Customised Virtual file system

- Technology used : System Programming using C.
- User Interface used : Command Line Interface.
- Platform required : Windows platform, Linux.
- Hardware requirement : Any modern processor can run the given operating system and compiler efficiently . RAM : Enough RAM to compile and run the programme
- Technology used : System Programming using C.
- Description of the project :

The code defines four structures (SUPERBLOCK, INODE, FILETABLE & UFDT)

1. SUPERBLOCK : Tracks the total number of inodes and count of free inodes.

2. INODE : Represent an inode int the file system storing attributes like file name , file size file type the number if links and permission.

3. FILETABLE : Keep track of file operations and referenced inode.

4. UFDT(User file descriptor table) : It's a array of pointer which holds the address of specific entry from the file table. First three entries from the UFDT are reserved for stdin, stdout, stderror.

- Data Structures used in project :

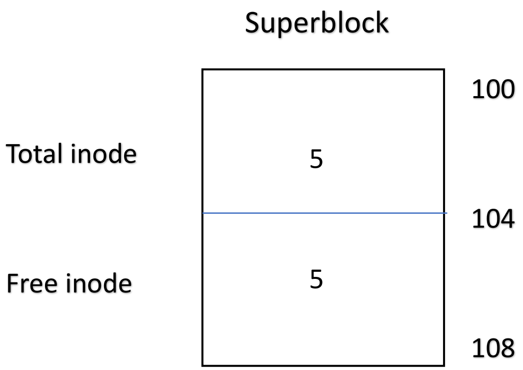
1. Superblock

2.Inode

3.Filetable

4.UFDT

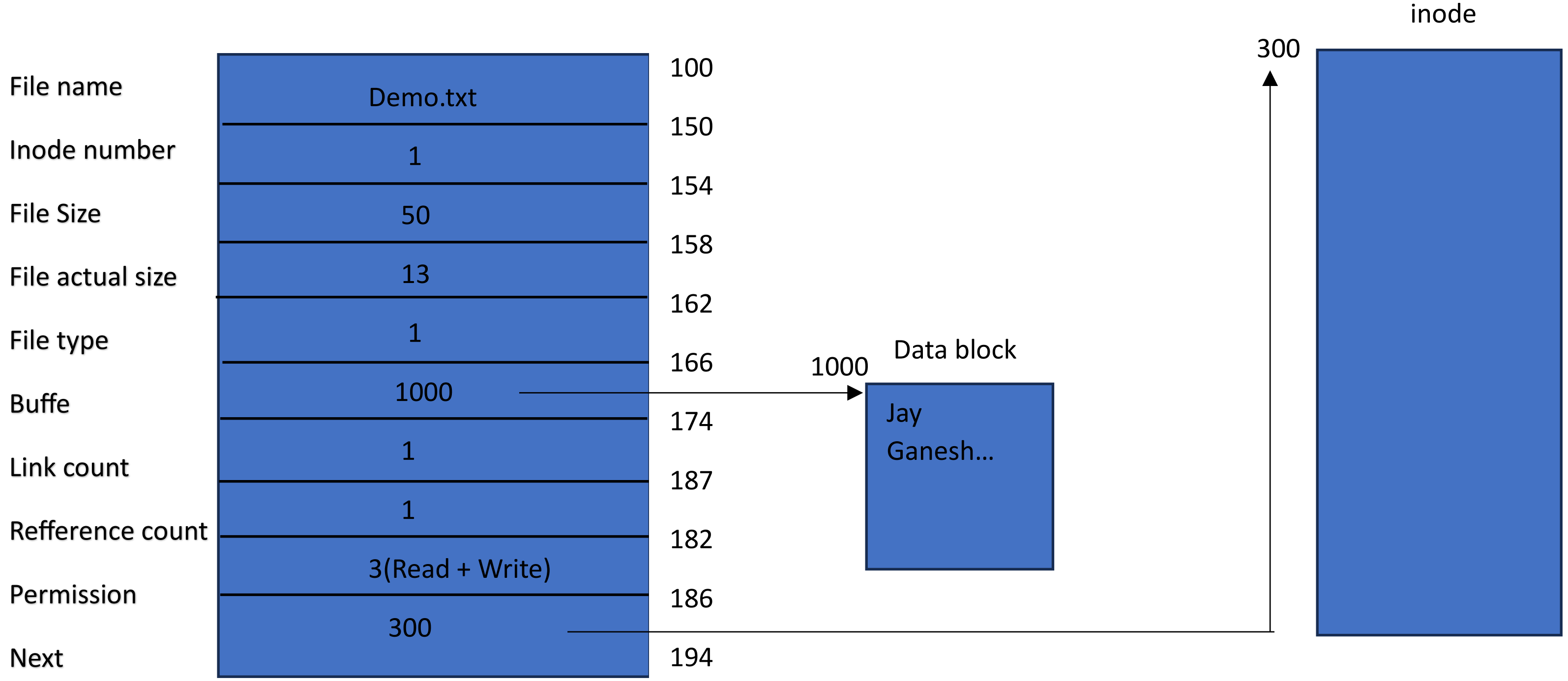
- Diagram of data structures used in project :



User define macro :

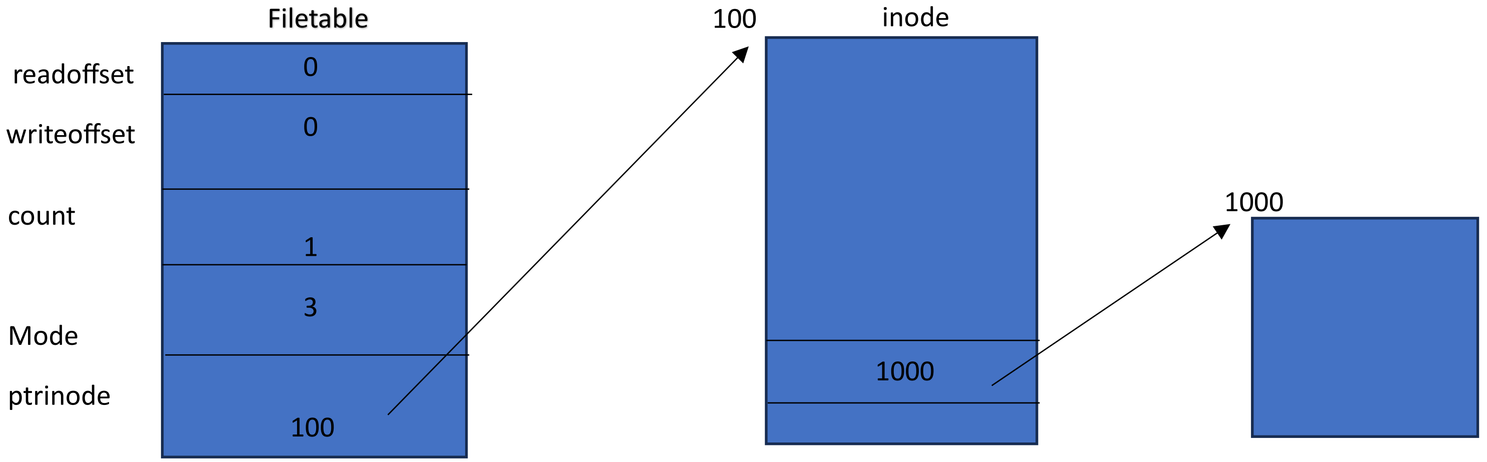
MAXINODE	5	SPECIAL	5	WRITE	2	Read	1
START	0	CURRENT	1	MAXSIZE	100	REGULAR	1
				END	2		

Inode



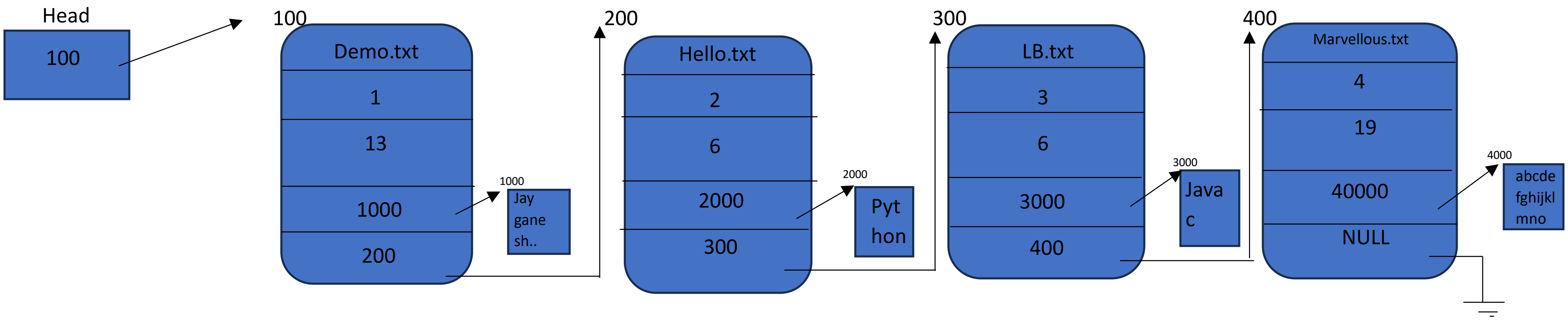
File name	Demo.txt	100
Inode number	1	150
File Size	50	154
File actual size	13	158
File type	1	162
Buffe	1000	166
Link count	1	174
Refference count	1	187
Permission	3(Read + Write)	182
Next	300	186
		194

Filetable :



Filetable	
readoffset	0
writeoffset	0
count	1
Mode	3
ptrinode	100

DILB :



UFDT :

–	–	–	100	

➢ Flow of the project :

This project includes various functionalities to manage files such as creating, reading, writing, opening, closing, deletion and displaying file information.

➢ Initialisation :

Initializes the superblock and UFDT (user file descriptive table)array.

Creates the DILB (Disk inode list block which is a linked list of inodes.

➢ Command line interface :

The main() function runs a loop to take user command and execute corresponding operations.

➢ Command Supported :

- ls : List of existing files.

- clear : Clears the consol.
- close : Closes a file by name.
- open : Opens an existing file.
- closeall : Closes all open files.
- read : Reads data from a file.
- write : Writes data to a file.
- exit : Terminates the CVFS.
- stat : Display file information by name.
- fstat : Display file information by file descriptor.
- truncate : Removes all data from a file.
- rm : Delete a file.

➢ Function Details :

- Man() : Provides a manual for various commands.
- DisplayHelp() : Display a list of available commands.
- GetFDFromName() : Gets file descriptor from the file name.
- Get_Inode() : Retrives an inode based on the file name.
- CreateFile() :Creates a new file with given name and permission.
- rm_file : Rmoves a file by name.
- ReadFile() : Reads data from the file descriptor into buffer.
- WriteFile() : Write a data from a buffer to a file descriptor.
- OpenFile() : Opens a file with a given name and mode.
- CloseFileByName() : Cloases a file by name.
- CloseAllFile() : Closes all file.
- LseekFile() : changes the file offset.
- ls_file() : List al files with their details.
- fstat_fle() : Display file information using file name.
- stat_file() : Display file information using a file name
- truncate_file() : truncates a file by name.

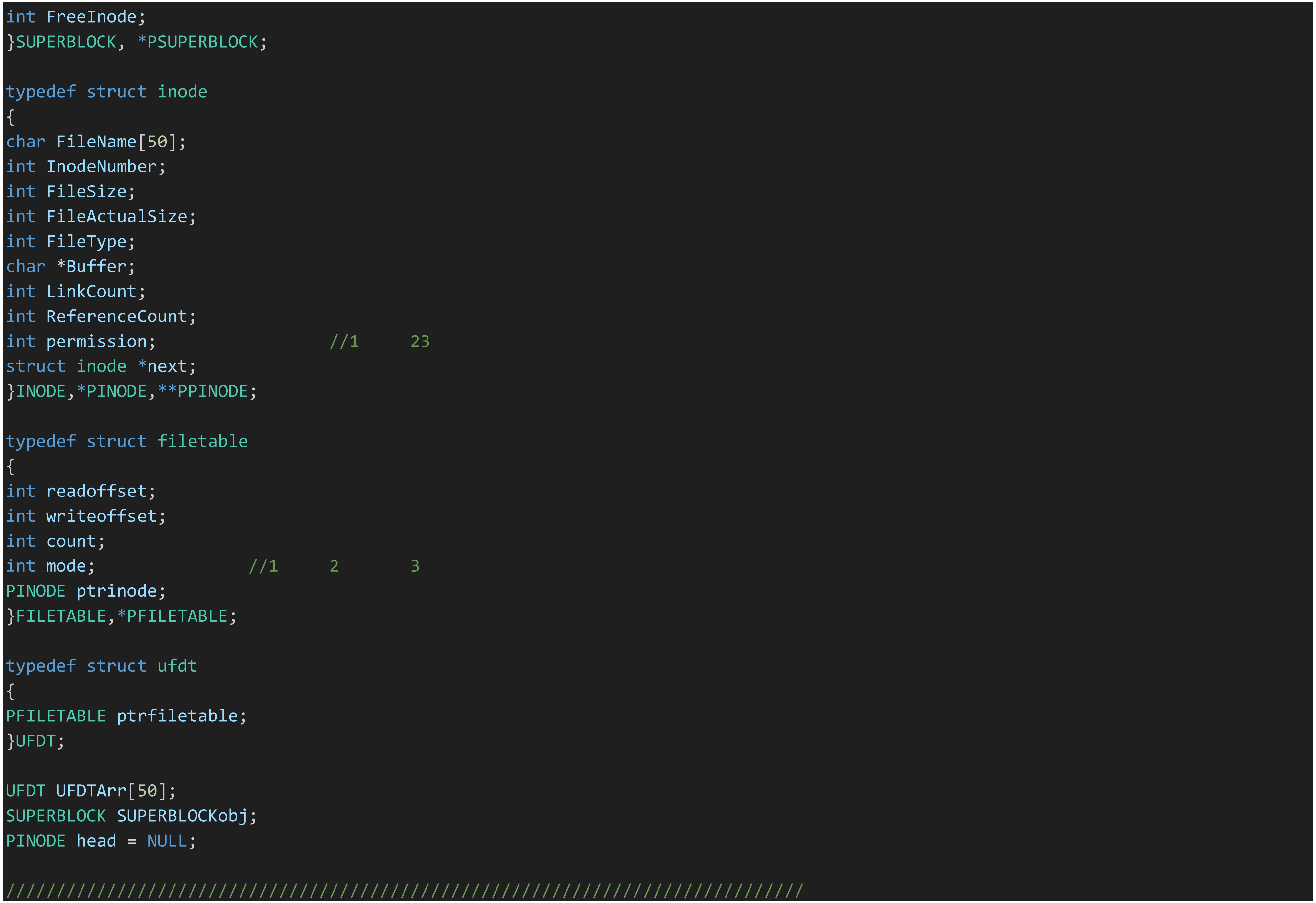
➢ Execution Flow :

- The CVFS is initialise the superblock and DILB
- The main loop reads command from user
- Based on the command , the corresponding function is called

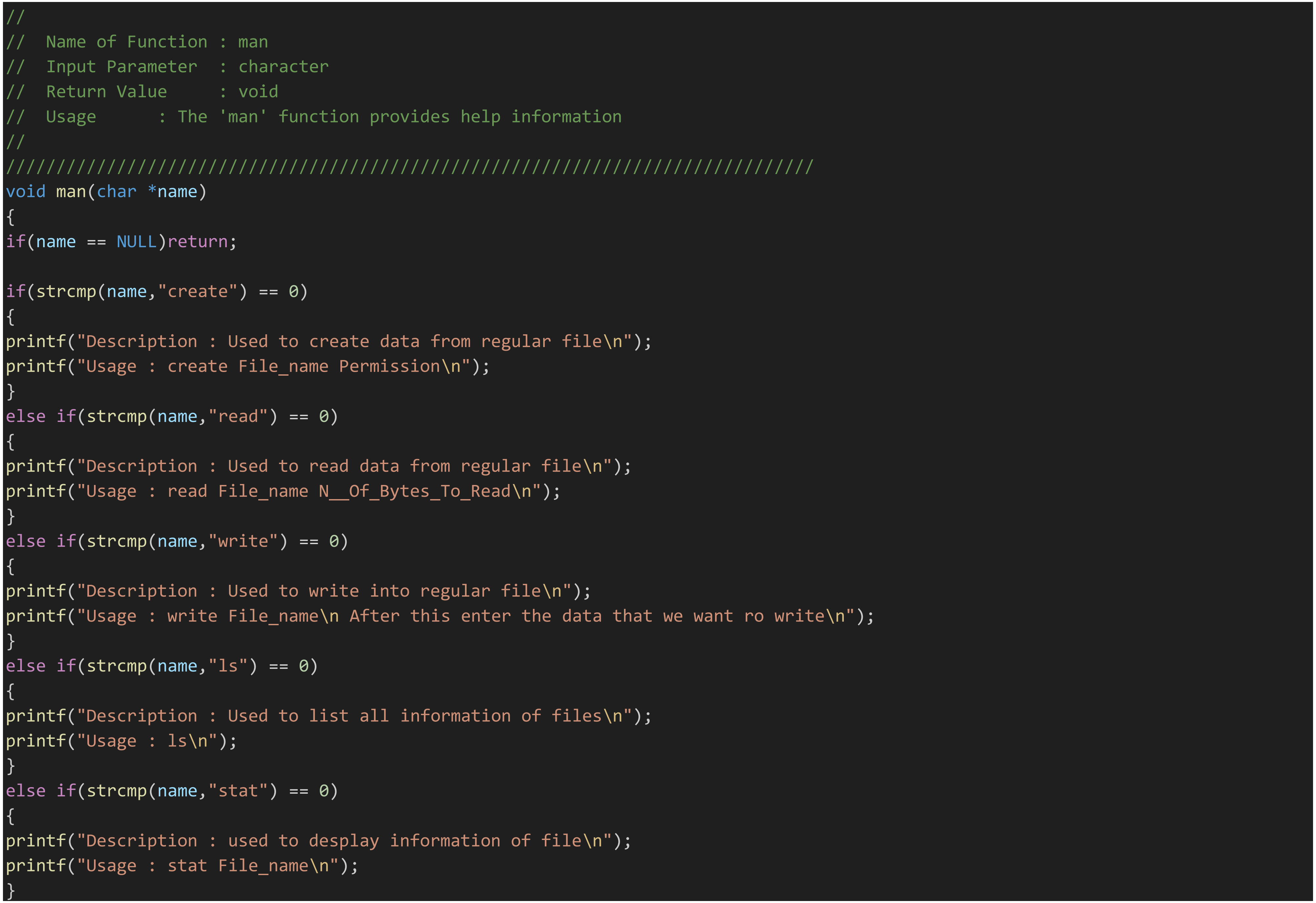
- The function performs the desired operation, handling errors as needed
- The loop continues until the user exists the CVFS

