CS 105
Lab 1: Manipulating Bits

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

You will work on a team of two people in solving the problems for this assignment. The only "hand-in" will be electronic; see Section 7 for instructions on how to submit. Any clarifications and revisions to the assignment will be posted on the course Web page. **We strongly recommend that you and your partner brainstorm before coding**. Be sure to read this entire writeup before you begin.

3 About Working on Wilkes

All of the labs in CS 105 must be done on Wilkes, the department machine that has been specifically set up for this course. To run programs on Wilkes, you must `ssh` to access the machine and work from the command line.

3.1 Getting and Using `ssh`

If you are working on one of the lab Macs, if you have a Mac of your own, or if you have a Linux box, `ssh` should be preinstalled. In that case you merely need to type "`ssh` *username*`@wilkes.cs.hmc.edu`" into a terminal (command-line) window, where *username* is your Knuth/Wilkes login ID (usually first initial and last name). You will be prompted for your password and, after you type it, you will get a normal Linux command prompt.

If you are running Windows 10, a search for "Windows ssh" will lead you to this Web page, which will give you instructions on how to make `ssh` work for you. If you are running an earlier version of Windows, search for "putty ssh" to find downloads (from `www.putty.org`) and instructions.

3.1.1 Editing Files

If you are logged into a lab Mac as yourself (not the guest account), you can edit directly on that machine using whatever editor you prefer, and your changes will show up on Wilkes.

If you prefer a text-based editor that is available on Wilkes, such as `emacs`, you can simply invoke it from the Wilkes command line, regardless of how you got to Wilkes.

If you are running on a machine of your own, life is slightly more complex. A good option for Linux users is to use sshfs to make things appear in a directory on your local machine; `sshfs` is also available on Macs. In that case, you will have your choice of editor.

On all platforms, another sensible option is to use Visual Studio Code (`vscode`) as your editor. If you do, there is a plugin that will allow you to use Visual Studio Code locally and have the changes show up on Wilkes.

4 Lab Setup and Overview

The materials for the bits lab are in a `tar` file on the course webpage for Lab 1. Be sure to first follow the directions for getting set up on the CS department server `wilkes` if you haven't yet already.

Then create a (protected) directory in which you plan to work. **Working on Wilkes,** change into that directory and type either "`wget` *link*" or "`curl` *link*", where *link* is the link to the tar file (right-click or control-click on the link on the lab Web page, choose "Copy link location", "Copy link address", "Copy link", or just "Copy", depending on your browser, and then paste the link into your terminal window). That will download `bits-handout.tar` into your directory. Then give the command

```
unix> tar xvf bits-handout.tar
```

(Don't type the "`unix>`" part; that represents the command prompt.) This will cause a number of files to be unpacked 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 14 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**. In other words, all constants must fall in the decimal range $[0, 255]$. See the comments in`bits.c` for detailed rules and a discussion of the desired coding style.

BE SURE TO PUT YOUR NAMES AND YOUR KNUTH LOGIN IDS IN THE COMMENTS AT THE TOP OF BITS.C!

5 The Puzzles

This section describes the puzzles that you will be solving in `bits.c`. There are two sets of puzzles: (1) bit manipulations and (2) two's complement arithmetic.

5.1 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 notes below as well as comments in `bits.c` for more details on the desired behavior of the functions.

Name	Description	Rating	Max Ops
`bitXor(x,y)`	`^` using only `&` and `~`	1	14
`isNotEqual(x,y)`	`x != y?`	2	6
`getByte(x,n)`	Extract byte `n` from `x`	2	6
`copyLSB(x)`	Set all bits to least significant bit of `x`	2	5
`logicalShift(x,n)`	Logical right shift `x` by `n`	3	20
`bitCount(x)`	Count number of 1's in `x`	4	40
`bang(x)`	Compute `!x` without using `!` operator	4	12
`leastBitPos(x)`	Mark least significant 1 bit	2	6

Table 1: Bit-Level Manipulation Functions.

- Function `bitXor` should duplicate the behavior of the bit 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 `getByte` extracts a byte from a word. The bytes within a word are ordered from 0 (least significant) to 3 (most significant).
- Function `copyLSB` replicates a copy of the least significant bit in all 32 bits of the result.
- Function `logicalShift` performs logical right shifts. You may assume the shift amount n satisfies $0 \leq n \leq 31$.
- Function `bitCount` returns a count of the number of 1's in the argument.
- Function `bang` computes logical negation without using the `!` operator.
- Function `leastBitPos` generates a mask consisting of a single bit marking the position of the least significant one bit in the argument. If the argument equals 0, it returns 0.

5.2 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 notes below and the comments in `bits.c`.

Name	Description	Rating	Max Ops
`tmax(void)`	largest two's complement integer	1	4
`isNonNegative(x)`	`x >= 0?`	3	6
`isGreater(x,y)`	`x > y?`	3	24
`divpwr2(x,n)`	`x/(1<<n)`	2	15
`abs(x)`	absolute value	4	10
`addOK(x,y)`	Does `x+y` overflow?	3	20

Table 2: Arithmetic Functions

- Function `tmax` returns the largest integer.
- Function `isNonNegative` determines whether `x` is less than or equal to 0.
- Function `isGreater` determines whether `x` is greater than `y`.
- Function `divpwr2` divides its first argument by 2^n, where n is the second argument. You may assume that $0 \leq n \leq 30$. It must round toward zero.
- Function `abs` is equivalent to the expression `x<0?-x:x`, giving the absolute value of `x` for all values other than $TMin$.
- Function `addOK` determines whether its two arguments can be added together without overflow.

6 Evaluation and Autograding

This section describes how your work will be evaluated. Note that included in your lab materials are the same autograding programs that will be used to grade your submission.

Evaluation

Your score will be computed out of a maximum of 69 points:

36 Correctness points.

28 Performance points.

5 Style points.

Correctness points. The 14 puzzles you must solve have been given a difficulty rating between 1 and 4, such that their weighted sum totals to 36. We will evaluate your functions using the `btest` program, which is described in the next subsection. 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.

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:

  ```
  unix> make
  unix> ./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:

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

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

```
unix> ./btest -f bitAnd -1 7 -2 0xf
```

Check the file `README` for documentation on running the `b`test 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
  ```

  You may need to make dlc executable with this command: `chmod +x dlc`. Note that 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:

  ```
  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
  ```

  Your instructors will use `driver.pl` to evaluate your solution.

7 Handin Instructions

Be sure you've run BOTH `btest` and `dlc` to check for issues before you submit! Also check that you've included both partners names/logins in the comments at the top of `bits.c` and that you've removed any extraneous print commands.

From the command line on `wilkes` in your lab directory type

```
cs105submit -a 01 bits.c
```

You should get a response that 1 file has been submitted and that you will get an email confirming your submission. Note, you can resubmit as often as you want up to the deadline. Only the most recent submission will be graded. Do not make any submissions after the deadline unless you are using late days.

8 Advice and Notes

- 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. Be sure to remove any printf statements before running driver.pl.
- Variable declarations: the dlc program enforces a stricter form of C declarations than is the case for C/C++ or that is enforced by gcc. In particular, any and all declarations 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;      /* A statement that is not a declaration */
  int b = a;  /* ERROR: Declaration of b not allowed here */
}
```

- Be sure that your code compiles and runs correctly on wilkes!