CSE312 OPERATING SYSTEMS
HW2 REPORT

200104004111

R.Hilal Saygin

Introduction

Usage

Compile with make command
Initialize the system file by running ./makeFileSystem 1 makeFileSystem.data
Run the operation with

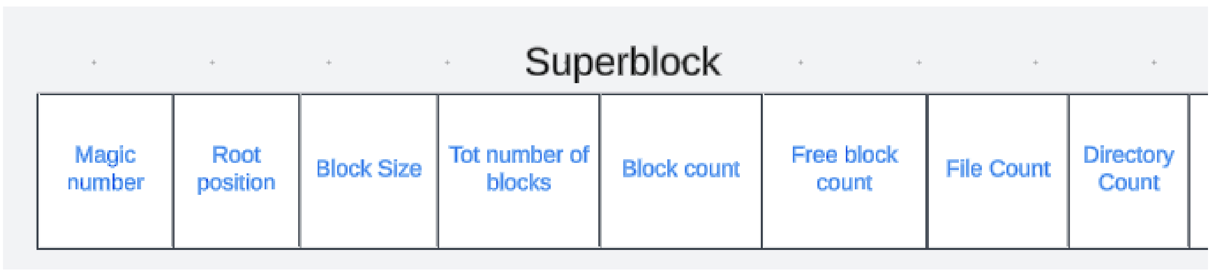
```
./fileSystemOper makeFileSystem.data mkdir "\usr"
./fileSystemOper makeFileSystem.data write "\usr\hilal.txt" filetes.txt
```

Implementation

FAT12 System Design

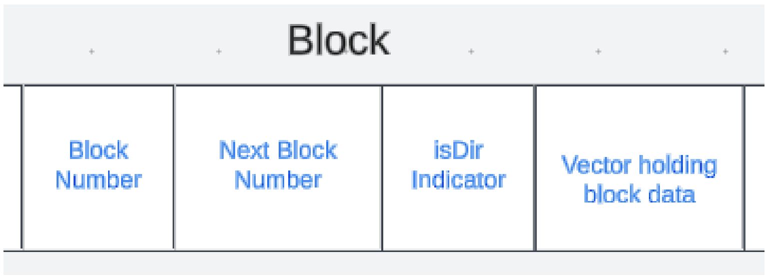
SuperBlock

Superblock structure used in my FAT12 file system



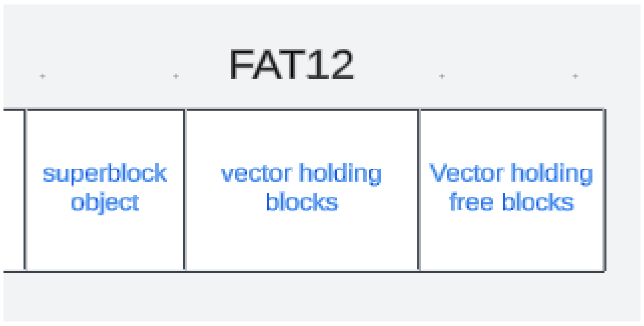
Block

The block are structured as;

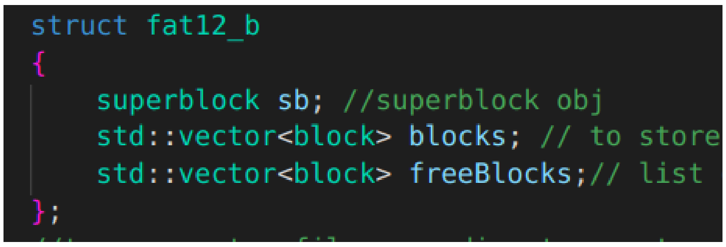


FAT Table

And the fat12 struct object hod the superblock, vector of multiple blocks initialised with the provided block size. And it hold the vector of freeblock exist in the file system currently.



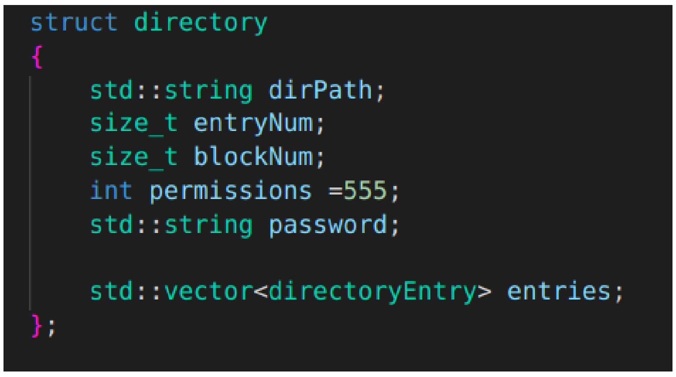
The fat12 FAT table object represented with the struct.



```
struct fat12_b
{
    superblock sb; //superblock obj
    std::vector<block> blocks; // to store
    std::vector<block> freeBlocks;// list
};
```

Directories

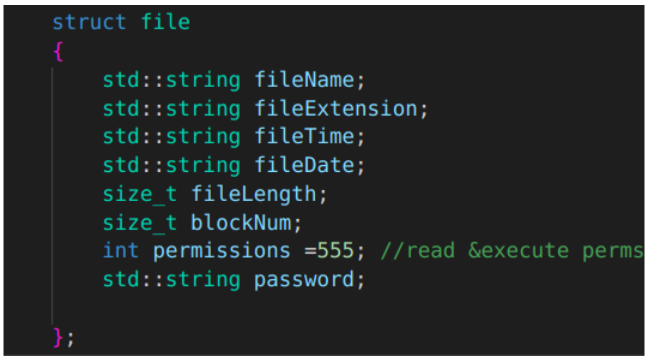
Directories are represented with



```
struct directory
{
    std::string dirPath;
    size_t entryNum;
    size_t blockNum;
    int permissions =555;
    std::string password;

    std::vector<directoryEntry> entries;
};
```

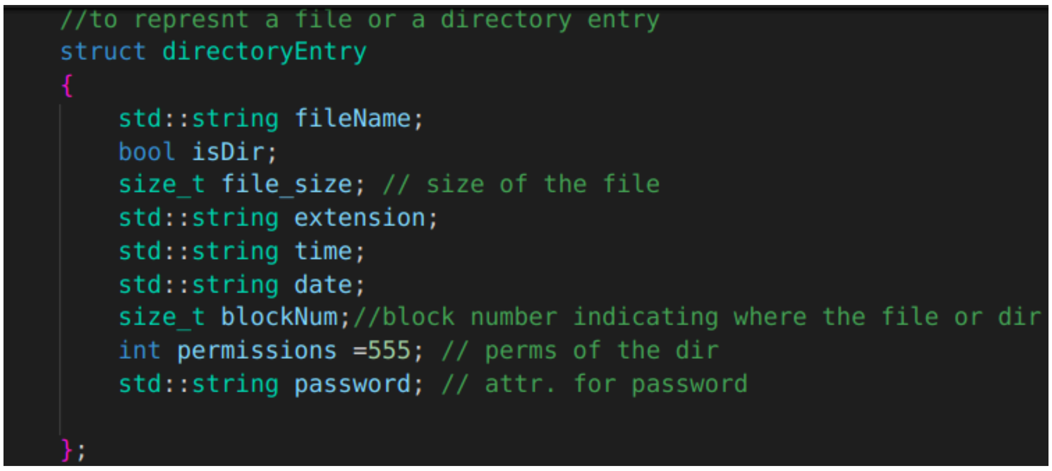
Files

File objects are represented with



```
struct file
{
    std::string fileName;
    std::string fileExtension;
    std::string fileTime;
    std::string fileDate;
    size_t fileLength;
    size_t blockNum;
    int permissions =555; //read &execute perms
    std::string password;

};
```

