Programming with C

Terry Marris November 2010

1 Hello World

Perhaps the only way to learn programming is to write programs, dozens of them. In doing so you not only learn the peculiarities of the language, but you also learn how to solve problems. Being stuck is a programmer's way of life. Strategies for getting unstuck include:

- ask an expert
- look for a similar problem that has already been solved, adapt the solution to the current problem
- attack the problem from a different angle
- have a break, sleep on the problem then come back to it with a fresh mind
- use concrete examples to aid thinking. Think on paper.
- be prepared to throw away efforts that do not work: you have learned how not to do it
- have alongside you, and refer to, textbooks on the language.

The C Programming Language by Brian Kernighan and Dennis Ritchie is the kind of book you grow into. A Lexical Guide to ANSI C by the Mark Williams Company describes each component of the standard C library. Too many times I have seen students staring at a blank screen, with no supporting material to hand, expecting a program to magically appear out of thin air. It just does not happen.

The C programming language was initially designed for expert programmers implementing the Unix operating system, and it does have it peculiarities. But it has also been used for teaching students programming since the 1980's. There is nothing to drag and drop, there are no hints that appear automatically as you type, no complex and confusing program development environments, just pure programming. Students who learn how to program in C find it relatively easy to transfer their skills to other programming languages. C is a small language, not much to learn. Some of it is strange to the beginner. But with practice comes familiarity. What seems strange to begin with becomes normal with practice. You begin to think in the language.

You could work sequentially through the chapters, attempting every exercise. Some chapters are easy - you whiz through them. Others are a little trickier, and they slow you down a bit. This is normal and to be expected. Answers to most exercises are given, you can learn from them. How to install the C compiler system and how to configure and use a program development environment are covered in the appendices.

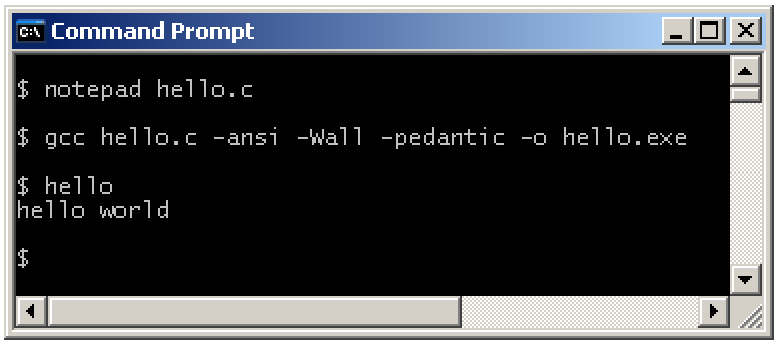
1.1 Hello World

The first program in any language displays *hello world* on the screen.

```
/* hello.c: displays a greeting */

#include <stdio.h>

int main()
{
  printf("hello world\n");
  return 0;
}
```



```
$ notepad hello.c

$ gcc hello.c -ansi -Wall -pedantic -o hello.exe

$ hello
hello world

$
```

We go through the program line-by-line.

1.2 Comments

The line

```
/* hello.c: displays a greeting */
```

is an example of a *comment*. Comments are written for the benefit of the person reading your program. That person could be yourself, your colleague or your supervisor. Here the comment announces the name of the program and what it does.

Comments always begin with /* and end with */. You must get the oblique stroke the right way round and the asterisk in the right place - the asterisks are always innermost. And comments cannot be nested - you cannot have comments inside comments.

You can spread comments over several lines like this:

```
/* hello.c: displays a greeting
   Priti Manek 1 Nov 2010 */
```

Including your name in a comment at the top of a program is especially useful if several people share the same printer.

1.3 The Standard Library

A function usually performs a useful task. C comes with a collection of functions organised into libraries. The standard input/output library, named *stdio.h*, contains functions for getting data from the keyboard and for writing data onto the screen.

Our program uses a function named *printf()* to display text on the screen. We are obliged to include *stdio.h* in our program since it contains the definition of *printf()*.

Other functions will be introduced as required, as will other libraries.

The C compiler is a program that translates your program text into a form that can be executed, or run, by the computer's operating system. Operating systems include Microsoft Windows and Linux.

The line

```
#include <stdio.h>
```

instructs the compiler to include the contents of the file named *stdio.h* in the translation process. The angle brackets indicate that the file is located in a standard directory or folder for the C library, which is set up when the C compiler software is installed.

