Programming Assignment #6

DEADLINE: **Friday, 4/19/2019, 11:59PM.**
Late-submission DEADLINE: Sunday, 4/21/**2019, 11:59PM (20% deduction).**

ABET/SMALC Assessment

This assignment is designated as one of the course assignments being used to assess basic programming skills for ABET/SMALC requirements (see the syllabus for more details). In addition to the normal grading scales, each student's submission will be judged on several aspects on a scale of "Highly Effective", "Effective", or "Ineffective", as specified by ABET/SMALC outcome assessment procedures.
A student's submission that earns at least 85% of the available points will count as "**Highly Effective**", earning between 70-85% of the available points will count as "**Effective**", and getting less than 70% of the points (i.e., a score under 28 points for this assignment) will count as "**Ineffective**".

In order to pass the class you must obtain at least 70% of the points ("Effective") on ONE of the two ABET/SMALC assignments, with this assignment being one of them. For this assignment, that means getting at least 28 points out of the total 40 points.

Objective: To gain experience with base and derived classes, virtual functions, and using applications of **polymorphism**. Also, to gain further practice with file I/O, as well as dynamic allocation.

Task: You will design a set of classes for storing information regarding bank accounts, along with a class that will manage a list of accounts. Data must be imported from files for storage in the list, and summary reports with computed balances must be saved to output files.

Note: You should have already seen basic C++ file I/O techniques and I/O formatting in your pre-requisite course. If you need a refresher, see the following note sets from Bob Myers' COP3014:
Basics of File I/O and Stream Objects: http://www.cs.fsu.edu/~myers/c++/notes/fileio.html
Output Stream Formatting: http://www.cs.fsu.edu/~myers/c++/notes/formatting.html

Classes Details

1. Design a set of classes to store bank account information. There should be one base class called **Account** to store common data about bank accounts, and three derived classes that divide the set of accounts into three categories: **Savings**, **Checking**, and **Investment** accounts. All data stored in these classes should be `private` or `protected`. Any access to class data from outside (besides accessing base class data from derived classes) should be done through public member functions. **Make use of virtual and abstract functions wherever appropriate**.

2. The base class **Account** should allocate storage for the following data (and only this data):
    - Account holder's **first name** (you may assume it is up to 20 characters long)
    - Account holder's **last name** (you may assume it is up to 20 characters long)
    - The **type** of account (**Savings**, **Checking**, or **Investment**)

3. Each class (base class and derived classes) should have a function that will compute and return the account's projected balance based on the stored information. All balances are in dollars. Here is the information that needs to be stored for each account, along with the breakdown for computing each projected balance:

    Savings Account
    Stores: -- Current Balance
    -- Interest rate

$$\boldsymbol{ProjectedBalance = CurrentBalance * (1 + interest\ rate)}$$

    Checking Account
    Stores: -- Current Balance

$$\boldsymbol{ProjectedBalance = CurrentBalance + 0.1}$$

    Investment Account
    -- An investment account is made up of five Exchange-Traded Funds (ETFs). The investment class should store the information for the five ETs. Each ETF has the following four pieces of information: amount invested (A), initial value per share (IVS), current value per share (CVS), and interest rate (I).
    The projected balance of the investment account is given by the current value (CV) of the five ETFs + their dividends.
    The current value of an ETF is computed as follows

$$\boldsymbol{CV = \left(\frac{A}{IVS}\right) * CVS}:$$

    The dividends yielded by each ETF is computed as follows:

$$\boldsymbol{DIV = I * A}$$

    The projected balance of an investment account is computed as follows:

$$\boldsymbol{ProjectedBalance = \sum_{ETFs} CV + DIV}$$

4. Write a class called **Portfolio**, which will be used to store the list of various accounts, using **a single array of flexible size**. Note that this array will act as a heterogeneous list, and it will need to be dynamically managed. Each item in the array should be an Account pointer (so you will have a dynamically allocated array of Account pointers, i.e., *Account*** *alist*), which should point to the appropriate type of object. **Your list should only be big enough to store the accounts currently in it.**

Your Portfolio class needs to have at least the following public functions available:

- ***Portfolio()*** -- default constructor. Sets up an empty list of accounts.

- ***~Portfolio()*** -- destructor. Needs to clean up all dynamically allocated data being managed inside a *Portfolio* object, before it is deallocated.

- ***bool importFile(const char* filename)***
  This function should add all data from the file given as parameter into the internally managed list of accounts. The file format is specified below in this writeup. Note that if there is already data in the list from a previous import, this call should add MORE data to the list. Records should be added at the end of the given list in the same order they appear in the input file.
  If the file does not exist or cannot be opened, return false to indicate failure of the import. Otherwise, after importing the file, return true for success.

- ***bool createReportFile(const char* filename)***
  This function should create a banking report and write it to the given output file (filename given in the parameter). The format of the banking report file is described below in this writeup. If the output file cannot be opened, return false for failure. Otherwise, after writing the report file, return true for success.

