Chapter 17

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

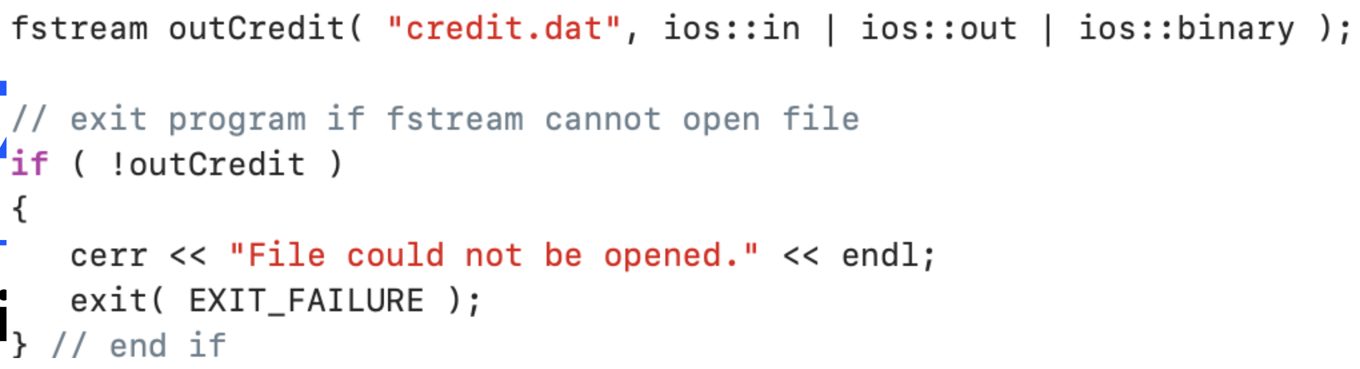
- **17.1 Introduction**
- **17.2 Scenario A: Handle exception thrown by C++ standard lib**
- **17.3 Scenario B: Define, throw and handle your own exception**
- **17.4 Stack Unwinding**

17.1 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

1



```
fstream outCredit( "credit.dat", ios::in | ios::out | ios::binary );

// exit program if fstream cannot open file
if ( !outCredit )
{
   cerr << "File could not be opened." << endl;
   exit( EXIT_FAILURE );
} // end if
```

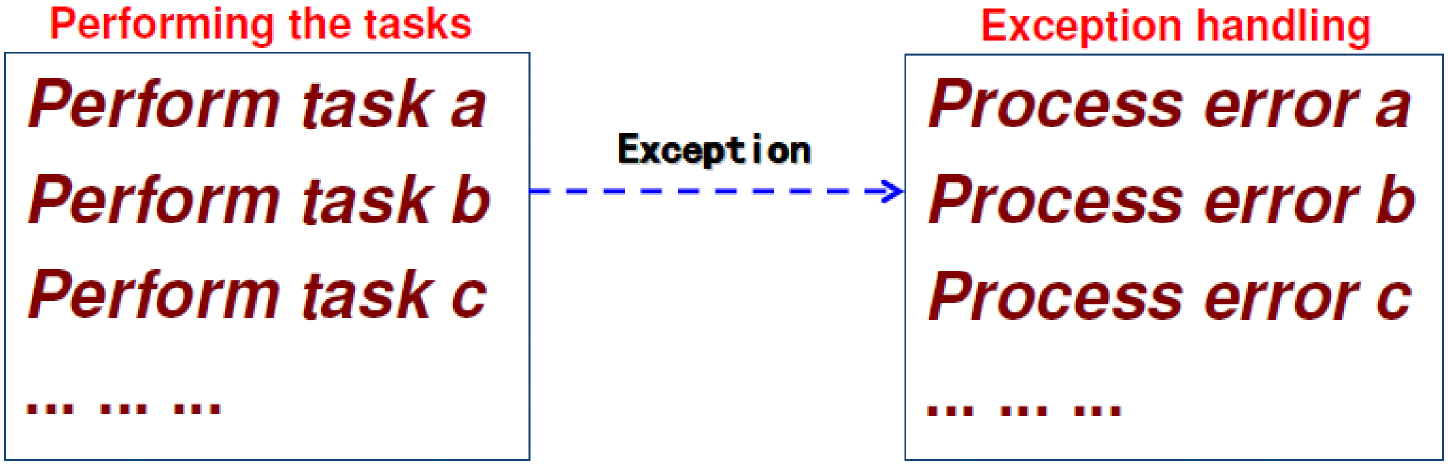
- ❑ **Intermixi**

```
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**

17.1 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.**



Performing the tasks

Perform task a

Perform task b

Perform task c

... ...

Exception

Exception handling

Process error a

Process error b

Process error c

... ...

- try-catch

robust(健壮性) and fault-tolerant(容错)

17.1 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.

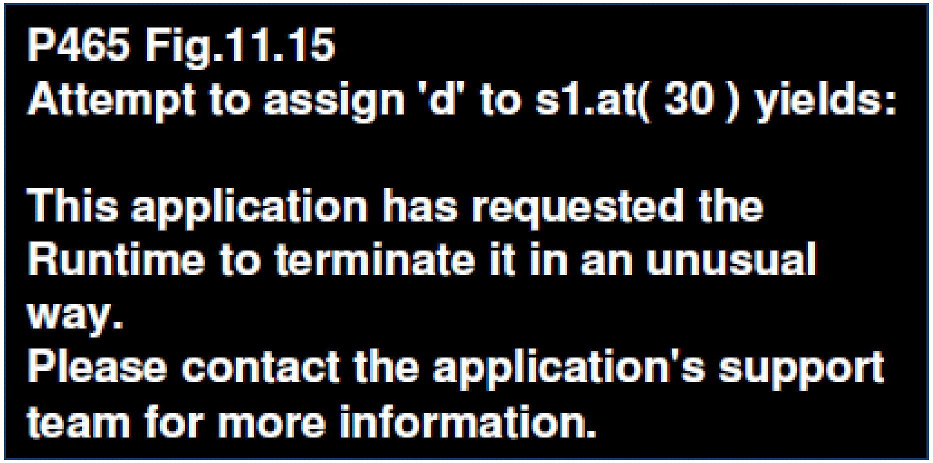
Topics

- 17.1 Introduction
- **17.2 Scenario A: Handle exception thrown by C++ standard lib**
- 17.3 Scenario B: Define, throw and handle your own exception
- 17.4 Stack Unwinding

17.2 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.
```

