

1. How does the C library wrap a filedescriptor?

```

01 // How the C library wraps a filedescriptor
02 typedef struct _FILE { /* Simplified!*/
03     int fd;
04     void* buffer; // reduce # of write() calls
05     size_t capacity;
06     size_t size;
07     int buffering; // _IONBF / _IOLBF / _IOFBUF
08 } FILE;
09
10 FILE* fdopen(int fd, char* rmode) {
11     ?
12
13
14
15
16 }
17 void fputs(char*str, FILE* file) {
18     ?
19
20
21
22
23
24
25
26
27
28
29
30
31
32

```

fprintf(FILE*, format,...)

uses write (buffering reduces number of writes => performance)

fseek(FILE*, offset, whence) SEEK_CUR | SEEK_SET | SEEK_END

uses lseek(int fd, off_t offset, int whence);

long pos=ftell(FILE*) uses return lseek(fd, 0, SEEK_CUR)**2. Challenge: Implement C library function rewind(FILE*)**

Hint: You will need fflush and lseek or fseek (that will flush for you)

```

01 // Use the struct above to extract the filedes
02 void rewind(FILE* f) {
03
04
05 }

```

3. Reading & Writing binary data**fread** (void * ptr, size_t size, size_t nitems, FILE * stream);**fwrite** (void * ptr, size_t size, size_t nitems, FILE * stream);

```

01 int num_pts;
02 typedef struct { float x,y,z } p_t;
03 p_t* points;
04
05 void load_point_cloud() {
06     FILE* f = fopen("points.dat","r");
07     fread( &num_pts , _____ , 1, f);
08     points = calloc( sizeof p_t, num_pts);
09     ?
10

```

Error handling/What could go wrong? Why #include<stdint.h> and using uint32_t be better?

4. Challenge: Read in the first half of a file as C string

Hints: fopen, fseek, ftell, fread, malloc, fclose may be useful

```

01 char* half(char*filename) {
02
03
04
05
06
07
08

```

5. Implement `fflush`

Hint: Use & reset the FILE's output buffer, `write` will be useful

```
01 void fflush(FILE*f) {  
02  
03  
04
```

6. Amdahl's law.

With a single core it takes 100 milliseconds to calculate and render my VR graphics (ie. 10FPS). 15% of that time is spent inside `read()` & `write`, and 10% inside unmodifiable library code and the rest inside some embarrassingly-parallel code that I can improve to be multi-threaded.

If I can use 3 cores for graphics rendering can I achieve 20FPS?

7. Pipes Putting it all together

Write a complete program to perform the following. The parent process will copy the contents (4KB at a time) of a file 'input.txt' into stdin of the child process which exec's a bash shell

Assume *read* and *write* always complete. `dup2` may be useful.