Project Code

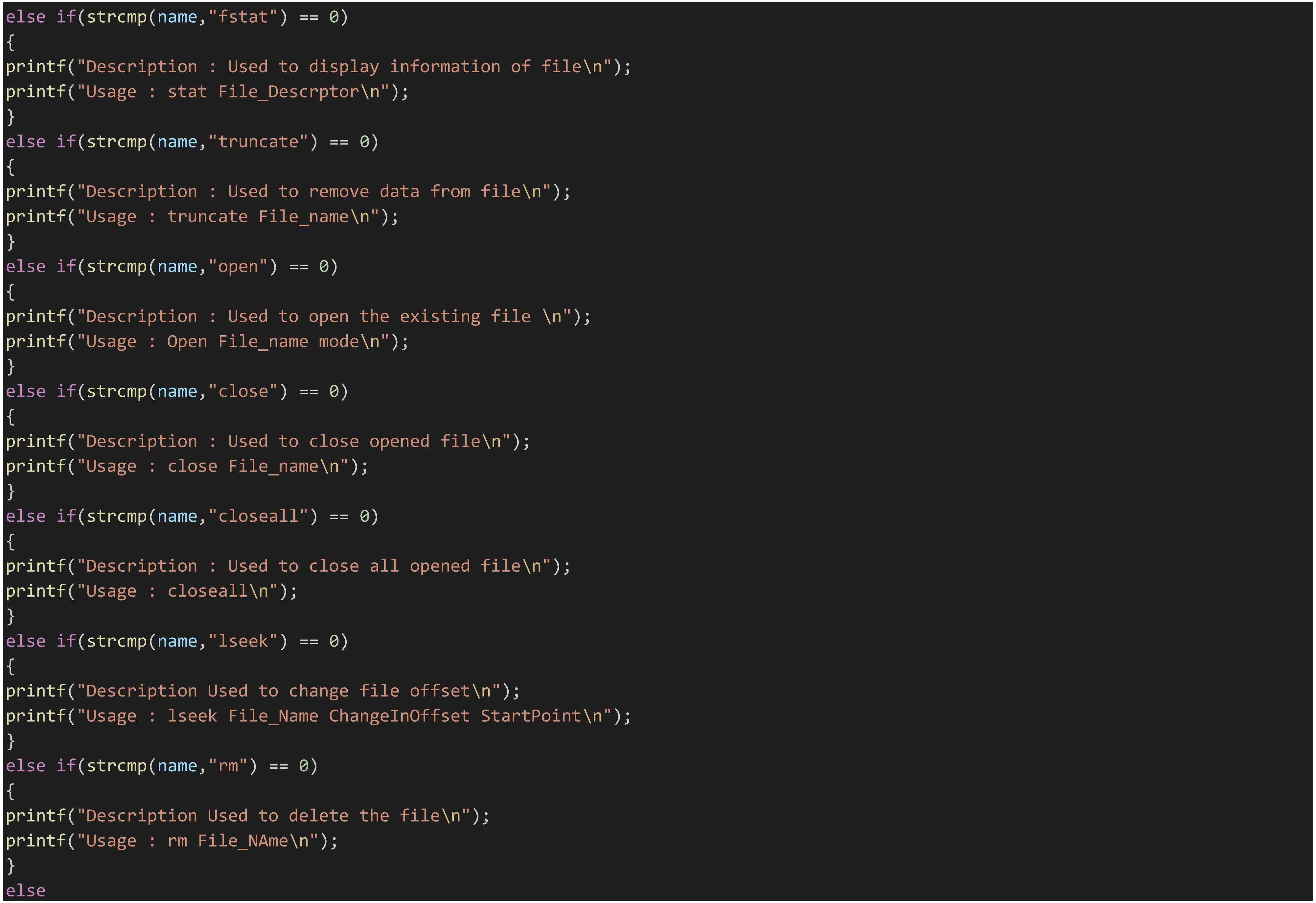
```
////////////////////////////////////////////////////////////////////////
//
//  Project Name : Customised Virtual File System
//
///////////////////////////////////////////////////////////////////////
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<iostream>
//#include<io.h>

#define MAXINODE 5

#define READ 1
#define WRITE 2

#define MAXFILESIZE 150

#define REGULAR 1
#define SPECIAL 2

#define START 0
#define CURRENT 1
#define END 2

typedef struct superblock
{
int TotalInodes;
```



```
int FreeInode;
}SUPERBLOCK, *PSUPERBLOCK;

typedef struct inode
{
char FileName[50];
int InodeNumber;
int FileSize;
int FileActualSize;
int FileType;
char *Buffer;
int LinkCount;
int ReferenceCount;
int permission;                 //1     23
struct inode *next;
}INODE,*PINODE,**PPINODE;

typedef struct filetable
{
int readoffset;
int writeoffset;
int count;
int mode;               //1     2       3
PINODE ptrinode;
}FILETABLE,*PFILETABLE;

typedef struct ufdt
{
PFILETABLE ptrfiletable;
}UFDT;

UFDT UFDTArr[50];
SUPERBLOCK SUPERBLOCKobj;
PINODE head = NULL;

///////////////////////////////////////////////////////////////////////////////
```

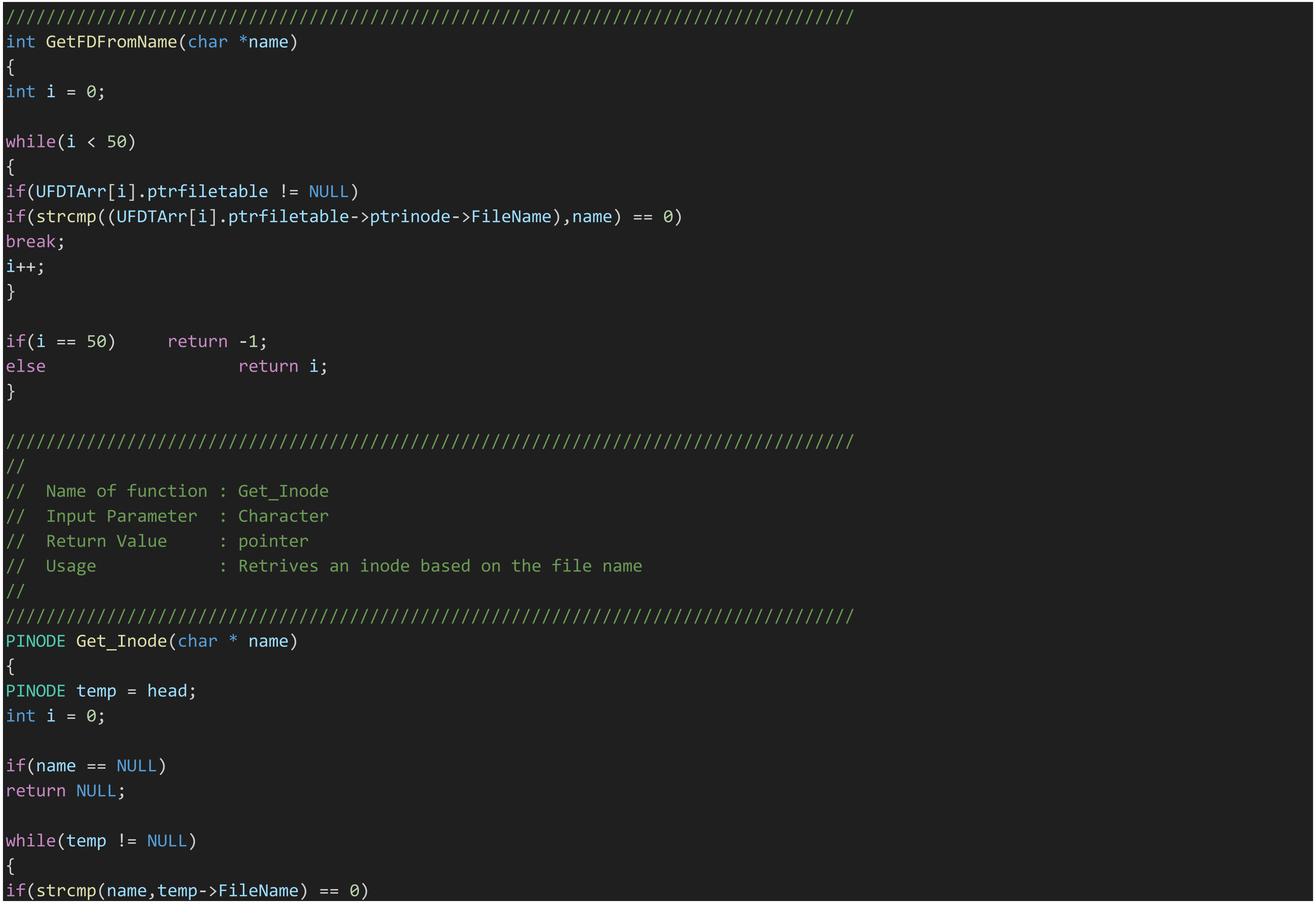


```
//
//  Name of Function : man
//  Input Parameter  : character
//  Return Value     : void
//  Usage      : The 'man' function provides help information
//
////////////////////////////////////////////////////////////////////////////////
void man(char *name)
{
if(name == NULL)return;

if(strcmp(name,"create") == 0)
{
printf("Description : Used to create data from regular file\n");
printf("Usage : create File_name Permission\n");
}
else if(strcmp(name,"read") == 0)
{
printf("Description : Used to read data from regular file\n");
printf("Usage : read File_name N__Of_Bytes_To_Read\n");
}
else if(strcmp(name,"write") == 0)
{
printf("Description : Used to write into regular file\n");
printf("Usage : write File_name\n After this enter the data that we want ro write\n");
}
else if(strcmp(name,"ls") == 0)
{
printf("Description : Used to list all information of files\n");
printf("Usage : ls\n");
}
else if(strcmp(name,"stat") == 0)
{
printf("Description : used to desplay information of file\n");
printf("Usage : stat File_name\n");
}
```

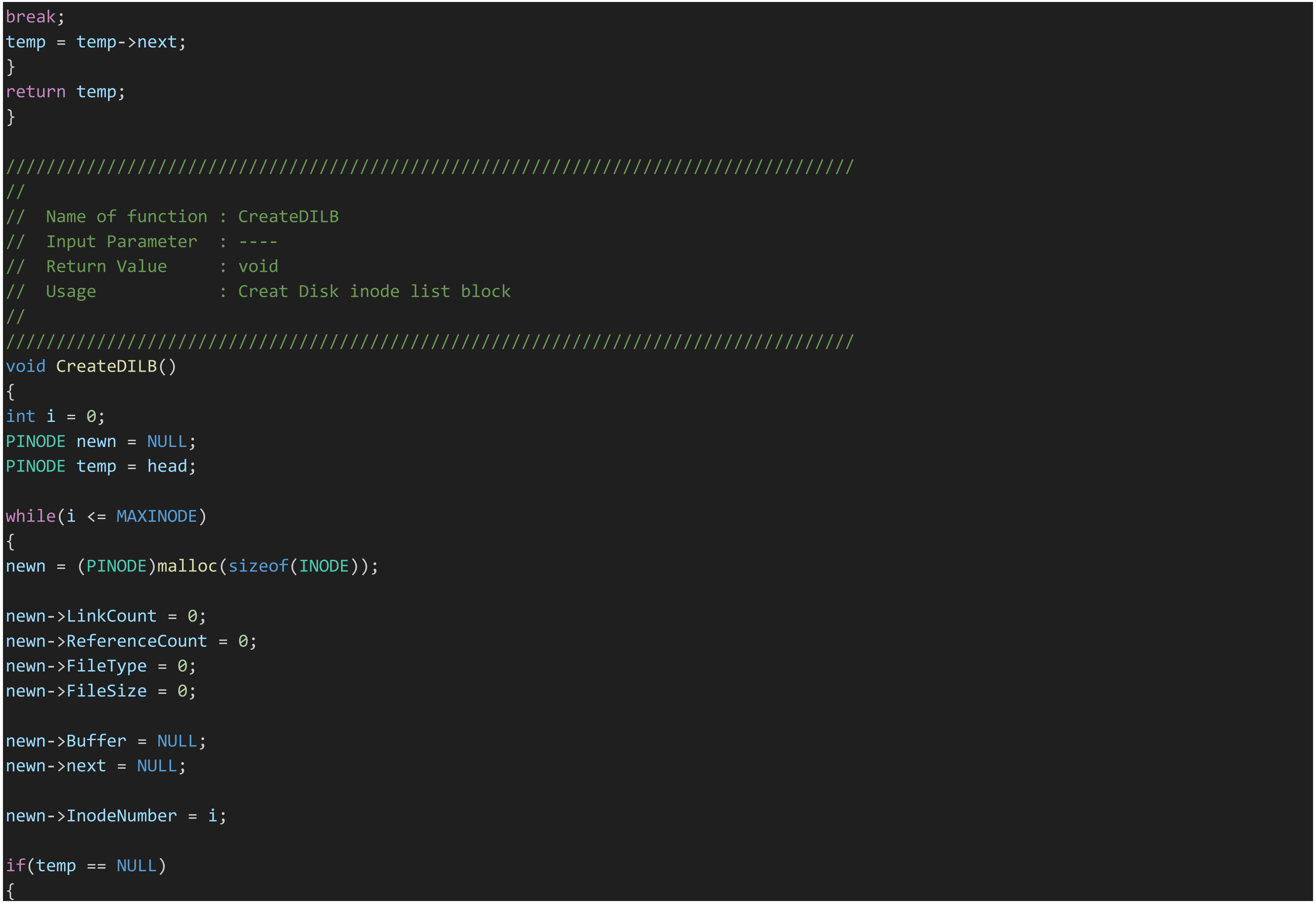


```c
else if(strcmp(name,"fstat") == 0)
{
printf("Description : Used to display information of file\n");
printf("Usage : stat File_Descrptor\n");
}
else if(strcmp(name,"truncate") == 0)
{
printf("Description : Used to remove data from file\n");
printf("Usage : truncate File_name\n");
}
else if(strcmp(name,"open") == 0)
{
printf("Description : Used to open the existing file \n");
printf("Usage : Open File_name mode\n");
}
else if(strcmp(name,"close") == 0)
{
printf("Description : Used to close opened file\n");
printf("Usage : close File_name\n");
}
else if(strcmp(name,"closeall") == 0)
{
printf("Description : Used to close all opened file\n");
printf("Usage : closeall\n");
}
else if(strcmp(name,"lseek") == 0)
{
printf("Description Used to change file offset\n");
printf("Usage : lseek File_Name ChangeInOffset StartPoint\n");
}
else if(strcmp(name,"rm") == 0)
{
printf("Description Used to delete the file\n");
printf("Usage : rm File_NAme\n");
}
else
```

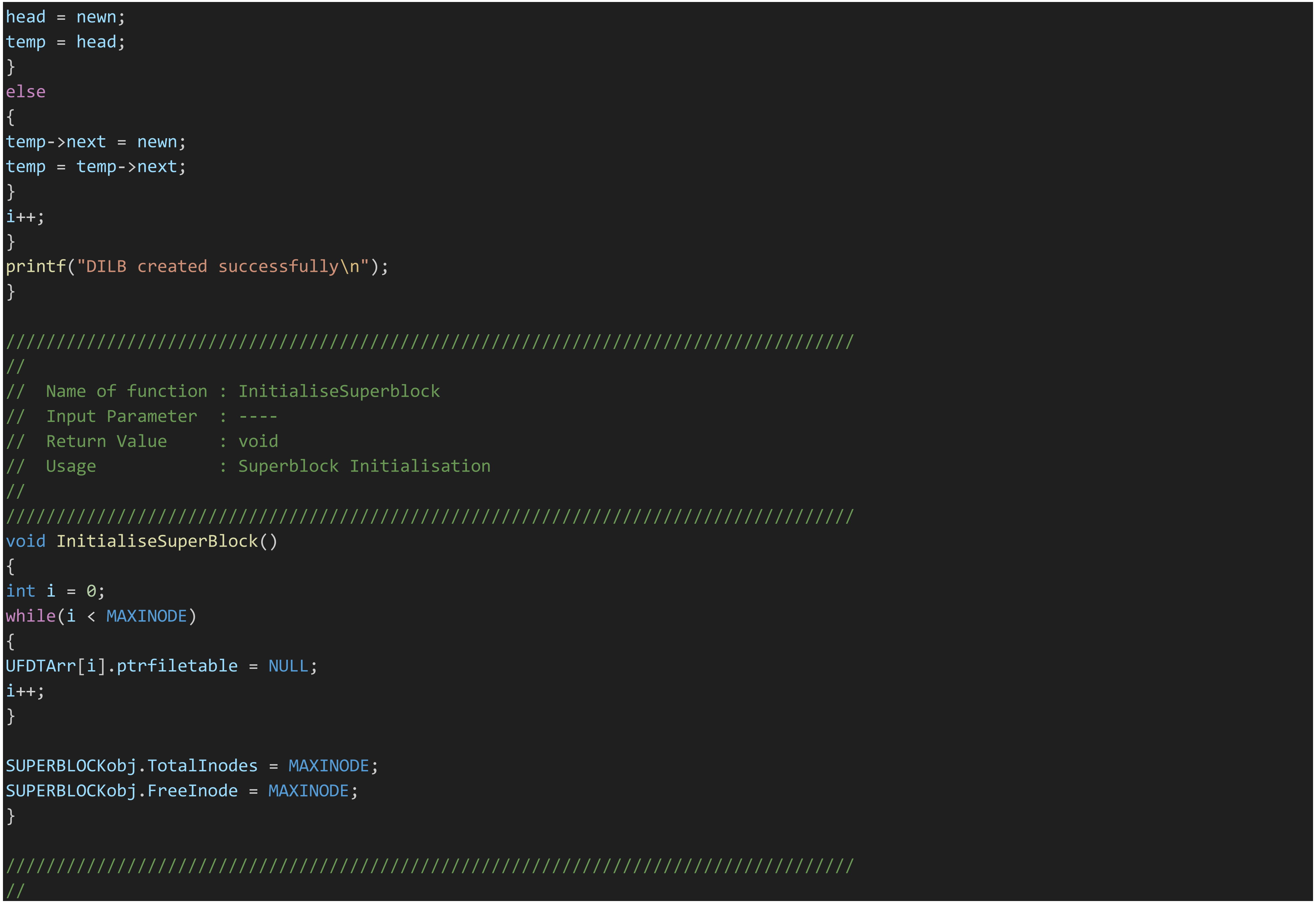
```
{
printf("ERROR : No manual entry available.\n");
}
}

///////////////////////////////////////////////////////////////////////////////
//
//  Name of Function : Display
//  Input Parameter  : ---
//  Return Value     : void
//  Usage            : This function uses printf statements to display a
//                       list of commands along their description
////////////////////////////////////////////////////////////////////////////////
void DisplayHelp()
{
printf("ls : To list out all file\n");
printf("clear : To clear console\n");
printf("open : To open the file\n");
printf("close : To close the file\n");
printf("closeall : To close al opend file\n");
printf("read : To read the contents of the file\n");
printf("write : To write the contents in to file\n");
printf("exit : To terminate file systm\n");
printf("stat : To Display information of file using name \n");
printf("fstat : To Display information of file using file descriptor\n");
printf("truncate : To remove all data from file\n");
printf("rm : To Delete the file");
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : GetFDFromName
//  Input Parameter  : Character
//  Return Value     : int
//  Usage            : Gets file descriptor from the file name
//
```



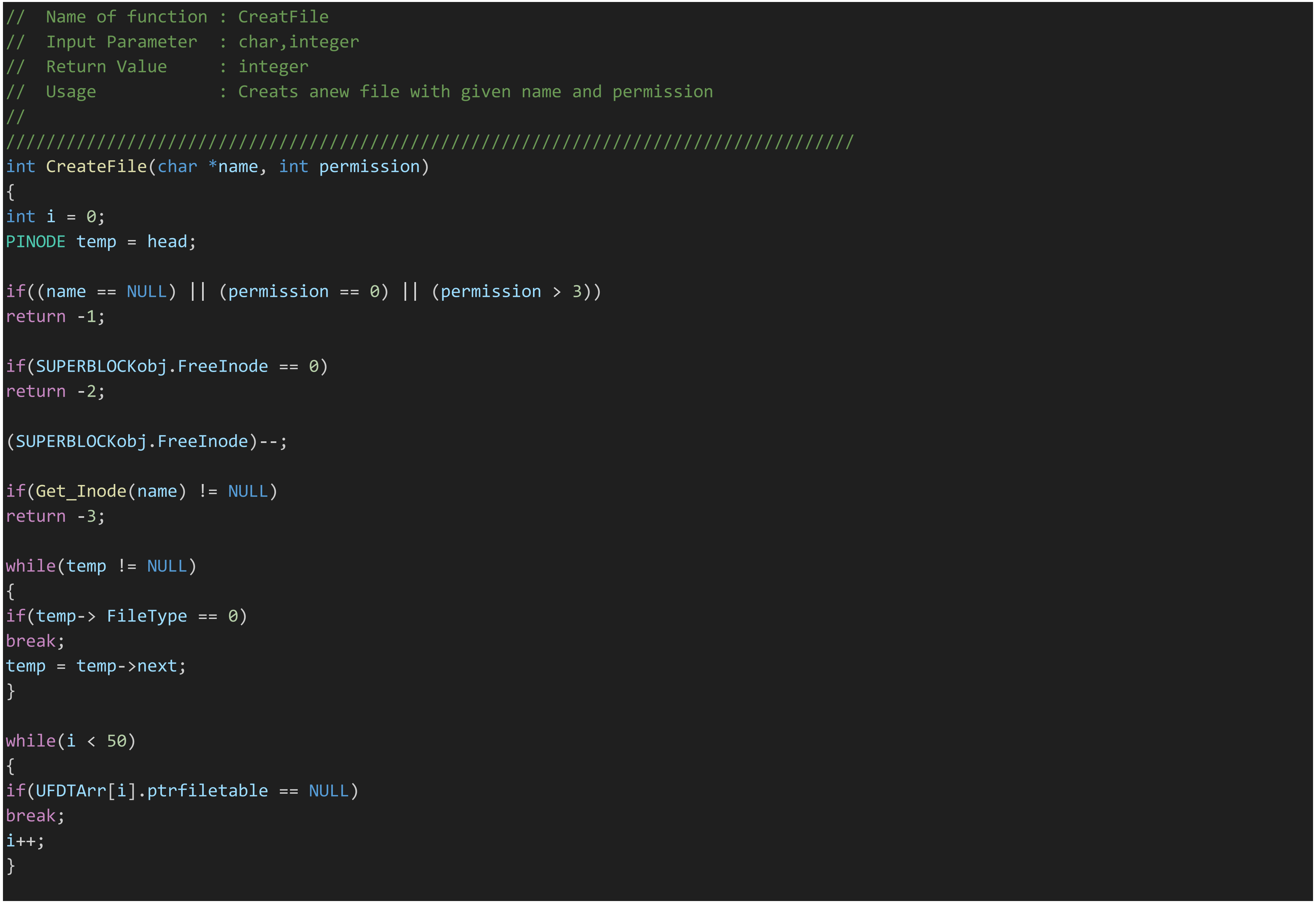
```
////////////////////////////////////////////////////////////////////////////////////
int GetFDFromName(char *name)
{
int i = 0;

while(i < 50)
{
if(UFDTArr[i].ptrfiletable != NULL)
if(strcmp((UFDTArr[i].ptrfiletable->ptrinode->FileName),name) == 0)
break;
i++;
}

if(i == 50)     return -1;
else                   return i;
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : Get_Inode
//  Input Parameter  : Character
//  Return Value     : pointer
//  Usage            : Retrives an inode based on the file name
//
////////////////////////////////////////////////////////////////////////////////////
PINODE Get_Inode(char * name)
{
PINODE temp = head;
int i = 0;

if(name == NULL)
return NULL;

while(temp != NULL)
{
if(strcmp(name,temp->FileName) == 0)
```