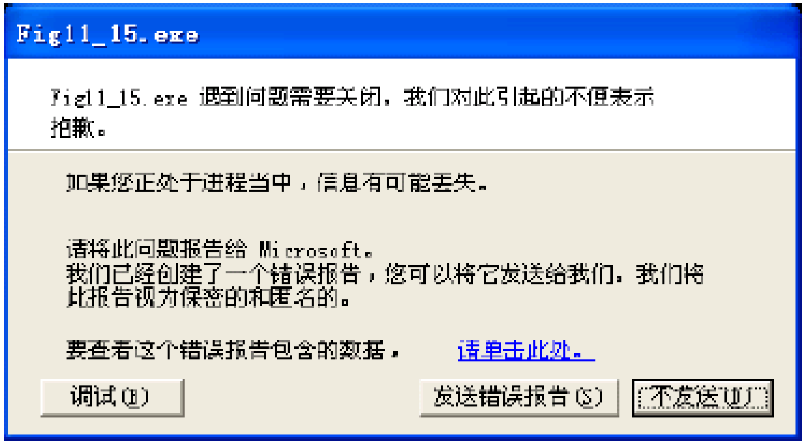


Fig11_15.exe

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。

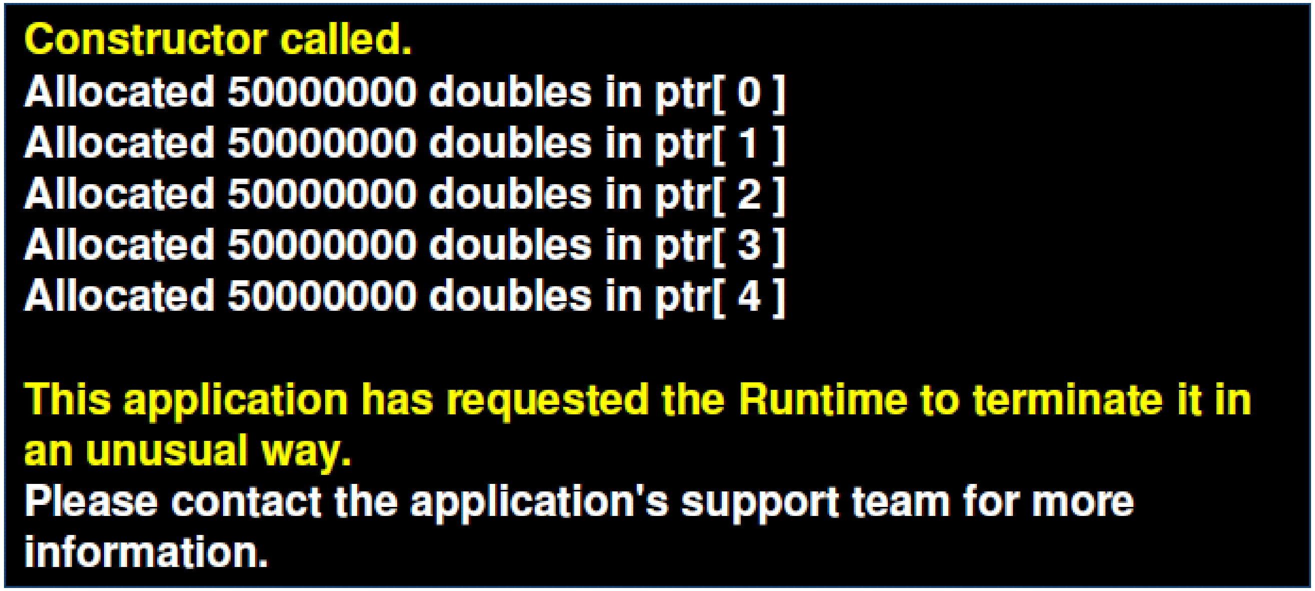
要查看这个错误报告包含的数据，请单击此处。

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

```
class Test{
public:
    Test(){ cout << "Constructor called." << endl; }
    ~Test(){ cout << "Destructor ok." << endl; }
};
int main()
{
    Test t;
    double *ptr[ 50 ];

    for ( int i = 0; i < 50; i++ )
    {
        ptr[ i ] = new double[ 50000000 ];
        cout << "Allocated 50000000 doubles in ptr[ " << i << " ]\n";
    }
    return 0;
}
```

17.2 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.
```

17.2 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

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

关键词try, “包裹”可能出现异常的compound statement

