* *A brief description of notable obstacles you overcame.*

There were several errors and obstacles I encountered which taught me more about the programming environment I was using. The first obstacle I had to overcome was learning how to correctly using the File I/O since the syntax was something I was working with for the first time. I initially kept getting an error when I was trying to read in the file and I realized that for XCode the full path name should be included whereas g32 accepts simply “words.txt.” These two syntactical differences taught me how these two environments operate differently.

I also had to overcome obstacles regarding the implementation of the functions. I struggled to create the shuffleChars function and realized that it would be easier to approach the permutations step by step. First step was to create a printPermutations function which can find all the anagrams/permutations of an inputted word. I used the pseudocode given in the FAQ to implement this function and then used box trace diagrams to understand how the loop worked. I then implemented the loop using a recursion and thus this function was co-recursive. My strategy was then to replace the ‘print’ part of the code with a cross-checker that did the following things: checked if the permutation/anagram existed in the dictionary, ensured this word did not already exist in the results array, ensured the results array was not greater than MAXRESULTS. Once these checks were completed, I added the word to the results array and incremented its size.

Another tricky obstacle to overcome was memory and efficiency since the program must complete its task whilst taking up no more than 8MB. This taught me the tradeoff between efficiency and memory since when I tried to create algorithms which would shorten the dictionary array or create an array of all possible permutations my algorithm would eat up too much memory and crash on Linux. After trying several methods I realized the best way to do it was to go through each anagram and check it against the dictionary.

* *A list of the test data that could be used to thoroughly test your program, along with the reason for each test. You do not have to include the results of the tests, but you must note which test cases your program does not handle correctly.*

Test data set 1:

These tests were carried out with the provided words.txt to ensure all lengths of words worked and numbers. The following words were inputted into the program and worked correctly.

One match only: “AAA”, “a”, “1st”, “10th”

One letter words: “a”, “b”, etc.

Two letter words: “at”, “as”, “so”, “is”, “ok”

Three letter words: “rat”, “one”, “nun”, “you”, “y’s”, “9th”, “bar”

Four letter words: “brag”, “bank”, “trap”, “bash”, “pass”

Five letter words: “sunny”, “Wills”, “yules”

*More than 2 seconds taken*

Six letter words: “muzzle”, “zygote”, “zeroes”, “UNESCO”

*More than 10 seconds taken*

Seven letter words: “electro”, “America”

*More than 50 seconds taken*

Eight letter words: “follicle”, “atrocity”

*More than 10 minutes taken*

Nine letter words: “astronaut”

*More than an hour taken*

Ten letter words: “regardless”

20 letter word: “electrocardiogram”

(Note: the above word took too long to process and was essentially unsuccessful, i.e. the program was unable to handle it).

Jumbled up words: “pto”, “ohp”, “ermo”, “yeob”, “lnul”, “erot”,

Non-existent matches: “uni”, “lis”, etc.

Test data set 2: (main routine provided)

These tests were carried out by creating custom dictionaries. Their purpose was to ensure that revealOutcomes and makeDictionary were working correctly with regards to the implementation of MAXDICTWORDS and MAXRESULTS. To test this out, both these global variables were set to a much smaller value and dictionaries/results having larger sizes were passed. The custom dictionaries were also used to ensure that repeats were not added into the results array (i.e. each anagram was unique).

A screenshot of the main routine used in this testing is provided below:

Text

Description automatically generated

The above is an example of a word which has repeats and whose MAXRESULTS is smaller than the size of matches and thus only the first MAXRESULTS matches are printed out as expected. Several main routines like the one above were used to thoroughly test the program however for conciseness they aren’t included.