

北京交通大学操作系统实验报告

姓名：程维森

学号：21231264

# 实验内容

*1.1 实验简介*

*本实验要求在假设的 I/O 系统之上开发一个简单的文件系统，这样做既能让实验者对文*

*件系统有整体了解，又避免了涉及过多细节。用户通过 create, open, read 等命令与文件*

*系统交互。文件系统把磁盘视为顺序编号的逻辑块序列，逻辑块的编号为 0 至 L-1。I/O 系*

*统利用内存中的数组模拟磁盘。*

*1.2 I/O 系统*

*实际物理磁盘的结构是多维的：有柱面、磁道、扇区等概念。I/O 系统的任务是隐藏磁*

*盘的结构细节，把磁盘以逻辑块的形式呈现给文件系统。逻辑块顺序编号，编号取值范围为*

*0 至 L-1，其中 L 表示磁盘的存储块总数。实验中，可用类似于字符数组 ldisk[L][B]构建*

*磁盘模型，其中 B 表示每个存储块的长度，一般为 512 字节。I/O 系统从文件系统接收命令，*

*根据命令指定的逻辑块号把磁盘块的内容读入命令指定的内存区域，或者把命令指定的内存*

*区域内容写入磁盘块。文件系统和 I/O 系统之间的接口由如下两个函数定义：*

* int read\_block(int i, char \*p);*

*该函数把逻辑块 i 的内容读入到指针 p 指向的内存位置，拷贝的字符个数为存储块的长度 B，*

*返回实际读取的字节数。*

* int write\_block(int i, char \*p);*

*该函数把指针 p 指向的内容写入逻辑块 i，拷贝的字符个数为存储块的长度 B，返回实际写*

*入的字节数。*

*此外，为方便测试，我们还需要实现另外两个函数：一个用来把数组 ldisk 存储到文件；*

*另一个用来把文件内容恢复到数组。*

*1.3 文件系统*

*文件系统位于 I/O 系统之上。*

*1.3.1 用户与文件系统之间的接口*

*文件系统需提供如下函数：create, destroy, open, read, write。*

* create(filename):根据给定定的文件名创建新文件。*

* destroy(filename):删除指定文件。*

* open(filename):打开文件。该函数返回的索引号可用于后续的 read, write, lseek*

*或 close 操作。*

* close(index):关闭指定的文件。*

* read(index, mem\_area, count):从指定文件顺序读入 count 个字节 mem\_area 指定的内*

*存位置。读操作从文件的读写指针指示的位置开始。*

* write(index, mem\_area, count):把 memarea 指定的内存位置开始的 count 个字节顺序*

*写入指定文件。写操作从文件的读写指针指示的位置开始。*

* lseek(index, pos):把文件的读写指针移动到 pos 指定的位置。pos 是一个整数，表示*

*从文件开始位置的偏移量。文件打开时，读写指针自动设置为 0。每次读写操作之后，*

*它指向最后被访问的字节的下一个位置。lseek 能够在不进行读写操作的情况下改变读*

*写指针能位置。*

* directory:列表显示所有文件及其长度。*

*1.3.2 文件系统的组织*

*磁盘的前 k 个块是保留区，其中包含如下信息：位示图和文件描述符。位示图用来描述*

*磁盘块的分配情况。位示图中每一位对应一个逻辑块。创建或者删除文件，以及文件的长度*

*发生变化时，文件系统都需要进行位示图操作。前 k 个块的剩余部分包含一组文件描述符（思*

*考：可以有多少个文件？）。每个文件描述符包含如下信息：*

* 文件长度，单位字节*

* 文件分配到的磁盘块号数组。该数组的长度是一个系统参数。在实验中我们可以把它设*

*置为一个比较小的数，例如 3（请思考，这个数对文件的大小意味着什么）。*

*1.3.3 目录*

*我们的文件系统中仅设置一个目录，该目录包含文件系统中所有的文件。除了不需要显*

*示地创建和删除之外，目录在很多方面和普通文件相像。目录对应 0 号文件描述符。初始状*

*态下，目录中没有文件，所有，目录对应的描述符中记录的长度应为 0，而且也没有分配磁*

*盘块。每创建一个文件，目录文件的长度便增加一分。目录文件的内容由一系列的目录项组*

*成，其中每个目录项由如下内容组成：*

* 文件名*

* 文件描述符序号*

*1.3.4 文件的创建与删除*

*创建文件时需要进行如下操作；*

* 找一个空闲文件描述符(扫描 ldisk[0]～ldisk[k-1])*

* 在文件目录里为新创建的文件分配一个目录项（可能需要为目录文件分配新的磁盘块）*

* 在分配到的目录项里记录文件名及描述符编号*

* 返回状态信息（如有无错误发生等）*

*删除文件时需要进行如下操作（假设文件没有被打开）*

* 在目录里搜索该文件的描述符编号*

* 删除该文件对应的目录项并更新位示图*

* 释放文件描述符*

* 返回状态信息*

*1.3.5 文件的打开与关闭*

*文件系统维护一张打开文件表，打开文件表的长度固定，其表目包含如下信息：*

* 读写缓冲区*

* 读写指针*

* 文件描述符号*

*文件被打开时，便在打开文件表中为其分配一个表目；文件被关闭时，其对应的表目被*

*释放。读写缓冲区的大小等于一个磁盘存储块。打开文件时需要进行的操作如下：*

* 搜索目录找到文件对应的描述符编号*

* 在打开文件表中分配一个表目*

* 在分配到的表目中把读写指针置为０，并记录描述符编号*

* 读入文件的第一块到读写缓冲区中*

* 返回分配到的表目*

*在打开文件表中的索引号关闭文件时需要进行的操作如下：*

* 把缓冲区的内容写入磁盘*

* 释放该文件在打开文件表中对应的表目*

* 返回状态信息*

*1.3.6 读写*

*文件打开之后才能进行读写操作．读操作需要完成的任务如下：*

*1.计算读写指针对应的位置在读写缓冲区中的偏移*

*2.把缓冲区中的内容拷贝到指定的内存位置，直到发生下列事件之一：*

* 到达文件尾或者已经拷贝了指定的字节数。这时，更新读写指针并返回相应信息*

* 到达缓冲区末尾。这时，把缓冲区内容写入磁盘，然后把文件下一块的内容读入磁盘。*

*最后返回第 2 步。*

*其他操作请同学们自己考虑。*

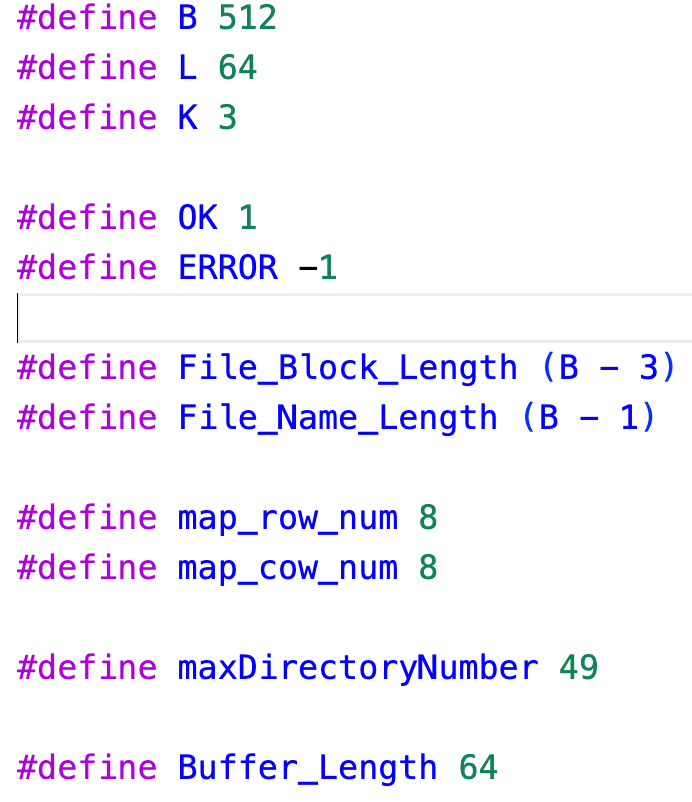
*1.4 测试*

*为了能够对我们的模拟系统进行测试，请编写一个操纵文件系统的外壳程序或者一个菜*

*单驱动系统。*

# 实验变量说明

2.1宏定义说明



#define B 512, #define L 64, #define K 3: 这些宏定义将 B 设置为512，L 设置为64，K 设置为3。它们代表了文件系统或程序中使用的固定常量值或者缓冲区大小。

#define OK 1, #define ERROR -1: 这里定义了两个宏，OK 的值是1，用于表示程序中的正常操作，而 ERROR 的值是-1，通常用于指示错误或异常的情况。

#define File\_Block\_Length (B - 3): 这个宏定义计算了文件块的长度，它是 B（512）减去3的结果。这是为了在文件系统中定义文件块的大小。

