

操作系统课程设计报告

——模拟实现 **UNIX** 的文件系统

学 院 计算机科学与工程

专 业 计算机科学与技术

班 级 1308 班

组长学号 20133647

组长姓名 纪钰婷

组员学号 20134028 20134030 20134017

组员姓名 袁堃皓 张旭 刘皓冰

指导教师 吴宏林

2016 年 7 月

设计分工

组长学号及姓名：20133647 纪钰婷
分工：数据结构设计、多级目录的实现、创建目录、删除目录、查看目录、文件(夹)复制、数据块的分配与回收、用户注册
组员 1 学号及姓名：20134028 袁堃皓
分工：建立文件、查看文件、写入文件、文件重命名、删除文件
组员 2 学号及姓名：20134030 张旭
分工：用户登录、用户退出、系统初始化、系统保存、文件格式化
组员 3 学号及姓名：20134017 刘皓冰
分工：多级用户的实现、文件打开、文件关闭、读文件、主函数设计、系统测试

摘 要

由于计算机中的内存是易失性设备，断电后所存储的信息即会丢失，其容量又十分有限，所以在现代计算机系统中，都必须配置外设，将系统和用户需要用到的大量程序和数据以文件的形式存放在外存中，需要时再随机将它们调入内存，或将它们打印出来。如果由用户直接管理存放在外存上的文件，不仅要求用户熟悉外存特性，了解各种文件属性，以及它们在外存中的位置，而且在多用户环境下还必须能保持数据的安全性和一致性。显然这不是用户可以胜任的，于是操作系统增加了文件管理功能。

文件系统是操作系统用于明确存储设备或分区上的文件的方法和数据结构；即在存储设备上组织文件的方法。操作系统中负责管理和存储文件信息的软件机构称为文件管理系统，简称文件系统。文件系统由三部分组成：文件系统的接口，对对象操纵和管理的软件集合，对象及属性。从系统角度来看，文件系统是对文件存储设备的空间进行组织和分配，负责文件存储并对存入的文件进行保护和检索的系统。具体地说，它负责为用户建立文件，存入、读出、修改、转储文件，控制文件的存取，当用户不再使用时撤销文件等。

关键词：操作系统，Unix 文件系统，多级目录

目 录

摘 要	3
1. 概述	6
2. 课程设计任务及要求	6
2.1 设计任务	6
2.2 设计要求	6
3. 算法及数据结构	8
3.1 算法的总体思想	8
3.2 算法的总体数据结构	8
3.3 初始化模块	11
3.3.1 功能	11
3.3.2 数据结构	11
3.3.3 流程图	12
3.4 登录模块	13
3.4.1 功能	13
3.4.2 数据结构	13
3.4.3 流程图	14
3.5 登出模块	14
3.5.1 功能	14
3.5.2 数据结构	15
3.5.3 流程图	16
3.6 加载模块	16
3.6.1 功能	16
3.6.2 数据结构	17
3.6.3 流程图	18
3.7 主菜单模块	18
3.7.1 功能	18
3.7.2 数据结构	19
3.7.3 流程图	20
3.8 打开文件模块	21
3.8.1 功能	21
3.8.2 数据结构	21
3.8.3 流程图	22
3.9 读文件模块	23
3.9.1 功能	23
3.9.2 数据结构	23
3.9.3 流程图	24
3.10 关闭文件模块	25
3.10.1 功能	25
3.10.2 数据结构	25
3.10.3 流程图	26
3.11 创建文件模块	27
3.11.1 功能	27

3.11.2 数据结构	27
3.11.3 流程图	28
3.12 写入文件模块	28
3.12.1 功能	28
3.12.2 数据结构	28
3.12.3 流程图	30
3.13 删除文件模块	31
3.13.1 功能	31
3.13.2 数据结构	31
3.13.3 流程图	31
3.14 重命名模块（新增功能）	32
3.14.1 功能	32
3.14.2 数据结构	32
3.14.3 流程图	33
3.15 数据块分配与回收模块	33
3.15.1 功能	33
3.15.2 数据结构	34
3.15.3 流程图	34
3.16 文件和文件夹的复制模块（新增功能）	36
3.16.1 功能	36
3.16.2 数据结构	36
3.16.3 流程图	36
3.17 显示文件和文件夹模块	38
3.17.1 功能	38
3.17.2 数据结构	39
3.17.3 流程图	39
3.18 删除目录模块（新增功能）	40
3.18.1 功能	40
3.18.2 数据结构	40
3.18.3 流程图	41
3.19 目录切换模块	42
3.19.1 功能	42
3.19.2 数据结构	42
3.19.3 流程图	43
3.20 创建目录模块	44
3.20.1 功能	44
3.20.2 数据结构	45
3.20.3 流程图	46
3.21 用户注册模块	46
3.21.1 功能	46
3.21.2 数据结构	46
3.21.3 流程图	47
4. 程序设计与实现	48
4.1 程序流程图	48

4.2 程序说明.....	49
4.3 实验结果.....	49
5. 参考文献	63
6. 收获、体会和建议	63

1. 概述

文件系统是操作系统用于明确存储设备（常见的是磁盘，也有基于 **NAND Flash** 的固态硬盘）或分区上的文件的方法和数据结构；即在存储设备上组织文件的方法。操作系统中负责管理和存储文件信息的软件机构称为文件管理系统，简称文件系统。文件系统由三部分组成：文件系统的接口，对对象操纵和管理的软件集合，对象及属性。从系统角度来看，文件系统是对文件存储设备的空间进行组织和分配，负责文件存储并对存入的文件进行保护和检索的系统。具体地说，它负责为用户建立文件，存入、读出、修改、转储文件，控制文件的存取，当用户不再使用时撤销文件等。

本次实验我们实现了多级目录下的文件管理系统，具备文件系统的文件创建、删除、读写以及目录的创建、删除等操作，并在内存中开辟一块空间，模拟虚拟磁盘，成功地展示出文件系统的功能和属性。

2. 课程设计任务及要求

2.1 设计任务

多用户、多级目录结构文件系统的设计与实现——模拟实现 **UNIX** 的文件系统。

2.2 设计要求

模拟 **UNIX**（或 **LINUX**，或 **FAT**）系统的文件管理功能。

包括：

多用户: **usr1, ..., usr8** (1-8 个用户)

多级目录: 可有多级子目录

具有 **login** (用户登录)系统初始化 (建文件卷、提供登录模块)

文件的创建: **create**

文件的打开: **open**

文件的读: **read**

文件的写: **write**

文件关闭: **close**

删除文件: **delfile**

创建目录: **mkdir**

改变目录: **chdir**

列出文件目录: **dir**

退出: **logout**

格式化: **format**

以菜单列表方式给出功能选择, 然后给出参数, 再执行文件管理操作。或者模拟命令行方式输入操作命令, 接收命令, 分析命令, 执行命令。如:

```
$dir <cr>
```

```
$mkdir subdir<cr>
```

```
$creat(user_id,Filename,mode)<cr>
```

的文件系统可以保存, 以便下次开机时再用。

3. 算法及数据结构

3.1 算法的总体思想

实现 UNIX 文件系统，第一步是确定整体的数据结构，之后的功能实现直接使用结构体来实现。主要的结构体有文件系统(fileSystem)，用户信息(user)，数据块(dataBlock)，目录(director)，文件(file)，打开文件(openFile)。最关键的 fileSystem 的结构体中主要有数据域 dataArea，超级栈 superStack，成组链数组 free_list，用户信息 user_info，文件信息 vector_file，目录信息 vector_director。在目录信息结构中主要含有目录索引号 id，本目录中的文件链表 file_list，本目录中的文件夹链表 director_list。在文件信息结构中主要有文件索引号 id，在数据域中的第一个位置 beginning_in_dataArea。

除了数据结构外，最根本的功能是数据块的分配与回收。其中，分配超级块的功能主要通过让 free 数组中的数据块进入超级栈的方式来实现，即 allocdatablock 通过调用 intosuperstack()方法实现功能。释放超级块的过程通过调入 intosuperstack 和 intofreearray 两个方法共同实现，当 superstack 栈满了之后，就让数据块进入 freearray 中，成组链表增加。

其他的功能都是基于数据块的分配和回收来实现。

3.2 算法的总体数据结构

使用的开发环境是 Visual Studio 2012。

```
const int DATA_BLOCK_LENGTH = 512 ; // 数据块内容大小
const int DATA_BLOCK_NUMBER = 512 ; // 数据块数目
const int INDEX_LIST_LENGTH = DATA_BLOCK_LENGTH/sizeof(int); // 成
组链中每组个数
```



```

const int INDEX_LIST_NUMBER = sizeof(int);          // 成组链组数z
const int MAX_FILE_LENGTH = 10;                    // 允许的最大文件长度，即占数据块的最大
数目
const int User_NUMBER = 8;                          // 用户数
const string root_path = "d:\\MyFileSystem";

typedef struct file
{
    int id;                // 文件索引号
    string file_name;      // 文件名
    int beginning_in_dataArea; // 在数据域中的第一个位置
    string owner;          // 文件所有者的名字
    int file_length;       // 文件所占数据块块数
    int begining_in_memory; // 在内存中的第一个位置
}file;

typedef struct director
{
    int id;                // 目录索引号
    string name;           // 目录名
    list<int> file_list;    // 本目录中的文件链表
    list<int> director_list; // 本目录中的文件夹链表
    string owner;          // 文件夹拥有者
    int last_director;     // 上一级目录，根目录的上一级 设为 -1
}director;

typedef struct dataBlock //数据块
{
    int used;             // 已使用的空间
    int next;             // 下一个数据块
    char content[DATA_BLOCK_LENGTH+1]; // 数据块内容 最后一位放'\0'
}dataBlock;

typedef struct user
{
    string name;
    string password;
}user;

typedef struct openFile
{
    int file_index; // 文件索引
    int director_index; // 目录索引
}openFile;

```

```

typedef struct fileSystem    //文件系统
{
    vector<dataBlock> dataArea;    // 数据域
    stack<int>          superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH];
// 成组链数组，被使用了置成-1
// 从free_list[0][127]—freelist[0][0]—free_list[1][127]—free_list[3][0]
    vector<user> user_info;    // 用户信息
    vector<director> vector_director;    // 存放所有目录信息
    vector<file> vector_file;    // 存放所有文件信息
}fileSystem;

```

◁主要函数▷

```

extern file current_file;    // 当前打开的文件
extern int next_free_list_index;    // 下一组要使用的组数
0~INDEX_LIST_NUMBER-1
extern int super_stack_number;    // 超级栈中可以使用的块数
extern int current_director_index; // 当前目录的索引
extern user current_user;    // 当前登录的用户
extern char memory[MAX_FILE_LENGTH][DATA_BLOCK_LENGTH];    //
内存10×512
extern int memory_index;    // 可用内存索引,即开始的位置
extern fileSystem myFileSystem;    // 文件系统
extern list<openFile> open_file_list;    // 打开的文件链表
extern bool Initialize();    // 初始化文件系统
extern bool LoginIn();    // 登录模块
extern bool SignIn();    //注册模块
extern void print();    // 画分割线
extern void mkdir(string director_name);    // 新建目录
extern void dir();    // 显示该目录下内容
extern bool ntmdir(string director_name);    // 前往下一级目录
extern void ltdir();    // 前往上一级目录
extern int AllocDataBlock();    // 分配数据块
extern void ReleaseDataBlock(int index);    // 释放数据块
extern bool create(string file_name); //创建文件
extern bool delfile(string file_name); // 删除文件
extern bool deldir(string director_name); //删除目录
extern int IsFileInCurrentDirector(string file_name);
// 判断文件是否在当前目录中    返回文件索引号
extern int IsDirectorInCurrentDirector(string file_name);
// 判断文件是否存在于当前目录中    返回目录索引号

```

```

extern list<openFile>::iterator IsInOpenFileList(int pos);
// 判断文件是否已打开
extern bool write(string file_name,string content);    // 向文件写入内容
extern bool wrmore(string file_name,string content,int time);
//向文件写入同一内容多次
extern bool open(string file_name);
// 打开文件，将文件内容读到内存
extern bool read(string file_name);    // 从内存中将文件内容读出来
extern bool close(string file_name);    // 将打开的文件关闭
extern bool LoginOut();                // 退出系统，保存已有信息
extern bool Load();                   // 载入保存的信息
extern void Initialize_Memory();    //初始化内存
extern bool rename(string old_name,string new_name); //重命名
extern bool copy(string name,string path);    // 复制

```

3.3 初始化模块

3.3.1 功能

本模块实现：初始化数据块，初始化成组链，初始化用户信息，初始化目录信息，初始化内存。实际操作即分别将各个数据结构中的数据置零置空，使用户得到一个初始空白的系统。

3.3.2 数据结构

```

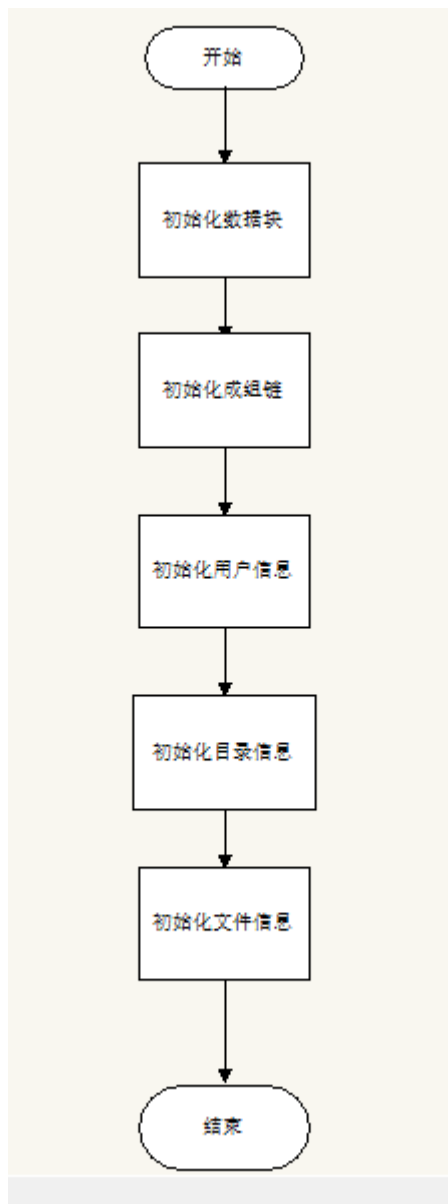
typedef struct fileSystem    //文件系统
{
    vector<dataBlock> dataArea;    // 数据域
    stack<int> superStack;    // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH];    // 成组链数组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info;    // 用户信息
    vector<director> vector_director;    // 存放所有目录信息
    vector<file> vector_file;    // 存放所有文件信息
}fileSystem;

extern char memory[MAX_FILE_LENGTH][DATA_BLOCK_LENGTH];    // 内存
extern int memory_index;    // 可用内存索引,即开始的位置
extern fileSystem myFileSystem;    // 文件系统

