XỬ LÝ LỖI

Bắt đầu từ phiên bản 3.0, C++ cung cấp cơ chế xử lý lỗi (exception handling) do người sử dụng điều hành. Trong chương trình có thể đưa ra quyết định rằng một phép xử lý nào đó bị lỗi bằng cách sử dụng từ khoá **throw**. Khi có lỗi, việc xử lý bị ngắt và quyền điều khiển sẽ trao trả cho đoạn xử lý lỗi mà người sử dụng bắt được.

1. BẪY VÀ BẮT LỖI

Ta lấy ví dụ viết một hàm tính giá trị của một phân số với đầu vào là tử số và mẫu số. Rắc rối sẽ nảy sinh khi người sử dụng hàm truyền vào cho mẫu số giá trị bằng 0. Để giải quyết trường hợp này, C++ sẽ tự động tạo sinh bẫy lỗi "gặp trường hợp mẫu số bằng 0". Sau đây là chương trình ví dụ cho hàm tính giá trị phân số có xử lý lỗi.

```
#include <iostream.h>
// lớp kiểu lỗi mà không có thành phần nào
class Loi_Chia_0 {};
float GiaTriPS(int ts, int ms){
        // phát lỗi nếu mẫu = 0
        if ( ms==0 ) throw( Loi_Chia_0 );
        return float(ts)/ms;
}
void main(){
        int ts, ms;
        cout << "Tinh gia tri phan so\n";
        cout << "TS = "; cin >> ts;
        cout << "MS = "; cin >> ms;
        try    {       // thực hiện có bắt lỗi
                float gt = GiaTriPS(ts, ms);
                cout << "Gia tri PS la: " << gt;
        }
```

```
        catch ( Loi Chia 0 ) // bắt lỗi kiểu Loi_Chia_0
        {
                cout << "Loi: Mau so bang 0";
        }
}
```

```
Tinh gia tri phan so
TS = 1
MS = 0
Loi: Mau so bang 0
```

Chương trình này có hai phần bẫy lỗi và bắt lỗi. Hàm tính phân số sẽ phát sinh một bẫy lỗi bằng từ khoá **throw** khi mẫu số bằng 0. Phía sau từ khoá throw là đối tượng thuộc lớp phục vụ bắt lỗi. Ở đây ta sử dụng lớp *Loi_Chia_0* không có thành phần nào bên trong để phục vụ cho việc bắt lỗi này. Khi một lỗi được phát sinh, toàn bộ những lệnh tiếp theo trong hàm xử lý bị huỷ bỏ. Trong thân chương trình chính chúng ta sử dụng cấu trúc **try... catch...** để bắt lỗi. Hàm `GiaTriPS` được đặt trong **try**, do vậy khi nó phát sinh lỗi (lỗi loại Loi_Chia_0 được phát sinh khi mẫu số truyền vào là 0) thì chương trình dừng hoạt động và trao quyền điều khiển cho đoạn mã bắt lỗi này. Ta đã dùng **catch** *(Loi_Chia_0)* để bắt lỗi. Như vậy, khi có một lỗi chia 0 thì chương trình sẽ in ra dòng thông báo "Loi: Mau so bang 0". Khi trong chương trình có một lỗi phát sinh mà không có đoạn bắt lỗi tương ứng thì chương trình sẽ tự động kết thúc bất thường. Điều này sẽ gây trở ngại đáng kể cho việc gỡ rối chương trình.

Một chương trình có thể phát sinh nhiều loại lỗi khác nhau. Loại lỗi phát sinh thể hiện ở kiểu lớp lỗi được sử dụng trong các lệnh **throw**. Do vậy, ta cũng có thể sử dụng nhiều lần **catch** để bắt các loại lỗi khác nhau trong một chương trình như trong ví dụ sau.

```
#include <iostream.h>
class Loi_A {};
class Loi_B {};
void PhatLoi(int i){
// neu i khac 0 thi phat Loi_A con khong Loi_B
if (i) throw ( Loi_A );
throw ( Loi_B );
```

```
}
void main(){
	int i;
	cout << "i = "; cin >> i;
	try
	{
		PhatLoi( i );
	}
	catch ( Loi_A )
	{
		cout << "Loi loai A";
	}
	catch ( Loi_B )
	{
		cout << "Loi loai B";
	}
}
```

```
i = 1
Loi loai A

i = 0
Loi loai B
```

Ta cũng có thể sử dụng `catch(...)` để bắt tất cả các loại lỗi. Lớp lỗi có thể có các thành phần giống như lớp bình thường dùng để làm các thông số xử lý lỗi khi bắt được. Chẳng hạn, trong chương trình xử lý lỗi chia 0 ở trên ta có thể thêm vào thuộc tính để lưu lại tử số của phép chia lỗi dùng in ra khi bắt lỗi.

```
#include <iostream.h>
class Loi_Chia_0{
public:
	int ts;
```

```
        Loi Chia 0(int t): ts(t) {}
};
float GiaTriPS(int ts, int ms){
        if ( ms==0 ) throw( Loi_Chia_0(ts) );
        return float(ts)/ms;
}
void main(){
        int ts, ms;
    cout << "Tinh gia tri phan so\n";
        cout << "TS = "; cin >> ts;
        cout << "MS = "; cin >> ms;
        try    {
                float gt = GiaTriPS(ts, ms);
                cout << "Gia tri PS la: " << gt;
        }
        catch ( Loi_Chia_0 loi )
        {
                cout << "Loi chia " << loi.ts << " cho 0";
        }
}
```

```
Tinh gia tri phan so
TS = 1
MS = 0
Loi chia 1 cho 0
```

