CS 271
Lab Assignment # 6

40 points total

There are quite a few files for this lab assignment. Zip them up and submit a zip file.

makefile
Time.h
Time.cpp
Date.h
Date.cpp
Calendar.cpp (this is the driver program)

Zip all 6 files together and submit one .zip file.

Programs must compile.
Programs that have syntax errors receive a grade of zero.

- Documentation and Style - This is a large part of the grade on this assignment. See the "Documentation and Style Guidelines" document in the Lecture Notes Module. The TA will be checking for:
  - Header and inline comments
  - Indentation
  - Spacing
  - Use of blank lines
- C++ input and output - You must use cin and cout. Use of scanf or printf is prohibited.
- Follow these conventions for identifiers:
  - function names - start with lowercase letter, then use camel case
  - variable names and data member names p- start with lowercase letter, then use camel case
  - class names - start with uppercase letter, then use camel case
  - constants - all uppercase letters
- Follow convention for naming accessors and mutators.
  - mutator - starts with lowercase "set" followed by, first letter capitalized, name of the data member
  - accessor - starts with lowercase "get" followed by, first letter capitalized, name of the data member
- Programs must declare ALL variables at the top of the function. C++: The only exception is a variable that is used to control a for loop.

```
int main ( void ) {

   // all variable declarations first

   // then executable statements
}
```

Avoid Magic Numbers

Programs cannot have any "magic numbers" in the executable statements. All numeric literals other than 0 or 1 must be declared as symbolic constants or as local constants with initial values.

- Creating a symbolic constant (pg 222 in the text).

```
#include <stdio.h>
#define SIZE 10
```

- Declaring a local constant using "const".

```
void checkHour(int x) {

   const int HOUR_LIMIT = 12;

   // executable statements begin here

   if (x <= HOUR_LIMIT)
       cout << "x is a valid hour" << endl;
}
```

Random Number Generation

We did this in C. Here's the C++ version of random number generation:

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main ( ) {

   // generate a random number between 1 and 10

   const int MAX = 10, num;
   srand(time(NULL));
   num = rand() % MAX + 1;
   cout << "num is " << num << endl;

}
```

Time.h

Create the class definition for class Time. This class represents a 24-hour time. Values may range from 00:00:00 to 23:59:59.

Remember that the class definition contains only data members and member function prototypes.

The data members are:

1) hour (unsigned int)
2) minute (unsigned int)
3) second (unsigned int)

The member functions are:

1) default constructor - initializes Time object to 00:00:00
2) constructor Time(int h, int m, int s) - initializes Time object using the parameters. This constructor **must** use the mutators to set values of the data members.
3) mutator for hour, mutator for minute, mutator for second (for invalid data, do not change the data member and do not print any error messages, do not throw an exception)
4) accessor for hour, accessor for minute, accessor for second
5) print ( ) - prints the time in 24-hour format `hh:mm:ss` (two digits each). Use the accessors to get the values of the data members.
6) print12 ( ) - prints the time in 12-hour format `hh:mm:ss` (two digits each) followed by one space, then AM or PM. Use the accessors to get the values of the data members.

Incremental Programming : You can compile a .h file. The compiler will check your syntax. Of course, the best thing is to proofread syntax as you type. But, just to make sure…compile the .h file before you go on to the next step.

Date.h

Create the class definition for class Date. This class represents a typical date with month, day and year. Values may range from 1/1/1980 to 12/31/2100.

Remember that the class definition contains only data members and member function prototypes.

The data members are:

1) month (unsigned int)
2) day (unsigned int)
3) year (unsigned int)

The member functions are:

1) default constructor - initializes Date object to 1/1/1980
2) constructor Date(int m, int d, int y) - initializes Date object using the parameters. This constructor **must** use the mutators to set values of the data members.
3) mutator for month, mutator for day, mutator for year (for invalid data, do not change the data member and do not print any error messages, do not throw an exception)
4) accessor for month, accessor for day, accessor for year
5) print ( ) - prints the Date in the format `mm/dd/yyyy` (two digits each for month and day, four digits for year). Use the accessors to get the values of the data members.

Incremental Programming : Compile Date.h before you go on to the next step.

Time.cpp

Write the function definitions for the 10 member functions of the Time class.

Remember that you cannot have any magic numbers in the executable portion of the code.

> **Incremental Programming : Compile Time.cpp before you go on to the next step.**

Date.cpp

Write the function definitions for the 9 member functions of the Date class.

Remember that you cannot have any magic numbers in the executable portion of the code.

> **Incremental Programming : Compile Date.cpp before you go on to the next step.**

Makefile

Yes, you need a makefile. If you haven't already created it, do that now. From here on… just type

```
make
```

to compile all files.

If the make command gives you errors (related to the commands you have in the makefile) or if it doesn't produce the executable you expect, debug the makefile before proceeding.

Calendar.cpp

Create a driver program with a main function. Use symbolic constants or "const" declarations here, also.

In the main function, create an array of 5 Time objects and set them to random times.

Print the array, one element per line, with 24 hour format.

Print the array again, one element per line, with 12 hour format.

Create an array of 5 Date objects. Use a loop to input the month, day and year for each Date. Then call the mutators to set those data members.

Print the elements of the array, one element per line.

Add calls to the other member functions as needed to test them thoroughly.

Zip all files into one zip file and submit the .zip file.