

CS33, Spring 2020

Lab 1: Data Lab

Assigned: Wed., April 1, Due: Fri., April 17, 11:59PM

1 Introduction

The purpose of this assignment is to become more familiar with bit-level representations of integers and floating point numbers. 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 Logistics

You must work on this lab independently, but you may discuss the assignment with other students. All code you write and submit must be your own.

3 Handout Instructions

We will use the shared directory `/w/class.1/cs/cs33/csbin/` on SEASnet machine `lnxsrv.seas.ucla.edu` to distribute the files needed for each lab. Log in to your account on SEASnet machine and start copying the folder `datalab-handout` to your local directory using this command:

```
cp -r /w/class.1/cs/cs33/csbin/datalab-handout .
```

This will make a copy of the handout folder in your directory. The only file you will be modifying and turning in is `bits.c`. The file `btest.c` allows you to evaluate the functional correctness of your code. The `README` file contains additional documentation about `btest`. Use the command `make btest` to generate the test code and run it with the command `./btest`. The file `dlc` is a compiler binary

that you can use to check your solutions for compliance with the coding rules. The remaining files are used to build btest.

The `bits.c` file contains a skeleton for each of the ¹⁰~~15~~ 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.

4 The Puzzles

This section describes the puzzles that you will be solving in `bits.c`. 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.

4.1 Bit Manipulations

Table 1 describes a set of functions that manipulate and test sets of bits.

Name	Description	Rating	Max Ops
<code>bitParity(x)</code>	returns 1 if x contains an odd number of 0’s.	4	20
<code>rotateRight(x,n)</code>	Rotate x to the right by n.	3	25
<code>byteSwap(x,n,m)</code>	swaps the nth byte and the mth byte.	2	25
<code>fitsShort(x)</code>	return 1 if x can be represented as a 16-bit, tc integer.	1	8
<code>bitAnd(x,y)</code>	Compute x&y using only ~ and .	1	8

Table 1: Bit-Level Manipulation Functions.

4.2 Two's Complement Arithmetic

Table 2 describes a set of functions that make use of the two's complement representation of integers. Again, refer to the comments in `bits.c` and the reference versions in `tests.c` for more information.

Name	Description	Rating	Max Ops
<code>subOK(x,y)</code>	Determine if can compute x-y without overflow	3	20
<code>isGreater(x,y)</code>	if $x > y$ then return 1, else return 0	3	24
<code>fitBits(x,n)</code>	return 1 if x can be represented as an n-bit, tc integer.	2	15
<code>negate(x)</code>	return $-x$	2	5
<code>isTmax(x)</code>	returns 1 if x is the maximum, tc number, 0 otherwise	1	10

Table 2: Arithmetic Functions

5 Evaluation

Your score will be computed out of a maximum of 36 points based on the following distribution:

20 Correctness points.

16 Performance points.

Correctness points. The 8 puzzles you must solve have been given a difficulty rating between 1 and 4. We will evaluate your functions using the `btest` program, which is described in the next section. You will get full credit for a puzzle if it passes all of the tests performed by `btest`, and no credit otherwise.

Performance points. Our main concern at this point in the course is that you can get the right answer. However, we want to instill in you a sense of keeping things as short and simple as you can. Furthermore, some of the puzzles can be solved by brute force, but we want you to be more clever. Thus, for each function we've established a maximum number of operators that you are allowed to use for each function. This limit is very generous and is designed only to catch egregiously inefficient solutions. **You will receive two points for each correct function that satisfies the operator limit.**

6 Handin Instructions

- Make sure it compiles, passes the dlc test, and passes the btest tests on the class machines, i.e. `lnxsrvc.seas.ucla.edu`.

- Make sure you have included your identifying information in your file `bits.c`.
- Remove any extraneous print statements.
- Submit your `bits.c` file to CCLE where indicated under Lab 1.

7 Advice

You are welcome to develop your solution using any system or compiler you choose. However, make sure that the version you turn in can compile and run correctly on our class machine (`lnxsrv.seas.ucla.edu`). If it does not compile, we cannot grade it.

The `dlc` program is a modified version of an ANSI C compiler that you can use to check for compliance with the coding rules for each puzzle. The typical usage is:

```
./dlc bits.c
```

Type `./dlc -help` for a list of command line options. The `README` file is also helpful. Some notes on `dlc`:

- The `dlc` program runs silently unless it detects a problem
- 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.