CSED 211, Fall 2024
Data Lab: Manipulating Bits
Assigned: Sept. 05, Due: Wed., Sept. 11, 11:59PM

Instructor: Prof. Jisung Park

Yonggon Park is the lead person for this assignment. If you have problem doing assignment, check the *FAQ* for Data Lab1 or use the Q&A board in PLMS.

1 Introduction

The purpose of this assignment is to become more familiar with bit-level representations of integers. You'll do this by solving a series of programming "puzzles." Many of these puzzles are quite artificial, but you'll find yourself thinking much more about bits in working your way through them.

2 Handout Instructions

Download the `datalab-handout.tar` from PLMS. Upload the file to a programming server, move it to the directory you want to work on. Then give the command

```
unix> tar xvf datalab-handout.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 puzzles. Your assignment is to complete each function skeleton using only *straightline* code for the integer puzzles (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.

3 Problems

This section describes the problems that you will be solving in `bits.c`. Table 1 describes a set of functions that manipulate and test sets of bits. The "Rating" field gives the difficulty rating (the number of points) for the puzzle, and the "Max ops" field gives the maximum number of operators you are allowed to use to implement each function. See the comments in `bits.c` for more details on the desired behavior of the functions. You may also refer to the test functions in `tests.c`. These are used as reference functions to express the correct behavior of your functions, although they don't satisfy the coding rules for your functions.

Name	Description	Rating	Max Ops
`bitNor(x,y)`	Compute `~(x | y)` using only `~` and `&`	1	8
`isZero(x,n)`	Return 0 if `x` is non-zero, else 1	1	2
`addOk(x,y)`	Determine if can compute `x+y` without overflow	3	20
`absVal(x)`	Return absolute value of `x`	4	10
`logicalShift(x,n)`	Perform logical right shift to `x` by `n`	3	20

Table 1: Bit-Level Manipulation Functions.

4 Evaluation

Your score will be computed out of a maximum of 12 points. We will evaluate your functions using the `btest` program, which is described in the next section. 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 `bits.c`. 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 bitAnd
  ```

You can feed it specific function argument using the option flags `-1`, `-2`, and `-3`:

```
unix> ./btest -f bitAnd -1 7 -2 0xf
```

Check the file `README` for documentation on running the `btest` program.

- **`dlc`:** This is a modified version of an ANSI C compiler from the MIT CILK group that you can use to check for compliance with the programming rules for each problem. The typical usage is:

  ```
  unix> ./dlc bits.c
  ```

  The program runs silently unless it detects a problem, such as an illegal operator, too many operators, or non-straightline code in the integer puzzles. Running with the `-e` switch:

  ```
  unix> ./dlc -e bits.c
  ```

  causes `dlc` to print counts of the number of operators used by each function. Type `./dlc -help` for a list of command line options.

- **`driver.pl`:** This is a driver program that uses `btest` and `dlc` to compute the correctness and performance points for your solution. It takes no arguments:

  ```
  unix> ./driver.pl
  ```

  TA will use `driver.pl` to evaluate your solution.

5 Handin Instructions

Upload your `bits.c` file and report in PLMS with the following format:

- Rename your `bits.c` file as: `[student id].c` (e.g., `20231234.c`)
- Submit your report with **pdf** format: `[student id]`**`.pdf`** (e.g., `20231234`**`.pdf`**)

Submissions with incorrect format will not be graded, no exception (i.e., you will get 0 points for this entire lab).

6 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 */
}
```