ECS 40 Program #1 (50 points, 2.5 hours) Spring 2017

New concepts: strtok.. Executable name: calendar.out
File names: main.cpp, calendar.cpp, calender.h,, appt.cpp, appt.h, day.cpp, day.h, time.cpp, time.h, Makefile, authors.csv
Due: Wednesday, April 19th at 11:59pm in p1 of cs40 using handin.

authors.csv should be in the following format (-5 points for incorrect format):
First line: <e-mail of first partner> <comma> <family name of first partner> <comma> <given name of first partner>
Second line (if needed): <e-mail of second partner> <comma> <family name of second partner> <comma> <given name of second partner>

For example:

```
hpotter@ucdavis.edu,Potter,Harry
fflintstone@ucdavis.edu,Flintstone,Fred
```

Use handin to turn in just your files to cs40. For this program you would type:

```
handin cs40 p1 authors.csv main.cpp calendar.cpp calendar.h appt.cpp appt.h \
      day.cpp day.h time.cpp time.h Makefile
```

This will be the first of a series of programming assignments to create an appointment calendar program. For this program, you will write a program to read in the information from the file appts.csv, and search for appointments based on date or subject. The information should be stored in an array named days.

The style of your program must obey the programming style specified in the style handout. Remember the body of any function cannot be more than 30 lines long. The output of your program must be identical to that of mine. Your program should compile with no warnings. Though the name of the file ends with .cpp, you may write your program in either C or C++. In either case, you will use the g++ compiler because it allows // comments. You will find my executable, and appts.csv in ~ssdavis/40/p1.

Specifications:

1. appts.csv is a "comma separated value" file of indeterminate size.
   1.1. The first line contains the title of each column.
   1.2. Each succeeding line contains the information for an appointment.
   1.3. You should use strtok() available in string.h to parse each line. I also found atoi(), strlen(), strcpy, and strchr() useful.
2. main.cpp may only contain two functions.
   2.1. getChoice() prints the menu, range checks the user's choice, and returns a valid choice to main().
   2.2. main() declares a Calendar named "calendar", calls create(), calls readFile(), and then has a loop based on the user's choice, and calls either dateSearch(), or subjectSearch(). When the user is done, main() should call destroy(). The address of the Calendar is the only parameter passed to the four functions.
3. calendar.h has the declaration of the typedef struct Calendar, and the prototypes for the functions that have a Calendar* as their parameters.
   3.1. The Calendar struct contains a Day pointer named "days", an integer named "size", and an integer named "count".
4. calendar.cpp contains the implementations for all of the functions that have a Calendar pointer as a parameter.
   4.1. create() dynamically allocates 30 Days in days, and initializes size to 30, and count to zero.
   4.2. readFile() reads appts.csv line by line into the days array.
      4.2.1. The function reads an <u>entire</u> line from the file, and then uses strtok() to parse the three parts of the date. It then calls create() with the address of a local Day variable (named dayTemp), the day, the month, and the year as parameters.
      4.2.2. The function then tries to find a Day with the corresponding date in the array by passing the address of the dayTemp, and the address of an existing Day to equal() of day.h. If a day with the proper date is found then go to 4.2.5

4.2.3. The days array is a dynamically allocated of typedef structs Day, with an initial size of 30. If the number of days encountered (stored in count) is equal to the current size of the array (stored in size), then the array must be doubled in size. To keep the length of readFile() under 36 lines, you will need to add a resize() function that takes the Calender pointer as its sole parameter. (Described in 4.3) Note that it resize() will be called more than once while the program reads large files!

4.2.4. Now that there is room for at least one more Day, rely on passing the address of dayTemp, and the address of each Day in days to lessThan() to determine where the new Day should be inserted. By starting at the end of the array and copying (by simply using the assignment operator) each larger Day into the next larger index position you can make room into which to copy dayTemp.

4.2.5. After finding, or creating a Day with the correct date, the function passes the address of the Day struct to the read() function declared in day.h.

4.2.6. It is good practice to close the file at the end of the function.

4.3. resize() dynamically allocates an array of Days that is twice as large as the current array, copies the Days in the original array into the new array simply using the assignment operator, deallocates the original array, and then copies the new address into days.

