CS105 – Computer Systems Spring 2020

Assignment 2: Data Lab

Due: Friday, February 7, 2020 at 5pm

The purpose of this assignment is to give you familiarity with bit-level representations of integers and floating point numbers and with various operations performed on these data types. 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. *We strongly recommend that you and your partner brainstorm before coding!*

Getting Started

The materials for the data lab are available on the course web page and on the course VM.

First `ssh` into `pom-itb-cs2.campus.pomona.edu`. Then copy/download `datalab-handout.tar` to a (protected) directory in which you plan to do your work. Next give the command

```
% tar xvf datalab-handout.tar
```

which 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 and userids in the comments at the top of the `bit.c` file. Do this right away!!

The `bits.c` file contains a skeleton for each of the 13 programming puzzles. Each function heading tells you what operations are allowed. For the first two parts, you must complete each function skeleton using only *straightline* code (no loops or conditionals) and a limited number of C arithmetic and logical operators. Further, you are not allowed to use any constants longer than 8 bits. (These rules are relaxed for the float puzzles.) 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 `btest` and `dlc` will work there; in the past, many students have found that these programs do not run correctly on other machines. In any case, make sure that the version you turn in compiles and runs correctly on `pom-itb-cs2`. If it fails to compile there, we cannot grade it.

Check the file `README` for documentation on running the compiler dlc and the `btest` program. You will find it helpful to work through the functions one at a time, testing each one as you go. You can use the `-e` flag to instruct `dlc` to count the number of operations you use (in addition to checking for disallowed operations). Once you pass the tests with `dlc`, you can test your function with `btest`. Note: you can use the `-f` flag to instruct `btest` to test only a single function, as in `./btest -f bitAnd`.

Dig more deeply into the `README` file for information on some helper programs.

We have given you a `Makefile` to ease the burden of running the compiler. You can open the file and look

at if you like. It is slightly more complicated than the Makefile we were using last week, but it is still the same general idea. Type

```
% make btest
```

to compile the program `btest`, or simply

```
% make
```

to compile everything.

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.
- 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).
- Running with the `-e` switch causes `dlc` to print counts of the number of operators used by each function.
- ANSI C, and hence `dlc`, disallows `//` comments.
- 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);
```

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

Evaluation

Your code will be run and tested on the course VM (`pom-itb-cs2`). Your score will be computed out of a maximum of 65 points. Each function will be evaluated separately for correctness and performance.

- **Correctness (33 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 or another error.
- **Performance (26 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 (4 points):** Your `bits.c` file will be evaluated by the graders and given up to 4 points for style. 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.

Note: 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. If you are a competitive type, my solution uses 82 total operations. :-)

Submission Instructions

When you have finished, submit two files, `bits.c` and `feedback.txt`, to the course submission page.

- Make sure you have included all your team members' names in your files.
- Remove any extraneous print statements before submitting the files.
- Use the submission system, linked from the course web page, to submit the files.
- Use all the team members' names when submitting (tag your partner as a collaborator!) and submit both files in the same submission.
- If you discover a mistake in your code, simply submit the files again.

Part I: Bit Manipulations

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

Function `isNotEqual` compares `x` to `y` for inequality. As with all *predicate* operations, it should return `1` if the tested condition holds and `0` otherwise. Function `copyLSB` replicates a copy of the least significant bit in all 32 bits of the result. Function `condition` returns `y` if `x` is true and `z` otherwise.

Function `logicalShift` performs logical right shifts. Function `rotateLeft` rotates the bits to the left by `n`. You may assume the shift/rotation amount n satisfies $0 \leq n \leq 31$.

Function `bang` computes logical negation without using the `!` operator.

Name	Description	Rating	Max Ops
`bitAnd(x,y)`	`x&y` using only `~` and `|`	1	8
`bitXor(x,y)`	`^` using only `&` and `~`	1	14
`isNotEqual(x,y)`	`x != y?`	2	6
`copyLSB(x)`	Set all bits to LSB of `x`	2	5
`conditional(x,y,z)`	`x ? y : z`	3	16
`logicalShift(x,n)`	Logical right shift `x` by `n`	3	16
`bang(x)`	Compute `!x` without using `!` operator	4	12

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 `tmax` returns the largest integer.

Function `isNonNegative` determines whether `x` is less than or equal to 0.

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

Function `isPower2` determines whether `x` is a power of 2.

Name	Description	Rating	Max Ops
`tmax(void)`	largest two's complement integer	1	4
`isNonNegative(x)`	`x >= 0`?	3	6
`addOK(x,y)`	Does `x+y` overflow?	3	20
`isPower2(x)`	Is `x` a power of 2?	4	20

Table 2: Arithmetic Functions

Part III: 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 loops, conditionals, and/or large constants.

Function `float_neg` returns the argument multiplied by `-1`.

Function `float_f2i` casts a float to an int.

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

Table 3: Arithmetic Functions

Part IV: 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.