CSCI 2132: Software Development

File Manipulation in C

Norbert Zeh

Faculty of Computer Science
Dalhousie University

Winter 2019

Files and Streams

C’s view of files mirrors Unix’s: Files are streams of bytes

File operations manipulate streams of bytes

Standard streams: `stdin`, `stdout`, `stderr`

Example:

- `printf` prints to `stdout`, `fprintf` prints to a file
- The following are equivalent

```
printf(“Hello, world!”);
```

```
fprintf(stdout, “Hello, world!”);
```

File Pointers

In C, files are accessed through file pointer or file descriptors:

File Pointers

In C, files are accessed through file pointer or file descriptors:

File descriptor: Low-level Unix identifier for a file

- Used with system calls:
  `open`, `close`, `read`, `write`, ...
- No buffering

File Pointers

In C, files are accessed through file pointer or file descriptors:

File descriptor: Low-level Unix identifier for a file

- Used with system calls:
  `open`, `close`, `read`, `write`, ...
- No buffering

File pointer: C library construct that wraps a file descriptor

- Used with C library functions:
  `fopen`, `fclose`, `fread`, `fwrite`, ...
- Buffering

File Pointers

In C, files are accessed through file pointer or file descriptors:

File descriptor: Low-level Unix identifier for a file

- Used with system calls:
  `open`, `close`, `read`, `write`, ...
- No buffering

File pointer: C library construct that wraps a file descriptor

- Used with C library functions:
  `fopen`, `fclose`, `fread`, `fwrite`, ...
- Buffering

You almost always want to use file pointers!

File Types

Text files:

- Newline characters may be treated specially
- May have special marker byte at the end

Binary files:

- Raw access to bytes in the file

The difference is mostly in how we access the file:

- `fread`, `fwrite`: Raw byte access
- `fscanf`, `fprintf`, `getline`: Interpret file contents as text

Opening Files

```
FILE *fopen(const char *filename, const char *mode);
```

Modes:

"r":	Read
"w":	Write (Overwrite if exists, create if not)
"a":	Append
"r+":	Read and write, start at beginning
"w+":	Read and write, delete old content
"a+":	Read and write, write at end position
" ... b":	Open binary file (ignored on Linux and BSD)

Return value: file pointer or NULL if unsuccessful

Closing a File

```c
int fclose(FILE *file);
```

Return value:

`0`	on success
`EOF`	otherwise

Formatted I/O with Files

```
int fprintf(FILE *stream, const char *format, ...);
int fscanf (FILE *stream, const char *format, ...);
```

```
printf( ... ) = fprintf(stdout, ... )
scanf ( ... ) = fscanf (stdin,  ... )
```

Print error message: `fprintf(stderr, ... )`

Example

```
#include <stdio.h>

int main() {
    FILE *stream;
    stream = fopen(“hello.txt”, “w”);
    if (!stream) {
        fprintf(stderr, “Cannot open hello.txt\n”);
        exit(EXIT_FAILURE);
    }
    fprintf(stream, “Hello, world!\n”);
    fclose(stream);
    return 0;
}
```

Character I/O

```
int putc (int c, FILE *stream);
int fputc(int c, FILE *stream);

int getc(FILE *stream);
int fgetc(FILE *stream);
```

`getc` and `putc` may be macros
(Do not use `getc(fopen("file.txt", "r"))`)

`putchar( … )` = `putc( … , stdout)`

`getchar( … )` = `getc( … , stdin)`

Reading and Writing Blocks of Data

```
fread(void *restrict ptr,
      size_t element_size, size_t nitems,
      FILE *restrict stream);

fwrite(const void *restrict ptr,
       size_t element_size, size_t nitems,
       FILE *restrict stream);
```

Checking for End of File

```
int feof(FILE *stream);
```

Return value:

- “True” ($\neq$ `0`) if at end of file
- “False” ($==$ `0`) if not at end of file

File Positioning

Reset file position to beginning of file:

```
void rewind(FILE *stream);
```

Get and set the file position:

```
long int ftell(FILE *stream);
int fseek(FILE *stream, long int offset,
          int whence);
```

Does not work for very large files (beyond `long int` capacity).

Values for whence:

- `SEEK_SET` relative to beginning of file (absolute positioning)
- `SEEK_END` relative to end of file
- `SEEK_CUR` relative to current position (relative positioning)

File Positioning

```
int fgetpos(FILE *restrict stream,
            fpos_t *restrict pos);
int fsetpos(FILE *stream, const fpos_t *pos);
```

- Similar to `ftell` and `fseek`
- Position information stored in an opaque object
- Can handle arbitrary file sizes

An Example

```c
#include <stdio.h>

struct point { int x, y; };

int main() {
    struct point p = { 1, 2 };
    FILE *f = fopen("tmp.txt", "w+");
    fwrite(&p, sizeof(struct point), 1, f);
    fseek(f, (char *) &p.y - (char *) &p, SEEK_SET);
    fread(&p.x, sizeof(int), 1, f);
    rewind(f);
    fread(&p.y, sizeof(int), 1, f);
    printf("(%d, %d)\n", p.x, p.y);
    return 0;
}
```