General Instructions

This assignment consists of 6 programs. Each program should be implemented in a separate .c file (see Section 7). A skeleton code for these 6 .c 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 6 files with the skeleton code and the makefile.

Make sure that your programs compile well using the provided makefile on our ENGR Unix host. You may test your compiling on flop.engr.oregonstate.edu. Zero credit if we cannot compile your programs.

The function prototypes, the function names, and the names of variables in the provided 6 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 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.

1. **Program1.c** (8 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 \( x \).

Grading:

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

2. **Program2.c** (8 points)

Write Program2.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()`.
- 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?

**Grading:**
1) Printing x, y, and z before the call to `foo()` (1 point)
2) Printing x, y, and z after the call to `foo()` (2 points)
3) Return value of `foo()` (2 points)
4) Comparison of the return value of `foo()` and z (2 points)
5) Your coding style (1 point)

### 3. Program3.c  (16 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 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 (2 points)
2) Generate (3 points)
3) Output (3 points)
4) Summary (5 points)
5) Deallocate (2 points)
6) Your coding style (1 point)

4. Program4.c (20 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 (4 points)
2) Correctly sorted array of numbers in the ascending order (14 points)
3) Your coding style (2 points)

5. Program5.c (24 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 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() as in Program3.c.

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 (22 points). Note a penalty of negative 8 points if your code does not correctly address the case when two students have the same first initial.
2) Your coding style (2 points)
6. Program6.c  (24 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.

NOTE 1: You can use the `toupper()` and `tolower()` functions provided in the skeletal code to change the case of a character. Notice that `toupper()` assumes that the input character is currently in lower case. Therefore, you would have to check the case of a character before calling `toupper()`, and similarly applies for `tolower()`.

NOTE 2: 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` (6 pts)
2) Properly converts by `lowerCase()` any input word to lower case letters (8 pts)
3) Properly converts by `upperCase()` any input word to upper case letters (8 pts)
4) Your coding style (2pts)

7. 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 test whether your programs compile well on flop.engr.oregonstate.edu. Do not submit compiled .o files.