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 questions regarding the assignment, please post it on Piazza or email ehsans@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 (10 points will be deducted if you have done so).

Output should be clear and understandable. Do not include getchar(..) statements in your final submission. Your program should exit once the output is printed.

This assignment is made up of a series of 4 separate 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 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]. Then print the value and address (using the address of operator) of **x.**
2. In **fooA(int * iptr)** function, print the value of the integer pointed to by **iptr** and the address pointed to by **iptr**. Change the value of the integer pointed to by **iptr** as instructed in the skeleton code. Print the address of **iptr** itself at the end.
3. In the **main()** function, following the call to **fooA(..)**, print the value of **x**. Answer the following question in a comment right after printing the value of **x**:

   Is the value of **x** different than the value that was printed at first? Why or why not?
4. In **fooB(int * jptr)** function, print the value of the integer pointed to by **jptr** and the address pointed to by **jptr**. Then change the address pointed to by jptr as instructed in the skeleton code. Print the address of **jptr** itself at the end.
5. In the main function, following the call to **fooB(..)**, print the value and address of **x**. Answer the following question in a comment right after printing the value and address of **x**:

   Are the value and address of **x** different than the value and address that were printed before the call to **fooB(..)**? Why or why not?

Scoring:

- Prints the values and addresses of **x** correctly (2 + 2 + 2 pts)
- Prints the values and addresses in the function **fooA(..)** correctly (1 + 2 + 1 pts)
- Prints the values and addresses in the function **fooB(..)** correctly (1 + 2 + 1 pts)
- Answers the questions correctly (3 + 3 pts)

Q1.c

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

```
struct student{
        int score;
        Int id;
};
```

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. 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 needs 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:

- Allocates memory for ten students correctly (4 pts)
- Generates random and unique IDs correctly (2 + 5 pts)
- Outputs IDs and scores correctly (2 + 2 pts)
- Prints the summary correctly (1 + 1 + 1 pts)
- Deallocates memory correctly (2 pts)

Q2.c

Write a program (Q2.c) with the following:

1) The function int **foo(int *a, int *b, int c)** should perform the following computations –
   - **Swap the addresses of variables pointed to by the pointer variables a and b** (not the values of the integers pointed to by **a** and **b**).
   - Decrement the value of **c**.
   - Return the value of **c**.

2) In the main function, declare three integers **x, y,** and **z**, and assign them random integer values in the interval [0, 10].

3) Print the values of the integers **x, y,** and **z**. Call **foo(..)** appropriately passing **x, y,** and **z** as arguments. Print out the values of integers **x, y**, and **z** after calling the function **foo(..)**. Also, print the value returned by **foo(..)**.

4) Answer the following questions in a comment at the bottom of the file:

a) Is the return value different than the value of integer **z**? Why or why not?

b) Are the values of integers **x** and **y** different before and after calling the function **foo(..)**? Why or why not?

Scoring:

- Prints the values of integers **x, y**, and **z** before and after the call to function **foo(..)** properly (3 + 3 pts)
- Performs computations in the function **foo(..)** correctly (4 + 2 + 2 pts)
- Prints the return value of function **foo**(..) correctly (3 pts)
- Answer the questions correctly (4 + 4 pts)

Q3.c

Write a function void camelCase(char* word) where the argument "word" to the function camelCase(..) consists of at least two groups of lowercase alphabetic characters, not necessarily forming actual words in the dictionary, separated by underscores, such as “ran_dom_word” or "this_is_my_first_programming_assignment". The function camelCase(..) should remove underscores from the string and rewrite it in lower camel case (several words joined together where the first letter of the entire string is lowercase, but subsequent first letters are uppercase). Watch out for the end of the string, which is denoted by ‘\0’. It’s possible that a user provided input string may contain uppercase alphabetic characters, numeric characters, special symbols, whitespace etc. as well. You should accept such an input string too. For convenience, convert the string into a valid form before you use it for the function camelCase(..). Below are some conversion examples of what you should implement -

a) "_random_ _word_provided" should be converted to "random_word_provided" first, then finally to "randomWordProvided" using the function camelCase(..).
b) "@$ptr*4con_" should be converted to "ptr_con" first, then finally to "ptrCon" using the function camelCase(..).
c) " ran dom word" should be converted to "ran_dom_word" first, then finally to "ranDomWord" using the function camelCase(..).
d) "example word " should be converted to "example_word" first, then finally to "exampleWord" using the function camelCase(..).
e) "ANOTHER_Word" should be converted to "another_word" first, then finally to "anotherWord" using the function camelCase(..).
f) One or more combinations from options a, b, c, d and e should be also taken care of.

Examples of the strings that you must not allow as inputs are - "__", "_ _ _", " ", "435%7_$$", "random". Print “invalid input string” for such cases.

You are allowed to limit the size of the input string you will provide. Finally, do not use the library functions **isupper(..)**, **islower(..)** or **strlen(..)**. Write the code on your own to perform such string operations if required.

Scoring:

- Properly checks for the invalid inputs (5 pts)
- Properly converts to lower camel case (30 pts, 5 pts for each case presented above)

What to turn in:

You will turn in 4 files Q0.c, Q1.c, Q2.c and Q3.c via TEACH and Canvas. 15% of the grade will be deducted if you do not submit to both sites. You must use the provided makefile to compile your programs on our ENGR Unix host. Also, make sure that your programs compile well using the provided makefile on our ENGR Unix host. Zero credit if we cannot compile your programs. Finally, please don’t submit in zipped format to TEACH. 15% of the grade will be deducted if you submit in zipped format.