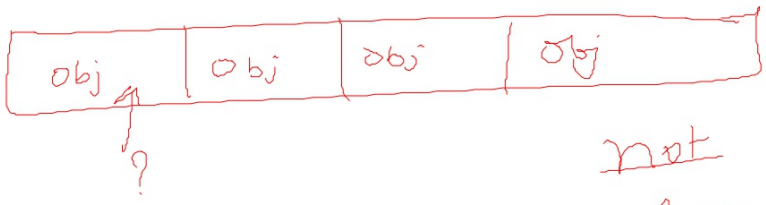
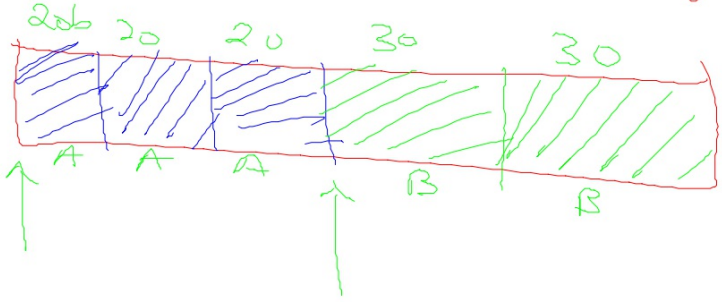
Binary I/O

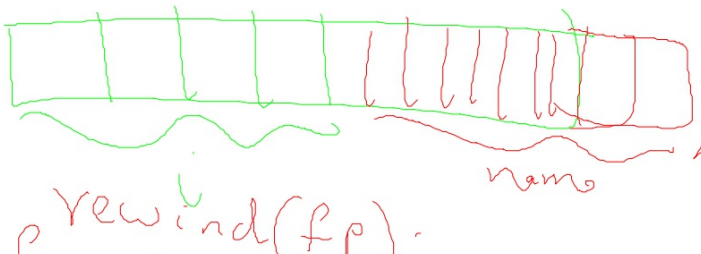
Binary file



not equi-sized objects



```
int i = 10;        char name[10];
int j;
// fopen
:
fwrite(&i, sizeof(int), 1, fp);
fwrite(name, sizeof(char), 10, fp);
```




```
rewind(fp);
fread(&j, sizeof(int), 1, fp);
```

SEEK_END SEEK_CUR

```
fseek(fp, 4, SEEK_SET);
                "whence"
fread(name, sizeof(char), 10, fp);
```

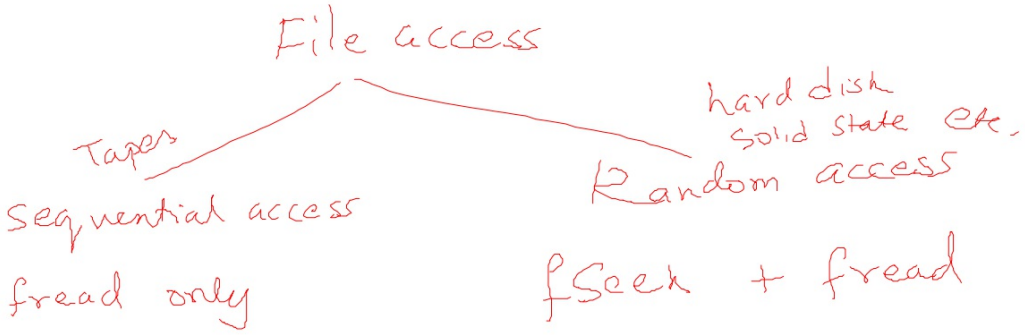
```
struct Vfs_File {
    int file_id;
    int file_size;
};
```

4 == MAX_FILES

```
struct Vfs_File all_files[MAX_FILES];
int num_files;
```


```
fwrite( &numfiles . . . . );
fwrite( all_files, sizeof(struct Vfs_file), num_files, fp);
```

```
struct Vfs_File *files;
int nfiles;

fp = fopen( ... )

fread( &nfiles, ... );

files = (struct Vfs_file *)
        malloc(sizeof(struct Vfs_File)
               *nfiles);

fread(files, sizeof(struct Vfs_File),
      nfiles, fp);
```

