# OS Lab -(CSLR42) ENDSEM -04/05/2021

-----------------------------------

## 106119100 - Rajneesh Pandey

---------------------------

## SET-B

------------------------------------------------------------------------------------

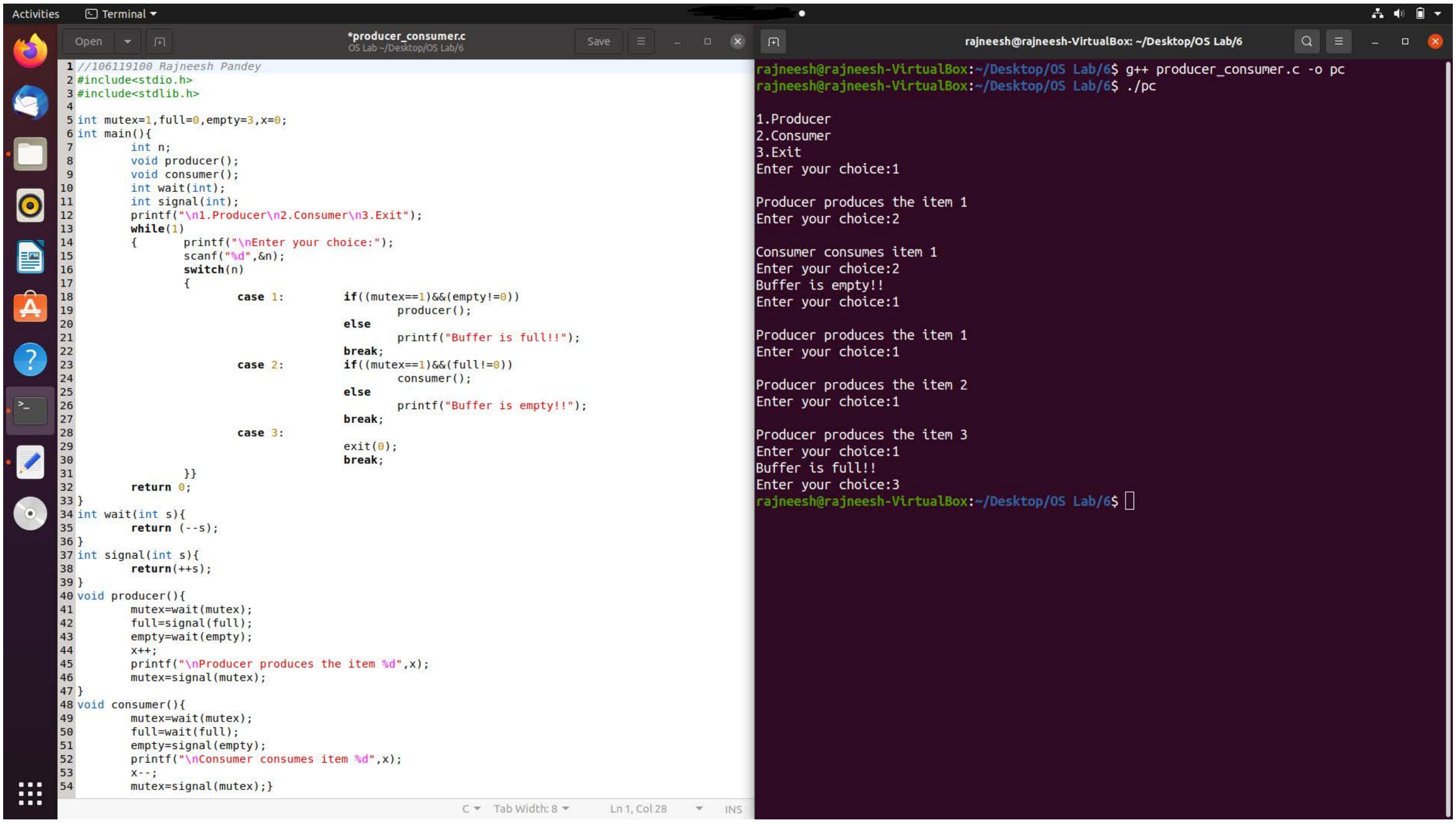
## Question 1 Write and Execute a Program to demonstrate Producer Consumer Problem Using Semaphores.

we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.

When producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of “full” is increased by 1. The value of mutex is also increased by 1 beacuse the task of producer has been completed and consumer can access the buffer.

As the consumer is removing an item from buffer, therefore the value of “full” is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of “empty” by 1. The value of mutex is also increased so that producer can access the buffer now. producer can access the buffer now.

## Program and Input/Output



[redacted]

```c
//106119100 Rajneesh Pandey
#include<stdio.h>
#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;
int main(){
        int n;
        void producer();
        void consumer();
        int wait(int);
        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)
        {       printf("\nEnter your choice:");
                scanf("%d",&n);
                switch(n)
                {
                        case 1:         if((mutex==1)&&(empty!=0))
                                                producer();
                                        else
                                                printf("Buffer is full!!");
                                        break;
                        case 2:         if((mutex==1)&&(full!=0))
                                                consumer();
                                        else
                                                printf("Buffer is empty!!");
                                        break;
                        case 3:
                                        exit(0);
                                        break;
                }}
        return 0;
}
int wait(int s){
        return (--s);
}
int signal(int s){
        return(++s);
}
void producer(){
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}
void consumer(){
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer consumes item %d",x);
        x--;
        mutex=signal(mutex);}
```

```
rajneesh@rajneesh-VirtualBox:~/Desktop/OS Lab/6$ g++ producer_consumer.c -o pc
rajneesh@rajneesh-VirtualBox:~/Desktop/OS Lab/6$ ./pc

1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3
rajneesh@rajneesh-VirtualBox:~/Desktop/OS Lab/6$ 
```

**Question 2.** **Write a Program in a desired object oriented programming language of your choice to demonstrate the concept of LRU Page Replacement Algorithm**

Let capacity be the number of pages that memory can hold. Let set be the current set of pages in memory.

