Exceptions

C++ Interlude 3

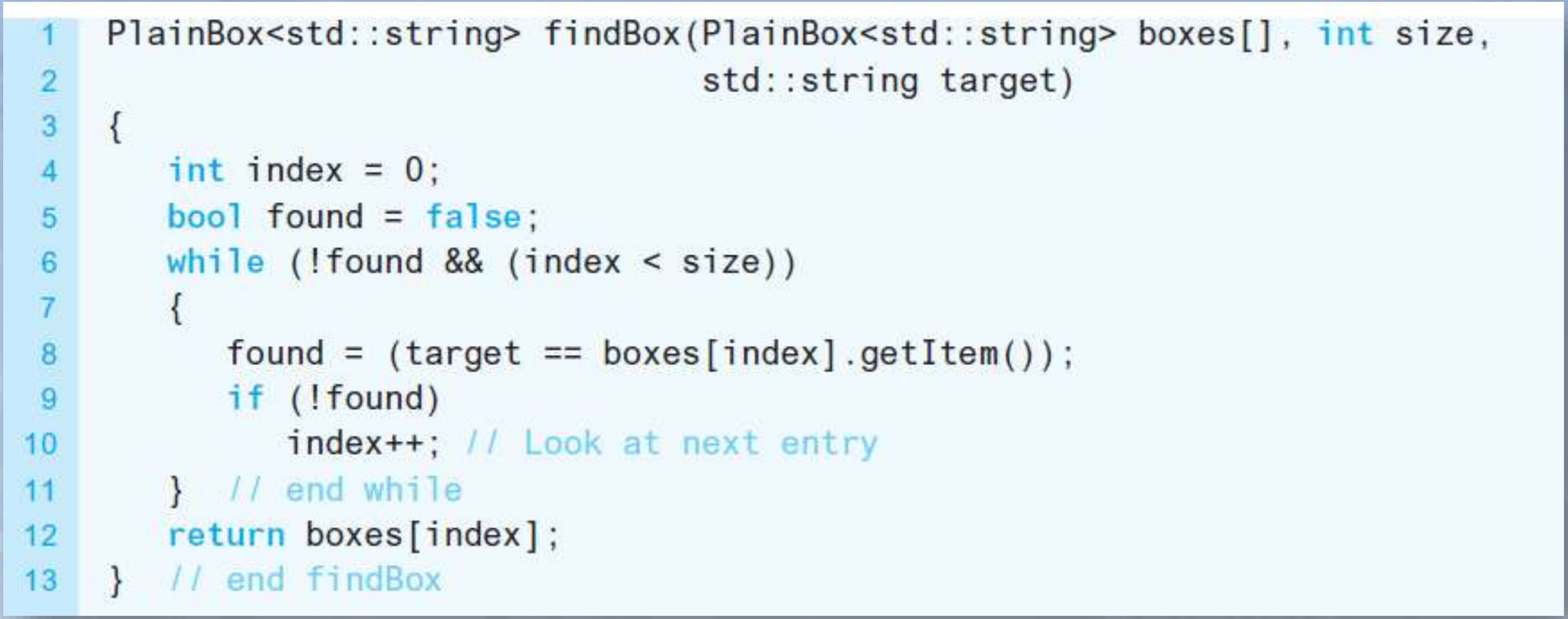
Background

- 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

- 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



```
PlainBox<std::string> findBox(PlainBox<std::string> boxes[], int size,
                              std::string target)
{
   int index = 0;
   bool found = false;
   while (!found && (index < size))
   {
      found = (target == boxes[index].getItem());
      if (!found)
         index++; // Look at next entry
   }  // end while
   return boxes[index];
}  // end findBox
```

