Algorithm Analysis & Design

CMPT 435

Homework #2 –Sequential Structures – 100 points

Goals

To implement common sequential data structures and their operations.

Preparation

For this assignment, you must have already completed Homework #1, which includes creating your *{YourLastname}-Work* repository and setting up a solution/project named *Homework* using a Java or C++ IDE. You must also download and extract the `Treasure.zip` archive to your repository, which contains ten (10) text files for us in this assignment.

Instructions

Demonstrate a practical and theoretical understanding of elementary, sequential data structures by implementing the following in your *Homework* project:

★ *Create an interface for* ***`EnchantedContainer`****s with the following methods:* **20pts**

- `stow` – takes an item name and adds it to the container
- `peek` – gets the name of the "next" item (meaning differs depending on container)
- `use` – discards the "next" item from the container

★ *Create a* ***`HaversackOfHolding`*** *class that realizes* ***`EnchantedContainer`****.* **30pts**

- Develop your own singly-linked list structure to provide the underlying storage representation for this class. Tip: I recommend using an inner class for `Node`s.
- This container must behave like a *stack*, with LIFO semantics, so that "next" means the item most recently `stow`ed (of those remaining).

★ *Create a* ***`BandolierOfConvenience`*** *class that realizes* ***`EnchantedContainer`****.* **30pts**

- Develop your own doubly-linked list structure to provide the underlying storage representation for this class. Tip: I recommend using an inner class for `Node`s.
- This container must behave like a *queue*, with FIFO semantics, so that "next" means the item first `stow`ed (of those remaining).

★ *Test your structures on the* **`chest*.txt`** *input data.* **20pts**

- For each treasure chest, you'll use two `EnchantedContainer`s in tandem to check if it was packed using the revered "palindrome packing" method, meaning the first and last items are the same, the second and penultimate items are the same, etc.
- Use (with modification if necessary) your file-reading routine from Homework #1 to read each input file line by line, storing each line in an array. **Note: Each treasure chest contains two instances of each item name.**
- Create an instance of each of your concrete classes for use a described below.
- To perform the checking, `stow` all items from a given chest in both the `Haversack` and the Bandolier, then `use` all the items, checking one-by-one whether the removed items are the same. If they are all the same, then the chest clearly was packed using the "palindrome packing" method.
- Print out the file names of all, and only, the "palindrome-packed" treasure chests.

★ *Follow software best practices.* **up to −∞pts**

- Consistent and readable coding style appropriate to your chosen language.
- Appropriate comments both at the top of each source file and throughout.
- Commit and push to GitHub frequently with concise meaningful commit messages.
- You must have at least one commit for an initial implementation of each interface/class, along with additional commits as you develop and test each of the methods of your classes.

Advice

Be sure to stage your changes a little at a time and commit frequently. Push any local work to your remote GitHub repository regularly. Don't forget to:

- Write short but meaningful messages for every commit. (Tip: Consider using the instructions themselves in your commit messages.)
- Look at the differences between successive versions of your code. Do this on GitHub as well as in Terminal/Git Bash using the `git diff` command.

Test, test, test… and test again. Then test some more. When you think you've tested enough, go back and test yet again. Then get someone else to test for you while you test theirs. Etc.

Submitting

You must push your changes to GitHub before the due date.

Note: Pushing regularly will reduce the risk of losing your work, so do not wait until after you have made all changes and commits before pushing.