

Lab 6: Bit manipulating Problems

CPSC 2310 – Fall 2020

# Due Date: Sunday, October 11, 2020 @ Midnight

# Lab Objective

This lab is going to be a hodgepodge of functions or expressions you will implement/write that will give you practice using bit manipulation.

# Introduction

Below I will describe several task you are to complete. Each of these tasks are designed to give you practice with bit manipulation and conditional expressions. Please read the instructions for each Task.

Below is a set of rules that you must follow when the instructions indicate you must follow the **Bit-Level Rules**. In other words, when required, any code you write must follow these rules:

**Assumptions**

Integers are represented in two’s complement form.

Right shifts of signed data are performed arithmetically.

Data type int is ***w*** bits long. For some of the problems, you will be given a specific value for w, but otherwise your code should work as long as w is a multiple of 8. You can use the expression sizeof(int) << 3 to compute w.

**Forbidden**

Conditionals (if or ?:(terinary operation)), loops, switch statements, function calls, and macro invocations.

Division, modulus, and multiplication.

Relative comparison operators (<, >, <=, and >=)

**Allowed operations**

All bit-level and logic operations.

Left and right shifts, but only with shift amounts between 0 and w-1.

Addition and subtraction.

Equality(==) and inequality (!=) test. Some of the problems do not allow these so pay attention to the individual instructions.

Integer constants INT\_MIN AND INT\_MAX

Casting between data type int and unsigned, either explicitly or implicitly

Task 1.

**For this problem you must follow the Bit-Level Rules listed above, with the additional restriction that you may not use equiality (==) or inequality (!) tests. If you violate any of these Rules you will receive a 0 on this task.**

You are to write a C expression that evaluates to 1 when the following conditions are true and 0 when false.

A. Any bit of a given int equals 1.

B. Any bit of a given int equals 0.

C. Any bit in the least significant byte of a given int equals 1.

D. Any bit in the most significant byte of a given int equals 0.

I have written a program, called Task1.c, that you will fill in the expression to test your answer. We will test your answer with more than just this one, so make sure you test each expression with several values.

Hint:

These exercises require thinking about the logical operation ! in a nontraditional way. Normally we think

of it as logical negation. More generally, it detects whether there is any nonzero bit in a word. In addition,

it gives practice with masking.

For Task 2 and Task 3 I created a functions.c and functions.h file. You will implement the functions for Task 2 and 3 in the functions.c file.

Task 2:

You are going to write a function that returns 1 when ran on a machine that uses arithmetic right shifts for data type int and yields 0 otherwise. FYI – I tested the solution on the SoC servers on a joey machine so you can assume that these machines use arithmetic right shifts. This will be useful when testing your algorithm.

I have provided you with starter code called Task2.c You are to implement the isArithmetic() function. You are **not** allowed to change the prototype of the function. You will receive a 0 for this task if you change the prototype.

Task 3:

**For this problem you must follow the Bit-Level Rules listed above. If you violate any of these Rules you will receive a 0 on this task.**

You are going to write a function that returns 1 if any of the odd bits in an unsigned int equal one. As an example, the bit pattern 0100 0000 has a one in the 2nd bit position so this would return 0. However, bit pattern 1000 0000 has a 1 in bit position 1 therefore it will return 1.

I have provided you with starter code called Task3.c You are to implement the isOddOne function. You are **not** allowed to change the prototype of the function. You will receive a 0 for this task if you change the prototype.

You are to add a header to each file with the following information.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\*Your name \*

\*CPSC2310 Lab7 \*

\*UserName: \*

\*Lab Section: \*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

If you do not add a header to each of the files 5 points for each infraction will be deducted. Your files must compile and run in order to receive credit. If any of the files do not compile you will receive a zero for that task. In other words, if Task1.c does not compile you will receive a 0 for Task1, same applies for Task2.c and Task3.c .

Each task will be graded with additional test cases so please do not write your function to the specific test cases in the mains provided.

# Submission Instructions

Tar all the files and use handin (http://handin.cs.clemson.edu) to submit the tarred file to the Lab8 folder. It is your responsibility to check your files after you have submitted. This will take time so do not wait until the last minute to submit. If any of your files are corrupt or missing you will receive a 0 on this lab.