Assignment #5 (50 points; due 12/1/2017 @ 11:59 P.M)

Some word games, like Scrabble, require rearranging a combination of letters to make a word. This type of arrangement is generally referred to as an anagram, it's known as a permutation in mathematics. This assignment will give you some experience thinking about and writing recursive functions. Write a C++ program that searches for ``anagrams'' in a dictionary. An anagram is a word obtained by scrambling the letters of some string. For example, the word ``pot'' is an anagram of the string `'otp.'' A sample run of the program is given below. Your output does not have to be formatted exactly the same as that shown in the sample, but should be in a similar style. You can use words.txt as your dictionary file and anagram.cpp as an example of a main program.

Since the purpose of this assignment is to give you experience using recursion, you may not use any of C++'s iteration constructs (do, while, for, and goto) or any STL algorithms (if you have no idea what this means, you're OK). All repetition must be accomplished using recursion. This applies to every operation in the program, even file operations. Obviously, you would never write a program like this in industry but as an exercise it should be useful to gain experience with recursion.

Sample Runs

Here are two examples of how the program might work:

```
Please enter a string for an anagram: opt
Matching word opt
Matching word pot
Matching word top

Please enter a string for an anagram: blah
No matches found
```

Requirements

You must write these three functions with the exact same function signature (include case):

int readDictionary(istream &dictfile, string dict[]);

Places each string in dictfile into the array dict. Returns the number of words read into dict. This number should not be larger than MAXDICTWORDS since that is the size of the array.

int recursivePermute(string word, const string dict[], int size, string results[]);

Places all the permutations of word, which are found in dict into results. Returns the number of matched words found. This number should not be larger than MAXRESULTS since that is the size of the array. The size is the number of words inside the dict array.

void recurPrint(const string results[], int size);

Prints size number of strings from results. The results can be printed in any order.

For words with double letters you may find that different permutations match the same word in the dictionary. For example, if you find all the permutations of the string kloo using the algorithm we've discussed you may find that the word look is found twice. The o's in kloo take turns in front. Your program should ensure that matches are unique, in other words, the results array returned from the recursivePermute function should have no duplicates. A nice way to test this, and your function in general, might be to use the assert facility from the standard library. If done properly the following code should run without a runtime error being generated.

```
string exampleDict[] = {"kool", "moe", "dee"};
int numResults = recursivePermute("look", exampleDict, 3, results);
assert(numResults == 1 && results[0] == "kool");
```

Again, your solution must not use the keywords while, for, or goto or any STL algorithm functions. You must not use global variables or variables declared with the keyword static, and you must not modify the function parameter lists. You must use the integer constants MAXRESULTS and MAXDICTWORDS, as the declared sizes of your arrays, as in the anagram.cpp example provided to you.

Helpful Tips

In this project, you will also have to deal with one of the drawbacks of using recursive functions. Repeated recursive calls may exhaust the stack space (we will talk about stacks soon) that's been allocated for your program. If you use the sample dictionary file provided, you are almost guaranteed to have a default stack size that is not large enough. Here is how to change the stack size on different platforms:

Visual Studio:

In the Property Pages dialog, in the left panel, select Configuration Properties / Linker / System. In the right panel, select Stack Reserve Size, and in the drop down list to its right, type in a new stack size (8000000 is approximately 8MB). Click OK.

Xcode:

Click on the Project Name, Select Build Settings at the top then scroll below to find the Linker subsection. Add -Wl,-stack_size,8000000 to the Other Linker Flags field.

Linux:

Run the command ulimit -s 8000 before compiling your program.

While completing this assignment you may find it helpful to review file operations and using the substr function.

The source file you turn in will contain all the functions and a main routine. You can have the main routine do whatever you want, because we will rename it to something harmless, never call it, and append our own main routine to your file. Our main routine will thoroughly test your functions. You'll probably want your main routine to do the same. If you wish, you may write functions in addition to those required here. We will not directly call any such additional functions.

What to turn in for this Project

What you will turn in for this assignment is a zip file containing these two files:

- A text file named anagrams.cpp that contains the source code for your C++ program. Your source code should have helpful comments that tell the purpose of the major program segments and explain any tricky code.
- A file named report.doc or report.docx (in Microsoft Word format) or report.txt (an ordinary text file) that contains of:
    - A brief description of notable obstacles you overcame.
    - 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.

Please use Canvas to turn in your zip file electronically. Remember that most computing tasks take longer than expected. Start this assignment now!