ECE 71/191T – Data Structures and Algorithms

Dr. Gregory R. Kriehn, Fresno State
C++ Homework Assignment: Chapter 13

Code Due By: Midnight on Mon, Mar 13
Write-up Due By: Class on Tue, Mar 14

HOMEWORK #27 – Stock Market

Write a program to help a local stock trading company automate its systems. The company invests only in the stock market. At the end of each trading day, the company would like to generate and post the listing of its stocks so that investors can see how their holdings performed that day. Assume that the company invests in, say, up to 50 different stocks.

The input data is provided in a file called stockdata.txt in the following format:

```
symbol openingPrice closingPrice todayHigh todayLow
     prevClose volume
```

For example, the sample data may be:

```
MSMT 112.50 115.75 116.50 111.75 113.50 6723823
CBA 67.50 75.50 78.75 67.50 65.75 378233
```

The first line indicates that the stock symbol is MSMT, today's opening price was $112.50, the closing price was $115.75, today's high price was $116.50, today's low price was $111.75, yesterday's closing price was $113.50, and the number of shares currently being held is 6,723,823.

The desired output is to produce two listings, one sorted alphabetically by stock symbol and another sorted by percent gain from the highest to lowest.

The listing by stock symbol must be of the following form:

```
          *********  First Investor's Heaven   **********
          *********     Financial Report       **********

Stock   Today   Today   Today   Today   Previous  Percent
Symbol  Open    Close   High    Low     Close     Gain        Volume
------  ------  ------  ------  ------  --------  -------     ------
   ABC  123.45  130.95  132.00  125.00    120.50    8.67%     10,000
  AOLK   80.00   75.00   82.00   74.00     83.00   -9.64%      5,000
  CSCO  100.00  102.00  105.00   98.00    101.00    0.99%     25,000
   IBD   68.00   71.00   72.00   67.00     75.00   -5.33%     15,000
  MSET  120.00  140.00  145.00  140.00    115.00   21.74%     30,920

Closing Assets: $9,628,300.00
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Likewise, the listing by percent gain must be of the following form:

```
           *********  First Investor's Heaven   **********
           *********     Financial Report       **********
           *********        Gain/Loss           **********

Stock   Today   Today   Today   Today   Previous  Percent
Symbol  Open    Close   High    Low     Close     Gain          Volume
------  -----   -----   -----   -----   --------  -------       ------
  MSET  120.00  140.00  145.00  140.00    115.00   21.74%       30,920
   ABC  123.45  130.95  132.00  125.00    120.50    8.67%       10,000
  CSCO  100.00  102.00  105.00   98.00    101.00    0.99%       25,000
   IBD   68.00   71.00   72.00   67.00     75.00   -5.33%       15,000
  AOLK   80.00   75.00   82.00   74.00     83.00   -9.64%        5,000
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

You will design this program in several steps.

Specifications:

1. Start by implementing the **`newString`** class (both the `.h` and `.cpp` files) described in **`PROGRAMMING EXAMPLE: newString`** in Chapter 13, pp. 960-963. The class will be used to store, copy, and pull strings from the data file, print strings to the screen, etc.

2. Design and implement a **`stockType`** class (both the `.h` and `.cpp` files) to store individual stock objects.

   The main components of a stock object are the stock symbol, stock price, and number of shares. However, the object must also store the opening price, closing price, high price, low price, previous price, and the percent gain/loss for the day. These variables should be stored as private members of the class.

   ```
   private:
       newString stockSymbol;
       double todayOpenPrice;
       double todayClosePrice;
       double todayHigh;
       double todayLow;
       double yesterdayClose;
       double percentChange;
       int noOfShares;
   ```

   The class should also be able to set the stock information, get the stock symbol, open price, close price, high price, low price, previous price, and the number of shares, as well as calculate the percent gain/loss for the day. Use the following public members:

   ```
   public:
       stockType();
   ```

```
stockType(newString symbol, double openPrice,
      double closePrice, double High, double Low,
      double prevClose, int shares);

void setStockInfo(newString symbol, double openPrice,
      double closePrice, double High, double Low,
      double prevClose, int shares);

newString getSymbol();
double getPercentChange();
double getOpenPrice();
double getClosePrice();
double getHighPrice();
double getLowPrice();
double getPrevPrice();
int getNoOfShares();
```

Since the natural order of the stock list is to be ordered by stock symbol, overload all of the relational operators appropriately:

```
bool operator==(const stockType &other);
bool operator!=(const stockType &other);
bool operator<=(const stockType &other);
bool operator>=(const stockType &other);
bool operator>(const stockType &other);
bool operator<(const stockType &other);
```

Finally, the class should be able to extract or print all relevant stock information from or to an input or output stream. Recall that the << and >> operators must be implemented as friend functions:

```
friend ostream& operator<< (ostream&, const
      stockType&);
friend ifstream& operator>> (ifstream&, stockType&);
```

If `inFile` is an ifstream object, and `myStock` is a stock object, then

```
inFile >> myStock;
```

reads all data from a given line in the text file and stores it in the object `myStock`. Likewise, the << operator should print all relevant stock information to the output stream.

