Oregon State UNIVERSITY | College of Engineering

Programming 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 email ehsan@eecs.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. 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 or extraneous printf() statements in your final submission. Your program should exit once the output is printed.

Comment your code following the guidelines given in the overview slides from the very first lecture .

This assignment is made up of a series of 6 small programs and will be graded for a total of 100 points. Out of this, 10 points will be awarded if your programs compile successfully on flip.engr.oregonstate.edu. 10 points are allocated for coding style, documentation, variable naming, indentation etc. 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 .zip file below and use them as the starting point for answering these questions.

- Assign1Skeleton.zip ▵
- Assign1Makefile ▵ (You must download and save this file with the name 'makefile')

Q0.c

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

1. In the main function, declare an integer, **x**. Print the address of **x** (using the address of operator). Pass **x** as an argument to a function **void fooA(int* iptr)**.
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.
3. In the main function, following the call to **fooA(...)**, print the value of **x**.

Scoring:

- Address of **x** (2pts)
- Value of what **iptr** points to (2 pts)
- Address pointed to by **iptr** (2pts)
- Address of **iptr** itself (2 pts)
- Value of **x** (2 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 make use of the **rand()** function to generate random numbers. Ensure that IDs are unique and between 1 and 10 (both inclusive) and score is between 0 and 100(both inclusive).
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 (2 pts)
- Generate(5 pts)
- Output(2 pts)
- Summary(3 pts)
- Deallocate(3 pts)

Q2.c

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

1. a function **int foo(int* a, int *b, int c)** which should perform the following computations
    1. Set the value of **a** to twice its original value.
    2. Set the value of **b** to half of its original value.
    3. Assign **a** + **b** to **c**.
    4. Return the value of **c**

2. In the main function, declare three integers **x**,**y**, and **z**, and assign them values 5, 6, and 7 respectively. Print the values of **x**,**y**, and **z**. Call **foo(...)** appropriately passing **x**,**y**, and **z** as arguments and print the returned value. After the function call, print out the values of **x**,**y**, and **z** again. Answer the following question in a comment **at the bottom of the file**: Is the return value different than the value of z? Why?

Scoring:

- **x**,**y**, and **z** before call to **foo(...)** (2 pts)
- **x**,**y**, and **z** after call to **foo(...)** (2 pts)
- Return value of **foo(...)** (2 pts)
- Explanation of return value difference ...if there is a difference. (4 pts) (**put answer at the bottom of the file**)

Q3.c

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

1. In the main function, declare an integer **n** and assign it a value of 20. 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()**. Print the contents of the array.
2. Pass this array along with **n** to the **sort(...)** function above. Print the contents of the sorted array following the call to **sort(...)**. You may *not* use the C provided sort function (e.g. **qsort()**).

Scoring:

- Creation of the array of random numbers (5 pts)
- Correctly sorted array of numbers (10 pts)

Q4.c

Consider the structure **student** in Q1.c. Modify the above **sort(...)** function from Q.3 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()**). Also you must make sure that IDs are unique.

Scoring:

- Sorts array of student structures correctly (15 pts)

Q5.c

Write a function **void sticky(char* word)** where word is a single word such as “sticky” or “RANDOM”. **sticky()** should modify the word to appear with “sticky caps” (http://en.wikipedia.org/wiki/StudlyCaps) , that is, the letters must be in alternating cases(upper and lower), starting with upper case for the first letter. For example, “sticky” becomes “StIcKy” and “RANDOM” becomes “RaNdOm”. Watch out for the end of the string, which is denoted by ‘\0’. You can assume that legal strings are given to the **sticky()** function.

NOTE: You can use the **toUpperCase(...)** and **toLowerCase(...)** functions provided in the skeletal code to change the case of a character. 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()**. The same applies for **toLowerCase()**.

Scoring:

- properly converts to sticky caps (15 pts)

What to turn in:

You will turn in 6 files Q0.c, Q1.c, Q2.c, Q3.c, Q4.c, and Q5.c via TEACH and Canvas . 20% of the grade will be deducted if you don't do that. Use the provided makefile to compile on flip. Please don’t submit in zipped format.