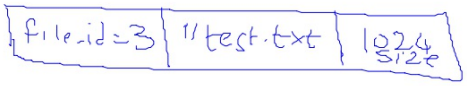

File access
Tapes
Sequential access
fread only
hard disk
solid state etc.
Random access
fSeek + fread

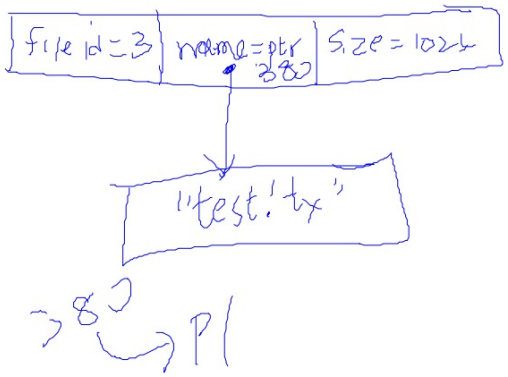
```
#define MAX_LEN 10
struct Vfs_File {
    int file_id;
    char name[MAX_LEN];
    int size;
};
```

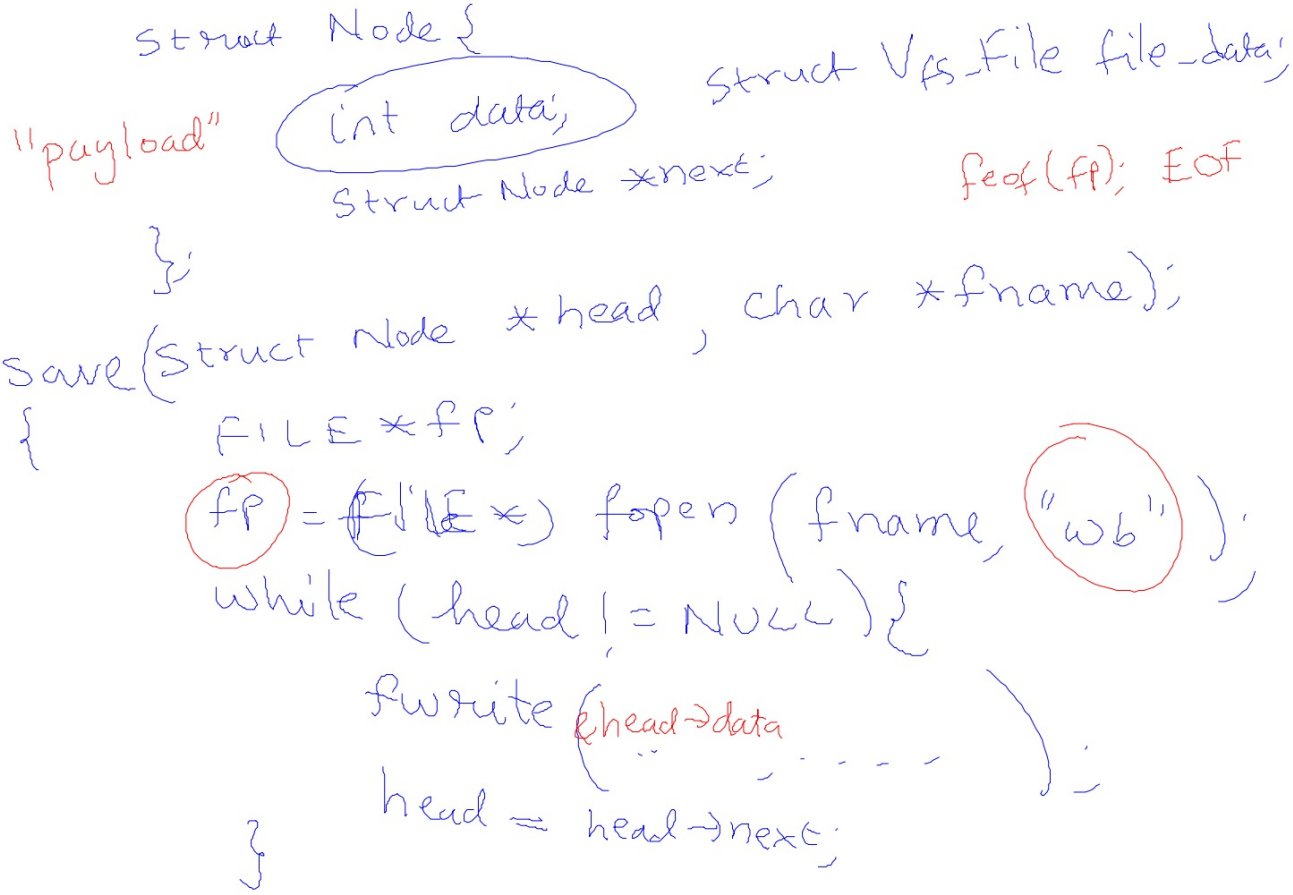
sizeof(Vfs_file) == 18

```
struct Vfs_File {
    int file_id;
    Char *name;
    int size;
};
```

