ECS 40 **Program # 3 (25 points, 1 hour)** **Spring 2017**

New concepts: new, delete, classes, constructors, destructors, cin, cout, ifstream, ostream.
Executable name: calendar.out Due: Wednesday, April 26th at 11:59pm in p3 of cs40 using handin.
File names (exactly): authors.csv, main.cpp, calendar.cpp, day.cpp, appt.cpp, time.cpp, calendar.h, day.h, appt.h, time.h,, Makefile, and Makefile-Debug.mk.

This is an extension of program #1. For this assignment, you will be converting your program #1 C code to C++ code using the NetBeans Integrated Development Environment. You will convert all of your structs to classes, use "const" appropriately, update the I/O and memory functions to that of C++, and take care of the two time format issues.

Netbeans is installed on the CSIF computers under Applications->Programming. You can download your own free copy from netbeans.org. From the Download page choose the "C/C++" version. There are many tutorials available at http://www.netbeans.org/kb/trails/cnd.html. It is up to you to choose what tutorials you need. To match the indentation requirement of 2 spaces, within NetBeans you should go to Tools->Options->Editor->Formatting and set the "Number of Spaces per Indent" and "Tab Size" to 2.

To prove that you used NetBeans, we want you to handin Makefile-Debug.mk for your house project. Makefile-Debug.mk is located in the nbproject directory of the directory NetBeans will create for your house project. Even though NetBeans creates its own Makefile for your project, we will be testing with the Makefile provided in ~davis/40/p3. You should make sure that your program also compiles without warnings using that Makefile!!!

(30 minutes) Assuming you are using my programming code as a basis here are the steps to convert each typedef struct to a class. You should do one class at a time, and then compile to make sure it works. Work from Time (5 minutes) to Appointment (8 minutes) to Day (8 minutes) to Calendar (5 minutes).

Here is the three step pattern for each typedef struct to class conversion with Time as an example. Your program should compile and run properly each time you finish this set of instructions for a class. Once you get the hang of this, you will find paste, and global search and replace will speed the transitions.

1. Create the class within the header file. This based on the assumption that your typedef statement above the prototypes.
   1.1. Replace the "typedef" line with a class heading, e.g. "class Time"
   1.2. Delete the "}" at the end of the typedef struct statement, and add "}; // class Time" above the #endif
   1.3. Above the first prototype place "public:" at the left margin. Indent all function prototypes 2 spaces from the left margin.
   1.4. Remove the first Struct* variable from all of the function prototypes, e.g. "bool lessThan(Time *time1, Time *time2);" would become "bool lessThan(Time *time2);" If the only parameter is a struct pointer, then there will be no parameters, e.g. "void read();"
2. Update the classes .cpp file
   2.1. Update each function header
      2.1.1. As with 1.4, remove the first Struct* variable from all the function prototypes.
      2.1.2. Between the return type and the function name, insert the name of the class followed by the scope operator, "::", e.g. void Time::read()
   2.2. Eliminate all mention class' pointer parameter in the function body, e.g. "time1->hour" becomes simply "hour". Do not eliminate the class' pointer for those parameters that continue to exist, e.g. "time2->hour"
3. Update the calls to the class functions in the file that calls it, e.g. appointment.cpp for Time.
   3.1. Move the struct pointer, from being a parameter to before the function call using the period operator to connect the object to its function, e.g.
   `"read(&appointment->startTime);"` would become `"appointment->startTime.read();"`
      3.1.1. You will often use "->" in day.cpp for the Appointment class.
   3.2. You will need to be careful that you don't have two variables with the same name in day.cpp.

(5 minutes) Once the code works properly again, you need to add "const" to the parameters and functions where appropriate. const parameters are those that are not changed within the function, e.g. "bool Time::equal(const Time *time2)". Parameters that are passed by value should never be labelled const even if they are not changed. Const functions do not change the class object, e.g. "bool Time::equal(const Time *time2) const", and "Time::print() const

(20 minutes) The third stage is replacing the I/O and memory functions. The I/O functions of stdio.h are found in <iostream> for cin, cout, and getline; <fstream> for ifstream(); and <iomanip> for setw() and setfill(). The memory management functions of C that are found in <stdlib.h>, are an integral part of C++ and require no #include. Most of this is one-to-one transformations, with only printf() requiring a radically different syntax. I advise that you convert one file at a time by replace #include <stdio.h> with the appropriate #include(s). To aid debugging, you should list all system #includes above any local #includes. After the last of the #includes, add "using namespace std;" Here are the conversions.

C	C++
`scanf("%s%d", subject, &num);`	`cin >> subject >> num;`
`fgets(line, 80, stdin);`	`cin.getline(line, 80);`
`FILE *fp = fopen("stuff.txt, "r");`	`ifstream inf("stuff.txt")`
`fgets(line, 80, fp);`	`inf.getline(line, 80);`
`fclose(fp);`	`inf.close();`
`printf("Your choice >> ");`	`cout << "Your choice >> ";`
`printf("%08d%-7dJ\n", a, b);`	`cout << setw(8) << setfill('0') << a << left << setw(7) << setfill(' ') << b << 'J' << endl;`
`int *ptr = (int*) malloc(sizeof(int))`	`int *ptr = new int;`
`int *array = (int*) malloc(sizeof(int) * 20);`	`int *array = new int[20];`
`free(ptr);`	`delete ptr;`
`free(array);`	`delete [] array;`

(5 minutes) The last stage is changing the format of the times. Change so any time in the 12:00am hour is printed properly, e.g. 12:10am is 00:10. Change so any time in the 12:00 pm hour is printed properly, e.g. 12:30pm is 12:30.

Further specifications:

1. The output of your program must be identical to that of mine, which is identical to program #1, except for changes to the time format.
2. All implementation code for a class must be in its .cpp file.
3. You may not #include stdio.h, stdlib.h, nor cstdio.
4. If #include <stdio.h>, or #include <cstdio>, or "struct" is found in any file, then you will receive a **zero** for the assignment.
5. You should #include <cstring> to access strtok(), and <cstdlib> for atoi().
6. **const** must be used wherever possible in the function headers.
7. If you have any of the proscribed header files, or a struct in any of your files you will receive a zero for the assignment.