Homework #4
Hotdog Empire Simulator

Due: Feb. 23, by 11:59:59 **Assigned:** Feb 9, 2018

We will simulate a basic hotdog stand empire.

Requirements:
***General** Requirements*

- You will be submitting eight (8) files for this assignment
  - Be sure to split your code among the files appropriately
- The name of your executable shall be `hw04`
- Create two classes and name their types `HotdogStand` and `Money`
- All relevant classes, functions, and data shall be placed in a namespace called `MyAwesomeBusiness`
- Use initialization sections where possible for all constructors
- With the sole exceptions being the << operator, **no other functions should be printing to the screen unless an error message must be displayed**

***Money** Class Requirements*

- Negative amounts of `Money` shall be stored by making **both** the dollars and cents negative
- The `Money` class shall have four (4) constructors
  - A default constructor that initializes your `Money` object to $0.00
  - A constructor that takes two integers, the first for the dollars and the second for the cents
  - A constructor that takes one integer, for an clean dollar amount
  - A constructor that takes one double, and can assign dollars and cents accordingly
- `int getPennies() const`
  - Returns the equivalent amount of money as a number of pennies
- `bool isNegative() const`
  - Returns true if your amount of money is negative
- The following functions will be **removed**:
  - `void add(const Money &m)`
  - `void subtract(const Money &m)`
  - `bool isEqual(const Money &m) const`
  - `void print() const`

- The following operators shall be overloaded as **member** operators
  - Unary minus (negative sign)
  - Prefix **&** Postfix increment
  - Prefix **&** Postfix decrement
- The following operators shall be overloaded as **friend** operators
  - Addition
  - Multiplication of Money and int
  - Multiplication of int and Money
  - Multiplication of Money and double
  - Multiplication of double and Money
  - Less than
  - Less than or equal to
  - Greater than
  - Greater than or equal to
  - Equality
  - Inequality
  - Insertion
  - Extraction
- The following operator shall be overloaded as a **non-member, non-friend** operator
  - Subtraction
- The `Money` class shall have two private data members
  - An integer that represents how many dollars you have.
  - An integer that represents how many cents you have.

***HotdogStand** Class Requirements*

- The `HotdogStand` class shall have three (3) constructors
  - A default constructor with the following values
    - Price of a hotdog = $3.50
    - Daily dogs sold = 0
    - Total stand dogs sold = 0
    - Hourly wage of $7.25
    - Hotdog supply = 60
    - Max hotdogs = 60
    - Wholesale hotdog cost = $0.15
    - Your starting cash will be a negative value representing the cost of buying your maximum amount of hotdogs

  - A constructor that takes a double that represents the price of a hotdog. Initialize the remaining values as follows:
    * Daily dogs sold = 0
    * Total stand dogs sold = 0
    * Hourly wage of $8.00 **IF** the price of a hotdog is greater than $3.50, or $7.25 otherwise
    * Hotdog supply = 30 **IF** the price of a hotdog is greater than $3.50, or 60 otherwise
    * Max hotdogs = see hotdog supply
    * Wholesale hotdog cost = $0.50 **IF** the price of a hotdog is greater than $3.50, or $0.15 otherwise
    * Your starting cash will be a negative value representing the cost of buying your maximum amount of hotdogs
  - A constructor that takes a Money object that represents the price of a hotdog. Initialize the remaining values according to the second constructor

- `const Money getCash() const`
  - This returns the total cash the `HotdogStand` has
- `const Money getPrice() const`
- `int getDailyDogsSold() const`
  - *ATTN*: This function is changed from the previous assignment
  - This returns the number of hotdogs sold by the stand in one day
- `void replenishSupplies()`
  - You must use your cash to buy hotdogs so that you have your maximum supply
- `void payWorker()`
  - The wages of your HotdogStand employee are set aside from your cash daily
- `void dailyReset()`
  - Resets the appropriate private data members in order to correctly simulate another day
- `const Money getStandRevenue() const`
  - This calculates the total money made by selling hotdogs
- `void setPrice(double price)`
- `void sellHotdog()`
  - Increments all appropriate data members accordingly
- `static int getNumStands()`
- `static int getTotalHotdogsSold()`
- `static const Money getTotalRevenue()`
- The `HotdogStand` class shall have twelve (12) private data members

- A `Money` object that will represent how much money the stand has made
- A `Money` object that will represent the price of a hotdog
- An integer that will describe how many hotdogs a stand has sold in a single day
- An integer that will describe how many cumulative hotdogs a stand has sold
- A `Money` object representing the hourly wage of your stand employee
- A constant that represents how many hours your employee works in a day, initialized to 8
- An integer that represents your hotdog supply
- An integer that represents the maximum amount of hotdogs you are allowed to have
- A `Money` object that represents the wholesale cost of a hotdog
- A `static` integer that represents the total number of `HotdogStand` objects
- A `static` integer that represents the total number of hotdogs sold
- A `static Money` object that represents total revenue for all `HotdogStand` objects