LISTING C3-1 First try at the function findBox

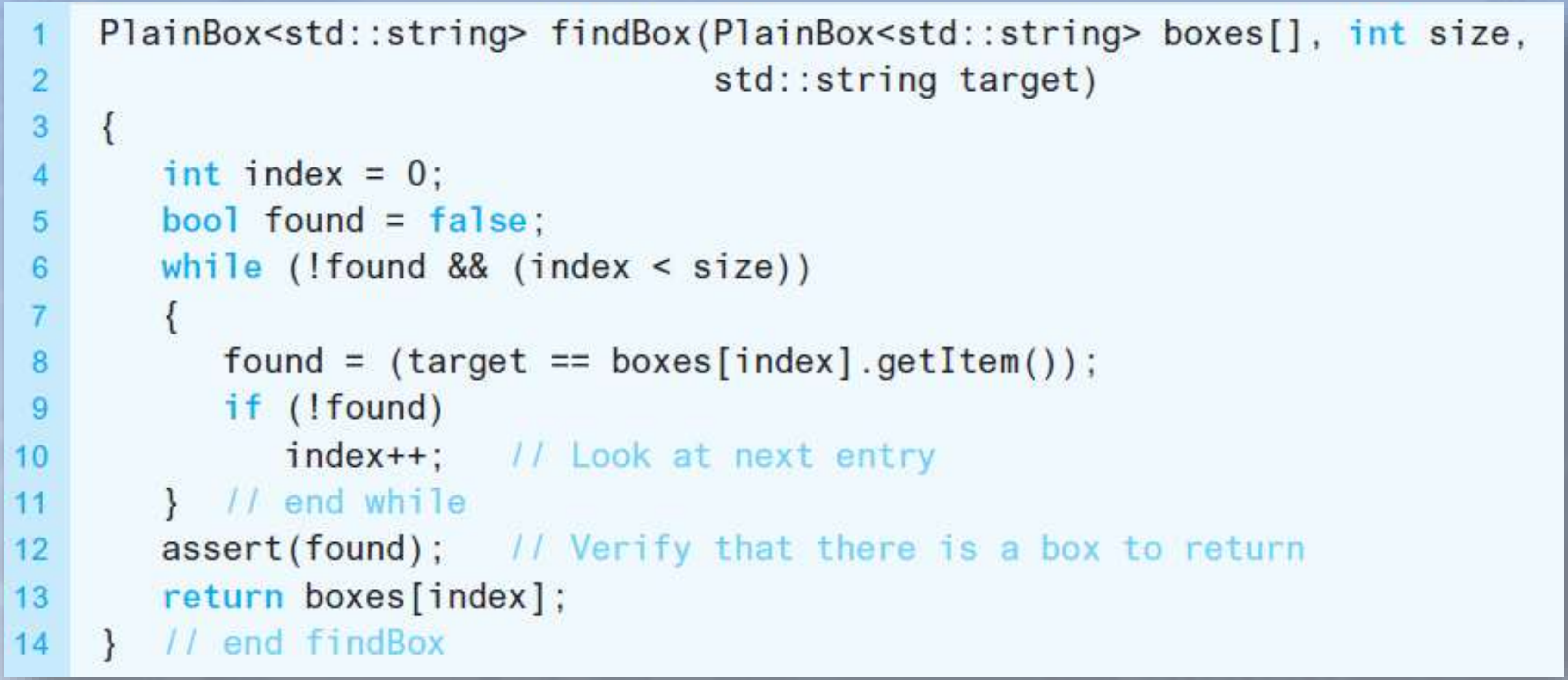
Problem to Solve

- 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

- 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



```cpp
PlainBox<std::string> findBox(PlainBox<std::string> boxes[], int size,
                              std::string target)
{
   int index = 0;
   bool found = false;
   while (!found && (index < size))
   {
      found = (target == boxes[index].getItem());
      if (!found)
         index++;    // Look at next entry
   }  // end while
   assert(found);    // Verify that there is a box to return
   return boxes[index];
}  // end findBox
```