```

```
extern int current_director_index; // 当前目录的索引
extern int next_free_list_index;
// 下一组要使用的组数 0~INDEX_LIST_NUMBER-1
extern int super_stack_number; // 超级栈中可以使用的块数
extern bool Initialize(); // 初始化文件系统
const string root_path = "d:\\MyFileSystem";
extern void Initialize_Memory(); //初始化内存
```

3.3.3 流程图



3.4 登录模块

3.4.1 功能

本功能模块实现文件系统的用户登录功能。用户可通过该模块进入系统，管理文件，本模块可供 **user1** 到 **user8** 登录，接收用户名和密码并判断是否符合，如果符合则可进入系统。

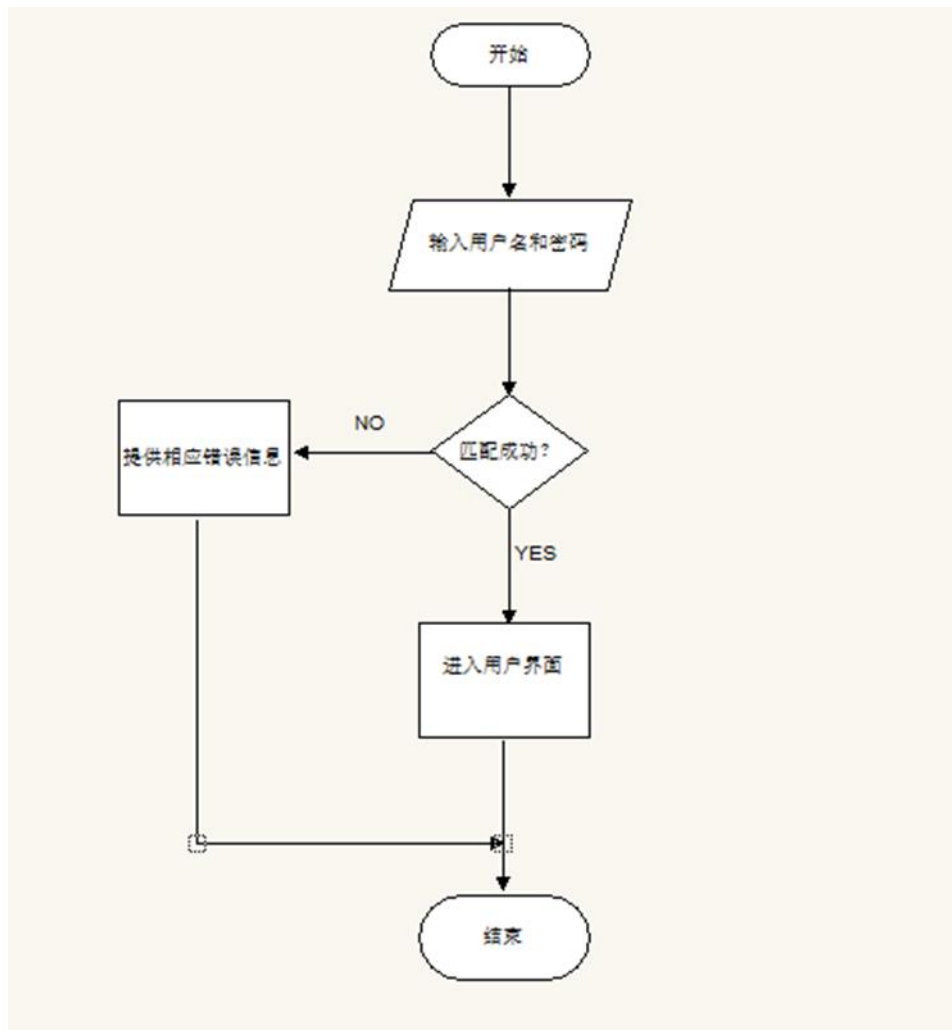
3.4.2 数据结构

```
typedef struct fileSystem      //文件系统
{
    vector<dataBlock> dataArea;    // 数据域
    stack<int>          superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info;        // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file;       // 存放所有文件信息
}fileSystem;

typedef struct user
{
    string name;
    string password;
}user;

extern user current_user;        // 当前登录的用户
extern bool LoginIn();          // 登录模块
```

3.4.3 流程图



3.5 登出模块

3.5.1 功能

本模块实现退出系统时保存数据信息的功能，其中包括保存目录信息，保存文件信息，保存用户信息，保存成组链，保存超级栈，保存数据域内容。其中每个模块都以相应的格式输出到指定文件中，逐步保存信息，以便以后登录时信息查看和操作。但在当前的文件处于打开状态时无法保存。

3.5.2 数据结构

由于在前面已给出具体的数据结构，所以这里就简单给出该模块所使用的数据结构名称和函数。

```
typedef struct file //文件
{ }file;

typedef struct director //目录
{ }director;

typedef struct dataBlock //数据块
{ }dataBlock;

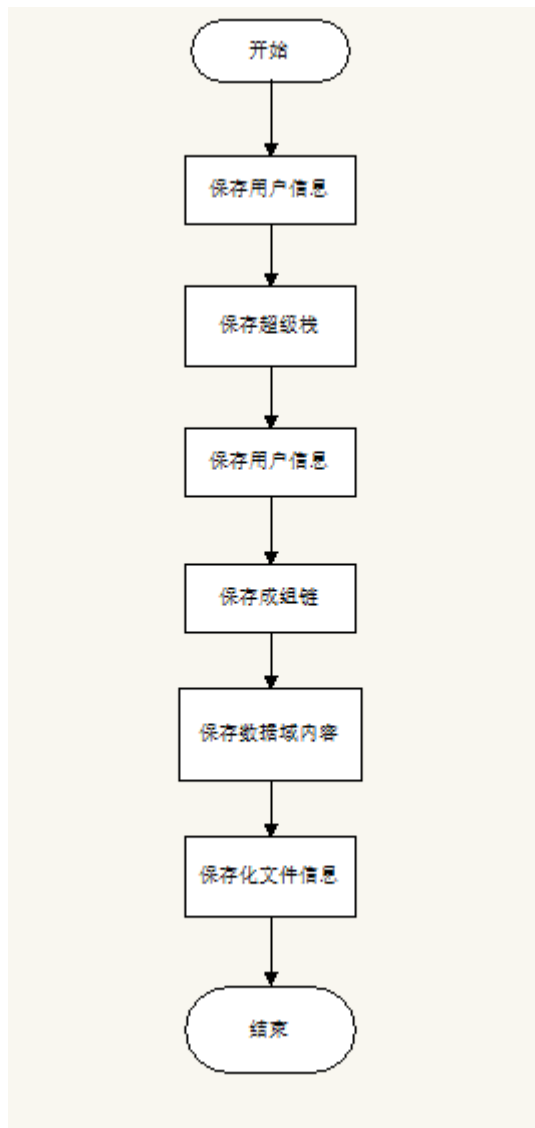
typedef struct user //用户
{ }user;

typedef struct openFile //打开文件
{ }openFile;

typedef struct fileSystem //文件系统
{ }fileSystem;

extern bool LoginOut(); // 退出系统，保存已有信息
const string root_path = "d:\\MyFileSystem";
const int DATA_BLOCK_LENGTH = 512 ; // 数据块内容大小
const int DATA_BLOCK_NUMBER = 512 ; // 数据块数目
const int INDEX_LIST_LENGTH = DATA_BLOCK_LENGTH/sizeof(int);
// 成组链中每组个数
const int INDEX_LIST_NUMBER = sizeof(int); // 成组链组数
```

3.5.3 流程图



3.6 加载模块

3.6.1 功能

本模块实现:加载目录信息,加载文件信息,加载用户信息,加载成组链,加载超级栈信息,加载数据域信息。在每一个分模块中又通过 **fscanf** 写入内存中,逐个遍历以达到加载的效果。

3.6.2 数据结构

由于在前面已给出具体的数据结构，所以这里就简单给出该模块所使用的数据结构名称和函数。

```
typedef struct director  
{ }director;
```

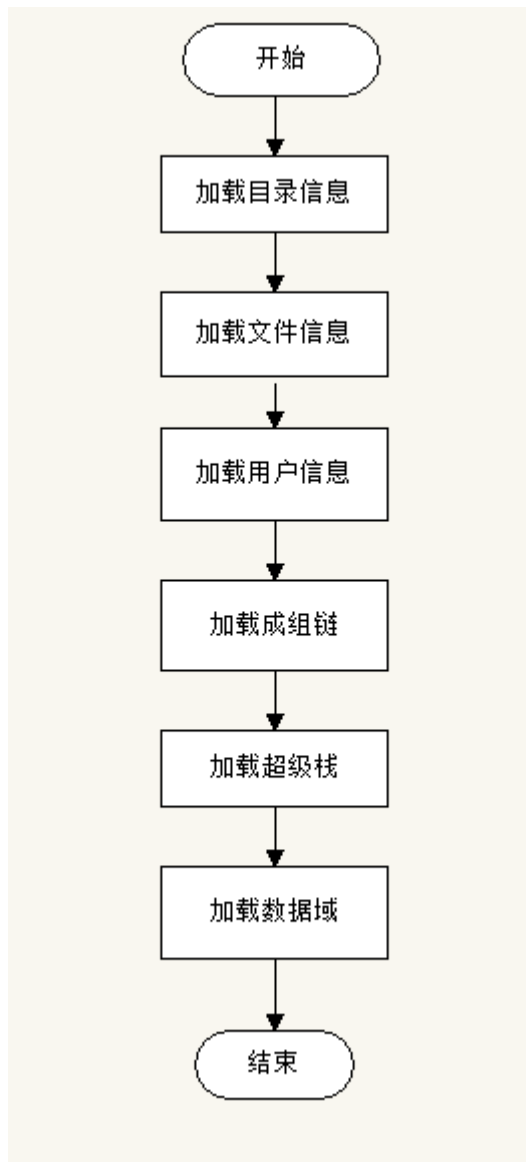
```
typedef struct file  
{ }file;
```

```
typedef struct user  
{ }user;
```

```
typedef struct fileSystem      //文件系统  
{ }fileSystem;
```

```
const int INDEX_LIST_LENGTH = DATA_BLOCK_LENGTH/sizeof(int);  
//成组链中每组个数  
const int INDEX_LIST_LENGTH = DATA_BLOCK_LENGTH/sizeof(int);  
//成组链中每组个数  
const int INDEX_LIST_NUMBER = sizeof(int); //成组链组数  
const int DATA_BLOCK_LENGTH = 512 ; // 数据块内容大小  
const int DATA_BLOCK_NUMBER = 512 ; // 数据块数目  
const string root_path = "d:\\MyFileSystem";  
extern int memory_index;      // 可用内存索引,即开始的位置
```

3.6.3 流程图



3.7 主菜单模块

3.7.1 功能

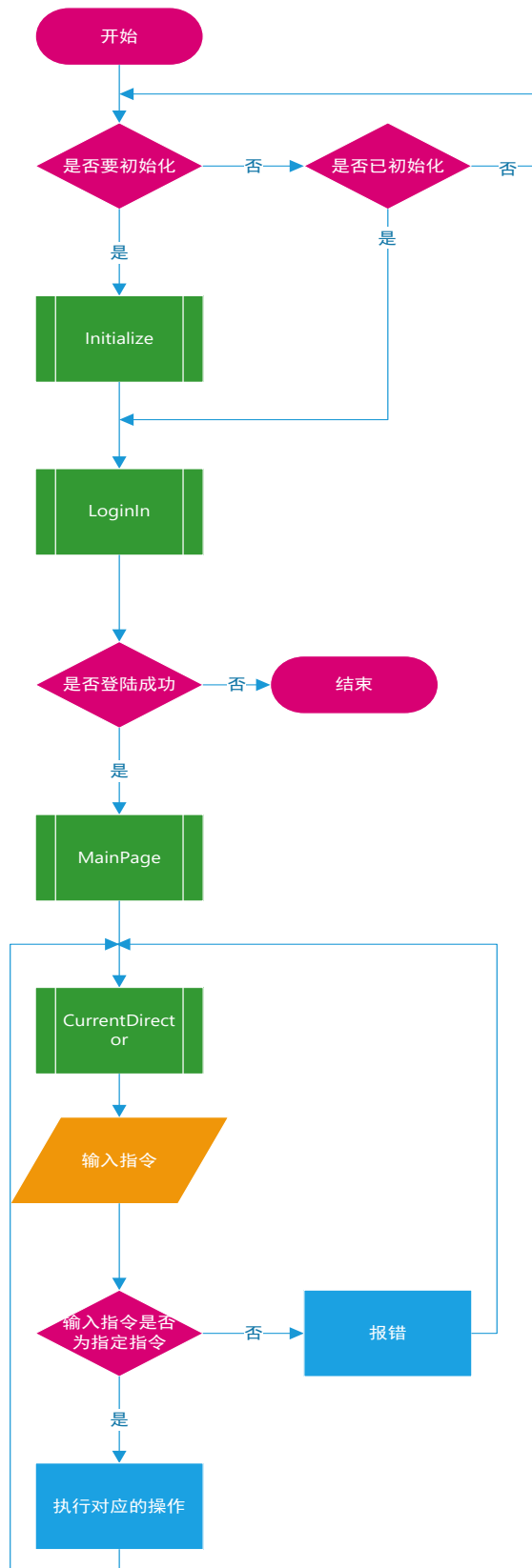
主函数的逻辑实现。从初始化开始，到用户登录界面、再到执行指令的菜单界面，最后读取执行用户输入指令。

3.7.2 数据结构

```
typedef struct director
{
    int id;                // 目录索引号
    string name;           // 目录名
    list<int> file_list;    // 本目录中的文件链表
    list<int> director_list; // 本目录中的文件夹链表
    string owner;          // 文件夹拥有者
    int last_director;     // 上一级目录      // 根目录的上一级 设为 -1
}director;

typedef struct fileSystem //文件系统
{
    vector<dataBlock> dataArea; // 数据域
    stack<int> superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info; // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file; // 存放所有文件信息
}fileSystem;
```

3.7.3 流程图



3.8 打开文件模块

3.8.1 功能

将用户给定的文件打开，把文件内容存入内存，方便用户读取，把文件加入打开文件表。通过文件索引找到文件的起始位置，再连续读入直到读到文件在内存中的最后一块将整个文件打开。