#define File\_Name\_Length (B - 1): 这个宏定义计算了文件名的最大长度，它是 B（512）减去1的结果。这个宏限制了文件名的最大长度。

#define maxDirectoryNumber 49: 这个宏定义将 maxDirectoryNumber 设置为49，是表示文件系统中最大的目录数量。

#define Buffer\_Length 64: 这个宏定义设置了缓冲区的长度为64。它用于定义在程序中用作临时存储数据的缓冲区大小

#define map\_row\_num 8, #define map\_cow\_num 8: 这些宏定义设置了地图的行数和列数，分别为8行和8列。这用于某种位示图或矩阵数据结构。

2.2全局变量说明



typedef struct FileDescriptor { ... } FileDescriptor;:

这是一个结构体定义，定义了一个名为 FileDescriptor 的结构体类型，包含了文件描述符的相关信息。

它包括了文件长度 fileLength、文件分配块的编号数组 file\_allocation\_blocknumber[File\_Block\_Length]、文件块长度 file\_block\_length、文件的起始位置 beginpos、读写指针 rwpointer、以及读写缓冲区 RWBuffer。

typedef struct Directory { ... } Directory;:

这是另一个结构体定义，定义了一个名为 Directory 的结构体类型，用于表示文件目录的相关信息。

它包含了索引 index、计数 count、文件名 fileName、是否为文件标志 isFileFlag、是否为打开文件标志 isOpenFlag，以及一个 FileDescriptor 类型的结构体 fileDescriptor。

char ldisk[L][B];:

这是一个二维字符数组，名为 ldisk，用于模拟虚拟磁盘。它具有 L 行和 B 列，用于存储字符数据。

char memory\_area[L \* (B - K)];:

这是一个字符数组，名为 memory\_area，用于存储大小为 L \* (B - K) 的内存块，可能用于模拟某种内存区域。

char mem\_area[L \* (B - K)] = {'\0'};:

这是一个与 memory\_area 类似的字符数组，初始化了大小为 L \* (B - K) 的内存块，并将所有元素初始化为 '\0'，即空字符。

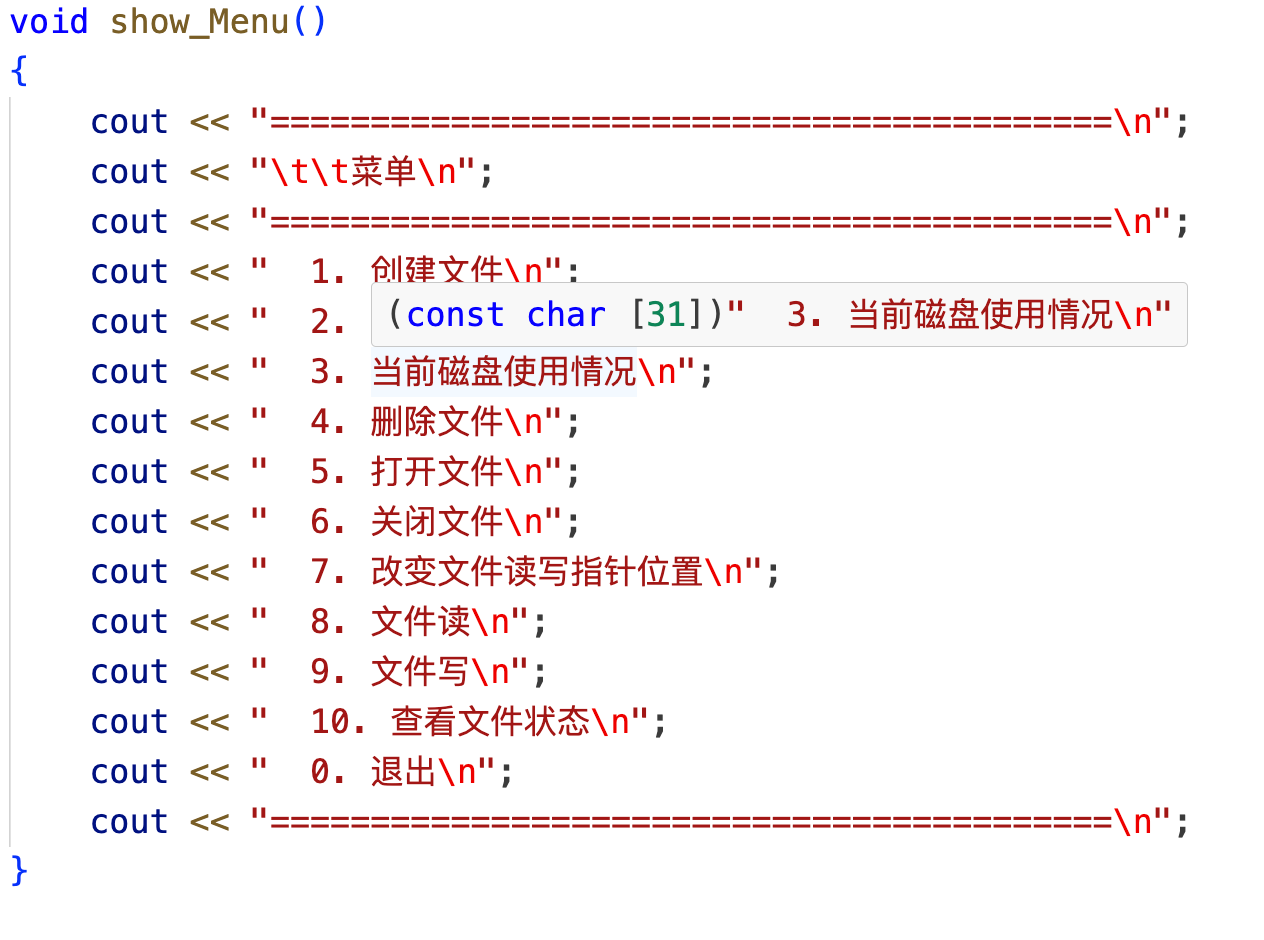
Directory Directorys[maxDirectoryNumber + 1];:

这是一个数组，名为 Directorys，用于存储 maxDirectoryNumber + 1 个 Directory 结构体，即用于存储文件目录的信息。

int bitMap[map\_row\_num][map\_cow\_num];:

这是一个二维整数数组，名为 bitMap，可能用于表示位图。它有 map\_row\_num 行和 map\_cow\_num 列，可能用于跟踪磁盘上的块或某些资源的分配情况。

# 实验函数说明与函数之间的联系



这个函数用来显示操作菜单，并且在每次操作结束后再次显示菜单



这个函数名为 Init()，它主要用于在程序运行开始时对全局变量进行初始化，为程序中使用的数据结构和数组赋予初始值。

函数内部的操作：

memset(ldisk, 0, sizeof(ldisk));:

这行代码使用 memset 函数将 ldisk 数组中的所有元素都初始化为0。ldisk 数组是一个模拟虚拟磁盘的二维字符数组。

for (int i = 0; i <= maxDirectoryNumber; i++) :

这个循环迭代处理 Directorys 数组中的每个元素（maxDirectoryNumber + 1 个元素）。

memset(Directorys[i].fileName, 0, sizeof(Directorys[i].fileName));:

对于每个 Directorys 数组的元素，将其 fileName 字符数组的所有元素都初始化为0。

if (i == 0)

else :