LISTING C3-2 Revised findBox function with assertions

Throwing Exceptions

- 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

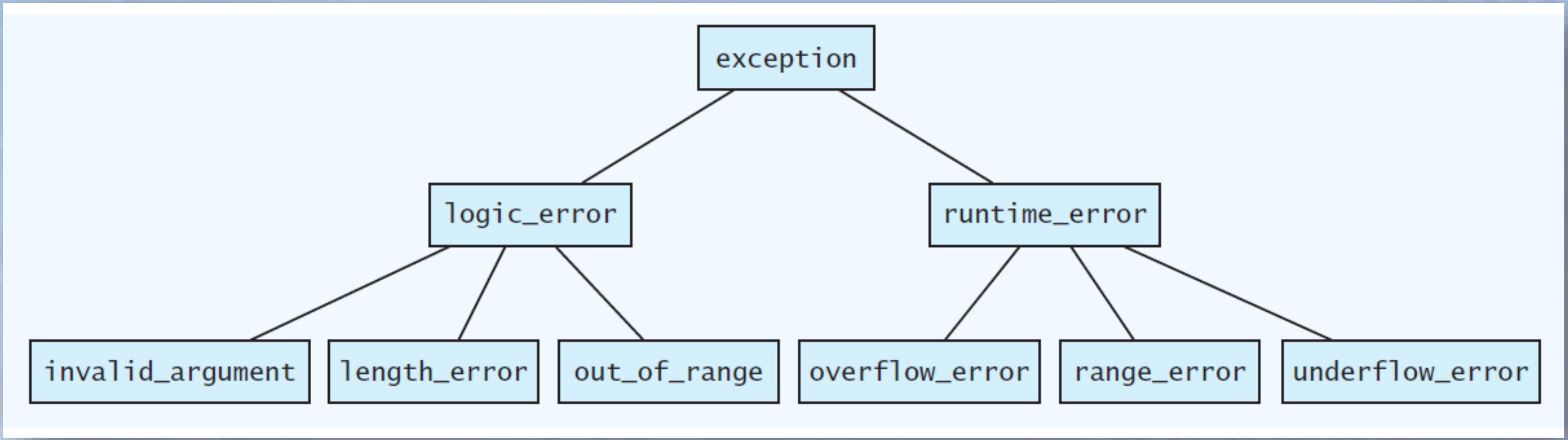


FIGURE C3-1 Hierarchy of C++ exception classes

Throwing Exceptions

```
PlainBox<std::string> findBox(PlainBox<std::string> boxes[], int size,
                              std::string target) throw(std::logic_error)
{
   int index = 0;
   bool found = false;
   while (!found && (index < size))
   {
      found = (target == boxes[index].getItem());
      if (!found)
         index++; // Look at next entry
   }  // end while

   if (!found)
      throw std::logic_error("Target not found in a box!");
   return boxes[index];
}  // end findBox
```

LISTING C3-3 Revised findBox function that throws an exception

Handling Exceptions

- 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

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

General syntax for a try block followed by one catch block

Handling Exceptions

- 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

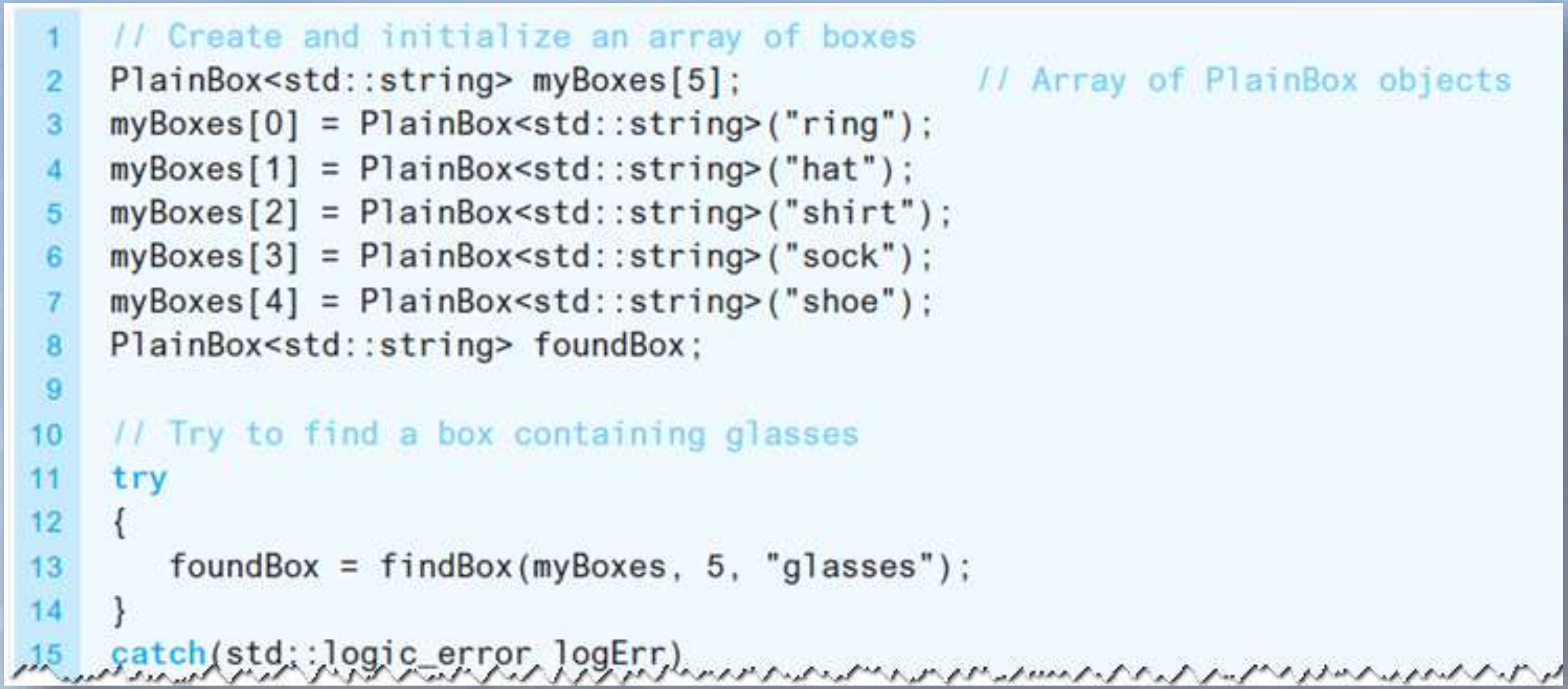
Handling Exceptions

- 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

- 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

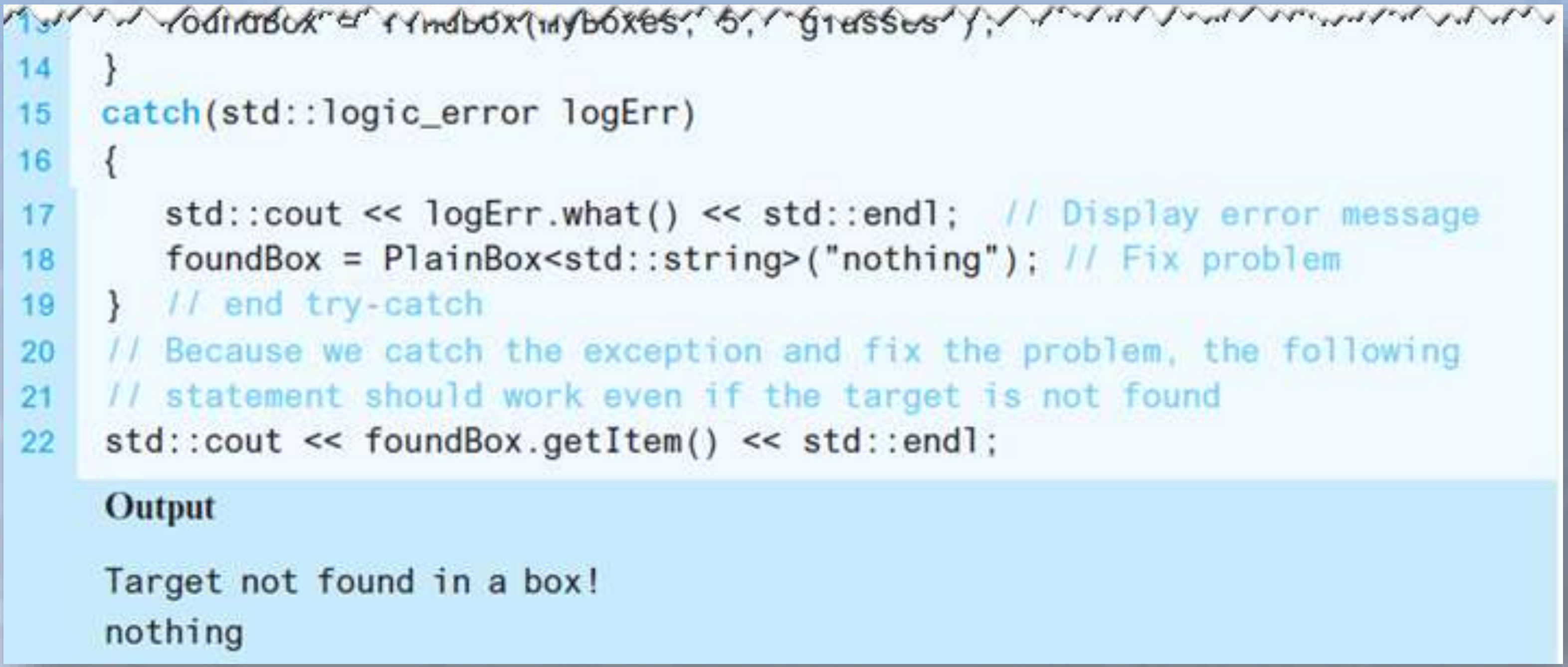
Handling Exceptions



```
// Create and initialize an array of boxes
PlainBox<std::string> myBoxes[5];          // Array of PlainBox objects
myBoxes[0] = PlainBox<std::string>("ring");
myBoxes[1] = PlainBox<std::string>("hat");
myBoxes[2] = PlainBox<std::string>("shirt");
myBoxes[3] = PlainBox<std::string>("sock");
myBoxes[4] = PlainBox<std::string>("shoe");
PlainBox<std::string> foundBox;

// Try to find a box containing glasses
try
{
   foundBox = findBox(myBoxes, 5, "glasses");
}
catch(std::logic_error logErr)
```

LISTING C3-4 Trying the function findBox

Handling Exceptions



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

Output

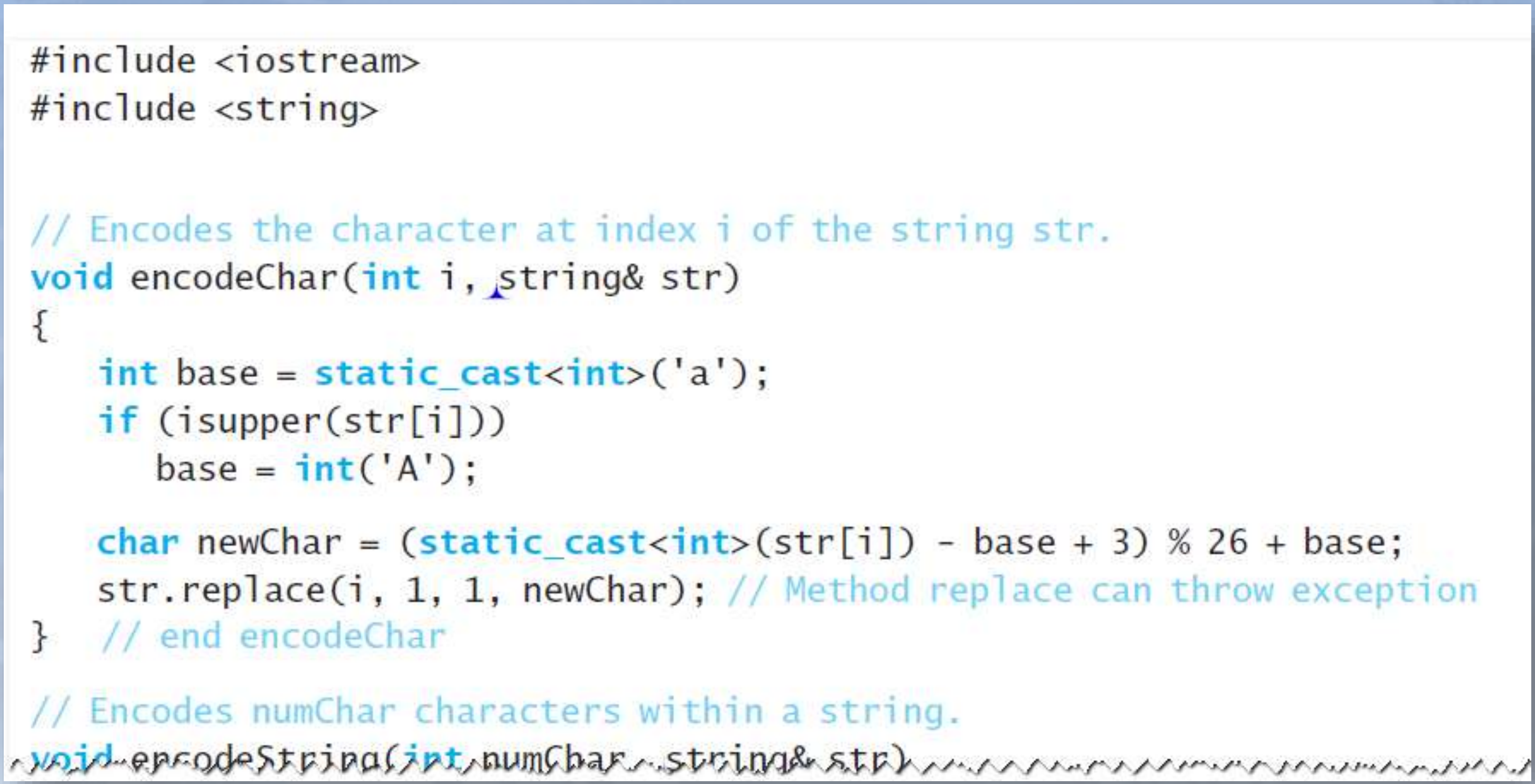
```
Target not found in a box!
nothing
```

LISTING C3-4 Trying the function findBox

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

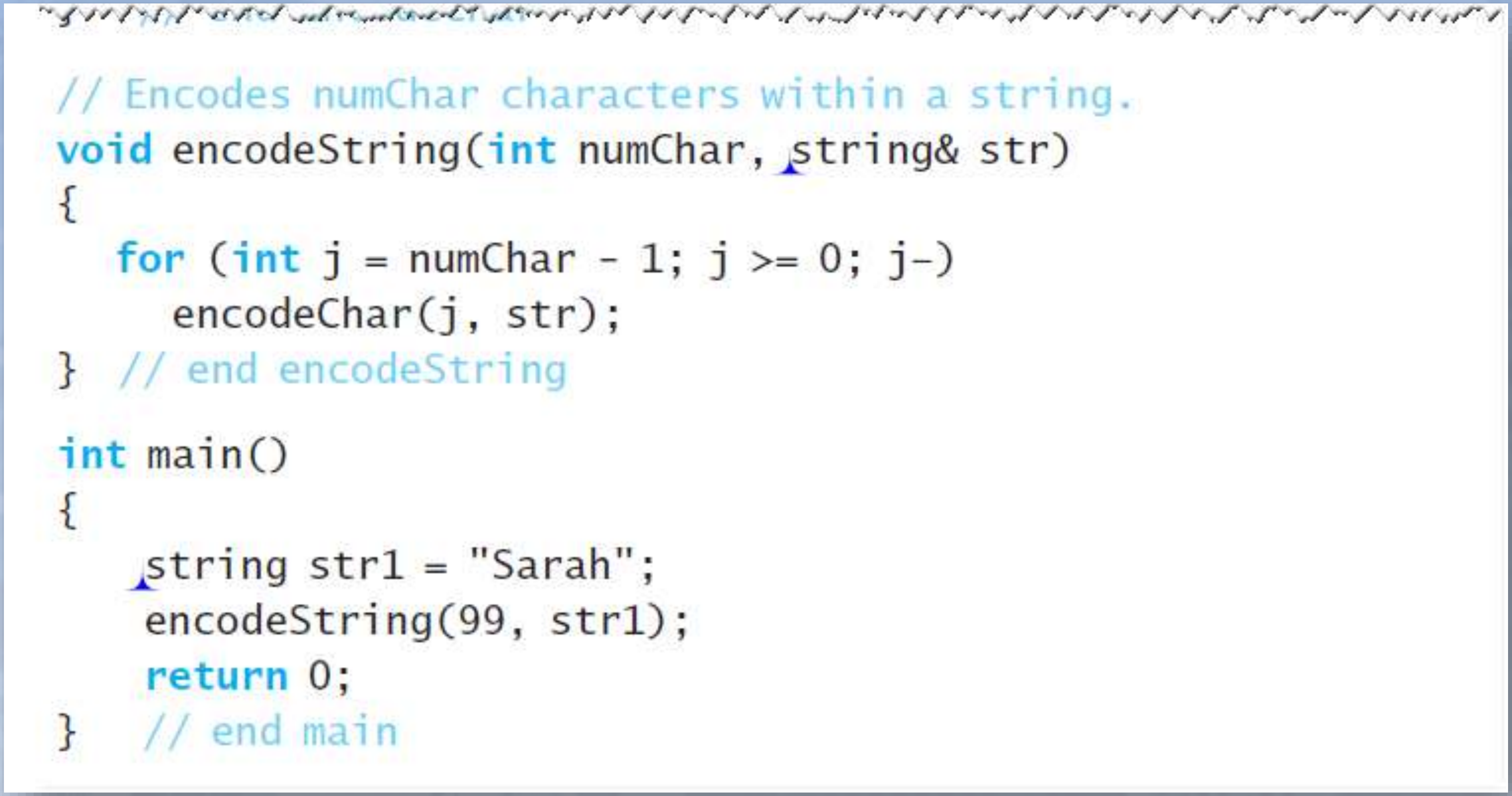
Uncaught Exceptions



```cpp
#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)
```