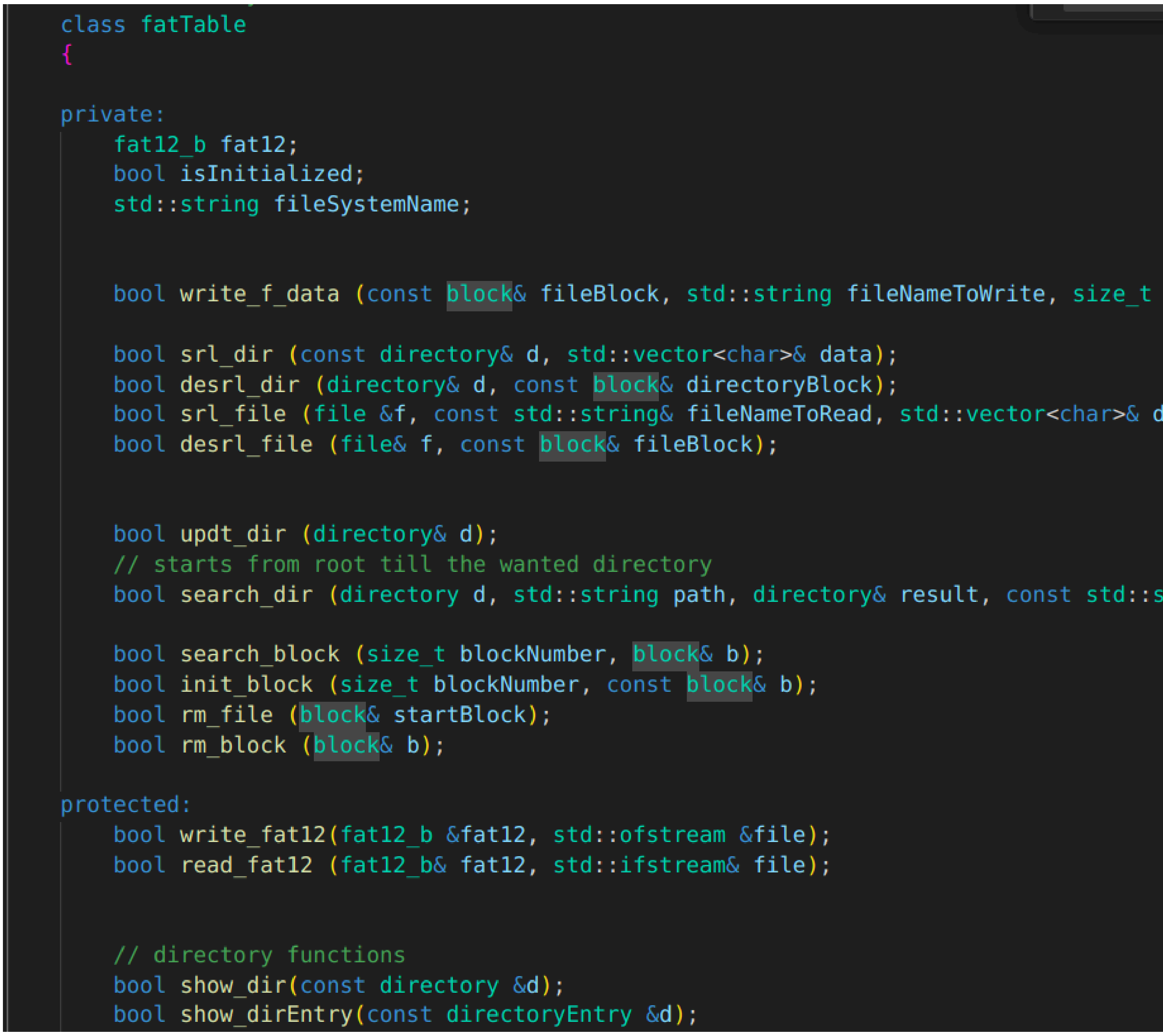
And to hold the file or directory entries in teh same vector attribute of te fat12 objects blocks variable, directoryEntry structure defined.



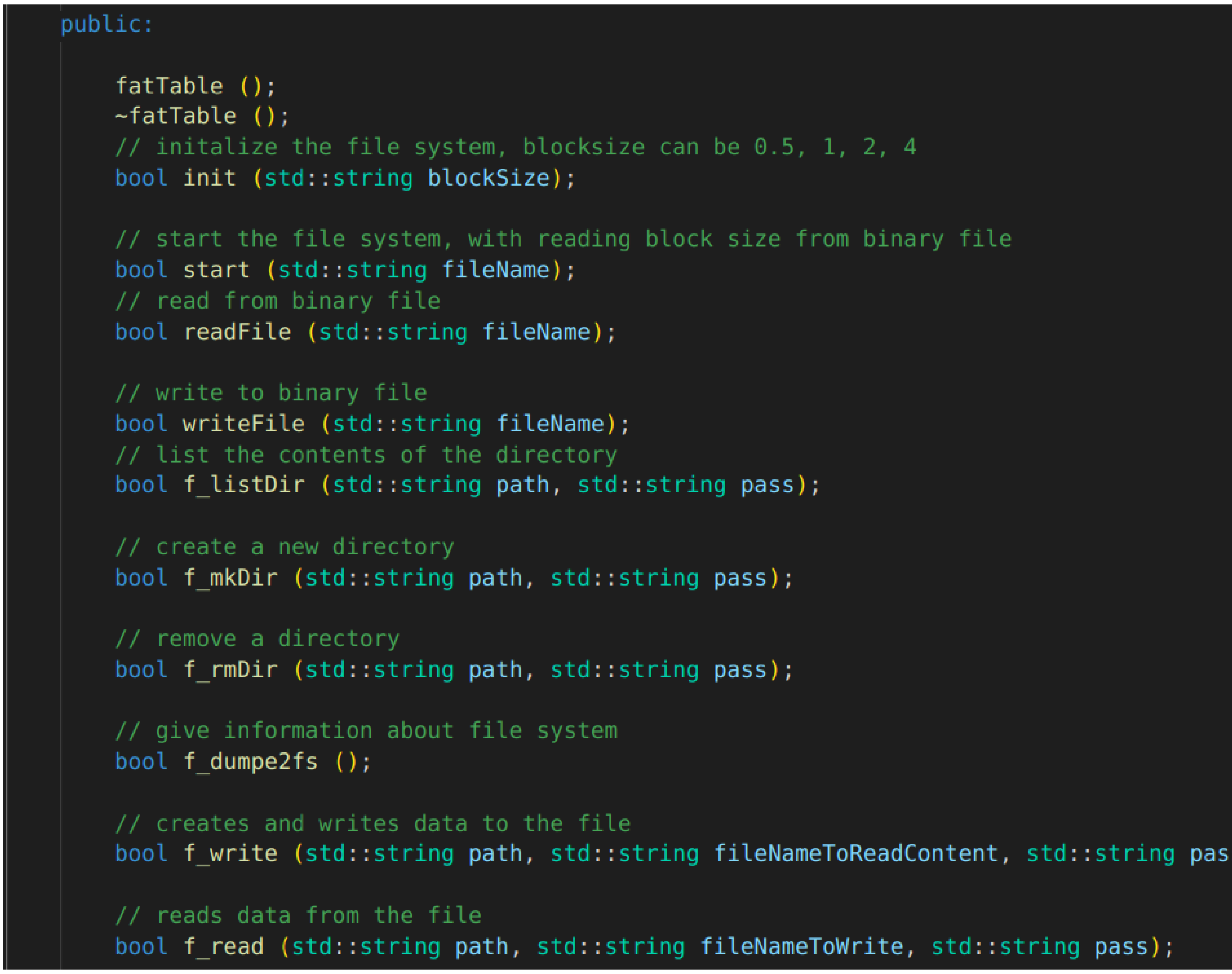
```
//to represnt a file or a directory entry
struct directoryEntry
{
    std::string fileName;
    bool isDir;
    size_t file_size; // size of the file
    std::string extension;
    std::string time;
    std::string date;
    size_t blockNum;//block number indicating where the file or dir
    int permissions =555; // perms of the dir
    std::string password; // attr. for password

};
```

