FATİH DOĞAÇ
1901042654
CSE312 HW#2 REPORT

Part 1:

SUPERBLOCK 99 byte	FAT 8192 byte	ROOT DIRECTORY 90 byte	DATA BLOCKS 4185923 byte

FILE SYSTEM
4 MB FOR 1 KB BLOCK SIZE
2 MB FOR 0.5 KB BLOCK SIZE
BLOCK COUNT IS ALWAYS 4096

```
typedef struct __attribute__ ((packed)){
    char filename[32]; // 32 byte name
    int size;
    char user_perm; // Should be initialized to 0.
    char last_modification_date[DATE_LENGTH];
    char file_creation_date[DATE_LENGTH];
    uint32_t first_block_addr;
    char password[9];

} DirectoryEntry; // 90 byte
```

Directory Entries and Directory Table:
Here is the Directory Entry struct. It has 32 bytes for file/directory names. 4 byte for size value. 20 bytes for last modification date and 20 for creation date. 32 bit (4 byte) for the first address block it points. And 8 bytes for password. Total 90 bytes.

I don't have a directory table. In File Allocation Table (FAT), It holds the index of the next block of this file in the file system's data blocks part. In this block it has the directory entry of the file. File's data starts with first_block_addr and can be gathered by continuing on FAT.

Free Blocks:
When initializing the file system, I put -2 in every byte which is 2's complement representation is 254. That indicates it is empty.
When a block is used, say we have 1 KB block size and we created a file with size of 0.5 KB. Remaining 0.5 KB of the block is still empty. But the block is now considered not empty. If we want to create a new file, we have to start from another block.

File Names:
Every file has 32 bytes for their names.

Permissions:

```
#define READ_PERM 0x01  // 0001
#define WRITE_PERM 0x04 // 0100
```

```
char user_permissions = 0;
user_permissions |= READ_PERM;
user_permissions |= WRITE_PERM;
```

I give permissions like this. And for checking the permission;

```
if (user_permissions & READ_PERM)
{
    printf("user can read\n");
}
if (user_permissions & WRITE_PERM)
{
    printf("user can write\n");
}
if ((user_permissions & READ_PERM) && (user_permissions & WRITE_PERM))
{
    printf("user has both\n");
}
```

I use the AND operator. If file doesn't have the permission it can't go into the if statement.

Password Protection:

```
uint32_t simple_hash(const char *str) {
    uint32_t hash = 0;
    while (*str) {
        hash = hash * 31 + *str++;
    }
    return hash;
}

void set_password(DirectoryEntry *entry, const char *password) {
    uint32_t hash = simple_hash(password);
    snprintf(entry->password, sizeof(entry->password), "%08x", hash);
}

int verify_password(const DirectoryEntry *entry, const char *password) {
    uint32_t hash = simple_hash(password);
    char hash_str[9];
    snprintf(hash_str, sizeof(hash_str), "%08x", hash);
    return strcmp(entry->password, hash_str) == 0;
}
```

For protection purposes, I add the ascii equivalent of the char and multiply it by 31. set_password hashes the password and verify password is checking if the hashed password is equal to the password.

Superblock:

```
typedef struct __attribute__ ((packed)){
    int block_size;
    int block_count;
    uint8_t* fat_first_addr;
    DirectoryEntry root;
} superblock; // 99 byte
```

Superblock keeps the information of block size (4 byte), block count (4096 as default and 4 byte space), fat table's start address in file system (1 byte), Directory Entry of the root because someone has to keep it somewhere (90 byte). Total of 99 byte.