Lập trình hướng đối tượng
13CTT21
Tuần 10
EXCEPTION

Hồ Tuấn Thanh

2

Xử lí ngoại lệ

cuu duong than cong . com

Khi có lỗi lúc chạy chương trình

- Thông báo lỗi và kết thúc chương trình
- Trả về giá trị đặc biệt thể hiện chương trình có lỗi
- Sử dụng biến toàn cục lưu trạng thái lỗi
- Gọi một hàm qui ước khi có lỗi

Lợi ích của xử lí ngoại lệ

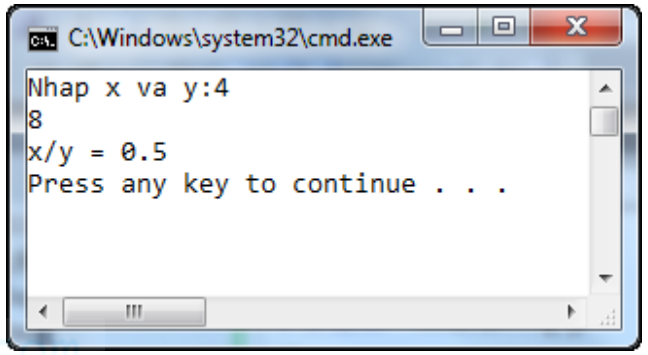
- Tiếng Anh: **exception handling**
- Tách biệt phần xử lí lỗi ra đoạn chương trình bình thường
- Thông báo lỗi ra bên ngoài cho các hàm mức cao hơn xử lí

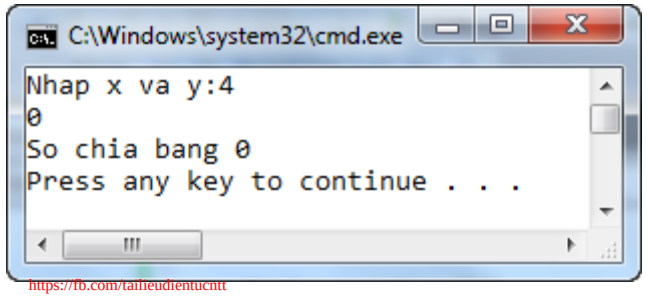
try, throw và catch

- Cơ chế xử lí ngoại lệ trong C++/C#/Java được xây dựng với 3 từ khóa chính: try, throw và catch.
- Khối lệnh try chứa đoạn code mà ta cho rằng sẽ có trường hợp phát sinh ngoại lệ, cần được theo dõi.
- Lệnh throw dùng để tạo ra một ngoại lệ.
  - Lệnh throw có thể do máy tính sinh ra.
  - Hoặc do người lập trình throw khi cần.
- Khối lệnh catch chứa đoạn code xử lí khi ngoại lệ đã xảy ra trong khối lệnh try.

Ví dụ


```
void main()
{
    int x;
    int y;
    try
    {
        cout<<"Nhap x va y:";
        cin>>x>>y;
        if(y==0)
            throw 0;
        cout<<"x/y = "<<1.0*x/y<<endl;
    }
    catch(int i)
    {
        cout<<"So chia bang 0"<<endl;
    }
}
```





