```c
// cs2.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "cs2.h"

static CMDVAL s_cmds [] = {{"cd",CD_CMD},{"mkdir",MKDIR_CMD},{"ls",LS_CMD},{"touch",TOUCH_CMD},{"exit",EXIT_CMD},{NULL,INVALID_CMD}};

int ValidateCmd(char *m_cmd){
    int i;
    for(i=0; s_cmds[i].cmd!=NULL;i++){
        if (strcmp(m_cmd,s_cmds[i].cmd)==0)break;
    }
    return (s_cmds[i].val);
}

bool CheckDupe(M_DIR *p_dir, char *dirname){
    bool retval = false;
    M_DIR *temp;
    for(temp=p_dir; temp!=NULL; temp=temp->next){
        if (strcmp(temp->dirName,dirname)==0){
            retval = true;
            break;
        }
    }
    return retval;
}

bool CheckDupeFile(MYFILE *p_dir, char *filename){
    bool retval = false;
    MYFILE *temp;
    for(temp=p_dir; temp!=NULL; temp=temp->next){
        if (strcmp(temp->filename,filename)==0){
            retval = true;
            break;
        }
    }
    return retval;
}

void InitDirName(M_DIR *p_mdir, char *dirname){
    if((p_mdir->dirName = (char*)malloc(strlen(dirname)+1))==NULL){
        perror("Memory allocation failed in InitDirName.\n");
        exit(0);
    }
    strcpy(p_mdir->dirName,dirname);
}

void AddSibling(M_DIR *cur, M_DIR *new){
    M_DIR *p_cur, *p_prev;
    for (p_cur = cur; p_cur≠NULL; p_prev = p_cur, p_cur = p_cur→next);
    p_prev→next = new;
}

M_DIR *ChangeDir(M_FILESYSTEM* p_fs, char* dirname){
    M_DIR *p_dir;
    for (p_dir=p_fs→root→subdir; p_dir≠NULL;p_dir=p_dir→next){
        if (strcmp(dirname, p_dir→dirName)==0) break;
    }
    return p_dir;
}


M_DIR *Allocate_dir(char *dirname,M_DIR *parent){
    M_DIR *p_mdir;
    if ((p_mdir=(M_DIR*)malloc(sizeof(M_DIR)))==NULL){
        perror("Memory allocation failed.\n");
        exit(0);
    }
    p_mdir→subdir=NULL;
    p_mdir→next=NULL;
    p_mdir→files=NULL;
    p_mdir→parent=parent;
    InitDirName(p_mdir,dirname);
    return p_mdir;
}

M_FILESYSTEM *Allocate_fs(char *dirname){
    M_FILESYSTEM *p_mfs;
    if((p_mfs = (M_FILESYSTEM*)malloc(sizeof(M_FILESYSTEM)))==NULL){
        perror("Memory allocation failed in Allocate_fs.\n");
        exit(0);
    }
    p_mfs→root = Allocate_dir(dirname,NULL);
    return p_mfs;
}


int MakeDir(M_DIR *cur,char *dirname){
    M_DIR *p_mdir;
    int retval=0;
    if (cur→parent ≠ NULL){
        retval=-1;
        printf("No subdirectory system implemented in this version.\n");
    }
    else{
        if (!CheckDupe(cur→subdir,dirname)){
            if ((p_mdir = (M_DIR*)malloc(sizeof(M_DIR)))==NULL){
