CS261: HOMEWORK 1
Due 04/12/2017 by noon (12pm time on the server)

Submit a zipped folder HW1.zip with four *.c files via the TEACH website:

`https://secure.engr.oregonstate.edu:8000/teach.php?type=want_auth`

General Instructions

This assignment consists of four programs. Each program should be implemented in a separate .c file (see Section 5). A skeleton code for these four files is already provided on the class website, and your task will be to insert your code in certain places. For compiling your code, we also provide a makefile on the class website. Download these four files with the skeleton code and the makefile.

Test and make sure that your programs compile well using the provided makefile on our ENGR Unix server. Zero credit if we cannot compile your programs on this server.

The function prototypes, the function names, and the names of variables in the provided four files are to be taken as is. Do not modify them. Any modification will slow down, and complicate the TA's job to grade your homework. Strictly follow the input and output formats specified for each program. Unless otherwise specified, all input/output should be accomplished using only `scanf()/printf()` functions, respectively. Your program should exit once the output is printed. Comment your code to help us grade your work.

1. Program1.c (10 points)

Write Program1.c to do the following:

- The function `int foo(int* a, int *b, int c)` should perform the following computations
  1) Increment `a`.
  2) Decrement `b`.
  3) Assign `a - b` to `c`.
  4) Return the value of `c`.
- In the main function, declare three integers `x`, `y`, and `z`, and assign them random integer values in the interval $[0, 10]$. You may use the C math library random number generator `rand()` to generate random numbers. Make sure that your use of `rand()` correctly generates nonnegative integers `x`, `y`, and `z` that are less than 11. Print the values of `x`, `y`, and `z`. Call `foo()` appropriately passing `x`, `y`, and `z` as arguments. Print out the values of `x`, `y`, and `z` after calling the function `foo()`. Also, print the value returned by `foo()`.

Grading:

1) Printing `x`, `y`, and `z` before the call to `foo()` (2 points)
2) Printing `x`, `y`, and `z` after the call to `foo()` (4 points)
3) Return value of `foo()` (4 points)

2. Program2.c (25 points)

Write Program2.c in which you will consider the following structure:

```
struct student{
   char initials[2];
   int score;
};
```

and the declaration in the main function:

```
struct student *st = 0;
```

Implement the following functions and demonstrate their functionality by calling them (in the order given) from the main function:

- `struct student* allocate()` that allocates memory for 10 students, and returns the pointer.
- `void generate(struct student* students)` that generates random initials and scores for each of the 10 students, and stores them in the array `students`. You may use the C math library random number generator `rand()` to generate random numbers. To generate the two initial letters of a student you may use the following command:

  ```
  char c1, c2;
  c1 = rand()%26 + 'A';
  c2 = rand()%26 + 'A';
  ```

  where % sign indicates the modulo operation, so that the student's initial letters take values in the English alphabet from A to Z (only capital letters), i.e., the value of (`rand()` %26) is an integer in the interval between 0 and 25 (both inclusive). Ensure that the `score` is between 0 and 100 (both inclusive).
- `void output(struct student* students)` that prints the `initials` and `scores` of all the students.
- `void summary(struct student* students)` that prints the minimum score, maximum score and average score of the 10 students.
- `void deallocate(struct student* stud)` that frees the memory allocated to `students`. Check that `students` is not `NULL` , that is `==0`, before you attempt to free it.

Grading:

1) Allocate (5 points)
2) Generate (5 points)
3) Output (5 points)
4) Summary (5 points)
5) Deallocate (5 points)

3. Program3.c (25 points)

Write a function `void sort(int* numbers, int n)` to sort a given array of `n` integers in the **ascending order**.

- In the main function, declare an integer `n`. Allocate memory for an array of `n` integers using `malloc`. Fill this array with random numbers, using the C math library random number generator `rand()`. Since `rand()` returns a random integer, you will need a `for` loop in which you will call `rand()` and set the value for each element of the array.
- Print the contents of the array.
- Pass this array along with `n` to the `sort()` function.
- Print the contents of the sorted array following the call to `sort()`.

Grading:

1) Creation of the array of random numbers (10 points)
2) Correctly sorted array of numbers in the ascending order (15 points)

4. Program4.c (40 points)

Consider the structure `student` in Program2.c. Modify the `sort()` function from Program3.c to sort an array of `n` students based on their first initial. If two students have the same first initial, you *should compare their second initial*. The function prototype is `void sort(struct student* students, int n)`. As in Program2.c, `initials` and `scores` of the students are to be generated randomly by `rand()`.

NOTE: Make sure that you can handle the case when the user provides input `n=0`.

Grading:

1) Sorts an array of student structures correctly (40 points). Note 1: a penalty of negative 10 points if your code does not correctly address the case when two students have the same first initial. Note 2: a penalty of negative 10 points if your code does not correctly address an empty array of students.

5. What to turn in?

You will turn in a single HW1.zip file — representing a folder with your four files Program1.c, Program2.c, Program3.c, Program4.c — via TEACH. Use the provided makefile to test whether your programs compile well on our ENGR LINUX server. Do not submit compiled .o files.