对于 Directorys 数组中的第一个元素（i == 0），将 index、isFileFlag 和 count 设置为0。

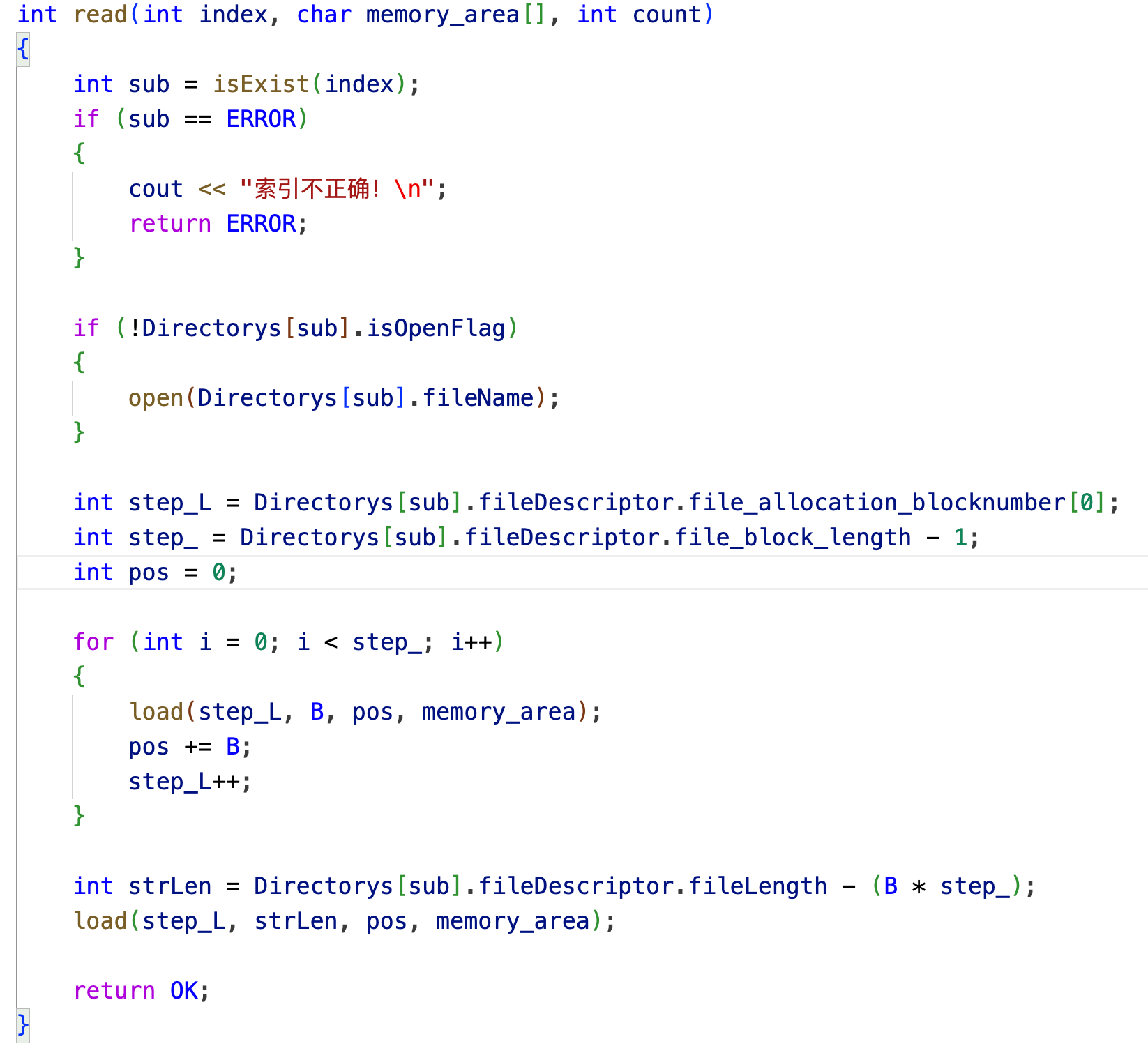
对于其余的 Directorys 数组元素，将 index、isFileFlag 和 count 分别设置为-1、1 和-1，然后对 fileDescriptor 结构体中的相关元素进行初始化。这个部分将所有其他目录项初始化为一个默认的非空目录项。

for (int i = 0; i < map\_row\_num; i++):

这个嵌套的循环遍历 bitMap 数组中的每个元素，并根据条件为 bitMap[i][j] 赋值。

bitMap[i][j] 的值根据 (i \* map\_cow\_num + j < K) ? 1 : 0 条件进行设置，如果当前索引小于 K，则为1，否则为0。

这用于初始化位示图或类似的数组，在模拟磁盘或资源分配时标记某些块或资源的状态（例如，0表示未使用，1表示已使用）。



这是一个名为 read 的函数，它用于读取文件的内容到 memory\_area 中。下面是这个函数的功能解释：

int read(int index, char memory\_area[], int count):

这个函数接受三个参数：index 表示文件索引，memory\_area[] 是用于存储读取数据的缓冲区，count 表示要读取的字符数。

int sub = isExist(index);:

这里调用了一个名为 isExist() 的函数来检查文件索引是否存在。如果 isExist() 返回 ERROR（可能表示索引不存在或不正确），则会输出一条错误信息并返回错误代码 ERROR。

if (!Directorys[sub].isOpenFlag) { open(Directorys[sub].fileName); }:

如果文件未打开（isOpenFlag 为假），则会调用 open() 函数打开该文件，传入文件名 Directorys[sub].fileName。这可能是为了确保文件被打开以便读取。

读取文件内容：

接下来的部分使用循环和 load() 函数从文件中读取数据。

首先，根据文件描述符中存储的信息确定文件的块大小 step\_L 和块数 step\_。

通过循环从磁盘的块中加载数据到 memory\_area 中。循环迭代了除了最后一个块之外的所有块，每次加载一个完整的块大小（B）的数据。

最后一个块的大小可能不足 B，所以计算 strLen 表示最后一块的实际长度，并将其加载到 memory\_area 中。

返回结果：

函数返回 OK，表示读取操作成功完成。

总的来说，read 函数负责根据文件的索引读取数据块并加载到 memory\_area 缓冲区中，然后返回表示读取成功的 OK。如果文件索引不存在或出现其他错误，则输出错误信息并返回 ERROR。



这个代码定义了一个名为 write 的函数，它用于向文件中写入数据。下面是这个函数的功能解释：

int write(int index, char memory\_area[], int count):

这个函数接受三个参数：index 表示文件索引，memory\_area[] 是要写入文件的数据，count 表示要写入的字符数。

int sub = isExist(index);:

这里调用了一个名为 isExist() 的函数来检查文件索引是否存在。如果 isExist() 返回 ERROR（可能表示索引不存在或不正确），则会输出一条错误信息并返回错误代码 ERROR。

if (!Directorys[sub].isOpenFlag) { open(Directorys[sub].fileName); }:

如果文件未打开（isOpenFlag 为假），则会调用 open() 函数打开该文件，传入文件名 Directorys[sub].fileName。这可能是为了确保文件被打开以便写入。

写入文件内容：

函数使用 while 循环，逐个字符从 memory\_area 中取出，并将其写入到文件描述符的 RWBuffer 中，这个缓冲区大小为 Buffer\_Length。

每当 RWBuffer 的大小达到 Buffer\_Length 时，会将其内容存储到磁盘中的相应块中。这个过程使用了 save() 函数。

如果 count 还有剩余字符需要写入，但 RWBuffer 不满，将剩余的字符写入最后一块缓冲区，然后进行存储。

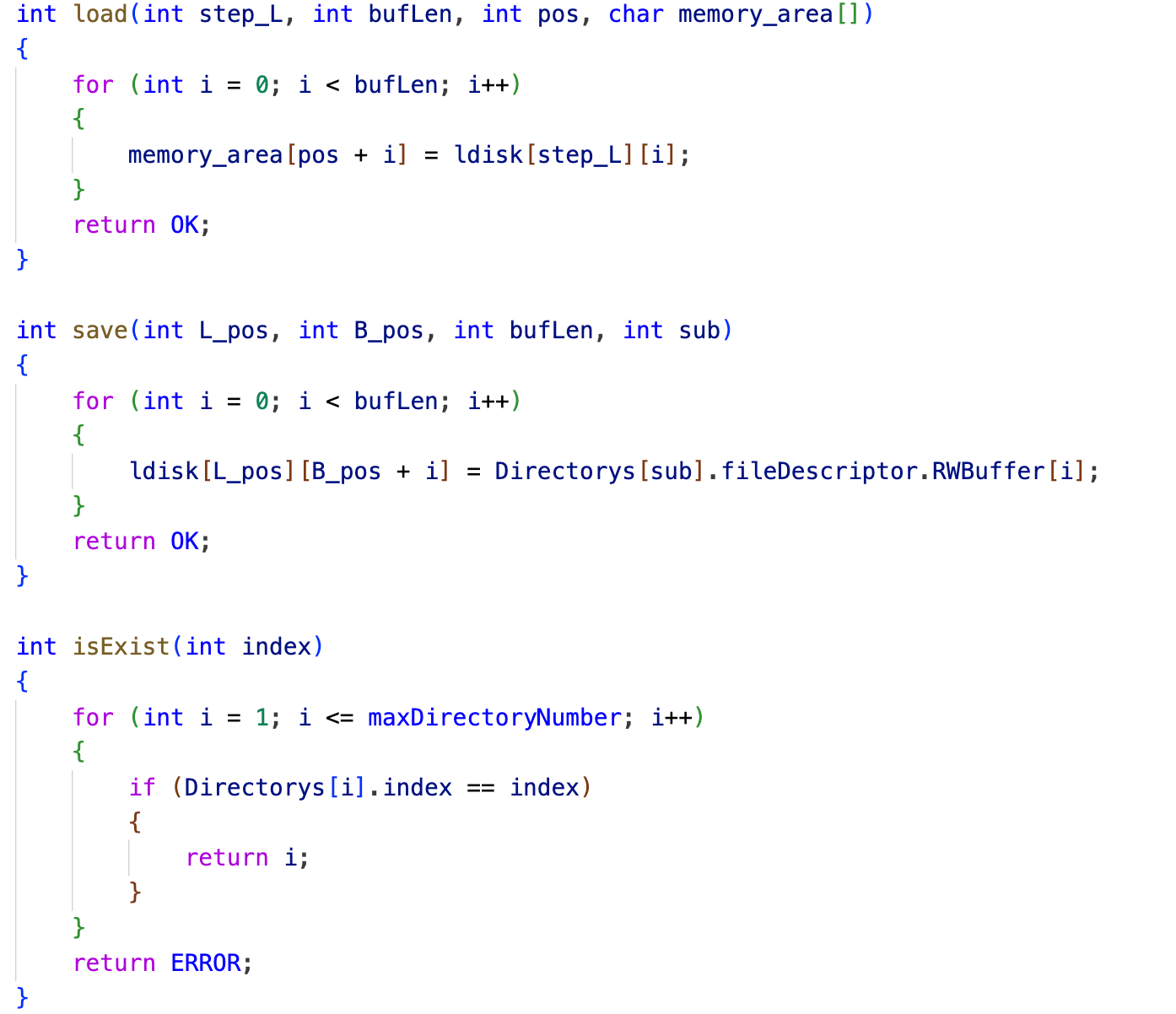
memset(Directorys[sub].fileDescriptor.RWBuffer, '\0', Buffer\_Length);:

最后，将文件描述符中的 RWBuffer 清空，以便下次写入使用。

返回结果：

函数返回 OK，表示写入操作成功完成。

总的来说，write 函数负责将 memory\_area 缓冲区中的数据写入到文件中。它通过 RWBuffer 缓冲区逐步写入到磁盘的相应块中，确保数据的持久性存储，并在写入完成后清空缓冲区。如果文件索引不存在或出现其他错误，则输出错误信息并返回 ERROR。



这些函数是模拟文件系统中读取和存储数据的关键部分。以下是对这些函数的功能解释：

int load(int step\_L, int bufLen, int pos, char memory\_area[])：

这个函数用于从虚拟磁盘 ldisk 中加载数据到 memory\_area 中的指定位置。

参数 step\_L 表示要加载数据的虚拟磁盘块索引，bufLen 表示要加载的数据长度，pos 表示在 memory\_area 中的起始位置，memory\_area[] 是要加载数据的目标缓冲区。

函数通过循环从 ldisk[step\_L] 中复制数据到 memory\_area 中的指定位置，加载指定长度的数据。

int save(int L\_pos, int B\_pos, int bufLen, int sub)：

这个函数用于将数据从文件描述符的缓冲区 RWBuffer 存储到虚拟磁盘 ldisk 的指定位置。

参数 L\_pos 表示虚拟磁盘块的行索引，B\_pos 表示虚拟磁盘块中的起始位置，bufLen 表示要存储的数据长度，sub 表示文件的索引。

函数通过循环从 Directorys[sub].fileDescriptor.RWBuffer 中复制数据到 ldisk[L\_pos] 的指定位置，实现数据的持久化存储。

int isExist(int index)：

这个函数用于检查特定索引的文件是否存在于文件目录中。

参数 index 表示要检查的文件索引。

函数遍历文件目录 Directorys，如果找到指定索引的文件，则返回该文件在目录中的索引位置；如果未找到，则返回 ERROR。

总的来说，load() 函数用于从虚拟磁盘加载数据到内存，save() 函数用于将数据从内存保存到虚拟磁盘，isExist() 函数用于检查文件索引是否存在于文件目录中。这些函数共同模拟了文件系统中读取和存储数据的关键操作。



这段代码定义了一个名为 create 的函数，其功能是创建文件并在模拟的文件系统中进行管理。  
逐步解释这个函数的主要步骤：

检查文件是否已存在：

首先，函数遍历文件目录 Directorys，如果发现有相同文件名的文件已经存在，则输出消息并返回错误码 ERROR。

查找空闲位置创建文件：

接着，函数寻找文件目录中第一个 index 为 -1 的空闲位置（表示空余的目录项），如果找到则跳出循环准备在此位置创建新文件。如果遍历完文件目录都没有找到空闲位置，则输出消息表示磁盘已满，无法创建新文件，并返回错误码 ERROR。

设置文件属性：

在找到空闲位置后，将文件名复制到对应目录项中，并根据文件索引设置文件在目录中的索引位置。

用户被提示输入文件的长度，并根据计算确定文件所需的虚拟磁盘块数量，以及从哪个虚拟磁盘块开始存储文件内容。

进行检查以确认是否有足够的虚拟磁盘块用于存储文件，如果没有足够的连续空间则输出消息表示内存不足，删除刚创建的文件并返回错误码 ERROR。

分配虚拟磁盘块并标记位示图：

如果有足够的空间，将文件所需的虚拟磁盘块分配给该文件，并将相应的位示图设置为已使用（设为1）。

记录文件的相关属性，如文件块长度、起始位置、文件的打开状态等。

创建成功后输出消息提示文件创建成功，并更新文件目录中的文件计数。

总的来说，create 函数负责在模拟文件系统中创建文件，首先进行各种检查以确保文件的有效创建，然后分配虚拟磁盘块并标记位示图，最后更新文件目录。



这段代码定义了一个名为 destroy 的函数，其主要功能是删除文件。下面是对函数的主要步骤解释：

查找文件：

函数首先遍历文件目录 Directorys，查找要删除的文件名是否存在。如果找到了该文件名对应的目录项，记录该文件的索引 sub，并跳出循环。如果遍历完整个目录都没有找到文件名对应的文件，则输出消息表示文件不存在，并返回错误码 ERROR。

检查文件是否打开：

如果要删除的文件处于打开状态（isOpenFlag 为真），则输出消息表示文件处于打开状态，无法删除，并返回错误码 ERROR。

释放虚拟磁盘块并清空文件属性：

如果文件存在且未打开，开始释放该文件占用的虚拟磁盘块，并在位示图 bitMap 中将相应的块标记为未使用（设为 0）。

清空该文件在文件目录中的相关属性，包括文件名、索引、文件块分配情况、文件长度、起始位置、读写缓冲区等。

更新文件计数并输出消息：

删除成功后，更新文件目录中的文件计数，并输出消息提示文件删除成功。

总的来说，destroy 函数负责在模拟文件系统中删除文件。它首先查找要删除的文件名是否存在，然后检查文件是否打开，接着释放文件占用的虚拟磁盘块并清空文件属性，最后更新文件目录中的文件计数。



这些函数是模拟文件系统中处理文件打开、关闭和定位的关键部分。以下是对这些函数的功能解释：

int open(char \*filename)：

此函数用于打开指定文件。

函数首先遍历文件目录 Directorys，寻找指定文件名对应的文件项，如果找到则将该文件的 isOpenFlag 设置为 1（表示打开状态）并返回 OK，否则返回 ERROR。

int lseek(int index, int position)：

此函数用于改变文件指针的位置。

函数首先根据文件的索引号 index 在文件目录中寻找对应的文件项，然后将文件的读写指针 rwpointer 设置为 position，表示将文件指针移动到指定位置。

如果成功找到对应文件项，则返回 OK，否则输出消息表示索引数据错误，返回 ERROR。

int close(int index)：

此函数用于关闭指定文件。

函数根据文件的索引号 index 在文件目录中寻找对应的文件项。如果找到文件且文件为打开状态（isOpenFlag 为 1），则将文件数据从缓冲区写回虚拟磁盘 ldisk 中，并将文件的 isOpenFlag 设置为 0（表示关闭状态）。

如果成功关闭文件，则返回 OK；如果文件已经为关闭状态，输出消息表示文件已关闭，返回 ERROR；如果找不到文件，输出消息表示文件不存在，返回 ERROR。

总的来说，这些函数提供了文件操作的关键功能，包括打开文件、改变文件指针位置以及关闭文件。它们模拟了文件系统中的关键行为，允许对文件进行读写操作并控制文件的打开和关闭状态。



这些函数用于显示文件系统中的信息或特定文件的详细信息：

void directory()：

此函数用于列出当前文件系统中存在的文件以及它们的文件长度。

如果文件目录中的文件数量为零（Directorys[0].count == 0），则输出消息表示当前没有文件。

否则，遍历文件目录，输出每个文件的序号、文件名以及文件长度。

void show\_ldisk()：

此函数用于显示虚拟磁盘 ldisk 中的内容。

遍历虚拟磁盘 ldisk，逐行输出每个块的内容。

int show\_File(char \*filename)：

此函数用于显示特定文件的详细信息。

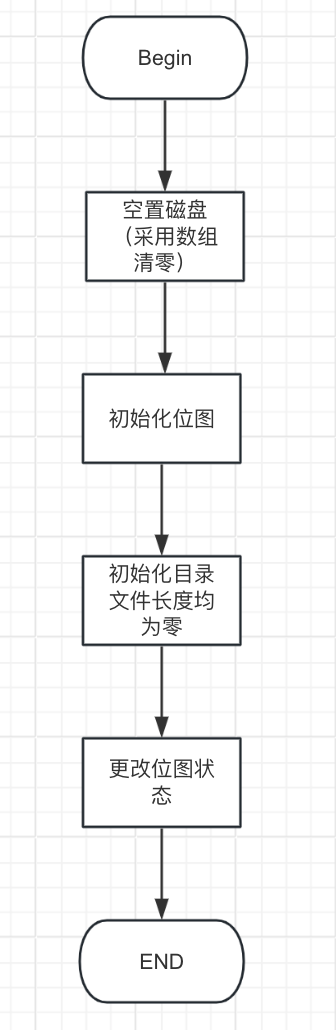
首先遍历文件目录 Directorys，查找指定文件名对应的文件项。如果找到该文件，则输出该文件的文件名、打开状态、索引值、文件长度等详细信息；如果未找到该文件，则输出消息表示未找到文件并返回 ERROR。

