16

Exception Handling

OBJECTIVES

In this chapter you will learn:

- What exceptions are and when to use them.
- To use `try`, `catch` and `throw` to detect, handle and indicate exceptions, respectively exceptions.
- 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 use `auto_ptr` to prevent memory leaks.
- To understand the standard exception hierarchy.

Assignment Checklist

Name: ____________________ Date: ____________________

Section: ____________________

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities		
Matching	YES NO	
Fill in the Blank	9, 10, 11, 12, 13, 14	
Short Answer	15, 16, 17	
Programming Output	18, 19	
Correct the Code	20, 21	
Lab Exercises		
Lab Exercise 1 — `numberVerifier`	YES NO	
Follow-Up Questions and Activities	1, 2	
Lab Exercise 2 — Destructors	YES NO	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
Postlab Activities		
Coding Exercise	1	

Prelab Activities

Matching

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

After reading Chapter 16 of *C++ How to Program: Fifth Edition*, answer the given questions. These questions are intended to test and reinforce your understanding of key concepts and may be done either before the lab or during the lab.

For each term in the column on the left, write the corresponding letter for the description that best matches it from the column on the right.

	Term	Description
___	1. `catch` block	a) Helps improve a program's fault tolerance.
___	2. `auto_ptr`	b) Encloses the code that may generate an exception.
___	3. Exception handling	c) Exception thrown when `new` fails.
___	4. `catch(...)`	d) Indicates that a function does not throw exceptions.
___	5. `try` block	e) Location in the program at which an exception occurs.
___	6. `bad_alloc`	f) "Catch all" handler that catches any exception.
___	7. Throw point	g) Encloses the code that is executed when an exception is caught.
___	8. `throw()`	h) Class template that helps avoid memory leaks.

Prelab Activities

Name:

Fill in the Blank

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

Fill in the blank for each of the following statements:

9. When an exception is not caught in a program, function _____________ is called.

10. Exception handling is designed for dealing with _____________ errors (i.e., errors that occur as the result of a program's execution).

11. Typically, exception handling deals with errors in a different _____________ from that which detected the error.

12. Class _____________ is the base class for the exception hierarchy.

13. Once an exception is thrown, control cannot return directly to the _____________.

14. _____________ catches all exceptions.

Prelab Activities

Name:

Short Answer

Name: ______________________________ **Date:** ______________________________

Section: ______________________________

In the space provided, answer each of the given questions. Your answers should be as concise as possible; aim for two or three sentences.

15. Describe two ways of detecting memory allocation failures.

16. If an exception is thrown, but not caught in a particular function's scope, to what point does control return? What is the name for this process?

17. If an exception of type `X` is thrown, which catch handlers would be able to catch this exception?

Prelab Activities

Name:

Programming Output

Name: ______________________ **Date:** ______________________

Section: ______________________

For each of the given program segments, read the code and write the output in the space provided below each program. [*Note:* Do not execute these programs on a computer.]

18. What is output by the following program?

```cpp
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

// class DivideZero definition
class DivideZero {

public:

   // constructor
   DivideZero()
      : out( "EXCEPTION: Division by zero attempted." )
   {
      // empty
   } // end class DivideZero exception

   // function display definition
   string display() const
   {
      return out;
   } // end function display
private:
   string out;
}; // end class DivideZero

// function arithmetic definition
double arith( int n, int d )
{
   if ( d == 0 )
      throw DivideZero();

   return static_cast< double > ( n ) / d;
} // end function arithmetic

int main()
{
   try
   {
      cout << arith( 24, 6 ) << endl;
      cout << arith( 1, 3 ) << endl;
```

Prelab Activities

Name:

Programming Output

```
      cout << arith( 9, 0 ) << endl;
   } // end try
   catch ( DivideZero &e )
   {
      cout << e.display() << endl;
   } // end catch

   return 0;
} // end main
```

Your Answer:

19. What is output by the following program?

```
#include <iostream>

using std::cout;
using std::endl;

#include <stdexcept>

using std::runtime_error;

// function function3 definition
void function3() throw ( runtime_error )
{
   throw runtime_error( "runtime_error in function3" );
} // end function function3

// function function2 definition
void function2() throw ( runtime_error )
{
   function3();
} // end function function2

// function function1 definition
void function1() throw ( runtime_error )
{
   try
   {
      function2();
   } // end try
   catch ( runtime_error e )
   {
      cout << "Exception occurred:\n" << e.what()
           << "; caught in function1\n";
      throw;
   } // end catch
} // end function function1
```

