# OPERATING SYSTEMS

(CECSC09 -I)


Submitted By:-

**Ashish Kumar**

**2019UCO1518**

# INDEX

**Program 1 : Linux Shell Commands**

**mkdir**: Used to create a directory if not already exist. It accepts the directory name as an input parameter.

**rmdir**: It is used to delete a directory if it is empty.

**touch**: Used to create or update a file.

**cat**: It is generally used to concatenate the files. It gives the output on the standard output.

**more**: It is a filter for paging through text one screenful at a time.

**cd**: Used to change the directory.

**touch**: Used to create or update a file.

**ls**: To get the list of all the files or folders.

**rm**: Used to remove files or directories.

**wc**: Used to count the number of characters, words in a file.

**sort**: This command is used to sort the contents of files.

**grep**: This command is used to search for the specified text in a file.

**head**: Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.

**tail**: The tail command displays the last part (10 lines by default) of one or more files or piped data. It can be also used to monitor the file changes in real time

**PROGRAM 2 : Write C programs using fork(), getpid(), getppid() and exec() system calls.**

**CODE:**

FILE 1 :

```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
using namespace std;
int main(int argc, char *argv[])
{
        cout << "Process Creation and Termination\n";
        pid_t id1, id2;
        id1 = fork();
        id2 = fork();
        if (id1 == 0 && id2 > 0)
        {
                cout << "This is First child Process; pid= " << getpid() << "
                   Parent's pid=" << getppid() << endl;
                exit(0);
        }
        else if (id1 > 0 && id2 == 0)
        {
                cout << "This is Second child Process; pid= " << getpid() <
                   < "Parent's pid=" << getppid() << endl;
                exit(0);
        }
        else if (id1 == 0 && id2 == 0)
        {
                cout << "This is Third child Process; pid= " << getpid() << "
                   Parent's pid=" << getppid() << endl;
                exit(0);
        }
        else if (id1 > 0 && id2 > 0)
        {
                cout << "This is Parent Process; pid= " << getpid() << endl;
                pid_t Id = wait(NULL);
                cout << "Parent Process returns from wait after process "
                   << Id << " terminates\n";
                cout << "Calling exec() system call\n";
                char *args[] = {"a", "b", NULL};

                execv("./prog", args);
        }
        else
                cout << "Error\n";
        return 0;
}
```

FILE2 :

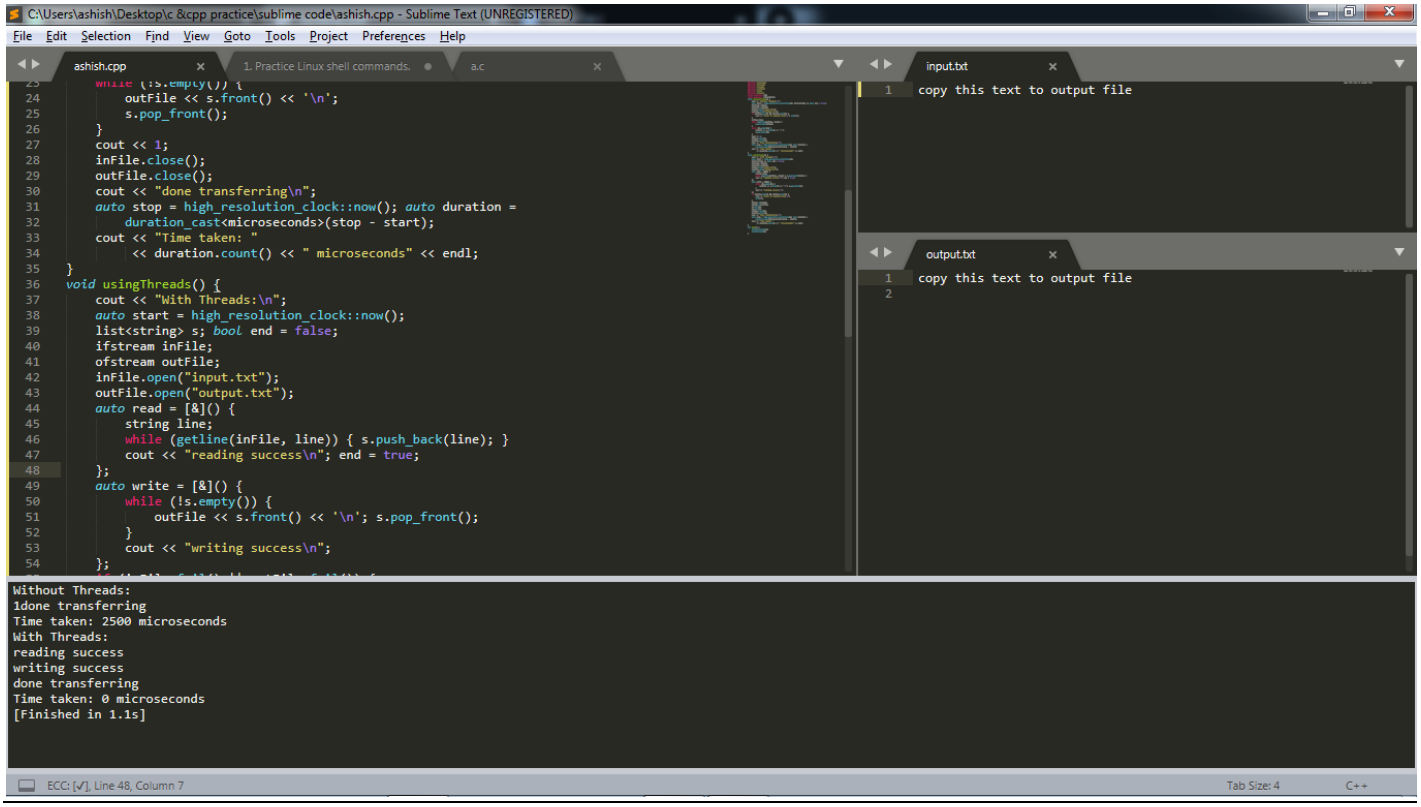
```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
using namespace std;
int main(int argc, char *argv[])
{
        cout << "This is the replaced file\n";
        cout << "Process Id= " << getpid() << endl;
        return 0;
}
```

