Introduction to Computational Physics

Meeting 1: Zero to Loops

Today:
- Simple C Programs
- Variables
- Loops
- Putting it all together....

Welcome!

In this short-course we'll try to get you up and programming! I hope you'll learn at least four important skills:

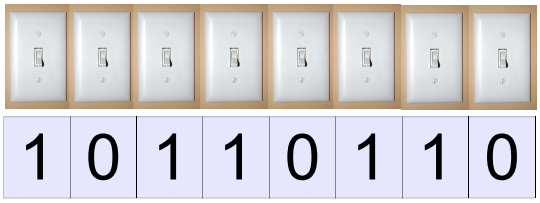
* How to connect to a remote computer,
* create programs there,
* run them,
* and view the results.

We'll be using the C programming language under the Linux operating system. The goal is for you to understand the mechanics and concepts of scientific programming.

Bits and Bytes:

The data in your computer is all stored in bunches of microscopic switches. Each switch can only have two values, “1” or “0” (“on” or “off”). The amount of information stored by one switch is called a “bit”, and we often talk about flipping bits on or off.

These bits are usually grouped together in sets of eight. A group of eight bits is called a “byte”.



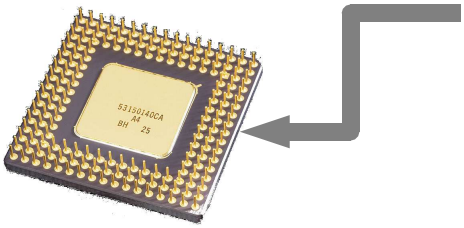
Why eight bits? First, because eight is a power of two (2^3), making it convenient for binary (base-2) arithmetic. (Just as 10, 100 or 1000 are convenient in base-10.) Second, because the very popular early Intel CPUs used data in 8-bit chunks.

Some people claim that “bit” is a shortened form of “binary digit”, but I'm skeptical.

How a Program Runs:

When running a program, a computer's CPU fetches instructions from the computer's memory.

CPU



Memory

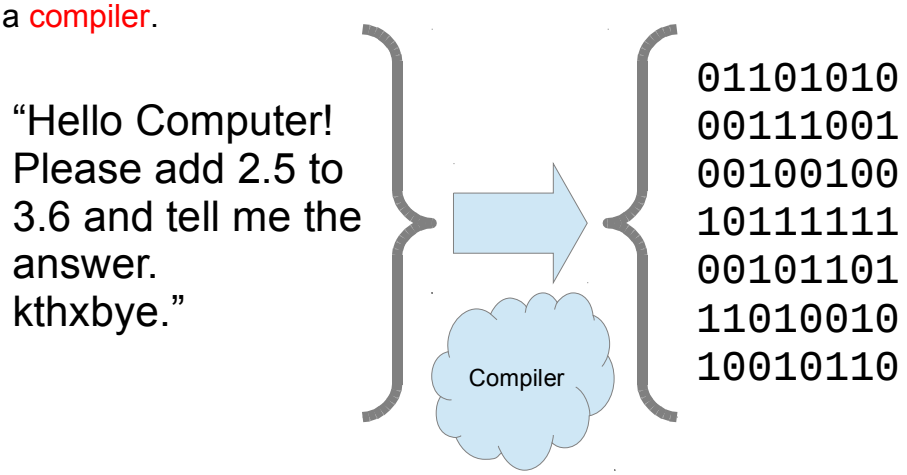
```
01101010
00111001
00100100
10111111
00101101
11010010
10010110
```

The memory also contains data. The instructions tell the CPU to add, compare or do other simple operations on the data.