```
try {
    // code that may throw exceptions
}
catch (exception-declaration) {
    // code that executes when
    // exception-declaration is thrown
}
catch (exception-declaration) {
    // code that handles another exception type
}
catch (exception-declaration) {

```

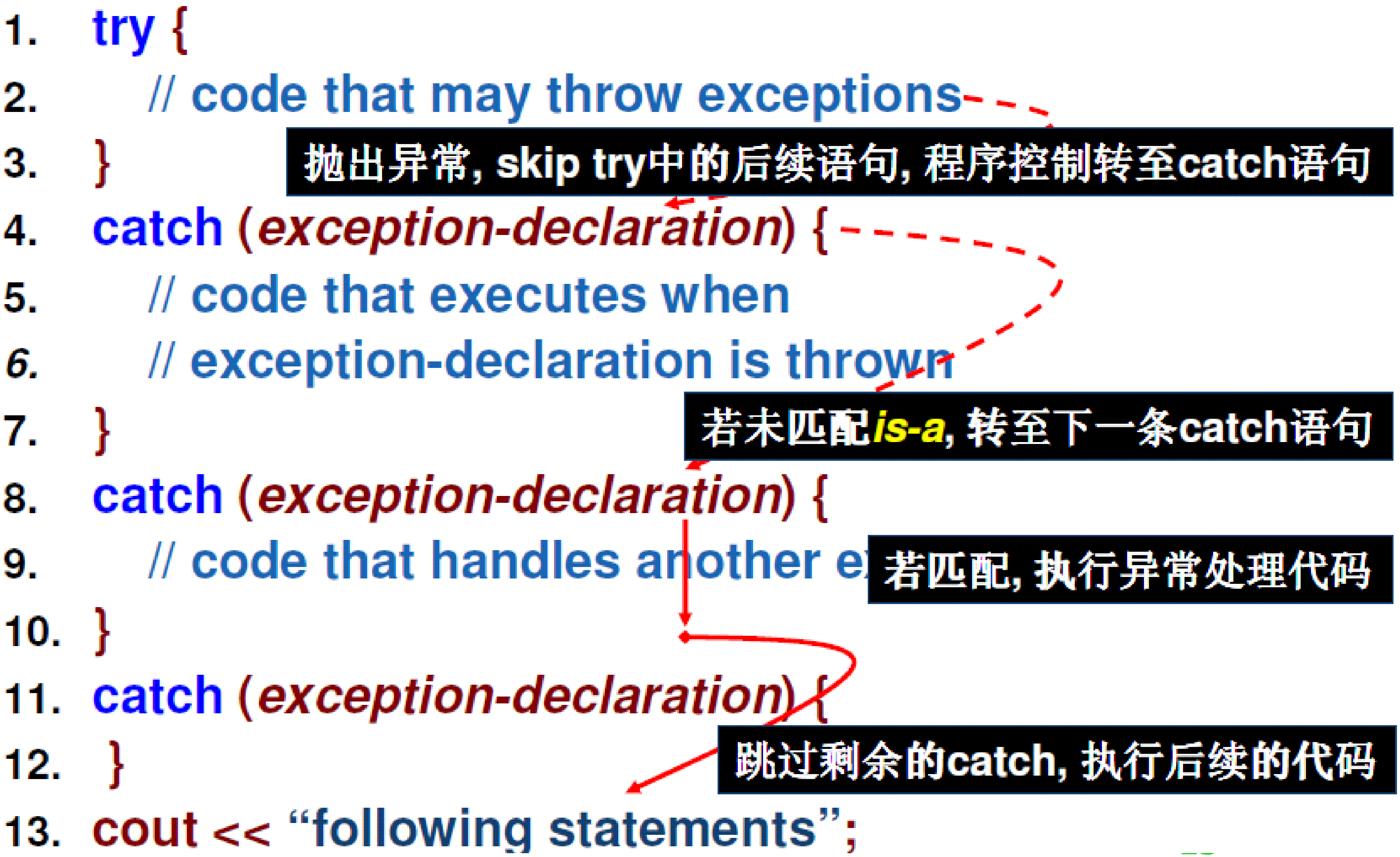
1. 特定异常类型变量的声明, 如: catch(*bad_alloc& theexception*)
2. 如要捕捉所有的异常, 则: catch(…)

17.2 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)**

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



```
try {
    // code that may throw exceptions
}
catch (exception-declaration) {
    // code that executes when
    // exception-declaration is thrown
}
catch (exception-declaration) {
    // code that handles another e
}
catch (exception-declaration) {
 }
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

