CS33, Spring 2018
Lab 0: Mini Data Lab
Assigned: Monday, April 2, **Due:** Friday, April 6 11:59PM

1 Introduction

Lab 0 is the simpler version of Lab 1: Data Lab. For this lab, you need to implement a single ”programming puzzle” described in Part B. The purpose of this warm-up assignment is to make sure all students:

1. can access their SEAS account remotely
2. are comfortable using basic Linux commands and programming editors
3. are familiar with Data lab structure and debugging techniques

2 Part A: Setup Your Environment

We will use the shared directory /w/class.1/cs/cs33/csbin/ on SEASnet machine lnxsrv.seas.ucla.edu to distribute the files needed for each lab. Log in to your account on SEASnet machine and start copying the folder datalab0-handout to your local directory using this command:

```
unix>      cp -r /w/class.1/cs/cs33/csbin/datalab0-handout .
```

This will make a copy of the handout folder in your current directory . The only file you will be modifying and turning in is bits.c. The README file contains additional documentation for each file in the folder. While you are expected to implement 8 puzzles in Lab 1, you will implement a single puzzle in Lab 0 just to get familiar with the SEASnet machine and Datalab infrastructure

You could either code and debug on the SEASnet machine or copy the handout to your personal machine and code and debug locally. For either case, refer to **setupenvironment.pdf** for more information.

3 Part B: Get Familiar with Data Lab structure

As mentioned in Part A, you will only modify and hand in the bits.c. Looking at the file bits.c, you will notice a C structure studentID into which you should insert the requested identifying information about yourself. Do this immediately so that you do not forget.

Your assignment is to implement the function ezThreeFourths(x) = (x*3/4) (described in the class) using only *straightline* code (i.e., no loops or conditionals) and a limited number of C arithmetic and logical operators:

! ~ & ^ | + << >>

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. 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:

  ```
  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 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.

  Note: Depending on how you copied the handout directory into your local directory it is possible that some of the permission flags are reset. You may get an error: **Permission denied**. You can reset the execute permission flag by using the command:

  ```
  unix>    chmod +x dlc
  ```

  In general, you can reset the execute permission flag for any file using: chmod +x <file-name>

- **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.

4 Handin Instructions

- Make sure it compiles, passes the dlc test, and passes the btest tests on the class machines, i.e. lnxsrv.seas.ucla.edu.
- Make sure you have included your identifying information in your file bits.c.
- 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.
- Remove any extraneous print statements.
- Submit your bits.c file to CCLE where indicated under Lab 0.