Ada — Jason Nguyen (XXXXXXX)

Learning Ada was somewhat of a paradigm shift from other languages like Python, JavaScript, and to some extent, C. It is probably the strictest language I've ever had the pleasure of learning, and it helped me establish a sense of appreciation for extremely strong typing and super strict compiler checks. As much as parts of it were annoying, Ada will hold a special place in my heart.

Some of the benefits to using Ada are that it is strongly typed (safe!), variables can be described in detail (i.e. selecting an exact range), and that the compiler is very strict on sloppy code. I might have taken a while to program the scrambler, but I knew that when I did, it was very robust. Some of the downsides to Ada are that it is a big departure from most languages people are familiar with and it can be hard to learn as a result. I found that one of the most underrated difficulties of this project was how little documentation there was for many functions.

Ada was not a suitable language for this assignment. Most can agree that Ada's crux is how it handles strings, which is probably the biggest differentiating factor between it and other languages. What immediately strikes people's eyes is the fact that the default string type, fixed, must contain exactly the number of characters it says it contains. That means if one were to declare and initialize a 10-character string, "HELLOWORLD", you would not be able to re-assign "HELLO" to it, as "HELLO" only contains five characters. For an assignment that is heavy on string manipulation, a better language could have definitely been chosen for that. Some people used **`unbounded_string`** to cope.

I personally found **`unbounded_string`** to be tedious and annoying. As such, I did not use any in my project. The only two instances where one would want to use **`unbounded_string`** were during file reading and filename input. For the latter, most UNIX systems do not support a path greater than 4095 characters, so I made my fixed filename string very long to accommodate this. For the former, I used a **`declare`** block at line 76 in my file reading loop. Every time I would call **`Get_Line()`** to pull in a new line from the file, it would be assigned to an entirely new string in this declare block, thus allocating an entirely new string variable every time. Whereas an **`unbounded_string`** solution would be creating **a single string** outside of the loop and using it **many times**, my solution was to create **many strings** in the loop, each used **only once**, thanks to the block scope used in my file loop.

Ada's object attributes were likely the redeeming factor of the language in this assignment. When working with strings, I would use **`'Length`** to instantly find the length of a given string (without using **strlen()** or having to use a dedicated length variable). Then, when I would need to find the upper and lower bounds of a string, I would use **`'First`** and **`'Last`** and it would make character iteration a breeze. As such, I found accessing a string's characters to be a breath of fresh air in Ada. There is a myriad of attributes in Ada that I haven't used, such as **`'Size`** and **`'Range`**.

In addition to that, another Ada structure I really liked was how it lets you slice strings using two indexes (e.g. Str(5 .. 23)). This was the bread and butter of my text parsing algorithm as it let me walk through the string using any subset of characters I wanted. This was what fueled my greedy word growing algorithm (shown on the next page). Using Ada's string slicing capabilities coupled with the attributes feature mentioned above made for some extremely intuitive code that I am proud of.

In programming the word scrambler, I took a bottom up approach. As I like rigorously testing every bit of a program, I figured it made more sense to program (and unit test) isWord(), scrambleWord(), and randomInt() before everything else, as they can be tested independently of each other. It was only after I finished programming these individual functions did I start putting them together into the big picture (in processText()), along with numerous amounts of regression testing.

This was the overall program flow at the time of submission:

1. Ask the user for a file to open and parse, refusing to advance until a valid one is specified.
2. Open the file and print out each line, unedited (for comparison purposes).
3. Re-open the file for parsing.
4. Tokenize each line into words using two string indexes to ‘greedily’ extract the largest valid word from the current seek position, which will be sent off for scrambling.
5. For each word tokenized using this process, send it to the scrambleWord() function in order to scramble the middle of the word in-place.
6. Print the resultant scrambled word out.
7. After each word is tokenized, print out as many non-alphabetic characters as possible until a new valid word is found (and can be greedily extended).
8. Advance the two string indexes (the seek position) one character past the last character printed in order to go back to step 4 and extract the next word.
9. Repeat steps 4 – 8 until there is nothing left to read in from the file.

A more detailed synopsis of this process can be found on line 86.

Overall, learning Ada was a positive experience. I learned some programming quirks inherent to Ada that I have not seen in other languages and I learned to appreciate extremely strict type/compiler checking.