Prelab Activities

Name:

Programming Output

```

int main()
{
   try
   {
      function1();
   } // end try
   catch ( runtime_error &e )
   {
      cout << "Exception occurred:\n" << e.what()
           << "; caught in main\n";
   } // end catch

   return 0;
} // end main
```

Your answer:

Prelab Activities

Name:

Correct the Code

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic error or a syntax compilation error, circle the error in the program, and write the corrected code in the space provided after each problem. If the code does not contain an error, write "no error." [*Note*: It is possible that a program segment may contain multiple errors.]

20. The following program segment should `catch` two exception types thrown by `function1`.

```
try
{
   function1();
} // end try
catch ( runtime_error &r, DivideByZeroException &z )
{
   cout << "Exception( s ) occurred:\n" << r.what() << " and "
        << z.what() << "; caught in main\n";
} // end catch
```

Your answer:

Prelab Activities

Name:

Correct the Code

21. The following program segment should `catch` `runtime_errors`, `DividebyZeroExceptions` or any other exception thrown by `function2`.

```
try
{
   function2();
} // end try
catch ( ... )
catch ( runtime_error &r )
{
   cout << "Exception occurred:\n" << r.what()
} // end catch
catch ( DivideByZeroException &z )
{
   cout << "Exception occurred:\n" << z.what()
} // end catch
```

Your answer:

Lab Exercises

Lab Exercise 1 — NumberVerifier

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 16.1)
5. Problem-Solving Tips
6. Follow-Up Questions and Activities

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up questions. The source code for the template is available at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 16 of *C++ How To Program: Fifth Edition*. In this lab, you will practice:

- Writing a class to determine whether a string contains digit characters.
- Using exception handling to handle non-numeric inputs.

The follow-up questions and activities also will give you practice:

- Using exception handling to handle inputs that are too large.
- Contrasting `catch` statements with `catch(...)` statements.

Problem Description

Write a short program that reads a number from the user and stores the number as a string of characters. Convert this string to an integer. Before conversion, test for a `NonNumber` exception, which occurs if one or more of the characters is not a digit. Your program should not `throw` an exception if it detects a `-` sign before the number.

Lab Exercises

Name:

Lab Exercise 1 — NumberVerifier

Sample Output

```
Please enter a number (end-of-file to terminate): 4
The number entered was: 4

Please enter a number (end-of-file to terminate): 28
The number entered was: 28

Please enter a number (end-of-file to terminate): -257
The number entered was: -257

Please enter a number (end-of-file to terminate): a23
INVALID INPUT: non-integer detected

Please enter a number (end-of-file to terminate): 34k3
INVALID INPUT: non-integer detected

Please enter a number (end-of-file to terminate): -3413-3
INVALID INPUT: non-integer detected

Please enter a number (end-of-file to terminate): -4-8
INVALID INPUT: non-integer detected

Please enter a number (end-of-file to terminate): ^Z
```

Template

```cpp
// Lab 1: numberverifier.cpp
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

#include <cmath>

#include <string>
using std::string;

#include <stdexcept>
using std::runtime_error;

// class NonNumber definition
class NonNumber : public runtime_error
{
public:
   // constructor
   NonNumber()
      : runtime_error( "non-integer detected" )
   {
      // empty
   } // end class NonNumber definition

   /* write definition for member function what */
private:
   string message;
}; // end class NonNumber

```

Fig. L 16.1 | `numberverifier.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 1 — NumberVerifier

```
// function castInput definition
int castInput( string input )
{
   int result = 0;
   int negative = 1;

   // check for minus sign
   if ( input[ 0 ] == '-' )
     negative = -1;

   for ( int i = input.length() - 1, j = 0; i >= 0; i--, j++ )
   {
      if ( negative == -1 && i == 0 )
         continue;

      if ( input[ i ] >= '0' && input[ i ] <= '9' )
         result += static_cast< int >( input[ i ] - '0' ) * pow( 10.0, j );
      else
         /* Write code to throw NonNumber exception */
   } // end for

   return result * negative;
} // end function castInput

int main()
{
   string input;
   int convert;

   cout << "Please enter a number (end-of-file to terminate): ";

   while ( cin >> input )
   {
      /* Write try block that calls castInput */
      /* Write catch handler that catches any exceptions
         that the call to castInput might have thrown */

      cout << "\n\nPlease enter a number (end-of-file to terminate): ";
   }

   cout << endl;
   return 0;
} // end main
```

Fig. L 16.1 | `numberverifier.cpp`. (Part 2 of 2.)