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

```
int main()
{
    Test t;
    double *ptr[ 50 ];
    try
    {
        for ( int i = 0; i < 50; i++ )
        {
            ptr[ i ] = new double[ 50000000 ]; // may throw exception
            cout << "Allocated 50000000 doubles in ptr[ " << i << " ]\n";
        }
    }
    catch ( bad_alloc &memoryAllocationException )// handle exception
    {
        cerr << "Exception occurred: "
            << memoryAllocationException.what() << endl;
    }
    cout << "Exception handled." << endl;
    return 0;
}
```

```
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.

17.2 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 → …

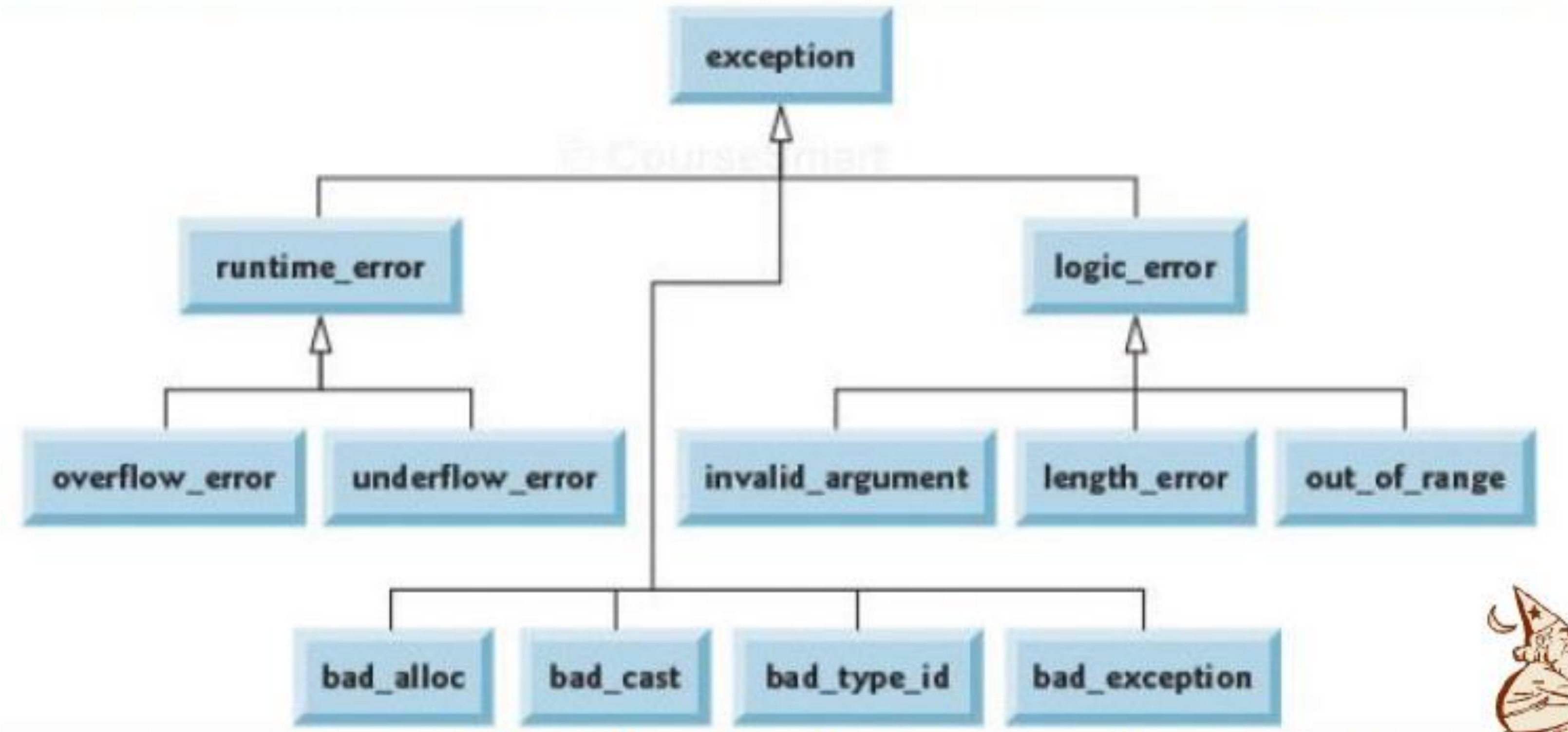


