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 between the languages and my experiences with them while writing the applications. What I found useful or difficult, 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 languages (screenshot taken before this document was added to the repository):

Figure 1: The percentage of languages for part 1.

The figure shows that the amount of code for each language 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 in one language may take more code than another, that same language may do something much simpler than the others.

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. Overall, I found the simplicity of C and C++ helped me in writing the code in the most simplest way I could think of.

Code Clarity

The code for all three applications is quite clear in it's operations, and I feel this is down to the way I wrote the code and comments. Through the use of good naming conventions and meaningful variables/methods/functions, the flow of each application can be followed and understood through the names of the components. It can get a little unclear in some aspects, such as setting the Latitude and Longitude from a char array in C, but any obscurities can be cleared up with comments.

Each language has it's own syntax and quirks when it comes to the way that applications should be written. In this case, the general logic of the applications was the same and any language-specific quirks weren't too difficult for me to understand when programming, or to understand when reading back over the code after some time of not working with it.

```c
void set_latitude(stream * streamer, char * string_lat, char * lat_facing) {
    char *decimal;

    /* Find the decimal point in the string*/
    decimal = strchr(string_lat, '.');
    /* Get the index of the start of the 'time' through -2 of the decimal*/
    int start_of_time = ((int)(decimal - string_lat)) - 2;

    /* Put aside some c style strings for the degrees and time. */
    char * degrees = calloc(1, sizeof string_lat);
    char * time = calloc(1, sizeof string_lat);

    /* Take apart the latitude string to get the components for
     the degrees and time */
    strncpy(degrees, string_lat, start_of_time);
    strncpy(time, string_lat + start_of_time, sizeof string_lat);

    /* Convert the strings into numbers. */
    int degrees_int = atoi(degrees);
    double time_dbl = atof(time);

    /* Convert the NMEA coordinates into decimal.*/
    time_dbl = time_dbl / 60;
    time_dbl = time_dbl + degrees_int;

    /* If the facing is South, turn the Latitude negative. */
    if (strcmp(lat_facing, "S") == 0) {
        time_dbl = time_dbl * -1;
    }

    /* Set it! */
    streamer->latitude = time_dbl;

    /* Free up the calloc'd memory. */
    free(degrees);
    free(time);
}
```

(a) An excerpt from my C application: Setting Latitude from a char array and converting it from NMEA to decimal.

```java
public void setLatitude(String stringLatitude, char compass) {
    /**
     * Move the period index of the Latitude to be correct for
     * GPX format.
     */
    int periodIndex = stringLatitude.indexOf(".");
    int start_of_minutes = periodIndex - 2;
    int degrees = Integer.parseInt(stringLatitude.substring(0, start_of_minutes));
    String string_minutes = stringLatitude.substring(start_of_minutes, stringLatitude.length());
    double minutes = Double.parseDouble(string_minutes);

    minutes = degrees + (minutes / 60);

    /**
     * If the compass is South, make the Latitude negative.
     */
    if (compass == 'S') {
        minutes = minutes * -1;
    }

    latitude = minutes;
}
```

(b) An excerpt from my Java application: Setting Latitude from a String and converting it from NMEA to decimal.

Figure 2: The difference between Java and C, in code clarity when setting the Latitude.

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. All three languages, while different, were relatively helpful with reading in from a file.

Data Accessibility & Manipulation

With Java and C++ having objects rather than simple source files with functions and variables, I was able to put a variable as private, and restrict access to it from outside of the class except through the use of defined getters and setters. Data is relatively easy to access in C, and it isn't as secure as setting something as private. In this regard, C lets itself down in that it is not 'open for extension, closed for modification' when data is in a struct that has been included in a file. This can lead to future developers circumventing the way that the original programmer intended data to be accessed, should this application be built upon in the future.

In terms of manipulating the data, there were generally no problems. The only issues I encountered were in C and C++, when I had to remember to set values to something before allowing them to be used/compared with something, in the off chance that they had not been initialized to 0 or null, and instead contained data from elsewhere. While not a huge hinderence to program, it sometimes made debugging provide very interesting results if I had forgotten to do this.

Memory Allocation

It was useful to have the 'garbage collection' in Java, saving me from having to use free() and delete(). In C and C++ I had to trace the bits of memory to where they should correctly end and remove them appropriately, without accidentally causing the application's memory to lose it's integrity with a double free situation. On the other hand, the freedom of allocating and freeing memory gave me much more control over how data was stored in the applications. While the applications were only small and so didn't take up too much memory, it is something that I will be keeping in mind for future applications where I have a choice of language and want to be memory space conscious.

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 other hand, using the time library was perfect for comparing timestamps, 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, updating the date and time provided by a sentence with both date and time was fine. However, when it came to updating a time without a provided date, it became much more difficult to do and I spent a good while writing and re-writing code to use an older Calendar object to create a new one with the updated time.

In C and C++, this was simple to do by turning the most recent 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 ended up quite 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, which slowed down my progress when writing the Java application.

Libraries

I chose not to use any libraries outside of those provided with the standard API of the native languages. Below are some 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 having to dissect and extract the data[**?**][**?**] myself. 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 projects where their functions may be invaluable in saving me time and effort.

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 being so little. 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.

Conclusion & 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 acceptably suitable for this process.

However, if I had to choose, 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 methods are all relatively simplistic. it has the simplicity of C, while providing the bonuses of Object Oriented programming. The fact that C++ has access to it's own libraries and functions, while also having access to those provided in C, made C++ a joy to write the application in with ease. Adding to this that there are libraries like Boost and NMEA parsers, and it becomes clear to me that C++ is perhaps the most suited for this application out of all in my opinion.

References