Name: Alex Lai Student#: 0920158 Email: Alai02@uoguelph.ca Date: 06/04/2018

CIS*3190 - A4 Reflection Report - Sieve of Eratosthenes

I chose to complete the sieve of eratosthenes programs for the A4 legacy software assignment. The programs were coded in C, Fortran, Ada, COBOL, and Python. I began the design process by reading over the specification and getting a stronger understanding of I needed to do for the program I was to implement.

The first language I decided to write the program in was C. Firstly, I constructed the skeleton of the C program by including the libraries I needed and asking for user input for the upper limit of primes to be calculated. Next, I had to make the decision on which data structure would be most appropriate for this algorithm and decided on a boolean array since I only cared about if the number was prime or not (A binary choice). Furthermore, I figured out how to initialize the array to the size of the number inputted and set all the values to true. After, the next step was to alter the array of boolean values to set the composite values to false, giving solution to the program. Looking at the specification, I remembered that I only needed to check numbers that are less than the square root of the upper limit so I set my first for loop to start at 2 and increment until the index squared was greater than the upper limit. Next was to construct the inner loop. I checked if the array at the index was true so that I would only loop through prime numbers and then have my inner loop to set all multiples of the prime to false since they cannot be prime. After that, the main functionality of the program was complete and I was able to print out the prime numbers to ensure the accuracy of the algorithm. Moreover, I changed output from stdout to open and write to a file with the results. Finally, I used the time C library to time the execution of the program and tested with some sample inputs. C was was the easiest program to write with my extensive knowledge about the language and practice with its structures. It supported the algorithm with ease and had a very good execution time, making it a great choice for this algorithm. Also I could see C having an edge on the other languages with the vast options in type choices if dealing with larger prime numbers.

The second program that I chose to write was the Fortran program. I started using the same process as the C program. I created a basic program and prompted for user input. I had to research how to create a dynamic sized boolean array in Fortran. I found a solution using an allocate function which allocated the size of the array after the user inputed the upper limit. This required a function because Fortran requires that all variables are declared before any functionality of the program, thus requiring this work around. I wrote the main function to be as close in logic as the C program to ensure consistency in work done for an accurate execution time comparison. I then outputted the results and checked to see if the program was working correctly. Moreover, I went back to the Fortran slides and figured out how to write to a file and

implemented the code. Lastly, I referenced the professors blog on timing a Fortran function and used it to calculate the execution time of the functions runtime. Fortran was similar to C in usability and I found it to be a very usable language with support for the necessary structures, making it a great choice for this algorithm as well.

Ada was the language I chose to write the program in next. The first step in reengineering Ada was to create a simple program to read users input for the upper limit using get and put functions. I then researched how to create a array of boolean values in Ada which happened to be quite similar to the array of strings from the second assignment. Next, I initialized the array to have all true values and set 1 to false with the size that was inputted from the user. I used a declare statement to set the size and values after the user inputted the limit since like Fortran, the language requires all variables to be initialized before any computation in the function. I used a while loop for my iterations with an exit when statement to be as similar to the other implementations as possible. After implementing the same algorithm, I printed out the result and checked for consistency with the prime values to my other programs. After that I researched how to write values to a file and found a solution that requires you to convert the integer to a unbounded string and then write that to the file. Lastly, I read the professors blog post on timing a function in Ada and used the solution to compute the execution time of my program. There were not any significant barriers about Ada that made the program difficult and I found that the language supported all the necessary structures needed to complete this task. With the overall usability being great in Ada for this program and also the fast execution time that the language has makes this a great choice for this algorithm.

COBOL was my next choice for the implementation. The first step, just as the others was to create a basic program that accepts user input for the upper limit of prime numbers. Next, the hardest step was to figure out how to create an array of boolean values since it is not part of the core COBOL language by default. I read the COBOL documents the professor posted and researched online how to create this array. I found a solution that requires the occurs type that allows for array time structures. The array is dependant on the upper limit and can be manipulated by using the set function that allows for setting true and false values. I declared the value 1 for true and false for 0 for the array. I then computed the primes using the same algorithm and loop structure as the other programs. Furthermore, I researched how to write to a file and found a solution that requires you to first convert the prime value to a record type and then write the record to the file buffer. There seemed to be no solution to write an integer to a file so this seemed to be the best solution. I declared a record type in the file section under where I declare the file pointer. Finally, I read the professor's email on timing a program in COBOL since it has no dedicated way to compute the runtime and implemented his solution of using a shell script. Since the program requires user input and writes the data to a file, I removed the parts to get a more accurate runtime of the program. I hardcoded the input values

and removed the file IO, allowing me to use the shell script to calculate an accurate run time of the COBOL implementation of the algorithm. COBOL was the hardest language to implement and I found that it was the most unqualified for this program. From the lack of runtime support and the lack of dedicated boolean array types made the program have the least usability and overall the “hackiest” solution.

Python was the last language I implemented. I had used python before and was familiar with the syntax but had to do some research on some of the specifics of this program. Firstly, I asked for and accepted user input from the command line and converted the upper limit to an integer to allow me to use it when looping through the array. I did some research on how to create an array in python and found a simple solution to create the array using the size of the upper limit. Next I wrote the same algorithm to compute the prime numbers and moved on to writing the values to a file. I used the python file IO open and close statements and used a string conversion to write the primes to the file. Finally, I researched how to time a python function and used the time library to compute the execution time. There weren't any limitations to creating this program in python and found that the language had great solutions for all of the components needed. Especially the time library and ease with file IO that the language supports. The great usability of this language makes this a great choice for an implementation of this program unless execution time is a factor which would make me more inclined to use a different language.

There were quite noticeable differences between the execution times of the different languages. C, Ada, and Fortran were noticeably faster in the tests ran but they differed slightly with no clear fastest time across all tests. The slowest language was python by far. Since the code is not compiled and requires an interpreter when it runs, it requires more work to be done and makes sense to be the slowest program.

Language / Upper Limit	100000	1000000
Ada	0.002413 sec	0.014369 sec
C	0.001999 sec	0.015838 sec
COBOL	0.029606 sec	0.165154 sec
Fortran	0.004000 sec	0.012000 sec
Python	0.030824 sec	0.204589 sec