Sheharyar Khan
CS-116
Lab 3 - Money

Source Code

moneyclass.hpp

```
#ifndef MONEY_H
#define MONEY_H

#include <iostream>
#include <string>
#include <cmath>
#include <iomanip>

class Money
{
   private:
   //member variables
   double value;
   int dollars;
   int cents;
   //private member functions
   void simplify();
   void distribute();
   void combine();

   public:
   //constructors
   Money(double money);
   Money(int d, int c);
   Money();

   void setValue(int d, int c);//mutator
   double getValue() const;//accessor

   //operators
   double operator+    (Money const &right);
   double operator-    (Money const &right);
   double operator*    (double const &right);
   bool operator==    (Money const &right);
   bool operator>     (Money const &right);
   bool operator<     (Money const &right);
   Money operator= (double const &right);
};

//Definitions
//*constructors
   Money::Money(int d, int c)
   {
       dollars = d;
       cents = c;
       simplify();
   }

   Money::Money(double input)
   {
       value=input;
       distribute();
   }

   Money::Money()
   {
       value = 0;
       distribute();
   }
//*private methods
   void Money::combine()
   {
       value = dollars*100; //$8 becomes 800
```

```
        value+= cents;//adding 35 cents you get 835
        value = value/100;//you get 8.35
    }

    void Money::distribute()
    {
        double temp = value;
        dollars = floor(temp);//assuming value equals 8.35 this will set dollars to 8 as
flooring would ignore everything after the decimal
        cents = (value*100)-(dollars*100);//8.35 becomes 835 and 8 becomes 800. 835-800 = 35
and we set that equal to the cents
        simplify();
    }

    void Money::simplify()
    {
        if(cents<0)
        {
            dollars-=1;//we borrow one dollar like we borrow in basic subraction
            cents = 100 + cents;//adding because cents is already negative and +*- = -
        }
        else if(cents>=100)
        {
            dollars+=1;//carries over 1 to the next digit like in simple addition
            cents-=100;// subracts 100 because we basically carried that over
        }
        combine();

    }
//*public methods
    void Money::setValue(int d, int c)
    {
        dollars = d;
        cents = c;
        simplify();
    }

    double Money::getValue() const
    {
        return value;
    }

//overloaded operators
    double Money::operator+    (Money const &right)
    {
       return value+right.value; //adds value variables
    }

    double Money::operator-    (Money const &right)
    {
        return value-right.value; //calculates the difference between the value variables
    }

    double Money::operator*    (double const &right)
    {
        return value*right; //multiplies the value variable
    }

    bool Money::operator==    (Money const &right)
    {
        if(value==right.value)//compares the value variable
        {
            return true;
        }
        else
        {
```

```
            return false;
        }
    }

    bool Money::operator>     (Money const &right)
    {
        if(value>right.value)//compares the value variable
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    bool Money::operator<     (Money const &right)
    {
        if (value<right.value)//compares the value variable
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    Money Money::operator= (double const &right)
    {
        //sets the value variable then distributes it
        value = right;
        distribute();
    }

//cout function
    std::ostream& operator<< (std::ostream& output, Money const &right)
        {   //             setting the number of decimal places
            output << "$"<<setprecision (2) << fixed<<right.getValue();
            return output;
        }

#endif
```

Lab3.cpp

```
/**
*TODO:To perform calculations with classes and practice overloading operators
@param name - Sheharyar Khan
@param courseInfo - CS-116
@return - none
*/

#include <iostream>
#include <string>
#include <cmath>
#include "printmefirst.hpp"
#include "moneyclass.hpp"

int main()
{
Money m1(8, 75); // set dollars to 8 and cents to 75
Money m2 (5, 80); // set dollars to 5 and cents to 80
Money m3; // initialize dollars to 0 and cents to 0
```

```
printMeFirst("Sheharyar Khan", "CS-116 2018 Fall - Lab Money"); // use your name
//multiply by 2
cout << m1 << " * 2 = " << m1 * 2 << "\n";
//subract
cout << m1 << " - " << m2 << " = " << m1 - m2 << "\n";
//addition
m3 = m1 + m2;
cout << m1 << " + " << m2 << " = " << m3 << "\n";
if (m1 < m2) // check to see if Money object m1 is less than m2 or not
cout << m1 << " < " << m2 << "\n";
else
cout << m1 << " > " << m2 << "\n";
m1.setValue(10,1); // set m1.dollars to 10; m1.cents to 1
m2.setValue(10,1);
//checks if equals
if (m1 == m2)
cout << m1 << " equals to " << m2 << endl;
else
cout << m1 << " NOT equals to " << m2 << endl;
m2.setValue(10,45);
if (m1 == m2) // compare Money object m1 and m2
cout << m1 << " equals to " << m2 << endl;
else
cout << m1 << " NOT equals to " << m2 << endl;
return 0;
}
```

Purpose

The purpose of this lab is to make a class that works with the provided driver function to print out the desired output and to practice overloading operators.

Logic

Member Variables

I'm using three member variables. Two ints and a double. I'm using ints for *dollar* and *cents* variables because they have to be whole numbers. I'm using double for *value* because it needs to have decimal places.

Member Functions

I'm using 3 private member functions because I don't want anyone to access them. These are *simplify*, *distribute*, and *combine*. The rest are public functions constructors, accessors, mutators, and overloaded operators.

The *simplify* function checks if the *cents* are positive or negative. If they're negative then it decrements the *dollar* value by 1 and adds the *cents* value to hundred. Since cents is already negative we end up finding the difference which is what we wanted.

The distribute function distributes the quantity stored in the double variable *value*. It does the by first flooring the *value* and getting the *dollars*. Then it multiplies the *value* by 100 and subtracts the *dollars* it calculated multiplied by 100 to get the *cents*. We're basically moving the decimal place two to the right and then getting rid of the hundreds,thousands, ten thousands, etc.

The combine function is the opposite of the distribute function. It multiplies *dollars* by 100 and sets it equal to the *value* variable, then adds the *cents*, then divides the *value* by 100 to produce a decimal quantity which is how money normally is.

There are 3 constructors.

```
Money(int d, int c)
```

This constructor takes two ints and assigns them to the *dollars* and *cents* variables.

`Money(double input)`

This constructor takes a double and then distributes the *dollars* and *cents* into their variables via the *distribute* function.

`Money()`

This constructor just initializes all the variables to 0.

There are a whole bunch of overloaded operators the addition, subtraction, and multiplication operators are similar, the greater than, less than and equals operators are similar, the output operator is different than the rest.

The addition, subtraction, and multiplication operators use the *value* variable to perform calculations as it is easier that way and they also return a double as that is easier to print and also can be used as input in one of the constructors.

The boolean operators the compare the *value* variables.

The output operator is different because unlike the others it is not defined inside the class function. It should be defined in the ostream class but since cannot edit that class we must use the global scope to define the overloaded operator.

Test Case