CS105 – Computer Systems Fall 2024

Assignment 2: Data Lab

Due: Tuesday, September 10, 2024 at 11:59pm PT

The purpose of this assignment is to give you familiarity with bit-level representations of signed integers and floating point numbers and with various operations performed on binary values. You will accomplish the goal by solving a series of programming "puzzles." Even though many of the puzzles are quite artificial, you will find yourself thinking much more about bits in working your way through them.

You must work in a group of two people in solving the problems; partners will be assigned for this assignment. You should complete this assignment using pair programming, and you and your partner should submit one solution. *I strongly recommend that you and your partner brainstorm before coding!*

Getting Started

The materials for this lab are available on the course web page and on the course VM. I strongly recommend that you complete this assignment on the VM.

First, ensure that you are connected to the Pomona network or the Pomona VPN. Then `ssh` to the VM using your Pomona username (e.g., abcd1234):

```
% ssh USERNAME@itbdcv-lnx04p.campus.pomona.edu
```

and unpack the starter code into your home directory on the VM:

```
% tar xvf /cs105/starters/datalab.tar
```

This will cause a number of files to be unpacked in the directory. The only file you will be modifying and is `bits.c`.

Begin by opening the file in an editor and *put both your names* in the comments at the top of the `bits.c` file. Do this right away!!

The `bits.c` file contains a skeleton for each of the 8 programming puzzles. Your assignment is to complete each function skeleton using limited types and numbers of C logical and arithmetic operators. Additionally, the first 6 puzzles must be completed using only *straightline* code (no loops or conditionals) and no function calls; also, you are not allowed to use any constants longer than 8 bits. (The rules are relaxed for the two float puzzles). You may use local variables, but you should declare all such variables at the top of functions before doing anything else. *Failure to do so may break the autograder.* You have been warned!

Each function heading tells you what is allowed. See the comments in `bits.c` for detailed rules and a discussion of the desired coding style.

Compiling the Code

We strongly suggest that you do all your work on the course VM. You can be sure that the support programs `dlc` (for testing compliance with the coding rules) and `btest` (for testing correctness) will work there. In the past, some students have found that these programs do not run correctly on other machines.

I strongly recommend that you work through the functions one at a time, testing each one as you go. We

have given you a `Makefile` to ease the burden of running the compiler. You can use it to compile everything, including all testing code, by typing

```
% make
```

You may ignore the warning about a "non-includable file."

The `dlc` Program

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.
- You can use the `-e` flag to instruct `dlc` to count the number of operations you use (in addition to checking for disallowed operations).
- The `dlc` program runs silently unless it detects a problem.
- Do not include `<stdio.h>` in your `bits.c` file (it confuses `dlc` results in non-intuitive errors).
- In ANSI C, you must make all variable declarations at the beginning of a function. The following code is not accepted by `dlc`.

```
int mask = 0x55 + (0x55 << 8);
mask = mask + (mask << 16);
int shift = (x >> 1);
int sum = (shift & mask) + (x & mask);
```

The `btest` Program

Once you pass the style-checker with `dlc`, you can test your function for correctness with `btest`. Note that you will need to re-compile `btest` every time you make changes to `bits.c`. You can do this with the general command

```
% make
```

or by typing

```
% make btest
```

To test your solution for correctness, you should then run the test code

```
% ./btest
```

If you only want to test one function, you can use the `-f` flag, for example

```
% ./btest -f bitXor
```

The `driver.pl` Program

I will use `./btest` and `./dlc` to grade your assignment via the Gradescope autograder. You can check your current grade yourself by running the exact same grading script as follows:

```
% ./driver.pl
```

Evaluation

Your score will be computed out of a maximum of 40 points. Each function will be evaluated separately for correctness and performance.

- **Correctness (20 points):** We will use the programs `driver.pl`, `btest`, and `dlc`, supplied with the starter code, to evaluate your solution. No points will be given for a function if `dlc` reports an illegal operator or another error.
- **Performance (16 points):** We will use the programs `driver.pl` and `dlc`, supplied with the laboratory materials, to evaluate your code. No points will be given for a function if `dlc` reports an illegal operator, too many operators, or another error.
- **Style (2 points):** For this assignment, "good style" is easy to attain. It means that your files are submitted correctly, your names are present at the top of each file, that your code is understandable and consistently indented, that comments—when necessary to explain—are present and easy to read, and that there is no extraneous material.
- **Feedback (2 points):** An additional 2 points will be awarded for submitting a completed feedback file.

Hint: Remember that you can run the Perl script `driver.pl` to see your current Correctness and Performance scores. It will also report the total number of operations you used.

Submission Instructions

When you have finished, submit two files, `bits.c` and `feedback.txt`, on Gradescope. As always, you can download files from the VM to your local machine by running the `scp` command from your local machine. Be sure to tag your partner as your group member and submit both files in the same submission!

Part I: Bit Manipulatinos

Table 1 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.

Function `bitOr` computes the bitwise or. You may only use the operators `&` and `~`. Function `bitXor` should duplicate the behavior of the operation `^`, using only the operations `&` and `~`.

Function `copyLSB` replicates a copy of the least significant bit (the "ones" column) in all 32 bits of the result.

Function `conditional` returns `y` if `x` is true and `z` otherwise. **Hint:** note that the parameters for these functions are declared as signed integers. Unlike unsigned integers, right shifting values of type `int` uses *arithmetic* shifting.

Name	Description	Rating	Max Ops
`bitOr(x,y)`	`x|y` using only `&` and `~`	1	8
`bitXor(x,y)`	`^` using only `&` and `~`	1	14
`copyLSB(x)`	Set all bits to LSB of `x`	2	5
`conditional(x,y,z)`	`x ? y : z`	3	16

Table 1: Bit-Level Manipulation Functions.

Part II: Two's Complement Arithmetic

Table 2 describes a set of functions that make use of the two's complement representation of integers.

Function `addOK` determines whether its two arguments can be added together without overflow.

Function `absVal` returns the absolute value `|x|` .

Name	Description	Rating	Max Ops
`addOK(x,y)`	Does `x+y` overflow?	3	20
`absVal(x)`	returns `|x|`	4	10

Table 2: Arithmetic Functions

Part II: Float Arithmetic

Table 3 describes a set of functions that make use of single precision floating point representation of integers. For these puzzles, you may use

Function `float_abs` returns the absolute value of the argument.

Function `float_f2i` casts a float to an int.

Name	Description	Rating	Max Ops
`float_abs(f)`	Returns `|f|`	2	10
`float_f2i(f)`	Returns `(int) f`	4	30

Table 3: Arithmetic Functions

Important Note: The coding rules are relaxed for floats: you may use conditionals and large constants to solve these puzzles. See notes in the starter code for details.

Part III: Feedback

Create a file called `feedback.txt` that answers the following questions:

1. How long did each of you spend on this assignment?
2. Any comments on this assignment?

How you answer these questions **will not affect your grade**, but whether you answer them will.