COP 3502 Birthday Program - Suggested Functional Breakdown

Here are some hints about the general design of the birthday program:

1) No need for too much dynamic memory allocation. My recommendation is to make a statically allocated array of pointers (to struct student). Make this array size 1000 since you'll never have more than 1000 students. For each student, you'll dynamically allocate the appropriate pointer to point to a single struct. The reason for this choice is so swapping items in the array is simple and not error prone, since you'll just be swapping pointers and not the structs themselves.

2) Hard code constant arrays to store the names of the months, as well as the number of days that have elapsed before the beginning of a month. Here is one example, for a regular year:

```
const int noleap[] = {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334};
```

3) The sorting in this assignment is 100% independent of the rest of it. So, write appropriate functions for the sorting that have nothing else to do with the problem at hand. Test these functions separately as well.

4) Write a function that compare that takes in two pointers to student structs, say ptrFirst, and ptrSecond. Have the compare function return a negative integer if the struct pointed to by ptrFirst should come before the one pointed to by ptrSecond, 0 if the structs being pointed to are equal, and a positive integer if the struct pointed to by ptrFirst comes AFTER the one pointed to by ptrSecond, based on the program description. This function should be called from your sort. Your sort should look very, very similar to what is posted on line. The reason we want to write this function is so that we don't have to change the sort function too much.

<u>Suggested Functional Breakdown</u>

Here is a suggested breakdown of functions, each of which can be individually tested:

1) Write a function that reads in the data into the array of pointers to struct.

2) Write a function that takes in a month as a string and returns the appropriate month. This function should be short and use the constant array mentioned previously.

3) Write a function that calculates the "day number" for a person's birthday. The day number is what day in the year that day is. This calculation is dependent upon whether or not someone in the input case has a leap year birthday.

4) Write a function that takes in a name and returns the index at which the student with that name is found.

5) Write a function that takes in a query and returns its solution (or processes the query.)

6) Write another function that reads in all of the queries for a case and processes all of them. This function should call the function in #5 inside of a loop.

7) Write a quick sort function - this should be nearly identical to what was shown in class.

8) Write a swap function which simply swaps to pointers in the array of pointers.

9) Write the previously mentioned compare function which takes in two pointers to structs and returns a negative integer, 0 or a positive integer depending on the relative order of the two structs, as determined by the problem statement.

10) Write the partition function. This will be similar to the one shown in class, but different in that it will call the compare function mentioned in #9 instead of using > or <.

<u>For the merge sort version, use the following functions:</u>

7) merge sort function

8) swap function

9) compare function - should be identical to the one mentioned before, as this portion is independent of the sorting algorithm used.

10) merge function - this one will call the compare function