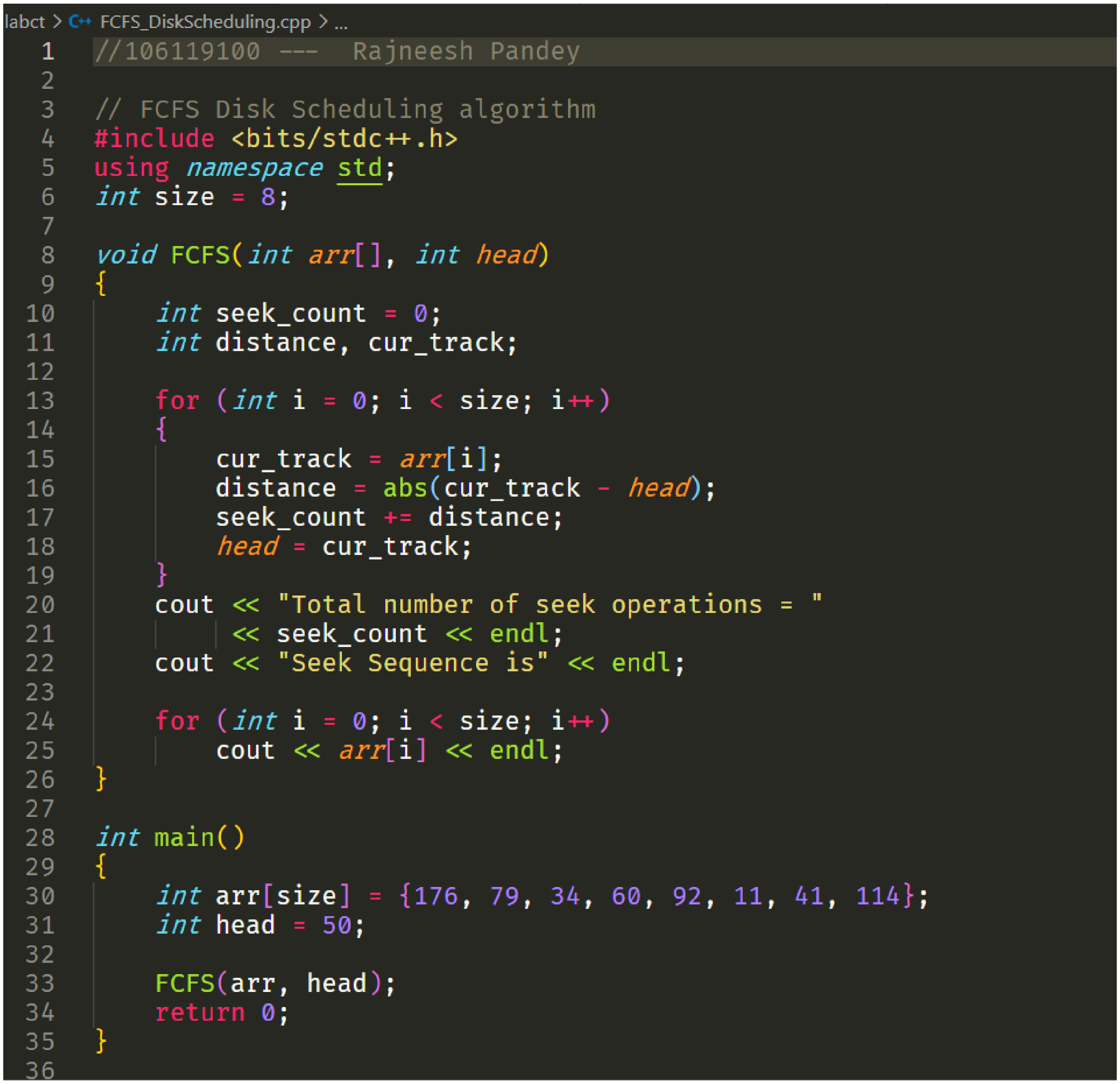
OS Lab - 10 (CSLR42) -27/04/2021

106119100 - Rajneesh Pandey

---

1. FCFS Disk Scheduling Algorithm

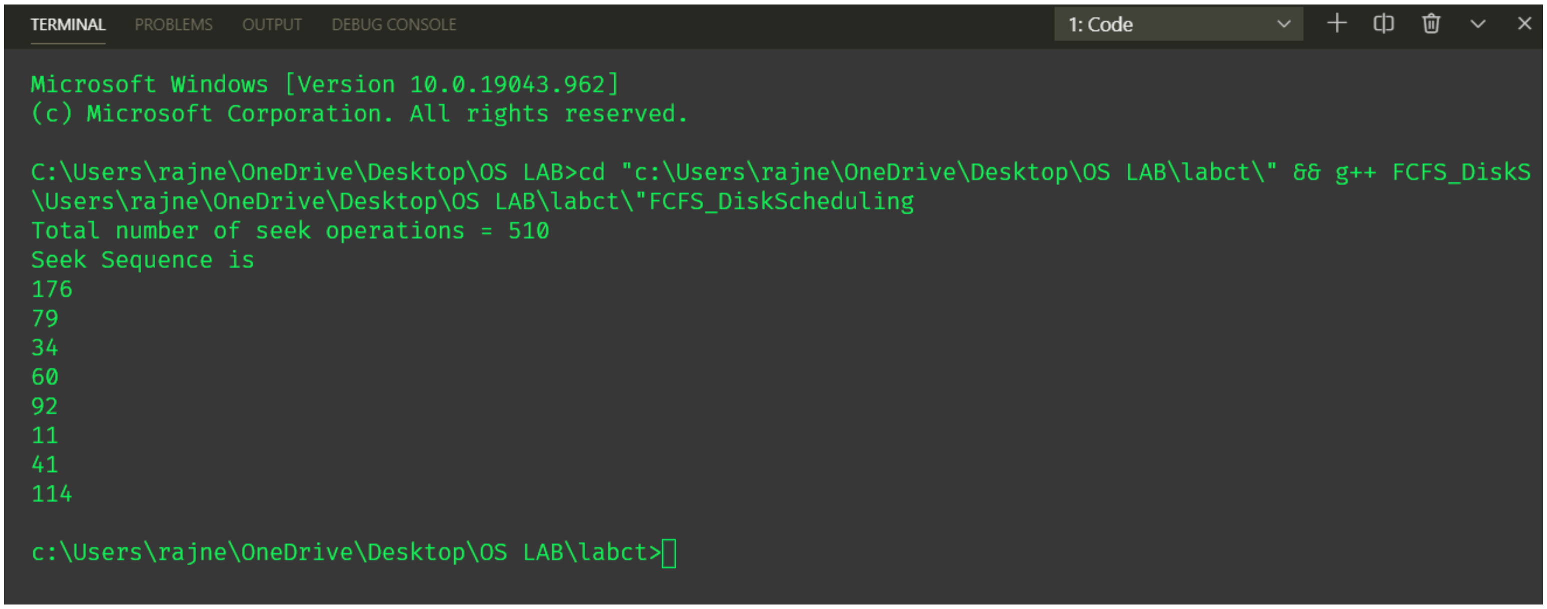
FCFS is the simplest disk scheduling algorithm. As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue. The algorithm looks very fair and there is no starvation (all requests are serviced sequentially) but generally, it does not provide the fastest service.



labct > FCFS_DiskScheduling.cpp > ...

