# File I/O in C

Adapted from materials by Dr. Carrier

Behnam Norouzi - Unsplash

# Overview

Files are treated like streams

# Overview

Files are treated like streams

Files are managed via **file pointers**

# Overview

Files are treated like streams

Files are managed via **file pointers**

All file functions are in `<stdio.h>`

# Overview

Files are treated like streams

Files are managed via **file pointers**

All file functions are in `<stdio.h>`

Generally, the process looks like this:

1. Create file pointer

# Overview

Files are treated like streams

Files are managed via **file pointers**

All file functions are in `<stdio.h>`

Generally, the process looks like this:

1. Create file pointer
2. Open file in the correct mode

# Overview

Files are treated like streams

Files are managed via **file pointers**

All file functions are in `<stdio.h>`

Generally, the process looks like this:

1. Create file pointer
2. Open file in the correct mode
3. Read and/or edit the file

# Overview

Files are treated like streams

Files are managed via **file pointers**

All file functions are in `<stdio.h>`

Generally, the process looks like this:

1. Create file pointer
2. Open file in the correct mode
3. Read and/or edit the file
4. Close the file

# Opening a file

```
FILE* fp_in;
fp_in fopen(“in_filename”, “r”);
// Read file
fclose(fp_in);
```

# Opening a file

```
FILE* fp_in;
fp_in fopen(“in_filename”, “r”);
// Read file
fclose(fp_in);

FILE* fp_out;
fp_out = fopen(“out_filename”, “w”);
// Write to file
fclose(fp_out);
```

# fopen

```
FILE* fp = fopen(filename, mode);
```

# fopen

```
FILE* fp = fopen(filename, mode);
```

Modes:

- “r” to read
- “w” to write (discards old contents)
- “a” to append

# fopen

```
FILE* fp = fopen(filename, mode);
```

Modes:

- “r” to read
- “w” to write (discards old contents)
- “a” to append

Can also add “+” to both read/write (docs)

# fopen

```
FILE* fp = fopen(filename, mode);
```

Modes:

- “r” to read
- “w” to write (discards old contents)
- “a” to append

Can also add “+” to both read/write (docs)

You *can* add “b” to the end for binary mode
This does not matter on Unix systems

# fopen

| Mode | File exists? | File doesn’t exist? |
|---|---|---|
| read | Open to read | Error |
| write | Overwrite it | Create it |
| append | Append to it | Create it |

# fopen

| Mode | File exists? | File doesn't exist? |
|---|---|---|
| read | Open to read | Error |
| write | Overwrite it | Create it |
| append | Append to it | Create it |

fopen returns NULL on failure, good to check!

# fopen

| Mode | File exists? | File doesn't exist? |
|---|---|---|
| read | Open to read | Error |
| write | Overwrite it | Create it |
| append | Append to it | Create it |

fopen returns NULL on failure, good to check!

When can it fail?

# fopen

| Mode | File exists? | File doesn’t exist? |
|---|---|---|
| read | Open to read | Error |
| write | Overwrite it | Create it |
| append | Append to it | Create it |

fopen returns NULL on failure, good to check!

When can it fail?

- Reading a file that doesn’t exist

# fopen

| Mode | File exists? | File doesn’t exist? |
|---|---|---|
| read | Open to read | Error |
| write | Overwrite it | Create it |
| append | Append to it | Create it |

fopen returns NULL on failure, good to check!

When can it fail?

- Reading a file that doesn’t exist
- Incorrect permissions

# fopen

| Mode | File exists? | File doesn't exist? |
|---|---|---|
| read | Open to read | Error |
| write | Overwrite it | Create it |
| append | Append to it | Create it |

fopen returns NULL on failure, good to check!

When can it fail?

- Reading a file that doesn't exist
- Incorrect permissions

```
FILE* fp = fopen("file.txt", "r");
if(fp == NULL){
    printf("Error! Could not open file\n");
    return 1;
}
```

# fclose

```
fclose(file_pointer);
```

# fclose

```
fclose(file_pointer);
```

Closing your files is important!

File output is often buffered – changes are only written to file when its closed!

# fclose

```
fclose(file_pointer);
```

Closing your files is important!

File output is often buffered – changes are only written to file when its closed!

Even if your system doesn't buffer, we want to close files to make our code portable!

# Reading from a file

*Very* similar to reading input from stdin! (lecture)

# Reading from a file

*Very* similar to reading input from stdin! (lecture)

`scanf` switches to `fscanf`

`fgets` and `getline` work out of the box!

# Reading from a file

*Very* similar to reading input from stdin! (lecture)

`scanf` switches to `fscanf`
`fgets` and `getline` work out of the box!

```
fscanf(fileptr, formatstr, memaddr1, …);
```

# Reading from a file

*Very* similar to reading input from stdin! (lecture)

`scanf` switches to `fscanf`
`fgets and getline` work out of the box!

```
fscanf(fileptr, formatstr, memaddr1, …);
```

```
fgets(char* s, int size, FILE* stream);
```

# Reading from a file

*Very* similar to reading input from stdin! (lecture)

`scanf` switches to `fscanf`
`fgets` and `getline` work out of the box!

```
fscanf(fileptr, formatstr, memaddr1, …);

fgets(char* s, int size, FILE* stream);

getline(char ** lineptr, size_t n,
   FILE* stream);
```

# Writing to a file

```
fprintf(FILE* fp, format_str, args…);
```

Identical to printf, but writes to a file, not stdout

Don’t forget the first `f`!

# Writing to a file

`fprintf(FILE* fp, format_str, args…);`

Identical to printf, but writes to a file, not stdout

Don’t forget the first `f`!

`fputc(int c, FILE* fp);`

Writes a single character to file

there’s also a fgetc)

# Writing to a file

`fprintf(FILE* fp, format_str, args…);`

Identical to printf, but writes to a file, not stdout

Don’t forget the first `f`!

`fputc(int c, FILE* fp);`

Writes a single character to file

there’s also a fgetc)

`fputs(char* s, FILE* fp);`

Writes the string to file

# Working with bytes

```
fread(void* buffer, size_t size,
  size_t n, FILE* fp);
```

- Reads n objects (which are `size` bytes each) from file and stores data in `buffer`.

# Working with bytes

```
fread(void* buffer, size_t size,
   size_t n, FILE* fp);
```

- Reads n objects (which are `size` bytes each) from file and stores data in `buffer`.

```
fwrite(void* buffer, size_t size,
   size_t n, FILE* fp);
```

- Writes n objects (which are `size` bytes each) from buffer to `fp`

# Working with bytes

```
fseek(FILE* fp, long offset,
   int origin);
```

- Think of your cursor in a text file, next read/write will occur here

# Working with bytes

```
fseek(FILE* fp, long offset,
  int origin);
```

- Think of your cursor in a text file, next read/write will occur here
- This also applies to reading/writing files in C
- `fseek` will move this cursor (position indicator)

# Working with bytes

```
fseek(FILE* fp, long offset,
  int origin);
```

- Think of your cursor in a text file, next read/write will occur here
- This also applies to reading/writing files in C
- `fseek` will move this cursor (position indicator)

Options for origin:

- `SEEK_SET` - start of file
- `SEEK_CUR` - current position
- `SEEK_END` - end of file

# Final thoughts

Pay attention to return values!!

- e.g., input functions will treat errors and EOF differently
- This will save you future headaches

# Final thoughts

Pay attention to return values!!

- e.g., input functions will treat errors and EOF differently
- This will save you future headaches

If you are switch between reading and writing in the same file pointer

- Use `fseek()` or `fflush()` before switching
- This forces buffer to be flushed