# File I/O; Bit Operations

*CS 350: Computer Organization & Assembler Language Programming*

*Lab 3, due Wed Sep 24 ~~Mon Sep 22~~* [*]

## A. Why?

- Reading from files is popular.
- Bit operations are needed to select and manipulate bitstrings.

## B. Outcomes

At the end of this lab you should be able to (in C):

- Read from a file using fscanf
- Read and manipulate hex numbers
- Create and use bitmasks.

## C. Study Sample Program

- First of all, study the `Lab03_sample.c` program. It does two things:
  - It uses command line arguments so that you can pass information to the program when you execute it. The main program contains two arguments: `argc` is the number of words on the command line; `argv[0]`, `argv[1]`, .... are strings, namely the words on the command line.
  - E.g., if you call the sample program using `a.out myfile.txt` then `argc` is `2`, `argv[0]` is `"a.out"`, and `argv[1]` is `"myfile.txt"`.
- Second, it opens a file (`mydata.dat`) for input and reads a sequence of decimal numbers from it. It uses `fopen` to open the file, `fscanf` to read the file, and `fclose` to close the file once the end of the data has been reached.
  - `fscanf` is like `scanf` but begins with the `FILE` to read from, then has a format string and the sequence of `&`*variables* to read data into. Note `fscanf` returns the number of items it read; when that number is zero,

[*] No extension for attending lab Sep 18 or 19.

we quit reading. (Either we've hit the end of the file or the file contained something that didn't look like a decimal number.

## *D. Programming Assignment [100 points]*

You are to write a C program for `alpha.cs.iit.edu` that repeatedly reads input from a file and processes it.

1. [5 pts] The file should be specified on the command line as the second word. E.g.,

   ```
   ./a.out myfile.dat
   ```

2. [10 pts] If the filename is not specified, use `default.txt` as the default file. Say what file you're opening (possibly the default) and `fopen` the file.
3. [5 pts] Make sure `fopen` succeeded; if it returns `NULL`, the input file couldn't be opened. In that case, print a message saying so and quit the program using `return 1` . (Returning a non-zero value is the standard way to indicate that a program had an error on Unix-like systems.)
4. [10 pts] Repeatedly read and process three integers (one in hex, two in decimal). (See step 5 below.) Use `fscanf` to read the three integers. If `fscanf` returns $<3$, we're done processing input; go to step 6 below.
5. For discussion purposes, let $X$, $L$, and $R$ be the three values we just read. In the sample output below, $X$ is `0xabcdefab`, $L$ is `6`, and $R$ is `17`.

   ```
   value    0xabcdefab = -1412567125
   mask     0x00007fc0, bits 6:14 = 1be
   bits set 0xabcdffeb
   cleared  0xabcd802b
   flipped  0xabcd906b
   ```

   Let's analyze the output line-by-line.

   a. [5 pts] `value 0xabcdefab = -1412567125` gives $X$ in hex and decimal.

   b. [15 pts] `mask 0x00007fc0, bits 6:14 = 1be` says we want to select bits $X[6:14]$ (i.e., $X[R:L]$). To do this we use the mask `0x00007fc0` (has 1 bits in positions `6 : 14` and 0 bits everywhere else), and we get hex `1be` for the 9 bits selected when we bitwise AND $X$ with the mask.

   c. [5 pts] `bits set 0xabcdffeb` says what get if we set $X[6:14]$ to all `1`s.

   d. [5 pts] `cleared 0xabcd802b` says what get if we set $X[\texttt{6}:\texttt{14}]$ to all `0`s.
   e. [5 pts] `flipped 0xabcd906b` says what get if we flip bits $X[\texttt{6}:\texttt{14}]$. (We flipped bits relative to the original $X$; we're not updating $X$ as we go.)

6. [5 pts] Once you've hit the end of the input, use `fclose` to close the input file. If `fclose` returns 0, the close succeeded; say you've closed the file and quit the program normally (`return 0`). If `fclose` failed, print an error message saying so and quit with error (`return 1`).
7. [5 pts] Your output doesn't have to look exactly like the sample output above, but it should be readable. Don't forget to, print your name.
8. [15 pts] You should comment and indent your program to make it readable and understandable.
9. [10 pts] The general structure of your program should be reasonable. This includes using conditional and loop statements well and avoiding repetitive code.

## *E. Sample Solution*

- I've posted an executable on `alpha`; you'll can run it using the command `~sasaki/Lab03_soln` at the shell prompt. I've also posted a sample data file in `default.txt` but you can (and should) try running it with your own data too.

## *F. What to Submit*

- Just the *.c file, thank you.