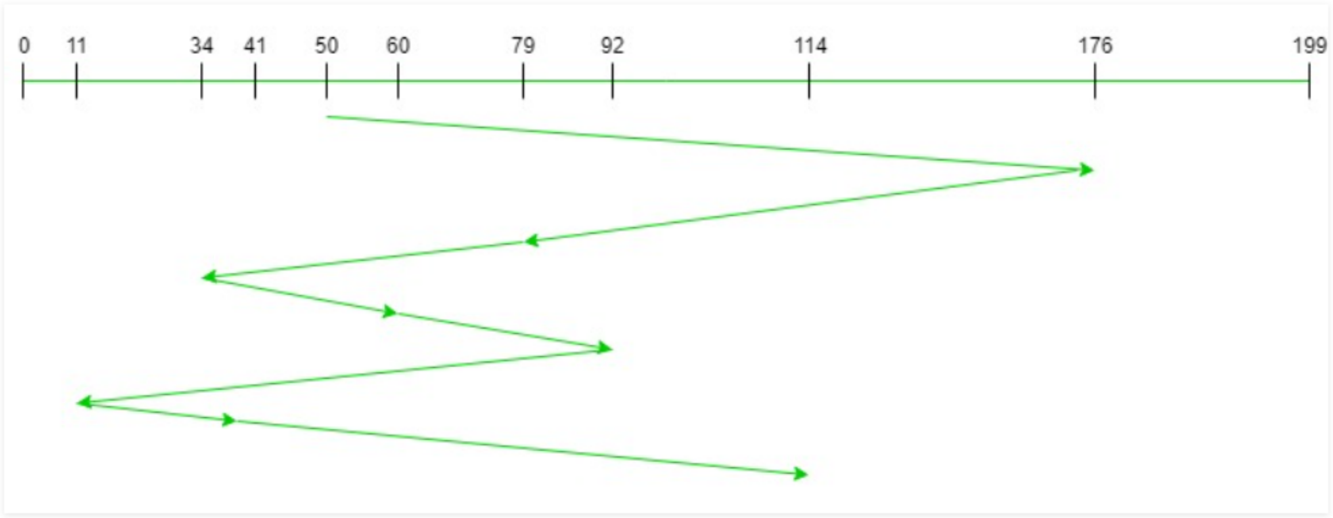
```cpp
//106119100 ---  Rajneesh Pandey

// FCFS Disk Scheduling algorithm
#include <bits/stdc++.h>
using namespace std;
int size = 8;

void FCFS(int arr[], int head)
{
    int seek_count = 0;
    int distance, cur_track;

    for (int i = 0; i < size; i++)
    {
        cur_track = arr[i];
        distance = abs(cur_track - head);
        seek_count += distance;
        head = cur_track;
    }
    cout << "Total number of seek operations = "
         << seek_count << endl;
    cout << "Seek Sequence is" << endl;

    for (int i = 0; i < size; i++)
        cout << arr[i] << endl;
}

int main()
{
    int arr[size] = {176, 79, 34, 60, 92, 11, 41, 114};
    int head = 50;

    FCFS(arr, head);
    return 0;
}

```

Input / Output



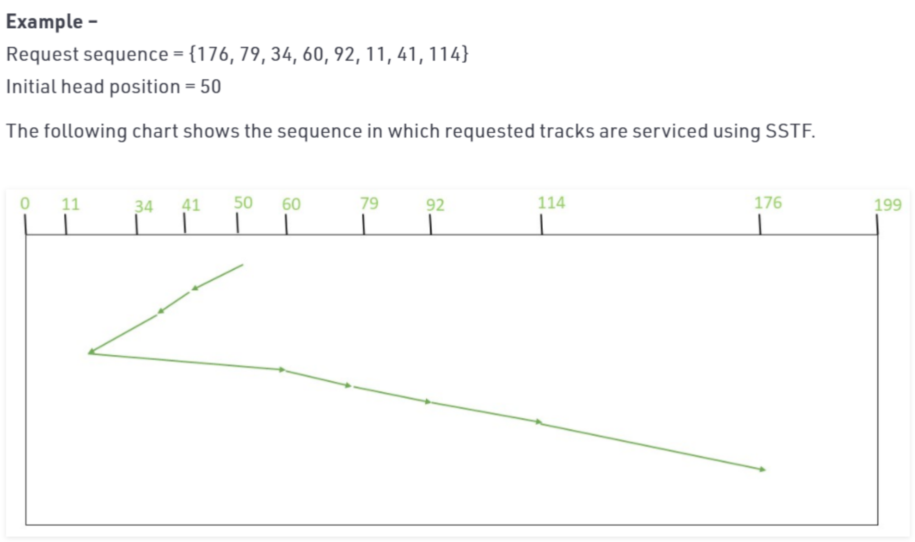
```
Microsoft Windows [Version 10.0.19043.962]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rajne\OneDrive\Desktop\OS LAB>cd "c:\Users\rajne\OneDrive\Desktop\OS LAB\labct\" && g++ FCFS_DiskS
\Users\rajne\OneDrive\Desktop\OS LAB\labct\"FCFS_DiskScheduling
Total number of seek operations = 510
Seek Sequence is
176
79
34
60
92
11
41
114

c:\Users\rajne\OneDrive\Desktop\OS LAB\labct>
```

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



2. SSTF Disk Scheduling Algorithm

Given an array of disk track numbers and initial head position, our task is to find the total number of seek operations done to access all the requested tracks if Shortest Seek Time First (SSTF) is a disk scheduling algorithm is used.
Basic idea is the tracks which are closer to current disk head position should be serviced first in order to minimize the seek operations.

