

```
1 // cs2.c
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <stdbool.h>
7 #include "cs2.h"
8
9 static CMDVAL s_cmds [] = {{ "cd", CD_CMD }, { "mkdir", MKDIR_CMD }, { "ls", LS_CMD },
10 {"touch", TOUCH_CMD }, { "exit", EXIT_CMD }, { NULL, INVALID_CMD } };
11
12 int ValidateCmd(char *m_cmd){
13     int i;
14     for(i=0; s_cmds[i].cmd!=NULL;i++){
15         if (strcmp(m_cmd,s_cmds[i].cmd)==0)break;
16     }
17     return (s_cmds[i].val);
18 }
19
20 bool CheckDupe(M_DIR *p_dir, char *dirname){
21     bool retval = false;
22     M_DIR *temp;
23     for(temp=p_dir; temp!=NULL; temp=temp->next){
24         if (strcmp(temp->dirName,dirname)==0){
25             retval = true;
26             break;
27         }
28     }
29     return retval;
30 }
31
32 bool CheckDupeFile(MYFILE *p_dir, char *filename){
33     bool retval = false;
34     MYFILE *temp;
35     for(temp=p_dir; temp!=NULL; temp=temp->next){
36         if (strcmp(temp->filename,filename)==0){
37             retval = true;
38             break;
39         }
40     }
41     return retval;
42 }
43
44 void InitDirName(M_DIR *p_mdir, char *dirname){
45     if((p_mdir->dirName = (char*)malloc(strlen(dirname)+1))==NULL){
46         perror("Memory allocation failed in InitDirName.\n");
47         exit(0);
48     }
49     strcpy(p_mdir->dirName,dirname);
50 }
51
52 void AddSibling(M_DIR *cur, M_DIR *new){
53     M_DIR *p_cur, *p_prev;
```

```
53     for (p_cur = cur; p_cur≠NULL; p_prev = p_cur, p_cur = p_cur→next);
54     p_prev→next = new;
55 }
56
57 M_DIR *ChangeDir(M_FILESYSTEM* p_fs, char* dirname){
58     M_DIR *p_dir;
59     for (p_dir=p_fs→root→subdir; p_dir≠NULL;p_dir=p_dir→next){
60         if (strcmp(dirname, p_dir→dirName)=0) break;
61     }
62     return p_dir;
63 }
64
65
66 M_DIR *Allocate_dir(char *dirname,M_DIR *parent){
67     M_DIR *p_mdir;
68     if ((p_mdir=(M_DIR*)malloc(sizeof(M_DIR)))=NULL){
69         perror("Memory allocation failed.\n");
70         exit(0);
71     }
72     p_mdir→subdir=NULL;
73     p_mdir→next=NULL;
74     p_mdir→files=NULL;
75     p_mdir→parent=parent;
76     InitDirName(p_mdir,dirname);
77     return p_mdir;
78 }
79
80 M_FILESYSTEM *Allocate_fs(char *dirname){
81     M_FILESYSTEM *p_mfs;
82     if((p_mfs = (M_FILESYSTEM*)malloc(sizeof(M_FILESYSTEM)))=NULL){
83         perror("Memory allocation failed in Allocate_fs.\n");
84         exit(0);
85     }
86     p_mfs→root = Allocate_dir(dirname,NULL);
87     return p_mfs;
88 }
89
90
91 int MakeDir(M_DIR *cur,char *dirname){
92     M_DIR *p_mdir;
93     int retval=0;
94     if (cur→parent ≠ NULL){
95         retval=-1;
96         printf("No subdirectory system implemented in this version.\n");
97     }
98     else{
99         if (!CheckDupe(cur→subdir,dirname)){
100             if ((p_mdir = (M_DIR*)malloc(sizeof(M_DIR)))=NULL){
101                 perror("Memory allocation failed in MakeDir.\n");
102                 exit(0);
103             }
104             p_mdir→parent = cur;
105             InitDirName(p_mdir,dirname);
106             if(cur→subdir=NULL)cur→subdir=p_mdir;
107             else{
```

```
108         AddSibling(cur→subdir,p_mdir);
109         p_mdir→next=NULL;
110         p_mdir→files=NULL;
111     }
112 }
113 else{
114     retval=-1;
115     printf("No duplicate names allowed.\n");
116 }
117 }
118 return retval;
119 }
120
121 void ShowFiles (M_FILESYSTEM *p_fs){
122     M_DIR *p_mdir;
123     M_DIR *p_temp;
