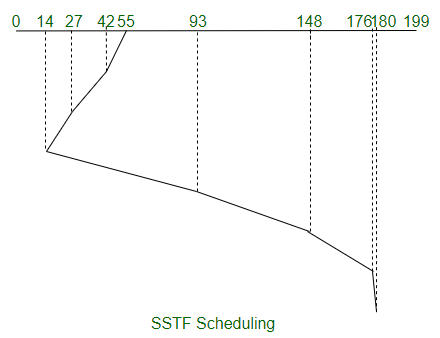
**Experiment 8**

**SSTF**

SSTF is abbreviation of [Shortest Seek Time First (SSTF)](https://www.geeksforgeeks.org/program-for-sstf-disk-scheduling-algorithm/) which is a [disk scheduling algorithm](https://www.geeksforgeeks.org/disk-scheduling-algorithms/).

It selects the request which is closest to the current head position before moving the head away to service other requests. This is done by selecting the request which has the least seek time from the current head position.

**Example:**  
Consider a disk with 200 tracks (0-199) and the disk queue having I/O requests in the following order as follows: 93, 176, 42, 148, 27, 14, 180. The current head position of the Read\Write head is 55. Calculate the total number of track movements of Read/Write head using SSTF.



Total Seek Time,

= (55-14) + (180-14)

= 207

**Advantages:**

1. Throughput is better than FCFS.
2. Average Response Time is minimized due to less number of head movements.

**SSTF program:**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

scanf("%d",&RQ[i]);

printf("Enter initial head position\n");

scanf("%d",&initial);

// logic for sstf disk scheduling

/\* loop will execute until all process is completed\*/

while(count!=n)

{

int min=1000,d,index;

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

{

d=abs(RQ[i]-initial);

if(min>d)

{

min=d;

index=i;

}

}

TotalHeadMoment=TotalHeadMoment+min;

initial=RQ[index];

// 1000 is for max

// you can use any number

RQ[index]=1000;

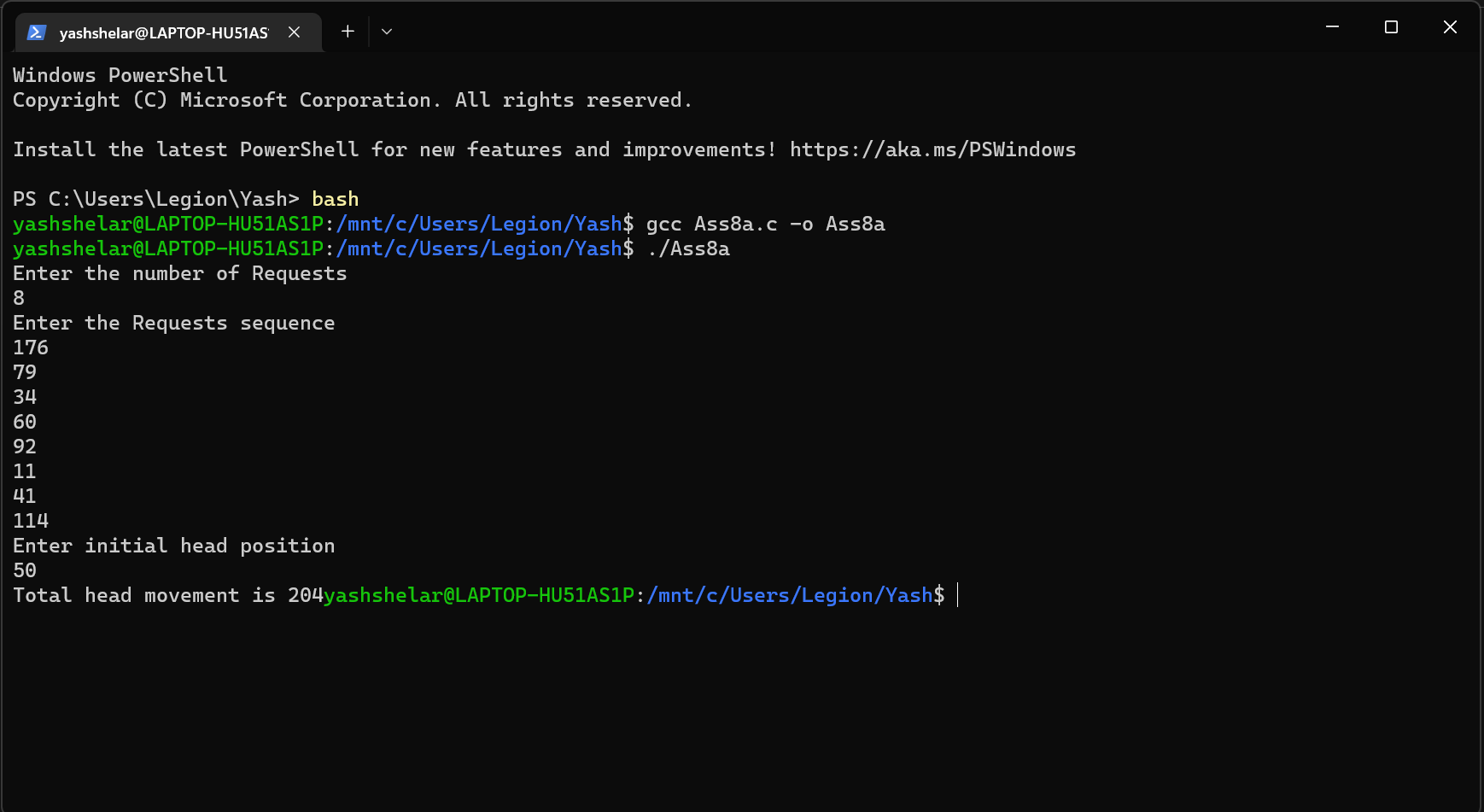
count++;

}

printf("Total head movement is %d",TotalHeadMoment);

return 0;

}



**SCAN(Elevator)algorithm**   
In SCAN disk scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk. So, this algorithm works as an elevator and hence also known as the **elevator algorithm**. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

**Advantages of SCAN (Elevator) algorithm**

1. This algorithm is simple and easy to understand.
2. SCAN algorithm have no starvation.
3. This algorithm is better than FCFS Scheduling algorithm .

**Example:**

**Input:**

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

Initial head position = 50

Direction = left (We are moving from right to left)

**Output:**

Total number of seek operations = 226

Seek Sequence is

41

34

11

0

60

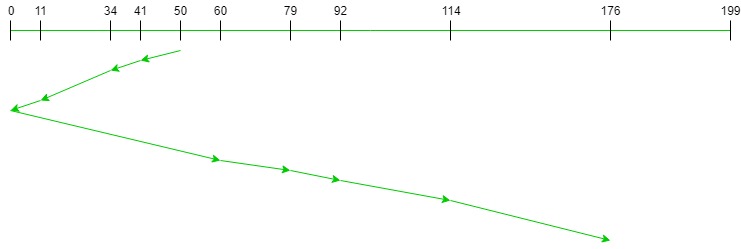
79

92

114

176

The following chart shows the sequence in which requested tracks are serviced using SCAN. 



Therefore, the total seek count is calculated as:

= (50-41)+(41-34)+(34-11)

+(11-0)+(60-0)+(79-60)

+(92-79)+(114-92)+(176-114)

= 226

**Scan algorithm**

#include<conio.h>

#include<stdio.h>

int main()

{

int i,j,sum=0,n;

int d[20];

int disk; //loc of head

int temp,max;

int dloc; //loc of disk in array

printf("enter number of location\t");

scanf("%d",&n);

printf("enter position of head\t");

scanf("%d",&disk);

printf("enter elements of disk queue\n");

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

{

scanf("%d",&d[i]);

}

d[n]=disk;

n=n+1;

for(i=0;i<n;i++) // sorting disk locations

{

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

{

if(d[i]>d[j])

{

temp=d[i];

d[i]=d[j];

d[j]=temp;

}

}

}

max=d[n];

for(i=0;i<n;i++) // to find loc of disc in array

{

if(disk==d[i]) { dloc=i; break; }

}

for(i=dloc;i>=0;i--)

{

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

}

printf("0 -->");

for(i=dloc+1;i<n;i++)

