



Chapter 16

Exception Handling



OBJECTIVES



- ☐ What exceptions are and when to use them.
- ☐ To use **try**, **catch** and **throw** to detect, handle and indicate exceptions, respectively.
- ☐ To process uncaught and unexpected exceptions.
- ☐ To declare new exception classes.
- ☐ How stack unwinding enables exceptions not caught in one scope to be caught in another scope.
- ☐ To handle **new** failures.
- ☐ To understand the standard exception hierarchy.



Topics



- ☐ **16.1 Introduction**
- ☐ 16.2 Scenario A: Handle exception thrown by C++ standard lib
- ☐ 16.3 Scenario B: Define, throw and handle your own exception
- ☐ 16.4 Stack Unwinding



16.1 Introduction



- ❑ **Exception(异常):** An exception is an indication of a **problem** that occurs during a program's **execution**. 程序执行期间, 可检测到的不正常情况.
- ❑ **例子:** 0作除数; 数组下标越界; 打开不存在的文件; 内存分配失败



16.1 Introduction



☐ Intermixing program logic

Perform a task

If the preceding task did not execute correctly

Perform error processing

Perform next task

If the preceding task did not execute correctly

Perform error processing

... ..

☐ Difficult to read, modify, maintain and debug especially in large applications

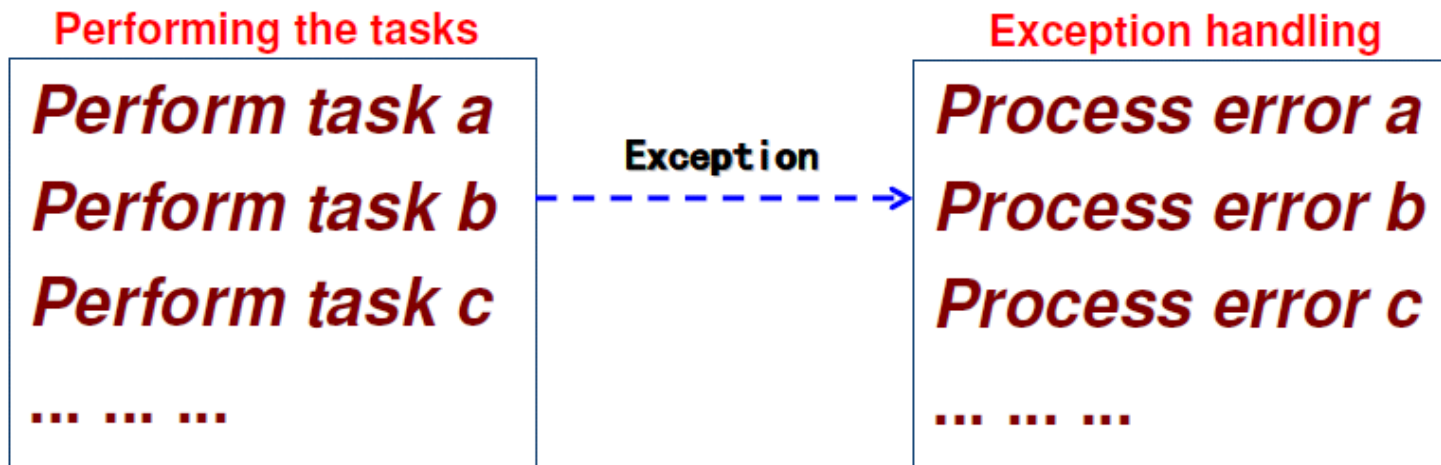
☐ Low performance



16.1 Introduction



- ❑ **Exception handling(异常处理):** In many cases, handling an exception allows a program to continue executing as if no problem had been encountered.



❖ **try-catch**

— **robust(健壮性) and fault-tolerant(容错)** —



16.1 Introduction



- ❑ How to **define** our own Exception?
- ❑ How to **throw** Exception?
- ❑ How to **catch** and **handle** Exception?
- ❑ Stack Unwinding (栈展开机制)

- ❑ **Scenario A:** Handle exception thrown by C++ standard lib.
- ❑ **Scenario B:** Define, throw and handle your own exception.