                perror("Memory allocation failed in MakeDir.\n");
                exit(0);
            }
            p_mdir→parent = cur;
            InitDirName(p_mdir,dirname);
            if(cur→subdir==NULL)cur→subdir=p_mdir;
            else{
```

```c
                AddSibling(cur->subdir,p_mdir);
                p_mdir->next=NULL;
                p_mdir->files=NULL;
            }
        }
        else{
            retval=-1;
            printf("No duplicate names allowed.\n");
        }
    }
    return retval;
}

void ShowFiles (M_FILESYSTEM *p_fs){
    M_DIR *p_mdir;
    M_DIR *p_temp;
    MYFILE *p_mfile;
    if(p_fs!=NULL){
        p_mdir = p_fs->root;
        printf("%s\n", p_mdir->dirName);
        p_mdir = p_mdir->subdir;

        if(p_mdir!=NULL){
            printf("\t%s",p_mdir->dirName);
            for(p_temp = p_mdir->next; p_temp!=NULL;p_temp=p_temp->next)printf("\t%s",p_temp->dirName);

            printf("\n");
            p_mdir = p_fs->root;

            if(p_mdir!=NULL){
                for (p_mfile = p_mdir->files; p_mfile!=NULL; p_mfile = p_mfile->next) printf("%s\n",p_mfile->filename);
            }

            p_mdir = p_fs->root->subdir;

            for(p_temp=p_mdir; p_temp!=NULL; p_temp = p_temp->next){
                printf("Directory: %s\n", p_temp->dirName);
                for(p_mfile=p_temp->files; p_mfile!=NULL; p_mfile=p_mfile->next) printf("%s\t",p_mfile->filename);
                printf("\n");
            }
            printf("\n");
        }
    }
}

int CreateFile(M_DIR *cur, char *filename){
    MYFILE *p_myfile;
    MYFILE *p_temp, *p_prev;
    int len = strlen(filename)+1;
    int retval=0;

    if (!CheckDupeFile(cur->files,filename)){
        if ((p_myfile = (MYFILE*)malloc(sizeof(MYFILE)))==NULL){
            perror("Memory allocation failed in CreateFile.\n");
            exit(0);
        }
        p_myfile->next = NULL;

        if(cur->files==NULL){
            cur->files=p_myfile;
            if ((cur->files->filename = malloc(len))==NULL){
                printf("Memory allocation failed in CreateFile.\n");
                exit(0);
            }
            strcpy(cur->files->filename,filename);
        }
        else{
            for (p_temp = cur->files; p_temp!=NULL; p_prev = p_temp, p_temp = p_temp->next);
            p_prev->next = p_myfile;
            if ((p_prev->next->filename = (char *)malloc(len))==NULL){