这些函数允许用户查看文件系统中存在的文件列表、虚拟磁盘中的内容以及特定文件的详细信息。它们为用户提供了管理和监视文件系统状态的功能。

主函数不做说明，仅为switch case的函数用来显示不同的输出。

# 实验主要函数流程框图

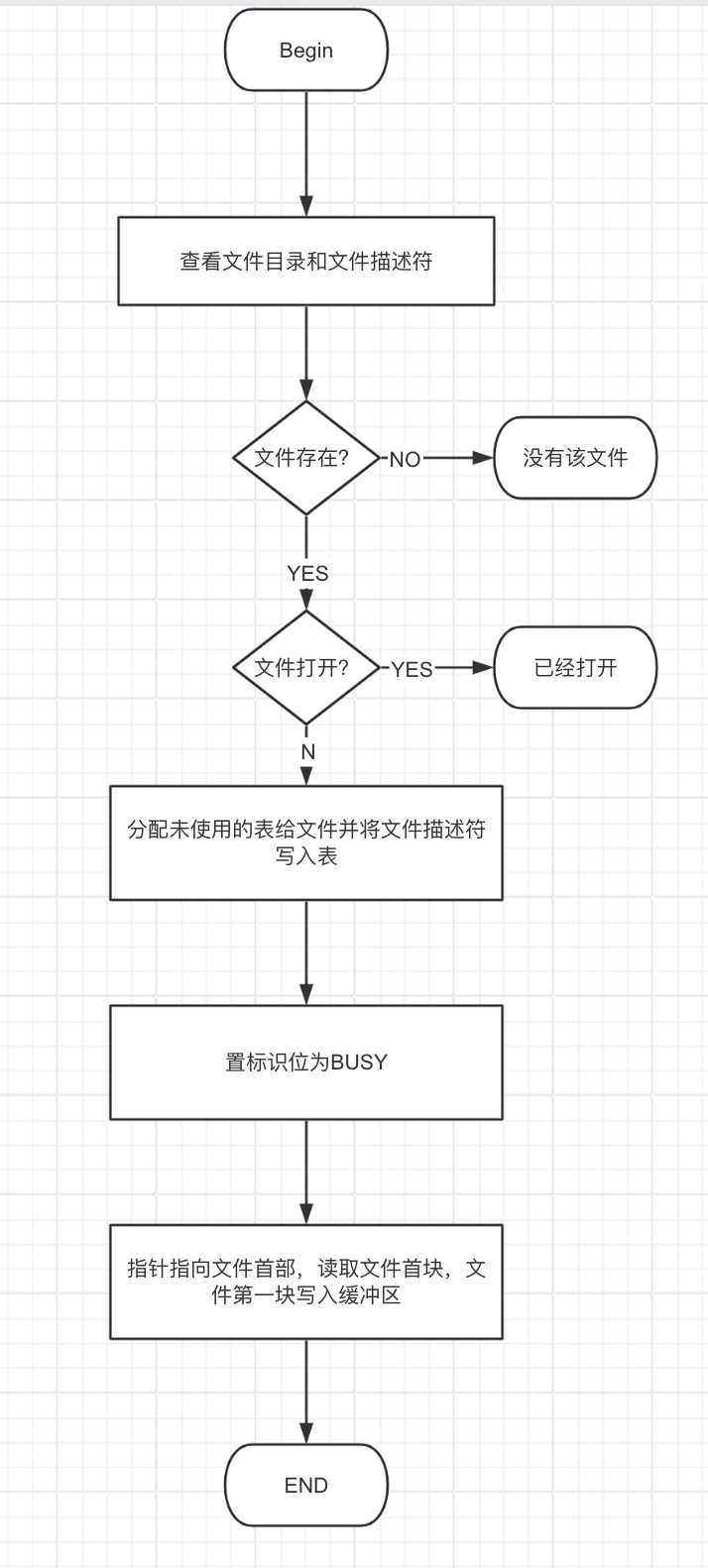
Init():