Topics



- ❑ 16.1 Introduction
- ❑ **16.2 Scenario A: Handle exception thrown by C++ standard lib**
- ❑ 16.3 Scenario B: Define, throw and handle your own exception
- ❑ 16.4 Stack Unwinding



16.2 Scenario A: Handle exception thrown by C++ standard lib



- ❑ 需求: 如何处理C++库调用时抛出的异常?
- ❑ try-catch语句

Termination Model of Exception Handling

P465 Fig.11.15
Attempt to assign 'd' to s1.at(30) yields:

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.

Fig11_15.exe

Fig11_15.exe 遇到问题需要关闭, 我们对此引起的不便表示抱歉。

如果您正处于进程当中, 信息有可能丢失。

请将此问题报告给 Microsoft。
我们已经创建了一个错误报告, 您可以将它发送给我们, 我们将此报告视为保密的和匿名的。

要查看这个错误报告包含的数据, [请点击此处。](#)

调试(D)

发送错误报告(S)

不发送(N)



16.2 Scenario A: Handle exception thrown by C++ standard lib



```
1. class Test{
2. public:
3.     Test(){ cout << "Constructor called." << endl; }
4.     ~Test(){ cout << "Destructor ok." << endl; }
5. };
6. int main()
7. {
8.     Test t;
9.     double *ptr[ 50 ];
10.
11.     for ( int i = 0; i < 50; i++ )
12.     {
13.         ptr[ i ] = new double[ 50000000 ];
14.         cout << "Allocated 50000000 doubles in ptr[ " << i << " ]\n";
15.     }
16.     return 0;
17. }
```





16.2 Scenario A: Handle exception thrown by C++ standard lib



- ❑ 由于new操作失败, 程序abort
- ❑ 危害: 剩余对象全部不调用析构函数等

Constructor called.

Allocated 50000000 doubles in ptr[0]

Allocated 50000000 doubles in ptr[1]

Allocated 50000000 doubles in ptr[2]

Allocated 50000000 doubles in ptr[3]

Allocated 50000000 doubles in ptr[4]

This application has requested the Runtime to terminate it in an unusual way.

Please contact the application's support team for more information.



16.2 Scenario A: Handle exception thrown by C++ standard lib



- ❑ If **new** fails to allocate memory and **set_new_handler** did not register a new -handler function, **new** throws a **bad_alloc** exception.
- ❑ • **Choice 1**: Handle **bad_alloc** exception



16.2 Scenario A: Handle exception thrown by C++ standard lib



```
1. try {  
2.     // code that may throw exceptions  
3. }  
4. catch (exception-declaration) {  
5.     // code that executes when  
6.     // exception-declaration is thrown  
7. }  
8. catch (exception-declaration) {  
9.     // code that handles another exception type  
10. }  
11. catch (exception-declaration) {  
12.
```

1. 特定异常类型变量的声明, 如: `catch(bad_alloc& theexception)`
2. 如要捕捉所有的异常, 则: `catch(...)`



16.2 Scenario A: Handle exception thrown by C++ standard lib



Termination Model of Exception Handling

- ❑ 1. 抛出异常时, **try block**结束执行;
- ❑ 2. 寻找匹配的**catch handler** (*is-a*);
- ❑ 3. 执行**catch handler**代码;
- ❑ 4. 程序控制跳至最后一个**catch handler**后的首条语句. (注意: 不再执行**try block**中抛出异常点的后续语句)



16.2 Scenario A: Handle exception thrown by C++ standard lib



```
1.  try {
2.      // code that may throw exceptions
3.  }
4.  catch (exception-declaration) {
5.      // code that executes when
6.      // exception-declaration is thrown
7.  }
8.  catch (exception-declaration) {
9.      // code that handles another e
10. }
11. catch (exception-declaration) {
12. }
13. cout << "following statements";
```

