Assignment 3: Date class

Collaboration Policy

You may not use code from any source (another student, a book, online, etc.) within your solution to this assignment. In fact, you may not even look at another student's solution or partial solution to this assignment. You also may not allow another student to look at any part of your solution to this exercise. You should get help on this assignment by coming to the instructor's or TA's office hours or by posting questions on Piazza (you still must not post assignment code publically on Piazza.) See the full Course Collaboration Policy here: Collaboration Policy

Problem Definition

In this assignment, you will build a Date class and a test harness (main function) to test it.

Specifications

Below is the interface for the Date class: it is our "contract" with you: you have to implement everything it describes, and show us that it works with a test harness that puts it through its paces. The comments in the interface below should be sufficient for you to understand the project (**use these comments in your Date declaration**), without the need of any further documentation. But of course, as always, you can ask us any questions you may have on Piazza.

Date Class interface:

Note: Changing the public interface will result in a 0 on the assignment.

Private Data Fields

● day - unsigned

● month - unsigned

● monthName - string

● year - unsigned *(NOTE: this class is NOT designed to handle negative years, aka "BC")*

You must declare all 4 of these variables in this exact order.

Public Member Functions

This is the public interface of the class - your contract with the user.

Date( )

default constructor: creates the date January 1st, 2000.

Date(unsigned m, unsigned d, unsigned y)

parameterized constructor: month number, day, year - e.g. (3, 1, 2010) would construct the date March 1st, 2010

If any of the arguments are invalid (e.g. 15 for month or 32 for day) then the constructor must construct instead a valid Date as close as possible to the arguments provided - e.g. in above example, Date(15, 32, 2010), the Date would be corrected to Dec 31st, 2010. In case of such invalid input, the constructor will issue a console error message:

```
Invalid date values: Date corrected to 12/31/2010.
```
(with a newline at the end).

You are required to use the daysPerMonth helper function in this constructor (see below).

Date(const string &mn, unsigned d, unsigned y)

parameterized constructor: month name, day, year e.g. (December, 15, 2012) would construct the date December 15th, 2012

If the constructor is unable to recognize the string argument as a **valid month name**, then it will issue a console error message:

```
Invalid month name: the Date was set to 1/1/2000
```
(with a newline at the end).

(Note: This is a not necessarily a good way to handle constructor errors, but until you learn about exceptions it's the best we can do)

If month name is correct but the **day is incorrect** it should correct it as in the **previous constructor**.

Your constructor must recognize both "december" and "December" as month name, but not necessarily "dec" or "Dec" (though you are encouraged to attempt that).

void printNumeric( ) const

const accessor function to output a Date exactly in the format "`3/1/2012`".

Do **not** output a newline (endl) at the end!

void printAlpha( ) const

const accessor function to output a Date exactly in the format "`March 1, 2012`". The first letter of the month name is upper case, and the month name is printed in full - so Jan, january, jan are not valid formats.

Do **not** output a newline (endl) at the end!

Private Member Functions

These are "helper" functions - member functions that will never be needed by a user of the class, and so do not belong to the public interface (which is why they are "private"). They are, however, needed by the interface functions ("public member functions"), which use them to test the validity of arguments and construct valid dates.

bool isLeap(unsigned) const

Tests for leap year.

The rule is:

> (year % 4 == 0) implies leap year
>
> *except* (year % 100 == 0) implies NOT leap year
>
> *except* (year % 400 == 0) implies leap year

So for instance the year 2000 is a leap year, but 1900 is not; 2004, 2008, 2012, 2016, etc. are all leap years

unsigned daysPerMonth(unsigned m, unsigned y) const

Returns number of days allowed in a given month e.g. daysPerMonth(9, 2000) returns 30.

It must correctly calculate February's days for leap and nonleap years, and therefore it also needs the year as a parameter.

QUESTION: what will you return if an invalid month number (e.g. 13) is passed in?