1.4 The main() Function

```
int main()
{
  ...
  return 0;
}
```

Every executable C program, every C program that runs, has just one function named *main()*. Program execution always starts with the *main()* function.

int stands for integer, a whole number. Our *main()* function returns zero, an integer. Here, the *int* is the function's return type, and zero is its return value. We shall have more to say about types and functions later.

Notice the braces, *{* and *}*. They always occur in pairs, and are used to group a collection of program statements.

```
{
  printf("hello world\n");
  return 0;
}
```

Each statement, an instruction to perform some task, is terminated with a semi-colon.

1.5 printf()

```
  printf("hello world\n");
```

displays *hello world* on the screen. Here, *printf()* is an example of a *function call*. It says: *hey, printf(), go and do your job*.

The text between quotation marks is an example of a *string constant* or a *string literal*. The text is displayed literally as you see it. The quotation marks themselves are not displayed.

\n is the newline character. Its effect is to move the print head to the beginning of the next line down on the screen. *printf()* does not supply a new line unless you include \n. So, for the same result you could write

```
      printf("hello ");
      printf("world\n");
```

If you want several lines of text on the screen you might write, for example,

```
      ...
      printf("I was born in a cross-fire hurricane\n");
      printf("I cried at my ma in the driving rain\n");
      printf("But it's all right now, in fact it's a gas\n");
      printf("But it's all right. I'm Jumping Jack Flash\n");
      printf("It's a Gas! Gas! Gas!\n");
      printf("Jagger/Richards/Wyman 1968\n");
      ...
```

printf("\n"); will write a blank line, if you need one.

1.6 Escape Sequences

\n is an example of an *escape sequence*. An escape sequence is several characters that represent just one character. So, for example, \n represents the newline character. Escape sequences include:

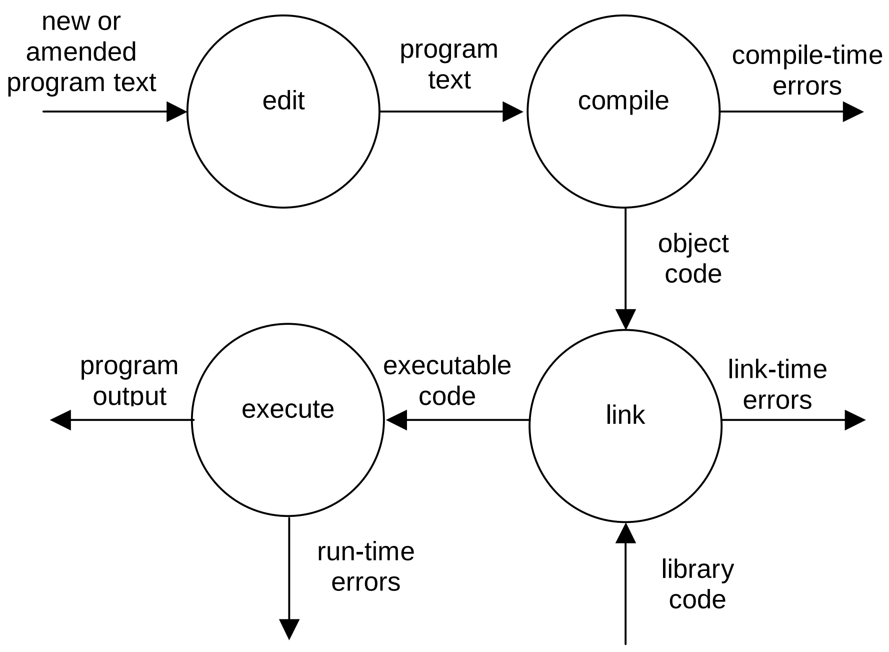
\n	newline
\t	tab
\\"	quotation marks
\\\\	backslash

So, if you want to print quotation marks themselves you might write

```
      ...
      printf("She said: \"Jo King\" \n");
      ...
```

