Instructor's Guide
for
C++ Without Fear
Third Edition

Brian Overland

ISBN-10: 0134314301

PRENTICE
HALL

CONTENTS

Chapter 7: Pointers: Data by Location

Chapter 8: Strings: Analyzing the Text

Chapter 12: Two Complete OOP Examples

Chapter 13: Easy Programming with STL

Chapter 17: New Features of C++14

Chapter 18: Operator Functions: Doing It with Class

Introduction

This guide is written for people who already have mastery of C++ but have to teach it to others. Here, more so than in the book itself, I discuss alternative learning paths and pedagogical issues: for example, depending on the length and the overall scope of the course, you may want to spend a lot of time with some chapters and skip others.

For each chapter, I present chapter scope and chapter Objectives. These Objectives form a different kind of summary than that found at the end of each chapter. The Objectives here provide general course goals, not specific information. In most cases, I have listed only a few major objectives for each chapter, to stress what (in the author's opinion) are the most crucial things for the students to master before going onward. Making sure that students have success with these objectives should ensure that they have the necessary knowledge to tackle subsequent chapters; missing some of this information, subsequent chapters are likely to be baffling.

Remember that, as in mathematics, computer-language concepts build on each other, so it is crucial to master the basics of one chapter before going onto the next.

The most important part of this guide is probably the additional exercises provided here for each chapter, along with all the solutions for these additional exercises.

There is some danger, of course, in putting answers to exercises in a publicly available place; the students may always find them. For this reason, the instructor may want to make up his or her own exercises. But the ones in this guide, as well as the ones in the book, should hopefully provide a good starting point that will inspire the instructor's own creativity.

Be that as it may, the programming challenges at the end of Chapter 6 (the Game of Life) and Chapter 16 (a menu/command system) are likely to be particularly useful and challenging to new programmers. Perhaps only the most ambitious students should have these challenges thrown at them… or, these exercises may be challenging enough for group projects. In any case, the instructor should feel free to provide hints and clues for some of the more esoteric and difficult C++ statements that have to be written.

By the way, I do not give the answers to book exercises here, for they can all be found online. To download these answers, a student need only start by my personal website for the book:

brianoverland.com/books

Chapter 1: Start Using C++

Scope of Chapter 1

To have early success learning a language (human or otherwise), a student needs to start with simple communication; in the case of computer languages, that means to be successful writing, compiling, and testing programs that 1) get some input; 2) do simple calculations; and 3) print some output.

Therefore, students should start with a "Hello, World" program and progress from there to simple I/O. Everything else that a program does can be built on top of these simple operations.

For this reason, I don't recommend learning a language by sitting down and memorizing lists of keywords or trying to absorb the entire syntax of the language--although you can get an overview of language syntax in Appendix B (starting on page 491) if desired. In addition, Appendix A (page 475) gives an overview of C++ operators; but again, it is the author's opinion that the novice doesn't really need to start by memorizing these.

Chapter 1 Objectives

Upon completing Chapter 1, the student should be able to:

- Successfully install Microsoft Visual Studio or another Interactive Development Environment (IDE) supporting C++.
- Understand the basic write-compile-test-debug cycle. (Page 10.)
- Become familiar with basic programming vocabulary, including "source code" and "compiler." (Pages 5 and 6.)
- Write and test a program that uses **cout** to print a simple message. (Pages 11-15.)
- Use **endl** to print extra lines. (Page 16.)
- Get input with **cin**.

Notes on Alternative Learning Paths

Different C++ compilers—including several free compilers—are available. As an instructor, you should know what compilers your students are using; pay special attention to making sure the students know the exact procedures for whatever compiler they have.

Note: If you or students have any problems installing Microsoft Visual Studio, check the book errata at:

/brianoverland.com/books

In particular, installation may have some issues; so look at the "Errata" down below.

This text uses **cin** and **cout** for input and output exclusively (as opposed to **printf** and **scanf**), for the following reasons:

- Use of **cin** and **cout** provides early success with objects (because **cin** and **cout** are built-in objects provided by the C++ standard library).
- Use of **cin** and **cout** is extensible with new data types—that is, classes—that the programmer provides him or herself. Later on, the student will learn how to print his or her own objects with **cout**.
- For a beginner, one's first programs are actually easier to write with **cin** and **cout** than with **printf**, in the author's opinion.

However, **printf** and **scanf** work in C++ and are fully backward compatible with C. C programmers may want to know why **printf** and **scanf** are not being used.

Note: Appendix G (beginning on page 525) shows how to use I/O stream functions and objects to format output just as you would with **printf** and **scanf**.

