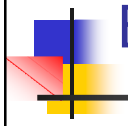
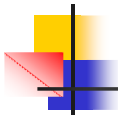
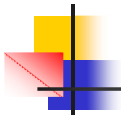


Bài 8: Ngoại lệ (Exception)



Nội dung

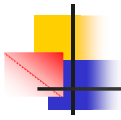
- Các loại lỗi thường gặp
- Cách xử lý lỗi trong chương trình
- Ngoại lệ (exception), cách sử dụng ngoại lệ
- Lớp exception.



CÁC LOẠI LỖI THƯỜNG GẶP

- Cú pháp
  - int a;
  - char a, a, b;
  - for (;);
- Biên dịch
  - int a, b = 0, c =1; a = c / b;
- Lôgic
  - for (int i = 0; i< 10; i--)

3

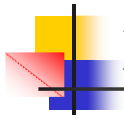


XỬ LÝ LỖI THƯỜNG GẶP

- Cú pháp: sai đâu sửa đó ➔ cẩn thận, nắm vững cú pháp
- Biên dịch: …?
- Lôgic: xem lại thuật toán, mã nguồn
- Ví dụ: hàm xử lý lỗi chia cho 0

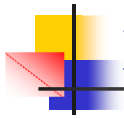
```
double MyDivide(double numerator, double denominator) {
        if (denominator == 0.0) {
                // Do something to indicate that an error occurred
        }
        else {
                return numerator / denominator;
        }
}
```

4



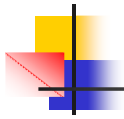
Xử lý lỗi truyền thống

- Xử lý lỗi truyền thống: thường là mỗi hàm lại thông báo trạng thái thành công/thất bại qua một mã lỗi
  - biến toàn cục
  - giá trị trả về
    - **int remove (const char * *filename*);**
  - tham số phụ là tham chiếu
    - double MyDivide(double *numerator,*
      double *denominator,* **int& *status);***



Xử lý lỗi truyền thống

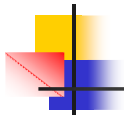
- Hạn chế của xử lý lỗi truyền thống
  - Phải có lệnh kiểm tra lỗi sau *mỗi lời gọi hàm*
    - code trông rối rắm, dài, khó đọc
  - Lập trình viên ứng dụng quên kiểm tra, hoặc cố tình bỏ qua
  - Rắc rối khi đẩy thông báo lỗi từ hàm được gọi sang hàm gọi vì từ một hàm ta chỉ có thể trả về một kiểu thông báo lỗi



Ngoại lệ (exception) trong C++

- Exception – ngoại lệ là cơ chế thông báo và xử lý lỗi giải quyết được các vấn đề kể trên
- Tách được phần xử lý lỗi ra khỏi phần thuật toán chính
- cho phép 1 hàm thông báo về nhiều loại ngoại lệ
- không thể bỏ qua ngoại lệ, nếu không, chương trình sẽ kết thúc
- Tóm lại, cơ chế ngoại lệ mềm dẻo hơn kiểu xử lý lỗi truyền thống

7



Các kiểu ngoại lệ

- Một ngoại lệ là một đối tượng chứa thông tin về một lỗi và được dùng để truyền thông tin đó tới cấp thực thi cao hơn
- Ngoại lệ có thể thuộc kiểu dữ liệu bất kỳ của C++
  - có sẵn, chẳng hạn **int, char* ...**
  - hoặc kiểu người dùng tự định nghĩa (thường dùng)
  - các lớp ngoại lệ trong thư viện <**exception**>

8

Cơ chế ngoại lệ

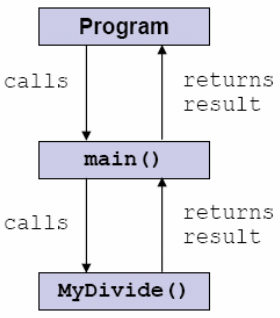
- Quá trình truyền ngoại lệ từ ngữ cảnh thực thi hiện hành tới mức thực thi cao hơn gọi là ném một ngoại lệ (throw an exception)
  - vị trí trong mã của hàm nơi ngoại lệ được ném được gọi là điểm ném (throw point)
- Khi một ngữ cảnh thực thi tiếp nhận và truy nhập một ngoại lệ, nó được coi là bắt ngoại lệ (catch the exception)

