Programming Assignment 1: Stocks, the Stock Marked and Linked Lists

Due: ~~Wednesday, March 1st at 11:59 pm~~ Saturday, March 11th at 11:59 pm

In this assignment you will:

- implement the Stock class (Stock.h and Stock.cpp)
- implement a List class (List.h)
- implement a stock market class (Market.h and Market.cpp)
- your implementations will be evaluated both on their ability to compile cleanly the tester files and produce output as seen in the files below, and on their adherence to good coding style and practices
- Tester programs:
    - StockTester.cpp, StockTester1.cpp
    - ListTester.cpp
    - MarketTester.cpp
- Output files:
    - StockTester.output.txt, StockTester1.output.txt/
    - ListTester.output.txt
    - MarketTester.output.txt
- A Makefile is also provided.

The Stock class: t

The stock class encapsulates the data associated with a particular stock. Specifically, a name (string), ticker symbol(string), price(double) and quantity(integer). The below listed methods should be provided. Remember to make appropriate use of the const modifier :

- **constructors (3 versions)**
  - ~~a no-argument constructor that sets the data members as follows: name to the string "null", ticker to the string "nul", price to 0.0 and quantity to 0.~~
  - ~~A constructor that receives the data members in the order name, ticker symbol, price and quantity.~~
    - **replace both of these with a single constructor that takes 4 arguments and sets the default parameters as described in the video.**
  - ~~A copy constructor that receives a constant reference to a Stock as a parameter~~
    - **as we saw in the video, the compiler automatically generates a copy constructor for us!**
- "getter" methods for the data members: **getName**, **getTicker**, **getPrice**, **getQuantity**
  - return appropriate data types
- "setter" methods for the data members: **setName**, **setTicker**, **setPrice**, **setQuantity**
  - void methods, appropriate data types
- a **getKey**() method - supports the retrieval of stock by a key value in the list class and stock market classes; behavior identical to getTicker
- overloaded operators for comparison of two Stock objects based on their key values, for the operators <, >, ==, >=, and <=

(Note: you could implement these as member functions or as non-member functions. EK implemented them as member functions in her solution.)

- overloaded operator for comparison of a Stock by its key value with a ticker symbol (string) using the == operator (you'll have two versions of operator==).

(Note: this one could also be implemented as either a member function or as a non-member function. EK implemented it as a member function in her solution.)

- overloaded operator<< that produces output as seen in the StockTester3.output.txt file.

(See the video demo for the Date class and apply that approach to the Stock class to get the same output format as seen in the StockTester3.output.txt.)

The StockList class:

The StockList class provides operations for a list of nodes, each which contains a pointer to a Stock and a pointer to the next node in the list. The list should be maintained in order by the key values of the stocks, and it should not permit duplicates.

The only data member of the StockList class is "head", a pointer to a StockNode. The node class, StockNode.h, is provided for you (contains both interface and implementation).

You should define the files StockList.h and StockList.cpp. Be sure to make appropriate use of the **const** modifier.

The following methods should be implemented:

- a **constructor** that sets head to nullptr
- a **destructor** that deletes each node in the list
- **getHead**(), which returns the value of head
- void **insert**(Stock *sPtr):
    - if the item to be inserted is already represented in the list, do nothing
    - otherwise, create a new StockNode that points to the Stock object indicated by the parameter sPtr. Place the new StockNode at the appropriate location in the list, based on the key value of the Stock. Note: this is where the overloaded operators for the Stock class are useful. If you find that additional overloaded operators for the Stock class would be useful to your implementation, you may add them to the Stock class.
- void **remove**(Stock *sPtr):
    - removal is based on the key found in the Stock object pointed to by sPtr
    - if the list does not contain that key value, do nothing
    - otherwise, remove the Stock node that refers to the Stock with the same key as *sPtr
- void **remove**(string key):
    - if the list does not contain a node with that key, do nothing
    - otherwise, remove the node containing the value with that key from the list
    - Note: you do not need to implement the **remove** logic twice. You can have one version of your remove method simply invoke the other
- StockNode *__find__(string key)
    - returns a pointer to the StockNode in the list that refers to the Stock with the key value
    - or a nullptr if not found
- bool **contains**(Stock *sPtr)
    - return true if the list contains a Stock with the key value found in *sPtr
    - else return false
- bool **contains**(string key)
    - return true if a node with the specified key is found in the list
    - else return false
    - Note: you do not need to implement the **contains** logic twice. You can have one version of your remove method simply invoke the other
- void **display**()
    - if the list is empty, display "empty list" as seen in the sample output
    - else, traverse the list, displaying the value portion of each node

The **Market class:**

The Market class encapsulates the data associated with a simulated stock market. Specifically, a list of stocks in this market (using your StockList class), the name(string) of the market, and a current day(int) for the market values listed.

The class definition should be supplied in Market.h and the class implementation should be supplied in Market.cpp.
Remember to make appropriate use of the **const** modifier. The following methods should be provided:

- **constructor(s) -** invocation of the no-argument constructor should sets the market's name to "Default" and the current day to 0. Invocation of the constructor with a single string argument should set the market's name to the supplied string and the current day to 0. Invocation of the constructor with a string and an integer should set the market's name to the supplied string and the current day to the supplied value.
- **readStockFile**(int dayNum) - read in the file named $MARKET_NAME.$DAYNUM.txt, where $MARKET_NAME is this market's stored name, and $DAYNUM is the dayNum parameter. The method should generate a string containing the name of the file, open the file, and then read each line of the file, creating and adding the stock described. The files EKSE.0.txt and Default.0.txt provide examples of the input file format. Be sure to close the file.
- **readStockFile**() - performs as above, but uses currentDay as the day number in opening the file.
- **writeStockFile**(int dayNum) - writes the market's list of stocks out to a file named $MARKET_NAME.$DAYNUM.txt, where $MARKET_NAME is this market's stored name, and $DAYNUM is the dayNum parameter. The method should generate a string containing the name of the file, open the file, and then write out each stock to the file. The files EKSE.1.txt and Default.1.txt provide examples of the output file format (same as the input file format). Be sure to close the file.
- **writeStockFile**() - performs as above, but uses currentDay as the day number in opening the file.
- void **showStocks**() - should produce output as seen in MarketTester.output.txt. Be sure to make maximal use of Stock and StockList class methods.
- void **step**() - should take a "step" of the market, adjusting the price of each stock in the list "randomly" up or down in the range +/- 5%. (See the video tutorial for help). Be sure that you update the price of the stock in the list (not just a copy).
- int **nextDay**() - increments the current day
- int **getDay**() -returns the current day number
- void **addStock**(Stock *s) - adds a Stock to the market's list (actually, adds a StockNode that points to that stock).
- void **removeStock**(string t) - removes a Stock object with the matching key from market's list; does nothing if no matching key is found (removes and deletes the corresponding StockNode and the Stock it refers to).
- Stock ***getStockPtr**(string t) - returns a pointer to the stock in the list with the matching key value. For now, should only be invoked if the list contains the provided symbol. In a later iteration we'll deal with this case via exception handling.

Submission: Via Canvas, submit a zip file (Proj1.zip) containing **only** the following files:

- Stock.h
- Stock.cpp
- StockList.h
- StockList.cpp
- Market.h
- Market.cpp

We'll combine this with our own Makefile and provided code (StockNode.h and tester files).