COMP1521

WEEK 8

CONTENT

Announcements

- Lab start this week!
  - Lab8 has been released, due on **Week 9 Monday 12:00:00 (midday)**

- weekly quiz 7 be **due Week 8 Thursday 21:00:00**
- **Assignment2** will due on **Week 10 Friday 18:00:00**

Style

Computer Systems Fundamentals

Course Resources

Tutors: GitHub

Administrivia: Course Outline | COMP1521 Handbook

Administrivia: Course Timetable | Help Sessions

Meet the Team: Our Team

Platforms: Lecture Recordings | Online Tut-Labs and Help Sessions (via BbCollaborate) | Course Forum

Style Guides: COMP1521 C Style Guide | Assembly Style Guide

MIPS Resources: MIPS Documentation | Text Editors for Assembly

mipsy: mipsy-web | mipsy source code | Debugging with mipsy (video)

Revision: Linux Cheatsheet | C Reference

Assessment: Autotests, Submissions, Marks | Give online: submission | Give online: sturec

Assignments: Assignment 1 | Assignment 2

Style

Rule | Example

Header Comment

All program should have a header comment.

For small lab exercises the header comment should contain at least:

- The names of the author(s) - include your UNSW zID with any code you submit at UNSW.
- The date it was written.
- A description of what the program does.

For larger programs such as assignments, also include:

- An overview of how the program works, including any bugs or limitations.
- Any other information that might help a reader understand the code.
- References to resources used in constructing the code.

It is not necessary to include licensing information, but it is commonly present in header comments of code outside uni.

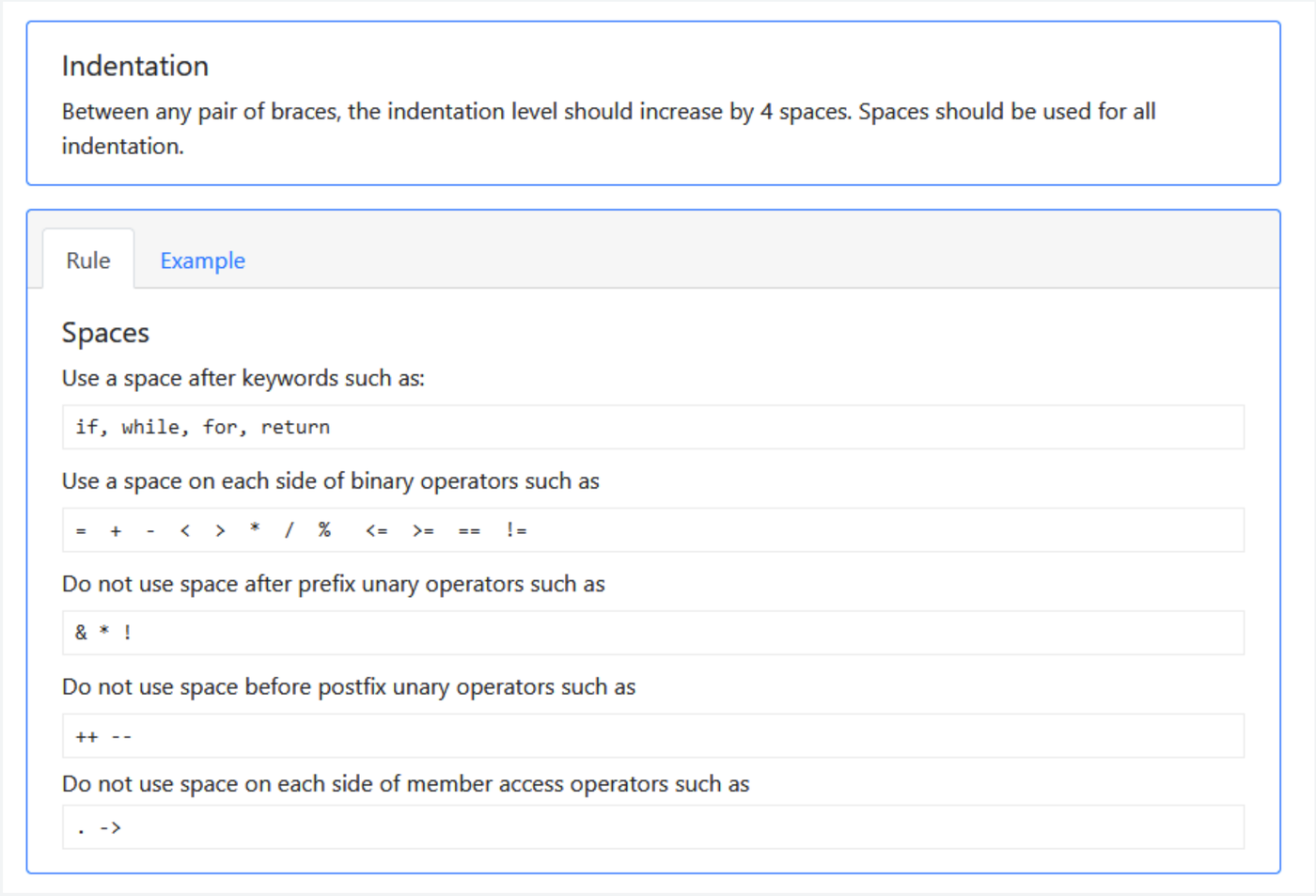
```
//
// COMP1511 Lab 10 Exercise - Turing's test
//
// Pretend to be a human in text-based conversation.
// https://en.wikipedia.org/wiki/Turing_test
//
// Authors:
// Grace Hopper (z1234567@unsw.edu.au)
// Ada Lovelace (z5000000@unsw.edu.au)
//
//
//
// Written: 19/03/2019
//

#include <stdio.h>
#include <string.h>
#include <math.h>

#define MEANING_OF_LIFE 42
```

