1

What Is Programming?

If you've picked up a book on programming, you must have a pretty good idea what programming is. After all, why would you want to learn something if you don't know what it is? However, many students who are new to the wonderful world of programming, or even practitioners of the art who have learned a little programming to do a particular job and built on their experience from there, might benefit from a quick rundown of the history of programming, what programming is, and where it's at right now.

The History of Programming

The history of programming spans more years than most people would imagine. Many people think programming is an invention of the late twentieth century. In fact, the history of modern programming and programming languages dates back nearly 60 years to the mid-1940s.

However, before we pick up the story in the 1940s, we need to go still farther back in time, all the way back to 1822 and Charles Babbage. While he was studying at Cambridge University in Britain, Babbage came upon the realization that many of the calculating devices of the time, such as astronomical tables, tidal charts, and navigational charts, all contained critical errors and omissions. These errors caused the loss of many ships at sea, along with lives and cargo. Because he considered human error to be the source of these inaccuracies, his idea was to take the guesswork out of creating and maintaining such tables and charts by using steam-powered machines. One such machine, dubbed the Difference Engine, was to occupy much of Babbage's time for the rest of his life. He even approached the British government for financial assistance — the first (but by no means last) request for a government grant to fund computer science research.

After 10 years of working on the Difference Engine, Babbage realized that it was a single-purpose machine that could ultimately only carry out one operation. He realized that this was a major limitation and, for a time, abandoned it in favor of the more versatile Analytical Engine. This engine would contain the basic components of a modern computer and led to Babbage's being called the "father of the computer." The Analytical Engine did not gain wide acceptance because Babbage suffered from an affliction that has plagued programmers and computer scientists across the centuries: an inability to clearly document and convey his ideas!

Work on the Analytical Engine continued until 1842, when the British government, tired of the lack of progress and results, abandoned the idea and withdrew its funding. However, Babbage still continued to work on it until 1847 when he returned to work on the Difference Engine, and between then and 1849 he completed 21 detailed drawings for the construction of the second version of the engine. However, Babbage never got around to actually completing either of the devices. In fact, it wasn't until after Babbage's death in 1871 that his son, Henry Prevost, built several copies of the simple arithmetic unit of the Difference Engine and sent them to various institutions around the world, including Harvard University, to ensure their preservation.

Progress continued throughout the 1800s, and in 1854 Charles Boole described the symbolic logic system that bears his name (Boolean logic) which is still in use today (and is covered later in this book). This system took logic terms such as greater than, less than, equal to, and not equal to and created a symbolic system to represent them.

It's said that necessity is the mother of invention, and necessity came to a head in 1890 when the U.S. Congress wanted to add more questions to the census. The increasing U.S. population meant that processing this data took longer and longer, and it was estimated that the census data for 1890 would not be processed before the 1900 census unless something could be done to improve and speed up the process.

So, a competition was created to drum up interest in computing and to deliver data processing equipment to the government. This contest was won by Herman Hollerith, and after successfully proving the technology, he went on to process census information in other countries and later founded a company called Hollerith Tabulating Co. (This company eventually became one of three that joined together in 1914 to create CTR, the Calculating Tabulating Recording Company. That name might not be familiar to you, but 10 years later the company was renamed International Business Machines, or IBM — a name you will undoubtedly have heard of!)

> *Hollerith also has another first to his name — his machines were the first ever to appear on a magazine cover!*

Progress then seemed to slow down a little and by the mid-1920s digital computing tools were rarely used in business, industry, or engineering. In fact, the tool most commonly used was the humble analog slide rule. Things began to change in 1925, however, when, while at MIT, Vannevar Bush built a large-scale differential analyzer that incorporated integration and differentiation capability. This massive creation was funded by the Rockefeller Foundation and was the largest "computer" in the world in 1930.

The next major player in this story is a German scientist by the name of Konrad Zuse. In 1935, Zuse developed his Z-1 computer. What was more remarkable than the fact that he built this computer in his parents' living room was that this was the first computer to make use of relays and to be based on the binary system of counting. It was the first of its kind and heralded the modern age of computers.