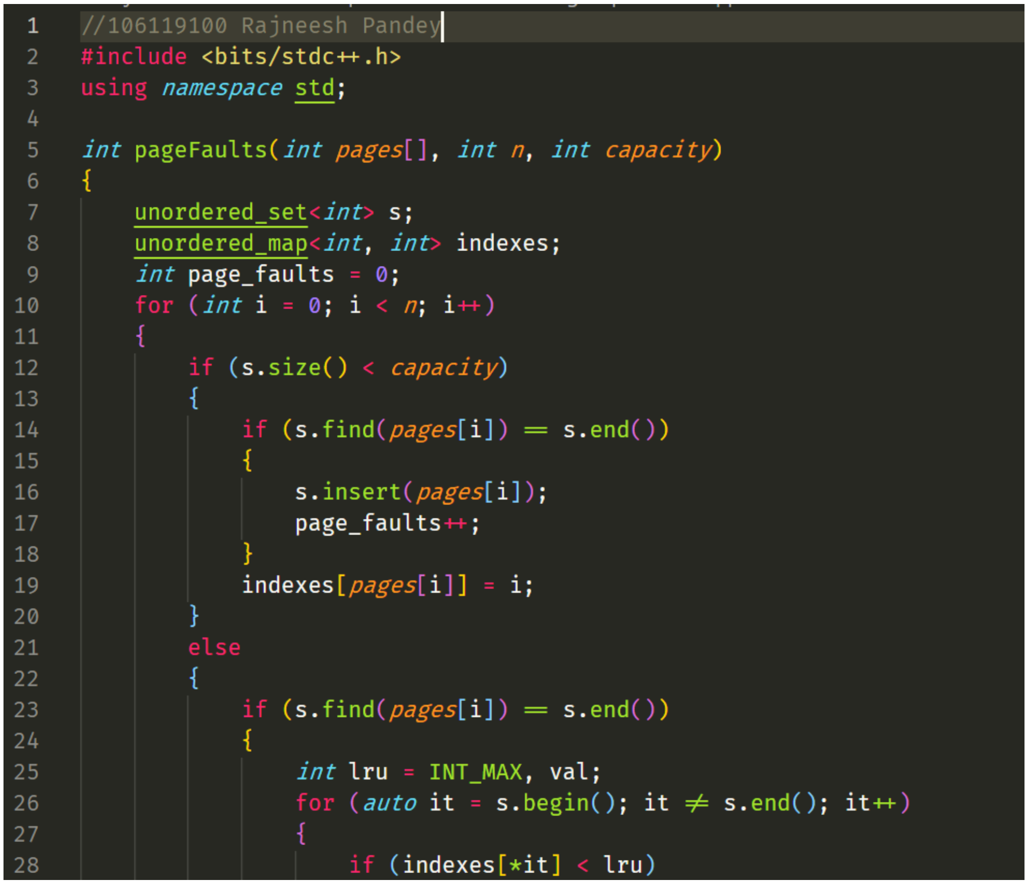
1- Start traversing the pages.

i) If set holds less pages than capacity.
  a) Insert page into the set one by one until the side of set reaches capacity or all page requests are processed.
  b) Simultaneously maintain the recent occurred index of each page in a map called indexes.
  c) Increment page fault

