Object Oriented Programming by C++

Exception Handling

Dealing with Runtime Errors

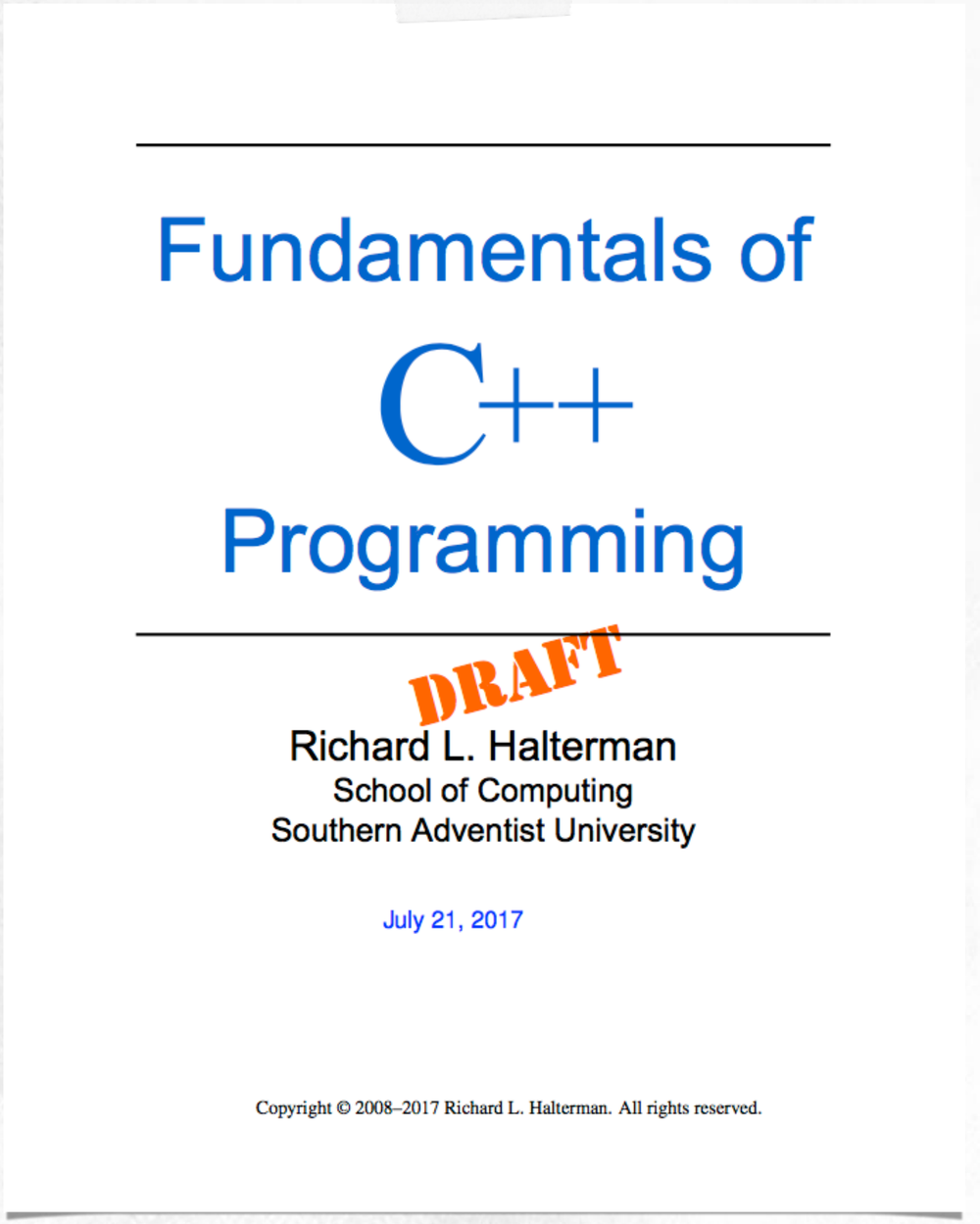
2017. 8.

