CS302

Assignment 8: Stock Brokerage Account

Description

As Elon Musk always says, the rich invest. And what simpler way to invest money other than the stock market? Yes and put it all on crypto currency, nothing can go wrong with that. It can be a challenge to day trade while having other obligations, thus we will design the optimal way to invest, using a greedy method. First you need to create the following custom type.

```
struct stockType
{
  std::string name;
  int timesPurchased;
  double dividend;
  double price;

  bool operator?(const stockType& rhs)
  {

  }
};
```

Each stock contains a name, amount of times purchased, dividend, and its price. The operator will be needed for comparison to maintain a priority queue of available stocks (so we pick the next best available stock). The operator will allow the priority queue determine which stock has the higher priority. Between any two stocks, the stock with the higher priority will be

1. The stock with the cheapest price has the higher priority
2. If the prices are the same, the stock that was purchased more has the higher priority
3. If the amount of times purchased is the same, the stock with larger dividend has the higher priority
4. If the dividend amount is the same, the stock that would go earlier in the alphabet has the higher priority

Below is the priority queue class that is either a min or max binary heap

```
template <class Type>
class priorityQ
{
public:
  priorityQ(int = 10);
  priorityQ(vector<Type>);
  priorityQ(const priorityQ<Type>&);
  ~priorityQ();
  const priorityQ<Type>& operator=(const priorityQ<Type>&);
  void insert(Type);
  void deletePriority();
  Type getPriority() const;
  bool isEmpty() const;
  void bubbleUp(int);
  void bubbleDown(int);
  int getSize() const;
private:
  int_t capacity;
  int_t size;
  Type * heapArray;
};
```

Each member contains/performs the following

- `Type * heapArray` - a dynamic array of elements contained in the heap structure
- `int capacity` - denotes the max size of the heap structure
- `int size` - denotes the amount of elements stored in the heap
- `priorityQ<Type>::priorityQ(int cap)` - constructor that sets the capacity of the heap with cap, allocates the heapArray with this capacity and sets the size variable
- `priorityQ<Type>::priorityQ(vector<Type> v)` - constructor that sets the capacity with v.size() + 1 (if you have index 1 as your root), sets the size with the same amount as capacity, assigns each element of the vector into the heapArray, and performs the steps that implements build heap
- `priorityQ<Type>::priorityQ(const priorityQ<Type>& copy)` - copy constructor
- `priorityQ<Type>::~priorityQ()` - destructor
- `const priorityQ<Type>& priorityQ<Type>::operator=(const priorityQ<Type>& rhs)` - assignment operator
- `void priorityQ<Type>::insert(Type item)` - assigns the item to the back of the heapArray (size denotes the back of the heap array), increments size and then bubbles up the element (must double the size of the heapArray if maxed out)
- `void priorityQ<Type>::deletePriority()` - overwrites the root with the last element in the heap, decrements the size by 1, and then bubbles down the element
- `Type priorityQ<Type>::getPriority() const` - returns the root item
- `bool priorityQ<Type>::isEmpty() const` - returns `true` if the heap is empty and `false` otherwise, the value of size will tell you if the heap is empty or not
- `void priorityQ<Type>::bubbleUp(int index)` - bubbles up the element at heapArray[index], compares the item with its parent and swaps heapArray[index] with its parent if violates heap order, continues this as long as the heap order is not maintained
- `void priorityQ<Type>::bubbleDown(int index)` - bubbles down the element at heapArray[index] with one of its children (larger or smaller child), swaps heapArray[index] with the left or right child and continues this process as long as the array is not in heap order
- `int priorityQ<Type>::getSize() const` - returns the amount of elements in the heap array

When writing the code for bubble up or bubble down, you need to use the overloaded operator of the `stockType`, do not use the dot operator to access the price field, so heapArray[x].price would be be implementation since it would be too specific for this type, this is a templated class so it needs to work on potentially any data type with the appropriate operator overloaded

Contents of Main

In main, you will be given two input files (both are comma separated .csv files). The first file contains a list of n stocks with their dividends, each line contains the stock name on the first column and the dividend amount on the second column. The another .csv file will given, which is be the stock exchange file that contains the stock price at the beginning and end of each day for that particular stock exchange. This will have n columns, each column gives the price of each stock (in the same order as they appear in the stocks.csv file). Each pair of 2 lines denotes each stock price at the beginning and end of a day. Also read in the amount of days to simulate and the initial amount assign to the brokerage account.

The steps your program will take

1. Read one line from the stock exchange file to store each stock's price at the start of the day
2. Insert all the stocks into the priority queue
3. If the priority queue is not empty and you have the available funds to buy the next available stock in the priority queue, add the price of the stock to a variable or decrease the profit amount, increment the stock purchased amount and remove this stock from the heap, and run this step again if possible
4. Read the next line from the stock exchange file to store each stock's price at the end of the day and compute the gain/losses for the day, output the gain/losses and update your brokerage account by adding the gain/losses
5. If more days needed to be simulated and you still have available funds, return back to step 1 otherwise continue to the next step
6. Add up the dividend amount by multiplying each stock's dividend percentage with the amount of times it was bought, add those up and output this amount along with the total amount (the brokerage account amount plus the dividends)

Sample Run

```
Output files provided on canvas
```

Specifications

- No memory leaks
- Do not modify or add any public functions
- Use a dynamic array for the heapArray (not a vector)
- Use the `stockType` operator to compare parent/children in the heapArray, do not access any of the members of the `stockType` within the `priorityQ` class since that would be a bad template implementation

Submission

Submit the source files to code grade by the deadline

References

- Supplemental Video https://youtu.be/ooyFrBDcJIQ
- Link to the top image can be found at https://www.downloadclipart.net/browse/90129/stock-market-png-transparent-image-clipart