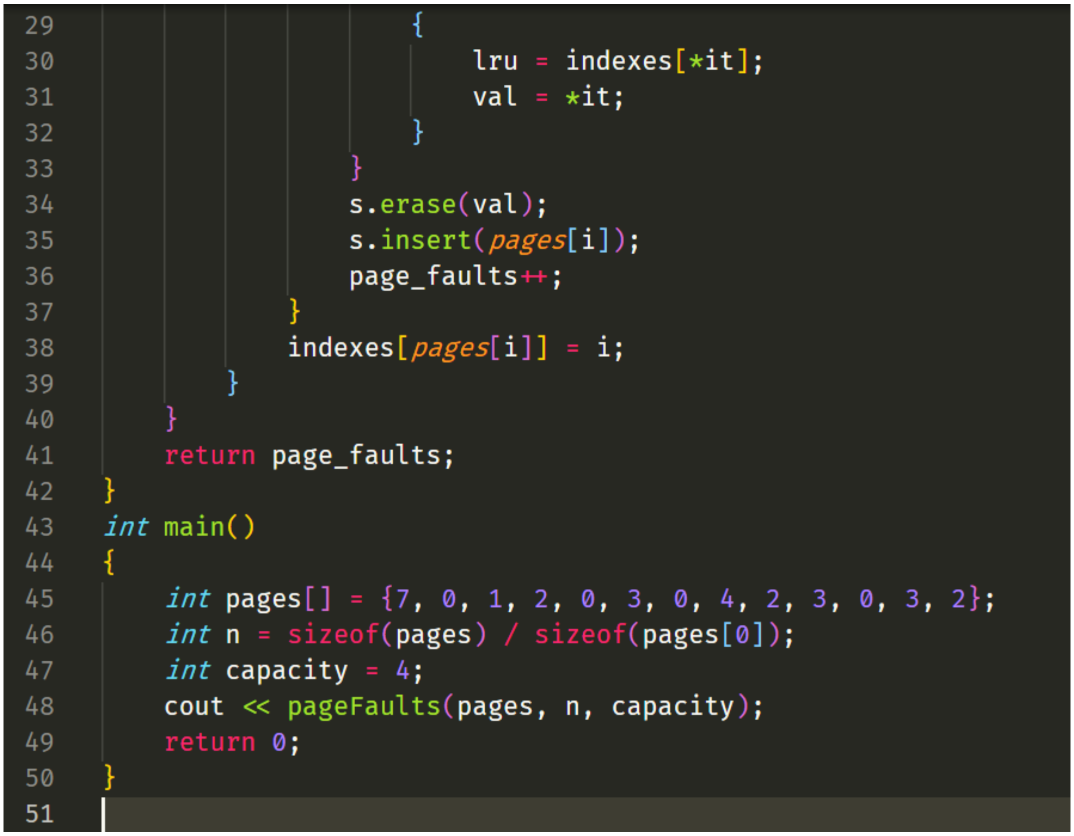
ii) Else
  If current page is present in set, do nothing. Else
  a) Find the page in the set that was least recently used. find it using index array. We basically need to replace the page with minimum index.
  b) Replace the found page with current page.
  c) Increment page faults.
  d) Update index of current page.

iii) Return page faults.

**Program and Input / Output**

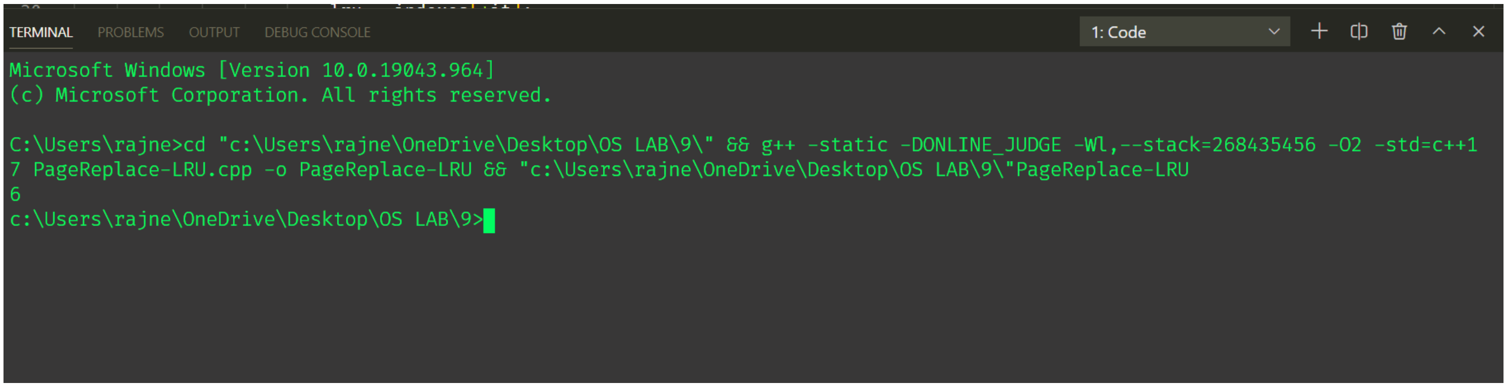


```
//106119100 Rajneesh Pandey
#include <bits/stdc++.h>
using namespace std;

int pageFaults(int pages[], int n, int capacity)
{
    unordered_set<int> s;
    unordered_map<int, int> indexes;
    int page_faults = 0;
    for (int i = 0; i < n; i++)
    {
        if (s.size() < capacity)
        {
            if (s.find(pages[i]) == s.end())
            {
                s.insert(pages[i]);
                page_faults++;
            }
            indexes[pages[i]] = i;
        }
        else
        {
            if (s.find(pages[i]) == s.end())
            {
                int lru = INT_MAX, val;
                for (auto it = s.begin(); it != s.end(); it++)
                {
                    if (indexes[*it] < lru)
```



```
                    {
                        lru = indexes[*it];
                        val = *it;
                    }
                }
                s.erase(val);
                s.insert(pages[i]);
                page_faults++;
            }
            indexes[pages[i]] = i;
        }
    }
    return page_faults;
}
int main()
{
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int n = sizeof(pages) / sizeof(pages[0]);
    int capacity = 4;
    cout << pageFaults(pages, n, capacity);
    return 0;
}
```

**Input / Output:**



```
Microsoft Windows [Version 10.0.19043.964]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rajne>cd "c:\Users\rajne\OneDrive\Desktop\OS LAB\9\" && g++ -static -DONLINE_JUDGE -Wl,--stack=268435456 -O2 -std=c++1
7 PageReplace-LRU.cpp -o PageReplace-LRU && "c:\Users\rajne\OneDrive\Desktop\OS LAB\9\"PageReplace-LRU
6
c:\Users\rajne\OneDrive\Desktop\OS LAB\9>
```