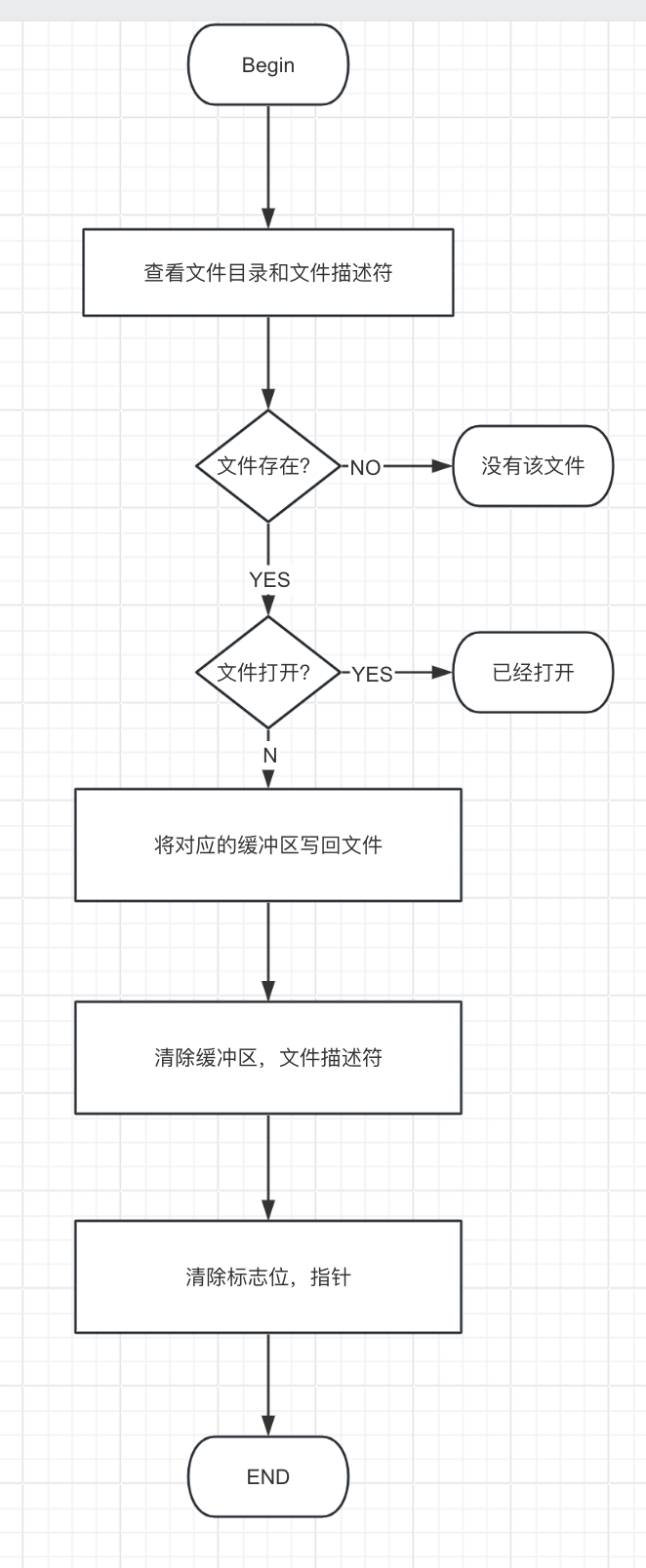
create(char filename[])：



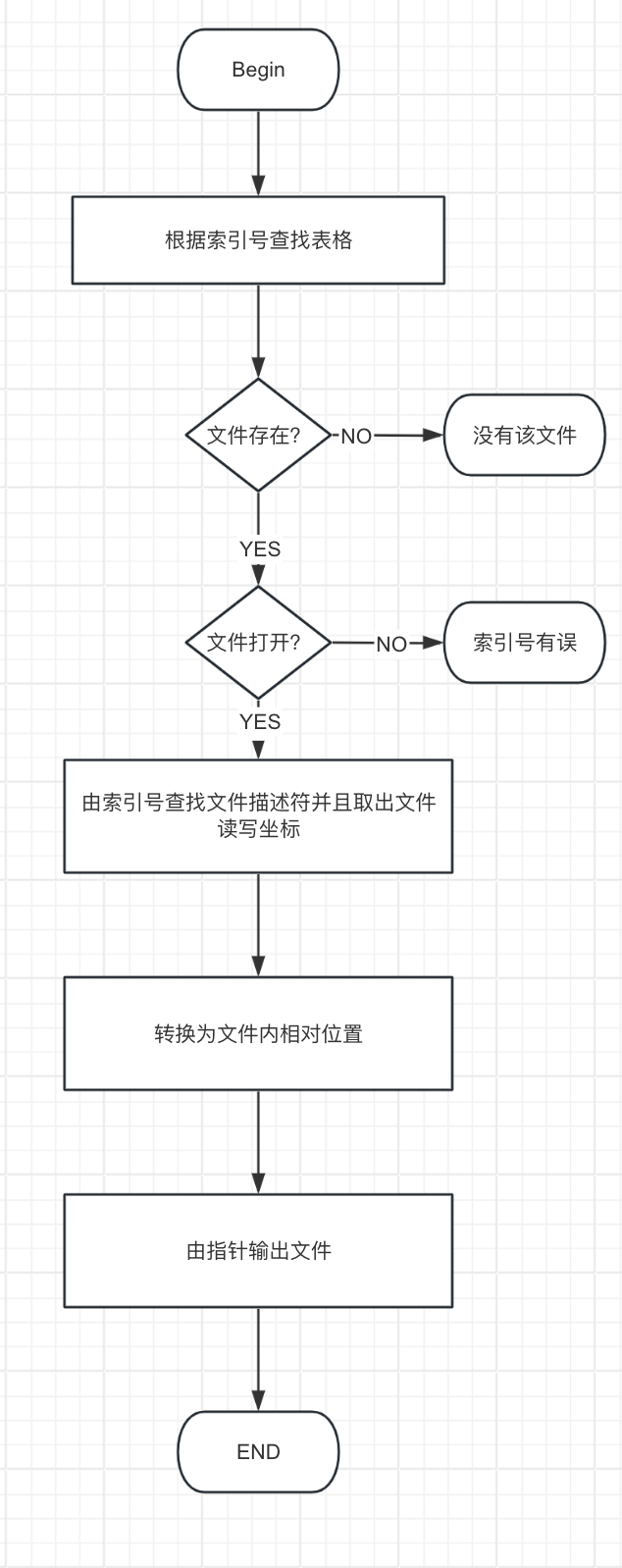
 open(char\* filename)：



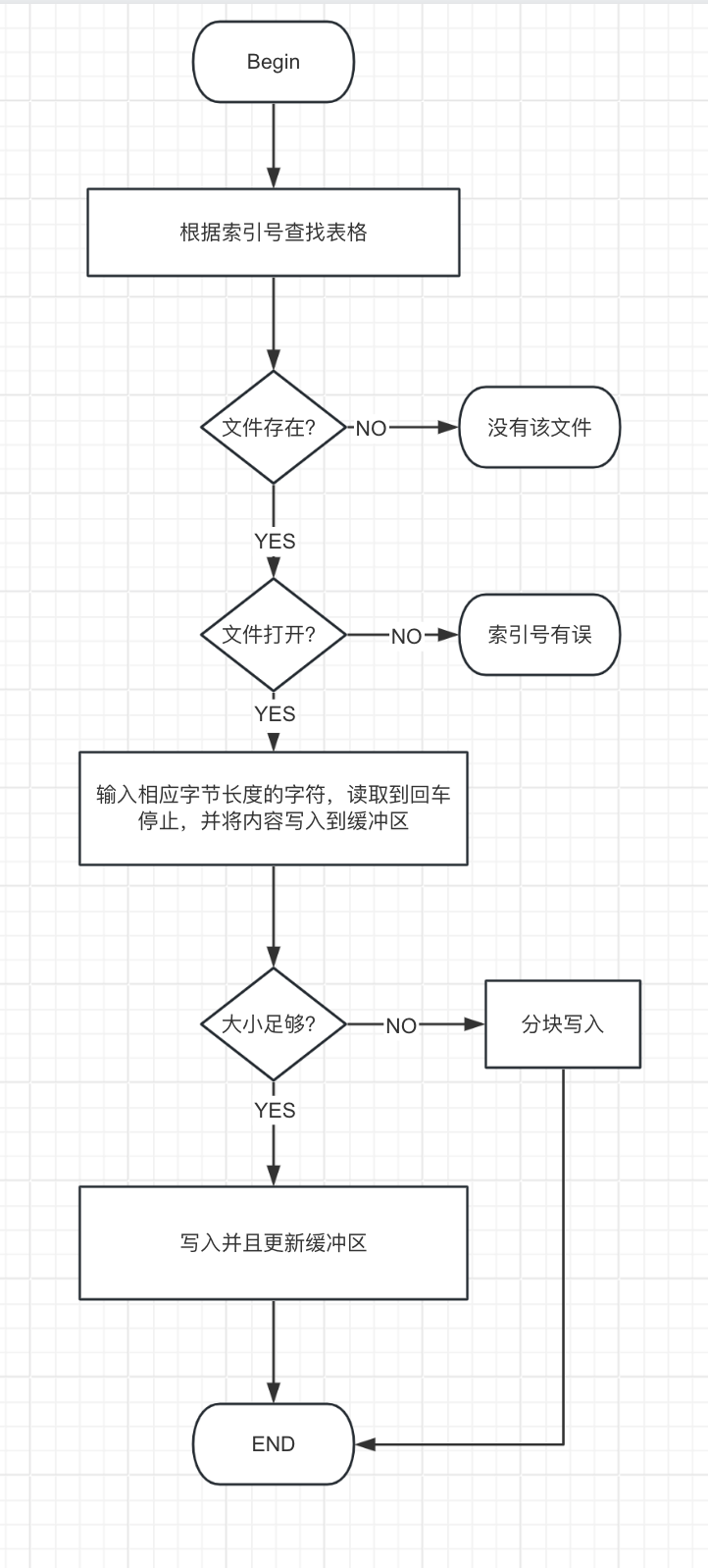
close(int index)：



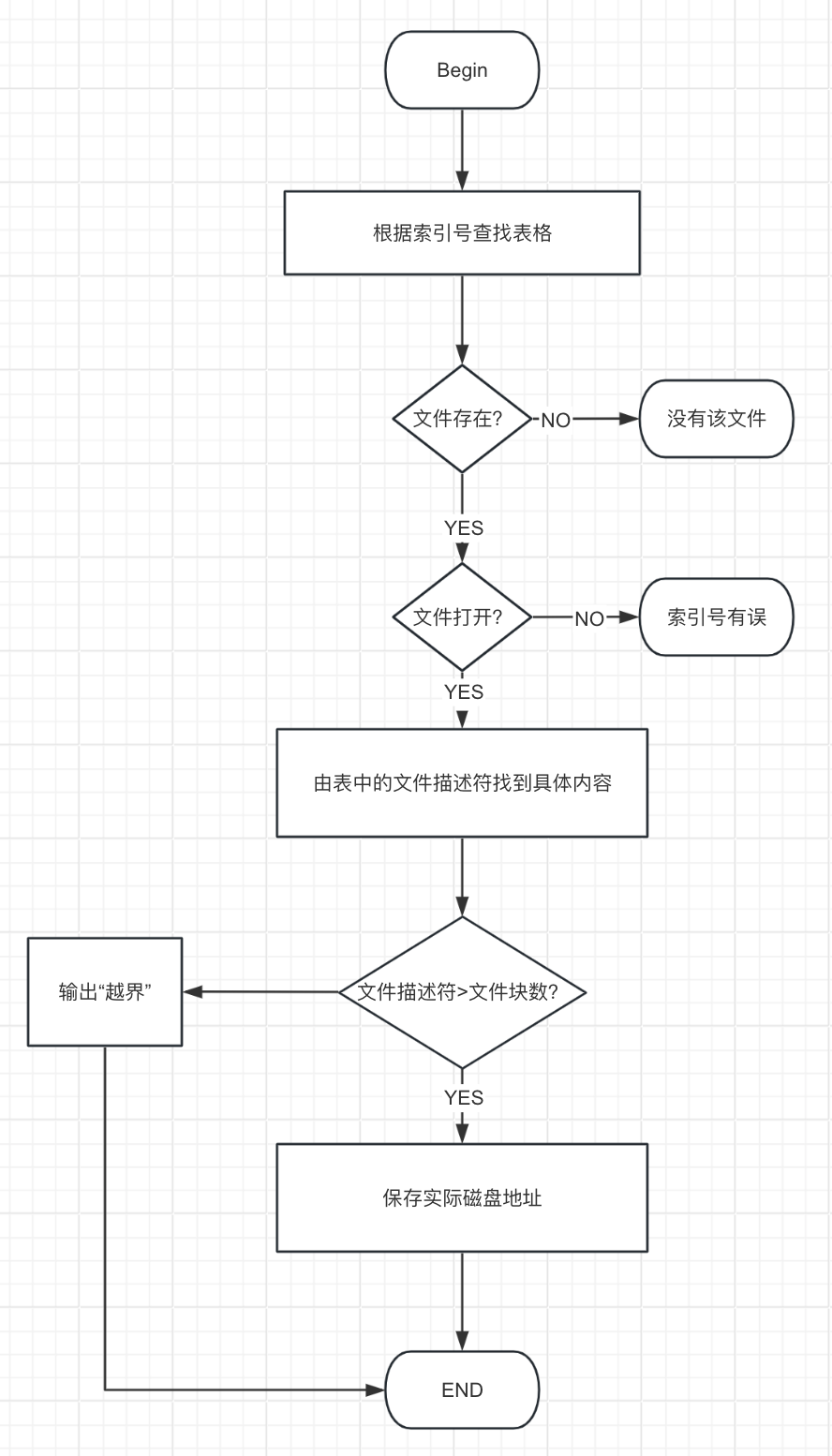
 read(int index, int mem\_area, int count)：



