CMPE-50 Object-Oriented Concepts and Methodologies, Tarng, Spring 2021
Homework #3

Due: 3/27/2021 midnight

The submission of the homework should be the cpp files with the output in the code comment. Each problem needs to have a complete program, meaning, it needs to contain the main() function and some test code and data. Therefore, you need to submit six cpp files in total. Do not zip the files. Name the files in the following way: CMPE50-HW-3-1.cpp, CMPE50-HW-3-2.cpp, etc. If the solution includes some input files, also submit the input files.

Total 50 points

1. [10 pts] (Chapter 10: Class) (Based on Programming Exercise 10.3)
   Redefine `CDAccount` from Display 10.1 (see Chapter 10 slides) so that it is a class rather than a structure. Use the same member variables as in Display 10.1 but make them private. Include member functions for each of the following:
   a. one to return the initial balance, one to return the balance at maturity,
   b. one to return the interest rate, and
   c. one to return the term.
   d. Include a constructor that sets all of the member variables to any specified values, as well as
   e. a default constructor.

   Embed your class definition in a test program.

2. [10 pts] (Chapter 10: Class) (Based on Programming Exercise 10.6)
   Define a class called `Month` that is an abstract data type for a month. Your class will have one member variable of type `int` to represent a month (1 for January, 2 for February, and so forth). Include all the following member functions:
   a. a constructor to set the month using the first three letters in the name of the month,
   b. an input function that reads the month as an integer,
   c. an output function that outputs the month as an integer,
   d. an output function that outputs the month as the first three letters in the name of the month, and
   e. a member function that returns the next month as a value of type Month.

   The input and output functions will each have one formal parameter for the stream. Therefore, you can invoke the input or output function with either the console I/O or file I/O objects. Embed your class definition in a test program.

3. [15 pts] (Chapter 11: Class friend functions)
   Redefine the `DayOfYear` class from Display 11.2 (see Chapter 11 slides) by adding two **friend functions**. The source code of Display 11.2 is available in the `SavitchCPP9_SourceCode.zip` (on Canvas). The first friend function is named `isAfter` and will take two argument of the type `DayOfYear`. The function after `isAfter` returns `true` if the first argument represents a date that comes after the date represented by the second argument; otherwise, the function returns `false`. The second friend function is `isBefore`. It works the same way as `isAfter` but the result is the reverse of `isAfter` -- if the first argument is a date that comes before the date of the second argument, it returns `true`. Otherwise, it returns `false`.

   Add another friend function called `add`, which takes two arguments of type `DayOfYear`. It returns a `DayOfYear` object. The add function adds the dates of the two input arguments and return the result.

   For simplicity purpose, you can assume each month has 30 days. If the resulting date exceeds December 30th, it is wrapped around from January 1st. Embed your class definition in a test program.

4. [15 pts] (Chapter 11: Operator Overloading)
   The following is a definition of a called `Duple`. Objects of type `Duple` can be used in any situation where ordered pairs are needed. Write the implementation of the overloaded operators >> and << so that object class `Duple` are to be input and output. During input, users need to enter two integer numbers. The output needs to be in the form (5, 6), (7, -4), (-12, 55), or (-38, -49), so forth. Also implement the two constructors where one is the default constructor and the other with two integers as the arguments. Add an overloaded binary operator + to add two `Duple` objects according to the rule

   (a, b) + (c, d) = (a + c, b + d)

   Implement the overloaded binary operator – analogously.

```
#include <iostream>
using namespace std;
class Duple
{
public:
    Duple();
    Duple(int first, int second);
    friend istream& operator >>(istream& ins,
            Duple& second);
    friend ostream& operator <<(ostream& outs,
            const Duple& second);
private:
    int f;
    int s;
};
```