ICS SS09, Spring 2010
Lab Assignment L1: Manipulating Bits
Assigned: Wed., Mar. 24, Due: Mon., Apr. 5, 15:59

TAs (YKQ YLW HZJ WZG) would in charge of this assignment for their groups.

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 should work **individually** in solving the problems for this assignment. Any clarifications and revisions to the assignment will be posted on the News page of ICS course websit (http://10.132.143.100/note.htm).

Obtain Lab Materials

You should use svn tools to get lab1. The URL of svn repository is "`svn://10.132.143.1/ics-ss09/[account]`" (the example of `[account]` is "`ics09302010001`"). You need select a directory named "`lab1`" to check out, which contains 11 files:

> **Makefile, README, bits.c, bits.h, btest.c, btest.h, decl.c, dlc, tests.c,datalab.pdf and datalab.ps**

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` 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 about only **ONE** individuals comprising your programming team. Do this right away so you don't forget. The "`team name`" and the "`login ID`" are all your student id (i.e. 09302010001), and the "`student name`" is the PINYIN of your name (i.e. haile ding)

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

Your code will be compiled with GCC and run and tested on one of the class machines. Your score will be computed out of a maximum of 75 points based on the following distribution:

45 Correctness of code running on one of the class machines.

30 Performance of code, based on number of operators used in each function.

The 15 puzzles you must solve have been given a difficulty rating between 1 and 4, such that their weighted sum totals to **45**. 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. You will receive two points for each function that satisfies the operator limit.

Advice

We recommend that you should use linux and work directly on it, which could make sure that the version you turn in compiles and runs correctly on our test machines. 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`.

Last but not least, if you are confused by the meaning of the function, you could read the corresponding code in tests.c, which is the C version code of bits.c.

handin

You only need to commit your bits.c file, but you need to commit it at least three times. If you violate the rule or exceed the deadline, you can't get any point!

- The 1st time: you finish 5 puzzles.
- The 2nd time: you finish 10 puzzles.
- The last time: you finish lab1.

NOTE: There is no requirement to the order of puzzles to be done.