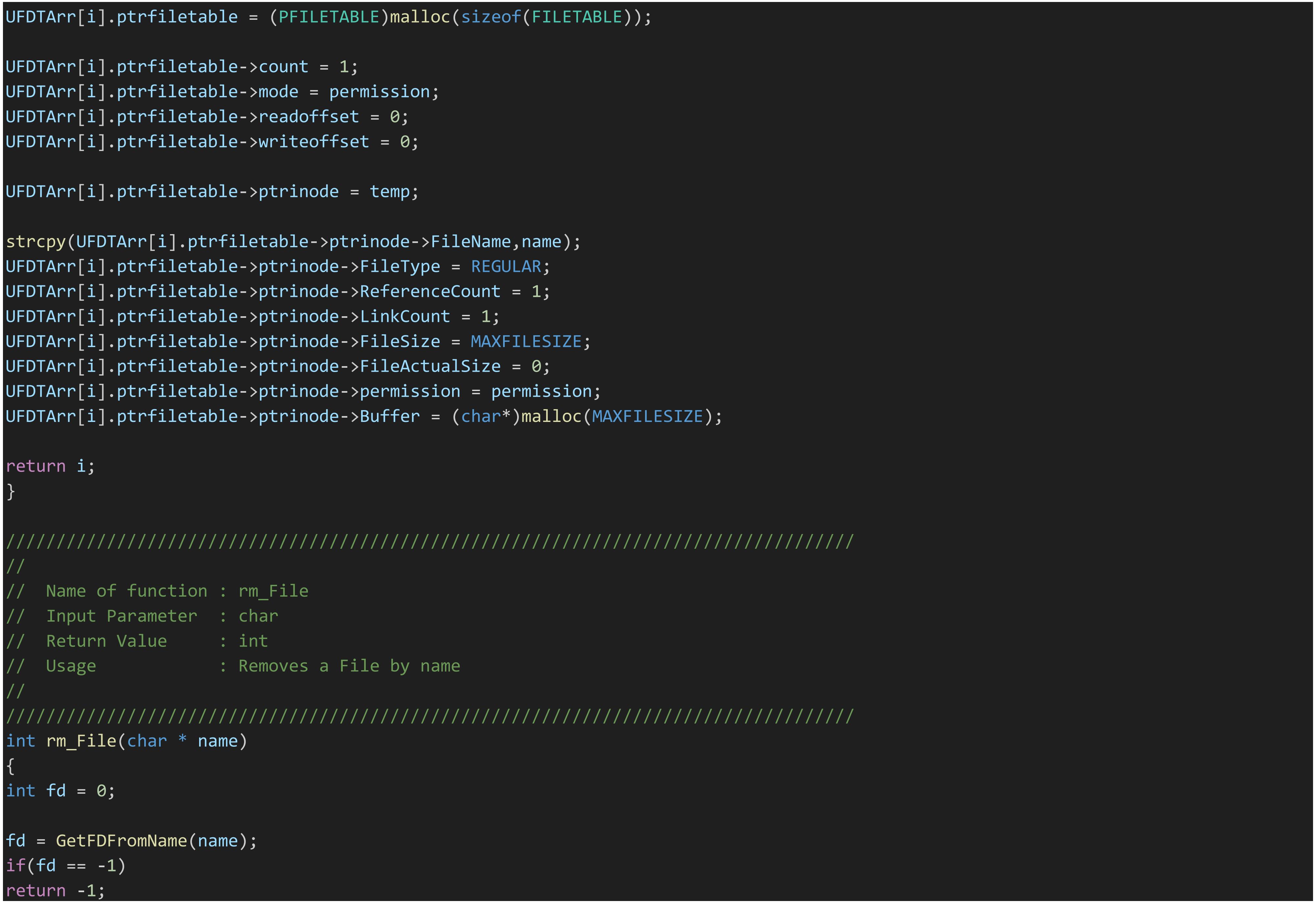
```
break;
temp = temp->next;
}
return temp;
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : CreateDILB
//  Input Parameter  : ----
//  Return Value     : void
//  Usage            : Creat Disk inode list block
//
////////////////////////////////////////////////////////////////////////////////////
void CreateDILB()
{
int i = 0;
PINODE newn = NULL;
PINODE temp = head;

while(i <= MAXINODE)
{
newn = (PINODE)malloc(sizeof(INODE));

newn->LinkCount = 0;
newn->ReferenceCount = 0;
newn->FileType = 0;
newn->FileSize = 0;

newn->Buffer = NULL;
newn->next = NULL;

newn->InodeNumber = i;

if(temp == NULL)
{
```



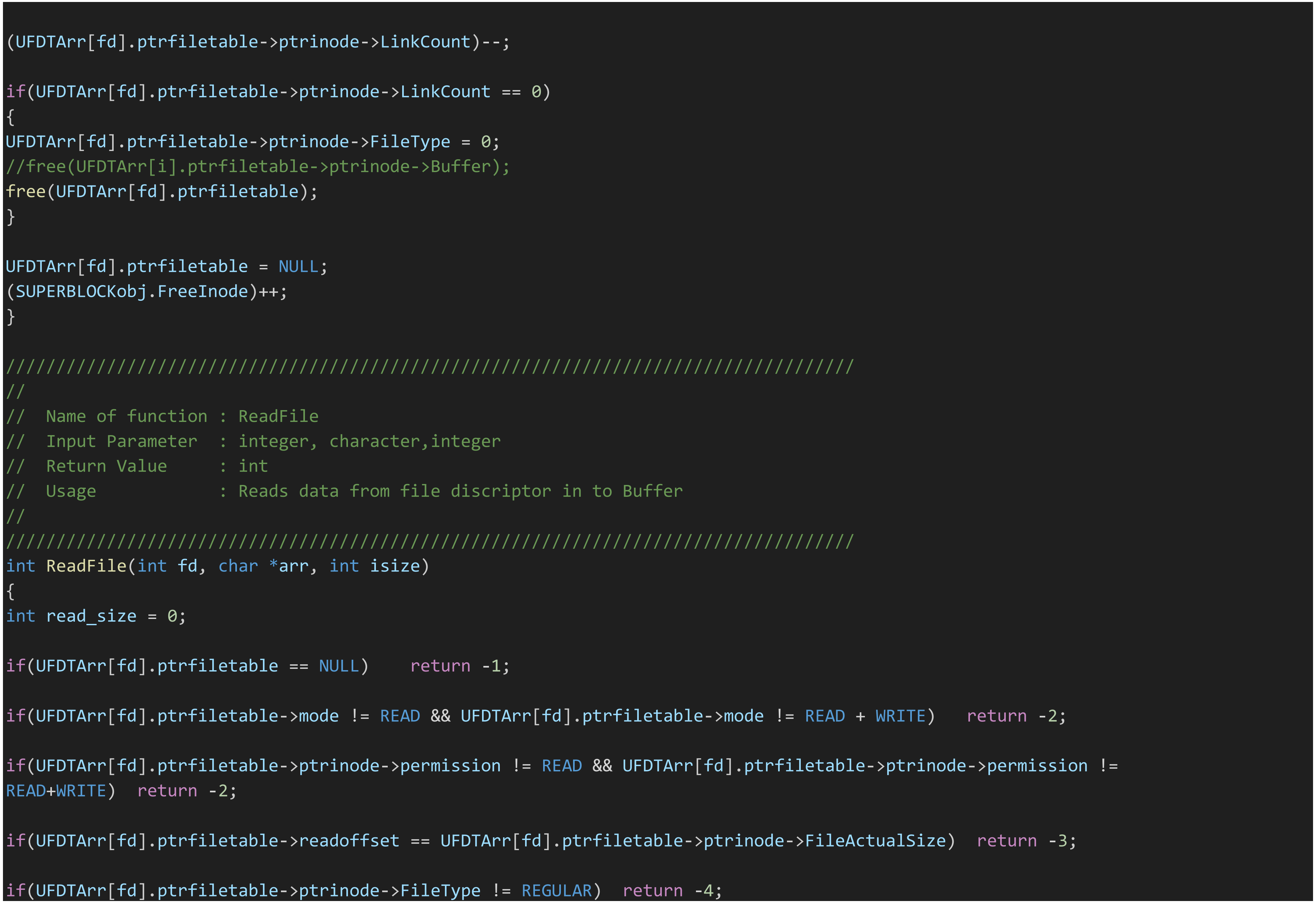
```c
head = newn;
temp = head;
}
else
{
temp->next = newn;
temp = temp->next;
}
i++;
}
printf("DILB created successfully\n");
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : InitialiseSuperblock
//  Input Parameter  : ----
//  Return Value     : void
//  Usage            : Superblock Initialisation
//
////////////////////////////////////////////////////////////////////////////////////
void InitialiseSuperBlock()
{
int i = 0;
while(i < MAXINODE)
{
UFDTArr[i].ptrfiletable = NULL;
i++;
}

SUPERBLOCKobj.TotalInodes = MAXINODE;
SUPERBLOCKobj.FreeInode = MAXINODE;
}

////////////////////////////////////////////////////////////////////////////////////
//
```



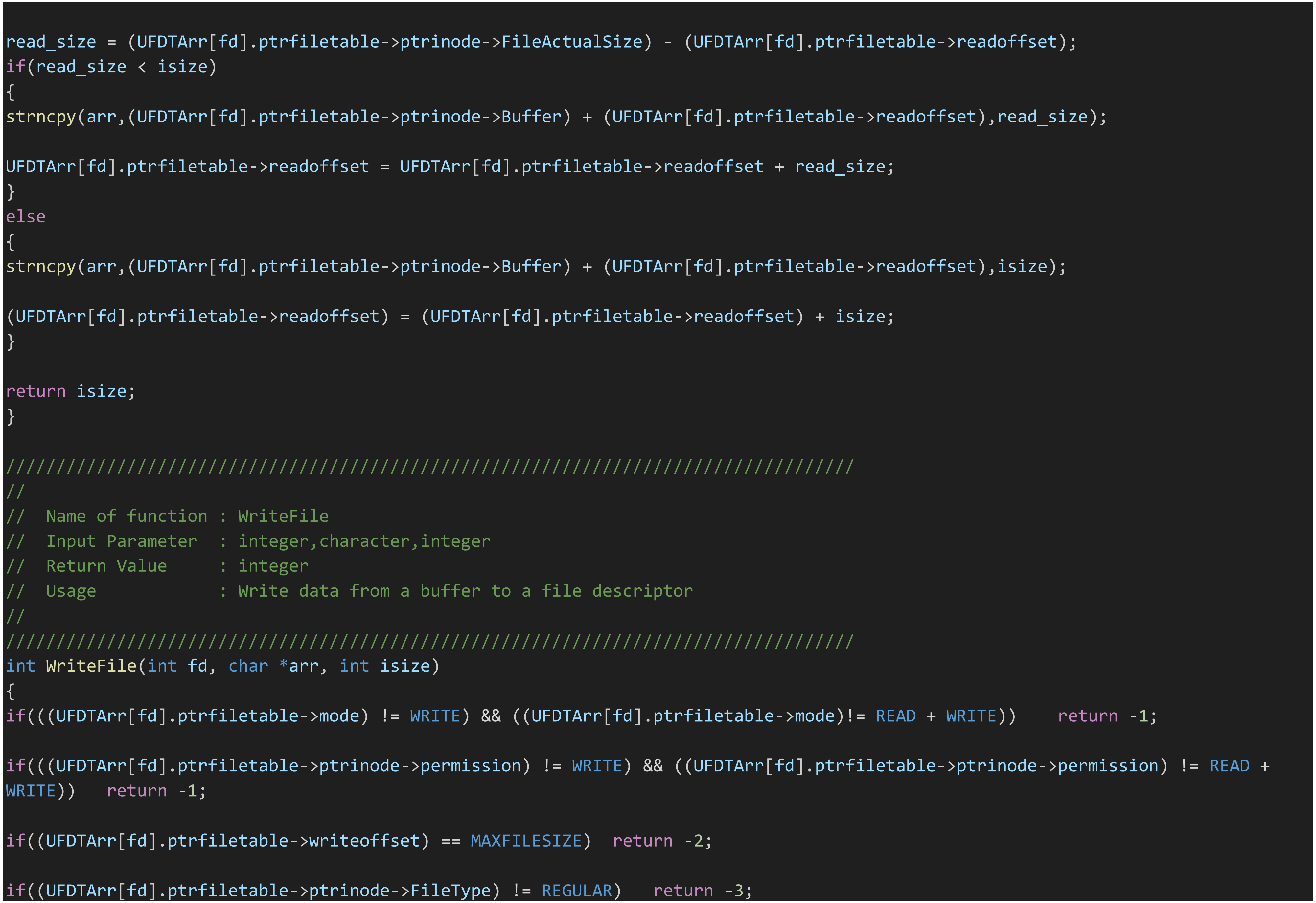
```
//  Name of function : CreatFile
//  Input Parameter  : char,integer
//  Return Value     : integer
//  Usage            : Creats anew file with given name and permission
//
////////////////////////////////////////////////////////////////////////////////////
int CreateFile(char *name, int permission)
{
int i = 0;
PINODE temp = head;

if((name == NULL) || (permission == 0) || (permission > 3))
return -1;

if(SUPERBLOCKobj.FreeInode == 0)
return -2;

(SUPERBLOCKobj.FreeInode)--;

if(Get_Inode(name) != NULL)
return -3;

while(temp != NULL)
{
if(temp-> FileType == 0)
break;
temp = temp->next;
}

while(i < 50)
{
if(UFDTArr[i].ptrfiletable == NULL)
break;
i++;
}
```



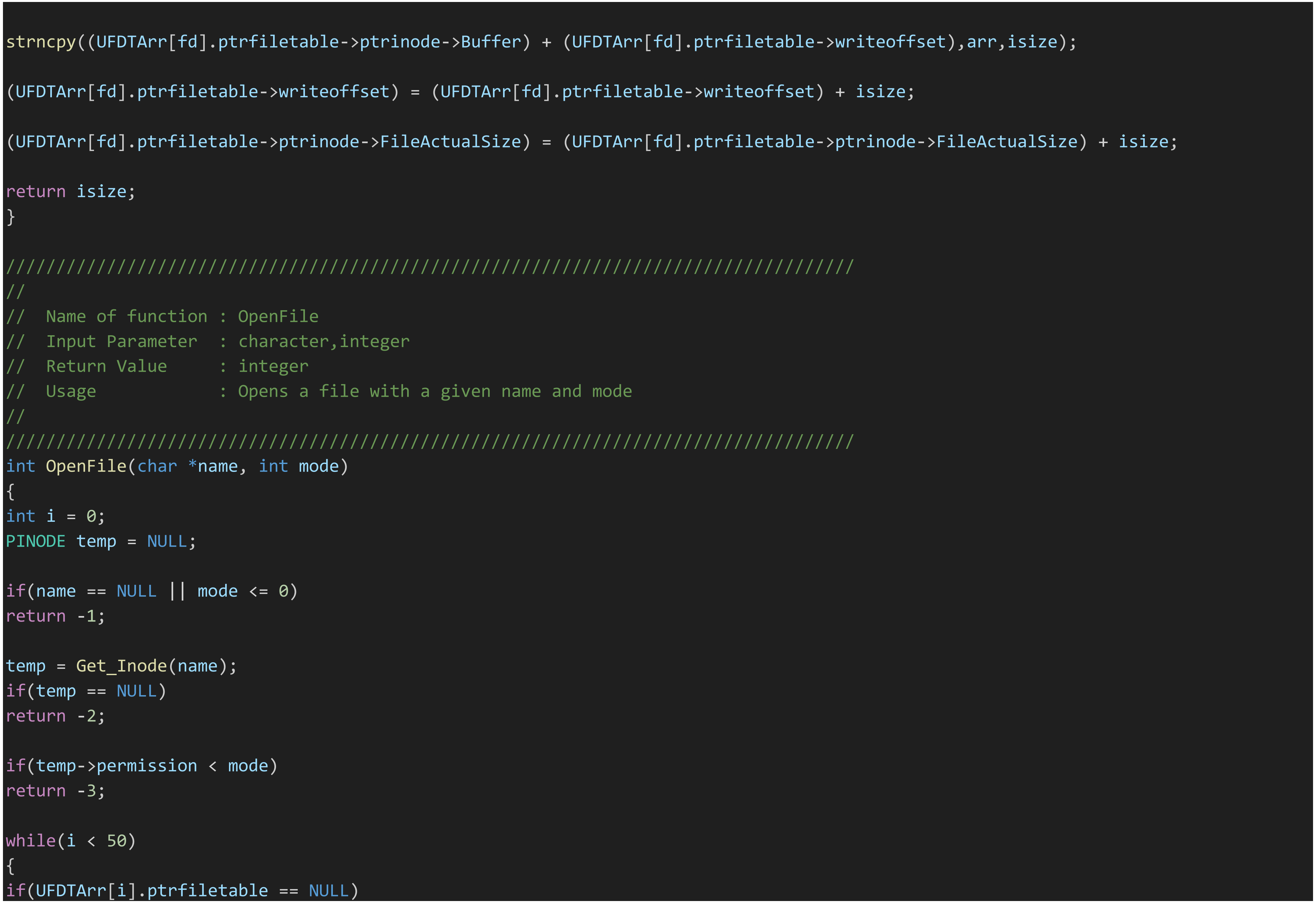
```
UFDTArr[i].ptrfiletable = (PFILETABLE)malloc(sizeof(FILETABLE));

UFDTArr[i].ptrfiletable->count = 1;
UFDTArr[i].ptrfiletable->mode = permission;
UFDTArr[i].ptrfiletable->readoffset = 0;
UFDTArr[i].ptrfiletable->writeoffset = 0;

UFDTArr[i].ptrfiletable->ptrinode = temp;

strcpy(UFDTArr[i].ptrfiletable->ptrinode->FileName,name);
UFDTArr[i].ptrfiletable->ptrinode->FileType = REGULAR;
UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount = 1;
UFDTArr[i].ptrfiletable->ptrinode->LinkCount = 1;
UFDTArr[i].ptrfiletable->ptrinode->FileSize = MAXFILESIZE;
UFDTArr[i].ptrfiletable->ptrinode->FileActualSize = 0;
UFDTArr[i].ptrfiletable->ptrinode->permission = permission;
UFDTArr[i].ptrfiletable->ptrinode->Buffer = (char*)malloc(MAXFILESIZE);

return i;
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : rm_File
//  Input Parameter  : char
//  Return Value     : int
//  Usage            : Removes a File by name
//
////////////////////////////////////////////////////////////////////////////////////
int rm_File(char * name)
{
int fd = 0;

fd = GetFDFromName(name);
if(fd == -1)
return -1;
```



