Lab #2 – Class Definitions & Header Files

Handed out: Tue, 9/25/2018	Due date: Tue, 10/2/2018

Goal

The goal of this assignment is to get you started with class definitions, as well as with using header files.

Submission instructions

Inside that the homework subdirectory, create a directory for homework #2, and call it `lab2`. When you finish the assignment, go to the homework directory and submit directly to our TA as follows:

```
$ submit jwang lab2 lab2
```

Please note that starting with this homework, your submission should include a Makefile!

Problems

1. `Account` Class
   - Create an `Account` class that a bank might use to represent customers' bank accounts.
   - Include a data member of type `double` to represent the account balance.
   - Provide a constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to ensure that it's greater than or equal to 0. If not, set the balance to 0 and display an error message indicating that the initial balance was invalid.
   - Also provide a default constructor that creates an account with the balance of zero.
   - Provide the following member functions:
     - Member function `credit` should add an amount to the current balance.
     - Member function `debit` should withdraw money from the `Account` and ensure that the debit amount does not exceed the Account's balance. If it does, the balance should be left unchanged and the function should print a message "Debit amount exceeded account balance."
     - Member function `getBalance` should return the current balance.
     - In addition to the validity checking described in the exercise, your `debit()` and `credit()` functions should insure that the debit or credit amount is positive.
     - Member function `addAccountBalance` should allow the user to add the balances in two accounts and return the total.

- Create a program that tests the member functions of class Account. Your user code should do the following:
  - Create two accounts, one using the default constructor, one using the non-default constructor with a balance of 100 dollars.
  - Get a withdrawal amount from the user for each account, debit the accounts and display the new balances.
  - Get a deposit amount from the user for each account, credit the accounts and display the new balances.
  - Display the total amount of money in both accounts.
- Your implementation should include a header defining the class interface, a class implementation file, and a file with client code. Submit your code in three files: Account.h, Account.cc, and account_client.cc.

2. `Invoice` Class. Create a class called `Invoice` that a hardware store might use to represent an invoice for an item sold at the store.
   - An `Invoice` should include four data members – a part number (type string), a part description (type string), a quantity of the item being purchased (type int) and a price per item (type double).
   - Your class should have a constructor that initializes the four data members.
   - Provide a set and a get function for each data member.
   - Provide a member function named `getInvoiceAmount` that calculates the invoice amount (multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.
   - Following the example from Problem 1, design the appropriate test functions for your implementation yourself. Submit your code in three files called Invoice.h, Invoice.cc, and invoice_client.cc

Grading Criteria

The grading procedure for this assignment will be as follows:

1. Run the provided user code to check if the specified functionality was implemented.
2. Run a test suite for each problem.
3. Read the code to examine implementation details.

The following will be checked for each of the problems:

Problem 1

- Does the non-default constructor validate the initial balance and set it to zero if negative, displaying the error message?
- Do debit() and credit() member function check that the debited or credited amount is positive?

Problem 2

- Do the constructor and setter for quantity and price do the validation?
  - If the quantity is not positive, it should be set to zero
  - If the price is not positive, it should be set to zero
- Do the test functions test the functionality along with boundary cases, i.e.

  1) creating an invoice with a non-positive quantity

  2) creating an invoice with a non-positive price