Tanushri S.V.

192321163

CSA0428 Operating Systems for UI Design

31. Construct a C program to simulate the First in First Out paging technique of memory management.

**AIM:**

To simulate FIFO page replacement algorithm in memory management using C.

**ALGORITHM:**

1. Initialize empty frames.
2. For each page:

* If already in frame → hit.
* Else → replace oldest page (FIFO), and count page fault.

1. Print the current frame content.

**CODE:**

#include <stdio.h>

int main() {

int pages[20], frames[3] = {-1, -1, -1}, n, i, j, pos = 0, faults = 0, hit;

printf("Enter number of pages: ");

scanf("%d", &n);

printf("Enter pages: ");

for (i = 0; i < n; i++) scanf("%d", &pages[i]);

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

hit = 0;

for (j = 0; j < 3; j++)

if (frames[j] == pages[i]) hit = 1;

if (!hit) {

frames[pos] = pages[i];

pos = (pos + 1) % 3;

faults++;

}

printf("Frames: %d %d %d\n", frames[0], frames[1], frames[2]);

}

printf("Total Page Faults: %d\n", faults);

return 0;

}

**SAMPLE INPUT:**

Enter number of pages: 4

Enter page numbers: 1 2 3 4

**SAMPLE OUTPUT:**

Frames: 1 -1 -1

Frames: 1 2 -1

Frames: 1 2 3

Frames: 4 2 3

Total Page Faults = 4

**A screenshot of a computer

AI-generated content may be incorrect.**

**RESULT:**

FIFO paging simulation program executed successfully.

32. Construct a C program to simulate the Least Recently Used paging technique of memory management.

**AIM:**

To simulate the LRU page replacement algorithm using arrays and timestamps in C.

**ALGORITHM:**

1. Initialize 3 frames with -1.
2. For each page:

* If it's a hit → update its time.
* If it's a miss → replace the least recently used page.

1. Count and print page faults and frames.

**CODE:**

#include <stdio.h>

int main() {

int pages[20], n, frames[3] = {-1, -1, -1}, used[3] = {0}, time = 0, faults = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

printf("Enter page numbers: ");

for (int i = 0; i < n; i++) scanf("%d", &pages[i]);

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

int hit = 0, min = 0;

for (int j = 0; j < 3; j++) {

if (frames[j] == pages[i]) {

hit = 1;

used[j] = ++time;

}

}

if (!hit) {

for (int j = 1; j < 3; j++)

if (used[j] < used[min]) min = j;

frames[min] = pages[i];

used[min] = ++time;

faults++;

}

printf("Frames: %d %d %d\n", frames[0], frames[1], frames[2]);

}

printf("Total Page Faults = %d\n", faults);

return 0;

}

**SAMPLE INPUT:**

Enter number of pages: 4

Enter pages: 1 2 3 1

**SAMPLE OUTPUT:**

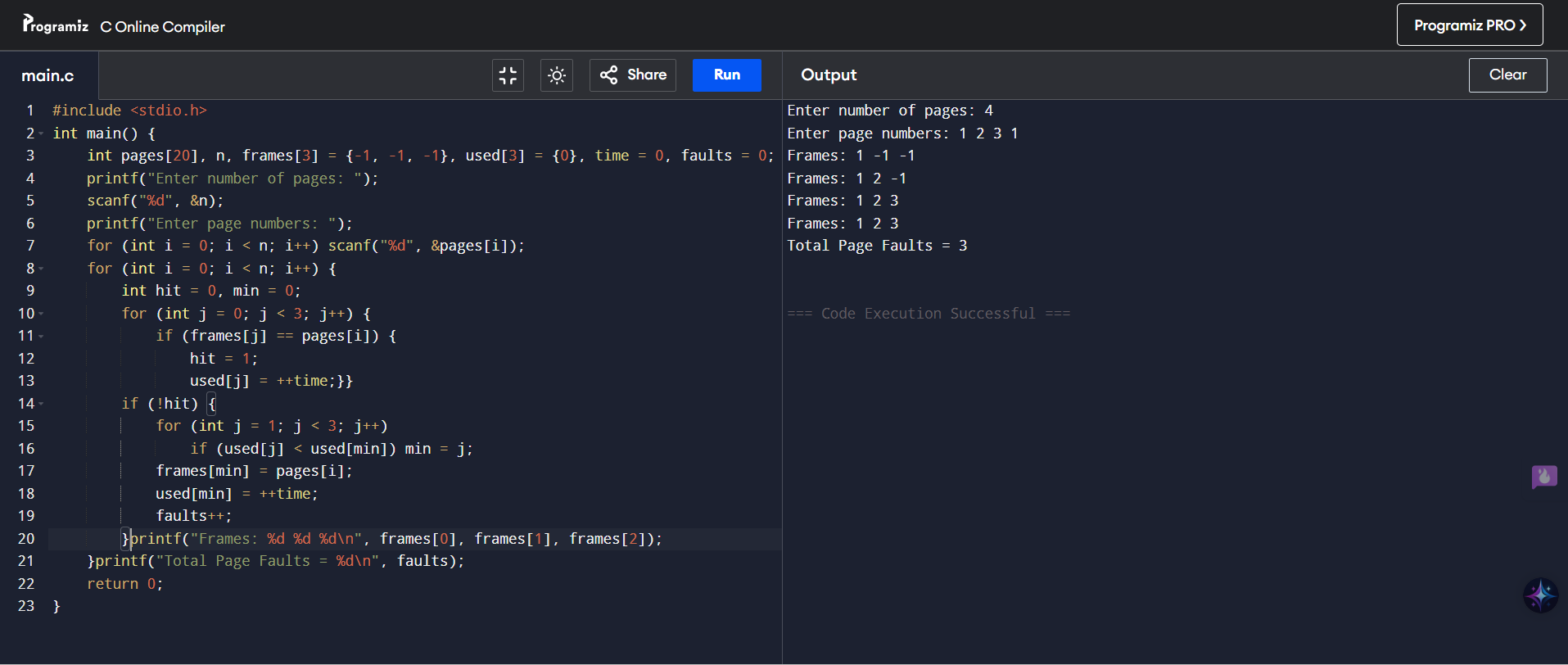
Frames: 1 -1 -1

Frames: 1 2 -1

Frames: 1 2 3

Frames: 1 2 3

Total Page Faults = 3

****

**RESULT:**

LRU paging simulation program executed successfully.

33. Construct a C program to simulate the optimal paging technique of memory management

**AIM:**

To simulate Optimal page replacement using C by predicting which page won't be used for the longest time in the future and replacing that page.

**ALGORITHM:**

1. Initialize empty frames.
2. For each page:

* If in frame → hit.
* Else → Replace page not used soonest in future.

1. Count page faults.
2. Print total faults.

**CODE:**

#include <stdio.h>

int predict(int p[], int f[], int n, int idx) {

int far = -1, pos = -1;

for (int i = 0; i < 3; i++) {

int j;

for (j = idx; j < n; j++)

if (f[i] == p[j]) break;

if (j == n) return i;

if (j > far) far = j, pos = i;

}

return pos;

}

int main() {

int p[20], f[3] = {-1, -1, -1}, n, i, j, hit, pos, faults = 0;

printf("Pages: "); scanf("%d", &n);

for (i = 0; i < n; i++) scanf("%d", &p[i]);

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

hit = 0;

for (j = 0; j < 3; j++)

if (f[j] == p[i]) hit = 1;

if (!hit) {

pos = (f[0] == -1 || f[1] == -1 || f[2] == -1) ?

