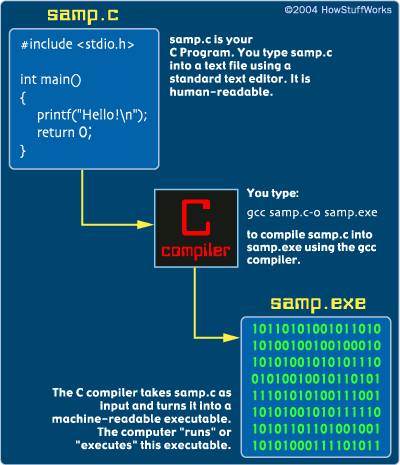
**The Basics of C Programming**

**by Marshall Brain & Chris Pollette**

(Available at <<<https://computer.howstuffworks.com/c1.htm>>>)

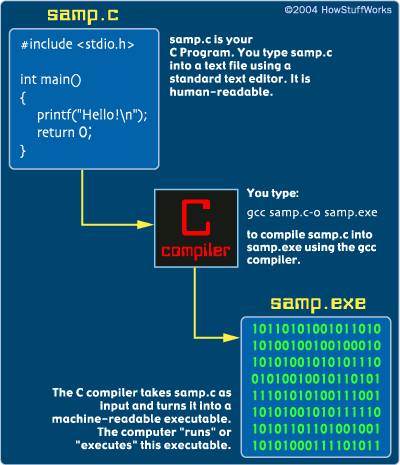
**What is C?**

C is a **computer programming language**. That means that you can use C to create lists of instructions for a computer to follow. C is one of thousands of programming languages currently in use. C has been around for several decades and has won widespread acceptance because it gives programmers maximum control and efficiency. C is an easy language to learn. It is a bit more cryptic in its style than some other languages, but you get beyond that fairly quickly.

C is what is called a **compiled language**. This means that once you write your C program, you must run it through a **C compiler** to turn your program into an **executable** that the computer can run (execute). The C program is the human-readable form, while the executable that comes out of the compiler is the machine-readable and executable form. What this means is that to write and run a C program, you must have access to a C compiler. If you are using a UNIX machine (for example, if you are writing CGI scripts in C on your host's UNIX computer, or if you are a student working on a lab's UNIX machine), the C compiler is available for free. It is called either "cc" or "gcc" and is available on the command line. If you are a student, then the school will likely provide you with a compiler -- find out what the school is using and learn about it. If you are working at home on a Windows machine, you are going to need to download a free C compiler or purchase a commercial compiler. A widely used commercial compiler is Microsoft's Visual C++ environment (it compiles both C and C++ programs). Unfortunately, this program costs several hundred dollars. If you do not have hundreds of dollars to spend on a commercial compiler, then you can use one of the free compilers available on the Web. See <http://delorie.com/djgpp/> as a starting point in your search.

We will start at the beginning with an extremely simple C program and build up from there. I will assume that you are using the UNIX command line and gcc as your environment for these examples; if you are not, all of the code will still work fine -- you will simply need to understand and use whatever compiler you have available.

Let's get started!

**The Simplest C Program**

Let's start with the simplest possible C program and use it both to understand the basics of C and the C compilation process. Type the following program into a standard text editor (vi or emacs on UNIX, Notepad on Windows or TeachText on a Macintosh). Then save the program to a file named **samp.c**. If you leave off **.c**, you will probably get some sort of error when you compile it, so make sure you remember the **.c**. Also, make sure that your editor does not automatically append some extra characters (such as .txt) to the name of the file. Here's the first program:

#include <stdio.h>

int main()

{

printf("This is output from my first program!\n");

return 0;

}

**A computer program is the key to the digital city: If you know the language, you can get a computer to do almost anything you want. Learn how to write computer programs in C.**

When executed, this program instructs the computer to print out the line "This is output from my first program!" -- then the program quits. You can't get much simpler than that!

