XỬ LÝ LỖI & NGOẠI LỆ

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

GVHD: Trương Toàn Thịnh

NỘI DUNG

- Giới thiệu
- Kỹ thuật try-catch
- Giới hạn lỗi có thể throw
- Vấn đề rò rỉ
- Cách khớp khối lệnh catch-throw
- Xử lý lỗi với hàm tạo lập/hàm hủy
- Các vấn đề khác
- Kế thừa và ngoại lệ

GIỚI THIỆU

- Các cách mà lập trình viên thường áp dụng khi gặp lỗi chương trình:
  - Thông báo lỗi & kết thúc chương trình
  - Trả về giá trị đặc biệt thể hiện trạng thái lỗi
  - Sử dụng biến toàn cục cập nhật trạng thái lỗi
  - Gọi hàm quy ước sẵn khi gặp lỗi
- Các hạn chế:
  - Toàn bộ kết quả đang tính toán có thể bị hủy nếu dừng chương trình
  - Giá trị đặc biệt có thể trùng với miền giá trị
  - Các chương trình chạy song song không phù hợp dùng biến toàn cục để cập nhật lỗi

GIỚI THIỆU

- C++ cung cấp cơ chế bắt lỗi với một số tính chất sau:
  - Sử dụng đơn giản
  - Tách biệt đoạn mã xử lý lỗi với đoạn mã thực thi tác vụ
  - Cơ chế xử lý sẽ thông báo lỗi cho các hàm ở mức cao hơn xử lý

NỘI DUNG

- Giới thiệu
- Kỹ thuật try-catch
- Giới hạn lỗi có thể throw
- Vấn đề rò rỉ
- Cách khớp khối lệnh catch-throw
- Xử lý lỗi với hàm tạo lập/hàm hủy
- Các vấn đề khác
- Kế thừa và ngoại lệ

KỸ THUẬT TRY-CATCH

- Kỹ thuật này có thể thông báo lỗi, sửa chữa hoặc xử lý các lỗi khi cần thiết.

```
double tinh(double x, double y){
  if(x == y)
    throw “divide by zero”;
  return (x*y)/(x - y);
}
int main(){
  double a = 7, b = 7;
  try{
    a = tinh(a, b);
  }
  catch(char* s){
    cout << “Error: ” << s << endl;
  }
  return 0;
}
```

KỸ THUẬT TRY-CATCH

- Khối lệnh try-throw-catch
  - Try: khối này chứa các lệnh có nguy cơ gây lỗi, dùng kèm với từ khóa ‘throw’

```
try{
    //các dòng lệnh có khả năng gây lỗi
   //throw…
}
```

  - Throw:
    - Khi gặp câu lệnh này, chương trình dừng lại và nhảy ra cấp cao hơn xử lý lỗi
    - Nên throw ra kiểu đối tượng kế thừa lớp exception

KỸ THUẬT TRY-CATCH

- Khối lệnh try-throw-catch
  - Catch: khối này bắt các lỗi phát sinh trong ‘try’

```
try{
   //các dòng lệnh có khả năng gây lỗi
}
catch(<Kiểu dữ liệu> tên_biến_1){
    //xử lý lỗi
}
catch (<Kiểu dữ liệu> tên_biến_2){
    //xử lý lỗi
}
```

KỸ THUẬT TRY-CATCH

- Các lưu ý:
  - Có thể ‘catch’ theo kiểu tham trị, tham chiếu hay tham con trỏ
  - ‘catch’ có thể truy xuất giá trị của biến lỗi nhưng mọi thay đổi không ảnh hưởng tới biến gốc (cho dù truyền tham chiếu hay tham con trỏ)
  - Nếu lệnh ‘throw’ không có giá trị trả về thì khối ‘catch’ không hoạt động và chương trình kết thúc
  - Có ít nhất một khối ‘catch’ sau khối ‘try’
  - Nếu muốn bắt tất cả các exception bất kể kiểu dữ liệu thì dùng cú pháp ‘catch(…){}’

NỘI DUNG

- Giới thiệu
- Kỹ thuật try-catch
- Giới hạn lỗi có thể throw
- Vấn đề rò rỉ
- Cách khớp khối lệnh catch-throw
- Xử lý lỗi với hàm tạo lập/hàm hủy
- Các vấn đề khác
- Kế thừa và ngoại lệ

GIỚI HẠN LỖI CÓ THỂ THROW

- Có thể quy định sẵn một vài exception mà một hàm có thể throw
- Cú pháp:

  <Kiểu trả về> Tên_hàm(<Tham_số>) throw(các_exception)

  int Tinh(int a, int b) throw(Err1, Err2)
- Có thể yêu cầu hàm không được throw
- Cú pháp

  <Kiểu trả về> Tên_hàm(<Tham_số>) throw()
- Dùng hàm **set_unexpected**(Tên_hàm_mới) để xử lý khi hàm throw về exception không có trong danh sách. Khi đó Tên_hàm_mới sẽ thay thế xử lý

VẤN ĐỀ RÒ RỈ

- Khi throw lỗi ra hàm khác xử lý, rất có thể sẽ bị rò rỉ phần bộ nhớ đang sử dụng
- Xét đoạn mã

```
void test(int t, int a, int b){
  int *p = new int[t];
  if(a == b){
    throw “error: ”;
  delete []p;
}
```