124     MYFILE *p_mfile;
125     if(p_fs≠NULL){
126         p_mdir = p_fs→root;
127         printf("%s\n", p_mdir→dirName);
128         p_mdir = p_mdir→subdir;
129
130         if(p_mdir≠NULL){
131             printf("\t%s",p_mdir→dirName);
132             for(p_temp = p_mdir→next; p_temp≠NULL;p_temp=p_temp-
>next)printf("\t%s",p_temp→dirName);
133
134             printf("\n");
135             p_mdir = p_fs→root;
136
137             if(p_mdir≠NULL){
138                 for (p_mfile = p_mdir→files; p_mfile≠NULL; p_mfile =
p_mfile→next) printf("%s\n",p_mfile→filename);
139             }
140
141             p_mdir = p_fs→root→subdir;
142
143             for(p_temp=p_mdir; p_temp≠NULL; p_temp = p_temp→next){
144                 printf("Directory: %s\n", p_temp→dirName);
145                 for(p_mfile=p_temp→files; p_mfile≠NULL; p_mfile=p_mfile-
>next) printf("%s\t",p_mfile→filename);
146                 printf("\n");
147             }
148             printf("\n");
149         }
150     }
151 }
152
153 int CreateFile(M_DIR *cur, char *filename){
154     MYFILE *p_myfile;
155     MYFILE *p_temp, *p_prev;
156     int len = strlen(filename)+1;
157     int retval=0;
158
159     if (!CheckDupeFile(cur→files,filename)){
```

```
160     if ((p_myfile = (MYFILE*)malloc(sizeof(MYFILE)))==NULL){
161         perror("Memory allocation failed in CreateFile.\n");
162         exit(0);
163     }
164     p_myfile->next = NULL;
165
166     if(cur->files==NULL){
167         cur->files=p_myfile;
168         if ((cur->files->filename = malloc(len))==NULL){
169             printf("Memory allocation failed in CreateFile.\n");
170             exit(0);
171         }
172         strcpy(cur->files->filename,filename);
173     }
174     else{
175         for (p_temp = cur->files; p_temp!=NULL; p_prev = p_temp, p_temp =
p_temp->next);
176         p_prev->next = p_myfile;
177         if ((p_prev->next->filename = (char *)malloc(len))==NULL){
178             printf("Memory allocation failed in CreateFile.\n");
179             exit(0);
180         }
181         strcpy(p_prev->next->filename,filename);
182     }
183 }
184 else{
185     retval=-1;
186     printf("No duplicate names allowed.\n");
187 }
188 return retval;
189 }
190
191 void free_mem(M_FILESYSTEM *p_fs){
192     M_DIR *cur_d, *temp_d;
193     MYFILE *cur_f, *temp_f;
194     cur_d = p_fs->root;
195     temp_d = cur_d;
196     cur_f = cur_d->files;
197
198     while (cur_f!=NULL){
199         temp_f = cur_f;
200         cur_f=cur_f->next;
201         //printf("Freed %s\n",temp_f->filename); //debug
202         free(temp_f);
203     }
204
205     cur_d=cur_d->subdir;
206     free(temp_d);
207     //printf("Freed root\n"); //debug
208
209     while(cur_d!=NULL){
210         cur_f = cur_d->files;
211         while (cur_f!=NULL){
212             temp_f = cur_f;
213             cur_f=cur_f->next;
```

```
214         //printf("Freed %s\n",temp_f->filename); //debug
215         free(temp_f);
216     }
217     temp_d = cur_d;
218     cur_d = cur_d->next;
219     //printf("Freed %s\n",temp_d->dirName); //debug
220     free(temp_d);
221 }
222 }
```

```
1 //cs2.h
2
3 #define MAXSTRING 50
4
5 typedef struct myfile{
6     char *filename;
7     struct myfile *next;
8 } MYFILE;
9
10 typedef struct m_directory{
11     char* dirName;
12     struct m_directory *subdir;
13     struct m_directory *next;
14     MYFILE *files;
15     struct m_directory *parent;
