CS261: HOMEWORK 2  
Due 04/13/2012, at 2pm

Submit six *.c files via the TEACH website:

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

1. Introduction

The purpose of HW2 is to help you get started with programming in C, and give you practice with pointer manipulation. If you have any questions regarding HW2, please email cs261-sp12@engr.orst.edu.

This assignment consists of a series of 6 small programs, and will be graded for a total of 100 points. The points for each program are indicated at the beginning of the corresponding section in this writeup. Each program should be implemented in a separate .c file (see Section 9). The skeleton code of these 6 programs is already provided to you in 6 hand-out .c files on the class website. In the hand-out files, specific requirements are inserted as comments to explain what you need to do for each function. Download these 6 files of the skeleton code from

http://classes.engr.oregonstate.edu/eecs/spring2012/cs261/CS261Homework.html

and use them as the starting point for HW2. We have also prepared a makefile for compiling your programs. The file can be downloaded from

http://classes.engr.oregonstate.edu/eecs/spring2012/cs261/CS261Homework.html

Make sure that your programs compile well using the provided makefile, i.e., gcc on Unix. You may test your compiling on flop.engr.oregonstate.edu.

2. General Instructions

The function prototypes, the function names, and the names of variables in the hand-out 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. Do not include getchar() statements, or extraneous printf() statements in your final submission. 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 following the guidelines given in the overview slides from the very first lecture. Note that points are also allocated for your coding style, clarity of comments, variable naming, indentation, etc.
3. Program1.c  (12 points)

Write Program1.c to do the following:

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

Scoring:
1) Address of x (2pts)
2) Value of what iptr points to (2 pts)
3) Address pointed to by iptr (3 pts)
4) Address of iptr itself (3 pts)
5) Your coding style (2pts)

4. Program2.c  (12 points)

Write Program2.c with the following:

- A function int foo(int * a, int * b, int c) which 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 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().
- In the main function, answer the following question in the comment at the bottom of the file: Is the return value of foo() different from the value of z? Why?

Scoring:
1) Printing x, y, and z before the call to foo() (2 pts)
2) Printing x, y, and z after the call to foo() (2 pts)
3) Return value of foo() (2 pts)
4) Comparison of the return value of foo() and z (4 pts)
5) Your coding style (2pts)
5. Program3.c  (19 points)

Write Program3.c in which you will consider the following structure:

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

and the declaration in the main function:

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

  where % sign indicate 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.

Scoring:
1) Allocate (3 pts)
2) Generate (5 pts)
3) Output (3 pts)
4) Summary (3 pts)
5) Deallocate (3 pts)
6) Your coding style (2pts)
6. Program4.c (19 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`, and assign a value of 20 to it. 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()`.

Scoring:
1) Creation of the array of random numbers (5 pts)
2) Correctly sorted array of numbers in the ascending order (12 pts)
3) Your coding style (2pts)

7. Program5.c (19 points)

Consider the structure `student` in Program3.c. Modify the `sort()` function from Program4.c to sort an array of `n` students based on their first initial. If two students have the same first initial, you may optionally want to compare their second initial. This option will bring you additional points only if you lose some points on the other mandatory tasks of HW2. 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()` as in Program3.c.

Scoring:
1) Sorts array of student structures correctly (17 pts)
2) Checks if two students have the same first initial, and compares their second initial (optional 8pts)
3) Your coding style (2pts)
8. Program6.c  (19 points)

In the main function, Program6.c is supposed to read in two variables from the console using the function `scanf()`. The first variable is a character array `word` consisting of only letters from the English alphabet, such as “America”. The second variable is Boolean `flag` that indicates whether `word` needs to be converted to lower case, `flag==0`, or upper case, `flag==1`.

- Write two functions `void lowerCase(char * word)` and `void upperCase(char * word)`. `lowerCase()` modifies the input `word` to have all lower case letters. `upperCase()` modifies the input `word` to have all upper case letters. For example, the output of `lowerCase()` for “America” is “america”, and for `upperCase()` is “AMERICA”. Watch out for the end of the string, which is denoted by ‘\0’.
- Call in the main function `lowerCase()` or `upperCase()` depending on the user input.

NOTE1: 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 input character is currently in lower case. Therefore, you would have to check the case of a character before calling `toUpperCase()`. Similarly applies for `toLowerCase()`.

NOTE2: Make sure that you can handle the case when the user has not provided the input word.

Scoring:
1) Properly inputs the `word` and `flag` (3 pts)
2) Properly converts by `lowerCase()` any input word to lower case letters (7 pts)
3) Properly converts by `upperCase()` any input word to upper case letters (7 pts)
4) Your coding style (2pts)

9. What to turn in

You will turn in six files Program1.c, Program2.c, Program3.c, Program4.c, Program5.c, and Program6.c via TEACH. Use the provided makefile to compile on flop. engr. oregonstate.edu.