OPERATING SYSTEM LAB ASSIGNMENT

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

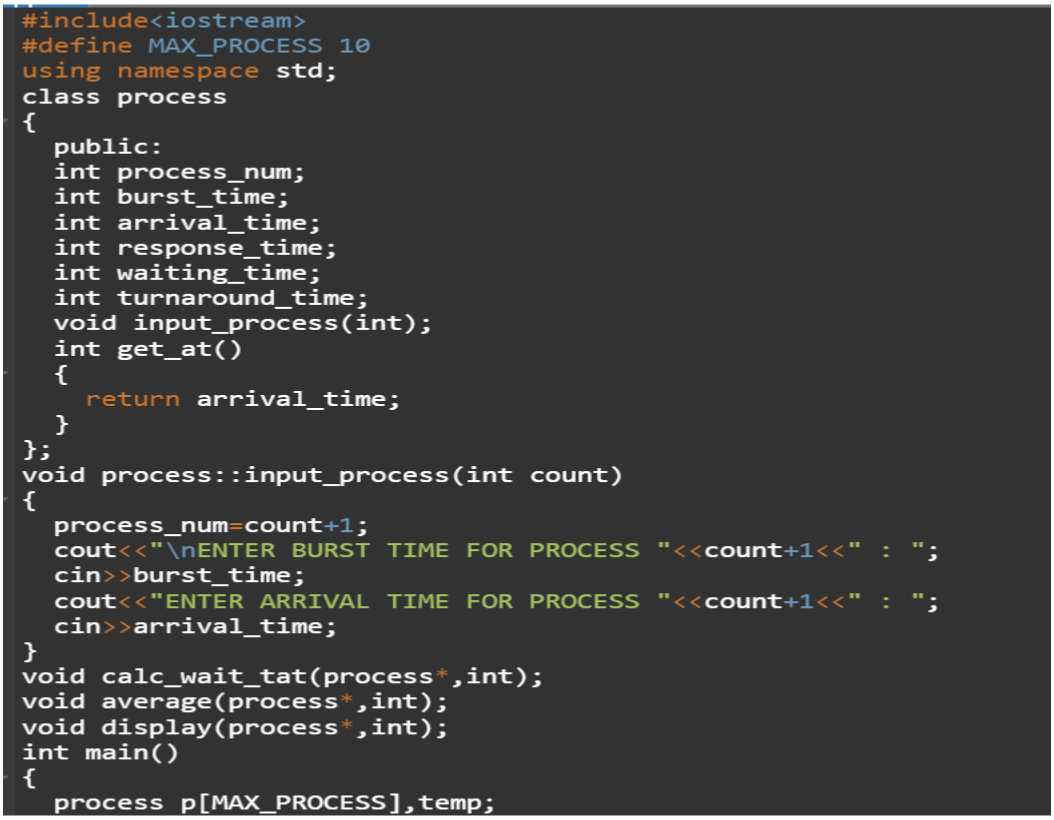
November 2022

SUBMITTED BY:
Krishnanshu Shoor
102103660
2CO24

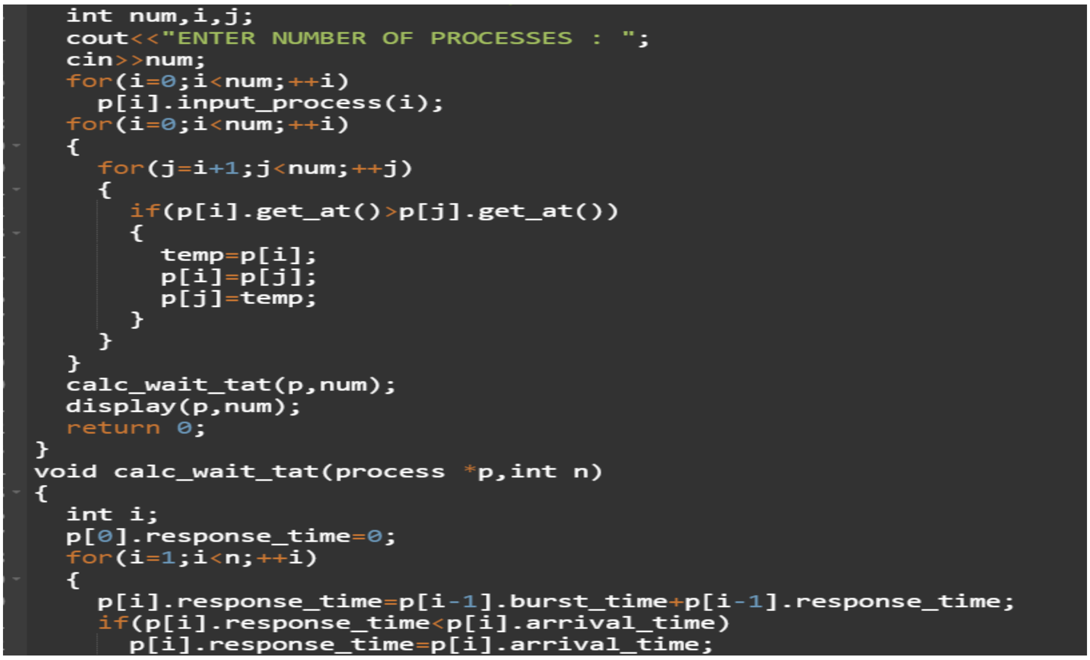
SUBMITTED TO:
Ms. Reaya Aggarwal

CPU SCHEDULING

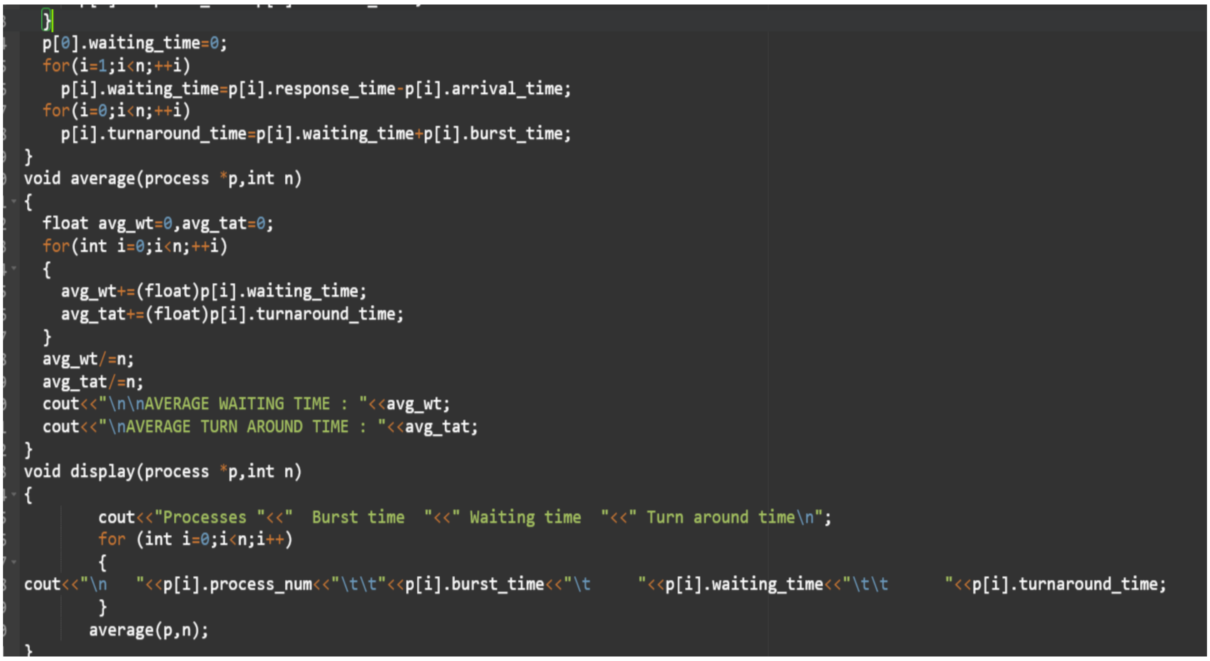
1. FCFS



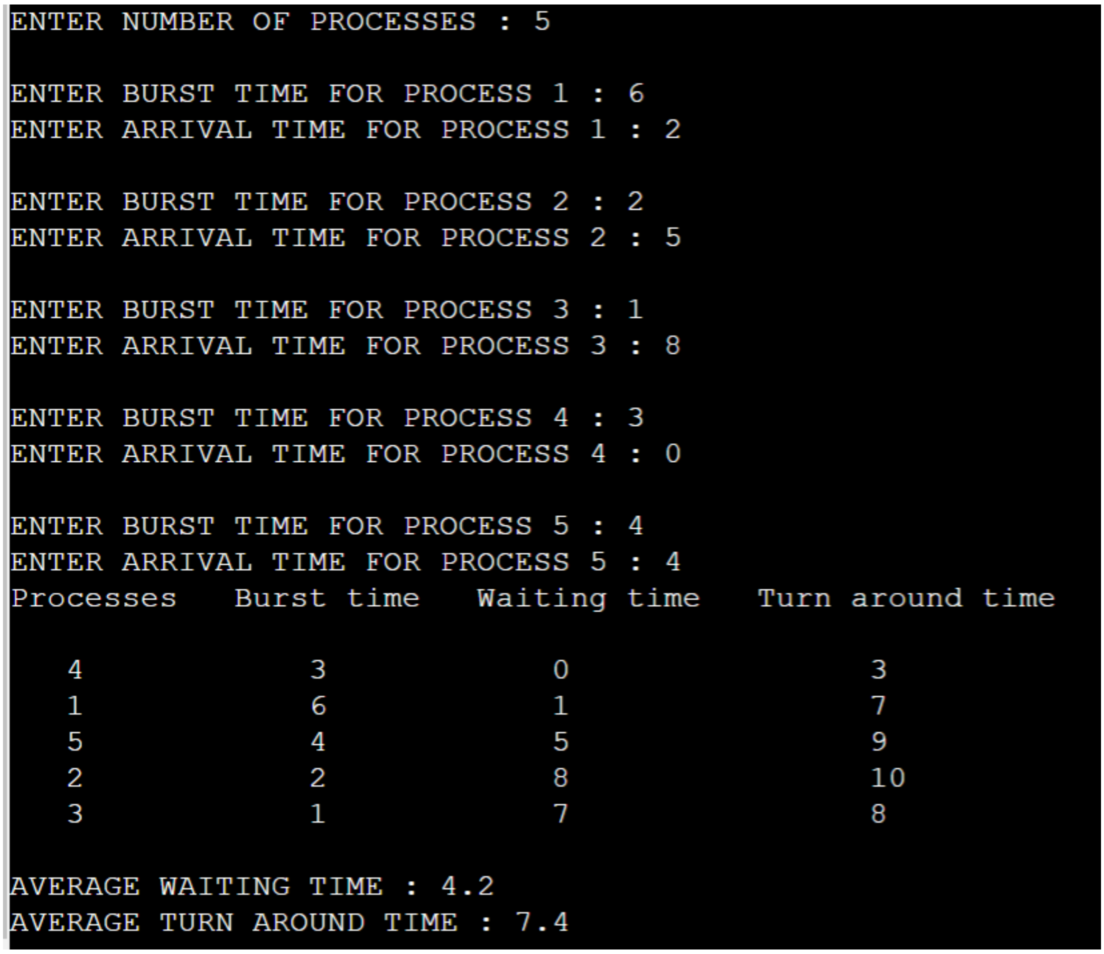
```
#include<iostream>
#define MAX_PROCESS 10
using namespace std;
class process
{
  public:
  int process_num;
  int burst_time;
  int arrival_time;
  int response_time;
  int waiting_time;
  int turnaround_time;
  void input_process(int);
  int get_at()
  {
    return arrival_time;
  }
};
void process::input_process(int count)
{
  process_num=count+1;
  cout<<"\nENTER BURST TIME FOR PROCESS "<<count+1<<" : ";
  cin>>burst_time;
  cout<<"ENTER ARRIVAL TIME FOR PROCESS "<<count+1<<" : ";
  cin>>arrival_time;
}
void calc_wait_tat(process*,int);
void average(process*,int);
void display(process*,int);
int main()
{
  process p[MAX_PROCESS],temp;
```



```
  int num,i,j;
  cout<<"ENTER NUMBER OF PROCESSES : ";
  cin>>num;
  for(i=0;i<num;++i)
    p[i].input_process(i);
  for(i=0;i<num;++i)
  {
    for(j=i+1;j<num;++j)
    {
      if(p[i].get_at()>p[j].get_at())
      {
        temp=p[i];
        p[i]=p[j];
        p[j]=temp;
      }
    }
  }
  calc_wait_tat(p,num);
  display(p,num);
  return 0;
}
void calc_wait_tat(process *p,int n)
{
  int i;
  p[0].response_time=0;
  for(i=1;i<n;++i)
  {
    p[i].response_time=p[i-1].burst_time+p[i-1].response_time;
    if(p[i].response_time<p[i].arrival_time)
      p[i].response_time=p[i].arrival_time;
```