FILL IN HEADER COMMENT!

Style



Indentation

Between any pair of braces, the indentation level should increase by 4 spaces. Spaces should be used for all indentation.

Rule | Example

Spaces

Use a space after keywords such as:

```
if, while, for, return
```

Use a space on each side of binary operators such as

```
=  +  -  <  >  *  /  %   <=  >=  ==  !=
```

Do not use space after prefix unary operators such as

```
& * !
```

Do not use space before postfix unary operators such as

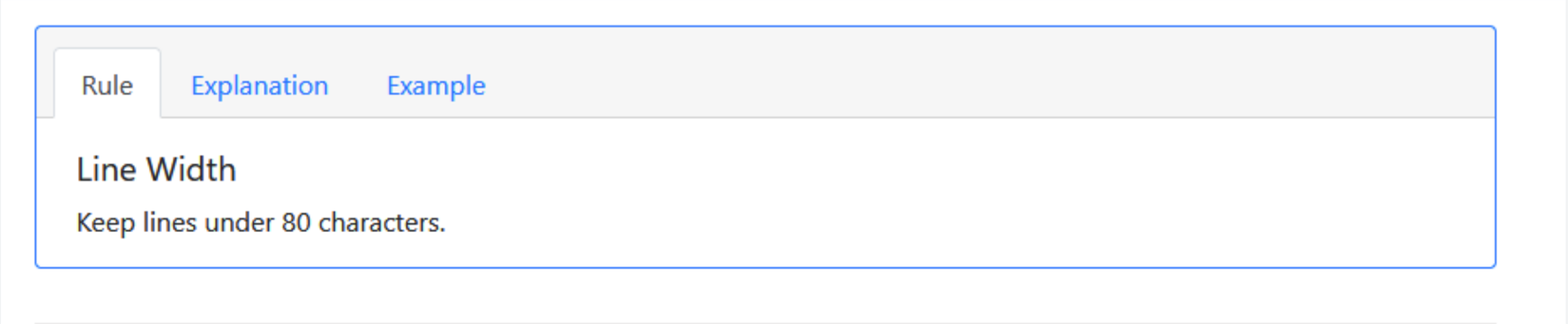
```
++ --
```

Do not use space on each side of member access operators such as

```
. ->
```

keep indentation and spacing consistent

Style


Rule
Explanation
Example
Line Width
Keep lines under 80 characters.