通过两个主要函数 `IsDirectorInCurrentDirector` 、
`IsFilerInCurrentDirector` 实现查找目录和文件。

3.8.2 数据结构

由于在前面已给出具体的数据结构，所以这里就简单给出该模块所使用的数据结构名称和函数。

```
typedef struct openFile  
{ }openFile;
```

```
typedef struct file  
{ }file;
```

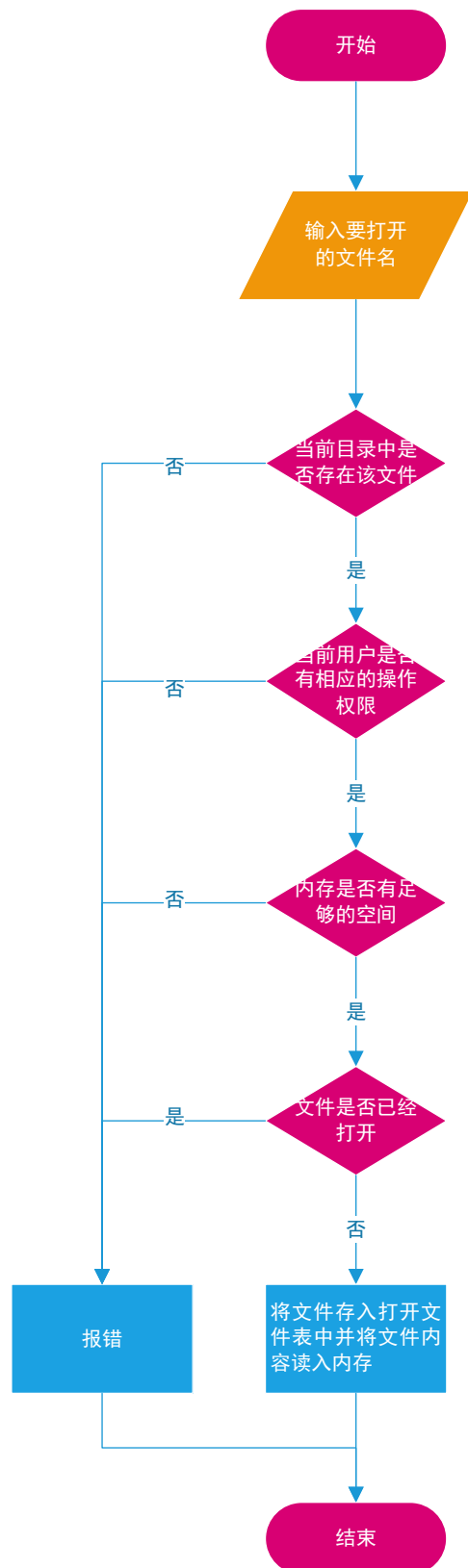
```
typedef struct fileSystem           //文件系统  
{ }fileSystem;
```

```
typedef struct user  
{ }user;
```

```
typedef struct dataBlock    //数据块  
{ }dataBlock;
```

```
int memory_index;  
char memory[MAX_FILE_LENGTH][DATA_BLOCK_LENGTH]; //内存
```

3.8.3 流程图



3.9 读文件模块

3.9.1 功能

将用户给定的已打开并已经将文件内容存入内存的文件从内存中读取出来，并把内容打印到屏幕上。同样通过两个主要函数实现查找目录和文件 `IsDirectorInCurrentDirector`、`IsFilerInCurrentDirector` 找到文件后创建临时文件对象 `Temp` 来保存该文件并从文件块开始读到结束。

3.9.2 数据结构

```
typedef struct openFile
{
    int file_index; // 文件索引
    int director_index; // 目录索引
}openFile;

typedef struct file
{
    int id; // 文件索引号
    string file_name; // 文件名
    int beginning_in_dataArea; // 在数据域中的第一个位置
    string owner; // 文件所有者的名字
    int file_length; // 文件所占数据块块数
    int begining_in_memory; // 在内存中的第一个位置
}file;

list<openFile> open_file_list;

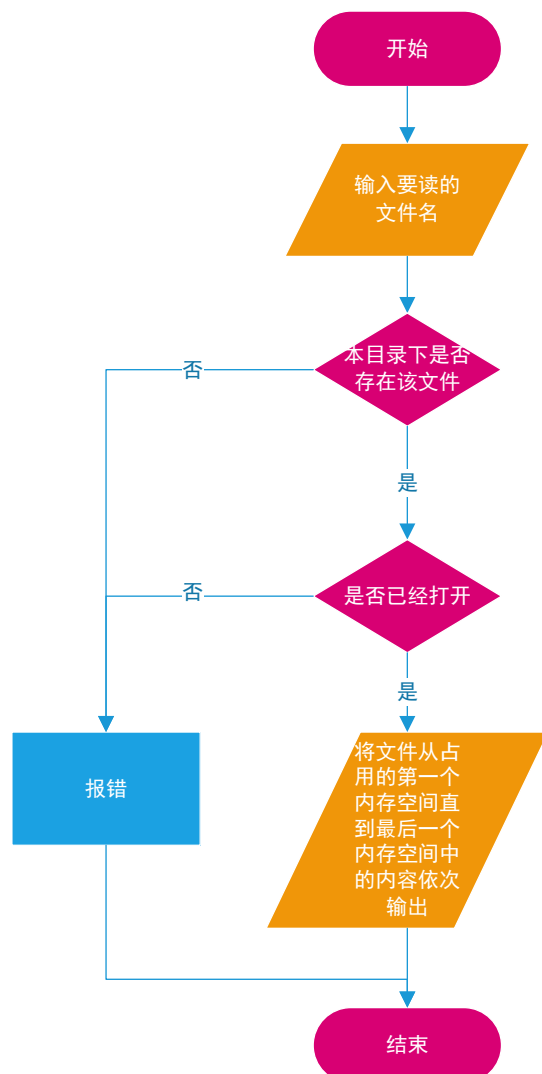
typedef struct fileSystem //文件系统
{
    vector<dataBlock> dataArea; // 数据域
    stack<int> superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info; // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file; // 存放所有文件信息
}
```

```
}fileSystem;
```

```
typedef struct user  
{  
    string name;  
    string password;  
}user;
```

```
char memory[MAX_FILE_LENGTH][DATA_BLOCK_LENGTH];
```

3.9.3 流程图



3.10 关闭文件模块

3.10.1 功能

将用户给定的文件关闭，把文件在内存所占用的空间释放，方便用户读取其他文件进入内存，把文件从打开文件表删除。首先检查打开文件链表有无文件，若没有则提示关闭失败，若存在该文件则遍历打开文件链表找到该文件，再擦除该文件在打开文件链表的数据。

3.10.2 数据结构

由于在前面已给出具体的数据结构，所以这里就简单给出该模块所使用的数据结构名称和函数。

```
typedef struct fileSystem      //文件系统
{
    vector<dataBlock> dataArea;    // 数据域
    stack<int>          superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info;        // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file;      // 存放所有文件信息
}fileSystem;

typedef struct openFile
{
    int file_index; // 文件索引
    int director_index; // 目录索引
}openFile;

typedef struct file
{
    int id; // 文件索引号
    string file_name; // 文件名
    int beginning_in_dataArea; // 在数据域中的第一个位置
    string owner; // 文件所有者的名字
    int file_length; // 文件所占数据块块数
    int begining_in_memory; // 在内存中的第一个位置
```

```

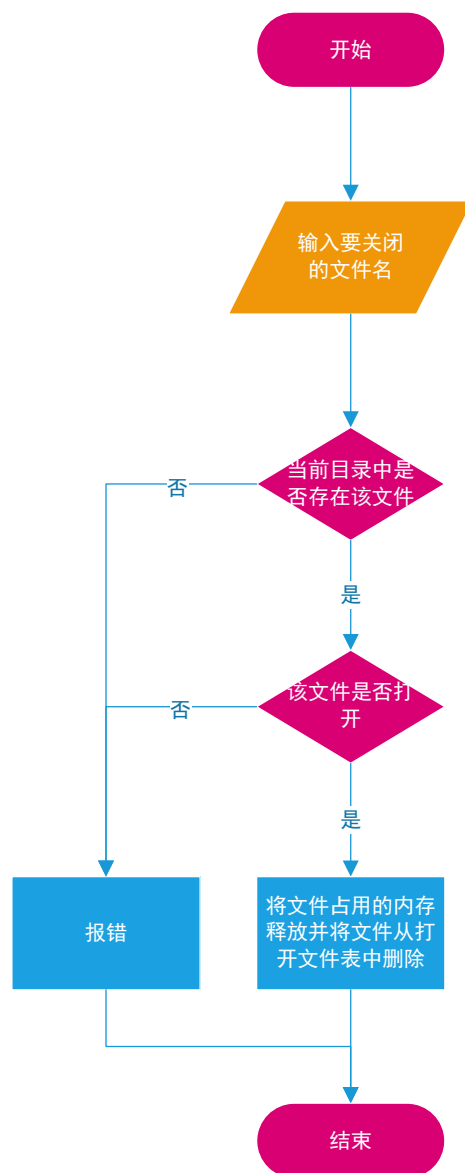
}file;

typedef struct dataBlock //数据块
{
    int used; // 已使用的空间
    int next; // 下一个数据块
    char content[DATA_BLOCK_LENGTH+1]; // 数据块内容 最后一位放'\0'
}dataBlock;

int memory_index;
char memory[MAX_FILE_LENGTH][DATA_BLOCK_LENGTH]; //内存

```

3.10.3 流程图



3.11 创建文件模块

3.11.1 功能

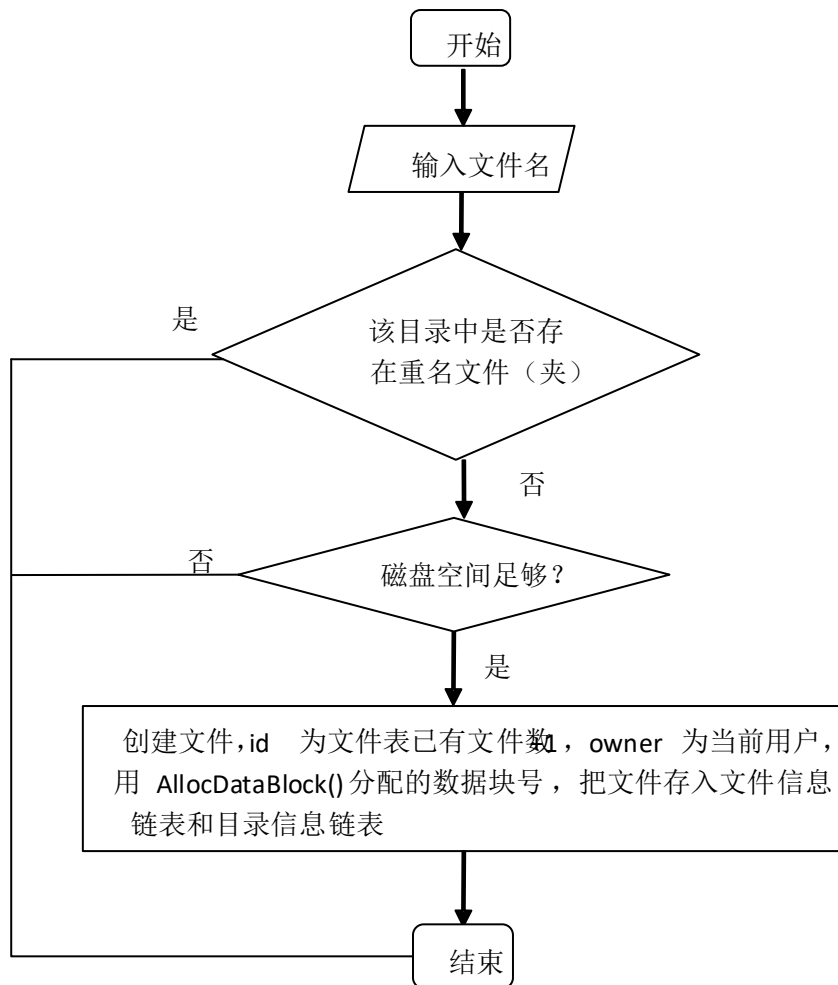
在系统中创建一个文件，这个文件需要由用户自己命名，然后存储在用户指定的位置，占用一定的磁盘空间，从空闲块链表最尾端开始使用，如果写入大小大于一个块，再依据上述原理重新分配一个块。最后在文件链表中添加相应信息。

3.11.2 数据结构

```
typedef struct file
{
    int id;                // 文件索引号
    string file_name;      // 文件名
    int beginning_in_dataArea; // 在数据域中的第一个位置
    string owner;          // 文件所有者的名字
    int file_length;       // 文件所占数据块块数
    int beginning_in_memory; // 在内存中的第一个位置
}file;

typedef struct fileSystem //文件系统
{
    vector<dataBlock> dataArea; // 数据域
    stack<int> superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info; // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file; // 存放所有文件信息
}fileSystem;
```

3.11.3 流程图



3.12 写入文件模块

3.12.1 功能

把用户给定的数据写入指定的文件，并保存在文件中。文件有自己的结构体，其中包含了文件索引和所在数据域位置等信息，还有文件所有者信息定义了该文件的读写和使用权限，其余 **user** 不能使用（固定权限）。

