Homework #1
A Structured Account

Due: Jun. 6, by 11:59:59* **Assigned:** June 4, 2018
*Due date may be subject to change depending on how quickly the handin account is created

We will use `structs` to represent bank accounts and money respectively, and write functions that allow us to interact with our accounts.

Requirements:

- Download the file `program1.cpp` that was provided with the assignment
- Create two structures and name the types `Account` and `Money`
- The `Money` struct contains two integers.
  - One represents how many dollars you have
  - The other represents how many cents you have
- The `Account` struct has three variables
  - A `Money` struct that will represent how much money is in the account
  - One variable to represent the interest rate as a decimal value.
  - The final variable will contain the name of the account. (Checking, Savings, CD, etc.)
- Negative amounts of `Money` are stored by making **both** variables of the `Money` object negative
- Use the following function prototypes
  - `Account createAccount(std::string file)`
    - ⋆ The function shall look for a file matching the name of the parameter that is passed to the function. The file contains the name, interest rate, and starting balance *in that order*
    - ⋆ You may assume that each of the three items are contained on their own line
    - ⋆ You may **NOT** assume that the name of the account contains only a single word
    - ⋆ You may also assume that a positive and valid (whole number or no more than two decimal places) amount of money will be contained in the file
    - ⋆ An Account object with the proper values is then returned
    - ⋆ If the file does not exist or otherwise cannot be opened, a default Account with name "Savings," an interest rate of 1%, and a starting balance of $100.00 is returned
  - `createAccount(std::string name, double rate, Money balance)`
    - ⋆ The function will return an account with the values already set based on the parameters passed
  - `Account deposit(Account account, Money deposit)`
    - ⋆ The function shall not accept negative amounts of money

        · If a negative amount of money is attempted to be deposited, an error message will be displayed, and the original account will be returned
    * A message shall be printed to the screen that takes the form "$X.XX deposited into [NAME]." **only** if a successful deposit is made.
        · The message appears on its own line
        · [NAME] shall be replaced with the name of the account
    * The balance of the account shall be updated accordingly

  - `Money withdraw(Account &account, Money withdraw)`
    * The function shall not accept negative amounts of money
        · If a negative amount of money is attempted to be withdrawn, an error message will be displayed, and a Money object equivalent to $0.00 will be returned
    * You may allow the account to be overdrawn, but by no more than $50.00
    * A message shall be printed to the screen that takes the form "$X.XX withdrawn from [NAME]." **whether or not** a successful withdrawl is made.
        · The message appears on its own line
        · [NAME] shall be replaced with the name of the account
    * The balance of the account shall be updated accordingly

  - `void accrue(Account &account)`
    * A message shall be printed to the screen that takes the form "At X.XX%, your [NAME] account earned $X.XX."
        · The message appears on its own line
        · [NAME] shall be replaced with the name of the account

  - `void print([SINGLE PARAMETER])` **x2**
    * The function shall be overloaded to accept either a `Money` object or an `Account` object
    * The functions shall print **ONLY** the amount of money
        · This means no extra phrases like "You have *blah blah*"
    * The amount of money shall be printed with a '$', a decimal point, and **only** 2 decimal digits
    * There shall be no extra whitespace before or after the amount of money when it is printed
    * Negative amounts of money shall be represented in the following manner: ($X.XX)

- Unless otherwise specified, a successful run of the program will result in multiple tests being run, and test is considered successful if it results in a 1. A test fails if it results in a 0

- A sample run of your program shall look something like this:

```
--- Homework 1 Tests ---
createAccount(filename)..........................1
createAccount(bad filename)......................1
createAccount(parameters)........................1

--- Begin Depost Testing ---
Deposit Message.................................
$10.00 deposited into Checking
```

```
deposit()........................................1

Deposit Message................................
Cannot make negative deposit.

deposit() (negative deposit).....................1
--- End Deposit Testing ---


--- Begin Withdraw Testing---
Withdrawl Message................................
$10.36 withdrawn from Index Fund.
withdraw().......................................1

Withdrawl Message................................
$60.10 withdrawn from Index Fund.

withdraw() (partial overdraft)...................1

Withdrawl Message................................
$100.00 withdrawn from Index Fund.

withdraw() (full overdraft)......................1

Withdrawl Message...............................
Cannot make negative withdrawl.
$0.00 withdrawn from Index Fund.

withdraw() (negative withdraw)...................1
--- End Withdraw Testing---


--- Begin Accrue Test ---
Accrue Message...................................
At 2.00%, your Savings account earned $0.97.

accrue().........................................1
--- End Accrue Test ---

print() (Money) [Expect $567.32].................$567.32
print() (Money) (negative) [Expect ($567.32)].....($567.32)
print() (Account) [Expect $567.32]................$567.32
print() (Account) (negative) [Expect ($567.32)]...($567.32)
```

Hints:

- The functions listed in the *Requirements* are required (shocker!), but you may find it useful to write other "helper" functions
- Converting a `double` to a `Money` object can cause rounding errors
  - You may want to look up the `round()` function

- Converting an amount of money to an equivalent amount of pennies makes a lot of logical work go away
- Take note of what statements are required to be printed from within functions, the rest are printed in the `main()` function

Reminders:

- Be sure to include a comment block at the top of the file with the required information
  - Refer to the General Homework Requirements handout on Blackboard
- Provide meaningful comments
  - If you think a comment is redundant, it probably is
  - If you think a comment is helpful, it probably is
  - Remember that you are writing comments for other programmers, not people who know nothing (obligatory Jon Snow) about coding
- There will be no extensions

Preparing & Submitting

- Your code must be able to compile and run on the EECS Linux Lab Servers
  - You are responsible for testing your code
  - "But it runs fine on my machine!" will **not** earn you any points after the fact
- Submit **ONLY** source code files
- **These instructions are subject to change**
  Homework submissions will be handled exclusively through the `handin` tool in the Linux Lab. You may submit your homework using the following command:
  `~cs311/bin/handin 1 program1.cpp`