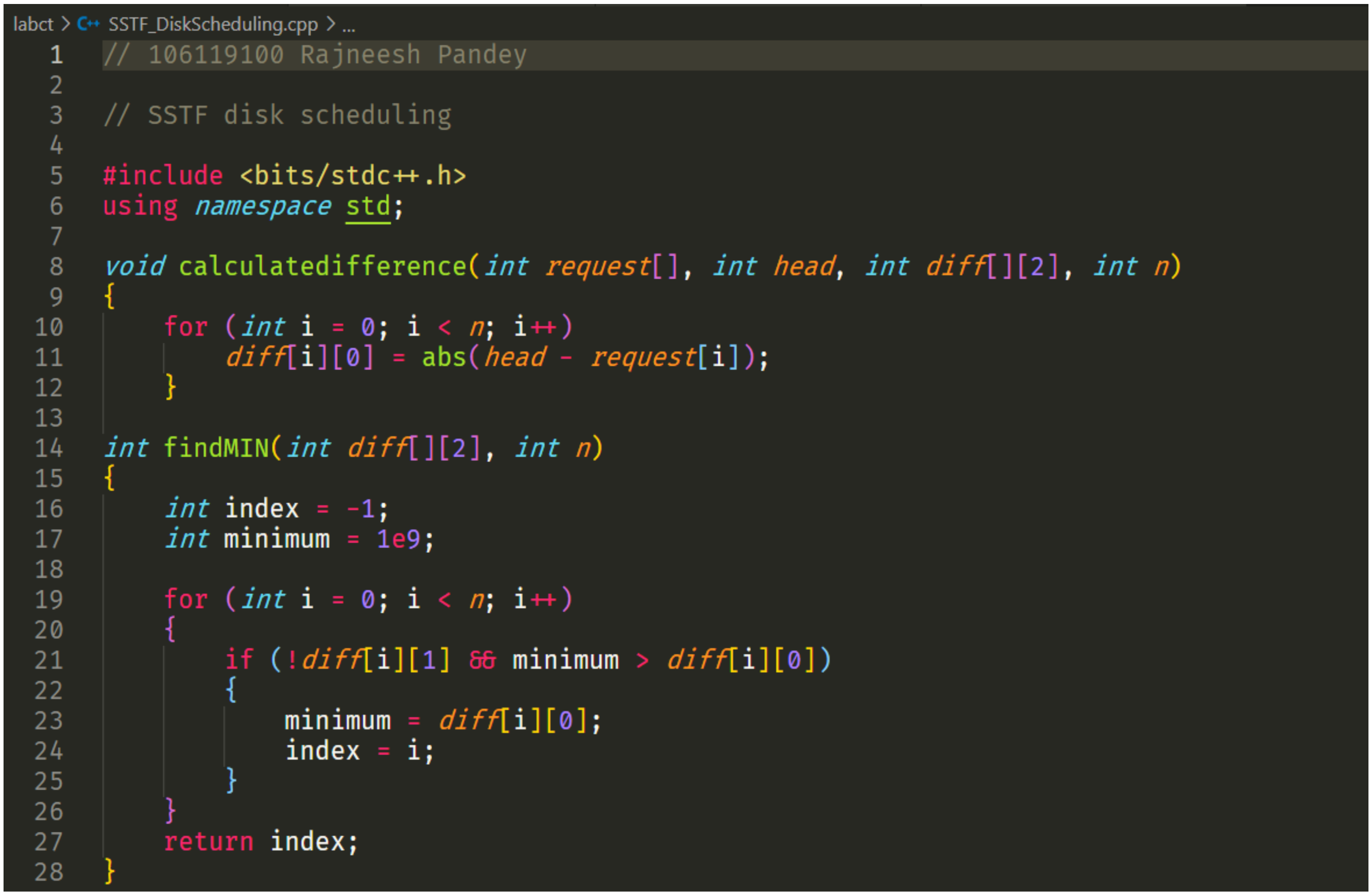


Example –

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

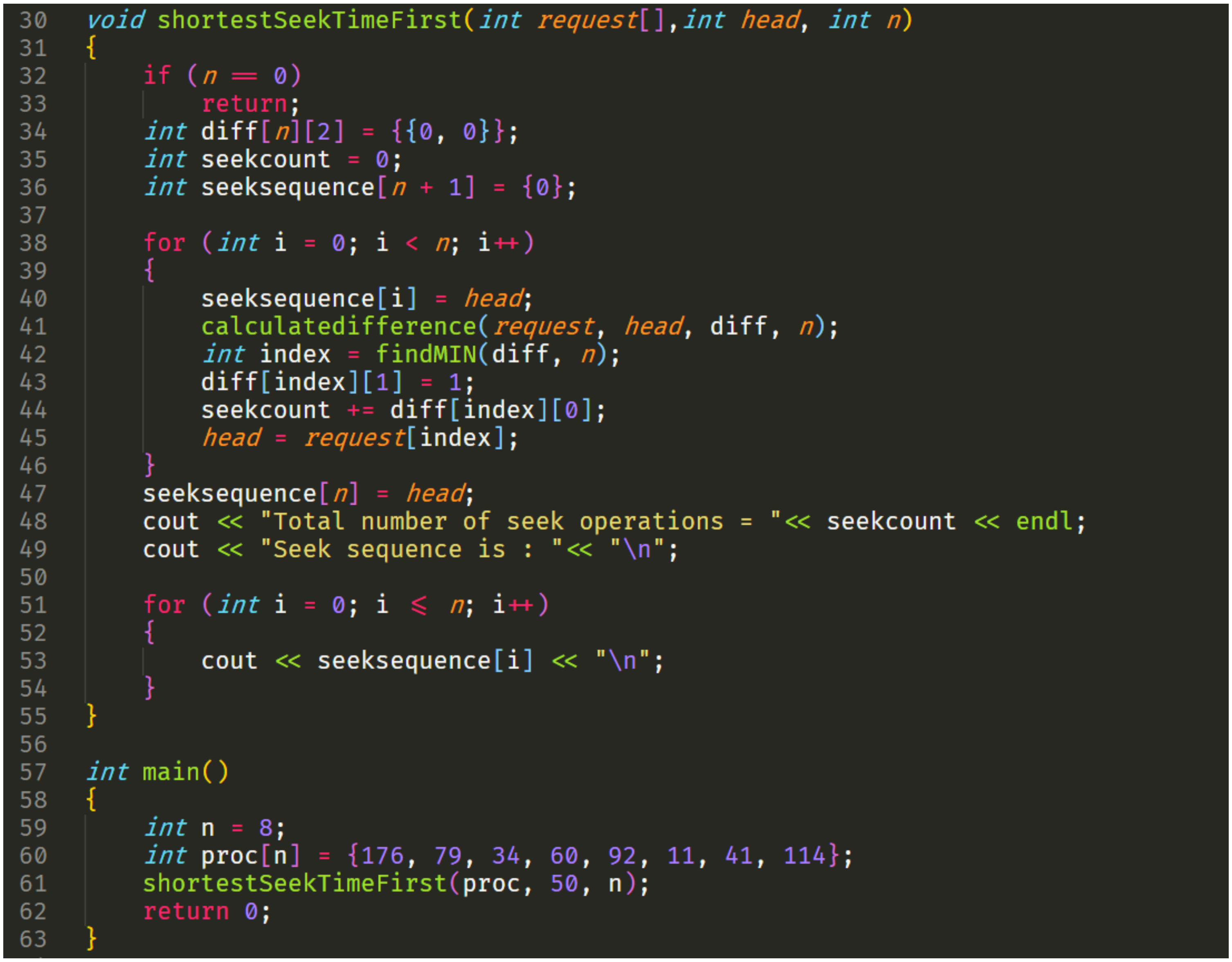
Initial head position = 50

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



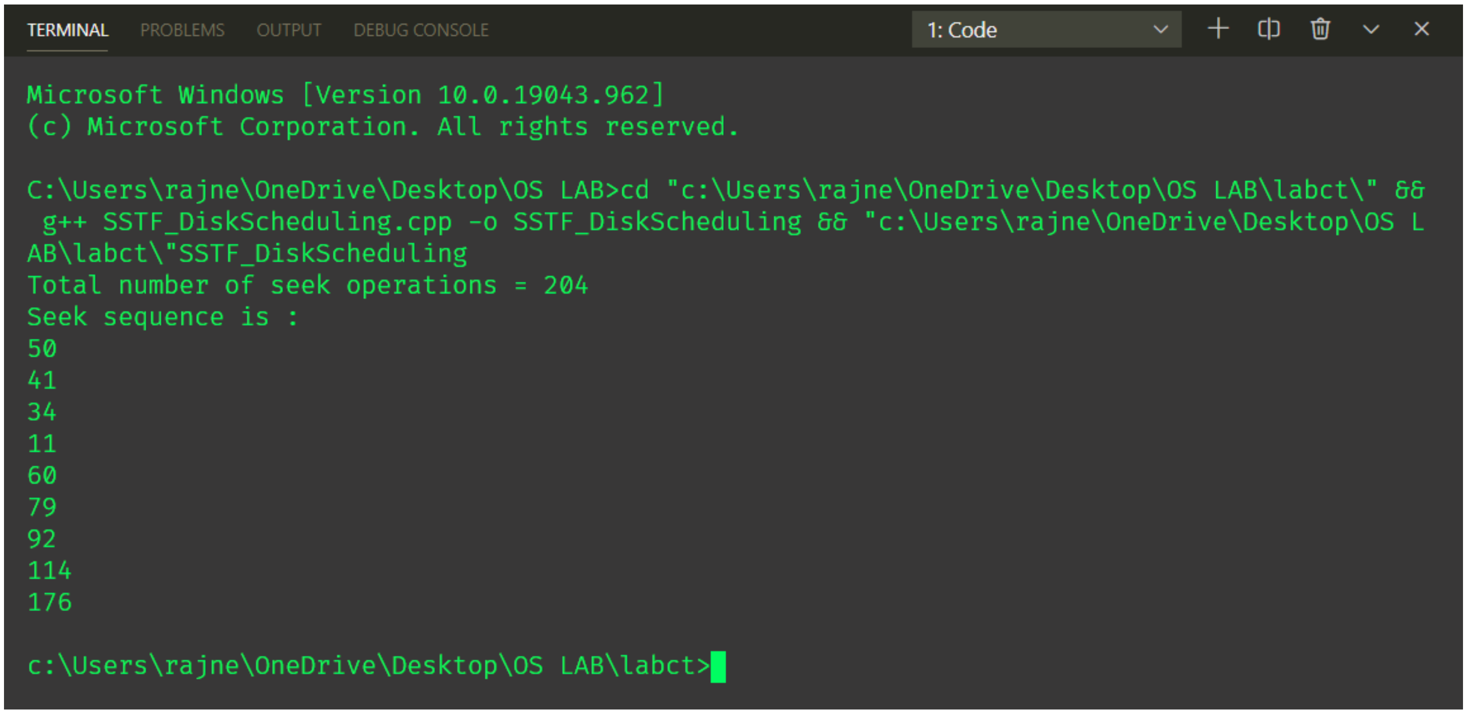
labct > SSTF_DiskScheduling.cpp > ...

```cpp
// 106119100 Rajneesh Pandey

// SSTF disk scheduling

#include <bits/stdc++.h>
using namespace std;

void calculatedifference(int request[], int head, int diff[][2], int n)
{
    for (int i = 0; i < n; i++)
        diff[i][0] = abs(head - request[i]);
    }

int findMIN(int diff[][2], int n)
{
    int index = -1;
    int minimum = 1e9;

    for (int i = 0; i < n; i++)
    {
        if (!diff[i][1] && minimum > diff[i][0])
        {
            minimum = diff[i][0];
            index = i;
        }
    }
    return index;
}
```



```cpp
void shortestSeekTimeFirst(int request[],int head, int n)
{
    if (n == 0)
        return;
    int diff[n][2] = {{0, 0}};
    int seekcount = 0;
    int seeksequence[n + 1] = {0};

    for (int i = 0; i < n; i++)
    {
        seeksequence[i] = head;
        calculatedifference(request, head, diff, n);
        int index = findMIN(diff, n);
        diff[index][1] = 1;
        seekcount += diff[index][0];
        head = request[index];
    }
    seeksequence[n] = head;
    cout << "Total number of seek operations = "<< seekcount << endl;
    cout << "Seek sequence is : "<< "\n";

    for (int i = 0; i <= n; i++)
    {
        cout << seeksequence[i] << "\n";
    }
}

int main()
{
    int n = 8;
    int proc[n] = {176, 79, 34, 60, 92, 11, 41, 114};
    shortestSeekTimeFirst(proc, 50, n);
    return 0;
}
```