抛出异常, skip try中的后续语句, 程序控制转至catch语句

若未匹配is-a, 转至下一条catch语句

若匹配, 执行异常处理代码

跳过剩余的catch, 执行后续的代码



16.2 Scenario A: Handle exception thrown by C++ standard lib



```
1.  int main()
2.  {
3.      Test t;
4.      double *ptr[ 50 ];
5.      try
6.      {
7.          for ( int i = 0; i < 50; i++ )
8.          {
9.              ptr[ i ] = new double[ 50000000 ]; // may throw exception
10.             cout << "Allocated 50000000 doubles in ptr[ " << i << " ]\n";
11.         }
12.     }
13.     catch ( bad_alloc &memoryAllocationException )// handle exception
14.     {
15.         cerr << "Exception occurred: "
16.             << memoryAllocationException.what() << endl;
17.     }
18.     cout << "Exception handled." << endl;
19.     return 0;
20. }
```

Constructor called.

Allocated 50000000 doubles in ptr[0]

Allocated 50000000 doubles in ptr[1]

Allocated 50000000 doubles in ptr[2]

Allocated 50000000 doubles in ptr[3]

Exception occurred: bad allocation

Exception handled.

Destructor ok.

exception类定义的虚函数, returns error message.



16.2 Scenario A: Handle exception thrown by C++ standard lib



- ❑ 修改1: `bad_alloc` → `exception`
- ❑ 修改2: `bad_alloc` → `logic_error`
- ❑ 修改3: `bad_alloc` → ...