Sungwon Lee / Professor

Email: drsungwon@khu.ac.kr
Web: http://mobilelab.khu.ac.kr/

Textbook & Copyright

- Textbook: http://python.cs.southern.edu/cppbook/progcpp.pdf
- Sample Codes: https://github.com/halterman/CppBook-SourceCode



Fundamentals of

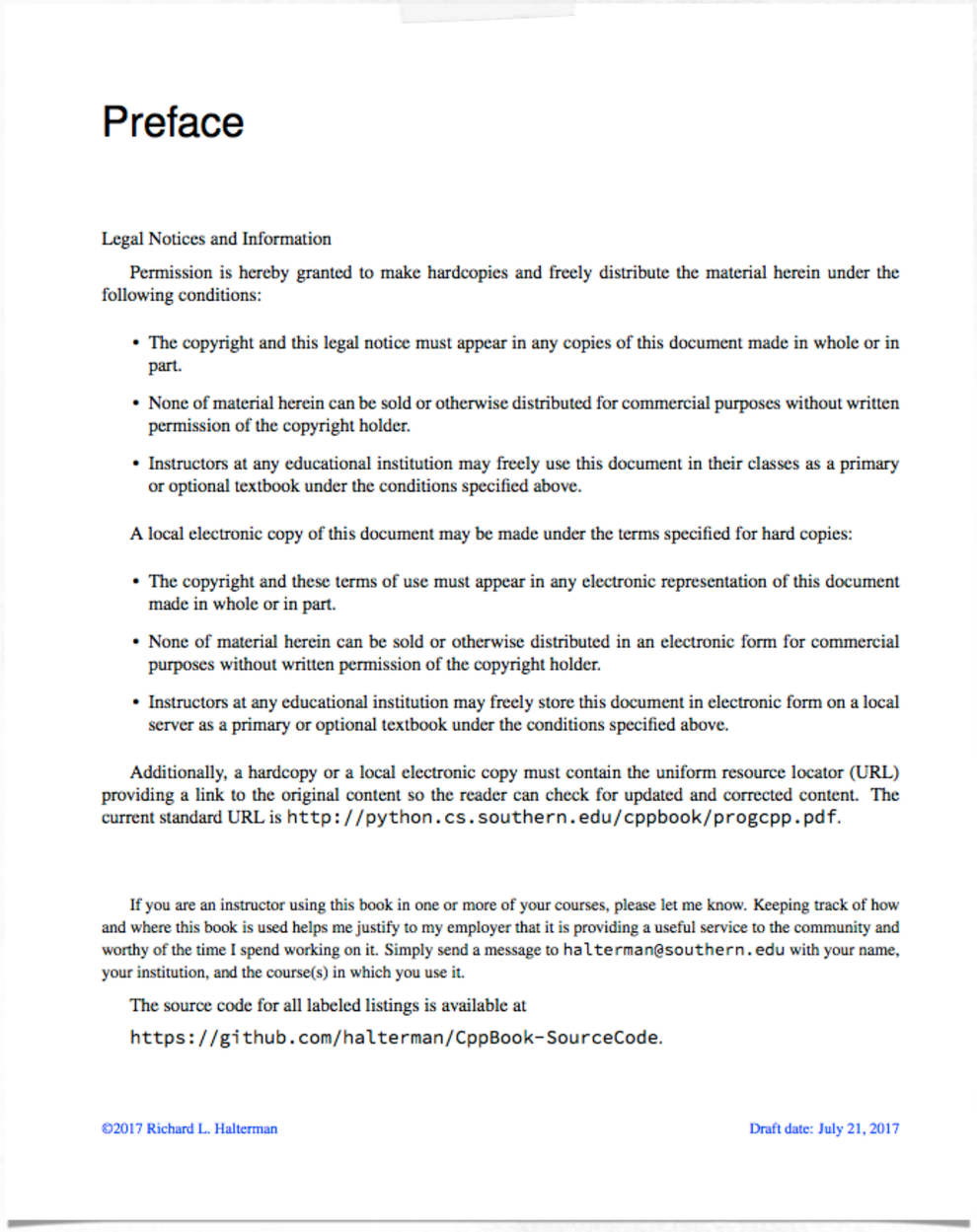
C++

Programming

DRAFT

Richard L. Halterman
School of Computing
Southern Adventist University

July 21, 2017



Preface

If you are an instructor using this book in one or more of your courses, please let me know. Keeping track of how and where this book is used helps me justify to my employer that it is providing a useful service to the community and worthy of the time I spend working on it. Simply send a message to `halterman@southern.edu` with your name, your institution, and the course(s) in which you use it.

The source code for all labeled listings is available at

`https://github.com/halterman/CppBook-SourceCode`.

