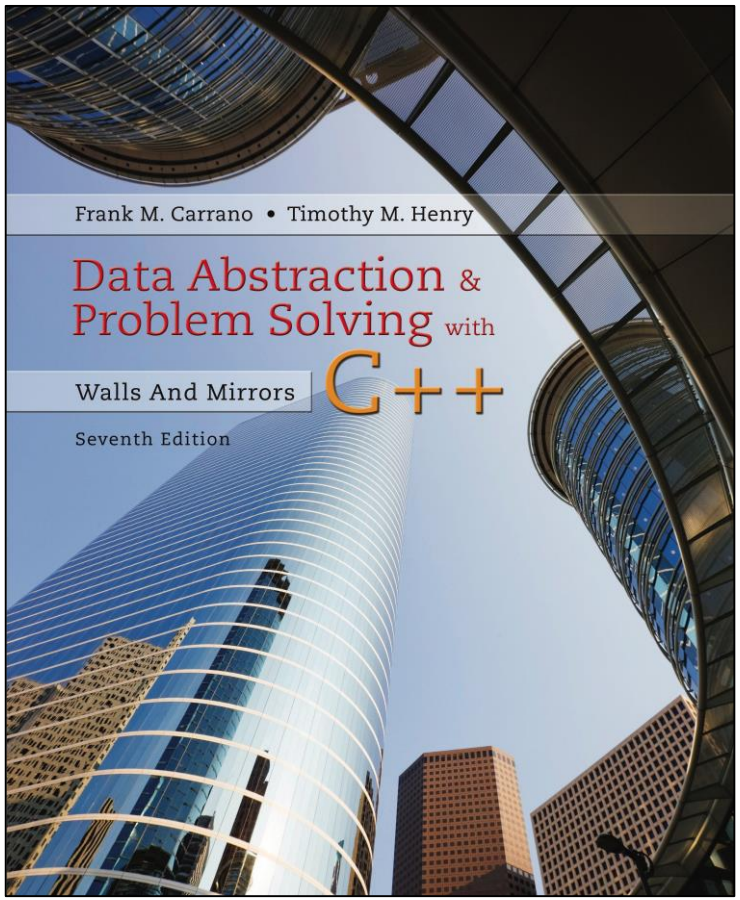
Data Abstraction & Problem Solving with C++: Walls and Mirrors

Seventh Edition



C++ Interlude 3

Exceptions

Background

- Preconditions for a method not always met
  - Method might return a false to indicate this
  - But not always possible
- Example
  - Stack method **peek()** called on empty stack which contains items of type **bool**
  - Return cannot be sure if return is normal or an exception

Problem to Solve (1 of 3)

- Previous C++ Interlude worked on video game
- Next task
  - Create function that searches for given string in a number of boxes
- Function parameters
  - Array of **string** objects
  - Integer represents number of objects in array
  - String to be located

Problem to Solve (2 of 3)

Listing C3-1 First try at the function **findBox**

```
1  PlainBox<std::string> findBox(PlainBox<std::string> boxes[], int size,
2                                std::string target)
3  {
4     int index = 0;
5     bool found = false;
6     while (!found && (index < size))
7     {
8        found = (target == boxes[index].getItem());
9        if (!found)
10          index++; // Look at next entry
11    }  // end while
12    return boxes[index];
13 }  // end findBox
```

Problem to Solve (3 of 3)

- Must deal with problem of a box containing target string not in the array
  - If target not found, function returns **boxes[size]** which is undefined
  - Problems occur when client tries to use this “box”
- What to return when target not found?