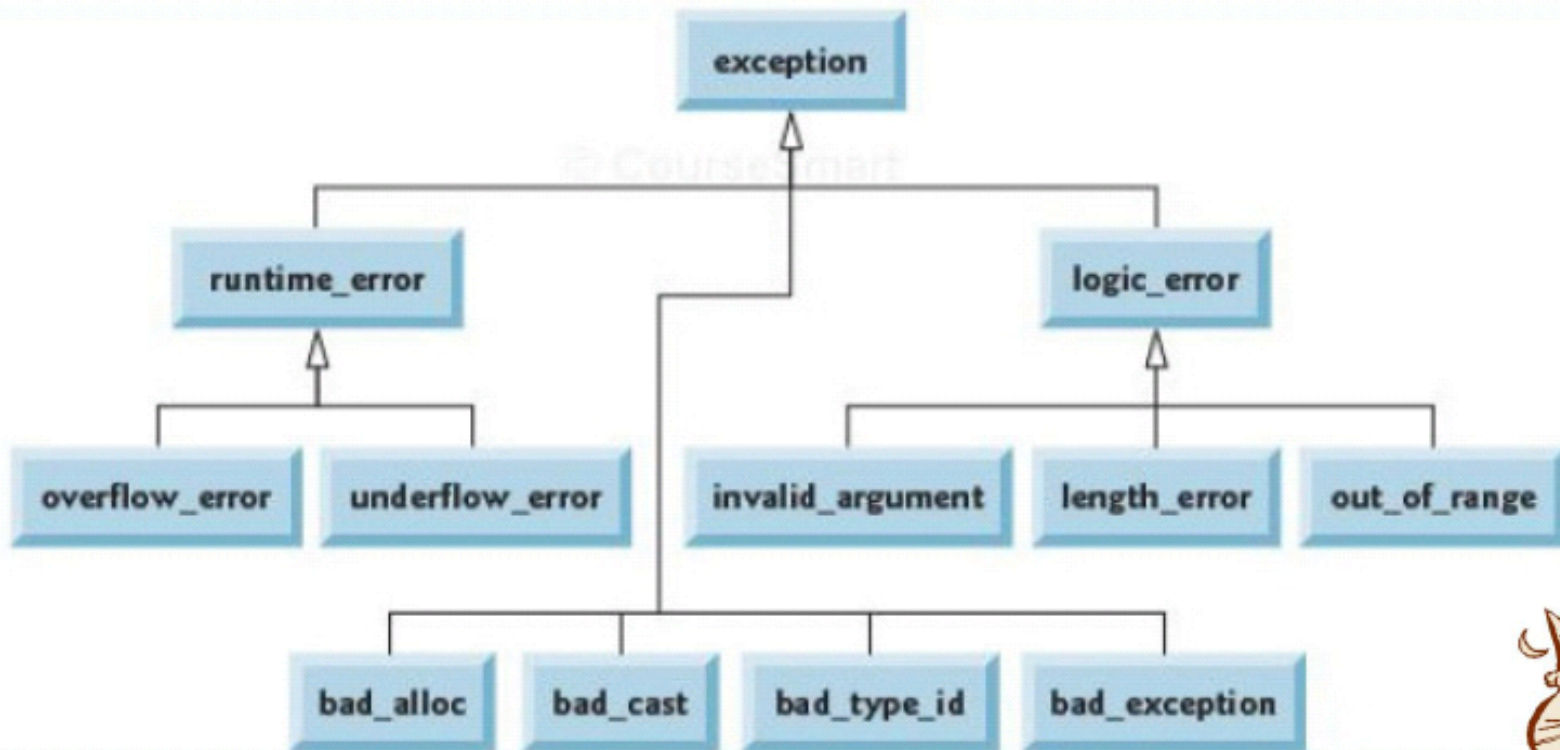


Figure 16.11. Standard Library exception classes





Topics



- ❑ 16.1 Introduction
- ❑ 16.2 Scenario A: Handle exception thrown by C++ standard lib
- ❑ **16.3 Scenario B: Define, throw and handle your own exception**
- ❑ 16.4 Stack Unwinding



16.3 Scenario B: Define, throw and handle your own exception



- ❑ 如何在自定义的函数中抛出异常?
- ❑ 需求: 设计 **quotient** 函数, 对用户输入的两个数进行除法操作, 希望输入的除数为0时能抛出异常, 由调用函数捕获并处理该异常
- ❑ **Exception Specifications** 异常说明



16.3 Scenario B: Define, throw and handle your own exception



// P612. Figure 16.1. Class DivideByZeroException definition

```
3.  #include <stdexcept>
4.  using std::runtime_error;
5.
6.  class DivideByZeroException : public runtime_error
7.  {
8.  public:
9.      DivideByZeroException::DivideByZeroException()
10.         : runtime_error( "attempted to divide by zero" ) {}
11. };
```

// P612. Figure 16.2. throws and handle exceptions

```
13. double quotient( int numerator, int denominator )
14. {
15.     if ( denominator == 0 )
16.         throw DivideByZeroException(); // terminate function
17.     return static_cast< double >( numerator ) / denominator;
18. }
```

what()输出的信息



16.3 Scenario B: Define, throw and handle your own exception



```
36. try
37. {
38.     result = quotient( number1, number2 );
39.     cout << "The quotient is: " << result << endl;
40. } // end try
41. catch ( DivideByZeroException &divideByZeroException )
42. {
43.     cout << "Exception occurred: "
44.         << divideByZeroException.what() << endl;
45. } // end catch
46.
47. cout << "\nEnter two integers (end-of-file to end): ";
```

Enter two integers (end-of-file to end): 10 6

The quotient is: 1.66667

Enter two integers (end-of-file to end): 10 0

Exception occurred: attempted to divide by zero



Topics



- ☐ 16.1 Introduction
- ☐ 16.2 Scenario A: Handle exception thrown by C++ standard lib
- ☐ 16.3 Scenario B: Define, throw and handle your own exception
- ☐ **16.4 Stack Unwinding**



16.4 Stack Unwinding



Stack Unwinding(栈展开机制)

- 1. 当某个函数(异常源)抛出异常, 将立即结束该函数的执行, 根据函数调用链回溯(可以是本函数)寻找可以catch该异常的Handler;
- 2. 如果找到了匹配的Handler, 则执行Stack Unwinding, 即依次释放从异常源Handler所在函数的所有局部对象;
- 3. 如果在main函数中仍没有找到匹配的Handler, 则调用terminate函数(该函数缺省调用abort, 不执行栈展开), 结束程序.