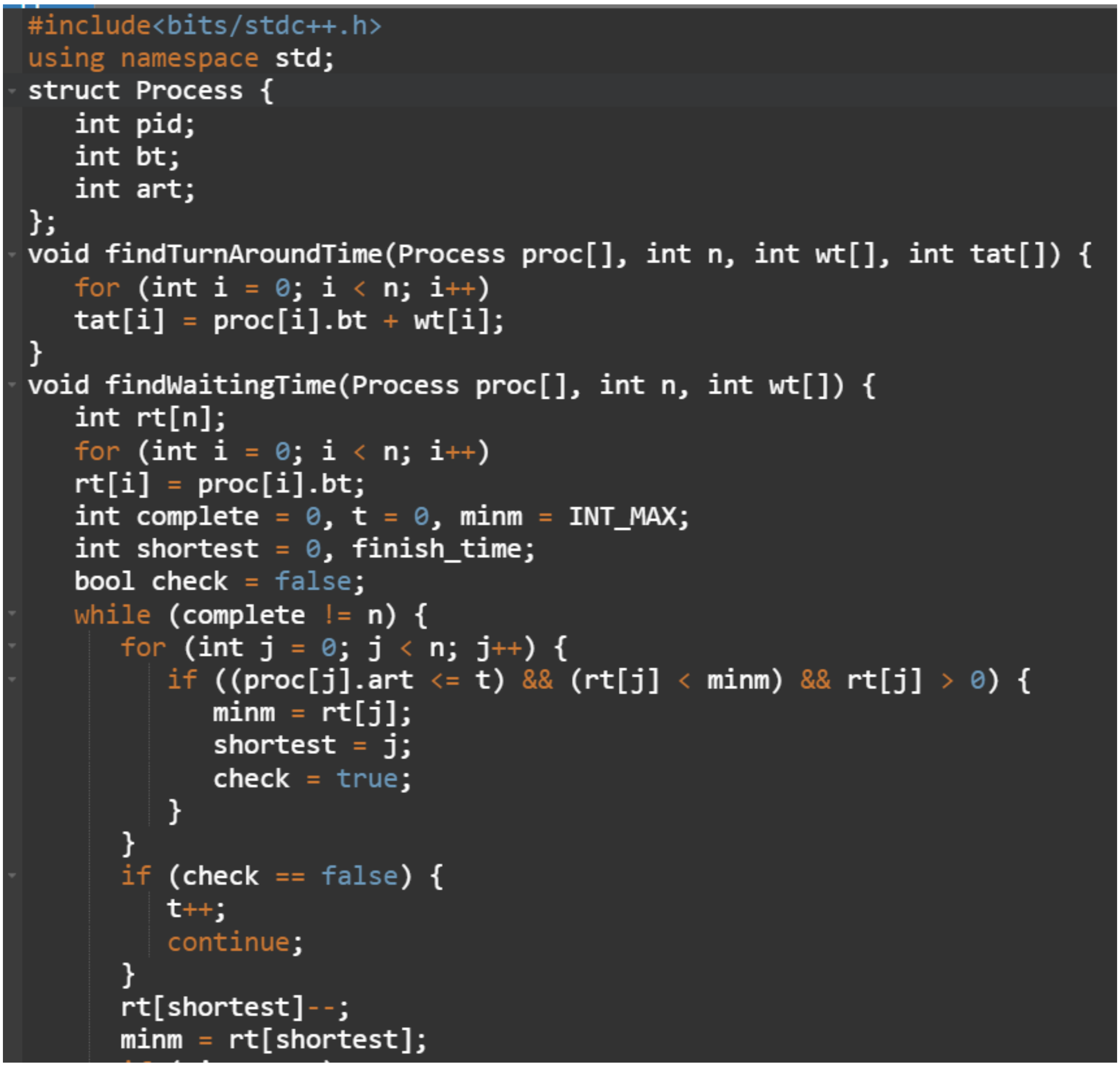


```
  }
  p[0].waiting_time=0;
  for(i=1;i<n;++i)
    p[i].waiting_time=p[i].response_time-p[i].arrival_time;
  for(i=0;i<n;++i)
    p[i].turnaround_time=p[i].waiting_time+p[i].burst_time;
}
void average(process *p,int n)
{
  float avg_wt=0,avg_tat=0;
  for(int i=0;i<n;++i)
  {
    avg_wt+=(float)p[i].waiting_time;
    avg_tat+=(float)p[i].turnaround_time;
  }
  avg_wt/=n;
  avg_tat/=n;
  cout<<"\n\nAVERAGE WAITING TIME : "<<avg_wt;
  cout<<"\nAVERAGE TURN AROUND TIME : "<<avg_tat;
}
void display(process *p,int n)
{
        cout<<"Processes "<<"  Burst time  "<<" Waiting time  "<<" Turn around time\n";
        for (int i=0;i<n;i++)
        {
cout<<"\n   "<<p[i].process_num<<"\t\t"<<p[i].burst_time<<"\t     "<<p[i].waiting_time<<"\t\t     "<<p[i].turnaround_time;
        }
       average(p,n);
}
```

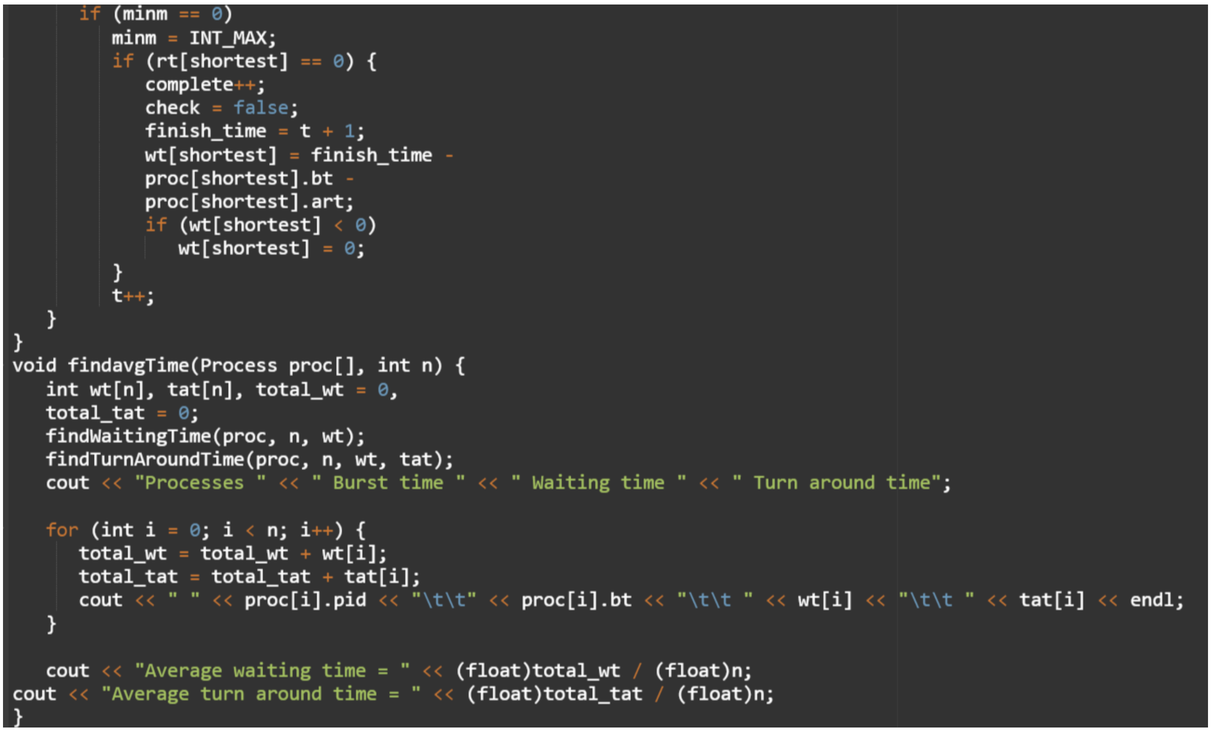


```
ENTER NUMBER OF PROCESSES : 5

ENTER BURST TIME FOR PROCESS 1 : 6
ENTER ARRIVAL TIME FOR PROCESS 1 : 2

ENTER BURST TIME FOR PROCESS 2 : 2
ENTER ARRIVAL TIME FOR PROCESS 2 : 5

ENTER BURST TIME FOR PROCESS 3 : 1
ENTER ARRIVAL TIME FOR PROCESS 3 : 8

ENTER BURST TIME FOR PROCESS 4 : 3
ENTER ARRIVAL TIME FOR PROCESS 4 : 0

ENTER BURST TIME FOR PROCESS 5 : 4
ENTER ARRIVAL TIME FOR PROCESS 5 : 4
Processes   Burst time   Waiting time   Turn around time

   4            3            0               3
   1            6            1               7
   5            4            5               9
   2            2            8               10
   3            1            7               8

AVERAGE WAITING TIME : 4.2
AVERAGE TURN AROUND TIME : 7.4
```

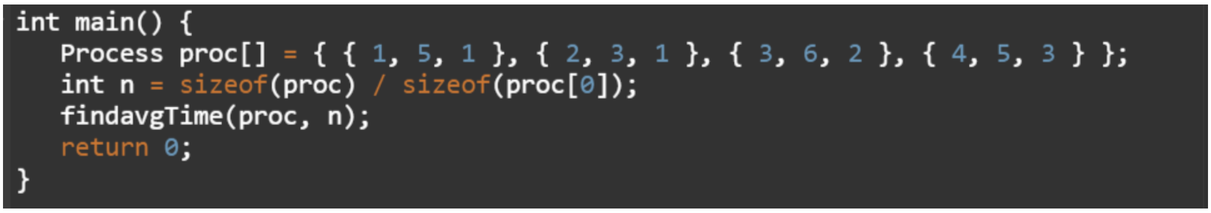
2. SJF



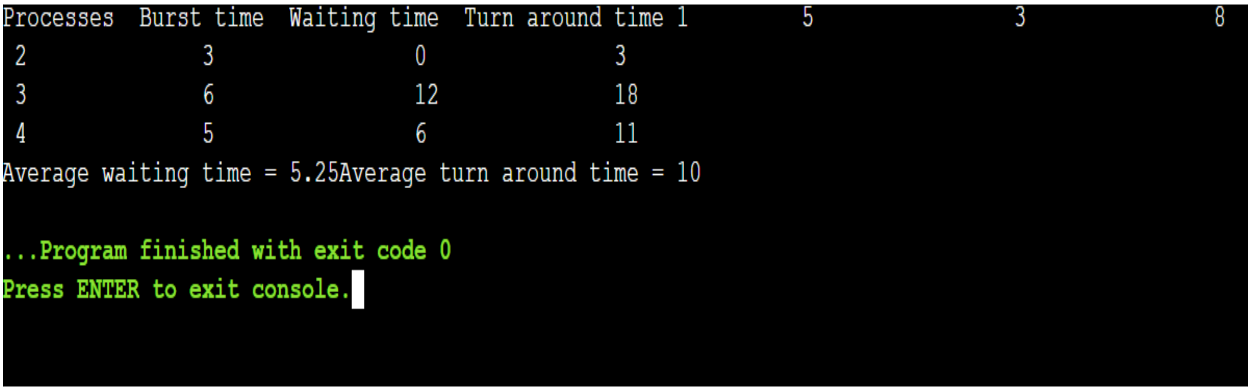
```
#include<bits/stdc++.h>
using namespace std;
struct Process {
    int pid;
    int bt;
    int art;
};
void findTurnAroundTime(Process proc[], int n, int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
    tat[i] = proc[i].bt + wt[i];
}
void findWaitingTime(Process proc[], int n, int wt[]) {
    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 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";

    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 << "Average waiting time = " << (float)total_wt / (float)n;
cout << "Average turn around time = " << (float)total_tat / (float)n;
}
```

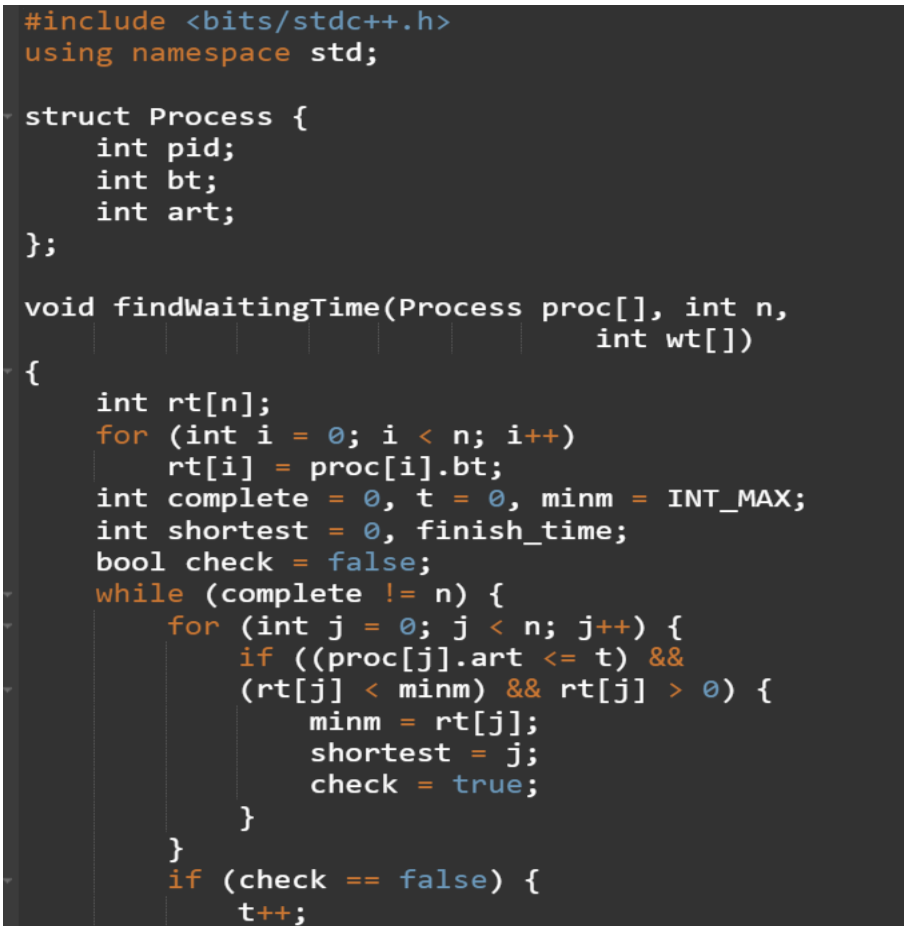


```c
int main() {
    Process proc[] = { { 1, 5, 1 }, { 2, 3, 1 }, { 3, 6, 2 }, { 4, 5, 3 } };
    int n = sizeof(proc) / sizeof(proc[0]);
    findavgTime(proc, n);
    return 0;
}
```