Assertions (1 of 2)

- Express an assertion either as a comment or by using the C++ function **assert**
  - Make assertions about variables, objects
  - Assertion in form of boolean expression that should be true at that point in program
  - False halts program execution
- Mainly used to validate pre- or postconditions
- This is a debugging tool
  - Not a substitute for an **if** statement

Assertions (2 of 2)

Listing C3-2 Revised **findBox** function with assertions

```
1  PlainBox<std::string> findBox(PlainBox<std::string> boxes[], int size,
2                                std::string target)
3  {
4     int index = 0;
5     bool found = false;
6     while (!found && (index < size))
7     {
8        found = (target == boxes[index].getItem());
9        if (!found)
10          index++;   // Look at next entry
11    }  // end while
12    assert(found);   // Verify that there is a box to return
13    return boxes[index];
14 }  // end findBox
```

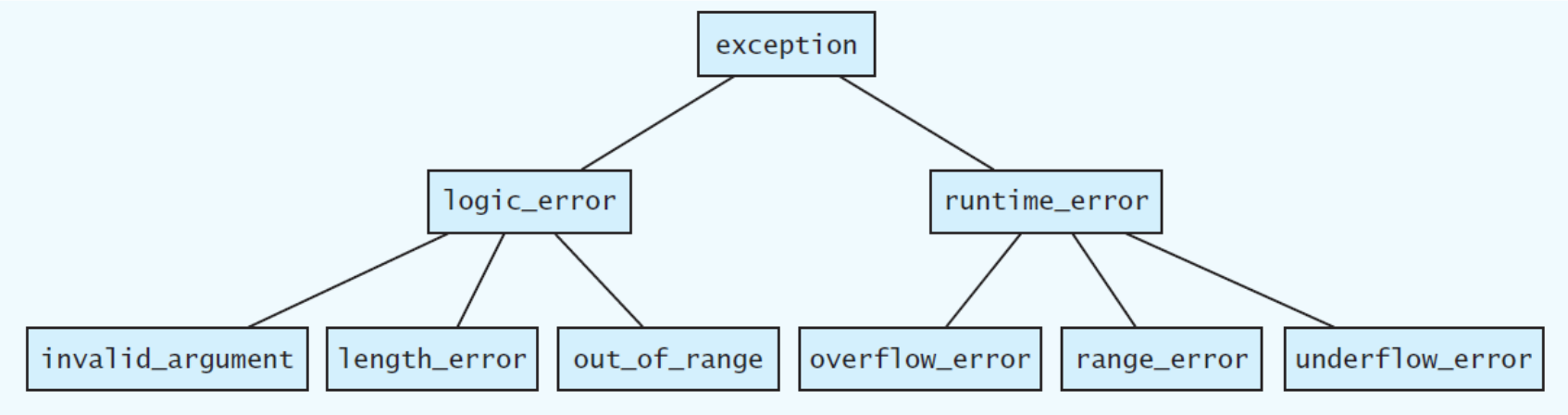
Throwing Exceptions (1 of 3)

- Alternate way of communicating or returning information to function's client
- Thrown exception bypasses normal execution,
  - Control immediately returns to client.
- Syntax

```
throw ExceptionClass(stringArgument);
```

Throwing Exceptions (2 of 3)

Figure C3-1 Hierarchy of C++ exception classes



Throwing Exceptions (3 of 3)

Listing C3-3 Revised **findBox** function that throws an exception

```
1  PlainBox<std::string> findBox(PlainBox<std::string> boxes[], int size,
2                                std::string target) throw(std::logic_error)
3  {
4     int index = 0;
5     bool found = false;
6     while (!found && (index < size))
7     {
8        found = (target == boxes[index].getItem());
9        if (!found)
10          index++; // Look at next entry
11    }  // end while
12
13    if (!found)
14       throw std::logic_error("Target not found in a box!");
15    return boxes[index];
16 }  // end findBox
```

Handling Exceptions (1 of 7)

- Code for handling exception
  - **try** block–contains statements that might cause or throw an exception
  - **catch** block–immediately follows **try** block with code to react to or catch a particular type of exception

Handling Exceptions (2 of 7)

General syntax for a **try** block followed by one **catch** block

```
try
{
    < statement(s) that might throw an exception >
}
catch (ExceptionClass  identifier)
{
    < statement(s) that react to an exception of type ExceptionClass >
}
```

Handling Exceptions (3 of 7)

