CS 261
DATA STRUCTURES

Oregon State University

CS 261: Data Structures in CS : Assignment 1

C Programming Practice

The purpose of this assignment is to help you get started with programming in C and give you practice with pointer manipulation.

The following specification is quite long and wordy. We are providing code skeleton files below. In those files, the specific wordy requirements (from this spec) are inserted as comments to make it quite clear what we expect for each function.

If you have any questions regarding the assignment, please post it on Piazza or email ehehsans@onid.oregonstate.edu.

In the following specification, function prototypes and names of variables that appear in questions are in **bold**. They are to be taken as is. *Do not modify function prototypes*. For this purpose, skeleton code has been provided with the questions in comments. It allows you to fill in the appropriate statements. Please don't add any new header file.

Output should be clear and understandable. Follow the input and output formats specified in each question strictly. Do not include getchar() statements in your final submission. Your program should exit once the output is printed.

Comment your code following one of the guidelines provided in Canvas .

This assignment is made up of a series of 5 small programs and will be graded for a total of 100 points. The points for each question are indicated at the end of each question. Each question should be implemented in a separate .c file (see What to Turn In below).

Unless otherwise specified, all input/output should be accomplished using scanf()/printf() respectively. For example, if a question asks you to get the value of an integer as keyboard input, you would use

```
scanf("%d",&intvar), where intvar is an integer variable.
```

scanf is very much like printf(), except that it makes the program wait for keyboard input. Notice also that it requires the the address of the target variable be given. The example above reads an integer input value from the keyboard and stores it in an int, named intvar.

Again, please make sure you download the skeleton code in the provided .zip file and use them as the starting point for answering these questions.

Q0.c

Write a program (Q0.c) to do the following:

1. In the main function, declare an integer, **x**. Then assign it to a random integer value in the interval [0, 10]. You should use the C math library random number generator rand() function to generate a random number. Then Print the value and address (using the address of operator) of **x** .
2. In **fooA(int * iptr)**, print the value of the integer pointed to by **iptr**, the address pointed to by **iptr**, and the address of **iptr** itself. Then change the value of of **iptr** as ***iptr** = 100**;**
3. In the main function, following the call to **fooA(...)**, print the value of **x**. Answer the following question in a comment at the bottom of the file: Is the value of **x** different than the value that was printed at first? Why or why not?

Scoring:

- Value and address of **x** (4 pts)
- Value of what **iptr** points to (4 pts)
- Address pointed to by **iptr** (4 pts)
- Address of **iptr** itself (4 pts)
- Answer to the question (4 pts)

Q1.c

Write a program (Q1.c) in which you consider the following structure:

```
struct student {

     int id;
     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 --

1. Write a function **struct student* allocate()** that allocates memory for ten students and returns the pointer.

2. Write a function **void generate(struct student* students)** that generates random IDs and scores for each of the ten students and stores them in the array of students. You can only make use of the **rand()** function to generate random numbers(even though that will result in the same set of random values being generated on each execution. Ensure that IDs are unique also and between 1 and 10 (both inclusive) and scores are between 0 and 100 (both inclusive). To generate unique IDs you can either use the brute-force random/check/repeat (generate a random integer between 1- 10 and then confirm that it hasn't been used yet for a student ID) or you can use the Fisher Yates shuffle - (https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle).
3. Write a function **void output(struct student* students)** that prints the ids and scores of all the students. the output of the function need not to be sorted.

4. Write a function **void summary(struct student* students)** that prints the minimum score, maximum score and average score of the ten students.

5. Write a function **void deallocate(struct student* stud)** that frees the memory allocated to students. Check that students is not NULL (NULL == 0) before you attempt to free it.

Scoring:

- Allocate (4 pts)
- Generate(10 pts)
- Output (4 pts)
- Summary(4 pts)
- Deallocate(3 pts)

Q2.c

Write a program (Q2.c) with 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 should 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().

Scoring:

1) Value of x, y, and z before the call to foo() (5 points)
2) Value of x, y, and z after the call to foo() (5 points)
3) Return value of foo() (5 points)

Q3.c

Consider the structure student in Q1.c.

1) In the main function, declare an integer n and assign it to a positive integer value. Allocate memory for n students.

2) Write a function to sort an array of n students based on their scores in ascending order. The function prototype is void sort(struct student* students, int n). The IDs and scores of the students are to be generated randomly (see use of rand()). Ensure that IDs are unique also and between 1 and n (both inclusive) and scores are between 0 and 100 (both inclusive).You should *not* use the C provided sort function (e.g. qsort()).

Scoring:

- Sorts array of student structures correctly (20 pts)

Q4.c

Write a function **void camelCase(char* word)** where word consists of more than two words separated by underscore such as "random_word" or "this_is_my_first_programming_assignment". **camelCase()** should remove underscores from the sentence and rewrite in lower camel case" (https://en.wikipedia.org/wiki/Camel_case). Watch out for the end of the string, which is denoted by '\0'. You have to ensure that legal strings are given to the camelCase**()** function.

NOTE: You can use the **toUpperCase(...)** a function provided in the skeletal code to change the case of a character from lowercase to upper case . Notice that **toUpperCase()** assumes that the character is currently in lower case. Therefore, you would have to check the case of a character before calling **toUpperCase()**.

Scoring:

- Properly inputs the string (5 pts)
- Properly converts to Camel case (15 pts)

What to turn in:

You will turn in 5 files Q0.c, Q1.c, Q2.c, Q3.c and Q4.c via TEACH and Canvas. 15% of the grade will be deducted if you don't do that. Use the provided makefile to compile on flip. Make sure that your programs compile well using the provided makefile on our ENGR Unix host. You must test your compiling on flip.engr.oregonstate.edu. Zero credit if we cannot compile your programs. Finally, please don't submit in zipped format to **TEACH**.