CS 215 - Fundamentals of Programming II
Fall 2019 - Programming Project 2
20 points

Out: September 16, 2019
Due: September 25, 2019 (Wednesday)

Reminder: Programming Projects (as opposed to Homework problems) are to be your own work. See syllabus for definitions of acceptable assistance from others.

Consider the following specification for a Date class that models a calendar date in the Common Era. The analysis of each operation is given formally and the parameters must be declared in the order specified, but the design of each operation is given informally in English.

Specification for Date Class
Data Attributes:

Objects	Type	Name
month of the date	int	mm
day of the date	int	dd
year of the date	int	yyyy

The class invariant (i.e., all valid objects must meet this condition) is:

- month is 1 to 12
- day is 1 to 31 for January, March, May, July, August, October, and December; 1 to 30 for April, June, September, and November; 1 to 28 for February in a non-leap year; and 1 to 29 for February in a leap year
- year is greater than 0

For those who do not recall the definition of a leap year, it is any year that is divisible by 4 but not by 100, or is divisible by 400. For example, 2016 is a leap year, 1900 is not a leap year, and 2000 is a leap year.

Operations

- Default/Explicit-value constructor - initialize attributes from passed values
  Must have default argument values of 1, 1, 2019, for initial_month, initial_day, and initial_year (i.e., January 1, 2019) - there is **no other constructor** for this class.
  Precondition: Initial values must meet class invariant above. If they do not, the default values should be stored and an error message should be displayed saying so.

  Analysis

Objects	Default	Type	Movement	Name
initial month	1	int	received	initial_month
initial day	1	int	received	initial_day
initial year	2019	int	received	initial_year

- month - Returns the month of this date

  Analysis

Objects	Type	Movement	Name
month of date	int	returned	mm

- day - Returns the day of this date

  Analysis

Objects	Type	Movement	Name
day of date	int	returned	dd

- year - Returns the year of this date

  Analysis

Objects	Type	Movement	Name
year of date	int	returned	yyyy

- to_string - Returns the string equivalent of this date in the same format as operator<<, e.g. "1/1/2019". Hint: you should use an ostringstream to do this. See design notes below.

  Analysis

Objects	Type	Movement	Name
string name of date	string	returned	-----

- string_name - Returns the written out equivalent of this date as a string, e.g. "January 1, 2019" for 1/1/2019. Hint: you should use an ostringstream to do this. See design notes below.

  Analysis

Objects	Type	Movement	Name
string name of date	string	returned	-----

- day_of_week - Returns the string name of the day of the week this date represents according to the following formula.† Let a, b, c, and d be integers defined as follows:

  a = The number of the month of the year, with March = 1, April = 2, and so on, with January and February being counted as months 11 and 12 **of the previous year**.
  b = The day of the month
  c = The year of the century
  d = The century

  For example, July 31, 1929 gives a = 5, b = 31, c = 29, d = 19; January 3, 1988 gives a = 11, b = 3, c = 87, d = 19. Now calculate the following integer quantities:

  w = The integer quotient of (13a -1) / 5
  x = The integer quotient of c / 4
  y = The integer quotient of d / 4
  z = w + x + y + b + c - 2d
  r = z reduced modulo 7, that is, r = z % 7 (except that the C++ % operator sometimes returns a negative number; if this happens, add 7 to the result), r = 0 represents Sunday, r = 1 represents Monday, and so on.

  Analysis

Objects	Type	Movement	Name
string name of day of the week	string	returned	-----

† From Nyhoff, *C++: An Introduction to Data Structures*, Prentice-Hall, Inc. 1999, Problem 47 on page 167.

- operator== - **friend** overloaded operator, returns true if the two date objects represent the same date

  Analysis

Objects	**Type**	**Movement**	**Name**
a date	Date	received	lhs
another date	Date	received	rhs
result of comparison	bool	returned	-----

- operator< - **friend** overloaded operator, returns true if the left-hand side date is earlier than the right-hand side date

  Analysis

Objects	**Type**	**Movement**	**Name**
a date	Date	received	lhs
another date	Date	received	rhs
result of comparison	bool	returned	-----

- operator <=, operator >, operator>=, operator!= - **friend** overloaded operators, receives two Date objects and returns true or false as appropriate. **Implement these functions using operator== and operator<.**

- operator<< - **friend** overloaded operator function that writes the date attributes to an output stream in format `MM/DD/YYYY`, single digit month and day are allowed.