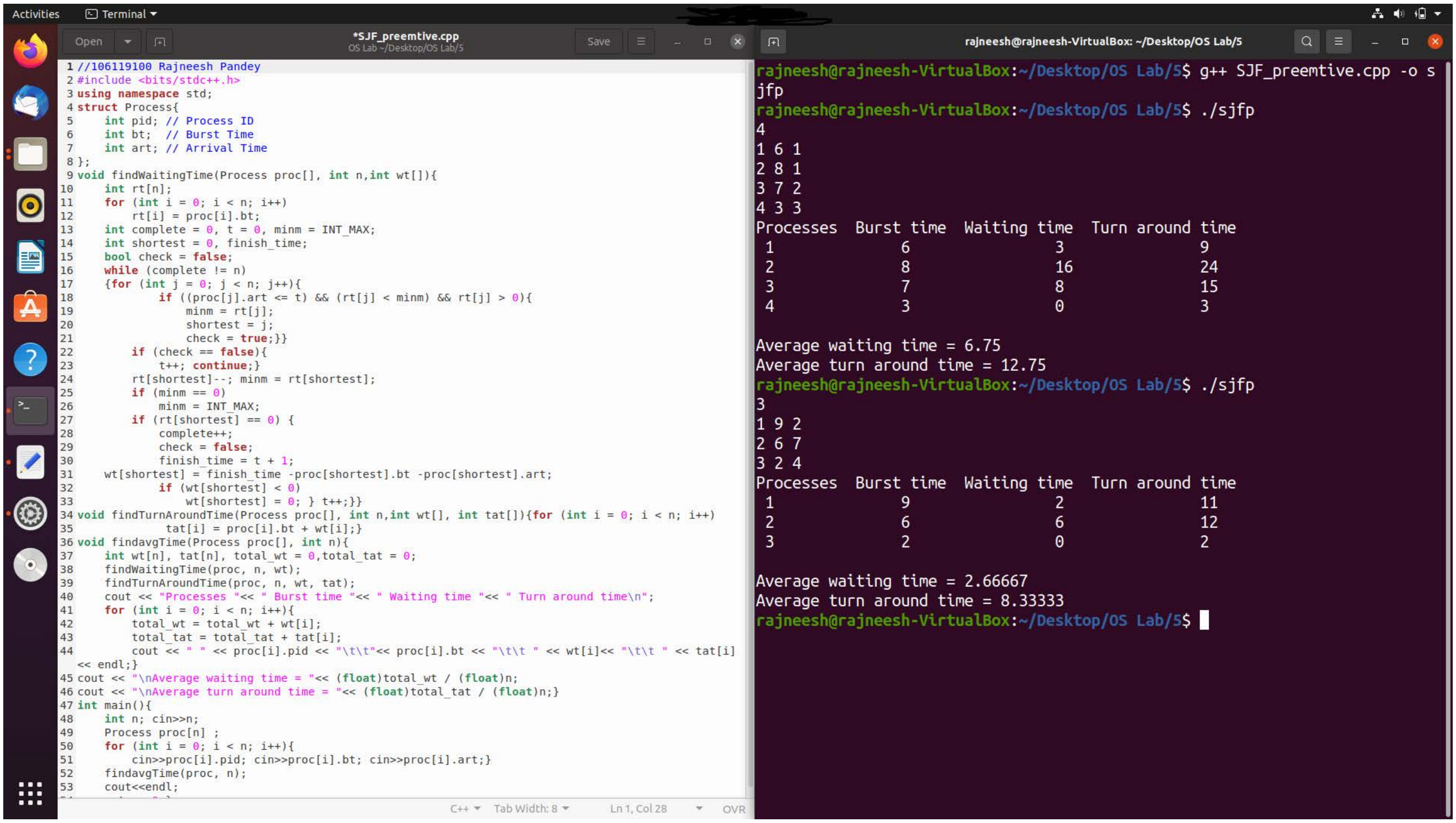
# Question 3:

## Depict the concept of Shortest Job First CPU Scheduling with a help of a program

In the Shortest Job First (SJF)/Shortest Remaining Time First (SRTF) scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