File System Creation

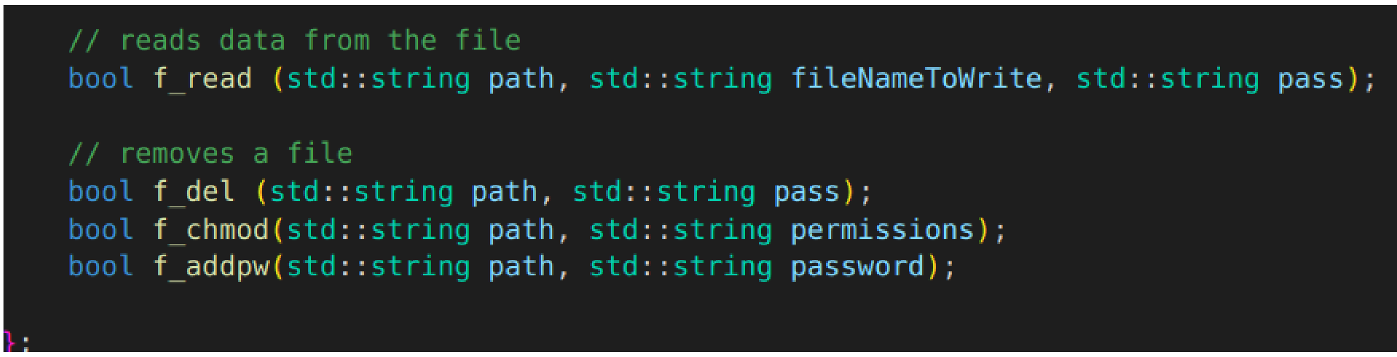
The complete system is combined within the fatTable class and the file operations are implemented under the class definition.



```
class fatTable
{

private:
    fat12_b fat12;
    bool isInitialized;
    std::string fileSystemName;


    bool write_f_data (const block& fileBlock, std::string fileNameToWrite, size_t

    bool srl_dir (const directory& d, std::vector<char>& data);
    bool desrl_dir (directory& d, const block& directoryBlock);
    bool srl_file (file &f, const std::string& fileNameToRead, std::vector<char>& d
    bool desrl_file (file& f, const block& fileBlock);


    bool updt_dir (directory& d);
    // starts from root till the wanted directory
    bool search_dir (directory d, std::string path, directory& result, const std::s

    bool search_block (size_t blockNumber, block& b);
    bool init_block (size_t blockNumber, const block& b);
    bool rm_file (block& startBlock);
    bool rm_block (block& b);

protected:
    bool write_fat12(fat12_b &fat12, std::ofstream &file);
    bool read_fat12 (fat12_b& fat12, std::ifstream& file);


    // directory functions
    bool show_dir(const directory &d);
    bool show_dirEntry(const directoryEntry &d);
```

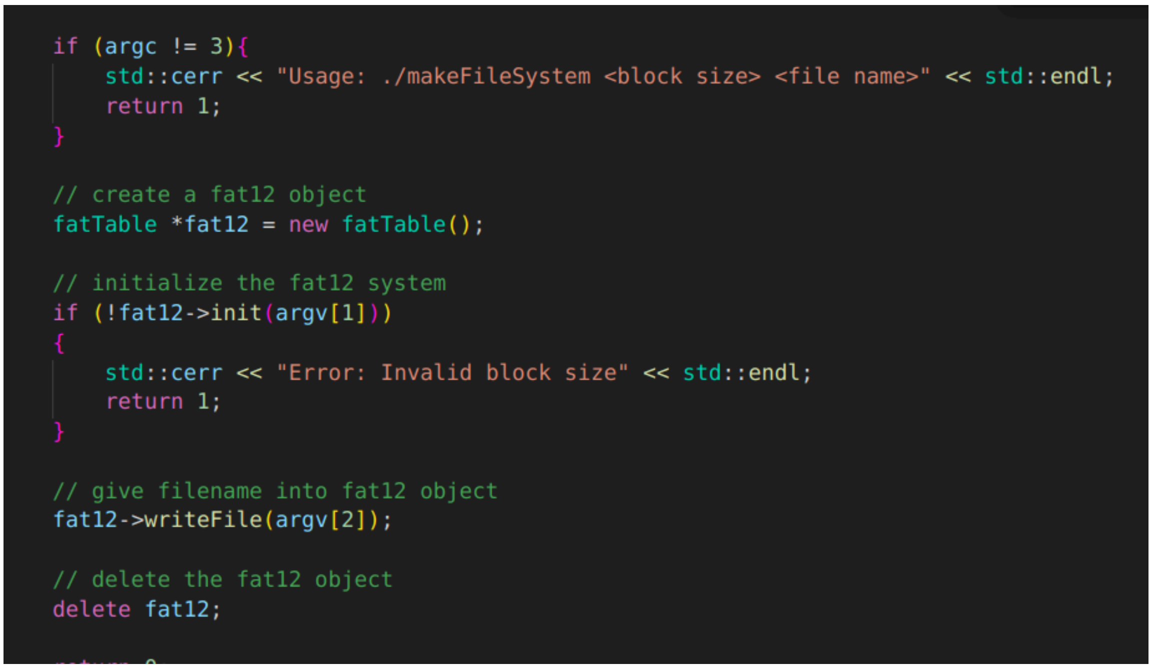


```
public:

    fatTable ();
    ~fatTable ();
    // initalize the file system, blocksize can be 0.5, 1, 2, 4
    bool init (std::string blockSize);

    // start the file system, with reading block size from binary file
    bool start (std::string fileName);
    // read from binary file
    bool readFile (std::string fileName);

    // write to binary file
    bool writeFile (std::string fileName);
    // list the contents of the directory
    bool f_listDir (std::string path, std::string pass);

    // create a new directory
    bool f_mkDir (std::string path, std::string pass);

    // remove a directory
    bool f_rmDir (std::string path, std::string pass);

    // give information about file system
    bool f_dumpe2fs ();

    // creates and writes data to the file
    bool f_write (std::string path, std::string fileNameToReadContent, std::string pas

    // reads data from the file
    bool f_read (std::string path, std::string fileNameToWrite, std::string pass);
```



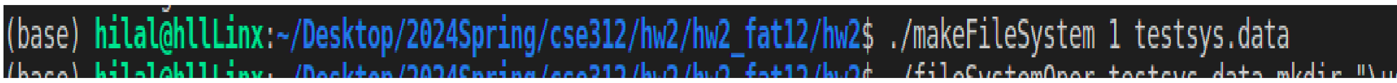
```
    // reads data from the file
    bool f_read (std::string path, std::string fileNameToWrite, std::string pass);

    // removes a file
    bool f_del (std::string path, std::string pass);
    bool f_chmod(std::string path, std::string permissions);
    bool f_addpw(std::string path, std::string password);

};
```

File system object file is created as



```
if (argc != 3){
    std::cerr << "Usage: ./makeFileSystem <block size> <file name>" << std::endl;
    return 1;
}

// create a fat12 object
fatTable *fat12 = new fatTable();

// initialize the fat12 system
if (!fat12->init(argv[1]))
{
    std::cerr << "Error: Invalid block size" << std::endl;
    return 1;
}

// give filename into fat12 object
fat12->writeFile(argv[2]);

// delete the fat12 object
delete fat12;
```

With the command



```
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./makeFileSystem 1 testsys.data
```

File System Operations

Dir

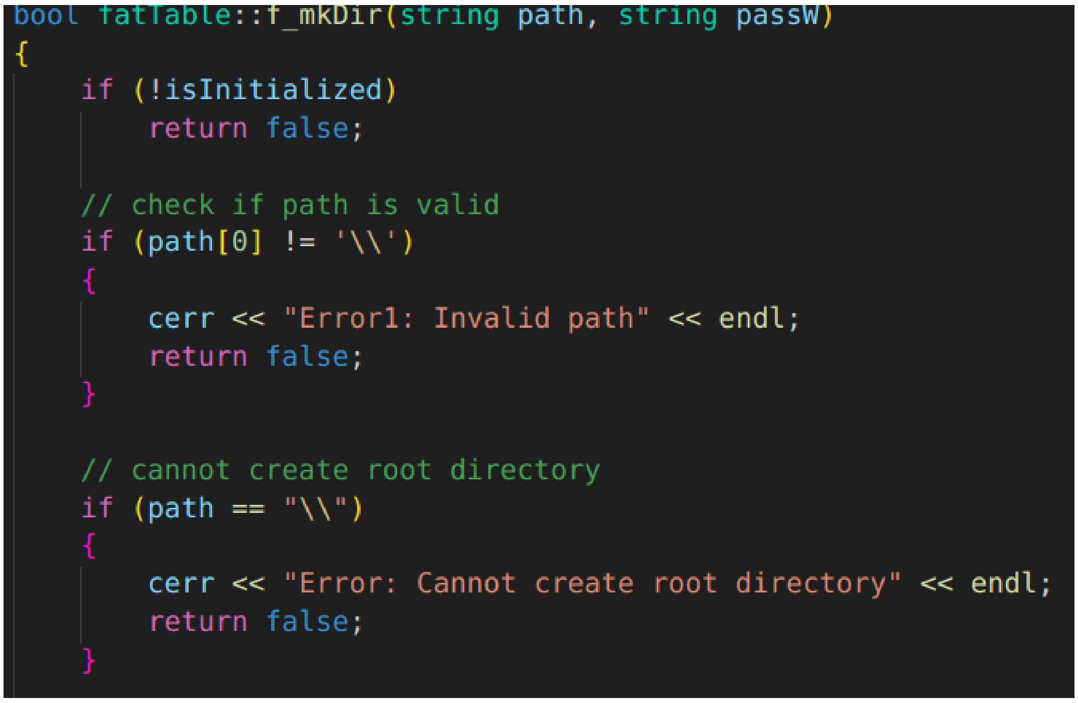
It lists the contents of a directory.