3.12.2 数据结构

```
typedef struct file
{
    int id;           // 文件索引号
```

```

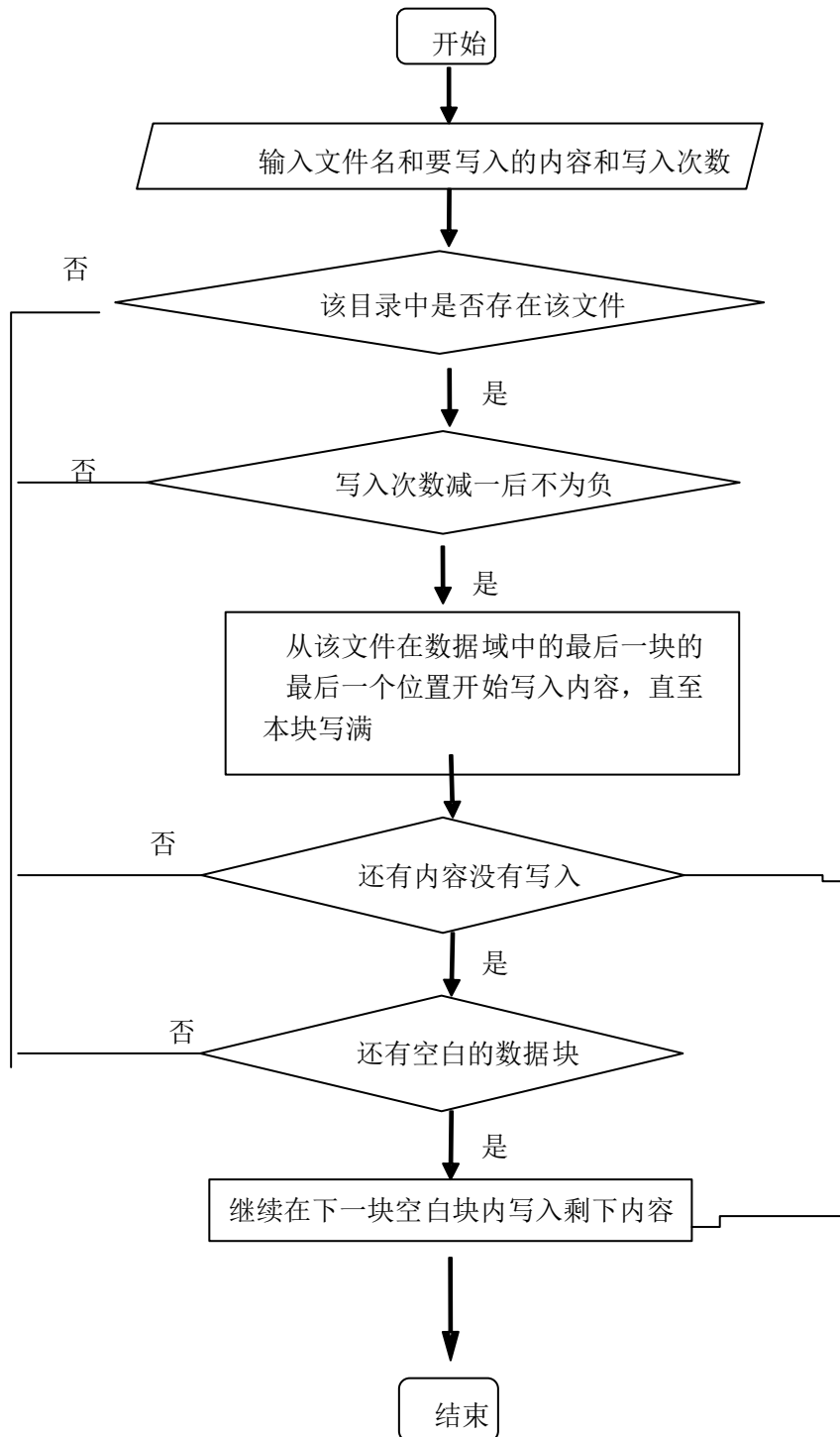
    string file_name;        // 文件名
    int    beginning_in_dataArea;    // 在数据域中的第一个位置
    string owner;            // 文件所有者的名字
    int    file_length;      // 文件所占数据块块数
    int    begining_in_memory;    // 在内存中的第一个位置
}file;

typedef struct fileSystem    //文件系统
{
    vector<dataBlock> dataArea;    // 数据域
    stack<int>        superStack;  // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数
    组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info;        // 用户信息
    vector<director> vector_director;    // 存放所有目录信息
    vector<file> vector_file;          // 存放所有文件信息
}fileSystem;

typedef struct dataBlock    //数据块
{
    int used;                // 已使用的空间
    int next;                // 下一个数据块
    char content[DATA_BLOCK_LENGTH+1]; // 数据块内容 最后一位放'\0'
}dataBlock;

```

3.12.3 流程图



3.13 删除文件模块

3.13.1 功能

系统提供删除文件的功能。用户可以删除不需要的文件。在释放数据块后修改文件链表的信息。

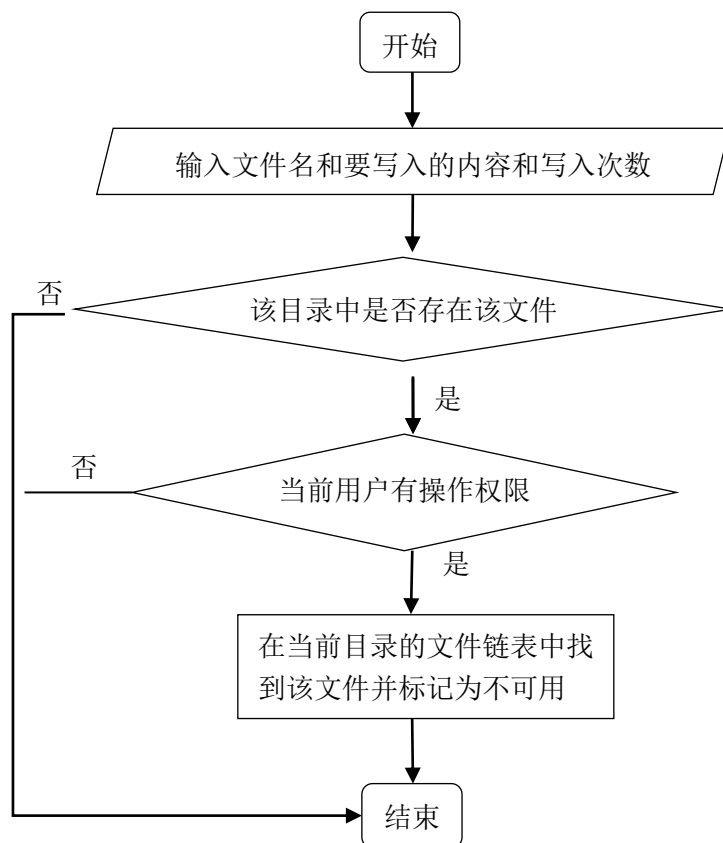
3.13.2 数据结构

由于在前面已给出具体的数据结构, 所以这里就简单给出该模块所使用的数据结构名称和函数。

```
typedef struct user  
{ }user;
```

```
typedef struct director  
{ }director;
```

3.13.3 流程图



3.14 重命名模块（新增功能）

3.14.1 功能

重命名时，先检查新名字是否与当前目录中的文件或文件夹重名，若重名则重命名失败，若不重名则重命名成功。

3.14.2 数据结构

```
typedef struct director
{
    int id;                // 目录索引号
    string name;           // 目录名
    list<int> file_list;    // 本目录中的文件链表
    list<int> director_list; // 本目录中的文件夹链表
    string owner;          // 文件夹拥有者
    int last_director;     // 上一级目录          // 根目录的上一级 设为 -1
}director;

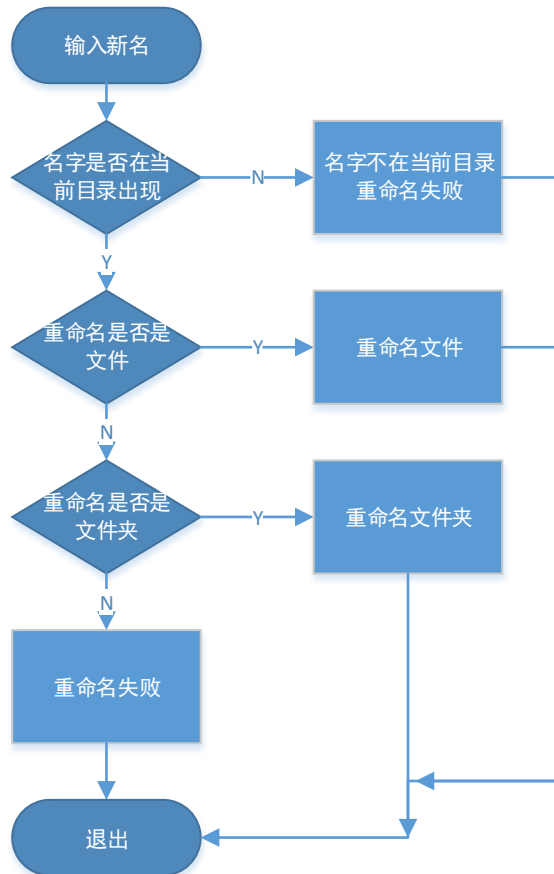
typedef struct fileSystem //文件系统
{
    vector<dataBlock> dataArea; // 数据域
    stack<int> superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数组
    // 使用的时候从小到大 被使用了置成-1
    // 即，从 free_list[0][127]开始，用到 freelist[0][0]，用完再 free_list[1][127]。。。
    // 最后是 free_list[3][0]
    user user_info[MAX_User_NUMBER]; // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file; // 存放所有文件信息
}fileSystem;

typedef struct file
{
    int id;                // 文件索引号
    string file_name;       // 文件名
    int beginning_in_dataArea; // 在数据域中的第一个位置
    string owner;          // 文件所有者的名字
    int file_length;       // 文件所占数据块块数
```



```
int begining_in_memory;    // 在内存中的第一个位置
}file;
```

3.14.3 流程图



3.15 数据块分配与回收模块

3.15.1 功能

数据块的分配与回收是文件系统实现的基础。

分配超级块的功能主要通过让 **free** 数组中的数据块进入超级栈的方式来实现，在 **free** 数组进入超级栈方法中，遍历成组链中的每一个数据块，使用的时候从小到大，被使用的就置成-1，如果遍历后没到最后一块就+1，最后，**allocdatablock** 通过调用 **intosuperstack()**方法实现功能。

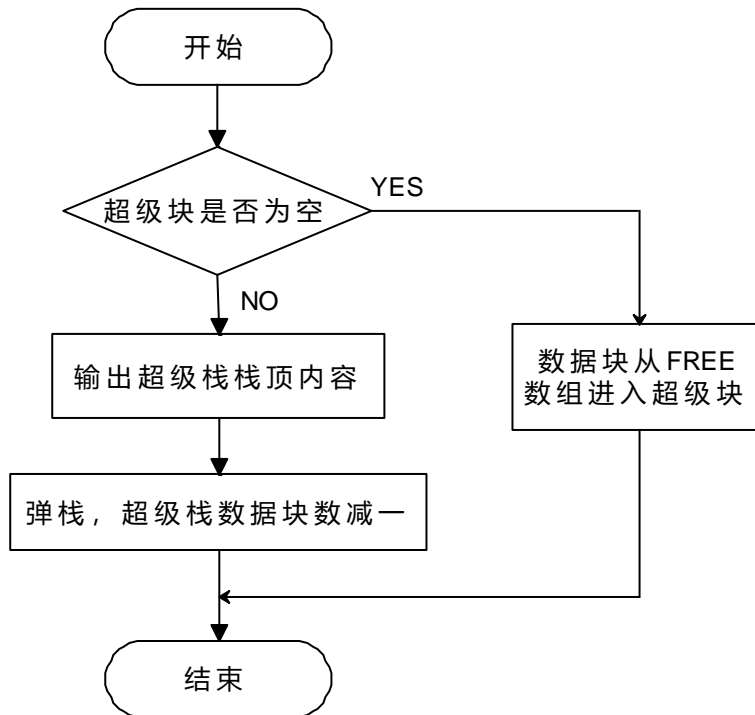
释放超级块的过程通过调入 `intosuperstack` 和 `intofreearray` 两个方法共同实现。设被释放空闲块为编号 `N`。查看超级块中是否栈已满；若不是，则 `N` 入栈，`++count`；当 `superstack` 栈满了之后，调用 `intofreearray`，将超级块中的栈写入到空闲块 `N`，然后把 `N` 放入内存空闲块栈中的栈顶。

3.15.2 数据结构

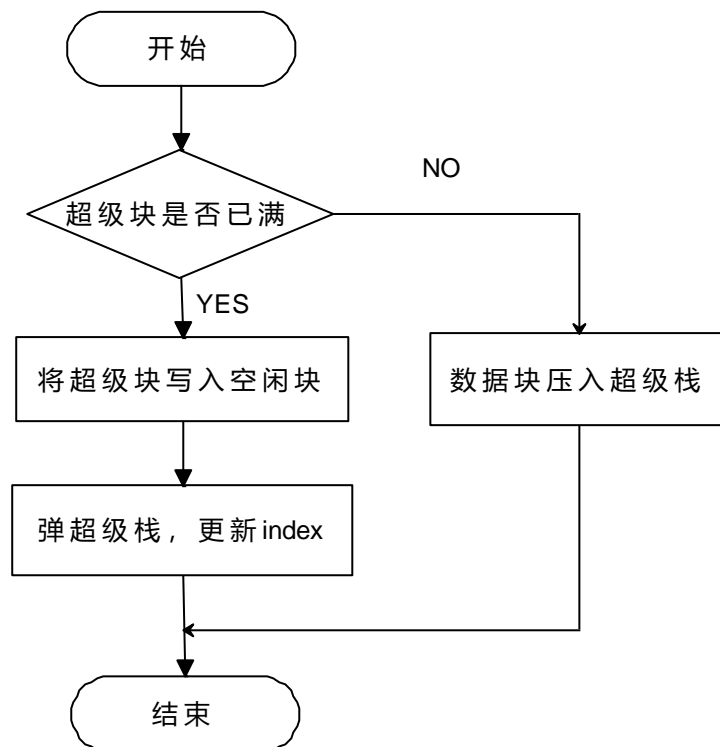
```
const int DATA_BLOCK_LENGTH = 512 ; // 数据块内容大小
const int DATA_BLOCK_NUMBER = 512 ; // 数据块数目
const int INDEX_LIST_LENGTH = DATA_BLOCK_LENGTH/sizeof(int); //
成组链中每组个数
const int INDEX_LIST_NUMBER = sizeof(int); // 成组链组数 z
typedef struct dataBlock //数据块
{
    int used; // 已使用的空间
    int next; // 下一个数据块
    char content[DATA_BLOCK_LENGTH+1]; // 数据块内容 最后一位放'\0'
}dataBlock;
extern int AllocDataBlock(); // 分配数据块
extern void ReleaseDataBlock(int index); // 释放数据块
```