(f[0] == -1 ? 0 : f[1] == -1 ? 1 : 2) :

predict(p, f, n, i+1);

f[pos] = p[i]; faults++;

}

printf("Frames: %d %d %d\n", f[0], f[1], f[2]);

}

printf("Faults: %d\n", faults);

}

**SAMPLE INPUT:**

Pages: 4

7 0 1 2

**SAMPLE OUTPUT:**

Frames: 7 -1 -1

Frames: 7 0 -1

Frames: 7 0 1

Frames: 2 0 1

Faults: 4

**A screenshot of a computer

AI-generated content may be incorrect.**

**RESULT:**

Optimal paging simulation program executed successfully.

34. Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a C program to simulate the file allocation strategy.

**AIM:**

To simulate sequential file allocation using a simple C program.

**ALGORITHM:**

1. Input number of files and their starting blocks + lengths.
2. Store blocks sequentially.
3. To access a record, read from the starting block up to the desired record.

**CODE:**

#include <stdio.h>

struct File {

int start, length;

};

int main() {

int n, i, j;

struct File f[10];

printf("Enter number of files: ");

scanf("%d", &n);

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

printf("File %d start block & length: ", i + 1);

scanf("%d %d", &f[i].start, &f[i].length);

}

printf("\nFile\tBlocks\n");

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

printf("%d\t", i + 1);

for (j = 0; j < f[i].length; j++)

printf("%d ", f[i].start + j);

printf("\n");

}

return 0;

}

**SAMPLE INPUT:**

Enter number of files: 2

File 1 start block & length: 5 3

File 2 start block & length: 10 2

**SAMPLE OUTPUT:**

File Blocks

1 5 6 7

2 10 11

**A screenshot of a computer

AI-generated content may be incorrect.**

**RESULT:**

Sequential file allocation program executed successfully.

35. Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a C program to simulate the file allocation strategy.

**AIM:**

To simulate Indexed File Allocation where the index block stores pointers to all the blocks of a file.

**ALGORITHM:**

1. For each file:

* Input the index block and block numbers used by the file.

1. Store and display:

* The index block and all its entries (pointers to file blocks).

**CODE:**

#include <stdio.h>

int main() {

int files, i, j, blocks, indexBlock, dataBlock[10];

printf("Enter number of files: ");

scanf("%d", &files);

for (i = 0; i < files; i++) {

printf("Enter index block for file %d: ", i + 1);

scanf("%d", &indexBlock);

printf("Enter number of blocks for file %d: ", i + 1);

scanf("%d", &blocks);

printf("Enter block numbers: ");

for (j = 0; j < blocks; j++)

scanf("%d", &dataBlock[j]);

printf("\nFile %d => Index Block: %d => Blocks: ", i + 1, indexBlock);

for (j = 0; j < blocks; j++)

printf("%d ", dataBlock[j]);

printf("\n\n");

}

return 0;

}

**SAMPLE INPUT:**

Enter number of files: 2

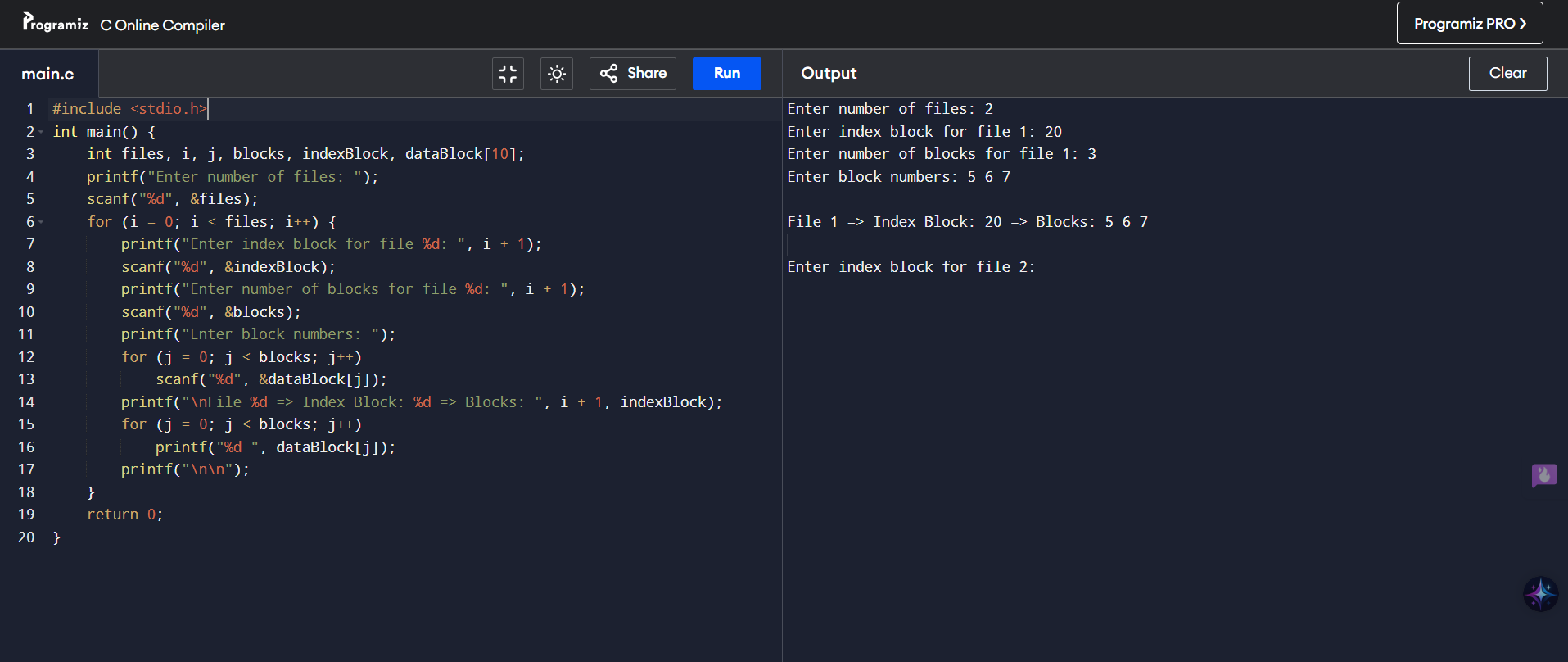
Enter index block for file 1: 20

Enter number of blocks for file 1: 3

Enter block numbers: 5 6 7

**SAMPLE OUTPUT:**

File 1 => Index Block: 20 => Blocks: 5 6 7

****

**RESULT:**

Indexed file allocation strategy program executed successfully.

36. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a C program to simulate the file allocation strategy.

**AIM:**

To simulate Linked Allocation where files are scattered blocks connected by pointers.

**ALGORITHM:**

1. For each file, input:

* Start and end block
* Number of blocks
* Sequence of blocks linked together

1. Display each file’s linked block path.

**CODE:**

#include <stdio.h>

int main() {

int files, i, j, blocks, start, end, chain[30];

printf("Enter number of files: ");

scanf("%d", &files);

for (i = 0; i < files; i++) {

printf("\nFile %d:\n", i + 1);

printf("Enter start and end blocks: ");

scanf("%d %d", &start, &end);

printf("Enter number of blocks: ");

scanf("%d", &blocks);

printf("Enter block chain: ");

for (j = 0; j < blocks; j++)

scanf("%d", &chain[j]);

printf("File %d => Start: %d, End: %d => Chain: ", i + 1, start, end);

for (j = 0; j < blocks; j++)

printf("%d -> ", chain[j]);

printf("NULL\n");

}

return 0;

}

**SAMPLE INPUT:**

Enter number of files: 1

File 1:

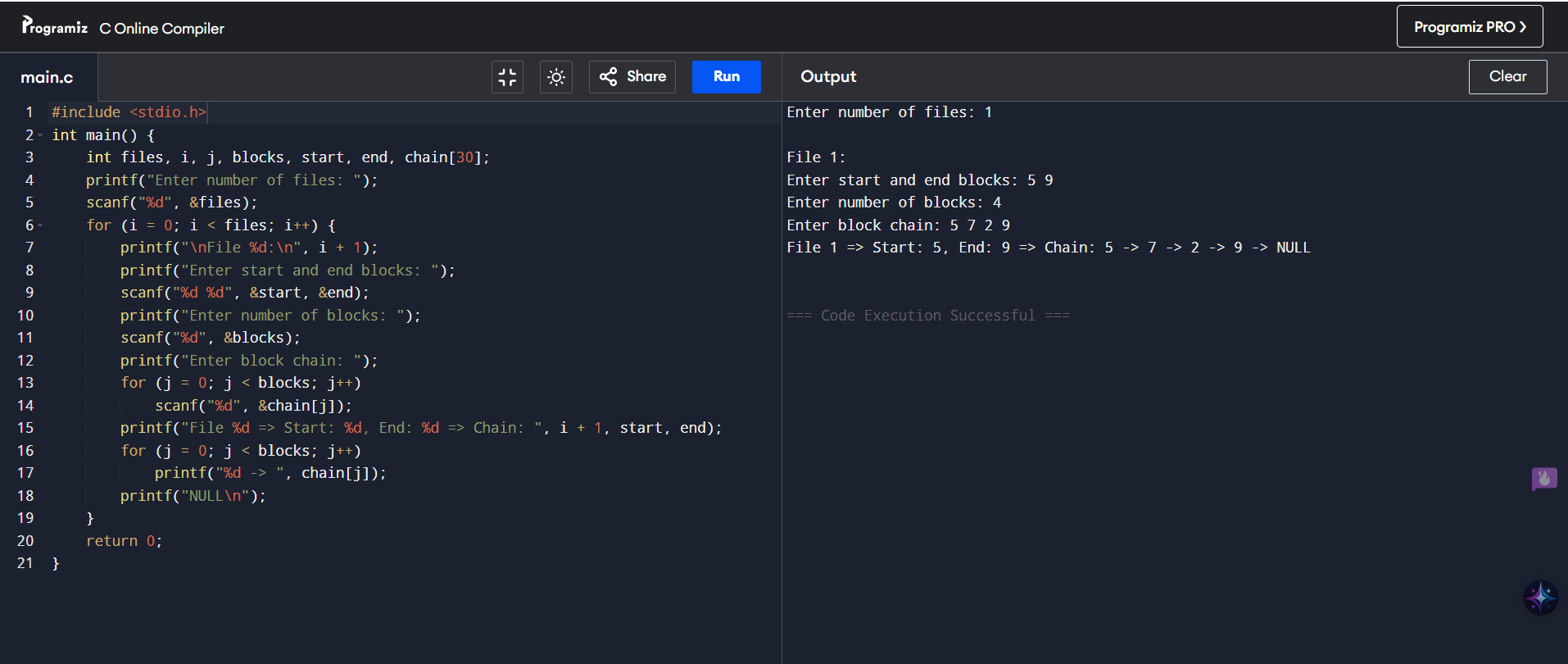
Enter start and end blocks: 5 9

Enter number of blocks: 4

Enter block chain: 5 7 2 9

**SAMPLE OUTPUT:**

File 1 => Start: 5, End: 9 => Chain: 5 -> 7 -> 2 -> 9 -> NULL

****

**RESULT:**

Linked file allocation strategy program executed successfully.

37. .Construct a C program to simulate the First Come First Served disk scheduling algorithm

**AIM:**

To simulate FCFS Disk Scheduling, where disk requests are handled in the order they arrive.

**ALGORITHM:**

1. Input number of requests and their positions.
2. Input initial head position.
3. For each request (in order), calculate the seek time.
4. Print total and average seek time.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, i, head, total = 0;

int req[100];

printf("Enter number of requests: ");

scanf("%d", &n);

printf("Enter request sequence: ");

for (i = 0; i < n; i++) scanf("%d", &req[i]);

printf("Enter initial head position: ");

scanf("%d", &head);

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

total += abs(req[i] - head);

head = req[i];

}

printf("Total seek time: %d\n", total);

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

return 0;

}

**SAMPLE INPUT:**

Enter number of requests: 5

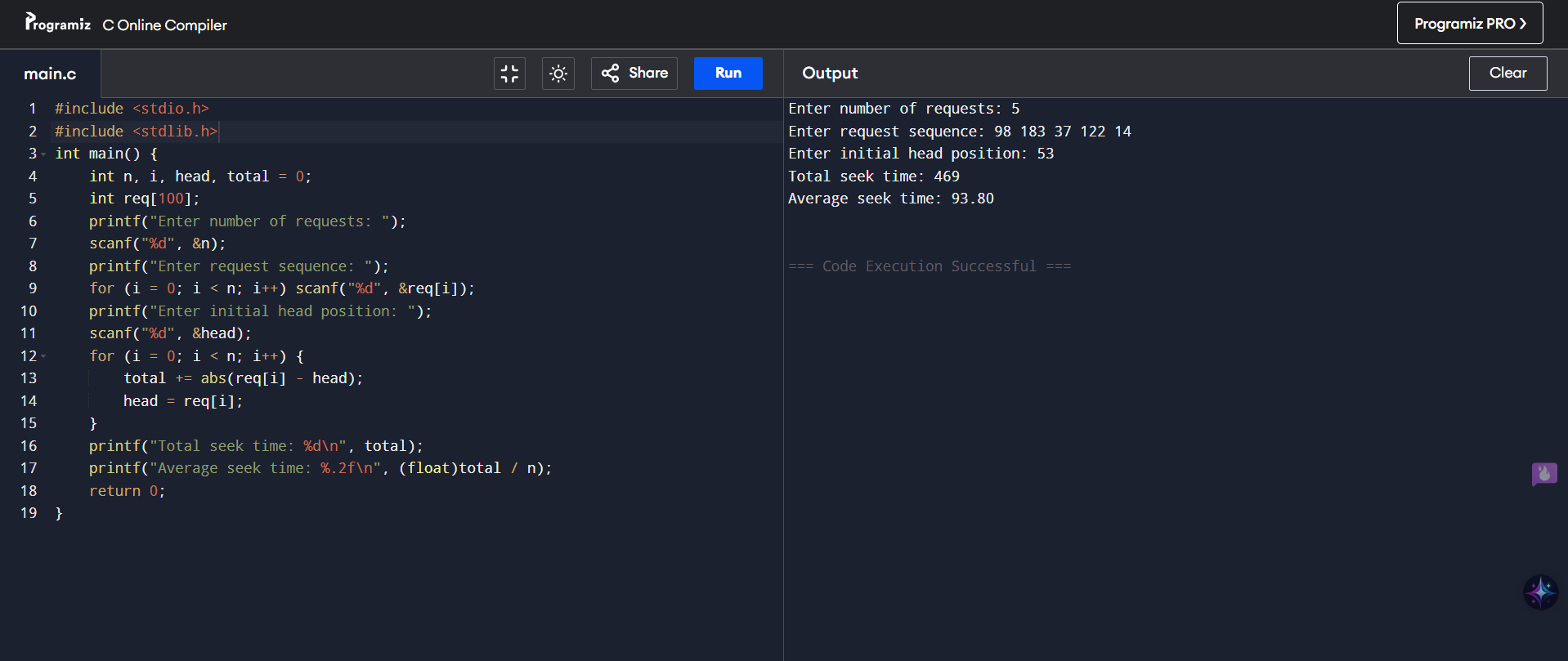
Enter request sequence: 98 183 37 122 14

Enter initial head position: 53

**SAMPLE OUTPUT:**

Total seek time: 640

Average seek time: 128.00

****

**RESULT:**

First Come First Served disk scheduling program executed successfully.

38. Design a C program to simulate SCAN disk scheduling algorithm.

**AIM:**

To simulate the SCAN disk scheduling algorithm where the disk head moves in one direction (like an elevator), then reverses.

**ALGORITHM:**

1. Input request queue and initial head position.
2. Input direction (toward 0 or toward max disk size).
3. Sort requests.
4. Move in the given direction, servicing all requests.
5. Reverse direction and service remaining.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int cmp(const void \*a, const void \*b) { return (\*(int\*)a - \*(int\*)b); }

int main() {

int r[100], n, h, i, d, t = 0;

printf("Requests: "); scanf("%d", &n);

for (i = 0; i < n; i++) scanf("%d", &r[i]);

printf("Head pos: "); scanf("%d", &h);

printf("Dir (0=left,1=right): "); scanf("%d", &d);

r[n++] = h;

qsort(r, n, sizeof(int), cmp);

int p; for (i = 0; i < n; i++) if (r[i] == h) { p = i; break; }

printf("Seq: ");

if (d) {

for (i = p; i < n; i++) t += abs(h - (h = r[i])), printf("%d ", r[i]);

for (i = p - 1; i >= 0; i--) t += abs(h - (h = r[i])), printf("%d ", r[i]);

} else {

for (i = p; i >= 0; i--) t += abs(h - (h = r[i])), printf("%d ", r[i]);

for (i = p + 1; i < n; i++) t += abs(h - (h = r[i])), printf("%d ", r[i]);

}

printf("\nTotal: %d\nAvg: %.2f\n", t, (float)t / (n - 1));

return 0;

