TCC CSCI 2843

Assignment 7

Description

1) This project will create a **base** account class that has the members:

```
std::string account_code;
std::string first_name;
std::string last_name;
double balance;
```

Provide a constructor that initializes **ALL** members in its initialization list with data passed in as arguments to the constructor. Provide any accessor functions you may need (e.g. to get the account code and balance and to set the balance).

In addition, include two pure **virtual** functions that look like the following:

```
virtual void monthly_update() = 0;
virtual char type() const = 0;
```

2) Design and implement the following **derived** classes that will inherit from account and implement the monthly_update function in the following manner:

Class Name	Implementation
simple_account	balance *= 1.05
bonus_account	balance = balance * (balance > 5000 ? 1.06 : 1.04)
service_account	balance = (balance - 5.0) * 1.04
balanced_account	balance = balance > 500.0 ? balance * 1.05 : (balance - 5.0) * 1.03

Each derived class will also override the type function to return a single character that describes the type as indicated in (3) below instead of storing the character.

3) Create a factory function or class that reads data in the following fixed format:

```
account code : 10 characters
first name : 15 characters
last name : 25 characters
type: 1 character
balance : numeric 8 digits, decimal place, 2 digits
```

and uses that data to dynamically allocate and construct one of the derived objects based on the value of *type* as indicated in the following table:

Type	Class to Dynamically Instantiate
'A'	simple_account
'B'	bonus_account
'C'	service_account
'D'	balanced_account

If the *type* is an 'X', skip the account record. In other words, the account should not be processed by the rest of the program and should not appear in the result.

If the *type* is anything else, log the error to a file named "account.log", but **DO NOT** stop processing the rest of the accounts.

Trim spaces from the right of the first_name and last_name members using the trim library you constructed in assignment 5.

4) Write a **manager** class that has a std::map<std::string, std::unique_ptr<account>> member. Note the use of std::unique_ptr for the "ownership" of the accounts added to the map. This will automatically clean up the accounts when the map goes out of scope.

5) The manager class will have a member function that performs the processing of the accounts. This processing is laid out as follows:

   a. Open an std::ifstream on the file, "account2.dat", which will be supplied to you. If the file does not exist, log this to the "account.log" file and exit the program (presumably using an exception).

   b. For each account in the input file, invoke the factory function/class written in part 3. Each object returned from the factory function will then be inserted into the map, using the account as the key. If the account is to be ignored, or if the factory function throws a fatal exception, no account should be inserted into the map (let alone even created). At the end of the stream, proceed with the next step.

   c. Open another std::ifstream on the transaction file, "transact.dat", which has the following format:

      account : 10 characters
      amount : numeric 8 digits, decimal place, 2 digits

      For each record in the transaction file, find the account in the map and apply the transaction amount (which could be negative) to the balance for that account. Output an error to the "account.log" file if the account is not found in the map, but do NOT stop processing the transaction file.

   d. When all transactions are processed, iterate through the map and invoke the monthly_update function for each account in the map.

   e. Using a std::ofstream, output each account in the map to a new file named "update.dat" in the SAME format as "account2.dat".

   f. Output the accounts in a "human readable format" to a file, "report.txt". The output format is of your choice, but please include all the members of each account per line (including the type).

6) In your program's main function, instantiate your manager class from part 4, and invoke the member function in (5) to process the records. Remember to take care of any exceptions that could possibly "leak" out.