ECS 40 **Program # 5 (50 points, 2 hours)** **Spring 2017**

dDue: Wednesday, May 10th at 11:59 pm in p5 of cs40 using handin.
New concepts: overloaded operators, linked lists, static function variables.
File names (exactly): authors.csv, main.cpp, calendar.cpp, calendar.h, day.cpp, day.h, appt.cpp, appt.h, time.cpp, time.h, DayOfWeek.cpp, DayOfWeek.h, linkedlist.cpp, linkedlist.h, and Makefile.

This is an extension of program #4. You are to modify program #4 so that it uses overloaded operators where appropriate, and replaces the array of Appointment pointers in the Day class with a linked list of Appointments (not pointers). The actual operation of the program will not change.

You will find my executable, DOW.dat, and appts.csv in ~ssdavis/40/p5. You are encouraged to use my source code from program #4 as the basis for your program #5

Replacing functions with overloaded operators (1 hour)

As usual, your code should compile without warnings and run perfectly after each major step.

1. (12 minutes) Replace all the equal(), and lessThan() methods with their equivalent overloaded operators, and update the calls to the methods.
   1.1. You should be passing const references instead of pointers in all cases except when passing const char*.
   1.2. So that you don't make the same mistake multiple times, start with changing the Time code, then compile and run, then Appointment, then compile and run, then Day, and finally compile and run.
2. (12 minutes) Replace all methods named print() with overloaded insertion operators, <<, and update the calls to the methods.
   2.1. These methods are friends and each has two parameters. Both parameters are references. Now you should be able to see an advantage of having the reference data type.
   2.2. You should use the ostream parameter identifier in these functions, and not cout. This offers more flexibility for later use to insert in a file if we wish.
   2.3. The Time and Appointments methods must contain only two statements, i.e. only two semicolons !
   2.4. Again, do one class at a time, and then compile and run to learn from your mistakes early.
3. (35 minutes) Replace all methods named read() and readFile() with overloaded extraction operators, and update the calls to the methods.
   3.1. main() will now open the file named in the first command line parameter and use the extraction operator to read the file into the calendar.
   3.2. You may not use strtok() anywhere in these methods. You can achieve the same result by using getline() and altering the delimiter specified as its third parameter. This makes this task tricky, and you would be well advised to work down from main() instead of up from Time.
   3.3. With each class, you can comment out the call to the next read(), and test dateSearch().
   3.4. To move to the next line when only part of a line will have been read, add a temporary getline(…) at the end of the Calendar method's loop. When done with the Day extraction operator, remove the temporary getline.
   3.5. When testing the Appointment method, before implementing the Time method, the times will be garbage, and the location will include the times in the file!
   3.6. To make the DayOfWeek extraction operator work, you will need to have Day::subjectSearch() open DOW.dat, and create a DayOfWeek constructor that takes the three date values as its parameters. Though this is the trickiest conversion, by now you should be able to figure out what aspects of the code you need to change to make this work. Please try to take this as a challenge, and don't resort to Piazza.

Linked list of Appointment* (1 hour coding)

1. Class ListNode should be declared in linkedlist.h, and implemented in linkedlist.cpp.
   1.1. A ListNode contains an Appointment*, and a next pointer.
   1.2. The only two methods of this class, are a constructor and a destructor, and they are not public.
2. Class LinkedList is a singly linked list sorted by the Appointment <.
   2.1. Class LinkedList is a friend of class ListNode.
   2.2. The class contains only a ListNode* head. Note that NULL is defined in <iostream>
   2.3. You must have a constructor and destructor for the class.

2.4. Write an overloaded assignment operator. This is the trickiest code to write of the whole assignment. After deleting the old list, use a tail pointer that always points at the last ListNode in the "this" list to append to the newly created list.
2.5. Replace all the code dealing with appts in Day::operator= with a single call to the overloaded assignment operator.
2.6. You will have two overloaded[] operators—one const, and one not. The const version should return a const Apointment*, and the non-const version should return an Appointment*&.
2.6.1. If the index requested is beyond the end of the list, append a new ListNode to the list. This may only be done in the non-const version. Why?
2.7. Replace the Appointments *appts[8] declaration in day.h with a LinkedList object with the same name.
2.7.1. Since the LinkedList has its own destructor, you can completely remove the Day destructor!
2.8. Surprisingly, at this point your code should run perfectly!

3. Write an operator+= to insert an Appointment in the list based on the Appointment < operator. This operator may not use the overloaded[] operator.
3.1. Replace the calls to Day::insert() with a call that uses the += operator. Increment the apptCount in the functions that used to call insert(). Remove the insert() method from the Day class.
3.2. Your code should again run perfectly.
4. Write an ostream insertion operator for the LinkedList class that prints the whole list. Call this from the Date insertion operator. This must be a friend of the ListNode class too.
5. Write a find() method for the LinkedList class that takes a char* subject as its sole parameter, returns a pointer to an Appointment.
5.1. Like strtok(), the function returns NULL when there are no more subjects to be found in the linked list.
5.2. This function will be like strtok() in that it will remember where it was in the linked list from one call to the next by using a static function ListNode* variable. Note that this is a static function variable, not a static LinkedList class variable.
5.3. Designing this function to work properly will take a little thought., but involves only nine lines of actual code (ignoring spaces). Please refrain from using Piazza. Hint: you will need another local variable other than the static ListNode*.
5.4. This allows a simple <u>while</u> loop to easily search an entire LinkedList without knowing how many Appointments are in the linked list. Replace the for loop in Day::subjectSearch() with a while loop calling find().
5.5. Since the loop in Day::subject was the last place that apptCount was used, eliminate apptCount from the entire Day class.