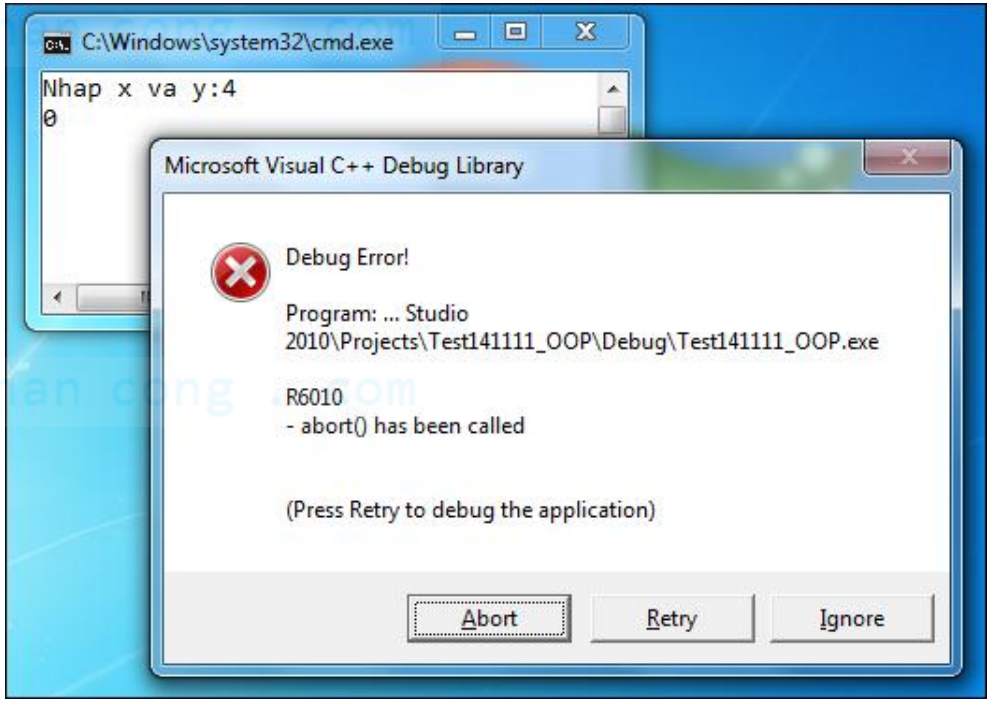
Ví dụ

- Với những trường hợp bình thường (y khác 0), các câu lệnh trong khối try sẽ được chạy.
  - Câu lệnh tính thương ở sau throw sẽ được chạy.
- Với trường hợp y bằng 0, ngay sau lệnh throw, các câu lệnh trong khối catch sẽ được chạy.
  - Câu lệnh tính thương ở sau throw sẽ ko được chạy.

throw và try

- Lệnh throw "ném" ra đối tượng kiểu gì thì lệnh "catch" phải bắt đúng kiểu đó.

```
void main()
{
    int x;
    int y;
    try
    {
        cout<<"Nhap x va y:";
        cin>>x>>y;
        if(y==0)
            throw 3.14;
        cout<<"x/y = "<<1.0*x/y<<endl;
    }
    catch(int i)
    {
        cout<<"So chia bang 0"<<endl;
    }
}
```

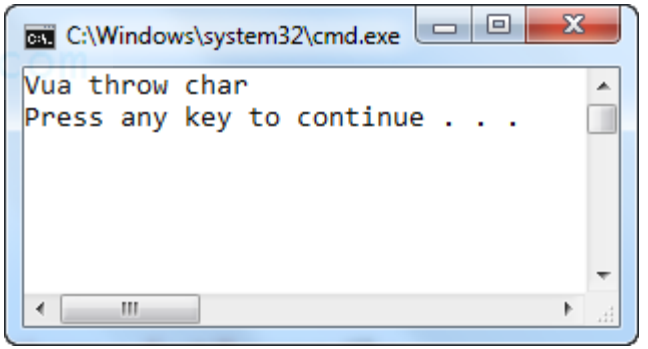


try và catch

- Một lệnh try có thể đi kèm với nhiều lệnh catch.
- Lệnh catch(…) dùng để bắt các ngoại lệ mà ko có lệnh catch nào xử lí được.
  - Thường là lệnh catch cuối cùng của một try.

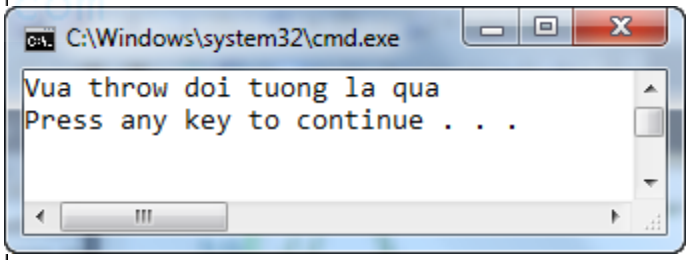
Ví dụ

```
void main()
{
    try
    {
        throw 'x';
    }
    catch(int i)
    {
        cout<<"Vua throw int"<<endl;
    }
    catch(double x)
    {
        cout<<"Vua throw double"<<endl;
    }
    catch(char c)
    {
        cout<<"Vua throw char"<<endl;
    }
    catch(...)
    {
        cout<<"Vua throw doi tuong la qua"<<endl;
    }
}
```