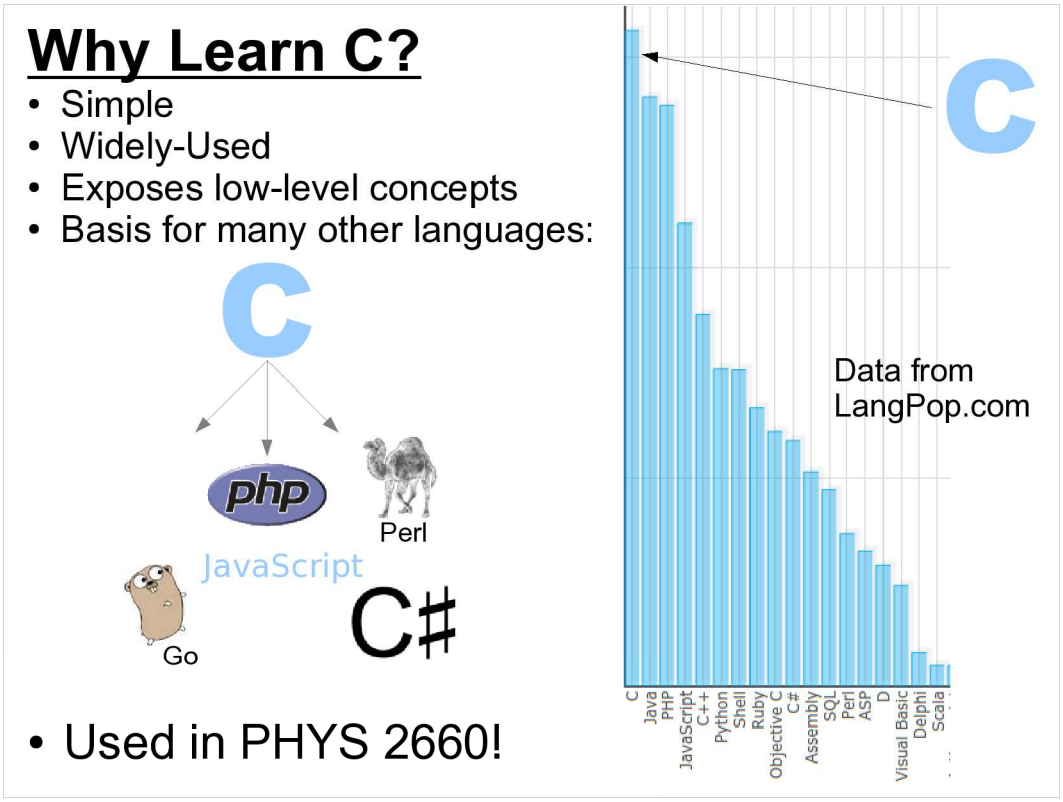
Of course, different CPUs use different instruction sets.

Programming Languages:

We usually write programs in human-readable languages like C, but CPUs don't understand C. The programs we write must be translated into a series of binary low-level instructions that the CPU can understand. This translation is done by a program called a compiler.



There are hundreds of different computer languages. Each has its own strengths and weaknesses, and no language is best for all tasks. When choosing a language for a particular project, programmers think about whether the language's strengths are a good match for that project.



C has been around for 40 years or so. It achieved its popularity for many reasons:

C is simple in that there aren't many words in the C language. The core functionality of C can be extended by writing “functions” (written in C) that can be stored in libraries and re-used. For example, the C language doesn't have a square-root command, but there's a library of math functions that contains a square-root function.

To implement a C compiler on a new kind of computer, a programmer just needs to write a compiler that can understand the small, core part of C. Then he or she can use this to compile all of the libraries containing math functions and other things.

This was very important back in the days when almost every computer manufacturer used a different CPU with a different set of instructions.

Don't be Afraid of the Computer!

