Appendix F What is C++?

Procedural Versus Object-Oriented Programming

C is a procedural language, as are BASIC, Pascal, FORTRAN, and most other programming languages. In a *procedural* language, functions (sometimes called *procedures*) take center stage, and the data plays second fiddle. If you think about what you've learned in this book, you'll probably agree that this is true. The entire structure of your program is designed around functions, with data being shuffled from one function to another. The emphasis is on what is done to the data--displaying it, sorting it, saving it to disk, and so on.

Procedural programming is a very powerful approach that has been used to create the majority of the commercial programs available today. Yet, as programs became more complex, certain shortcomings of the procedural approach started to become evident. The difficulty of writing, debugging, maintaining, and modifying procedural programs became unmanageable. Rather than trying to design an improved procedural language, the necessary next step seemed to be to create a whole new approach to programming.

That approach was *object-oriented programming* (OOP). C++ is an object-oriented language, as is SmallTalk, the only other object-oriented language you're likely to have heard of. OOP places the emphasis on data, and functions play a subsidiary role. That's what an object is--a chunk of data. However, as you'll see, an object in C++ is nothing like the data you're accustomed to working with in C.

What's the Object?

An *object* is a chunk of data. Using OOP, you design your program around the data it will handle. If you're using objects in your C++ program (and, as you'll soon see, you don't have to use objects), much of your programming time will be spent creating objects. What's the difference between a C++ object and a C data item? The difference is that a C++ object is smart--it knows what to do.

Confused? I can't blame you. Hang on, though, and I'll try to explain. A C++ object contains both data and code. To be more specific, an object contains data plus the code required to manipulate the data in the desired ways. An example should clarify this.

Suppose you need to maintain a long list of numbers and manipulate it in various ways. For example, you need to be able to get the largest value in the list, the smallest value, the average value, and the median value. You also need to sort the list in both ascending and descending order. In C, you would write a variety of functions to perform the needed actions, and you would declare an array to hold the numbers. The array would be passed to the functions as an argument, and the result might be passed back as the return value.

In C++, you would design an object (also called a *class*) to hold and manipulate the data. The object would contain an array to store the data and functions to manipulate the data. In effect, the data object would know how to find its minimum, its maximum, and so on. Most important, the data and the functions would be isolated from the rest of the program, removing the possibility of the sort of unintentional interaction that can cause problems in procedural programs.

There's more to OOP than this, but I hope this gives you some idea of what it's all about. There's more to C++ than its object-oriented capabilities, however. Now let's take a closer look at C++.

Good Old C, with a Plus

If you apply your deductive skills, you can tell something about C++ from its name alone. Remember what ++ means in the C language--it's the unary increment operator. So the expression "C++" evaluates to the value of C plus 1. And that's exactly why this new language was given the name C++--it's the C language plus something. Actually, there are several new somethings in C++, but that's not the point. The point is that the entire C language is part of C++. With a few minor changes, you can use everything you've learned about C in a C++ program. Figure F.1 illustrates the relationship between C and C++.

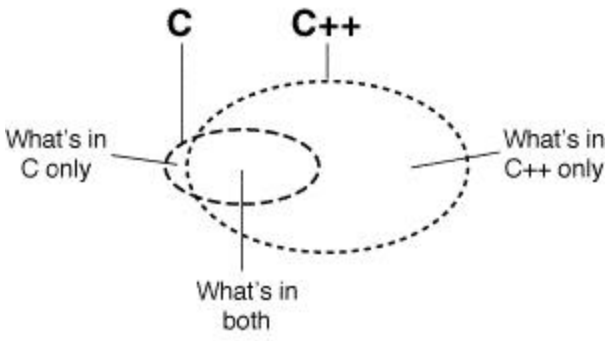

Figure F-1. The relationship between C and C++.

You can see that there's a region common to both languages. This means that if you know C, you're well on your way to learning C++. The large area representing things present in C++ but not in C shows you how much has been added to the new language. Objects are a big part of this, but certainly not all of it. Objects aside, there are lots of neat new things in C++. I can't go into details here, but once you have a little C programming under your belt, you might want to give C++ a try. I think you'll like it!

Your First C++ Program

Although I can't teach you C++ in a single chapter, I can show you a simple C++ program that illustrates how C++ doesn't look all that different from C. It also demonstrates one of C++'s new features. I have created a variation on the traditional "Hello, world" program, as shown in Listing F.1.