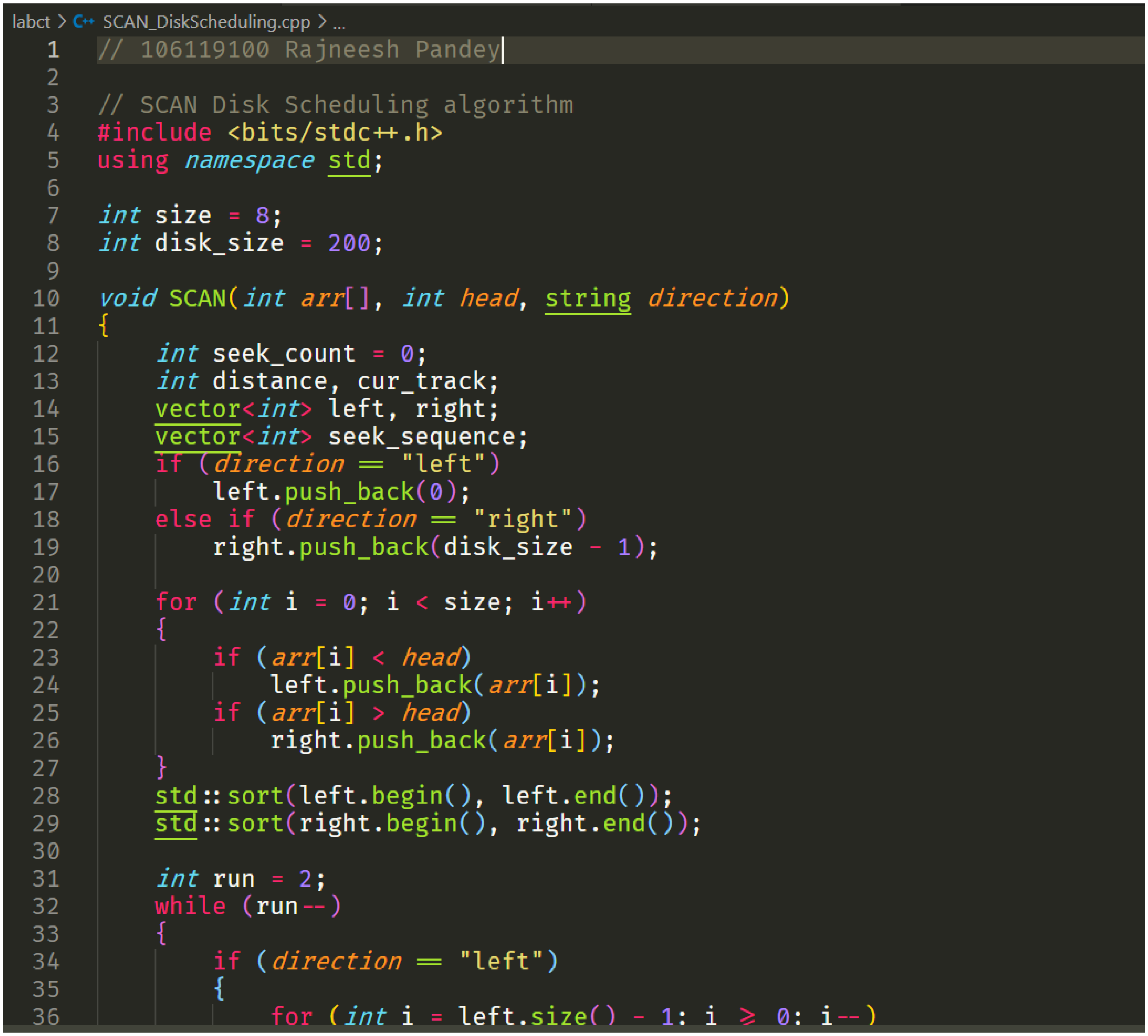
Input / Output



```
Microsoft Windows [Version 10.0.19043.962]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rajne\OneDrive\Desktop\OS LAB>cd "c:\Users\rajne\OneDrive\Desktop\OS LAB\labct\" &&
 g++ SSTF_DiskScheduling.cpp -o SSTF_DiskScheduling && "c:\Users\rajne\OneDrive\Desktop\OS L
AB\labct\"SSTF_DiskScheduling
Total number of seek operations = 204
Seek sequence is :
50
41
34
11
60
79
92
114
176

c:\Users\rajne\OneDrive\Desktop\OS LAB\labct>
```

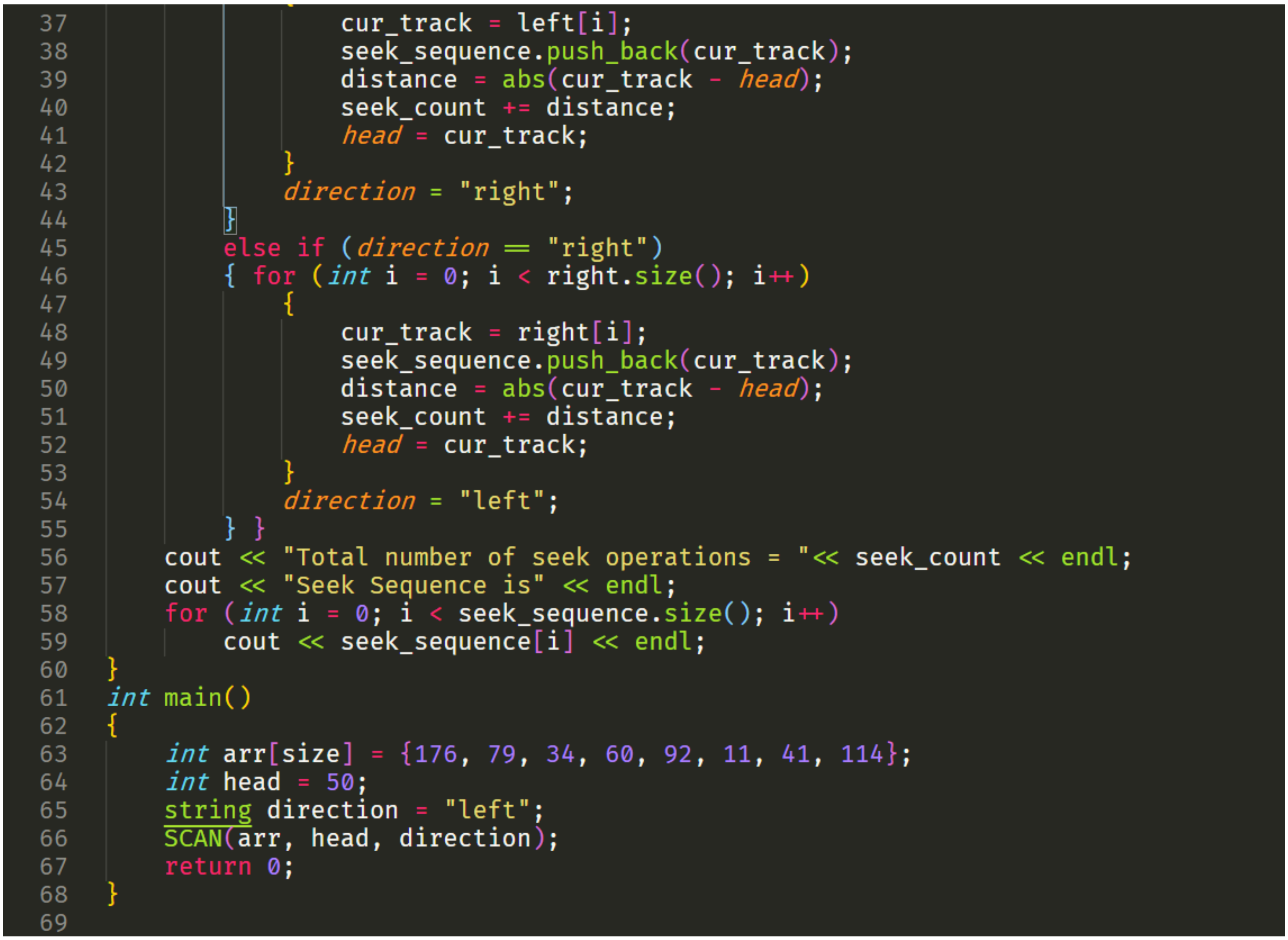
4. SCAN (Elevator) Disk Scheduling Algorithms

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.