16.4 Stack Unwinding



```
1. void function3() throw ( runtime_error )
2. {
3.     cout << "In fun3\n";
4.     Test t(3);
5.     throw runtime_error( "runtime_error in fun3" );
6.
7.     cout << "Reach here? fun3\n";
8. }
```

```
1. void function2() throw ( runtime_error )
2. {
3.     Test t(2);
4.     cout << "fun3 is called inside fun2\n";
5.     function3();
6.     cout << "Reach here? fun2\n";
7. }
```

```
1. void function1() throw ( runtime_error )
2. {
3.     Test t(1);
4.     cout << "fun2 is called inside fun1\n" ;
5.     function2();
6.     cout << "Reach here? Fun1\n";
7. }
```

```
1. int main()
2. {
3.     try {
4.         cout << "fun1 is called inside main\n";
5.         function1();
6.         cout << "Reach here? fun main\n";
7.     }
8.     catch ( runtime_error &error ) {
9.         cout << "Exception occurred: "
10.            << error.what() << endl;
11.         cout << "Exception handled in main\n";
12.     }
13.     return 0;
14. }
```

fun1 is called inside main
Constructor 1
fun2 is called inside fun1
Constructor 2
fun3 is called inside fun2
In fun3
Constructor 3



16.4 Stack Unwinding



```
1. void function3() throw ( runtime_error )
2. {
3.     cout << "In fun3\n";
4.     Test t(3);
5.     throw runtime_error( "runtime_error in fun3" );
6.
7. cout << "Reach here? fun3\n";
8. }
```

④

```
1. void function2() throw ( runtime_error )
2. {
3.     Test t(2);
4.     cout << "fun3 is called inside fun2\n";
5.     function3();
6. cout << "Reach here? fun2\n";
7. }
```

⑤

```
1. void function1() throw ( runtime_error )
2. {
3.     Test t(1);
4.     cout << "fun2 is called inside fun1\n";
5.     function2();
6. cout << "Reach here? fun1\n";
7. }
```

⑥

```
1. int main()
2. {
3.     try {
4.         cout << "fun1 is called inside main\n";
5.         function1();
6. cout << "Reach here? fun-main\n";
7.     }
8.     catch ( runtime_error &error ) {
9.         cout << "Exception occurred: "
10.            << error.what() << endl;
11.         cout << "Exception handled in main\n";
12.     }
13.     return 0;
14. }
```

⑦

⑧

stack unwinding occur

Destructor 3

Destructor 2

Destructor 1

**Exception occurred: runtime_error
in fun3**

Exception handled in main



16.4 Stack Unwinding



- ❑ As control passes from a throw expression to a handler, destructors are invoked for all automatic objects constructed since the try block was entered. The automatic objects are destroyed **in the reverse order** of the completion of their construction.
- ❑ The process of calling destructors for automatic objects constructed on the path from a try block to a throw expression is called "***stack unwinding***".



16.4 Stack Unwinding



- 如果Exception Handler无法处理捕获的异常，
可以re-throw重新抛出异常：

throw;

- 根据栈展开机制，解读Fig 16.3

程序解读 (P494)



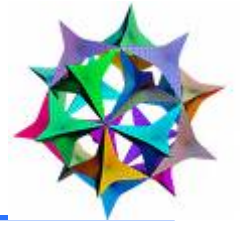
Summary



- 异常的概念
- try-throw-catch模块的语法和处理流程
- 栈展开过程（与构造和析构的关系）
- new异常的处理



Homework



☐ 实验必选题目:

16.23

☐ 实验任选题目:

☐ 作业题目: