

Execute | Source Code | Share | Help

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define MAX_RECORDS 10
5 #define MAX_RECORD_SIZE 100
6
7 // Structure to represent a file
8 typedef struct {
9     char records[MAX_RECORDS][MAX_RECORD_SIZE];
10    int totalRecords;
11 } File;
12 // Function to initialize the file with records
13 void initializeFile(File* file) {
14     file->totalRecords = 0;
15 }
16 // Function to add a record to the file
17 void addRecord(File* file, const char* record) {
18     if (file->totalRecords < MAX_RECORDS) {
19         strcpy(file->records[file->totalRecords], record);
20         file->totalRecords++;
21     } else {
22         printf("File is full. Cannot add more records.\n");
23     }
24 }
```

Enter record index to access (0 to 4) or -1 to exit: 2  
Accessing record 2: Record 3: Data for third record

Enter record index to access (0 to 4) or -1 to exit: 4  
Accessing record 4: Record 5: Data for fifth record

Enter record index to access (0 to 4) or -1 to exit: -1



main.c

Share

Run

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 // Structure to represent a file block
5 typedef struct Block {
6     char data[100]; // Data of the block (you can change the size)
7     struct Block* next; // Pointer to the next block in the linked
        list
8 } Block;
9 // Structure to represent a file
10 typedef struct File {
11     Block* firstBlock; // Pointer to the first block
12     Block* lastBlock; // Pointer to the last block
13 } File;
14 // Function to initialize a file
15 void initializeFile(File* file) {
16     file->firstBlock = NULL;
17     file->lastBlock = NULL;
18 }
19 // Function to add data to the file (add a block)
20 void addBlock(File* file, const char* data) {
```

Output

Clear

Block added: Block 0: Data for the first block.  
Block added: Block 1: Data for the second block.  
Block added: Block 2: Data for the third block.  
Block added: Block 3: Data for the fourth block.  
  
Accessing file data:  
Block data: Block 0: Data for the first block.  
Block data: Block 1: Data for the second block.  
Block data: Block 2: Data for the third block.  
Block data: Block 3: Data for the fourth block.  
  
Enter the block position to delete (0 to 3): 2  
Block at position 2 deleted.  
  
Accessing file data after deletion:  
Block data: Block 0: Data for the first block.  
Block data: Block 1: Data for the second block.  
Block data: Block 3: Data for the fourth block.  
Memory released.

main.c

Share

Run

1 #include <stdio.h>

2 #include <stdlib.h>

3 void findSeekTime(int tracks[], int n, int start) {

4 int totalSeekTime = 0;

5 int currentPosition = start;

6 printf("\nDisk Scheduling (FCFS):\n");

7 printf("Initial head position: %d\n", start);

8 printf("Order of disk access requests:\n");

9 // Process each request in the order they arrive (FCFS)

10 for (int i = 0; i < n; i++) {

11 printf("Accessing track %d\n", tracks[i]);

12 totalSeekTime += abs(currentPosition - tracks[i]); // Calculate the seek time

13 currentPosition = tracks[i]; // Move the disk arm to the current track

14 }

15 printf("\nTotal seek time: %d\n", totalSeekTime);

16 printf("Average seek time: %.2f\n", (float)totalSeekTime / n);

17 }

18 int main() {

19 int n, start;

20 }

Output

Clear

Enter the number of disk access requests: 5

Enter the disk access requests (track numbers):

98 183 41 122 15

Enter the initial position of the disk arm: 53

Disk Scheduling (FCFS):

Initial head position: 53

Order of disk access requests:

Accessing track 98

Accessing track 183

Accessing track 41

Accessing track 122

Accessing track 15

Total seek time: 460

Average seek time: 92.00

=== Code Execution Successful ===

main.c

Run

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 // Function to perform the SCAN algorithm
4 void scanDiskScheduling(int tracks[], int n, int start, int direction
    , int diskSize) {
5     int totalSeekTime = 0;
6     int currentPosition = start;
7     int i, j;
8
9     // Sort the track requests
10    for (i = 0; i < n - 1; i++) {
11        for (j = 0; j < n - i - 1; j++) {
12            if (tracks[j] > tracks[j + 1]) {
13                int temp = tracks[j];
14                tracks[j] = tracks[j + 1];
15                tracks[j + 1] = temp;
16            }
17        }
18    }
19    // The direction can be either 1 (towards the higher end) or 0
    (towards the lower end)
```

Output

Clear

Enter the number of disk access requests: 5  
Enter the disk access requests (track numbers):  
98 122 33 41 23  
Enter the size of the disk (total number of tracks): 200  
Enter the initial position of the disk arm: 53  
Enter the initial direction of the disk arm (1 for right, 0 for left): 1  
  
Total seek time: 188  
Average seek time: 37.60  
  
=== Code Execution Successful ===





Execute | Source Code | Share | Help

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/stat.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 void display_permissions(mode_t mode) {
7     printf("File Permissions: ");
8     // User permissions
9     printf("%c", (mode & S_IRUSR) ? 'r' : '-');
10    printf("%c", (mode & S_IWUSR) ? 'w' : '-');
11    printf("%c ", (mode & S_IXUSR) ? 'x' : '-');
12    // Group permissions
13    printf("%c", (mode & S_IRGRP) ? 'r' : '-');
14    printf("%c", (mode & S_IWGRP) ? 'w' : '-');
15    printf("%c ", (mode & S_IXGRP) ? 'x' : '-');
16    // Others permissions
17    printf("%c", (mode & S_IROTH) ? 'r' : '-');
18    printf("%c", (mode & S_IWOTH) ? 'w' : '-');
19    printf("%c\n", (mode & S_IXOTH) ? 'x' : '-');
20 }
21 void change_permissions(const char* filename, mode_t mode)
22 {
23     if (chmod(filename, mode) == -1) {
24         perror("Error changing permissions");
25     }
26 }
```

Current Permissions of testfile.txt:  
File Permissions: rw- r-- r--  
Permissions changed successfully.  
Updated Permissions of testfile.txt:  
File Permissions: rwx r-x r-x  
Owner and Group changed successfully.

 Execute  Source Code  Share  Help

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <sys/stat.h>
5 #include <dirent.h>
6 #include <string.h>
7 #include <stdlib.h>
8 int main() {
9     int fd;
10    struct stat fileStat;
11    DIR *dir;
12    struct dirent *entry;
13
14    const char *filename = "sample.txt";
15    const char *dirname = "."; // Current directory
16    // Create and open file
17    fd = open(filename, O_RDWR | O_CREAT, 0644);
18    if (fd == -1) {
19        perror("Error opening file");
20        return 1;
21    }
22    printf("File '%s' opened successfully.\n", filename);
23    // Use fcntl to get file descriptor flags
24    int flags = fcntl(fd, F_GETFL);
```

File 'sample.txt' opened successfully.  
File access mode: Read/Write  
File pointer moved to the end: 0 bytes

#### File Information:

Size: 0 bytes

Permissions: rw-

#### Directory Contents of '.':

-> .  
-> ..  
-> sample.txt  
-> main  
-> testfile.txt  
-> main.c



Execute | `</>` Source Code | Share | Help

```
1 #include <stdio.h>
2 #define MAX 100
3 typedef struct {
4     int pid;
5     int arrival_time;
6     int burst_time;
7     int priority;
8     int remaining_time;
9     int completion_time;
10    int turnaround_time;
11    int waiting_time;
12 } Process;
13 int main() {
14     Process p[MAX];
15     int n, time = 0, completed = 0;
16     int i, current = -1, prev = -1;
17     float total_tat = 0, total_wt = 0;
18     printf("Enter the number of processes: ");
19     scanf("%d", &n);
20     printf("Enter Arrival Time, Burst Time and Priority\nfor each process:\n");
21     for (i = 0; i < n; i++) {
22         p[i].pid = i + 1;
23         printf("Process %d: ", p[i].pid);
```

```
Enter the number of processes: 4
Enter Arrival Time, Burst Time and Priority for each process:
Process 1: 0 8 2
Process 2: 1 4 1
Process 3: 2 8 3
Process 4: 3 5 2
PID AT  BT  PR  CT  TAT WT
1   0   8   2  12  12  4
2   1   4   1   5   4   0
3   2   8   3  25  23  15
4   3   5   2  17  14   9

Average Turnaround Time: 13.25
Average Waiting Time: 7.00
```