 Draft date: July 21, 2017

Contents

Motivation

C++ Exception Handling

- C++'s `exception` handling infrastructure
  - allows programmers to cleanly ***separate the code*** that implements the ***focused algorithm*** from the code that deals with ***exceptional situations*** that the algorithm may face
  - is more modular and encourages the development of code that is ***cleaner and easier to maintain and debug***

- An `exception`
  - is an exceptional event that ***occurs during a program's execution***.
  - always is possible, but it ***should be a relatively rare*** event
  - almost always represents a problem, usually some sort of run-time error
    - example: **`v[i]`** access when '**`i`**' is out-of-range(= bounds)

Let's make some Errors

Listing 22.1: vectorboundscrash.cpp

```
#include <iostream>
#include <vector>

int main() {
    std::vector<double> nums { 1.0, 2.0, 3.0 };
    int input;
    std::cout << "Enter an index: ";
    std::cin >> input;
    std::cout << nums.at(input) << '\n';
}
```

```
Enter an index: 10
libc++abi.dylib: terminating with
uncaught exception of type
std::out_of_range: vector
```

- Program abnormally terminated

Exception Handling Statement

`try` and `catch`

- To ***intercept the problem at run time*** and prevent the program from terminating due to an error
  - we use a ***`try/catch` block***
  - which consists of two parts: a **`try`** block and a **`catch`** block

- To form a try/catch block we
  1. wrap the code that has the ***potential to throw an exception*** in a **`try`** block
  2. provide code to ***execute only in the event of an exception*** in a **`catch`** block
  3. the statements within a **`try`** block and statements within a **`catch`** block ***must appear within curly braces, even if only one statement*** appears in the section.

Enhanced Error Codes

Listing 22.2: vectorboundsexcept.cpp

```
#include <iostream>
#include <vector>

int main() {
    std::vector<double> nums { 1.0, 2.0, 3.0 };
    int input;
    std::cout << "Enter an index: ";
    std::cin >> input;
    try {
        std::cout << nums.at(input) << '\n';
    }
    catch (std::exception& e) {
        std::cout << e.what() << '\n';
    }
}
```

```
Enter an index: 10
vector
Program ended with exit code: 0
```

- Program normally terminated

Exception Handling Statement

`std::exception`

- ***Standard exception class***
- Base class for standard exceptions.
  - All objects thrown by components of the standard library are ***derived from this class***
  - Therefore, ***all standard exceptions can be caught by catching this type by reference***

Member functions

(constructor)	Construct exception (public member function)
operator=	Copy exception (public member function)
what (virtual)	Get string identifying exception (public member function)
(destructor) (virtual)	Destroy exception (public virtual member function)

Reference: http://www.cplusplus.com/reference/exception/exception/

Exception Handling Statement

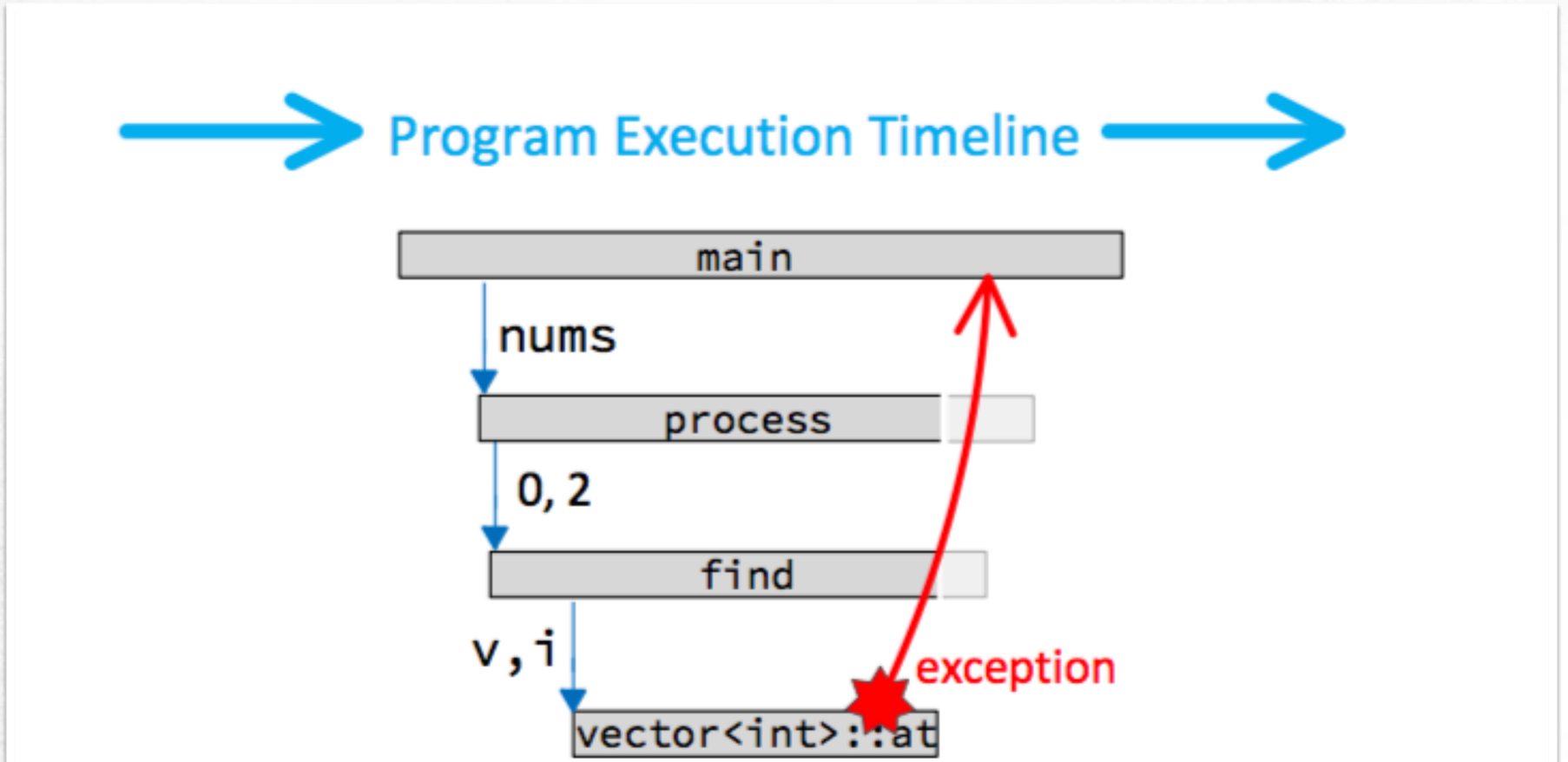
Code Example

```
#include <iostream>
#include <vector>

int main() {
    std::vector<double> nums { 1.0, 2.0, 3.0 };
    int input;
    while (true) {
        std::cout << "Enter an index: ";
        std::cin >> input;
        try {
            std::cout << nums.at(input) << '\n';
            break;  // Printed successfully, so break out of loop
        }
        catch (std::exception&) {
            std::cout << "Index is out of range.  Please try again.\n";
        }
    }
}
```

Finding Nearest `catch` Statement

- Listing 22.4 Case
  - Error in the function may be caught by the catch statement in the function
  - OR, the error traversed to the upper functions (which invoked the error causing function) ***until the first catch statement is encountered***



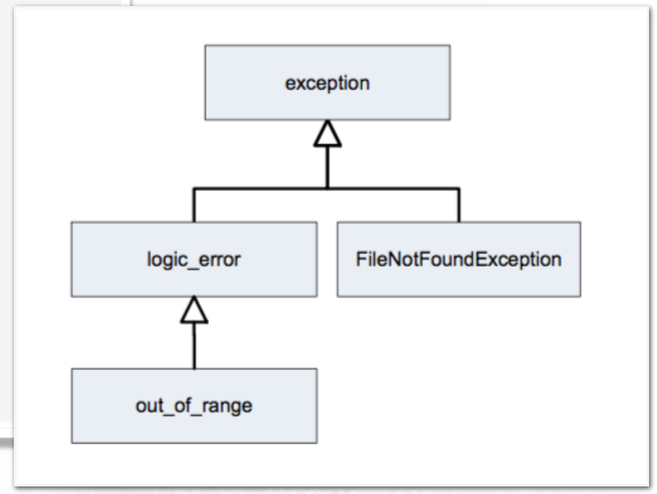
Programmer Defined Exception

Exception Class Inheritance

- ***The standard C++ library has a limited number of standard exceptions***
- ***Programmer can create our own custom exceptions*** for specialized error handling that our applications may require
- Programmer ***creates own constructor(), and override*** required functions

```
// Exception object to thrown when a client attempts to
// open a text file via a name that does correspond to a
// file in the current working directory.
class FileNotFoundException : public std::exception {
    std::string message;  // Identifies the exception and filename
  public:
    // Constructor establishes the exception object's message
    FileNotFoundException(const std::string& fname):
        message("File \"" + fname + "\" not found") {}

    // Reveal message to clients
    const char *what() const {
        return message.c_str();
    }
};
```