ITraverse the directory structure, retrieve the names of files and subdirectories, and display them to the user.

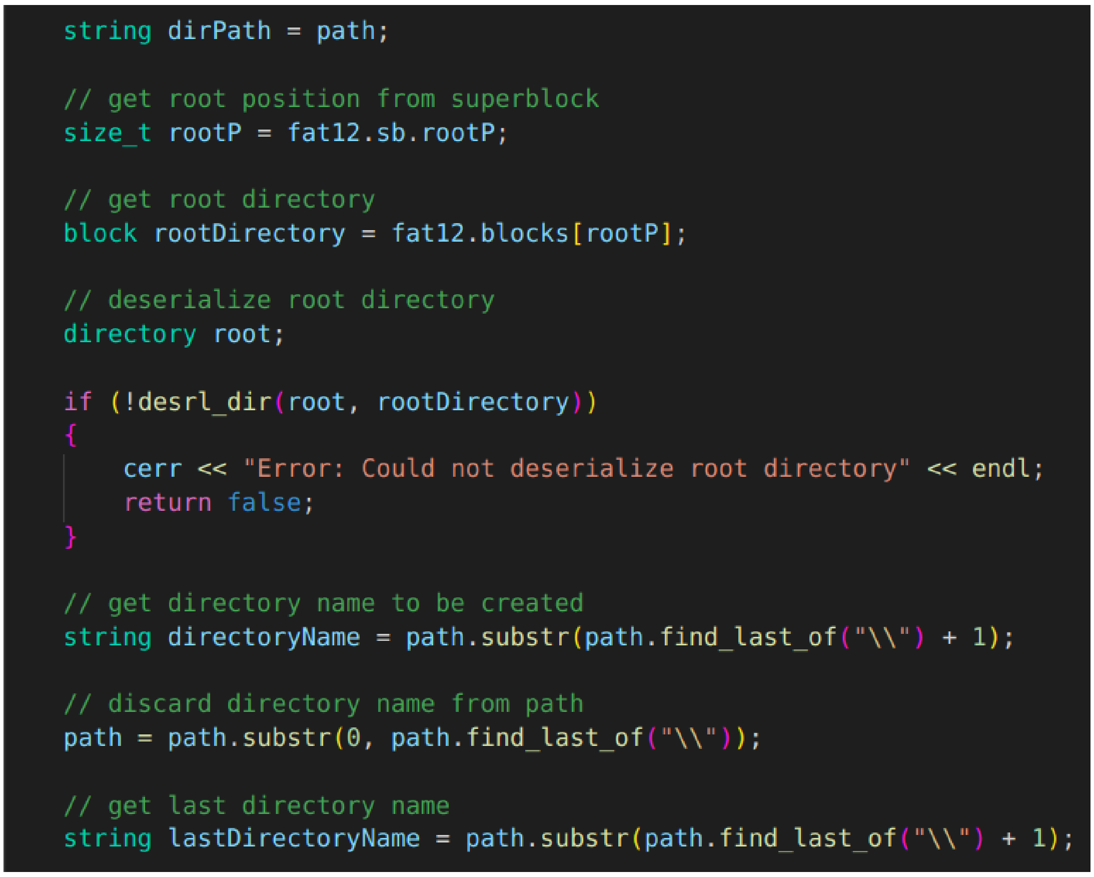
Mkdir

I explain the details of creating new directory steps below.

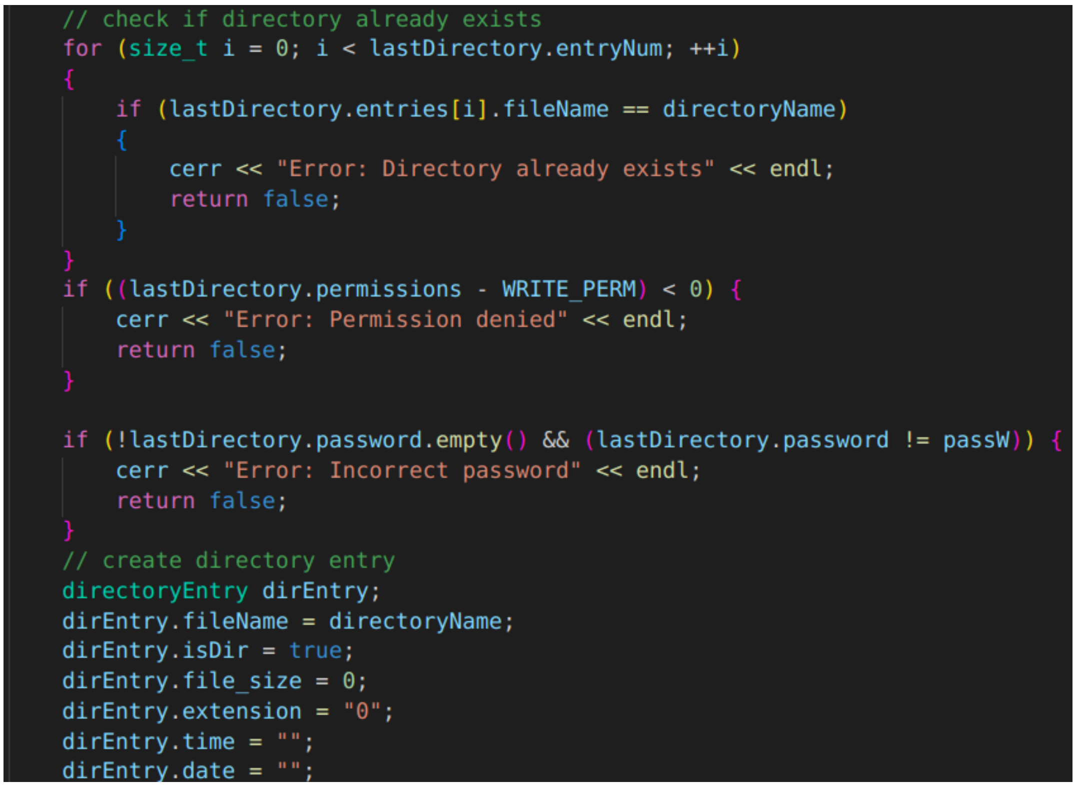
1. Check if the provided path is valid and not the root directory.Ensure there are free blocks available in the file system.
2. Retrieve the root directory block and deserialize it into a directory object.
3. Extract the name of the new directory from the given path. Then extract the parent path and last directory name.
4. Using search_dir to locate the parent directory.
5. Ensuring the directory to be created does not already exist.
6. Check write permissions and password (if any) for the parent directory.
7. Create a new directoryEntry object for the new directory.Then allocate a new block for the directory from the free blocks list and set necessary attributes.
8. Add the new directory entry to the parent directory. Then serialisation and update of the parent directory block.
9. Create a new directory object for the new directory and serialise it. Then it assigns the serialised data to the allocated block and update the file system structures.



```cpp
bool fatTable::f_mkDir(string path, string passW)
{
    if (!isInitialized)
        return false;

    // check if path is valid
    if (path[0] != '\\')
    {
        cerr << "Error1: Invalid path" << endl;
        return false;
    }

    // cannot create root directory
    if (path == "\\")
    {
        cerr << "Error: Cannot create root directory" << endl;
        return false;
    }
```

