

Programmazione Avanzata

Eccezioni

Pierluigi Roberti

`pierluigi.roberti@unitn.it`

Motivazione

- gli errori di runtime sono quegli errori che NON possono essere rilevati in fase di compilazione, perché si manifestano solo durante la fase di esecuzione del programma e solo in alcune particolari circostanze, ossia ai veri casi di “eventi eccezionali”.
- Alcuni esempi di errori di runtime sono:
 - divisione per 0;
 - l'utilizzo di un pathname non valido o che fa riferimento ad una periferica che in un certo momento risulta scollegata;
 - overflow su un tipo di dato da parte di un'operazione aritmetica;
 - un'operazione di casting non valido (ad esempio, quando si ha l'inserimento in input da parte dell'utente di una stringa di testo non interpretabile come un numero e che viene assegnata ad una variabile di tipo numerico).

Motivazione

- Si tratta di errori pericolosi perché se non gestiti, possono generare anomalie di funzionamento che possono determinare persino il blocco inaspettato del programma (crash dell'applicazione). La gestione di questo tipo di errori non è banale ed è per questo che per facilitarla i linguaggi di programmazione ad oggetti (OOP) mettono a disposizione il meccanismo delle **ECCEZIONI**.
- Quando si verifica un errore di runtime i linguaggi OOP, si dice, possono “sollevare” o permettono di “lanciare” (in inglese, to **throw**) un'eccezione. Ciò in generale si traduce nella creazione di un oggetto che appartiene ad una classe particolare, dipendente dallo specifico errore di runtime che si è verificato.
- Questo meccanismo, infatti, prevede che quando viene sollevata un'eccezione, il flusso di esecuzione del programma viene sospeso nel punto in cui si è verificato l'errore e salta in un altro punto del codice in cui esso viene gestito. Nel linguaggio C++ ciò viene realizzato utilizzando il costrutto **try-catch**.

Sintassi

```
try{  
    //istruzioni a rischio errori di runtime  
} catch (TipoErrore1){  
    //istruzioni che gestiscono le eccezioni di tipo 'TipoErrore1'  
}
```

- **try** è la parola chiave che introduce il blocco in cui si possono inserire le istruzioni che si vuole tenere “sotto controllo” perché durante l’esecuzione potrebbero generare degli errori di runtime.
- **catch** è la parola chiave che introduce un blocco in grado di riconoscere e “catturare” un certo tipo di eccezione e che contiene le istruzioni per gestire l’errore di runtime che l’ha generata.
- Un **blocco try** può avere ad esso associati **uno o più blocchi catch**: quando all’interno di un blocco try viene sollevata o lanciata un’eccezione, il flusso di esecuzione del programma salta ai suoi blocchi catch, partendo dal primo e proseguendo verso gli altri finché non si trova un blocco catch (se esiste) che cattura il tipo di eccezione sollevata. Quando si entra in un blocco catch, alla sua uscita il flusso del programma prosegue oltre l’ultimo blocco catch.

Sintassi

Un'eccezione può essere o sollevata direttamente dall'istruzione in corrispondenza della quale si verifica l'errore, oppure se per una certa istruzione ciò non è previsto dal linguaggio, può essere lanciata esplicitamente con un'istruzione **throw** con la seguente sintassi:

```
throw espressione;
```

throw è la parola chiave con cui è possibile lanciare un'eccezione per segnalare il fatto che si è verificato un errore.

L'eccezione viene “marcata” opportunamente, per permettere il suo riconoscimento da parte dei **blocchi catch**.

Esempio 1

```
double x, y;  
try {  
    cout<<"Inserisci il dividendo: ";  
    cin>>x;  
    cout<<"Inserisci il divisore: ";  
    cin>>y;  
    //monitora il verificarsi dell'errore di runtime di divisione per zero  
    if(y==0) {  
        //lancia un'eccezione di tipo 'costante stringa'  
        throw "Divisione per zero!";  
    }  
    cout<<x/y<<endl;  
}  
//individua e cattura un'eccezione di tipo 'costante stringa'  
catch (const char* messaggio) {  
    cerr<<messaggio<< endl;  
}
```

Inserisci il dividendo: 12
Inserisci il divisore: 4
2.4

Inserisci il dividendo: 6
Inserisci il divisore: 0
Divisione per zero!

