CS154, Autumn 2019
Project 1: Bit Manipulation in C ("`p1bitmanip`")
Due: Thursday October 10, 11:59pm

1 Introduction

The purpose of this assignment is to help students become more familiar with bit-level representations of integers. You'll do this by solving a series of programming "puzzles."

READ ALL INSTRUCTIONS CAREFULLY. Every year, some students are surprised that they lose unexpected points because they failed to follow all the instructions in this document.

2 Motivation

A deep understanding of a computer system requires a deep understanding of the way the computer represents data. This assignment is designed to get you familiar with thinking about the bit-level representation of data.

In addition, understanding how to manipulate bit-level representations can often open the door to code optimizations. For example, a professor in our department was once responsible for performance tuning a video encoder. This encoder had a strict performance requirement – to be considered correct, the code had to both produce a correct output and do so in a fixed amount of time. After profiling the code, he found that a significant amount of time was being spent performing integer division in software (the particular hardware being used did not support a divide instruction). His understanding of the bit-level representation of numbers allowed him to replace the general integer division code (which was slow) with a specific implementation for the small set of numbers that could appear as a divisor in this code. This custom implementation was significantly faster than the general version and allowed him to meet the timing requirement. This project emphasizes the same types of skills he used in this real-world problem of video encoding.

3 Logistics

Code that does not compile will receive no credit. Do not submit code that does not compile. This is a general guideline that is true beyond this class: if you break your company's build system because you did not test before checking in, the least bad thing that will happen is severe embarrassment. For this assignment, always run `driver.pl` (see below for more) before checking in to ensure that you get the credit you expect. If you can't solve one of the puzzles, make sure that you are not keeping the entire submission from compiling because of the incomplete code for that puzzle, for instance.

You must work alone to solve the problems for this project. The only "hand-in" will be electronic, via an SVN repository hosted on Phoenixforge (as you learned about in Lab 1). Any clarifications and revisions to

the assignment will be announced on Piazza.

You should not perform web searches for the tasks you are trying to complete in this assignment. While practicing programmers are able to resolve technical questions frequently with a quick web search, and this is a good thing, it is not appropriate for this assignment. The reason is that these puzzles do not lend themselves well to hints, and you are likely to inadvertently find complete answers. If you hand in answers you have found online, even if you were only exposed to the answers unexpectedly, you are committing an academic honesty violation. The safest course for this assignment is to puzzle through the questions yourself and restrict your attempts to get help to Piazza and office hours.

To your `CNetID-cs154-spr-19` SVN repository has been added a new directory `p1bitmanip`. This is where you will do your work. **The only file from you that we will be grading is `bits.c`, so that is the only file you should be modifying. Do not add additional files to your repository. Do not alter the testing files, as this could give you optimistically false results while you work on the assignment — but the grading we perform will use the original versions..**

4 Instructions

Inside `p1bitmanip` there are a number of files. You should start by reading the `README`. The file `btest.c` allows you to evaluate the functional correctness of your code. Use the command `make btest` to generate the test code and run it with the command `./btest`. The file `dlc` is a compiler binary that you can use to check your solutions for compliance with the coding rules on x86 machines running Linux. The remaining files are used to build `btest`.

The `bits.c` file contains a skeleton for each of the programming puzzles, and it is the only file you are supposed to modify. Your assignment is to complete each function skeleton using only *straightline* code (i.e., no loops or conditionals) for the puzzles 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.

5 The Puzzles

This section describes the puzzles that you will be solving in `bits.c`.

As you read through `bits.c` you will see the functions for which you have to provide the implementation, within the constraints described. Each function is one "puzzle". The "Rating" gives the approximate 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 above each function 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 do not satisfy the coding rules for your functions.

6 Evaluation

Your score will be computed out of a maximum of 103 points based on the following distribution:

59 Correctness points.

44 Performance points.

Correctness points. 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. The weighting for the individual puzzles in the correctness will be weighed by the difficulty score (between 1 and 4).

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. This limit is very generous and is designed only to catch egregiously inefficient solutions. You will receive performance credit for each correct function that satisfies the operator limit.

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
  ```

  The program runs silently unless it detects a problem, such as an illegal operator, too many operators, or non-straightline code in the 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
  ```

  We will use `driver.pl` to evaluate your solution.

7 Handin Instructions

We recommend checking your updates to the file `bits.c` into your phoenixforge account frequently. When you are ready to hand in your solution, check your final version of `bits.c` into your phoenixforge account. Always test before checking in code.

Please make sure you have included your identifying information in `bits.c` (write your name and CNetID as comment at the top) and removed any extraneous print statements.

8 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 in a block (what you enclose in curly braces) must appear 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 */
  }
  ```

  Simply reorder your code to place your declarations above any statements to avoid this issue, keeping in mind that you can declare a variable without assigning an initial value to it.
- The following warning is expected, acceptable, and can be ignored:

  ```
  /usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line>
  included from includable file /usr/include/stdc-predef.h.
  ```

Acknowledgments:

This lab was developed by the authors of the course text and their staff. It has been customized for use here by various instructors in the department.