Part Zero

Introduction

Suppose we want to write a function that computes the average of a list of numbers. One implementation is given here:

```
double GetAverage(double arr[], int numElems) {
    double total = 0.0;
    for(int h = 0; h < numElems; ++h)
        total += arr[h] / numElems;

    return total;
}
```

An alternative implementation is as follows:

```
template <typename ForwardIterator>
double GetAverage(ForwardIterator begin, ForwardIterator end) {
    return accumulate(begin, end, 0.0) / distance(begin, end);
}
```

Don't panic if you don't understand any of this code – you're not expected to at this point – but even without an understanding of how either of these functions work it's clear that they are implemented differently. Although both of these functions are valid C++ and accurately compute the average, experienced C++ programmers will likely prefer the second version to the first because it is safer, more concise, and more versatile. To understand why you would prefer the second version of this function requires a solid understanding of the C++ programming language. Not only must you have a firm grasp of how all the language features involved in each solution work, but you must also understand the benefits and weaknesses of each of the approaches and ultimately which is a more versatile solution.

The purpose of this course is to get you up to speed on C++'s language features and libraries to the point where you are capable of not only writing C++ code, but also critiquing your design decisions and arguing why the cocktail of language features you chose is appropriate for your specific application. This is an ambitious goal, but if you take the time to read through this reader and work out some of the practice problems you should be in excellent C++ shape.

Who this Course is For

This course is designed to augment CS106B/X by providing a working knowledge of C++ and its applications. C++ is an industrial-strength tool that can be harnessed to solve a wide array of problems, and by the time you've completed CS106B/X and CS106L you should be equipped with the skill set necessary to identify solutions to complex problems, then to precisely and efficiently implement those solutions in C++. This course reader assumes a knowledge of C++ at the level at which it would be covered in the first two weeks of CS106B/X. In particular, I assume that you are familiar with the following:

0. How to print to the console (i.e. `cout` and `endl`)
1. Primitive variable types (`int`, `double`, etc.)
2. The `string` type.
3. `enum`s and `struct`s**.**
4. Functions and function prototypes.
5. Pass-by-value and pass-by-reference.
6. Control structures (`if`, `for`, `while`, `do`, `switch`).
7. CS106B/X-specific libraries (`genlib.h`, `simpio.h`, the ADTs, etc.)

If you are unfamiliar with any of these terms, I recommend reading the first chapter of *Programming Abstractions in C++* by Eric Roberts and Julie Zelenski, which has an excellent treatment of the material. These concepts are fundamental to C++ but aren't that particular to the language – you'll find similar constructs in C, Java, Python, and other languages – and so I won't discuss them at great length. In addition to the language prerequisites, you should have at least one quarter of programming experience under your belt (CS106A should be more than enough). We'll be writing a lot of code, and the more programming savvy you bring to this course, the more you'll take out of it.

How this Reader is Organized

The course reader is logically divided into six sections:

0. **Introduction**: This section motivates and introduces the material and covers information necessary to be a working C++ programmer. In particular, it focuses on the history of C++, how to set up a C++ project for compilation, and how to move away from the `genlib.h` training wheels we've provided you in CS106B/X.

1. **A Better C**: C++ supports *imperative programming*, a style of programming in which programs are sequences of commands executed in order. In this sense, C++ can be viewed as an extension to the C programming language which makes day-to-day imperative programming more intuitive and easier to use. This section of the course reader introduces some of C++'s most common libraries, including the standard template library, and shows how to use these libraries to build imperative programs. In addition, it explores new primitives in the C++ language that originally appeared in the C programming language, namely pointers, C strings, and the preprocessor.

2. **Data Abstraction**. What most distinguishes C++ from its sibling C is the idea of *data abstraction*, that the means by which a program executes can be separated from the ways in which programmers talk about that program. This section of the course reader explores the concept of abstraction, how to model it concretely in C++ using the `class` keyword, and an assortment of language features which can be used to refine abstractions more precisely.

3. **Object-Oriented Programming**. Object-oriented programming is an entirely different way of thinking about program design and can dramatically simplify complex software systems. The key concepts behind object-orientation are simple, but to truly appreciate the power of object-oriented programming you will need to see it in action time and time again. This section of the course reader explores major concepts in object-oriented programming and how to realize it in C++ with inheritance and polymorphism.

4. **Generic Programming**. Generic programming is a style of programming which aims to build software that can tackle an array of problems far beyond what it was initially envisioned to perform. While a full treatment of generic programming is far beyond the scope of an introductory C++ programming class, many of the ideas from generic programming are accessible and can fundamentally change the ways in which you think about programming in C++. This section explores the

main ideas behind generic programming and covers several advanced C++ programming techniques not typically found in an introductory text.

5. **More to Explore**. C++ is an enormous language and there simply isn't enough time to cover all of its facets in a single course. To help guide further exploration into C++ programming, this course reader ends with a treatment of the future of C++ and a list of references for further reading.

Notice that this course reader focuses on C++'s standard libraries before embarking on a detailed tour of its language features. This may seem backwards – after all, how can you understand libraries written in a language you have not yet studied? – but from experience I believe this is the best way to learn C++. A comprehensive understanding of the streams library and STL requires a rich understanding of templates, inheritance, functors, and operator overloading, but even without knowledge of these techniques it's still possible to write nontrivial C++ programs that use these libraries. For example, after a quick tour of the streams library and basic STL containers, we'll see how to write an implementation of the game Snake with an AI-controlled player. Later, once we've explored the proper language features, we'll revisit the standard libraries and see how they're put together.

To give you a feel for how C++ looks in practice, this course reader contains several extended examples that demonstrate how to harness the concepts of the previous chapters to solve a particular problem. I *strongly* suggest that you take the time to read over these examples and play around with the code. The extended examples showcase how to use the techniques developed in previous chapters, and by seeing how the different pieces of C++ work together you will be a much more capable coder. In addition, I've tried to conclude each chapter with a few practice problems. Take a stab at them – you'll get a much more nuanced view of the language if you do. Solutions to some of my favorite problems are given in Appendix One. Exercises with solutions are marked with a diamond (♦).

C++ is a large language and it is impossible to cover all of its features in a single course. To help guide further exploration into C++ techniques, most chapters contain a "More to Explore" section listing important topics and techniques that may prove useful in your future C++ career.

Supplemental Reading

This course reader is by no means a complete C++ reference and there are many libraries and language features that we simply do not have time to cover. However, the portions of C++ we do cover are among the most-commonly used and you should be able to pick up the remaining pieces on a need-to-know basis. If you are interested in a more complete reference text, Bjarne Stroustrup's *The C++ Programming Language, Third Edition* is an excellent choice. Be aware that *TC++PL* is *not* a tutorial – it's a reference – and so you will probably want to read the relevant sections from this course reader before diving into it. If you're interested in a hybrid reference/tutorial, I would recommend *C++ Primer, Fourth Edition* by Lippman, Lajoie, and Moo. As for online resources, the C++ FAQ Lite at www.parashift.com/c++-faq-lite/ has a great discussion of C++'s core language features. cplusplus.com has perhaps the best coverage of the C++ standard library on the Internet, though its discussion of the language as a whole is fairly limited.

Onward and Forward!