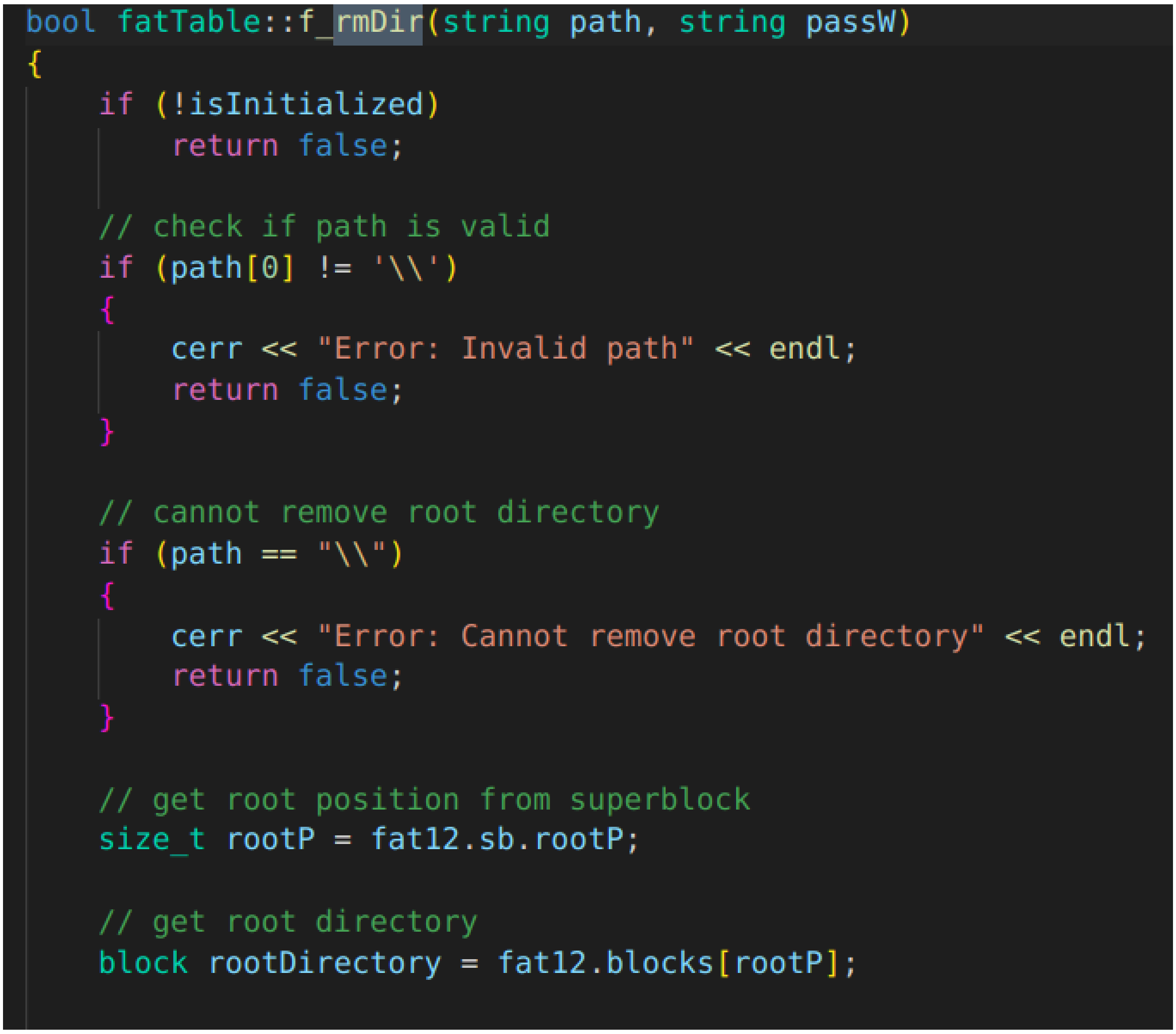


```cpp
    string dirPath = path;

    // get root position from superblock
    size_t rootP = fat12.sb.rootP;

    // get root directory
    block rootDirectory = fat12.blocks[rootP];

    // deserialize root directory
    directory root;

    if (!desrl_dir(root, rootDirectory))
    {
        cerr << "Error: Could not deserialize root directory" << endl;
        return false;
    }

    // get directory name to be created
    string directoryName = path.substr(path.find_last_of("\\") + 1);

    // discard directory name from path
    path = path.substr(0, path.find_last_of("\\"));

    // get last directory name
    string lastDirectoryName = path.substr(path.find_last_of("\\") + 1);
```

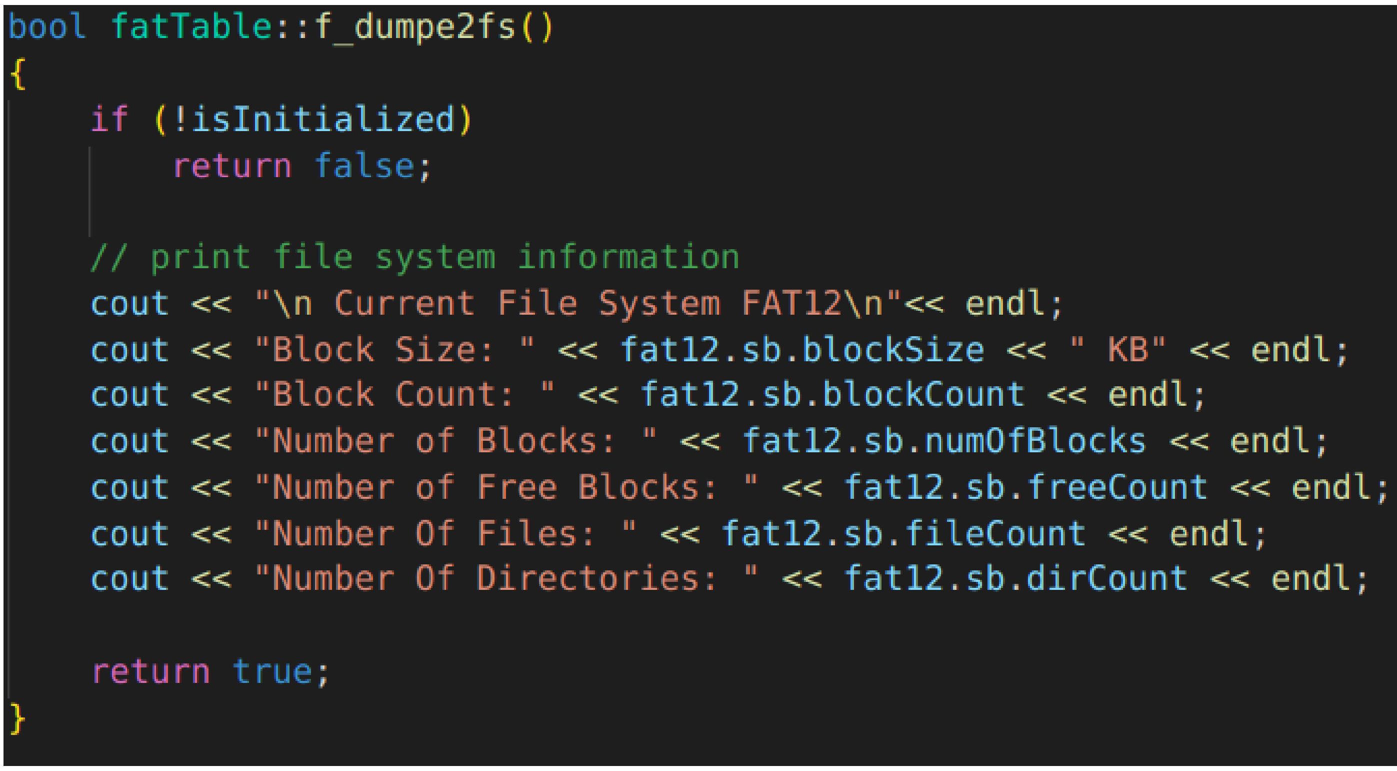


```
    // check if directory already exists
    for (size_t i = 0; i < lastDirectory.entryNum; ++i)
    {
        if (lastDirectory.entries[i].fileName == directoryName)
        {
            cerr << "Error: Directory already exists" << endl;
            return false;
        }
    }
    if ((lastDirectory.permissions - WRITE_PERM) < 0) {
        cerr << "Error: Permission denied" << endl;
        return false;
    }

    if (!lastDirectory.password.empty() && (lastDirectory.password != passW)) {
        cerr << "Error: Incorrect password" << endl;
        return false;
    }
    // create directory entry
    directoryEntry dirEntry;
    dirEntry.fileName = directoryName;
    dirEntry.isDir = true;
    dirEntry.file_size = 0;
    dirEntry.extension = "0";
    dirEntry.time = "";
    dirEntry.date = "";
```