3.15.3 流程图

1. 分配数据块



2. 回收数据块



3.16 文件和文件夹的复制模块（新增功能）

3.16.1 功能

将用户给定的文件或文件夹，复制到用户给定的目录下。

如果复制的是文件，则在目标目录下创建一个新的文件，分配新的数据块，再将原文件的数据块内容拷贝到新的数据块中。

如果复制的是文件夹，那么该文件夹下的所有文件也被复制到新的目录下。并且如果文件夹中存在子文件夹，则使用递归方法继续拷贝子文件夹。

3.16.2 数据结构

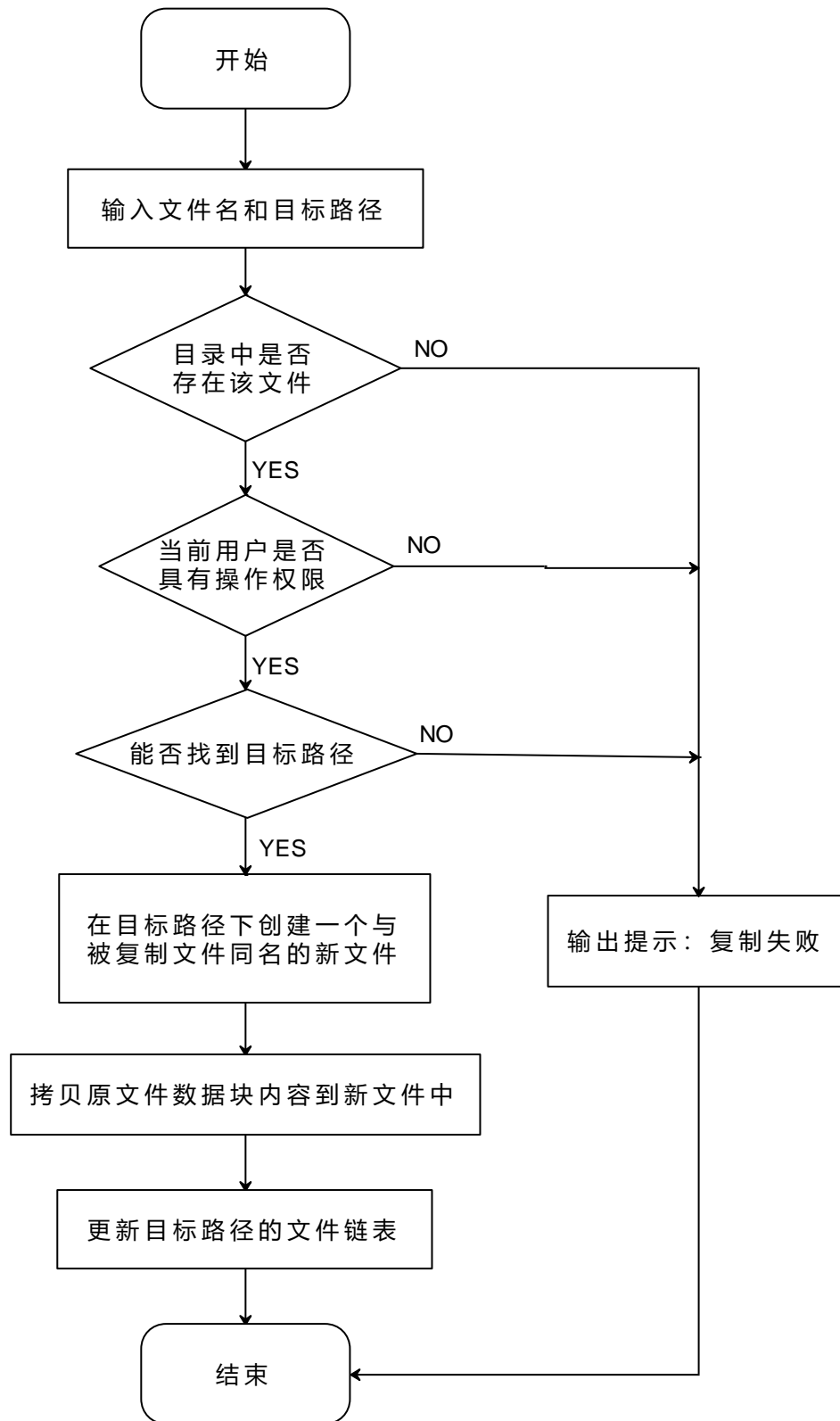
```
extern int current_director_index; // 当前目录的索引

typedef struct fileSystem //文件系统
{
    vector<dataBlock> dataArea; // 数据域
    stack<int> superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info; // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file; // 存放所有文件信息
}fileSystem;

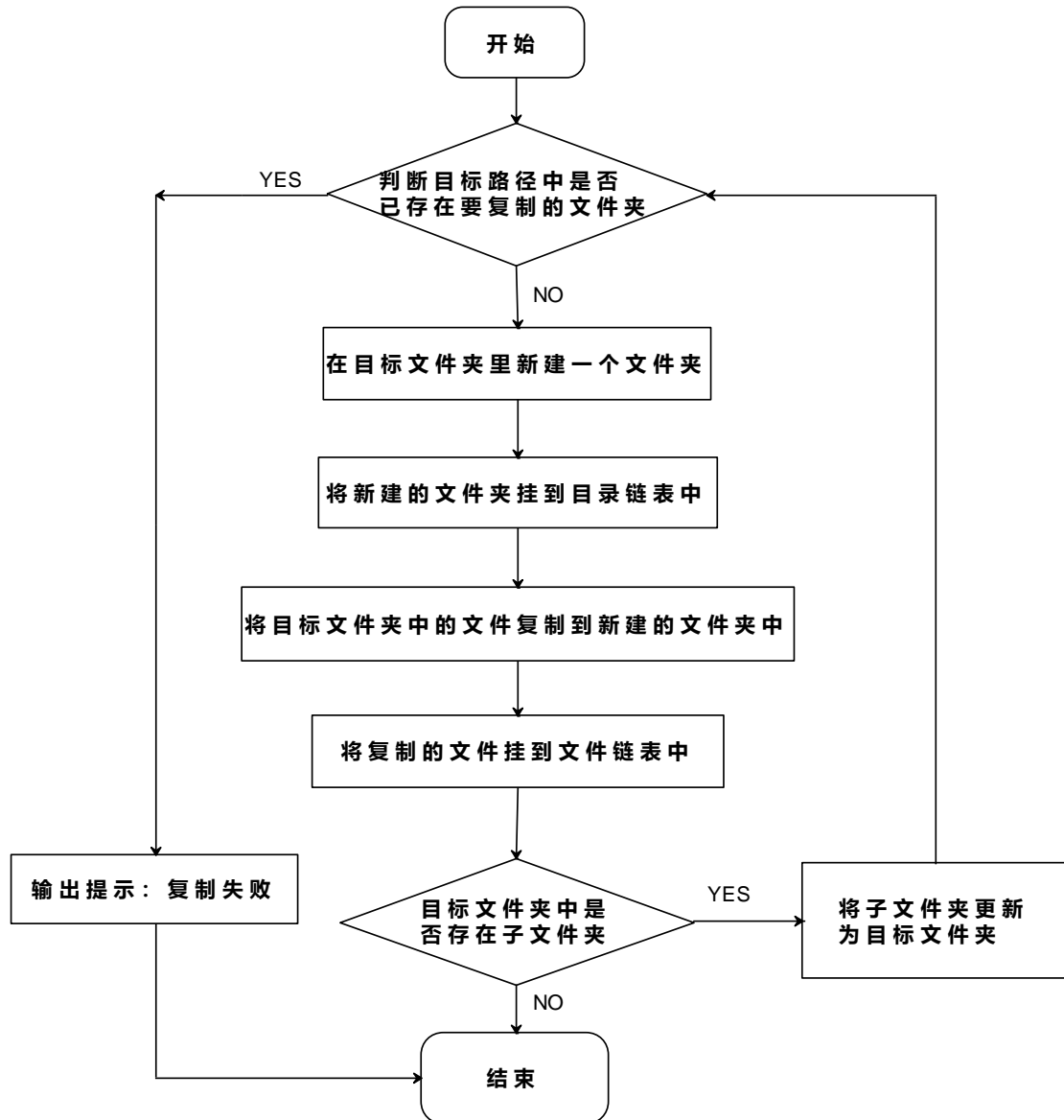
typedef struct user
{
    string name;
    string password;
}user;
```

3.16.3 流程图

1.文件的复制



2. 文件夹的复制



3.17 显示文件和文件夹模块

3.17.1 功能

提供显示该目录下的文件夹和文件的功能。目录下的文件与文件夹都是通过链表方式组织的，只要通过指针将链表从头结点逐个扫描至尾节点，依次显示即可。

3.17.2 数据结构

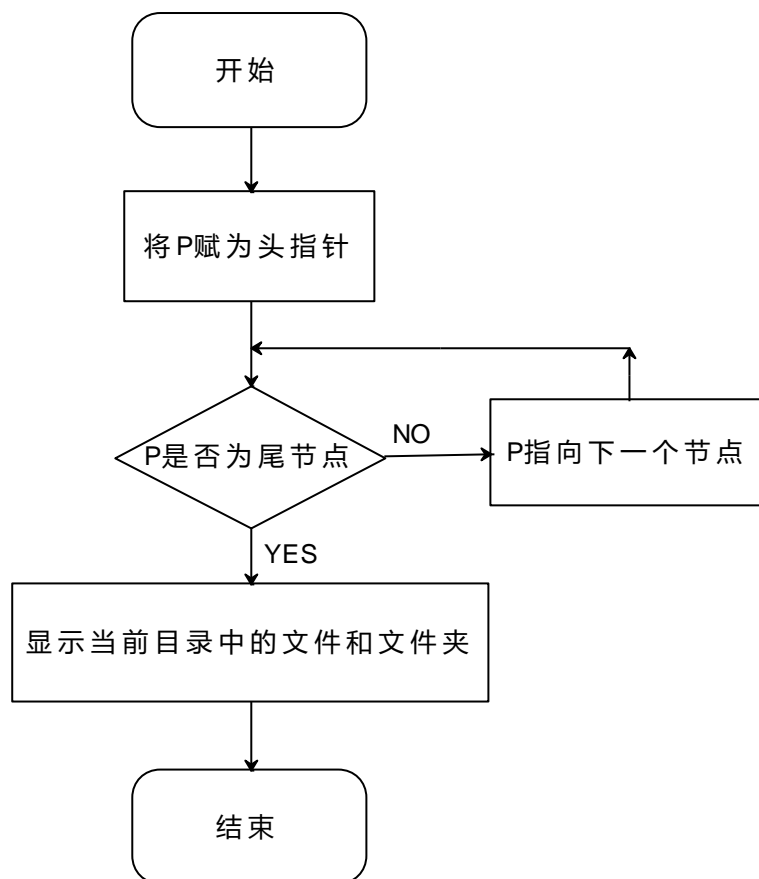
由于在前面已给出具体的数据结构，所以这里就简单给出该模块所使用的数据结构名称和函数。

```
typedef struct director  
{ }director;
```

```
typedef struct fileSystem    //文件系统  
{ }fileSystem;
```

```
typedef struct file  
{ }file;
```

3.17.3 流程图



3.18 删除目录模块（新增功能）

3.18.1 功能

提供删除文件夹的功能。用户可以选择删除不需要的文件夹。删除目录分为两种情况，即删除空文件夹和删除非空文件夹。空文件夹直接删除；非空文件夹先删除文件夹下的所有内容后调用删除空文件夹函数将空文件夹删除。

3.18.2 数据结构

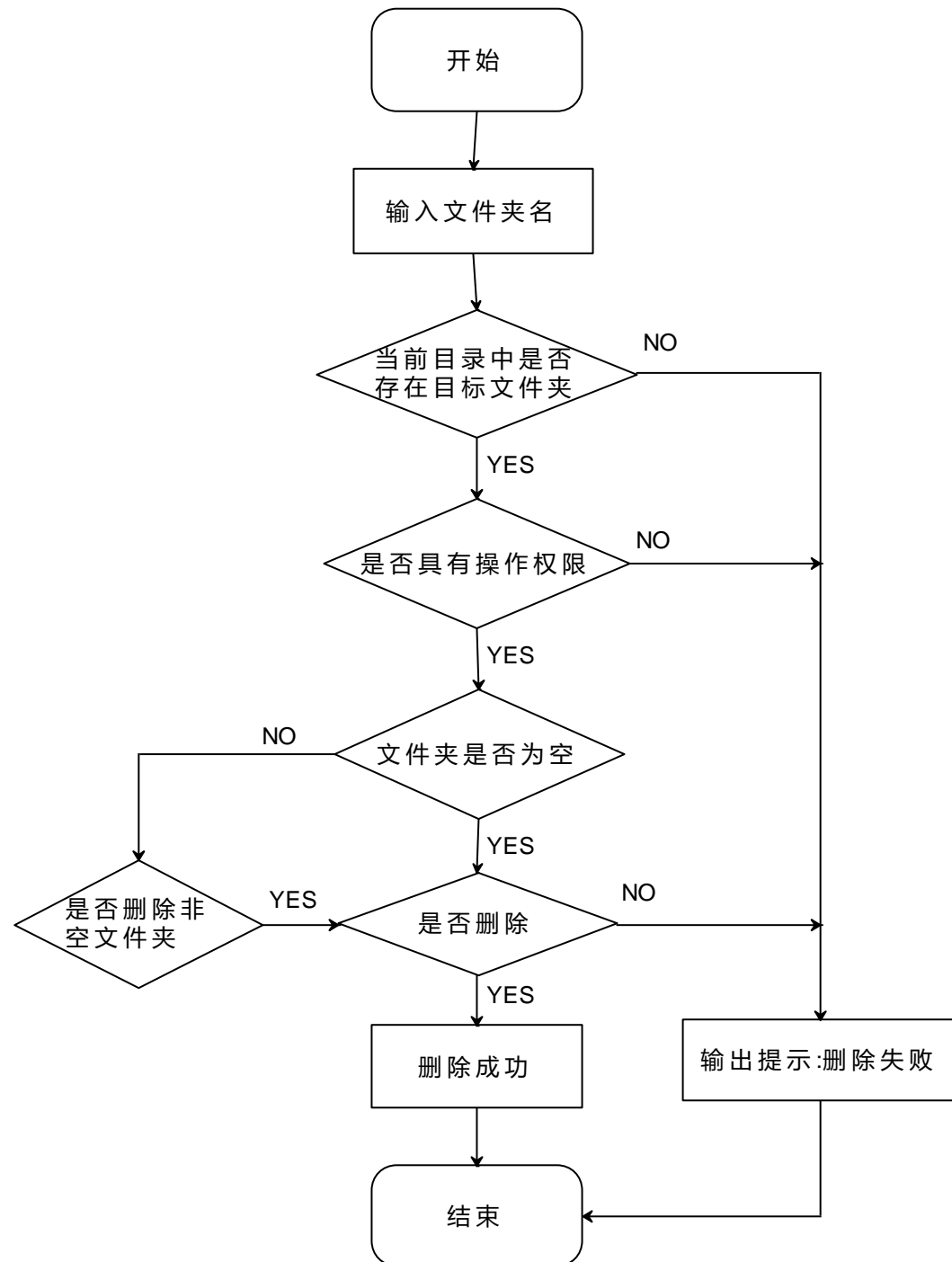
```
typedef struct director
{
    int id;                // 目录索引号
    string name;           // 目录名
    list<int> file_list;    // 本目录中的文件链表
    list<int> director_list; // 本目录中的文件夹链表
    string owner;          // 文件夹拥有者
    int last_director;     // 上一级目录          // 根目录的上一级 设为 -1
}director;

typedef struct fileSystem //文件系统
{
    vector<dataBlock> dataArea; // 数据域
    stack<int> superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info; // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file; // 存放所有文件信息
}fileSystem;

typedef struct file
{
    int id;                // 文件索引号
    string file_name;      // 文件名
    int beginning_in_dataArea; // 在数据域中的第一个位置
    string owner;          // 文件所有者的名字
    int file_length;       // 文件所占数据块块数
    int begining_in_memory; // 在内存中的第一个位置
}
```