3. A list of stock objects will be implemented in a class called **`stockListType`**. However, **`stockListType`** will be derived from the class **`listType`**, which should be implemented as a class template. The declaration of the **`listType`** class is provided by the following:

```
template <class Type>
class listType
{
    public:
        bool isEmptyList() const;
        // Function returns a nonzero value (TRUE) if list
        // is empty. Otherwise it returns the value 0
        // (FALSE).

        bool isFullList() const;
        // Function returns a nonzero value (TRUE) if list
        // is full.  Otherwise it returns the value 0
        // (FALSE).

        void setLength(int len);
        // Function that sets the length of a list.
        // Postcondition:  length = len

        int showLength() const;
        // Function that returns the length of a list
        // Postcondition:  returns length

        void search(Type searchItem) const;
        // Search the list for searchItem
        // Postcondition:  Internal variable found is set
        // to a nonzero value (TRUE) if searchItem is found
        // in the list, otherwise found is set to 0
        // (FALSE). If the searchItem is found, the value
        // and location within the list is
        // printed to the screen, otherwise, a message
        // indicating that the item cannot
        // be found should be printed.

        void insert(Type newElement);
        // Insert newElement in the list
        // Prior to insertion list must not be full
        // Postcondition:  List is old list plus the
        // newElement

        void deleteItem(Type deleteElement);
        // If deleteElement is found in the list it is
        // deleted
        // If list is empty output the message:
        // "Cannot delete from the empty list"
        // Postcondition:  List is old list minus the
        // deleteItem if deleteItem is found in the list
```

```
    void sort();
    // Sort the list
    // Precondition:  List must exist
    // Postcondition:  List elements are in ascending
    // order

    void print() const;
    // Output the elements of the list

    void getList(ifstream &);
    // Read and store elements in the list
    // Postcondition:  length = number of elements in
    // the list
    // elements = array holding the input data

    void destroyList();
    // Postcondition:  length = 0

    void printList() const;
    // Output the elements of the list

    listType(int listSize);
    // Constructor with parameters
    // Create an array of size specified by the
    // parameter listSize
    // Postcondition:  elements contains the base
    // address of the array, length = 0, and
    // maxsize = listSize

    listType();
    // Default constructor
    // Create an array of 50 components
    // Postcondition: elements contains the base
    // address of the array, length = 0, and
    // maxsize = 50

    ~listType();
    // Destructor
    // Delete all elements of the list
    // Postcondition:  Array elements is deleted

protected:
    void binarySearch(Type searchItem, int &found, int
           &index);
    int maxSize;
    // Maximum number that can be stored in the list
```

```
        int length;
        // Number of elements in the list
        Type *elements;
        // Pointer to the array that holds list elements
};
```

Implement the template definitions for each member function of the class, as appropriate.

NOTE: `binarySearch()` should implement a binary search, which has yet to be covered in class. For convenience, the definitions of the member functions **`search()`** and **`binarySearch()`** are provided here:

```
//-----------------------------------------------------------//
template <class Type>
void listType<Type>::search(Type searchItem) const
{
    int found;
    int index;

    binarySearch(searchItem, found, index);

    if (found)
        cout << "Item" << searchItem
             << "is in the list at" << index << endl;
    else
        cout << "Item" << searchItem
             << "is not in the list" << endl;
} // End search
//-----------------------------------------------------------//

//-----------------------------------------------------------//
template <class Type>
void listType<Type>::binarySearch(Type searchItem, int
&found, int &index)
{
    int first = 0;
    int last = length - 1;
    int mid;

    found = 0;

    while (!found && (first<=last) )
    {
        mid = (first+last)/2;

        if (elements[mid] == searchItem)
            found = 1;
```

```
        else if (elements[mid] > searchItem)
            last = mid - 1;
        else
            first = mid + 1;
    }

    if (found)
        index = mid;
    else
        index = -1;
} // End binarySearch
//--------------------------------------------------------//
```

4. Implement the derived class **`stockListType`**. Since the class only works with a list of stock objects, the class should not be declared as a template. Furthermore, objects of type **`stockListType`** should be declared as arrays, since the purpose of the list is to store all information about each stock in an ordered format.

The following statement will derive the class **`stockListType`** from the class **`listType`**:

```
class stockListType:public listType<stockType>
{
     Member List
};
```

Use the following public and private members for the class:

```
public:
    void printBySymbol();
    void printByGain();
    void printReports();
    void sortBySymbol();
    void sortByGain();

    stockListType();
    stockListType(int size);

private:
    int *sortIndiciesGainLoss;
```

The **`printBySymbol()`** and **`printbyGain()`** member functions print the list of stocks by symbol order or by gain order, as appropriate. To print both, use **`printReports()`**. To order the list by symbol or by gain, the **`sortBySymbol()`** and **`sortByGain()`** functions are used. Because the member variables used to hold the list elements, the length of the list, and the maximum size of the list were declared as

protected members off the class **`listType`**, the members can be directly accessed in class **`stockListType`**.

Since the company also requires you to produce the list ordered by the percent gain/loss via **`printByGain()`** and **`sortByGain()`**, you need to sort the stock list appropriately. However, you will not physically sort the list by percent gain/loss, as the list should have already been sorted by symbol order via the **`sortBySymbol()`** function. Instead, you will provide a logical ordering via the **`sortIndiciesGainLoss`** pointer.

The **`sortIndiciesGainLoss`** pointer should be used to create a dynamic array of integer values whose length is equal to the number of stocks in your data file. Each element of the array will store the appropriate index from your **`stockListType`** object to ensure that each stock is printed in the correct order by percent gain/loss.

5. Write a program prompts the user for the name of a stock data file, and uses file to read in each stock into your **`stockListType`** object. Then sort the list by symbol and by gain, and print the list by symbol and by percent gain/loss.