  Analysis

Objects	**Type**	**Movement**	**Name**
output stream	ostream	received, passed back, and returned	out
date object	Date	received	a_date

- operator>> - **friend** overloaded operator function that reads date values from an input stream without prompting in format `MM/DD/YYYY` (month and day may be a single digit) and stored in the attributes. Must check that the input meets the class invariant above. If not met, the date object is unchanged and the input stream should be put in the failed state and returned. Also, if the input stream fails for any reason (e.g., entering a letter instead of a digit), it should just be return as is. See design notes below.

  Analysis

Objects	**Type**	**Movement**	**Name**
input stream	istream	received, passed back, and returned	in
date object	Date	passed back	a_date

Assignment

This assignment will be distributed by the GitKeeper submission system. Follow the directions in the assignment email. Be sure to commit your changes early and often. **The first commit for this assignment is expected to be by Friday, September 20.**

The Date class definition and friend operator function prototypes must be put in header file `date.h` with suitable compilation guards. The implementations of the Date class and friend operator functions must be put in source file `date.cpp`. Note that all public names (the class name, the operation names, and the file names) must be as specified above including (lack of) capitalization.

Write a main program that adequately tests your Date class in file **`datedriver.cpp`**. This program should demonstrate that your Date class meets all of the specifications given above. Part of your grade will depend on how well you test your class. In addition, the submission system will run a specific driver program to test your Date class.

You must submit a makefile named **`Makefile`** for your project that has a first target for an executable file named **`datedriver`** created from linking objects files and targets for the individual object files that are created from compiling the source files. It should conform to the examples demonstrated in class.

REMINDER: Your project must compile for it to be graded. Submissions that do not compile will be returned for resubmission and assessed a late penalty. Submissions that do not substantially work also will be returned for resubmission and assessed a late penalty.

Follow the guidelines in the *C++ Programming Style Guideline* handout. As stated in the syllabus, part of the grade on a programming project depends on how well you adhere to the guidelines. The grader will look at your code listing and grade it according to the guidelines.

Design Notes

1. Since there are multiple places (the constructor and operator>>) where the class invariant is checked, it might be worthwhile writing a function that receives the month, day, and year numbers and returns true if they represent a valid date and false otherwise. **Such utility functions are free functions and should have their prototypes placed at the beginning of the source file in which it is used and their function definitions placed at the end of the source file.**

2. By convention, when an operator>> encounters invalid input, it changes the input stream state to a failed state. For this project, the input data should be read into local variables first. **Note that there are 3 integer and 2 character inputs.** If the stream has already failed after reading in the input (tested using **`in.fail()`**, which returns true when the stream is in a failed state), the function should return **`in`** immediately. After this test, the input integers should be checked against the class invariant and the input characters checked whether they are '/'. If either test fails, the stream is put into a failed state using **`in.setstate (ios_base::failbit)`**. In other words, operator>> will have the following structure:

```
<read data values into local variables here>

// check if input stream has failed or eof
if (in.fail())
   return in;  // if fail, return immediately

// check class invariant and formatting
if (<invariant test not true> or <format not correct>)
{
   in.setstate (ios_base::failbit);  // set fail bit then return
   return in;
}

<set the attribute data members here>
```

3. Reminder: As demonstrated for the Rational class **`to_string`** operation, a stringstream is a stream object that is attached to a string. (This is analogous to an fstream begin attached to a file.) Stringstreams are defined in the **`<sstream>`** library header. As with fstreams, there is an input stringstream (**`istringstream`**) and an output stringstream (**`ostringstream`**) type. Since these objects are a kind of I/O stream, the extraction (>>) and insertion (<<) operators are used. Note: only use ostringstream if some of the objects to be "output" are not strings or characters. When all of the objects are strings, use the concatenation operator (+).

How to submit

Reminder: all public names (the class name, the operation names, and file names) must be as specified including capitalization, and all parameters must be implemented in the order listed. Internal names (private data member, private member functions, function parameters, local variables) can be whatever you wish. It also requires that the format of any created strings or output be exactly as expected including whitespace, capitalization, and spelling. Submit projects by first adding and committing changes, then pushing the project. See handout *Using GitKeeper for Submitting Assignments* for additional information.