CSI 240

Module 11 Project

Bank Management System

The coded solution to the following problem is to be done by you and only you. You may ask for help only from the professor and the tutors, and no one else. You may use your texts, notes, online tutorials, etc., but the code must be your own. Bear in mind that the professor and tutor are helping you with your C++ problem and not writing your program! If you have trouble compiling or debugging your code, see the professor or tutors as soon as possible. If none of us are available, you may email your question to your professor directly.

Problem Statement

Prolog

You have inherited a bank system written by another programmer.

The following features are currently implemented

- User Management
  - Provide Login/Logout for users.
    - Password verification required.
- Account Management
  - Add a new account
  - Delete an existing account
  - Update customer name on an existing account.
  - Search for an account by account number.
- Funds Management
  - Deposit money into an account
  - Withdraw money from an account
  - Check balance on an account

The system must work continuously until the program is shutdown. Also, a bank teller must be allowed to log in or out of the system at any time.

Problem

The above has been implemented using a more legacy approach utlizing functions and files to organize the functionality. Your job is to upgrade the code base to be class based. In addition you are need to implement additional languages features like operator overloading, inheritance, and polymorphism. In addition, the following features need to be added:

- Transactions
  - Tracking
    - Deposits
    - Widthdrawals
  - Viewing
  - Filtering

Implementation Specification

Teller Data

Each teller is provided with an ID (unique) and password, and this information is used to verify the identity of the teller logging in.

The ID and password are a combination of letters, numbers, underscores and hyphens (yes, no space allowed). This data file must be named as tellers.dat and its format is given as follows. The (ID password) combination for the bank teller is unique.

```
ID Password
```

Customer Data

There is a file used to record of all customers' personal information, and it must be named as `accounts.dat` The format for this file is given as follow:

```
Account Number
Social Security Number
Name
Address
Phone Number
```

- The account number must be unique and it must be all numeric
- The format for SSN should be xxx-xx-xxxx, but it might be difficult to maintain. If you system is able to enforce this format, it is a 5 points bonus.
- The format for phone number should be (xxx) xxx-xxxx, but it might be difficult to maintain. If you system is able to enforce this format, it is a 5 points bonus.

Account Balance Data

The main account data file does not store the balance for the accounts; the balance of each account is stored in a data file that is associated to the account number. For example, if the account number for the customer is `04126`, then the name of the associated data file is `04126.dat`. This file stores only one piece of information that is the current balance for the given account.

Account Transaction Data

The main account data file does not store the transactions for the account; the transactions for each account is stored in a data file that is associated to the account number. For example `04126-t.dat`

This file will store a multiple line record for each transaction.

The first line will indicate the type of transaction.

```
Type (W/D)
```

The second set of lines will be for each type

Deposit

```
Date
Amount
```

Withdrawal

```
Date
Amount
CheckNumber
```

Sample Data

Samples files are provided for the main account and teller.

Windows

Windows stores line endings with Carrige Return (CR) and Line Feed (LF) characters

- accounts-crlf.dat
- tellers-crlf.dat

macOS and linux

These platforms store line endings with just a Line Feed (LF) character.

- accounts-lf.dat
- tellers-lf.dat

Course Material Requirements

You must incorporate the following constructs into your code:

- Classes
  - Inheritance
  - Polymorphism
- Operator Overloading
- Pointers

Additional Requirements

- All data files must be stored in a directory named `data`.
- If any of the data file is missing, your program must not terminate.
  - For example, if tellers.dat is not available, your system must continue to run. The consequence of missing this file is that no one can log in.

Bonus Point Opportunities

- Update additional information on an existing account
  - Your system must be allowed to update customer's name
    - Any additional update is considered as a 5 points bonus each.
- Additional Search options
  - Your system must be allowed to search by account number
    - Any addition search option is considered as a 5 points bonus each.

Final Project Grading Criteria

These are in addition to all of the general grading guildelines found in `Grading.pdf`.