9

Cơ chế ngoại lệ

- Quy trình gọi hàm và trả về trong trường hợp bình thường

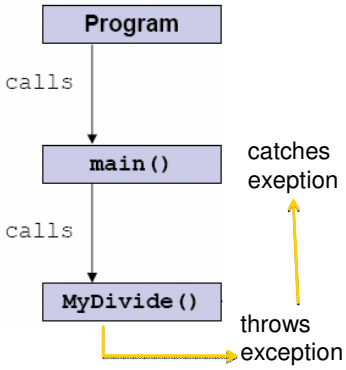
```
int main() {
int x, y;
cout << "Nhập 2 số: ";
cin >> x >> y;
cout << "Kết quả chia số thứ nhất cho thứ hai: "
cout<< MyDivide(x, y) << "\n";
}
```



10

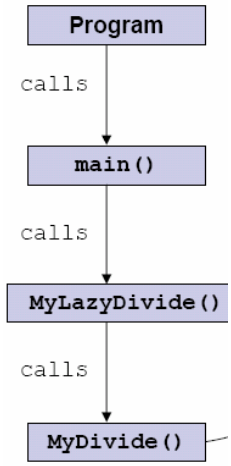
Cơ chế ngoại lệ

- Quy trình ném và bắt ngoại lệ:
  - giả sử người dùng nhập mẫu số bằng 0
  - Mã chương trình trong MyDivide() tạo một ngoại lệ (bằng cách nào đó) và ném
  - Khi một hàm ném một ngoại lệ, nó lập tức kết thúc thực thi và gửi ngoại lệ đó cho nơi gọi nó
  - Nếu main() có thể xử lý ngoại lệ, nó sẽ bắt và giải quyết ngoại lệ
    - Chẳng hạn yêu cầu người dùng nhập lại mẫu số



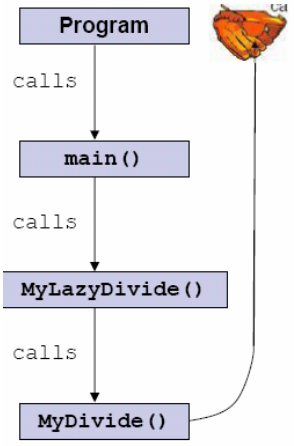
Cơ chế ngoại lệ

- Nếu một hàm không thể bắt ngoại lệ
  - giả sử hàm **MyLazyDivide() không** thể bắt ngoại lệ do **MyDivide() ném**
    - Không phải hàm nào bắt được ngoại lệ cũng có thể bắt được ***mọi loại ngoại lệ***
    - Chẳng hạn hàm **f() bắt được các ngoại** lệ loại **E1 nhưng không bắt được các** ngoại lệ loại **E2**
  - Ngoại lệ đó sẽ được chuyển lên mức trên cho **main() bắt**



Cơ chế ngoại lệ

- Nếu không có hàm nào bắt được ngoại lệ?
- Tại mức thực thi cao nhất, chương trình tổng (nơi gọi hàm **main()) sẽ bắt mọi** ngoại lệ còn sót lại mà nó nhìn thấy
- Khi đó, chương trình lập tức kết thúc



Cú pháp xử lý ngoại lệ

- Cơ chế xử lý ngoại lệ của C++ có 3 tính năng chính
  - khả năng tạo và ném ngoại lệ (sử dụng từ khoá **throw)**
  - khả năng bắt và giải quyết ngoại lệ (sử dụng từ khoá **catch)**
  - khả năng tách lôgic xử lý ngoại lệ trong một hàm ra khỏi phần còn lại của hàm (sử dụng từ khoá **try)**

throw – Ném ngoại lệ

- Để ném một ngoại lệ, ta dùng từ khoá **throw, kèm theo đối tượng** mà ta định ném

  throw <object>;
- Ta có thể dùng ***mọi thứ làm ngoại lệ, kể cả giá trị thuộc kiểu có sẵn***

  throw 15;

