Lab: Problem Solving with `std::string`s

Learning Outcomes

- Gain experience with interface/implementation and abstract data type methodology
- Gain experience with functional decomposition and algorithm design
- Gain experience with solving textual problems using the C++ standard library

Task

Your task is to implement a constructed language game called Pig Latin. This game applies a simple set of rules to convert an English word into Pig Latin so that people can speak an argot for fun. For the purposes of this exercise, a word is a sequence of non-whitespace characters consisting of lower-case Latin characters (a through z), upper-case Latin characters (A through Z), digits (0 through 9), period (.), and apostrophe (’). Some examples of words: Zebra, doesn’t, PhD., 12alpha34, and apple. The specific rules of the game are:

1. If the word starts with a vowel (a, e, i, o, u and their upper case versions), the Pig Latin version is the original word with suffix -yay added at the end. Some examples:

English word	Pig Latin word
apple	apple-yay
eat	eat-yay
Omelet	Omelet-yay

2. If a word starts with a consonant (a character that is not a vowel), or a series of consecutive consonants, the Pig Latin version transfers all consonants up to the first vowel to the end of the word, and adds suffix -ay to the end. Some examples:

English word	Pig Latin word
box	oxb-ay
stupid	upidst-ay
whisker	iskerwh-ay

3. If a word contains no vowels, the Pig Latin version is the original word with suffix -way added to the end. Some examples:

English word	Pig Latin word
psst	psst-way
hmm	hmm-way
1234	1234-way

4. The letter `y` or `Y` should be treated as a consonant if it is the first letter of the word, otherwise, it is treated as a vowel. Some examples:

English word	Pig Latin word
yellow (y is treated as consonant)	ellowy-ay
bye (y is treated as vowel)	yeb-ay
aye (here it doesn't matter whether y is consonant or vowel since word starts with vowel)	aye-yay
yww (y is treated as consonant)	yww-way

5. If the English word is capitalized, the Pig Latin version of the word should be capitalized in the first letter. The previous capital letter is no longer capitalized. Some examples:

English word	Pig Latin word
There	Ereth-ay
Apple	Apple-yay
Flower	Owerfl-ay
Dr.	Dr.-way

To implement the constructed game with rules described above, define function

```
std::string to_piglatin(std::string word);
```

that takes as parameter an English word `word` and returns the corresponding Pig Latin version. Here's a use case of function `to_piglatin`:

```
std::array<std::string, 4> vocab {"Apple", "box", "psst", "Yellow"};
for (std::string const& word : vocab) {
  std::cout << word << " --> " << to_piglatin(word) << "\n";
}
```

The above code fragment should generate the following output:

```
Apple --> Apple-yay
box --> oxb-ay
psst --> psst-way
Yellow --> Ellowy-ay
```

You must declare the function in header file `q.hpp` and define the function in source file `q.cpp`. Both the declaration and definition should be in namespace `hlp2`. You should write the solution only using the C++ standard library. You can't use C headers such as `<cstring>`, `<cctype>`, and so on in your code since such headers are unnecessary and the auto grader will not accept your submission.

Read this again: ***Your implementation cannot use the C standard library but can use any and all aspects of the C++ standard library. Addition of C standard library headers such as*** `<cstring>`, `<cctype>`, `<cstdio>`, `<cstdlib>` ***and so on [even in code comments] will prevent the auto grader from accepting your submission.***

Algorithm development and implementation

There are many ways to design an algorithm that solves the translation problem. You can choose to implement your own technique. Or, you can use the algorithm described below. In any case, make sure to read the sections below detailing what to submit, how to submit, and the rubrics that will be used to assign grades before jumping to the implementation details.

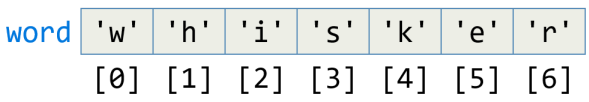
Let's recap the rules imposed on the algorithm to convert English words to Pig Latin:

1. If a word starts with a vowel (a, e, i, o, u and their upper case versions), the Pig Latin version is the original word with suffix -yay added at the end. Some examples: apple to apple-yay; Omelet to Omelet-yay.
2. If a word starts with a consonant, or a series of consecutive consonants, the Pig Latin version transfers all consonants up to the first vowel to the end of the word, and adds suffix -ay to the end. Some examples: box to oxb-ay; stupid to upidst-ay.
3. If a word contains no vowels, the Pig Latin version is the original word with suffix -way added to the end. Some examples: psst to psst-way; 1234 to 1234-way.
4. The letter y or Y should be treated as a consonant if it is the first letter of the word, otherwise, it is treated as a vowel. Some examples: yellow (y is a consonant) to ellowy-ay; bye (y is a vowel) to yeb-ay.
5. If the English word is capitalized, the Pig Latin version of the word should be capitalized in the first letter. Some examples: Flower to Owerfl-ay; There to Ereth-ay.