### Implementation

1- Traverse until all process gets completely executed.
   a) Find process with minimum remaining time at every single time lap.
   b) Reduce its time by 1.
   c) Check if its remaining time becomes 0
   d) Increment the counter of process completion.
   e) Completion time of current process = current_time +1;
   e) Calculate waiting time for each completed process.
         wt[i]= Completion time - arrival_time-burst_time
    f)Increment time lap by one.
2- Find turnaround time (waiting_time+burst_time).



```cpp
//106119100 Rajneesh Pandey
#include <bits/stdc++.h>
using namespace std;
struct Process{
    int pid; // Process ID
    int bt;  // Burst Time
    int art; // Arrival Time
};
void findWaitingTime(Process proc[], int n,int wt[]){
    int rt[n];
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;
    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;
    while (complete != n)
    {for (int j = 0; j < n; j++){
            if ((proc[j].art <= t) && (rt[j] < minm) && rt[j] > 0){
                minm = rt[j];
                shortest = j;
                check = true;}}
        if (check == false){
            t++; continue;}
        rt[shortest]--; minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;
        if (rt[shortest] == 0) {
            complete++;
            check = false;
            finish_time = t + 1;
    wt[shortest] = finish_time -proc[shortest].bt -proc[shortest].art;
            if (wt[shortest] < 0)
                wt[shortest] = 0; } t++;}}
void findTurnAroundTime(Process proc[], int n,int wt[], int tat[]){for (int i = 0; i < n; i++)
            tat[i] = proc[i].bt + wt[i];}
void findavgTime(Process proc[], int n){
    int wt[n], tat[n], total_wt = 0,total_tat = 0;
    findWaitingTime(proc, n, wt);
    findTurnAroundTime(proc, n, wt, tat);
    cout << "Processes "<< " Burst time "<< " Waiting time "<< " Turn around time\n";
    for (int i = 0; i < n; i++){
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"<< proc[i].bt << "\t\t " << wt[i]<< "\t\t " << tat[i]
<< endl;}
cout << "\nAverage waiting time = "<< (float)total_wt / (float)n;
cout << "\nAverage turn around time = "<< (float)total_tat / (float)n;}
int main(){
    int n; cin>>n;
    Process proc[n] ;
    for (int i = 0; i < n; i++){
        cin>>proc[i].pid; cin>>proc[i].bt; cin>>proc[i].art;}
    findavgTime(proc, n);
    cout<<endl;
```

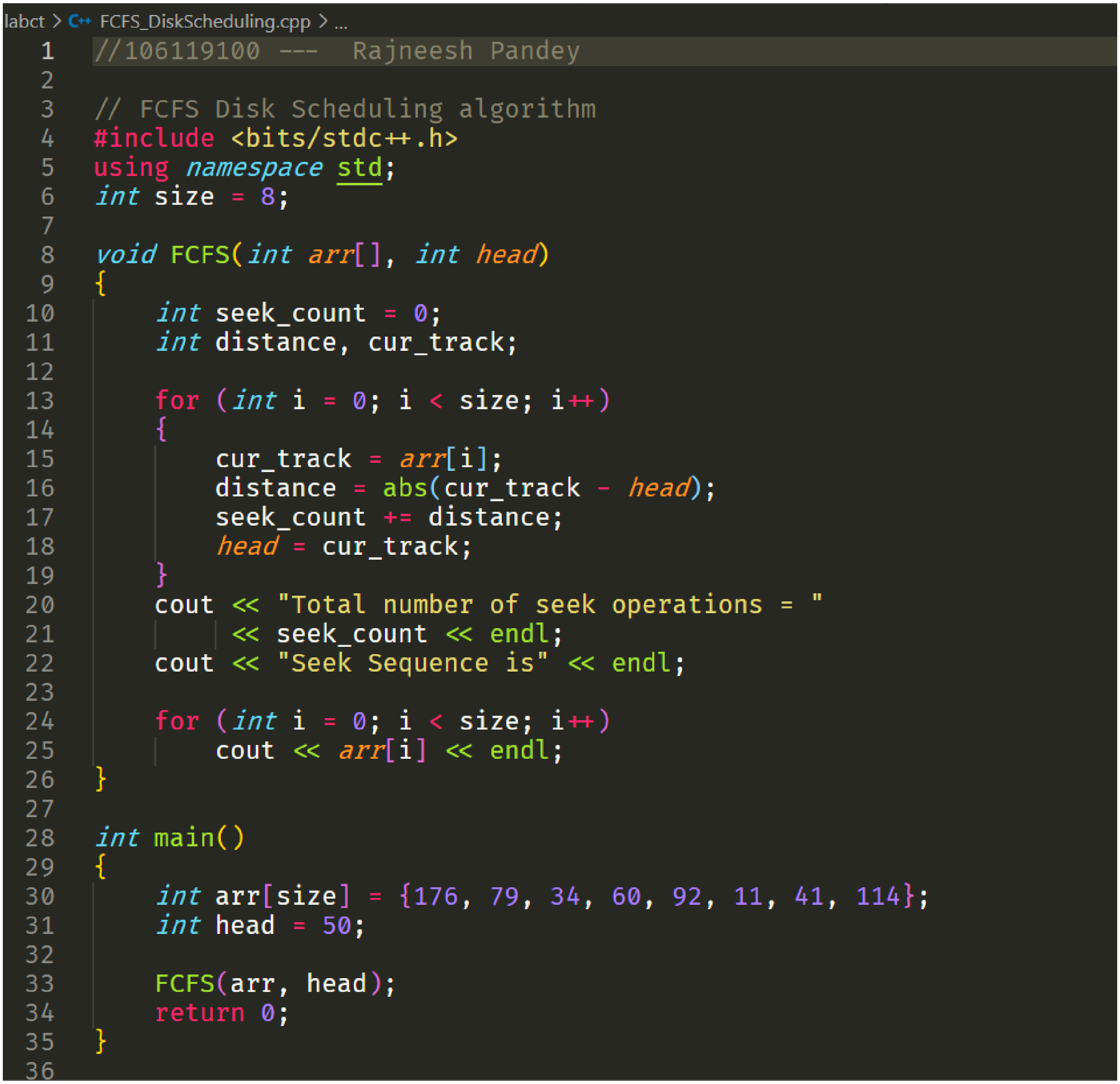
```
rajneesh@rajneesh-VirtualBox:~/Desktop/OS Lab/5$ g++ SJF_preemtive.cpp -o s
jfp
rajneesh@rajneesh-VirtualBox:~/Desktop/OS Lab/5$ ./sjfp
4
1 6 1
2 8 1
3 7 2
4 3 3
Processes  Burst time  Waiting time  Turn around time
 1               6              3               9
 2               8              16              24
 3               7              8               15
 4               3              0               3

Average waiting time = 6.75
Average turn around time = 12.75
rajneesh@rajneesh-VirtualBox:~/Desktop/OS Lab/5$ ./sjfp
3
1 9 2
2 6 7
3 2 4
Processes  Burst time  Waiting time  Turn around time
 1               9              2               11
 2               6              6               12
 3               2              0               2

Average waiting time = 2.66667
Average turn around time = 8.33333
rajneesh@rajneesh-VirtualBox:~/Desktop/OS Lab/5$ 
```