***Non-Member** Requirements*

- Define the following non-member functions in the `MyAwesomeBusiness` namespace:
  - `void runSimulation(std::vector<HotdogStand> &franchises, int days)`
    - This function runs the simulation for an end-user specified number of days
  - `void printRundown(const vector<HotdogStand> &franchises)`
    - *ATTN*: This function is changed slightly from the previous assignment
    - This function will print the summary that is shown in the sample run below

***main()** Requirements*

- Declare a vector of `HotdogStand` objects in your main function
- The end-user shall be prompted for how many hotdog stands they own
- The end-user will then be prompted for how many of those hotdog stands sell fancy hotdogs
- If the above input is greater than zero (0), the end-user shall be prompted for the price of the fancy hotdogs
  - The end-user may type the price with or without the $
- The end-user will be prompted to specify for how many days the simulation will run
- The simulation is then run and the output is printed to the screen

***Simulation** Requirements*

- The last n stands are always fancy, where n is the number of stands designated as fancy by the end-user
- Regular hotdog stands can sell a random number in the range 20 - 60 (inclusive) hotdogs a day
- Fancy hotdog stands can sell a random number in the range 1 - 30 (inclusive) hotdogs a day
- At the end of a day, a hotdog stand must purchase as many hotdogs as they sold to reach their maximum supply

- At the end of a day, cash will be deducted to pay the employee for that day's work
- Totals for cumulative hotdogs sold, cumulative revenue, and total cash is only displayed at the of the simulation
- All money amounts in the daily table **must** be right-aligned

A sample run of your program shall look like this (Your numbers will **not** match):
NOTE: There are some subtle changes to the output that are required

```
$ ./hw04
Welcome!
How many hotdog stands do you own? 3
How many of these sell classy hotdogs? 1
How much does a classy hotdog cost? $8.99
How many days will you simulate? 3

Stand  Sales    Price    Revenue  Remaining Cash
=====  =====  =======  =========  ==============
    1     52    $3.50    $182.00         $107.20
    2     50    $3.50    $175.00         $100.50
    3      8    $8.99     $71.92         ($1.58)

Stand  Sales    Price    Revenue  Remaining Cash
=====  =====  =======  =========  ==============
    1     24    $3.50     $84.00         $129.60
    2     60    $3.50    $210.00         $243.50
    3     18    $8.99    $161.82          $91.74

Stand  Sales    Price    Revenue  Remaining Cash
=====  =====  =======  =========  ==============
    1     41    $3.50    $143.50         $208.95
    2     48    $3.50    $168.00         $346.30
    3     19    $8.99    $170.81         $193.80

TOTALS   320            $1367.05         $749.05
Stands: 3
$
```

***makefile** Requirements*

- Your makefile must still include the requisite comment block
- Your makefile must compile each class separately
- Your makefile must include a clean rule
- Your makefile will use variables for the compiler name, -Wall flag, and the complete list of objects
- Your makefile will include the special built-in target `.PHONY`

Hints:

- The functions `getCash()` and `getStandRevenue()` should **NOT** return the same amount of money in **this** assignment.
- It is possible to "typedef" a namespace. It looks something like this: `namespace` *`alias`* `=` *`namespace`*`;`
- To ensure a proper count of Hotdog Stands, you may want to look up the vector function `emplace_back()`
- While not enforced in this assignment, this is a great time to practice not using `using` directives, or at least placing them more smartly
- The C++11 library <random> is much more robust than the older C library
- With the change of a variable name in the `HotdogStand` class, and the new operator overloads, you will need to re-examine many of your other functions that you hoped wouldn't need to be touched again
- When first testing that your calculations work as intended, hard-code the sales numbers to check your math against the sample run
- While required, the right alignment of the Money values is cosmetic. Save it for near the end
- When your first attempts at right-aligning ultimately fail, consider how `setw()` works, what your overloaded insertion operator is doing, and what change would be required to have it work as expected
- The hints from the first homework also apply here and are repeated
  - 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
  - You can create a file that holds user inputs and use it to streamline your testing. http://linuxcommand.org/lc3_lts0070.php

Reminders:

- Include a makefile!
- 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
- Homework submission will be handled exclusively through the `handin` tool in the Linux Lab. You may submit your homework using the following command:
  For the 11:00 AM lecture section: ~cs311a/bin/handin 4 [ALL OF YOUR FILES]

  For the 4:05 PM lecture section: ~cs311b/bin/handin 4 [ALL OF YOUR FILES]