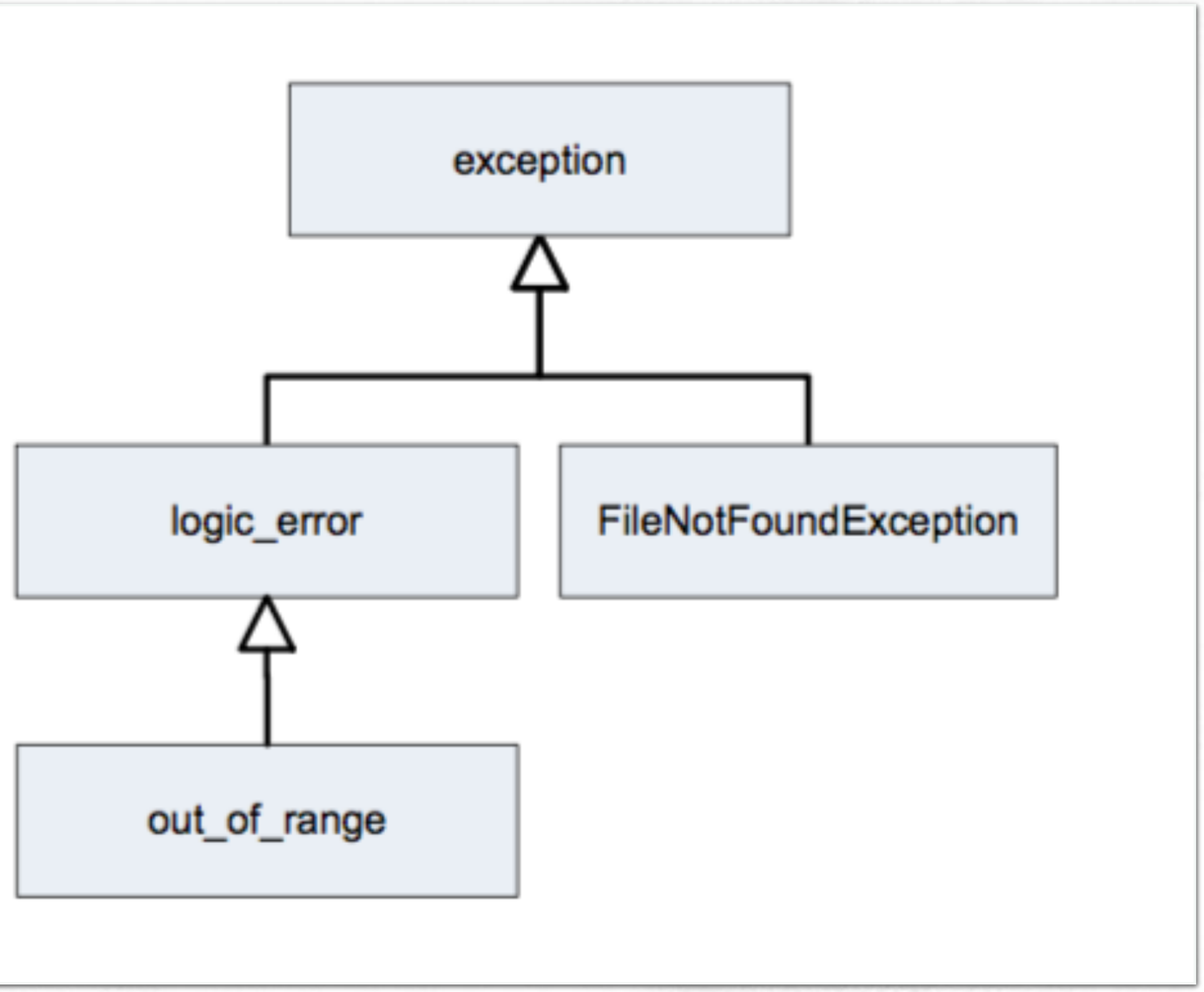


Programmer Defined Exception

Exception Class Inheritance

- The standard C++ library has a limited number of standard exceptions
- Programmer can create our own custom exceptions for specialized error handling that our applications may require
- Programmer creates own constructor(), and override required functions

```
// Exception object to thrown when a cl
// open a text file via a name that does
// file in the current working directory
class FileNotFoundException : public std
    std::string message;  // Identifies
  public:
    // Constructor establishes the excep
    FileNotFoundException(const std::str
        message("File \"" + fname + "\"
    // Reveal message to clients
    const char *what() const {
        return message.c_str();
    }
};
```