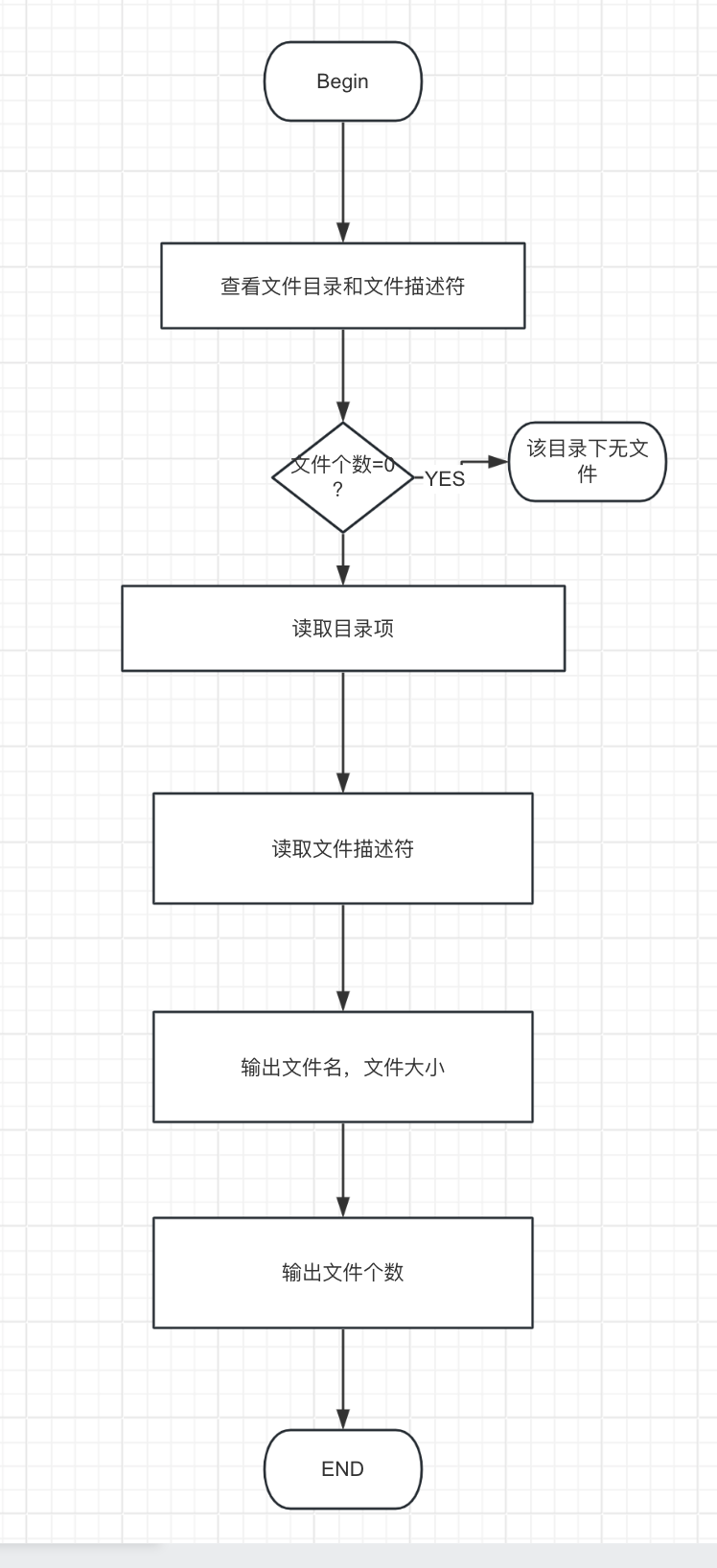
 write(int index, int mem\_area, int count)：



