Reflection Report A4

For this final assignment, I decided to do the first topic: calculating e using the ALGOL 60 algorithm. I had to translate the algorithm into Python, C, Fortran and Ada, and then save the results and put it into a file for comparison to see which language was the most accurate.

To begin, I decided to translate the algorithm into Python. I have been using Python for over 6 years, so I am most comfortable translating pseudocode into it. I also find that once I have the code in Python, translating it into other languages is much easier. In addition, a massive benefit about working with Python is that the syntax translates very well into pseudocode. Dealing with file IO is also intuitive in Python, making that part of the assignment trivial. However, I did run into some issues with how the algorithm was written. Since all Python arrays start at 0, I had to slightly change up the algorithm so that some array that began at 1 now begin at 0. For example, coef starts from 2 to m, so I had to modify that.

Next, I translated the Python code into C. The code itself is very similar to the structure of the Python code, but in this case, I had to manually allocate memory for the array of digits. The reason I had to do this is because the user defines how many digits of e they want to be during runtime. For Python, this is not problem it allocates this memory during runtime. In C the size of arrays must be declared beforehand, and we don't know how many digits we need to calculate. This just meant that I had to make sure to free the memory I allocated at the end of the main loop of the program. Once I got that done, I simply followed the Python code to create the C code. A benefit to using C is that it is more low level than Python, so it tends to run a bit faster. However, coding it is more difficult because of the need to manually allocate memory. Another note is that the logarithm function is not default with standard C, so I had to make sure that when compiling the code I included that package.

After the code for C was done, I moved onto Fortran. In this case, I used my C code as a point of reference because it is lower level. I ran into a similar problem when I was developing in C and I found I needed to manually allocate memory here as well. A benefit of Fortran compared to C is that I found allocating memory in Fortran to be easier. Something about the syntax made it clearer, possibly because in C you need to use sizeof rather than just putting in the array into the allocate call. Like C, this just meant I had to free up the memory when I was done. The basic format of the program is structured like C. In this case, I could follow the original algorithm in terms of the size of the arrays since Fortran lets you declare the start of an array wherever, but I decided to just follow the structure I used for the other programs since I already had that code.

Finally, I wrote the Ada program. I had more trouble than I expected writing in Ada compared to the other three languages. I took my function from the 2nd assignment for checking if a file exists, since I needed that functionality here for writing the result to a file. The algorithm for calculating the number is the same as the other programs, but I had trouble figuring out how to declare an array of an unknown size at runtime. I figured it out eventually by making a nested declare statement with the coef inside the calculation. Compared to C and Fortran, declaring the size of an array during runtime was a massive limitation of Ada since it is nowhere near as intuitive. However, writing to the file was easy since I already did that

functionality in A2. In addition, having exceptions to check if the file was properly opened or if it needs to be created was a huge plus to finishing that portion of the assignment.

The table below has the following results of running each program. This test data was obtained by calculating e to 100 decimal places, then running each file against each other with t linux function diff to see if they returned different results. In the first row, we have the language that was used. The second row contains the results of the calculation of e to 100 decimal places. Finally, the third column has the approximate runtime of the calculation in seconds for 10,000 digits. The calculation of the runtime was done on my personal Linux VM, however I verified with the school computers and the results were very similar. Note that I stopped the timer once the results were calculated and a prompt for the filename was displayed.

When comparing the different languages, there were two main things I noticed aside from ease of development. In terms of accuracy, in each case the diff function returned nothing after adding in a newline character to each file. Therefore, if you are simply looking to translate the ALGOL 60 pseudocode into a functioning program, it will not yield any difference. However, there was a significant difference in runtime for the calculation of 10,000 digits. Python ran significantly slower than all the other languages. This is most likely due to it being an interpreted rather than compiled language. Fortran and C ran at about the same speed every time, and Ada was noticeably quicker than every other program. These are lower level languages, so it makes sense for them to be faster than Python. Therefore, if taking into consideration that the accuracy of each language was not a factor, Ada is the clear winner in terms of computing mathematical algorithms simply because it was the fastest. Python provides ease of development, but the speed of the program will be noticeably slower. Fortran and C are comparable, but I would prefer writing in Fortran because allocation of memory is easier.

Fortran	**Ada**	**C**	**Python**
2.718281828459045	2.718281828459045	2.718281828459045	2.718281828459045
23536028747135266	23536028747135266	23536028747135266	23536028747135266
24977572470936999	24977572470936999	24977572470936999	24977572470936999
59574966967627724	59574966967627724	59574966967627724	59574966967627724
07663035354759457	07663035354759457	07663035354759457	07663035354759457
13821785251664274	13821785251664274	13821785251664274	13821785251664274
~ 0.5 seconds	< 0.5 seconds	~ 0.5 seconds	15 seconds