

# CS 2400, Fall 2010

## Lab Assignment L1: Manipulating Bits

### Introduction

The purpose of this assignment is to become more familiar with bit-level representations and manipulations. 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.

### Logistics

You need to work by yourself on this assignment but you may ask for assistance in tools and the process of compiling your programs. You shouldn't ask other students for help about the substance of the assignment – you're going to have to understand the problems well enough to be interviewed about them. The only "hand-in" will be electronic. Any clarifications and revisions to the assignment will be posted on the course Moodle page.

### Getting ready for the next assignment

This is the only lab or programming assignment where you'll be working on your own. Please use the next two weeks to meet and talk with people in class. For the next assignment, you'll be asked to work with a programming partner; we encourage you to use the "Extreme Programming" style. We're doing this because the next sequence of assignments are actually very difficult to do without someone to talk with and reason with. These labs are learning mechanisms, and working together will help with that learning.

### Hand Out Instructions

You need to use a Linux system for this lab because there is a "pre-built" binary that only runs on Linux. You had two options to do this lab. The first would be to download the files in `data-lab-handout.tar` at the assignment page and unpack them on your own or a CSEL linux machine. The second is download a VMware image that contains a Linux environment and the assignment, unpacked.

If you use the tar file, start by copying `datalab-handout.tar` to a (protected) directory in which you plan to do your work. Then give the command: `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 file `btest.c` allows you to evaluate the functional correctness of your code. The file `README` 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`.

Looking at the file `bits.c` you'll notice a C structure `team` into which you should insert the requested identifying information. Do this right away so you don't forget. Despite the description in the file, we're not using programming teams and you're expected to work out your own solutions.

The `bits.c` file also contains a skeleton for each of the 15 programming puzzles. Your assignment is to complete each function skeleton using only *straightline* code (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.

## Evaluation

You will be evaluated for two things - having working code and (more importantly) the ability to explain why your code is correct. This latter explanation will take place in a grading meeting with a TA.

This lab is worth 100 points. 40% of those points will arise from having working or correct solutions, following the grading rubric shown on the course website.

Your code will be compiled with GCC and run and tested on one of the class vmware machines. We will evaluate your functions using the test arguments in `btest.c`. You will get full credit for a puzzle if it passes all of the tests performed by `btest.c`, half credit if it fails one test, and no credit otherwise.

Regarding performance, 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.

The other 60% of the assignment score will arise from your ability to explain the assignment. You will be asked about  $\approx 4$  of the problems on this assignment, including ones of rank 1, 2, 3 and 4.

## Problems

The `bits.c` file describes a set of functions that manipulate and test sets of bits. Each problem as a “Rating” and a specified “Max ops”. 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.

For example, function `isZero` determines if the argument is zero. Other problems deal with two’s complement arithmetic. They tend to be harder and need more steps.

The problems are ordered by complexity; do the earlier problems first.

## Advice

You should do your development using the virtual machine we provided or something similar. Just make sure that the version you turn in compiles and runs correctly on our class machines (*i.e.* the CSEL machines or your VMware image). If it doesn’t compile, we can’t grade it.

The `dlc` program, a modified version of an ANSI C compiler, will be used to check your programs for compliance with the coding style rules. 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 `<stdio.h>` in your `bits.c` file, as it confuses `dlc` and results in some non-intuitive error messages.

Check the file `README` for documentation on running the `btest` program. 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, e.g., `./btest -f isPositive`.

## Hand In Instructions

Use the Moodle assignment page to upload your `bits.c` file.

- Make sure you have included your identifying information in your file `bits.c`.
- Remove any extraneous print statements.