**Obstacles**

One of the more general obstacles I had was simply rethinking how to code repetition using recursion. On default, I would want to make a for loop or a while loop, so to make it easier, I began to write forloops and then I would convert them into recursive statements. After I got used to it, I was able to write my code recursively without this added step. Another step I had trouble figuring out was how to create a recursive algorithm without having a parameter to keep track of where I was. For this, I wrote helper functions with the added parameter. I also found difficulty in updating my variables appropriately so that it didn’t get incremented too many or little times, but printing the variable out intermittently helped me keep track of them.

**Test Cases**

1. ***Test createDict and viewAnswers***

dictfile.open("words.txt");

if (!dictfile) {

cout << "File not found!" << endl;

return (1);

}

nwords = createDict(dictfile, dict);

viewAnswers(dict, nwords);

This code segment tests the basic case of **createDict** copying the values of the dictfile with the use of the helper function into an array and **viewAnswers** printing out all of the values.

Variations

1. I tested the code with an empty dictfile and made sure it outputted nothing
2. I tested viewAnswers by inputting a larger value than MAXDICTWORDS for the size parameter to make sure it would print the right number of words
3. I tested createDict by making MAXDICTWORDS smaller to make sure the base case worked

//maxresults = 20

viewAnswers(results, MAXRESULTS);

viewAnswers(results, 2);

//maxresults = 2

viewAnswers(results, 3);

I tested **viewAnswers** with my results array as well. Here I made sure that if size parameter was larger than the number of results, the remaining empty lines would be printed without “Matched word”. I tested a size with a smaller amount of how many elements there were to make sure the base case worked. Finally I tested a size that was larger than MAXRESULTS to make sure that the constraint was implemented appropriately.

1. ***Test characterMixer and helper functions***

cout << "Please enter a string for an anagram: ";

cin >> word;

int numMatches = characterMixer(word, dict, nwords, results);

if (!numMatches)

cout << "No matches found" << endl;

else

viewAnswers(results, MAXRESULTS);

To check **characterMixer** and it’s helper functions**,** I utilized the code in the anagrams.cpp file provided for us by Professor Ambrosio. As I progressed through the code I would usually add print statements so I can see the progress of my functions as well.

Inputs

1. “rat”

I first started with a shorter word. With this I tested my **Loop** to make sure that all permutations were found. Later on, to test longer words I implemented a counter variable and compared it against the number of permutations it should have found.

I also used this word to test my **mixer** function, which actually checks the permutation against the dictionary and adds it to the array. In turn, this tests my **check** function, which will recursively run through the given array to find matches of the string.

1. “book”

For this word, I again first tested it by printing out all permutations of it. When I realized that book was printed twice because of the doubles o’s, I implemented the check function.

After this, I printed out the results array and ensured that book was only added once which means that check would be working correctly.

1. “anagrams” & “additiona”

I tried to use longer words on g32 to make sure my stack size was not being exceeded. It took much longer than the shorter words but in the end the correct answer was outputted.

“additiona” also made sure that when there were no matching words, nothing was printed.

1. “--” & “33”

I wanted to make sure that my code ran correctly with truly ANY string so I first tested them without adding to the dictionary file and ensured that it didn’t find any matches, and then I added both to the dictionary file and made sure it found a match but also that it registered not to print twice because of the double characters.