To compile this code, take the following steps:

* On a UNIX machine, type **gcc samp.c -o samp** (if gcc does not work, try cc). This line invokes the C compiler called gcc, asks it to compile samp.c and asks it to place the executable file it creates under the name **samp**. To run the program, type **samp** (or, on some UNIX machines, **./samp**).
* On a DOS or Windows machine using [DJGPP](http://www.delorie.com/djgpp/), at an MS-DOS prompt type **gcc samp.c -o samp.exe**. This line invokes the C compiler called gcc, asks it to compile samp.c and asks it to place the executable file it creates under the name **samp.exe**. To run the program, type **samp**.
* If you are working with some other compiler or development system, read and follow the directions for the compiler you are using to compile and execute the program.

You should see the output "This is output from my first program!" when you run the program. Here is what happened when you compiled the program:

If you mistype the program, it either will not compile or it will not run. If the program does not compile or does not run correctly, edit it again and see where you went wrong in your typing. Fix the error and try again.

POSITION

When you enter this program, position **#include** so that the pound sign is in column 1 (the far left side). Otherwise, the spacing and indentation can be any way you like it. On some UNIX systems, you will find a program called **cb**, the C Beautifier, which will format code for you. The spacing and indentation shown above is a good example to follow.

**The Simplest C Program: What's Happening?**

Let's walk through this program and start to see what the different lines are doing ([Click here](https://computer.howstuffworks.com/c2.htm) to open the program in another window):

* This C program starts with **#include <stdio.h>**. This line **includes** the "standard I/O library" into your program. The standard I/O library lets you read input from the [keyboard](https://computer.howstuffworks.com/keyboard.htm) (called "standard in"), write output to the [screen](https://computer.howstuffworks.com/monitor.htm) (called "standard out"), process text files stored on the [disk](https://computer.howstuffworks.com/hard-disk.htm), and so on. It is an extremely useful library. C has a large number of standard libraries like stdio, including string, time and math libraries. A **library** is simply a package of code that someone else has written to make your life easier (we'll discuss libraries a bit later).
* The line **int main()** declares the main function. Every C program must have a function named **main** somewhere in the code. We will learn more about functions shortly. At run time, program execution starts at the first line of the main function.
* In C, the **{** and **}** symbols mark the beginning and end of a block of code. In this case, the block of code making up the main function contains two lines.
* The **printf** statement in C allows you to send output to standard out (for us, the screen). The portion in quotes is called the **format string** and describes how the data is to be formatted when printed. The format string can contain string literals such as "This is output from my first program!," symbols for carriage returns (\n), and operators as placeholders for variables (see below). If you are using UNIX, you can type **man 3 printf** to get complete documentation for the printf function. If not, see the documentation included with your compiler for details about the printf function.
* The **return 0;** line causes the function to return an error code of 0 (no error) to the shell that started execution. More on this capability a bit later.

## Variables

As a programmer, you will frequently want your program to "remember" a value. For example, if your program requests a value from the user, or if it calculates a value, you will want to remember it somewhere so you can use it later. The way your program remembers things is by using **variables**. For example:

int b;

This line says, "I want to create a space called b that is able to hold one integer value." A variable has a **name** (in this case, b) and a **type** (in this case, int, an integer). You can store a value in b by saying something like:

b = 5;

You can use the value in b by saying something like:

printf("%d", b);

In C, there are several standard types for variables:

* **int** - integer (whole number) values
* **float** - floating point values
* **char** - single character values (such as "m" or "Z")

We will see examples of these other types as we go along.

## Printf

The **printf statement allows you to send output to standard out.** For us, standard out is generally the screen (although you can redirect standard out into a text file or another command).

Here is another program that will help you learn more about printf:

#include <stdio.h>

int main()

{

int a, b, c;

a = 5;

b = 7;

c = a + b;

printf("%d + %d = %d\n", a, b, c);

return 0;

}

Type this program into a file and save it as **add.c**. Compile it with the line **gcc add.c -o add** and then run it by typing **add** (or **./add**). You will see the line "5 + 7 = 12" as output.

