Pset 4: Train Conductor
CPSC 223
Due Date: 4/6/2023

Boston's MBTA has been adding extensions for its Green Line service. They want to keep track of each stop and it's accessibility options. They want to be able to easily add and remove possible stops. The MBTA has given you a file `glx.txt` that contains the names of all the planned additional stops and whether they provide accessibility options (Y/N). They want to be able to quickly add and remove stops from this list. They also want the ability to neatly print the stops in order - similar to a map.

The driver

In this Pset, you will be provided the driver. Please note that you compile with `make train-conductor`, and execute with `./train-conductor`.

The Assignment

The driver `train-conductor.cpp` comes fully implemented this time for you to better understand and to analyze C++ language. The driver opens a file, reads the station names, inserts them into a linked list (and also deletes a few of them). Please be aware that you **should not modify this driver**.

For this assignment you must complete the implementation of the linked list and stations classes. You are allowed to modify and add extra functions and attributes to your code to better modularize the program. Remember that these functions and attributes are to be only added under the **private** section of your code.

The **Station** class you must implement the following functions:

1. Parameterized Constructor
    a. Takes in all information of a Station object, which contains
        i. station name (string)
        ii. accessibility (boolean: true if accessible, false if not)
2. isEqual
    a. Takes in a station, returns a boolean
    b. Returns true if the station given as a parameter is identical to the object whose function is being invoked (that is, they have the same name and accessibility). Returns false otherwise.

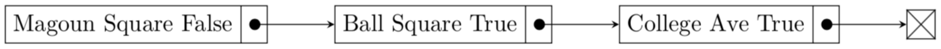
3. Print
    a. Takes in a reference to an ostream object, returns void
    b. Should print a) the string containing the station name and b) an `A` or `U` to indicate whether accessibility options are available (A = available, U = unavailable). Example:

        ```
        Ball Square A
        ```

        i. Please take **careful note** of this printout format. It is **imperative** that you follow this exactly; `diff`, used in our testing, will mark output as incorrect even if one character (or the spacing) is different.
        ii. There is no newline character at the end of Station's print function.

In the **LinkedList** class implement the following operations

4. ~LinkedList
    a. Standard destructor that deletes all memory associated with the class
5. insertStation
    a. Takes in Station as parameter, returns void
    b. Inserts the given station into the **head** of the list
6. removeStation
    a. Takes in station, returns void
    b. Deletes the first instance of a station that is in the list and isEqual to the one given as part of the input (see isEqual function above).
    c. Needless to say, the operation should deallocate any memory reserved for the station just deleted.
    d. This function should **still work** even in the event that the station is not in the list.
7. makeEmpty
    a. Takes no parameters, returns void
    b. This function should delete all elements in the list and deallocate any allocated memory. (Hint: this operation should be very similar to the destructor.)
8. Print
    a. Takes in a reference to an ostream object, returns void
    b. Prints all the stations in the list in order from front to the end in the following format:
        i. First, you list the Station name and accessibility (with print function in the station class)
        ii. Then a Space character
        iii. Distance to the final stop (to be calculated here, not stored)
        iv. If there is a next station:
            1. Space character
            2. "Train tracks" (==)
            3. Space character
            4. Go back to step i above
        v. If there is no next station: write newline
        vi. To help illustrate output, please refer to the following diagram:
        vii. For example, in the following diagram:



The **exact** string printed that we expect is:

```
Magoun Square U 2 == Ball Square A 1 == College Ave U 0
```

9. getNextStation
    a. Takes in no parameters, returns a station
    b. The aim of this function is to, every time it is called, return the next station in the list. It uses the private variable `currPos` to determine which station to return.
    c. Refer to the following definitions to walk you through the behavior of currPos:
        i. First time we invoke it (or whenever currPos currently NULL)
            1. Returns: first station
            2. currPos should point to the second station
        ii. Second time the function is executed
            1. Returns: second station
            2. currPos should point to third station
        iii. And so on
        iv. Eventually currPos will point to the last station. If we invoke getNextStation one more time we must
            1. Return the last station
            2. Make currPos NULL
        v. If this operation is executed on a list that is empty you return the default station
    d. **Note:** currPos is a variable that keeps track of a virtual train within the station. This variable is undelated to where you should insert or delete in the linked list.
    e. **Note** that there are a couple of functions that interact with currPos already implemented (do not modify them).
    f. If you delete the element that is being pointed by currPos, then currPos should be updated (to NULL)

Style Points

Please make sure that you adhere to the styleGuide document given this semester.

- File headers should be on top of every file: name of the file, your name, pest#, date
- Complete the function headers by not exceeding 80 line of characters.
- Comment properly: comment as if you will pick this work up a year later so that comments will help you understand what you have implemented.
- Remove unused code, print statements.