LEARNING C++ CODING ON LINUX

Quick Note: This was written by a fairly new programmer (4-6 months using C++ at the time of writing). Please comment or email me at ccmachine[at]gmail[dot]com if there are mistakes or things you believe i should/should not have said. However this also gives me a good perspective into what somebody completely new should know.

Why would you want to learn C++? It's a time-tested language that's been around over 20 years and as such, C++ code can be run on pretty much any computer or device you come across (so long as you compile it for such, more on that later).

What do you need to learn to code?

1. A computer
2. Logical and maybe mathematical skills (particularly for game programming), or at least common sense in both.
3. A text editor and C++ compiler.

Obviously, coding can be pretty complex. For this I'm going to assume you're using Ubuntu or any other Linux distribution with a packaging system (apt, yum, synaptic, etc).

You probably have the first two. In Ubuntu and most modern Linux systems, installing a text editor and C++ compiler is easy. I recommend the packages "kate" (the KDE text editor) and "g++" (the GCC C++ compiler). we can do this in Ubuntu by one command: (note it is case sensitive)

sudo apt-get install kate g++

Hit enter and type your password. You can also install kate and g++ in synaptic if you wish, but we should definitely get used to using the commandline if we want to begin coding.

```
#include <iostream>
using namespace std;

int main()
{
cout << "hello world" << endl;
}
```

Confused yet? ^_^ Don't worry. The above is a "hello world" program - mostly use to prove that the code compiles and runs, and also to teach how a C++ program is put together. Let's analyse it.

If you really don't understand some of the notes, don't worry. All will become clear later on.

```
#include <iostream>
```

This line is an include line. What it does is declare that we want to include another set of code so we can use the functions it provides, in this case iostream. iostream provides the ability to send text to the terminal and also get input back. We use #include to use any library or any code file that we may need to use.

Note that all of C++ is case sensitive, including filenames. If we instead wrote:

```
#include <Iostream>
```

"Iostream" would not be found and the program would not compile.

using namespace std;

This is a command, setting that we want to use the std namespace. This allows us to use cout and cin to print text to the terminal prompt or get typed text from the user. Notice that it ends with a semicolon (;). As it's a statement, C++ requires ; marks to indicate the end of every statement. This may sound annoying, But it's also useful, as it allows us to write many statements on one line or one statement on many lines if we want, just so long as each statement is separated with a ; mark.

int main()

This declares the beginning of the main() function. main() is where all C++ programs start running from. Why the ()? this is because main is a function. Any function being defined must set the parameters to be run. A parameter of a function is some form of information being fed in - for exmaple an adder function would require a number input, a number parameter. In this example, main requires no parameters, so we just use two brackets "()". When declaring a function, we must give it a type. main() is always of type int, so we write "int main()" to declare a function called "main" of type int with no paramaters. Parameters would be inside the two brackets if the function needed any.

{

This is the beginning of main(). Any function being defined must have { } brackets around the code to be run when the function is called.

cout << "hello world" << endl;

This is the statement that prints "hello world" onto the commandline. As it's a statement it has a semicolon (;) after it. We write "cout" first as it's the command we want to use. The "<<" indicates that we're feeding some form of

data into it (in this case the text to be printed). Text must always have quote marks around it so the compiler knows we aren't trying to call two statements being "hello" and "world". The second "<<" indicates that we want to feed more data in that appears after the first. "endl" is a linebreak: the same as hitting the Enter key on the keyboard.

In programming there is nearly always no one definitive way to do something. To help you understand cout and "<<" usage, we could feed the "hello" and "world" in separately if we wish:

```
cout << "hello " << "world" << endl;
```

Notice i left a space after hello inside the quotes. This is done so the program will write "hello world" and not "helloworld" - spaces are not automatically placed. There is also more than one way of doing a linebreak - "endl" in this example.

```
}
```

Ending bracket for the main() function. This doesn't need to be on a separate line as the code... We could have written this, instead:

```
#include <iostream>
using namespace std;


int main()
{ cout << "hello world" << endl; }
```

Likewise, you can have the { } brackets and code on the same line as int main() if you want.