Rmdir

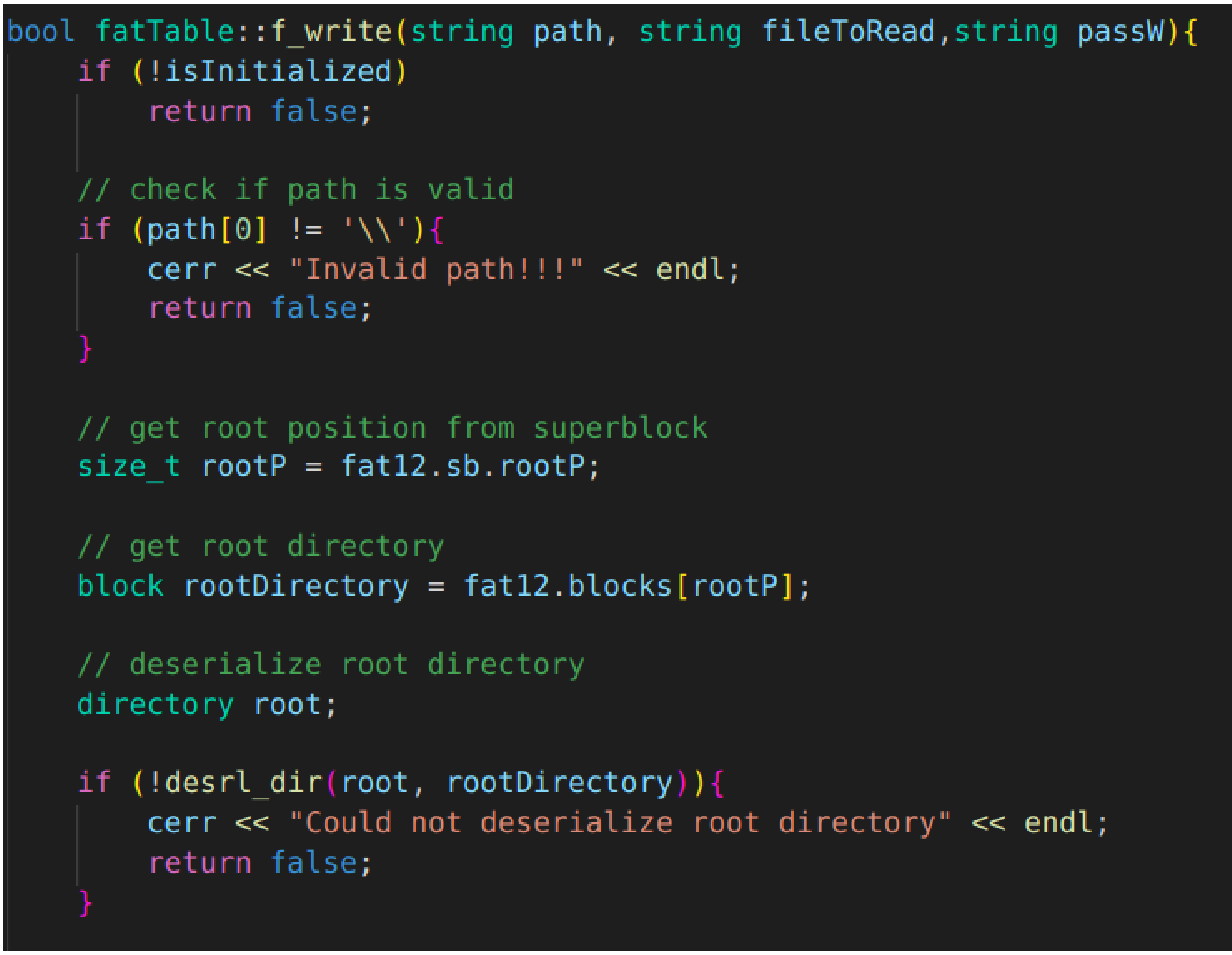

```cpp
bool fatTable::f_rmDir(string path, string passW)
{
    if (!isInitialized)
        return false;

    // check if path is valid
    if (path[0] != '\\')
    {
        cerr << "Error: Invalid path" << endl;
        return false;
    }

    // cannot remove root directory
    if (path == "\\")
    {
        cerr << "Error: Cannot remove root directory" << endl;
        return false;
    }

    // get root position from superblock
    size_t rootP = fat12.sb.rootP;

    // get root directory
    block rootDirectory = fat12.blocks[rootP];

```

Dumpe2fs



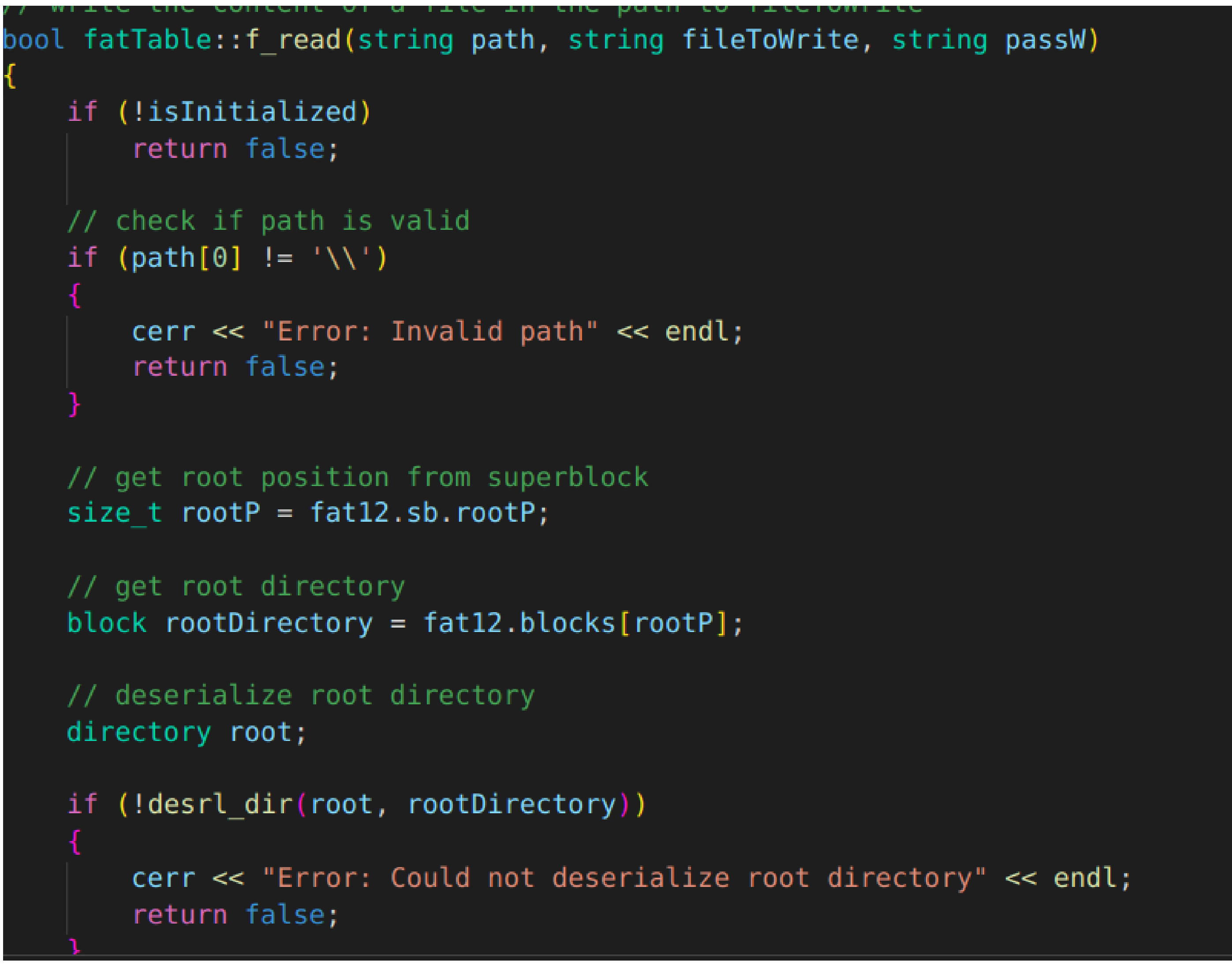
```cpp
bool fatTable::f_dumpe2fs()
{
    if (!isInitialized)
        return false;

    // print file system information
    cout << "\n Current File System FAT12\n"<< endl;
    cout << "Block Size: " << fat12.sb.blockSize << " KB" << endl;
    cout << "Block Count: " << fat12.sb.blockCount << endl;
    cout << "Number of Blocks: " << fat12.sb.numOfBlocks << endl;
    cout << "Number of Free Blocks: " << fat12.sb.freeCount << endl;
    cout << "Number Of Files: " << fat12.sb.fileCount << endl;
    cout << "Number Of Directories: " << fat12.sb.dirCount << endl;

    return true;
}
```