## PROGRAM : 3 - Write a C program to represent a family of processes as a tree.

**Code:**

```
#include <iostream>
#include <fstream>
#include <thread>
#include <cstdlib>
#include <list>
#include <chrono>
using namespace std;
using namespace std::chrono;
void withoutThreads() {
	cout << "Without Threads:\n";
	auto start = high_resolution_clock::now(); list<string>
s; bool end = false;
	ifstream inFile;
	ofstream outFile;
	inFile.open("input.txt");
	outFile.open("output.txt");
	if (inFile.fail() || outFile.fail()) {
		cout << "error in opening files\n"; exit(1);
	}
	string line;
	while (getline(inFile, line)) {
		s.push_back(line);
	}
	while (!s.empty()) {
		outFile << s.front() << '\n';
		s.pop_front();
	}
	cout << 1;
	inFile.close();
	outFile.close();
	cout << "done transferring\n";
	auto stop = high_resolution_clock::now(); auto
duration =
	   duration_cast<microseconds>(stop - start);
	cout << "Time taken: "
	    << duration.count() << " microseconds" << endl;
}
void usingThreads() {
	cout << "With Threads:\n";
	auto start = high_resolution_clock::now();
	list<string> s; bool end = false;
	ifstream inFile;
	ofstream outFile;
	inFile.open("input.txt");
	outFile.open("output.txt");
	auto read = [&]() {
	string line;
	while (getline(inFile, line)) { s.push_back(line); }
		cout << "reading success\n"; end = true;
	};
	auto write = [&]() {
		while (!s.empty()) {
			outFile << s.front() << '\n';
s.pop_front();
		}
		cout << "writing success\n";
	};
	if (inFile.fail() || outFile.fail()) {
		cout << "error in opening files\n";
		exit(1);
	}
	thread t1(read);
	thread t2(write);
	t2.join();
	t1.join();
	inFile.close();
	outFile.close();
	cout << "done transferring\n";
	auto stop = high_resolution_clock::now(); auto
duration =
	   duration_cast<microseconds>(stop - start);
	cout << "Time taken: "
	    << duration.count() << " microseconds" << endl;
}
int main() {
	withoutThreads();
	usingThreads();
}
```

**OUTPUT:**


C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
1. Practice Linux shell commands.
a.c
input.txt
copy this text to output file
output.txt
copy this text to output file
Without Threads:
1done transferring
Time taken: 2500 microseconds
With Threads:
reading success
writing success
done transferring
Time taken: 0 microseconds
[Finished in 1.1s]
ECC: [✓], Line 48, Column 7
Tab Size: 4
C++