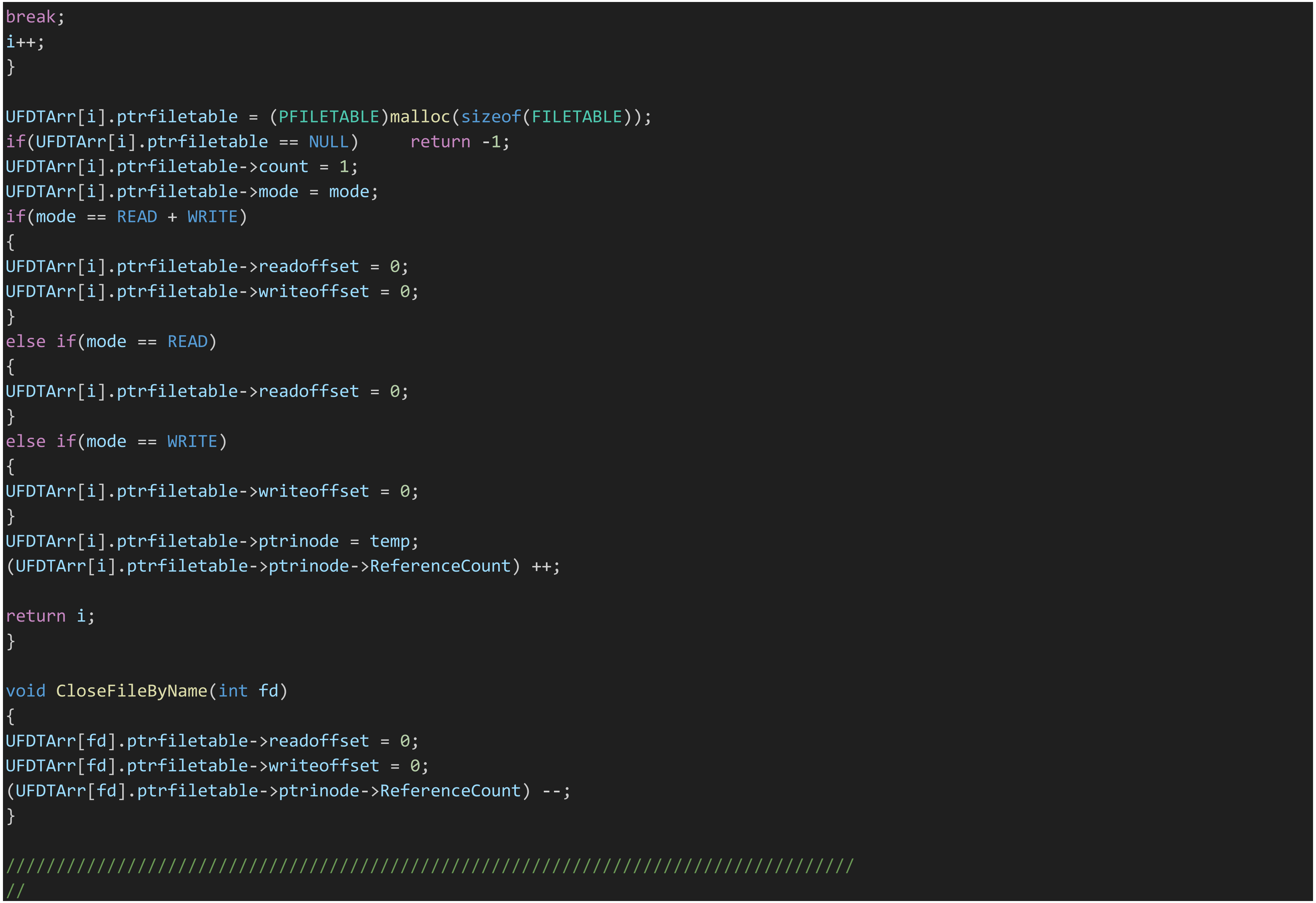
```
(UFDTArr[fd].ptrfiletable->ptrinode->LinkCount)--;

if(UFDTArr[fd].ptrfiletable->ptrinode->LinkCount == 0)
{
UFDTArr[fd].ptrfiletable->ptrinode->FileType = 0;
//free(UFDTArr[i].ptrfiletable->ptrinode->Buffer);
free(UFDTArr[fd].ptrfiletable);
}

UFDTArr[fd].ptrfiletable = NULL;
(SUPERBLOCKobj.FreeInode)++;
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : ReadFile
//  Input Parameter  : integer, character,integer
//  Return Value     : int
//  Usage            : Reads data from file discriptor in to Buffer
//
////////////////////////////////////////////////////////////////////////////////////
int ReadFile(int fd, char *arr, int isize)
{
int read_size = 0;

if(UFDTArr[fd].ptrfiletable == NULL)    return -1;

if(UFDTArr[fd].ptrfiletable->mode != READ && UFDTArr[fd].ptrfiletable->mode != READ + WRITE)   return -2;

if(UFDTArr[fd].ptrfiletable->ptrinode->permission != READ && UFDTArr[fd].ptrfiletable->ptrinode->permission !=
READ+WRITE)  return -2;

if(UFDTArr[fd].ptrfiletable->readoffset == UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize)  return -3;

if(UFDTArr[fd].ptrfiletable->ptrinode->FileType != REGULAR)  return -4;
```



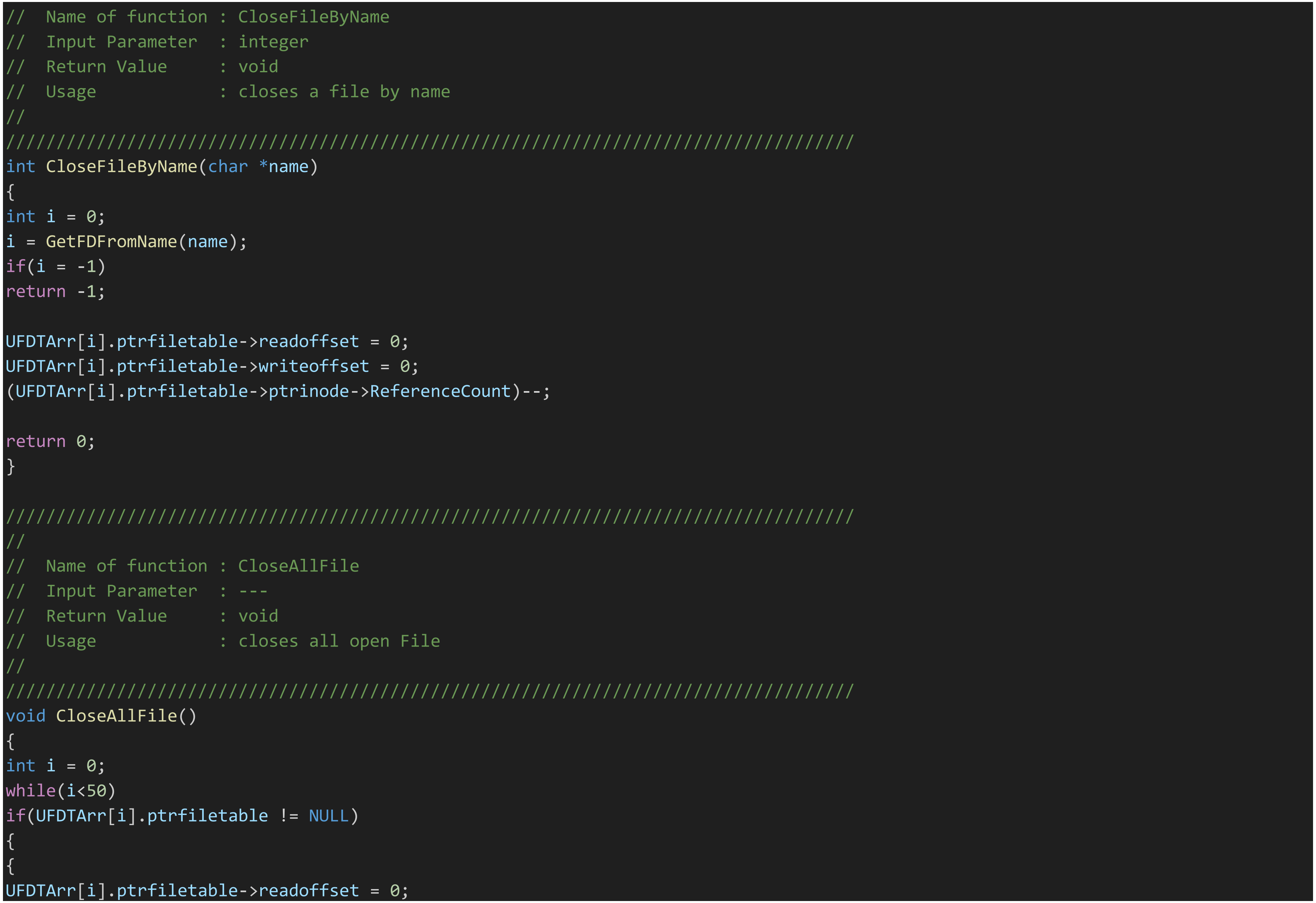
```
read_size = (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) - (UFDTArr[fd].ptrfiletable->readoffset);
if(read_size < isize)
{
strncpy(arr,(UFDTArr[fd].ptrfiletable->ptrinode->Buffer) + (UFDTArr[fd].ptrfiletable->readoffset),read_size);

UFDTArr[fd].ptrfiletable->readoffset = UFDTArr[fd].ptrfiletable->readoffset + read_size;
}
else
{
strncpy(arr,(UFDTArr[fd].ptrfiletable->ptrinode->Buffer) + (UFDTArr[fd].ptrfiletable->readoffset),isize);

(UFDTArr[fd].ptrfiletable->readoffset) = (UFDTArr[fd].ptrfiletable->readoffset) + isize;
}

return isize;
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : WriteFile
//  Input Parameter  : integer,character,integer
//  Return Value     : integer
//  Usage            : Write data from a buffer to a file descriptor
//
////////////////////////////////////////////////////////////////////////////////////
int WriteFile(int fd, char *arr, int isize)
{
if(((UFDTArr[fd].ptrfiletable->mode) != WRITE) && ((UFDTArr[fd].ptrfiletable->mode)!= READ + WRITE))    return -1;

if(((UFDTArr[fd].ptrfiletable->ptrinode->permission) != WRITE) && ((UFDTArr[fd].ptrfiletable->ptrinode->permission) != READ +
WRITE))   return -1;

if((UFDTArr[fd].ptrfiletable->writeoffset) == MAXFILESIZE)  return -2;

if((UFDTArr[fd].ptrfiletable->ptrinode->FileType) != REGULAR)   return -3;
```



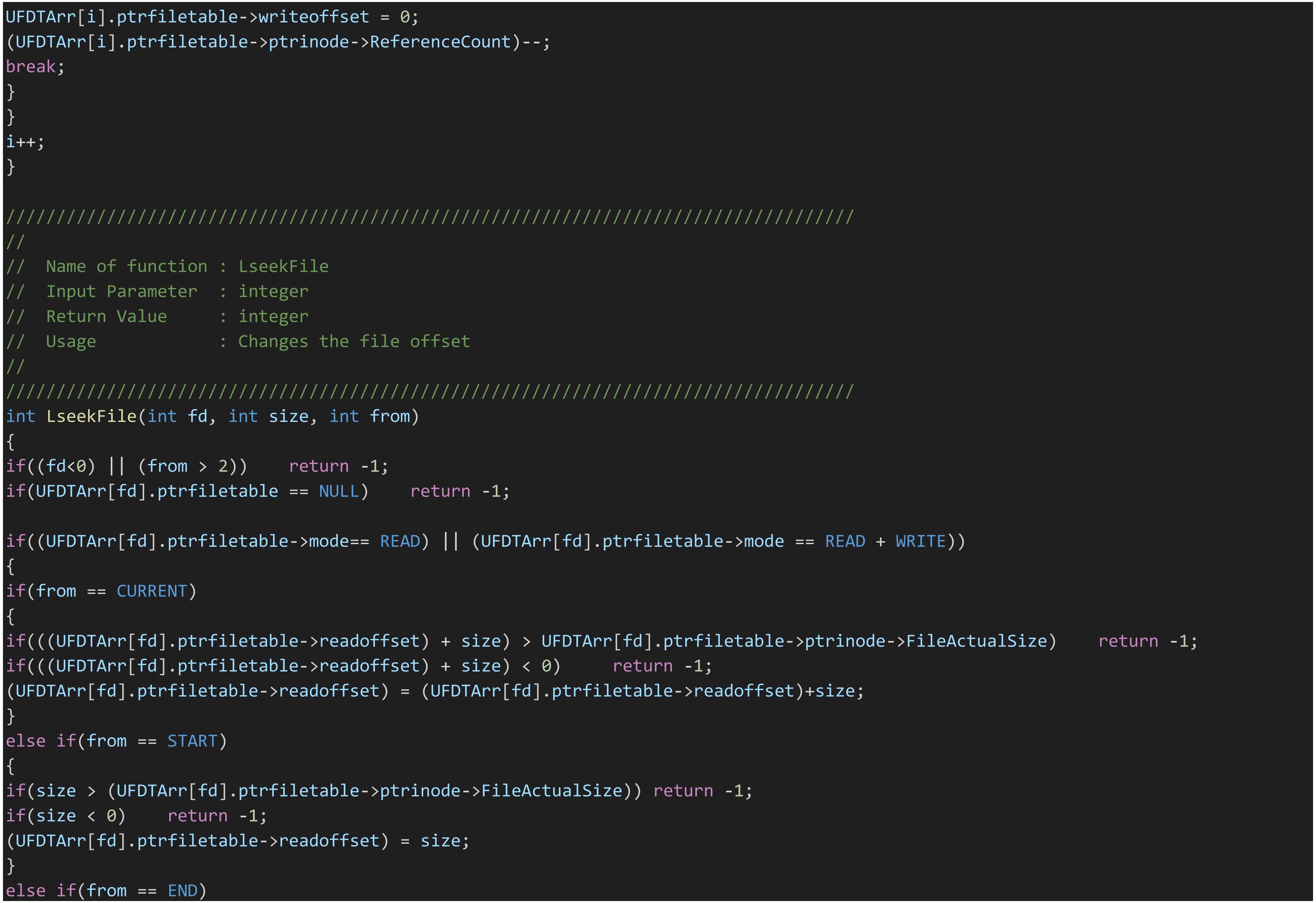
```
strncpy((UFDTArr[fd].ptrfiletable->ptrinode->Buffer) + (UFDTArr[fd].ptrfiletable->writeoffset),arr,isize);

(UFDTArr[fd].ptrfiletable->writeoffset) = (UFDTArr[fd].ptrfiletable->writeoffset) + isize;

(UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) = (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + isize;

return isize;
}

/////////////////////////////////////////////////////////////////////////////////////
//
//   Name of function : OpenFile
//   Input Parameter  : character,integer
//   Return Value     : integer
//   Usage            : Opens a file with a given name and mode
//
/////////////////////////////////////////////////////////////////////////////////////
int OpenFile(char *name, int mode)
{
int i = 0;
PINODE temp = NULL;

if(name == NULL || mode <= 0)
return -1;

temp = Get_Inode(name);
if(temp == NULL)
return -2;

if(temp->permission < mode)
return -3;

while(i < 50)
{
if(UFDTArr[i].ptrfiletable == NULL)
```



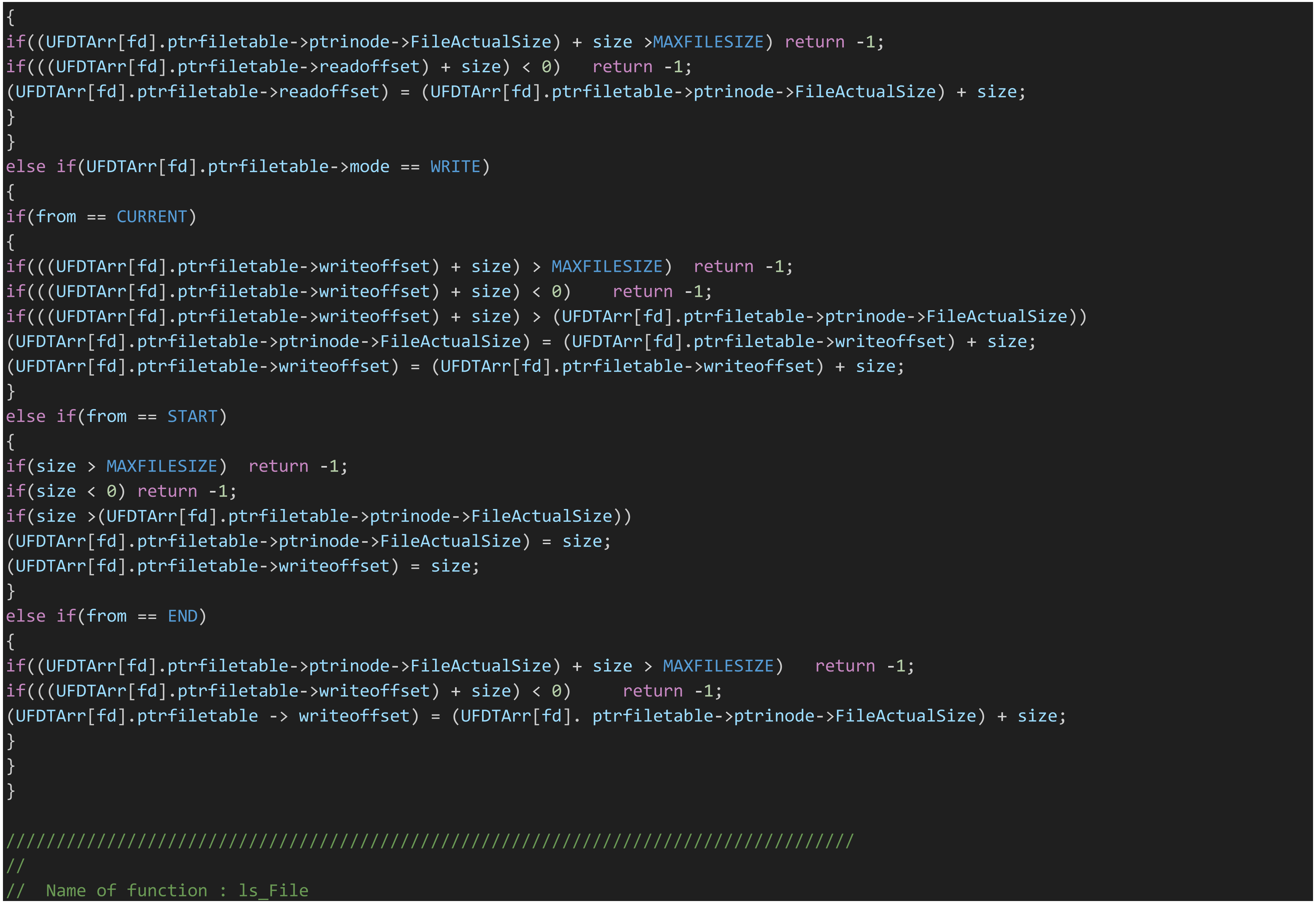
```
break;
i++;
}

UFDTArr[i].ptrfiletable = (PFILETABLE)malloc(sizeof(FILETABLE));
if(UFDTArr[i].ptrfiletable == NULL)     return -1;
UFDTArr[i].ptrfiletable->count = 1;
UFDTArr[i].ptrfiletable->mode = mode;
if(mode == READ + WRITE)
{
UFDTArr[i].ptrfiletable->readoffset = 0;
UFDTArr[i].ptrfiletable->writeoffset = 0;
}
else if(mode == READ)
{
UFDTArr[i].ptrfiletable->readoffset = 0;
}
else if(mode == WRITE)
{
UFDTArr[i].ptrfiletable->writeoffset = 0;
}
UFDTArr[i].ptrfiletable->ptrinode = temp;
(UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount) ++;

return i;
}

void CloseFileByName(int fd)
{
UFDTArr[fd].ptrfiletable->readoffset = 0;
UFDTArr[fd].ptrfiletable->writeoffset = 0;
(UFDTArr[fd].ptrfiletable->ptrinode->ReferenceCount) --;
}

////////////////////////////////////////////////////////////////////////////////////
//
```



