```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
#include <string.h>
#include <ext2fs/ext2_fs.h>
//#include <linux/ext2_fs.h>
//#include <ext2fs/ext2_types.h>
//#include "ext2_fs.h"

//constants
#define BLKSIZE 1024
#define MAGIC 0xEF53

//typedefs
typedef struct ext2_inode INODE;
typedef struct ext2_super_block SUPER;
typedef struct ext2_dir_entry_2 DIR;
typedef struct ext2_group_desc GD;

//global variables
char ibuf[BLKSIZE], sup_block[BLKSIZE], group_desc[BLKSIZE], buf[BLKSIZE], ubuf[BLKSIZE];
INODE *ip, *root;
int bmap, imap, inode_start;
//int ninodes, nblocks;
char *name[256];
uint32_t *up, *uup;
int dev;
SUPER *sup;

//function prototypes
int readGroupDesc(char *gdp);
int show_dir(INODE *ip);
int search(INODE *ip, char *name);
INODE * find_inode(int dev, char *pathname);

// get_block() reads a disk BLOCK into a char buf[BLKSIZE].
int get_block(int dev, int blk, char *buf)
{
	lseek(dev, blk*BLKSIZE, SEEK_SET);
	return read(dev, buf, BLKSIZE);
}

int main(int argc, char *argv[]) {
	dev = open("diskimage", O_RDONLY);   // OR  O_RDWR
	char pathname[256] = "/Z/hugefile";
	int i, x, y;

	//read super block, verify ext2
	if (get_block(dev, 1, ibuf)) {
		sup = (SUPER *)ibuf;
		if (sup->s_magic != MAGIC) {
			printf("diskimage is not ext2\n");
			exit(1);
		}
	}
```

```
	//read group descriptor
	get_block(dev, 2, group_desc);
	readGroupDesc(group_desc);


	//read root inode
	get_block(dev, inode_start, ibuf);
	ip = (INODE *)ibuf + 1;
	show_dir(ip);

	printf("Enter pathname : ");
	fgets(pathname, 256, stdin);         // get a line (end with \n) from stdin
	line[strlen(line) - 1] = 0;
	find_inode(dev, pathname);
	////print dir info
	for (i = 0; i < 15; ++i) {
		if (i < 12) {
			printf("block %d\n", ip->i_block[i]);

		}
		else if (i == 12) {
			//iterate over indirect block list
			get_block(dev, ip->i_block[i], buf);
			up = (uint32_t *)buf;
			while (*(up) != 0) {
				printf("indirect block %d\n", *(up));
				++up;
			}
		}
		else  if (i == 13) {
			//iterate over double indirect block list
			get_block(dev, ip->i_block[i], buf);
			up = (uint32_t *)buf;
			while (*(up) != 0) {
				get_block(dev, ip->i_block[i], ubuf);
				uup = (uint32_t *)ubuf;
				while (*(uup) != 0) {
					printf("double indirect block %d\n", *(uup));
					++uup;
				}

				++up;
			}
		}
		else {

		}
	}
}

int readGroupDesc(char *gdp) {
	//typecast buf to Group Descriptor
	GD *group_desc = (GD *)gdp;
	bmap = group_desc->bg_block_bitmap;
	imap = group_desc->bg_inode_bitmap;
	inode_start = group_desc->bg_inode_table;

	//print
```

```
        printf("bmap = %d\nimap = %d\ninode_start = %d\n", bmap, imap, inode_start);

}

int show_dir(INODE *ip) {
        char sbuf[BLKSIZE], temp[256];
        DIR *dp;
        char *cp;
        int i;

        for (i = 0; i < 12; ++i) {//assume at most 12 direct blocks
                if (ip->i_block[i] == 0) {
                        break;
                }
                printf("i_block = %d\n", ip->i_block[i]);
                get_block(dev, ip->i_block[i], sbuf);

                dp = (DIR *)sbuf;
                cp = sbuf;

                while (cp < sbuf + BLKSIZE) {
                        strncpy(temp, dp->name, dp->name_len);
                        temp[dp->name_len] = 0;
                        printf("%4d %4d %4d %s\n", dp->inode, dp->rec_len, dp->name_len,
temp);
                        cp += dp->rec_len;
                        dp = (DIR *)cp;

                }
        }
}

int search(INODE *ip, char *name) {
        char sbuf[BLKSIZE], temp[256];
        DIR *dp;
        char *cp;
        int i, inum = 0;
        char tp[24];

        for (i = 0; i < 12; ++i) {
                if (ip->i_block[i] == 0) {
                        break;
                }

                get_block(dev, ip->i_block[i], sbuf);
                dp = (DIR *)sbuf;
                cp = sbuf;
                putchar('\n');


                while (cp < sbuf + BLKSIZE) {
                        strncpy(temp, dp->name, dp->name_len);
                        //may not need
                        temp[dp->name_len] = 0;
                        if (!strcmp(temp, name)) {
                                inum = dp->inode;
                                return inum;
                        }
```

```
                cp += dp->rec_len;
                dp = (DIR *)cp;

            }

        }

        return inum;
}

INODE * find_inode(int dev, char *pathname) {
        int ino, blk, offset, n = 0, i = 0;
        char *name[256], *s, tp[24];

        //tokenize pathname
        s = strtok(pathname, "/");
        while (s) {
                name[i] = s;
                s = strtok(NULL, "/");
                ++i;
        }
        n = i;

        //set ip to root
        get_block(dev, inode_start, ibuf);
        ip = (INODE *)ibuf + 1;


        for (i = 0; i < n; ++i) {
                ino = search(ip, name[i]);

                if (ino == 0) {
                        printf("can't find %s\n", name[i]);
                        exit(1);
                }

                //mailmans algorithm: convert (dev, ino) to INODE pointer
                blk = (ino - 1) / 8 + inode_start;
                offset = (ino - 1) % 8;
                get_block(dev, blk, ibuf);
                ip = (INODE *)ibuf + offset;

        }
        return ip;
}
```