# CEN 591, Fall 2012, Assignment 1
# Familiarizing with Linux/Unix and C
# Assigned: Tue., Sept. 4, Due: Mon., Sept. 17, by noon

## 1 Introduction

The purpose of this assignment is to become familiar with C Linux/UNIX programming and review some of computer system basic such as memory hierarchy and data representation in a computer system. The assignment has three questions, in the first question you are asked to state some Linux command lines. In the second and third questions you are asked to develop some C functions to work with memory hierarchy and bit representation of integer and floating point numbers.

## 2 Logistics

This assignment must be done individually. All handins are electronic. Clarifications and corrections will be posted on the blackboard

## 3 Handout Instructions

You can download the assignment handout, `HW1-handout.tar`, from the course Web page.

Start by copying `HW1-handout.tar` to a directory on a Linux machine in which you plan to do your work. Then give the command

```
unix> tar xvf HW1-handout.tar.
```

This will cause a number of files to be unpacked in the directory that you will need them to develop the bit manipulation functions.

If you do not have access to a Unix-like OS on your personal machine, you can use the following options:

- Remote access to `general.asu.edu`. Download "SSH for windows" from `http://myapps.asu.edu`, install SSH, create a profile in SSH to connect to `general.asu.edu`, and login into it using your ASURITE ID and password.

- Emulate Linux on your windows: Install cygwin to emulate Linux on your windows based machine
- CIDSE open lab, , second floor, BYENG 214 : There are some MAC machines in the lab where you can access. Also some of computers have Linux boot option which you can use.

## 4 Linux/Unix Assignment

The purpose of this assignment is to familiarize you with the command line Linux environment that you will be using this class. This assignment covers only some of the basic usages of Linux/UNIX command line including file system, file permission Linux pipeline and filtering.

**Basic commands to create file and directory**

- Create a folder named `CENHW1`, and change your path to the folder using `cd` command
- Use `vi` command to create two files named `file1` and `file2`.

**Setting the permissions of files**

- Use the `ls -l` command to determine the permission of the files. Use one command line to add the permission of 'executing' of `file1` to your id and 'writing' permission to your group (the group that your id is belonged) (command 1)
- set `file2`'s permission to be the same as `file1`'s permission using a single command (assume you do knot know what the permissions of `file1` are) (command 2)

In Unix-like operating you can take the output of one program and send it to another as the input. This is called piping. With pipes, you can merge multiple commands together and make your own powerful commands. In the following you'll learn how to use pipelining to filter the output of the commands or apply the same function to a group of data.

**pipelining and filtering in Linux command lines**

- Using `who` and and `wc` commands in a pipe line sequence to determine how many users are logged in your system. (command 3)
- In the previous command, if a user is logged in more than once, there may be multiple entries for that person. Now devise a pipeline sequence of commands to display a sorted list of people currently logged in with no duplication in names. Use `awk` command to remove multiple entries. (command 4)
- Using `ls` and awk in a pipe line sequence rename the `file1` and `file2` to `file1.txt` and `file1.txt` (command 5)
- The very first column in the output of `ls -l` can be a 'd', a '-' or a 'l' depending on whether the file object is a directory, regular file or symbolic link. Using the `ls -l`, `grep` and `wc` commands in a pipeline sequence determine how many directories, regular files and symbolic links exist in the `\usr\bin` directory (commands 6-8)

### 4.1 Submission format

You need to modify and turn in the file `linuxcommands` for this assignments. Similar to the 3 first commands that have been written in the file, you need to write the 8 mentioned commands on every line sequentially. Note that the order of the commands is important.

## 5 Memory hierarchy

This assignment deals with optimizing memory access by leveraging spatial and temporal locality. Well-written programs tend to exhibit good locality. That is they tend to reference data items that are near to recently used data (spatial locality) or data items that were recently referenced themselves (temporal locality). Trivial change to a program such as the arrangement of a loop or arrangement of data can have a big impact on its locality. For example the following codes (i.e., code a and code b) both compute the sum of the matrix `a[n][m]`. The only difference is that we have interchanged the *i* and the *j* loop. Since `C` arrays are laid out in memory row-wise, code b suffers from poor spatial locality because it scans the array column-wise instead of row-wise.

