Kevin Hahn

CS 32: Project 3

Report

Obstacles:

There were many obstacles encountered in completing this project. In the broadest sense, figuring out what helper functions were needed was the most difficult part, especially regarding the number of parameters needed in these functions. After this, implementing the recursive pseudocode was probably the next hurdle because without knowing how to recursively loop and generate permutations, then it will render the implementations of the other functions incomplete. Ultimately, the realization came as I created a permutational loop function to work in conjunction with a function that finds the permutations. Another obstacle was reading in each word, because there are many ways to check and read in stuff from the infile, like .eof() or char-by-char, or dictfile >> word, and some methods are more convenient and efficient than others. I compiled and ran all three methods but settled on the shift operator method of reading in input. After taking care of the input file processing issues, this made the rest of the implementation of the dictionaryReader helper function a lot easier.

Integrating the duplicate word checker without messing up the 3 primary functions was another difficulty. Debugging the edge cases, such as the number of permutations exceeding MAXRESULTS, number of words exceeding MAXDICTWORDS, and handling empty text files were tough to take care of. I had to revise some of my control statements by adding in these simple checks along with the existing conditionals. I had similar issues with my other helper function, checkTheDictionary, as it was used inside more than one function.

Test Cases:

///////////////////////////////////////////////////////////////////////////////

Part I:

string results[MAXRESULTS];

string dict[MAXDICTWORDS];

ifstream dictfile; // file containing the list of words

int nwords; // number of words read from dictionary

string word;

// if words.txt is an empty file

dictfile.open("words.txt");

if (!dictfile) {

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

return (1);

}

int numberOfWords = dictionaryReader(dictfile, dict);

// if a word isn't found:

int numOfMatches = recurCombos("CodyBellinger", dict, nwords, results);

assert(numOfMatches == 0); // results array should also be empty

// if words.txt was an empty file, then dictionaryReader should return 0.

nwords = dictionaryReader(dictfile, dict);

assert(nwords == 0);

// if words.txt contained more than MAXDICTWORDS,

// then dictionaryReader should return MAXDICTWORDS

nwords = dictionaryReader(dictfile, dict);

assert(nwords == MAXDICTWORDS);

// For handling words with more permutations than the results array can hold:

// if MAXRESULTS = 2, and words.txt didn't contain more than MAXDICTWORDS:

int num = recurCombos("race", dict, nwords, results);

assert(num == 2);

// results will only contain two out of the following: "race", "care", "acre"

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Part II:

string results[MAXRESULTS];

string dict[MAXDICTWORDS];

ifstream dictfile; // file containing the list of words

int nwords; // number of words read from dictionary

string word;

// assumes words.txt is the same file provided by the professor

dictfile.open("words.txt");

if (!dictfile) {

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

return (1);

}

nwords = dictionaryReader(dictfile, dict);

int numResults = recurCombos("ecra", dict, nwords, results);

assert(numResults == 3); // results should contain "race", "care", "acre"

numResults = recurCombos("dou'y", dict, nwords, results);

assert(numResults == 1 && results[0] == "you'd"); // check to handle case-sensitive and non-alphanumeric characters

numResults = recurCombos("", dict, nwords, results);

assert(numResults == 0); // empty string should have no matching permutations

numResults = recurCombos(" ", dict, nwords, results); // whitespace should have no matching permutations

assert(numResults == 0);

numResults = recurCombos("..h ", dict, nwords, results);

// alphanumerics + whitespace should have no matching permutations

assert(numResults == 0);

numResults = recurCombos("AAB", dict, nwords, results);

assert(numResults == 1 && results[0] == "ABA"); // handling all caps letters correctly (and duplicates)

numResults = recurCombos("1t0h", dict, nwords, results);

assert(numResults == 1 && results[0] == "10th"); // handling strings composed of alpha and numeric characters

numResults = recurCombos("...", dict, nwords, results);

assert(numResults == 0); // handling strings solely composed of non-alphanumeric characters

numResults = recurCombos("a", dict, nwords, results);

assert(numResults == 1 && results[0] == "a"); // handling one letter strings

numResults = recurCombos("AC", dict, nwords, results);

assert(numResults == 2 && results[0] == "AC" && results[1] == "CA"); // handling 2 letter strings

numResults = recurCombos("clla", dict, nwords, results);

assert(numResults == 1 && results[0] == "call"); // make sure theres no duplicates since call has 2 L's

numResults = recurCombos("DDO", dict, nwords, results);

assert(numResults == 1 && results[0] == "DOD"); // handling palindromes properly

numResults = recurCombos("evenhandde", dict, nwords, results);

assert(numResults == 1 && results[0] == "evenhanded"); // for handling words requiring large permutations

string ex[] = { "Gung-ho", "#P66gMyFriend", "do-or-die" };

numResults = recurCombos("Gungho-", ex, 3, results);

assert(numResults == 1 && results[0] == "Gung-ho"); // handling capital letters and non-alphanumerics

numResults = recurCombos("PuigMyFriend#", dict, nwords, results);

assert(numResults == 1 && results[0] == "#P66gMyFriend"); // handling long words with alphanumerics

// and non-alpanumeric characters

string ex2[1] = {};

numResults = recurCombos("", ex2, 0, results);

assert(numResults == 0); // handling an empty string array

string ex3[1] = {"Kershaw"};

numResults = recurCombos("Kershwa", ex3, 0, results);

assert(numResults == 1 && results[0] == "Kershaw"); // handling an input array of just one string

/\*

Additional cases:

Passing an input file with more than MAXDICTWORDS into the dictReader function.

This should return MAXDICTWORDS (300000)

Passing a string with more than 20 permutations into recurCombos function.

This should return MAXRESULTS (20)

\*/