4.4. dateSearch() reads and range checks for a valid date, then create() a Day from the date, and repeatedly uses equal() for every Day in the days array to find the corresponding day. Because the range checking is complex and long, you will need to write a separate getDate() function (described in 4.5). If it finds a Day with the correct date, it passes the address of the Day to print(), and then returns.

4.5. getDate() reads and range checks for a valid date.

4.6. subjectSearch() accepts subjects up to 79 chars in length. It then repeatedly calls the subjectSearch() function in day.* that takes the address of a Day, and the entered subject as its only parameters.

4.7. destroy() repeatedly passes the address of each Day to a function called destroy(), and afterward deallocates the days array.

5. day.h has the declaration of the typedef struct Day, and the prototypes of for the functions that have a Day pointer as a parameter.

5.1. The typedef struct Day contains a short named "day", a short named "month", a short named “year”, and an array named "appts" of eight pointers to typedef struct Appointment, and a short named "apptCount".

6. day.cpp contains the implementations for all of the functions that have a Day pointer as a parameter.

6.1. create() stores the values of the passed parameters in their corresponding variables, and sets apptCount to zero.

6.2. equal() returns true if the date of the two Days are the same.

6.3. lessThan() returns true if the date of the first Day is less than the date of the second Day.

6.4. read() dynamically allocates an Appointment, passes that address to the read() function declared in appt.h, and then inserts the address into the properly sorted place in the appts array by relying on the bool returned when passing the address of the appointment and the address of an appointment already in the array to lessThan().

6.5. print() prints the heading, and then simply passes the address of each appointment in appts to the print() function declared in appt.h.

6.6. subjectSearch() loops through the appts array, passes the address of the Appointment and the subject to equal(). If equal() returns true, then subjectSearch() passes the address of the Appointment to print().

6.7. destroy() passes the address of each Appointment in appts to destroy() of appt.h, and afterward deallocates each of the Appointments in appts.

7. appt.h has the declaration of the typedef struct Appointment, and the prototypes of for the functions that have an Appointment pointer as a parameter.

7.1. Typedef struct Appointment contains: "startTime" and "endTime" of typedef struct Time, a dynamically allocated array of char named "subject", and a dynamically allocated array of char named "location".

8. appt.cpp contains the implementations for all of the functions that have an Appointment pointer as a parameter.

8.1. read() uses strtok() to isolate the subject, then copies it into subject (after allocating for its size), then passes the address of startTime to read() of time.h, then calls read() with the address of endTime, and finally uses strtok() one more time and copies the location in the line into location (after allocating for its size).

8.2. print() passes the address of startTime to print(), then the address of endTime to print(), and then prints the subject and location.

8.3. lessThan() returns the value returned when it passes the addresses of the startTimes of the two appointments to the lessThan() function of time.h.

8.4. destroy() deallocates the subject and location.

9. time.h has the declaration of the typedef struct Time, and the prototypes of for the functions that have a Time pointer as a parameter.