Problem-Solving Tips

1. To determine whether the input is a valid number, the program checks for any non-digit character in the input `string`. The only non-digit character that is allowed is a minus sign (-) at the beginning of the `string`.
2. Make sure that any code that could throw an exception is enclosed within a `try` statement with a matching `catch` handler.

Lab Exercises

Name:

Lab Exercise 1 — `NumberVerifier`

Follow-Up Questions and Activities:

1. Modify the program by creating an exception class `Overflow` for detecting whether the user input “fits” into an `int` variable. For the purpose of this exercise, any input longer than 10 digits should generate an overflow error. Modify function `castInput` to check for this error, and add an appropriate `catch` handler in `main` to handle this type of exception. A typical run of your program should look like this:

```
Please enter a number (end-of-file to terminate): 44
The number entered was: 44

Please enter a number (end-of-file to terminate): -44
The number entered was: -44

Please enter a number (end-of-file to terminate): p25
INVALID INPUT: non-integer detected

Please enter a number (end-of-file to terminate): 123456789
The number entered was: 123456789

Please enter a number (end-of-file to terminate): 12345678901
INVALID INPUT: overflow detected

Please enter a number (end-of-file to terminate): ^Z
```

2. In your solution to *Follow-Up Question 1*, replace the second `catch` with a `catch( ... )` statement. What changes occur?

Lab Exercises

Name:

Lab Exercise 2— Destructors

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into five parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 16.2–Fig. L 16.4)
5. Problem-Solving Tips

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. The source code for the template is available at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 16 of *C++ How To Program: Fifth Edition*. In this lab, you will practice:

- Observing how destructors are called when an exception is thrown.

Problem Description

Write a program illustrating that all destructors for objects constructed in a block are called before an exception is thrown from that block.

Sample Output

```
TestObject 1 constructor
TestObject 2 constructor
TestObject 3 constructor

TestObject 3 destructor
TestObject 2 destructor
TestObject 1 destructor
This is a test exception
```

Lab Exercises

Name:

Lab Exercise 2— Destructors

Template

```cpp
// Lab 2: TestObject.h
// Class TestObject definition.
class TestObject
{
public:
   TestObject( int ); // constructor takes int parameter
   ~TestObject(); // destructor
private:
   int value;
}; // end class TestObject
```

Fig. L 16.2 | `TestObject.h`.

```cpp
// Lab 2: TestObject.cpp
// Class TestObject member function definition.
#include <iostream>
using std::cout;

#include "TestObject.h"

// constructor takes int parameter
TestObject::TestObject( int val ) : value( val )
{
   /* Write code to display a message announcing that this
      constructor has been called, include data member value */
} // end TestObject constructor

// destructor
TestObject::~TestObject()
{
   /* Write code to display a message announcing that this
      destructor has been called, include data member value */
} // end TestObject destructor
```

Fig. L 16.3 | `TestObject.cpp`.

```cpp
// Lab 2: destructors.cpp
#include <iostream>
using std::cout;
using std::cerr;

#include <stdexcept>
using std::runtime_error;

#include "TestObject.h"

int main()
{
   try // create objects and throw exception
   {
      // create three TestObjects
      /* Write declarations for three TestObjects */
      cout << '\n';
```

Fig. L 16.4 | `destructors.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 2— Destructors

```

      // throw an exception to show that all three Objects created above
      // will have their destructors called before the block expires
      /* Write code to throw a runtime_error */
   } // end try
   /* Write a catch header to catch the runtime_error */
   {
      /* Write code to output the error message
         to the standard error stream */
   } // end catch

   return 0;
} // end main
```

Fig. L 16.4 | `destructors.cpp`. (Part 2 of 2.)