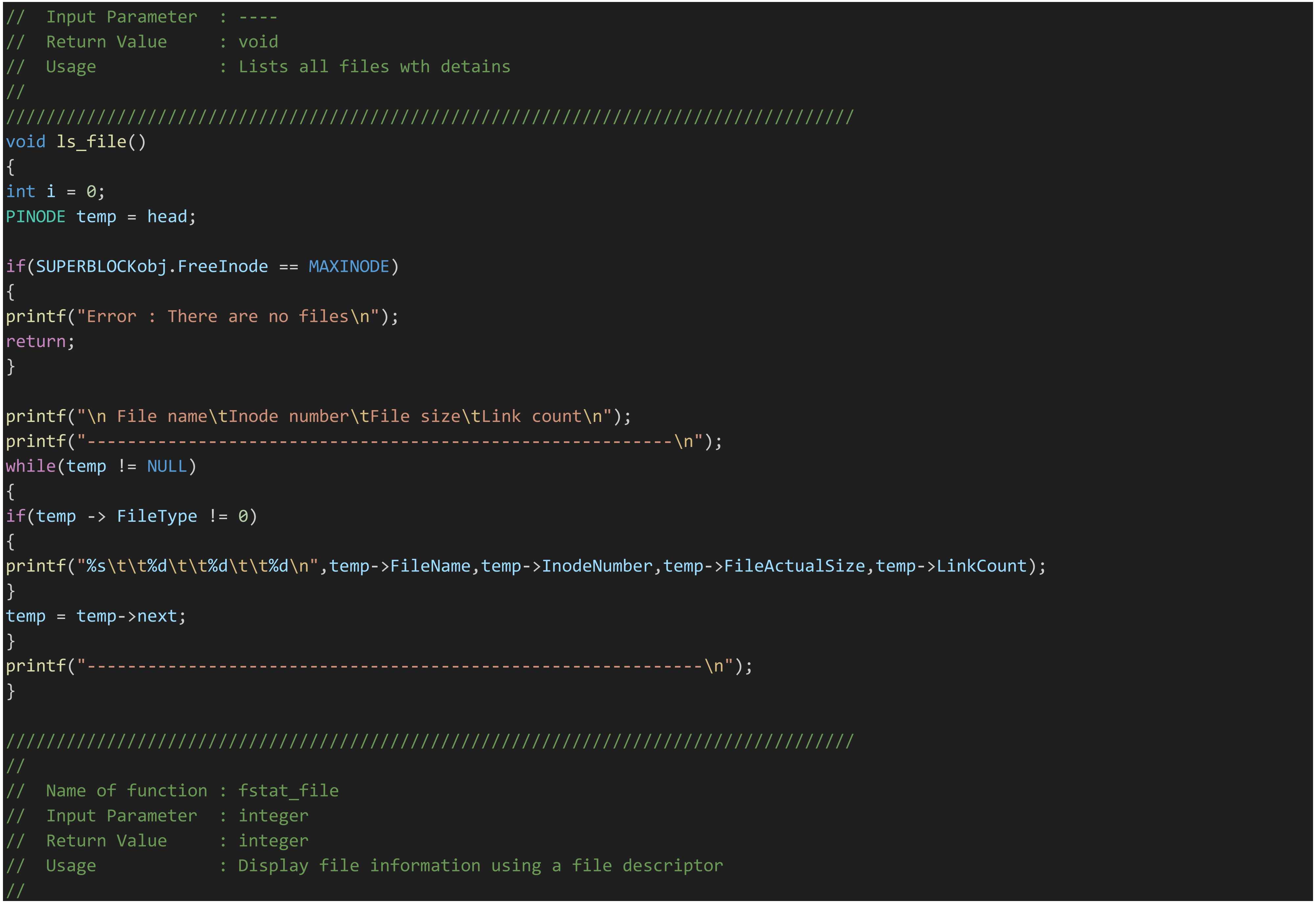
```
//  Name of function : CloseFileByName
//  Input Parameter  : integer
//  Return Value     : void
//  Usage            : closes a file by name
//
////////////////////////////////////////////////////////////////////////////////////
int CloseFileByName(char *name)
{
int i = 0;
i = GetFDFromName(name);
if(i = -1)
return -1;

UFDTArr[i].ptrfiletable->readoffset = 0;
UFDTArr[i].ptrfiletable->writeoffset = 0;
(UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount)--;

return 0;
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : CloseAllFile
//  Input Parameter  : ---
//  Return Value     : void
//  Usage            : closes all open File
//
////////////////////////////////////////////////////////////////////////////////////
void CloseAllFile()
{
int i = 0;
while(i<50)
if(UFDTArr[i].ptrfiletable != NULL)
{
{
UFDTArr[i].ptrfiletable->readoffset = 0;
```



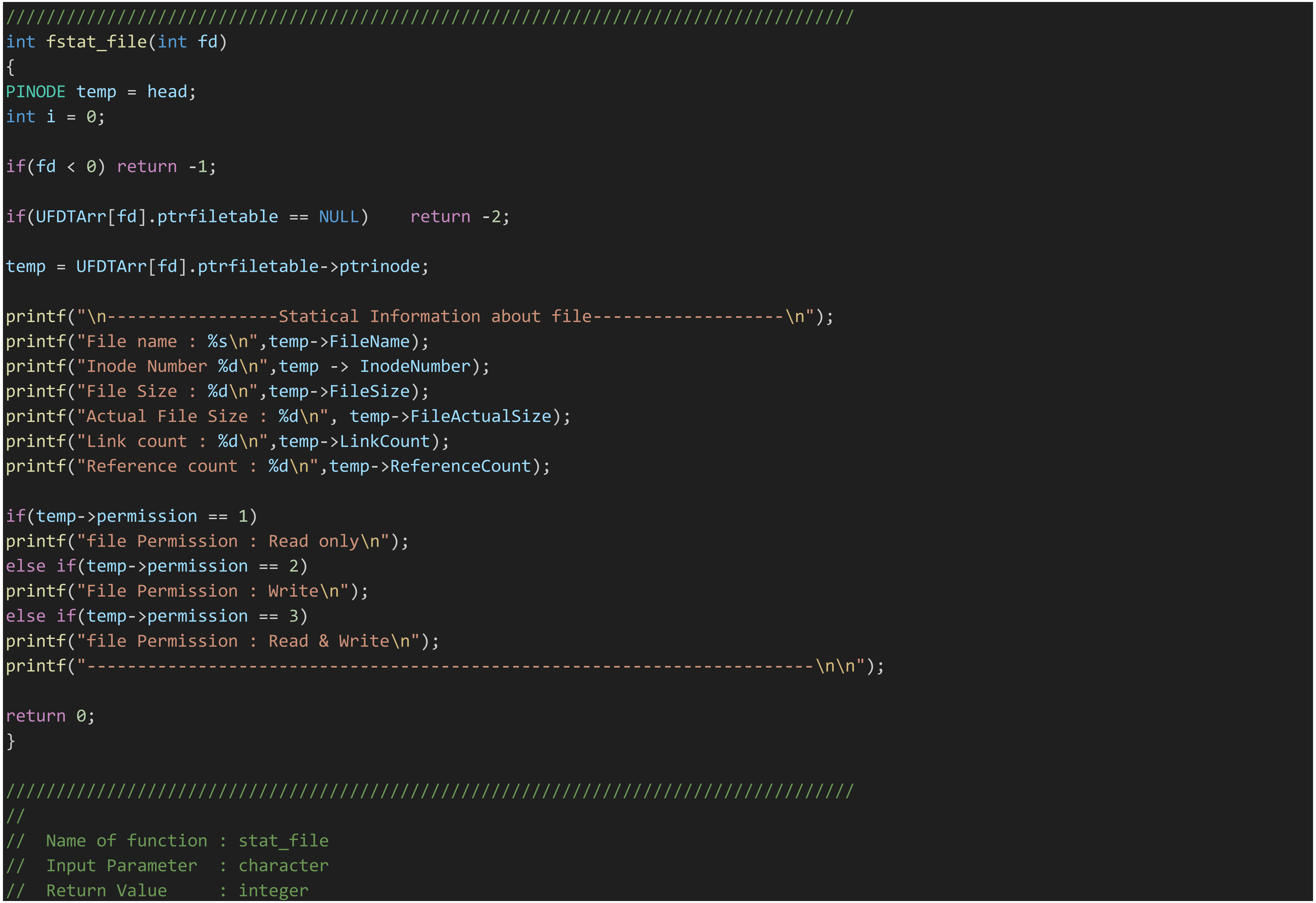
```
UFDTArr[i].ptrfiletable->writeoffset = 0;
(UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount)--;
break;
}
}
i++;
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : LseekFile
//  Input Parameter  : integer
//  Return Value     : integer
//  Usage            : Changes the file offset
//
////////////////////////////////////////////////////////////////////////////////////
int LseekFile(int fd, int size, int from)
{
if((fd<0) || (from > 2))    return -1;
if(UFDTArr[fd].ptrfiletable == NULL)    return -1;

if((UFDTArr[fd].ptrfiletable->mode== READ) || (UFDTArr[fd].ptrfiletable->mode == READ + WRITE))
{
if(from == CURRENT)
{
if(((UFDTArr[fd].ptrfiletable->readoffset) + size) > UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize)    return -1;
if(((UFDTArr[fd].ptrfiletable->readoffset) + size) < 0)     return -1;
(UFDTArr[fd].ptrfiletable->readoffset) = (UFDTArr[fd].ptrfiletable->readoffset)+size;
}
else if(from == START)
{
if(size > (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize)) return -1;
if(size < 0)    return -1;
(UFDTArr[fd].ptrfiletable->readoffset) = size;
}
else if(from == END)
```



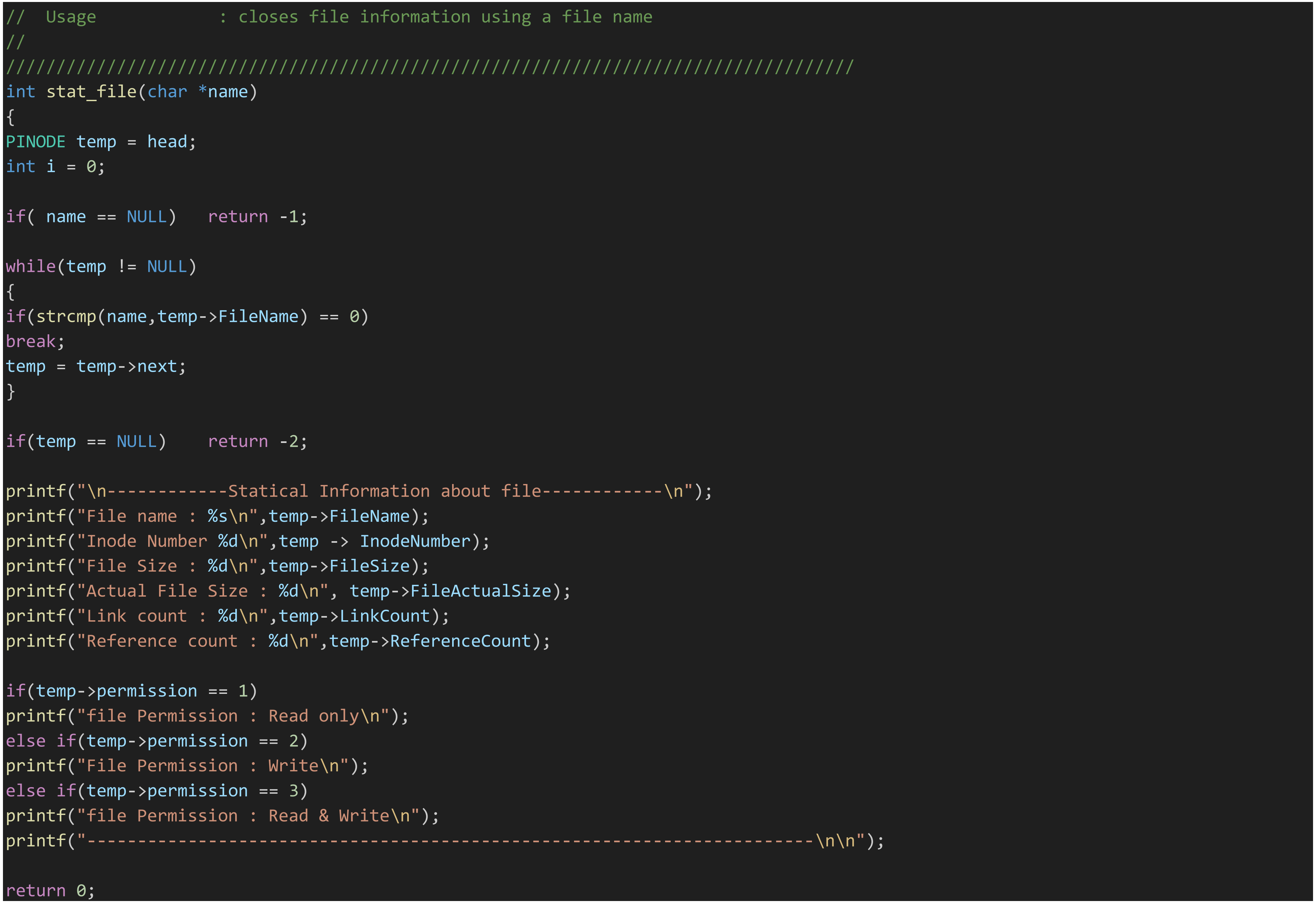
```
{
if((UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + size >MAXFILESIZE) return -1;
if(((UFDTArr[fd].ptrfiletable->readoffset) + size) < 0)   return -1;
(UFDTArr[fd].ptrfiletable->readoffset) = (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + size;
}
}
else if(UFDTArr[fd].ptrfiletable->mode == WRITE)
{
if(from == CURRENT)
{
if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) > MAXFILESIZE)  return -1;
if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) < 0)    return -1;
if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) > (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize))
(UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) = (UFDTArr[fd].ptrfiletable->writeoffset) + size;
(UFDTArr[fd].ptrfiletable->writeoffset) = (UFDTArr[fd].ptrfiletable->writeoffset) + size;
}
else if(from == START)
{
if(size > MAXFILESIZE)  return -1;
if(size < 0) return -1;
if(size >(UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize))
(UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) = size;
(UFDTArr[fd].ptrfiletable->writeoffset) = size;
}
else if(from == END)
{
if((UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + size > MAXFILESIZE)   return -1;
if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) < 0)     return -1;
(UFDTArr[fd].ptrfiletable -> writeoffset) = (UFDTArr[fd]. ptrfiletable->ptrinode->FileActualSize) + size;
}
}
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : ls_File
```



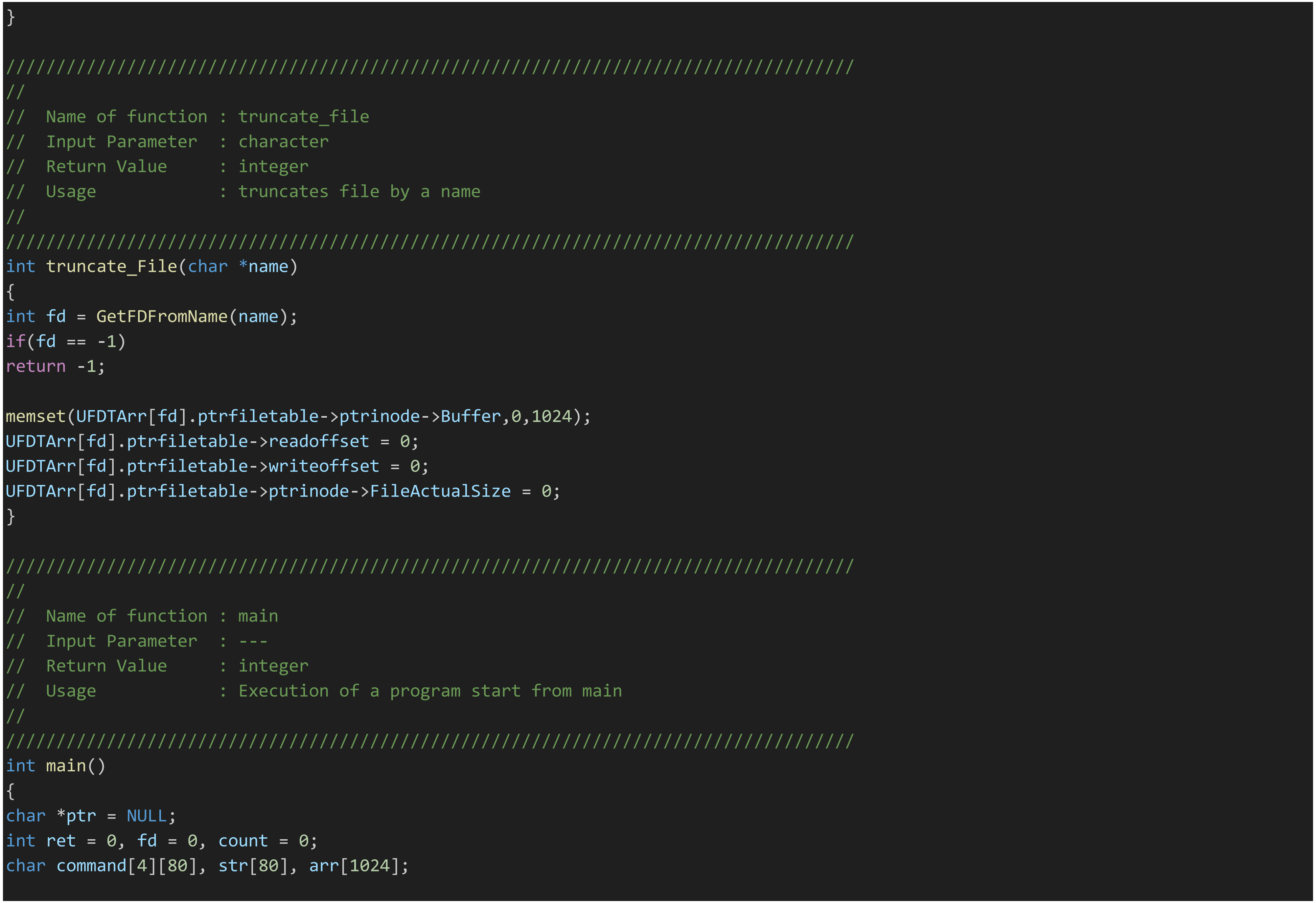
```
//  Input Parameter  : ----
//  Return Value     : void
//  Usage            : Lists all files wth detains
//
////////////////////////////////////////////////////////////////////////////////////
void ls_file()
{
int i = 0;
PINODE temp = head;

if(SUPERBLOCKobj.FreeInode == MAXINODE)
{
printf("Error : There are no files\n");
return;
}

printf("\n File name\tInode number\tFile size\tLink count\n");
printf("----------------------------------------------------------\n");
while(temp != NULL)
{
if(temp -> FileType != 0)
{
printf("%s\t\t%d\t\t%d\t\t%d\n",temp->FileName,temp->InodeNumber,temp->FileActualSize,temp->LinkCount);
}
temp = temp->next;
}
printf("-------------------------------------------------------------\n");
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : fstat_file
//  Input Parameter  : integer
//  Return Value     : integer
//  Usage            : Display file information using a file descriptor
//
```