Propagazione delle Eccezioni

- Un **blocco try/catch** viene utilizzato per rilevare le eccezioni.
- Il codice nella sezione try è il codice che può generare un'eccezione e il codice nella clausola catch gestisce l'eccezione.
- È possibile utilizzare più clausole di catch per gestire più tipi di eccezioni. Se sono presenti più clausole catch, il meccanismo di gestione delle eccezioni tenta di farli corrispondere in ordine di aspetto nel codice.
- Un'eccezione non necessariamente deve essere gestita all'interno della stessa funzione in cui viene sollevata o lanciata, ma al contrario è opportuno che questa gestione avvenga al suo esterno, all'interno di una funzione chiamante.
- Spesso, infatti, accade che la sua gestione dipende e può essere diversa a seconda del punto in cui quella funzione viene chiamata. Questo è reso possibile dal meccanismo proprio delle eccezioni, il quale prevede che un'eccezione sollevata in una funzione e che non venga in essa gestita, sia propagata all'indietro verso il chiamante, si dice risalendo lo stack delle chiamate.

Esempio 2a

```
int compare(int a, int b) {
    if (a < 0 || b < 0) { //individua l'errore di runtime
        //lancia l'eccezione utilizzando una stringa
        throw "ERR a or b negative";
    }
    return a == b;
}

void test1() {
    try {
        cout << compare(-1, 0) << endl;
        /*una volta sollevata una eccezione il programma
        non prosegue con le istruzioni successive*/
        cout << "FINE PROGRAMMA" << endl;
    } catch (const char* errorMessage) {
        //intercetta un'eccezione di tipo costante stringa
        cout << errorMessage << endl;
    }
}

int main(int argc, char *argv[]) {
    test1();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```


Esempio 2b

```
int compare(int a, int b) {
    if (a < 0 || b < 0) { //individua l'errore di runtime
        //lancia l'eccezione utilizzando una stringa
        throw "ERR a or b negative";
    }
    return a == b;
}

void test2() {
    /*attenzione non è gestito il catch dell'errore e
    questo blocca l'esecuzione del programma*/
    cout << compare(-1, 0) << endl;
}

int main(int argc, char *argv[]) {
    test2();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Esempio 2c

```
int compareADV(int a, int b) {
    if (a < 0 ) {
        throw "ERR a negative";
    }
    if (b < 0) {
        //soluzione per ritornare una "stringa" con dettagli sul valore errato
        string message = "ERR b negative: ";
        cerr << message << b << endl;
    }
    return a == b;
}

void test3() {
    try {
        cout << compare(5, 5) << endl;
        cout << "FINE Confronto 1" << endl;
        //eccezione con messaggio errore personalizzato
        cout << compareADV(6, -3) << endl;
        /*una volta sollevata una eccezione il programma
        non prosegue con le istruzioni successive*/
        cout << "FINE Confronto 2" << endl;
    } catch (const char* errorMessage) {
        cout << errorMessage << endl;
    }
}

int main(int argc, char *argv[]) {
    test3();  system("PAUSE");  return EXIT_SUCCESS;
}
```

Esempio 3 – eccezioni multiple

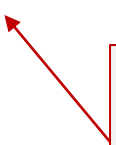
Poiché le clausole di catch vengono provate in ordine, assicurarsi di scrivere prima clausole di cattura più specifiche, altrimenti il codice di gestione delle eccezioni potrebbe non essere mai chiamato.