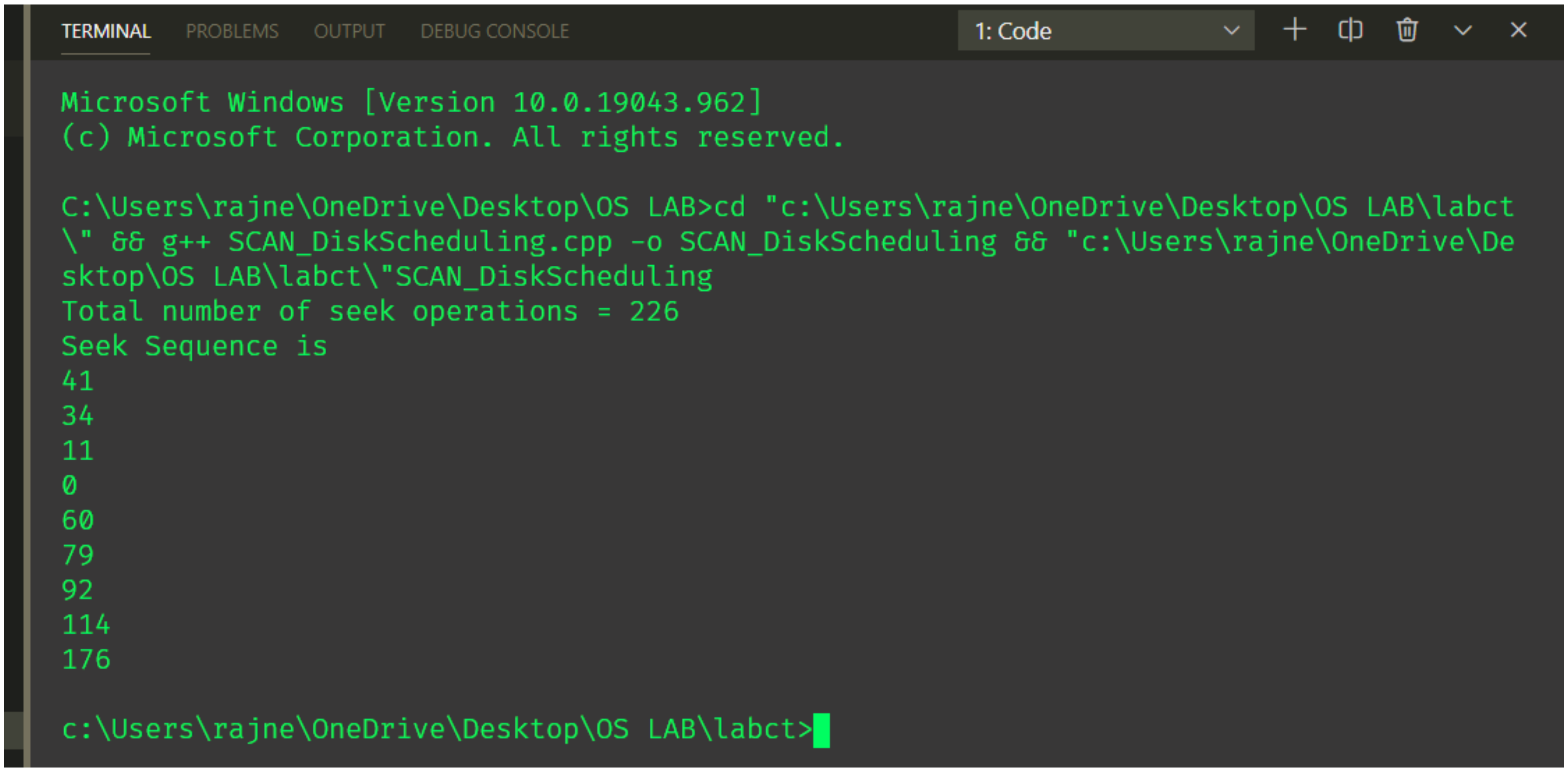
labct > SCAN_DiskScheduling.cpp > ...

```cpp
// 106119100 Rajneesh Pandey

// SCAN Disk Scheduling algorithm
#include <bits/stdc++.h>
using namespace std;

int size = 8;
int disk_size = 200;

void SCAN(int arr[], int head, string direction)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;
    if (direction == "left")
        left.push_back(0);
    else if (direction == "right")
        right.push_back(disk_size - 1);

    for (int i = 0; i < size; i++)
    {
        if (arr[i] < head)
            left.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());

    int run = 2;
    while (run--)
    {
        if (direction == "left")
        {
            for (int i = left.size() - 1; i >= 0; i--)
```



```cpp
                cur_track = left[i];
                seek_sequence.push_back(cur_track);
                distance = abs(cur_track - head);
                seek_count += distance;
                head = cur_track;
            }
            direction = "right";
        }
        else if (direction == "right")
        { for (int i = 0; i < right.size(); i++)
            {
                cur_track = right[i];
                seek_sequence.push_back(cur_track);
                distance = abs(cur_track - head);
                seek_count += distance;
                head = cur_track;
            }
            direction = "left";
        } }
    cout << "Total number of seek operations = "<< seek_count << endl;
    cout << "Seek Sequence is" << endl;
    for (int i = 0; i < seek_sequence.size(); i++)
        cout << seek_sequence[i] << endl;
}
int main()
{
    int arr[size] = {176, 79, 34, 60, 92, 11, 41, 114};
    int head = 50;
    string direction = "left";
    SCAN(arr, head, direction);
    return 0;
}
```

