National University of Sciences & Technology

Institute of Geographical Information Systems

CS-212: Object Oriented Programming; BEGI-2021 Fall 2022

ASSIGNMENT		
CLO 3 Demonstrate the ability to create and use OOP constructs to map real world scenarios.		
Maximum Marks: 100	Date: 9th December 2022	Due date: 18th Dec 2022
Name:_____		Reg No_____

Objective

This assignment will provide further practice with implementing classes.

Task

For this assignment 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.

Program Details and Requirements

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.

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 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.

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 of the functionality described here.

Public Interface

1. *Constructor*
   The Date class should have a constructor that allows the user to specify the values for the month, day, and year using integer values. 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. You should also allow a Date object to be declared without specified values in which case it should initialize to 1/1/2000.

   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
```

2. `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 (e.g. "Invalid date. Try again: ") and make the user reenter the whole date. 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.

Examples:

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

3. `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. When a Date object is create, 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 the 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, Dec

4. `bool Set(int m, int d, int y)`

This function should set the date to the specified values of month, day, and year respectively. 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.

5.
```
int GetMonth()
int   GetDay()
int GetYear()
```

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

6. `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 the future uses of the Show function will display in this given format until the format is changed.

   Valid setting codes that can be passed in:
   `D` = Default format
   `T` = Two-Digit format
   `L` = Long format

7. `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
```

8. `int Compare(const Date&  d)`

   This function should compare two Date objects (the calling object and the parameter) without modifying either object. It returns one of the following:

   - `-1`, if the calling object comes first chronologically
   - `0`, if the objects are the same date
   - `1`, if the parameter object comes first chronologically

   Examples:

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

General Requirements

- No global variables, other than constants
- All member data of your class must be private
- The only libraries that may be used in these class files are `<iostream>`, `<iomanip>`, and `<string>`. While this class can be written without the string class, you may use it to store words like “January”
- 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 screen output should only be done where described. Do not add any extraneous input/output
- You are not required to handle leap years in this class. You may make the general assumption that a year always has 365 days.
- When you write source code, it should be readable and well documented
- Your `date.h` file should contain the class declaration only. The `date.cpp` file should contain the member function definitions.
- If you change formatting properties of the cout stream object from inside a member function, you must put it back the way it was when your function started. You do not want to mess up anybody else's outputs (e.g. main function calling your features).

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 should not submit these test programs, but you should write them to verify your solution works correctly. I included a sample driver to get you started, but it does not test every possible test case.

Submitting

Archive your `date.h`, `date.cpp`, and `README` files into a simple tar ball (no compression).

General Advice

- Periodically (e.g. nightly) make a backup of your assignment to another machine (e.g. personal computer, email). Computers die and accidents happen, having a backup prevents you from having to start from scratch.

Extra Credit

1. Make your Date class handle leap years where appropriate. Remember that a leap year has one extra day in it (Feb 29). 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 (e.g. 2000 is a leap year but 1900 is not).

2. 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 onemore 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 `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