Profs. Jingke Li, Classes: TR 14:00-15:50, KMC 390, Office Hours: TR 1300-1400 (in-person @FAB 120-06).

Assignment 2 (Bit-Level Manipulation)

(Due Thursday 2/6/25 @11:59pm)

This assignment is aimed at helping you become more familiar with bit-level representations and operations. You'll do this by solving a series of programming "puzzles." You'll find yourself thinking much more about bits in working your way through them.[1]

This assignment carries 20 points in total. Individual puzzles' points are specified in the listing table on the next page.

Setup

Unzip `Assignment 2` zip file into a (protected) directory on a Linux machine in which you plan to do your work. You'll see the following program files in the directory:

```
bits.c            - The file you will be modifying and handing in
bits.h            - Header file
btest.c, btest.h  - The main btest program
decl.c, tests.c   - Support programs for building btest
dlc               - Rule checking compiler binary
ishow.c, fshow.c  - Utility programs for examining int & float bit representations
sample.c          - Two sample puzzles with solutions
Makefile          - For compiling relevant programs
```

Your Task

The `bits.c` file contains skeletons of ten programming puzzles. Your task is to complete each function skeleton (except the last one) using:

- only straightline code (*i.e.*, no loops or conditionals);
- a limited number of C arithmetic and logical operators; and
- no no constants larger than 8 bits (*i.e.*, 0 - 255 inclusive).

For the last function skeleton, `float_twice()`, the constraints are more relaxed. You are allowed to use both `if` and `while` statements, and constants of any size.

We assume a 32-bit word size for all these puzzles. (A compiler flag `-m32` is used to enforce this requirement in the `Makefile`.)

The table below describes the ten puzzles, along with their difficulty rating (which translates to the number of points they each carry). Some hints are also provided. More complete (and definitive, should there be any inconsistencies) documentation is found in the `bits.c` file itself.

Carefully read the instructions in the `bits.c` file before you start. These give the coding rules that you will need to follow if you want full credit.

Checking Code for Compliance

Use the provided `dlc` compiler (`./dlc`) to automatically check your version of `bits.c` for compliance with the coding guidelines:

[1] This assignment is adapted from [B&O]'s Datalab.

Puzzle	Description	Rating
`int tmin(void)`	Returns Tmin (*Hint:* words are 32-bit)	1
`int isEqual(int,int)`	Returns 1 if $x = y$, 0 otherwise (*Hint:* 0 is false, anything else is true	1
`int evenBits(void)`	Sets all even bits to 1 (*Hint:* construct from bytes)	1
`int getByte(int,int)`	Extracts nth byte from int x (*Hint:* bytes are 8 bits)	2
`int anyOddBit(int)`	Returns 1 if any odd-bit is 1 (*Hint:* use a mask)	2
`int isNegative(int)`	Returns 1 if $x < 0$, 0 otherwise (*Hint:* consider shifting)	2
`int isPositive(int)`	Returns 1 if $x > 0$, 0 otherwise (*Hint:* handling 0 is the key)	2
`int sign(int)`	Returns 1 if pos, 0 if zero, -1 if neg (*Hint:* use shifting)	2
`int replaceByte(int,int,int)`	Replaces byte n in x with c (*Hint:* use mask and shifting)	3
`unsigned float_twice(unsigned)`	Doubles the value of f (*Hint:* treat different cases separately)	4

```
linux> ./dlc bits.c
```

(*Note:* `Dlc` will always output the following warning, which can be ignored:
`/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-predef.h.`)

`Dlc` returns silently if there are no problems with your code. Otherwise it prints messages that flag any problems. Running `dlc` with the `-e` switch:

```
linux> ./dlc -e bits.c
```

causes `dlc` to print counts of the number of operators used by each function. Note that the `dlc` compiler is built for use on linux machines only.

Testing with `btest`

Once you have a legal solution, you can test it for correctness using the `./btest` program.

The `Makefile` in this directory compiles your version of `bits.c` with additional code to create a program (or test harness) named `btest`. To compile and run the `btest` program, type:

```
linux> make btest
linux> ./btest [optional cmd line args]
```

You will need to recompile `btest` each time you change your `bits.c` program. When moving from one platform to another, you will want to get rid of the old version of `btest` and generate a new one. Use the commands:

```
linux> make clean
linux> make btest
```

`Btest` tests your code for correctness by running millions of test cases on each function. It tests around well known corner cases such as `Tmin` and zero. When `btest` detects an error in one of your functions, it prints out the test that failed, the incorrect result, and the expected result, and then terminates the testing for that function. Here are some usage examples:

- Test all functions for correctness and print out error messages:

```
linux> ./btest
```

- Test all functions in a compact form with no error messages:

```
linux> ./btest -g
```

- Test a specific function for correctness:

```
linux> ./btest -f isEqual
```

- Test a specific function with specific arguments:

```
linux> ./btest -f isEqual -1 5 -2 5
```

"`-1`" and "`-2`" specify the first and second arguments to the function `isEqual`.

`Btest` does not check your code for compliance with the coding guidelines. Use `dlc` to do that.

The `ishow` and `fshow` Helper Programs

The two provided programs, `ishow` and `fshow`, can help you decipher integer and floating-point representations, respectively. Run `make` first to build these programs. Example usages:

```
linux> ./ishow 27
Hex = 0x0000001b,   Signed = 27,    Unsigned = 27

linux> ./fshow 3.14
Floating point value 3.140000105
Bit Representation 0x4048f5c3, sign = 0, exponent = 0x80, fraction = 0x48f5c3
Normalized.  +1.5700000525 X 2^(1)
```

Submission and Grading

Upload your modified copy of `bits.c` (with your name and Odin id inserted at the top) on Canvas through the "File Upload" tab in the "Assignment 2" folder. (You need to press the "Start Assignment" button to see the submission options.)

This assignment is to be graded automatically by two programs, `dlc` and `btest`, for compliance and correctness. Since both programs are provided, you should test your program yourself before submission.