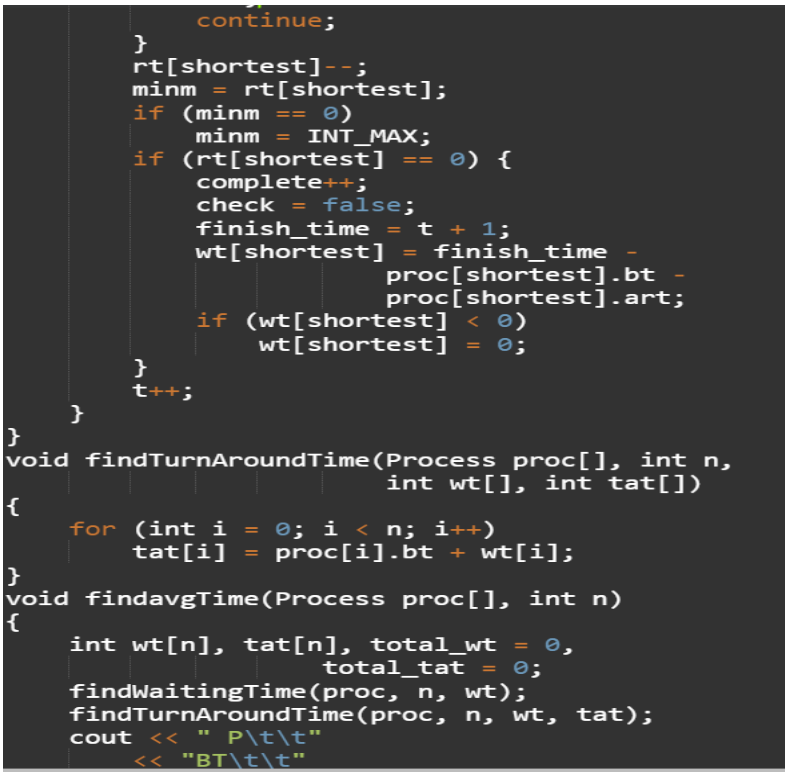


```
Processes  Burst time  Waiting time  Turn around time 1         5               3              8
 2              3               0              3
 3              6               12             18
 4              5               6              11
Average waiting time = 5.25Average turn around time = 10

...Program finished with exit code 0
Press ENTER to exit console.
```

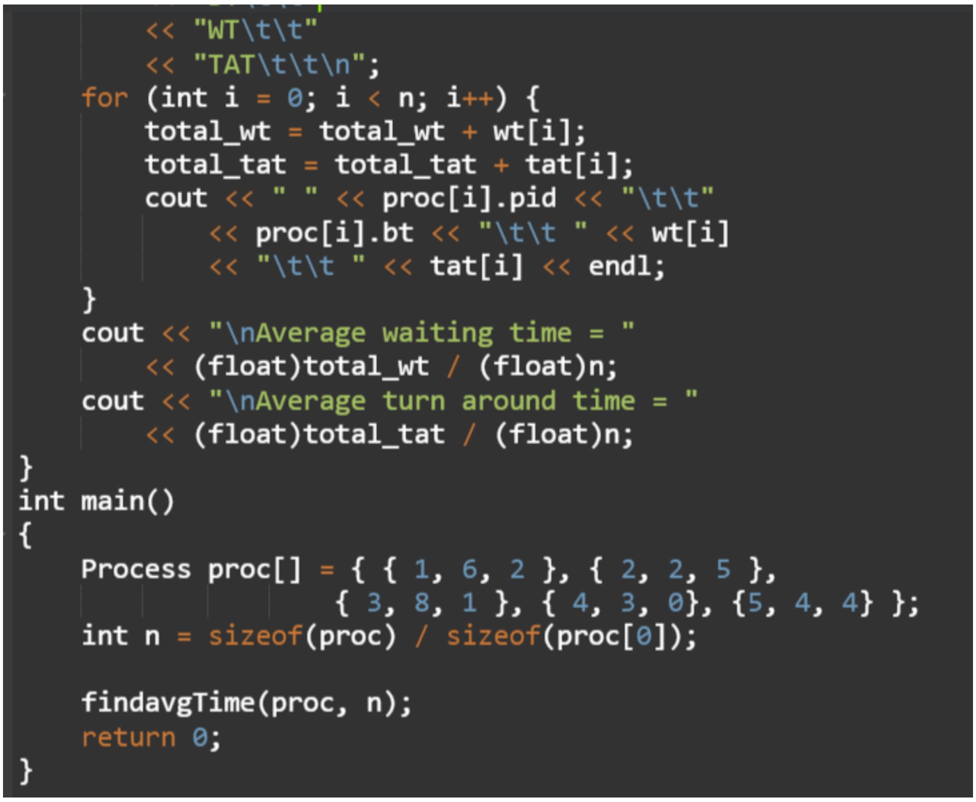
3.SRTF



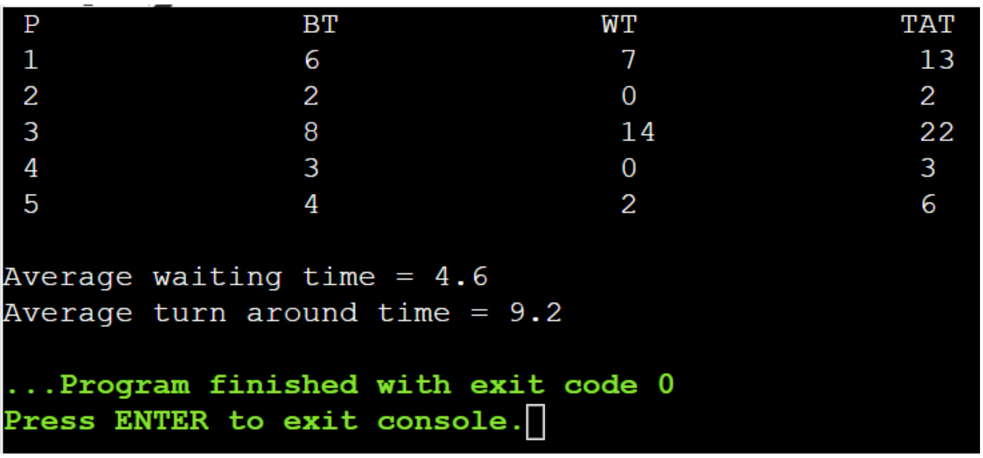
```
#include <bits/stdc++.h>
using namespace std;

struct Process {
    int pid;
    int bt;
    int art;
};

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 << " P\t\t"
        << "BT\t\t"
```

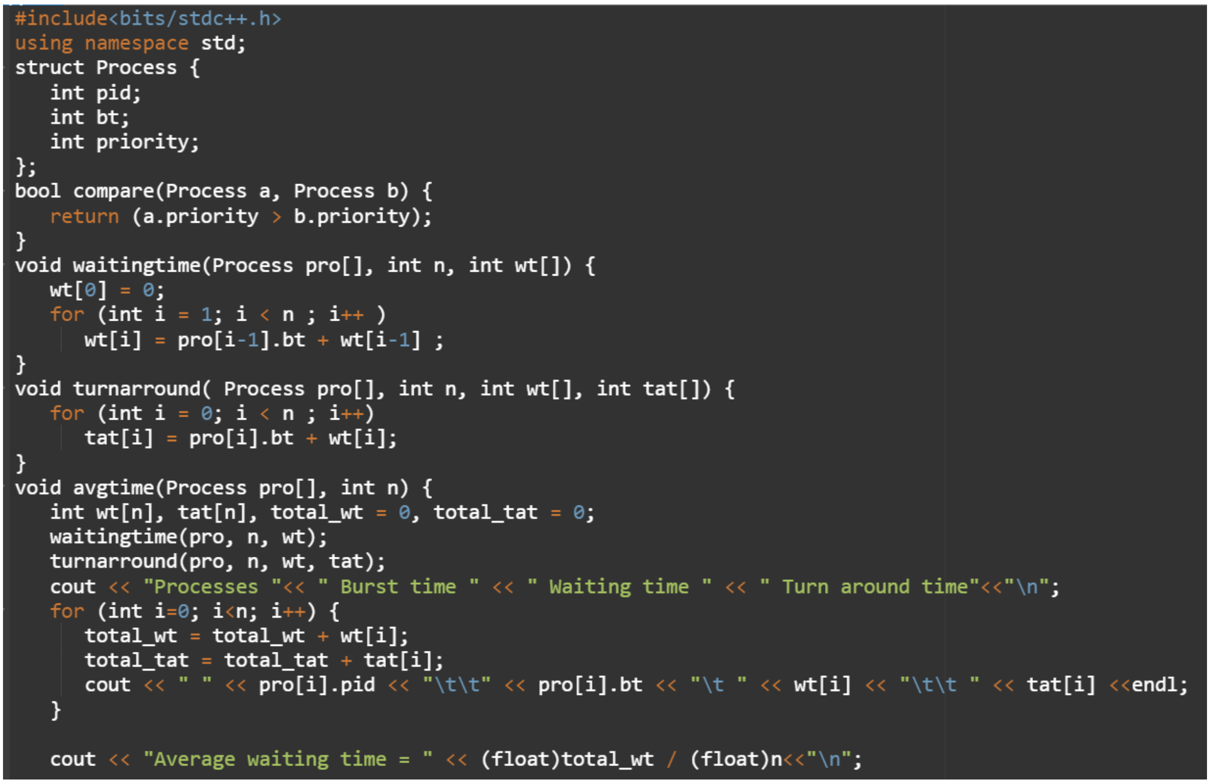


```
        << "WT\t\t"
        << "TAT\t\t\n";
    for (int i = 0; i < n; i++) {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
            << proc[i].bt << "\t\t " << wt[i]
            << "\t\t " << tat[i] << endl;
    }
    cout << "\nAverage waiting time = "
        << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
        << (float)total_tat / (float)n;
}
int main()
{
    Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },
                    { 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };
    int n = sizeof(proc) / sizeof(proc[0]);

    findavgTime(proc, n);
    return 0;
}
```

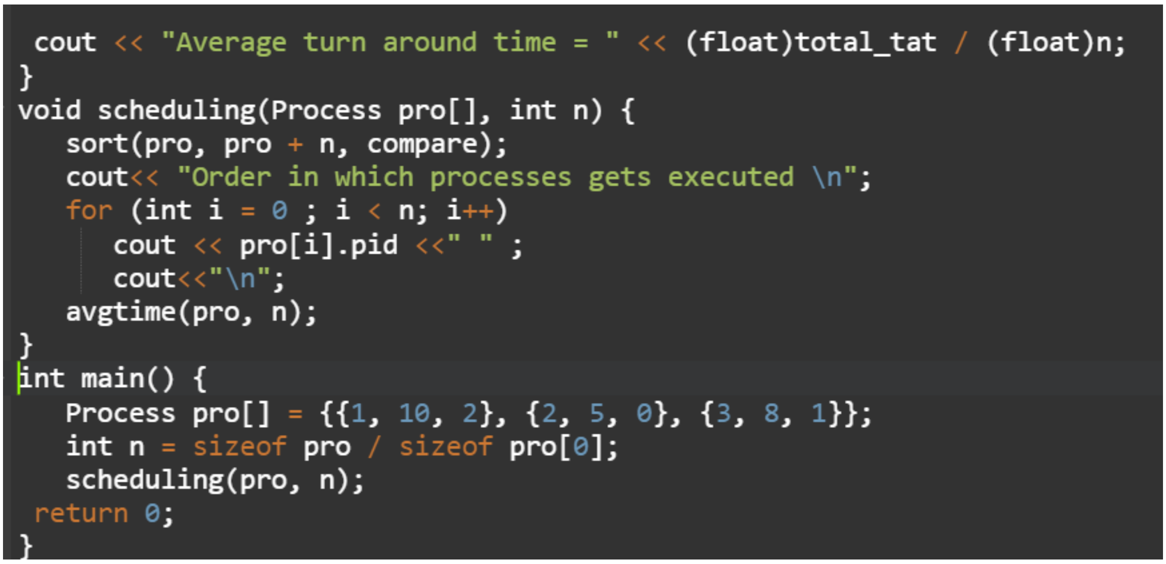


```
 P                  BT                  WT                  TAT
 1                  6                    7                   13
 2                  2                    0                   2
 3                  8                    14                  22
 4                  3                    0                   3
 5                  4                    2                   6

Average waiting time = 4.6
Average turn around time = 9.2

...Program finished with exit code 0
Press ENTER to exit console.
```

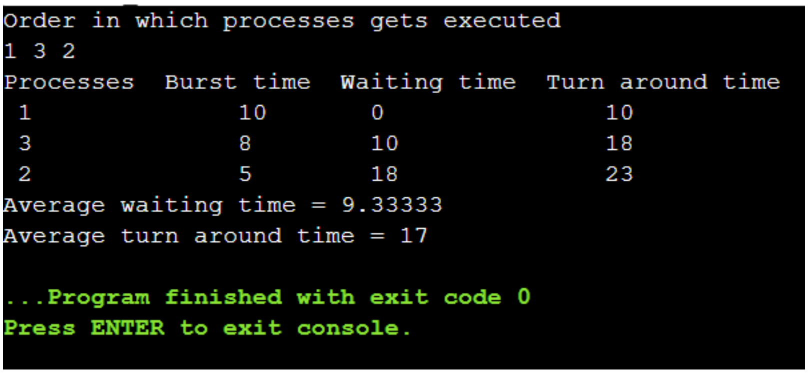
4. PRIORITY



```cpp
#include<bits/stdc++.h>
using namespace std;
struct Process {
   int pid;
   int bt;
   int priority;
};
bool compare(Process a, Process b) {
   return (a.priority > b.priority);
}
void waitingtime(Process pro[], int n, int wt[]) {
   wt[0] = 0;
   for (int i = 1; i < n ; i++ )
      wt[i] = pro[i-1].bt + wt[i-1] ;
}
void turnarround( Process pro[], int n, int wt[], int tat[]) {
   for (int i = 0; i < n ; i++)
      tat[i] = pro[i].bt + wt[i];
}
void avgtime(Process pro[], int n) {
   int wt[n], tat[n], total_wt = 0, total_tat = 0;
   waitingtime(pro, n, wt);
   turnarround(pro, 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 << " " << pro[i].pid << "\t\t" << pro[i].bt << "\t " << wt[i] << "\t\t " << tat[i] <<endl;
   }

   cout << "Average waiting time = " << (float)total_wt / (float)n<<"\n";
```

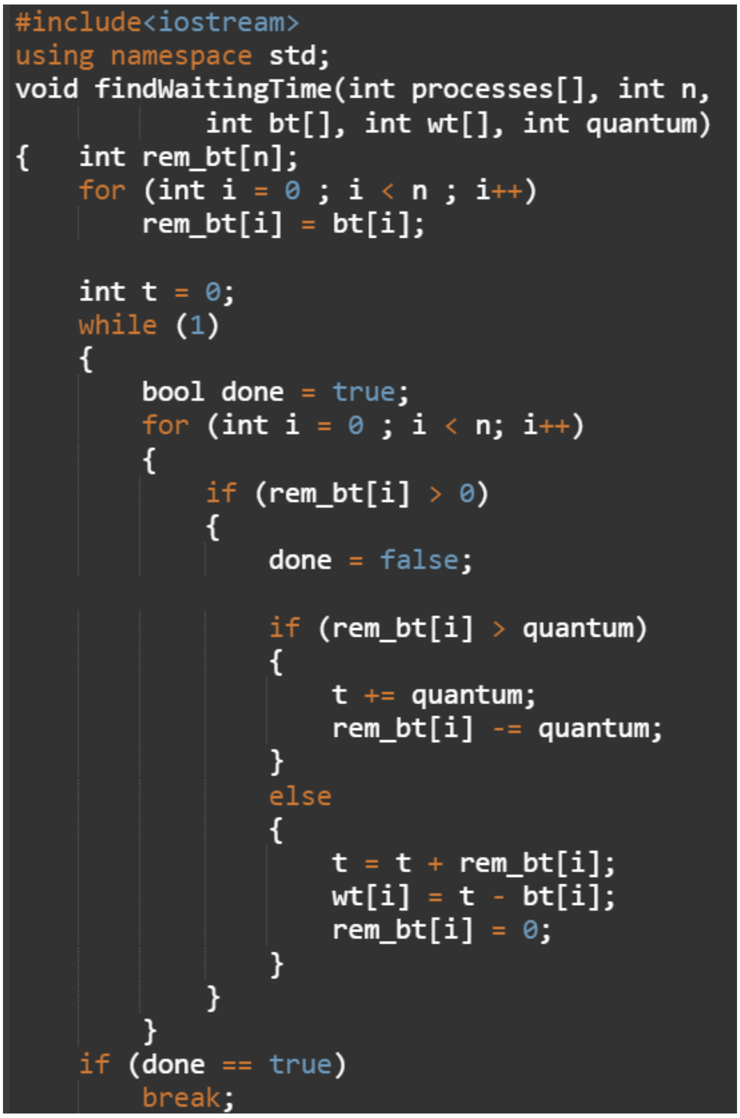


```cpp
 cout << "Average turn around time = " << (float)total_tat / (float)n;
}
void scheduling(Process pro[], int n) {
   sort(pro, pro + n, compare);
   cout<< "Order in which processes gets executed \n";
   for (int i = 0 ; i < n; i++)
      cout << pro[i].pid <<" " ;
      cout<<"\n";
   avgtime(pro, n);
}
int main() {
   Process pro[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};
   int n = sizeof pro / sizeof pro[0];
   scheduling(pro, n);
 return 0;
}
```

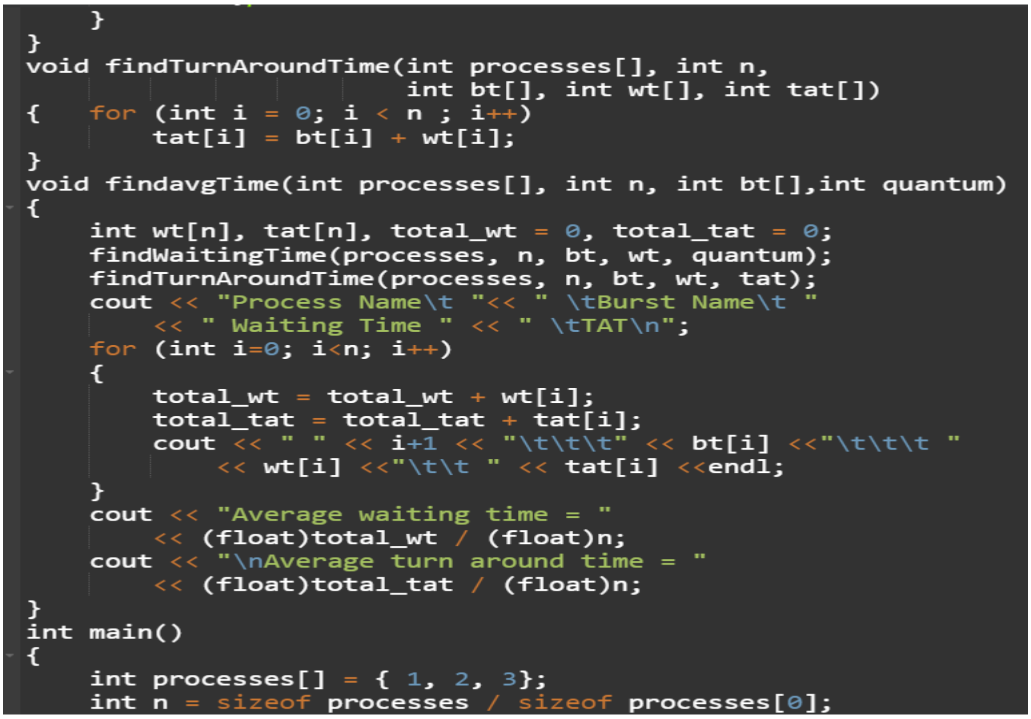


```
Order in which processes gets executed
1 3 2
Processes  Burst time  Waiting time  Turn around time
 1              10        0             10
 3              8         10            18
 2              5         18            23
Average waiting time = 9.33333
Average turn around time = 17

...Program finished with exit code 0
Press ENTER to exit console.
```

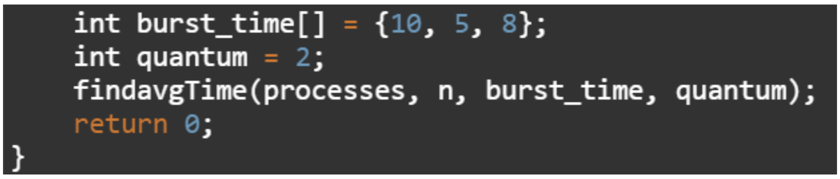
5. Round Robin



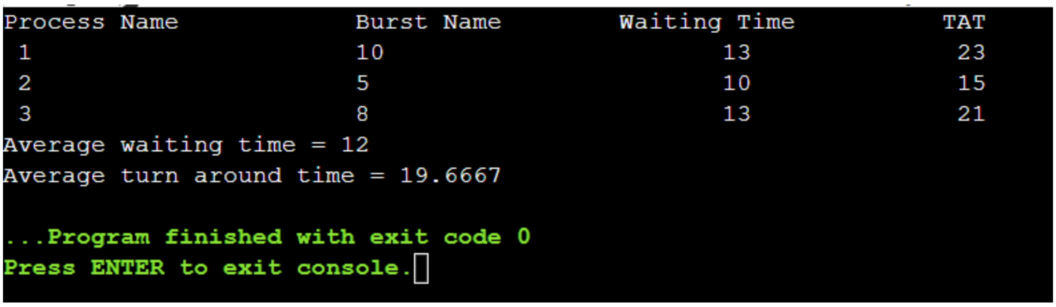
```
#include<iostream>
using namespace std;
void findWaitingTime(int processes[], int n,
            int bt[], int wt[], int quantum)
{   int rem_bt[n];
    for (int i = 0 ; i < n ; i++)
        rem_bt[i] = bt[i];

    int t = 0;
    while (1)
    {
        bool done = true;
        for (int i = 0 ; i < n; i++)
        {
            if (rem_bt[i] > 0)
            {
                done = false;

                if (rem_bt[i] > quantum)
                {
                    t += quantum;
                    rem_bt[i] -= quantum;
                }
                else
                {
                    t = t + rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
                }
            }
        }
        if (done == true)
            break;
```



```
    }
}
void findTurnAroundTime(int processes[], int n,
                        int bt[], int wt[], int tat[])
{   for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}
void findavgTime(int processes[], int n, int bt[],int quantum)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    findWaitingTime(processes, n, bt, wt, quantum);
    findTurnAroundTime(processes, n, bt, wt, tat);
    cout << "Process Name\t "<< " \tBurst Name\t "
        << " Waiting Time " << " \tTAT\n";
    for (int i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << i+1 << "\t\t\t" << bt[i] <<"\t\t\t "
            << wt[i] <<"\t\t " << tat[i] <<endl;
    }
    cout << "Average waiting time = "
        << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
        << (float)total_tat / (float)n;
}
int main()
{
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];
```



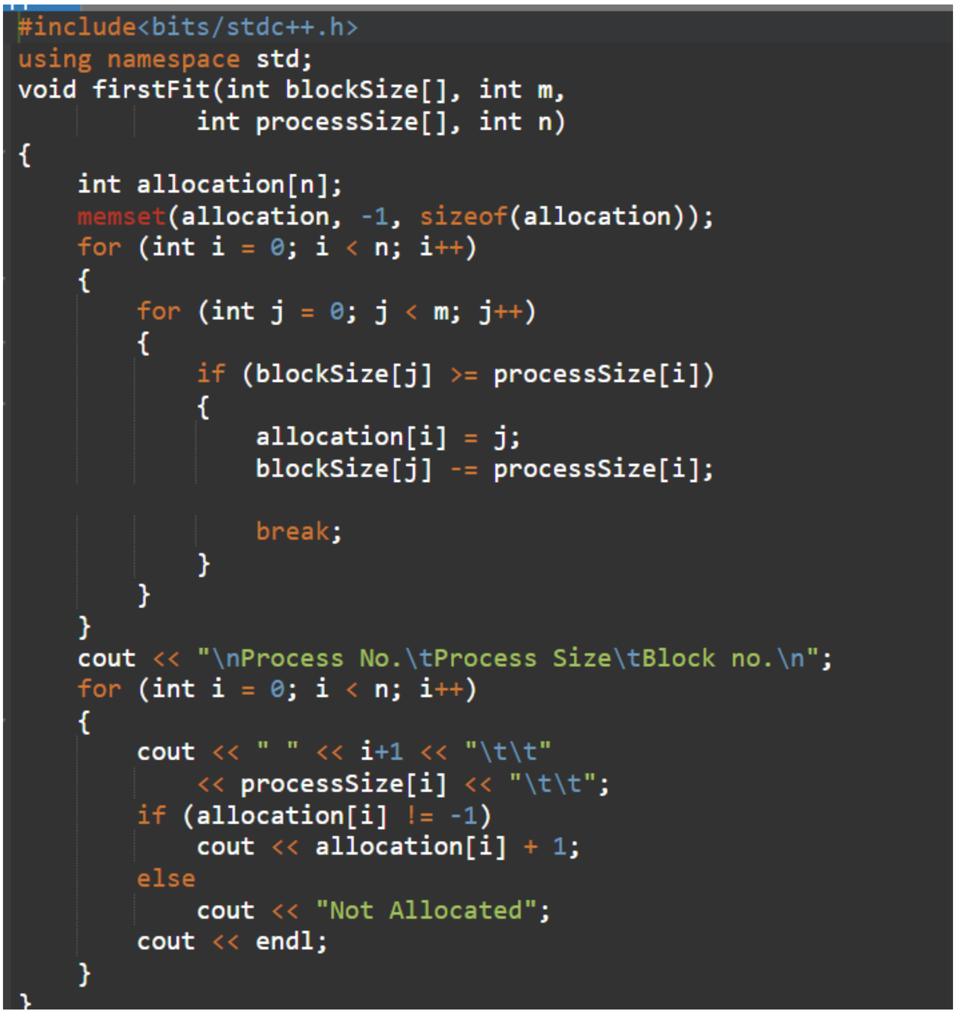
```
    int burst_time[] = {10, 5, 8};
    int quantum = 2;
    findavgTime(processes, n, burst_time, quantum);
    return 0;
}
```



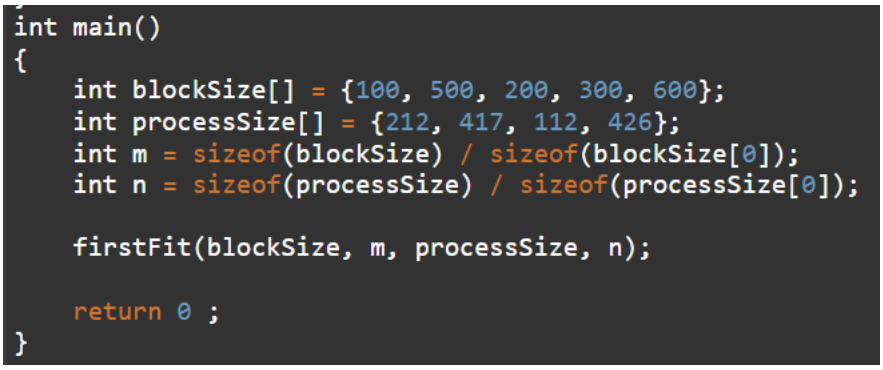
```
Process Name              Burst Name          Waiting Time          TAT
 1                        10                         13              23
 2                        5                          10              15
 3                        8                          13              21
Average waiting time = 12
Average turn around time = 19.6667

...Program finished with exit code 0
Press ENTER to exit console.
```

MEMORY MANAGEMENT

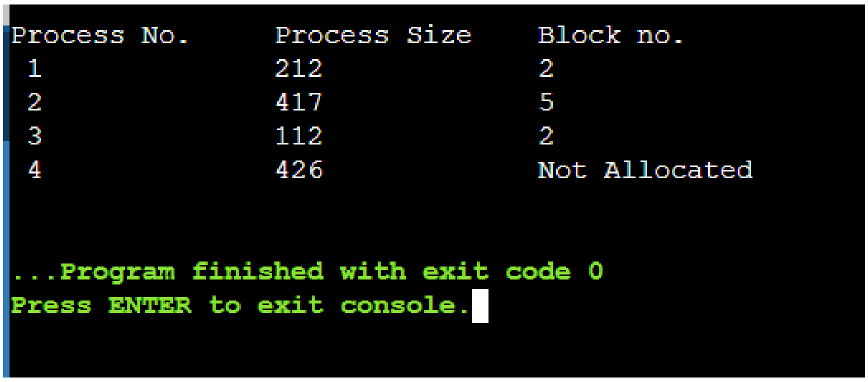
1. First Fit



```
#include<bits/stdc++.h>
using namespace std;
void firstFit(int blockSize[], int m,
              int processSize[], int n)
{
    int allocation[n];
    memset(allocation, -1, sizeof(allocation));
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                allocation[i] = j;
                blockSize[j] -= processSize[i];

                break;
            }
        }
    }
    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << " " << i+1 << "\t\t"
             << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}
```

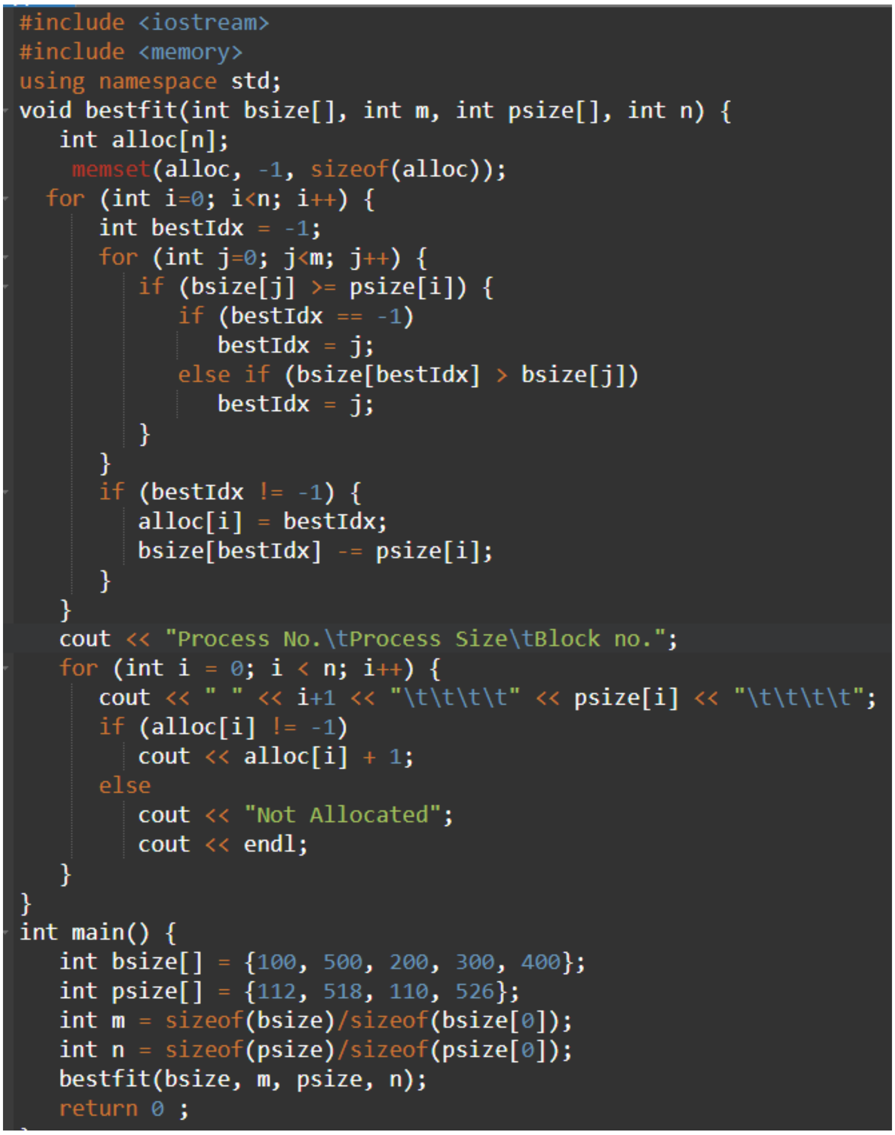


```
int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

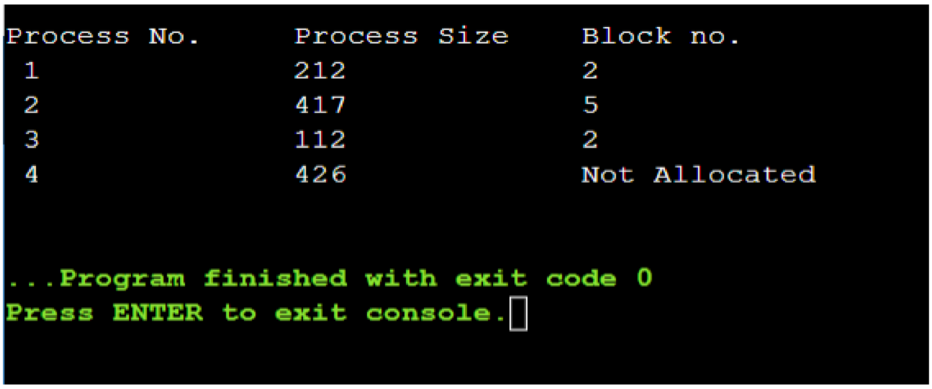


```
Process No.     Process Size    Block no.
 1              212             2
 2              417             5
 3              112             2
 4              426             Not Allocated


...Program finished with exit code 0
Press ENTER to exit console.
```

2. Best Fit

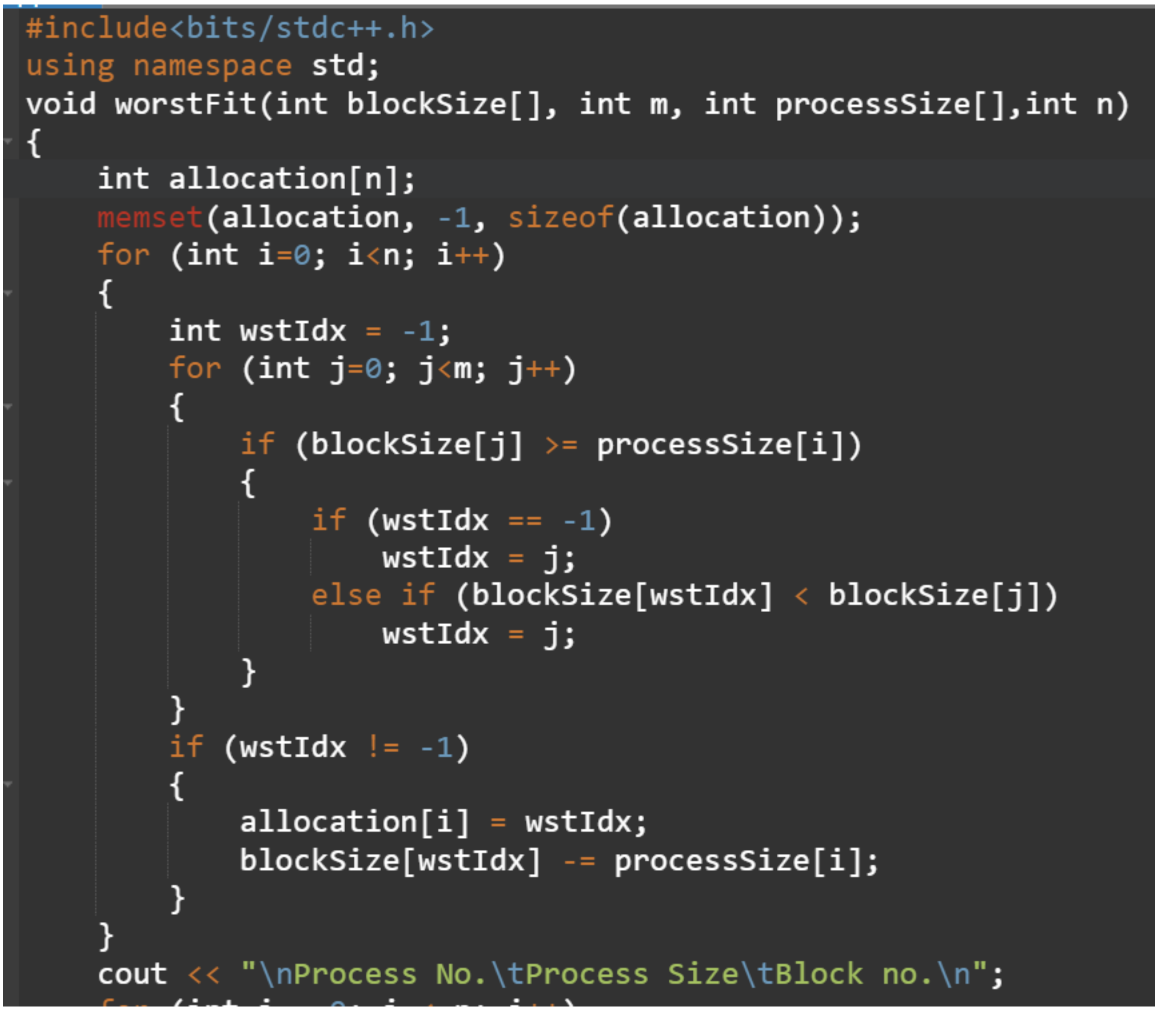


```cpp
#include <iostream>
#include <memory>
using namespace std;
void bestfit(int bsize[], int m, int psize[], int n) {
    int alloc[n];
     memset(alloc, -1, sizeof(alloc));
    for (int i=0; i<n; i++) {
        int bestIdx = -1;
        for (int j=0; j<m; j++) {
            if (bsize[j] >= psize[i]) {
                if (bestIdx == -1)
                    bestIdx = j;
                else if (bsize[bestIdx] > bsize[j])
                    bestIdx = j;
            }
        }
        if (bestIdx != -1) {
            alloc[i] = bestIdx;
            bsize[bestIdx] -= psize[i];
        }
    }
    cout << "Process No.\tProcess Size\tBlock no.";
    for (int i = 0; i < n; i++) {
        cout << " " << i+1 << "\t\t\t\t" << psize[i] << "\t\t\t\t";
        if (alloc[i] != -1)
            cout << alloc[i] + 1;
        else
            cout << "Not Allocated";
            cout << endl;
    }
}
int main() {
    int bsize[] = {100, 500, 200, 300, 400};
    int psize[] = {112, 518, 110, 526};
    int m = sizeof(bsize)/sizeof(bsize[0]);
    int n = sizeof(psize)/sizeof(psize[0]);
    bestfit(bsize, m, psize, n);
    return 0 ;
```

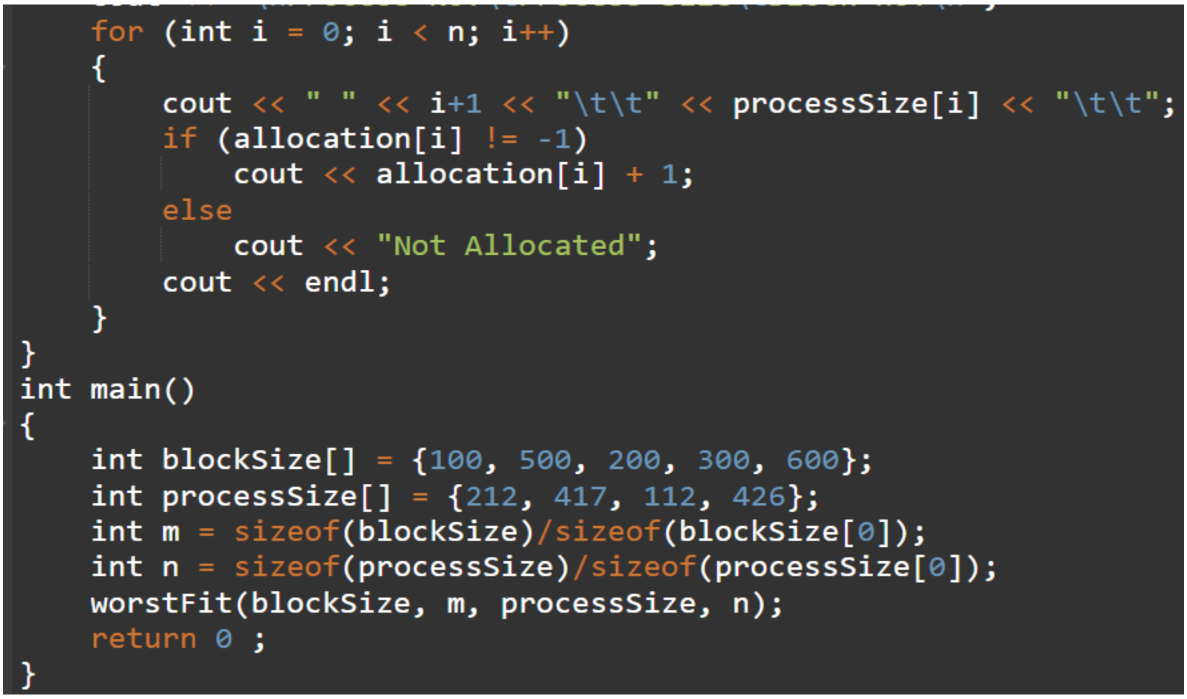


```
Process No.         Process Size        Block no.
 1                  212                 2
 2                  417                 5
 3                  112                 2
 4                  426                 Not Allocated


...Program finished with exit code 0
Press ENTER to exit console.
```

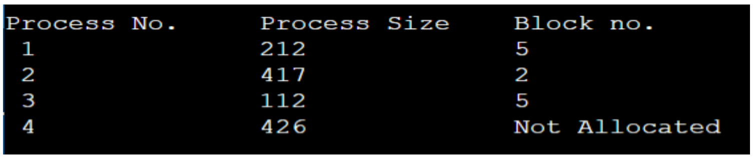
3. Worst Fit



```
#include<bits/stdc++.h>
using namespace std;
void worstFit(int blockSize[], int m, int processSize[],int n)
{
    int allocation[n];
    memset(allocation, -1, sizeof(allocation));
    for (int i=0; i<n; i++)
    {
        int wstIdx = -1;
        for (int j=0; j<m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (wstIdx == -1)
                    wstIdx = j;
                else if (blockSize[wstIdx] < blockSize[j])
                    wstIdx = j;
            }
        }
        if (wstIdx != -1)
        {
            allocation[i] = wstIdx;
            blockSize[wstIdx] -= processSize[i];
        }
    }
    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
```



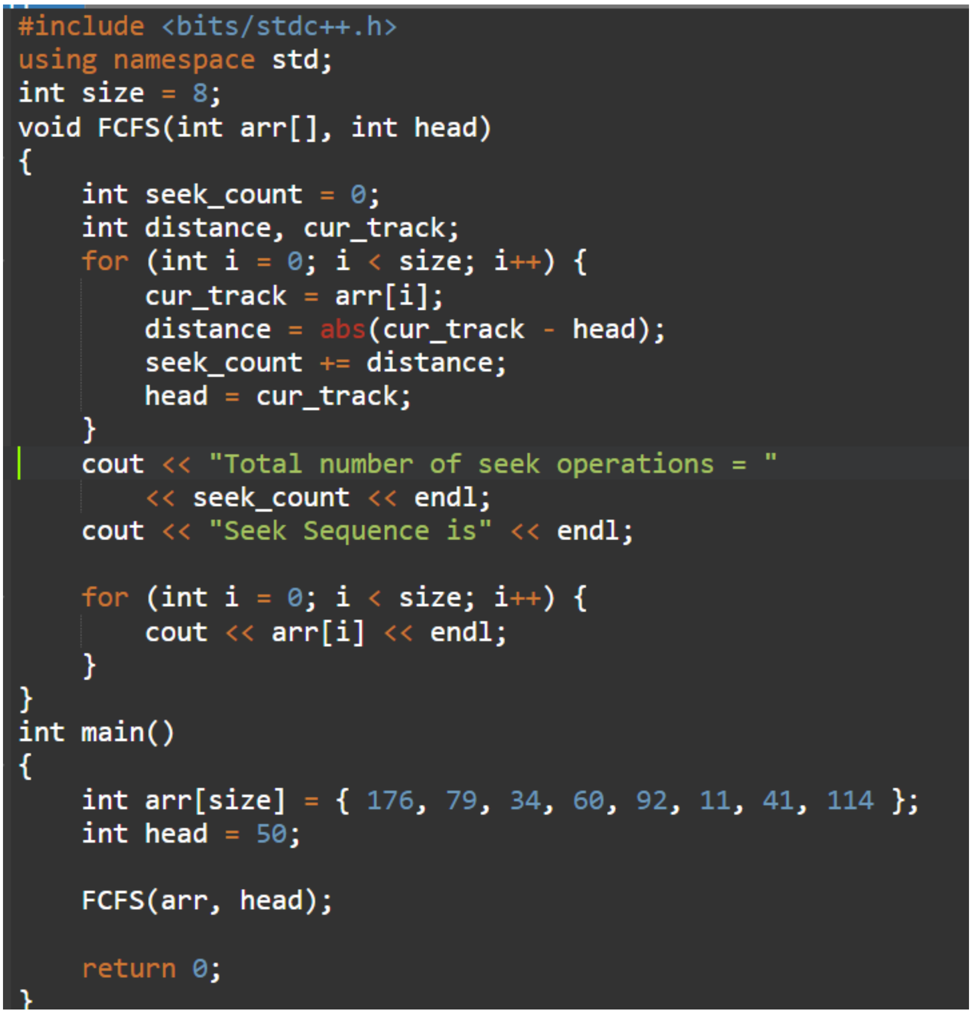
```
    for (int i = 0; i < n; i++)
    {
        cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}
int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);
    worstFit(blockSize, m, processSize, n);
    return 0 ;
}
```



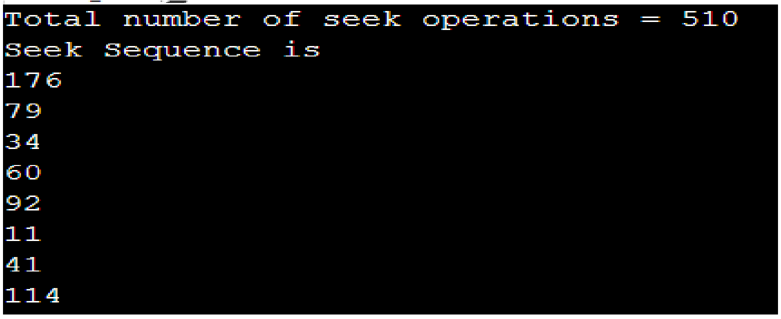
```
Process No.     Process Size    Block no.
 1              212             5
 2              417             2
 3              112             5
 4              426             Not Allocated
```

DISK SCHEDULING

1. FCFS

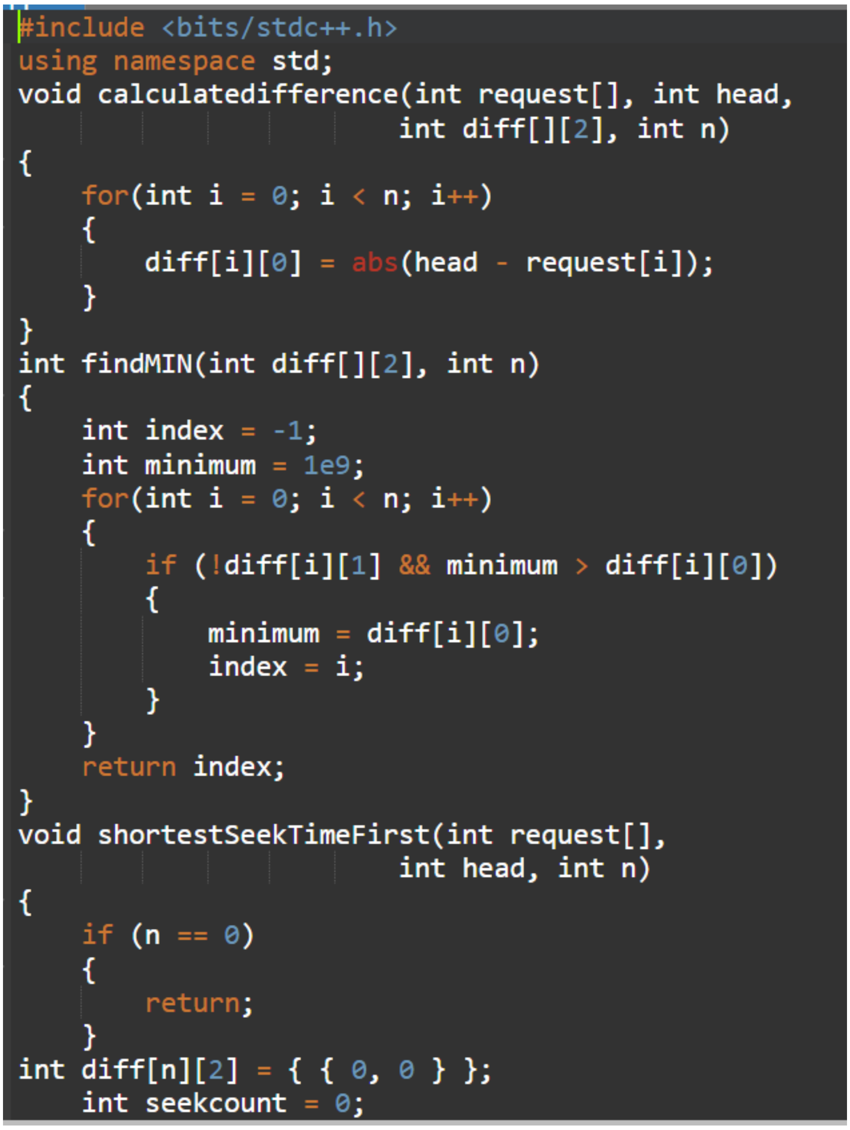


```
#include <bits/stdc++.h>
using namespace std;
int size = 8;
void FCFS(int arr[], int head)
{
    int seek_count = 0;
    int distance, cur_track;
    for (int i = 0; i < size; i++) {
        cur_track = arr[i];
        distance = abs(cur_track - head);
        seek_count += distance;
        head = 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;
}
```

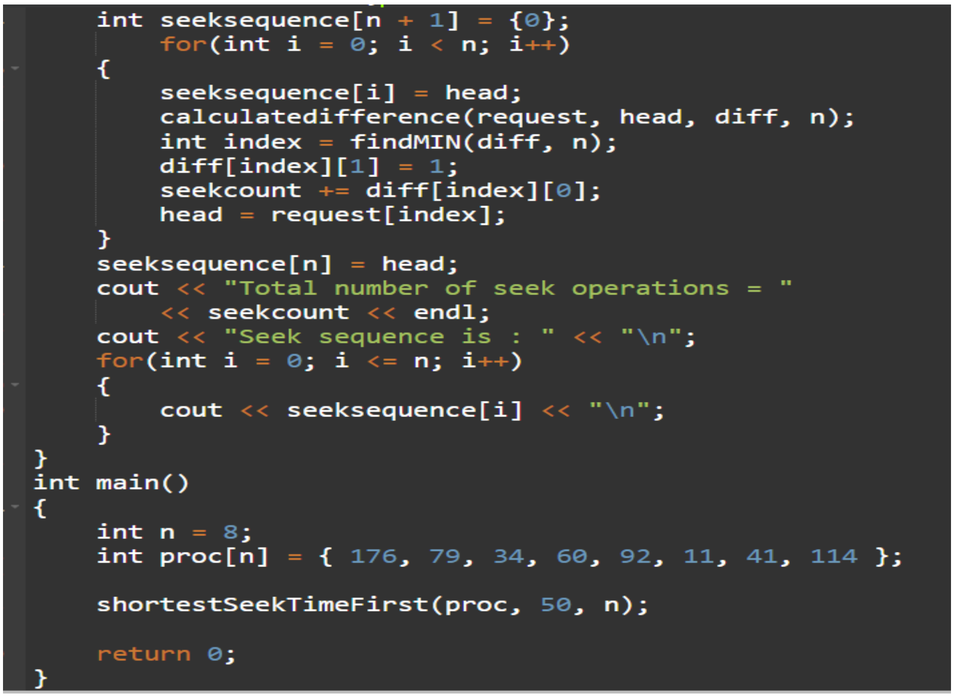


```
Total number of seek operations = 510
Seek Sequence is
176
79
34
60
92
11
41
114
```

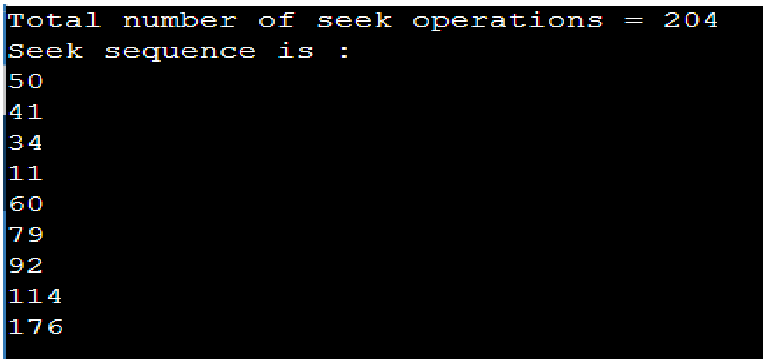
2. Shortest seek time first



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

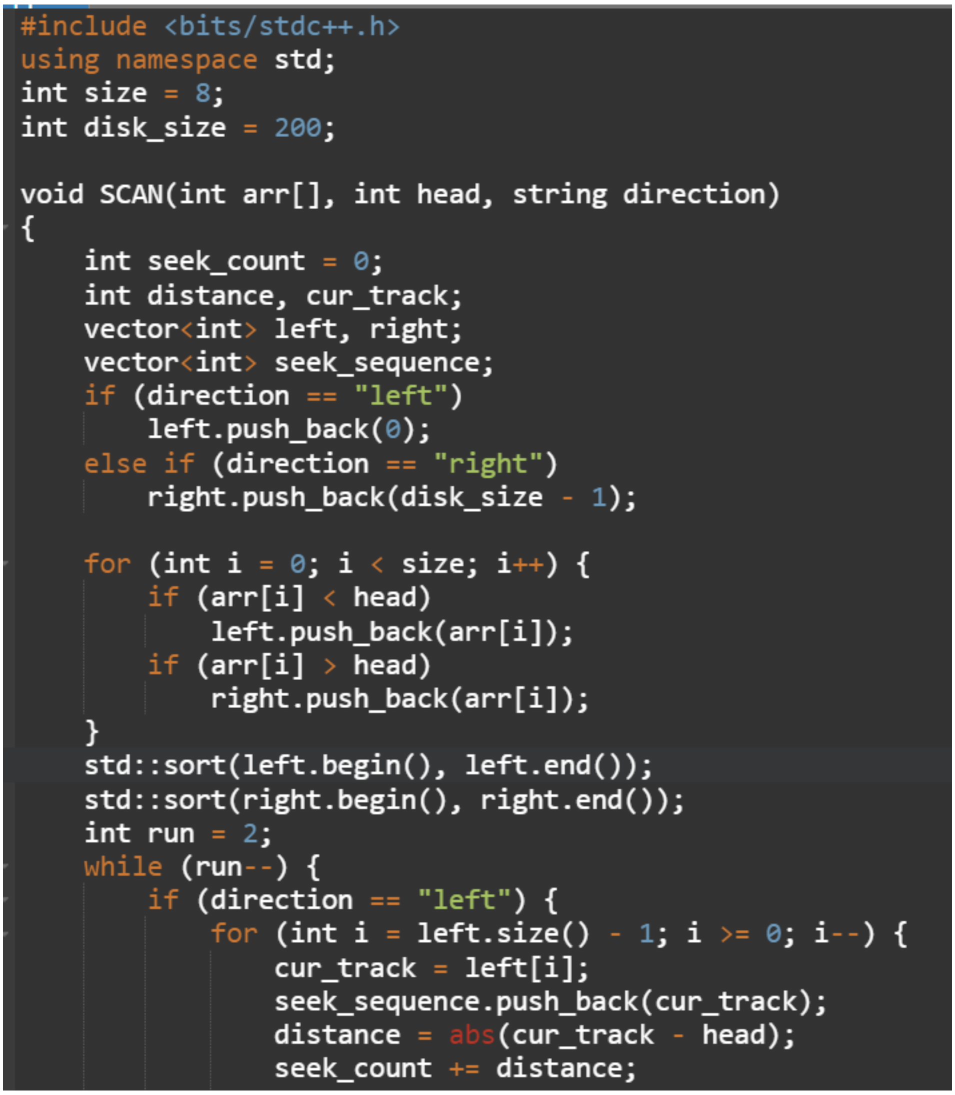


```
    int seeksequence[n + 1] = {0};
        for(int i = 0; i < n; i++)
    {
        seeksequence[i] = head;
        calculatedifference(request, head, diff, n);
        int index = findMIN(diff, n);
        diff[index][1] = 1;
        seekcount += diff[index][0];
        head = request[index];
    }
    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;
}
```

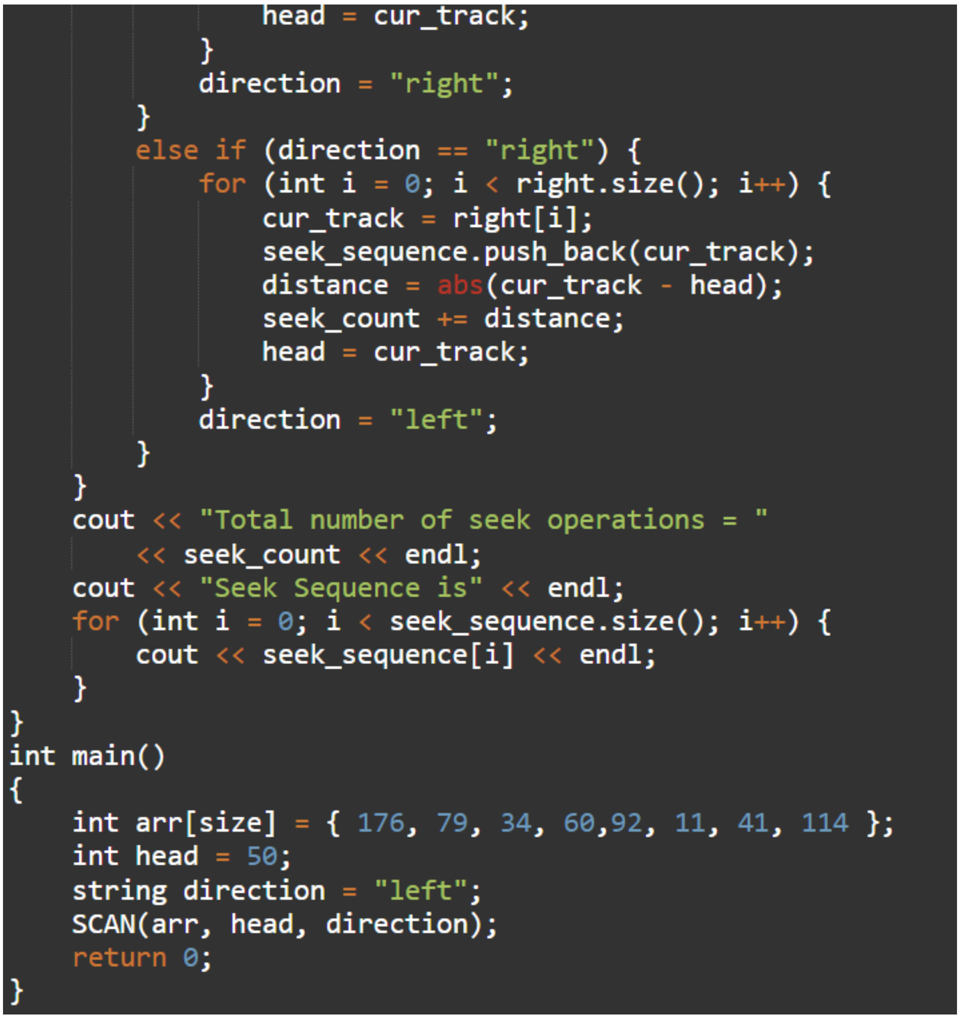


```
Total number of seek operations = 204
Seek sequence is :
50
41
34
11
60
79
92
114
176
```

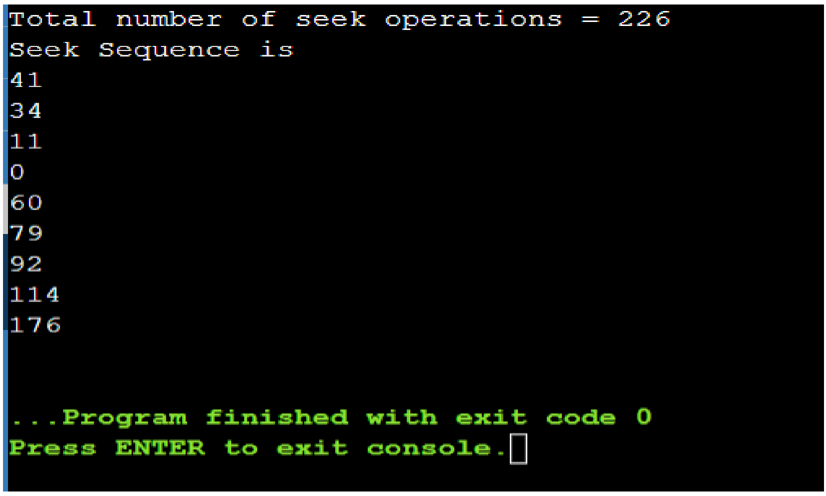
3. SCAN DISK SCHEDULING



```cpp
#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--) {
                cur_track = left[i];
                seek_sequence.push_back(cur_track);
                distance = abs(cur_track - head);
                seek_count += distance;
```

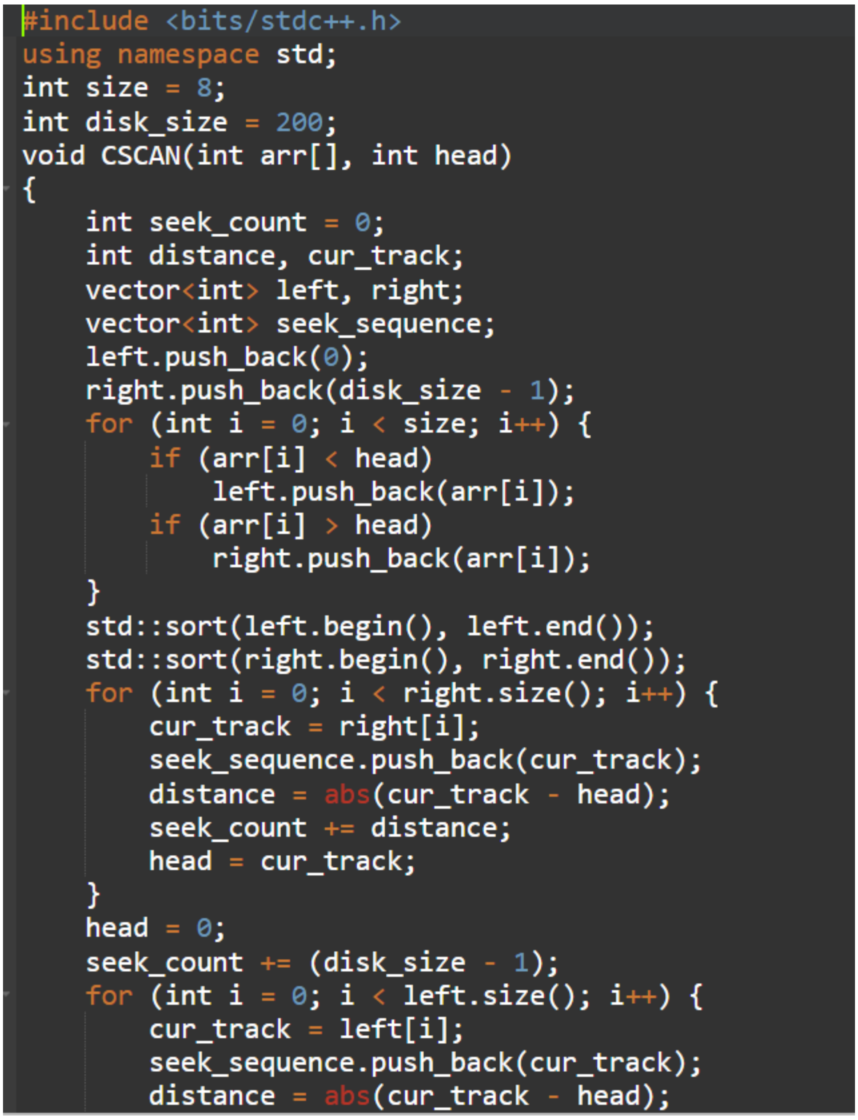


```
                head = cur_track;
            }
            direction = "right";
        }
        else if (direction == "right") {
            for (int i = 0; i < right.size(); i++) {
                cur_track = right[i];
                seek_sequence.push_back(cur_track);
                distance = 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;
}
```

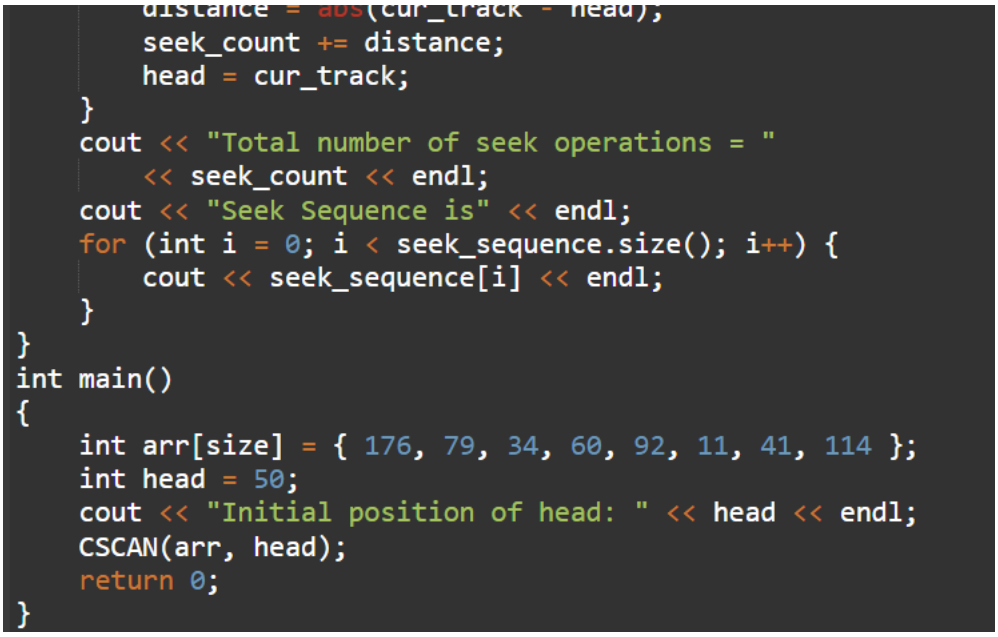


```
Total number of seek operations = 226
Seek Sequence is
41
34
11
0
60
79
92
114
176


...Program finished with exit code 0
Press ENTER to exit console.
```

4. C-SCAN DISK SCHEDULING

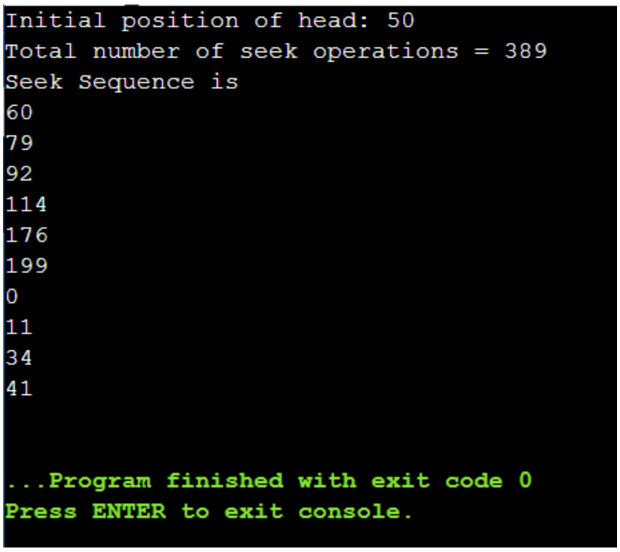


```
#include <bits/stdc++.h>
using namespace std;
int size = 8;
int disk_size = 200;
void CSCAN(int arr[], int head)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;
    left.push_back(0);
    right.push_back(disk_size - 1);
    for (int 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);
        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);
```



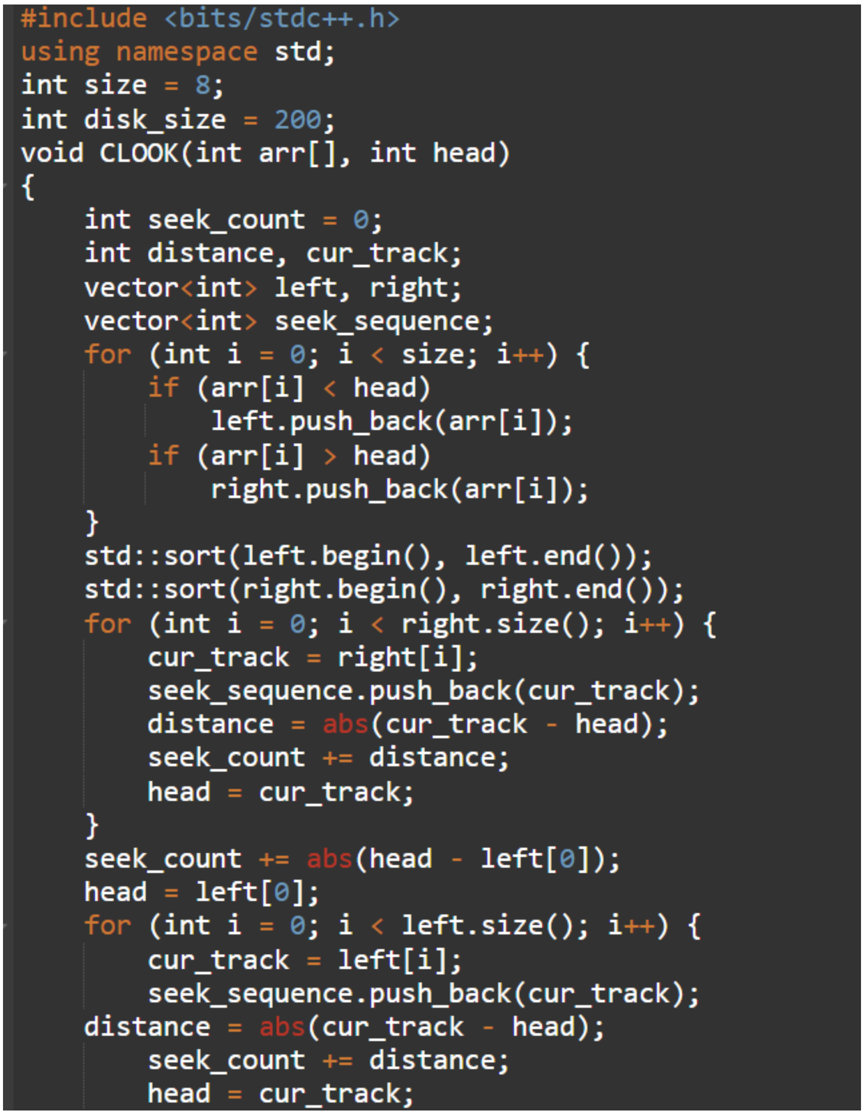
distance = abs(cur_track - head);

