Programming Assignment #2

Due: 11:59pm, Wednesday, Feb. 13th

Objective:

This assignment will provide further practice with classes and objects, and deepen the understanding of basic OO programming.

Task:

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++ on linprog before you hand it in**. You should write test programs **of your own** to test the functionality of the class.

Details:

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 no less than 1900. 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, and so on up to 12 for December. A valid day value will be an integer between 1 and the number of days in the month (definitely no more than 31). Valid year values are positive integer numbers greater than or equal to 1900.

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 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. For each function and each parameter, you need to think carefully if **`const`** is needed for declaration and definition.

   2.1. Constructor(s):

   The Date class should have one (or multiple) 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 1/1/2019 (January 1, 2019) instead. Also, you should allow a Date object to be declared without

specified values, in which case it should initialize to 1/1/2019 also. **Examples:** These declarations should be legal, and the comment gives the initialized date:

```
Date d1;            // initializes to Jan 1, 2019
Date d2(3,4,1992);  // initializes to March 4, 1992
Date d3(13,30,1990); // invalid month, initializes to Jan 1, 2019 instead.
```

2.2. void Input():

This function should prompt the user to enter a date (like **"Input date in month/day/year format: "**), 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, 12/31/1945
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 (/ with the ASCII code 47) are **always** present.

2.3. int GetMonth()
int GetDay()
int GetYear()

These are "accessor" functions, and they should return the month, day, and year, respectively, to the caller.

2.4. 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).

2.5. void Increment()

This function should move the date forward by ONE calendar day. Examples:

```
Date d1(10, 31, 1998); // Oct 31, 1998
Date d2(2, 28, 2016);  // Feb 28, 2016

d1.Increment();        // d1 is now Nov 1, 1998
```

```
d2.Increment();        // d2 is now Feb 29, 2016 (a leap year)
```

2.6. int DayofWeek()

This function determines day of the week for a valid Date object, and the returned values range from 0 to 6: 0 stands for Sunday, 1 for Monday, ….., and 6 for Saturday. Here is one algorithm to compute day of the week, called Sakamoto's Algorithm, from Wikipedia for your reference:

```
int dayofweek(y, m, d) /* 1 <= m <= 12,  y > 1752 (in the U.K.) */
{
      static int t[] = {0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};
      y -= m < 3;
      return (y + y/4 - y/100 + y/400 + t[m-1] + d) % 7;
}
```

2.7. 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)
```

2.8. void ShowByDay()

This function will print out the detailed information for a Date object. The output format is like day-of-week month/day/year. Here day-of-week should be a literal string, namely Sunday, Monday, ……, Saturday, rather than 0, 1, ……, 6. Example:

```
Date d1(2,17,2019);           // Feb 17, 2019
d1.ShowByDay();               // Output: Sunday 2/17/2019
```

2.9. void ShowByMonth()

This function will print out all the days of the month in which the Date object is in a calendar format. Specifically, the first line of output will be **a header** consisting of a static line of string like

Su Mo Tu We Th Fr Sa

where Su stands for “Sunday”, Mo stands for “Monday”, and so forth. Each two-letter abbreviation is separated from the right one (if there exists) with FOUR spaces. Su (for Sunday) is at the beginning of the line, and Sa (for Saturday) is at the end of the line.

In the following **content** lines, each day of the month, from the first day to the last day of that month, is printed out that is perfectly aligned with its corresponding day-of-week (Note that a line need to be wrapped up to the next new line after a Saturday has been printed out). For a day that is in the range of 1 and 9 (inclusive), a character ‘0’ should be added as a prefix for output.

Example:

```
Date d1(2,17,2019);         // Feb 17, 2019
d1.ShowByMonth();
```

And the output should look like:

```
Su    Mo    Tu    We    Th    Fr    Sa
                                01    02
03    04    05    06    07    08    09
10    11    12    13    14    15    16
17    18    19    20    21    22    23
24    25    26    27    28
```

Hints:

You may consider implement a series of private functions that are invisible to end-users of your class, but frequent invoked by public member functions. Here are some examples:

1. This class may need to determine if a given year is a **leap year** or not. 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**).
2. This class may need to determine for a given month and a year, how many days there are in that month. For example, there are 29 days in February 2016 (a leap year), and there are 31 days in July 2018.
3. This class may need to determine if a date (created by constructors or input, for instance) is valid or not.

Testing your class:

We will provide a sample program (called test.cpp) that uses objects of type Date and illustrates the usage of the member functions. We also provide the output from the execution of the

test.cpp program. Your class declaration and definition files must work with our program, as-is (do not change our program to make your code work!). You should write your own test routines to further test the functionality of your class. **Please keep in mind**, test.cpp is just a **sample**. Your class must meet the specified requirements listed above -- not **just** satisfy **this** sample program. Your class will be tested with a larger and significantly more thorough set of calls than this sample program represents.

General Requirements

- **You need to implement the Date class all by yourself!** You may use the iostream library for output. DO NOT use any other C++ standard libraries, such as chrono, local_t, time and so on, or any third-party libraries, such as Boost, for API calls;
- No global variables, other than constants!
- All member data of your class must be private;
- The const qualifier should be used on any member functions and parameters where it is appropriate;
- 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);
- When you write source code, it should be readable and well-documented.

Submitting:

Program submissions should be done through Canvas. Do **not** send program submissions through e-mail -- e-mail attachments will **not** be accepted as valid submissions.

General Advice - e-mail a copy of your finished homework files to your own FSU account. This e-mail will have a time stamp that shows when they were sent (i.e. before the due date would be the best idea), and they will also serve as a backup. It's not a bad idea to keep a copy on your CS account (as well as on a personal computer) -- backing up your work is a GOOD thing!

For HW #2, submit a zipped package (named **`proj2.tar.gz`**) including the following files

```
date.h
date.cpp
makefile (the final executable is named proj2)
readme (optional)
```

Make sure your filenames are these exact names, and do not submit your test.cpp file (or any other main program you might create for testing purposes).