Write

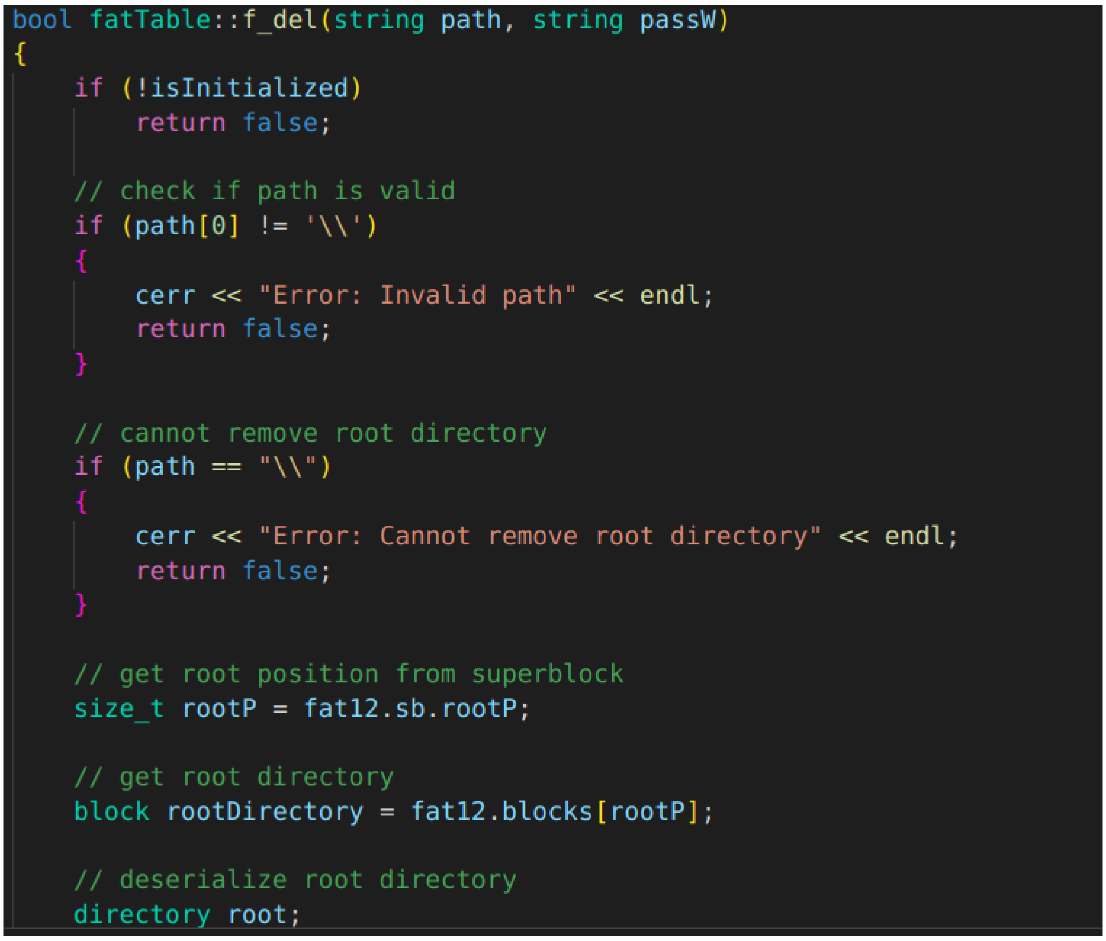


```
bool fatTable::f_write(string path, string fileToRead,string passW){
    if (!isInitialized)
        return false;

    // check if path is valid
    if (path[0] != '\\'){
        cerr << "Invalid path!!!" << endl;
        return false;
    }

    // get root position from superblock
    size_t rootP = fat12.sb.rootP;

    // get root directory
    block rootDirectory = fat12.blocks[rootP];

    // deserialize root directory
    directory root;

    if (!desrl_dir(root, rootDirectory)){
        cerr << "Could not deserialize root directory" << endl;
        return false;
    }
```

Read



```
// write the content of a file in the path to fileToWrite
bool fatTable::f_read(string path, string fileToWrite, string passW)
{
    if (!isInitialized)
        return false;

    // check if path is valid
    if (path[0] != '\\')
    {
        cerr << "Error: Invalid path" << endl;
        return false;
    }

    // get root position from superblock
    size_t rootP = fat12.sb.rootP;

    // get root directory
    block rootDirectory = fat12.blocks[rootP];

    // deserialize root directory
    directory root;

    if (!desrl_dir(root, rootDirectory))
    {
        cerr << "Error: Could not deserialize root directory" << endl;
        return false;
    }
```



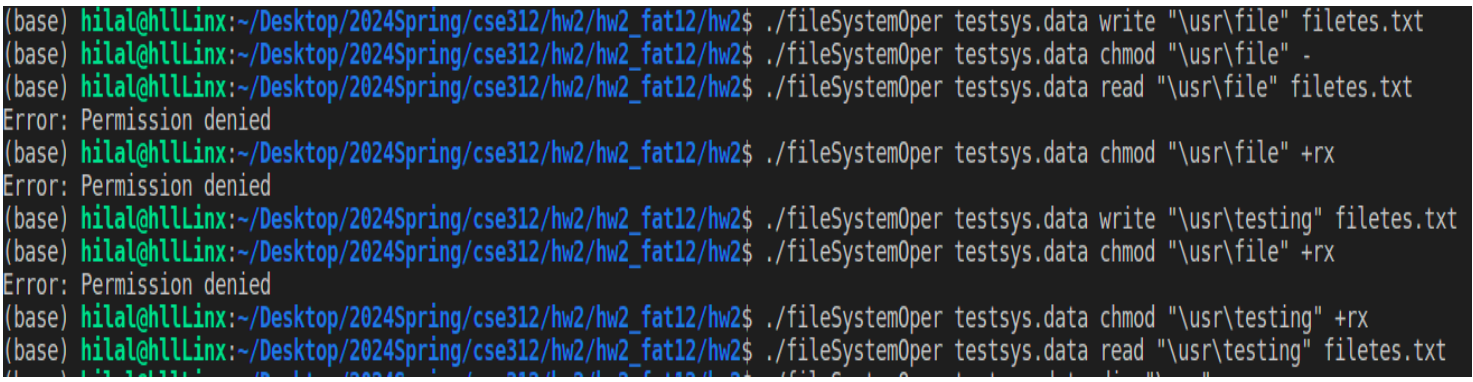
```
bool fatTable::f_del(string path, string passW)
{
    if (!isInitialized)
        return false;

    // check if path is valid
    if (path[0] != '\\')
    {
        cerr << "Error: Invalid path" << endl;
        return false;
    }

    // cannot remove root directory
    if (path == "\\")
    {
        cerr << "Error: Cannot remove root directory" << endl;
        return false;
    }

    // get root position from superblock
    size_t rootP = fat12.sb.rootP;

    // get root directory
    block rootDirectory = fat12.blocks[rootP];

    // deserialize root directory
    directory root;
```

