Computer Science II — Homework 1 — Word Circle Puzzle.

Preliminaries

This assignment is due Thursday, January 24 at 11:59:59pm and is worth 60 points toward your homework grade. The primary component of the assignment is your first programming problem. The secondary component consists of the answers to two short questions that analyze the number of operations required by your program (see the notes from Lecture 1 and the associated examples and solution). You must submit both a C++ source code file and `readme.txt` file, where the latter contains the answers to the short questions. A template for `readme.txt`, including the questions, is provided on the course web page. The rest of this handout is about the programming problem.

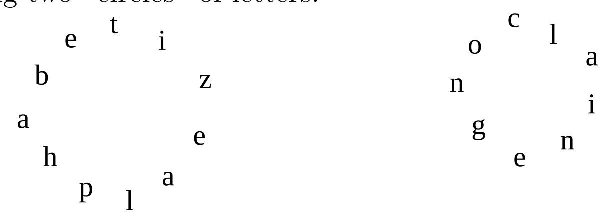
In solving the problem you may use any C++ construct or algorithm you wish from Lectures 1 and 2, but not beyond. Of particular importance are the examples associated with Lecture 2 which are posted on-line, including

- `alphabetize.cpp` and
- the recursive and non-recursive versions of binary search.

You may use as little or as much of this code in your own solution as you choose. Our tests for academic integrity violations account for code provided to you.

The Puzzle

Consider the following two "circles" of letters.



By starting in the lower right of the first circle with the letter `'a'`, moving clockwise through the letters, and using all the letters, the word `alphabetize` is spelled out. By starting on the top of the second circle with the letter `'c'`, moving counter-clockwise through the letters, and using all the letters, the word `congenial` is spelled out. The puzzle is to be able to automatically determine if a word can be formed from a circle of letters. The rules are:

1. any starting point on the circle is allowed,
2. the word may be formed by moving either clockwise or counter-clockwise,
3. all letters of the circle must be included in the word, and
4. the word must appear in the dictionary.

As a result, if a circle has n letters, then there are $2n$ possible words. Your job in Homework 1 is to solve this puzzle.

A String as a Letter Circle

The first issue that must be addressed is how to consider a string of letters as a circle, since a string is a linear sequence. The trick is to think using modular arithmetic, so that the "successor" of the last letter is the first letter, and the predecessor of the first letter is the last letter. The `%` operator in C++ helps enormously, because for a sequence of length n, the letter after the `i`-th letter is at location

```
(i+1) % n
```

even when `i==n-1`, and the letter before the `i`-th letter is at location

```
(i+n-1) % n
```

even when `i==0`. We will refer to a string with this type of wrap-around operation as a "letter circle". This is an example of creating a conceptual structure or an "abstraction" on top of a more basic structure.

The Problem, More Precisely

Your program will be provided two input files — one containing the dictionary of possible words and the other containing the letter "circles" to test as puzzles — and it must produce one output file giving the results of the puzzle tests. Therefore, the command-line form of the program is

```
word_circle dictionary.txt puzzle.txt results.txt
```

(See `alphabetize.cpp`, referenced above, for an example illustrating use of command-line arguments.) Both input files will contain a sequence of alphabetical strings, separated by whitespace. Each string will be formed by lower-case letters. No words will be repeated in `dictionary.txt`, but this file will not be alphabetized — your program must do so. Once the dictionary is alphabetized, you must use binary search on the dictionary (a vector of strings).

The file `results.txt` produced by your program should contain each letter circle (a string) from `puzzle.txt` on a separate line of output. If a word can be formed from a letter circle (as determined by finding the word in the dictionary of words), then on the line of output the string should be followed by the word formed. If a word can not be formed, the string <`none`> should be output. **Please see the example output on the course web page and follow the format of this output as precisely as you can.** You may assume at most one word in the dictionary can be formed from each letter circle and therefore you do not need to check for multiple words.

Developing Partial Solutions

The heart of the problem is generating each of the $2n$ possible words from a letter circle and testing it against the dictionary. In writing your program, you should consider these as two separate problems:

1. Write and test code to generate all $2n$ possible words from a string, demonstrating that you have the concept of a "letter circle" implemented correctly.

2. Write and test code to implement the dictionary, including reading, alphabetizing, and searching.

Combining the solution to these two problems will lead to the solution for the overall problem.

Questions to Answer in the readme.txt File

1. In your program, suppose n is the length of a “word circle”. Give an order notation analysis of the time required to generate the $2n$ possible words from the word circle.

2. If there are m words in the dictionary, what is the worst-case number of operations (again, using “O” notation) to solve a word circle puzzle. Your answer will be a function of both m and n.