                printf("Memory allocation failed in CreateFile.\n");
                exit(0);
            }
            strcpy(p_prev->next->filename,filename);
        }
    }
    else{
        retval=-1;
        printf("No duplicate names allowed.\n");
    }
    return retval;
}

void free_mem(M_FILESYSTEM *p_fs){
    M_DIR *cur_d, *temp_d;
    MYFILE *cur_f, *temp_f;
    cur_d = p_fs->root;
    temp_d = cur_d;
    cur_f = cur_d->files;

    while (cur_f!=NULL){
        temp_f = cur_f;
        cur_f=cur_f->next;
        //printf("Freed %s\n",temp_f->filename); //debug
        free(temp_f);
    }

    cur_d=cur_d->subdir;
    free(temp_d);
    //printf("Freed root\n"); //debug

    while(cur_d!=NULL){
        cur_f = cur_d->files;
        while (cur_f!=NULL){
            temp_f = cur_f;
            cur_f=cur_f->next;
```

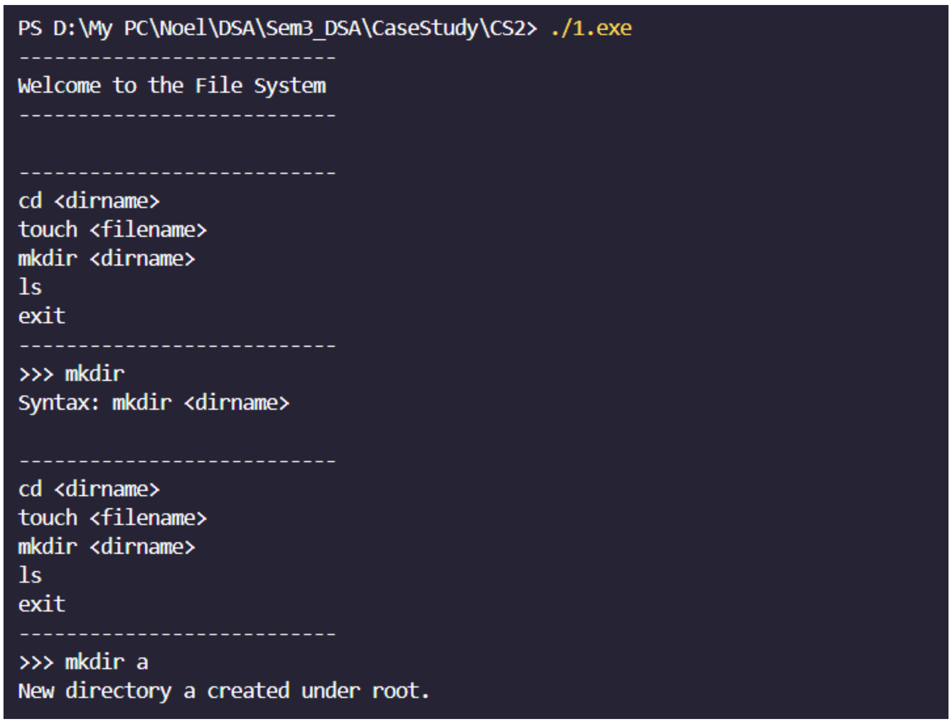
```
                //printf("Freed %s\n",temp_f->filename); //debug
                free(temp_f);
            }
            temp_d = cur_d;
            cur_d = cur_d->next;
            //printf("Freed %s\n",temp_d->dirName); //debug
            free(temp_d);
        }
}
```

```c
//cs2.h

#define MAXSTRING 50

typedef struct myfile{
    char *filename;
    struct myfile *next;
} MYFILE;

typedef struct m_directory{
    char* dirName;
    struct m_directory *subdir;
    struct m_directory *next;
    MYFILE *files;
    struct m_directory *parent;
} M_DIR;

typedef struct FileSystem{
    M_DIR *root;
} M_FILESYSTEM;

enum CmdVal {CD_CMD,MKDIR_CMD,LS_CMD,TOUCH_CMD,EXIT_CMD,INVALID_CMD};

typedef struct cmdval{
    char *cmd;
    enum CmdVal val;
} CMDVAL;

int ValidateCmd(char *);
M_FILESYSTEM *Allocate_fs(char *);
M_DIR *Allocate_dir(char *,M_DIR *);
int MakeDir(M_DIR *,char *);
M_DIR *ChangeDir(M_FILESYSTEM *,char *);
void ShowFiles (M_FILESYSTEM *);
int CreateFile(M_DIR *, char *);
void AddSibling(M_DIR *,M_DIR *);
void free_mem(M_FILESYSTEM *);
```

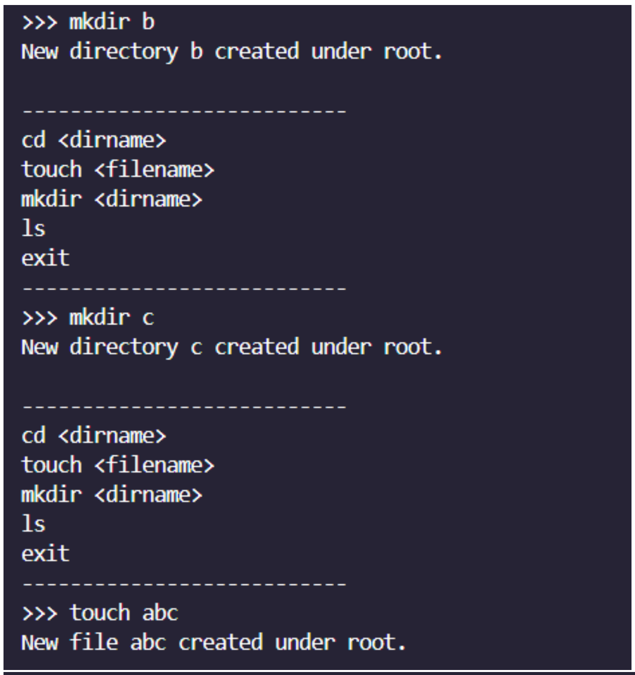
```c
//cs2main.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "cs2.h"

int main (void){
    char choice[MAXSTRING];
    enum CmdVal ValCmd;
    int retval;
    bool go_on = true;
    char m_cmd[10];
    char m_arg[10];
    M_FILESYSTEM *fs_head;
    M_DIR *cur;

    fs_head = Allocate_fs("root");
    cur = fs_head->root;

    printf("---------------------------\nWelcome to the File System\n---------------------------\n");
    while(go_on){
        m_arg[0]=0;