## Question 4 :

### Demonstrate FCFS and Shortest Seek Time First (SSTF) Algorithms for Disk Scheduling.

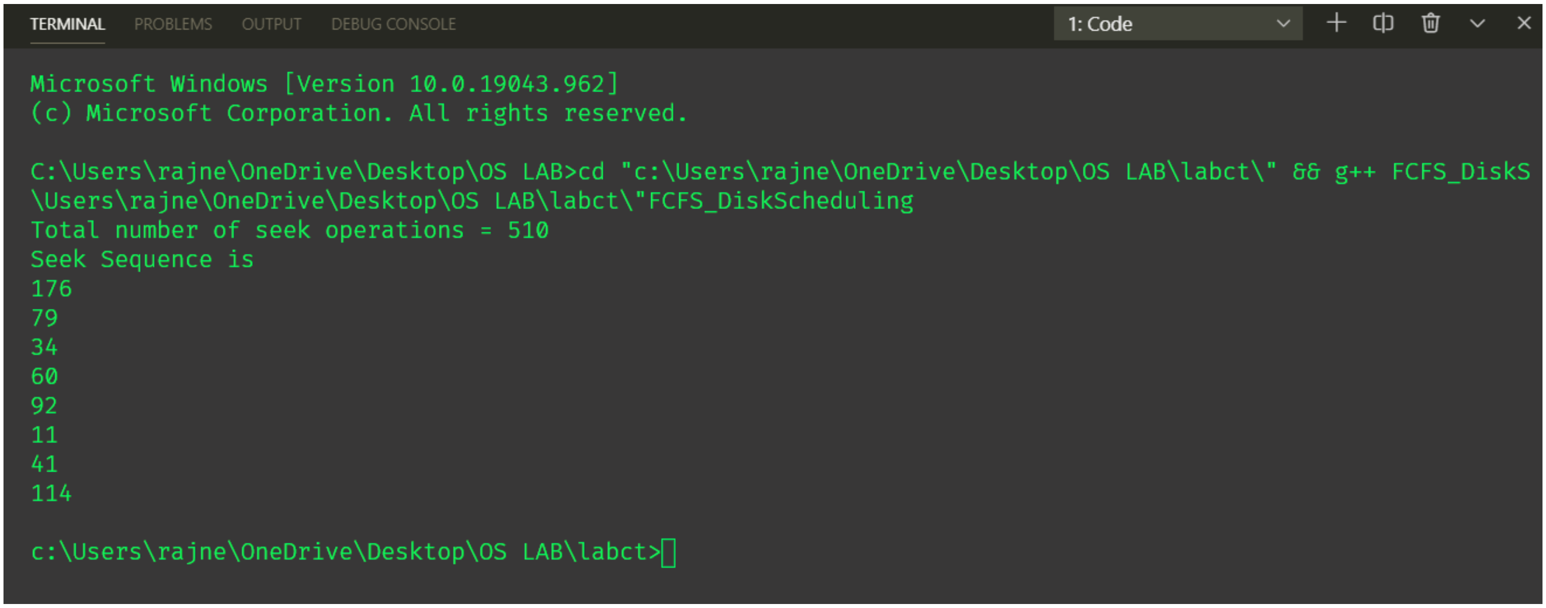
### 1. FCFS Disk Scheduling Algorithm

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.



```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;
}