}file;

3.18.3 流程图



3.19 目录切换模块

3.19.1 功能

提供目录切换操作，包括返回上一级和进入下一级目录。

返回上一级目录，先判断当前是否在根目录中，若在显示已在根目录中，不能向上；若不在则将上一层目录索引返回给当前索引。

进入下一级目录，逐个检索目录，判断目录名是否匹配，匹配成功后还需检测用户权限，都满足才可以进入下一级目录，否则失败。

3.19.2 数据结构

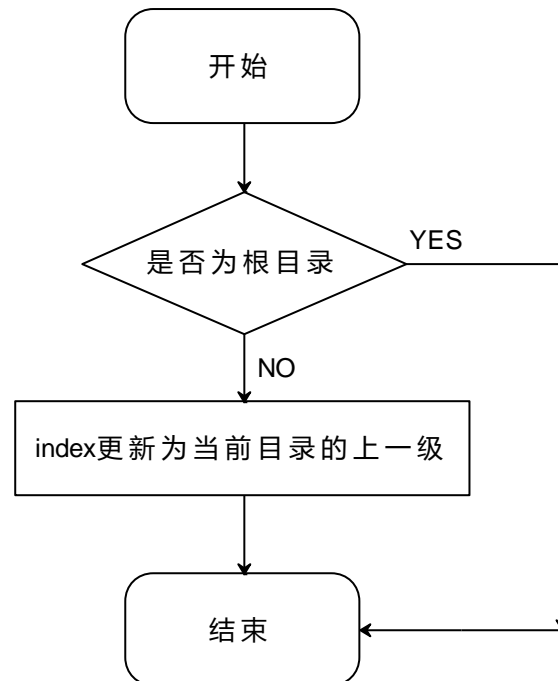
```
typedef struct director
{
    int id;                // 目录索引号
    string name;           // 目录名
    list<int> file_list;    // 本目录中的文件链表
    list<int> director_list; // 本目录中的文件夹链表
    string owner;          // 文件夹拥有者
    int last_director;     // 上一级目录，根目录的上一级设为-1
}director;

typedef struct fileSystem //文件系统
{
    vector<dataBlock> dataArea; // 数据域
    stack<int> superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链数组，被使用了置成-1
    //从free_list[0][127]——freelist[0][0]——free_list[1][127]——free_list[3][0]
    vector<user> user_info; // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file; // 存放所有文件信息
}fileSystem;

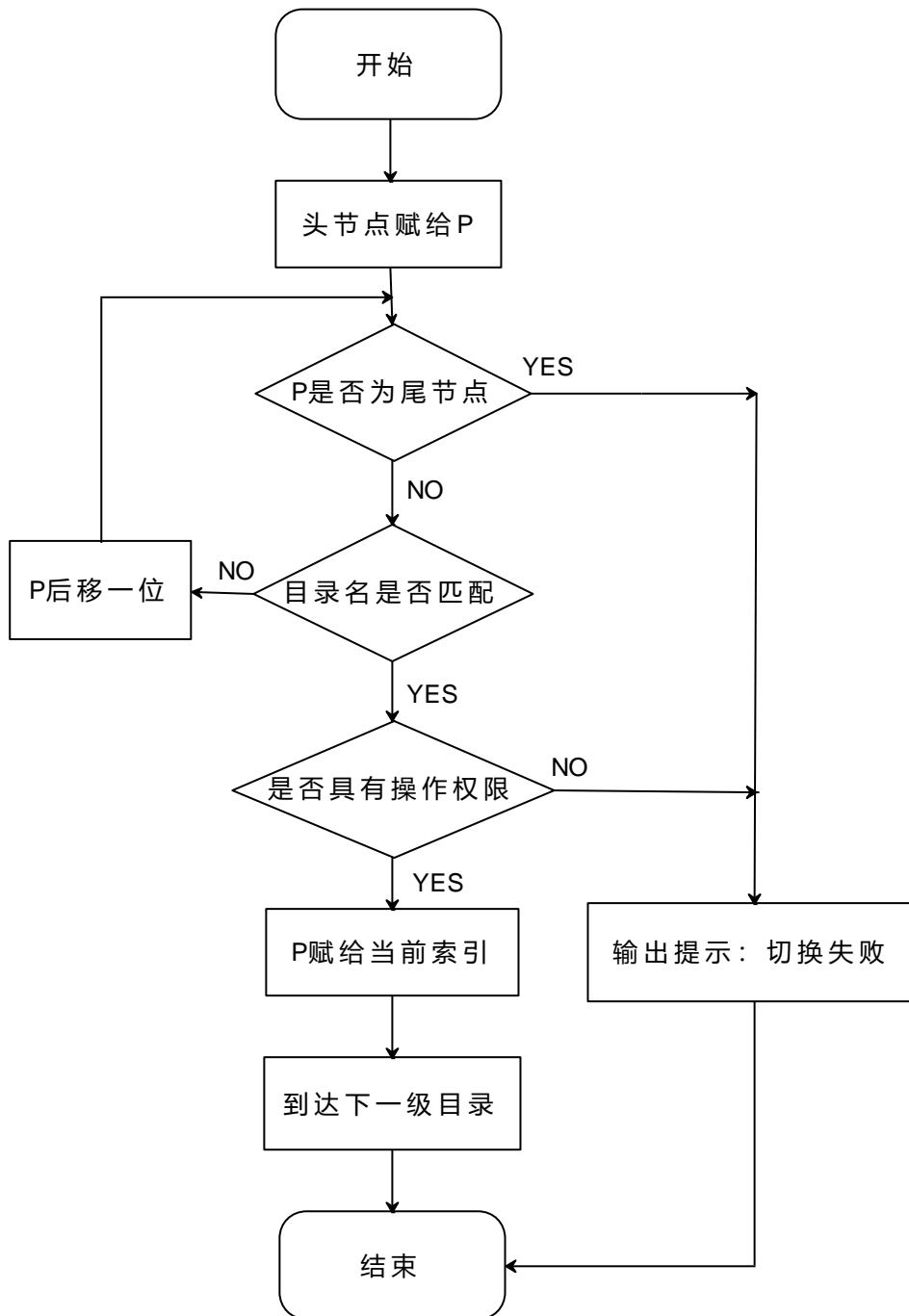
typedef struct user
{
    string name;
    string password;
}user;
```

3.19.3 流程图

1.返回上一级



2.进入下一级



3.20 创建目录模块

3.20.1 功能

提供创建文件夹的功能。判断是否存在同名文件或文件夹，若存在则创建失败，若不存在则创建成功。最后修改相应目录链表的信息。

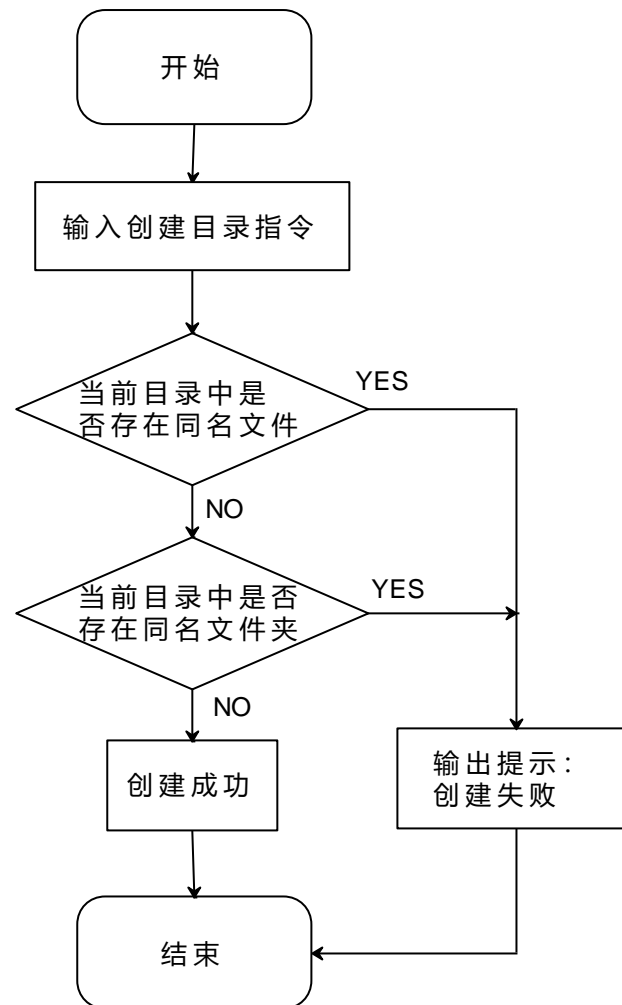
3.20.2 数据结构

```
typedef struct director
{
    int id;                // 目录索引号
    string name;           // 目录名
    list<int> file_list;    // 本目录中的文件链表
    list<int> director_list; // 本目录中的文件夹链表
    string owner;          // 文件夹拥有者
    int last_director;     // 上一级目录          // 根目录的上一级 设为 -1
}director;

typedef struct fileSystem //文件系统
{
    vector<dataBlock> dataArea; // 数据域
    stack<int> superStack; // 超级栈
    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链
    //从
    free_list[0][127]--freelist[0][0]--free_list[1][127]--free_list[3][0]
    vector<user> user_info; // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file; // 存放所有文件信息
}fileSystem;

typedef struct file
{
    int id;                // 文件索引号
    string file_name;      // 文件名
    int beginning_in_dataArea; // 在数据域中的第一个位置
    string owner;          // 文件所有者的名字
    int file_length;       // 文件所占数据块块数
    int beginning_in_memory; // 在内存中的第一个位置
}file;
```

3.20.3 流程图



3.21 用户注册模块

3.21.1 功能

提供用户注册功能。用户可创建新的用户账号，输入新的用户名和密码，进入文件管理系统。

3.21.2 数据结构

```
typedef struct fileSystem    //文件系统
{
    vector<dataBlock> dataArea;    // 数据域
    stack<int> superStack;    // 超级栈
}
```

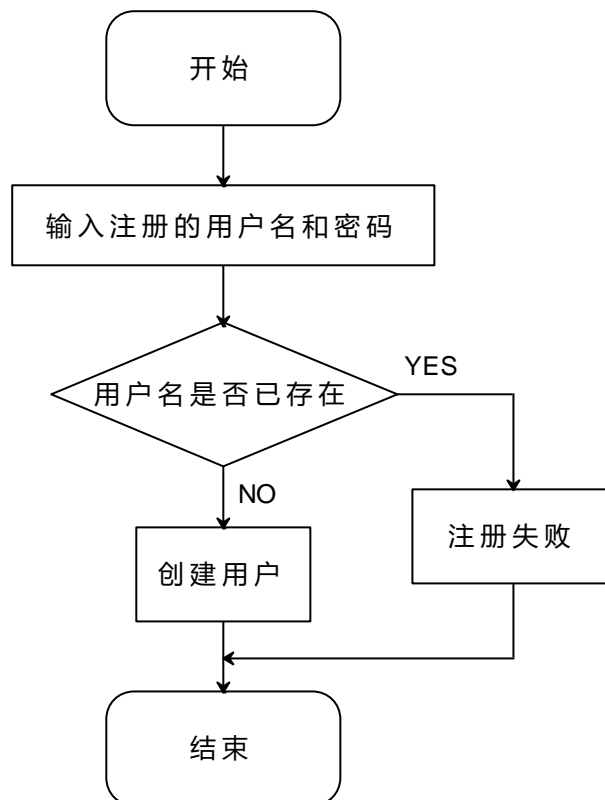
```

    int free_list[INDEX_LIST_NUMBER][INDEX_LIST_LENGTH]; // 成组链
    数组，被使用了置成-1
    //从
    free_list[0][127]--freelist[0][0]--free_list[1][127]--free_list[3][0]
    vector<user> user_info; // 用户信息
    vector<director> vector_director; // 存放所有目录信息
    vector<file> vector_file; // 存放所有文件信息
}fileSystem;
typedef struct user
{
    string name;
    string password;
}user;

extern bool SignIn(); //注册模块

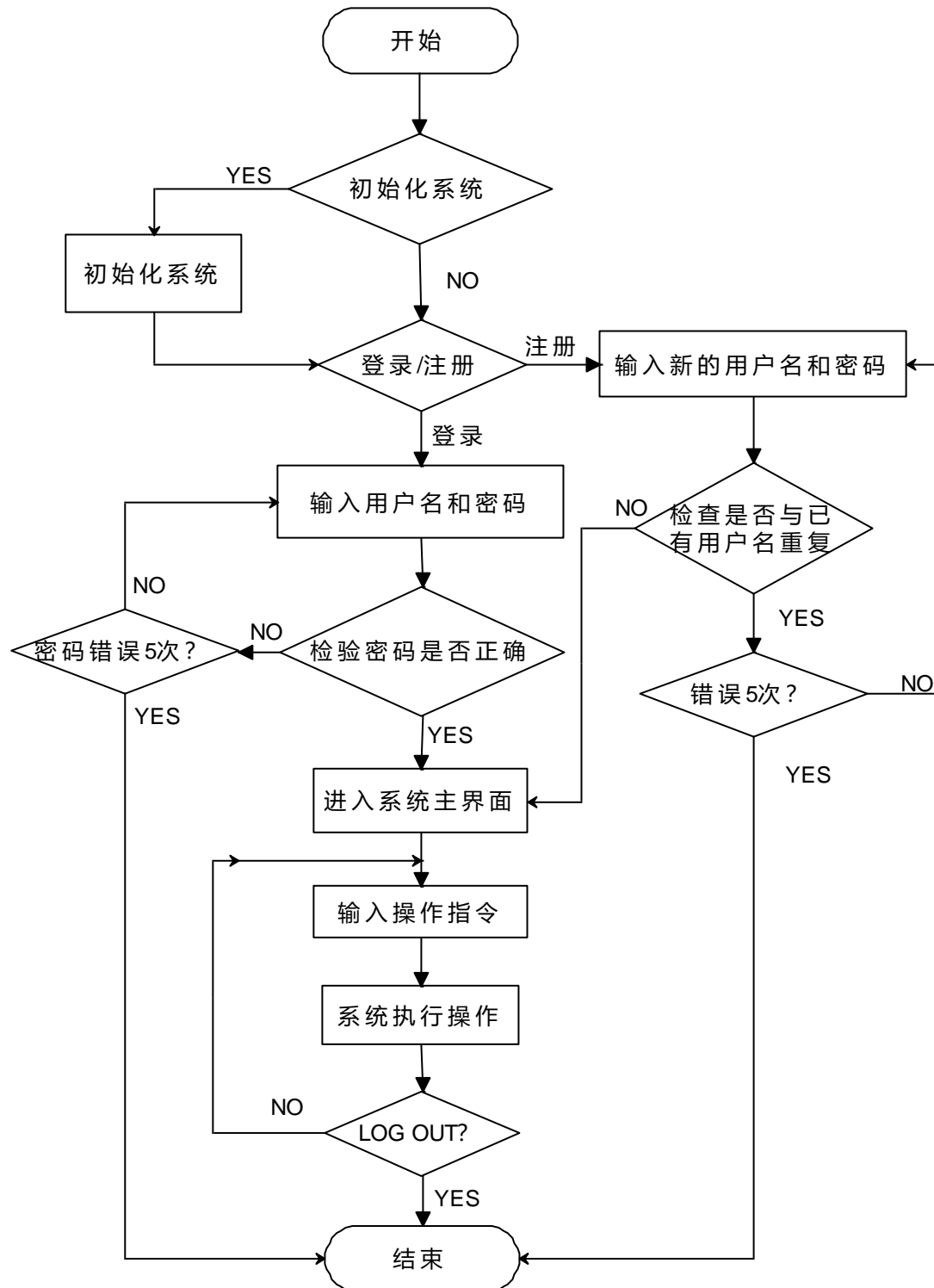
```