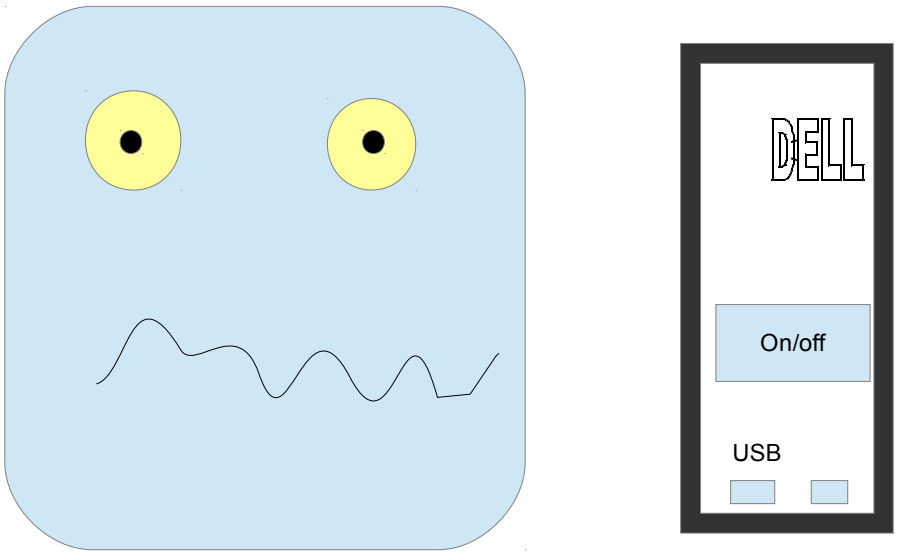
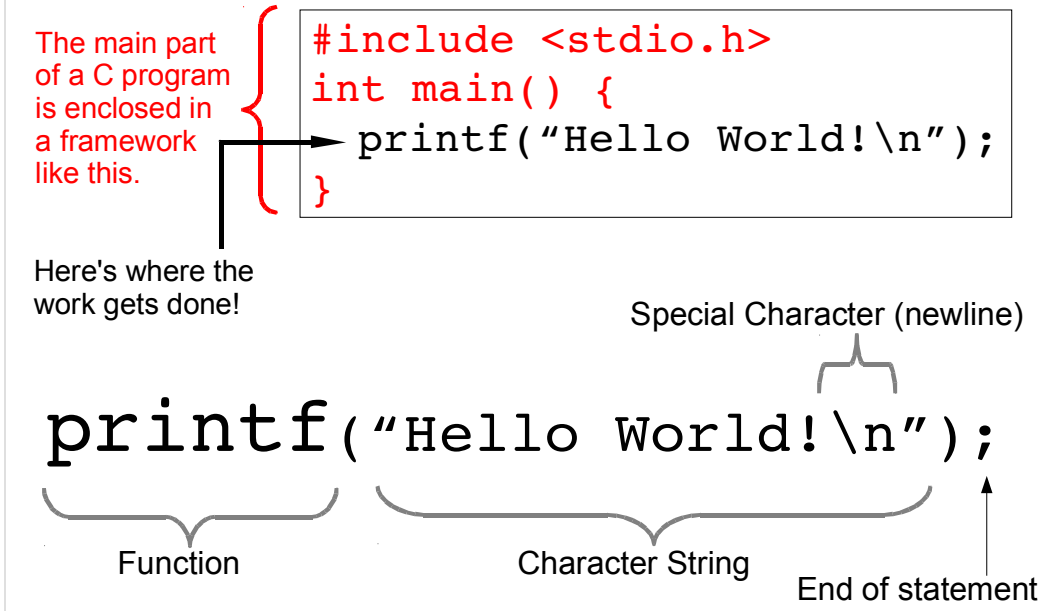


Image Credit: Finian Wright.

This is important advice! Don't be daunted just because something doesn't make sense at first. Ask questions. Try things out. And remember that other people are probably confused, too. This short-course is intended to help you learn stuff!

A Simple C Program:

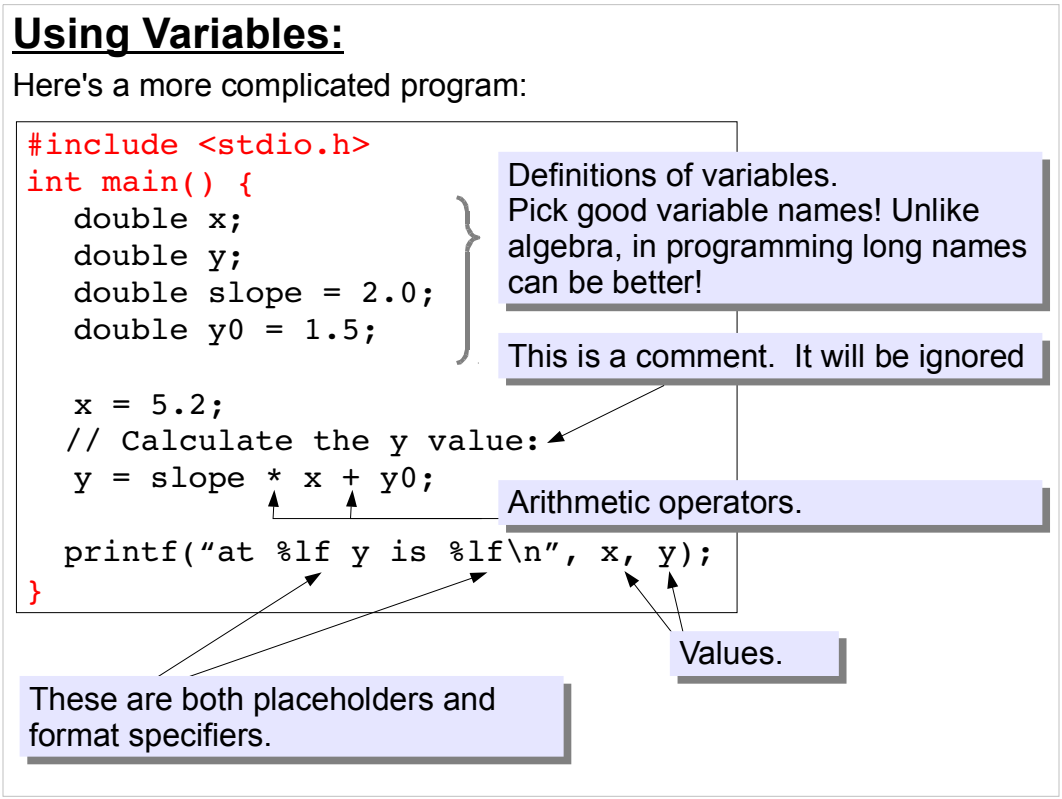
Here's a C program that prints “Hello World!”



