

Emphasis on: **Creating and testing a class**

Calculations involving monetary values can be a problem when your language offers only a floating-point type for representing fractional values. Fractional money values represented as type float are stored with extra digits of precision which can introduce errors into monetary calculations. Create a new class to represent positive dollars-and-cents values with operations that will allow accurate calculations using integer arithmetic rather than float. Your new class should define a single money object (so the client can declare as many of them as necessary for the particular client application) represented by private variables of an integer dollars value and an integer cents value. You should provide operations which allow a client to:

- 1) create a money object using default values (0.00)
- 2) create a money object using client-supplied values (parameters: dollars, cents)
- 3) input a value for a money object (allow input in user-friendly form, like 14.95, but input as two integer values rather than as one float value)
- 4) assign client-supplied values to an existing money object (parameters: dollars, cents)
- 5) output a money value (like 14.95)
- 6) return the integer dollar value (like 14)
- 7) return the integer cents value (like 95)
- 8) add two money amounts and return a money sum
- 9) subtract two money amounts and return a money difference
- 10) multiply a money amount by an integer parameter and return a money product
- 11) divide a money amount by an integer parameter and return a money quotient (truncate any fractions of cents)
- 12) compare two money amounts for equal, less than, and greater than, returning a boolean true or false in each case

Requirements

- 1) You must create and use a class which represents **one** money object with a possible range of 0.00 to +1000.00. No negative values are expected to be input or calculated.
- 2) At least one function must be an overloaded operator function.
- 3) Wherever possible, the client program should call class functions rather than recreating a task available in a class function.
- 4) The class functions themselves (with the exception of #5 in the list) should not do any output.

Extra Credit Opportunity

For an extra 10%, modify your class so that negative money amounts can be represented. To test your program, enter a value for a which is smaller than the value entered for b or enter a negative value for one or both money values.

Create a client program to test your class. Since we're not writing a real application using this class, we just need to call each of the functions to make sure each one operates as it should. The program should read in from the keyboard values for two money objects and test the class functions, producing output similar to the following example (where a and b are the names of money objects). You don't have to demonstrate all possible ways of declaring and assigning values to money variables; the client can either call the class function to input a money value (#3 in the list of required functions), or the client can input the value and call the class function to assign the value to the money variable (#4 in the list of required functions).

```
For a = 29.99, dollars = 29 and cents = 99
For b = 14.95, dollars = 14 and cents = 95
29.99 + 14.95 = 44.94
29.99 - 14.95 = 15.04
29.99 * 5 = 149.95      Note: the 5 used here can be a constant
14.95 * 5 = 74.75
29.99 / 3 = 9.99      Note: the 3 used here can be a constant
14.95 / 3 = 4.98
29.99 < 14.95 is false
14.95 < 29.99 is true
29.99 = 14.95 is false
29.99 > 14.95 is true
14.95 > 29.99 is false
```

Then prompt the user to input new values for a and b and repeat the sequence to produce similar output using the new values.

Example of a partial client program:

```
int main ()
{
    Money a, b, c;    // a, b, and c are initialized to 0.00

    cout << "Enter a dollars-and-cents value, like 14.75 => ";
    cin >> a;        // or a.Input(cin);
    cout << "Enter another dollars-and-cents value    => ";
    cin >> b;        // or b.Input(cin);

    cout << "For a = " << a << ", dollars = " << a.GetDollars()
        << " and cents = " << a.GetCents() << endl;

    c = a + b;      // or c = a.Add(b);
    cout << a << " + " << b << " = " << c << endl;
    // or if you don't have the output operator overloaded:
    // a.Output(cout);
    // cout << " + ";
    // b.Output(cout);
    // cout << " = ";
    // c.Output(cout);
    // cout << endl;

    c = a * 5;      // or c = a.Multiply(5);
    cout << a << " * 5 = " << c << endl;

    cout << a << " < " << b << " is " << (a < b) << endl;

    return 0;
}
```

