CSci 1113: Introduction to C/C++ Programming for Scientists and Engineers Homework 6

Due Date: Mon, Apr. 1 before 5:30pm.

Purpose: This homework involves practice with strings as well as with other topics.

Instructions: *Important: As discussed in class this week, this is an* **individual** *homework assignment. Make sure you design and implement your solutions on your own without any disallowed types of help from others, including disallowed help from others not in this class and help from online sources.*

There are two problems worth 30 points each. The rules and reminders are the same as for past homework, so see the previous homework write-ups if you need a reminder.

Problem A: String Overlap (30 points)

In this problem you will design and implement C++ code that identifies overlap in strings. Specifically, design and implement a C++ program that does the following:

1. Asks a user to input a filename and then opens that file. If the file open fails, then print the message "Unable to open file" and terminate the program using `exit(1)`.
2. Reads the file contents, in order, into an array of strings. (See the file format explanation below.)
3. Computes the string overlapping order described below, and then prints the strings out, one per line, in that order.
4. Closes the file.

File Format: The data file consists of a number of strings, each on its own line. You do not know in advance how many strings will be in the file, other than there will be no more than 30 strings. Assume (i) there is at least one string in the file, (ii) the strings do not have any whitespace in their interior, and (iii) the strings consist entirely of alphabetic, upper case characters.

Here is an example of a data file containing four strings:

```
AGGTGTGGA
AAAATTA
AATTGTCGCTGA
GGAAAA
```

Overlapping Order: Here is the explanation of the string overlapping your program is to find in Step (3) above. Suppose we have the strings above read, in order, into an array of strings. We start with the first string in the array, AGGTGTGGA. What we want to determine is which of the other strings' beginning overlaps the most with the end of the this first string. Note this is a nontrivial problem since we do not know without further analysis what the size of the overlapping substring will be. For example, if we just look at the last character of AGGTGTGGA, the A, there are two other strings that begin with A. If we look at the last two characters, GA, there are no strings starting with GA. If we look at the last three, GGA, there is one other string starting at GGA. If we continue along this line, we notice that the largest overlap is the three characters GGA, and the overlap is with the fourth string. So move the fourth string into the second position in the array by exchanging it with the current second string:

```
AGGTGTGGA
GGAAAA
AATTGTCGCTGA
AAAATTA
```

Now which of the remaining strings (i.e., the third and fourth ones), has the largest overlap at its beginning with the end of GGAAAA? Both of the remaining strings have an overlap of one character (A), and of two characters (AA). But only one has a larger overlap, the fourth string, with maximum overlap AAAA. So move the fourth string into the third position by exchanging it with the third string:

```
AGGTGTGGA
GGAAAA
AAAATTA
AATTGTCGCTGA
```

Then we look for the maximum overlap between AAAATT and the remaining strings. However, since there is only one string left, the final string trivially has the maximum overlap. So the strings are now ordered as desired, and the program's output should be the strings printed one per line in the order immediately above.

Additional clarification, requirements, and advice:

- If more than one string has the maximum overlap (e.g., two string overlap with the current string in a substring of 3 characters) choose the one that occurs first in the current array.
- It is possible for the maximum overlap to be of length 0.
- Your program should work on a data file with any name, and with any reasonable number of strings, as long as the number of strings is at most 30.
- Your program just needs to print the strings in the final order. It does not need to print any additional information such as the amount of overlap.
- Make sure you understand how the overlap works here with the end of the "current" string overlapping with the beginning of the other strings.
- Finding the length of overlap between any two strings is not a simple task, but should not be extremely lengthy or difficult either. Think about the problem carefully, and

remember you can use string functions as useful. Moreover, to organize your program, write a function `int findOverlap(string s1, string s2)` that finds the number of characters of overlap between the beginning of `s2` and end of `s1`, and returns the length of this overlap. For example, when `s1 = AGGTGTGGA` and `s2 = GATTTAG` the function should return 2 since the largest overlap is two characters long, namely `GA`.