9.1. Typedef struct Time uses contains a short named "hour" (0 – 23), and a short named "minute".

10. time.cpp contains the implementations for all of the functions that have a Time pointer as a parameter.
    10.1. read() uses strtok() to parse the time stored in the char array, determines the corresponding hour and minute.
    10.2. print() prints the time in the proper format.
    10.3. lessThan() returns true if the time specified in the first parameter is less than the time specified in the second parameter.
11. Makefile must use g++, and every compilation and linking line must have -g, –Wall, and –ansi options. You must have a "clean" rule that explicitly removes every file created by a call to make. Thus, you may not have "rm *.o"
    11.1. You will lose one point for each warning.
12. Hints and suggestions:
    12.1. Since gets() will cause a warning, use fgets(s, 80, stdin) to read entire lines from the user. Beware that the string in the array is terminated with a '\n' when fgets() is used. You will usually have to get rid of that trailing '\n'. You should also use fgets() after reading the choice to read the remaining '\n' from the keyboard buffer.
    12.2. I used strtok() to parse the date entered by the user. Be careful to allow for the user just pressing the Enter key!
    12.3. For formatting the output, remember that printf() has formatting options that can change the padding character in your output, and can determine the justification for variable.
    12.4. I alphabetize my functions in both the header and .cpp files. This often makes it quicker to move to a desired function.
    12.5. Do not try to write this whole program all at once, and then debug it!!! By writing it in stages it is easier to debug, and you will learn from your mistakes.
    12.6. Here is a suggested order of development. You program should compile without warnings, and run (albeit with only partial functionality) after you complete each step.
        1. Write main(), and getChoice(). Write the Calendar struct, and stubs for functions in day.* that are called from main(). Stubs are simply empty functions. Write the Day struct in day.h. Write the Appointment struct in appt.h. Write the Time struct in time.h. Write a Makefile with only rules for calendar.out, main.o, day.o, and clean. You can now test your getChoice() function. You need only range check for incorrect integers in getChoice().
        2. Write create(), readFile(), and resize() in calendar.*. Write create(), equal(), and lessThan() in day.*. Create a stub for read() in day.*. There is no need to change the Makefile. You need not worry about invalid, nor a missing appts.csv. Your program should run the same as in step 1.
        3. It would be nice to know if readFile() is working properly. Write dateSearch() in calendar.cpp, and print() in day.*, and a stub for print() in appt.*. You will need to use strtok(), and atoi(). Be careful to deal with strtok() returning NULL. Add a rule for appt.o to the Makefile. Now you can test whether your program really reads the file, and handles invalid dates by searching for a variety of dates. You may assume all months have 31 days, and only the years 2000 to 2017 are valid. Beware the user can enter anything under 80 chars.
        4. Write read() for day.cpp. Write stubs for read(), and lessThan() in appt.*. You will need to add a "return true;" statement in lessThan() to avoid a warning. This should run the same as the previous step.
        5. Write read(), print(), and lessThan() for appt.*. Write a stubs for read(), print(), and lessThan() for time.*. Add a rule for time.o to the Makefile. When you search for a date now you should see a listing for appointment(s). Since you haven't implemented the read() for time.*, the listing for each appointment's location will include the times.
        6. Write the read(), print(), and lessThan() in time.cpp. I used strchr() as well as strtok() in read(). Searching for a date should now produce the proper result.
        7. Write subjectSearch() in calendar.cpp. Write subjectSearch() in day.*. Write equal() in appt.*. The program should now output correctly.