Zuse continued his work and developed the Z-2 in 1938 with the help of Helmut Schreyer. He applied for financial assistance from the German government in developing and constructing his machines, but he was turned down because the project would take longer to complete than the war was expected to last. Zuse eventually fled to Hinterstein when the war came to an end, later making his way to Switzerland where he rebuilt his Z-4 machine at the University of Zurich.

It is Zuse who also gave birth to modern programming in 1946 when he developed the world's first programming language, called Plankalkül. He even went as far as writing code for the Z-3 to play chess against him. This language was groundbreaking and contained much of what is seen in modern languages, such as tables and data structures.

Zuse later went on to found a computer company that was eventually integrated into the Siemens Corporation.

The year 1945 saw another important development in computing, the introduction of the one word that we are all pretty sick of hearing — bug! In 1945, Grace Murray Hopper (later Admiral Hopper) was working on the Harvard University Mark II Aiken Relay Calculator. Machines in those days experienced continuous problems, and during one of these incidents on September 9, 1945, an investigation by a technician revealed that there was a moth trapped in between the circuitry (the record shows that it was in the points of Relay #70, on Panel F). The technician removed the moth and affixed it to the logbook that documented the computer's use and problems. The entry in the logbook reads: "First actual case of bug being found." The phrase that was born from this was that the machine had been "debugged" and the terms "debugging a computer" and later "debugging a computer program" were born.

> *History does show that although Grace Hopper was always careful to admit that she was not there when it actually happened, it was one of her favorite stories that she recounted often.*

Things now begin to speed up. The year 1949 saw the development of a language called *Short Code*. This code had to be made into machine-readable code by hand (a process known as *compiling*), so there was very little about it that was "short."

In 1954, IBM began developing a language called FORTRAN (which stands for *FORmula TRANslator*). FORTRAN was published in 1959 and became an immediate success because of its easy-to-use input and output system and because the code was short and easy to follow (and is still in use in some places today). FORTRAN also was the first commercial high-level programming language, meaning that the code was easier for a human to read because it follows the familiar rules of syntax, grammar, and structure.

In 1958, FORTRAN II and ALGOL were published, as well as work being carried out on LISP, while in 1959 another popular and long-lived language called COBOL (Common Business Oriented Language) was created at the Data Systems and Languages conference (CODASYL). COBOL is a commercial institutional language used primarily on large machines and is still used in many companies today.

Things continued at breakneck speed with many new languages and variants on existing languages developed and released. During 1968, work started on a language called *Pascal*. Pascal was released in 1970 and is still used extensively for educational purposes. The year 1970 also saw the publication of two other languages that revolutionized computing. These are Smalltalk and B-language. Smalltalk was important because it was a language based completely on objects (don't worry about this just yet), and B-language was important because of what it led to.

What did B-language lead on to? Well, in 1972 Dennis Ritchie created a language based on B-language which he eventually called C (after calling it NB for a while). C brought with it simplicity, efficiency, and flexibility and heralded a new age in programming in which it was possible to do a lot more with less code, more quickly and easily than before.

The year 1975 saw the release of TinyBASIC by Dr Wong. TinyBASIC took only 2 KB of memory and was loaded onto the computer via paper tape. It was groundbreaking too because it was the first known freeware (that is, it was free to use and did not need to be purchased) program.

> *The text strings "All Wrongs Reserved" and "Copyleft" were found within this program.*

Coincidently, 1975 was also the year that a young Bill Gates and Paul Allen wrote and sold to MIT their own version of the BASIC language.

Things continued at breakneck speed throughout the 1970s, 1980s, and 1990s, with more and more developments that lead us to where we are now: with a myriad of different programming languages that each has its own strengths and weaknesses. We've also seen the appearance of the Internet, which has bought with it a whole host of languages. The other huge advantage of the Internet is that it is now easy to share information and programs with others, meaning that the interest in programming and programming languages has grown and grown, fueled by the free exchange of information, ideas, and applications.