- ***void showAccounts() const***
  This function should print to the screen the current list of accounts, one account per line. The only information needed in this printout is last name, first name, account type, and current balance (in the case of investment accounts, instead of current balance you will print **the total amount invested**, i.e., the sum of the invested amounts (A) in the five ETFs). Format this output so that it lines up in columns.

Note that while this class will need to do dynamic memory management, you do not need to create a copy constructor or assignment operator overload (for deep copy) for this class. (Ideally, we would, but in this case, we will not be using a *Portfolio* object in a context where copies of such a list will be made)

5. Write a main menu program in a separate file called **main.cpp** that creates a single *Portfolio* object and then implements a menu interface to allow interaction with the object. Your main program should implement the following menu loop (any single letter options should work on both lower- and upper-case inputs):

***** Portfolio Management menu *****

I Import accounts from a file
S Show accounts (brief)
E Export a banking report (to file)
M Show this menu
Q Quit program

The import and export options should start by asking for user input of a filename (you may assume it will be up to 30 characters long, no spaces), then call the appropriate class function for importing accounts from a file or printing the banking report to a file, respectively. The program must print a message indicating the task was unsuccessful if the import/export fails.

The "Show accounts (brief)" option should call the class function that prints the brief account info to screen (names, account type, and current balance/invested amount only).

The "Show this menu" option should re-display the menu.

"Quit" should print "Goodbye" and exit program. (Until this option is selected, keep looping back for menu selections).

6. File formats

The following examples of input files are provided: **test1.txt**, **test2.txt**.
An example run is provided in file: **samplerun.txt**
The output file of the provided run is provided in file: s**ummary.txt**

Input File -- The first line of the input file to import accounts from will contain **the number of accounts** listed in the file. This will tell you how many accounts are being imported from this file. After the first line, every set of two lines constitutes a single account entry. The first line of an entry is the account holder's full name, in the format **lastName, firstName**. Note that a name could include spaces -- the comma will delimit last name from first name.

The second line will contain the type of account ("Savings", "Checking", or "Investment"), followed by a list of account information, all separated by spaces. There will be no extra spaces at the end of the line in the file. The order of the account information for each type is as follows:

- for Savings accounts: current Balance, then Interest rate (mind that the values on a line will be separated by space)
- for Checking accounts: current Balance
- for Investment accounts: amount invested for ETF1, initial value for ETF1, current value for ETF1, interest rate for ETF1, amount invested for ETF2, initial value for ETF2, current value for ETF2, interest rate for ETF2, amount invested for ETF3, initial value for ETF3, current value for ETF3, interest rate for ETF3, amount invested for ETF4, initial value for ETF4, current value for ETF4, interest rate for ETF4, amount invested for ETF5, initial value for ETF5, current value for ETF5 , interest rate for ETF5 (mind that the values on a line will be separated by space)

Output File -- The output file that you print should list each account holder's name (firstName lastName - no extra punctuation between), Initial Balance, and projected balance. All the balances should be printed to two decimal places. An example output file is provided in file: **summary.txt**

The Output should be separated by subject, with an appropriate heading for each section, and each account's information listed on a separate line, in an organized fashion. (See example file). The order of the accounts within any given category should be the same as they appear in the portfolio. Data must line up appropriately in columns when multiple lines are printed in the file. At the bottom of the output file, print the account distribution of ALL accounts in the portfolio (i.e., how many Saving accounts, Checking accounts, and Investment accounts there are) and **the average projected value per type of account**.

7. Extra Credit

Extra Credit #1 - Sorting

Write a function called **sort()** and add in a menu option (using the letter 'O') for sorting the accounts list.

The letter 'O' option should cause the accounts list to be sorted alphabetically by the holder's last name, and if needed by first name when the last names are the same. Do not change any of the stored data in the account records (i.e. don't change the stored case -- upper/lower case -- of the names in the objects). Just re-order the array in the Portfolio object so that it is now alphabetized by name.

Extra credit #2 - Aggregation

Implement the Exchange-Traded Funds (ETFs) as a class. The class stores the following info: amount invested (A), initial value per share (IVS), current value per share (CVS), and interest rate (I).

Implement the Investment account in such a way that it has/stores an array of ETF objects (i.e., it has an array of ETF objects as member data).

General Requirements

1. All member data of the classes must be private or protected.
2. Use the const qualifier on member functions wherever it is appropriate.
3. Adhere to the good programming practices discussed in class (e.g., no global variables, other than constants or enumerations, **DO NOT** #include .cpp files, properly document your code).
4. Do not use language or third-party library features in this assignment. You may use the following libraries: *iostream, iomanip, fstream, cmath, cstring, cctype.* You may also use the *string* library. You may NOT use any of the other STL (Standard Template Libraries) except string.
5. Your ***.h*** files should contain the class declaration only, whereas the ***.cpp*** files should contain the member function definitions.

Deliverables

Submit all source code files in one package, **proj6.tar.gz**, on Canvas. This should include *a set of account classes (base and derived)*, the *Portfolio* class, and the menu program described

above. As usual, the makefile is also required (the final executable is named **proj6**, and the file containing main() function is named **main.cpp**) and an optional readme file.

Do **NOT** send program submissions through e-mail. E-mail attachments will **NOT** be accepted as valid submissions.