CSC373, Winter 2016-2017
Data Lab: Manipulating Bits
Assigned: January 10
Due: January. 23, 11:59PM

1 Introduction

The purpose of this assignment 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.

2 Logistics

This is an individual project. All handins are electronic. Clarifications and corrections will be posted to the discussion forum on D2L.

3 Handout Instructions

When you login, you should see a directory named `datalab` in your home directory. It contains a number of files, but the only file you will be modifying and turning in is `bits.c`.

The `bits.c` file contains a skeleton for each of the 11 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.

4 The Puzzles

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

4.1 Bit Manipulations

Table 1 describes a set of functions that manipulate and test sets of bits. The "Rating" field gives the number of points for the puzzle. (Note that the **easier** puzzlers are worth more points.) 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 `~`	10	8
`oddBits()`	returns word with all odd-numbered bits set to 1	10	8
`reverseBytes(x)`	reverse the bytes of `x`.	8	25
`allEvenBits(x)`	returns 1 if all even-numbered bits of `x` are 1, else 0	6	12
`conditional(x,y,z)`	same as `x ? y : z`.	4	16
`bitCount(x)`	Count the number of 1's in `x`.	2	40

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
`minusTwo()`	return the int value -2	12	2
`tmax()`	greatest two's complement integer	10	4
`negate(x)`	`-x` without using -	8	5
`times34(x)`	multiply by 34	6	6
`isLess(x,y)`	return 1 if `x < y`, else 0	2	24

Table 2: Arithmetic Functions

5 Evaluation

Your score will be computed out of a maximum of 100 points based on the following distribution:

78 Correctness points.

22 Performance points.

Correctness points. The 11 puzzles you must solve have been given a point value rating between 2 and 12, such that their sum totals to 78. I 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 it obeys all the coding restricitons, and no credit otherwise.

Performance points. My main concern at this point in the course is that you can get the right answer. However, I 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 I want you to be more clever. Thus, for each function I'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 also satisfies the operator limit.

Autograding your work

I 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:

  ```
  unix> make
  unix> ./btest
  ```

  Notice that you must rebuild `btest` (rerun make) 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. For example, to test only the bitAnd function, use:

  ```
  unix> ./btest -f bitAnd
  ```

  You can feed it specific function arguments using the option flags `-1`, `-2`, and `-3`. For example, to test bitAnd with 7 as the first argument and 0xf as the second argument, use:

  ```
  unix> ./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:

  ```
  unix> ./dlc bits.c
  ```

The program runs silently unless it detects a problem, such as an illegal operator, too many operators, or non-straightline code. Running with the `-e` switch:

```
unix> ./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:

  ```
  unix> ./driver.pl
  ```

  I will use `driver.pl` to evaluate your solution. You can check your current score at any time by running driver yourself.

6 Handin Instructions

To submit your solution, make sure you are in the directory containing the bits.c file you want to submit and type:

```
unix> /csc373/datalab_submissions/submit
```

Verify that you have submitted the correct file by typing:

```
unix> /csc373/datalab_submissions/verify
```

This will dump your submission to the screen. You can also redirect to a file that you can open in emacs with:

```
unix> /csc373/datalab_submissions/verify > current_submission
```

which will create a file in the current directory called current_submission.

You may submit as often as you like. Each new submission overwrites the previous submission.

7 Advice

- Submit early and often. You can submit as often as you like. Everytime you do something that earns you more points, submit. It takes less than a minute to guarantee you the better score.
- In each function, declare all your local varibles first, before you make assignments. The autograding tools are based on an older C standard that does not allow variable declarations after statements.

- 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. You will still be able to use `printf` in your `bits.c` file for debugging without including the `<stdio.h>` header, although `gcc` will print a warning that you can ignore.
- The `dlc` program enforces a stricter form of C declarations than is the case for C++ or that is enforced by `gcc`. In particular, any declaration must appear in a block (what you enclose in curly braces) before any statement that is not a declaration. For example, it will complain about the following code:

```
int foo(int x)
{
  int a = x;
  a *= 3;     /* Statement that is not a declaration */
  int b = a;  /* ERROR: Declaration not allowed here */
}
```

  So just make sure to declare **all** the variables you will use in your function **before** you write any actual statements.

8 The "Beat the Prof" Contest

For fun, I'm 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 Data Lab puzzle using the fewest number of operators. Students who match or beat the instructor's operator count for each puzzle win bragging rights!

To enter the contest, type:

```
unix> ./driver.pl -u ''Your Nickname''
```

Nicknames are limited to 35 characters and can contain alphanumerics, apostrophes, commas, periods, dashes, underscores, and ampersands. 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://marrero373.cstcis.cti.depaul.edu:6010
```

Entry in the contest does <u>not</u> constitute submission of your lab assignment. To get credit you must submit as described in the Handin Instructions section of this writeup.