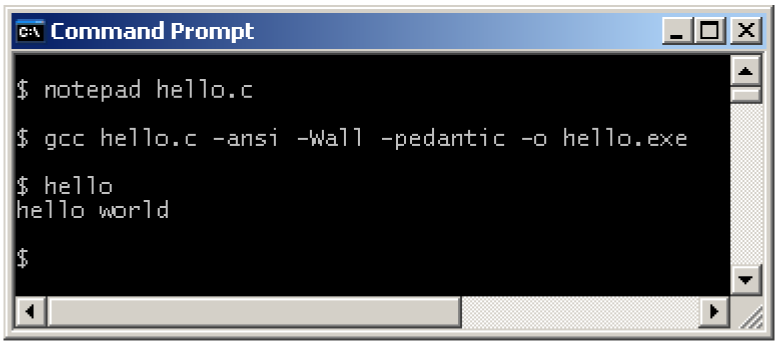
1.7 Program Development Environment

An *editor* allows you to create, amend and save program text. A *compiler* translates your program text into object code and reports compile-time errors. A compile-time error will occur if, for example, you missed a semi-colon or a quotation mark out. A *linker* combines your program object code with other code from libraries, if it can find them, to create an executable program. An executable program is one that runs.



If errors were reported, you would return to the editor to correct your program text.

The program development environment used here is the Microsoft Windows MSDOS window, reached from *Start*, *Program Accessories*, *Command Prompt*.



```
$ notepad hello.c

$ gcc hello.c -ansi -Wall -pedantic -o hello.exe

$ hello
hello world

$
```

$ is my system command prompt. *notepad* is the program editor. *gcc* loads the GNU C compiler. *hello.c* is the filename of the program to be compiled. *-ansi* instructs the compiler to check for compliance with the ANSI C standard. *-Wall* and *-pedantic* instructs the compiler to report warnings. C was originally designed for professional programmers who were expected to know what they were doing. Us amateurs need all the help we can get. *-o* is for output from the compiler. *hello.exe* is the executable file to be output. *hello* typed at the command prompt loads and runs the program. And *hello world* is the result of running the program.

The program development environment is discussed in more detail in the appendix on the GNU C Compiler.

In other programming environments you might need to insert

```
printf("Press return to continue ... ");
getchar();
```

at the bottom of *main()* but before the *return* statement. The purpose of these two lines is to hold program output on the screen until you press the return key. You will need to include these two lines if your program run flashes on the screen and then disappears. Here is an entire program and its run.

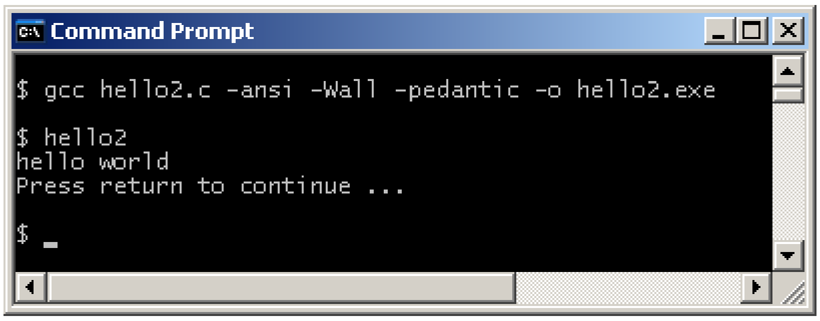
```
/* hello2.c: displays a greeting */

#include <stdio.h>

int main()
{
  printf("hello world\n");

  printf("Press return to continue ... ");
  getchar();

  return 0;
}
```



```
$ gcc hello2.c -ansi -Wall -pedantic -o hello2.exe

$ hello2
hello world
Press return to continue ...

$ _
```

1.8 Layout

Programmers take pains to get the layout consistently neat and tidy. Why? Because it helps them to write error-free programs, and its helps them, and others, to understand what has been written.

```
/* hello.c: displays a greeting */

#include <stdio.h>

int main()
{
  printf("hello world\n");
  return 0;
}
```

Here, we have:

- left a blank line before and after the *#include* statement
- indented the block of statements between the braces by two spaces
- lined up the statements within the block one directly below the other
- lined up the matching pair of braces, *{* and *}*, one brace directly under the other

OK. The program is a small one and easy to understand. But good habits started now make it easier when programs become large and complex.

Exercise 1.1

Perhaps the only way to learn how to program is to write programs.

1. Try out the program, shown above, that displays *hello world* on the screen.

2. Investigate the effect of leaving out parts of the program written in exercise 1 above. Remember to save your amended program before compiling and running it.

3. Write and run a program that displays several lines of the song or poem of your choice. Remember to acknowledge the source of your song or poem.

4. Write and run a program that demonstrates the effect of using the escape sequences shown above.

We have seen how to display a string constant. **Next** we begin looking at string variables.

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988 pp 1..8
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988