Here's a complete example using another data file. Suppose the file is the following:

```
AGCTG
GCGGACGGG
GGG
TGAGCGGA
GGGGC
```

Then here is the program input and output. As usual, user input is underlined, and there is a single space between the input prompt and the input file name. And — also as usual — follow the input and output example carefully.

```
Input filename:  testData.dat
AGCTG
TGAGCGGA
GCGGACGGG
GGG
GGGGC
```

And here is an example of an invalid filename:

```
Input filename:  testData.cat
Unable to open file
```

When you are done, name the source code file <username>_6A.cpp (where you replace <username> with your U of M email address) and submit it using the HW 6 Problem A submission link on the class website. Remember to follow the naming convention diligently.

Problem B: Quasi-Latin Generator (30 points)

In this problem you will design and implement C++ code that translates string data. Specifically, design and implement a C++ program that does the following:

Reads in lines, one at a time, from a data file "input.txt". If the file open fails, then print the message "Unable to open file" and terminate the program using `exit(1)`. If the file opens, then for each line translate each word in the line to "quasi-Latin" using the following rules:

- If a word starts with a consonant followed immediately by a vowel, then put the first letter of the word at the end of the word and add "oi." *For the purpose of this homework problem a vowel is defined as any english letter in the following list: a,e,i,o,u,y; note 'y' is included.. A consonant is any other English letter.*

    - Example: happy = appyh + oi = appyhoi

- If a word starts with two consonants move the two consonants to the end of the word and add "ah."
    - Example: child = ildch + ah = ildchah
- If a word starts with a vowel (a,e,i,o,u,y) add the string "eeh" at the end of the word.
    - Example: awesome = awesome +eeh = awesomeeeh

Output these translated sentences to a file ("output.txt") in the same order they were read in.

Additional rules:

1. All sentences should end with the punctuation of the original line.
2. Honor any commas or semi-colons in the line. That is, if there is a comma or semi-colon after a word in the original sentence, then there should be a comma or semi-colon after the translated word in the output sentence.
3. Honor capitalization. Each line (sentence) will begin with a capitalized letter; the only other times a word will be capitalized is if it is a proper noun, such as a person's name. When translating to quasi-Latin, the translated word should be capitalized if the original word was capitalized.
    - Zach $\rightarrow$ Achzoi
    - Shawn $\rightarrow$ Awnshah

Input guarantees:

Only alphabetic characters (upper and lower case a-z), spaces, periods, exclamations, question marks, and semi-colons will appear in any line (sentence). So there will be no other characters (like ' or ‘ or - or /, etc) and no numeric characters (0-9) will appear in the file. Periods and question marks will appear only at the end of the sentence. All words will be single space delimited. If a comma or semi-colon appears in a line, it will appear immediately after a word, and followed by a single space. So there will be no acronyms (e.g. U.S.A. or F.B.I. or A.S.A.P.). Each line will be separated from the next line by a newline character. Only the first letter of any word may be capitalized.

Additional Comments, Rules, and Clarifications:

- Use string objects, not C-style strings (i.e., not arrays of chars) in this program.
- There will be no terminal input or output for this program. Instead the program will always read data from the input file "input.txt" and write it to the output file "output.txt".

- You may assume the input file will always be named "input.txt" and the output file will always be named "output.txt".
- An example input file will be posted to the class site site. You should be able to figure out the correct output for the lines in this example input file, so you can use it to test your program. However, as usual, make sure your program runs correctly on any valid input file.
- Remember you can read in an entire line from a file using `getline`.
- Remember you can use string functions such as `substr`, `find`, etc.
- Advice on the program design and development: (i) Before writing code, think carefully about the program logic: what does the program need to do, and in what order? (ii) There are a number of subtasks your program will need to perform. Break the program up into subtasks, write functions for these subtasks as appropriate, and then develop your code incrementally.

Examples:

See Spot run, run Spot run! → Eesoi Otspah unroi, unroi Otspah unroi!

All good boys deserve fudge. → Alleeh oodgoi oysboi eservedoi udgefoi.

Can you hear me Tom? → Ancoi youeeh earhoi emoi Omtoi?

When you are done, name the source code file <username>_6B.cpp (where you replace <username> with your U of M email address) and submit it using the HW 6 Problem B submission link on the class website. Remember to follow the naming convention diligently.