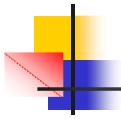
  throw MyObj(…);
- Ví dụ, **MyDivide() ném ngoại lệ là một string**

```
double MyDivide(double numerator, double denominator) {
        if (denominator == 0.0) {
                throw string("So bi chia khong the bang 0");
        } else {
                return numerator / denominator;
        }
}
```

throw – Ném ngoại lệ

- Trường hợp cần cung cấp nhiều thông tin hơn cho hàm gọi, ta tạo một class dành riêng cho các ngoại lệ
- Ví dụ, ta cần cung cấp cho người dùng 2 số nguyên.Ta có thể tạo một lớp ngoại lệ:

```
class MyExceptionClass {
        public:
                MyExceptionClass(int a, int b);
                ~MyExceptionClass();
                int getA();
                int getB();
        private:
                int a, b;
};
int x, y;
...
if (...) throw MyExceptionClass(x, y);
...
```



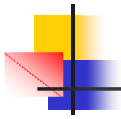
Khối **try – catch**

- Khối **try – catch dùng để:**
  - Tách phần giải quyết lỗi ra khỏi phần có thể sinh lỗi
  - Quy định các loại ngoại lệ được bắt tại mức thực thi hiện hành
- Cú pháp chung cho khối **try – catch:**

```
try {
// Code that could generate an exception
}
catch (<Type of exception>) {
// Code that resolves an exception of that type
};
```

  - Mã liên quan đến thuật toán nằm trong khối **try**
  - Mã giải quyết lỗi đặt trong (các) khối **catch**

17

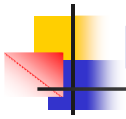


Khối try – catch

- Có thể có nhiều khối **catch, mỗi khối chứa mã để giải** quyết một loại ngoại lệ cụ thể

Dấu chấm phẩy đánh dấu kết thúc của toàn khối **try-catch**

```
try {
// Code that could generate an exception
}
catch (<Exception type1>) {
// Code that resolves a type1 exception
}
catch (<Exception type2>) {
// Code that resolves a type2 exception
}
...
catch (<Exception typeN>) {
// Code that resolves a typeN exception
};
```



Khối try – catch

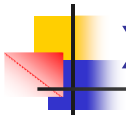
```
#include<iostream>
using namespace std;
double MyDivide(double numerator, double denominator) {
        if (denominator == 0.0) {
                throw string("So bi chia khong the bang 0");
        }
        else
         {
                return numerator / denominator;
        }
}
```

19

Khối try – catch

```
int main() {
    int x, y;
    double result;
    cout << "Nhập 2 số: ";cin >> x >> y;
    try {
          result = MyDivide(x, y);
    }
    catch (string &s) {
          cout << s << endl;
    };
    cout << "Kết quả chia số thứ nhất cho thứ hai: ";
    cout<< result << "\n";
}
```

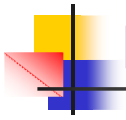
20



Xử lý ngoại lệ

- Khi một ngoại lệ được ném từ trong một khối try, hệ thống xử lý ngoại lệ sẽ kiểm tra các kiểu được liệt kê trong khối catch theo thứ tự liệt kê:
  - Khi tìm thấy kiểu ăn khớp, ngoại lệ được coi là được giải quyết, không cần tiếp tục tìm kiếm
  - Nếu không tìm thấy, mức thực thi hiện hành bị kết thúc, ngoại lệ được chuyển lên mức cao hơn
- Khi tìm các kiểu dữ liệu khớp với ngoại lệ, trình biên dịch nói chung sẽ không thực hiện đổi kiểu tự động
  - Nếu một ngoại lệ kiểu float được ném, nó sẽ không khớp với một khối catch cho ngoại lệ kiểu int
- Một đối tượng hoặc tham chiếu kiểu dẫn xuất sẽ khớp với một lệnh catch dành cho kiểu cơ sở
  - Giả sử lớp **Car** dẫn xuất từ lớp cơ sở MotorVehicle.
  - Nếu một ngoại lệ kiểu **Car** được ném, nó sẽ khớp với một khối catch cho ngoại lệ kiểu **MotorVehicle.**