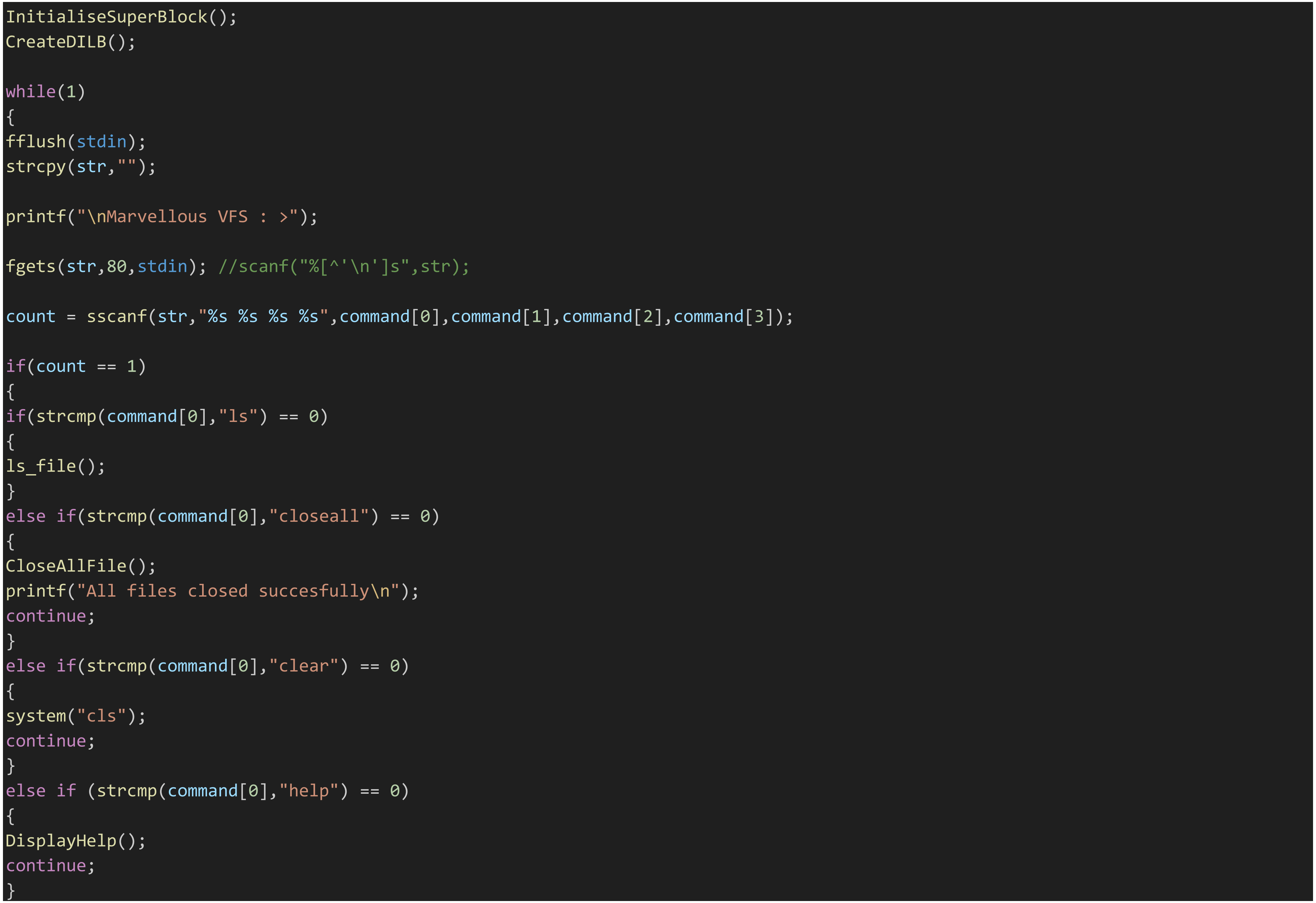
```c
////////////////////////////////////////////////////////////////////////////////////
int fstat_file(int fd)
{
PINODE temp = head;
int i = 0;

if(fd < 0) return -1;

if(UFDTArr[fd].ptrfiletable == NULL)    return -2;

temp = UFDTArr[fd].ptrfiletable->ptrinode;

printf("\n-----------------Statical Information about file-------------------\n");
printf("File name : %s\n",temp->FileName);
printf("Inode Number %d\n",temp -> InodeNumber);
printf("File Size : %d\n",temp->FileSize);
printf("Actual File Size : %d\n", temp->FileActualSize);
printf("Link count : %d\n",temp->LinkCount);
printf("Reference count : %d\n",temp->ReferenceCount);

if(temp->permission == 1)
printf("file Permission : Read only\n");
else if(temp->permission == 2)
printf("File Permission : Write\n");
else if(temp->permission == 3)
printf("file Permission : Read & Write\n");
printf("------------------------------------------------------------------------\n\n");

return 0;
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : stat_file
//  Input Parameter  : character
//  Return Value     : integer
```



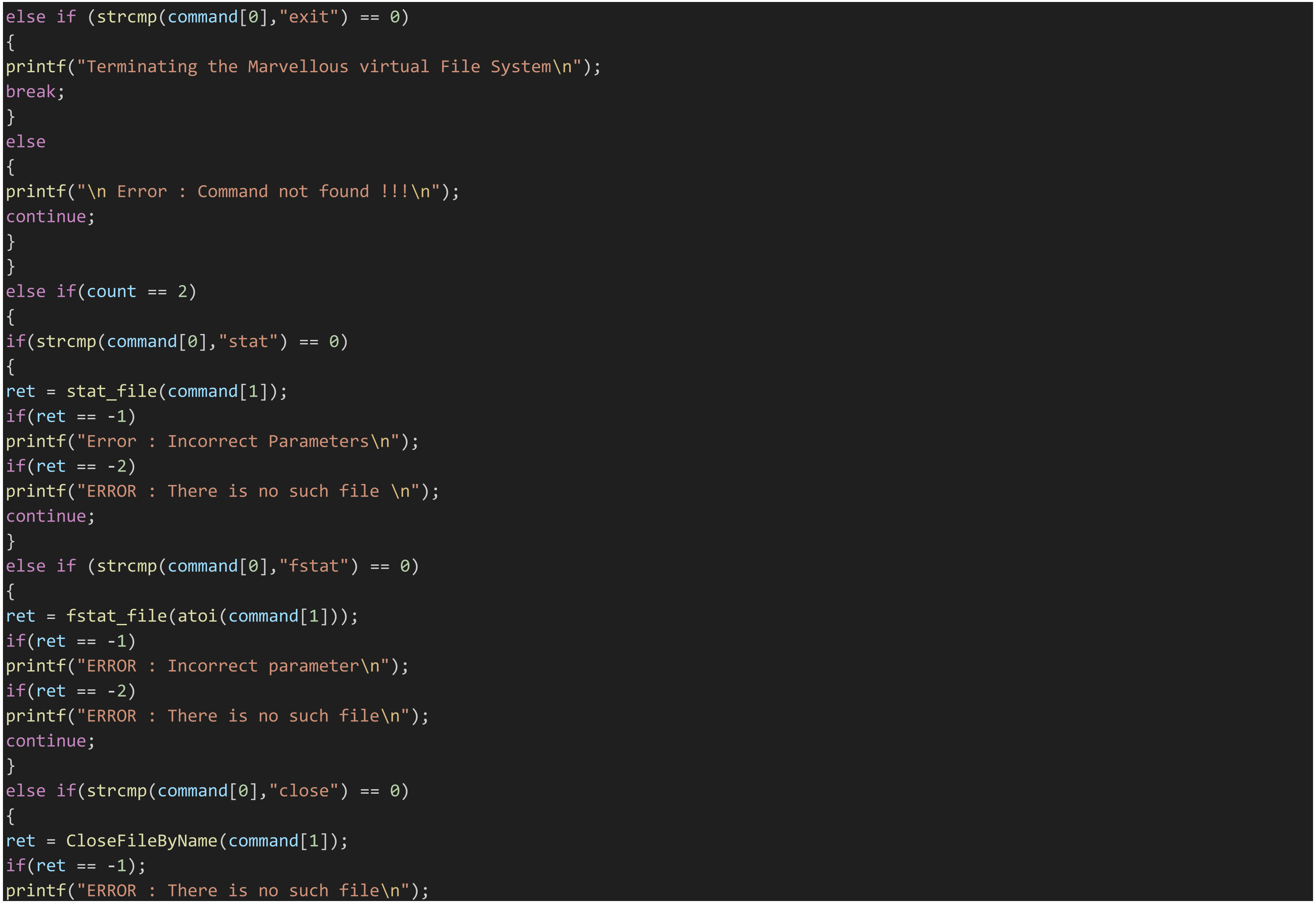
```
//  Usage            : closes file information using a file name
//
////////////////////////////////////////////////////////////////////////////////////
int stat_file(char *name)
{
PINODE temp = head;
int i = 0;

if( name == NULL)   return -1;

while(temp != NULL)
{
if(strcmp(name,temp->FileName) == 0)
break;
temp = temp->next;
}

if(temp == NULL)    return -2;

printf("\n------------Statical Information about file------------\n");
printf("File name : %s\n",temp->FileName);
printf("Inode Number %d\n",temp -> InodeNumber);
printf("File Size : %d\n",temp->FileSize);
printf("Actual File Size : %d\n", temp->FileActualSize);
printf("Link count : %d\n",temp->LinkCount);
printf("Reference count : %d\n",temp->ReferenceCount);

if(temp->permission == 1)
printf("file Permission : Read only\n");
else if(temp->permission == 2)
printf("File Permission : Write\n");
else if(temp->permission == 3)
printf("file Permission : Read & Write\n");
printf("------------------------------------------------------------------------\n\n");

return 0;
```



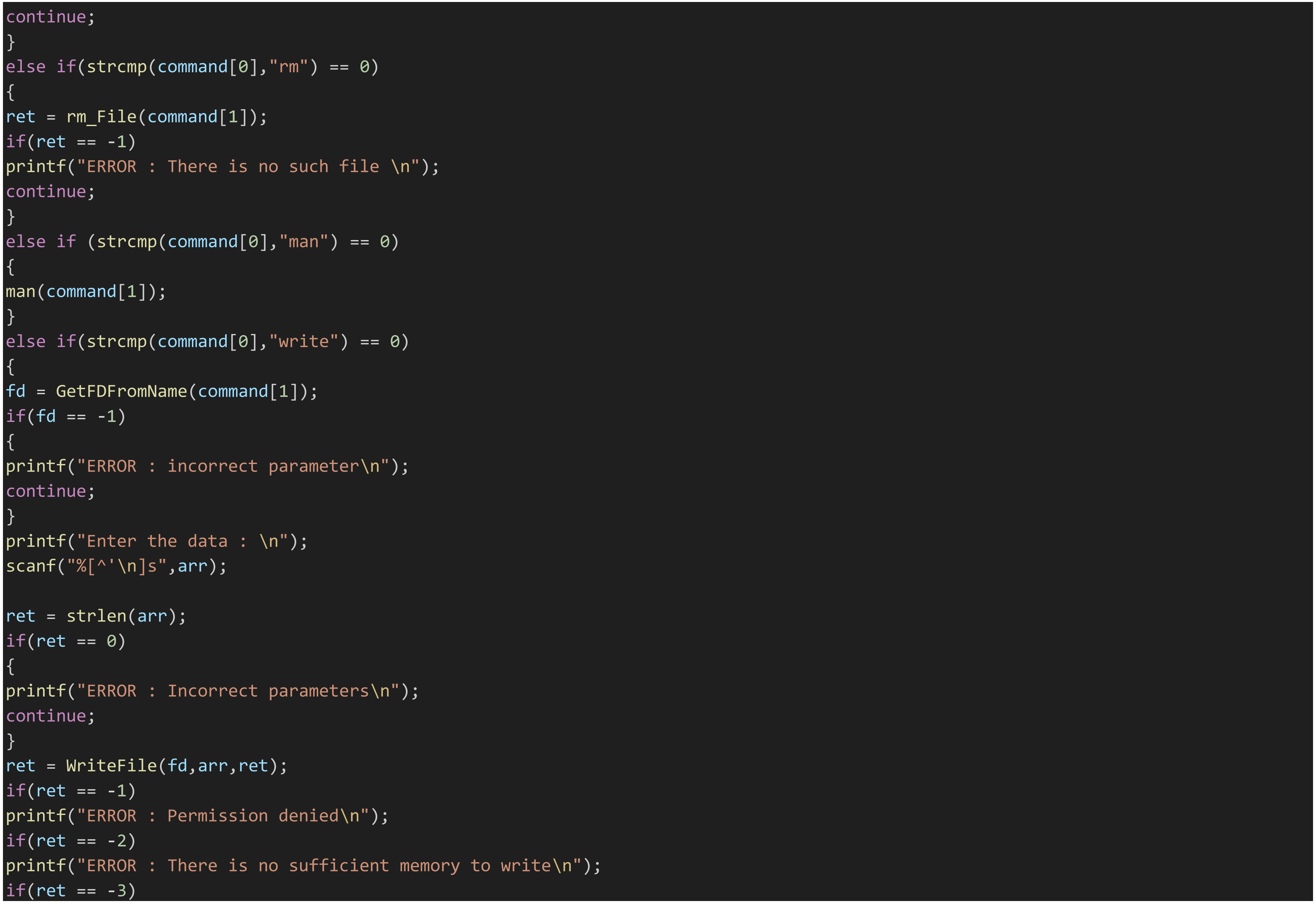
```
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : truncate_file
//  Input Parameter  : character
//  Return Value     : integer
//  Usage            : truncates file by a name
//
////////////////////////////////////////////////////////////////////////////////////
int truncate_File(char *name)
{
int fd = GetFDFromName(name);
if(fd == -1)
return -1;

memset(UFDTArr[fd].ptrfiletable->ptrinode->Buffer,0,1024);
UFDTArr[fd].ptrfiletable->readoffset = 0;
UFDTArr[fd].ptrfiletable->writeoffset = 0;
UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize = 0;
}

////////////////////////////////////////////////////////////////////////////////////
//
//  Name of function : main
//  Input Parameter  : ---
//  Return Value     : integer
//  Usage            : Execution of a program start from main
//
////////////////////////////////////////////////////////////////////////////////////
int main()
{
char *ptr = NULL;
int ret = 0, fd = 0, count = 0;
char command[4][80], str[80], arr[1024];
```



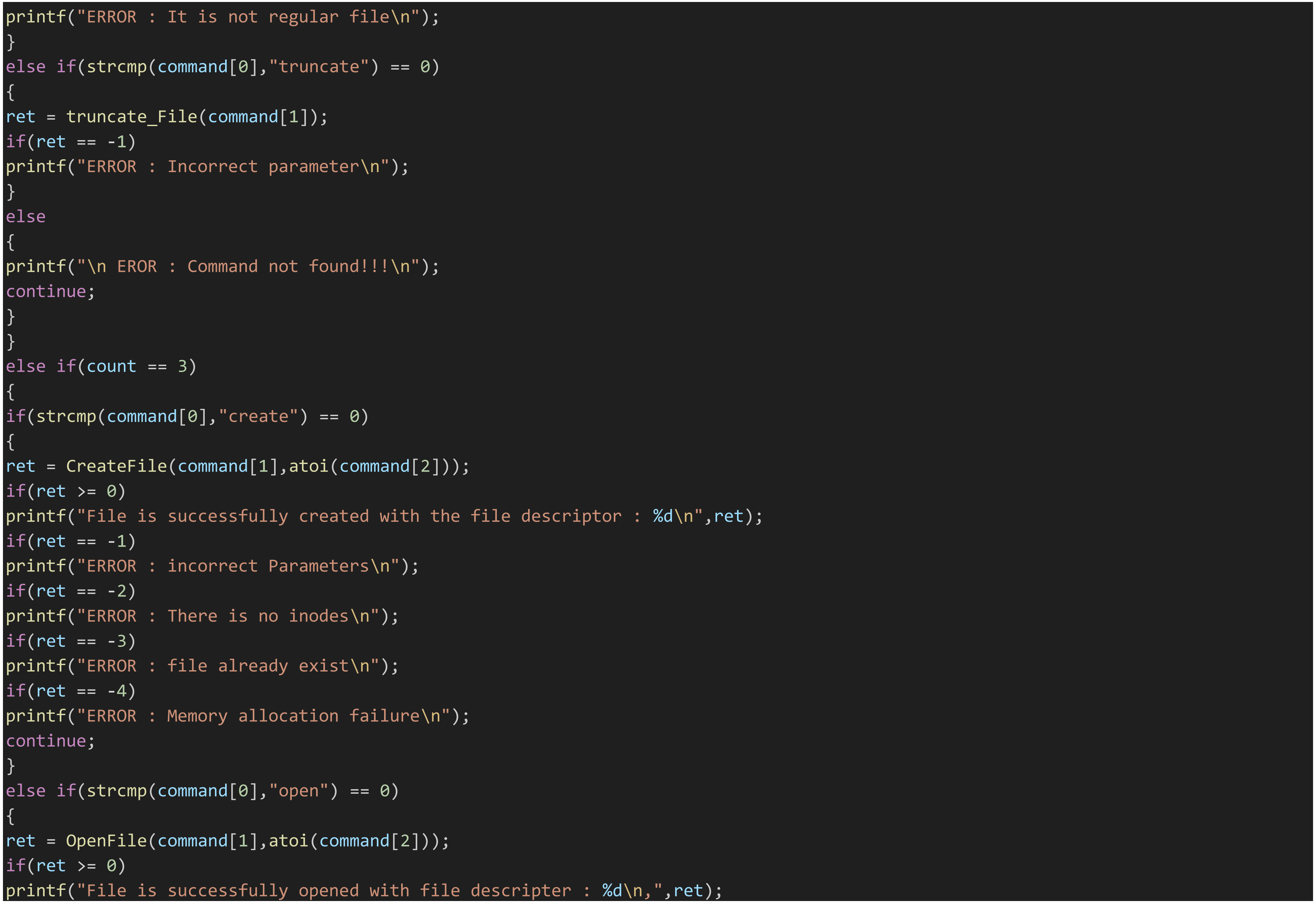
```
InitialiseSuperBlock();
CreateDILB();

while(1)
{
fflush(stdin);
strcpy(str,"");

printf("\nMarvellous VFS : >");

fgets(str,80,stdin); //scanf("%[^'\n']s",str);

count = sscanf(str,"%s %s %s %s",command[0],command[1],command[2],command[3]);

if(count == 1)
{
if(strcmp(command[0],"ls") == 0)
{
ls_file();
}
else if(strcmp(command[0],"closeall") == 0)
{
CloseAllFile();
printf("All files closed succesfully\n");
continue;
}
else if(strcmp(command[0],"clear") == 0)
{
system("cls");
continue;
}
else if (strcmp(command[0],"help") == 0)
{
DisplayHelp();
continue;
}
```



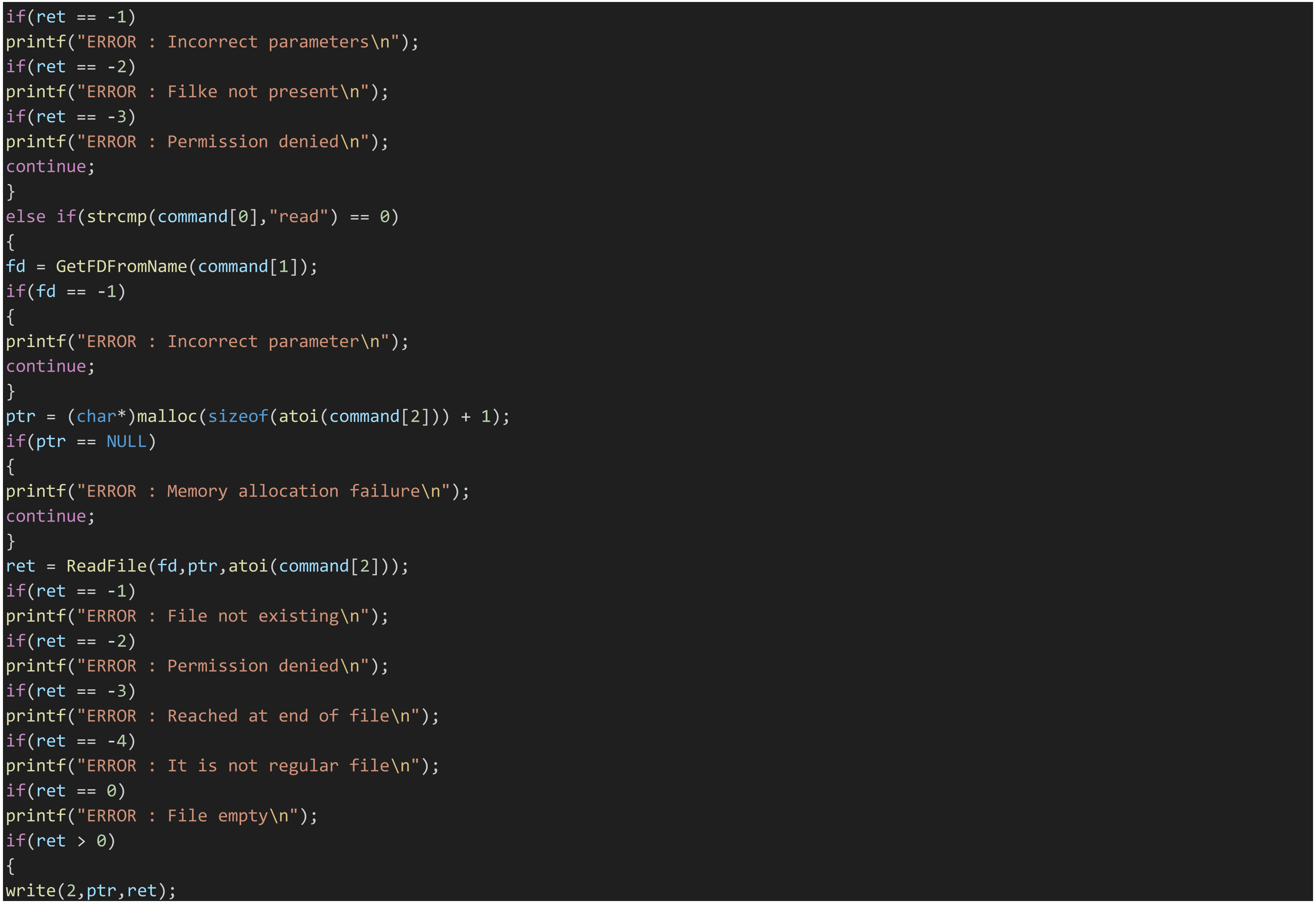
```c
else if (strcmp(command[0],"exit") == 0)
{
printf("Terminating the Marvellous virtual File System\n");
break;
}
else
{
printf("\n Error : Command not found !!!\n");
continue;
}
}
else if(count == 2)
{
if(strcmp(command[0],"stat") == 0)
{
ret = stat_file(command[1]);
if(ret == -1)
printf("Error : Incorrect Parameters\n");
if(ret == -2)
printf("ERROR : There is no such file \n");
continue;
}
else if (strcmp(command[0],"fstat") == 0)
{
ret = fstat_file(atoi(command[1]));
if(ret == -1)
printf("ERROR : Incorrect parameter\n");
if(ret == -2)
printf("ERROR : There is no such file\n");
continue;
}
else if(strcmp(command[0],"close") == 0)
{
ret = CloseFileByName(command[1]);
if(ret == -1);
printf("ERROR : There is no such file\n");
```