        printf("\n---------------------------\ncd <dirname>\ntouch <filename>\nmkdir <dirname>\nls\nexit\n---------------------------\n>>> ");
        scanf(" %[^\n]s",choice);
        sscanf(choice,"%s%s",m_cmd,m_arg);
        ValCmd = ValidateCmd(m_cmd);

        switch(ValCmd){
            case CD_CMD:
                if (m_arg[0]!=0){
                    if (strcmp(m_arg,"..")==0){
                        if (cur==fs_head->root)printf("Already at root.\n");
                        else{
                            cur = cur->parent;
                        }
                    }
                    else{
                        M_DIR *temp = cur;
                        if (cur==fs_head->root){
                            cur = ChangeDir(fs_head,m_arg);
                            if (cur==NULL){
                                printf("Directory does not exist.\n");
                                cur = temp;
                            }
                        }
                        else{
                            printf("Directory does not exist.\n");
                        }
                    }
                    printf("Current working directory: %s\n",cur->dirName);
                }
                else{
                    printf("Syntax: cd <dirname>\n");
                }
            break;

            case MKDIR_CMD:
                if (m_arg[0]!=0){
                    retval = MakeDir(cur, m_arg);
                    if (retval==0)printf("New directory %s created under %s.\n",m_arg,cur->dirName);
                }
                else{
                    printf("Syntax: mkdir <dirname>\n");
                }
            break;

            case LS_CMD:
                ShowFiles(fs_head);
            break;

            case TOUCH_CMD:
                if (m_arg[0]!=0){
                    retval = CreateFile(cur, m_arg);
                    if (retval==0)printf("New file %s created under %s.\n",m_arg,cur->dirName);
                }
                else{
                    printf("Syntax: touch <filename>\n");
                }
            break;

            case EXIT_CMD:
                printf("Exiting...\n");
                go_on=false;
            break;

            default:
                printf("Unsupported command.\n");
            break;
        }
    }
    free_mem(fs_head);
    return 0;
}
```

Input:

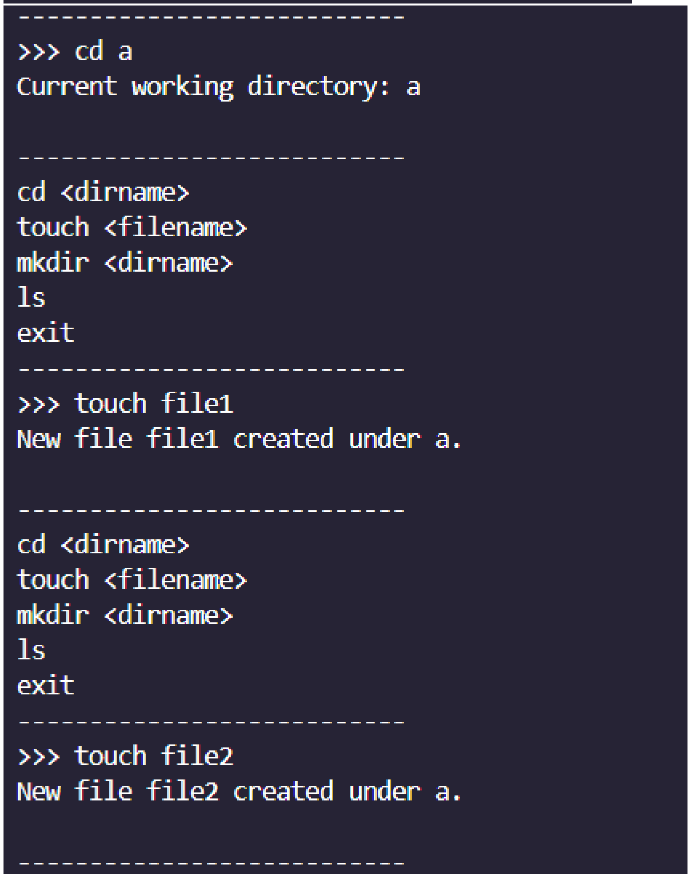
```
mkdir
mkdir a
mkdir b
mkdir c
touch abc
cd a
touch file1
touch file2
cd b
cd ..
