A Brief History of C

The reason the language is called C is that it is the successor to the language called B. That language was developed by Ken Thompson in 1970 while working on a DEC PDP-7, a much less powerful machine than most modern PCs. The original UNIX operating system ran on that machine, and that's also where B got its start. (B itself was the successor to a language called BCPL, which had been written by Martin Richards.)

However, B was a little restricted. In 1972, Dennis Ritchie and Ken Thompson created the C language to augment B's power. C did not become immediately popular after its creation; in fact, it remained almost an esoteric topic for the next six years. In 1978, however (a historic year for C programmers), Brian Kernighan and Dennis Ritchie wrote a famous book — The C Programming Language (Prentice Hall, 1978). And that simple book changed everything.

Once the word was out, there was an explosion of interest, and C was implemented on 8-bit computers under the CP/M operating system. It wasn't until the introduction of the IBM PC in 1981, however, that C really came into its own. When the PC revolution began, C was in a perfect place to take ad vantage of it. As the number of PCs shot upwards, so did the number of C users. C broke away from its original UNIX background and became a popular language on microcomputers.

It's worth stressing that C became popular for a very good reason — programmers liked using it. Unlike many other languages, C gives the programmer a great deal of control over the computer. With that control comes responsibility — there are many things you can do in C that will ruin your program or crash your computer. That is, you have the power to do things in C that other languages would never allow you to do. And programmers liked that very much; they liked finding a language that was a tool, not an obstacle.

Until that time, the only true way to get complete control over your computer was to use assembly language — the native language of the 80x86 microprocessor itself (I'll use the term 80x86 to stand for any of the Intel chip series from the 8088 up). The 80x86 interprets the bytes and words it sees in memory as instructions, and when we give English language names to those low-level instructions, it's called assembly language. This is the true language of your computer, the instructions that are built into the actual microprocessor, the 80x86.

Yet it is difficult to write long or user friendly programs in assembly language — in fact, it may take you dozens of pages of frustration to get where you want to go. C builds on that original foundation; many C instructions are quite close to their assembly language equivalents. But many more powerful instructions have been added, as well as whole libraries of prewritten programs ready for us to use. To programmers, C was much like assembly language without the drawback of having to do everything for yourself — in other words, C was the perfect combination of control and programming power.

ANSI Standardizes C

All this made for such a popular language that different companies started to bring out their own versions of C, and each one began to go in a different direction. The C revolution was in danger of splintering into many incompatible programming packages. For that reason, the American National Standards Institute (ANSI) created a special subcommittee named X3J1 1 to create a standard version of C. This was an extremely important development for C programmers; the language, which had been going off in all directions at the same time, became standardized and coherent once more. For that reason, ANSI C did indeed become standard C.

On the other hand, ANSI was interested in codifying C for all computers, not just the PC. But there are many things that are specific to the PC which does not really apply to, say, mainframe computers. For example, one important area is the way the PC handles memory, which is quite unlike any computer which is not 80x86-based. For that reason, most implementations of C now adhere to the ANSI standard as far as it goes, and then they add their own extensions. The two versions of C that we're covering in this book, one from Borland and one from Microsoft, agree in most particulars; however, they start to differ when there is no ANSI standard to agree on. The ANSI standard says nothing about screen graphics, for example, so we'll see that there is considerable difference between Turbo and Microsoft C when it comes to drawing pictures.

Why Use C?

In today's world of computer programming, there are many high-level languages to choose from, such as C, Pascal, BASIC, and Java. These are all excellent languages suited for most programming tasks. Even so, there are several reasons why many computer professionals feel that C is at the top of the list:

→ C is a powerful and flexible language. What you can accomplish with C is limited only by your imagination. The language itself places no constraints on you. C is used for projects as diverse as operating systems, word processors, graphics, spreadsheets, and even compilers for other languages.

→ C is a popular language preferred by professional programmers. As a result, a wide variety of C compilers and helpful accessories are available.

→ C is a portable language. Portable means that a C program written for one computer system (an IBM PC, for example) can be compiled and run on another system (a DEC VAX system, perhaps) with little or no modification. Portability is enhanced by the ANSI standard for C, the set of rules for C compilers.

→ C is a language of few words, containing only a handful of terms, called keywords, which serve as the base on which the language's functionality is built. You might think that a language with more keywords (sometimes called reserved words) would be more powerful. This isn't true. As you program with C, you will find that it can be programmed to do any task.

→ C is modular. C code can (and should) be written in routines called functions. These functions can be reused in other applications or programs. By passing pieces of information to the functions, you can create useful, reusable code.

→ As these features show, C is an excellent choice for your first programming language. What about C++? You might have heard about C++ and the programming technique called object-oriented programming. Perhaps you're wondering what the differences are between C and C++ and whether you should be teaching yourself C++ instead of C.

C++ is a superset of C, which means that C++ contains everything C does, plus new additions for object-oriented programming. If you do go on to learn C++, almost everything you learn about C will still apply to the C++ superset. In learning C, you are not only learning one of today's most powerful and popular programming languages, but you are also preparing yourself for object-oriented programming.

Another language that has gotten lots of attention is Java. Java, like C++, is based on C. If later you decide to learn Java, you will find that almost everything you learned about C can be applied.