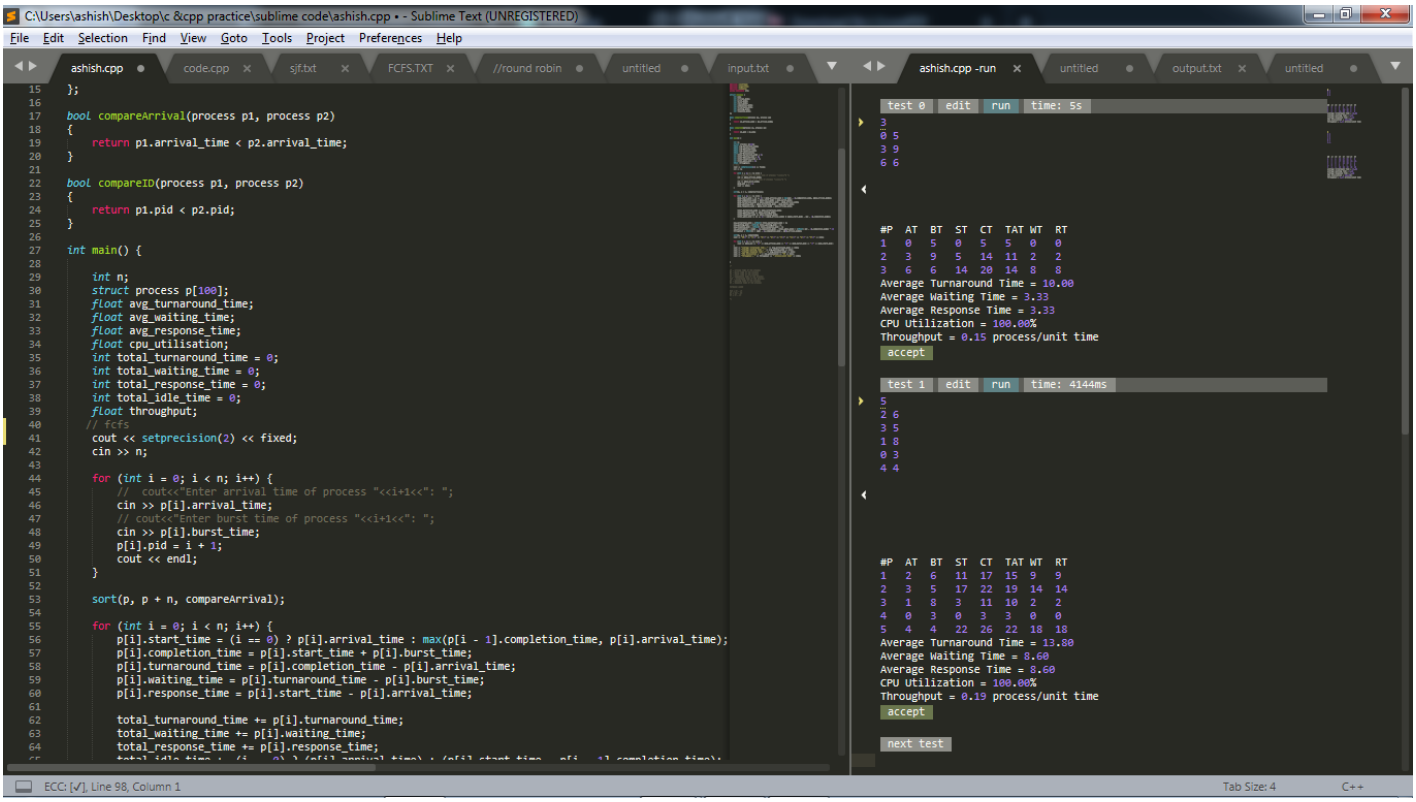
## PROGRAM 4 - CPU SCHEDULING ALGORITHMS – FCFS

**Code:**

```
#include <iostream>
#include <algorithm>
#include <iomanip>
using namespace std;

struct process {
	int pid;
	int arrival_time;
	int burst_time;
	int start_time;
	int completion_time;
	int turnaround_time;
	int waiting_time;
	int response_time;
};

bool compareArrival(process p1, process p2)
{
	return p1.arrival_time < p2.arrival_time;
}

bool compareID(process p1, process p2)
{
	return p1.pid < p2.pid;
}

int main() {

	int n;
	struct process p[100];
	float avg_turnaround_time;
	float avg_waiting_time;
	float avg_response_time;
	float cpu_utilisation;
	int total_turnaround_time = 0;
	int total_waiting_time = 0;
	int total_response_time = 0;
	int total_idle_time = 0;
	float throughput;
  // fcfs
   cout << setprecision(2) << fixed;
	cin >> n;

	for (int i = 0; i < n; i++) {
		cin >> p[i].arrival_time;
		cin >> p[i].burst_time;
		p[i].pid = i + 1;
		cout << endl;
	}

	sort(p, p + n, compareArrival);
for (int i = 0; i < n; i++) {
		p[i].start_time = (i == 0) ? p[i].arrival_time : max(p[i - 1].completion_time, p[i].arrival_time);
		p[i].completion_time = p[i].start_time + p[i].burst_time;
		p[i].turnaround_time = p[i].completion_time - p[i].arrival_time;
		p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
```

```
                p[i].response_time = p[i].start_time - p[i].arrival_time;

                total_turnaround_time += p[i].turnaround_time;
                total_waiting_time += p[i].waiting_time;
                total_response_time += p[i].response_time;
                total_idle_time += (i == 0) ? (p[i].arrival_time) : (p[i].start_time - p[i - 1].completion_time);
        }

        avg_turnaround_time = (float) total_turnaround_time / n;
        avg_waiting_time = (float) total_waiting_time / n;
        avg_response_time = (float) total_response_time / n;
        cpu_utilisation = ((p[n - 1].completion_time - total_idle_time) / (float) p[n - 1].completion_time) * 100;
        throughput = float(n) / (p[n - 1].completion_time - p[0].arrival_time);

        sort(p, p + n, compareID);
        cout << "#P\t" << "AT\t" << "BT\t" << "ST\t" << "CT\t" << "TAT\t" << "WT\t" << "RT\t" << endl;

        for (int i = 0; i < n; i++) {
                cout << p[i].pid << "\t" << p[i].arrival_time << "\t" << p[i].burst_time << "\t" << p[i].start_time << "\t" <<
p[i].completion_time << "\t" << p[i].turnaround_time << "\t" << p[i].waiting_time << "\t" << p[i].response_time << "\t" <<
endl;
        }
        cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
        cout << "Average Waiting Time = " << avg_waiting_time << endl;
        cout << "Average Response Time = " << avg_response_time << endl;
        cout << "CPU Utilization = " << cpu_utilisation << "%" << endl;
        cout << "Throughput = " << throughput << " process/unit time" << endl;

return 0;
}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:

TAT = CT - AT
WT = TAT - BT
RT = ST - AT

*/
```

**Output:**


C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
code.cpp
sjf.txt
FCFS.TXT
//round robin
untitled
input.txt
ashish.cpp -run
untitled
output.txt
untitled
};
bool compareArrival(process p1, process p2)
{
return p1.arrival_time < p2.arrival_time;
}
bool compareID(process p1, process p2)
{
return p1.pid < p2.pid;
}
int main() {
int n;
struct process p[100];
float avg_turnaround_time;
float avg_waiting_time;
float avg_response_time;
float cpu_utilisation;
int total_turnaround_time = 0;
int total_waiting_time = 0;
int total_response_time = 0;
int total_idle_time = 0;
float throughput;
// fcfs
cout << setprecision(2) << fixed;
cin >> n;
for (int i = 0; i < n; i++) {
// cout<<"Enter arrival time of process "<<i+1<<": ";
cin >> p[i].arrival_time;
// cout<<"Enter burst time of process "<<i+1<<": ";
cin >> p[i].burst_time;
p[i].pid = i + 1;
cout << endl;
}
sort(p, p + n, compareArrival);
for (int i = 0; i < n; i++) {
p[i].start_time = (i == 0) ? p[i].arrival_time : max(p[i - 1].completion_time, p[i].arrival_time);
p[i].completion_time = p[i].start_time + p[i].burst_time;
p[i].turnaround_time = p[i].completion_time - p[i].arrival_time;
p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
p[i].response_time = p[i].start_time - p[i].arrival_time;
total_turnaround_time += p[i].turnaround_time;
total_waiting_time += p[i].waiting_time;
total_response_time += p[i].response_time;
test 0 edit run time: 5s
3
0 5
3 9
6 6
#P AT BT ST CT TAT WT RT
1 0 5 0 5 5 0 0
2 3 9 5 14 11 2 2
3 6 6 14 20 14 8 8
Average Turnaround Time = 10.00
Average Waiting Time = 3.33
Average Response Time = 3.33
CPU Utilization = 100.00%
Throughput = 0.15 process/unit time
accept
test 1 edit run time: 4144ms
5
2 6
3 5
1 8
0 3
4 4
#P AT BT ST CT TAT WT RT
1 2 6 11 17 15 9 9
2 3 5 17 22 19 14 14
3 1 8 3 11 10 2 2
4 0 3 0 3 3 0 0
5 4 4 22 26 22 18 18
Average Turnaround Time = 13.80
Average Waiting Time = 8.60
Average Response Time = 8.60
CPU Utilization = 100.00%
Throughput = 0.19 process/unit time
accept
next test
ECC: [✓], Line 98, Column 1
Tab Size: 4
C++