size == 12






Struct Node {
int data;
Struct Node *next;
};
"payload"
Struct Vfs_File file_data;
feof(fp); EOF
Save(Struct Node *head, char *fname);
{
FILE *fp;
fp = fopen(fname, "wb");
while (head != NULL){
fwrite(&head->data ...);
head = head->next;
}

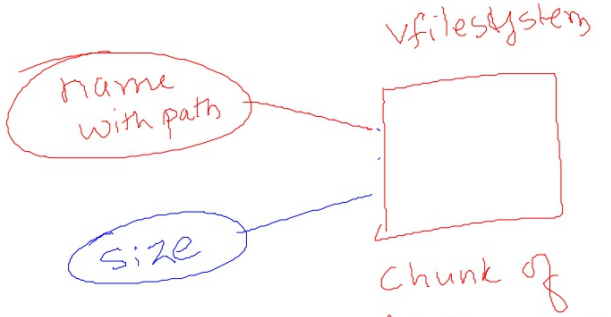
```
while ( fread( . . . ) != EOF )

while ( !feof(fp) ) }

    :

}
```

1. fclose( ) must always be called on an open file

2) Read and write possible in same process depending upon mode

3) Call fclose(fp) ONLY ONCE per file

Version 1



1. Create a file system
   creat_vfs executable

---

2. a) load vfs (open)
   b) Save file into vfs
   Save_vfs executable

---

3. a) load vfs
   b) read file from vfs

extract_vfs

1) vfs.h

2) vfs_utils.c

```
int load_vfs(char *name_with_path);
```

3) save_vfs.c

```
int save_vfs(char *saved_name,
             char *name_with_path);
```

4) read_vfs.c

```
int extract_vfs(char *saved_name,
                char *name_with_path);
```