- **try** block
  - Contains statements that might cause or throw an exception
- **catch** block
  - One or more **catch** blocks immediately follow **try** block
  - Contain code to react to or catch particular type of exception

Handling Exceptions (4 of 7)

- If no exception occurs and **try** block completes
  - Execution continues with statement after **catch** block
- If statement within **try** block causes exception of type specified in **catch** block
  - Remainder of **try** block abandoned
  - Execution transfers to statements in **catch** block
  - After **catch** block statements finish, execution jumps to statement after last catch **block**

Handling Exceptions (5 of 7)

- The syntax for **catch** block resembles that of a function definition
  - Specifies type of exception, and an identifier
  - The catch block parameter provides name for caught exception
- Steps taken in **catch** block vary
  - Simple message
  - Elaborate update of variables, retry of offending function

Handling Exceptions (6 of 7)

Listing C3-4 Trying the function **findBox**

```
1  // Create and initialize an array of boxes
2  PlainBox<std::string> myBoxes[5];              // Array of PlainBox objects
3  myBoxes[0] = PlainBox<std::string>("ring");
4  myBoxes[1] = PlainBox<std::string>("hat");
5  myBoxes[2] = PlainBox<std::string>("shirt");
6  myBoxes[3] = PlainBox<std::string>("sock");
7  myBoxes[4] = PlainBox<std::string>("shoe");
8  PlainBox<std::string> foundBox;
9
10 // Try to find a box containing glasses
11 try
12 {
13    foundBox = findBox(myBoxes, 5, "glasses");
14 }
15 catch(std::logic_error logErr)
```

Handling Exceptions (7 of 7)

Listing C3-4 [Continued]

```
13    foundBox = PlainBox<std::string>(myBoxes, 5, "glasses");
14 }
15 catch(std::logic_error logErr)
16 {
17    std::cout << logErr.what() << std::endl;  // Display error message
18    foundBox = PlainBox<std::string>("nothing"); // Fix problem
19 }  // end try-catch
20 // Because we catch the exception and fix the problem, the following
21 // statement should work even if the target is not found
22 std::cout << foundBox.getItem() << std::endl;
```

Output

```
Target not found in a box!
nothing
```

Multiple Catch Blocks

- **try** block may cause more than one type of exception
  - Can have many **catch** blocks associated with it
- **catch** blocks must be ordered
  - Most specific classes first
  - More general classes last

Uncaught Exceptions (1 of 4)

Listing C3-5 A program with an uncaught exception

```
#include <iostream>
#include <string>

// Encodes the character at index i of the string str.
void encodeChar(int i, string& str)
{
    int base = static_cast<int>('a');
    if (isupper(str[i]))
       base = int('A');

    char newChar = (static_cast<int>(str[i]) - base + 3) % 26 + base;
    str.replace(i, 1, 1, newChar); // Method replace can throw exception
}   // end encodeChar

// Encodes numChar characters within a string.
void encodeString(int numChar, string& str)
```

Uncaught Exceptions (2 of 4)

