Boolean Logic, argc/argv, File I/O, Bit Operations

CS 350: Computer Organization & Assembler Language Programming

Lab 4, due Wed Feb 10

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 and bitstrings

C. This Lab is Important

- The `argc/argv`, file I/O, and bit selection and manipulation operations will come up in the final project, so be sure to master this lab, even if you don't turn it in.

D. Study the Sample Program

- First, study the `Lab4_argv.c` program and the `Lab4_argv_data.txt` file.
- The program expects to be called from the command line with a filename argument.
  - E.g. `./a.out myfile.txt` at the unix prompt.
- Command-line argument information is passed to the main program as an integer `argc` and an array of strings `argv[0]`, `argv[1]`, ..., `argv[argc-1]`.
  - For `./a.out myfile.txt`, we have `argc` = 2, `argv[0]` is `"./a.out"`, and `argv[1]` is `"myfile.txt"`.
- If the program gets no argument or too many arguments, it complains and quits.

- Otherwise, it uses `argv[1]` as the filename to open for read via `fopen`. If opening fails, we complain and quit, otherwise we use `fscanf` in a loop to read a sequence of decimal numbers, printing them as we go. Once the end of the data has been reached, we use `fclose` to close the file .
  - `fscanf` is like `scanf` but begins with the `FILE` to read from, then has the format string and variables to read into.
  - `fscanf` returns the number of items it read. Since we're reading one number at a time, the returned value should be either 1 (we read something) or 0 (we hit the end of the file or the file contained something that didn't look like a decimal number). We keep looping as long as we see 1s from `fscanf`.
- Go ahead and compile and run the sample program on `Lab4_argv_data.txt`. But you should create your own sample data files and run with them until you feel you understand how the program works.

E. Programming Assignment [50 points]

You are to write a C program for `alpha.cs.iit.edu` that repeatedly reads input from a file and processes it. Each input has us break up an integer into three bitstrings and interpret them. For example, an input of `1234abcd 16 4` would produce

```
Value = 0x1234abcd = 305441741
Its leftmost  16 bits are 0x1234 = 4660 as an unsigned integer
Its next       4 bits are 0xa = -5 in 1's complement
Its remaining 12 bits are 0xbcd = -1075 in 2's complement
```

(Note only two bitstring lengths are input; we calculate the third length as what remains after subtracting the first two lengths. The exact form of your output doesn't have to look like the output above, but it should include all of the same information.

1. [3 pts] The file to read from should be specified on the command line as the second word. E.g.,

   ```
   ./a.out somefile.txt
   ```

   If the filename is not specified, use `Lab4_data.txt` as the default file. Say what file you're opening (possibly the default) and `fopen` the file.
2. [2 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.)

3. [4 pts] Use `fscanf` to read three integers (one in hex, two in decimal). If `fscanf` returns < 3, we're done processing input; go to step 10.
4. [2 pts] For discussion purposes, let *val*, *len1*, and *len2* be the three values we just read. Calculate *len3*, the length of the rightmost bitstring, the bits left over after breaking out the leftmost bitstring (of length *len1*) and the middle bitstring (of length *len2*). Since integers are 32 bits long, *len3* = 32 - (*len1* + *len2*).
5. [2 pts] All three lengths should be positive; if not, complain and go onto the next set of data.
6. For purposes of discussion, let's call the three bitstrings to calculate *x1*, *x2*, and *x3* (reading left-to-right). I recommend you calculate them from right to left:
   6.1. [3 pts] Calculate *x3* = the rightmost *len3* bits of *val*.
   6.2. [3 pts] Shift *val* right *len3* bits and calculate *x2* = the rightmost *len2* bits of the resulting string.
   6.3. [3 pts] Shift the string right another *len2* bits and calculate *x1* = the rightmost *len1* bits of the resulting string.
7. [8 pts] Since *x1* is unsigned, we won't have to calculate its decimal value separately (we can just print it twice, once in hex format and once in decimal. But we do need to calculate (let's call them) *dec2* = *x2* in 1's complement and *dec3* = *x3* in 2's complement.*
8. [8 pts] Print out *val* (in hex and decimal), *len1*, *len2*, *len3*, *x1*, *x2*, *x3*, and *x1* (in decimal), *dec2*, and *dec3*. (You figure out a nice format.) Go back to step 4.
9. [2 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 exit the program normally (`return 0`). If `fclose` failed, print an error message saying so and exit with an error (`return 1`).
10. [5 pts] Comment and indent your program to make it readable and understandable.

---

* Hint: To convert a bitstring from unsigned to 2's complement, you can subtract a power of 2. For example, bitstring 1101 = 13 unsigned; in 2's complement, 1101 = –3 = 13 – 2^4 = 13 – 16.

11. [5 pts] The general structure of your program should be reasonable. This includes using conditional and loop statements well and avoiding repetitive code.

F. Skeleton, Sample Data, and Sample Solution

- You can write your program by filling in the stubs in `Lab4_skel.c` and use `Lab4_data.txt` as a sample data file, but you should test your program with your own test data too.
- There's an executable sample solution on `alpha`; you can run it using the command `~sasaki/CS350/Lab4_soln` *dataFileName* at the shell prompt. There's also a copy of the sample data file; you can copy it to your current directory using

  ```
  cp ~sasaki/CS350/Lab4_data.txt .
  ```

  (Be sure to include the period at the end of the command.)
- When you run the solution, your data file should be in your current directory.

G. What to Submit?

- Just the `*.c` file, thank you.

