ASSIGNMENT 3

COMP-202, Winter 2015, All Sections

Due: Tuesday, February 24, 2015 (23:59)

Please read the entire pdf before starting.

You must do this assignment individually and, unless otherwise specified, you must follow all the general instructions and regulations for assignments. Graders have the discretion to deduct up to 10% of the value of this assignment for deviations from the general instructions and regulations. These regulations are posted on the course website. Be sure to read them before starting.

Question 1:	40 points
Question 2:	60 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment through automated tests. While these tests will not determine your entire grade, it will speed up the process significantly, which will allow the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. Marks can be removed if comments are missing, if the code is not well structured, and if the problems your solution does not respect the assignment requirement.

Assignment

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions.*

Warm-up Question 1 (0 points)

Write a method called *reverse* which takes as input an array of integers and reverses the order of the elements **in place**. This method should not create a new array, and should return nothing. For example, if the input is:

```
{3, 2, 98, 32, -12, 1}
```

after the method is executed the array shall be

```
{1, -12, 32, 98, 2, 3}
```

Warm-up Question 2 (0 points)

Write a method called *contains* which takes two input arguments: an array of Strings `strArray` and a String `str`. The method returns true if the `strArray` contains `str`, and false otherwise. For example, if the input is

`{"tree", "apple", "earth", "air"}, "air"`

the method returns true, but if the input is

`{"tree", "apple", "earth", "air"}, "water"`

the method returns false.

Warm-up Question 3 (0 points)

Write a method called *uniqueElements* which takes an array of integers, and returns true if there are no duplicate values in the array, and false otherwise. For example, the input {3, 2, 3} would return false, because 3 appears twice.

Part 2

The questions in this part of the assignment will be graded. This assignment asks you to implement methods that perform some specific function. To obtain full marks, you must adhere to the specified requirements for the method names and input parameters. You may in addition implement helper methods to perform some subtask as part of solving the questions below. All the methods you will be asked to write in this assignment take the `public` and `static` keywords.

Question 1: Longest Increasing Contiguous Subsequence (40 points)

A **contiguous subsequence** is a sequence whose elements can be found in the same order contiguously in another sequence. For example, some contiguous subsequences of the sequence A = {3, 6, 5, 1, 9, 3, 2, 3, 4, 5, 1} include {3, 6, 5} from the first, second, and third elements of A; and {5, 1}, from the third and fourth elements of A. On the other hand, {6, 9} is not a contiguous subsequence of A because the numbers do not appear contiguously in A, nor is {7, 1}, because 7 cannot be found in A. An **increasing** sequence is one in which each element is strictly greater than the previous one. For example, {1, 2, 3} , or {4} are increasing sequences, whereas {6, 2, 4} and {1, 2, 2, 3} are not. The **longest increasing contiguous subsequence** (LICS) is the contiguous subsequence of the greatest length that is also increasing.

In a class called `LongestSubsequence`, implement two methods, `longestForward` and `longestBackward`, as follows.

Method 1: Longest Increasing Contiguous Subsequence Forwards

First, implement `longestForward`, a method which takes an int array as input, and returns no output. This method prints two lines of text. The first line is a number, which is the length of the LICS. The second line is the actual elements of the subsequence itself, separated by commas. In case there are multiple LICSs of the same length, print the first one.

For example, passing the sample sequence A to `longestForward` would result in the following being printed to screen:

```
4
2, 3, 4, 5
```

This means that the LICS contains four elements, which are 2, 3, 4 and 5. You may assume that the input array always contains at least one element.

Suggested Algorithm

To solve the above problem, first write a loop that will determine the LICS, *assuming that it starts at a certain index of the input array.* For example, the LICS of A starting at index 0 is {3, 6} with a length

of 2, whereas the LICS of A starting at index 4 is $\{9\}$ with a length of 1. Run this code for all of the indices of the original array. The overall LICS is the longest subsequence that results over all possible starting points.

You will need to declare variables to keep track of the length of the longest increasing contiguous subsequence found so far, as well as the index of the array where this subsequence starts.

Hints

1. Define a number of int arrays in the main method, and pass them to the method you are implementing. See if you get the result you expected.
2. Be sure to test cases where the longest increasing contiguous subsequence occurs at the start, the middle, and the end of the original sequence.
3. Don't be afraid to define helper methods to help you solve subtasks.
4. `Integer.MIN_VALUE` is the smallest integer that is representable by an int in Java.

Method 2: Longest Increasing Contiguous Subsequence Backwards

Implement `longestBackward`, which is the same as `longestForward`, except it searches the original array backwards. In case of a tie, the method should print the LICS going backwards that starts at a *later* index in the input array. For example, calling `longestBackward` on A results in the following being printed:

```
3
2, 3, 9
```

Hints

1. This question may be easier than you think, if you've done the warm-up exercises!
2. The input array passed to `longestForward` or `longestBackward` may not be changed as a result of the methods. That is, if the code `longestBackward(inputArr);` is executed, the values of the array `inputArr` should be the same before the method is called, and after.