This is about the simplest C program you can write. If you search on Wikipedia, you'll find a long list of “Hello World” programs written in many different languages. Some of them are truly bizarre.

Using Variables:

Here's a more complicated program:



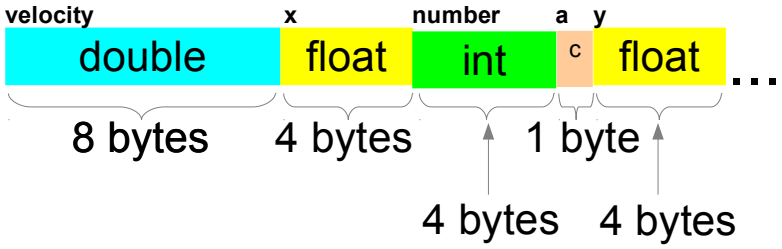
The “double” just means that these variables are “double-precision floating-point numbers”. This tells C how much space in memory it needs to reserve for storing them. As we'll see, most of the time your variables should be either “double” or “int”. Use “double” for anything that might have a decimal point, and “int” for integers.

In the “printf” statement above, “%lf” means “save a space for a 'double' value here”. The letters “lf” stand for “long float”, which is another way of referring to “double” variables. Did I mention that C has a 40-year history?

Notice that “printf” is a function, just like functions in algebra. Within the parentheses following it you find a list of arguments. In this case, a format string and two numbers.

Storing Variables:

When your program runs, it sets up an area in the computer's memory for storing the value of each of your variables:



Different types of variables are given different amounts of space. Bad things can happen if you try to stick the wrong type of data into a variable.

What would happen if you tried to stick a “double” value the variable named “x”, above?

For storing numbers, reach for “int” or “double” first, but there are lots of other types.

Answer: If you succeeded, the data would spill over into the adjoining variable (“number”) and corrupt it.

The C compiler tries to prevent this sort of thing two ways:

- It warns you when try to stick the wrong type of data into a variable, and
- It tries, when reasonable, to re-cast your data into a format that's appropriate for the variable into which you're putting it.

What's “Scientific Computing”?

The most powerful computers in the world aren't laptop or desktop computers. They're huge computing clusters that occupy whole rooms. Usually these clusters are located far away from the researchers who use them.

Researchers use local computers (perhaps a laptop) to communicate with the remote computing cluster. They connect to it through the Internet.

Once connected, the researcher can create and run programs on the cluster. Complicated programs might run for hours or days. After researchers have started a program running, they can disconnect from the cluster and come back again later to check on the progress of their program.

