EESC 388 Lab #1

C Programming Fundamentals

Objective:

In this lab, we will go over some essential topics of C programming which are required to know during our Embedded Systems Lab on the following weeks. We will use an online compiler for practicing and writing all the programs we learn today. Click here to launch an online compiler for C programming, or you can also use VSCode. First, we will understand the listed C programming topics below and practice them in the online compiler. After that, try to solve the assignment problems. Once you're finished, demonstrate the programs to your GTA and turn in the programs in the Canvas.

Assignment - 40 points (Max 42/40 with bonus)

Submission Deadline: You have until the end of your NEXT lab session to demo and submit your code. It is highly encouraged to get the demo done during your current lab session.

Milestone: The obtain the 10% (4 points) milestone marks for this lab, you must show progress in at least 3 of the 8 problems in this lab assignment to you GTA.

Demo: To obtain the 40% (16 points) marks for this lab, you must show your GTA at least 3 of the 8 problems completed in this lab assignment. Your TA will also ask you questions about your understanding of the code.

Some (but not all) possible questions:

- In problem #2, how many times did you have to read the array to get the max/min value? If you iterated through the array twice to get both values, is there a way to only iterate once?
- In problem #5, what happens if an element in the array occurs more than 2 times? How does your implementation handle this situation?
- In problem #7, walk me through what is happening when temp_ptr is assigned to &temp. What would happen if you tried to dereference temp_ptr before that assignment?
- Challenge question – In problem #1, if you ask the user for an integer operand first, and then the operation as a char operand second, you can find an issue with scanf() and the input buffer. Explain what is happening with scanf().

Code: To obtain the 50% (20 points) you should create a separate **.c** file for each problem and name them **p1.c** for problem 1, **p2.c** for problem 2 and so on. Once complete, upload your files individually to Canvas.

Bonus: To obtain the 2.5% (1 points) you need to complete the demo with your GTA and submit the files to Canvas by the end of your lab period. These exercises can take some time, so get a head start and come to the lab with questions!

To obtain the 2.5% (1 point) you need to complete problem #9 during your lab session and show it to the GTA during your demo of the other 3 problems. You will not get the point just for submitting the completed code for problem 9 unless you also demo it with your GTA.

Assignment

For your first assignment, you will write a few simple C programs that satisfy the requirements of the questions below. Each problem has a very specific input and output format that your program must use. This is to help automated testing of your code, so please make sure your programs conform to the specifications!

The specifications are described using examples. In the examples, [**prompt**] indicates something your program must print in order to ask users for an input. [**input**] indicates an example user input and [**output**] indicates what the correct output for the given example input should be. You shouldn't print the strings [input], [output] or [prompt] in your programs, these are just labels to help you understand.

The available point for each problem is given at the beginning. **Leave comments in your code explaining how your code solves the problem.**

1. Write a program that will work as a basic calculator that can perform addition, subtraction, multiplication, division, squaring (for e.g. 3^2), and cubing (e.g. 3^3), on user input data for two operands (or one for squaring and cubing). You can assume all operands will be of type **int**. The program should first ask for the operation, then ask for the operands. The input for type of operation will be a **char**, and '+' indicates addition, '-' indicates subtraction, '*' indicates multiplication, '/' indicates division, 's' indicates squaring, and 'c' indicates cubing. If the operation is **not** squaring or cubing, then the program asks for the second operand and then computes the result. Otherwise, it goes directly to computing the square or cube. Your program should output a single number as the result of the operation. An example of one possible run of the program is given below:

```
[prompt] Enter operation:
[input] +
[prompt] Enter 1st operand:
[input] 3
[prompt] Enter 2nd operand:
[input] 5
[output] 8
```

Another possible run could be:

```
[prompt] Enter operation:
[input] s
[prompt] Enter 1st operand:
[input] 3
[output] 9
```

2. Write a C program that can find the max value (m), minimum value (n), or both max and min (b) from the following array.

   int array[10] = {500, 1, 255, 7, -12, 40, 42, 999, 50, 227};

   For example:

   ```
   [prompt] Enter m,n, or b:
   [Input] m
   [Output] 999
   ```

   Another possible run could be:

   ```
   [prompt] Enter m,n, or b:
   [Input] b
   [Output] 999, -12
   ```

3. Write a C program to print all the prime numbers within a limit set by a user input. The input will be a positive integer and the output should be a list of numbers separated by space. For example:

   ```
   [Prompt] Enter a number:
   [Input] 6
   [Output] 2 3 5
   ```

4. Write a C program that can convert a hexadecimal number to a binary. The user will input the hex number and then the program should print the corresponding binary value. The binary input will be a series of 1s and 0s, while the hex number will be a series of digits containing the numbers 1-9 and/or the letters A-F, all capitals. Similarly, the output should be a series of 1s and 0s *only*. Be careful, however, since there are many ways to read the input, including as a string or a number and the implementation is up to you. One example could be:

   ```
   [Prompt] Please enter a hex number:
   [Input] 1A2F
   [Output] 0001101000101111
   ```

5. Write a program in C to count the total number of elements that have a duplicate in a list/array. For example, in the array (1,3,4,1,3,6), there are two elements that have duplicates, 1 and 3. Therefore the program should print 2. For the array (2,4,3,4,1,4), the program should print 1, since only one element has duplicates (i.e. 4). Your program should first ask the user how many numbers are in the array. Then it should prompt for

each element of that array. Finally, it should output the number of items which have duplicates as single integer. For example, this enters a 3-element array (1,1,8), which only has one element with duplicates:

```
[prompt] How big is the array?
[input] 3
[prompt] Enter element 1:
[input] 1
[prompt] Enter element 2:
[input] 1
[prompt] Enter element 3:
[input] 8
[output] 1
```

6. Write a program that calculates the length of a string (i.e. the number of characters in the string) using a pointer. The program first asks for a string and then prints its length. The implementation should use pointers to calculate the length. You should not use any built-in or 3rd party libraries for this task. For example:

```
[prompt] Enter string:
[input] Hello!
[output] 6
```

7. Write a program that takes an integer value from the user and stores it in an integer variable named 'temp'. Then, create an integer pointer variable named 'temp_ptr' and set it to point out to the address of your integer variable. Use printf statements to output:
   a The value of 'temp' using 'temp'
   b The value of 'temp' using the dereferenced 'temp_ptr'
   c The address of 'temp' using the reference of 'temp'
   d The address of 'temp' using 'temp_ptr'

   For example:

```
[prompt] Enter integer:
[input] 25
[output]
25,
25,
Memory address,
Memory address
```

8. Write a program that takes an integer input and a bit and returns whether that specific bit is set. Example: 7 decimal in binary is 0b0111. If the user requests bit 0,1, or 2, it will return TRUE (set), if the user requests bit 3, it will return FALSE (not set).

```
[prompt] Enter integer:
[input] 7
[prompt] Enter bit:
```

```
[input] 3
[output] FALSE
```

or

```
[prompt] Enter integer:
[input] 7
[prompt] Enter bit:
[input] 2
[output] TRUE
```

9. [BONUS] Write a program that takes an integer input, a bit, and an override value to set for that bit (either 0 or 1). Example: Integer = 7, Bit = 1, Override = 0. Output the value of the integer after writing the 0 or 1 in the specified bit location. In this example, the output would be 0b101 or 5 decimal.

```
[prompt] Enter integer:
[input] 7
[prompt] Enter bit:
[input] 2
[prompt] Enter override:
[input] 0
[output] 3
```

or

```
[prompt] Enter integer:
[input] 8
[prompt] Enter bit:
[input] 2
[prompt] Enter override:
[input] 1
[output] 12
```