Figure 16.11. Standard Library exception classes

Topics

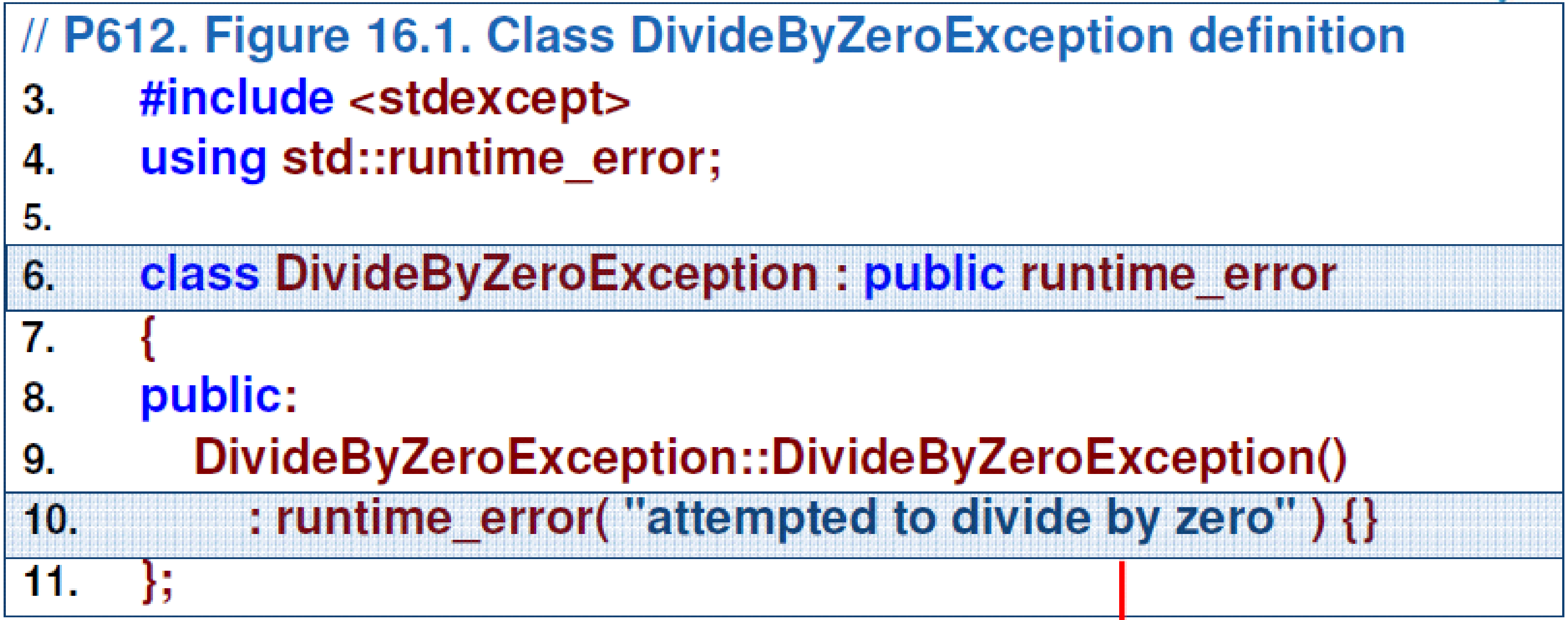
- 17.1 Introduction
- 17.2 Scenario A: Handle exception thrown by C++ standard lib
- **17.3 Scenario B: Define, throw and handle your own exception**
- 17.4 Stack Unwinding

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

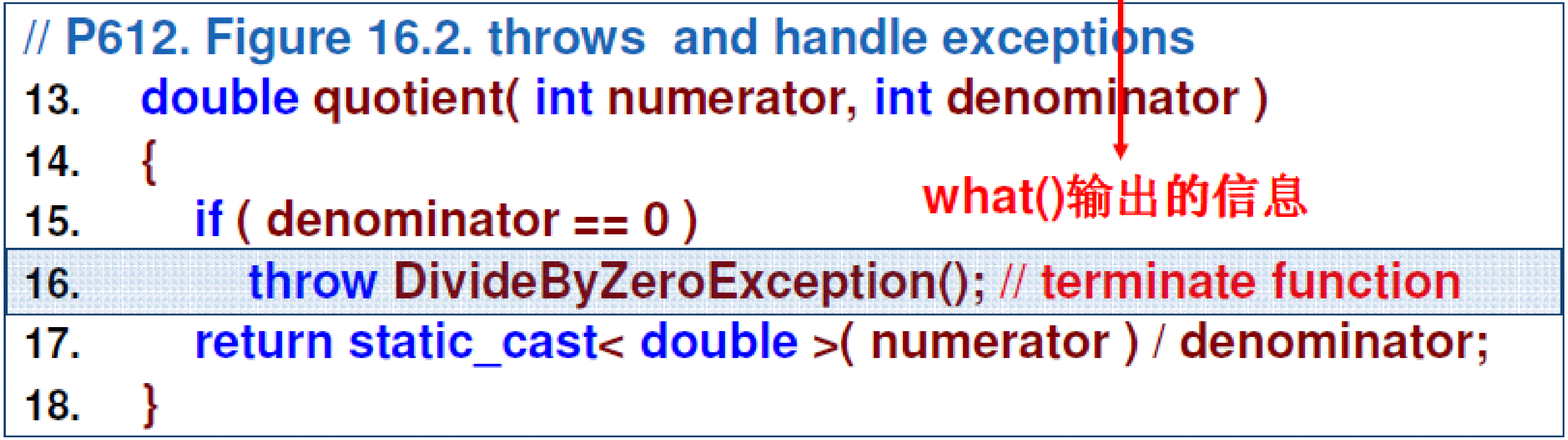
- ❑如何在自定义的函数中抛出异常?
- ❑需求: 设计quotient函数, 对用户输入的两个数进行除法操作, 希望输入的除数为0时能抛出异常, 由调用函数捕获并处理该异常

- ❑Exception Specifications 异常说明

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



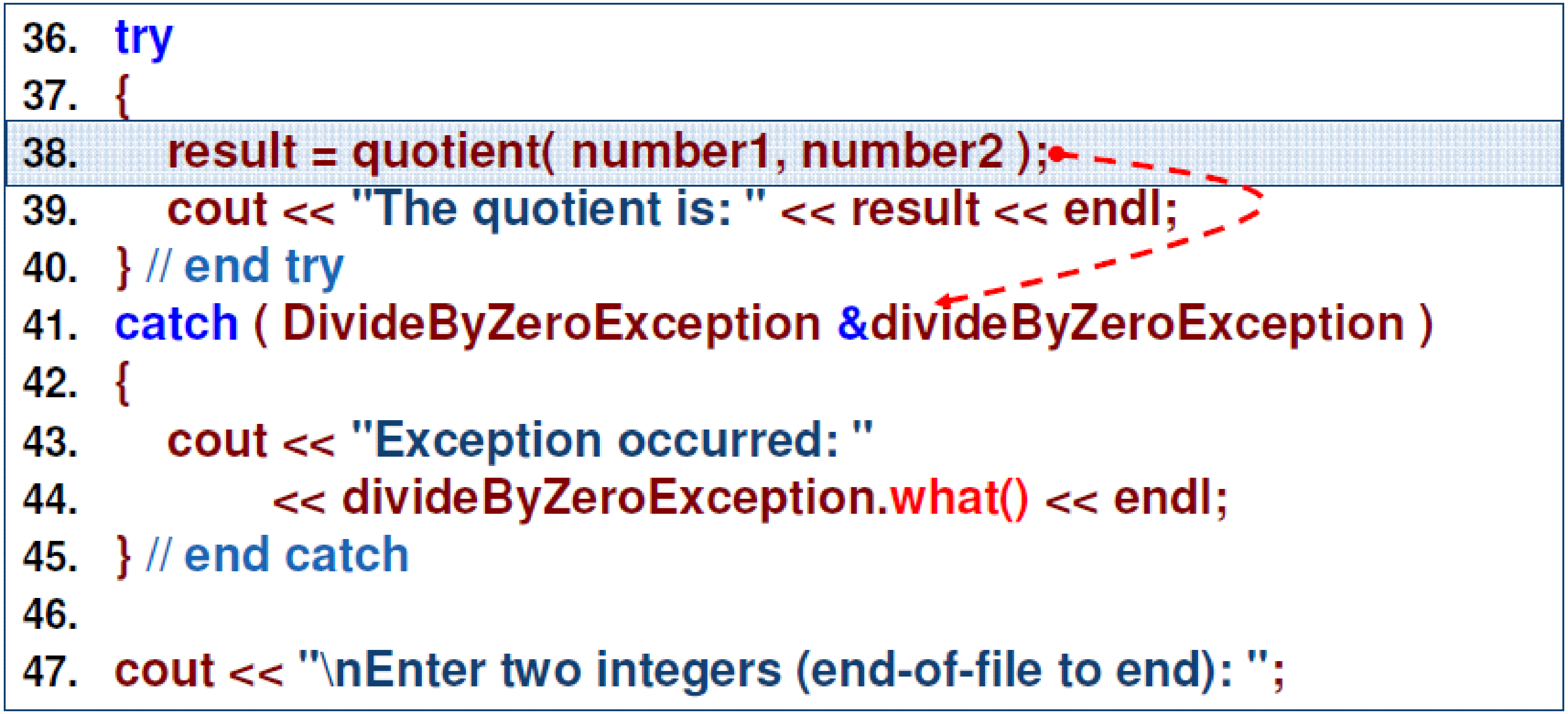
```
// P612. Figure 16.1. Class DivideByZeroException definition
#include <stdexcept>
using std::runtime_error;