We've taken a brief trip through the history of programming, highlighting some of the main events that bought us to where we are now. With that done, it's time to take a look at what programming actually is.

What Is Programming?

There is no single answer to this question. Programming means different things to different people. However, there is a definition I've come up with that I'm quite happy with:

Programming is the ability to talk to the computer in a language it can understand and using grammar and syntax that it can follow to get it to perform useful tasks for you.

Put into its simplest terms, that's all that programming is! You write code and the computer interprets your request and does something.

Doing something is a vital part of programming. You always program the computer to do something (even if that is waiting for another instruction!). Programming is all about doing something and moving forward. Whenever you are programming, no matter how simple or complicated your task, you are always giving the computer an instruction. Usually these instructions are given one at a time. Even though sometimes it can seem as if you're asking your computer to do a lot of different things "all at once," you are in fact giving it step-by-step instructions as to what to do.

The code also has to be correct and unambiguous. It can't contain any mistakes, errors or ambiguity. Any of these cause the code to fail and errors to occur. There is no capacity for the computer to make guesses about what you meant or to correct your code as it goes along.

Another thing you will notice as you begin to program is that rarely will you write code that does only one thing and one thing only. Even the simplest of projects usually consists of several steps, for example:

1. Program is executed.
2. Initial parameters are checked.
3. Parameters are changed.
4. Clear up after running.
5. Program exits.

Why So Many Programming Languages?

But if you are giving the computer instructions in a form that it can understand, how come there are so many different programming languages? Surely you write in code that the computer understands?

Well, while it's true that different types of computer understand different types of code (so a PC is fundamentally different from a Macintosh), each type of computer really only understands one language, and that's not the language you write code in. The code you type in is not the code that the computer then goes on to understand. A program (either an interpreter or a compiler) is required to convert the code from text into a binary language that the computer can decipher. This is the great part about computer language.

A computer operates by reading instructions in binary, which you might be familiar with. It is a number system but it differs from our usual base 10 system by only having two digits — 0 and 1. Figure 1-1 shows the decimal numbers 0 to 9 represented as binary.

Decimal digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Binary bits: 0, 1

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Figure 1-1

Now take a close look at Figure 1-2.

```
0100001001101001101101110011000010111001001111001001000000110100101110011001000
0011101000110010101100100011010010110111101110101011100011001000000110000101101110
011001000010000001101001011001100010000001110111011001010010000001101000011000
0000110100101101110001000000110001001101001011011100110000101110010011110010010 0
0000001110111011010010111010001101000001000001101111011101010111001000100000011
0001101101111011011010111000001110101011101000110010101110010011100110010000 0111
0101011100110110100101101110011001110010000001101001011101000010000000110000101 1
0100011011000010000000111010001101000011001010010000000111010000110100101101010 1
1001010010000000111010001101000011010010110111001100111011100110010000000111011101
1011110111010101101100011001000010000000110011101100101011101000010000000111011001
10001010111001001111001001000000110001101101111011011010111000001101100011010010 11
000110110000101110100011001010110010000100000001101001011011100110010001100101 01
100101011001000010000000010100001100001011011000111010001101000011011110111010 10
1100111011010000001000000110111101110101011101000010000000110101101100101011100101
1000100110111101100001011100100110010000111001100100000001110111011011110111010101 1
0110001100100001000000011000100110010100100000001100001001000000110110001101111
0111010000100000011100110110100100110110101110000001101100011001010111001000010000
1001010010010110
```

Figure 1-2

Binary is tedious and if we had to work in binary with our computers using it all the time things would get very complicated indeed (although our keyboards would be a lot simpler!).

Wondering what Figure 1-2 means? That's the last paragraph written out with binary used to represent the letters using a system called ASCII (American Standard Code for Information Interchange). Don't worry just yet about what all that means (later on you will be able to come back to that and read it if you want), for now, just be thankful that programming will not mean writing out endless streams of zeros and ones!