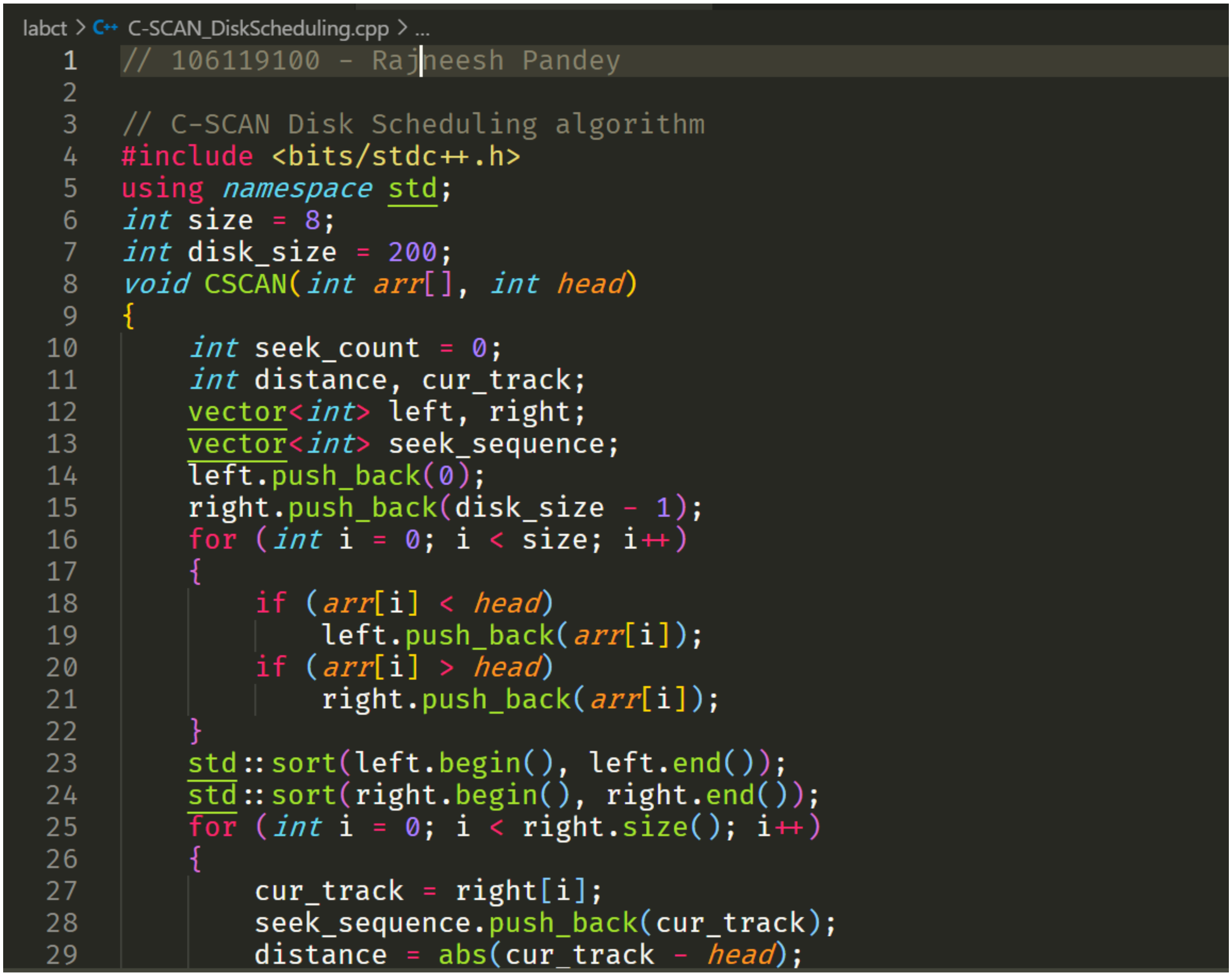
Input / Output



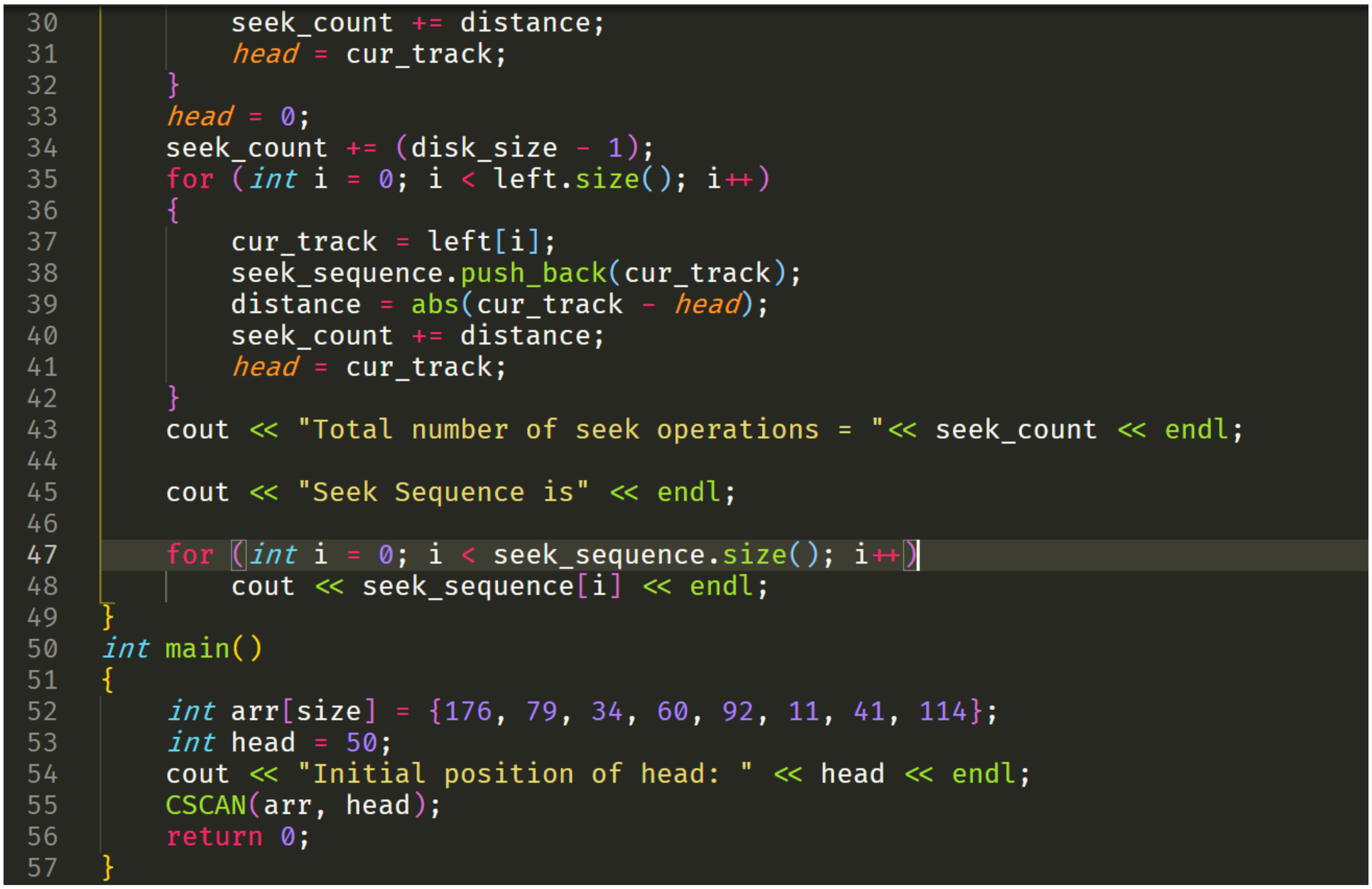
```
Microsoft Windows [Version 10.0.19043.962]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rajne\OneDrive\Desktop\OS LAB>cd "c:\Users\rajne\OneDrive\Desktop\OS LAB\labct
\" && g++ SCAN_DiskScheduling.cpp -o SCAN_DiskScheduling && "c:\Users\rajne\OneDrive\De
sktop\OS LAB\labct\"SCAN_DiskScheduling
Total number of seek operations = 226
Seek Sequence is
41
34
11
0
60
79
92
114
176

c:\Users\rajne\OneDrive\Desktop\OS LAB\labct>
```

4. C-SCAN Disk Scheduling Algorithm

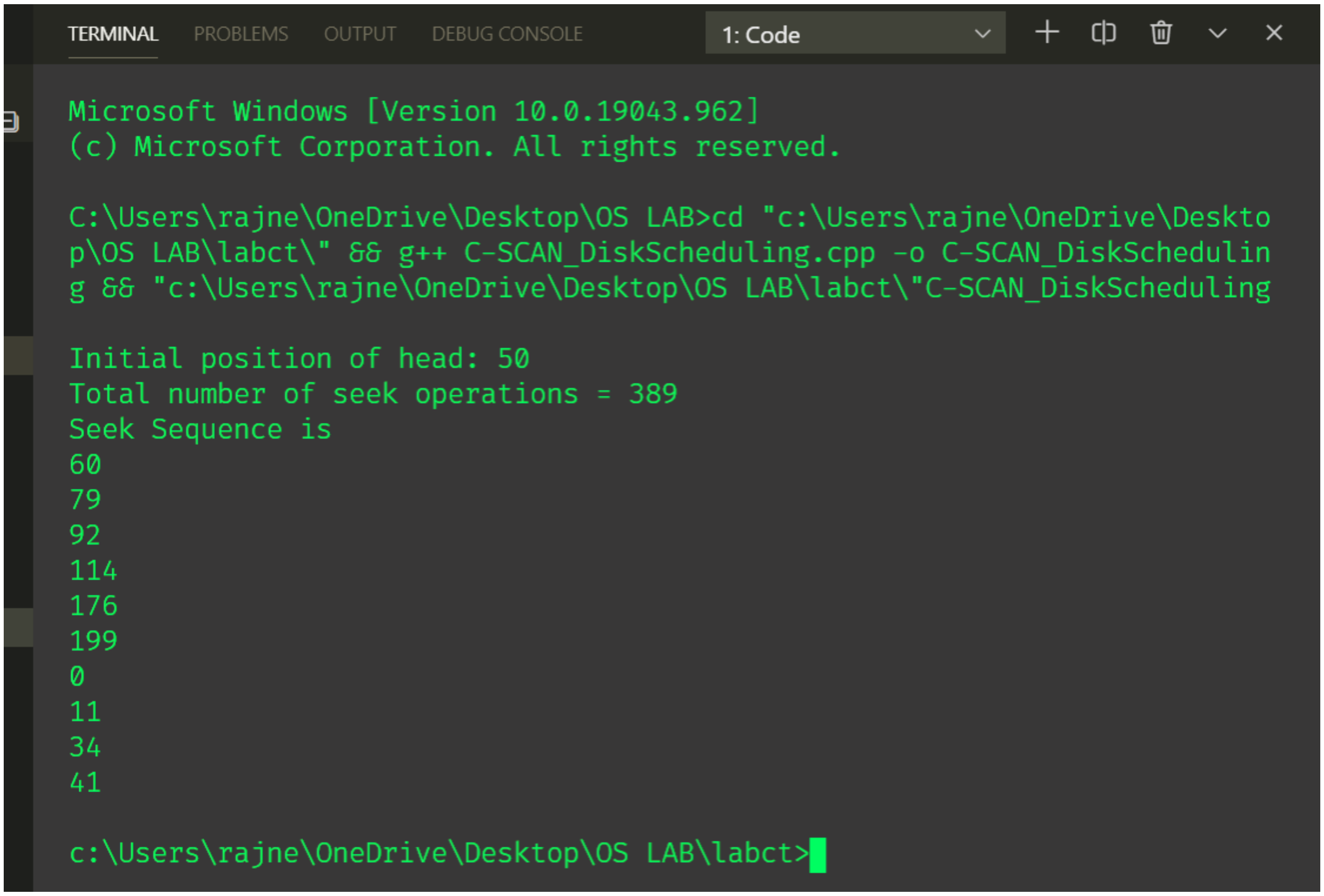
Circular SCAN (C-SCAN) scheduling algorithm is a modified version of SCAN disk scheduling algorithm that deals with the inefficiency of SCAN algorithm by servicing the requests more uniformly. Like SCAN (Elevator Algorithm) C-SCAN moves the head from one end servicing all the requests to the other end. However, as soon as the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip (see chart below) and starts servicing again once reaches the beginning. This is also known as the "Circular Elevator Algorithm" as it essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.