class DivideByZeroException : public runtime_error
{
public:
   DivideByZeroException::DivideByZeroException()
      : runtime_error( "attempted to divide by zero" ) {}
};
```



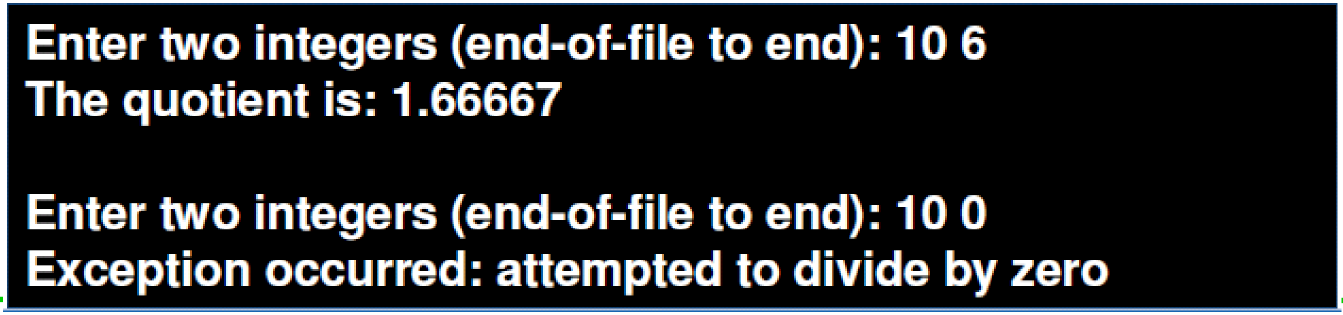
```
// P612. Figure 16.2. throws  and handle exceptions
double quotient( int numerator, int denominator )
{
   if ( denominator == 0 )
      throw DivideByZeroException(); // terminate function
   return static_cast< double >( numerator ) / denominator;