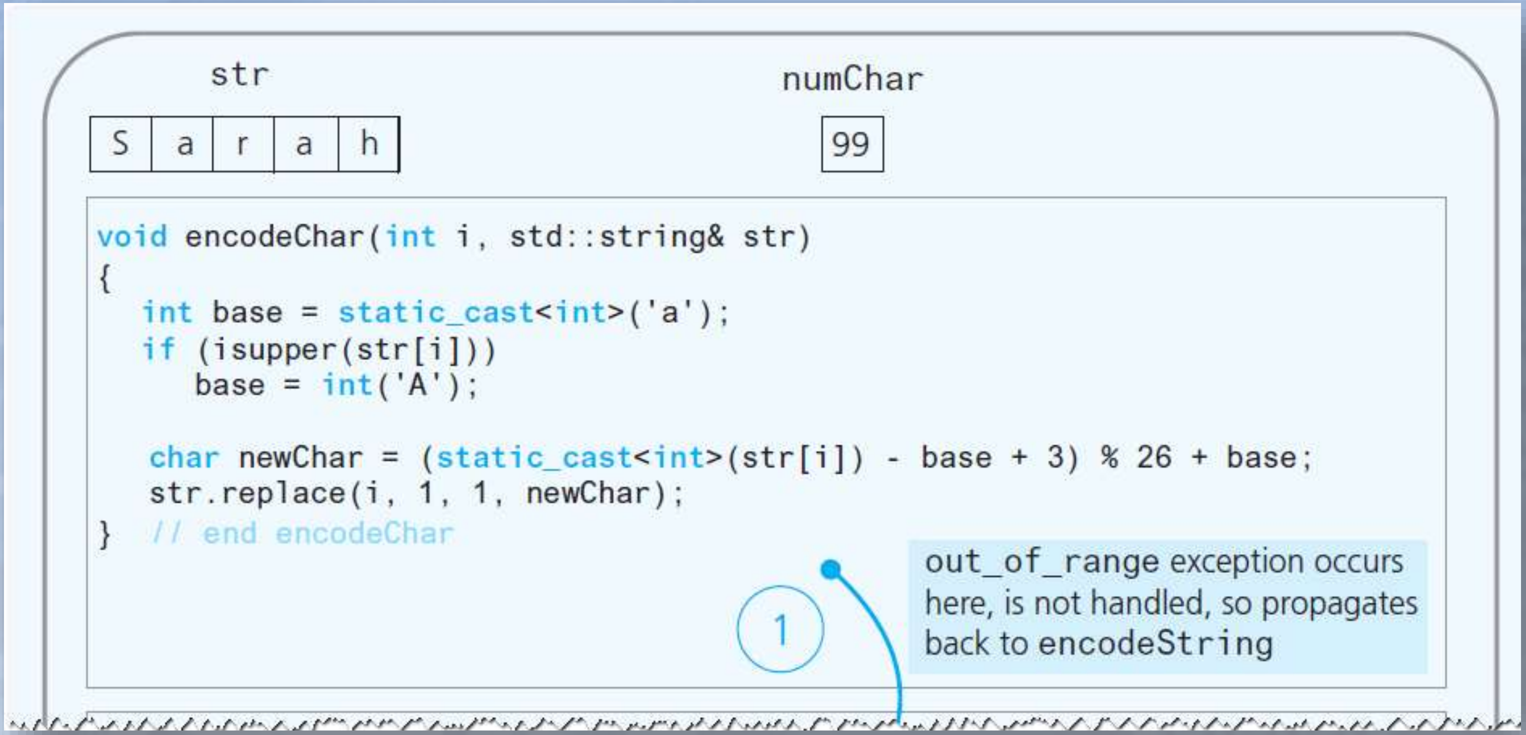
LISTING C3-5 A program with an uncaught exception

Uncaught Exceptions


```
// 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
```