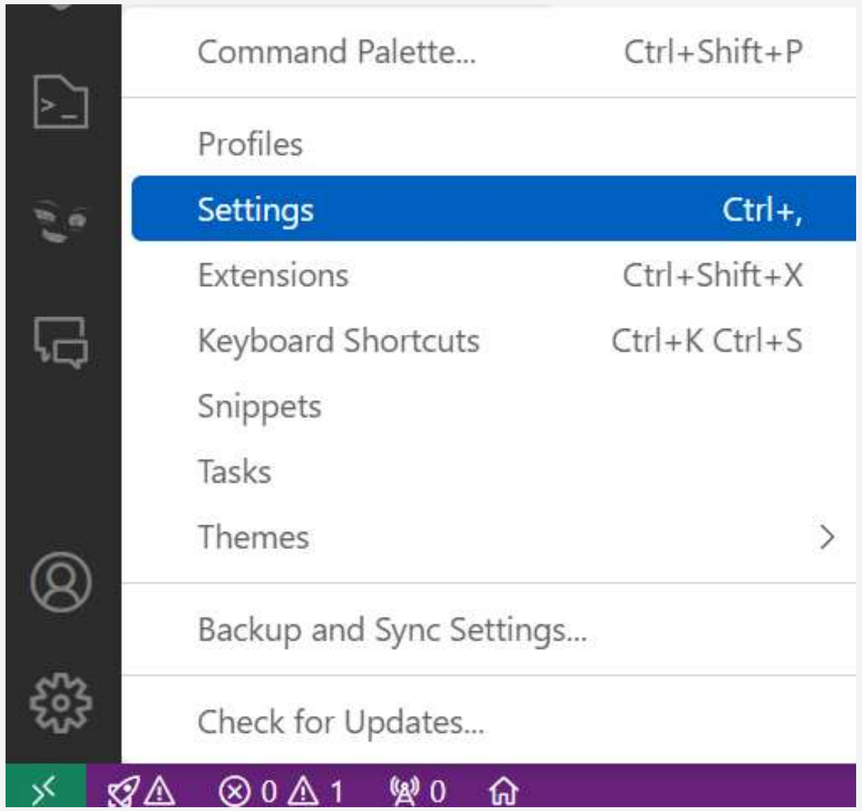
Style

Rule	Explanation	Example

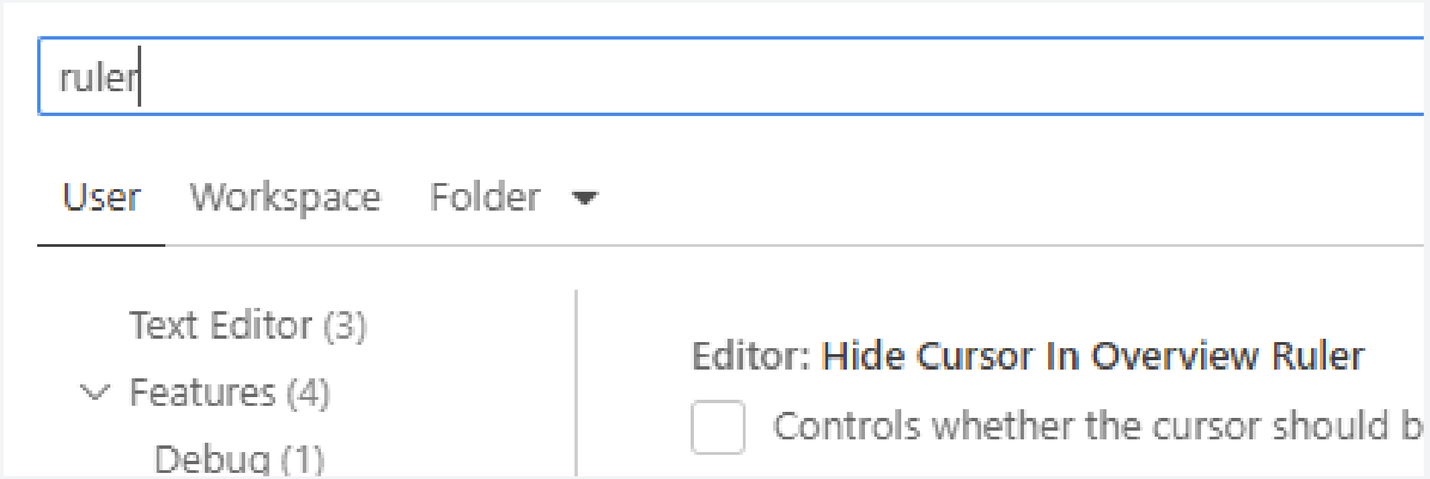
Line Width

Keep lines under 80 characters.

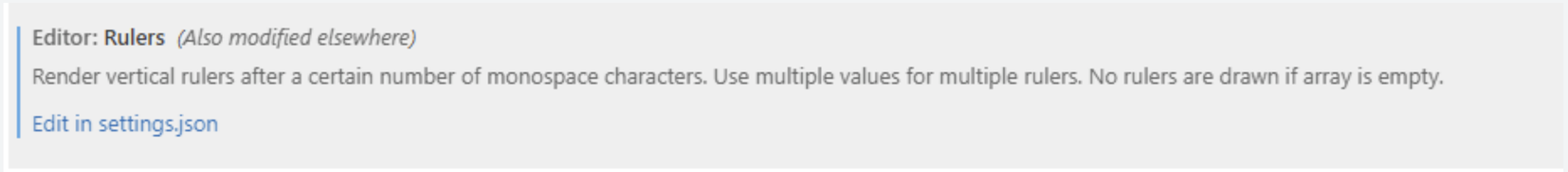
to get line ruler:



2.



