Ebad Babar

0954164

Cis 3190

Assignment 4

Sieve of Eranthoses

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1000 | 10000 | 100000 |
| C | 2.9 ms | 40.5 ms | 70.4 ms |
| Python | 1.3 ms | 31.2 ms | 62.7 ms |
| Fortran | 1.2 ms | 36.0 ms | 65.0 ms |
| Ada | 2.5 ms | 37.4 ms | 63.7 ms |

Average for 10 tests

C:

Compile = gcc -wall -std=c99 seive.c

Run = ./a.out

C was probably the easiest language to use make this program in. However, looking at the results it is easy to see how inefficient C is. Being able to dynamically allocate memory and structs made this program easy to make.

The algorithm starts with the checking for file output and opens it. The program then asks for the user input and starts the timer. It sets a Boolean array of user input size to true. An I starts at 2 for I \* I < n to check for prime numbers of 2. If the number, “I” is prime the program runs another loop turning all the multiples of that number to false. I run another for loop to print all the prime numbers to a file. Finally, I perform call the end time and calculate the total time the program ran. To end the program end 0.

Assumptions: file output.txt exists, user inputs valid data.

Python:

Compile and run: python Seive\_of\_Eranthoses.py

Python had the least amount of coding out of all and according to the data was the fastest. Being able to freely mess around with the variables was probably the best part about python. It is also the easier to go back and maintain.

The algorithm starts with asking for the user input. The program then starts the timer. Sets the array to true. Follows the same patter as C except uses a while instead of a for loop. Outputs the data to the file in a for loop and then ends

Assumptions: the user is willing to re run the program over and over, file exists, and user inputs valid data

Fortran:

Compile: gfortran SeiveFortran.f95

Run: ./a.out

Fortran was not the fastest not the easiest to code in. But it also wasn’t the slowest or the hardest. Being able to redeclare variable half way through the program was extremely useful as it allowed for me to create an array of proper size rather then have a very large, static array and hoping the user enter a value lower. Also, Fortran allowed for specific file type and status which I though was cool.

Algorithm: the program starts the with declarations. It asks for user data and if the user enters 0 a flag is set, and the program doesn’t run. Otherwise, continues. I set the entire array to true as in the previous programs. I follow the same algorithm as before and set the non-prime numbers to false. In a for loop is print the data to the file. Finally, I calculated the data that the program ran for.

Assumptions: The files output exists for the data to be stored in. The user inputs valid data.

Ada:

Compile: gnatmake sieve.abd -> gnatbind -x sieve.ali -> gnatlink sieve.ali

Run: ./sieve

Ada was a hard language to code in and according to the data was second slowest before C. took 70 lines of code without much error checking to in ada to do this program, where as in C with error checking it took 50 lines, and 30 in python. What I did like about for ada was being to declare everything at the start and redeclare or update data later if need be.

Algorithm: the program asks for the user data until the user enters a value over 0. The program set the array to true and follows the same basic algo as C and python and sets the non-primes to false. The primes are then put inside the file through a for loop.

Assumptions: The file output exists, the user inputs valid data

Cobol dnf:

Compile: cobc -free -x cobSeive.cob

I didn’t finish the program in Cobol as I couldn’t figure out how files work.

Algorithm: I start the program by asking for user input for the max range of 0 to end. If he user enters 0 the program will end. If the user enters a value greater then 0 the program continues. The algorithm is the same as in the previous programs, however, I could find Booleans in this language and therefor had to use 0’s and 1’s. Another problem I had with this language was how to dynamically allocate space. I have an array of 1000000000 manually defined, the program will break if the user enters a value higher then that.

Assumptions: The output file exists, and the user inputs valid arguments

Sources:

Time in Fortran: <https://gcc.gnu.org/onlinedocs/gfortran/CPU_005fTIME.html>

Files in Fortran: <https://www.tutorialspoint.com/fortran/fortran_file_input_output.htm>

Time in Python: <https://docs.python.org/2/library/time.html>

Compile in Ada: <https://gcc.gnu.org/onlinedocs/gcc-6.3.0/gnat_ugn/Running-a-Simple-Ada-Program.html>

Time in Ada: <http://www.iuma.ulpgc.es/users/jmiranda/gnat-rts/node29.htm>

Files in Cobol: <https://www.tutorialspoint.com/cobol/cobol_file_handling_verbs.htm>

Time in Cobol: <https://www.ibm.com/support/knowledgecenter/en/SSQ2R2_9.0.1/com.ibm.etools.cbl.win.doc/topics/rlpsacce.htm>