```

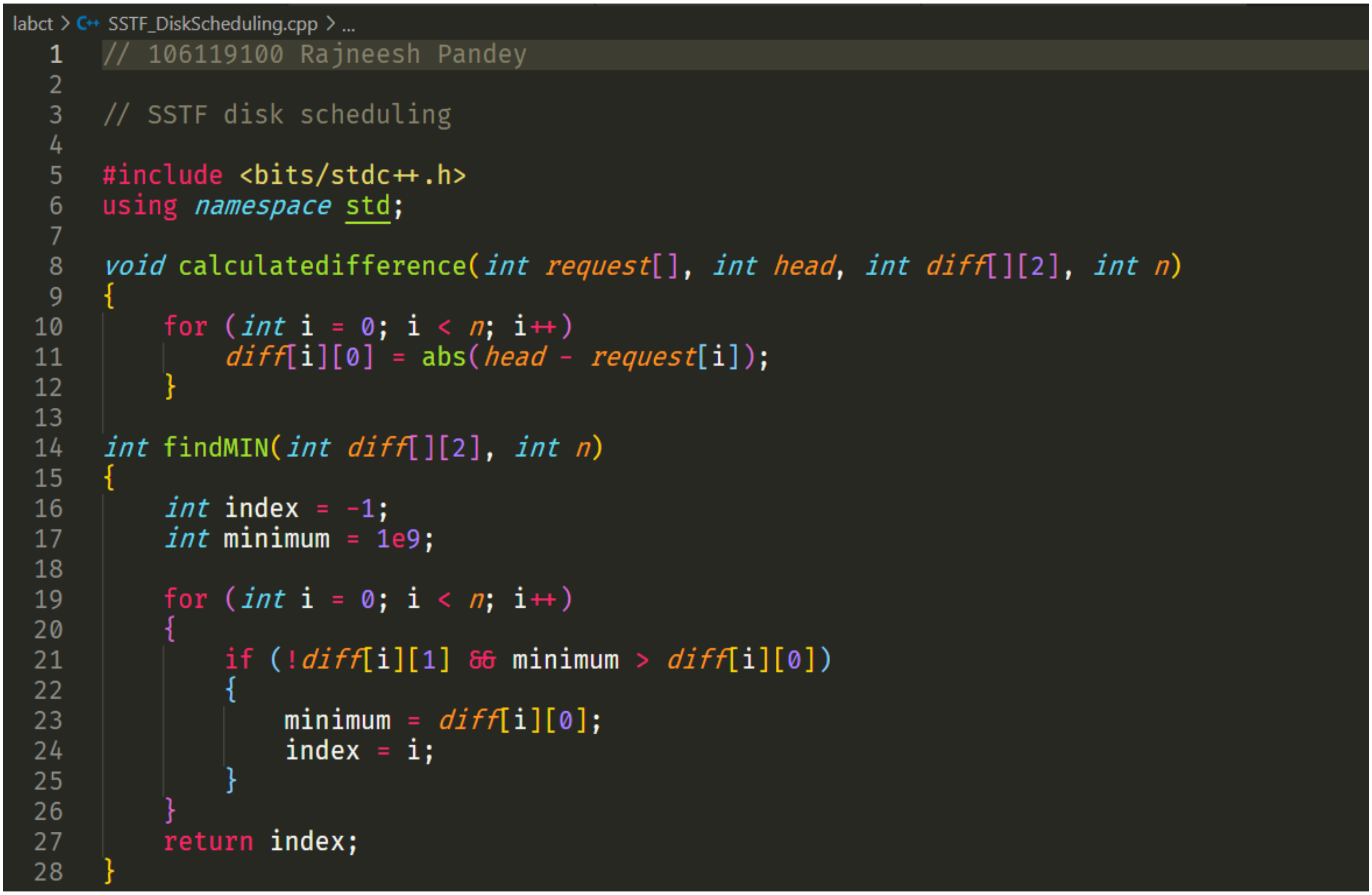
# Input / Output



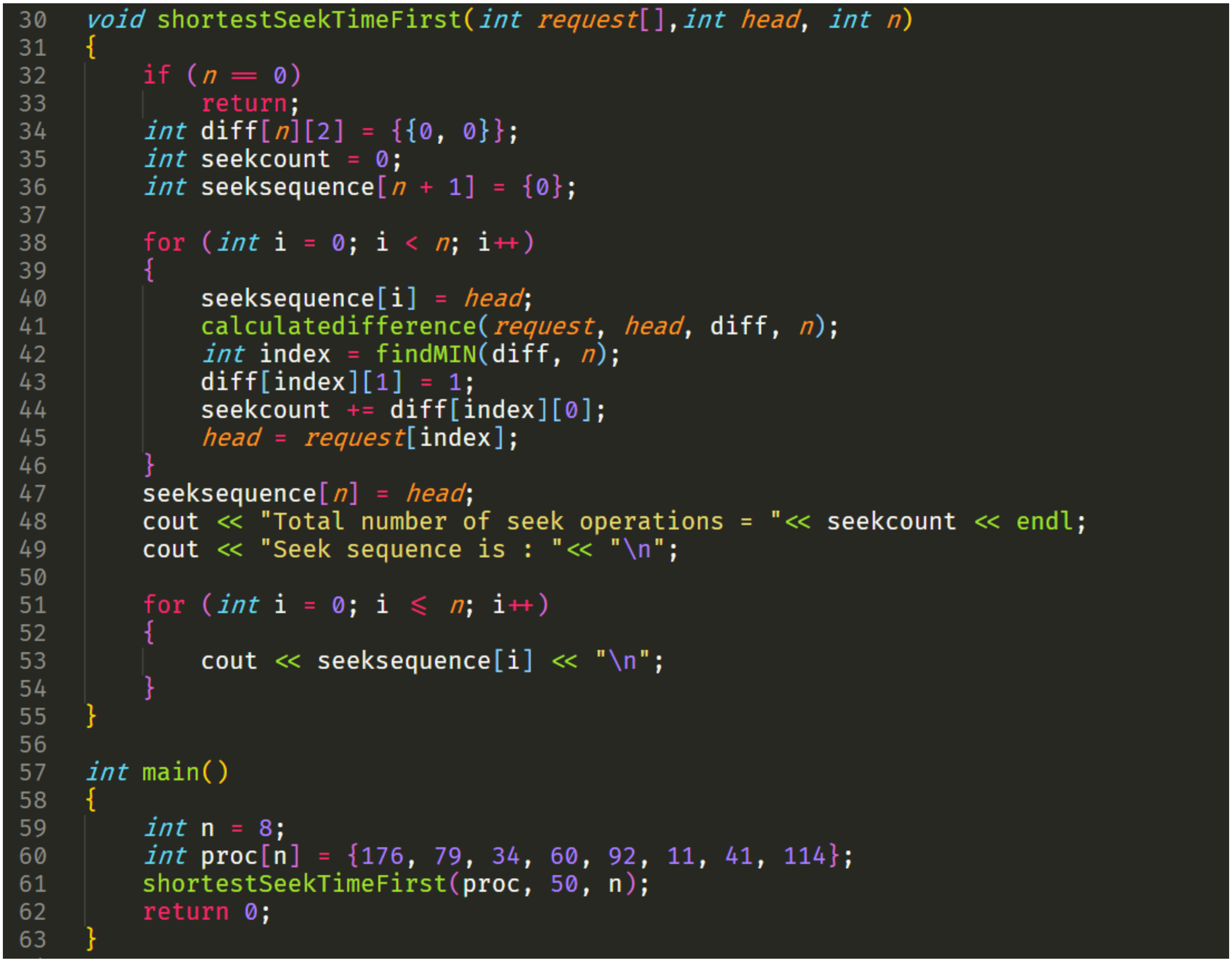
```
Microsoft Windows [Version 10.0.19043.962]
(c) Microsoft Corporation. All rights reserved.

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>
```