Ví dụ


```
void main()
{
    try
    {
        throw 'x';
    }
    catch(int i)
    {
        cout<<"Vua throw int"<<endl;
    }
    catch(double x)
    {
        cout<<"Vua throw double"<<endl;
    }
    catch(...)
    {
        cout<<"Vua throw doi tuong la qua"<<endl;
    }
}
```



try, throw và catch trong hàm

- Dĩ nhiên ta có thể try, throw và catch trong một hàm bất kì.

```
void f()
{
    try
    {
        throw 'x';
    }
    catch(int i)
    {
        cout<<"Vua throw int"<<endl;
    }
    catch(char c)
    {
        cout<<"Vua throw char"<<endl;
    }
}
void main()
{
    f();
}
```

throw trong, catch ngoài

- Dĩ nhiên ta có thể throw trong một hàm và để đoạn catch ở hàm cha (hàm mà gọi hàm chứa lệnh throw).
  - Đây là điều nên làm, vì thường khi có lỗi xảy ra, hàm hiện tại ko biết xử lí thế nào. Chỉ có hàm gọi hàm đó mới có đầy đủ thông tin để xử lí lỗi đó.

Ví dụ


```
void f()
{
    throw 'x';
}
void main()
{
    try
    {
        f();
    }
    catch(char c)
    {
        cout<<"Vua throw choi thoi, dung lo"<<endl;
    }
}
```

Loại đối tượng khi catch

- Thông thường ta ko catch các đối tượng có kiểu dữ liệu cơ sở (int, float, char,…) mà thường catch các đối tượng kiểu lớp.
  - Lí do, thông thường ta hay xây dựng lớp chứa thông tin lỗi.

Ví dụ


```
class MyException
{
private:
    string message;
public:
    MyException(string msg)
    {
        message=msg;
    }
    string getMessage()
    {
        return message;
    }
};
```

```
void main()
{
    int x;
    int y;
    try
    {
        cout<<"Nhap x va y:";
        cin>>x>>y;
        if(y==0)
            throw MyException("Divided by zero");
        cout<<"x/y = "<<1.0*x/y<<endl;
    }
    catch(MyException& ex)
    {
        cout<<ex.getMessage()<<endl;
    }
}
```

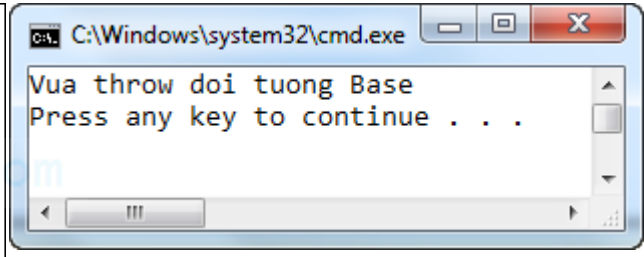
catch đối tượng lớp cơ sở

- Lệnh catch đối tượng lớp cơ sở, có thể bắt các lệnh throw đối tượng lớp dẫn xuất.
- Do đó, thường để lệnh catch lớp cơ sở ở dưới lệnh catch lớp dẫn xuất.
- Sau đây là một trường hợp sai.

Ví dụ


```
class Base{};
class Derived: public Base{};
```

```
void main()
{
    try
    {
        Derived d;
        throw d;
    }
    catch(Base& b)
    {
        cout<<"Vua throw doi tuong Base"<<endl;
    }
    catch(Derived& x)
    {
        cout<<"Vua throw doi tuong Derived"<<endl;
    }
}
```



Qui định các ngoại lệ của hàm

- Ta có thể qui định các ngoại lệ mà một hàm sẽ trả ra nếu có.
- Nếu trong hàm đó, ta "ném" ra một ngoại lệ ko được liệt kê thì hàm unexpected() sẽ được gọi.
  - Visual C++ ko ngăn một hàm "ném" ra một ngoại lệ mà chưa được liệt kê.

