Individual Programming Assignment

Dr. Xiaoming Liu

Tony Rocha

T13002597

Due date: 12/4/2018 (Tuesday)

Report On Simple Sorting Program Implemented In Java And C++

For the final personal project, I was tasked with creating the same program in two different languages, to compare and contrast the differences between the two languages chosen. To test something very argumentized, but also important, the topic chosen is sorting a list. The code is simple, and easily available – the project is hosted at <https://github.com/Rohzek/CSC420620> – using the ‘built-in’ sorting algorithms.

For the Java implementation, the Collections#sort algorithm is used, which took 196ms to run, almost a full 100ms longer than the C++ implementation (at 99ms). This due to not only the virtualizing of Java, using the bytecode, rather than being compiled to a lower level, like C++ but also the way the Collections library works. Java’s sort option converts the list (or any other option that implements Comparable) that’s passed in to an internal array, and then runs the swap on it. It then re-converts to a list and returns that. The swap itself is a quicksort algorithm – used for all primitive types, while objects are a little different. The conversion takes a significant amount of time, that the C++ implementation doesn’t have to deal with.

For the C++ implementation, the std::sort algorithm was used, rather than the qsort which would have been more comparable to the quicksort of Java’s Collections option, however sort is faster – being a hybrid algorithm based on quicksort, and heapsort: introsort. This also ignores the fact that C++ compiles down farther than Java, which floats on a mid-layer of bytecode to be compiled down at runtime. The idea was to choose the ‘default’ sort option, which here means calling ‘sort’ and nothing else.

As far as readability and writability are concerned, the two are basically matched, as Java was heavily influenced by C++ while it was being created. This rings true for most other comparison types, as well. They have the same level of complexity, and nearly identical orthogonality. The flexibility and reliability are where the two really split. Java’s floating bytecode structure means that in general it’s slower to begin with which could arguably skew it into the unreliability camp. It also means that it can be more flexible on it’s own, but not as strong on one system, as it can’t be natively compiled. The cost of maintenance and execution of the two is where things really deviate the most. As mentioned several times, Java is a heavier run than a C++ application, due to the floating nature of the bytecode being compiled at run time; but the maintenance of Java is much more of a dream. As the code is written once, and distributed once, maintaining the code is much easier than that of C++ which, while can be compiled to multiple systems, has to be done so by someone with the knowledge of compiling applications, where the Java Runtime Environment handles interpreting the code and nothing has to be done to update the compiled jar file when switching runtime updates, or even entire systems.

Deciding on rather or not you want to develop and application in C++ or Java, based on just these numbers would make C++ the clear winner, based on it’s speed. But the hidden cost of maintaining a single code and binary source, versus compiling for multiple systems could be a reason to choose Java instead. Distributing and maintaining your application is much easier with Java. If you only wanted to support one system (such as building for a Windows environment) however, C++ is a clear winner, as it’s similar in all ways that count, but have advantages of compiling to a native executable, and the pure speed of it’s implementation of libraries and algorithms in ‘default’ scenarios.