```
        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;
}
```

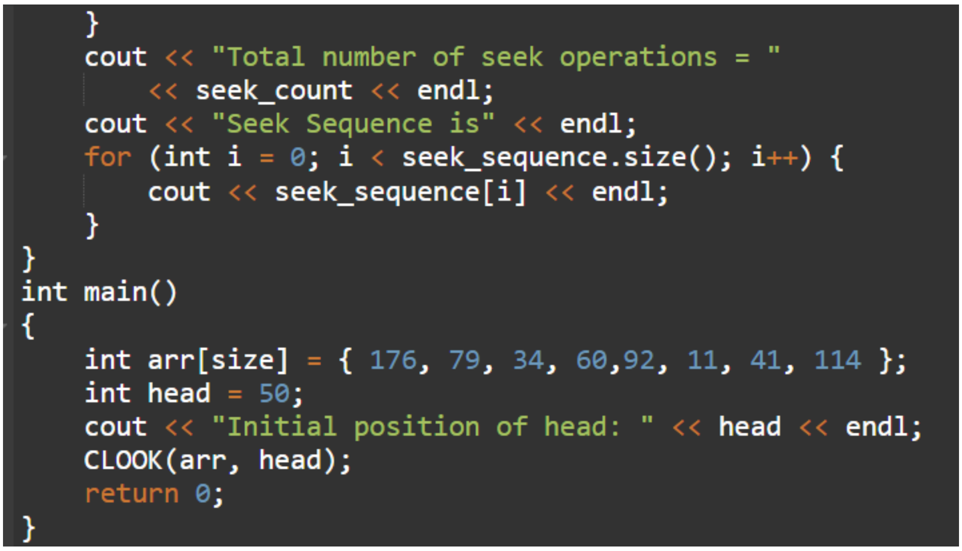


```
Initial position of head: 50
Total number of seek operations = 389
Seek Sequence is
60
79
92
114
176
199
0
11
34
41


...Program finished with exit code 0
Press ENTER to exit console.
```

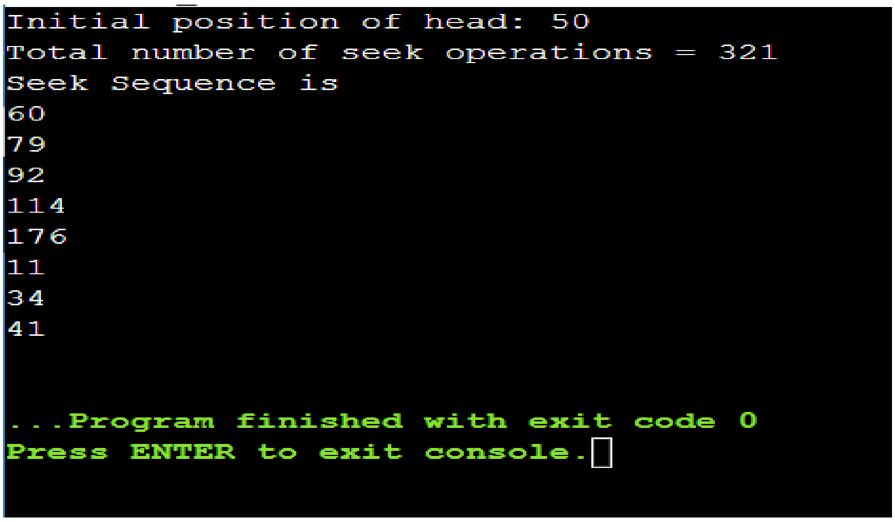
5. C-LOOK



```
#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);
        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;
```

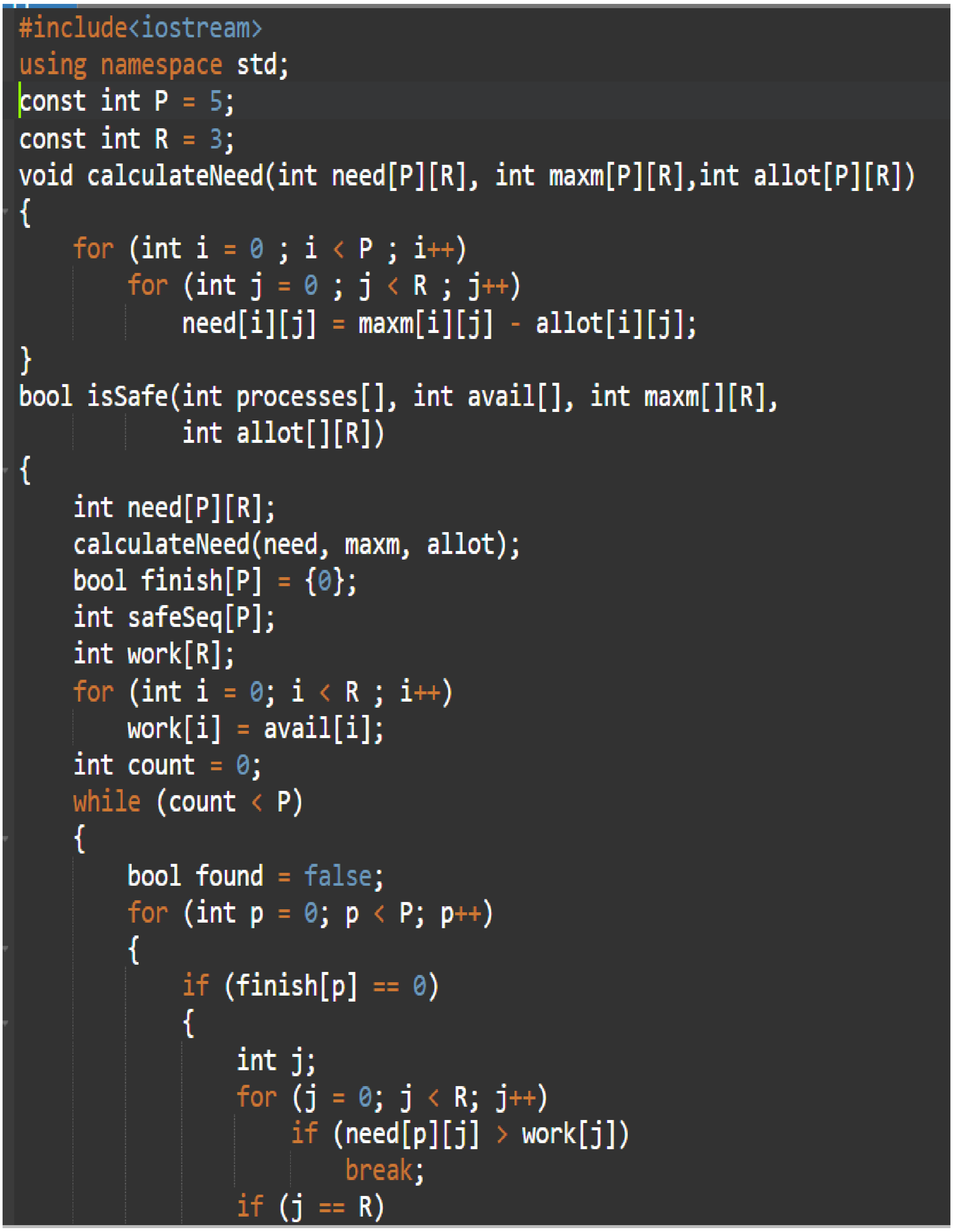


```
    }
    cout << "Total number of seek operations = "
        << seek_count << endl;
    cout << "Seek Sequence is" << endl;
    for (int i = 0; i < seek_sequence.size(); i++) {
        cout << seek_sequence[i] << endl;
    }
}
int main()
{
    int arr[size] = { 176, 79, 34, 60,92, 11, 41, 114 };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

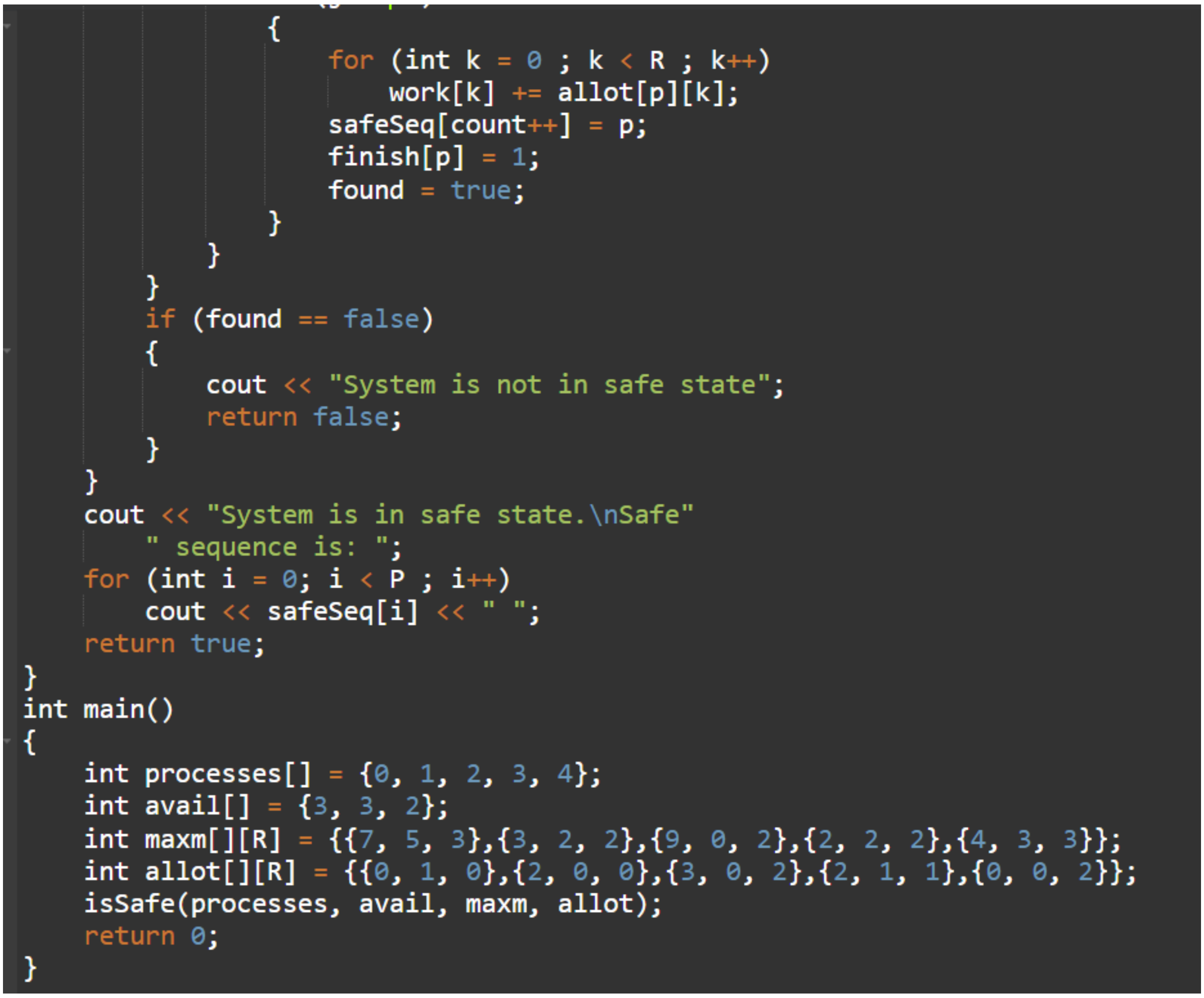


```
Initial position of head: 50
Total number of seek operations = 321
Seek Sequence is
60
79
92
114
176
11
34
41


...Program finished with exit code 0
Press ENTER to exit console.
```

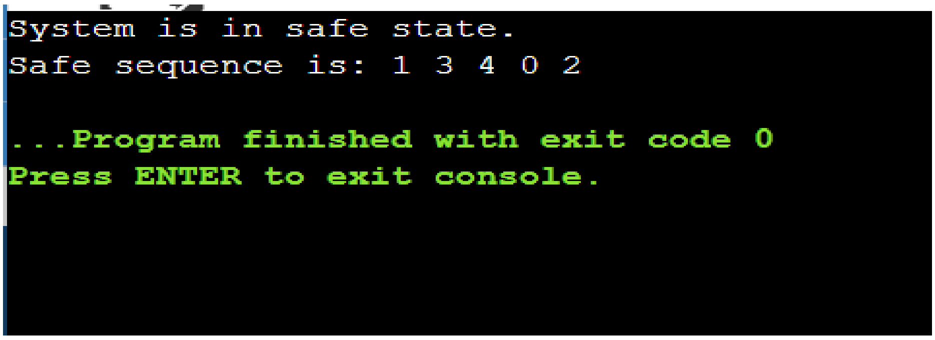
BANKER’S ALGORITHM



```
#include<iostream>
using namespace std;
const int P = 5;
const int R = 3;
void calculateNeed(int need[P][R], int maxm[P][R],int allot[P][R])
{
    for (int i = 0 ; i < P ; i++)
        for (int j = 0 ; j < R ; j++)
            need[i][j] = maxm[i][j] - allot[i][j];
}
bool isSafe(int processes[], int avail[], int maxm[][R],
            int allot[][R])
{
    int need[P][R];
    calculateNeed(need, maxm, allot);
    bool finish[P] = {0};
    int safeSeq[P];
    int work[R];
    for (int i = 0; i < R ; i++)
        work[i] = avail[i];
    int count = 0;
    while (count < P)
    {
        bool found = false;
        for (int p = 0; p < P; p++)
        {
            if (finish[p] == 0)
            {
                int j;
                for (j = 0; j < R; j++)
                    if (need[p][j] > work[j])
                        break;
                if (j == R)
```



```
                {
                    for (int k = 0 ; k < R ; k++)
                        work[k] += allot[p][k];
                    safeSeq[count++] = p;
                    finish[p] = 1;
                    found = true;
                }
            }
        }
        if (found == false)
        {
            cout << "System is not in safe state";
            return false;
        }
    }
    cout << "System is in safe state.\nSafe"
        " sequence is: ";
    for (int i = 0; i < P ; i++)
        cout << safeSeq[i] << " ";
    return true;
}
int main()
{
    int processes[] = {0, 1, 2, 3, 4};
    int avail[] = {3, 3, 2};
    int maxm[][R] = {{7, 5, 3},{3, 2, 2},{9, 0, 2},{2, 2, 2},{4, 3, 3}};
    int allot[][R] = {{0, 1, 0},{2, 0, 0},{3, 0, 2},{2, 1, 1},{0, 0, 2}};
    isSafe(processes, avail, maxm, allot);
    return 0;
}
```



```
System is in safe state.
Safe sequence is: 1 3 4 0 2

...Program finished with exit code 0
Press ENTER to exit console.
```