Programming Assignment #2

Learning objective : This assignment will provide further practice with implementing classes.

- Creating a class
- Creating private and public sections of a class
- Implementing the class
- Adding new member functions and features to an existing class
- Declaring objects
- Multiple file compiles
- Character array manipulation
- Good programming practices

Description:

For this homework, you will write a class called Date, in the files **`date.h`** and **`date.cpp`**, for creating and using objects that will store valid dates of the year.

This class should be portable, so it should work with any up-to-date C++ compiler. Make sure that it works with g++ before you hand it in. You should write at least one test driver (main routine that calls the class) **of your own** to test the functionality of the class.

Programming Specifications:

1) An object of type Date should represent a calendar date in terms of month, day, and year, as on a 12-month A.D. calendar. The valid months are January through December, a valid day must correspond to a valid day for the given month, and the year must be a positive number. Your object should also store a format setting, to be used for display of dates to the screen. There will be more than one possible format. The class features (public interface) should work exactly as specified, regardless of what program might be using Date objects.

Note: For purposes of easy input (from keyboard or into functions), date values will be specified with integers. Month values will be 1 for January, 2 for February, etc... on to 12 for December. A valid day value will be an integer between 1 and the number of days in the month. Valid year values are positive numbers.

2) Your Date class must provide the following services (i.e. member functions) in its public section. These functions will make up the interface of the Date class. Make sure you use function prototypes as specified here. (You may write any other private functions you feel necessary, but the public interface must include all the functionality described here).

The constructor(s):

The Date class should have a constructor that allows the user to specify the values for the month, day, and year, using integer values, when the object is declared. If any of the values would result in an invalid date, the constructor should throw out the erroneous information and initialize the object to represent 1/1/2000 (January 1, 2000) instead. Also, you should allow a Date object to be declared without specified values, in which case it should initialize to 1/1/2000 also.

Examples: These declarations should be legal, and the comment gives the initialized date

```
Date d1;              // initializes to Jan 1, 2000
Date d2(3,4,1992);    // initializes to March 4, 1992
Date d3(13,30,1990); // invalid month, initializes to Jan 1, 2000 instead.
```

`void Input()`

This function should prompt the user to enter a date, and then allow the user to input a date from the keyboard. User input is expected to be in the format *month/day/year*, where month, day, and year are integer values. Whenever the user attempts to enter an invalid date, the Input function should display an appropriate error message (like "Invalid date. Try again: ") and make the user re-enter the whole date. A few examples of some good and bad inputs:

```
Legal:    1/4/2000 , 2/28/1996 , 12/31/1845
Illegal:  13/12/1985 , 11/31/2002 , 8/30/-2000
```

You may assume that the user entry will always be of the form:
`M/D/Y` `where M, D, and Y are integers, and the slash characters are always present.`

`void Show()`

This function should simply output the date to the screen. There will be more than one possible format for this output, however, and your class will need to store a format setting. The Show function should use the format setting to determine the output. (There will be a member function that allows the setting to be changed). When a Date object is created, the format setting should start out at the "Default" setting. The possible formats are shown in the following table:

Name	Format	Example	Explanation
Default	*M/D/Y*	10/4/1998	This will look like the input from the Input function. Print the month, day, and year as integer values.
Two-Digit	*mm/dd/yy*	10/04/98	Fixed size format, in which the month, day, and year values are always 2 digits Some values may need to be padded with a leading zero, and year values always show the low 2 digits.
Long	*month D, Y*	Oct 4, 1998	This display format should show the abbreviated month name, then the day, and the full year. Month abbreviations are Jan, Feb, Mar, Apr, May, June, July, Aug, Sept, Oct, Nov, and Dec

`bool Set(int m, int d, int y)`
This function should set the date to the specified values (the first parameter represents the month, the second represents the day, and the third represents the year). If the resulting date is an invalid date, the operation should abort (i.e. the existing stored date should not be changed). This function should return `true` for success and `false` for failure (i.e. invalid date sent in).

