Name: Alex Lai
Student Number: 0920158
Email: alai02@uoguelph.ca

CIS*3190 - A2 Reflection Report - Ada Word Jumble

I chose implement the second program, the word jumble for my Ada assignment. The program takes in a users input asking them to enter in strings of characters on each line and then the program searches a dictionary with all anagrams of each word and printing out words that are found.

In order to learn this new language and build the program, I decided to break the process up into smaller steps and work my way towards to a fully working Ada program.
The first step that I took was to read the introduction to ada pdf and get a brief overview of the language and syntax. Next, I created the word jumble program file and imported some basic Ada packages to be able to handle strings and text io. I decided to implement the function inputJumble() first since it requires no other function to be implemented. I first found out how to input a string from the command line and then implementing a loop that stores the input on each line in a new string and store it in a list. I then decided to implement inputJumble() as a procedure rather than a function so I would be able to retain the length of the list in a variable after the procedure ends. I had to make a decision on which data structure to use for the list and I chose a array type of unbounded strings because i found it to solve the problem perfectly. After that, I then decided the next step was to implement a factorial function to get the number of factorials for a length of a string which would later help me with the size of the anagrams list. I decided this would be a great function and not a procedure since I only care about the integer the function produces and nothing else. I looked into the for loop structure for Ada, then wrote the small function and printed the result of the factorial function to make sure it was working properly. Next, I chose to implement the generateAnagrams() function since it was the next logical step in building the program. I googled the permutation algorithm and decided it was best to break the generateAnagrams() function up into three parts. First was a permute function that was recursive, adding each anagram to the list of anagrams and second a swap function to be able to swap characters in each string. I figured that it was best to implement these two functions inside of the generateAnagrams() function so that I would be able to insert each anagram directly into the list that I defied in the generateAnagrams() function. During this process I discovered that you cannot easily swap characters in an unbounded string. I found a work around by temporarily creating a fixed length string and swapping the characters in that and after setting unbounded string to the value of the new string. This required another Ada package import in order to use these fixed string functions. I tested the generateAnagrams() by printing out each anagram created to stdout and confirming that it was working properly. My next challenge was to find a way to generate anagrams for every input string entered. After some research I discovered that you can use a declare block where I declare the length of the anagrams list only after I get all the permutations of it. This allowed me to loop over each word

entered, storing the anagrams of each in a list so i'll be able to pass each list into the findAnagram function. After finishing the generateAnagrams() function I chose to implement the buildLEXICON() function since it is needed in order to write the findAnagram() function. I began with reading about file IO and figuring out how to store the dictionary the prof gave us in a data structure. My first thought was to use my wordsArray data structure to store each word in the dictionary but found the Ada package for ordered set containers to be a much simpler solution. I was then able to fill the set up with the dictionary words and be ready to pass it into the findAnagram function. Next, I was able to implement the findAnagram function with ease since the container package has a contains() function to quickly search the set of dictionary words. I tested the output using the suggested tests for the assignment and after some testing and small bug fixes, I was able to output the correct words from the jumbles. I noticed that when a jumble contained two of the same letters, it would then print the word out twice since I was storing the duplicate permutation in the anagrams list. I decided to fix that in the permute function by first checking to see if the anagram was already in the list before inserting it. After that, I did some code clean up and commented the whole program for readability. Finally, I was able to test the program and ensure that everything was working smoothly.

I do believe Ada is a suitable language to solve the problem but I also think there are better languages for this sort of problem like python or javascript. Before most modern languages, Ada would be a fine language to solve this problem but today the syntax differences and amount of packages and imports needed seems unnecessary since languages like javascript make working with strings very easy. Also the learning curve is much steeper for a language like Ada because of its very particular and unique syntax. Overall, the program is very straightforward and I think it was a good program for an introduction to Ada.

During the coding process of the assignment, I found that the ordered set data structure the most useful tool within Ada. Both inserting and checking if a string is contained in the structure were just single lines of code and simplified the dictionary aspect of the program greatly. Also the string functions in Ada were very helpful in solving this problem. Another part of Ada that I found useful was its error handling. Whenever I ran into any sort of error, the feedback was extremely informative and sped up the debugging process.

I found that the syntax of Ada the worst part about the language. I found that it was very dissimilar when it came to things like declarations, assignments, arguments, ect. It wasn't a huge deal and became easy over time but I disliked the small differences and just found it made learning Ada not as interesting compared to Fortran. I Also believe if someone not fluent in the language was to read some Ada code, they might understand certain things incorrectly. For example, the equals operator is a single = which most people would understand as an assignment operator.

I didn't find any limitations to Ada for this assignment and found that each task could be solved without too much complexity. Ada is definitely a powerful language with tons of functionality from different packages and can be useful for particular problems even in 2018.