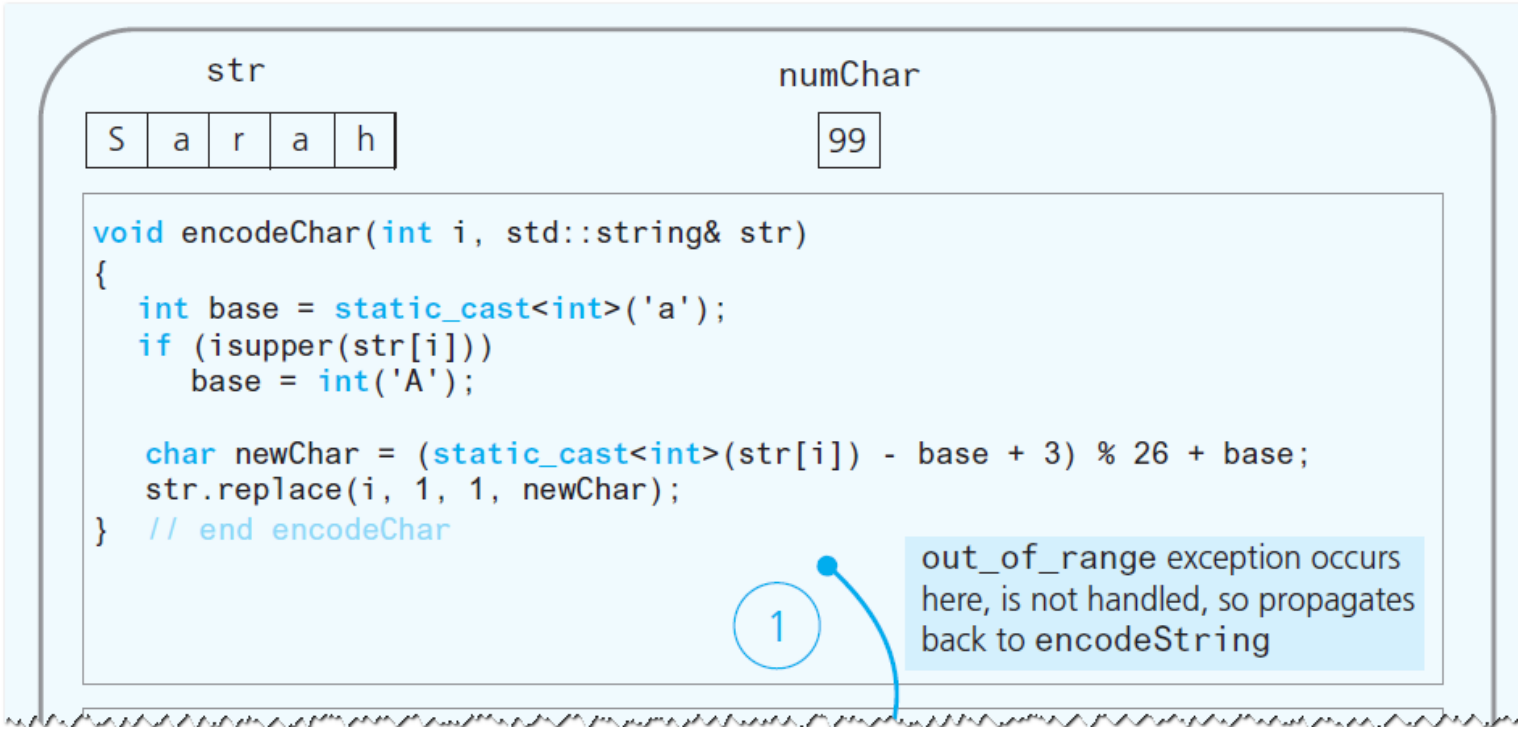
Listing C3-5 [Continued]

```
// Encodes numChar characters within a string.
void encodeString(int numChar, string& str)
{
   for (int j = numChar - 1; j >= 0; j-)
      encodeChar(j, str);
}  // end encodeString

int main()
{
    string str1 = "Sarah";
    encodeString(99, str1);
    return 0;
}   // end main
```

