Corso di Laurea in Ingegneria Informatica

Corso di Programmazione I

Gestione delle eccezioni

Concetti base

Il problema

- In alcuni casi, chi scrive una libreria può rilevare errori a run-time, ma non sa come gestirli
- Chi usa la libreria, invece, sa come gestire tali errori ma non può rilevarli
- Soluzioni ad-hoc possono essere definite di volta in volta
- È necessario un meccanismo generale, semplice da usare ed elegante per gestire situazioni “eccezionali”

Esempio: overflow a seguito di somma

mylib.cpp

```
#include <stdlib.h>
#include <climits>

short somma (short x, short y)
{
 // check for overflow
 if ((x>0 && y>0 && x>SHRT_MAX-y) ||
     (x<0 && y<0 && x<SHRT_MIN-y))
   ??? // cosa fare ?

 return x + y;
}
```

main.cpp

```
#include <climits>
#include <iostream>

using namespace std;

short somma (short x, short y);

int main()
{
      cout << somma(30000, 3000) << endl;
}
```

Soluzione 1: termina il programma

mylib.cpp

```
#include <stdlib.h>
#include <climits>

short somma (short x, short y)
{
 // check for overflow
 if ((x>0 && y>0 && x>SHRT_MAX-y) ||
     (x<0 && y<0 && x<SHRT_MIN-y))
   exit(0);

 return x + y;
}
```

main.cpp

```
#include <climits>
#include <iostream>

using namespace std;

short somma (short x, short y);

int main()
{
      cout << somma(30000, 3000) << endl;
}
```

- Vogliamo gestire l'eccezione, non terminare il processo

Soluzione 2: restituisce un valore di errore

- Non sempre possibile
  - Nessun valore del tipo short può essere usato per indicare una situazione eccezionale
- Si può utilizzare una variabile booleana passata per riferimento
  - Diventa complicato quando possono presentarsi diverse situazioni eccezionali

Soluzione 3: restituisce un valore legale e setta una variabile globale

mylib.cpp

```
#include <stdlib.h>
#include <climits>
#include <errno.h>

short somma (short x, short y)
{
 // check for overflow
 if ((x>0 && y>0 && x>SHRT_MAX-y) ||
     (x<0 && y<0 && x<SHRT_MIN-y))
   errno=10;

 return x + y;
}
```

main.cpp

```
#include <climits>
#include <iostream>
#include <errno.h>
using namespace std;

short somma (short x, short y);

int main()
{
      somma(30000, 3000);
      if (errno == 10) …
}
```

- Cosa succede se il chiamante non controlla **errno**?

Il meccanismo di gestione delle eccezioni

- Fornisce un'alternativa alle tecniche tradizionali quando queste sono insufficienti e prone ad errori
- L'idea è che una funzione che trova un errore che non sa gestire *lancia* (throw) un'eccezione, nella speranza che il suo chiamante (diretto o indiretto) possa gestire il problema
- Consente di separare il codice per la gestione dell'eccezione dal codice "ordinario"
  - Il programma diventa più leggibile

Gestione delle eccezioni

- Una funzione che vuole gestire un certo tipo di eccezione (handler) può farlo indicando che intende *catturare* (catch) quel tipo di eccezione
- Se viene lanciata un'eccezione e ripercorrendo a ritroso la catena di chiamanti non si incontra nessuna funzione che cattura l'eccezione, il programma viene terminato

Gestione delle eccezioni

mylib.cpp

```
#include <climits>

short somma (short x, short y)
{

 // check for overflow
 if ((x>0 && y>0 && x>SHRT_MAX-y) ||
     (x<0 && y<0 && x<SHRT_MIN-y))
   throw "Overflow";

 return x + y;
}
```

- La funzione somma *lancia* un'eccezione al verificarsi di una situazione che non sa gestire
  - throw [*oggetto*]
- Ovviamente, throw fa terminare l'esecuzione della funzione somma

Gestione delle eccezioni

main.cpp

```
#include <climits>
#include <iostream>
using namespace std;
short somma (short x, short y);
int main()
{
     try {
       short ris = somma(30000, 3000);
       cout << "La somma è " << ris;
     }
     catch (const char* errstr) {
       cout << errstr << endl;
     }
}
```

- Il costrutto **catch()** *(exception handler)* può essere usato solo dopo un blocco preceduto dalla keyword **try** o dopo un altro blocco catch()
- catch() ha tra parentesi una dichiarazione *simile* a quella degli argomenti di una funzione

