CS 22510: "Where Have I Been?" - Languages Comparison

Due on Friday, April 11, 2014

James Euesden - jee22

Contents

Introduction

This document is the second part (Assignment 2[**?**]) of the "Where have I been?" assignment. In this report, I will discuss the languages and my experiences with them. While the first part (Assignment 1) was about the programming for GPS Processing, this is about the differences and similarities between them. What I found useful, or difficult, between the languages, any libraries I may have chosen to use, and what language I felt was best suited to the task.

Language Comparisons

The amount of code between each language was rather similar. I used GitHub to keep track of my code and the versions of it as I completed part one of the assignment. One of the benefits of doing this was that it provided a great break down of how much percentage of the repo was taken up by the different components (screenshot taken before this document was added to the repository):

Figure 1: The percentage of languages on the GitHub repo for part 1.

The figure shows that the amount of code for each language that I wrote was very similar. I feel the reason C++ has more code than the other two may be due to each class having a Constructor, a Destructor and a Copy Constructor, whether they were used or not. With this in mind, it should be apparent that my writing of the application in each language was very similar, and while one particular section may take more code than another, that same language may do something much simpler than in another too.

An example would be the file operations in Java vs C, where it takes more for C to handle getting the data out of the file and char arrays than in Java. Opposite to this, my C program handles date and time much easier and with less code than that used in Java for the most part. Overall, I found the simplicity of C and C++ help me in writing the code in the most simplest way I could think of. A huge benefit over Java, where I sometimes felt I was writing bloated code that was unnecessary (and later refactored out).

File Operations

In all three languages, the file input and output was relatively simple and similar. With Java, I had a slight difference in that on every update of a timestamp I output the XML into a StringBuilder, and then output the full String at the very end of the application. While with the C and C++, I output directly to the file when a timestamp was updated. This was made very easy through the use of pointers to the FILE, and keeping a reference to the class for the file operations.

Data Accessibility & Manipulation

With Java and C++ having objects rather than simple source files with functions and variables, it was fantastic to be able to put a variable as private, and restrict access to it from outside of the class except through the defined use of getters and setters. While something similar can be done

Date & Time

For both C and C++ I used the 'time' library[**?**], and for Java used the Calendar class. I found the Calendar class cumbersome to use and in some ways hindered my programming. On the otherhand, using the time

library was perfect for checking the timestamps between sentences, comparing them, creating new timestamps from sentence data and even updating a timestamp with new sentence data without a date.

When programming in Java and using Calendar, making a new time with the date and time provided by a sentence was fine. When it came to updating a time without a provided date, it became much more difficult to do. My method for handling this in all three languages was similar. Just use the most recent date provided (potential pitfall for the application's future use where the time passes 00:00. This could be rectified in a future edition).

This was simple to do in C and C++, with simply turning the time_t struct back into a tm struct, updating the values and then using mktime() to get it back to time_t. While the process with Calendar is similar, I found that I had trouble getting the time to be correct, with milliseconds assigned randomly, and needing to use .clone() to get correct values for the date.

Libraries

I chose not to use any libraries outside of those provided with the standard API of the native languages. Below are those libraries that I could have used, why they could have been useful, and why I didn't use them.

NMEA parsing

For all three languages there are libraries that I could have used in order to parse the NMEA sentences into data, without me having to dissect them myself and extract the data[**?**][**?**]. However, I chose not to use any of these. My main reason for this is that I wanted to understand the sentences myself, and to only extract the data I wanted and then discard them, rather than extract all data from the sentences. On a larger project that intended to use the majority of data from the NMEA sentences, I would look into using one of these open source libraries to save time. For my own assignment in my own work, I felt that it was acceptable to not use these.

C++: Boost

There is a library for C++ called Boost[**?**]. This library provides many useful functions for C++ that aren't in the native language. A number of these could be have been useful for manipulating strings, and saved me time and code. However, as with the NMEA 0813 sentences, for an assignment of this size I would rather use my own code and work through the problem to better my understanding of C++. Boost is a useful library that I will remember for future, larger, projects where their functions may be invaluable in saving me time and effort.

Java: DocumentBuilder & XML Builders

While not an additional library, Java has the use for classes such as DocumentBuilder, which can XML tag a file (UTF-8) for us. Again, I chose not to use this, and provided my own methods for putting tags around the data. This is partially because of tags such as "¡wpt lat="" lon=""¿ where it is a tag built up of multiple components I found easier to do myself, and also because of the size and amount of XML tags being output. If the applications were required to output a larger variety of XML tags, I may have found it useful to find XML Builders. For what the problem consisted of, I felt it was easy enough to do it myself, and to further my understanding of building XML files in my programming.

Suitability

In terms of what language is best suited for writing this application, I would say that a lot of it comes down to personal opinion. I found all three languages had their challenges or parts that aided me, and so each of them is relatively suitable for this process.

However, out of all of them I felt that C++ was the most suitable for my implementation. My justification for this is that I found it the easiest and quickest to write in. The functions are all relatively simplistic, it has the simplicity of C, while providing the bonuses of Object Oriented programming like in Java. The fact that C++ has access to it's own libraries and functions, while also having access to those provided in C, made C++ an joy to write the application in.

Conclusion

References