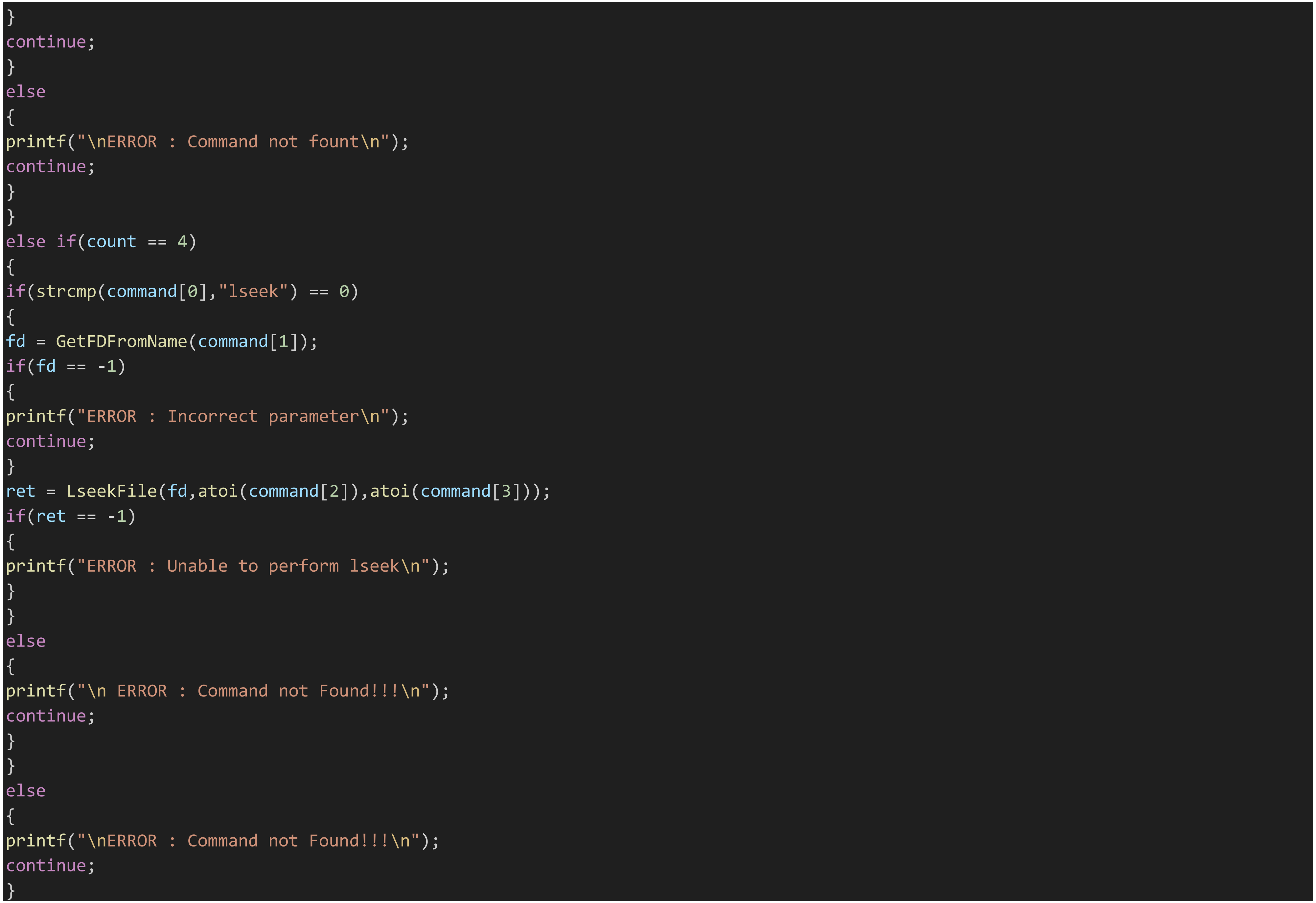
```
continue;
}
else if(strcmp(command[0],"rm") == 0)
{
ret = rm_File(command[1]);
if(ret == -1)
printf("ERROR : There is no such file \n");
continue;
}
else if (strcmp(command[0],"man") == 0)
{
man(command[1]);
}
else if(strcmp(command[0],"write") == 0)
{
fd = GetFDFromName(command[1]);
if(fd == -1)
{
printf("ERROR : incorrect parameter\n");
continue;
}
printf("Enter the data : \n");
scanf("%[^'\n]s",arr);

ret = strlen(arr);
if(ret == 0)
{
printf("ERROR : Incorrect parameters\n");
continue;
}
ret = WriteFile(fd,arr,ret);
if(ret == -1)
printf("ERROR : Permission denied\n");
if(ret == -2)
printf("ERROR : There is no sufficient memory to write\n");
if(ret == -3)
```



```
printf("ERROR : It is not regular file\n");
}
else if(strcmp(command[0],"truncate") == 0)
{
ret = truncate_File(command[1]);
if(ret == -1)
printf("ERROR : Incorrect parameter\n");
}
else
{
printf("\n EROR : Command not found!!!\n");
continue;
}
}
else if(count == 3)
{
if(strcmp(command[0],"create") == 0)
{
ret = CreateFile(command[1],atoi(command[2]));
if(ret >= 0)
printf("File is successfully created with the file descriptor : %d\n",ret);
if(ret == -1)
printf("ERROR : incorrect Parameters\n");
if(ret == -2)
printf("ERROR : There is no inodes\n");
if(ret == -3)
printf("ERROR : file already exist\n");
if(ret == -4)
printf("ERROR : Memory allocation failure\n");
continue;
}
else if(strcmp(command[0],"open") == 0)
{
ret = OpenFile(command[1],atoi(command[2]));
if(ret >= 0)
printf("File is successfully opened with file descripter : %d\n,",ret);
```



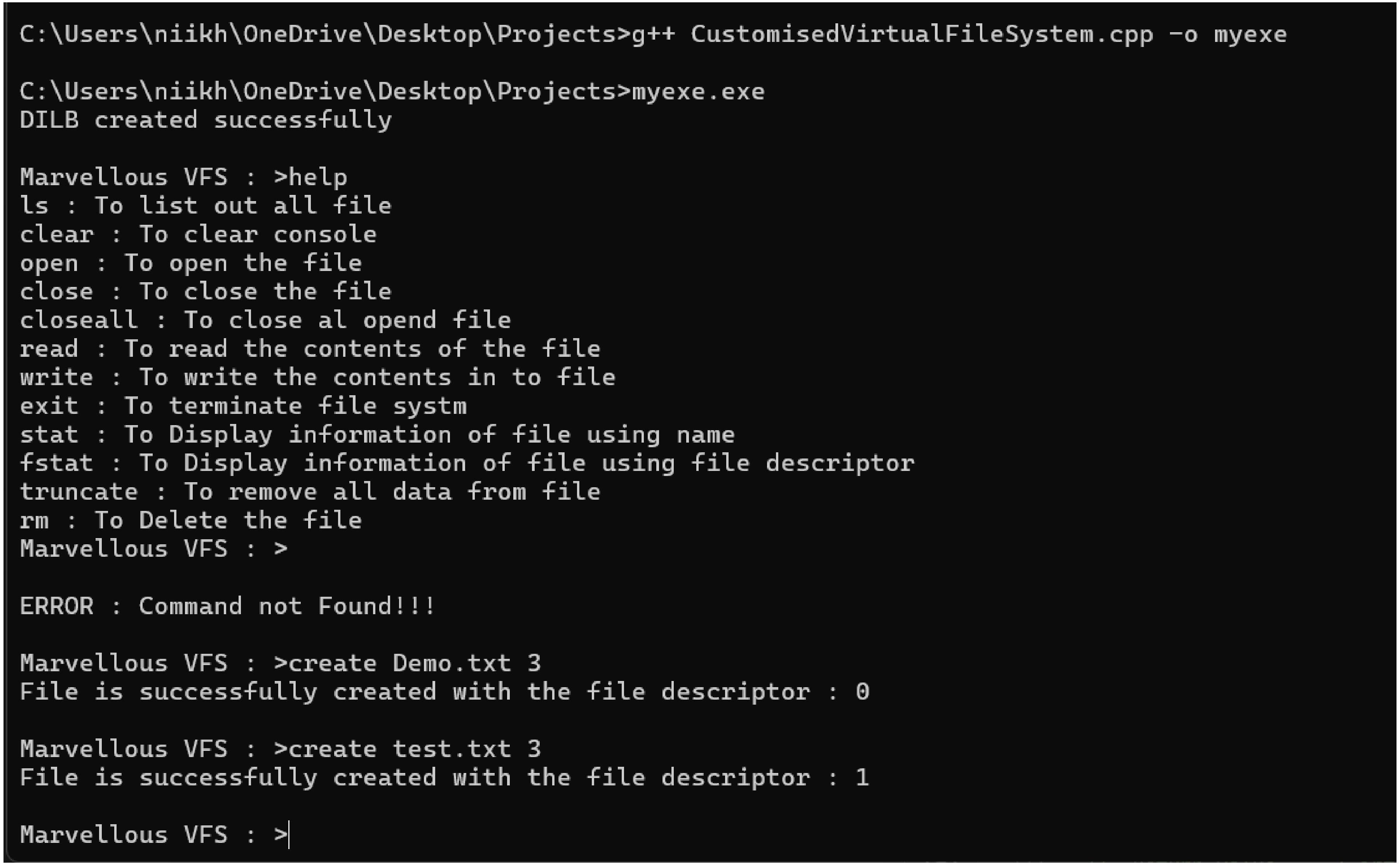
```
if(ret == -1)
printf("ERROR : Incorrect parameters\n");
if(ret == -2)
printf("ERROR : Filke not present\n");
if(ret == -3)
printf("ERROR : Permission denied\n");
continue;
}
else if(strcmp(command[0],"read") == 0)
{
fd = GetFDFromName(command[1]);
if(fd == -1)
{
printf("ERROR : Incorrect parameter\n");
continue;
}
ptr = (char*)malloc(sizeof(atoi(command[2])) + 1);
if(ptr == NULL)
{
printf("ERROR : Memory allocation failure\n");
continue;
}
ret = ReadFile(fd,ptr,atoi(command[2]));
if(ret == -1)
printf("ERROR : File not existing\n");
if(ret == -2)
printf("ERROR : Permission denied\n");
if(ret == -3)
printf("ERROR : Reached at end of file\n");
if(ret == -4)
printf("ERROR : It is not regular file\n");
if(ret == 0)
printf("ERROR : File empty\n");
if(ret > 0)
{
write(2,ptr,ret);
```



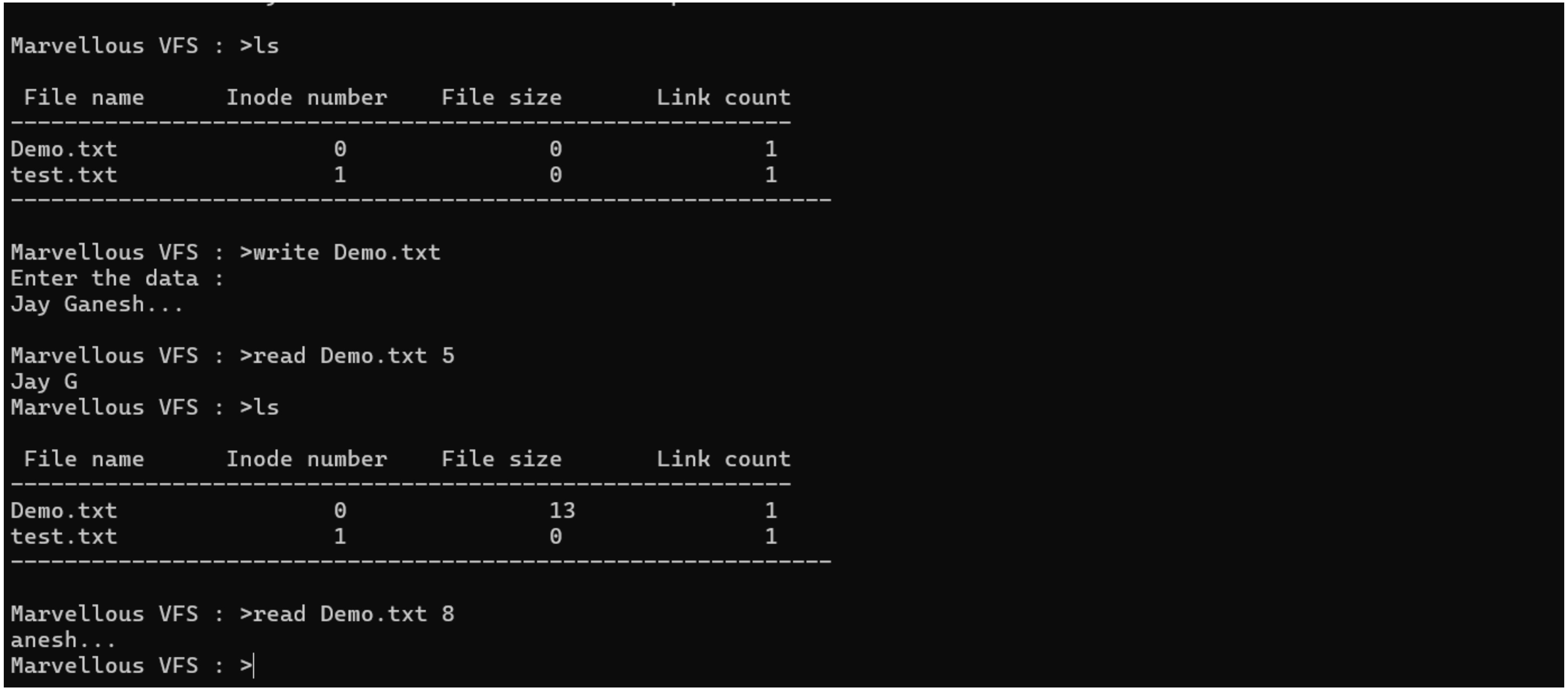
```
}
continue;
}
else
{
printf("\nERROR : Command not fount\n");
continue;
}
}
else if(count == 4)
{
if(strcmp(command[0],"lseek") == 0)
{
fd = GetFDFromName(command[1]);
if(fd == -1)
{
printf("ERROR : Incorrect parameter\n");
continue;
}
ret = LseekFile(fd,atoi(command[2]),atoi(command[3]));
if(ret == -1)
{
printf("ERROR : Unable to perform lseek\n");
}
}
else
{
printf("\n ERROR : Command not Found!!!\n");
continue;
}
}
else
{
printf("\nERROR : Command not Found!!!\n");
continue;
}
```

```
}
return 0;
}
```

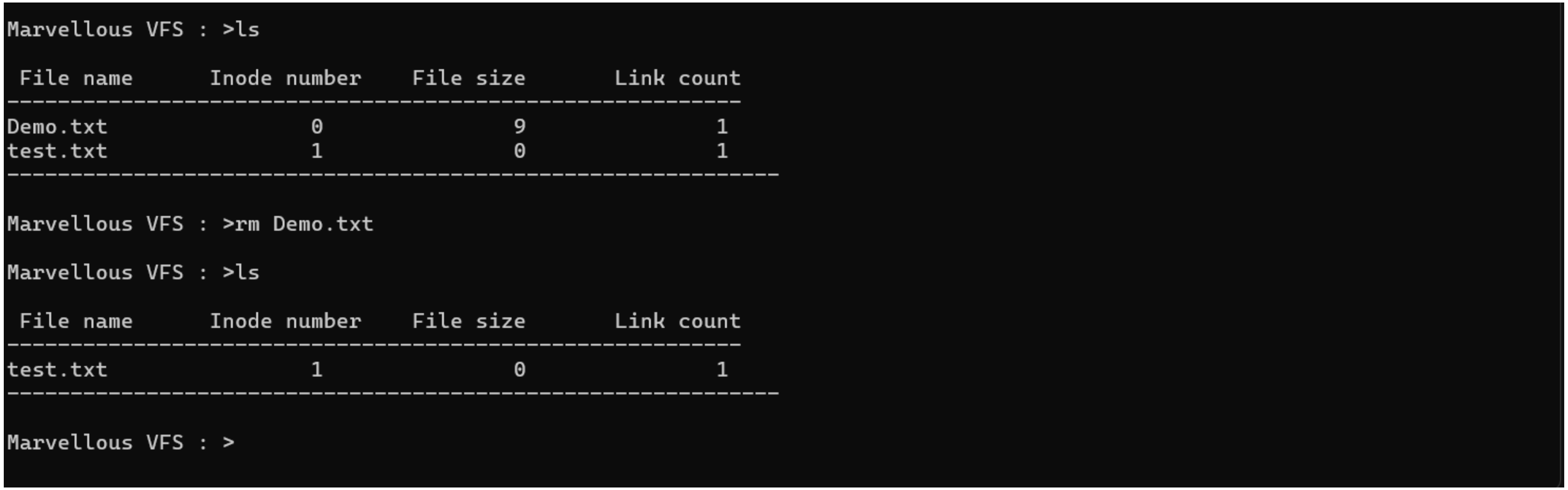
Output :



```
C:\Users\niikh\OneDrive\Desktop\Projects>g++ CustomisedVirtualFileSystem.cpp -o myexe

C:\Users\niikh\OneDrive\Desktop\Projects>myexe.exe
DILB created successfully

Marvellous VFS : >help
ls : To list out all file
clear : To clear console
open : To open the file
close : To close the file
closeall : To close al opend file
read : To read the contents of the file
write : To write the contents in to file
exit : To terminate file systm
stat : To Display information of file using name
fstat : To Display information of file using file descriptor
truncate : To remove all data from file
rm : To Delete the file
Marvellous VFS : >

ERROR : Command not Found!!!

Marvellous VFS : >create Demo.txt 3
File is successfully created with the file descriptor : 0

Marvellous VFS : >create test.txt 3
File is successfully created with the file descriptor : 1

Marvellous VFS : >
```



```
Marvellous VFS : >ls

 File name      Inode number    File size       Link count
----------------------------------------------------------
Demo.txt                0               0               1
test.txt                1               0               1
-------------------------------------------------------------

Marvellous VFS : >write Demo.txt
Enter the data :
Jay Ganesh...

Marvellous VFS : >read Demo.txt 5
Jay G
Marvellous VFS : >ls

 File name      Inode number    File size       Link count
----------------------------------------------------------
Demo.txt                0               13              1
test.txt                1               0               1
-------------------------------------------------------------

Marvellous VFS : >read Demo.txt 8
anesh...
Marvellous VFS : >
```



```
Marvellous VFS : >ls

 File name      Inode number    File size       Link count
----------------------------------------------------------
Demo.txt                0               9               1
test.txt                1               0               1
-------------------------------------------------------------

Marvellous VFS : >rm Demo.txt

Marvellous VFS : >ls

 File name      Inode number    File size       Link count
----------------------------------------------------------
test.txt                1               0               1
-------------------------------------------------------------

Marvellous VFS : >
```

- System call Working :
  - open() : The open system call will use to open the existing file. For system call we have to pass to two parameter as name of file and the mode in which we want to open the file. For open system call we have to pass two parameter as name of file and the mode in which we want to open the file. The mode which open the file is considered as a macro. For the mod there are three important option. Checks the file existence and permission, allocates a file descriptor, and initializes internal structure.
    - O_RDONLY = Read mode
    - O_WRONLY = Write node
    - O_RDWR = Read + Write

      Ex. fd = open(File_Name, O_RDWR)

  - close() : It is good programming practice to close an opened file after its use gets completed. For the close system call we have to pass only one parameter as a file descriptor and there is no return value of that system call. Validates the file descriptor, updates internal tables, and release resources.

