CS 215 - Fundamentals of Programming II
Fall 2019 - Homework 5
20 points

Out: September 13, 2019
Due: September 20, 2019 (Friday)

Computers have long been used to encode and decode messages. The Caesar Shift Cipher from Homework 1 is one simple scheme that encodes each letter in a message separately. A simple scheme to encode words in a message is Pig Latin. For this assignment, we will translate English words in lowercase to a very simple form of Pig Latin. (Note: This is much simpler than a similar CS 210 assignment that you may or may not have completed.) There are only two rules:

1. If a word start with a vowel, simply append "ay" to the end of the word. Example: *insect* becomes *insectay*.
2. Otherwise, remove the first letter of the word, append it to the end of the word, and then append "ay". Example: *dog* becomes *ogday*.

Write a program in a file named **`piglatin.cpp`** that will read lines of text from a file. For each word in a line, the program will translate the word into Pig Latin and output it to a file. Assume that the input file meets the following specification:

- **All** characters are lowercase alphabetic characters or spaces. I.e., there are no uppercase letters, digits, or symbols in the input.
- The first character in a line is not a space.
- The last character of a line (i.e., before the newline) is not a space.
- There is exactly one space between each word in a line.
- There are no blank lines.

The output file must meet the same specification. That is, the translated words of a line of input must appear in the same line in the output separated by one space and each line must end with a non-whitespace character and a newline.

The program **must be implemented** using (at least) the following functions:

- A function that ***receives*** a C++ string that is a line of words in all lowercase letters and ***passes back*** a string that is the Pig Latin translation of the word. E.g.,
  `translate_sentence (english_sentence, piglatin_sentence);`
- A function that ***receives*** a C++ string that is a word in all lowercase letters and ***passes back*** a string that is the Pig Latin translation of the word. E.g.,
  `translate_word (english_word, piglatin_word);`

The idea is for the main program to repeatedly read in a line of the file, call the translate_sentence function, and print out the translated sentence. The translate_sentence function find the words in the received English sentence and calls the translate_word function on each word to build up the translated sentence. Reminder: C++ strings are defined in the **`<string>`** library.

All functions must have a prototype written above the main program, and their function definitions must be written below the main program. Here are some notes that may be helpful.

- A link to a reference page for the C++ string type and its functions is available on the course webpage. As demonstrated in class, most string functions are called using object dot notation (<stringvar>.<functionname>) unless they are operator functions. Functions you might want to use include: indexing operator[ ], concatenation operator+, find, and substr. Use of C strings and the C string library functions (strlen, strcmp, etc.) **are not allowed** for this assignment.
- Since the newlines need to be preserved in the output, the input should be read line by line using getline( ). Like operator>> and get( ), getline returns the stream object that converts to true when a read is successful and false when it is not. Thus the main processing loop condition should be the call to getline().
- There are numerous ways to tokenize a line (i.e., break it into "words"). For this problem, you should use only string operations. E.g., find the spaces in the word and then get out the substring between the spaces. The constant **`string::npos`** is used to indicate an "undefined" index. It should be used to determine whether a find operation returned a valid index. Some example code, using various string operations:

```
getline(infile, str_var);            // read a line
index_of_space = str_var.find(' ');  // look for the first space
if (index_of_space == string::npos)  // no space found
   cout << "No spaces in this string" << endl;
else
{
   index_of_next_space = str_var.find(' ', index_of_space+1);
   if (index_of_next_space != string::npos)
      word = str_var.substr(index_of_space+1,
                            index_of_next_space - index_of _space-1);
   else
      word = str_var.substr(index_of_space+1);  // to end of str_var
}
```

- String indexes are of type **`size_t`**. Use this type instead of **`int`** to eliminate warnings about mixing signed and unsigned integers in for-loops.
- A sentence can be thought of as a word followed by one or more sets of a space and a word. Thus the first word of a line is slightly different from the rest of the words in a line since there is no space in the output before the first word, and can be handled specially before the looping for the rest of the words. Alternately, a sentence can be thought of a series of a word and a space with the last word being the special case and not having a space afterward. In this case, the last word is handled specially.

The program must accept the input and output file names as command-line arguments. **Proper error checking of the number of command-line arguments and successful file opens must be made.**

An input file named **`sample.txt`** is provided and contains the following data:

```
this is a test of english
into pig latin
```

The program is compiled and run using:

```
$ clang++ piglatin.cpp -o piglatin -Wall -std=c++11
$ ./piglatin sample.txt output.txt
```

After the program is run, the output file named **output.txt** should contain the following result:

```
histay isay aay esttay ofay englishay
intoay igpay atinlay
```

Coding Notes

This assignment is being distributed by the GitKeeper submission system. You should have received an email with the URL to clone. A reminder of the commands needed to set up this assignment:

```
$ cd <path to your class directory>
$ git clone <URL given in the assignment email>
```

This will create an working directory called **hwk05-piglatin** containing example input file **input.dat**. Change to this directory and create **piglatin.cpp**.

```
$ cd hwk05-piglatin
$ emacs piglatin.cpp &
```

Be sure to commit your changes early and often. **The first commit for this assignment is expected to be by Sunday, September 15.** A reminder of the commands needed to do this (run in the working directory):

```
$ git add .
$ git commit -m "<description of what was completed>"
```

How to submit

The program file must have a comment at the top of the file with your name and assignment number.
Please note that the automated submission system requires all files be named exactly as specified in an assignment (**piglatin.cpp**, in this case), and also requires that the output of the program be exactly as expected including whitespace, capitalization, and spelling.

Assignments are submitted to GitKeeper by pushing committed changes.

```
$ git push
```

Only a clean repository (one where all changes have been committed) can be pushed. If a repository is not clean, git will say everything is up to date, and refuse to do the push. To see which files have been changed, but not committed, use command:

```
$ git status
```

You can also see all the log messages by using command:

```
$ git log
```

Note that the first commit was done by the GitKeeper system.