CSE 303, Spring 2009
Homework 7: Date/Birthday (40 points)
Due Saturday, June 6, 2009, 11:30 PM

This **individual (not group)** program focuses on C++ programming with classes and objects. The assignment has two parts: finishing a client program that uses `Date`s, and your own implementation of a `Date` class. Turn in these files:

- `Date.h`, `Date.cpp` - header and implementation for `Date` class
- `birthday.cpp` - client program that computes birthdays/dates using `Date` objects *(skeleton provided)*
- `Makefile` - basic compilation manager script to build your program

`Date` class (`Date.h` / `Date.cpp`):

Part of this assignment asks you to implement a class named `Date`, whose declarations are stored in file `Date.h` and whose definitions are stored in `Date.cpp`. Your `Date` class must have the following public behavior:

- `Date(`***`year`***`, `***`month`***`, `***`day`***`)`
  In this constructor you should initialize a new `Date` representing the given year, month and day.
  You may assume that the parameter values passed are valid.

- `getYear()`
  In this method you should return the `Date` object's year. For example, if the `Date` object represents August 17, 2005, in this method you should return 2005.

- `getMonth()`
  In this method you should return the `Date` object's month of the year, between 1 and 12. For example, if the `Date` object represents August 17, 2005, in this method you should return 8.

- `getDay()`
  In this method you should return the `Date` object's day of the month, between 1 and the number of days in that month (which will be between 28 and 31). For example, if the `Date` object represents August 17, 2005, this method should return 17.

- `setDate(`***`year`***`, `***`month`***`, `***`day`***`)`
  In this method you should set the `Date` to represent the given year, month and day. Assume valid parameters.

- `isLeapYear()`
  In this method you should return `true` if the date represented by the `Date` object is a leap year, otherwise `false`. Normally February has 28 days, and the year is 365 days long. But "leap years" have a 29th day in February, making the year 366 days long. Leap years are those that are multiples of 4, except for multiples of 100 that are not also multiples of 400 (such as 1700 or 1900). For example, 1600, 1876, 1996, 2000, 2004, and 2008 are leap years, so you would return `true`, but 1700, 1800, 1900, 1973, and 2009 are not, so you would return `false`.

- `daysInMonth()`
  In this method you should return the total number of days in the month represented by the `Date` object. The following table lists the number of days in each of the twelve months of the year.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Days	31	28/29	31	30	31	30	31	31	30	31	30	31

  For example, if the `Date` object represents August 17, 2005, this method should return 31. If the `Date` object represents February 14, 2009, you should return 28. If it is February 14, 2008, return 29.

- `dayOfYear()`
  In this method you should return the absolute day of the year for the date represented by the `Date` object. January 1 is absolute day #1, January 2 is day #2, ..., and December 31st is absolute day #365 (#366 in a leap year). The following table lists the number of days in each of the twelve months:

	1	2	3	4	5	6	7	8	9	10	11	12
Date	Jan 1	Feb 1	Mar 1	Apr 1	May 1	Jun 1	Jul 1	Aug 1	Sep 1	Oct 1	Nov 1	Dec1
dayOfYear	1	32	60	91	121	152	182	213	244	274	305	335

  For example, calling this method on a date representing February 13th, 2005 should return `44`, and on a date representing September 19th, 1979 should return 262. March 3, 2004 should return 63.

- `nextDay()`
  In this method you should modify the state of the `Date` object by advancing it 1 day in time. For example, if the `Date` object represents September 19, a call to this method should modify the `Date` object's state so that it represents September 20. Note that depending on the date, a call to this method might advance the `Date` object into the next month or year. For example, the next day after August 17 is August 18; the next day after February 28, 2009 is March 1, 2009; the next day after February 28, 2008 is February 29, 2008; and the next day after December 31, 2008 is January 1, 2009.

- `< (`***`date`***`)`
  You should define an overloaded < operator that returns a boolean value indicating whether one `Date` is "less than" another, that is, whether it comes chronologically earlier. For example, May 30, 2009 is less than September 19, 2009, and December 31, 2008 is less than January 1, 2009, so your < operator would return `true` in these cases. The `Date`s should be compared by reference, as shown in lecture.

- `<< (`***`stream, date`***`)`
  You should define an overloaded << operator so that a client can output a `Date` object to any output stream (`ostream`). This operator is defined outside your class, but still in the same files.