NOTE: in this function and the next two, the numbers 1 through 12, the numbers {28, 29, 30, 31}, and the strings "January" through "December" are NOT considered to be "magic" numbers or strings, so you do not need to make constants for these values.

string name(unsigned m) const

Returns the name of a given month e.g. name(12) returns the string "December"

unsigned number(const string &mn) const

Returns the number of a given named month e.g. number("March") returns 3

Testing:

Make sure to thoroughly test your class before submitting. Here is a sample of some tests you should run (you should think of and develop more tests to help you find potential flaws in your logic):

Sample main.cpp

```
//includes...
void test1()
{
      cout << "1. Default" << endl;
      Date x = Date();
      cout << "numeric:\t" ;
      x.printNumeric(); cout << endl;
      cout << "alpha:\t";
      x.printAlpha();
      cout << endl;
      cout << "-------------" << endl;
}
void test2()
{
      cout << "2. Numeric Date: 7.3.1991" << endl;
      Date x = Date(7,3,1991);
      cout << "numeric:\t" ;
      x.printNumeric();
      cout << endl;
      cout << "alpha:\t";
      x.printAlpha();
      cout << endl;
      cout << "-------------" << endl;
}
void test3()
{
      cout << "3. Alpha Date: July 3, 1991" << endl;
      Date x = Date("July",3,1991);
      cout << "numeric:\t" ;
      x.printNumeric();
      cout << endl;
      cout << "alpha:\t";
      x.printAlpha();
      cout << endl;
      cout << "------------- " << endl;
}
void test4()
{
      cout << "4. Invalid Numeric Date: 0.3.1991  > 1.3.1991" << endl;
      Date x = Date(0,3,1991);
      cout << "numeric:\t" ;
      x.printNumeric();
      cout << endl;
      cout << "alpha:\t";
      x.printAlpha();
      cout << endl;
      cout << "-------------" << endl;
}
void test5()
{
      cout << "5. Invalid Alpha Date: 1.adf.1991  > 1.1.2000" << endl;
      Date x = Date("adf",1, 1991);
      cout << "numeric:\t" ; x.printNumeric();
      cout << endl;
      cout << "alpha:\t";
      x.printAlpha();
      cout << endl;
      cout << "-------------" << endl;
}
int main()
{
      test1();
      test2();
      test3();
      test4();
      test5();
      //you should make many more than these...
}
```

Output Specifications:

Read the specifications for the print functions carefully. The only 'cout' statements in your Date code should be:

1. the "Invalid Date" warnings in the constructors(see constructor definitions); and
2. in your two print functions(see print functions)

Here are a few examples.

```
1. Default
numeric: 1/1/2000
alpha: Jan 1, 2000
-------------
2. Numeric Date: 7.3.1991
numeric: 7/3/1991
alpha: July 3, 1991
-------------
 3. Alpha Date: July 3, 1991
numeric: 7/3/1991
```

```
alpha: July 3, 1991
-------------
4. Invalid Numeric Date: 0.3.1991  > 1.3.1991
Invalid date values: Date corrected to 1/3/1991.
numeric: 1/3/1991
alpha: January 3, 1991
-------------
5. Invalid Alpha Date: 1.adf.1991  > 1.1.2000
Invalid month name: the Date was set to 1/1/2000.
numeric: 1/1/2000
alpha: January 1, 2000
-------------
```

Extra Credit:

For 15 points of extra credit(out of a possible 100 points), implement the function:

```
// Add a number of days to one date to get another:
// this number may be positive or negative.
// Obviously, the new date will have to be a valid date,
// accounting for the correct number of days in each month,
// and for leap years.
```

Date addDays(int) const

Note: Always be careful when mixing ints and unsigneds!! g++ won't generate any warnings or errors when you do so (not even with all the flags turned on!) - but what happens when you add say -20 to unsigned 10? Or assign a negative int to an unsigned variable?

Think about this, and be ready to use static_casts if necessary!

What to Submit:

For this assignment you will turn in the following files (**case sensitive**):

- main.cpp
- Date.h
- Date.cpp

You will again be submitting your files to R'Sub.