    Ex. close(fd)

- write() : Write system call is a call is used to write the data into regular file. For the write system call we have to pass three parameters File descriptor, the data we want to write, number of bytes we want to write. After the successful execution of a write system call it returns number of bytes successfully. Copied data from user space to kernel space and updates the file offset.

  EX. write(fd, data, 22);

- read() : The read system call is used to read the data from the file . for the read system call we have to pass first parameter as a file descriptor .second parameter is a number of empty array . And the third parameter is number of bytes we want read. The read system call returns the number of bytes successfully read from the file. Copies data from kernel space to user space and updates the file offset.

  Ex. Read(fd, data, 10);

- lseek() : This system call is used to access the file in random way. In lssek function we change the file offset to the specific function.

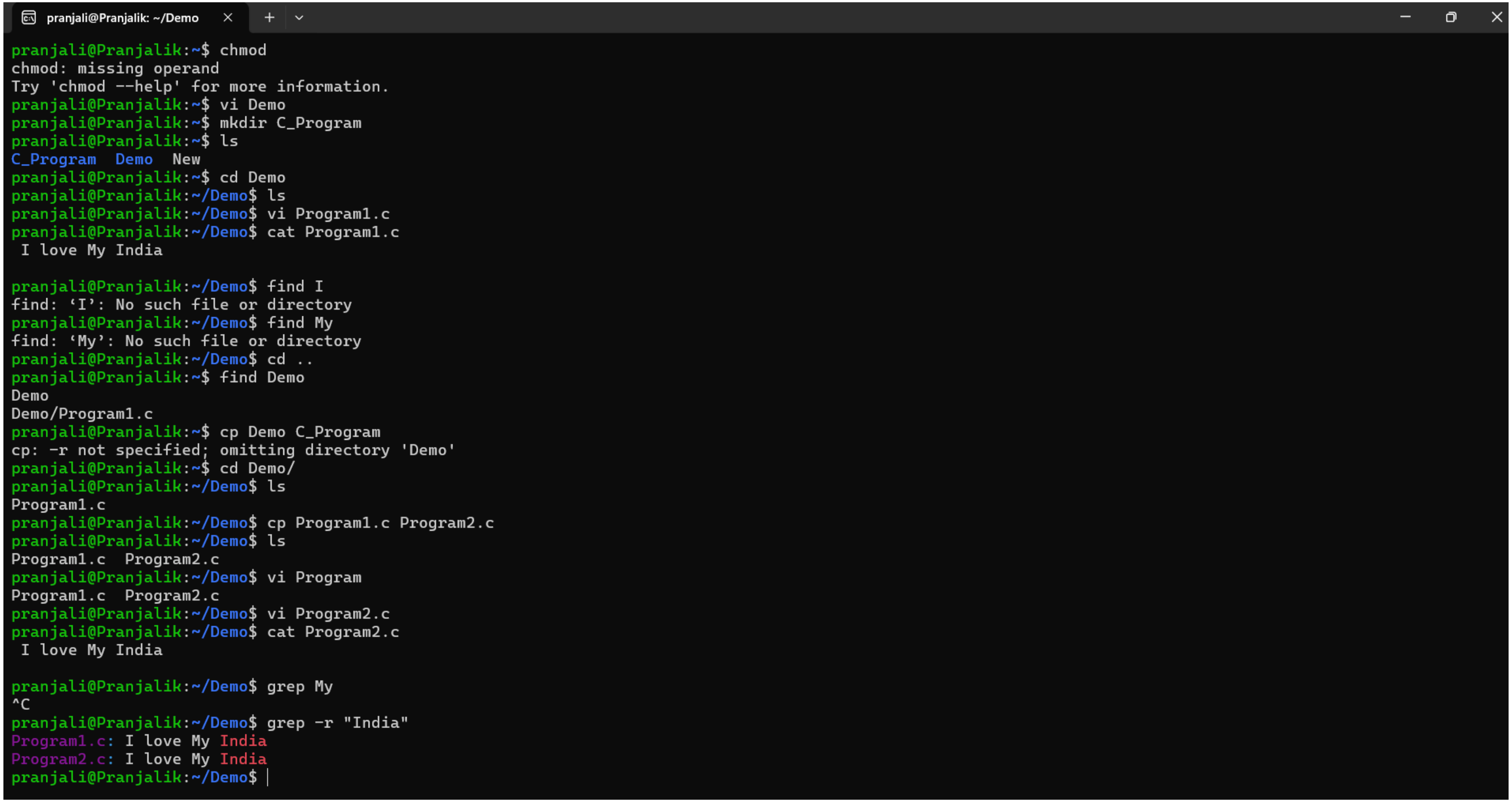
  Ex. lseek(fd, 10, SEEK_SET);

- stat() : Retrieves file status information. Fills a structure with file metadata from the file system.
-
- unlink() : Removes a file. Decrement the link count and, if it reaches 0, deallocates the inode and file data.
-
- chmod() : Changes the permission of the file. Updates the mode bites of the files inode in the file system.

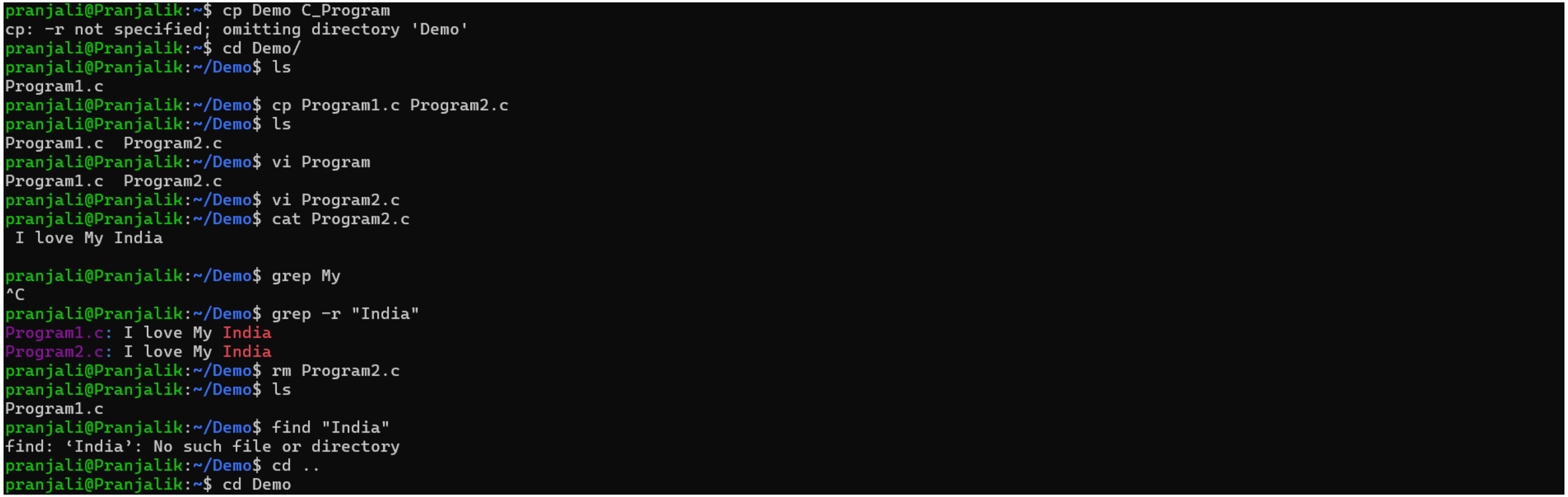
Use of Commands :

- ls: List out all the existing files
- man : Used to display the manual page for the commands.
- ls – l : Is used to list the content of a directory in long format.
- ls – a : Is used to list all files and directories int the specific directory, including hidden file.
- cat : It is used to concatenate and display the content of files.
- cd : it Is used to change the current working directory in the command line interface.
- chmod : Is used to change the file system modes of files and directories.
- cp : is used to copy files and directories.
- df : Is used to display information about the disk space usage of the file system.
- find : Is used to searching for files and directories within a directory hierarchy.
- grep : Is used to search for pattern within the file.
- ln : Used to create link between files.
- mkdir : Is used to create a new directories.

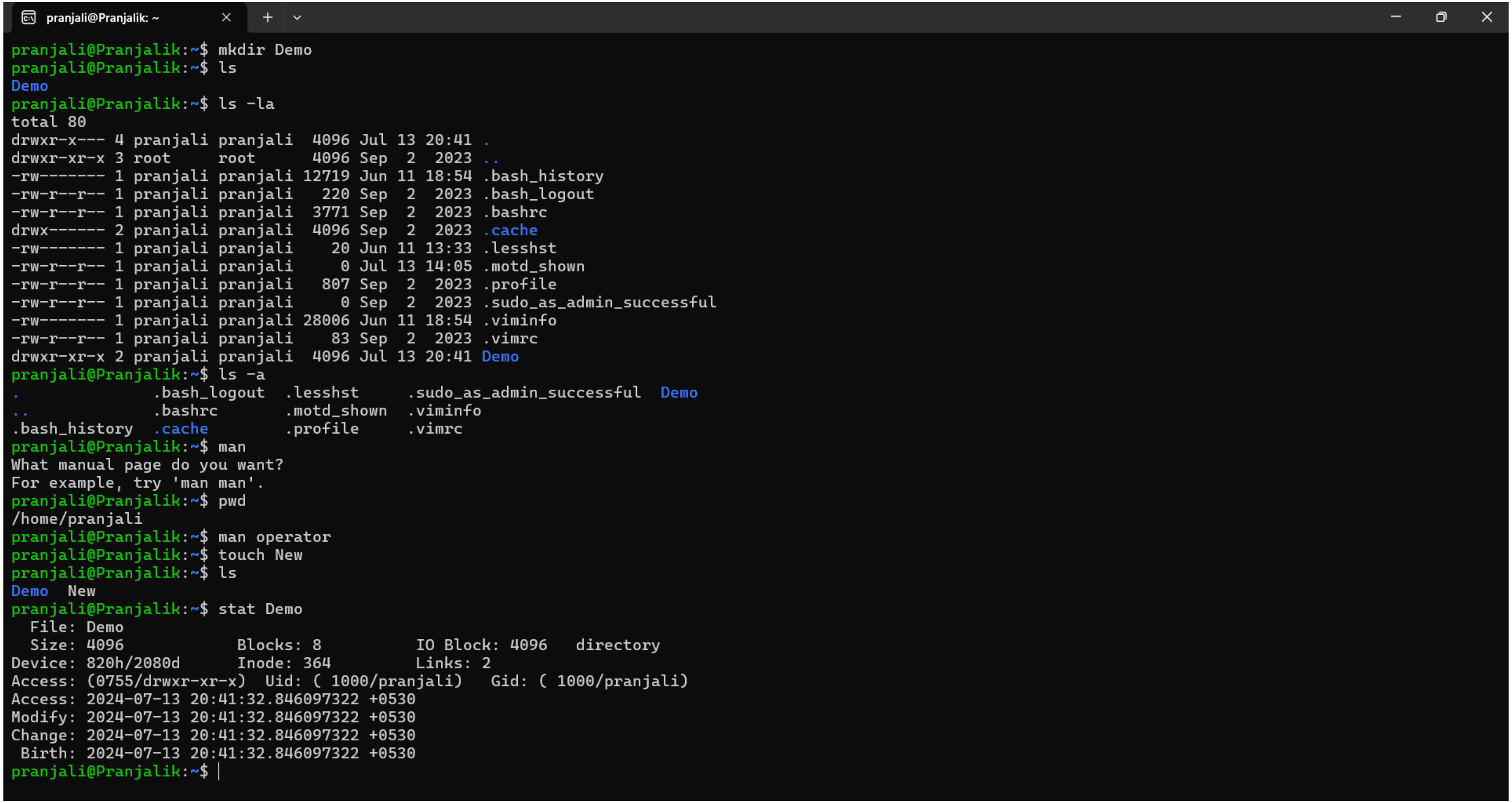
- pwd : It is used to display the current directory you are in within the command line interface.
- touch : Is used to create a empty file or update the timestamp of existing file.
- uname : Is used to display system information.
- stat : Is used to display all the information from the inode of the file.
- mkfs (make file system) : Is used to create file system on a storage device of partition.



```
pranjali@Pranjalik:~$ chmod
chmod: missing operand
Try 'chmod --help' for more information.
pranjali@Pranjalik:~$ vi Demo
pranjali@Pranjalik:~$ mkdir C_Program
pranjali@Pranjalik:~$ ls
C_Program  Demo  New
pranjali@Pranjalik:~$ cd Demo
pranjali@Pranjalik:~/Demo$ ls
pranjali@Pranjalik:~/Demo$ vi Program1.c
pranjali@Pranjalik:~/Demo$ cat Program1.c
 I love My India

pranjali@Pranjalik:~/Demo$ find I
find: ‘I’: No such file or directory
pranjali@Pranjalik:~/Demo$ find My
find: ‘My’: No such file or directory
pranjali@Pranjalik:~/Demo$ cd ..
pranjali@Pranjalik:~$ find Demo
Demo
Demo/Program1.c
pranjali@Pranjalik:~$ cp Demo C_Program
cp: -r not specified; omitting directory 'Demo'
pranjali@Pranjalik:~$ cd Demo/
pranjali@Pranjalik:~/Demo$ ls
Program1.c
pranjali@Pranjalik:~/Demo$ cp Program1.c Program2.c
pranjali@Pranjalik:~/Demo$ ls
Program1.c  Program2.c
pranjali@Pranjalik:~/Demo$ vi Program
Program1.c  Program2.c
pranjali@Pranjalik:~/Demo$ vi Program2.c
pranjali@Pranjalik:~/Demo$ cat Program2.c
 I love My India

pranjali@Pranjalik:~/Demo$ grep My
^C
pranjali@Pranjalik:~/Demo$ grep -r "India"
Program1.c: I love My India
Program2.c: I love My India
pranjali@Pranjalik:~/Demo$ 
```



```
pranjali@Pranjalik:~$ cp Demo C_Program
cp: -r not specified; omitting directory 'Demo'
pranjali@Pranjalik:~$ cd Demo/
pranjali@Pranjalik:~/Demo$ ls
Program1.c
pranjali@Pranjalik:~/Demo$ cp Program1.c Program2.c
pranjali@Pranjalik:~/Demo$ ls
Program1.c  Program2.c
pranjali@Pranjalik:~/Demo$ vi Program
Program1.c  Program2.c
pranjali@Pranjalik:~/Demo$ vi Program2.c
pranjali@Pranjalik:~/Demo$ cat Program2.c
 I love My India

pranjali@Pranjalik:~/Demo$ grep My
^C
pranjali@Pranjalik:~/Demo$ grep -r "India"
Program1.c: I love My India
Program2.c: I love My India
pranjali@Pranjalik:~/Demo$ rm Program2.c
pranjali@Pranjalik:~/Demo$ ls
Program1.c
pranjali@Pranjalik:~/Demo$ find "India"
find: ‘India’: No such file or directory
pranjali@Pranjalik:~/Demo$ cd ..
pranjali@Pranjalik:~$ cd Demo
```



```
pranjali@Pranjalik:~$ mkdir Demo
pranjali@Pranjalik:~$ ls
Demo
pranjali@Pranjalik:~$ ls -la
total 80
drwxr-x--- 4 pranjali pranjali  4096 Jul 13 20:41 .
drwxr-xr-x 3 root     root      4096 Sep  2  2023 ..
-rw------- 1 pranjali pranjali 12719 Jun 11 18:54 .bash_history
-rw-r--r-- 1 pranjali pranjali   220 Sep  2  2023 .bash_logout
-rw-r--r-- 1 pranjali pranjali  3771 Sep  2  2023 .bashrc
drwx------ 2 pranjali pranjali  4096 Sep  2  2023 .cache
-rw------- 1 pranjali pranjali    20 Jun 11 13:33 .lesshst
-rw-r--r-- 1 pranjali pranjali     0 Jul 13 14:05 .motd_shown
-rw-r--r-- 1 pranjali pranjali   807 Sep  2  2023 .profile
-rw-r--r-- 1 pranjali pranjali     0 Sep  2  2023 .sudo_as_admin_successful
-rw------- 1 pranjali pranjali 28006 Jun 11 18:54 .viminfo
-rw-r--r-- 1 pranjali pranjali    83 Sep  2  2023 .vimrc
drwxr-xr-x 2 pranjali pranjali  4096 Jul 13 20:41 Demo
pranjali@Pranjalik:~$ ls -a
.              .bash_logout  .lesshst     .sudo_as_admin_successful  Demo
..             .bashrc       .motd_shown  .viminfo
.bash_history  .cache        .profile     .vimrc
pranjali@Pranjalik:~$ man
What manual page do you want?
For example, try 'man man'.
pranjali@Pranjalik:~$ pwd
/home/pranjali
pranjali@Pranjalik:~$ man operator
pranjali@Pranjalik:~$ touch New
pranjali@Pranjalik:~$ ls
Demo  New
pranjali@Pranjalik:~$ stat Demo
  File: Demo
  Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: 820h/2080d      Inode: 364         Links: 2
Access: (0755/drwxr-xr-x)  Uid: ( 1000/pranjali)   Gid: ( 1000/pranjali)
Access: 2024-07-13 20:41:32.846097322 +0530
Modify: 2024-07-13 20:41:32.846097322 +0530
Change: 2024-07-13 20:41:32.846097322 +0530
 Birth: 2024-07-13 20:41:32.846097322 +0530
pranjali@Pranjalik:~$ |
```

➢ Question & Answer :

1. What is mean by file system?

Ans -> A file system is a structure used by an operating system to organise and manage files on storage device.

2. Which file system used by Linux and Windows operating system?

Ans-> Windows use NTFS(new technology file system)

Linux use EXT4 (fourth extended file system)

3. What are the parts of the file system?
Ans-> Boot block
DILB(disk inode list block)
Super block
Data block

4.Explain UAREA and its contents?
Ans->It's a specific area allocated by the operating system to the process which contents the information about that process for every process there is a separate UAREA.

5.Explain the use of the file table and its contents?
Ans->File table is a table which contents information about the file that we want wo open
File table contents below entries
Mode : it contains mode we open the file
Offset : It's a point from when we read or write the data
Pointer : Which points to the specific entry from Incore inode table

6.Explain the use of IIT?
Ans-> It's a table which contains the inodes which are loaded into the memory. All the files inodes are present in IIT. The complete information about the inode is stored inside the IIT.

7.What does inode mean?
Ans-> Inode is a structure which contains complete information about file. Every file contains its unique inode.

8.What are the contents of superblock?
Ans->Superblock is block which contains complete information about file

- Total number of block
- Free block
- Total number of inodes
- List of free inodes
- Size of system file

9.What are the type of file?

Ans->NTFS,fat32,fat64,UFS.

10.What are the content of inodes?

Ans->Permission

Count

File size

Pointer

Ownership

Timestamp

11.What is the use of Directory files?

Ans->Directory file play a crucial role is the file system by organizing and managing files and other directories.

12.How does the operating system maintain security of files?

Ans->The operating system uses access control list (ACLS) to determine which user or process have permission to access specific resources or perform specific action.

13.What happens when a user wants to open a file?

Ans->When a user wants to open a file the operating system will open its directory and fetch the inode number. From that inode number gets fetch

from DILB from that inode the operating system gest the block number and after getting the block number the operating system open the block and read the actual data of the file.

14.What happen when user calls the lseek system call?

Ans-> lseek system call is used to change the file offset.

15.What is difference between library function & system calls?

Ans->A system call is a function provide by the kernel to enter into kernel mode to access the hardware resources.

A library call is a function provided by the programming library to perform task.

System calls are provided by the system and are executed by the kernel.

Library calls included the ANSI C standard library.

16.What is the use of this project?

Ans->The CVFS project simulates a basic file system in C. allowing users to perform operations like creating the file, reading, writing, opening, closing , deleting files. It serves educational purpose, helping user to understand file system concept such as inodes, superblock & file descriptor.

17.What difficulties that you faces in this project?

Ans->understanding file system and advance C programming. Providing error handling , create, read , write, delete, open & close operations.

18.Is there any improvement needed in this project?

Ans->Enhanced error handling , improve security features, robust error recovery.