EECS402 Fall 2024 "Mini-Project" 1

This mini programming assignment will ensure you are able to write a program, compile it, run it, test it, and submit it. Since I'm trying to keep this at a minimal level, it will be quite short – in no way does this project represent the complexity of future projects, which will vary significantly. Despite this, especially if you don't have much or any programming experience, you should start it right away in case you run into problems with compiling, etc…

As described in lecture, your final grade depends on more than just correct behavior, so be sure to follow the specs exactly, and also write easy to read and easy to understand code, use the best types of loops, avoid magic numbers, etc., as appropriate. In addition to correctness, your grade will also consider all other aspects of your implementation, including style, organization, readability, etc.

Submission and Due Date

You will submit your program using an email-based submission system by attaching *one C++ source file* only (named exactly "**project1.cpp**" – be sure to use that exact filename so that you don't get a deduction!). Do not attach any other files with your submission – specifically, do *not* include your compiled executable, etc. The due date is **Thursday, September 19, 2024 at 4:30pm**.

No submissions will be accepted after the submission deadline. As discussed in lecture, please double check that you have submitted the correct file (the correct version of your *source code* file named exactly as specified above). Submission of the "wrong file" will not be grounds for an "extension" or late submission of the correct file, and will result in a score of 0.

High-Level Description

For this project, you will implement functionality to accrue interest for an investment account. For a real bank, there are many aspects to this calculation, but for the purposes of this "mini-project", we will keep things short and simple. Following are some important aspects of how the bank we are designing this project for handles things:

- Our bank will assume interest is accrued monthly, not daily, and the length of the month will not affect the amount of interest - that is: every month will accrue "one month's worth of interest", whether the month has 28 days or 31 days, etc.
- For the investment account we're considering, additional deposits or withdrawals are not allowed, so the balance cannot change in any way, except for accrual of interest - that is: our accrual approach can use the account balance for the month without concern for the balance changing mid-month due to other withdrawals or deposits.
- Our bank has three different interest rates, depending on the balance of the customer's account. Higher balances will get a higher rate in order to persuade customers to have a larger balance. The interest rates work as follows:
  - Customer balances under $1000.00 (exclusive) will be given a monthly interest rate of 0.15%, which our bank calls the "minimum interest rate".

- Customer balances over $15000.00 (inclusive) will be given a monthly interest rate of 0.4%, which our bank calls the “maximum interest rate”.
- Other balances (i.e. between $1000.00 (inclusive) and $15000.00 (exclusive)) will be given a monthly interest rate of 0.225%, which our bank calls the “standard interest rate” since the majority of our customer’s balances are within this range.
- Interest is accrued once per month and is calculated simply by multiplying the month’s balance by the appropriate monthly interest rate and adding the result back to the balance. Only one interest rate is used during a month - that is, if a balance is increased to the next-level interest rate during a month, the new rate is not utilized until the next month. This is to keep things very simple - don’t make it more complicated than described.

Requirements

You will implement the following functions, whose *exact specifications* are provided below. You must implement exactly these functions, as described, with the same name, parameter types and names, return types, etc. Do not implement any additional or fewer functions, or change the given specifications in ***any*** way.

```
bool accrueInterest(
     double &balanceAmt,
     const int numMonths,
     const bool doPrintEachMonth
     )
```

This function accrues interest, as per the way our "bank" allows, over the specified number of months. The parameter balanceAmt is used as both input (the initial balance of the account prior to interest accrual) and output (the final balance after the specified number of months' worth of interest have been accrued). If doPrintEachMonth is false, the function performs silently, but if true, the amount of interest added each month, along with the balance after that month, will be printed. Returns true if the function was able to perform as expected, or false otherwise. Failure will be indicated if the input parameter(s) are illogical in some way - specifically if the initial balance is negative, or if the number of months specified is not positive.

```
bool accrueOneMonthsInterest(
     double &balanceAmt,
     const bool doPrintInfo
     )
```

