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 working with a partner on this lab.

2 Handout Instructions

Download `lab4.tar` from our course website to a directory on your home directory of a lab machine 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.

3 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.

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

4 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.

5 Autograding your work

We have included an autograding tools in the handout directory — `driver.pl` — to help you check the correctness of your work. It takes no arguments:

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

6 The "Beat the Prof" Contest

For fun, we're offering an optional "Beat the Prof" contest that allows you to compete with other students and the instructor to develop the most efficient puzzles. The goal is to solve each puzzle using the fewest number of operators. Students who match or beat the instructor's operator count for each puzzle are winners!

To submit your entry to the contest, type:

```
$ ./driver.pl -u <Your Team Name>
```

You can submit as often as you like. Your most recent submission will appear on a real-time scoreboard, identified only by your nickname. You can view the scoreboard by pointing your browser at

```
http://lab.cs.trincoll.edu:8082
```