CS 3432 – Computer Organization

Lab 1 Author: Robert Alvarez

This lab will introduce you to the bitwise operators not (~), and (&), or (|), xor (^), left-shift (<<), and right-shift (>>) through different exercises explained below. Pointers and memory addresses would also be introduced/used for problems 6 and 7; only 7 use heap memory.

**Background:**

Please read and study the following sections of The C Programming Language textbook as preparation for this lab:

* K&R – 2.9: Bitwise Operators.
* K&R – 5.1-4: Pointer and Arrays.
* K&R – 6.1, 6.4-5: Structures.

**Instructions:**

Assignments must be done in pairs and submitted through GitHub Classroom. Use the following link to access the assignment: <https://classroom.github.com/a/izBHhWZv>

Note that only assistance from your teammates, instructor, TA, or IA will be permitted.

**Bitwise Operators and Introduction to Memory Addresses and Structs**

In the preamble of each function definition, there is a set of parentheses containing bitwise operators, which are mandatory for constructing the function. Failure to employ each of these operators at least once within the function will result in the function’s lack of grading. This requirement is implemented to mitigate the potential for plagiarism to the greatest extent possible.

1. [10 points] (^) Develop a function capable of exchanging the values of two short numbers without necessitating using a third variable.
2. [10 points] (~, &) Given a 16-bit unsigned integer, create a function that returns “false” if the variable contains an even number of set bits (1s) and returns “true” if it contains an odd number of set bits. Your solution must have a time complexity of O(# of set bits).

Hint: How can you get the least significant bit in O(1)? Do it “# of set bits” times.

1. [15 points] Given an integer n and a short k with values between 0 and 15, make functions to
   1. (&, ~, <<) turn off,
   2. (&, <<) check and
   3. (^, <<) toggle

the bit at the specific location.

Hint: Like turn k bit on from class.

1. [15 points] (<<, |, &, >>) Design a function that determines whether the binary representation of a given number is a palindrome or not.
2. [15 points] Given a 16-bit number, create a function that returns its bits in reverse (&, |, <<). Your solution must have a time complexity of O(# of set bits).

Hint: This is the function that uses log2().

1. [20 points] (<<, &) Given an array of 120 unsigned characters, each consecutive 12 entries would be used to draw a number from 0 to 9. Make a function that receives the 2D array, one number that represents which number to draw, and an enumerate variable of how to draw it. The enumerate variable is declared as follows:

enum reflex\_num{ NO, x\_axis, y\_axis, x\_y\_axis };

Note: Number is drawn by printing the binary representation of each number per row, ‘ ‘ when 0 and ‘\*’ when 1. For example, the entry with value 0x7E would generate “ \*\*\*\*\*\* ”.

1. [15 points] (&, <<, |) Create the function “int is\_unique(const char \*s)” that receives a pointer to a sequence of characters, and it returns 1 if all its characters are unique, 0 otherwise. The function must run in O(n) time and O(1) space. [Hint: Use an array with a constant number of fixed-sized integers (sizes must be maintained regardless of the architecture)].

Hint: Come up with a O(n) time and space solution for the general “find unique elements” problem and now instead of using a set use an array declare as

“int32\_t checker[] = {0, 0, 0, 0, 0, 0, 0, 0};”

Hint2: “checker[]” is an array of 8 integers of size 32 bits each. We can see that 32\*8 = 256 and a char variable can only have values from 0 to 255.

Extra credit:

1. (^) Two files, XORDoubleLL.h and XORDoubleLL.c, are given with the declaration of the double head linked list struct and five functions, where only three are implemented. One is to print the linked list data in traversing forward on the list, one is to insert a node at the beginning of the list, and one is to perform the XOR operation between two addresses. Your job is to:
   1. [15 points] Implement insert\_node function such that it inserts the new node so that the node preceding it possesses a lower memory address, while the node following it has a higher memory address.
   2. [15 points] Implement remove\_node function, which removes a node (if it exists) based on its address and updates the linked list accordingly as required.
   3. Note: Make sure to return an integer that is 0 if the execution was successful, and 1 otherwise.

A rudimentary implementation and a brief explanation of a single-field doubly linked list can be found on the [geeksforgeeks](https://www.geeksforgeeks.org/xor-linked-list-a-memory-efficient-doubly-linked-list-set-1/) website.

**Note 1:** The use of “!” or “!=” is prohibited in the implementation of the functions declared in bitwise\_f.h. If used, a 0 would be given to any function that uses it.

**Note 2:** Code can be compiled by using the command “make” in the terminal where your project is located, and it can be compiled and run using the command “make run”.

**Permitted standard libraries:**

* math.h - log2()
* stdio.h - putchar(), malloc(), free()
* stdint.h - uintptr\_t

No additional standard C library, function, or variable is allowed. If there is a desire to use anything else, please get in touch with the AI or the TA for assistance in finding an alternative approach that does not rely on said library, function, or variable. Failure to adhere to this rule will result in a score of 0 points for the compromised function(s).

**Deadline (GitHub Classroom):** February 6th, 2024, by 11:59 pm.

1. Source code (Only .c and .h files)
   1. If another file that was not given is included, the Makefile must also be modified for a successful compilation and linkage. Failure to do so would result in a 10-point deduction.

**Grading:** (Total of 130 points where 30 points are extra credit)

Each function has the weight grade value of its implementation.