}
```

what()输出的信息

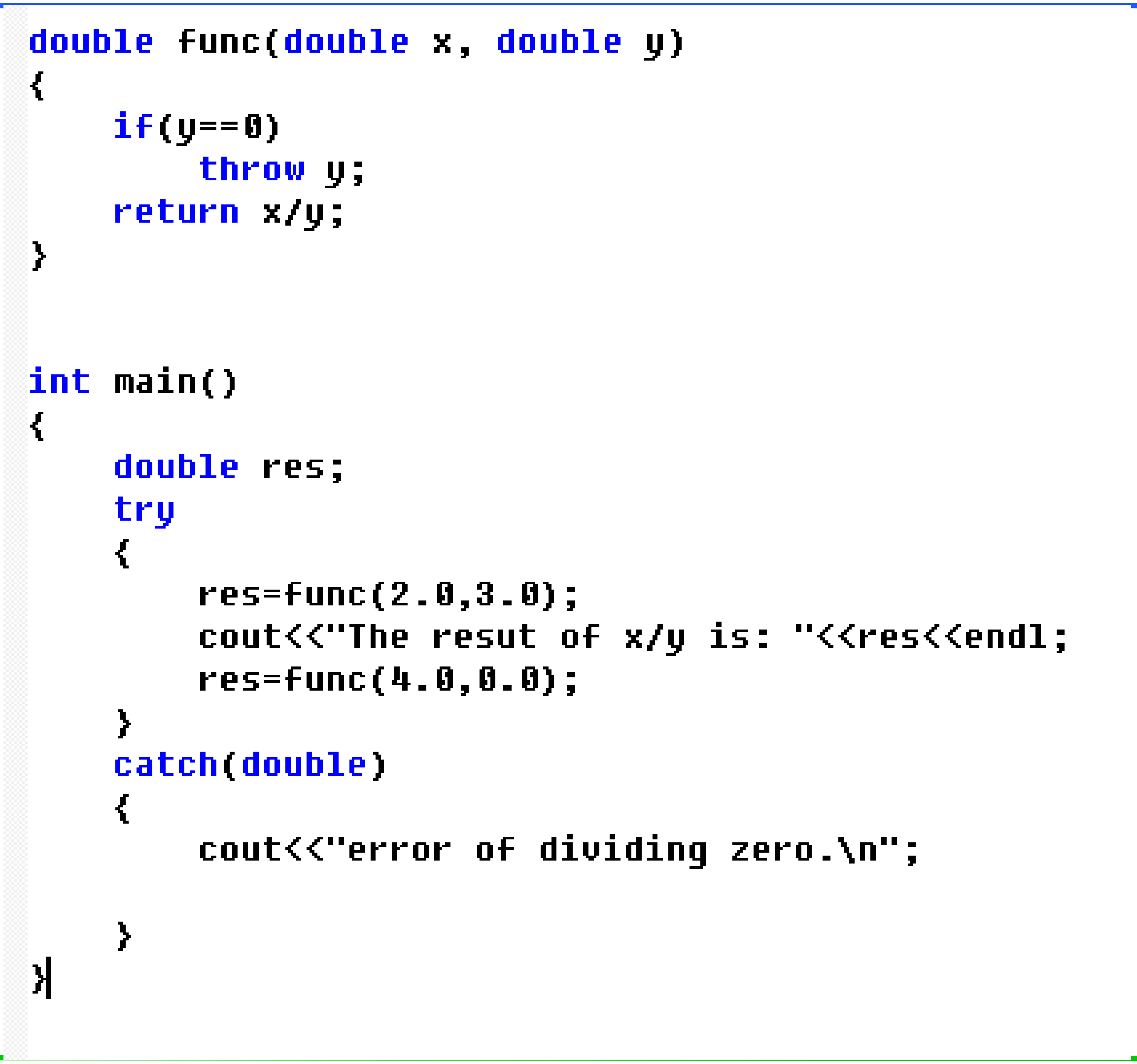
17.3 Scenario B: Define, throw and handle your own exception



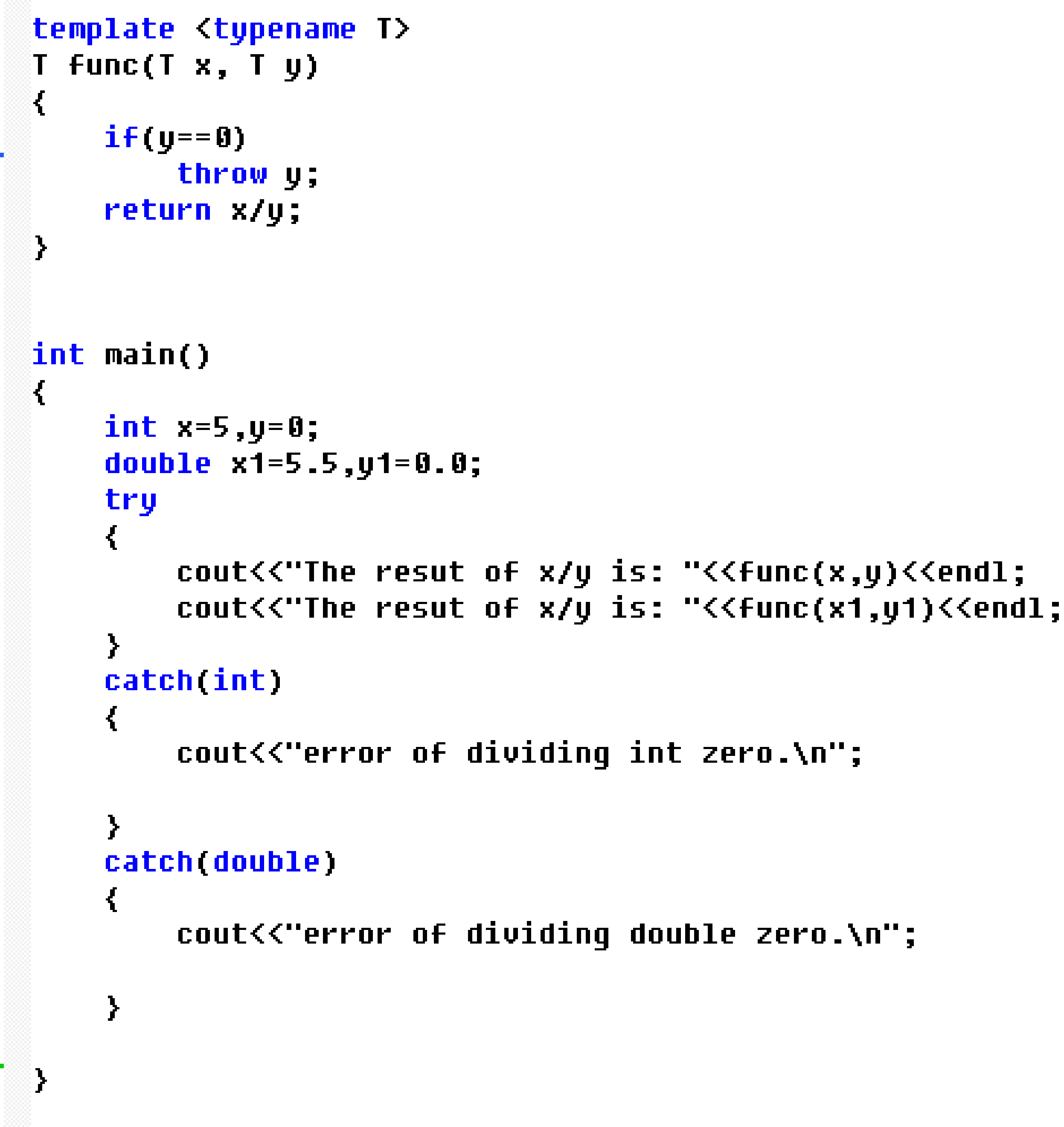
```cpp
try
{
   result = quotient( number1, number2 );
   cout << "The quotient is: " << result << endl;
} // end try
catch ( DivideByZeroException &divideByZeroException )
{
   cout << "Exception occurred: "
         << divideByZeroException.what() << endl;
} // end catch

