CS 223 – Wednesday September 4th, 2013

Particulars of the C++ Language

Review:

- Time complexity
- Manipulating data with algorithms
- C/C++
  - Encapsulation of data
  - Classes/objects
- Coupling and Cohesion
  - Something that'll be covered in the future
- Inheritance/Polymorphism
- Generics/templates
  - More flexible code to handle new data types
- Operators
  - You can control/overlaod how operators are dealt with in a class
  - Check out the "class.cpp" file on Angel

Constants and References

- Constants
  - Unmodifiable

References

- Uses the `"&"` symbol during initialization
- Less powerful, but safer than a pointer
- References are literally just another name for the original value.
- You must initialized a reference to a variable
- They cannot be re-assigned
- It's a new variable of the same type as the original, but with it's address set to the address of something that already exists. So it's literally another name (another "reference"!) for something that already exists.

Pointers

As I've outlined in previous notes, the way to think about pointers is that they're variables just like any other in a language; they have a value, and they have an address somewhere in memory where that value is actually stored. The only thing that makes pointers any different is that a pointer is a variable that has a value that is actually the address of some other value variable.

Indirection

Indirection is the act of having pointers that point to other pointers (**superpointers!**). `Pointers` can have as much nested indirection as they want, while `references` can only have the one level of indirection.

Arrays/Pointers

`C-strings` are actually just pointers to a series of bytes in memory, with each byte being the ascii code of a character. The way you can access a `c-string` so simple is these series of characters are sequential in memory, and thus you can go from one to the next by just going to the next address in memory.

Indeed, that's actually how pretty much all arrays work.

`Consts`

`Consts` offer a "safe", read only way of passing something into something else. It marks a variable as totally unchangeable.

Inlining

Inlining is an optimization technique that's carried out by the compiler. By specifying something as to be inlined, you tell the compiler to copy-paste that code into wherever it's called, instead of actually having it be a separate function. This reduces the number of jumps between programs, and the amount of data to be managed on the call stack, but it increases the compiled size of the program.

Classes

Something to note about having base-classes is that the base-class in almost all cases cannot be instantiated directly. Most base classes exist so that derived classes can actually implement the methods outlined in the base class. Additionally, the base class doesn't have the details necessary to have it's variables and values populated properly.

Protected

Protected methods / variables in a class can only be accessed by other instances of the same class.

Virtual Functions / Virtual Tables

Allows the method to the overwritten in the derived class.

Each virtual function has it's function pointer entered into the virtual table. Then when the virtual function is implemented, the function pointer is set to point back to the implemented function.