Kevin Pirabaharan 2017-04-05
ID: 0946212

CIS 3190 A4: Fun with Legacy Languages Handout –Option 2

The first step of this assignment was to find or create and algorithm for the Sieve of Atkin. This program is supposed to find all the prime numbers that are less or equal to the upper bound that is specified by the user. The way I went about it was to create a Boolean array the size of the limit and then make all of the indexes false. After that I would have a series of loops that would take values and multiply them together and see if there were an odd number of solutions. If there were, then those numbers would be a prime number and the corresponding index would have its value changed to true. Finally, I would have a for loop read through the array again and print out only the true indexes (which are the prime values). Numbers 2 and 3 are hardcoded because if the loop started at 2 then there were errors. The first implementation I started with was C because it was my most familiar languages and the for loops were a lot easier and simpler in this language. One error I stumbled upon was that for some reason negative values would show up for unknown reasons during testing, but after doing some reading I learnt about unsigned characters which allowed for only positive numbers and that fixed that whole problem. The hardest part was trying to find the specific libraries I would need to time the runtime of the program so that it would work on both my computer and the server as some of the methods would just return a timing of 0 on the legacy servers. All in all, the C component took about an hour to code and test to make sure it works. I never malloc' d the arrays or used anything like that so once you type a number past 10 million it will return a segmentation fault. Now that I look back I could have easily added this functionality without changing anything else in my code because it was all done in main.

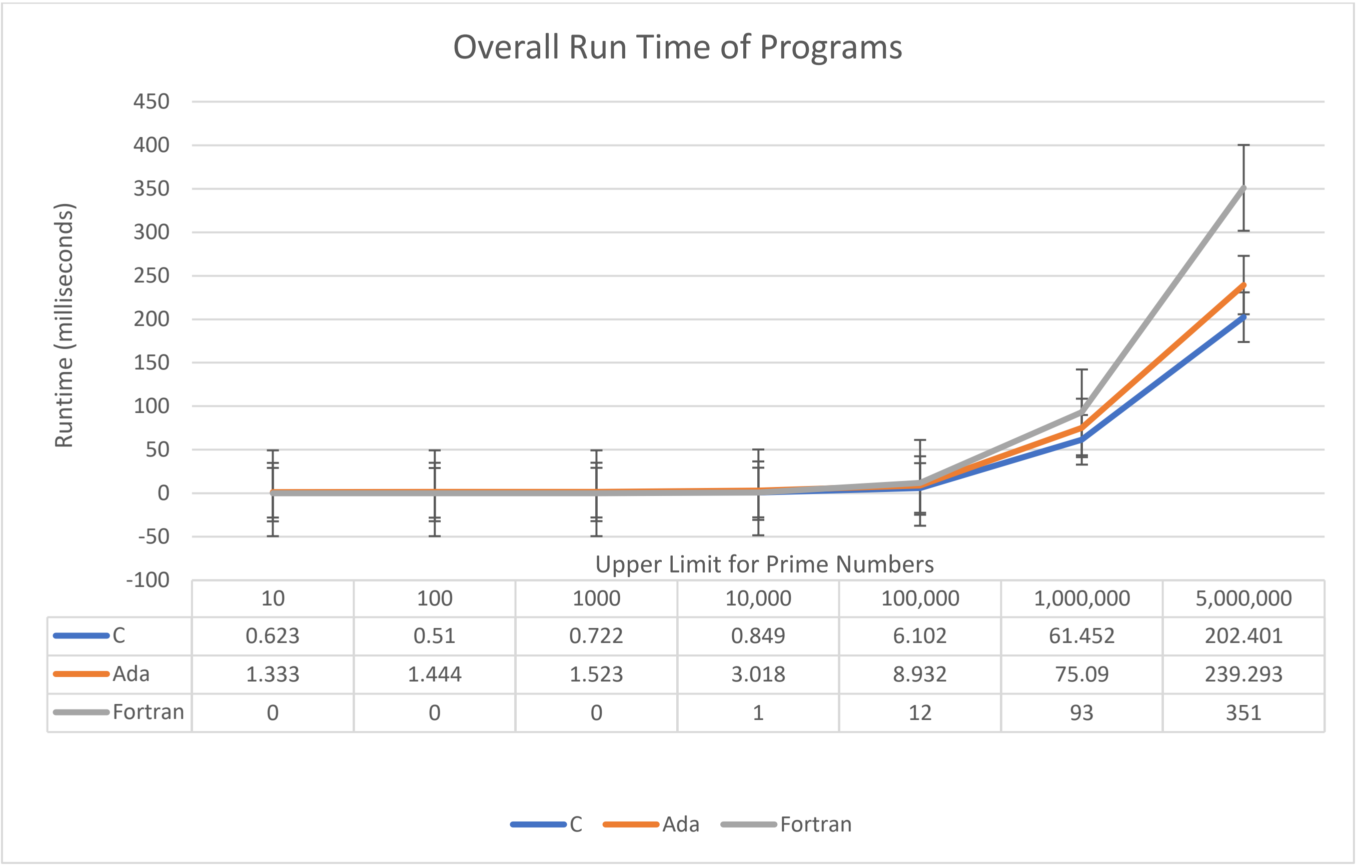
Next up was taking this algorithm and coding it in Ada, I chose to do Ada next because it was the most similar of the two remaining languages to C. I was a bit rusty in so I had to have my old assignment to see how the syntax for for–loops and other aspects were different. Also for loops were different in this language so I had to use while loops instead so I can start at a certain value and increment in different ways (like squaring a number). The rest of it was simple and it just took a little longer that C to code. The Ada program worked without a hitch and returned some interesting results for timing. The timing code was taken from the Professor' s blog on how to time Ada programs. I didn' t know how to use memory allocation for the array in this language so this program also may run into error when the number gets really really high. I' m glad that my program didn' t require the use of square rooting values as I ran into a lot of problems using the math library in Ada during practice coding sessions. Ada was a fun experience to code in, maybe because all I had to do was essentially copy and paste my C code and just change the syntax up but it was still a lot better than the Cobol assignment that we had to do previously. There were some bumps a long the way though, Ada doesn’ t overwrite files that it outputs and so I had to “create” the file just in case so it can be deleted right away and created again but this time with the output. I also had to multiply the seconds by 1000 to milliseconds which may have also added a margin of error to the execution time.

