A COMPARISON OF FUNCTIONAL AND MAINSTREAM PROGRAMMING LANGUAGES

CMPT 383 D100, FALL 2021

Prepared by: Ahmed Ali
301329325
aaa123@sfu.ca

Implementation

Performance:

My performance testing was done with a file that contained a million ballots. I generated this file by pasting from the *"example100000"* file ten times.

1. **Java:** Java took the least amount of time to run with this large ballot file. It took less than two seconds to complete. The program read the input file twice to get everything needed. I was able to calculate the empty and total ballots using if statements and incrementing during each iteration throughout the file reading process. I was not able to do this in the other two languages, which is why Java ran the quickest.

2. **Haskell:** Haskell was much slower than Java when running the file. It took exactly 12 seconds to read the large ballot file. This is because the program had to go through the entire file various times to calculate each of the full ballots, empty ballots, and approved candidates.

3. **Racket:** Racket was the slowest of them all taking about 19 seconds to complete. Just like Haskell, it also had to read the file various times to complete, but it was slower and had more than twice the number of functions Haskell did.

Ease of development:

1. **Java:** It took me around six hours in total to get my Java implementation running. Three of those hours were spent looking for optimizations and to display the results by sorted values. It took me 30 minutes to debug and test my program, which was easy to do with Java. This implementation took me the least amount of time to work on because I was very familiar with this type of programming. I found it easy for me to code and it was quite enjoyable. When I got stuck on trying to figure out a way to display the HashMap content in descending order, I was able to easily search for resources online because of Java's popularity.

2. **Haskell:** Haskell took around ten hours in total for me to complete. Around five of those hours were spent on debugging, which can be difficult to do in a

language like Haskell. After using Racket and becoming more familiar with functional programming, I was able to create my Haskell implementation more efficiently as compared to Racket. What also helped me was watching and understanding the professor's implementation of the *"approvalCount"* function. I began to really appreciate using *"$"* in Haskell. Currying was very pleasant in Haskell, as compared to Racket. Using the *"where"* clause made it easy to define variables and create calculations needed for the rest of the function. I enjoyed implementing this program in Haskell.

3. **Racket:** I struggled with the Racket implementation. It took me around 15 hours to get my program working. Most of my time was spent on debugging, which was around nine hours. I also found it difficult to format the code and no matter how I formatted it, it did not look clean enough to me. I started off by first trying to create several different tasks within one function but began to realize that it is something difficult to do in Racket and sometimes not possible. It would also make the code less readable and confusing. I decided to split each task into its own function, which made it easier for me to implement.

   I enjoyed using the map and filter functions, as it made my code more concise, and I got comfortable using it frequently. It made me appreciate the simplicity and efficiency of certain functions within Racket. As I got more comfortable with higher order functions, I began to avoid using recursion whenever possible. Overall, I did not enjoy programming in Racket.

Quality:

I considered a line of code to be simply based on the line that it was on in the text editor. For example, if the text editor indicated that the last line of the program is on line 45, then I consider my program to have 45 lines of code.

1. **Haskell:** My Haskell implementation was shorter than the other two with a total of 39 lines of code. I had a total of five functions in my program, including the main. I find that the Haskell code, while still being concise, was easy enough to read. Having minimal brackets and aligning similar parts together made the code look beautiful.

2. **Java:** My Java implementation contained 81 lines of code and a total of four functions, including the main. The code is easy to understand compared to

Haskell and Racket. While Java libraries can be very useful, a lot of built-in methods are made up of long names and can make the program look longer compared to the other two.

3. **Racket:** My Racket implementation was the longest with 13 functions and a total of 95 lines of code. There were too many brackets, and the program can look quite confusing for what it does. When someone is reading the Java implementation, they can understand and get a good idea of what the program is trying to do. Someone reading Racket will need to pay a little more attention to understand what it is doing. A lot of functions were needed in my Racket implementation. However, if I were to implement it using Racket again, I am certain that my code would be shorter after the experience that I have gotten from this project.

Recommendation

Overall, after comparing my experience with implementing this program in three different languages, I would recommend coding it in Haskell.

The Haskell implementation took the best from both Java and Racket when it came to readability. While still being concise, the code was readable and most of it can be understood with what it is doing with ease. The overall code looks beautiful and is quite powerful. For me, having all Haskell functions being curried made it much easier to implement and helped me avoid a lot of the errors that I faced in Racket. Having the *"where"* clause also made it possible to reduce the number of functions needed for me.

Haskell seemed to be faster than Racket, but not as fast as Java, so it was a good middle ground between the two when it came to performance.

I think that the support and resources available for Haskell is decent enough to get certain tasks done; Rackets community is smaller, and its resources are limited.