Introduction to C++

June 11, 2017

1 Motivation

1.1 What is C++?

C++ is another programming language.

1.2 Why do we need another language?

That is a pretty good question. We already know python, so why should we need another language?

It turns out that C++ is much better than python for writing code which runs fast.

which is crucial in co

That sounds pretty dumb, why can't python also be faster? They run on the same computer, so they should have same speed no?

No, it turns out that C++ is a much less forgiving language, and as the programmer, you have to do a lot more work. This sounds like a bad deal, but it turns out that this lets you make better decisions (essentially micro-manage), which leads to faster code overall.

What do we mean by "less forgiving"? Let's find out.

2 Basic structure

2.1 What does a C++ program look like?

Program 1: this does nothing

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

Before we run through an explanation, you should try to type this program out (don't copy and paste, it will not work) on an online IDE, and run it.

Now, the first line of the program contains the words `int main()`. `int` is a data-type in C++ that is used to represent integers.

Wait, what's that again? We've never had to deal with this "data-type" nonsense in python.

Remember how we had variables in python? We have them here too. Except this time, instead of just asking the computer for a "box" where we store some value, we also have to tell the computer what kind of value we will store.

Phew, ok, `int` is a way to tell the computer we want to store an integer. But we aren't storing any integers here!

Actually, `int main()` is a *function*, which returns an `int`. Why are we already looking at a function (aren't those the scary things we did after a week of learning python)?
It turns out that when you run a C++ program, the computer doesn't simply start running from the first line of code it sees (like python). Instead, it politely looks for an *entry point*, like a door to enter your house. Now, of course, a programming language has no doors. So the *entry point* is instead a function that the computer executes when you run the program.

Hmph, this seems so dumb. Anyway, what are those `{}` things? The part between the `{}` is the actual function definition.

Remember how in python, when we define a function, we place everything which is part of the function one indent level deeper than `def function_name`? It turns out C++ does not care for indents. It uses `{` to denote the beginning of the function, and the corresponding `}` to denote the end.

But unindented code is e

Now, of course we come to the statement in line 3, which says `return 0;`. Note how this line is terminated by a semi-colon `;`. In C++, unlike python, every line of code is terminated by a `;`. The reasons for this choice are beyond the scope of this article.

0 is an example of exit c

We're familiar with all of the other things in line 3 (ie. `return` and `0`). And we're returning an `int`, just like we promised in line 1, so the computer should be happy. But why `0`? Well, it turns out that if you return `0`, the computer takes this to mean that everything went off without a hitch, right according to plan.

Try changing the program, to return `1` instead, run, and read the error message that pops up.

Try changing the program to blank (remove the entire function `main`), run, and read the error message that pops up.

Phew, that was a lot of work for a program that doesn't even do anything.