cd b
touch file3
cd ..
cd d
cd c
touch file4
ls
exit
```



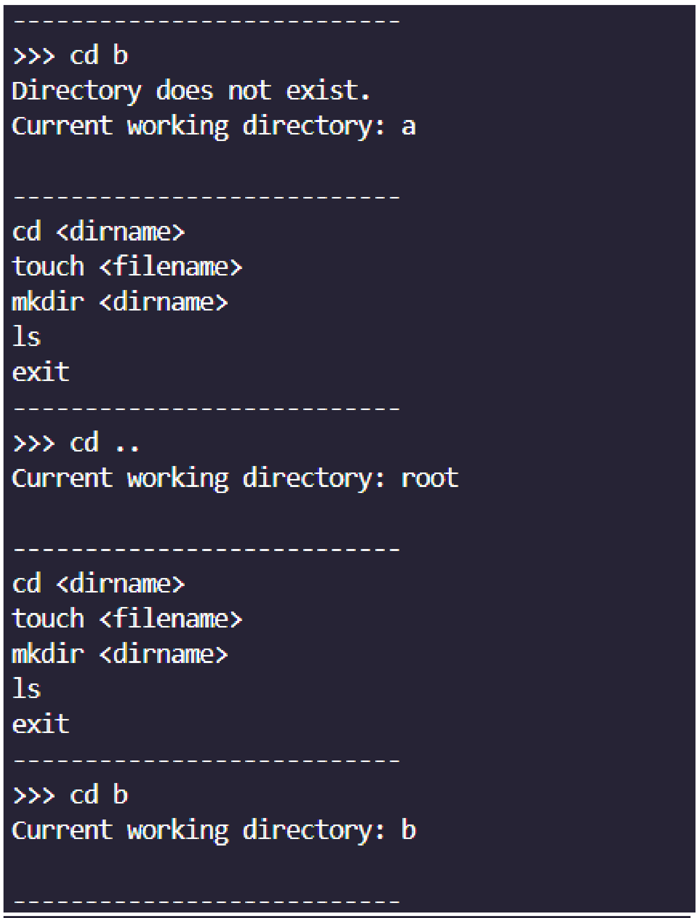
```
PS D:\My PC\Noel\DSA\Sem3_DSA\CaseStudy\CS2> ./1.exe
---------------------------
Welcome to the File System
---------------------------

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> mkdir
Syntax: mkdir <dirname>

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> mkdir a
New directory a created under root.
```



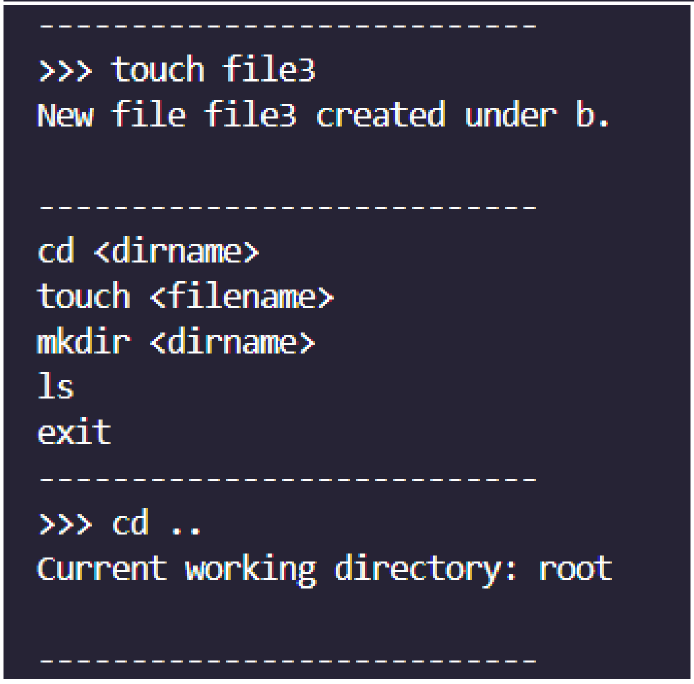
```
>>> mkdir b
New directory b created under root.

