Assignment 10

Due Dec 2 by 7pm **Points** 120

Please note that there are two separate submission folders for this week's projects - one for 10.a and one for 10.b. For project 10.b, remember to submit your makefile to TEACH together with the files for 10.b.

Project 10.a

Write a class called Board that represents a tic-tac-toe board. It should have a 3x3 array as a data member, which will store the locations of the players' moves. It should have a default constructor that initializes the 3x3 array to being empty. It should have a method called makeMove that takes the x and y coordinates of the move (see the example below) and which player's turn it is as parameters. If that location is unoccupied, makeMove should record the move and return true. If that location is already occupied, makeMove should just return false. There should be a method called gameState that takes no parameters and returns a value indicating one of four possibilities: 'x' has won, 'o' has won, the game is a draw, or the game is still in progress - use an **enum** for this (the enum definition should go in Board.hpp). There should also be a method called print, which just prints out the current board to the screen.

Write a class called TicTacToe that allows two people to play a game. This class will have a field for a Board object and a field to keep track of which player's turn it is. It should have a constructor that takes a char parameter that specifies whether 'x' or 'o' should have the first move. It should have a method called play that starts the game. The play method should keep looping, asking the correct player for their move and sending it to the board (with makeMove) until someone has won or it's a draw (as indicated by gameState), and then declare what the outcome was.

Write a main method (in TicTacToe.cpp) that asks the user which player should go first, creates a new TicTacToe object and starts the game. **For this project (10.a) only, you will not comment out your main method.**

Input validation: If someone tries to take an occupied square, tell them that square is already occupied and ask for a different move.

Here's an example portion of a game (already in progress):

```
  0 1 2

0 x . .

1 . . .

2 . . .
```

Player O: please enter your move.

```
1 2
```

```
  0 1 2

0 x . .

1 . . o

2 . . .
```

Player X: please enter your move.

```
1 2
```

That square is already taken.

```
  0 1 2

0 x . .

1 . . o

2 . . .
```

Player X: please enter your move.

The files must be named: **Board.hpp**, **Board.cpp**, **TicTacToe.hpp** and **TicTacToe.cpp**

Project 10.b

You will be writing a (rather primitive) online store simulator. It will have three classes: Product, Customer and Store. To make things a little simpler for you, I am supplying you with the three .hpp files. You will write the three implementation files. You should not alter the provided .hpp files.

Here are the .hpp files: **Product.hpp** , **Customer.hpp** and **Store.hpp**

Here are descriptions of methods for the three classes:

Product:

A Product object represents a product with an ID code, title, description, price and quantity available.

- constructor - takes as parameters five values with which to initialize the Product's idCode, title, description, price, and quantity available
- get methods - return the value of the corresponding data member
- decreaseQuantity - decreases the quantity available by one

Customer:

A Customer object represents a customer with a name and account ID. Customers must be members of the Store to make a purchase. Premium members get free shipping.

- constructor - takes as parameters three values with which to initialize the Customer's name, account ID, and whether the customer is a premium member
- get methods - return the value of the corresponding data member
- isPremiumMember - returns whether the customer is a premium member
- addProductToCart - adds the product ID code to the Customer's cart
- emptyCart - empties the Customer's cart

Store:

A Store object represents a store, which has some number of products in its inventory and some number of customers as members.

- addProduct - adds a product to the inventory
- addMember - adds a customer to the members
- getProductFromID - returns product with matching ID. Returns NULL if no matching ID is found.
- getMemberFromID - returns product with matching ID. Returns NULL if no matching ID is found.

- productSearch -for every product whose title or description contains the search string, prints out that product's title, ID code, price and description. The first letter of the search string should be case-insensitive, i.e. a search for "wood" should match Products that have "Wood" in their title or description, and a search for "Wood" should match Products that have "wood" in their title or description.
- addProductToMemberCart **-** If the product isn't found in the inventory, print "Product #***[idCode goes here]*** not found." If the member isn't found in the members, print "Member #***[accountID goes here]*** not found." If both are found and the product is still available, calls the member's addProductToCart method. Otherwise it prints "Sorry, product #***[idCode goes here]*** is currently out of stock." The same product can be added multiple times if the customer wants more than one of something.
- checkOut **-** If the member isn't found in the members, print "Member #***[accountID goes here]*** not found." Otherwise prints out the title and price for each product in the cart and decreases the available quantity of that product by 1. If any product has already sold out, then on that line it should print 'Sorry, product #***[idCode goes here]***, "***[product name goes here]***", is no longer available.' At the bottom it should print out the subtotal for the cart, the shipping cost ($0 for premium members, 7% of the cart cost for normal members), and the final total cost for the cart (subtotal plus shipping). If the cart is empty, it should just print "There are no items in the cart." When the calculations are complete, the member's cart should be emptied.

Here is an example of how the output of the Store::productSearch method might look (searching for "red"):

```
red blender
ID code: 123
price: $350
sturdy blender perfect for making smoothies and sauces

hot air balloon
ID code: 345
price: $700
fly into the sky in your own balloon - comes in red, blue or chartreuse
```

Here is an example of how the output of the Store::checkOutMember method might look:

```
giant robot - $7000
Sorry, product #345, "live goat", is no longer available.
oak and glass coffee table - $250
Subtotal: $7250
Shipping Cost: $0
Total: $7250
```

You must submit on TEACH: **Product.cpp**, **Customer.cpp**, and **Store.cpp**. You do not need to submit the .hpp files.

In the main method you use for testing, you should only need to #include Store.hpp. Remember that your compile command needs to list all of the .cpp files.