  You should output a `String` representation of the `Date` in a *year/month/day* format. For example, if the `Date` object represents May 24, 2009, output `2009/5/24`. If the `Date` object represents December 3, 1964, output `1964/12/3`. The `Date` should be passed by reference. Your operator should return a reference to the output stream itself, as shown in class.

No `Date` method should perform console I/O. You can add other data/behavior to your class as long as it is `private`.

Any method or operator that does not modify the `Date`, and any unmodified object parameter, should be declared `const`.

You may not utilize any behavior from any existing C++ date-related libraries such as `ctime`.

You can test your `Date` program by running the `birthday` program with it. But `birthday` is not a comprehensive testing program. You may want to create another client program of your own to help test your class.

We encourage you to build your `Date` class incrementally, writing a small amount of code at a time and testing it. You can test an incomplete `Date` class by writing some methods and then creating a small client to call just those methods.

Recall that code in an object's method is able to call any of the object's other methods. Do this to avoid redundancy.

`birthday` client program (`birthday.cpp`):

Another part of this assignment asks you to complete a partially written client program that uses your `Date` class. The reason for this part is to give you a bit more practice writing C++ code, specifically to practice creating and using `Date` objects from a client's perspective, and to give you an appreciation for the usefulness of your `Date` objects in general.

The program prompts the user for today's date and for his/her birthday, each as three integers, *year month day*. It uses this to figure out what day of the month/year the user's birthday falls on, the # of days from today to the user's next birthday, and the total number of days old the user is today. If the user's birthday is today, it prints a Happy Birthday message.

The following are several example logs of execution from the program; user input is bold and underlined. Your program's output should match these examples exactly given the same input. See the course web site for more logs with other input.

```
Please type your birthdate (y m d): 1979 9 19
Info about 1979/9/19:
It is day #19 of 30 of the month
It is day #262 of 365 of the year

Please type today's date (y m d): 2009 5 30
Info about 2009/5/30:
It is day #30 of 31 of the month
It is day #150 of 365 of the year

It will be your birthday in 112 days
You are 10846 days old
```

```
Please type your birthdate (y m d)? 1987 12 31
Info about 1987/12/31:
It is day #31 of 31 of the month
It is day #365 of 365 of the year

Please type today's date (y m d)? 2008 12 31
Info about 2008/12/31:
It is day #31 of 31 of the month
It is day #366 of 366 of the year

Happy birthday!
You are 7671 days old
```

```
Please type your birthdate (y m d)? 1944 11 30
Info about 1944/11/30:
It is day #30 of 30 of the month
It is day #335 of 366 of the year

Please type today's date (y m d)? 1999 1 1
Info about 1999/1/1:
It is day #1 of 31 of the month
It is day #1 of 365 of the year

It will be your birthday in 333 days
You are 19755 days old
```

```
Please type your birthdate (y m d)? 1803 2 28
Info about 1803/2/28:
It is day #28 of 28 of the month
It is day #59 of 365 of the year

Please type today's date (y m d)? 2008 2 29
Info about 2008/2/29:
It is day #29 of 29 of the month
It is day #60 of 366 of the year

It will be your birthday in 365 days
You are 74876 days old
```

Table 1 Expected output logs from four runs of `birthday` program

A skeleton of this client program is provided on the course web site. The skeleton handles the part of the code that prompts the user, creates the initial `Date` objects, and prints the info about them. You must finish the part that counts the number of days until the user's birthday and prints how many days old the user is today.

You can count the number of days between a given pair of `Date`s by advancing one until it reaches another date.

You may assume valid input: that the user will always type integers; the years typed will be at least 1753; the months typed will be between 1-12; and the days typed will be between 1 and the end of the corresponding month. You may also assume that the birthday the user types in will come on or before today's date.

You may put additional behavior in your `Date` class beyond what is specified, but we will test `birthday` with an instructor-provided `Date`, so `birthday` should not depend on unspecified features that exist only in your `Date`.

A sample solution to this program will be placed on `attu` for you to test its behavior.

Makefile and Compilation:

You should submit a basic `Makefile` with the following functionality (it can optionally have more if you like):

- The command `make` or `make all` must compile all `.c` files into `.o` and also build your `birthday` executable.
- The command `make clean` must clean up your directory by removing any `.o` files and built executables.
- There should be targets to make the `.o` object for each corresponding `.c` file in the system.