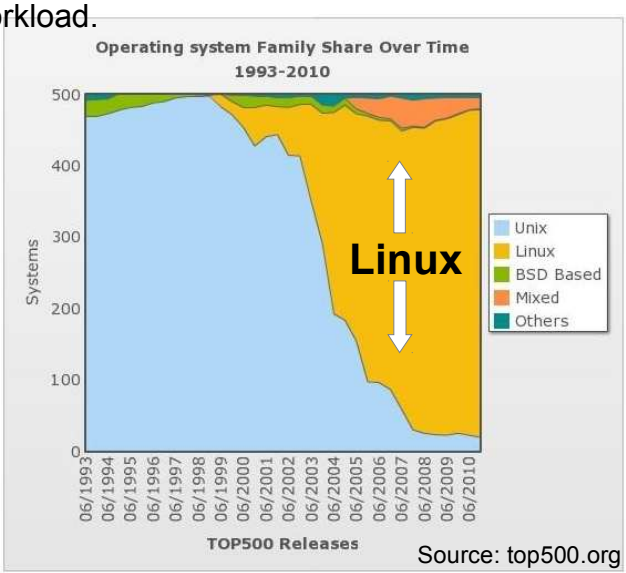
Scientific researchers push the limits of computing. The world's most powerful computers are used for Physics, Astronomy, Environmental Science and many other areas of scientific research.

If you pursue a career in computationally-intensive research, you'll need to learn how to use these remote computers.

Why Linux?:

Linux-based computer systems are a mainstay in the world of scientific computing. In any laboratory setting where the research requires large amounts of data processing or computationally intensive calculations, you will routinely find a Linux/Unix cluster of computers handling the workload.

As of Nov 2010 92% (96%, counting Unix) of the world's top performing computer systems operate on Linux. Linux is the overwhelming choice for building world-wide high performance computing Grids.



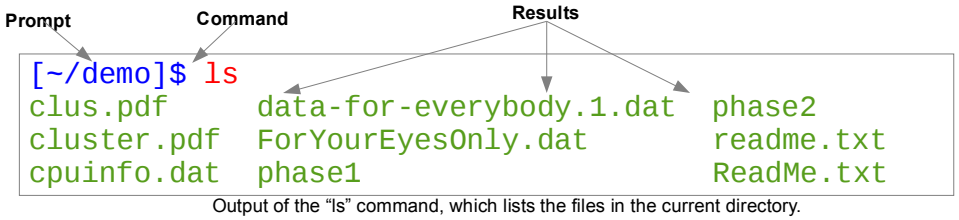
(Twice a year top500.org posts benchmark measurements for the top 500 fastest computers on earth.)

The chart above is out of date. As of September 2014, 97% of the top 500 computers are now running Linux.

The desktop environment on our lab computers will probably look similar to the desktop of other computers you've used. At the bottom left, there's a button that will pop up a menu of programs, and along the bottom of the screen there are a few icons for commonly-used programs, like the terminal emulator we'll be using.

Try clicking it to open a terminal window.

The Command Line:



Output of the "ls" command, which lists the files in the current directory.

Why do we do things from the command line?:

* Text commands are easily reproduced. It's easy to document what you've done, or to tell someone else exactly how to do it, or to automate what you've done. Being able to exactly replicate your procedures is especially important when analyzing scientific data.

* Text commands don't require much bandwidth. They can be used even over slow network connections.

* In Linux, graphical tools might provide a front-end to help you do tasks, but you can do more from the command line.

The answer is that the command line has its own advantage. Here are some of the things that might make someone choose to use the command line under Linux.

The first item is the most important, I think. This is true for all operating systems, not just Linux, and it's why all major operating systems still have a command-line interface and continue to improve it.

Case Sensitivity:

Important Note: when typing commands, file names, etc...