3.21.3 流程图



4. 程序设计与实现

4.1 程序流程图



4.2 程序说明

程序实现模拟 **UNIX** 系统的文件管理功能。具体操作流程如下。

(1)初始化界面。开始运行时，界面询问用户是否初始化本系统，若用户选择初始化本系统，则以前保存的系统操作的结果将被删除。

(2)选择注册或登录界面。用户根据需要选择登录或注册。

(3)若选择注册，则用户可创建新的账号，输入用户名和密码即可。该用户名不能与系统中已存在的用户名相同。注册成功后，进入系统主界面。

(4)若选择登录，用户需要输入已注册的用户名和密码。初始化后系统支持 **user1-user8**，密码为相应的用户名。如果密码连续输入错误 **5** 次，则自动退出系统。登录成功后，进入系统主界面。

(5) 系统主界面。进入系统后，系统会以菜单列表方式给出所有功能选择，用户根据自己的需要，输入相应的指令，系统将根据指令执行相应操作。

(6)执行完一次文件管理操作后，会等待用户输入下次要执行的操作，直到用户想要退出系统。

(7)输入退出系统指令后，系统自动将本次登陆后的操作结果保存，以便下次登录使用。

4.3 实验结果

1.初始化系统

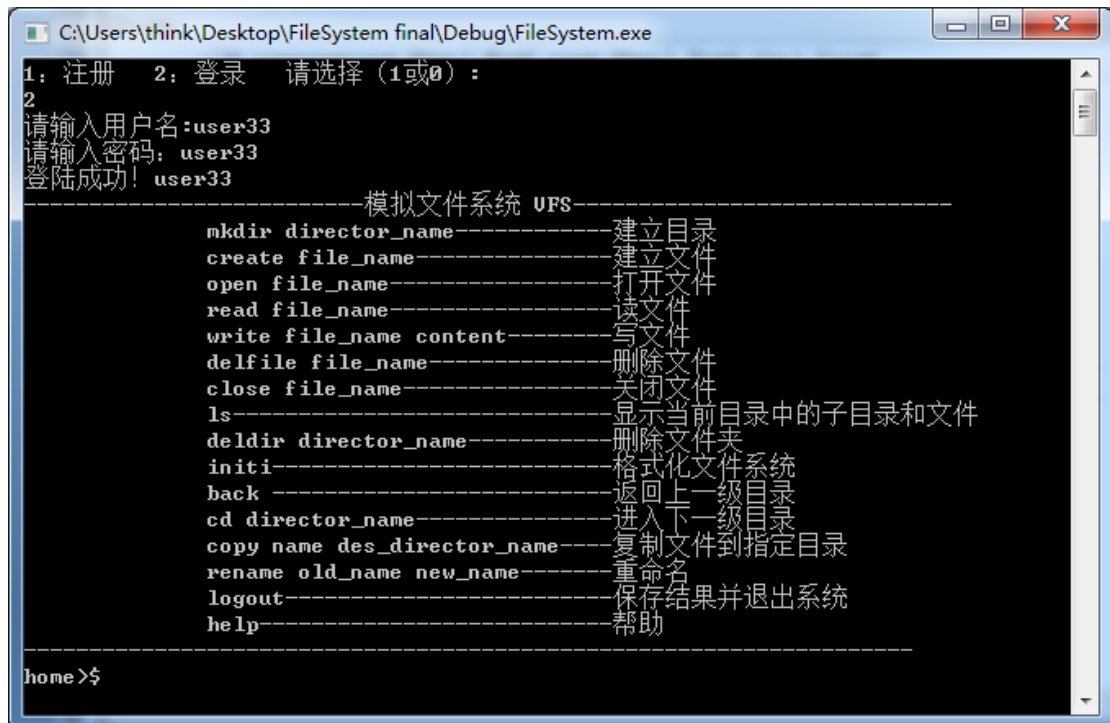


2.注册账号

注册时:



注册后登录:



3. 登录系统

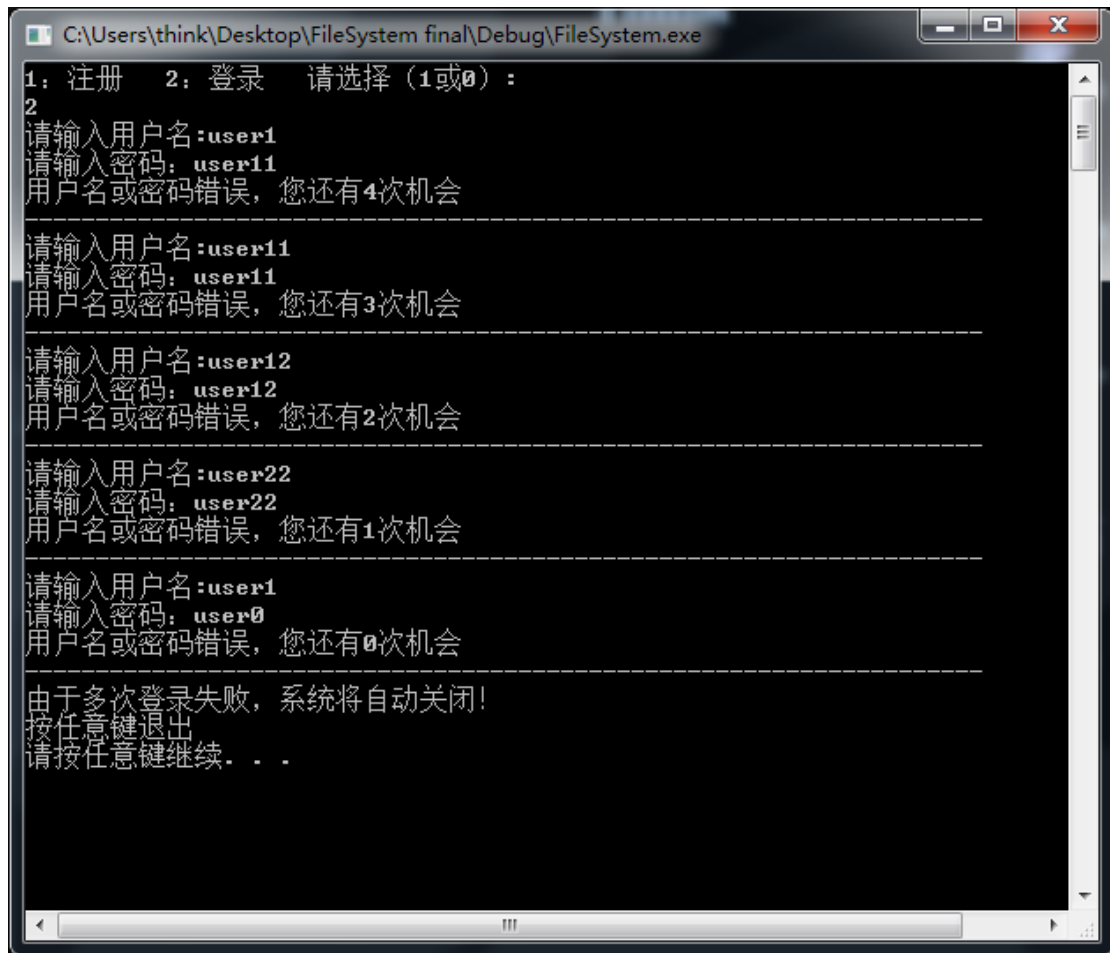
用户名和密码匹配成功则登录成功，显示菜单界面。可以模拟命令行方式输

入操作命令。连续 5 次输入错误则退出系统。

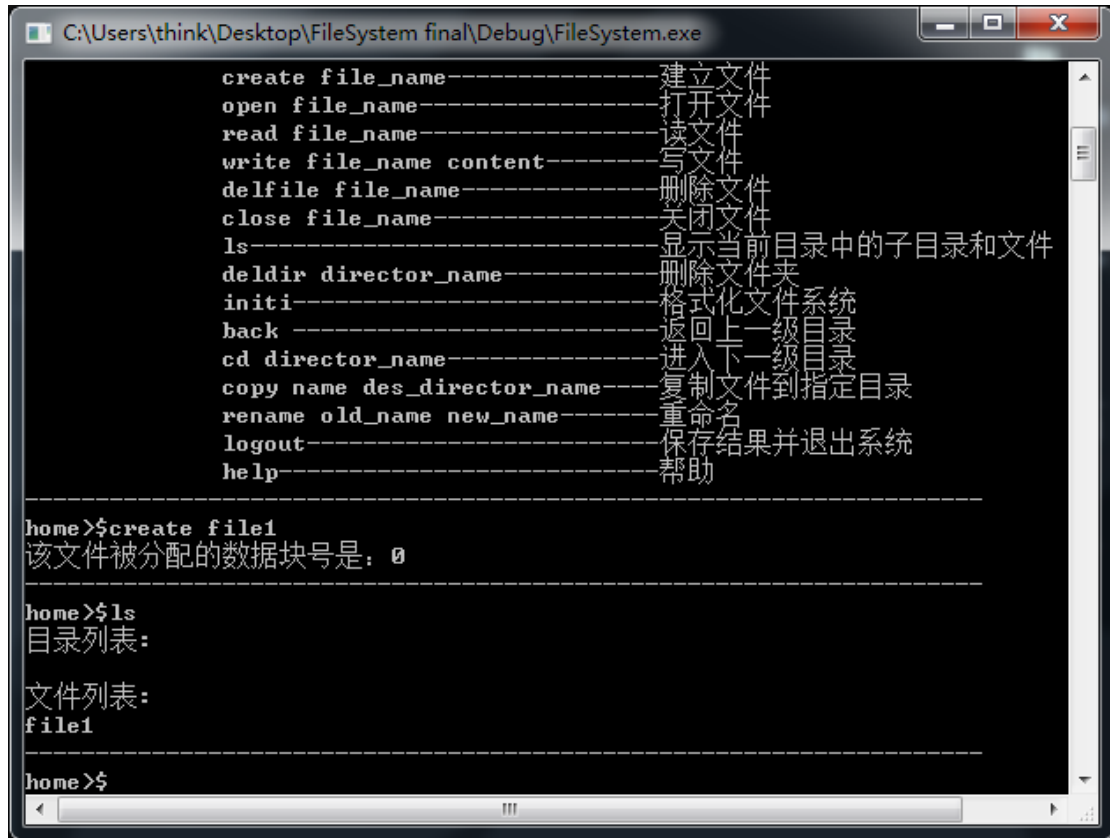


The screenshot shows a Windows command prompt window titled "C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe". The program is running and displays the following text:

```
1: 注册 2: 登录 请选择 (1或0) :  
2  
请输入用户名:user1  
请输入密码: user1  
登陆成功! user1  
-----模拟文件系统 UFS-----  
mkdir director_name-----建立目录  
create file_name-----建立文件  
open file_name-----打开文件  
read file_name-----读文件  
write file_name content-----写文件  
delfile file_name-----删除文件  
close file_name-----关闭文件  
ls-----显示当前目录中的子目录和文件  
deldir director_name-----删除文件夹  
init-----格式化文件系统  
back-----返回上一级目录  
cd director_name-----进入下一级目录  
copy name des_director_name-----复制文件到指定目录  
rename old_name new_name-----重命名  
logout-----保存结果并退出系统  
help-----帮助  
-----  
home>$
```



4. 创建文件(create file1)



```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe

create file_name-----建立文件
open file_name-----打开文件
read file_name-----读文件
write file_name content-----写文件
delfile file_name-----删除文件
close file_name-----关闭文件
ls-----显示当前目录中的子目录和文件
deldir director_name-----删除文件夹
initi-----格式化文件系统
back -----返回上一级目录
cd director_name-----进入下一级目录
copy name des_director_name---复制文件到指定目录
rename old_name new_name-----重命名
logout-----保存结果并退出系统
help-----帮助

home>$create file1
该文件被分配的数据块号是: 0

home>$ls
目录列表:

文件列表:
file1

home>$
```

5. 在文件中写入内容(write file1 virtual_file_system), 打开文件(open file1), 读取文件(read file1), 关闭文件(close file1), 删除文件(delfile file1)

```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe
home>$ls
目录列表:
文件列表:
file1

home>$write file1 virtual_file_system
写入成功! 已在0号数据块写入19字节

home>$open file1
内存读入:virtual_file_system
打开文件file1成功!
共读入内存19个字节

home>$read file1
virtual_file_system
共读入19个字节

home>$close file1
关闭文件file1成功

home>$delfile file1
释放了0号数据块
删除文件file1成功

home>$
```

6. 创建文件夹(mkdir dir2)

```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe
home>$close file1
关闭文件file1成功

home>$delfile file1
释放了0号数据块
删除文件file1成功

home>$mkdir dir1
创建目录dir1成功

home>$ls
目录列表:
dir1
文件列表:

home>$mkdir dir2
创建目录dir2成功

home>$ls
目录列表:
dir1    dir2
文件列表:

home>$
```

7. 进入下一级目录(cd dir1)、返回上一级目录(back)、显示目录(ls)