3
.



4. click on edit in settings.json

copy this into the end;

,"[c]": {
"editor.rulers": [80],
},

now you have a 80 line ruler for only c!

mport > glass > 4 > z5360323 > COMP1521-2313 > tutorial-code > week7 > q4.c

```
0 00000000 00000000000000000000000

sign: (-1 * 0) * frac: (1.0000000000) * exp: (2^0 - 127)
1 * 1* 2^-127
2^-127 (which is almost 0.)


1 00000000 00000000000000000000000
sign: (-1 * 1)* frac: (1.0000000000) * exp: (2^0 - 127)
-1 * 1 * 2^-127
-2^-127


0 01111111 10000000000000000000000
01111111 = 10000000 -1 = 128 - 1 = 127
1*2^-1 = 1/2 = 0.5
sign: (-1 * 0) * frac: (1.5) * exp: 2^(127-127)
1 * 1.5 * 1 = 1.5


0 01111110 00000000000000000000000
0 01111110 11111111111111111111111

0.1111111111
0.5 + 0.25 + 0.125 + ..... = 0.99999999

sign: (-1 * 0) * frac: (1.99999) * exp( 2^(126 - 127))
1.9999*2*-1
1/1.9999*2


0 10000000 01100000000000000000000
0 10010100 10000000000000000000000
0 01101110 10100000101000001010000
```

Functions

Rule | Explanation

Repeated Code

Do not cut-and-paste code. The same code should not appear twice in your program. Avoid repeating code by defining a function and calling it twice (or more).

Rule | Example

Function Comments

Every function must have a comment placed before the *function implementation* describing the purpose of the function and any *side-effects* the function has.

Common Linux File Types (Basic Three):

Symbol	Type Name	Meaning	Description
-	Regular File	Regular file	Most common type: text files, images, executables, binaries
d	Directory	Directory	A container that holds other files and directories
l	Symbolic Link	Symbolic (soft) link	A reference (alias) to another file path, like a shortcut

Files Overview:

Most common functions:

- fopen, fclose - open and close a file
- fgetc/getchar, fputc/putchar - read or write 1 B
- fread, fwrite - read or write multiple bytes
- fprintf/fscanf - formatted print/scan to targeted destination
- fseek - move the file pointer to a specified spot.

Protip: read the manual for each function very carefully as the details really matter a lot for functions. - students spend hours debugging just to realise they missed a minor detail in how the function works.

Files Overview:

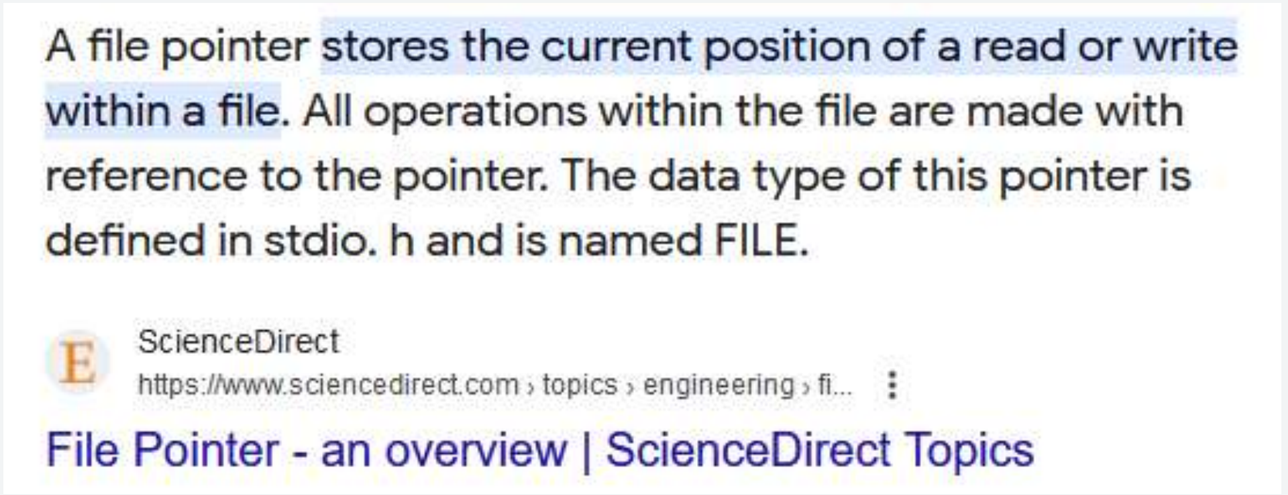
Discussion: what is a FILE pointer? (FILE *)

```
FILE *stream = fopen(“hello.txt”, “r”);
```

