```
#include <stdio.h>
#include<dos.h>
#include<memory.h>
#include<string.h>
#include<stdlib.h>
#include <iostream.h>

struct BOOT{ //for FAT16
   char jmp[3];   // o dia D
        char OME[8];   // ten hang san xuat
        int bytes_per_sector; // kich thuoc sector tinh bang byte
        char sectors_per_cluster; // so sectors trong 1 cluster
        int reserved;  // so luong sectors danh cho vung dau dia den truoc FAT gom boot sector va sector du phong
        char FAT_cnt;  // dem so luong bang FAT
        int ROOT_size;
        int total_sectors;  // tong so sectors tren dia
        char media;
        int FAT_size;
        int sectors_per_track;  // so sectors tren 1 ranh
        int head_cnt;
        long hidden_sectors; // so luong sectors an
        long total_sectors_long; // tong so sectors tren dia cho truong hop nhieu hon 65535
        char unknown[3];
        long serial;
        char volume[11];
        char FAT_type[8];
        char loader[448];
        char mark[2];
```

```
};
struct ROOT{
	char name[8];
	char ext[3];
	char attr;
	char reserved[10];
	char time[2];
	char date[2];
	int  first_cluster;
};
void main(){
	int drive = 3; // A=0, B=1, C=2, D=3, ...

	//1. Reading boot sector from disk
	BOOT boot;

	int res = absread(drive, 1, 0, &boot); // absread la ham doc dia

	if(res != 0){
		printf("Cannot read boot sector\n");
		return;
	}
	printf("Reading disk parameters\n");
	printf("------------------------\n");
	printf("Sector size: %d\n", boot.bytes_per_sector);
	printf("FAT type:");
	int i;
	for(i = 0; i < 8; i++)
		printf("%c", boot.FAT_type[i]);
```

```
//2.1 Reading FAT16 table from disk D
unsigned int *fat = (unsigned int *)malloc (boot.FAT_size * boot.bytes_per_sector);
if (fat == NULL){
        printf("Not enough memory\n");
        return;
}
printf("\n\n");
printf("Reading FAT16 parameters\n");
printf("-------------------------\n");
printf("FAT size: %d \n", boot.FAT_size);
printf("Reserved: %d \n", boot.reserved);

res = absread(drive, boot.FAT_size, boot.reserved, fat);
if(res != 0){
        printf("Cannot read FAT\n");
        return;
}
//2.2 Printing first 15 FAT cells
printf("Content of first 15 FAT cells:");
for(i = 0; i < 15; i++)
        printf("%u ", fat[i]);

//5. Counting number of free clusters fromm first 100 clusters
int free_count = 0;
for(i = 2; i < 100; i++){
        if(fat[i] == 0) free_count++;
}
printf("\n");
printf("Number of free clusters from first 100 clusters:");
printf("%d\n", free_count);
```

```
        //Printing clusters of a file from clusters n
        unsigned int n = 5;
        unsigned int cur = n;
        printf("Clusters of a file from %u: ", n);
        while(cur <0xFFF8){
                printf("->%u", cur);
                cur = fat[cur];
        }
        // Reading ROOT from disk D
        printf("\n\n");
        printf("Reading ROOT information:\n");
        printf(-------------------------\n");
        int num_byte = boot.ROOT_size * 32; // sizeof(ROOT)

        ROOT *root = (ROOT *)malloc(num_byte);
        if(root == NULL) return;

        int num_sector = num_byte / boot.bytes_per_sector;
        int root_begin = boot.reserved + boot.FAT_size * boot.FAT_cnt;

        res = absread(drive, num_sector, root_begin, (void*)root);
if(res != 0){
                printf("\n Cannot read ROOT\n");
                return;
        }

        //Printing first 3 items of root
        printf("3 first items of root:\n");
        for(i = 0; i < 3; i++){
```

```
        if(root[i].name[0] == ' ') continue;
        for(int j = 0; j < 8 && root[i].name[j] != ' '; j++)
                printf("%c", root[i].name[j]);
        printf("\t");
        printf("%d \t %ld\n", root[i].first_cluster, root[i].size);
}

// Find a file with name given in char filename[]
printf("\n");
printf("Clusters belong to file readme:");
int k;
char str[9];
int first_cluster = -1;
for(i = 0; i < boot.ROOT_size; i++){
        //Copy root[i].name to str to make null-terminated string
        for(k = 0; k < 8 && root[i].name[k] != ' '; k++)
                str[k] = root[i].name[k];
        str[k] = 0;
        // Comparing
        char filename[8];
        printf("\n Enter a file name:");
        scanf("%s", filename);

        if(strcmp(str, filename) == 0){
                first_cluster = root[i].first_cluster;
                break;
        }
}

// Printing clusters belong to the file
```

```
        if(first_cluster >= 0){
                cur = first_cluster;
                while(cur <0xFFF8){
                        printf("%u ", cur);
                        cur = fat[cur];
                }
        }
        free(root);
        free(fat);
        getchar();
}
```