CPSC 1160 SECTION 001 INSTRUCTOR: GLADYS MONAGAN

ASSIGNMENT #6: ABSTRACT DATA TYPE AS A C++ CLASS

DUE DATE WITH BRIGHTSPACE: OCTOBER 26 AT 11:50 PM

THE LEARNING OUTCOMES

- to design and implement an Abstract Data Type, ADT, for Money
- to implement a complete proper C++ class that overloads operators
  (the big 3 are still not implemented, more later in the course)
- to compile and link code from source and application files using a Makefile

READINGS IN THE TEXTBOOK

- chapter 9
- chapter 10
  - §10.8 is not covered in this course
  - §10.9 will be covered later
- chapter 14
  - §14.5, §14.7 are optional for now (see bonus part)
  - §14.13 will be covered later

TO IMPLEMENT

Define a class named **Money** to store an amount. Here is the functionality that your class should provide. Your class should

- create an object from two integers, the dollars and the cents
- create an object from a single integer: this integer corresponds to just the dollars
- create an object with a no-arg, default constructor that sets the Money amount to $0.00
- get the dollars (the value returned should be an int): this is an accessor function
- get the cents (the value returned should be an int): this is an accessor function
- get the Money amount as a double: this is an accessor function
- set the dollars and cents with two integers in a single function: this is a mutator function
- set the dollars and cents with an amount of type double: this is a mutator function
- overload the operator << to output in the format $ddd.cc where
  $ is the dollar sign
  d is one more digits corresponding to the dollars' digits (possibly 0)
  a fixed decimal point
  2 digits for the cents (possibly 00): the 2 is called Money::DIGITS_OF_CENTS in Money.h
  There could be a negative sign before the dollar sign as in -$7.31
- overload the operator >> for input
  the first integer read is the dollars, the second integer is the cents.
  Do not do any output in the input function operator >>

How should you handle negative numbers when creating an object or when setting the 'dollars and cents'? I would say intuitively. A negative number corresponds to an amount owed.
So, if the constructor's first argument corresponds to the dollars and the second argument to the cents:

- `Money m0(0, -32)` creates an object corresponding to the Money amount `-0.32`
- `Money m1(-10, -21)` creates an object corresponding to the Money amount `-10.21`
- `Money m2(-10, 25)` creates an object corresponding to the Money amount `-9.75`

Overload the following operators so that they apply to the data type `Money`:

== binary operator returns true if two `Money` objects are equal, false otherwise

< binary operator returns true if the first `Money` object is less than the second `Money` object, false otherwise

> binary operator returns true if the first `Money` object is greater than the second `Money` object, false otherwise

+ binary operator for adding two `Money` objects: the result is a `Money` object

− binary operator for subtracting two `Money` objects: the result is a `Money` object
interpret the subtraction as the first `Money` object minus the second `Money` object

* binary operator for multiplication: the result is a `Money` object
the first operand is a `Money` object the second operand is an integer `n`
this operation can be interpreted as multiply the Money amount `n` times

* binary operator for multiplication: the result is a `Money` object
the first operand is an integer `n` and the second operand is a `Money` object
this operation can be interpreted as multiply the Money amount `n` times

Include (in the interface, header file) the following named constants of type `Money`.
These `Money` named constants are not part of the class declaration.

- LOONIE corresponds to one dollar and no cents
- TOONIE corresponds to two dollars and no cents
- QUARTER corresponds to no dollars and 25 cents
- DIME corresponds no dollars and 10 cents
- NICKEL corresponds to no dollars and 5 cents
- (no PENNY!)

How should you store (implement) the amount in the Money amount? It's up to you.
Put in the README.txt the reason(s) for your choice. Possibilities are

- using two instance integer variables, one to store the dollars' part of the Money amount and another to store the cents' part of the Money amount.
- storing the Money amount in cents (an integer)
- storing the Money amount as a double

You may assume (a precondition) that the absolute value of the integer passed as cents

- to a constructor
- when setting the dollars and cents in a function
- being read from an input stream

is less than 100. No need for error checking.
Note, though, that -99 is valid. We are saying that the absolute value is less than 100.

TO SUBMIT WITH BRIGHTSPACE AS A SINGLE ZIP FILE:

1. Code that is properly documented. In the lab we will give you the files with a Makefile.
   Fill in the code as needed
   a. a header file Money.h
   b. an implementation file Money.cpp
   c. a unit test file Money_unittest.cpp that uses doctest
   d. a test file testMoney that has a main function and no doctest
   e. a Makefile : leave as is and make sure that your code works with the Makefile
   f. doctest.h
2. Write more unit tests for the missing operators and functions.
   Place these at the bottom of the file Money_unittest.cpp
   For instance, the accessor functions are currently tested in the Money_unittest and you are free to choose the names of these accessor functions.
   Similarly for the functions that set the dollars and cents.
3. A README.txt
   a. explain why you chose to represent the Money amount the way you did:
      why did you choose 2 integers? Or why did you use a double? Or an integer?
      what were the advantages vs disadvantages of your choice
   b. include in the README.txt that you did the BONUS with the test cases
      If the BONUS is not mentioned in the README.txt, you will not get the bonus marks.

BONUS AT MOST 40% EXTRA

I encourage you to start the bonus part **after** you have completed the rest of the program.

1) `-` unary operator minus for a negative Money amount

2) `/` binary operator for division where
the first operand is a `Money` object the second operand is an integer `n`: the result is a `Money` object
this operation can be interpreted as divide the Money amount by `n` people
assume that `n` is NOT zero
explain the division / operator: how does it relate to multiplication?
How did you divide $1 Money by 3 people?
This type of comment needs to be in the interface file.

3) code the function "get dollars" so that it rounds off (rounds up) the half cent i.e. 7.995 will return 8

4) tests for the bonus parts

Bibliography Books by Savitch