2. Hoạt động của chương trình khi một lỗi phát sinh

Khi một lỗi trong chương trình đã bị bắt thì chương trình tiếp tục hoạt động bình thường theo mã lệnh xử lý lỗi bắt được. Trong một hàm xử lý người sử dụng có thể bắt lỗi xảy ra và sau đó tiếp tục ném nó để duy trì lỗi của chương trình bằng từ khoá **throw** mà không có kiểu loại lỗi phía sau.

```
#include <iostream.h>
```

```
class Loi {};
void PhatLoi(){
throw ( Loi );
}
void BatLoi(){
      try   {
            PhatLoi();
      }
      catch ( Loi )      {
            cout << "Loi da bi bat va duoc nem lai\n";
            throw; // ném lại lỗi bị bắt
}
}
void main() {
      try   {
            BatLoi();
      }
      catch ( Loi )      {
            cout << "Loi bi bat lai lan hai";
      }
}
```

```
Loi da bi bat va duoc nem lai
Loi bi bat lai lan hai
```

Khi một lỗi xảy ra mà không có một bắt lỗi nào đáp ứng thì chương trình sẽ kết thúc và trước khi kết thúc nó sẽ thực hiện hàm xử lý được xác lập trong câu lệnh `set_terminate`.

```
#include <iostream.h>
class Loi {};

void KetThucLoi(){
```

```
        cout << "Chuong trinh bi loi va ket thuc bat thuong\n";
}
void PhatLoi(){
throw ( Loi );
}
void main(){
        // xác lập hàm kết thúc bất thường
        set_terminate(KetThucLoi);
        cout << "Goi ham phat loi ma khong bat\n";
        PhatLoi();
        cout << "Ket thuc binh thuong\n";
}
```

```
Goi ham phat loi ma khong bat
Chuong trinh bi loi va ket thuc bat thuong
```

Trong một hàm xử lý người xử dụng có thể xác định tất cả những lỗi có thể xảy ra bằng cách dùng từ khoá **throw** đứng ngay sau khai báo tham số hàm và liệt kê những lớp lỗi có thể xảy ra để trong cặp dấu ( ). Nếu không có lớp lỗi nào được liệt kê thì hiểu rằng hàm đó sẽ không có lỗi. Còn nếu không có từ khoá **throw** thì hàm đó có thể có bất kì lỗi nào.

```
// hàm có thể có Lỗi_A hoặc Lỗi_B
void fct1() throw(Loi_A, Loi_B);
// hàm sẽ không có lỗi nào
void fct2() throw();
// hàm có thể có bất kì loại lỗi nào
void fct3();
```

Trường hợp với một hàm nào đó tuy đã được xác định một số lỗi có thể xảy ra, nhưng khi chạy lại xuất hiện một lỗi không phải là lỗi đã xác định, thì chương trình sẽ kết thúc. Trước khi kết thúc nó sẽ thực hiện phép xử lý của hàm được xác lập bằng hàm `set_unexpected`.

```
#include <iostream.h>
class Loi_A {};
class Loi_B {}
```

```
void LoiKhongCho()
{
        cout << "Chuong trinh ket thuc vi gap loi khong cho\n";
}
void PhatLoi() throw(Loi_B){
throw (Loi_A);
}
void main(){
        // xác lập hàm kết thúc khi gặp lỗi không chờ đợi
        set_unexpected(LoiKhongCho);
        try     {
          cout << "Goi ham phat loi\n";
                PhatLoi();
        }
        catch ( ... ) {
                cout << "Loi da bi bat";
        }
}
```

```
Goi ham phat loi
Chuong trinh ket thuc vi gap loi khong cho
```

3. Xử lý lỗi trong lớp ứng dụng

Trong mục này chúng ta sẽ xây dựng một lớp ứng dụng mảng động có kiểm soát lỗi khởi tạo với số phần tử nhỏ hơn hoặc bằng không và lỗi truy nhập phần tử ngoài chỉ số.

```
#include <iostream.h>
#include <stdlib.h>
class Loi{
public:
        virtual void InLoi()=0;
};
```

```
// lỗi khởi tạo
class Loi_KT : public Loi {
public:
        int spt;
        Loi_KT(int n): spt(n) {}
        void InLoi() {
        cout << "Loi khoi tao voi " << spt << " phan tu\n";
        }
};
// loi truy cap
class Loi_TC : public Loi {
public:
        int cs;
        Loi_TC(int i): cs(i) {}
        void InLoi() {
        cout << "Loi truy cap chi so " << cs << "\n";
        }
};

class Array
{
        int spt; // so phan tu mang
        int *dl; // du lieu cua mang
public:
        Array(int n): spt(n)  {
                if ( spt <= 0 ) throw( Loi_KT(spt) );
                dl = new int[spt];
        }
        ~Array() { delete dl; }
        int & operator [] (int i){
                if ( i<0 || i>=spt ) throw( Loi_TC(i) );
                return dl[i];
```

```
        }
};
void main(){
        try     {
                Array a(-3);
                a[5] = 10;
        }
        // bat toan bo loi ke thua tu Loi
        catch ( Loi& l )        {
                l.InLoi(); // in loi tuong ung hoi
        }
}
```

```
Loi khoi tao voi -3 phan tu
```

Trong chương trình trên đã dùng kĩ thuật đa hình để tạo một lớp lỗi trừu tượng cơ sở có phương thức in lỗi ảo. Các lỗi cụ thể kế thừa từ lớp này và thi hành cụ thể việc in lỗi cho nó. Khi bắt lỗi, chỉ cần bắt lỗi lớp cơ sở một cách tổng quát thì tất cả các kiểu lỗi trong lớp kế thừa đều bị bắt. Khi gọi thủ tục in lỗi bị bắt, hệ thống sẽ in đúng lỗi của lớp nó thuộc vào tương tự như tính tương ứng bội.