CS 222: File I/O

Chris Kauffman

Week 5-2

Logistics

Reading

- Ch 8 (pointers)
- Ch 9 (file i/o)

Exam 2

Next Week Thursday

Homework

- HW5 now up
- Multidimensional allocation
- Some file processing

Goals

- HW 5 Overview
- File I/O

HW 5

- Outer product of vectors -> matrix
- Freeing a matrix
- Allocating arrays of `channel_params`
- Counting line lengths in a file

Files: First Guesses

- What's a file?
- How will it be represented?
- How will C communicate with it?
- How do we access that communication mechanism?
- Is the terminal different from file for reading and writing?

Old and New

- `printf` and `scanf`
  - What arguments do they take?
- `fprintf` and `fscanf`
  - How to "get at" them
  - What are their argument?

See `basic_input.c` and `basic_output.c`

Make Me a FILE as Quick as you can

`FILE *fopen(char *fname, char *mode)`

- Opens a file
- `fname` is name of file to open
- `mode` is usually

  `"r"` Read from file, text mode

  `"w"` write to file, text mode

  `"rb"` and `"wb"` read or write binary mode
- Returns a pointer to a `FILE struct`, used with `fscanf/fprintf`

`fclose(FILE *f)`

- Close a file

Standard Input and Standard Output

Original	Equivalent
`printf("Hi\n");`	`fprintf(stdout,"Hi\n");`
`scanf("%s",buf);`	`fscanf(stdin,"%s",buf);`

Can you do less or more with `fprintf/fscanf` compared to `printf/scanf`?

Note on Input Parsing

- `scanf()` and `fscanf()` both ignore whitespace
  - Whitespace includes multiple space, newline, and tab characters
  - Can be a little finicky at times so take care
- Functions see input as one long continuous string
- `fscanf()` moves notion of file position around
- Can reposition with some functions

```
FILE *f = fopen("something.txt","r");

1.23 4 2   \nhello world\t\thow are you?\n\n\nThis is the end\nEOF
012345678901 234567890123 4 5678901234567 8 9 0123456789012345 6
^          1           2            3             4          5
f

double x; int i;  fscanf("%lf %d",&x,&i);

1.23 4 2   \nhello world\t\thow are you?\n\n\nThis is the end\nEOF
012345678901 234567890123 4 5678901234567 8 9 0123456789012345 6
      ^    1           2            3             4          5
      f
```

EOF Character

- A special character which indicates the end of a file.
- Returned by many input functions to indicate no more input
- `scanf/fscanf/getc/getchar` all return at end of input

```
for(status = fscanf(f,"%s",buf);
    status != EOF;
    status = fscanf(f,"%s",buf)){
  printf("Word %2d: %s\n",i,buf);
  i++;
}
```

Modify `show_words_scanf.c`: to use file named on command line

EOF from Wikipedia

> *The actual value of EOF is system-dependent (but is commonly -1, such as in glibc) and is unequal to any valid character code.*
> – *Wikipedia/End-of-file*

Exercise: `sum_file()`

Pseudocode

- Write a `main()` method that accepts command line arguments
- Check that there is at least 1 command line argument
  - If not exit the program
- Open the file named in `argv[1]` for reading
- Read `doubles` until the end of the file
- Close the file
- Print the sum
- Note: no need for arrays here

Example Use

```
$> gcc -o sum_file sum_file.c
$> ./sum_file
usage: sum_file filename
$> ./sum_file file1.dat
file file1.dat sums to +1.00
$> ./sum_file file2.dat
file file2.dat sums to +1.00
$> ./sum_file file3.dat
file file3.dat sums to +0.00
$> ./sum_file file4.dat
file file4.dat sums to +20.00
```

A few more goodies

Only `stdin/stdout`

`int getchar()` read single character

`int putchar(char c)` print single character

`char *gets(char *buf)` read whole line (DANGEROUS)

Any file

`int getc(FILE *f)` read single character

`int putchar(FILE *f)` print single character

`char *fgets(char *buf, int n, FILE *f)` read whole line, up to `n` characters

`int fgetc(FILE *f)` analagous as `getc`

Exercise: Simple counts

The unix utility `wc` reports how many characters, lines, and words are in a file.

```
lila [w05-2-code]% wc file1.dat
 3  3 27 file1.dat
# 3 lines, 3 words, 27 characters

lila [w05-2-code]% wc parrot.c
 10  31 193 parrot.c
# 10 lines, 31 words, 193 characters
```

- Write a `main()` which counts
  - How many characters are in a file
  - How many lines are in a file
- `char input = fgetc();` makes this easy
- `input` will be `\n` on line breaks
- `input` will be `EOF` when file ends

Reminder: `FILE` is a struct like any other

- `fopen()` returns a pointer to one
- Fields are system dependent (different between Mac/Windows/Linux)
- How could I ask how big a `FILE` struct is?

See `file_struct_size.c`

Note on Output Buffering

Operating systems try to optimize I/O operations

- Data doesn't get pushed to disk right away
- Guaranteed when `fclose` is called
- See `buffering.c`
- Other ways to force writing (`fflush`)

Sample: Write A Range

See `range_cmdline.c`

- Write a range of numbers to the screen
- Adapt this to write range to file
- File is first arg on command line
- 2nd arg is first number
- 3rd arg is last number

```
> gcc range_cmdline_file.c
> a.out myfile.txt 3 7
> cat myfile.txt
3
4
5
6
7
> a.out otherfile.txt 8 10
> cat otherfile
8
9
10
```

Optional Exercise: Parrot

Define a parrot

- Whatever you type it spits back

Define a file copy

- Takes 2 command line arguments
- Copies contents of named file arg1 to named file arg2