Linux and C/C++ are

CaSe SeNsItIvE

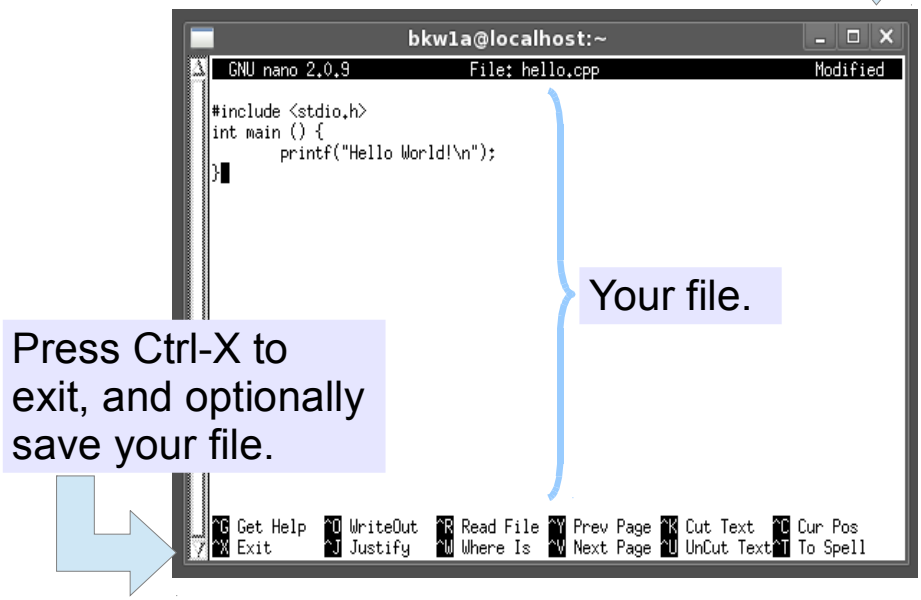
So, for example:

This is not the same as this,
VELOCITY is not the same as Velocity or velocity,
MyFile.dat is not the same as MyFile.DAT

For best results, stick to all lower-case unless there's a good reason to do otherwise.

Using the “nano” Text Editor:

```
[~/demo]$ nano hello.cpp
```



There are many text editors, but I recommend you start out with a simple editor called “nano”. To use nano to edit a file, just type “nano” followed by the file name. Once in nano, you can type normally, or you can use special control keystrokes to do things like saving your file or searching. These are listed at the bottom of the window.

The list at the bottom uses “^” to mean “hold down the Ctrl key”, so “^X” means “press Ctrl-X”.

Compiling and Running Your Program:

```
#include <stdio.h>
int main() {
  printf("Hello World\n");
}
```

Let's say you've saved the C code for our "Hello World" program in a file called "hello.cpp". To compile this into an executable binary file that you can run, you could type:

```
[~/demo]$ g++ -Wall -o hello hello.cpp
```

The qualifier "-o hello" tells the compiler that we want the output executable file to be named "hello". The "-Wall" tells the compiler to warn you about any possible problems it notices. We can run this executable file by just typing "./" followed by its name. Here's what will happen:

```
[~/demo]$ ./hello
Hello World!
```

Congratulations! You're a programmer.

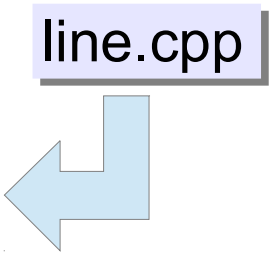
Writing a Second Program:

Now let's try our slope/y-intercept program. Call it "line.cpp":

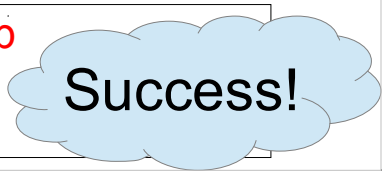
```
#include <stdio.h>
int main() {
   double x;
   double y;
   double slope = 2.0;
   double y0 = 1.5;

   x = 5.2;
  // Calculate the y value:
   y = slope * x + y0;

  printf("at %lf y is %lf\n", x, y);
}
```




```
[~/demo]$ g++ -Wall -o line line.cpp
[~/demo]$ ./line
at 5.200000 y is 11.900000
```