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> mkdir c
New directory c created under root.

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> touch abc
New file abc created under root.
```



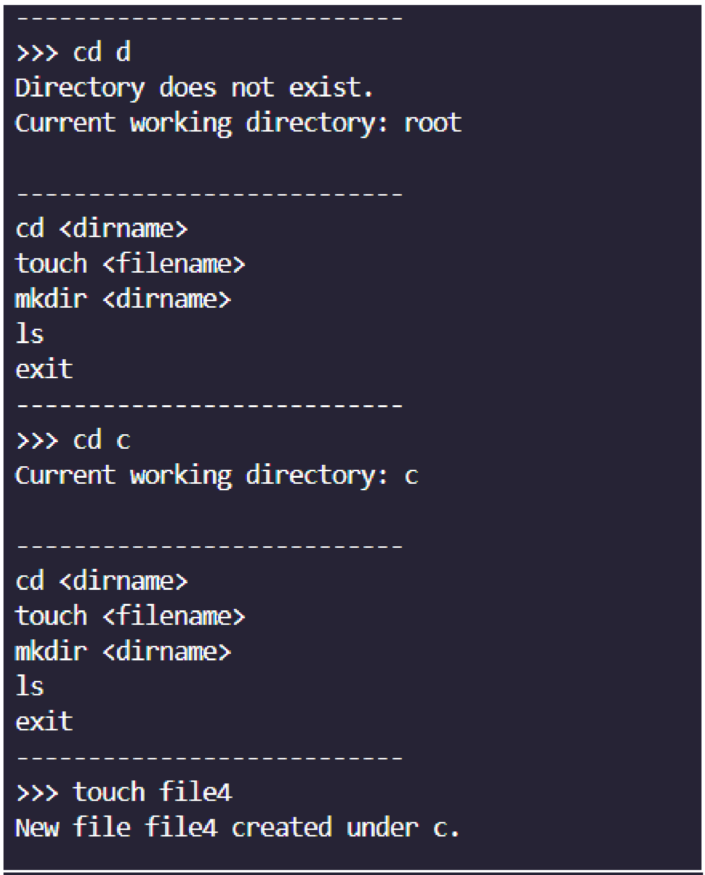
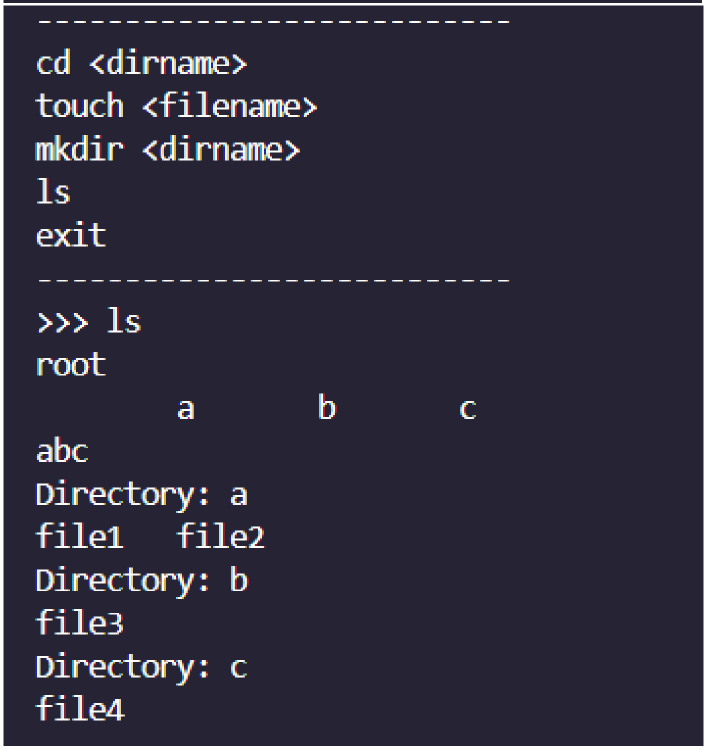
```
---------------------------
>>> cd a
Current working directory: a

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> touch file1
New file file1 created under a.