When you are programming, or, more specifically, writing code, what you are actually doing is writing code that another program can understand to translate that code into something that the computer can understand, hence the name *interpreter*. So, when you are writing code, it is not the rules of the computer you have to follow, but rather the rules of the interpreter (or, similarly, the compiler) which will translate that code into something the computer understands.

Figure 1-3 shows a simplified diagram showing the code-writing process:

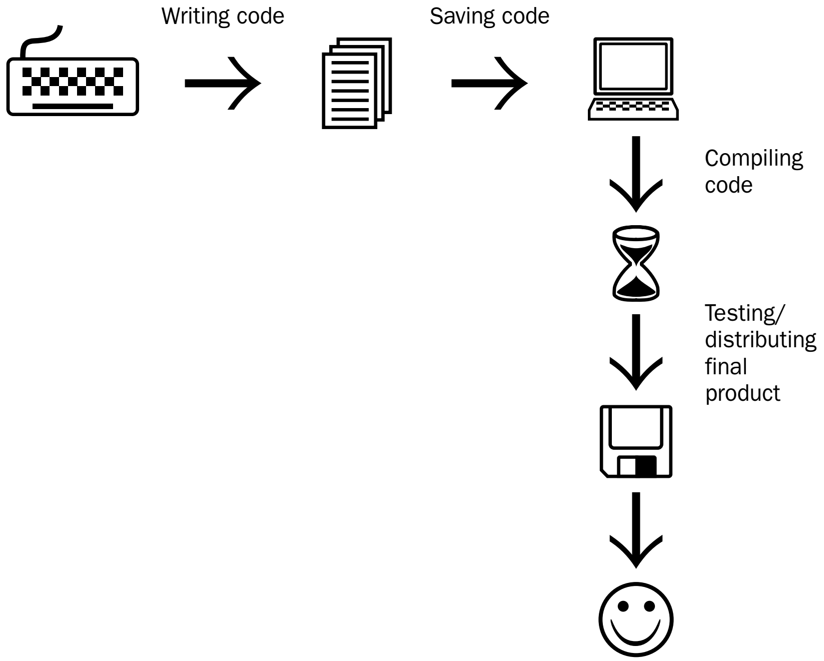


Figure 1-3

Different Code, Same Results

Just to show you how different code can achieve the same results, take a look at the following code in different languages, some of which you may have heard of.

BASIC

```
10 print "Hello World!"
20 goto 10
```

Atari BASIC

```
10 REM HELLO.BAS
20 POKE 764,255
30 PRINT "Hello World"
40 IF PEEK(764)=255 THEN GOTO 30
```

C

```
#include <stdio.h>

main()
{
for(;;)
{
printf ("Hello World!\n");
}
}
```

C++

Old version of C++ code:

```
#include <iostream.h>

main()
{
for(;;)
{
cout << "Hello World! ";
}}
```

Newer version of the code:

```
#include <iostream>

int main()
{
std::cout << "Hello, World!\n";
}
```

COBOL

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLOWORLD.
000300 DATE-WRITTEN. 02/09/04 17:24.
000400* AUTHOR FRED F
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400 DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500 DISPLAY "HELLO, WORLD." LINE 15 POSITION 10.
100600 STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800 EXIT.
```

FORTRAN

```
c
c Hello, world.
c
Program Hello

implicit none
logical DONE

DO while (.NOT. DONE)
write(*,10)
END DO
10 format('Hello, World.')
END
```

Java

```
class HelloWorld {
public static void main (String args[]) {
for (;;) {
System.out.print("Hello World ");
}
}
}
```

JavaScript

```
<title>
Hello World in JavaScript
</title>
<script>
alert("Hello, World!")
</script>
```

Mathematica

```
While[True, Print["Hello, World!"]]
```

Pascal

```
Program Hello (Input, Output);