}

**SAMPLE INPUT:**

Requests: 5

98 183 37 122 14

Head pos: 53

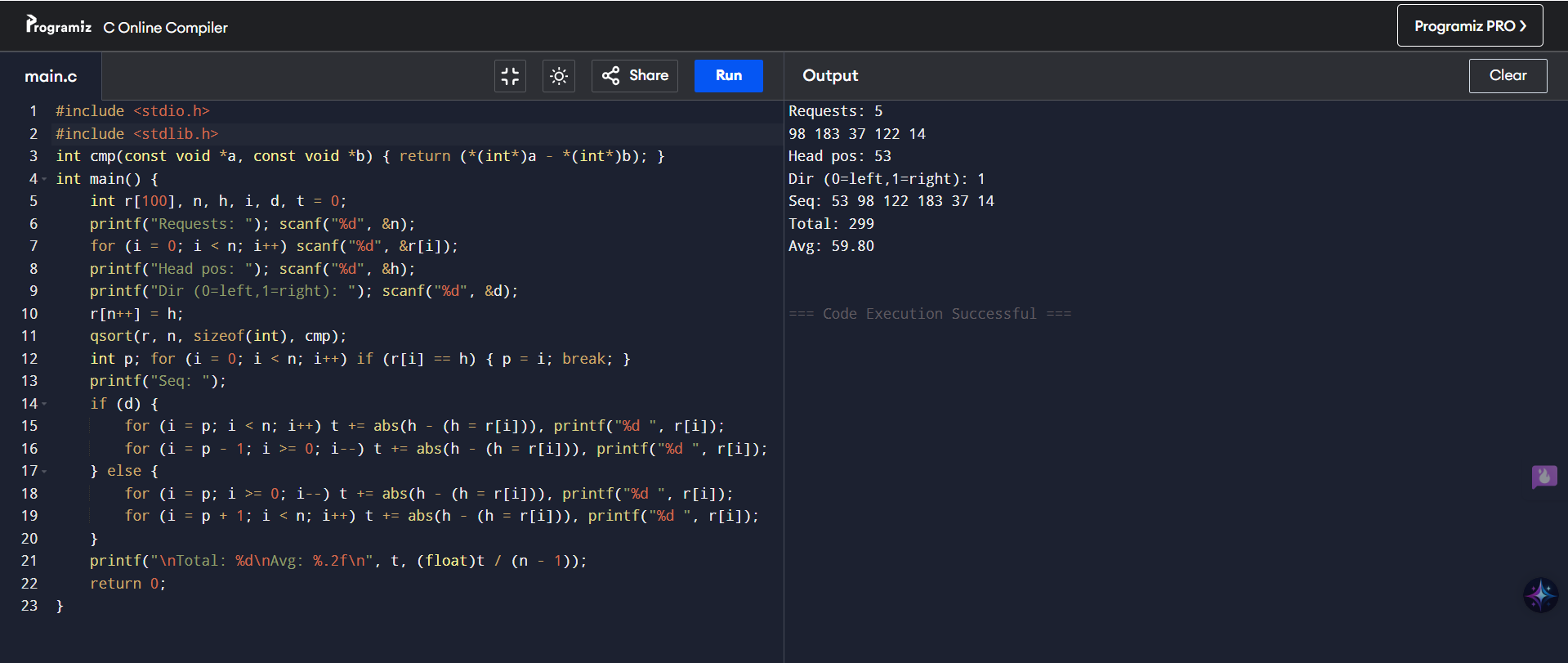
Dir (0=left,1=right): 1

**SAMPLE OUTPUT:**

Seq: 53 98 122 183 37 14

Total: 322

Avg: 64.40

****

**RESULT:**

SCAN disk scheduling algorithm program executed successfully.

39. Develop a C program to simulate C-SCAN disk scheduling algorithm.

**AIM:**

To simulate the C-SCAN (Circular SCAN) disk scheduling algorithm where the disk head moves in one direction only (like SCAN), but instead of reversing, it jumps to the beginning and continues.

**ALGORITHM:**

1. Input requests and initial head position.
2. Add head to the request queue.
3. Sort the request queue.
4. Move right from the head to the end, then jump to start and continue.
5. Calculate total and average seek time.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int cmp(const void \*a, const void \*b) { return (\*(int\*)a - \*(int\*)b); }

int main() {

int r[100], n, i, head, size, pos, t = 0;

printf("Requests: "); scanf("%d", &n);

for (i = 0; i < n; i++) scanf("%d", &r[i]);

printf("Disk size & Head: "); scanf("%d %d", &size, &head);

r[n++] = head;

qsort(r, n, sizeof(int), cmp);

for (i = 0; i < n; i++) if (r[i] == head) { pos = i; break; }

printf("Sequence: ");

for (i = pos; i < n; i++) { printf("%d ", r[i]); t += abs(head - r[i]); head = r[i]; }

if (pos > 0) {

t += abs(head - (size - 1)) + (size - 1); head = 0;

for (i = 0; i < pos; i++) { printf("%d ", r[i]); t += abs(head - r[i]); head = r[i]; }

}

printf("\nTotal Seek: %d\nAvg Seek: %.2f\n", t, (float)t / (n - 1));

return 0;

}

**SAMPLE INPUT:**

Requests: 5

95 180 34 119 11

Disk size & Head: 200 50

**SAMPLE OUTPUT:**

Sequence: 50 95 119 180 11 34

Total Seek: 391

Avg Seek: 78.20

**A screenshot of a computer program

AI-generated content may be incorrect.**

**RESULT:**

C-SCAN disk scheduling algorithm program executed successfully.

40. Illustrate the various File Access Permission and different types users in Linux.

**AIM:**

To illustrate file access permissions and user types in Linux.

**ALGORITHM:**

1. Create a file.
2. Use ls -l to view permissions.
3. Use chmod to modify permissions.
4. Check the result again using ls -l.

**CODE:**

#include <stdio.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <unistd.h>

int main() {

int fd = creat("myfile.txt", 0644); // rw-r--r--

if (fd < 0) {

perror("File creation failed");

return 1;

}

close(fd);

if (chmod("myfile.txt", 0754) == 0)

printf("Permissions changed to 754 (rwxr-xr--)\n");

else

perror("chmod failed");

return 0;

}

**SAMPLE OUTPUT:**

Permissions changed to 754 (rwxr-xr--)

A screenshot of a computer

AI-generated content may be incorrect.

**RESULT:**

File access permission and Linux user type program executed successfully.