Trevor Pinto

301394589

tpa31@sfu.ca

CMPT 383 D100

A study of Modern Programming Languages using an Approval Voting System

For this semester of CMPT 383 D100, the final project was to implement an approval voting system in three different languages. The first being a language of your choice, the second being Racket and the final one being Haskell. My language of choice was C++, and I have the following observations based on my three implementations of the task provided.

Performance:

	example100.txt	example100000.txt	example1000000.txt
Haskell	0.09s	3.893s	46.884s
Racket	0.06s	0.547s	6.743s
C++	?	?	?

From the times above, the order of fastest runtime goes as follows: Racket-> Haskell -> C++, where Racket is the fastest. The test to generate 1 000 000 ballots was completed by copy pasting example100000.txt 10 times in another text document.

Ease of development:

Haskell:

In terms of ease of development, Haskell was personally my favorite. The professor went over a Haskell implementation on November 16th, and it allowed me to understand the requirements of the project and implement the code accordingly. Overall, it took me 1 day to write both the code for the Haskell program and complete the debugging too. Moreover, it was a breath of fresh air to come to Haskell after Racket's infuriating use of brackets, pre-fix notation and dynamically typed language.

C++

The implementation for C++ took 4 days with 2 days of debugging/attempting to find a concrete solution. It was by far the least enjoyable experience out of all the three languages, and it shows in my futile implementation. Initially, the solution did not seem too bad, as this was my language of choice, however, the lack of high-level functions (like filter or map) in C++ compared to Haskell or Racket made it difficult for me to come up with a solution.

Racket

Finally, the racket implementation took 3 days to write the code, followed by 2 days of testing/debugging. A significant portion of the time was spent re-reading my notes and understanding how to use the high-level functions like filter and map, and the concept of currying. Once I was able to understand those concepts, the code itself took about 2 days to write. A particularly unenjoyable attribute of Racket compared to Haskell was the lack of automatic currying. There were many occasions where I would be lost with all the different data types and needed to write comments for myself to understand what was happening.

Quality of source code

	Number of Lines per program (line count on the side of IDE)
Racket	56
Haskell	67
C++	86

	Number of Named functions

Racket	7
Haskell	7
C++	1

In terms of readability, I personally feel that both Haskell and Racket have good overall aesthetical value, but I would give the edge to Haskell due to the immense number of brackets in Racket. However, both languages have an easy-to-understand interface, with high level languages similar to what we use in our day to day lives, which enhances its experience.

Recommendation Section:

Given this requirement of this project, along with the approval rating system and the testing for high input values, the ultimate winner for me is Racket. Although I dislike the prefix notation, its dynamic type and insane amount of brackets, it worked the best in terms of efficiency and utilized high level functions and concepts like currying and filter. However, my C++ code is a terrible representation of the language, which somewhat obscures the results. Finally, Haskell was enjoyable to create, but it lacked the efficiency when it came to high input, which is why it did not win.