- Cần dời lệnh delete lên trước ‘throw’
- Giải pháp tốt hơn là chuyển các biến xin cấp phát thành các lớp để khi lệnh throw thực thi thì hàm hủy sẽ tự động được gọi và dọn dẹp

CÁCH KHỚP KHỐI LỆNH CATCH-THROW

- Phần mã trong khối ‘catch’ cần có mối quan hệ với phần mã trong khối ‘try’

```
void test(){
  try {
    throw  E;
  }
   catch(H){
  }
}
```

- H và E cần cùng kiểu hoặc H là lớp cha của E (H & E có thể cùng là con trỏ hoặc H là tham chiếu)

XỬ LÝ LỖI VỚI HÀM TẠO LẬP/HÀM HỦY

- Cần giải phóng tài nguyên trước khi ‘throw’ lỗi trong constructor.
- **Không** ‘throw’ lỗi trong destructor.
- Có hai trường hợp khi destructor được gọi
  - Lời gọi thông thường do đối tượng hết phạm vi hoạt động
  - Lời gọi khẩn cấp do một exception nào đó được phát sinh và kết quả là các đối tượng liên quan bị hủy
- Rơi vào trường hợp hai sẽ dẫn tới việc có hai exception: một exception ban đầu và một exception trong destructor

XỬ LÝ LỖI VỚI HÀM TẠO LẬP/HÀM HỦY

- Ví dụ: test throw Err1 sau đó a throw Err2

```
class Err1{};
class Err2{};
class A{
  public:
    ~A(){throw Err2();}
}
void test(){
  A a;
  throw Err1();
}
void main(){
  try{ test();}
  catch(Err2& b){cout << "Bat duoc loi 2" << endl;}
  catch(Err1& a){cout << "Bat duoc loi 1" << endl;}
  catch(…){cout << "Bat duoc loi" << endl;}
}
```

CÁC VẤN ĐỀ KHÁC

- Cơ chế try-throw-catch sẽ dừng ngang chương trình và nhảy ra bên ngoài cho tới khi gặp khối lệnh catch phù hợp
- Có thể lợi dụng tính chất này để lập trình
- Lời khuyên: không nên lạm dụng vì bản chất của cơ chế bắt ngoại lệ là để xử lý lỗi
- Lạm dụng kĩ thuật này có thể dẫn tới việc phức tạp khi bảo trì mã nguồn

CÁC VẤN ĐỀ KHÁC

- Ví dụ

```
struct Data{//…};
struct Node{
  int k;
  Data info;
  Node* left, *right;
};
void TimNode(Node*p, int k){
  if(p->k == k) throw p;
  if(p->left != NULL) TimNode(p->left, k);
  if(p->right != NULL) TimNode(p->right, k);
}
void main(){
  Node* root;
  int giaTri;
  try {TimNode(root, giaTri)}
  catch(Node* p){}
}
```

NỘI DUNG

- Giới thiệu
- Kỹ thuật try-catch
- Giới hạn lỗi có thể throw
- Vấn đề rò rỉ
- Cách khớp khối lệnh catch-throw
- Xử lý lỗi với hàm tạo lập/hàm hủy
- Các vấn đề khác
- Kế thừa và ngoại lệ

KẾ THỪA VÀ NGOẠI LỆ

- Các lớp exception cho phép kế thừa và chuyển kiểu giữa các lớp
- Thực tế các lớp xử lý ngoại lệ đề là lớp con của các lớp exception

```
#include <iostream>
using namespace std;
class MyError{};
class MyLogicalError: public MyError{};

void main(){
  try {throw MyLogicalError();}
  catch(MyError& err){cout << "catch myerror…" << endl;}
  catch(…){cout << "catch something else…" << endl;}
}
```

KẾ THỪA VÀ NGOẠI LỆ

- Các lớp exception có thể đa kế thừa
- Khối lệnh catch có thể bắt một trong hai kiểu cơ sở của lớp con exception

```
#include <iostream>
class InputError{};
class DatabaseError{};
class CompoundError: public InputError, public DatabaseError{};
void UpdateData{throw CompoundError();}
void Input{throw CompoundError();}
void ProcessDatabase(){
  try{UpdataData();}
  catch(DatabaseError& err){std::cout << "Database error…" << endl;}
}
void ProcessInput(){
  try{Input();}
  catch(InputError& err){std::cout << "Input error…" << endl;}
}
```

KẾ THỪA VÀ NGOẠI LỆ

- Việc ép kiểu từ lớp kế thừa sang lớp cơ sở được thực hiện ngầm
- Sẽ khó khăn khi muốn bắt lại exception lớp kế thừa khi đã chuyển thành lớp cơ sở

```
#include <iostream>
using namespace std;
class Base{};
class Child: public Base{};
void test(Base& b){throw b;}
void main(){
  Child c;
  try {test(b);}
  catch(Child& c1){//Khong bat duoc loi o day}
  catch(Base& b1){cout << "catch base…" << endl;}
}
```

KẾ THỪA VÀ NGOẠI LỆ

- Lợi dụng đa xạ để giải quyết vấn đề này

```
#include <iostream>
using namespace std;
class Base{
public:
  virtual void generateError(){throw *this;}
};
class Child: public Base{
public:
  void generateError(){throw *this;}
};
void test(Base& b){b.generateError();}
void main(){
  Child c;
  try {test(b);}
  catch(Child& c1){//Bat duoc loi o day}
  catch(Base& b1){cout << "catch base…" << endl;}
}
```