16 } M_DIR;
17
18 typedef struct FileSystem{
19     M_DIR *root;
20 } M_FILESYSTEM;
21
22 enum CmdVal {CD_CMD,MKDIR_CMD,LS_CMD,TOUCH_CMD,EXIT_CMD,INVALID_CMD};
23
24 typedef struct cmdval{
25     char *cmd;
26     enum CmdVal val;
27 } CMDVAL;
28
29 int ValidateCmd(char *);
30 M_FILESYSTEM *Allocate_fs(char *);
31 M_DIR *Allocate_dir(char *,M_DIR *);
32 int MakeDir(M_DIR *,char *);
33 M_DIR *ChangeDir(M_FILESYSTEM *,char *);
34 void ShowFiles (M_FILESYSTEM *);
35 int CreateFile(M_DIR *, char *);
36 void AddSibling(M_DIR *,M_DIR *);
37 void free_mem(M_FILESYSTEM *);
```

```
1 //cs2main.c
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <stdbool.h>
7 #include "cs2.h"
8
9 int main (void){
10     char choice[MAXSTRING];
11     enum CmdVal ValCmd;
12     int retval;
13     bool go_on = true;
14     char m_cmd[10];
15     char m_arg[10];
16     M_FILESYSTEM *fs_head;
17     M_DIR *cur;
18
19     fs_head = Allocate_fs("root");
20     cur = fs_head->root;
21
22     printf("—————\nWelcome to the File System\n—————\n");
23     while(go_on){
24         m_arg[0]=0;
25         printf("\n—————\ncd <dirname>\ntouch <filename>\nmkdir <dirname>\nls\nexit\n—————\n>>> ");
26         scanf(" %[^\\n]s",choice);
27         sscanf(choice,"%s%s",m_cmd,m_arg);
28         ValCmd = ValidateCmd(m_cmd);
29
30         switch(ValCmd){
31             case CD_CMD:
32                 if (m_arg[0]≠0){
33                     if (strcmp(m_arg,"..")=0){
34                         if (cur=fs_head->root)printf("Already at root.\n");
35                         else{
36                             cur = cur->parent;
37                         }
38                     }
39                     else{
40                         M_DIR *temp = cur;
41                         if (cur=fs_head->root){
42                             cur = ChangeDir(fs_head,m_arg);
43                             if (cur=NULL){
44                                 printf("Directory does not exist.\n");
45                                 cur = temp;
46                             }
47                         }
48                     }
49                     else{
50                         printf("Directory does not exist.\n");
51                     }
52                 }
53             }
54         }
55     }
```

```
52         printf("Current working directory: %s\n",cur→dirName);
53     }
54     else{
55         printf("Syntax: cd <dirname>\n");
56     }
57     break;
58
59     case MKDIR_CMD:
60         if (m_arg[0]≠0){
61             retval = MakeDir(cur, m_arg);
62             if (retval=0)printf("New directory %s created under
63 %s.\n",m_arg,cur→dirName);
64         }
65         else{
66             printf("Syntax: mkdir <dirname>\n");
67         }
68         break;
69
70     case LS_CMD:
71         ShowFiles(fs_head);
72         break;
73
74     case TOUCH_CMD:
75         if (m_arg[0]≠0){
76             retval = CreateFile(cur, m_arg);
77             if (retval=0)printf("New file %s created under
78 %s.\n",m_arg,cur→dirName);
79         }
80         else{
81             printf("Syntax: touch <filename>\n");
82         }
83         break;
84
85     case EXIT_CMD:
86         printf("Exiting ... \n");
87         go_on=false;
88         break;
89
90     default:
91         printf("Unsupported command.\n");
92         break;
93 }
94 free_mem(fs_head);
95 return 0;
96 }
```


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...
```