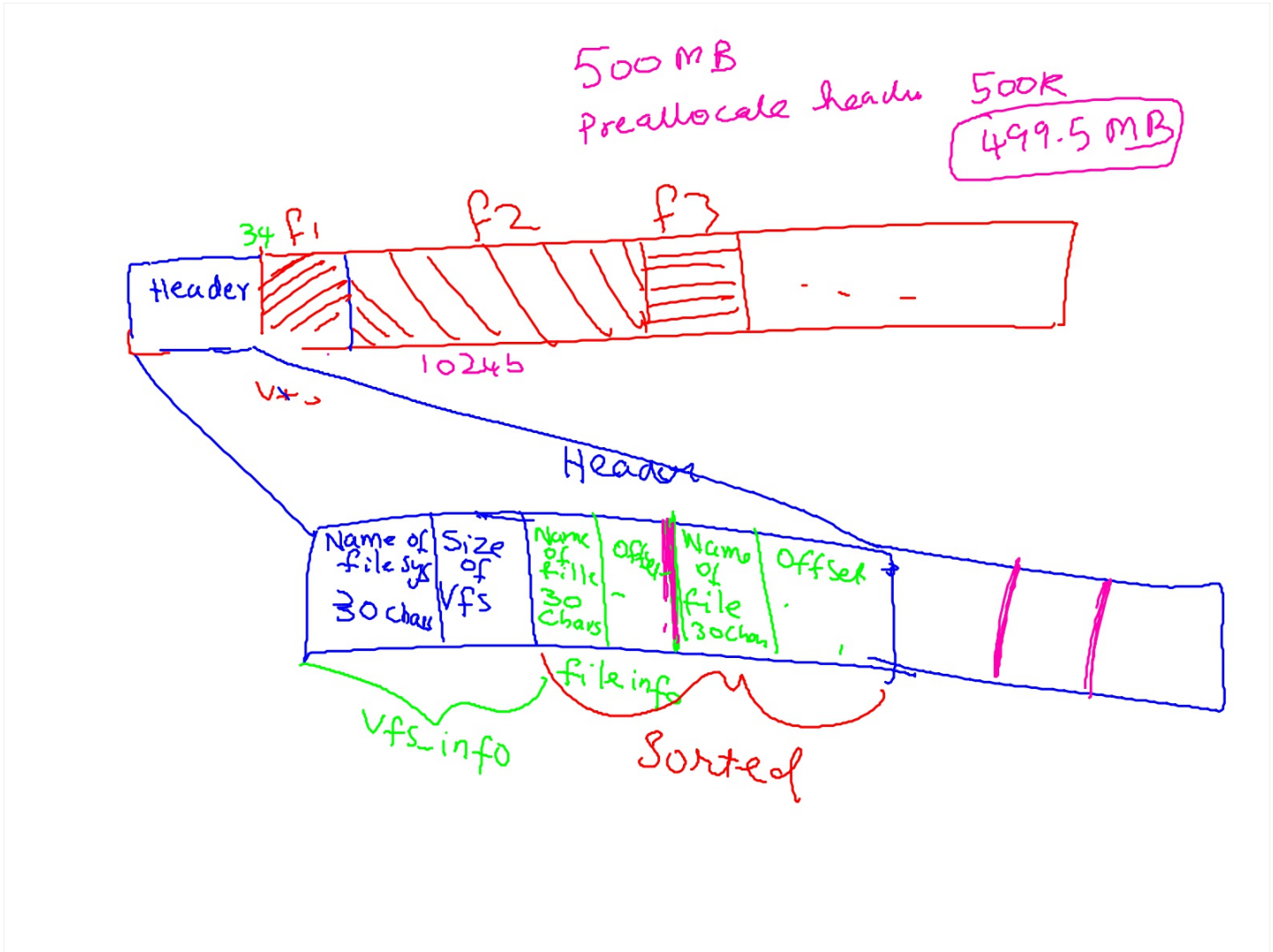
5) create_vfs.c

```
int create_vfs(char *name_with_path, int size);
```

6) vfs_driver.c

```
main();
```

7) `int unload_vfs();`


500 MB
preallocate header
500K
499.5 MB
34
f1
f2
f3
Header
1024b
Vfs
Header
Name of file sys
30 chars
Size of Vfs
Name of file
30 Chars
Offset
Name of file
30chars
Offset
file info
Vfs_info
Sorted

```
// vfs.h
Struct Vfs_File_Info{
        char fname[MAXLEN];
        int offset; int file_size;
};
Struct Vfs_Info {
        char vsf_name[MAXLEN];
        int num_files;
        int size;
};
Struct Vfs_Header_Info{
                Struct Vfs_Info vfs_info;
                Struct Vfs_File_Info vfs_files[MAX_FILES];
}
```

1) Vfs_create

1) Create the vfs file (fopen)
2) Initialize Header
3) Save header ~~to~~ at the beginning of the vfs file (fwrite)
4) Save dummy bytes (uninitialized) into vfs (fwrite) (Pre-allocate)
5) Close file system (fclose)

debatable
Not in VI

2) Vfs_open

1) Open a vfs file (fopen)

2) Load header info in the struct (fread)

3) Initialize Struct Vfs

V2(v++) Sort the Vfs_header_info.Vfs_files Using q_sort

3) Vfs_save

// Pre-conditions: Vfs is loaded.

1) Open external file to be saved (fopen)

2) Calculate offset where file needs to be stored

offset == End of file (fseek)

fseek(fp, 0, SEEK_END)

offset = ftell(fp);

3) Initialize Vfs file_info //

4) Write external file contents at offset/end

5) qsort header

// V2

```
// vfs.h (continued)

struct Vfs {
        struct Vfs_HeaderInfo  header;
        FILE * Vfs_fp;
        int    Vfs_status;
};
#define VFS_OPEN   0
#define VFS_CLOSED 1
```

4) Vfs_close.