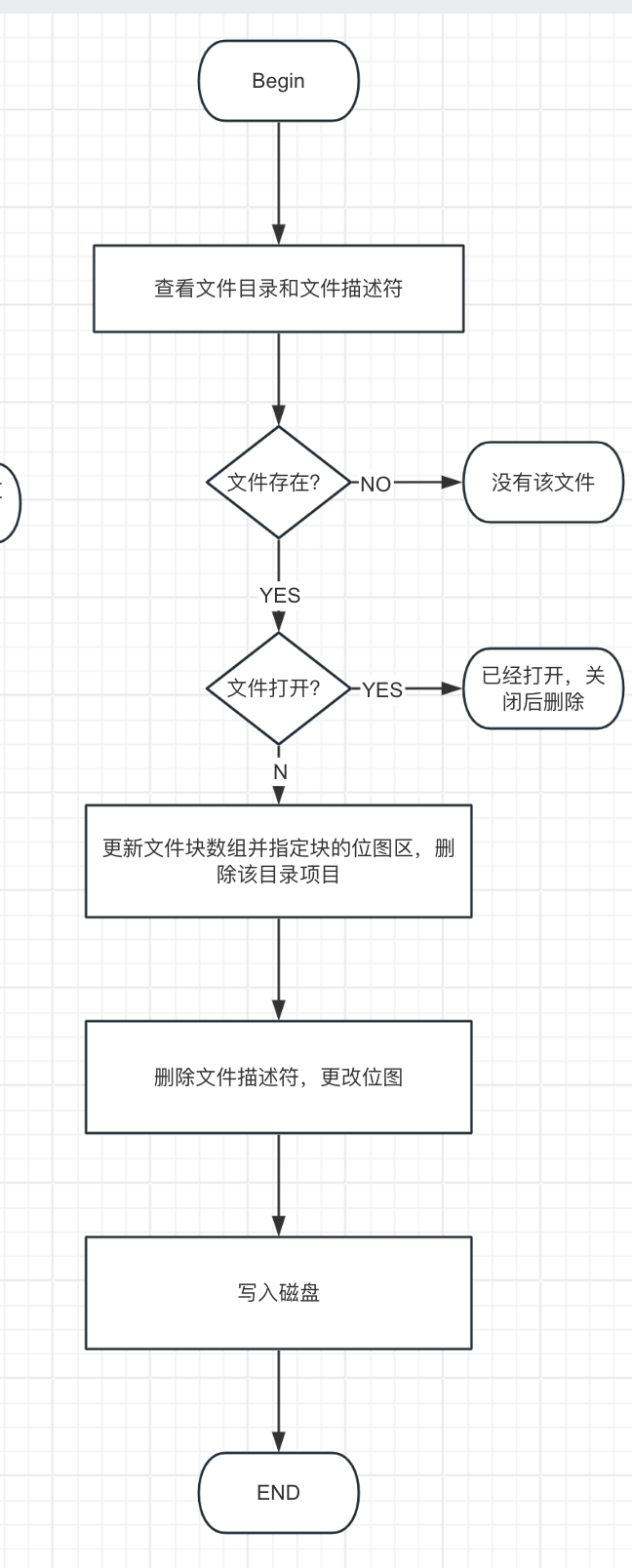
lseek(int index, int pos)：



directory()：



 destroy(char\* filename)：

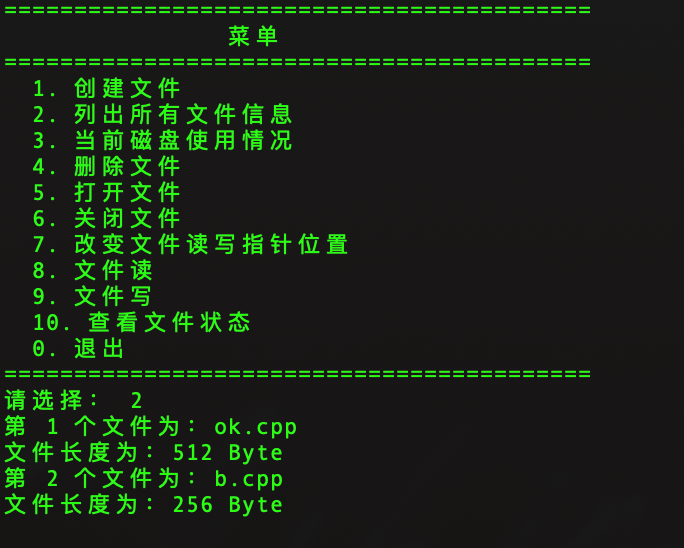


# 实验结果

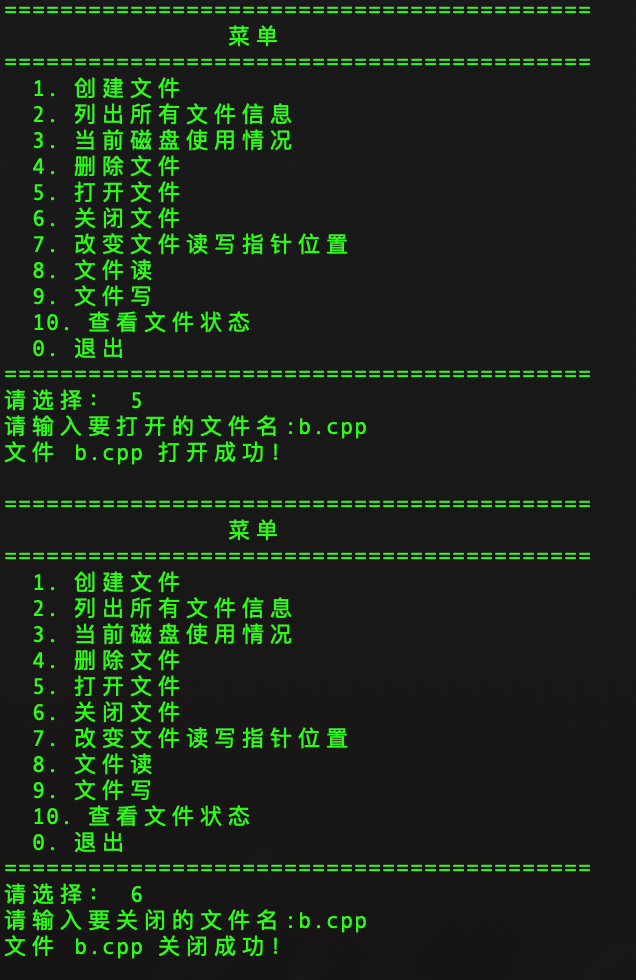
文件写入：

## 截屏2023-11-25 23.43.08

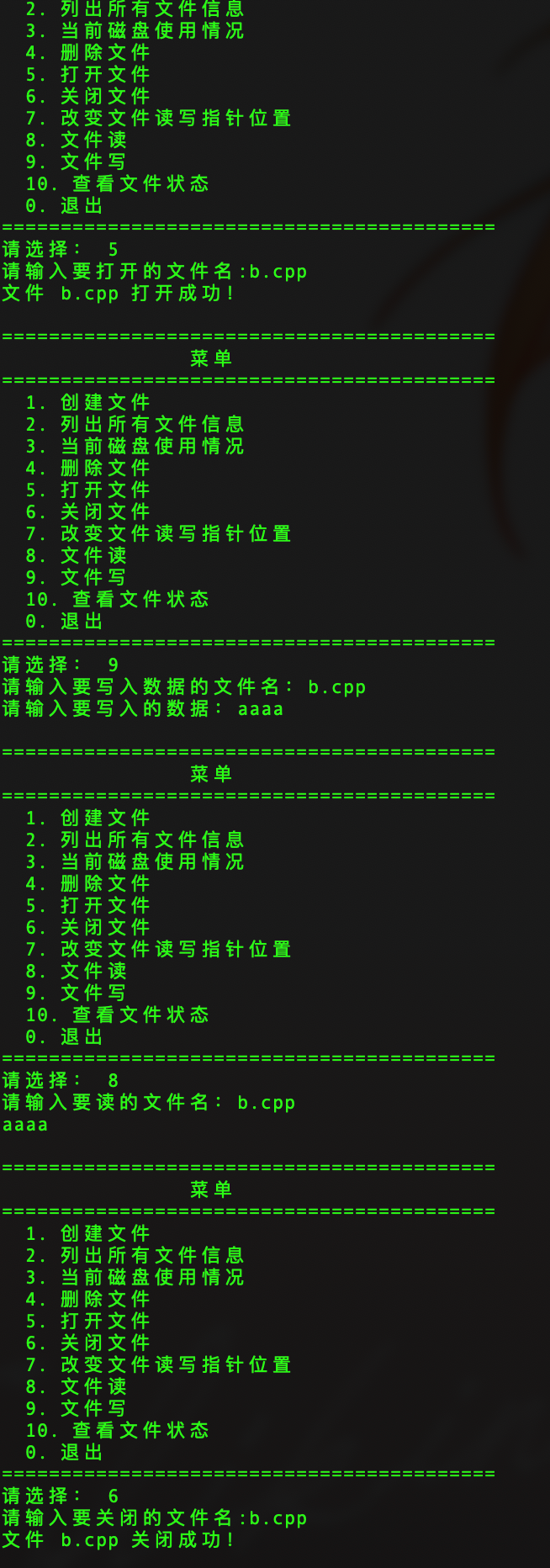
文件列出：



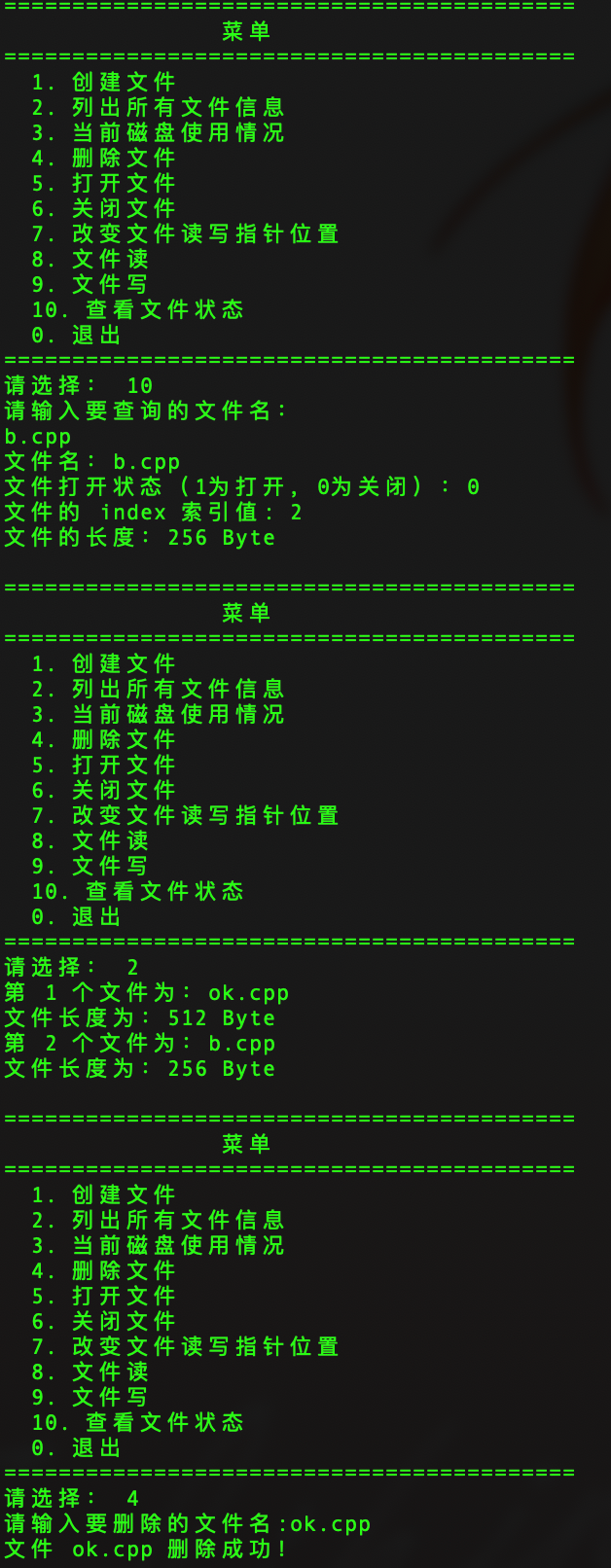
文件打开关闭：



文件写入读出：



文件删除与查看状态：



# 实验代码与仓库

请参考：https://github.com/sksx085/OSHomework/exp6