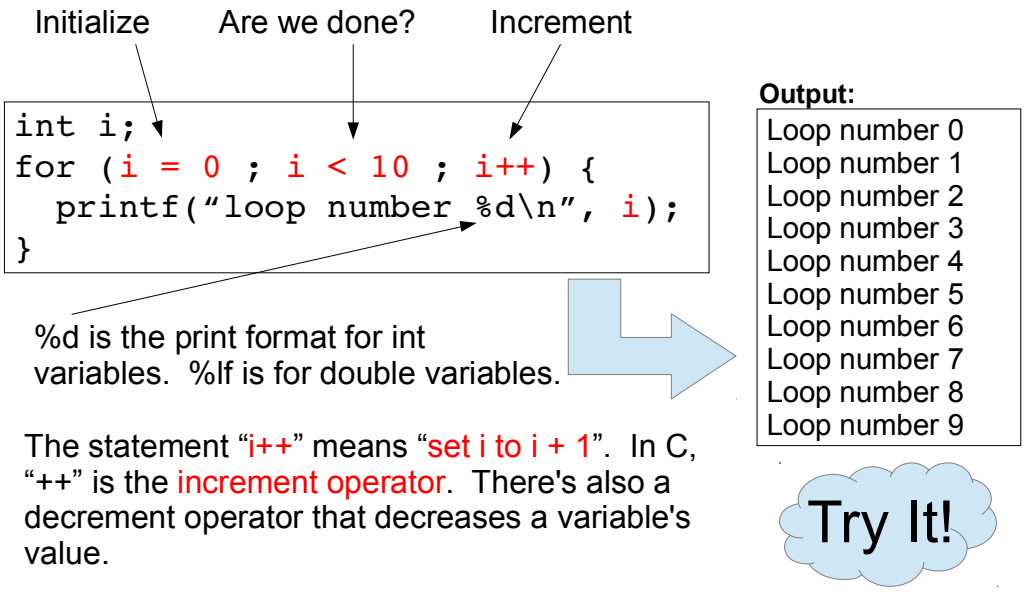


As with the "hello.cpp" program, you can start up the editor by typing "nano line.cpp". When you're done typing your program, press Ctrl-X to save it and exit from nano.

Later on, we'll learn how to ask the user for numbers, so we'll be able to ask the user to enter a value for x, instead of having the value written explicitly into the program.

The for Loop:

Loops are the reason computers were invented. It's easy to do something once, but computers excel at repetitive tasks. C offers several ways of doing loops. One of them is the “for” loop:

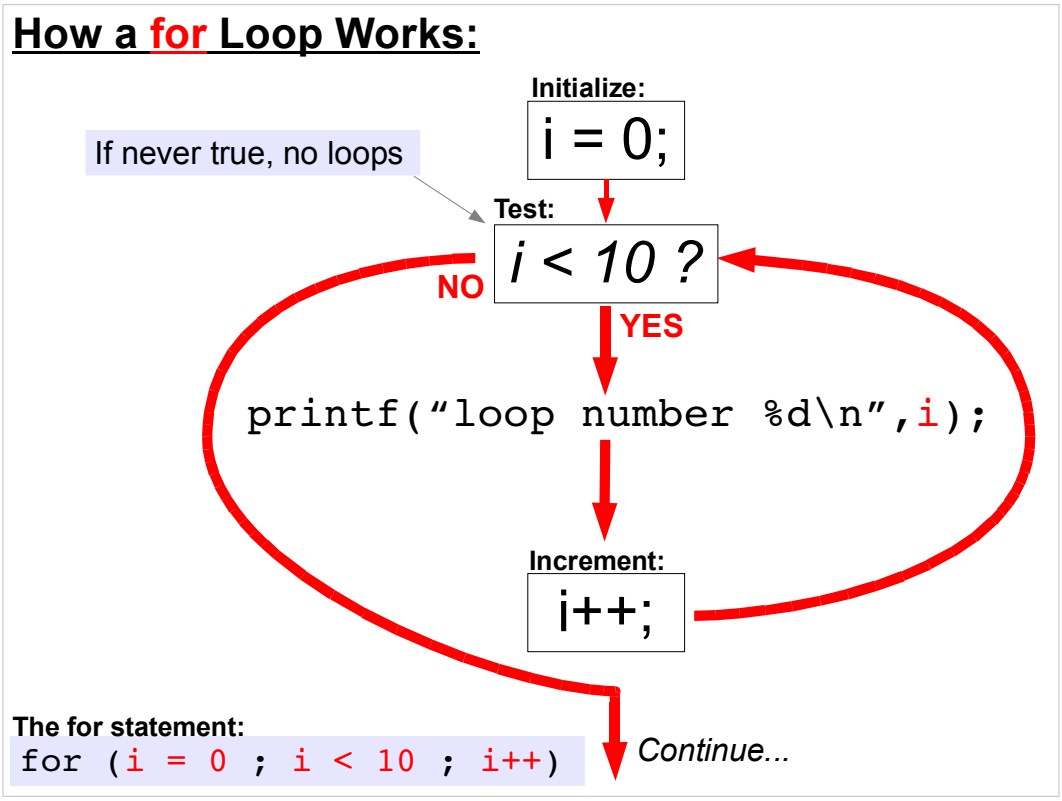


The statement “i++” means “set i to i + 1”. In C, “++” is the increment operator. There's also a decrement operator that decreases a variable's value.

