COMP 201 - Fall 2020
Assignment 1 - Data Lab: Manipulating Bits
Assigned: 18 October 2020, Due: 28 October 2020 11:59PM

Samet Demir (`sdemir20@ku.edu.tr`) is the lead person for this assignment.

1 Introduction

The purpose of this assignment is to become more familiar with bit-level representations of integers and floating point numbers. 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 announced on Blackboard.

3 Handout Instructions

3.1 How to start

I Accept the GitHub Classroom assignment using the link: https://classroom.github.com/a/1lzbJayA

II Clone the GitHub repository created for you to a Linux machine in which you plan to do your work (We advice you to do your work on our linux servers [linuxpool.ku.edu.tr]. See Section 8 for details.) :
`$ git clone https://github.com/COMP201-Fall/comp201-fall20-assign1-USER.git`
(Replace USER with your GitHub username that you use to accept the assignment)

3.2 Task

There are a number of files 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 15 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`.

4.1   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. 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
`isZero(x)`	Is `x == 0`?	1	2
`implication(x,y)`	`x -> y` in propositional logic[1]	2	5
`twoDigitNumberInBaseFour(x,y)`[2]	Integer equivalent of $(xy)_4$	2	4
`multThreeEighths(x)`	Multiply `x` by 3/8 rounding toward 0	3	12
`bang(x)`	Compute `!n` without using `!` operator	4	12

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
`tmax()`	Most positive two's complement integer	1	4
`isOppositeSigns(x,y)`	Do `x` and `y` have different signs?	3	6
`conditional(x,y,z)`	Same as `x ? y : z`	3	16

Table 2: Two's Complement Arithmetic Functions

4.3   Floating-Point Operations

For this part of the assignment, you will implement some common single-precision floating-point operations. In this section, you are allowed to use standard control structures (conditionals, loops), and you may use both `int` and `unsigned` data types, including arbitrary unsigned and integer constants. You may not use any unions, structs, or arrays. Most significantly, you may not use any floating point data types, operations, or constants. Instead, any floating-point operand will be passed to the function as having type `unsigned`, and any returned floating-point value will be of type `unsigned`. Your code should perform the bit manipulations that implement the specified floating point operations.

---

[1]See the following web page for the truth tables: http://sites.millersville.edu/bikenaga/math-proof/truth-tables/truth-tables.html

[2]For `twoDigitNumberInBaseFour(x,y)`, you can assume $x, y \in \{0, 1, 2, 3\}$

Table 3 describes a set of functions that operate on the bit-level representations of floating-point numbers. Refer to the comments in `bits.c` and the reference versions in `tests.c` for more information.

Name	Description	Rating	Max Ops
`float_abs(uf)`	Compute $\lvert f \rvert$	2	10
`float_f2i(uf)`	Compute `(int) f`	4	30

Table 3: Floating-Point Functions. Value `f` is the floating-point number having the same bit representation as the unsigned integer `uf`.

Functions `float_abs` and `float_f2i` must handle the full range of possible argument values, including not-a-number (NaN) and infinity. The IEEE standard does not specify precisely how to handle NaN's, and the IA32 behavior is a bit obscure. For `float_abs`, when the argument is NaN, return the argument. For `float_f2i`, anything out of range (including NaN and infinity) should return `0x80000000u`.

The included program `fshow` helps you understand the structure of floating point numbers. To compile `fshow`, switch to the handout directory and type:

```
$ make
```

You can use `fshow` to see what an arbitrary pattern represents as a floating-point number:

```
$ ./fshow 2080374784

Floating point value 2.658455992e+36
Bit Representation 0x7c000000, sign = 0, exponent = f8, fraction = 000000
Normalized.  1.0000000000 X 2^(121)
```

You can also give `fshow` hexadecimal and floating point values, and it will decipher their bit structure.

5 Evaluation

Your score will be computed out of a maximum of 50 points based on the following distribution:

25 Correctness points.

20 Performance points.

5 Style points.

Correctness points. The 10 puzzles you must solve have been given a difficulty rating between 1 and 4, such that their weighted sum totals to 25. 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.

Style points. Finally, we've reserved 5 points for a subjective evaluation of the style of your solutions and your commenting. Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they need not be extensive.

Important Note: We use automated plagiarism detection to compare your assignment submission with others and also the code repositories on GitHub and similar sites. Moreover, we plan to ask randomly selected 10% of students to explain their code verbally after the assignments are graded. And one may lose full credit if he or she fails from this oral part.

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 isZero
  ```

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

  ```
  $ ./btest -f isOppositeSigns -1 7 -2 10
  ```

  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

    I Write your name and student number in bits.c file - line 4 as comment.

    II Commit all the changes you make: `$ git commit -a -m "final commit"`

    III Push your work to GitHub servers: `$ git push origin master`

7   Advice

- 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 */
}
```

- Use linuxpool.ku.edu.tr linux servers to test your code in order to avoid compatibility issues.

8   How to use linuxpool.ku.edu.tr linux servers

    I Connect to KU VPN (If you are connected to the KU network, you can skip this step.)
      See for details: https://confluence.ku.edu.tr/kuhelp/ithelp/it-services/network-and-wireless/vpn-access

    II Connect to linuxpool.ku.edu.tr server using SSH (Replace USER with your Koç University username):
      `$ ssh USER@linuxpool.ku.edu.tr`
      (It will ask your password, type your Koç University password.)

    III When you are finished with your work, you can disconnect by typing: `$ exit`

Your connection to the server may drop sometimes. In that case, you need to reconnect.

We advice you to watch the following video about the usage of SSH, which is used to connect remote servers, and SCP, which is used to transfer files between remote servers and your local machine: https://www.youtube.com/watch?v=rm6pewTcSro

Figure 1: How to connect and disconnect using SSH

9   Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else. See Koç University - Student Code of Conduct.

10   Late Submission Policy

You may use up to 5 grace days (in total) over the course of the semester for the assignments. That is you can submit your solutions without any penalty if you have free grace days left. Any additional unapproved late submission will be punished (1 day late: 20% off, 2 days late: 40% off) and **no submission after 2 days will be accepted.**

Acknowledgement

This assignment is adapted from "Carnegie Mellon University, CS 15-213 Course - Data Lab Assignment".