Files Overview:

Discussion: what is a FILE pointer? (FILE *)

what it does:



what it is:

A file pointer is a variable that is used to refer to an opened file in a C program. The file pointer is actually a structure that stores the file data such as the file name, its location, mode, and the current position in the file. It is used in almost all the file operations in C such as opening, closing, reading, writing, etc.

https://www.geeksforgeeks.org/c-file-pointer/

Files Overview:

Discussion: what is a FILE pointer? (FILE *)

how to use?

Fopen creates the FILE pointer

Fseek moves the pointer to precise locations in the FILE

fgetc gets the character at the FILE pointer, then moves the FILE pointer to the next byte.

Files Overview:

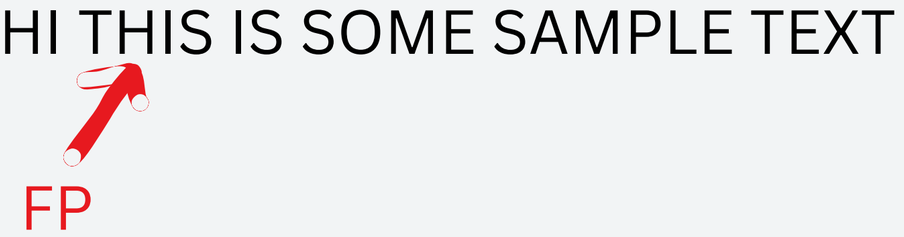
Discussion: what is a FILE pointer? (FILE *)

how to use?

Fopen creates the FILE pointer

Fseek moves the pointer to precise locations in the FILE

fgetc gets the character at the FILE pointer, then moves the FILE pointer to the next byte.



Files Overview:

Discussion: what is a FILE pointer? (FILE *)

how to use?

Fopen creates the FILE pointer

Fseek moves the pointer to precise locations in the FILE

fgetc gets the character at the FILE pointer, then moves the FILE pointer to the next byte.

HI THIS IS SOME SAMPLE TEXT

FP

txt = fgetc(stream)

Files Overview:

Discussion: what is a FILE pointer? (FILE *)

how to use?

Fopen creates the FILE pointer

Fseek moves the pointer to precise locations in the FILE

fgetc gets the character at the FILE pointer, then moves the FILE pointer to the next byte.

HI THIS IS SOME SAMPLE TEXT

FP

txt = fgetc(stream)

txt = s

Files Overview:

Discussion: what is a FILE pointer? (FILE *)

how to use?

Fopen creates the FILE pointer

Fseek moves the pointer to precise locations in the FILE

fgetc gets the character at the FILE pointer, then moves the FILE pointer to the next byte.

HI THIS IS SOME SAMPLE TEXT

FP

txt = fgetc(stream)

txt = ‘ ‘

Files Overview:

Discussion: what is a FILE pointer? (FILE *)

how to use?

Fopen creates the FILE pointer

Fseek moves the pointer to precise locations in the FILE

fgetc gets the character at the FILE pointer, then moves the FILE pointer to the next byte.

HI THIS IS SOME SAMPLE TEXT

FP

txt = fgetc(stream)

txt = I

Files Overview:

Discussion: what is a FILE pointer? (FILE *)

how to use?

Fopen creates the FILE pointer

Fseek moves the pointer to precise locations in the FILE

fgetc gets the character at the FILE pointer, then moves the FILE pointer to the next byte.

fclose then closes and destroys the FILE pointer and frees its resources.

