Notes on C/C++

Pointers:

There's a weird zen to the way pointers work, and it's quite powerful, though not immediately evident. You see, pointers aren't actually anything special in C. By that I mean, the compiler treats them just like any other variable in a program, because a pointer has all the same properties of any other variable:

- *An address*: every variable has some place in memory where it's actual data resides. That location is a number, just like any other.
- *A Value:* The value of the variable is it's actual data. It sits somewhere and is accessed via the it's address

The above is true for all variables, *including pointers!* The only thing that makes pointers any different from anything else in C is that the value of a pointer is actually a record of the address of some other variable/bit of data. Because of this, you can have all kinds of weird things like pointers to pointers (often called *superpointers*), or you can treat the values of things that are not pointers as references.

The thing that's different about pointers isn't an actual part of the pointers, it's the *reference* and *dereference* operators that are part of C. The *reference* operation returns the reference, or the address of that variables value. The dereference operation makes the compiler go get the data at a given reference and operate on it.

When using pointers, the location of the asterisk (*) is important.If you put the asterisk before a pointer, you are **dereferencing** that pointer and accessing the data underneath/inside it. The opposite of this action is called **referencing**, and is done by placing an ampersand (&) before a variable. When you reference something, you indicate that you want to access the address of that thing.

Examples:

```
int x = 20;

int *x_pointer;

x_pointer = &x;
```

Notice that here we don't refer to "x_pointer" using either a "*" or a "&", something we've not covered.

What's going on is we're accessing the value that is contained in x_pointer, namely, the number that is stored by x_pointer that is used as a reference to the location in memory of some other variable.

At this point, the value of `x_pointer` is the location in memory where the value of `x` is stored. The if you were to print `*x_pointer`, you'd see that it is the same value as `x`. If you printed `x_pointer` (without dereferencing) you'd be printing the value of `x_pointer` itself, the value of which is the location in memory of `x`.

Declaring Pointers

When you declare a pointer to something, there's more than the standard declaration. You specify that you're making a pointer, and you specify the *type* of the thing you're pointing to.

An example is the following:

```
int *someNum;
```

That declaration declares you're creating a pointer, and that it points something that is of the type `integer`.