{

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

}

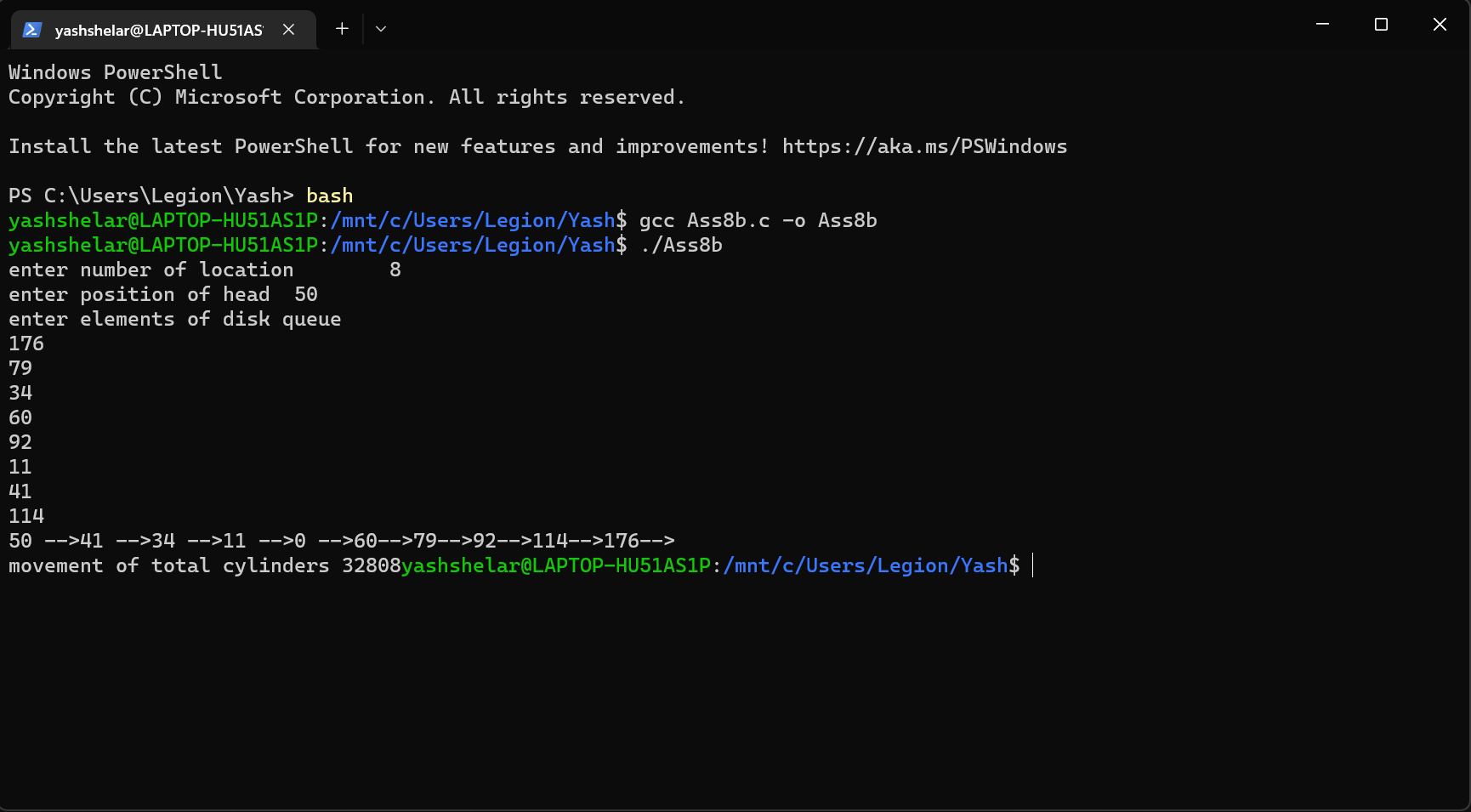
sum=disk+max;

printf("\nmovement of total cylinders %d",sum);

getch();

return 0;

}



**C-LOOK (Circular LOOK) Disk Scheduling Algorithm:**   
**C-LOOK** is an enhanced version of both **SCAN** as well as **LOOK** disk scheduling algorithms.

In this algorithm, the head services requests only in one direction (either left or right) until all the requests in this direction are not serviced and then jumps back to the farthest request on the other direction and service the remaining requests which gives a better uniform servicing as well as avoids wasting seek time for going till the end of the disk.

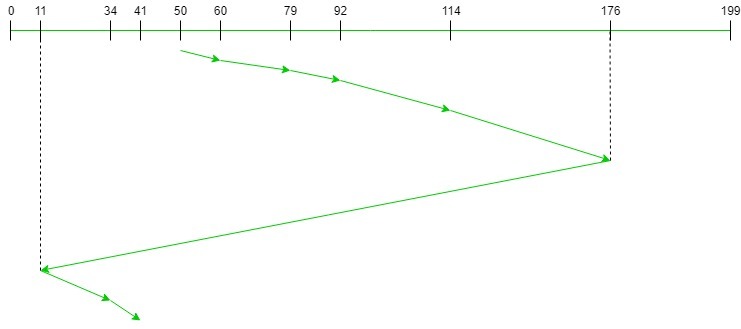
**Algorithm-**

1. Let Request array represents an array storing indexes of the tracks that have been requested in ascending order of their time of arrival and **head** is the position of the disk head.
2. The initial direction in which the head is moving is given and it services in the same direction.
3. The head services all the requests one by one in the direction it is moving.
4. The head continues to move in the same direction until all the requests in this direction have been serviced.
5. While moving in this direction, calculate the absolute distance of the tracks from the head.
6. Increment the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.
8. Go to step 5 until we reach the last request in this direction.
9. If we reach the last request in the current direction then reverse the direction and move the head in this direction until we reach the last request that is needed to be serviced in this direction without servicing the intermediate requests.
10. Reverse the direction and go to step 3 until all the requests have not been serviced.

**Examples:**

**Input:**   
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}   
Initial head position = 50   
Direction = right (Moving from left to right)   
**Output:**   
Initial position of head: 50   
Total number of seek operations = 156   
Seek Sequence is   
60   
79   
92   
114   
176   
11   
34   
41 

The following chart shows the sequence in which requested tracks are serviced using C-LOOK.



Therefore, the total seek count = (60 – 50) + (79 – 60) + (92 – 79) + (114 – 92) + (176 – 114) + (176 – 11) + (34 – 11) + (41 – 34) = 321