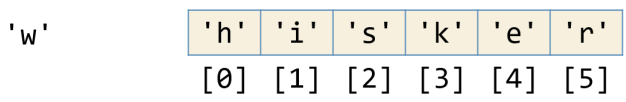
Assume the input to the algorithm is an English word encapsulated by object `word` of type `string`. Also, suppose that the algorithm's output is specified by object `word`.

Rule 1 requires checking whether the first character, `word[0]`, of `word` is a vowel. If `word[0]` is a vowel, add suffix `"-yay"` at the end of `word`. Research `std::string` member function `operator+=` for concatenating strings. The algorithm concludes if Rule 1 is satisfied. Functional decomposition recommends implementing a function `is_vowel` that returns `true` if its `char` parameter represents a vowel and `false` otherwise.

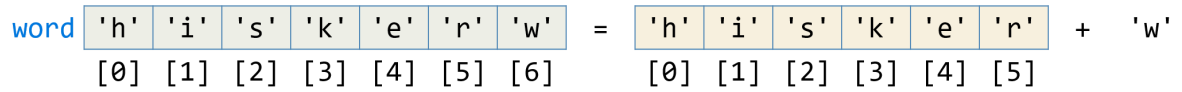
Suppose `word[0]` is not a vowel. For instance, this will happen if `word` encapsulates string `"whisker"`:



Slice `word` into two parts: the first part contains the first character `word[0]`; the second part contains the remaining characters of `word`:



Add the first part at the end of second part - this will now be new `word`. That is, the second character of `word` becomes the first character of `word` while the first character of `word` has become the last character:



Now, check if the current first character of `word` is a vowel. *You cannot use the previously defined* `is_vowel` *function because (according to Rule* 4*)* `'y'` *and* `'Y'` *must now be considered as vowels in addition to the original set of vowels.* If the current first character is a vowel, Rule 2 requires the addition of suffix `"-ay"` at the end of `word`. Otherwise, the slicing and adding part must be repeated until either the first character of `word` is a vowel or all the characters of `word` have been processed, in which case `word` doesn't contain any vowels. If there are no vowels, Rule 3 requires the addition of suffix `"-way"` at the end of the original input.

Slicing a `string` object into two parts and adding the first part at the end of the second part to create a new `string` object can be implemented using `string` member function `substr` and non-member function `operator+`. Since this operation may be required in multiple places in the algorithm, it is useful to encapsulate this operation in a function `rotate`.

The only thing left to solve is Rule 5 which is left for the reader to figure out.

The objective of this lab is to practice programming with abstract data types without worrying about their implementation details. Rather than defining arrays, using pointers, or dealing with run-time memory allocation and deallocation, you'll implement the algorithm using abstract data type `std::string` that provides a simpler and more importantly a *safer* alternative to C-style strings. You should practice functional decomposition techniques learnt in HLP1 to divide the algorithm into several smaller problems with each problem implemented by a *simple to understand*, *simple to code*, and *simple to test* function.

Submission Details

Header file `q.hpp`

Submit header file `q.hpp`.

Source file `q.cpp`

Submit source file `q.cpp`.

Compiling, executing, and testing

Download `q-driver.cpp`, `makefile`, an input file `english-words.txt` containing English words, and correct Pig Latin translations of these words in output file `pig-latin-words.txt`. Run `make` with the default rule to bring program executable `q.out` up to date:

```
$ make
```

Or, directly test your implementation by running `make` with target `test`:

```
$ make test
```

If the `diff` command in the `test` rule is not silent, then one or more of your function definitions is incorrect and will require further work.

File-level and function-level documentation

Every source and header file *must* begin with a *file-level* documentation block. This module will use Doxygen to tag source and header files for generating html-based documentation. In addition, every function that you declare and define and submit for assessment must contain *function-level documentation*. This documentation should consist of a description of the function, the inputs, and return value.

Submission and automatic evaluation

1. In the course web page, click on the appropriate submission page to submit the necessary files.
2. Read the following rubrics to maximize your grade. Your submission will receive:
    - F grade if your submission doesn't compile with the full suite of `g++` options.
    - F grade if your submission doesn't link to create an executable.
    - Your implementation's output doesn't match correct output of the grader (you can see the inputs and outputs of the auto grader's tests). The auto grader will assign 50% of the grade based on the input and output files given to you. The remaining 50% of the grade will be awarded based on the additional tests implemented by the auto grader.
    - The auto grade will provide a proportional grade based on how many incorrect results were generated by your submission. $A+$ grade if your output matches correct output of auto grader.
    - A deduction of one letter grade for each missing documentation block in your submissions. Each source file must have **one** file-level documentation block and a function-level documentation block for each defined function. A teaching assistant will physically read submitted source files to ensure that these documentation blocks are authored correctly. Each missing or incomplete or copy-pasted (with irrelevant information from some previous assessment) block will result in a deduction of a letter grade. For example, if the automatic grader gave your submission an $A+$ grade and one documentation block is missing, your grade will be later reduced from $A+$ to $B+$. Another example: if the automatic grade gave your submission a C grade and the two documentation blocks are missing, your grade will be later reduced from C to F.