NOTA: possibile ordine catch diverso da ordine throw


```
//divisione per soli valori interi positivi
int divideINT(int a, int b) {
    if (a < 0 || b < 0) {
        throw invalid_argument( "received negative value" );
    }
    return a/b;
}

void test1() {
    int a = 5, b = -6;
    try {
        cout << a << "/" << b << "=" << divideINT(a, b) << endl;
    } catch( const invalid_argument& e ) {
        //cout << e << endl; //bind ERROR
        cout << "a or b negative" << endl;
    }
}

int main(int argc, char *argv[]) {
    test1(); system("PAUSE"); return EXIT_SUCCESS;
}
```



#include <stdexcept>
using namespace std;
<http://www.cplusplus.com/reference/stdexcept/>



- domain_error
- invalid_argument
- length_error
- logic_error
- out_of_range
- overflow_error
- range_error
- runtime_error
- underflow_error

Esempio 3b

```
//divisione per soli valori interi positivi con valore massimo sotto il 10
int divideINT2(int a, int b) {
    if (a < 0 || b < 0) { throw invalid_argument( "received negative value" ); }
    if (b==0){ throw overflow_error( "division by 0" ); }
    if (a > 9 || b > 9) { throw out_of_range( "out of range value" ); }
    return a/b;
}

void test2() {
    int a = 7, b = 5; // => ok
    //a=5 & b=0  => overflow_error
    //a=15 & b=5 => out_of_range
    try {
        cout << a << "/" << b << "=" << divideINT2(a, b) << endl;
    } catch( const invalid_argument& e ) {
        cout << e.what() << endl;
        cout << "a or b negative" << endl;
    } catch( const out_of_range& e ) {
        //cout << e << endl; //bind ERROR
        cout << "a or b greather than 9" << endl;
        //return;
    } catch( const overflow_error& e ) {
        //cout << e << endl; //bind ERROR usare cout << e.what() << endl;
        cout << "b nullo" << endl;
    }
    //intercettato l'errore posso continuare con il mio codice
    cout << "prossima istruzione"<<endl;
}
```

Esempio 3c

```
//divisione per soli valori interi positivi con valore massimo sotto il 10
int divideINT2(int a, int b) {
    if (a < 0 || b < 0) { throw invalid_argument( "received negative value" ); }
    if (b==0){ throw overflow_error( "division by 0" ); }
    if (a > 9 || b > 9) { throw out_of_range( "out of range value" ); }
    return a/b;
}
/*
Un'altra possibilità è il gestore catch-all,
che catturerà qualsiasi oggetto lanciato
*/
void test3() {
    int a = 15, b = 5;
    //a=7 & b=5 => ok
    //a=5 & b=0 => overflow_error
    //a=15 & b=5 => out_of_range
    try {
        cout << a << "/" << b << "=" << divideINT2(a, b) << endl;
    } catch( ... ) {
        cout << "errore nei dati" << endl;
    }
    //intercettato l'errore posso continuare con il mio codice
    cout << "prossima istruzione"<<endl;
}
```

Esempio 3d

```
//divisione per soli valori interi positivi con valore massimo sotto il 10
int divideINT2(int a, int b) {
    if (a < 0 || b < 0) { throw invalid_argument( "received negative value" ); }
    if (b==0){ throw overflow_error( "division by 0" ); }
    if (a > 9 || b > 9) { throw out_of_range( "out of range value" ); }
    return a/b;
}
/*
Un'altra possibilità è il gestore catch-all,
che catturerà qualsiasi oggetto lanciato
*/
void test3() {
    int a = 15, b = 5;
    //a=7 & b=5 => ok
    //a=5 & b=0 => overflow_error
    //a=15 & b=5 => out_of_range
    try {
        cout << a << "/" << b << "=" << divideINT2(a, b) << endl;
    } catch( exception &e ) {
        cout << e.what() << endl;
    }
    //intercettato l'errore posso continuare con il mio codice
    cout << "prossima istruzione"<<endl;
}
```

Le eccezioni nelle classi

```
class Test {  
private:  
    unsigned int value;  
public:  
    Test(int v) {  
        try {  
            if (v < 0) {  
                throw "ERR negative value";  
            }  
            value = v;  
            cout << "class initialized with value:" << value << endl;  
        }  
        catch (...) {  
            // exceptions from the initializer list and constructor are caught here  
            cout << "wrong data" << endl;  
        }  
    }  
};  
  
int main(int argc, char *argv[]) {  
    Test var1(3);  
    Test var2(-6);  
    system("PAUSE");  
    return EXIT_SUCCESS;  
}
```

```
#include <stdexcept>  
using namespace std;  
http://www.cplusplus.com/reference/stdexcept/
```