`int GetMonth()`
`int GetDay()`
`int GetYear()`
These are "accessor" functions, and they should return the month, day, and year, respectively, to the caller.

`bool SetFormat(char f)`
This function allows the caller to change the format setting. The setting should be adjusted inside the object based on the character code passed in. This means that future uses of the Show function will display in this given format until the format is changed. The valid setting codes that can be passed in are:

```
'D' = Default format
'T' = Two-Digit format
'L' = Long format
```

If an invalid setting code is passed in, do not alter the current format setting. This function should return `true` for successful format change, and `false` for failure (invalid setting given).

`void Increment(int numDays = 1)`
This function should move the date forward by the number of calendar days given in the argument. Default value on the parameter is 1 day. Examples:

```
  Date d1(10, 31, 1998);       // Oct 31, 1998
  Date d2(6, 29, 1950);        // June 29, 1950

  d1.Increment();              // d1 is now Nov 1, 1998
  d2.Increment(5);             // d2 is now July 4, 1950
```

`int Compare(const Date& d)`
This function should compare two Date objects (the calling object and the parameter), and should return: -1 if the calling object comes first chronologically, 0 if the objects are the same date, and 1 if the parameter object comes first chronologically. The function should not change either object. Example:

```
  Date d1(12,25,2003);         // Dec 25, 2003
  Date d2(5,18,2002);          // May 18, 2002

  d1.Compare(d2);              // returns 1  (since d2 comes first)
  d2.Compare(d1);              // returns -1 (calling object is d2, comes
first)
```

General Requirements:

- all member data of the Date class must be **private**.
- The **`const`** qualifier should be used on any member functions where it is appropriate
- the only library that may be used in these class files is <iostream>.
- You only need to do error-checking that is specified in the descriptions above. If something is not specified (e.g. user entering a letter where a number is expected), you may assume that part of the input will be appropriate.
- user input and/or screen output should only be done where described (i.e. do not add in extraneous input/output).
- no global variables, other than constants
- You are not required to handle leap years in this class. You may make the general assumption that a year always has 365 days.

Extra Credit:

A) (10 points) Make your Date class handle leap years, wherever appropriate. Remember that a leap year has one extra day in it (Feb. 29), and the rule for leap years is as follows: A year is a leap year if it is divisible by 4, except for century years (years ending in 00). A century year is a leap year only if it is divisible by 400. (For instance, 2000 is a leap year, but 1900 is **not**).

B) (10 points) The *Julian Day* for a calendar year is defined as the number of the day in the calendar year. There are 365 days in a regular calendar year, so each day is numbered in order (1 - 365). For instance, January 15 has a Julian date of 15, and February 10 has a Julian date of

41. Add in one more formatting code ('J', for Julian Date), to be allowed by the SetFormat function. This setting should result in the output of a date in the Julian Date format, which should look like this: *YY-JJJ*. The Julian Day will always be printed as a 3-digit number, and the year as a 2-digit number (last two digits). Examples:

```
Feb  1, 1998 would print as:  98-032
May 17, 2002 would print as:  02-137
```

C) (10 Points) Overload the << operator to perform the same function as the Show().

Note: If you extra credit parts A and B, the Julian Date must handle leap years as well.

Testing Your Class:

You will need to test your class, which means you will need to write one or more main programs that will call upon the functionality (i.e. the public member functions) of the class and exercise all of the different cases for each function. You do not need to turn any test programs in, but you should write them to verify your class' features. I will post a small example on the Blackboard site.

Submitting:

Submit the following files on the Assignment Blackboard portal:

```
date.h
date.cpp
```

Grading Criteria:

- The program compiles. If the program does not compile no further grading can be accomplished. Programs that do not compile will receive a zero.
- (25 Points) The program executes without exception and produces output. The grading of the output cannot be accomplished unless the program executes.
- (25 Points) The program produces the correct output.
- (25 Points) The program specifications are followed.
- (10 Points)The program is documented (commented) properly.
- (5 Points)Use constants when values are not to be changed
- (5 Points)Use proper indentation
- (5 Points)Use good naming standards