## 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.

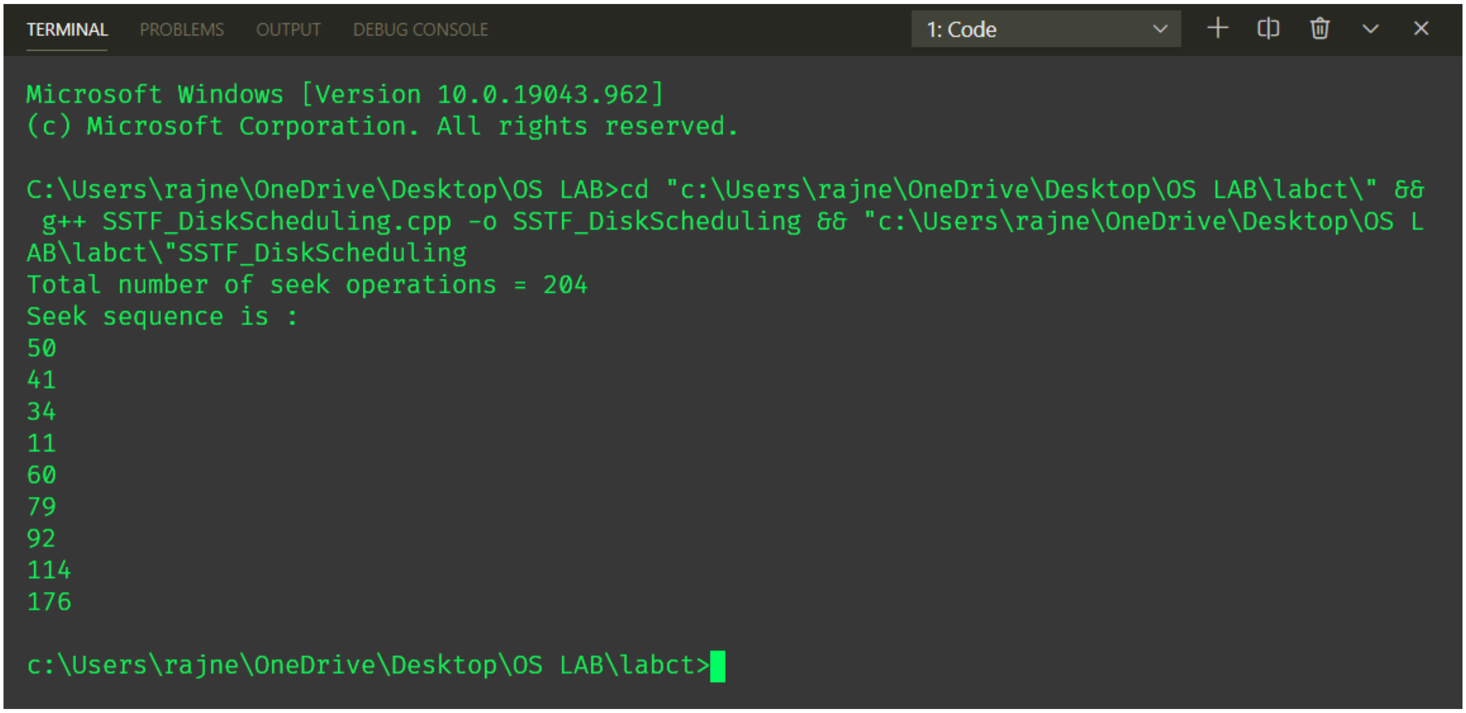


```cpp
// 106119100 Rajneesh Pandey

// 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>
```