CSCE 121-509,510,511,512: Introduction to Program Design and Concepts
Fall 2016
Project

Phase-1 Project Due Friday, Nov 18, 11:59 PM.
Phase-2 (Final) Project Due Friday, Dec 2, 11:59 PM.
Review the Project General Instructions!

---

1. **Phase-1, Bank Class:** Write code for a **Bank** class which initially need only deal in **USD** (US dollars). To support this **Bank** class you will also need the following:
    - A **Currency** struct to hold information about a particular currency type (a string for the currency type, e.g. "USD", and a double for its exchange rate into USD, e.g. 1.0 for USD's). For **BankUS** we will only deal with **Currency** of type **USD**'s.
    - A **Money** class to hold information about the money objects in the **Bank**, i.e. the money **Currency** type and the money amount (in that **Currency** type). Again for **BankUS** we will only deal with **Money** of **Currency** type **USD**'s. These **Money** objects should be used to track: 1) the total money held by the bank, 2) the money held by any customer of the bank in their account, and 3) the money involved in any transaction done by customers of the bank (i.e. deposits and withdrawals).
    - A **Patron** class to hold information about the customers in the **Bank**, i.e. their name, their account number, their account balance (in USD's). To avoid difficulties with reading into a string variable, use the underscore character (_) in place of spaces for **Patron** names.
    - A **Transaction** struct to hold information about customer deposits and withdrawals in the **Bank**. Each **Transaction** should include: 1) the patron name, account number and resulting account balance, 2) the transaction date (using **Chrono::Date**), 3) the transaction time (using **Chrono::Time**), 4) the transaction type (i.e. Deposit or Withdrawal), and the transaction amount in USD.
    - The **Bank** class should contain a list of **Patron**'s and a list of **Transaction**'s done by the patrons.
    - The complete **Bank** state (**Currency**'s **Money**'s **Patron**'s, **Transaction**'s) should be able to be initialized from a text file, and to be saved to a text file. When the main program first constructs a **Bank** the initialize from file option should be presented to the user, and if the user declines then the bank should be initialized to a state with no **Patron**'s or **Transaction**'s and a fixed amount (e.g. 6000.00) of **Money** of type **USD** held in the bank. When the main program exits (by user menu choice) the save **Bank** to file option should be presented to the user, and if the user declines then no state is saved.
    - **Bank** member/helper functions should be used to appropriately implement the main() program Menu operations listed below.

    Your **BankUSMain** main program should consist of a Menu loop that repeatedly queries the user for what to do next. Menu options must include:

    - adding **Money** of **Currency** type **USD** to the **Bank**,
    - removing **Money** of **Currency** type **USD** from the **Bank**,
    - displaying how much total **Money** in **USD** is in the **Bank**,
    - adding a new **Patron** to the **Bank**,
    - checking whether someone is already a **Patron**, and displaying their information,
    - displaying a list of information about all **Patron**'s,
    - making a deposit in **USD** by a **Patron**,
    - making a withdrawal in **USD** by a **Patron**,
    - displaying a list of information about all **Patron**'s that are overdrawn
    - displaying a list of **Transaction**'s done by all **Patron**'s

- quitting the program

Now derive a new type of **Bank** class (from your original **Bank** class) that extends its capabilities to support an International Bank that accepts the following **Currency** types, with these exchange rates from **USD** (as of 07/29/2016):

- `United States Dollars: "USD", exchange-rate   1.00/USD`
- `Great Britain Pounds:  "GBP", exchange-rate   0.76/USD`
- `European E. C. Euros:  "EUR", exchange-rate   0.89/USD`
- `Japanese Yens:         "JPY", exchange-rate 102.09/USD`
- `Russian Rubles:        "RUB", exchange-rate  65.97/USD`

NOTE!!! The new **Bank** base class should be modified to allow "*the default **Currency** type*" (i.e. the **Currency** type used to display all **Bank** and **Patron** holdings, which up to this point has always been **USD**) to be any of the five **Currency** types. The **Money** each **Patron** has in their "bank account balance" is now always shown in "*the default **Currency** type*", however the total collection of **Money**'s that the **Bank** has will be in the five different **Currency** types, so be careful in that while each **Patron**'s "bank account balance" is in "*the default **Currency** type*" that does NOT guarantee that the **Bank** has that much total **Money** of "*the default **Currency** type*" (since the **Bank**'s total holdings now would be the sum of "*the default **Currency** type*" equivalence of each of the five **Currency**'s held).