Errata for Chapter 1

This applies to the first printing only:

Page 1. The downloading program "vc_community," mentioned twice, should be "vs_community."

Page 2. Steps 3 and 4 are wrong. They should say that you choose Custom Installation, not Typical Installation. Then, in the Custom Install window, de-select every option but "Programming Languages." That option must be switched ON for generic C++ code to be compiled correctly. (This error is due in part to Microsoft's changing the project settings back and forth while I was writing the book.)

Page 4. When creating a project, instead of choosing "Finish" (as mentioned at top of page), choose "Next." Then, in the Project Options window, de-select "Precompiled Headers" and "Security Development Lifetime" checks. De-selecting Precompiled Headers will allow you to use 100% generic C++ code without change. This is recommended, although not 100% essential.

Answers to Exercises

Answers to all the exercises in the book are provided at brianoverland.com/books.

Suggested Additional Exercise

Any exercise for Chapter 1 is useful as long as it: 1) prompts for input; 2) does a simple calculation; and 3) prints the results.

Suggested problem: Write a program that prompts for a time value, in seconds. Then determine how far a free-falling object falls in that time according to the formula, which produces distance in feet, calculating it from 16 feet multiplied by time in seconds, squared:

d = 16.0 x (t * t)

Alternatively, you can produce the result in meters, again multiplying by seconds squared:

d = 4.9 x (t * t)

Solution to Additional Exercise

```
#include <iostream>
using namespace std;

int main() {
     double t = 0.0;
     cout << "Enter time elapsed, in seconds: ";
     cin >> t;
     cout << "Distance object will fall is : ";
     cout << 16.0 * t * t << endl;
     return 0;
}
```

Chapter 2. Decisions, Decisions

Scope of Chapter 2

Once a programmer can read input, do simple calculations, and print output, the next step in learning to master any programming language is to write programs that make decisions, using **if** and **else**. Once this step is mastered, it is a simple matter to go from there to understanding **while** loops.

Chapter 2 Objectives

Upon completing Chapter 2, the student should be able to:

- Understand how to use simple **if** and **if-else** statements
- Understand how the C++ language evaluates true and false conditions
- Understand the difference between test-for-equality (==) and assignment (=)
- Use simple Boolean operators AND (&&) and OR (||) to combine conditions
- Write simple **while** loops.

Notes on Additional Learning Paths

Although the book introduces **do-while** and switch statements in the next chapter (Chapter 3), you might want to briefly mention them earlier, to give students a broader overview.

If students are familiar with some version of Basic, you might compare and contrast with the syntax for Basic's **Do While** loops. Or you might want to make a comparison to Python's **do** keyword.

Pitfalls to Watch Out For

Every student should carefully note the difference between test-for-equality (==) and assignment (=), because C and C++ allow both inside a conditional test… even though test-for-equality (==) is usually what is meant. Not remembering to distinguish these can cause hours of debugging!

Answers to Exercises

Answers to all the exercises in the book are provided at brianoverland.com/books.

Suggested Additional Exercises for Chapter 2

First Additional Exercise: Triangle Numbers

When introducing this exercise, you may want to alert students to the use of the += operator, in which

```
amount += n;
```

means the same as

```
amount = amount + n;
```

Exercise: Write a program that prompts for an integer N and calculates Triangle(N), which is equal to:

1 + 3 + 4 + 5 + … + N

Print the results.

Second Additional Exercise: Factorial Numbers

When introducing this exercise, you may want to alert students to the use of the *= operator, in which

```
amount *= n;
```

means the same as

```
amount = amount * n;
```

Exercise: Write a program that prompts for an integer N and calculates Factorial(N), which is equal to:

1 * 3 * 4 * 5 * … * N

Print the results. (Note that you can calculate factorials only for relatively low numbers. Factorials quickly become so large that they result in arithmetic overflow.)

Solutions to Additional Exercises for Chapter 2

The solution for calculating triangle numbers is:

```
#include <iostream>
using namespace std;

int main() {
     int amount = 0;
     int i = 1;

     cout << "Enter n: ";
     cin >> n;
     while (n-- > 0) {
           amount += i++;

     }
     cout << "Triangle(N) = " << amount << endl;
     return 0;
}
```

The solution for calculating factorial numbers is:

```
#include <iostream>
using namespace std;

int main() {
     int amount = 1;
     int i = 1;

     cout << "Enter n: ";
     cin >> n;
     while (n-- > 0) {
           amount *= i++;
     }
     cout << "Factorial(N) = " << amount << endl;
     return 0;
}
```