CS270 Recitation 4
“C Structures”

DUE FRIDAY 9/16/2016 AT 11:59 PM (NO LATE SUBMISSIONS)

Goals

1. To write a C program essentially from scratch.
2. To learn how to allocate and access C structs, and pass them around as parameters.
3. To learn to use dynamic allocation in C.

The Assignment

Make a subdirectory called R4 for the recitation, all files should reside in this subdirectory. Copy the following files to the R4 directory:

https://www.cs.colostate.edu/~cs270/.Fall16/recitations/R4/src/struct.c
https://www.cs.colostate.edu/~cs270/.Fall16/recitations/R4/src/struct.h
https://www.cs.colostate.edu/~cs270/.Fall16/recitations/R4/src/main.c
https://www.cs.colostate.edu/~cs270/.Fall16/recitations/R4/src/Makefile

1) Extend the C *struct* called Student in the header file to add a last name, average homework score, average lab score, a midterm score, and a final exam score. All scores are of type **integer**, and all names are statically allocated **character** arrays (C strings) with 80 characters.

2) In main.c, declare (but don’t initialize) a global pointer to a Student struct, outside of any functions. This is meant to point to an array of students (but not yet, wait till step 4).

3) In the main entry point, print a prompt for the number of students and read an integer from the user. It should look like this:

```
Enter the number of students: 3
```

4) In the main entry point, **dynamically** allocate an array of Student structs based on the number entered. Make the global variable declared in step 2 point to this array. Add a statement to free the array at the end of the main entry point (before returning 0).

5) Implement the following function in struct.c (it is already declared in struct.h):

```
void inputScores(Student *student);
```

The inputScores function should ask the user for the names and all scores for a single student, and it should store them in the structure pointed to by the argument. You don’t need to handle spaces in the names. Here is a sample output of running the inputScores function (your prompts need to match):

```
Enter the first name: Andres
Enter the last name: Calderon
Enter the average homework score: 90
Enter the average lab score: 95
Enter the midterm score: 92
Enter the final exam score: 94
```

6) Implement the following function in struct.c (it is already declared in struct.h):

```
void outputScores(Student student);
```

The outputScores function should print all names and scores for a single student. Here is a sample output of running the function (your prompts need to match, ignore the last two lines of output until step 11):

```
First name: Andres
Last name: Calderon
Average homework score: 90
Average lab score: 95
Midterm score: 92
Final exam score: 94
Total points: 92.60
Letter grade: A
```

Note that <u>there is</u> a newline after the last line.

7) Add a loop to the main entry point to call the inputScores and outputScores function for each student in the array (in that order).

8) Build the program by typing the **make** command, and iterate until the program builds without warnings and you can input and output student records. Hint: you can hit Ctrl-C so that you don't have to enter more than a couple of student records.

9) Add the totalPoints (**float**) and letterGrade (**char**) fields to the Student structure in struct.h. Rebuild the program.

10) Add a declaration and implementation of the following function (the declaration goes in the .h header file and the implementation goes in the .c file):

```
void calculateScores(Student *student);
```

The calculateScores function will compute totalPoints and letterGrade for a single student. To calculate the totalPoints, use the following formula:

*(homework average * 0.30) + (lab average * 0.20) + (midterm score * 0.20) + (final score * 0.30)*

To calculate the letterGrade, use the following rules:

If 90.0 ≤ totalPoints, then letterGrade is A
If 80.0 ≤ totalPoints < 90.0, then letterGrade is B
If 70.0 ≤ totalPoints < 80.0, then letterGrade is C
If 60.0 ≤ totalPoints < 70.0, then letterGrade is D
If totalPoints < 60.0, then letterGrade is F

11) Add print statements for totalPoints (with 2 digits after the decimal point) and letterGrade to the outputScores function (see the sample run in step 13).

12) Call the calculateScores function for each student record. Where do you think is a good place to call it?

13) Rebuild and test your modified program. Here is a complete sample run:

```
Enter the number of students: 2
Enter the first name: John
Enter the last name: Doe
Enter the average homework score: 87
Enter the average lab score: 33
Enter the midterm score: 93
Enter the final exam score: 67
First name: John
Last name: Doe
Average homework score: 87
Average lab score: 33
Midterm score: 93
Final exam score: 67
```

```
Total points: 71.40
Letter grade: C
Enter the first name: Andres
Enter the last name: Calderon
Enter the average homework score: 98
Enter the average lab score: 95
Enter the midterm score: 56
Enter the final exam score: 77
First name: Andres
Last name: Calderon
Average homework score: 98
Average lab score: 95
Midterm score: 56
Final exam score: 77
Total points: 82.70
Letter grade: B
```

Note that there is a newline after the last line.

15) Run your program through **Valgrind** using the command below. Valgrind is a program used to diagnose memory problems in other programs (e.g., memory leaks and invalid memory reads/writes).

```
valgrind --leak-check=yes ./R4
```

You should see something like this at the end:

```
==12252== All heap blocks were freed -- no leaks are possible
==12252==
==12252== For counts of detected and suppressed errors, rerun with: -v
==12252== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Please ask your TA for assistance in interpreting Valgrind's output.

16) Type the following command to create a package of your source code and note how the package is built.

```
make package
```

17) Submit the resulting **R4.tar** to the Checkin tab. All grading is preliminary (there are no hidden test cases). **The grade that you get in the Checkin tab will be your grade for this recitation**.