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> touch file2
New file file2 created under a.

---------------------------
```

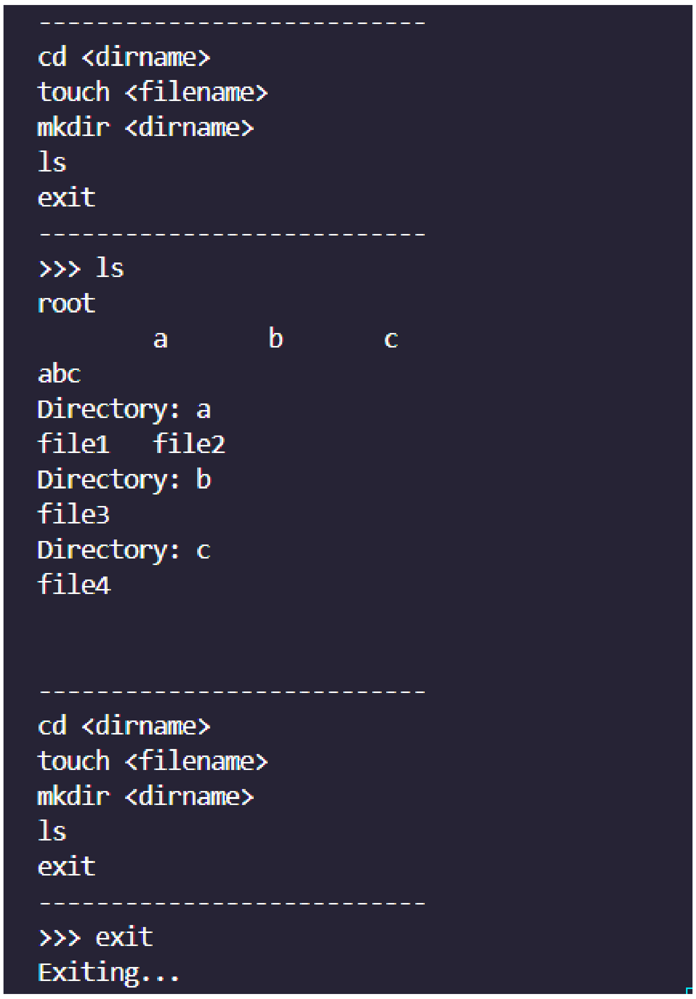


```
---------------------------
>>> cd b
Directory does not exist.
Current working directory: a

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> cd ..
Current working directory: root

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> cd b
Current working directory: b

---------------------------
```



```
---------------------------
>>> touch file3
New file file3 created under b.

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> cd ..
Current working directory: root

---------------------------
```


```
---------------------------
>>> cd d
Directory does not exist.
Current working directory: root

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> cd c
Current working directory: c

---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> touch file4
New file file4 created under c.
```

```
---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> ls
root
          a         b         c
abc
Directory: a
file1   file2
Directory: b
file3
Directory: c
file4
```



```
---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> ls
root
        a       b       c
abc
Directory: a
file1   file2
Directory: b
file3
Directory: c
file4


---------------------------
cd <dirname>
touch <filename>
mkdir <dirname>
ls
exit
---------------------------
>>> exit
Exiting...
```