当在根目录时不能再返回上一级目录。

```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe
home>$mkdir dir2
创建目录dir2成功

home>$ls
目录列表:
dir1    dir2
文件列表:

home>$cd dir1

home>dir1>$create file2
该文件被分配的数据块号是: 0

home>dir1>$back

home>$dir
错误指令, 请重新输入!

home>$ls
目录列表:
dir1    dir2
文件列表:

home>$
```

```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe

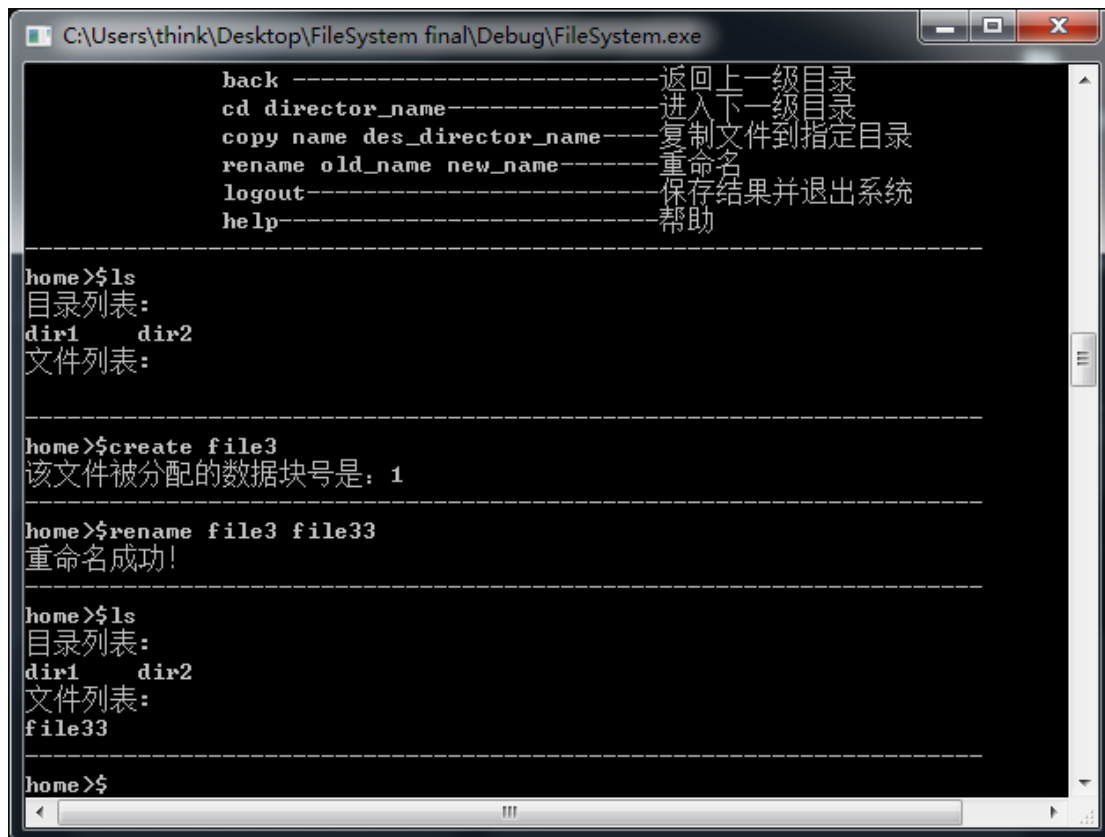
open file_name-----打开文件
read file_name-----读文件
write file_name content-----写文件
delfile file_name-----删除文件
close file_name-----关闭文件
ls-----显示当前目录中的子目录和文件
deldir director_name-----删除文件夹
init-----格式化文件系统
back-----返回上一级目录
cd director_name-----进入下一级目录
copy name des_director_name-----复制文件到指定目录
rename old_name new_name-----重命名
logout-----保存结果并退出系统
help-----帮助

home>$ls
目录列表:
dir1
文件列表:
file33

home>$back
您已经在根目录中, 不能再向上

home>$
```


8. 文件重命名(rename file3 file33)



```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe

back ----- 返回上一级目录
cd director_name ----- 进入下一级目录
copy name des_director_name ----- 复制文件到指定目录
rename old_name new_name ----- 重命名
logout ----- 保存结果并退出系统
help ----- 帮助

-----
home>$ls
目录列表:
dir1    dir2
文件列表:

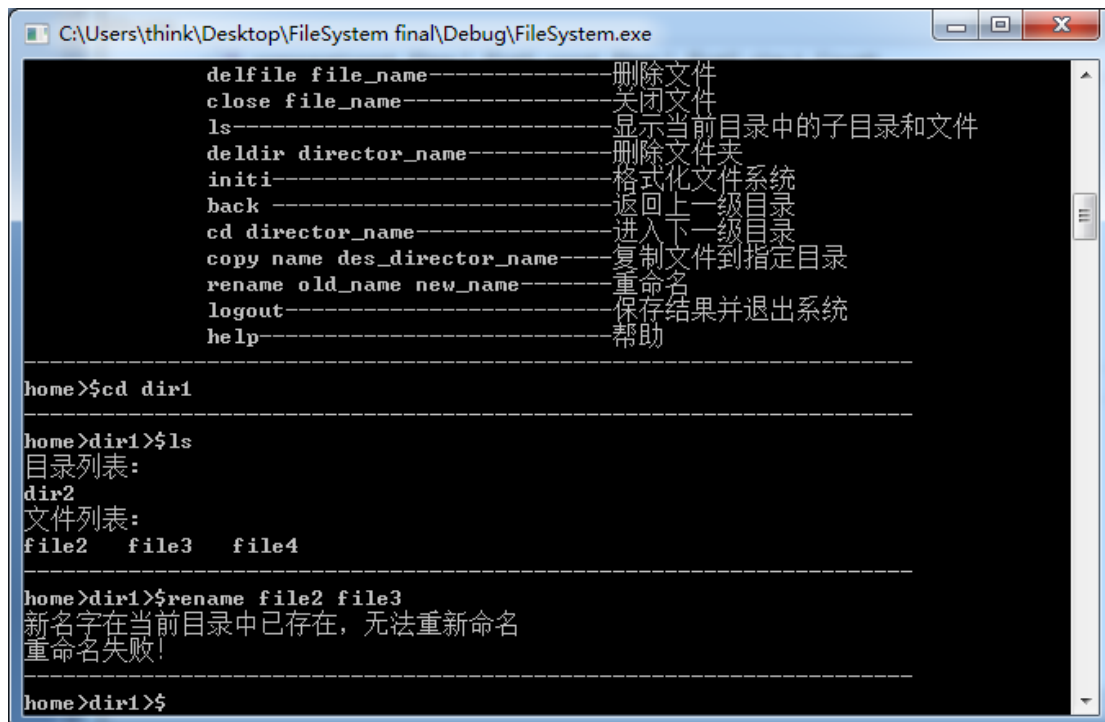
-----
home>$create file3
该文件被分配的数据块号是: 1

-----
home>$rename file3 file33
重命名成功!

-----
home>$ls
目录列表:
dir1    dir2
文件列表:
file33

-----
home>$
```

重命名失败时:



```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe

delfile file_name ----- 删除文件
close file_name ----- 关闭文件
ls ----- 显示当前目录中的子目录和文件
deldir director_name ----- 删除文件夹
initi ----- 格式化文件系统
back ----- 返回上一级目录
cd director_name ----- 进入下一级目录
copy name des_director_name ----- 复制文件到指定目录
rename old_name new_name ----- 重命名
logout ----- 保存结果并退出系统
help ----- 帮助

-----
home>$cd dir1

-----
home>dir1>$ls
目录列表:
dir2
文件列表:
file2   file3   file4

-----
home>dir1>$rename file2 file3
新名字在当前目录中已存在, 无法重新命名
重命名失败!

-----
home>dir1>$
```

9. 复制文件(copy file4 home\dir1)

```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe

home>dir2>$create file4
该文件被分配的数据块号是: 2

home>dir2>$write file4 where_is_bug
写入成功! 已在2号数据块写入12字节

home>dir2>$copy file4 home\dir1
跳转到home\dir1成功!
where_is_bug
3
写入成功! 已在3号数据块写入12字节
复制成功!

home>dir2>$back

home>$cd dir1

home>dir1>$ls
目录列表:

文件列表:
file2  file4

home>dir1>$open file4
内存读入:where_is_bug
打开文件file4成功!
共读入内存12个字节

home>dir1>$
```

打开文件可以看到拷贝的文件内容。

删除复制的文件 **dir1\file4**, 不会影响原路径中的文件 **dir2\file4**, 如下图所示。

```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe
目录列表:
文件列表:
file2  file4
-----
home>dir1>$open file4
内存读入:where_is_bug
打开文件file4成功!
共读入内存12个字节
-----
home>dir1>$delfile file4
释放了3号数据块
删除文件file4成功
-----
home>dir1>$back
-----
home>$cd dir2
-----
home>dir2>$ls
目录列表:
文件列表:
file4
-----
home>dir2>$open file4
内存读入:where_is_bug
打开文件file4成功!
共读入内存12个字节
-----
home>dir2>$
```

10. 复制文件夹(copy dir2 home\dir1)

```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe
file4
-----
home>dir2>$open file4
内存读入:where_is_bug
打开文件file4成功!
共读入内存12个字节
-----
home>dir2>$back
-----
home>$copy dir2 home\dir1
跳转到home\dir1成功!
复制成功!
-----
home>$cd dir1
-----
home>dir1>$ls
目录列表:
dir2
文件列表:
file2
-----
home>dir1>$cd dir2
-----
home>dir1>dir2>$ls
目录列表:
文件列表:
file4
-----
home>dir1>dir2>$
```

11.删除文件夹

删除非空文件夹(deldir dir2):

```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe
home>dir1>$back

home>$ls
目录列表:
dir1    dir2
文件列表:
file33

home>$deldir dir2
要删除的目录非空，是否删除其中所有文件及文件夹?Y/N
Y
释放了2号数据块
删除文件file4成功
删除文件夹dir2成功

home>$ls
目录列表:
dir1
文件列表:
file33

home>$cd dir1

home>dir1>$ls
目录列表:
dir2
文件列表:
file2

home>dir1>$
```

上图中可看出，拷贝在 `home\dir1` 中的 `dir2` 不受影响。

删除空文件夹(`deldir dir4`):

```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe

home>$cd dir1

home>dir1>$ls
目录列表:
dir2
文件列表:
file2

home>dir1>$back

home>$mkdir dir4
创建目录dir4成功

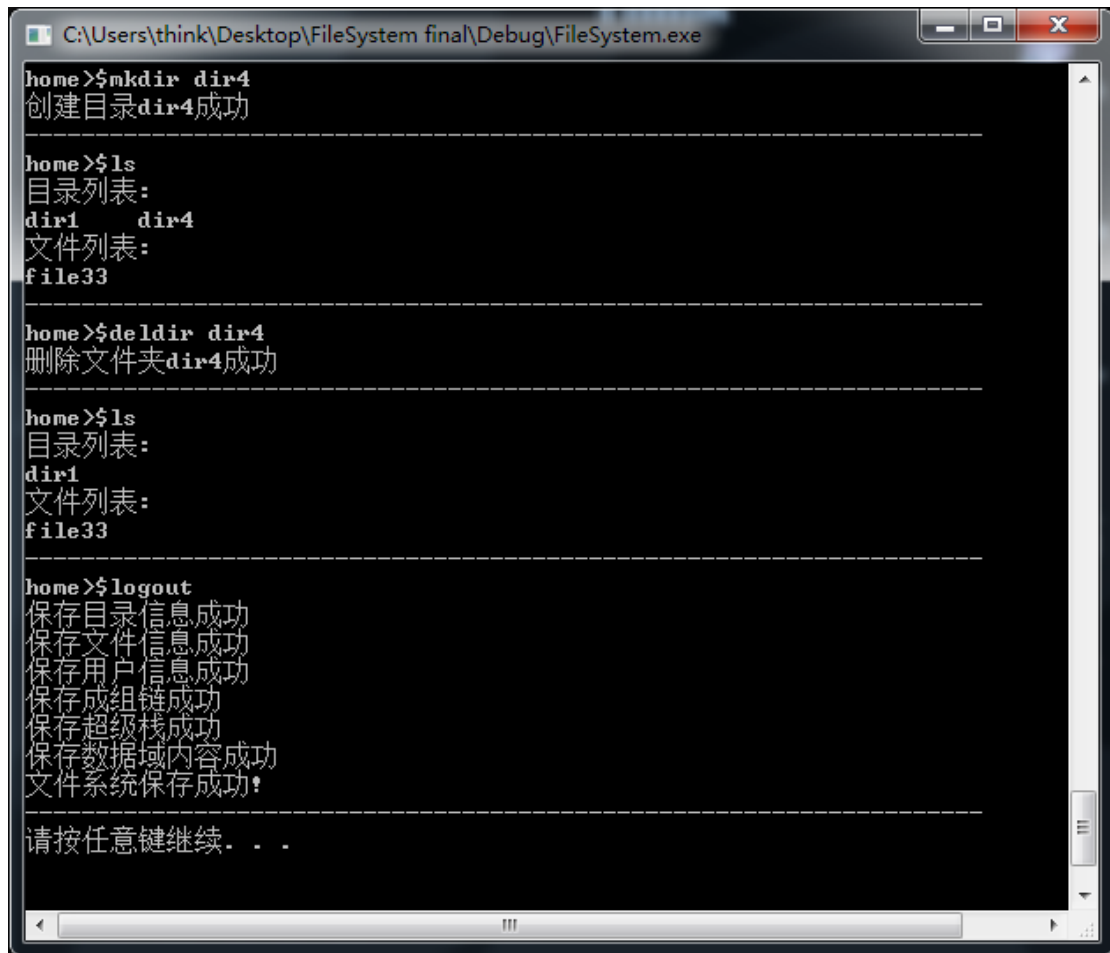
home>$ls
目录列表:
dir1    dir4
文件列表:
file33

home>$rmdir dir4
删除文件夹dir4成功

home>$ls
目录列表:
dir1
文件列表:
file33

home>$
```

12.退出系统(loginout)



```
C:\Users\think\Desktop\FileSystem final\Debug\FileSystem.exe

home>$mkdir dir4
创建目录dir4成功

-----

home>$ls
目录列表:
dir1    dir4
文件列表:
file33

-----

home>$rmdir dir4
删除文件夹dir4成功

-----

home>$ls
目录列表:
dir1
文件列表:
file33

-----

home>$logout
保存目录信息成功
保存文件信息成功
保存用户信息成功
保存成组链成功
保存超级栈成功
保存数据域内容成功
文件系统保存成功!

-----

请按任意键继续. . .
```

5. 参考文献

1. 汤小丹等, 计算机操作系统 (第 4 版), 西安电子科技大学出版社.
2. 张尧学编著. 计算机操作系统教程, 清华大学出版社.2000.
3. 李彤等, 操作系统分析与设计, 云南大学出版社
4. 张琨藏, 操作系统原理 DOS 篇, 清华大学出版社
5. 陈葆玉译, UNIX 操作系统设计, 北京大学出版社

6. 收获、体会和建议