Del

Chmod

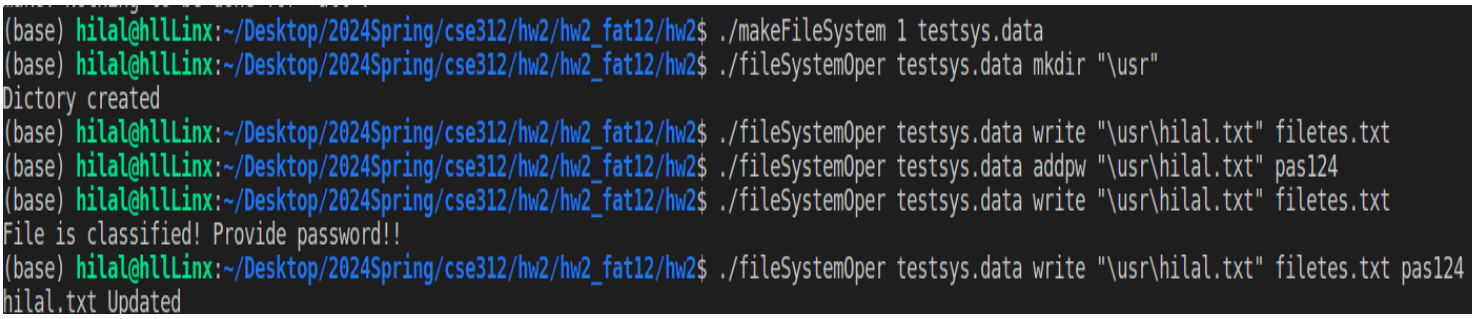
addpw

Results



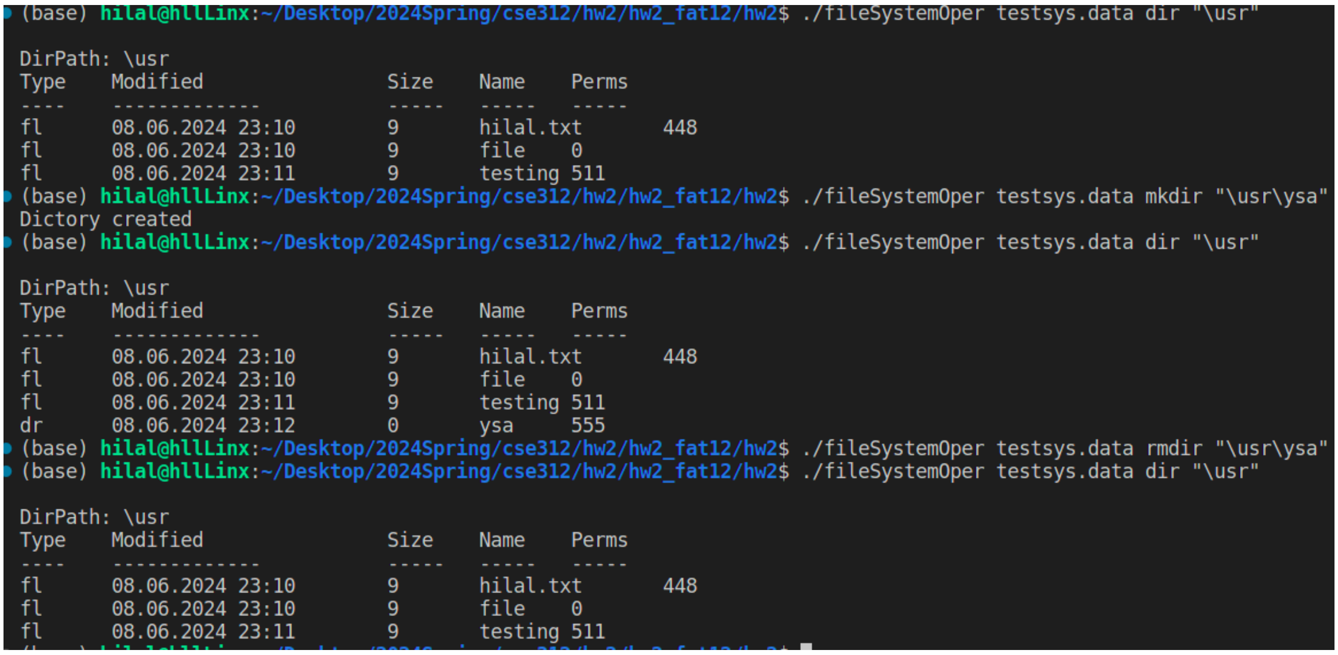
```
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data write "\usr\file" filetes.txt
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data chmod "\usr\file" -
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data read "\usr\file" filetes.txt
Error: Permission denied
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data chmod "\usr\file" +rx
Error: Permission denied
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data write "\usr\testing" filetes.txt
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data chmod "\usr\file" +rx
Error: Permission denied
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data chmod "\usr\testing" +rx
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data read "\usr\testing" filetes.txt
```

When password added to a file created, user cannot make any changes or cannot read or write to that classified file without provisioning the password.
Therefore we accomplish the file and directory protection via user password.



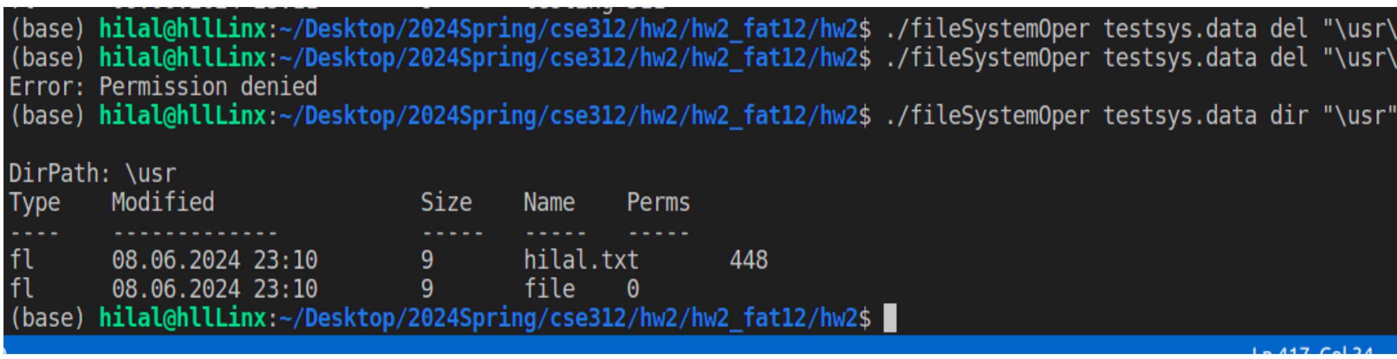
```
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./makeFileSystem 1 testsys.data
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data mkdir "\usr"
Dictory created
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data write "\usr\hilal.txt" filetes.txt
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data addpw "\usr\hilal.txt" pas124
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data write "\usr\hilal.txt" filetes.txt
File is classified! Provide password!!
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data write "\usr\hilal.txt" filetes.txt pas124
hilal.txt Updated
```

If permissions for the user are valid, user can change the the directory including removing the directory if no file present inside.



```
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data dir "\usr"

DirPath: \usr
Type    Modified                Size    Name    Perms
----    -------------           -----   -----   -----
fl      08.06.2024 23:10        9       hilal.txt       448
fl      08.06.2024 23:10        9       file    0
fl      08.06.2024 23:11        9       testing 511
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data mkdir "\usr\ysa"
Dictory created
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data dir "\usr"

DirPath: \usr
Type    Modified                Size    Name    Perms
----    -------------           -----   -----   -----
fl      08.06.2024 23:10        9       hilal.txt       448
fl      08.06.2024 23:10        9       file    0
fl      08.06.2024 23:11        9       testing 511
dr      08.06.2024 23:12        0       ysa     555
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data rmdir "\usr\ysa"
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data dir "\usr"

DirPath: \usr
Type    Modified                Size    Name    Perms
----    -------------           -----   -----   -----
fl      08.06.2024 23:10        9       hilal.txt       448
fl      08.06.2024 23:10        9       file    0
fl      08.06.2024 23:11        9       testing 511
```

If user have valid permissions, they can delta the created file in the provided directory path.



```
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data del "\usr\
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data del "\usr\
Error: Permission denied
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$ ./fileSystemOper testsys.data dir "\usr"

DirPath: \usr
Type    Modified                Size    Name    Perms
----    -------------           -----   -----   -----
fl      08.06.2024 23:10        9       hilal.txt       448
fl      08.06.2024 23:10        9       file    0
(base) hilal@hllLinx:~/Desktop/2024Spring/cse312/hw2/hw2_fat12/hw2$
```

As it can be displayed on the above output, users cannot delta the files which they dont have write permissions.
When “del” command executed on testing file, the file removed.
However, when file named ”file” attempted to be deleted. The operation isn’t executed on the file.