CSED 211, Fall 2022
Data Lab: Manipulating Bits
Assigned: Sept. 07, Due: Sept. 18, 11:59PM

Namgyu Park (namgyu.park@postech.ac.kr) is the lead person for this assignment.

1 Introduction

The purpose of this assignment is to become more familiar with bit-level representations of integers numbers. You'll solve five problems in the presentation.

2 Logistics

This is an individual project. All handins are electronic. Clarifications and corrections will be posted on the course Web page.

3 Handout Instructions

Start by copying datalab.tar to a (protected) directory on a Linux machine in which you plan to do your work. Then give the command

```
unix>  tar xvf datalab.tar
```

This will cause a number of files to be unpacked in the directory.
The only file you will be modifying and turning in is **bits.c**.

The **bits.c** file contains a skeleton for each of the 5 programming problems. Your assignment is to complete each function skeleton using only *straightline* code for the integer problems (i.e., no loops or conditionals) and a limited number of C arithmetic and logical operators. Specifically, you are *only* allowed to use the following eight operators:

```
! ~ & ^ | + << >>
```

A few of the functions further restrict this list. Also, you are not allowed to use any constants longer than 8 bits. See the comments in **bits.c** for detailed rules and a discussion of the desired coding style.

4 The Problems

This section describes the problems that you will be solving in **bits.c**.
We have 5 problems bitNor, isZero, addOK, logicalShift, and absVal. Using legal operation that we allow you, you need to solve problems to execute the desired behavior of the functions.
Please refer to the presentation for detailed instructions.

Name	Description	Rating
bitNor (x, y)	~(x | y) using only ~ and &	1
isZero (x)	return 0 if x is non-zero, else 1	1
addOk(x,y)	Determine if can compute x+y without overflow	3
absVal(x)	absoulte value of x	4
logicalShift(x, n)	Shift right logical.	3

Table 1: Bit-Level Manipulation Functions.

5 Evaluation

Your score will be computed out of a maximum of 12 points.
Correctness points. We will evaluate your functions . You will get full credit for a problem if it passes all of the tests, and no credit otherwise.

Autograding your work

We have included some autograding tools in the handout directory — `btest`, `dlc`, and `driver.pl` — to help you check the correctness of your work.

- **`btest`:** This program checks the functional correctness of the functions in . To build and use it, type the following two commands:

  ```
  unix> make
  unix> ./btest
  ```

  Notice that you must rebuild `btest`each time you modify your **bits.c** file.

  You'll find it helpful to work through the functions one at a time, testing each one as you go. You can use the `-f` flag to instruct `btest` to test only a single function:

  ```
  unix> ./btest -f bitNor
  ```

6 Handin Instructions

Upload your source file **bits.c** and report in plms. You need to explain your answer in the report. The format of file is (student number)_(your name).c / .doc.

7 Advice

- Don't include the `<stdio.h>` header file in your **bits.c** file, as it confuses `dlc` and results in some non-intuitive error messages. You will still be able to use printf in your **bits.c** file for debugging without including the `<stdio.h>` header, although `gcc` will print a warning that you can ignore.
- The `dlc` program enforces a stricter form of C declarations than is the case for C++ or that is enforced by `gcc`. In particular, any declaration must appear in a block (what you enclose in curly braces) before any statement that is not a declaration. For example, it will complain about the following code:

```
int foo(int x)
{
  int a = x;
  a *= 3;     /* Statement that is not a declaration */
  int b = a;  /* ERROR: Declaration not allowed here */
}
```