Your new **BankIntlMain** main program **Menu** items that deal with **Money** (i.e. adding/removing/displaying **Bank**'s **Money**, depositing/withdrawing **Patron Money**) must now present the user with the option to specify the **Currency** type for the **Menu** operation, and in each case when removing/withdrawing **Money** make show the **Bank** has a sufficient amount of that **Currency** type to complete the operation (if not then the operation must be declined).

To successfully complete **Phase-1** you must turn-in a main program (and all supporting files) that at start-up asks which country, United States (**USD**), Great Britain (**GBP**), European Economic Community (**EUR**), Japan (**JPY**) or Russia (**RUB**) the bank is located in, and then constructs the **Bank** making that country's **Currency** type as their **Bank**'s "*default **Currency** type*". All other operations (including initializing from and saving to a file) for the International Bank should work identically as they did for the original US Bank. Your **Phase-1** program will be tested to ensure it can correctly be any of the five types of **Bank**. All appropriate error conditions will also be tested for correct handling.

2. **Phase-2 (Final), Banking Network:** Starting from the Phase-2 code fully replace the original console Menu interface with a Graphics Window GUI interface. Widgets of type Button, In_Box, Out_Box, Menu, etc should be used to completely replace the original Menu operations implemented in **Bank** member/helper functions. The one allowable exception to this is that the initial "load from file" and the final "save to file" interface can continue to be thru Unix window (on build.tamu.edu) the program was run from.

   Now write a new main() program that constructs five **Bank**'s, one in the United States, one in Great Britain, one in the European Economic Community, one in Japan and one in Russia. Each of these Bank's must still support all five **Currency** types (using the above exchange rates from 07/29/2016), however each of the five **Bank**'s upon construction must specify their own country's **Currency** type as their **Bank**'s "*default **Currency** type*". When the new main() program starts up each of the five **Bank**'s (as they are constructed) will do their own "initialize from file" query (therefore each **Bank**'s state should be initialized-from/saved-to a different file), and then pop-up their Graphics/GUI Interface Window (so five windows will be opened, one per **Bank**). Each **Bank** window can be independently used/closed (no need to close all five at the same time), and upon closing each **Bank** will do their own "save to file" query.

An additional operation should be added to the **Bank** GUI, which is the ability for a **Patron** to transfer **Money** from their account in one **Bank** to their account in a different **Bank**. The **Patron** must have accounts in both **Bank**'s and sufficient funds in the "from" **Bank** to cover the transfer (i.e. the transfer should be rejected if it causes the **Patron** to be overdrawn in the "from" **Bank**). The transfer must always be initiated in the GUI window of the "from" **Bank**, and the transfer is always done in the "*default **Currency** type*" of the "from" **Bank** (therefore the "from" **Bank** must reject the transfer if it does not have a sufficient amount of that **Currency** to cover the transfer). The result of the transfer will not be visible in the window of the "to" **Bank** until some operation is done in the "to" **Bank** window which causes it to be redrawn.

Your **Phase-2 Final** program will be tested to ensure that all five **Bank**'s work independently and correctly (via their individual GUI Windows), and that transfers can be successfully made between the **Bank**'s (again via their GUI Windows). All appropriate error conditions will also be tested for correct handling.

Phase-1 of this Project is worth **70%** of the full Project grade and must be turned-in to eCampus by **11:59pm on Friday Nov 18, 2016**. **Phase-2** of this Project (i.e. the **Final** version of the Project) is worth **30%** of the full Project grade and must be turned-in to eCampus by **11:59pm on Friday Dec 2, 2016**. The Project grading rubric (not the Homework Program grading rubic) will be used to Grade each Phase of the Project.

Since the expectation is that Project Teams will spend part of each Lab working together on the Project, Lab attendance will be **MANDATORY** from 10/31/16 thru 12/2/16. Each unexcused Lab absence will cost the Student 1% of the total points available for the Project (note that this deduction is just on the individual Student, not on the other members of their Project Team). The TA's will track attendance at each Lab and it will be their decision as to whether or not to excuse an absence.