labct > C-SCAN_DiskScheduling.cpp > ...

```cpp
// 106119100 - Rajneesh Pandey

// C-SCAN Disk Scheduling algorithm
#include <bits/stdc++.h>
using namespace std;
int size = 8;
int disk_size = 200;
void CSCAN(int arr[], int head)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;
    left.push_back(0);
    right.push_back(disk_size - 1);
    for (int i = 0; i < size; i++)
    {
        if (arr[i] < head)
            left.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());
    for (int i = 0; i < right.size(); i++)
    {
        cur_track = right[i];
        seek_sequence.push_back(cur_track);
        distance = abs(cur_track - head);
```



```cpp
            seek_count += distance;
            head = cur_track;
        }
        head = 0;
        seek_count += (disk_size - 1);
        for (int i = 0; i < left.size(); i++)
        {
            cur_track = left[i];
            seek_sequence.push_back(cur_track);
            distance = abs(cur_track - head);
            seek_count += distance;
            head = cur_track;
        }
        cout << "Total number of seek operations = "<< seek_count << endl;

        cout << "Seek Sequence is" << endl;

        for (int i = 0; i < seek_sequence.size(); i++)
            cout << seek_sequence[i] << endl;
    }
    int main()
    {
        int arr[size] = {176, 79, 34, 60, 92, 11, 41, 114};
        int head = 50;
        cout << "Initial position of head: " << head << endl;
        CSCAN(arr, head);
        return 0;
    }
```