Here is an explanation of the different lines in this program:

* The line **int a, b, c;** declares three integer variables named **a**, **b** and **c**. Integer variables hold whole numbers.
* The next line initializes the variable named **a** to the value 5.
* The next line sets **b** to 7.
* The next line adds **a** and **b** and "assigns" the result to **c**. The computer adds the value in **a** (5) to the value in **b** (7) to form the result 12, and then places that new value (12) into the variable **c**. The variable **c** is assigned the value 12. For this reason, the = in this line is called "the assignment operator."
* The **printf** statement then prints the line "5 + 7 = 12." The **%d** placeholders in the printf statement act as placeholders for values. There are three %d placeholders, and at the end of the printf line there are the three variable names: **a**, **b** and **c**. C matches up the first %d with a and substitutes 5 there. It matches the second %d with b and substitutes 7. It matches the third %d with c and substitutes 12. Then it prints the completed line to the screen: 5 + 7 = 12. The **+**, the **=** and the spacing are a part of the format line and get embedded automatically between the %d operators as specified by the programmer.

## Printf: Reading User Values

The previous program is good, but it would be better if it read in the values 5 and 7 from the user instead of using constants. Try this program instead:

#include <stdio.h>

int main()

{

int a, b, c;

printf("Enter the first value:");

scanf("%d", &a);

printf("Enter the second value:");

scanf("%d", &b);

c = a + b;

printf("%d + %d = %d\n", a, b, c);

return 0;

}

Here's how this program works when you execute it:

<https://computer.howstuffworks.com/c6.htm>

Make the changes, then compile and run the program to make sure it works. Note that scanf uses the same sort of format string as printf (type **man scanf** for more info). Also note the & in front of a and b. This is the **address operator** in C: It returns the address of the variable (this will not make sense until we discuss pointers). You must use the & operator in scanf on any variable of type char, int, or float, as well as structure types (which we will get to shortly). If you leave out the & operator, you will get an error when you run the program. Try it so that you can see what that sort of run-time error looks like.

Let's look at some variations to understand printf completely. Here is the simplest printf statement:

printf("Hello");

This call to printf has a format string that tells printf to send the word "Hello" to standard out. Contrast it with this:

printf("Hello\n");

The difference between the two is that the second version sends the word "Hello" followed by a carriage return to standard out.

The following line shows how to **output the value of a variable using printf**.

printf("%d", b);

The **%d** is a placeholder that will be replaced by the value of the variable **b** when the printf statement is executed. Often, you will want to embed the value within some other words. One way to accomplish that is like this:

printf("The temperature is ");

printf("%d", b);

printf(" degrees\n");

An easier way is to say this:

printf("The temperature is %d degrees\n", b);

You can also use multiple %d placeholders in one printf statement:

printf("%d + %d = %d\n", a, b, c);

In the printf statement, it is extremely important that the number of **operators** in the format string corresponds exactly with the number and type of the variables following it. For example, if the format string contains three %d operators, then it must be followed by exactly three parameters and they must have the same types in the same order as those specified by the operators.

You can **print all of the normal C types with printf** by using different placeholders:

* **int** (integer values) uses **%d**
* **float** (floating point values) uses **%f**
* **char** (single character values) uses **%c**
* **character strings** (arrays of characters, discussed later) use **%s**

You can learn more about the nuances of printf on a UNIX machine by typing **man 3 printf**. Any other C compiler you are using will probably come with a manual or a help file that contains a description of printf.

**C ERRORS TO AVOID**

* Using the wrong character case - Case matters in C, so you cannot type Printf or PRINTF. It must be printf.
* Forgetting to use the & in scanf
* Too many or too few parameters following the format statement in printf or scanf
* Forgetting to declare a variable name before using it;
* […]

(If you got interested and would like to learn more about C language, just access the link at the beginning of the text.)