Finally, it was time to implement the algorithm in Fortran which was more similar to Ada than to C so that worked out pretty well. I again had to google how the syntax looked like and uses past assignments to remember how to code in Fortran. This time it was a lot easier to understand Fortran code and code in it, maybe it’ s from the practice but this time it went a lot smoother. During this process, I realized that we needed to output the primes and execution time to a text file and not to standard output and I had to go back and change that in all 3 programs to get it to work. Writing the code was simple enough but I got so many errors for certain values. These errors were because I was going out of array bounds somewhere in the code and this was probably due to the structure of how I had my do while loops laid out. So, the only way I could think of fixing this was to learn memory allocation for the array and then just change the loops altogether which kind of skews the runtime of the program but at least it works for most values now. But after that I had working code that ran in Fortran. This code could handle higher values which made it better in some respects to the other implementations. Then came the harder parts where I had to add if statements because it was still outputting multiples of 5 but after a lot of testing the programs all worked.

After coding this algorithm in 3 different languages, my opinion has stayed the same throughout this whole course. I still think C is a lot easier to code on and has a lot more usability with the use of structs and more modern for loops. I had to make the code a lot longer because there weren’ t for loops as usable as the ones in C and in Fortran’ s case, there weren’ t even any for loops. In addition, the steps were also a lot more tedious in timing the execution of the program in the other languages. Especially in Fortran, I had to find a work around because for some reason the on that the prof referenced wasn’ t working on the servers for me. Fortran gave me so many headaches and it was the reason I had to submit a day late, and Ada wasn’ t hard but it took longer than C to write and polish up. However as hard of a time I may give the other legacy languages, they weren’ t that unusable. They just seem like older (more primitive) versions of the C we’ re using today and that’ s more or less true.

Here is a chart and graph of the testing that I have done with all my programs. At the bottom is the chart with all the values, runtime in milliseconds, I got for various input values I gave them. Above that is a visual representations so you can see the exponential increases that all the programs experience as you give larger and larger inputs. There are also markings for my margin of errors as sometimes I' d get results that vary by a bit for the same input limit (I chalk that up to me just giving it too many commands repetitively during testing.) As you can see from the data you can see that Fortran starts of the fastest as the first few times are so small that they show up as 0 milliseconds, however, it takes a steep increase in execution time as we get into the 6 digits. Ada follow Fotran by slowing down at a decreased rate but it still takes longer than the C code. According to my findings, the C code is the fastest at find prime numbers as you scale up, while Ada and especially Fortran are quick to run when input values are much lower. This is probably because I kept C in mind when I found this algorithm and used it. The other 2 programs were more or less a translation from C and that probably slowed it down. But as for now C seems to win both usability and efficiency for this assignment.

I had a great time learning about legacy languages and I hope I will be able to use this knowledge in various things as I grow older and start doing harder work.



	10	100	1000	10,000	100,000	1,000,000	5,000,000
C	0.623	0.51	0.722	0.849	6.102	61.452	202.401
Ada	1.333	1.444	1.523	3.018	8.932	75.09	239.293
Fortran	0	0	0	1	12	93	351