Michael Camara

Honor Code Pledge: This work is mine unless otherwise cited

CMPSC 220

Due Date: 11/30/15

Final Project: Progress Report

Much of my time on this project has been spent reading tutorials and books about the various features of C++ and trying to isolate interesting components that differ from other programming languages. It's been both a boon and curse to realize how similar C++ is to Java, the language that I'm most familiar with. Both are object oriented languages with lexical, block-level scoping that are statically typed. Many of the basic constructs for selection, iteration, and class structure are very similar, and in many cases identical. However, there are a number of important differences, beginning with the fact that C++ is a compiled language, whereas Java is has characteristics of both compiled and interpreted. This requires the C++ compiler to check for errors and optimize the code before the user ever actually executes it. While this may result in faster performance compared to an interpreted language, it has also made the debugging process much more difficult, as the error logs are much less specific and less helpful.

Another, more specific, difference between C++ and other languages like Java is how parameter passing works. The ParameterPassing.cpp file shows the three ways that parameters can be passed in C++: by value, by reference, and by const-reference. By-value passing is similar to Java, where a copy of each parameter is made at the beginning of a method call to be used within the method itself. By-reference passing instead targets the actual memory address of the indicated parameter using the ampersand ("&") or "address-of" operator, which can then be directly used and permanently altered if desired. By-const-reference is similar to by-reference, except the parameter is declared as a constant using the "const" keyword; i.e. the value of the parameter cannot be changed. For instance, as the example code shows, if a list of values is used as a parameter, then by-const-

reference will allow the method to access the elements of the list but will prevent them from being changed. This kind of "read-only" access adds a degree of security and consistency that other languages like Java do not support. Both types of reference-passing can improve performance by omitting unnecessary copying of parameters. The ParameterPassing.cpp program uses some simple timing experiments to show the cost associated with the pass-by-value method, when the same result could potentially be achieved with minimal cost using the pass-by-reference method.

I know that I still have a lot of work to do with the project, particularly with creating various sample programs to show more language features. I've been struggling trying to think of creative ways to show these features that don't simply iterate on the many examples that I've come across in my research. However, I do have a general list of features that I think I could wrap into individual programs. These include:

1. Parameter passing (Pass by reference, default arguments, unspecified arguments)
2. Input/output streams and error handling
3. Using / #include / namespaces
4. #define / typedef / const / constexpression / extern
5. User defined operators
6. Records (unions, structs) / enumerations
7. Scope (global, namespace, local, class, statement) / scope resolution operator (::)
8. Multiple inheritance
9. Header files / interfaces / templates
10. New / delete operators
11. Pointers (*, & operators)

As I continue to make more programs I'll hopefully determine which features can be adequately showcased together, and which should be separated in their own programs. Any suggestions for topics I definitely should or should not include would be much appreciated, but I feel comfortable proceeding with this tentative list for now.