Creating Programmer Defined Exception

- `throw` Statement
  - Signals an erroneous condition and executes an error handler
  - In general, an exception is thrown by using the throw keyword from inside the try block, or explicit throwing operation
  - A throw expression accepts one parameter, which is passed as an argument to the exception handler
  - ***Multiple handlers (i.e., catch expressions) can be chained***; each one with a different parameter type. Only the handler whose argument type matches the type of the exception specified in the throw statement is executed
  - After an exception has been handled the program, ***execution resumes after the try-catch block, not after the throw statement!***

Code Review (Listing 22.8) (1/2)

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

// Exception object to thrown when a client attempts to
// open a text file via a name that does correspond to a
// file in the current working directory.
class FileNotFoundException : public std::exception {
    std::string message;  // Identifies the exception and filename
public:
    // Constructor establishes the exception object's message
    FileNotFoundException(const std::string& fname):
    message("File \"" + fname + "\" not found") {}

    // Reveal message to clients

//  ORIGINAL CODE (BELOW) Causes error and we modified:
//  const char *what() const {
    virtual const char* what() const throw () {
        return message.c_str();
    }
};
```

- Inherited from std:exception class

Programmer Defined Exception

Code Review (Listing 22.8) (2/2)

```
// Creates and returns a vector of integers from data stored
// in a text file.
// filename: the name of the text file containing the data
// Returns a vector containing the data in the file, if possible
std::vector<int> load_vector(const std::string& filename) {
    std::ifstream fin(filename);      // Open the text file for reading
    if (fin.good()) {                 // Did the file open successfully?
        std::vector<int> result;      // Initially empty vector
        int n;
        fin >> n;                     // Size of data set
        for (int i = 0; i < n; i++ ) {
            int value;
            fin >> value;             // Read in a data value
            result.push_back(value); // Append it to the vector
        }
        return result;                // Return the populated vector
    }
    else   // Could not open the text file
        throw FileNotFoundException(filename);

}

int main() {
    try {
        std::vector<int> numbers = load_vector("values.data");
        for (int value : numbers)
            std::cout << value << ' ';
        std::cout << '\n';
    }
    catch (std::exception& e) {
        std::cout << e.what() << '\n';
    }
}
```

```
File "values.data" not found
Program ended with exit code: 0
```

Catching Multiple Exceptions

- Multiple catch Statement after try Statement can solve problems
  - As we already saw in slide 13
  - ***only the handler whose argument type matches*** the type of the exception specified in the throw statement ***is executed***

Code Example (Listing 22.10) (main() only)

```
int main() {
    try {
        std::vector<int> numbers = load_vector("1.dat");
        for (int value : numbers)
            std::cout << value << ' ';
        std::cout << '\n';
    }
    catch (std::out_of_range& e) {
        std::cout << "Error: vector bounds exceeded\n";
        std::cout << e.what() << '\n';
    }
    catch (FileNotFoundException& e) {
        std::cout << "Error: cannot open file\n";
        std::cout << e.what() << '\n';
    }
}
```

- Protection from possible bound error
- Same with Listing 22.8

Exception Re-trhrowing

- An exception handler within a catch block may take a few steps to handle the exception and then ***re-throw the exception or throw a completely different exception***.

Layered Exception Handling

Code Example

```
#include <iostream>
#include <fstream>
#include <vector>

void filter(std::vector<int>& v, int i) {
    v.at(i)++;
}

void compute(std::vector<int>& a) {
    for (int i = 0; i < 6; i++) {
        try {
            filter(a, i);
        }
        catch (std::exception& ex) {
            std::cout << "*********************************\n";
            std::cout << "* For loop terminated prematurely\n";
            std::cout << "* when i = " << i << '\n';
            std::cout << "*********************************\n";
            throw ex;  // Rethrow the same exception
        }
    }
}

int main() {
    std::vector<int> list { 10, 20, 30, 40, 50 };
    try {
        compute(list);
    }
    catch (std::exception& e) {
        std::cout << "Caught an exception: " << e.what() << '\n';
    }
    std::cout << "Program finished\n";
}
```

- Exception caused when ‘i’ is ‘5’
- First catcher

```
*********************************
* For loop terminated prematurely
* when i = 5
*********************************
Caught an exception: std::exception
Program finished
Program ended with exit code: 0
```

- Second catcher

Object Oriented Programming by C++

Sungwon Lee / Professor

Email: drsungwon@khu.ac.kr

Web: http://mobilelab.khu.ac.kr/