Problem-Solving Tips

1. Display a message inside the constructor and destructor of class `TestObject`.
2. Declare three `TestObjects` inside the `try` block. Their destructors will be called when the exception is thrown.
3. Have the `catch` block display a message so the user knows when the exception has been handled.

Lab Exercises

Name:

Debugging

Name: ____________________________ Date: ____________________________

Section: ____________________________

The program (Fig. L 16.5) in this section does not run properly. Fix all the compilation errors so that the program will compile successfully. Once the program compiles, compare the output with the sample output, and eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the program's code has been corrected.

Sample Output

```
Enter an integer from 1 to 100 (-1 to end): 54
Enter an integer from 1 to 100 (-1 to end): 12
54 / 12 = 4.5

Enter an integer from 1 to 100 (-1 to end): 93
Enter an integer from 1 to 100 (-1 to end): 32
93 / 32 = 2.90625

Enter an integer from 1 to 100 (-1 to end): a
Exception occurred: entered input of the wrong data type
Enter an integer from 1 to 100 (-1 to end): -4
Exception occurred: entered a number not in the valid range
Enter an integer from 1 to 100 (-1 to end): 132
Exception occurred: entered a number not in the valid range
Enter an integer from 1 to 100 (-1 to end): -1
An unknown exception has occurred, exiting the program
```

Broken Code

```
// Debugging: debugging.cpp

#include <iostream>

using std::cout;
using std::cin;
using std::endl;
using std::ios;

#include <exception>

using std::exception;

```

Fig. L 16.5 | `debugging.cpp`. (Part 1 of 3.)

Lab Exercises

Name:

Debugging

```
// class InvalidInputTypeException definition
class InvalidInputTypeException
{
public:
   // constructor
   InvalidInputTypeException()
      : message( "entered input of the wrong data type" )
   {
      // empty

   } // end class InvalidInputTypeException

   // function what definition
   const char *what() const
   {
      return message.c_str();

   } // end function what
private:
   string message;
}; // end class InvalidInputTypeException

// class OutOfRangeException definition
class OutOfRangeException
{
public:
   // constructor
   exception OutOfRangeException()
      : message( "entered a number not in the valid range" )
   {
      // empty
   } // end class OutOfRangeException constructor

   // function what definition
   const char *what() const
   {
      return message.c_str();
   } // end function what
private:
   string message;
}; // end class OutOfRangeException

// function inputNumber definition
int inputNumber()
{
   int number;

   cout << "Enter an integer from 1 to 100 (-1 to end): ";
   cin >> number;

   if ( cin.fail() == 1 )
      throw( InvalidInputTypeException );

   if ( number > 100 || number < 1 )
      throw exception( OutOfRangeException() );

```

Fig. L 16.5 | debugging.cpp. (Part 2 of 3.)

Lab Exercises

Name:

Debugging

```
   if ( num == -1 )
      throw;

   return number;
} // end function inputNumber

int main()
{
   int num1 = 0;
   int num2 = 0;
   double result;

   // only way to exit this loop is an exception
   while ( true )
   {
      number1 = inputNumber();
      number2 = inputNumber();

      try
      {
         result = static_cast< double >( number1 ) / number2;
         cout << number1 << " / " << number2 << " = " << result
              << endl << endl;
      } // end try
      catch ( ... )
      {
         cout << "An unknown exception has occurred, "
              << "exiting the program\n"
              << e.what() << endl;
         exit( 0 );
      }; // end catch
      catch ( InvalidInputTypeException &e )
      {
         cout << "Exception occurred: " << e.what() << '\n';
         cin.clear();
         cin.ignore();
      } // end catch
      catch ( OutOfRangeException &&e )
         cout << "Exception occurred: " << e.what() << '\n';
   } // end while

   return 0;
} // end main
```

Fig. L 16.5 | `debugging.cpp`. (Part 3 of 3.)

Postlab Activities

Coding Exercise

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have completed the *Prelab Activities* and *Lab Exercises* successfully.

For the following problem, write a program or a program segment that performs the specified action:

1. A `try` block contains a function call to `mystery` that could produce any of the following exceptions: `DivideByZeroException`, `InvalidInputException`, `ArithmeticException` or `InvalidCastException`. Class `ArithmeticException` is a base class of `DivideByZeroException`. Write the `try` block and necessary `catch` blocks. Add a `catch` block to handle all other possible exceptions.