21

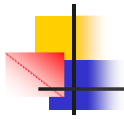


Lớp exception

- Để tích hợp hơn nữa các ngoại lệ vào ngôn ngữ C++, lớp **exception đã được đưa vào thư viện chuẩn**
  - sử dụng **#include <exception> và namespace std**
- Sử dụng thư viện này, ta có thể ném các thể hiện của **exception hoặc tạo các lớp dẫn xuất từ đó**
- Lớp **exception có một hàm ảo what(), có thể định** nghĩa lại **what() để trả về một xâu ký tự**

```
try {...}
catch (exception e) {
        cout << e.what();
}
```

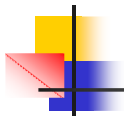
22



Lớp exception

- Một số lớp ngoại lệ chuẩn khác được dẫn xuất từ lớp cơ sở **exception**
- File header **<stdexcept> (cũng thuộc thư viện chuẩn** C++) chứa một số lớp ngoại lệ dẫn xuất từ **exception**
  - File này cũng đã **#include <exception> nên khi dùng không** cần **#include cả hai**
  - Trong đó có hai lớp quan trọng được dẫn xuất trực tiếp từ **exception:**
    - **runtime_error**
    - **logic_error**

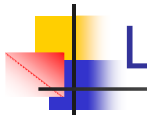
23



Lớp exception

- **runtime_error dùng để đại diện cho các lỗi trong** thời gian chạy (các lỗi là kết quả của các tình huống không mong đợi, chẳng hạn: hết bộ nhớ)
- **logic_error dùng cho các lỗi trong lôgic chương trình** (chẳng hạn truyền tham số không hợp lệ)
- Thông thường, ta sẽ dùng các lớp này (hoặc các lớp dẫn xuất của chúng) thay vì dùng trực tiếp **exception**
  - Một lý do là cả hai lớp này đều có constructor nhận tham số là một **string mà nó sẽ là kết quả trả về của hàm what()**

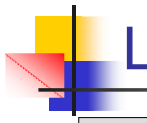
24



Lớp exception

- **runtime_error có các lớp dẫn xuất sau:**
  - **range_error** điều kiện sau (post-condition) bị vi phạm
  - **overflow_error** xảy ra tràn số học
  - **bad_alloc** không thể cấp phát bộ nhớ
- **logic_error có các lớp dẫn xuất sau:**
  - **domain_error** điều kiện trước (pre-condition) bị vi phạm
  - **invalid_argument** tham số không hợp lệ được truyền cho hàm
  - **length_error** tạo đối tượng lớn hơn độ dài cho phép
  - **out_of_range** tham số ngoài khoảng (chẳng hạn chỉ số không hợp lệ)

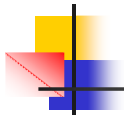
25



Lớp exception – Ví dụ

```
#include<iostream>
#include <stdexcept>
using namespace std;
int main()
{
try {
     int b = 0;
     if (b == 0) throw invalid_argument("Chia cho 0");
     // một số lệnh khác
}
catch (invalid_argument e)
{
     cout<<e.what()<<endl;
}
return 0;
}
```

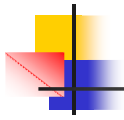
26



Review

- Các loại lỗi thường gặp
- Cách xử lý lỗi trong chương trình
- Ngoại lệ (exception), cách sử dụng ngoại lệ
- Lớp exception.

27



Bài tập

1. Thực hành các bài tập trong bài trên máy tính.
2. Sử dụng ngoại lệ để bắt lỗi trong các bài tập thực hành.

28