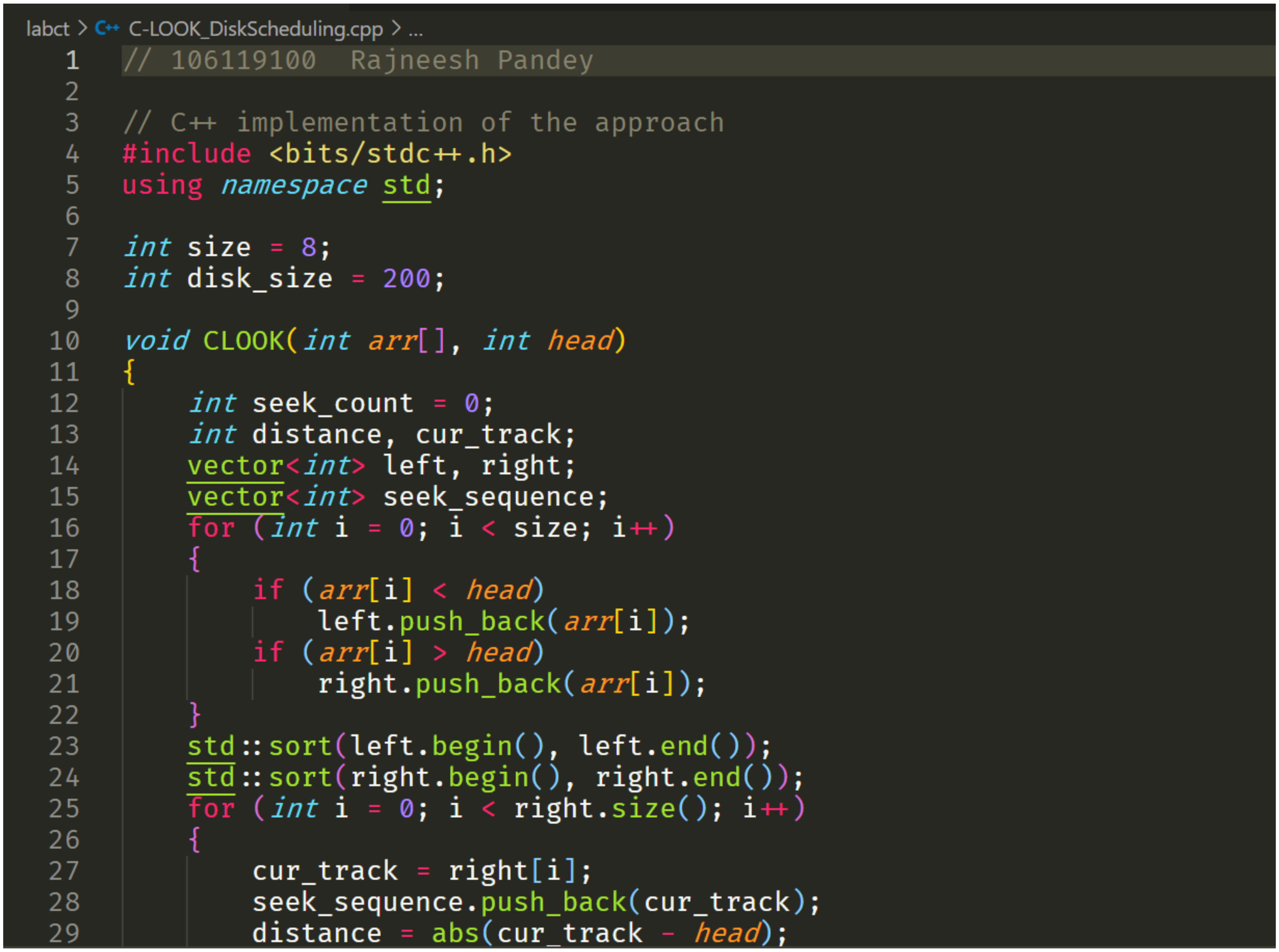


TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE 1: Code

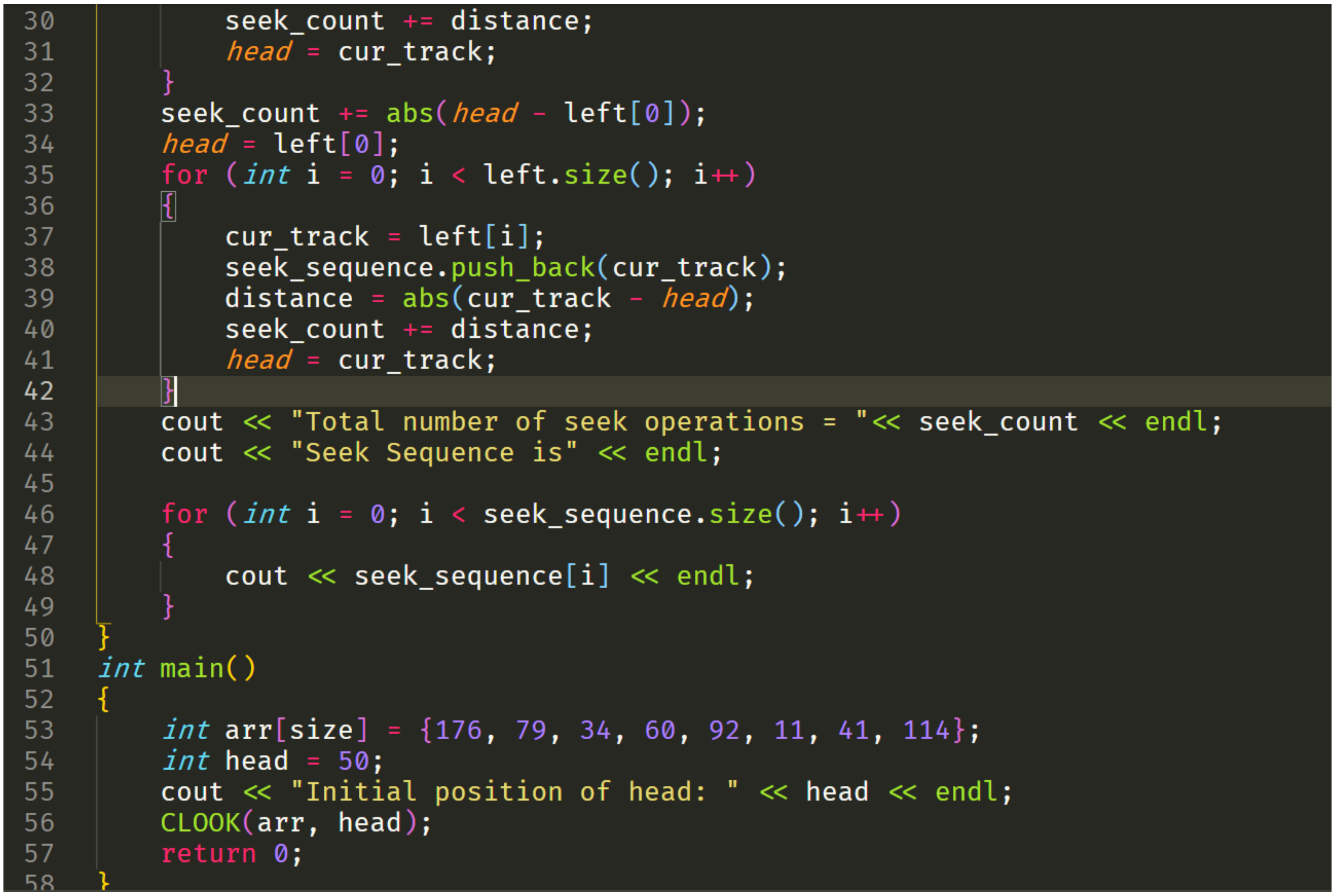
```
Microsoft Windows [Version 10.0.19043.962]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rajne\OneDrive\Desktop\OS LAB>cd "c:\Users\rajne\OneDrive\Deskto
p\OS LAB\labct\" && g++ C-SCAN_DiskScheduling.cpp -o C-SCAN_DiskSchedulin
g && "c:\Users\rajne\OneDrive\Desktop\OS LAB\labct\"C-SCAN_DiskScheduling

Initial position of head: 50
Total number of seek operations = 389
Seek Sequence is
60
79
92
114
176
199
0
11
34
41

c:\Users\rajne\OneDrive\Desktop\OS LAB\labct>
```

5. C - LOOK Disk Scheduling Algorithm

C-LOOK is an enhanced version of both SCAN as well as LOOK disk scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than C-SCAN algorithm. We know that C-SCAN is used to avoid starvation and services all the requests more uniformly, the same goes for C-LOOK.
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.

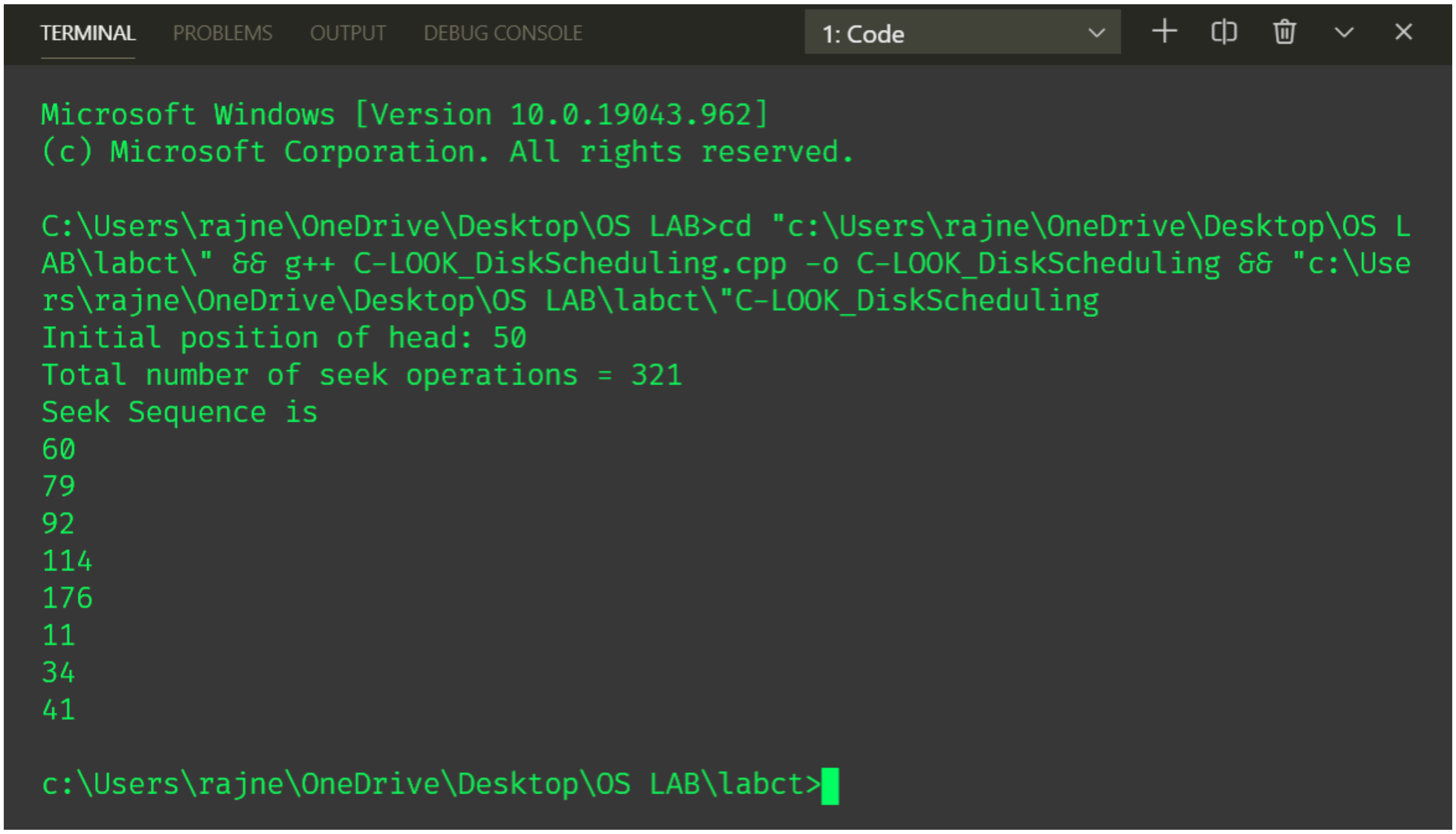


```cpp
// 106119100  Rajneesh Pandey

// C++ implementation of the approach
#include <bits/stdc++.h>
using namespace std;

int size = 8;
int disk_size = 200;

void CLOOK(int arr[], int head)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;
    for (int i = 0; i < size; i++)
    {
        if (arr[i] < head)
            left.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());
    for (int i = 0; i < right.size(); i++)
    {
        cur_track = right[i];
        seek_sequence.push_back(cur_track);
        distance = abs(cur_track - head);
```



```cpp
        seek_count += distance;
        head = cur_track;
    }
    seek_count += abs(head - left[0]);
    head = left[0];
    for (int i = 0; i < left.size(); i++)
    {
        cur_track = left[i];
        seek_sequence.push_back(cur_track);
        distance = abs(cur_track - head);
        seek_count += distance;
        head = cur_track;
    }
    cout << "Total number of seek operations = "<< seek_count << endl;
    cout << "Seek Sequence is" << endl;

    for (int i = 0; i < seek_sequence.size(); i++)
    {
        cout << seek_sequence[i] << endl;
    }
}
int main()
{
    int arr[size] = {176, 79, 34, 60, 92, 11, 41, 114};
    int head = 50;
    cout << "Initial position of head: " << head << endl;
    CLOOK(arr, head);
    return 0;
}
```

Input / Output



```
Microsoft Windows [Version 10.0.19043.962]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rajne\OneDrive\Desktop\OS LAB>cd "c:\Users\rajne\OneDrive\Desktop\OS L
AB\labct\" && g++ C-LOOK_DiskScheduling.cpp -o C-LOOK_DiskScheduling && "c:\Use
rs\rajne\OneDrive\Desktop\OS LAB\labct\"C-LOOK_DiskScheduling
Initial position of head: 50
Total number of seek operations = 321
Seek Sequence is
60
79
92
114
176
11
34
41

c:\Users\rajne\OneDrive\Desktop\OS LAB\labct>
```