CSCI 3110 Project 5

You are designing software for an online auction site "BidHere.com". One of the modules in the system is responsible for keeping track of all sellers registered at the site.

Create a **Person** class for a user of the system. Create person.h and person.cpp files for this class. The class should contain the following data of a person in the system:

- first name,
- last name,
- user ID, and
- email address.

These data should be accessible in its derived class. (Define the data with the appropriate access privilege: private, public, or protected)

This class should contain:

- a default constructor,
- a value constructor that receives all of the data to be placed into the data members as its parameters,
- a copy constructor.

The class should also contain:

- "get" methods to allow the client to obtain information about a person.
  - getFirstName, getLastName, getID, getEmail
- "set" methods to allow the client to assign information to a seller.
  - setFirstName, setLastName, setID, setEmail, and optionally setAllInfo
- overloaded operators
  - overloaded == method that returns true if two sellers have identical names and email address; otherwise it returns false.
  - overloaded !=, <, and > operators (for > and < operators, comparisons are based on last names)
- An overloaded assignment = operator.
- Overloaded << and >> operators these be friend functions)
- print() – a virtual function that displays on screen all the information of a person with appropriate messages
- read() – a virtual function that reads in the information of a Person from an input file stream

Designate methods as const methods as appropriate.

A list of system management users of Person type will be read from the data file "users.dat". Here is an example of the data file with two users. (firstname, lastname, ID, and email are one information per line)

Charles
McCartney
ckm2d
cmCartney@gmail.com
Rosa
Johnson
rtj8m
rosa.johnson@mac.com

Use the Person class as the base class, create two derived classes **Seller** class and the **PowerSeller** class. The Seller class inherits from the Person Class, and the PowerSeller class inherits from the Seller class. **Person ➔ Seller ➔ PowerSeller**

Implement the **Seller** class as a derived class of **Person class**. Create the seller.h and seller.cpp files for this class. The **Seller** class contains additional data members that hold a seller's:

- ***average star rating received from the buyers***, and
- ***total number of items sold.***

These data should be accessible by its derived classes. (Make sure to define these data with the appropriate access privilege: private, public, or protected)

The following methods are included in the seller class:

- A default constructor and a copy constructor
- A constructor that receives all of the data for a seller.

(for the constructors, make sure to call the base class constructor as appropriate)

- Appropriate get and set functions for the new data in this class
- print() – overrides the base class print() to print the base class data and print all additional seller data with appropriate messages. Make sure to call the base class print to print the base class data.
- read() – overrides the base class read() to read the base class data and reads all additional seller data items, one data per line in the given order. Make sure to call the base class read to read in the base class data.

  Here is an example data for a Seller

  S ← indicate a seller record
  Joseph ← first name
  Edmondson ← last name
  jed2k ← user ID
  jed2k@apple.com ← email
  3.6 ← seller rating
  1400 ← total number of items sold

Implement a **PowerSeller** class that is derived from the **Seller** class. Create the powerseller.h and powerseller.cpp files for this class. This derived class contains additional data members include:

- ***the seller's own web site address (string),***
- ***number of items sold in the current year.***

Methods included in this derived class are:

- A default constructor and a copy constructor
- A value constructor that receives all data for a power seller
- Appropriate get and set functions for the new data
- print() – overrides the parent class print() method. This method should call the Seller class print method to display all the information from a Person class and the seller class, and also display the additional power seller data with appropriate messages.
- read() – overrides the parent class read() method. This method should call the Seller class read method to read all the information for Person class and seller class, and also read in the information for the specific data for a Power Seller. The addition data: (1) the web site address, (2) the number of items sold in the current year, one per line, follow from the Person and Seller data.

Here is an example data for a Power seller

P	← stands for power seller record
Jack	← First name
Smith	← Last name
jsmith	← user ID
jack.smith@google.com	← email address
4.6	← seller rating
1800	← total number of items sold
https://www.bottledMusic.com	← seller's web site
900	← number of items sold current year

The main program will implement a seller management system with the following pseudocode design:

```
1. Initialization Step: Build the database of all the sellers.

2. Check for valid user.

   //Perform seller management:
3. while (not done)
       Display menu
       Get user choice

       choice 1: Print information of all sellers
       choice 2: Check information of a particular seller
       choice 3: Add a seller to the list
       choice 4: Remove a seller from the list
       choice 5: quit
   end while;
```

Details of each step of the main program is described below:

1. The **initialization** phase:
   - initializes the boolean flag/variable **done**
   - read information of the users from a file users.dat and add them into a list of **users.**
   - read information of the sellers from a file **seller.dat** and add them into a list of **sellers.** The file **seller.dat** will contain information concerning sellers. The first line before every seller will contain a letter 'S' or 'P' meaning that a *seller* is to be read or a *power seller* is to be read.
     - If 'S' is read, then the information will appear for a seller as described above.
     - If 'P" is read, then the data starts with all information as a regular seller, and then followed by the seller's own web site address (string) and the number of items sold in the current year, one per line.
     - All the sellers (whether a regular seller or a power seller) are to be added into the same list
   - **A STL list should be used to store all the sellers.**

2. The **Check for Valid User** function will ask the user for his/her name (first name and last name) . If these match a user in the valid user's list then the function will ask for

a magic password and if the user types is "**LETMEIN"** then they will be allowed to continue using the system. If the person is not a valid user or if the password is wrong, then the user should not be allowed to use the system. The program output an error message and terminates.

3. In the loop that performs seller management**:**
At the beginning of each iteration of the loop, a menu is displayed to
(1) allow the user to print all sellers,
(2) check a seller's record,
(3) add a seller to the list of sellers,
(4) remove a seller from the list, or
(5) quit.
Write a function to display the menu.

Write a separate function for each of the following four choice tasks:

(1) If the user selects the option to print all sellers, then all seller records should be printed. For each seller in the list, all information of that seller is to be displayed, and for a power seller in the list, all information of a power seller is to be displayed. Use dynamic binding to let the system determine whether the seller is just a seller or a power seller. After all the information is displayed, control should be returned to the seller management menu.

(1) If the user selects the option to check a seller's record, then the user should be prompted for the seller's name. If the seller is found, then **all** information pertaining to the seller should be displayed. If the seller is not found, then an error message should be displayed. Use dynamic binding to let the system determine whether the seller is just a seller or a power seller.

(2) If the user selects the option to add a seller, then the user should be asked whether the seller is a power seller or not. If not, then just read a seller's information and add it to the list of sellers. Otherwise, read a power seller's information and add it to the list of sellers.

(3) If the user selects the option to delete a seller, then the user should be asked to enter the firstname and lastname of the seller (or power seller) to be removed. If the seller is found, he/she is removed from the list, otherwise, display an error message.

<a sample run of the program is provided on the course page>