Listing F.1. A simple C++ program.

```
1: #include <iostream.h>
2:
3: main()
4: {
5:     float pi = 3.14;
6:     cout << "The value of pi is " << pi << "\n";
7:     return 0;
8: }
The value of pi is 3.14
```

ANALYSIS: Much of this program should make sense to you. There's an #include statement on line 1, and although the name of the file might be unfamiliar, you should be able to understand what's going on here. The main() function with its enclosing brackets on lines 3, 4, and 8 is familiar, as is the declaration and initialization of a type float variable on line 5.

Line 6 is certainly strange, though. What does cout refer to, and why are there two left-shift operators? cout is one of C++'s predefined streams. It's actually the same as stdout, but you use it differently. To send data to cout, you use the insertor operator << (so called because it inserts data into the stream). But isn't << the left-shift operator? Indeed it is, but here you're seeing one of C++'s new and most powerful features in action: operator overloading. An operator can have different meanings in different contexts. When placed after an integer variable, << is interpreted as the left-shift operator, but when placed after the name of an output stream, it's interpreted as the insertor operator. Line 6 inserts the string constant "The value of pi is" into the cout stream and then inserts the value of the variable pi. The cout stream is smart enough to apply default formatting to the numerical value.

Finally, there's a return statement on line 7. Isn't return used to return a value from a function to the program that called the function? Yes, indeed. Remember that main() itself is a function, and you can think of running a C++ program as calling the main() function from the operating system. The operating system might or might not make use of the return value. A C program could return a value from main(), but C++ requires it. This is an exam-ple of C++'s stricter rules that are designed to enforce consistent programming.

What About Compilers?

If you've been shopping for a commercial C compiler, you might have noticed that it's difficult to find one. Everything seems to be for C++ these days. Remember, however, that C is part of C++, so go ahead and buy that C++ compiler that's on sale. You can use it for your C programs, and you'll be ready if and when you decide to take up C++.

A C++ compiler is actually three compilers in one. If your program's source code file has a .C extension (such as PROGRAM.C), the compiler will treat it as a C program. All of C's rules apply, and all of its capabilities are available. Therefore, you should give your files a .C extension if you're using a C++ compiler to work through this book. On the other hand, if your program has a .CPP extension (or, with some compilers, .CXX), it is treated as a C++ program, and C++'s rules and features are available. In other words, you control the rules and restrictions that the compiler applies with the source-code file extension.

I've described only two compilers. Where's the third one I promised? This "third" compiler comes into play when you write a C program and then compile it as a C++ program (by giving it a .CPP extension). Just because you use a C++ compiler doesn't mean you have to use all of C++'s object-oriented features in your program--or any of them, for that matter! This approach offers two advantages. Your program is subjected to C++'s stricter rules, which are designed in part to prevent certain programming practices that, although perfectly legal in C, sometimes create problems. Also, you can slowly start incorporating elements of C++ into your programs, which can be an excellent way to learn that language as you work.

Where to Go from Here

If you're seriously interested in learning C++, I strongly recommend getting a good introductory book. You can't do better than *Sams' Teach Yourself C++ Programming in 21 Days* (Sams Publishing, 1994), which takes the same structured approach to C++ that was used in this book.

Summary

This appendix provided a brief introduction to the C++ language. You learned that C++ is an object-oriented extension of the C language. It includes everything C does plus a host of additions and enhancements that provide significantly more programming power and convenience. The major addition is support for object-oriented programming, a technique that gives priority to a program's data in the design of the program. C++ can be used for any size of program, but its advantages are particularly evident when you create large, complex programs. You also saw a simple C++ program that illustrates two of the language's new features: operator overloading and improved standard streams.

Q&A

Q I'm pretty comfortable with the material presented in this book. How can I continue improving my skills as a programmer?

A One way is to start learning C++. You can also explore more advanced C programming topics. There are numerous good books on this topic, including *Sams' Teach Yourself Advanced C in 21 Days* (Sams Publishing, 1994). The best way to improve your skills, however, is by programming. There's no substitute for writing real programs.

Q When are you most likely to see the advantage of C++ over C?

A The new features of C++ can be put to good use in almost any program, no matter how small. It's in large and complex programs, however, that these advantages can really make a difference.

Q I'm not sure whether I need to learn C++. I don't want to waste time learning something I won't ever use, but on the other hand, I don't want to fall behind. Do you have any advice?

A This is a tough question to answer, because I don't know the kind of programming you'll be doing. Generally speaking, the more serious you are about programming, the more strongly I recommend that you learn C++. In particular, if you plan to program for a living, you need to know C++.

Wrap-Up

I hope you've found this book useful, and maybe even enjoyable, in your quest to learn C programming. You might program only occasionally as a hobby, or you might become an ace programmer working for Microsoft, Borland, or Lotus. In any event, I think the lessons you've learned here will serve you well.