Features

Any of the functionality below is not working correctly

- Add a new account
- Delete an existing account
- Update information on an existing account
- Search account information4
- Deposit money into an account
- Withdraw money from an account
- Check balance on an account

```
 -5: Feature not working correclty
-10: Features Missing
```

General

```
-10: Using global variables, except const variables
-10: Must provide clear instructions and interface for program interaction to user
 -5: Waits and does nothing
 -5: Infinite error message each
-10: Output file's data corrupted
-10: Negative balance - should not let withdraw more money than account has
 -5: Input filename is not read in correctly
 -5: Output filename is not read in correctly
 -5: Input file stream is not opened correctly
 -5: Output file stream is not opened correctly
```

Function Design

When writing software its a good idea to design some of the aspects around how you will interact with each of your modules. Take this set of possible function definitions for dealing with accounts. You will need to decide on your design, and be consistent through out.

```
#include <string>
#include <array>

using namespace std;

// Create an alias for this very verbose type defintion for our array
using account_record = array<string, static_cast<int>(ACCOUNT_DETAIL::SIZE)>;

// Account detail enumeration to help access both C Style and C++ style array elements.
enum class ACCOUNT_DETAIL
{
    accountNumber = 0,
    ssn,
    name,
    address,
    phone,
    SIZE
};

// CREATE
void AddAccount(string accountNumber, string ssn, string name, string address, string phone);
void AddAccount(string accountNumber, string accountDetail[]);
void AddAccount(string accountDetail[]);
void AddAccount(array<string, static_cast<int>(ACCOUNT_DETAIL::SIZE)> accountDetail);

// READ
void GetAccount(string accountNumber, string &ssn, string &name, string &address, string &phone);
void GetAccount(string accountNumber, string accountDetail[]);
void GetAccount(string accouutNumber, array<string, static_cast<int>(ACCOUNT_DETAIL::SIZE)> &accountDetail

// UPDATE
void UpdateAccount(string accountNumber, string name);
void UpdateAccount(string accountDetail[]);
void UpdateAccount(array<string, static_cast<int>(ACCOUNT_DETAIL::SIZE)> &accountDetail);
void UpdateAccount(account_record &accountRecord);

// DELETE
void DeleteAccount(string accountToDelete);

//Example calls
int main()
{
    {
        string accountNumber;
        string ssn;
        string customerName;
        string address;
        string phone;

        accountNumber = "";
        ssn = "";
```

```
        customerName = "";
        address = "";
        phone = "";

        AddAccount(accountNumber, ssn, customerName, address, phone);
    }
    {
        string newCustomer[static_cast<int>(ACCOUNT_DETAIL::SIZE)];
        newCustomer[static_cast<int>(ACCOUNT_DETAIL::accountNumber)] = "01232";
        newCustomer[static_cast<int>(ACCOUNT_DETAIL::ssn)] = "123-23-1233";
        newCustomer[static_cast<int>(ACCOUNT_DETAIL::name)] = "123-23-1233";
        //etc.

        AddAccount(newCustomer);
    }
    {
        array<string, static_cast<int>(ACCOUNT_DETAIL::SIZE)> newCustomer;
        newCustomer[static_cast<int>(ACCOUNT_DETAIL::accountNumber)] = "01232";
        newCustomer[static_cast<int>(ACCOUNT_DETAIL::ssn)] = "123-23-1233";
        newCustomer[static_cast<int>(ACCOUNT_DETAIL::name)] = "customer name";

        AddAccount(newCustomer);
    }
    {
        account_record newCustomer;
        newCustomer[static_cast<int>(ACCOUNT_DETAIL::accountNumber)] = "accountNumber";
        newCustomer[static_cast<int>(ACCOUNT_DETAIL::ssn)] = "123-23-1233";
        newCustomer[static_cast<int>(ACCOUNT_DETAIL::name)] = "customer name";
        //etc.

        AddAccount(newCustomer);
    }
}
```