***code a***:
```
for (i = 0; i < n; i++)
for (j = 0; j < m; j++)
sum += a[i][j];
```
***code b***:
```
for (j = 0; j < m; j++)
for (i = 0; i < n; i++)
sum += a[i][j];
```

Matrix multiplication is an example that can benefit from locality. You will see in this assignment how a loop arrangement or data organization can make difference in the total execution time of a matrix multiplication program. Your task in this homework is to develop matrix multiplication for square matrices as follows:

- Develop 6 different C functions to multiply two matrices such that each has different locality properties (use loop arrangement and matrix transposing to result different locality)
- Develop a C program to use the functions and report their performance (i.e., total execution time)
- Plot a graph of the performance of your functions versus the array size of matrices
- Choose the array size in the following range 100 to 1000 with a step of 100

### 5.1 Hints

You can use C libraries such as `gettimeofday()` before and after the multiplication function in your C program to calculate the total execution time of your function. Type `man gettimeofday` on your Linux machine to see how this function can be used. You are free to use other functions to calculate the total execution time.

Further, you can develop some scripts to automate the whole process (i.e., running the program with different array sizes and multiplication function versions and plotting the result) as follows.

- Write a Makefile to compile the C program once it is changed
- Write a script( e.g., using `bash` scripts or `perl`) to invoke the C program for different multiplication versions and array sizes (using Makefile and the previous script) and generate the plot (e.g., using Gnuplot )

### 5.2 Submission format

You need to create two `C` files named `mulmain.c` and `mulfunc.c` to write the `main` program and the multiplication functions, respectively. Define your multiplication functions as follows `int** mul{x}(int** a,int** b, int n)`, where x is the function number (it should be 1-6) and n is matrices' length. `mulfunc.c` should has the implementation codes of all of your 6 functions, i.e., mul1, ... mul6 (see `mulfunc.c` in the handout). You also need to submit a sing-page pdf document to report the performance plot.

## 6 Integer representations

In this assignment you will become familiar with bit representation of integer and floating numbers. You will do this by solving a series of programming "puzzles". Many of these puzzles are quite artificial, but youll find yourself thinking much more about bits in working your way through them.

Your assignment is to develop 6 functions as shown in Table 1 using only *straightline* code for the integers (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 Table 1 the "Points" field gives the number of points for the function, and the "Max ops" field gives the maximum number of operators you are allowed to use to implement each function. Finally, the "Bonus pints" filed shows the extra credit assigned to 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.

The functions `tmin`, and `isPositive` in the table 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.

The functions `float_neg`, and `float_twice` in the table make use of floating point numbers. Both the argument and result are passed as unsigned int's, but they are to be interpreted as the bit-level representations of single-precision floating point values. For detail description of these functions see `bits.c`.

If you are not familiar with bit representation of integer and floating point see the slided provided in the hand-out named `IntFloatBitRepresentation.ppt`. For further details refer to the book `Computer Systems: A Programmer's Perspective, R. Bryant and D. O'Hallaron, 2nd ed, Ch 2, Sec 2.2 and 2.4.`

| Name | Description | Points | Max Ops | Bonus points |
|---|---|---|---|---|
| `bitAnd(x,y)` | `x & y` using only `\|` and `~` | 5 | 8 | 0 |
| `bitCount(x)` | Count the number of 1's in `x`. | 0 | 40 | 7 |
| `tmin()` | Most negative two's complement integer | 5 | 4 | 0 |
| `isPositive(x)` | `x > 0`? | 5 | 8 | 0 |
| `float_neg(uf)` | Compute −f | 5 | 10 | 0 |
| `float_twice(uf)` | Computer 2*f | 0 | 30 | 8 |

Table 1: Bit-Level Functions.

## 6.1 Submission format

The only file you will be modifying and turning in is `bits.c`. The `bits.c` file contains a skeleton 15 bit-wise functions,but you just need the complete the 6 functions shown in Table 1.

## 6.2 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 bit-wise functions.

- **`btest`:** This program checks the functional correctness of the functions in `bits.c` (you only need to check your grades on the 6 mentioned functions). 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 `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.

- **`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 Evaluation

Your score will be computed out of a maximum of 90 points ($+$ 35 bonus points) based on the following distribution:

**10** points for Linux assignment

**30** points for 6 different multiplication functions, each has 5 points

**30** points for the performance plot, i.e., array size versus total execution time for all different multiplication functions.

**20** bonus points for developing scripts to automate the performance plot generation.

**20** points and [15] bonus points for bit manipulation functions (see the field rating in Table 1 for details).

## 8 Submission Instructions

Zip all of your files including `linuxcommands`, `mulmain.c`, `mulfunc.c`, bits.c, your report document and any other files (e.g., scripts). Name your zip file as [ASUID]H1 (e.g. 222222222H1), and submit it using blackboard.