**Testing and Submission Instructions:**

1. For each problem, submit **1)** your program, and **2)** one or two screenshots in **jpg** format showing that your program really works (only include the executions, no need to include the gcc compilation in the screenshots). You must submit C programs (not C++, java, etc), which can be compiled using gcc on the virtual machine used in the course. Also test your program on the virtual machine. The grader will not give credits if your program cannot be compiled or run on the virtual machine.

2. Name your programs using the pattern Problem#.c , and name your screenshots using the pattern Problem#\_Index.jpg. Problem# is the problem number (e.g., 1, 2, 3, 4, etc).  Index is the index number showing the order of the screenshots (e.g., 1 or 2).

3. Submit individual files. **DO NOT SUBMIT A ZIP FILE**.

**Problem 1**

Write a C program that extracts some bits from an unsigned integer, and prints out the unsigned integer value of these bits.

For example, the binary representation of 789 is

0000 0000 0000 0000 0000 0011 0001 0101

In the binary representation, from the lowest bit, bit 0 is 1, bit 1 is 0, bit 2 is 1, bit 3 is 0 ..... If we extracts bit 2 to bit 8, we get 7 bits: 1000101. The unsigned integer value of these 7 bits is 69.

The program reads both the unsigned integer (e.g., 789) and the range (starting bit and ending bit, e.g., 2 and 8) from the user (scanf()), and prints out the value corresponding to these bits (e.g., 69). To get the bits, your program may form a bit mask to "mask off" the bits that are not needed. Then, it shifts the bits to get unsigned integer value.

Your program must use bit-wise operations on the binary data to extracts the bits. Translating the data into a string of character '0's and '1's and then scanning the string are not an efficient way and are prohibited.

**Problem 2**

Write a C program that can find the longest continuous sequence of 1s in the binary representation of an unsigned integer. The integer cannot be 0. So there is at least one 1 in its binary representation.

For example, if the integer is 61326, the program will print out 5.  The binary representation of 61326 is

0000 0000 0000 0000

The five 1s are bit 7~ bit 11.

The program reads the unsigned integer from the user, and prints out the range (e.g., "bit 7 ~ bit 11").

Your program must use bit-wise operations on the binary data to find the bits. Translating the data into a string of character '0's and '1's and then scanning the string are not an efficient way and are prohibited.

**Problem 3**

Write a C program that flips the lower 4 bits of every character in a string (i.e., 1->0 and 0->1 for bits 0~3 in each byte). The program reads the string (a normal string) from the user and prints out the string obtained after bit-flipping (also a normal string). For example, if the input is abcd, the output should be mnlk; if the input is ZYXW, the output is UVWX.

Your program can assume that the string length does not exceed 127 (total length not exceeding 128 with the NULL character at the end).

Note that your program should not flip the bits in the NULL character (i.e., 0) at the end of the string.

Your program must use bit-wise operations to operate the bits in the string. Translating the data into characters ('0's and '1's) and then scanning the characters are not an efficient way and are prohibited.

**Problem 4**

Write a C program that scans the bits in a normal string and prints out the total number of bits that are 0s . The string is provided by the user. Your program should not scan the NULL character at the end of the string. For example, for string abcd, your program needs to scan 4 bytes \* 8 bits/byte = 32 bits.  Your program can assume that the string length does not exceed 127 (still 128 with the NULL character).

Your program must use bit-wise operations to scan the bits in the string. Translating the data into characters ('0's and '1's) and then scanning the characters are not an efficient way and are prohibited.

**Some Other Instructions**

In problems 3 and 4, to make scanf read into a string containing space characters, you can use %[^\n] as format string, e.g., scanf("%[^\n]", buf\_str);

Bitwise operators usually have low precedence.  Use parentheses when necessary.  Refer to this table for operator precedence: https://en.cppreference.com/w/c/language/operator\_precedence