Question 2: Pig Latin (60 points)

Pig Latin is a **language game**, in which regular English words are changed and obfuscated according a number of simple rules, so that the resulting text becomes hard to understand. An example of a sentence in Pig Latin is

```
Inway isthay artpay ofway ethay assignmentway, ouyay illway itewray omesay odecay
atthay anslatestray Englishway intoway Igpay Atinlay.
```

which translates into

```
In this part of the assignment, you will write some code that translates English
into Pig Latin.
```

Here are the rules to translate an English word into Pig Latin. First, if the word starts with one or more consonants, the consonant(s) are moved to the end of the word. So, "pig" becomes "igp", and "this" becomes "isth". If the word starts with a vowel, a 'w' is added to the end of the word, so "assignment" becomes "assignmentw".

Then, the suffix "ay" is added to the result of the first step. So "pig" becomes "igpay", "this" becomes "isthway", and "assignment" becomes "assignmentway".

For this question, you will write a class called PigLatin. The main method of this class uses a Scanner to ask for an input sentence in English, then prints out its translation in Pig Latin. Assume that the input sentence is in all lower-case, and does not contain any punctuation or extra spacing. Here is a sample run of the program.

```
Sentence to translate:
the quick brown fox jumped over the lazy dog // input by user
ethay uickqay ownbray oxfay umpedjay overway ethay azylay ogday
```

To implement the above functionality, you will write the following three methods, and any additional helper methods as you deem appropriate.

Method 1: Tokenize the Input Sentence

Write a method `tokenize` which takes a single String as input, and returns a String array containing the words in the sentence, assuming that the words are separated by a single space character. You may not use the split method of String. For example, calling tokenize on the String "the quick brown fox jumped over the lazy dog" results in a String array containing {"the", "quick", "brown", "fox", "jumped", "over", "the", "lazy", "dog"}.

Implementation Hints

1. Declare a new String array with more entries than you will need, storing new words as you encounter space characters in this new array.
2. Then, at the end, create a new String array with just enough entries, and copy the non-null elements of the first array over.
3. Make sure you don't forget the last word of the sentence.

Method 2: Detect Consonant Prefix

Write a method, `detectPrefix`, which takes a String as input, and returns a String as the output, containing the consonant characters at the start of the word before the first vowel. Assume that the vowels are 'a', 'e', 'i', 'o', and 'u', and all other characters are consonants, including 'y'.

Here are some examples:

- `detectPrefix("quick")` should return "q",
- `detectPrefix("one")` should return "", and
- `detectPrefix("brown")` should return "br".

Method 3: Translate an English Word into Pig Latin

Implement a method, `pigLatinfy`, which takes a Strings as input, representing a word in English to be translated. The method returns the word in Pig Latin, as described by the rules above. Your implementation of `pigLatinfy` must call the method `detectPrefix` you wrote above. So, `pigLatinfy("quick")` should return "uickqay", and `pigLatinfy("one")` returns "oneway".

After implementing these three methods, be sure to write more code to print out the entire final sentence in Pig Latin, as shown by the sample run of the program above.

Testing Your Code

As you are developing the programs, you will need to test the code you have written so far to make sure that is correct. To do so, come up with several cases for which you have manually computed the expected results of the methods you are writing, then run the code you wrote on those cases as well in the main method. Do they match? If so, great! If not, it's time to go back and debug your code some more.

It is a good idea to make several test cases for each method, that test a range of possible cases within the execution of your method. For example, in Question 2's `detectPrefix`, you should have a test case where the prefix has length 0, a test case where the prefix has length 1, and a test case where the prefix is longer. As another example, in Question 1, make sure your code can find the LICS no matter what its length is (1, 2, or even the length of the entire original sequence), or where it starts and ends (the beginning, the middle, or the end).

You may leave your testing code in the main method for Question 1, because we do not ask you to put anything else there, but be sure to remove your testing code from the main method for Question 2 before submitting your assignment.

What To Submit

You have to submit one zip file that contains all your files to myCourses - Assignment 3. If you do not know how to zip files, please enquire that information from any search engine or friends. Google might be your best friend with this, and for a lot of different little problems as well.

`LongestSubsequence.java` - Java code
`PigLatin.java` - Java code

`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise he or she would not.

Marking Scheme

Up to 30% can be removed for bad indentation of your code as well as omitting comments, coding structure, or missing files. Marks will be removed as well if the class names are not respected.

Question 1

Correct method signatures	5	points
Finding LICS forwards	15	points
Finding LICS backwards	15	points
Correctly displays output	5	points
	40	**points**

Question 2

Tokenize	20	points
Detection of consonant prefix	20	points
Word translation	10	points
Main method and overall program	10	points
	60	**points**