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
```



```
double func(double x, double y)
{
    if(y==0)
        throw y;
    return x/y;
}


int main()
{
    double res;
    try
    {
        res=func(2.0,3.0);
        cout<<"The resut of x/y is: "<<res<<endl;
        res=func(4.0,0.0);
    }
    catch(double)
    {
        cout<<"error of dividing zero.\n";

    }
}
```



```
template <typename T>
T func(T x, T y)
{
    if(y==0)
        throw y;
    return x/y;
}


int main()
{
    int x=5,y=0;
    double x1=5.5,y1=0.0;
    try
    {
        cout<<"The resut of x/y is: "<<func(x,y)<<endl;
        cout<<"The resut of x/y is: "<<func(x1,y1)<<endl;
    }
    catch(int)
    {
        cout<<"error of dividing int zero.\n";

    }
    catch(double)
    {
        cout<<"error of dividing double zero.\n";

    }

}
```

Topics

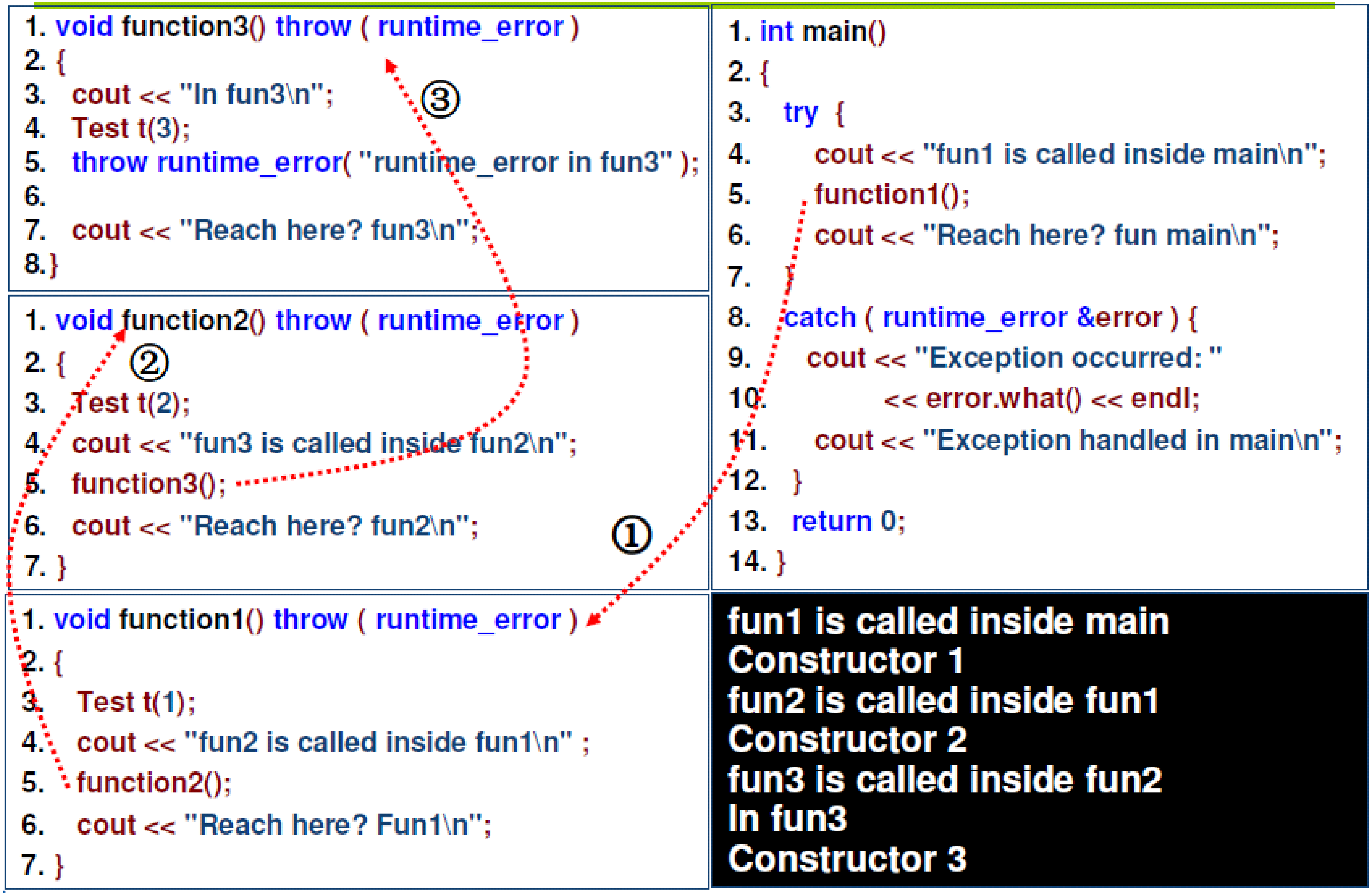
- ❑ 17.1 Introduction
- ❑ 17.2 Scenario A: Handle exception thrown by C++ standard lib
- ❑ 17.3 Scenario B: Define, throw and handle your own exception
- ❑ **17.4 Stack Unwinding**

17.4 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.

17.4 Stack Unwinding



```
void function3() throw ( runtime_error )
{
  cout << "In fun3\n";
  Test t(3);
  throw runtime_error( "runtime_error in fun3" );

  cout << "Reach here? fun3\n";
}
```

```
void function2() throw ( runtime_error )
{
  Test t(2);
  cout << "fun3 is called inside fun2\n";
  function3();
  cout << "Reach here? fun2\n";
}
```

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

```
int main()
{
  try  {
     cout << "fun1 is called inside main\n";
     function1();
     cout << "Reach here? fun main\n";
  }
  catch ( runtime_error &error ) {
    cout << "Exception occurred: "
              << error.what() << endl;
     cout << "Exception handled in main\n";
  }
  return 0;
}
```

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

17.4 Stack Unwinding

```
void function3() throw ( runtime_error )
{
  cout << "In fun3\n";
  Test t(3);
  throw runtime_error( "runtime_error in fun3" );

  ~~cout << "Reach here? fun3\n";~~
}
```

④

```
void function2() throw ( runtime_error )
{
  Test t(2);
  cout << "fun3 is called inside fun2\n";
  function3();
  ~~cout << "Reach here? fun2\n";~~
}
```

```
void function1() throw ( runtime_error )
{
  Test t(1);
  cout << "fun2 is called inside fun1\n" ;
  function2();
  ~~cout << "Reach here? fun1\n";~~
}
```

⑤ ⑥

```
int main()
{
  try  {
      cout << "fun1 is called inside main\n";
      function1();
      ~~cout << "Reach here? fun main\n";~~
  }
  catch ( runtime_error &error ) {
     cout << "Exception occurred: "
             << error.what() << endl;
      cout << "Exception handled in main\n";
  }
  return 0;
}
```

⑦ ⑧

stack unwinding occur

```
Destructor 3
Destructor 2
Destructor 1
Exception occurred: runtime_error
in fun3
Exception handled in main
```

17.4 Stack Unwinding

- **As control passes from a throwexpression 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*".**

17.4 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 17.3

```cpp
// Fig. 16.3: Fig16_03.cpp
// Demonstrating exception rethrowing.
#include <iostream>
using std::cout;
using std::endl;

#include <exception>
using std::exception;

// throw, catch and rethrow exception
void throwException()
{
   // throw exception and catch it immediately
   try
   {
      cout << "  Function throwException throws an exception\n";
      throw exception(); // generate exception
   } // end try
   catch ( exception & ) // handle exception
   {
      cout << "  Exception handled in function throwException"
         << "\n  Function throwException rethrows exception";
      throw; // rethrow exception for further processing
   } // end catch

   cout << "This also should not print\n";
} // end function throwException
```

Outline

Fig17_03.cpp

(1 of 2)

```cpp

int main()
{
   // throw exception
   try
   {
      cout << "\nmain invokes function throwException\n";
      throwException();
      cout << "This should not print\n";
   } // end try
   catch ( exception & ) // handle exception
   {
      cout << "\n\nException handled in main\n";
   } // end catch

   cout << "Program control continues after catch in main\n";
   return 0;
} // end main
```

```
main invokes function throwException
  Function throwException throws an exception
  Exception handled in function throwException
  Function throwException rethrows exception

Exception handled in main
Program control continues after catch in main
```

Outline

Fig17_03.cpp

(2 of 2)

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：

  Ex4