Begin
Writeln ('Hello World!');
End.
```

Perl

```
print "Hello, World!\n" while (1);
```

Python

```
while (1) :
print "Hello World";
```

QBASIC

```
begin:
print "Hello World!"
goto begin
```

Smalltalk

```
Transcript show:'Hello World';cr
```

Visual Basic

```
Private Sub Form_Load()
Static I
I = 1
for I = 1 to 10
msgbox "Hello, World!"
Next I
end sub
```

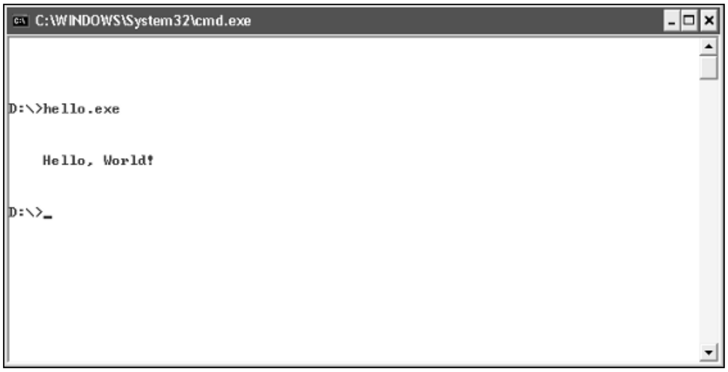
VRML

```
#VRML V1.0 ascii

AsciiText {
string "Hello, World!"
justification LEFT
}
```

All of this code written in a variety of languages does one thing: it shows the words Hello, World! on the screen in one form or another.

Figure 1-4 shows an example of what it looks like in C++:



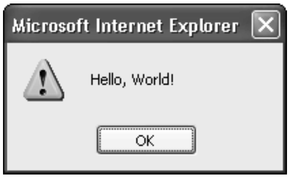
```
D:\>hello.exe

   Hello, World!

D:\>_
```

Figure 1-4

Figure 1-5 shows what is displayed by JavaScript:



Hello, World!

Figure 1-5

Why "Hello, World!"?

Why is it that "Hello, World!" is used in so many programming examples? The origin and history behind this statement is heavily laden with urban legend and myth. However, it does seem that the first tutorial that contained a reference to Hello, World! was for the B-language that predates C.

It seems that before that it was used as a simple way to determine that everything was set up right and the language worked. So, who am I to break with tradition?!

Programs Needed to Create Programs

So, can you create programs from scratch? Well, you could, and the people who trail-blazed before you needed to do this, but after creating the first languages one of the first things that computer scientists started creating were programs to make writing other programs easier. Once this happened it became easier and quicker to write code in a particular language. Soon one language was used to create tools to work with a completely different (or new) language. This enabled newer languages to have greater complexity and sophistication, which means that they end up being able to do more.

There are two kinds of program that help you write code and make programs:

- A development environment
- A compiler

The interpreter mentioned before is not needed to write code, only to run it.

Development Environment

A development environment is a program that you use to type the code into. Some languages need a specific development environment (for example, Visual Basic), while others need only a simple text editor (JavaScript is an example of a language you can work with using nothing more than a text editor). There is also a huge array of languages where you have a choice as to what development environment to use (C++ is one such language).

Development environments are discussed in more detail later, for now, just knowing that they exist is enough!

Compilers

A compiler is a program that changes the code you type into code that the computer can understand. This process is known as *compilation*.

The compiler reads the code that you type, checking it for errors and making sure that it follows all the rules and makes sense. If problems with your code are found, the compiler should (if it's a good compiler) tell you about it and halt the process. If no problems are found, the compiler outputs a file that you can run (or execute) on its own, and it should carry out the instructions as laid out in your code. On a PC this file is typically called an executable file and has a file extension *.exe,* as shown in Figure 1-6.

Figure 1-6

Summary

In this chapter, we've looked at what programming really is. Now I'm not expecting that after this chapter you should have any idea how to write a program, but you should have gotten a sense (albeit brief!) of the following:

- ❑ The history of programming
- ❑ An introduction to some programming languages
- ❑ A sneak peek at some code
- ❑ An overview of the programming process, from code to executable
- ❑ An exposure to some important programming terminology

In the next chapter, we'll look at why you *really do* need to learn how to program!