Files Overview:

Fopen

Fseek: move the FP to the correct place

File operation: fgetc/fread/fwrite/fputs

Fclose.

protip:

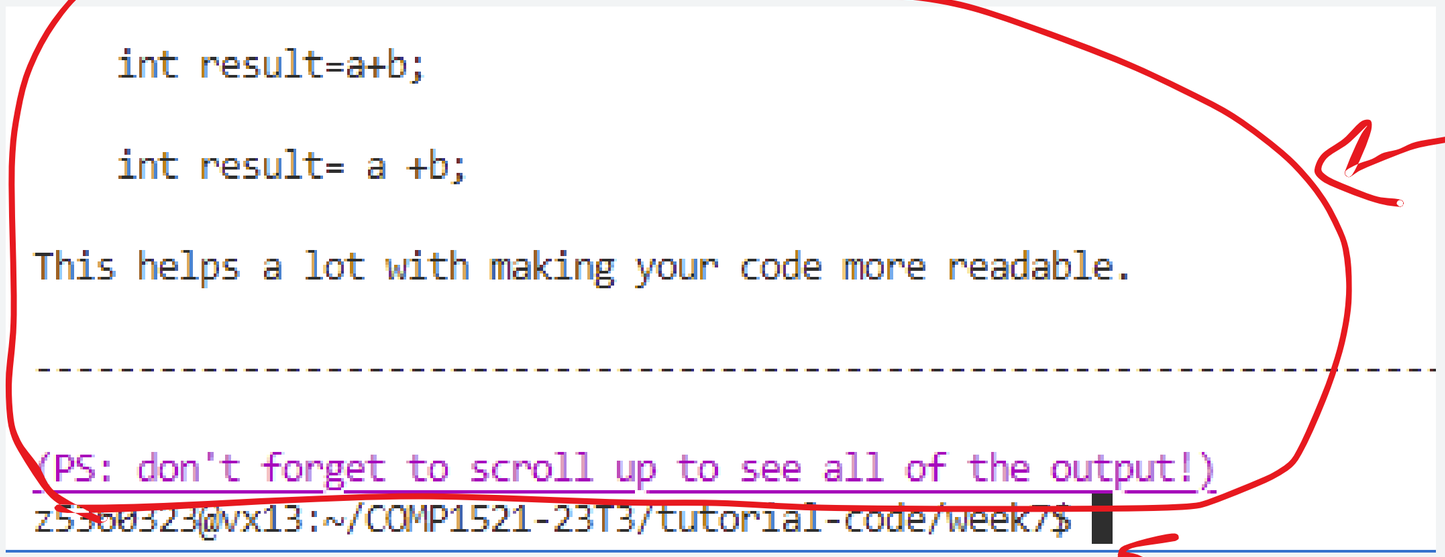
ALWAYS keep track of where your file pointer is. Use comments to help you keep track of it!.

70% of files is manipulating and keeping track of exactly where your FP is pointing to.

Files Overview:

Special Files:

stdin and stdout are what we are used to as terminals.



STDOUT

STDIN (inputs get put into a buffer which is where STDIN is read from)

Files Overview:

Special Files:

stdin and stdout are what we are used to as terminals.

```
fprintf(stdout, “i am %d years old”, 20);
```

is equivalent to

```
printf(“i am %d years old”, 20);
```

Endianness: Little vs Big

	Little-endian	**Big-endian**
Order	Least byte first	Most byte first
Example	0x12345678 → 78 56 34 12	0x12345678 → 12 34 56 78
Used in	x86, ARM (default)	Network protocols
Memory	Low address = LSB	Low address = MSB

Endianness: Little vs Big

	Little-endian	**Big-endian**
Order	Least byte first	Most byte first
Example	0x12345678 → 78 56 34 12	0x12345678 → 12 34 56 78
Used in	x86, ARM (default)	Network protocols
Memory	Low address = LSB	Low address = MSB

Value: 0x12345678

Address → 0x00 0x01 0x02 0x03

Little → 0x78 0x56 0x34 0x12
Big → 0x12 0x34 0x56 0x78

Q1 a, b, g

1. read from a file alphabets.txt and swap the first 13 letters with the last 13 letters.

2. q6

Week 9 → stats.h 12)

tute q1 answers

ANSWER:

a. COMP1521's `web` directory is `/home/cs1521/web`

b. `~jas` is shorthand for `/home/jas`, so `~jas/../..` is the root directory (`/`)

c. The link to the parent directory is also stored as an entry in the directory structure, called `..`

d. `cat` is a regular file, which happens to contain executable machine code.

e. `home` is a directory.

f. `tty0` is a *character special file*, a file that represents a *device* which can read and write a byte-stream, which is typically interpreted as characters.

g. A symbolic link is a special kind of file that simply contains the name of another file.

h. Symbolic links produce a graph because they allow arbitrary links between filesystem objects. Without symlinks, the only "connections" in the filesystem are parent/child links, which produce a tree.