Ví dụ

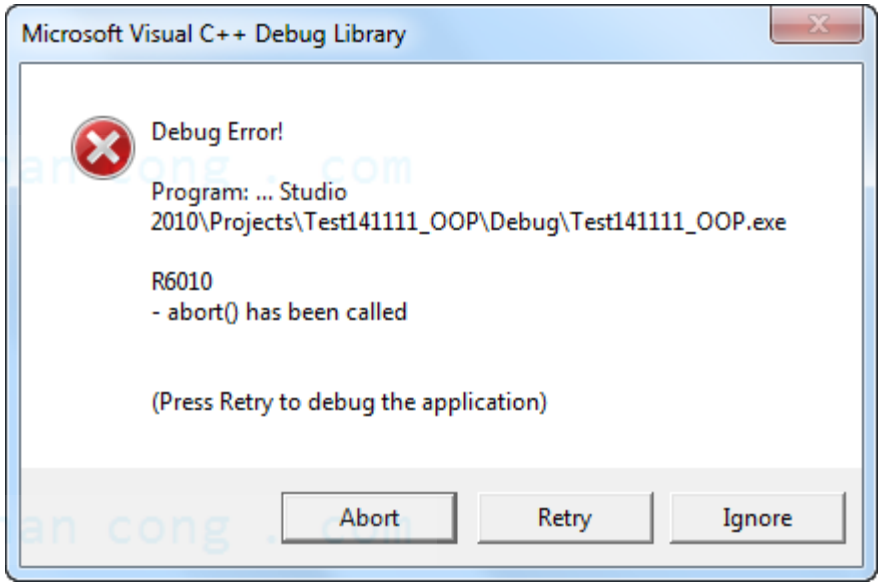

```
void f(int test) throw(int, char, double)
{
    if(test==1) throw test;
    if(test==2) throw 'x';
    if(test==3) throw 3.14;
}
```

```
void main()
{
    try
    {
        f(1);
    }
    catch(int n)
    {
        cout<<"Vua throw int = "<<n<<endl;
    }
    catch(char c)
    {
        cout<<"Vua throw char = "<<c<<endl;
    }
    catch(double x)
    {
        cout<<"Vua throw double = "<<x<<endl;
    }
}
```

Ví dụ


```
void f(int test) throw(int, char, double)
{
    if(test==1) throw test;
    if(test==2) throw 'x';
    if(test==3) throw 3.14;
    if(test==4) throw "abc";
}
```

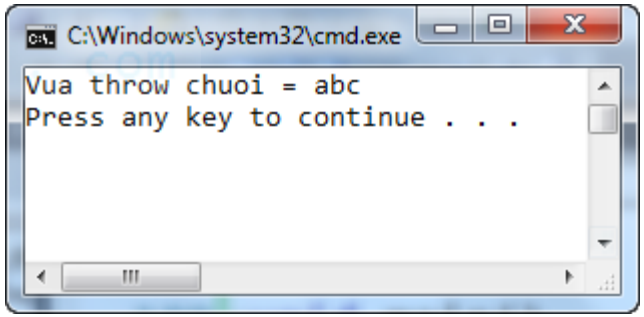
```
void main()
{
    try
    {
        f(4);
    }
    catch(int n)
    {
        cout<<"Vua throw int = "<<n<<endl;
    }
    catch(char c)
    {
        cout<<"Vua throw char = "<<c<<endl;
    }
    catch(double x)
    {
        cout<<"Vua throw double = "<<x<<endl;
    }
}
```



Ví dụ


```
void f(int test) throw(int, char, double)
{
    if(test==1) throw test;
    if(test==2) throw 'x';
    if(test==3) throw 3.14;
    if(test==4) throw "abc";
}
```

```
void main()
{
    try
    {
        f(4);
    }
    catch(char* s)
    {
        cout<<"Vua throw chuoi = "<<s<<endl;
    }
}
```



Ví dụ

- Nếu ta để danh sách rỗng thì có nghĩa rằng ta cam đoan hàm này ko có ngoại lệ.

```
void f(int test) throw()
{
    cout<<"Ham nay ko co ngoai le gi het!!!";
}
```

Rethrow một ngoại lệ

- Tình huống - h() gọi g(), g() gọi f():
  - Hàm f có ngoại lệ, throw lên cho g(); g() catch lại, xử lí xong, nhưng vẫn muốn throw lên cho h() biết.
  - Hàm f có ngoại lệ, throw lên cho g(); g() ko biết xử lí, lại throw lên cho h() xử lí.
- Có 2 cách rethrow:
  - f() throw gì cho g(), g() throw lại y chang cho h()
    - Dùng câu lệnh throw;
  - f() throw cho g(), g() tạo ngoại lệ mới rồi throw lại cho h()

Ví dụ


```
void f()
{
    throw 3.14;
}
```

```
void g()
{
    try
    {
        f();
    }
    catch(double x)
    {
        cout<<"Dang catch double o ham g"<<endl;
        throw;
    }
}
```

```
void main()
{
    try
    {
        g();
    }
    catch(double x)
    {
        cout<<"Dang catch double o ham main"<<endl;
    }
}
```

