MCIS-4130
Homework Assignment 2

Purpose:

This assignment will illustrate how programs are built of multiple files in a programming environment.

This assignment will also explore ways of establishing encapsulation and data abstraction using multiple source files.

This assignment will also explore some C++ output formatting capability.

What to hand in:

Hand in listings of the program which must be captured in 3 or more files: a header file, a test program, and at least one implementation file. Also, hand in the output of a sample run (or runs) of the program. All of these must be formatted in a reasonable style. Choose sufficient test cases to ensure your functions work properly.

Also, hand in one alternate idea about how to represent a Date, and why it might be better in some applications.

Problems:

Dating Skills:

1. Create a header file (date.h) containing a proper definition for a Date. Be sure to put the appropriate standard `#ifndef/#define/#endif` directives around the header file.
2. Add functions for setting and printing a Date. These are further described in the Notes section below.
3. Create a test program (testdate.cpp) that uses the functions for setting and printing a Date. You should add tests one at a time, and ensure your implementation supports the tests after each addition.
4. Create an implementation for each of the functions in the header file (date.cpp). You may use your best software engineering practices to factor the algorithms in the best way. Note: You may ***not*** include this file in your test program—you must use the linker to resolve the functions.
5. Compile and run your program, capturing the output for submission.

- *Optional: How can errors in setting a date be dealt with?*

Notes:

The goal of the exercise is to think in object-oriented terms. Your Date *structure* serves the role of the *object* in Object oriented terms (and is declared in the `date.h` file):

```
struct Date
{
   int month;
   int day;
   int year;
}; /* One implementation (there are others) */
```

The *variables* of that type serve the roles of *instances* of the object:

```
Date d1, d2, d3; /* Mostly found in the test main program */
```

Thus, each of the functions should manipulate the instances of the object–they are the *messages* of the object-oriented paradigm. The message interface is defined in the `date.h` file:

```
void PrintDate  ( Date d);
Date CreateDate ( );
```

```
   Date SetDate    ( int y, int m, int d );
```

Note that this approach begins to isolate clients of your code from the internal details: where you might have had a piece of client code (in, say, `testmain.cpp`) like:

```
         Date d1;
         d1.month = 3;
         d1.day = 31;
         d1.year = 1995;
         print ( "%d/%d/%d\n", d1.month, d1.day, d1.year);
```

which is dependent on your representation. You will now have code in `testmain.c` like:

```
         Date d1;
         d1 = SetDate (1995, 3, 31);
         PrintDate (d1);
```

Functions become simpler and smaller. The function `DateSet`, for example, in `date.c`, may resemble:

```
   Date SetDate ( int y, int m, int d ) {
      Date date;
      date.day = d;
      date.month = m;
      date.year = y;
      return date;
   }
```

Time-oriented functions in the standard library are in the standard header <ctime>. The function *time_t time(time_t *t)* returns the number of seconds since 1 Jan 1970 GMT on Unix systems. The function *struct tm *localtime(time_t *t)* takes (the address of) this result and converts it into a structure with tm_month, tm_day, tm_year, etc. as components. The function *time_t mktime (struct tm*)* takes one of the time structures, completes it reasonably, and returns the equivalent seconds count.

A function to set the date to the current time may look as follows:

```
   Date CreateDate ( ) {
      Date d = { 1, 1, 1970};
      time_t t = time(0L);            /* Stores the current time in t */
      struct tm *tp;
      if (t == (time_t)(-1))
          return d;
      tp = localtime(&t);             /* Turns t into a struct tm */

      d.day   = tp->tm_mday;
      d.month = tp->tm_mon + 1;     /* Perversely, months are 0-11 */
      d.year  = tp->tm_year + 1900; /* struct tm uses 2-digit years */
      return d;
   }
```

Evaluation Criteria:

Code will be evaluated both for proper style and for correct content. Your code should be in 3 (or more) files: date.h, date.cpp, and testdate.cpp (date.cpp could be subdivided into many other files if you choose).

10	Reasonable answer for the Date alternative.
30	Interface file: proper definitions and content, date type, compliant set of operations
30	Implementation file: proper use of C++ for implementation
30	Test program: full test of interfaces, including boundary conditions. Inclusion of results. ***Note: If your program doesn't compile, hand in the error listing as the results!***