1. The biggest challenge of the project was to implement the function that checks if a word is an anagram. In the first place, I thought of sorting and then comparing the dictionary word and the target word. I tried to implement the function using a nested for loop. However, I found it difficult translating the nested for loop into recursive methods. I even created three helper functions, but none of them helped me achieve what I wanted because I couldn’t use any pass-by reference in a recursive function. After spending hours on trying to sort the words, I decided to try a new approach. I found that rotating the words can help check every letter of the two words, even double letters. I created three helper functions: one for examining the letters of the two words recursively, one for deciding whether the dictionary word is a result or not, and one for returning the number of results. It also took me a while to think of how to stop the recursive check of letters. I noticed that anagrams must have the same length. If they are the anagrams and I cut the letters one by one off from them, their sizes should turn zero at the same time. If they are not anagrams, when one word’s size turns zero, the other must have letter(s) remaining. To check if the words have the same letters but different orders, I allow the letters of a word to rotate left so I can compare the target letter with the next letter in the dictionary word.

2. string results[MAXRESULTS];

string dict[MAXDICTWORDS];

ifstream dictfile; // file containing the list of words

**int** nwords; // number of words read from dictionary

string word;

//check if the dictfile can be read in and examined word by word

//check the number of dictionary words loaded

//let the user enter a word for check

//find all results and the number of results

//reveal the results using the function revealOutcomes

dictfile.open("/Users/clairez/Desktop/dictionary.txt");

**if** (!dictfile) {

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

**return** (1);

}

nwords = makeDictionary(dictfile, dict);

cout << nwords << endl;

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

cin >> word;

**int** numMatches = shuffleChars(word, dict, nwords, results);

**if** (!numMatches)

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

**else**

revealOutcomes(results, numMatches);

//make the dictionary bigger than MAXDICTWORDS = 30000

//check the number of dictionary words loaded (should not be greater than 30000

//let the user enter a word for check

//find all results and the number of results

//reveal the results using the function revealOutcomes

dictfile.open("/Users/clairez/Desktop/dictionary.txt");

**if** (!dictfile) {

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

**return** (1);

}

nwords = makeDictionary(dictfile, dict);

cout << nwords << endl;

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

cin >> word;

**int** numMatches = shuffleChars(word, dict, nwords, results);

**if** (!numMatches)

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

**else**

revealOutcomes(results, numMatches);

//regular test consisting of 3 anagrams

//reveal the results using the function revealOutcomes

string results1[MAXRESULTS];

string exampleDict[] = { "art", "tar", "rat"};

**int** numResults = shuffleChars("art", exampleDict, 3, results1);

assert(numResults == 3 && results1[0] == "art");

revealOutcomes(results1, numResults);

//test consisting of words of different sizes

//no anagrams are included, so numResults should be 0

//no need to reveal the results

string results2[MAXRESULTS];

string exampleDict2[] = { "abcd", "efgh", "ijkl","mno"};

numResults = shuffleChars("abc", exampleDict2, 4, results2);

assert(numResults == 0);

numResults = shuffleChars("", exampleDict2, 4, results2);

assert(numResults == 0);

//test consisting of longer anagrams

//all words should end up in the results array

//reveal the results using the function revealOutcomes

string results3[MAXRESULTS];

string exampleDict3[] = { "abcdefg", "gfedcba", "cbadgfe","agbfced"};

numResults = shuffleChars("gedfbca", exampleDict3, 4, results3);

assert(numResults == 4);

revealOutcomes(results3, numResults);

//test consisting of very long anagrams

//all words should end up in the results array

//reveal the results using the function revealOutcomes

string results4[MAXRESULTS];

string exampleDict4[] = { "abcdefghijk","kgfiejdcbah","hcibadkgfej","jkaighbfced","ghakbcfdeij"};

numResults = shuffleChars("gekdfjbcahi", exampleDict4, 5, results4);

assert(numResults == 5);

revealOutcomes(results4, numResults);

//test consisting of many anagrams

//numResults should not exceed MAXRESULTS

//reveal the results using the function revealOutcomes

string results5[MAXRESULTS];

string exampleDict5[] = { "abcd","abdc","acbd","acdb","adbc","adcb","bacd","badc","bcad","bcda","bdac","bdca","cabd","cadb","cbad","cbda","cdab","cdba","dabc","dacb","dbac"};

numResults = shuffleChars("abcd", exampleDict5, 21, results5);

assert(numResults == MAXRESULTS);

revealOutcomes(results5, numResults);

//test consisting of one letter anagrams

//reveal the results using the function revealOutcomes

string results6[MAXRESULTS];

string exampleDict6[] = { "a", "a", "a"};

numResults = shuffleChars("a", exampleDict6, 3, results6);

assert(numResults == 3);

revealOutcomes(results6, numResults);

//test consisting of words of the same length

//no matching words

//reveal the results using the function revealOutcomes (prints nothing)

string results7[MAXRESULTS];

string exampleDict7[] = { "yes", "how", "tub", "why"};

numResults = shuffleChars("ooh", exampleDict7, 4, results7);

assert(numResults == 0);

revealOutcomes(results7, numResults);

//test consisting of words of the same length

//some are matching words, and some are not

//reveal the results using the function revealOutcomes (shouldn’t print anything)

string results8[MAXRESULTS];

string exampleDict8[] = { "buy", "but", "tub", "rub"};

numResults = shuffleChars("ubt", exampleDict8, 4, results8);

assert(numResults == 2);

revealOutcomes(results8, numResults);

//test consisting of very long words

//some are matching words, and some are not

//reveal the results using the function revealOutcomes

string results9[MAXRESULTS];

string exampleDict9[] = { "abcdefghijk","kgfieudcbah","hcibadkgfej","jkaighbtced","ghakbcfdeij"};

numResults = shuffleChars("gekdfjbcahi", exampleDict9, 5, results9);

assert(numResults == 3);

revealOutcomes(results9, numResults);

//test consisting of empty strings

//no words are matching

//reveal the results using the function revealOutcomes (shouldn’t print anything)

string results10[MAXRESULTS];

string exampleDict10[] = { "","","","",""};

numResults = shuffleChars("hi", exampleDict10, 5, results10);

assert(numResults == 0);

revealOutcomes(results10, numResults);

cout << "all tests passed" << endl;