So, we now understand what this code does. How do we get it to run on a computer? For a C++ program, we need to compile it first. This is because computers only natively understand one "language" - machine code. We have

languages such as C and C++ to help people understand and write code more easily. With this analogy, we need a translator to change our code into machine code (otherwise known as a binary - as machine code is literally binary numbers - 1's and 0's.). Run kate (Applications -> Other -> Kate on Ubuntu 8.04), make a new text file and copy or retype in the 7 line long hello world example. You may use gedit ("Text Editor") if you wish, but kate is generally better for programming. If you can't find Kate on the menus, you can always open a terminal and type "kate" into it and enter - this will also run it. Save this anywhere you like but the filename must end with ".cpp". So "helloworld.cpp" or "myfirstprogram.cpp" etc. I would recommend saving it into your home folder or a folder inside your home to make the next step easier. Now, open a terminal. (Applications -> Accessories -> Terminal).

You will need to "cd" (change directory) to the folder you saved your .cpp file in. I saved mine to a "c++" folder inside my home directory. As the terminal starts in your home directory by default, I simply type "cd c++" and enter:

```
ccmachine@andrewiv-portable:~$ cd c++
ccmachine@andrewiv-portable:~/c++$
```

Now, type "g++ yourfilename.cpp" where yourfilename.cpp is what you saved the file as. Mine is "helloworld.cpp", so i do:

```
ccmachine@andrewiv-portable:~/c++$ g++ helloworld.cpp
ccmachine@andrewiv-portable:~/c++$
```

If there are errors listed on the commandline, it means that there are errors within your .cpp file. Make sure the file was copied exactly as it is written above. As C++ is case sensitive, make sure everything is the correct case (many times i have written "int Main()" which doesn't work as the compiler wants to run main(). If you copied and pasted it, you may need to type it out manually, as the copy-paste method can change the characters in strange

ways, so then the compiler doesn't recognize them, though they still look correct.

Well done! You've just compiled your first program. By default, g++ writes the output file to "a.out" - you will notice this if you go to the folder where your .cpp file is located. This similiar to a .exe file on windows - its a binary that the computer can run. Remember, any changes made to the .cpp will need to be recompiled in order to change the binary and hence the program being run. To run it in the terminal, we use

```
ccmachine@andrewiv-portable:~/c++$ ./a.out
hello world
ccmachine@andrewiv-portable:~/c++$
```

See? It prints "hello world" to the commandline. Not much, but that's all we told it to do. Remember, any program is just a set of instructions to be carried out by the computer. We can make it do more by adding more instructions. We can have multiple lines of cout, or even feed in more text after a "endl" to print multiple lines of text, like so:

```
#include <iostream>
using namespace std;

int main()
{
cout << "hello world," << endl;
cout << "hello world again!" << endl << "and again!" << endl;
}
```

Now we recompile it (run g++ filename.cpp again, don't forget) and run it:

```
ccmachine@andrewiv-portable:~/c++$ g++ helloworld.cpp
ccmachine@andrewiv-portable:~/c++$ ./a.out
```

hello world,
hello world again!
and again!
ccmachine@andrewiv-portable:~/c++$

Cool eh? As I wrote those three lines in quotes and with and endl after each one, It printed those three on separate lines in that order. I agree, this isn't very funtional in the practical world, but we must start small before doing anything complex. To a child just starting to learn to write on paper, completing a 5-page essay to earn him a GCSE qualification would be very very difficult without knowing grammar, learning to spell etc first.

I could go on and on teaching the basics and more, but I just wanted to give you an Ubuntu and Linux-specific start into the wide world of C++. The best place to continue would be the C++ tutorials on cplusplus.com:

http://www.cplusplus.com/doc/tutorial/

You should read "Structure of a program" as it does contain some things I didn't tell you about for simplicity (adding comments to a program, the "return 0;" statement to properly end a program, etc)

One final word: Play and Experiment! The best way to learn coding in any language is to play around and see what works, and what doesn't. This way, you learn better and also have the satisfaction being able to say "I made that :-)" as it runs in front of you.

Source : http://www.soslug.org/wiki/learning_c_coding_on_linux_your_first_c_program