Assignment 3

- Carefully read all questions (2-3 times) and then start implementation!
- Submit well written C++ code in 3-file-structure (.h,.cpp)
- Avoid Ctrl + C and Ctrl + V
- Example outputs are also provided for better understanding of problem statements. (You can use your own input data)
- Due Date: 20-April-2020 (11:55 pm) (Monday)

Q No. 1: Find the errors in the following class and explain how to correct them:

```
1   class Example
2   {
3       public:
4           Example( int y = 10 )
5           : data( y )
6           {
7               // empty body
8           } // end Example constructor
9           int getIncrementedData() const
10          {
11              return ++data;
12          } // end function getIncrementedData
13          static int getCount()
14          {
15              cout << "Data is " << data << endl;
16              return count;
17          } // end function getCount
18      private:
19          int data;
20          static int count;
21  }; // end class Example
```

Q No. 2: (SavingsAccount Class)

- Create a ***SavingsAccount*** class. Use a static data member *annualInterestRate* to store the annual interest rate for each of the savers. Each member of the class contains a private data member *savingsBalance* indicating the amount the saver currently has on deposit.

- Provide member function *calculateMonthlyInterest* that calculates the monthly interest by multiplying the balance by *annualInterestRate* divided by 12; this interest should be added to *savingsBalance*.

- Provide a static member function *modifyInterestRate* that sets the static *annualInterestRate* to a new value.

- Write a driver program to test class *SavingsAccount*. Instantiate two different objects of class SavingsAccount, *saver1* and *saver2*, with balances of $2000.00 and $3000.00, respectively. Set the *annualInterestRate* to 3 percent.

- Then calculate the monthly interest and print the new balances for each of the savers.
- Then set the *annualInterestRate* to 4 percent, calculate the next month's interest and print the new balances for each of the savers.
- Example output looks like:

```
Initial balances:
Saver 1: $2000.00       Saver 2: $3000.00

Balances after 1 month's interest applied at .03:
Saver 1: $2005.00       Saver 2: $3007.50

Balances after 1 month's interest applied at .04:
Saver 1: $2011.68       Saver 2: $3017.53
```

Q No. 3: (Package Inheritance Hierarchy)

Package-delivery services, such as FedEx®, DHL® and UPS®, offer a number of different shipping options, each with specific costs associated. Create an inheritance hierarchy to represent various types of packages.

- Use class ***Package*** as the base class of the hierarchy, then include classes ***TwoDayPackage*** and ***OvernightPackage*** that derive from Package.
  - Base class *Package* should include data members representing the name, address, city, state and ZIP code for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package.
  - Package's constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values.
  - Package should provide a public member function *calculateCost* that returns a double indicating the cost associated with shipping the package. Package's *calculateCost* function should determine the cost by multiplying the weight by the cost per ounce.

- Derived class *TwoDayPackage* should inherit the functionality of base class *Package*, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service.

- *TwoDayPackage*'s constructor should receive a value to initialize this data member. *TwoDayPackage* should redefine member function *calculateCost* so that it computes the shipping cost by adding the flat fee to the weight-based cost calculated by base class Package's calculateCost function.

- Class *OvernightPackage* should inherit directly from class *Package* and contain an additional data member representing an additional fee per ounce charged for overnight-delivery service. *OvernightPackage* should redefine member function *calculateCost* so that it adds the additional fee per ounce to the standard cost per ounce before calculating the shipping cost.

- Write a test program that creates objects of each type of *Package* and tests member function *calculateCost.* (i.e. Package package1, TwoDayPackage package2, OvernightPackage package3)

- Example output looks like:

```
Package 1:

Sender:
Lou Brown
1 Main St
Boston, MA 11111

Recipient:
Mary Smith
7 Elm St
New York, NY 22222

Cost: $4.25

Package 2:

Sender:
Lisa Klein
5 Broadway
Somerville, MA 33333

Recipient:
Bob George
21 Pine Rd
Cambridge, MA 44444

Cost: $8.82

Package 3:

Sender:
Ed Lewis
2 Oak St
Boston, MA 55555

Recipient:
Don Kelly
9 Main St
Denver, CO 66666

Cost: $11.64
```

Q No. 4: (Account Inheritance Hierarchy)

Create an inheritance hierarchy that a bank might use to represent customers' bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts. More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit).

- Create an inheritance hierarchy containing base class **Account** and derived classes **SavingsAccount** and **CheckingAccount** that inherit from class **Account**.

1. Base class *Account* should include one data member of type double to represent the account balance. The class should 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 0.0. If not, the balance should be set to 0.0 and the constructor should display an error message, indicating that the initial balance was invalid.

2. The class should provide three 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 the message "*Debit amount exceeded account balance.*" Member function *getBalance* should return the current balance.

3. Derived class *SavingsAccount* should inherit the functionality of an *Account*, but also include a data member of type double indicating the interest rate (percentage) assigned to the *Account*. *SavingsAccount*'s constructor should receive the initial balance, as well as an initial value for the *SavingsAccount*'s interest rate. *SavingsAccount* should provide a public member function *calculateInterest* that returns a double indicating the amount of interest earned by an account. Member function *calculateInterest* should determine this amount by multiplying the interest rate by the account balance. [Note: *SavingsAccount* should inherit member functions *credit* and *debit* as is without redefining them.]

4. Derived class *CheckingAccount* should inherit from base class *Account* and include an additional data member of type double that represents the fee charged per

transaction. *CheckingAccount*'s constructor should receive the initial balance, as well as a parameter indicating a fee amount. Class *CheckingAccount* should redefine member functions *credit* and *debit* so that they subtract the fee from the account balance whenever either transaction is performed successfully. *CheckingAccount*'s versions of these functions should invoke the base-class *Account* version to perform the updates to an account balance. *CheckingAccount*'s debit function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance). [Hint: Define *Account*'s debit function so that it returns a bool indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.]

- After defining the classes in this hierarchy, write a program that creates objects of each class and tests their member functions. Add interest to the *SavingsAccount* object by first invoking its *calculateInterest* function, then passing the returned interest amount to the object's *credit* function.

- Example Output looks like:

```
account1 balance: $50.00
account2 balance: $25.00
account3 balance: $80.00

Attempting to debit $25.00 from account1.

Attempting to debit $30.00 from account2.
Debit amount exceeded account balance.

Attempting to debit $40.00 from account3.
$1.00 transaction fee charged.

account1 balance: $25.00
account2 balance: $25.00
account3 balance: $39.00

Crediting $40.00 to account1.

Crediting $65.00 to account2.

Crediting $20.00 to account3.
$1.00 transaction fee charged.

account1 balance: $65.00
account2 balance: $90.00
account3 balance: $58.00

Adding $2.70 interest to account2.

New account2 balance: $92.70
```