This function accrues interest, as per the way our "bank" allows, for a single month (the period in which our "bank" accrues interest). The parameter balanceAmt is used as both input (the initial balance of the account prior to interest accrual) and output (the final balance after the months' worth of interest has been accrued). If doPrintInfo is false, the function performs silently, but if true, the amount of interest added for the month, along with the balance after interest is accrued, will be printed. Returns true if the function was able to perform as expected, or false otherwise. Failure will be indicated if the input parameter is illogical in some way - specifically if the initial balance is negative.

int main()

This function will be what is executed when your program starts, and for the purpose of this "mini-project" will serve primarily to test the other functionality you've implemented as described above. Therefore, we'll keep the main very minimal - in a "real program", we might choose to implement a menu-driven system that allows the user more flexibility. For this mini-project, though, your main function must ask the user for an initial bank account balance, then prompt the user for the number of months of accrual they are interested in seeing, then prompt the user for whether they'd like to see the results on a month-by-month basis, or just the final total at the end of the months requested. See the example output provided for specific details as to what prompts to use, etc. For this project, you can safely assume that the user (grader) will provide input using the appropriate data types. Specifically, when executing your program the user (grader) will always enter a floating point value (the initial balance) followed by an integer (the number of months), followed by a character (whether to show month-by-month results).

Output Detail

As humans, we are used to seeing money values printed to two digits after the decimal place. For this project, though, do not try to alter the way the money values will be printed. Simply use "cout" directly, and let the values print as cout prints them. Do not use iomanip, setprecision, setw, etc., in this project.

Special Testing Requirement:

In order to allow me to easily test your program using my own main() function in place of yours, some special preprocessor directives will be required. Immediately before the definition of the main() function, (but after ***all*** other prior source code), include the following lines EXACTLY:

```
#ifdef ANDREW_TEST
#include "andrewTest.h"
#else
```

and immediately following the main() function, include the following line EXACTLY:

```
#endif
```

Therefore, your source code should look as follows:

```
//library includes go here
//program header goes here
//global constant declarations and initializations here
//global function prototypes and documentation here
#ifdef ANDREW_TEST
#include "andrewTest.h"
#else
int main()
{
  //implementation of main function goes here
}
#endif
//global function definitions go here
```

Lines above in red are to be used *exactly* as shown. Other lines simply represent the location of those items within the source code file.

Additional Information

- When implementing your solution, use only topics that we've discussed in lecture by the date that the project was posted, or that course staff has explicitly allowed in writing. Experienced programmers may identify ways to solve problems in the project using more advanced topics that had not been discussed by the date the project was posted, but you may not utilize those techniques!
- Remember, "magic numbers" are bad and need to be avoided
- Remember, "duplicated code" is bad and needs to be avoided
- Remember, identifier naming is important. Name your variables, constants, etc., using a descriptive name using the style described in lecture
- To compile and execute on the Linux command line, refer to the "gPlusPlus" lecture posted on the Canvas site, which walks you through it step-by-step
- As mentioned above, for this project, you are required to implement the project *exactly* as described here using the *exact* output strings provided in the sample outputs. Make sure you implement and utilize the exact functions specified and do not add any additional functions, parameters, etc. You will have more "freedom" in your design and implementation as the course progresses.
- You may not use any functionality or data type from any library other than <iostream>. Specifically, be careful not to use anything from the math library or the string library.

"Specific Specifications"

These "specific specifications" are meant to state whether or not something is allowed. A "no" means you definitely may NOT use that item. In general, you can assume that you should not be using anything that has not yet been covered in lecture (as of the first posting of the project).

- Use of Goto: No
- Global Variables / Objects: No
- Global Functions: Yes (required)
- Use of Friend Functions / Classes: No
- Use of Structs: No
- Use of Classes: No
- Public Data In Classes: No
- Use of Inheritance / Polymorphism: No
- Use of Arrays: No
- Use of C++ "string" Type: No
- Use of C-Strings: No
- Use of Pointers: No
- Use of STL Containers: No
- Use of Makefile / User-Defined Header Files / Multiple Source Code Files: No
- Use of exit(): No
- Use of overloaded operators: No
- Use of float type: **No** (That is, all floating point values should be type double, not float)
- Use of any libraries, or functionality from any libraries other than <iostream>: **No**