The file should have a comment header, should appropriately variables to avoid redundantly repeating names of common files, should properly express dependencies between files, and should only rebuild files as needed when a file is modified. For example, if the developer modifies `Date.cpp`, the `make` command should not recompile `birthday.cpp`.

All files in your program should produce no errors or warnings from `gcc -Wall`.

Style Guidelines:

Encapsulate your `Date` objects by making their methods, constructors, and operators `public` and their fields `private`. Avoid redundancy and repeated logic. Avoid unnecessary fields; use fields to store important object data but not temporary values only used in one method. If you add "helper" methods to your `Date` class, make them `private`.

On both parts of the assignment, you should follow general past style guidelines, such as avoiding redundancy and properly using indentation, names, types, and variables. Comment your code in your own words with a descriptive heading in each file, on top of each method (either at declaration or at definition), and on complex sections of your code.

For reference, our files have the following approximate sizes (lines / without-comments-or-blank-lines / with extra credit):

`Birthday.cpp` 75/45/80, `Date.h` 40/22/55, `Date.cpp` 100/45/200

Extra Credit:

1. For **+1** extra point, implement all five of the following overloaded operators in your `Date` class:

- `> (date), == (date), != (date), <= (date), >= (date)`
  Define overloaded operators to indicate whether the `Date` object is "greater than" (later than), "less than" (earlier than), or "equal to" (the same date as) another `Date`. `Date`s should be compared by reference as shown in class.

2. For **+1** extra point, implement both of the following overloaded operators in your `Date` class:

- `++, --`
  Define overloaded `++` and `--` operators to shift a `Date` object forward or backward in time by 1 day respectively. Note that these operators might cause the `Date` to wrap into the previous/next month or year. Each operator should return a reference to the `Date` itself. Also use these operators as appropriate in your `birthday` program. You should support both pre- and post-increment, such as `date++;` or `++date;` . These operators have strange headers; see the link about overloading `++` on the Links page. Note that post-increment returns a copy object.

3. For **+1** extra point, implement the following constructor in your `Date` class:

- `Date()`
  In this second, parameterless constructor, you should initialize a new `Date` object that represents *today*, the date that the program is being run. That is, if the program is run on June 4, 2009, you should initialize a new `Date` object whose year is 2009, month is 6, and day is 4. You can figure out what date today is by using the `ctime` library (see links on web site). This is the *only* place that you are allowed to use a date-related system library.

If you do this extra credit feature, you should also modify your `birthday` client program so that it no longer prompts for today's date, but rather figures it out for itself. For example:

```
Today is 2009/5/31
Info about 2009/5/31:
It is day #31 of 31 of the month
It is day #151 of 365 of the year
```

Note that once you aren't prompting for today's date, it's harder to test the program with different "today" dates. So you may want to do this last, and/or have a preprocessor flag for turning the feature off for testing.

4. For **+1** extra point, implement both of the following methods in your `Date` class:

- `daysInYear()`
  In this method you should return the number of days in the year of this `Date` object. Leap years have 366 days, and all other years have 365 days.
- `dayOfWeek()`
  In this method you should return a `string` representing the day of the week on which the `Date` falls, such as `"Sunday"` or `"Thursday"`. For example, if the object represents June 1, 2009, your method would return `"Monday"`. It may help you to know that January 1, 1753 was a Monday. Hint: Try using an auxiliary object.

If you do this extra credit extension, you should also modify your birthday client program so that it prints what day of the week today and the user's birthday fall on, as shown below. See the logs on the web site.

```
Info about 1979/9/19:
It is day #19 of 30 of the month
It is day #262 of 365 of the year
It is a Wednesday
```

5. For **+1** extra point, implement *exception handling*. In your `Date` class, if the client ever tries to create a `Date` object with an invalid state, or set an existing `Date` object to an invalid state, throw an exception string. An invalid date is one with a year before 1753, a month before 1 or after 12, or a day before 1 or after the number of days in its month. Modify `birthday` so that if the client types an invalid date, it will print the exception and exit. For example:

```
Please type your birthdate (y m d)? 1997 29 150
Error: month of 29 must be between 1 and 12
```