        8. Add the destroy() functions to calendar.*, day.*, and appt.*. Congratualtions, you are done!

```
[ssdavis@lect1 p1]$ cat appts.csv
Date,Subject,Start Time,End Time,Location
10/7/2003,ECS 30-B,1:40:00 PM,3:00:00 PM,3 Temp
10/9/2003,ECS 30-B,1:40:00 PM,3:00:00 PM,3 Temp
// Edited by Sean
12/2/2003,ECS 30-B,1:40:00 PM,3:00:00 PM,3 Temp
12/4/2003,ECS 30-B,1:40:00 PM,3:00:00 PM,3 Temp
12/13/2003,ECS 30-B Final,8:00:00 AM,10:00:00 AM,3 Temp
10/6/2003,ECS 30-C,11:00:00 AM,11:50:00 AM,3 Temp
10/8/2003,ECS 30-C,11:00:00 AM,11:50:00 AM,3 Temp
// Edited by Sean
12/5/2003,ECS 30-C,11:00:00 AM,11:50:00 AM,3 Temp
12/9/2003,ECS 30-C Final,1:30:00 PM,3:30:00 PM,3 Temp
10/6/2003,ECS 30-D,1:10:00 PM,2:00:00 PM,126 Wellman
10/8/2003,ECS 30-D,1:10:00 PM,2:00:00 PM,126 Wellman
//Edited by Sean
12/5/2003,ECS 30-D,1:10:00 PM,2:00:00 PM,126 Wellman
12/12/2003,ECS 30-D Final,10:30:00 AM,12:30:00 PM,26
Wellman
10/15/2003,ECS 40-A,4:10:00 PM,5:00:00 PM,66 Roessler
10/17/2003,ECS 40-A,4:10:00 PM,5:00:00 PM,66 Roessler
10/20/2003,ECS 40-A,4:10:00 PM,5:00:00 PM,66 Roessler
//Edited by Sean
12/3/2003,ECS 40-A,4:10:00 PM,5:00:00 PM,66 Roessler
12/5/2003,ECS 40-A,4:10:00 PM,5:00:00 PM,66 Roessler
12/8/2003,ECS 40-A Final,4:00:00 PM,6:00:00 PM,66
Roessler
10/1/2003,Field Study 480,5:00:00 PM,7:00:00 PM,CSUS
9/26/2003,Group,9:00:00 AM,10:30:00 AM,SADVC Shelter
(Davis)
9/26/2003,Group,11:00:00 AM,12:30:00 PM,SADVC (Woodland)
10/3/2003,Group,9:00:00 AM,10:30:00 AM,SADVC Shelter
(Davis)
10/3/2003,Group,11:00:00 AM,12:30:00 PM,SADVC (Woodland)
9/30/2003,Group Supervision,11:00:00 AM,1:00:00 PM,SADVC
(Woodland)
10/1/2003,Individual Supervision,8:00:00 AM,9:00:00
AM,SADVC (Woodland)
10/7/2003,Office Hours,3:15:00 PM,5:00:00 PM,3052 Kemper
10/9/2003,Office Hours,3:15:00 PM,5:00:00 PM,3052 Kemper
10/13/2003,Office Hours,2:15:00 PM,5:00:00 PM,3052
Kemper
10/14/2003,Office Hours,3:15:00 PM,5:00:00 PM,3052
Kemper
// Edited by Sean
12/5/2003,Office Hours,2:15:00 PM,4:00:00 PM,3052 Kemper
12/8/2003,Office Hours,9:00:00 AM,12:00:00 PM,3052
Kemper
12/8/2003,Office Hours,1:00:00 PM,3:00:00 PM,3052 Kemper
12/9/2003,Office Hours,9:00:00 AM,12:00:00 PM,3052
Kemper
12/11/2003,Office Hours,1:00:00 PM,5:00:00 PM,3052
Kemper
12/12/2003,Office Hours,1:30:00 PM,5:00:00 PM,3052
Kemper
9/30/2003,Orals,2:00:00 PM,5:00:00 PM,CSUS
10/2/2003,Play Therapy,4:00:00 PM,7:00:00 PM,CSUS
9/29/2003,Practicum,4:00:00 PM,8:00:00 PM,CSUS
9/26/2003,SADVC,1:00:00 PM,6:00:00 PM,SADVC (Woodland)
10/1/2003,SADVC,12:00:00 AM,12:00:00 AM,Woodland
10/3/2003,SADVC,1:00:00 PM,6:00:00 PM,SADVC (Woodland)
 [ssdavis@lect1 p1]$ calendar.out
Calendar Menu
0. Done
1. Search for date.
2. Search for subject.

Your choice >> 1

Please enter the month, day, and year (mm/dd/yyyy)
>> 13/12/2012
13/12/2012 is not a valid date.
Please try again.

Please enter the month, day, and year (mm/dd/yyyy)
>> 9/26/2003
Start End   Subject      Location
09:00 10:30 Group        SADVC Shelter (Davis)
11:00 24:30 Group        SADVC (Woodland)
13:00 18:00 SADVC        SADVC (Woodland)

Calendar Menu
0. Done
1. Search for date.
2. Search for subject.

Your choice >> 1

Please enter the month, day, and year
(mm/dd/yyyy) >> 10/8/2003
Start End   Subject      Location
11:00 11:50 ECS 30-C     3 Temp
13:10 14:00 ECS 30-D     126 Wellman

Calendar Menu
0. Done
1. Search for date.
2. Search for subject.

Your choice >> 2

[ssdavis@lect1 p1]$ calendar.out
Calendar Menu
0. Done
1. Search for date.
2. Search for subject.

Your choice >> 2
Please enter the subject >> SADVC
Date       Start End   Subject      Location
 9/26/2003 13:00 18:00 SADVC        SADVC (Woodland)
10/ 1/2003 12:00 12:00 SADVC        Woodland
10/ 3/2003 13:00 18:00 SADVC        SADVC (Woodland)

Calendar Menu
0. Done
1. Search for date.
2. Search for subject.

Your choice >> 2
Please enter the subject >> Orals
Date       Start End   Subject      Location
 9/30/2003 14:00 17:00 Orals        CSUS

Calendar Menu
0. Done
1. Search for date.
2. Search for subject.

Your choice >> 2
Please enter the subject >> Wombat
Date       Start End   Subject      Location

Calendar Menu
0. Done
1. Search for date.
2. Search for subject.

Your choice >> 0
[ssdavis@lect1 p1]$
```