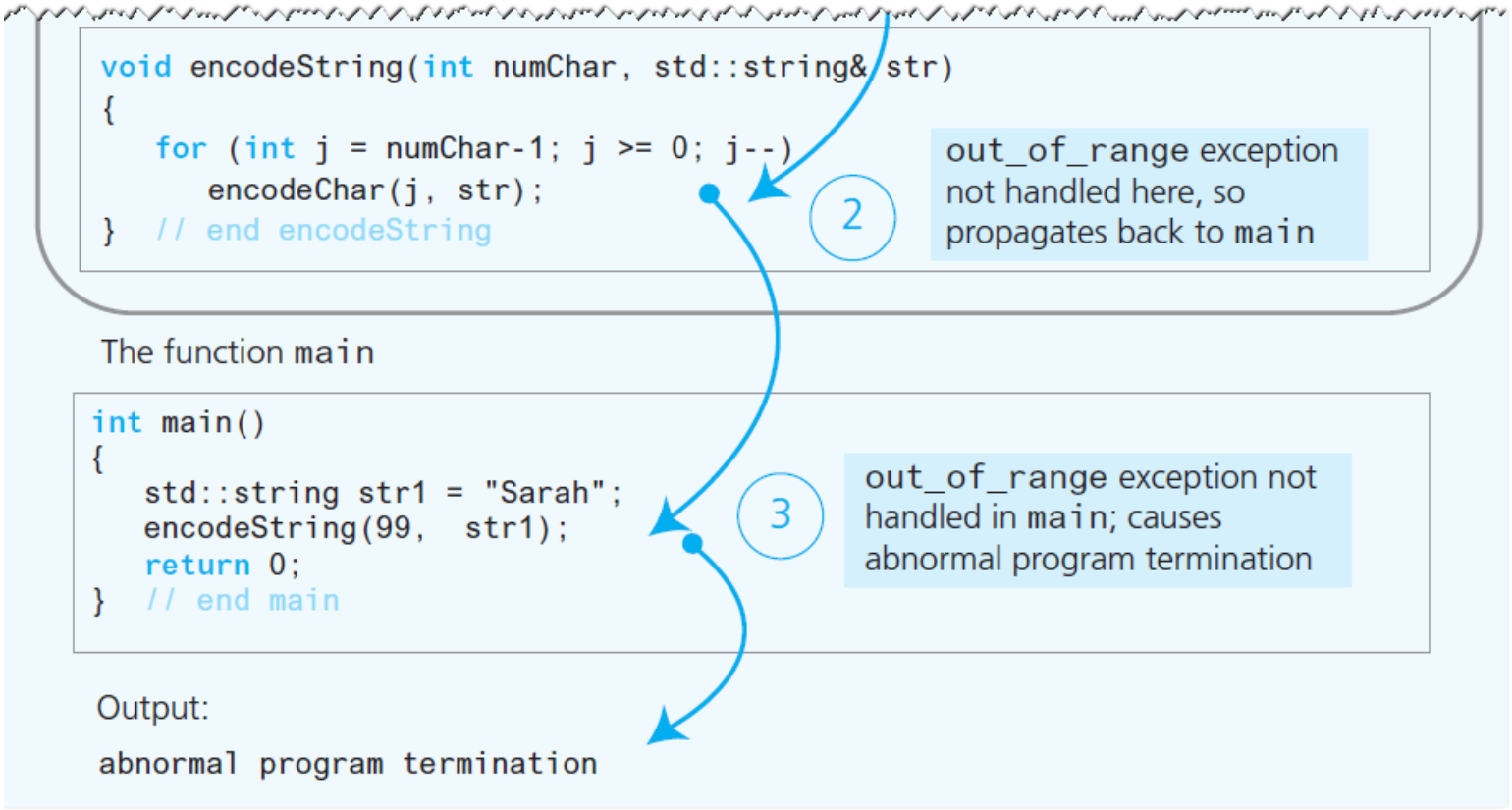
Uncaught Exceptions (3 of 4)

Figure C3-2 Flow of control for an uncaught exception



Uncaught Exceptions (4 of 4)

Figure C3-2 [Continued]



Programmer-Defined Exception Classes (1 of 2)

- Usually, C++ exception class **exception** , or one of its derived classes, is the base class
  - Provides a standardized interface for working with exceptions.
- Exception class typically consists of a constructor that has a string parameter

Programmer-Defined Exception Classes (2 of 2)

```
#include <stdexcept>
#include <string>
class TargetNotFoundException: public std::exception
{
public :
   TargetNotFoundException(const std::string& message = "")
                          : std::exception("Target not found: " + message)
   {
   } // end constructor
}; // end TargetNotFoundException
```

```
throw TargetNotFoundException (target + " not found in a box!");
```

Example–constructor provides way for throw statement to identify condition of exception.

Copyright