Gestione delle eccezioni

main.cpp

```
#include <climits>
#include <iostream>
using namespace std;
short somma (short x, short y);
int main()
{
    try {
      short ris = somma(30000, 3000);
      cout << "La somma è " << ris;
    }
    catch (const char* errstr) {
      cout << errstr << endl;
    }
}
```

- Se una funzione nel blocco try lancia un'eccezione
  - Le istruzioni nel blocco try seguenti tale funzione non vengono eseguite
  - Viene eseguito (se esiste) solo il primo handler trovato per il tipo di eccezione lanciata

Cattura delle eccezioni

```
try {
    // codice che lancia un'eccezione di tipo E;
}
catch (H) {
    // quando arriviamo qui?
}
```

1) H è dello stesso tipo di E (o una sua classe base)
2) H ed E sono di tipo puntatore ed 1) vale per i tipi puntati
3) H è un riferimento ed 1) vale per il tipo a cui H si riferisce

Gestione delle eccezioni

- **catch (…)** può essere usato per catturare qualunque tipo di eccezione
- Se posto in testa ad una lista di handler fa sì che gli altri handler non vengano mai considerati
  - Il compilatore g++ (versione 4.3) segnala errore se catch (…) non è l'ultimo handler della lista

Rilancio di un'eccezione

- Se un handler non è in grado di gestire completamente un'eccezione, può *rilanciarla*, assumendo che un altro handler possa completarne la gestione

```
try {
   // codice che può lanciare un'eccezione di tipo E
}
catch (E) {
   // fa qualcosa
   throw;  // rilancia l'eccezione originaria
}
```

Specificazione delle eccezioni

- La dichiarazione di una funzione può contenere la lista di eccezioni che possono essere lanciate da tale funzione

  short somma (short x, short y) throw (const char*);
- Il vantaggio è che la dichiarazione appartiene ad un'interfaccia che è visibile al chiamante
- Nota: se somma lancia un'eccezione diversa da const char*, viene chamata abort() che termina il programma
- Per specificare che una funzione non lancia eccezioni:

  int g () throw ();

L'operatore new e le eccezioni

memleak.cpp

```
#include <iostream>

using namespace std;

int main()
{
        int n = 1000000;
        int * vett;
        while (true) {
          vett = new int[n];
        }
        return 0;
}
```

- Quando la memoria heap è esaurita, il programma termina con il messaggio del tipo:

```
terminate called after throwing an
    instance of 'std::bad_alloc'
  what():  std::bad_alloc
Aborted
```

- Cosa è successo?

L'operatore new e le eccezioni

- Dall'header file <new> :

```
void* operator new(std::size_t) throw (std::bad_alloc);
void* operator new[](std::size_t) throw (std::bad_alloc);
void operator delete(void*) throw();
void operator delete[](void*) throw();
```

- new può lanciare un'eccezione di tipo std::bad_alloc
- delete non lancia alcuna eccezione

L'operatore new e le eccezioni

memleak_catch.cpp

```
#include <iostream>
using namespace std;
int main()
{
        int n = 1000000;
        int * vett;
   while (true) {
    try {
      vett = new int[n];
    }
    catch (bad_alloc e) {
      cout << "Memoria insufficiente" << endl;
      break;
    }
   }
   cout << "Il programma puo' continuare...";
   return 0;
}
```

- L'eccezione lanciata da new viene gestita facendo terminare il ciclo while

Memoria insufficiente
Il programma puo' continuare...

Un'altra versione dell'operatore new

- Dall'header file <new> :

```
struct nothrow_t { };
extern const nothrow_t nothrow;

void* operator new(std::size_t, const std::nothrow_t&) throw();
void* operator new[](std::size_t, const std::nothrow_t&) throw();
```

- new può essere invocato in modo da non lanciare alcuna eccezione
- new restituisce un puntatore NULL se non riesce ad allocare la memoria richiesta

L'operatore new e nothrow

memleak_nothrow.cpp

```
#include <iostream>
using namespace std;
int main()
{
   int n = 1000000;
   int * vett;
   while (true) {
     vett = new (nothrow) int[n];
    if (!vett) {
      cout << "Memoria insufficiente" << endl;
      break;
    }
     else cout << "allocati " <<  n*sizeof(int) << " byte" <<
endl;
   }
   cout << "Il programma puo' continuare...";
   return 0;
}
```

- new non lancia un'eccezione ma restituisce un puntatore nullo

```
Memoria insufficiente
Il programma puo' continuare...
```

Esercizio

- Scrivere una funzione che effettui la divisione di due numeri reali forniti in ingresso e lanci una eccezione in caso di divisione per zero.
- Scrivere un programma chiamante che invochi la funzione e preveda un handler per la gestione dell'eccezione