LAB 4
Manipulating Bits

1 Introduction

The purpose of this lab 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. You will be assigned a partner for this lab.

2 Warm-Up Exercises

1. Fill in the following table showing the results of evaluating Boolean operations on bit vectors.

Operation	Result
a	01101001
b	01010101
$\sim a$	
$\sim b$	
a & b	
$a \mid b$	
$a \hat{} b$	

2. Suppose that x and y have byte values 0x66 and 0x39, respectively. Fill in the following table indicating the byte values of the different C expressions:

Expression	Value	Expression	Value
x & y		x && y	
$x \mid y$		$x \mid\mid y$	
$\sim x \mid \sim y$		$!x \mid\mid !y$	
x & $!y$		x && $\sim y$	

3. Fill in the table below showing the effects of the different shift operations on single byte quantities. The best way to think about shift operations is to work with binary representations. Convert the initial values to binary, perform the shifts, and then convert it back to hexadecimal. Each of the answers should be 8 binary digits or 2 hexadecimal digits.

x		$x \ll 3$		(Logical) $x \gg 2$		(Arithmetic) $x \gg 2$	
Hex	Binary	Hex	Binary	Hex	Binary	Hex	Binary
0xC3							
0x75							
0x87							
0x66							

3 Handout Instructions

Download `lab4.tar` from our course websire to a directory on your home directory in which you plan to do your work. Then give the command

```
$ tar xvf lab4.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.

4 The Puzzles

This section describes the puzzles that you will be solving in `bits.c`. The following table 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
`bitAnd(x,y)`	`x & y` using only `|` and `~`	1	8
`getByte(x,n)`	Get byte `n` from `x`.	2	6
`logicalShift(x,n)`	Shift right logical.	3	20
`bitCount(x)`	Count the number of 1's in `x`.	4	40
`bang(x)`	Compute `!n` without using `!` operator.	4	12

5 Evaluation

Your score will be computed out of a maximum of 24 points based on the following distribution:

14 Correctness points.

10 Performance points.

Correctness points. The 5 puzzles you must solve have been given a difficulty rating between 1 and 4, such that their weighted sum totals to 14. 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.

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:

  ```
  $ make
  $ ./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:

  ```
  $ ./btest -f bitAnd
  ```

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

  ```
  $ ./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 coding rules for each puzzle. The typical usage is:

  ```
  $ ./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:

  ```
  $ ./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:

  ```
  $ ./driver.pl
  ```

  Your instructors will use `driver.pl` to evaluate your solution.

6 Handin Instructions

When completed, upload your `bits.c` solution file.