Assignment 4 – Sieve of Eratosthenes Apr. 9, 2018

Reflection Report

Mohammadamin Sheikhtaheri
0930853

This assignment was by far the most interesting one in this course since we truly see the difference between all of the languages we have learned by writing the same algorithm in each one. I have chosen to code the Sieve of Eratosthenes algorithm in C and Python, along with the 3 legacy languages we have learned: Fortran, Ada, and Cobol. The algorithm itself was very simple after I researched what it did more in detail, and this resulted in short code for each language.

I started coding the algorithm in C first as it is the language I am most familiar with. First, I created an integer array, and initialized it to every number from 0 to the inputted upper limit. Then I proceeded to implement the algorithm, which was a double for loop iterating through the array and setting the non-prime integers to 0. At the end, I printed every number in the array that was not a 0 to the file "cOutput.txt". Designing this algorithm did not take a very long time since I have a lot of experience with C and the algorithm itself was quite simple. After I finished the C portion of the assignment, it wasn't too difficult converting the same algorithm to the other languages listed.

After C, I moved on to Python, and started translating the same algorithm I wrote in C, to Python. Writing the algorithm in Python was even easier than writing it in C, which was a surprise to me since I do not have much experience with Python as a language. This goes to show that Python is by far the most usable language out of the ones I had the opportunity of working with, it only took around 20 lines of code to write the entire algorithm, whereas it took more than double that in C. As for the algorithm itself, it is structured exactly the same way as the C algorithm, I initialized an array from 0 to the inputted upper limit, used the same double for loop to iterate through the array and setting the non-primes to 0, and finally printing the array to the file "pythonOutput.txt".

Moving on to the legacy languages, I started with Fortran which I believe is the language that resembles C the most, and therefore is the easiest of the 3 we have learned to write code in (in my opinion). Since I already had knowledge from A1 (TicTacToe in Fortran), all I had to research in order to translate the algorithm fully was how to output to a file, and also how to make dynamic arrays at runtime. I had to make a slight change to the algorithm for it to work in Fortran, this change was just simply changing the for-loops to while loops since Fortran doesn't have the traditional for-loop. Declaring and using the integer array was very weird because even though it was a one dimensional array, I still had to access it with 2 indices and having one

of them being 1. It was still very easy to translate the algorithm to Fortran code and it only took around 40 lines to finish it.

The Ada program was also very easy and simple to translate after doing the Fortran program, this is because I find Ada and Fortran very similar in terms of program structure and variable types they both offer. I did have to research how dynamic arrays worked in Ada (Similar to Fortran), but other than that, the knowledge I already had from A2 was enough to successfully translate the sieve algorithm from Fortran to Ada.

I left the Cobol program for last because I knew it was going to be the hardest one to translate by far. I took a long time to study the language more, and I realized that it would be much easier to restructure my algorithm to using a sort of Boolean value to determine whether or not a number in the array is a prime number. The code from A3 definitely helped me re-design the algorithm but it took a very long time to do so, maybe even 4-5 hours. The re-designed algorithm starts with an output record that contains a variable which will be the number that is outputted to the file, then I declare the variables that I used to iterate until the maximum prime of the inputted upper limit. I then created the record that contained the array itself and the Boolean value that checked prime numbers. The Cobol portion of the assignment was extremely hard and I have to say a bit unpleasant, however, it was still a short program given that it is roughly 45 lines of code.

For comparing the efficiencies of the 5 different languages, I used the “time” command line argument which actually shows the execution time of the program. To do this, I had to remove the user input, and hard code the upper limit so the time it takes for the user input is not taken into consideration. I recorded the times it took for each program in a table below, and used 500, 50000, and 1500000 as the upper limits for testing. It is very clear that C and Ada are the most efficient algorithms, and it is also very clear that Cobol and Python are the most inefficient algorithms, with Fortran being somewhere in the middle. This goes to show that Python code may be the easy to write, but it is very inefficient. It is not a surprise that C is the most efficient, but I was surprised seeing how fast Ada really was and how slow Cobol was. Overall, this assignment certainly expanded my knowledge on legacy languages and their similarities and differences to modern languages such as Python.

Upper Limit	**C**	**Python**	**Fortran**	**Ada**	**Cobol**
500	5 ms	24 ms	7 ms	7 ms	48 ms
50,000	10 ms	58 ms	14 ms	13 ms	59 ms
1,500,000	66 ms	1015 ms	104 ms	74 ms	124 ms

Execution Instructions

C Program:

Compilation: gcc -Wall -std=c99 sieve.c

Execution: ./a.out

outputFile: cOutput.txt

Python Program:

Compilation & Execution: python sieve.py

outputFile: pythonOutput.txt

Fortran Program:

Compilation: gfortran -Wall sieve.f95

Execution: ./a.out

outputFile: fortranOutput.txt

Ada Program:

Compilation: gnatmake -Wall sieve.adb

Execution: ./sieve

outputFile: adaOutput.txt

Cobol Program:

Compilation: cobc -x -free -Wall sieve.cob

Execution: ./sieve

outputFile: cobolOutput.txt