LISTING C3-5 A program with an uncaught exception

Uncaught Exceptions

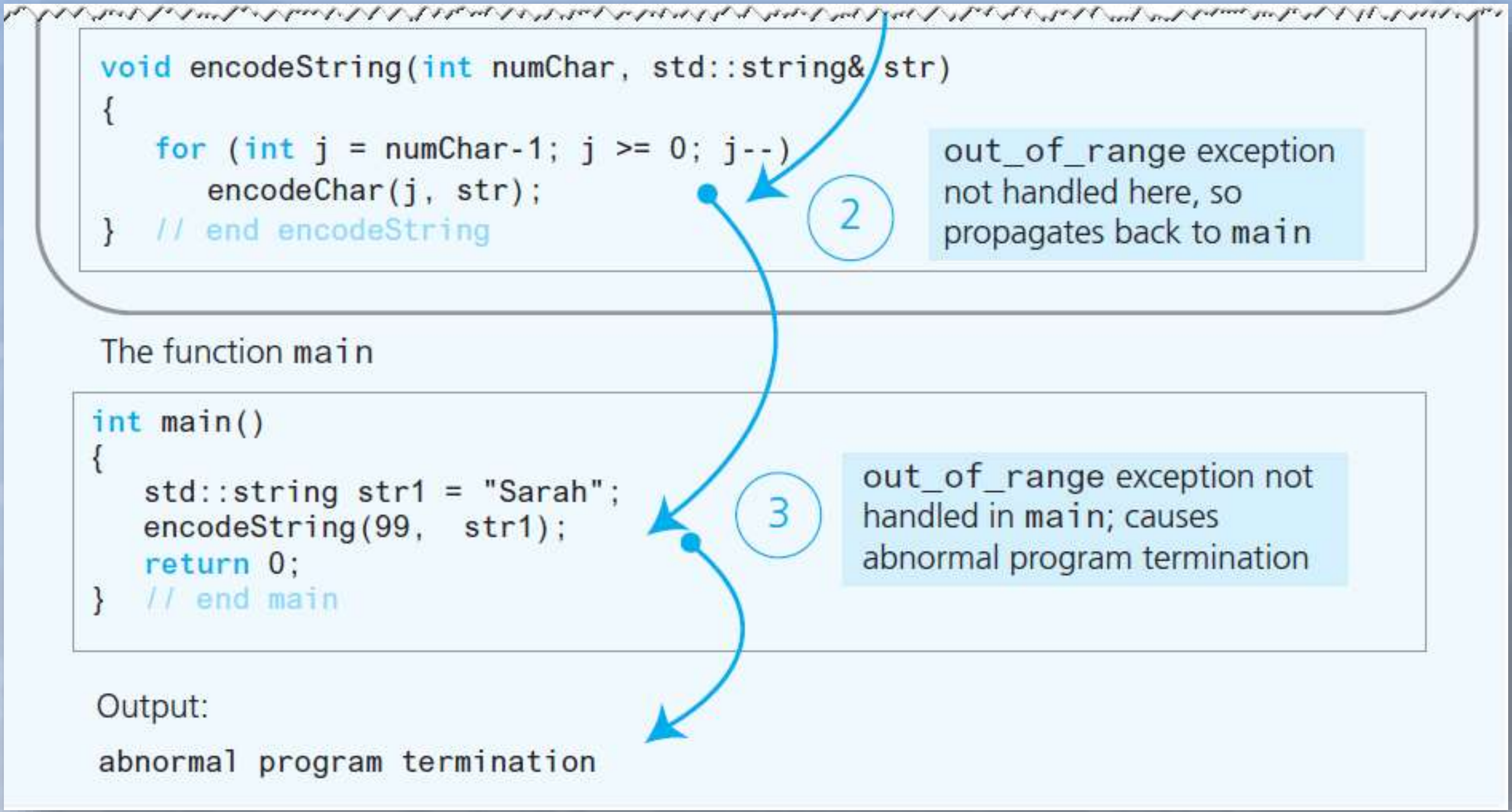


```
void encodeChar(int i, std::string& str)
{
   int base = static_cast<int>('a');
   if (isupper(str[i]))
      base = int('A');

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

out_of_range exception occurs here, is not handled, so propagates back to encodeString

FIGURE C3-2 Flow of control for an uncaught exception

Uncaught Exceptions



```
void encodeString(int numChar, std::string& str)
{
   for (int j = numChar-1; j >= 0; j--)
      encodeChar(j, str);
}  // end encodeString
```

2
out_of_range exception not handled here, so propagates back to main
The function main

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

3
out_of_range exception not handled in main; causes abnormal program termination
Output:

```
abnormal program termination
```

FIGURE C3-2 Flow of control for an uncaught exception

Programmer-Defined Exception Classes

- 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

```
#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.

End

C++ Interlude 3