Try It!

The loop continues until the test condition is no longer true. See the following graphical representation.

In the example above, I've left out the “framework” that we used in our other examples, but you'll need to put it in to make a complete program. Take a look at the “include” and “main” lines of the earlier examples, and remember to add a closing “}” at the bottom of your program.



This diagram may help you understand what's going on in a “for” loop.

Notice that if we gave i a value like 100 in the beginning, the program would never do the printf. Instead, it would just skip the loop entirely.

This is important, because later on we'll encounter another kind of loops that will always be acted on at least once.

Practice Problem #1:

What if we wanted our program to count to 1000 by hundreds (100,200,300,... up to 1000)? How could we do that without changing the “for” line in the preceding example?

Practice Problem #2:

As you do more programming, you'll find that randomly-chosen numbers can be amazingly useful. Try doing this to one of your “for” loop examples:

* At the top of the program, add:

```
#include <stdlib.h>
#include <time.h>
```

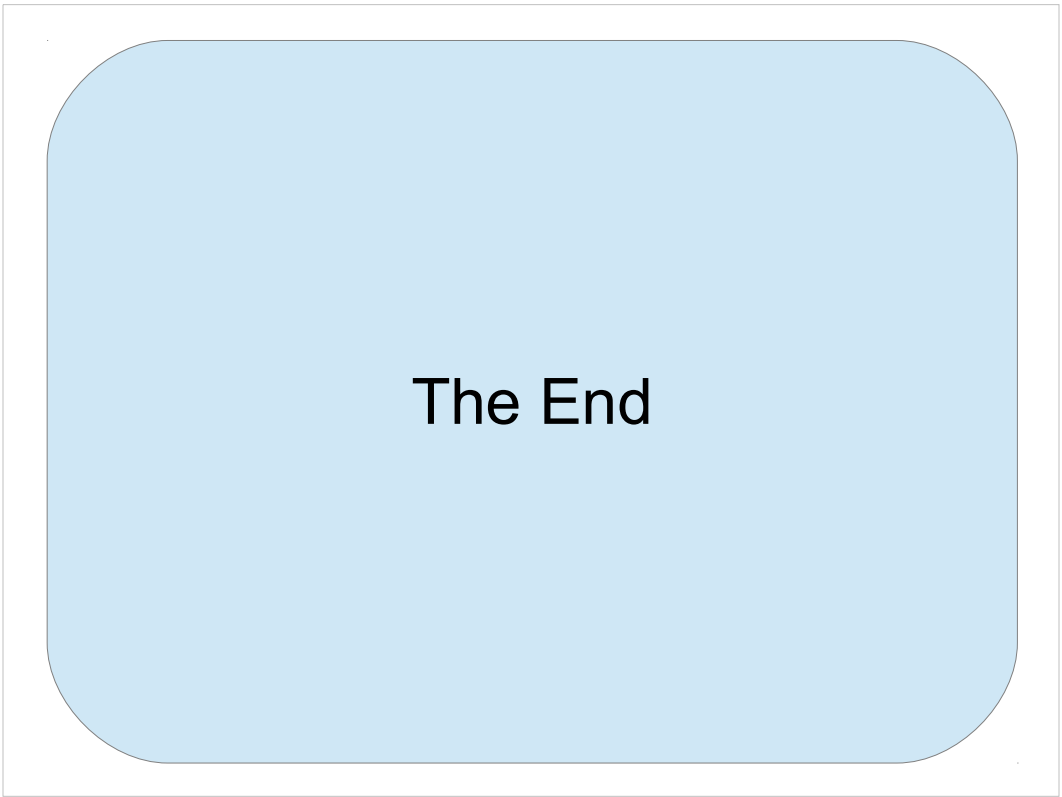
* Then, just above the “for” statement, add:

```
srand(time(NULL));
```

* Finally, inside the “for” loop put a statement like this:

```
printf( “%lf\n”, rand()/(double)RAND_MAX );
```

Here are some problems to challenge you! We'll look at solutions for them next time.



Thanks!