Ví dụ


```
void f()
{
    throw 3.14;
}
```

```
void g()
{
    try
    {
        f();
    }
    catch(double x)
    {
        cout<<"Dang catch double o ham g"<<endl;
        throw 'c';
    }
}
```

```
void main()
{
    try
    {
        g();
    }
    catch(char c)
    {
        cout<<"Dang catch char o ham main"<<endl;
    }
}
```

Rò rỉ bộ nhớ khi throw – Vấn đề


```
void test(int t, int a, int b)
{
    int *p;
    p=new int[t];
    // Blah blah blah
    if(a==b)
        throw "error";
    delete []p;
}
```

Rò rỉ bộ nhớ khi throw – Giải pháp 1


```
void test(int t, int a, int b)
{
    int *p;
    p=new int[t];
    // Blah blah blah
    if(a==b)
    {
        delete []p;
        throw "error";
    }
    delete []p;
}
```

Rò rỉ bộ nhớ khi throw – Giải pháp 2

- Với giải pháp 1, ta cần nhớ giải phóng vùng nhớ đã sử dụng trước khi "ném" lỗi.
- Một giải pháp tốt hơn là đưa các biến có sử dụng đến tài nguyên vào trong các lớp. Khi đó hàm hủy của lớp này sẽ đảm nhận vai trò giải phóng tài nguyên đang nắm giữ.
- Phương pháp này gọi là Resource Acquisition Is Initialization (RAII) do Bjarne Stroustrup đề nghị.

throw trong constructor

- OK, cứ throw bình thường.
- Nhớ hủy vùng nhớ đã cấp phát cho con trỏ nếu có

throw trong destructor

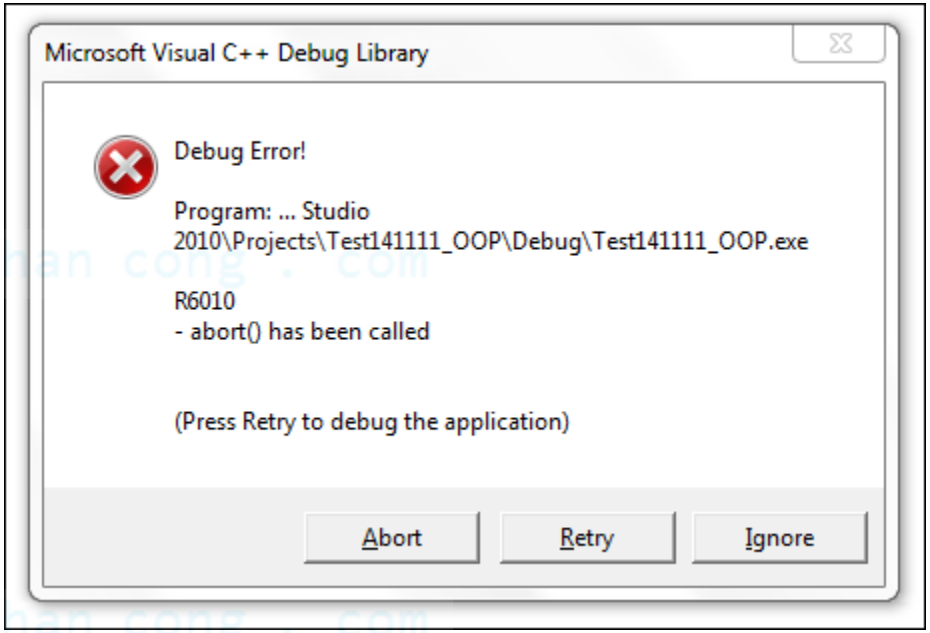
- Không nên "ném" ngoại lệ trong hàm hủy, mà chỉ ghi nhận lại lỗi (chẳng hạn dùng logger, ghi lỗi xuống file).

Ví dụ


```
class A
{
public:
    ~A()
    {
        throw 3.14;
    }
};
```

Ví dụ - trường hợp 1


```
void main()
{
    A a;
    // Blah blah blah
}
```



Ví dụ


```
class A
{
public:
    ~A()
    {
        throw 3.14;
    }
};

void test()
{
    A a;
    throw 1;
}
```

Ví dụ - trường hợp 2


```
void main()
{
    try
    {
        test();
    }
    catch(int i)
    {
        cout<<"Vua throw tu test"<<endl;
    }
    catch(double x)
    {
        cout<<"Vua throw tu ham huy cua A"<<endl;
    }
    catch(...)
    {
        cout<<"Tat cac loi deu phai vay day"<<endl;
    }
}
```