UNIVERSITÀ
DI PARMA

eccezioni

Alberto Ferrari

- C++ fornisce strumenti per gestire ***situazioni eccezionali***
- terminologia
  - ***sollevare*** un'eccezione (to ***throw*** an exception) = ***segnalare*** una situazione eccezionale
  - ***intercettare*** l'eccezione (to ***catch*** or ***handle*** the exception) = ***catturare*** o ***gestire*** la situazione eccezionale

esempio senza eccezioni

```
#include <iostream>
using namespace std;
int main( ) {
    int num, den;      // numeratore e denominatore
    double val;        // valore della frazione
    cout << "numeratore: ";   cin >> num;
    cout << "denominatore: "; cin >> den;
    if (den == 0)
        cout << "denominatore = 0" << endl;
    else {
        val = num/static_cast<double>(den);
        cout << "valore della frazione: " << val << endl;
    }
    return 0;
}
```

costrutto try – throw - catch

```
try
{
   Some_Statements
   if (Exceptional_Case)
     throw exception;
   Some_More_Statements
}
catch(Type e)
{
  Code to be performed if a value of type Type is thrown in the try block
}
```

blocco try

- contiene il codice per gestire le ***situazioni normali***
- può ***riconoscere*** e ***segnalare*** situazioni speciali ***sollevando eccezioni***
- se ***non*** si verificano ***eccezioni***, l'esecuzione del blocco try è quella ***standard***
- è buona norma inserire in un blocco ***try*** il ***solo codice*** che potenzialmente può ***sollevare un'eccezione***

istruzione throw

- è l'istruzione usata per "***lanciare***" un ***valore*** detto ***eccezione***
- il ***valore*** lanciato può essere di ***qualsiasi tipo***
- l'esecuzione del blocco ***try termina*** e il ***controllo*** passa a un blocco ***catch***

blocco catch

- contiene il ***codice*** per ***gestire*** la situazione eccezionale
- ha un ***parametro*** che
  - specifica quale ***tipo*** di ***valore*** può essere ***intercettato*** dal blocco
  - consente di ***utilizzare*** il ***valore*** intercettato all'interno del blocco
- se non viene sollevata ***nessuna*** eccezione l'esecuzione del blocco try viene completata e il blocco catch viene ignorato
- Una volta completato il blocco catch, viene eseguito il codice che segue
- Un blocco catch risponde solo a un blocco try immediatamente precedente
- Se non c'è un blocco catch del tipo opportuno il programma termina

esempio con eccezioni

```
#include <iostream>
using namespace std;
int main( ){
    int num, den;      // numeratore e denominatore
    double val;        // valore della frazione
    try {
        cout << "numeratore: ";   cin >> num;
        cout << "denominatore: "; cin >> den;
        if (den == 0)
                throw den;
        val = num/static_cast<double>(den);
        cout << "valore della frazione: " << val << endl;
    } catch(int e) {
         cout << "denominatore = " << e << endl;
    }
    return 0;
}
```

classi di eccezioni

- sono classi i cui oggetti contengono l'***informazione*** che si vuole lanciare al blocco ***catch***
- in questo modo si ottiene un ***diverso tipo*** per ***ogni*** possibile ***situazione eccezionale***
- può essere ***utile*** definire una ***gerarchia*** di classi di eccezioni

eccezioni multiple

- un ***blocco*** try può potenzialmente sollevare più ***eccezioni*** di ***tipi diversi***
- in ogni esecuzione verrà sollevata al massimo una eccezione
- ogni ***blocco catch*** può intercettare valori di ***un solo tipo***
- si possono avere ***più blocchi catch*** dopo un blocco try per gestire eccezioni di ***tipo diverso***

blocco catch di default

- quando in un blocco try viene sollevata un'eccezione, i blocchi ***catch*** che seguono sono considerati ***in ordine***, viene eseguito il ***primo*** che intercetta quel tipo di eccezione
- blocco catch ***speciale*** che intercetta ogni tipo di eccezione, da usare come ***default***
- `catch(…) { cout << "Unexplained exception";}`

sollevare un'eccezione in una funzione

- spesso è utile ***ritardare la gestione*** di un'eccezione
- una funzione può ***sollevare*** un'eccezione e ***non intercettarla***
- Sarà il programma che usa la funzione a gestire l'eccezione
- il programma metterà la ***chiamata*** della funzione in un blocco ***try*** seguito da un blocco ***catch*** che intercetta l'eccezione

esempio

```
class DivideByZero {};
double safeDivide(int top, int bottom) throw (DivideByZero);
int main( ){
    …
  try {
    quotient = safeDivide(numerator, denominator);
  } catch(DivideByZero) {
    cout << "Error: Division by zero! " << "Program aborting";
    exit(0);
  }
}
double safeDivide(int top, int bottom) throw (DivideByZero) {
    if (bottom == 0)
        throw DivideByZero( );
    return top/static_cast<double>(bottom);
}
```

specifica delle eccezioni

- elenca le ***eccezioni*** che possono essere sollevate da una funzione e ***non*** vengono da essa ***intercettate***
- ***deve*** apparire sia nella ***dichiarazione*** che nella ***definizione*** della funzione
- se viene sollevata un'eccezione che non viene intercettata e non compare nella specifica, viene chiamata la funzione ***unexpected*** che per default termina il programma, ma può essere ***ridefinita***

quando sollevare un'eccezione

- se la funzione è in grado di ***gestire in modo semplice*** il caso speciale, ***non*** deve ***sollevare*** l'eccezione
- se il modo in cui il caso speciale va gestito ***dipende*** da dove la funzione è ***usata***, si ***delega*** la gestione al ***livello superiore*** (le eccezioni non intercettate risalgono di scope)
- ***non*** usare le ***eccezioni*** nei ***distruttori***

esempio allocazione dinamica


```
#include <new>
using std::bad_alloc;

try
{
   int *p = new int[100];
}
catch(bad_alloc)
{
   cout << "Cannot alloc p";
    …
}
```

abuso nell'uso di eccezioni

- la gestione delle eccezioni genera ***overhead*** sia *temporale* che *spaziale*
- le istruzioni ***throw*** rendono ***contorto*** il flusso di controllo
- la gestione delle eccezioni va usata con ***moderazione***

eccezioni: vantaggi

- se il linguaggio che non supporta le eccezioni si può restituire un codice di errore
  - occorre ***controllarlo*** ogni volta
  - il programma può ***ignorarlo***
  - alcune funzioni non possono restituire un codice di errore
- in C++
  - gestione ***uniforme*** delle eccezioni per tutte le funzioni
  - una eccezione ***non*** può essere ***ignorata***
  - ***non*** si ***mescola*** la gestione dei casi ***speciali*** e dei casi ***normali***