1) Save header info intofile

a) fseek to beginning

```
fseek(fp, 0, SEEK_SET);
```

b) fwrite

2) Close the file

fclose

3) Update struct Vfs

5) Vfs.extract

//Precond : Vfs is loaded (opened)

a) Iterate on Vfs_File_Info array

Create empty external file (fopen)

b) Locate the offset

do bsearch( ) <- V2

c) Calculate bytes_to_read

Seek to offset (fseek)

d) Read bytes_to_read

e) Save bytes into external file

f) Close external file

25/7

Function Pointers

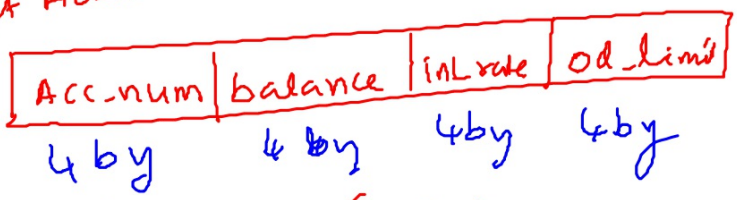
printStatement (struct Account *a)
{
}

Saving A/C
A/C Num
Balance
Interest Rate

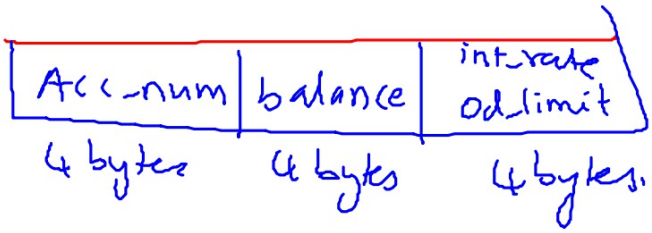
Current A/C
A/C Num
Balance
Overdraft_limit

```
struct Account{ // V1
        int acc_num;
        float balance;
        float interest_rate;
        float od_limit;
};
struct Account { // V2
        int acc_num;
        float balance; int acc_type; //discr
        union {
            float interest_rate;
            float od_limit;
        } acc_info;
};
```

Struct Account (v1)



Struct Account (v2)



```
{
    struct Account myacc;

    myacc.acc_num = 1234;
    myacc.balance = 10000;
    myacc.acc_type = SAVINGS;

    myacc.acc_info.int_rate = 0.05;
}

printf("%f",
    myacc.acc_info.od_limit);
    // Correct Syntactically
    // Incorrect semantically
```

```
// V1
printStatement(Struct Account *a)
{
    // Generic printing
    printf("Ac Name: ", a->acc_num);
    :
    :
    if(a->acc_type == SAVINGS){
        // print saving specific data
    }
    else if(a->acc_type == CURRENT){
        // print specific info.
    }
}
```

```
// V1
printStatement(Struct Account *a,
               void (*specific)(Struct Account *a))
{
    // Generic printing
    printf("Ac Numb: ", a->acc_num);
    :
    :

    (*specific)(a);

}
```

```
void printSavings(struct Account *a)
{
    printf("Interest rate: %f",
           a->acc_info.interest_rate);
}
printCurrent(struct Account *a)
{
    printf("OD Limit %f",
           a->acc_info.od_limit);
}
```

man qsort

```
qsort ( . . . . int (*cmp)(void *,
                           void *, )
                                   )
```

man bsearch

```
bsearch  - - - -
                                   )
```

"generic programming"

```
int    cmpAccNum(struct Account * a1
                 Struct Account *a2)
{
    if ( a1->acc_num < a2->acc_num)
        return -1;
        :
}
```

```
{
    struct Account myacc;

    myacc.acc_num = 1234;
    myacc.balance = 10000;
    myacc.acc_type = SAVINGS;

    myacc.acc_info.int_rate = 0.05;

    printStatement(&myacc, printSavings);
}
```

Name of function
NO arguments (Param)

```
int x;
float *y = &x;
*y = 3.5;
```

```
float y = 3.5;
printf("%d", y); // NOT Type safe
```

Variable arguments