## Program 5 : CPU SCHEDULING ALGORITHMS – SJF

**Code :**

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string.h>
using namespace std;

struct process {
        int pid;
        int arrival_time;
        int burst_time;
        int start_time;
        int completion_time;
        int turnaround_time;
        int waiting_time;
        int response_time;
};

int main() {

        int n;
        struct process p[100];
        float avg_turnaround_time;
        float avg_waiting_time;
        float avg_response_time;
        float cpu_utilisation;
        int total_turnaround_time = 0;
        int total_waiting_time = 0;
        int total_response_time = 0;
        int total_idle_time = 0;
        float throughput;
        int is_completed[100];
        memset(is_completed, 0, sizeof(is_completed));

        cout << setprecision(2) << fixed;

        cin >> n;

        for (int i = 0; i < n; i++) {
                cin >> p[i].arrival_time;
                cin >> p[i].burst_time;
                p[i].pid = i + 1;
                cout << endl;
        }

        int current_time = 0;
        int completed = 0;
        int prev = 0;

        while (completed != n) {
                int idx = -1;
                int mn = 10000000;
                for (int i = 0; i < n; i++) {
                        if (p[i].arrival_time <= current_time && is_completed[i] == 0) {
                                if (p[i].burst_time < mn) {
                                        mn = p[i].burst_time;
                                        idx = i;
                                }
                                if (p[i].burst_time == mn) {
                                        if (p[i].arrival_time < p[idx].arrival_time) {
```

```
                                                mn = p[i].burst_time;
                                                idx = i;
                                        }
                                }
                        }
                }
if (idx != -1) {
                        p[idx].start_time = current_time;
                        p[idx].completion_time = p[idx].start_time + p[idx].burst_time;
                        p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
                        p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
                        p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

                        total_turnaround_time += p[idx].turnaround_time;
                        total_waiting_time += p[idx].waiting_time;
                        total_response_time += p[idx].response_time;
                        total_idle_time += p[idx].start_time - prev;

                        is_completed[idx] = 1;
                        completed++;
                        current_time = p[idx].completion_time;
                        prev = current_time;
                }
                else {
                        current_time++;
                }

        }

        int min_arrival_time = 10000000;
        int max_completion_time = -1;
        for (int i = 0; i < n; i++) {
                min_arrival_time = min(min_arrival_time, p[i].arrival_time);
                max_completion_time = max(max_completion_time, p[i].completion_time);
        }

        avg_turnaround_time = (float) total_turnaround_time / n;
        avg_waiting_time = (float) total_waiting_time / n;
        avg_response_time = (float) total_response_time / n;
        cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_completion_time ) * 100;
        throughput = float(n) / (max_completion_time - min_arrival_time);

        cout << endl << endl;

        cout << "#P\t" << "AT\t" << "BT\t" << "ST\t" << "CT\t" << "TAT\t" << "WT\t" << "RT\t" << endl;

        for (int i = 0; i < n; i++) {
                cout << p[i].pid << "\t" << p[i].arrival_time << "\t" << p[i].burst_time << "\t" << p[i].start_time << "\t" <<
p[i].completion_time << "\t" << p[i].turnaround_time << "\t" << p[i].waiting_time << "\t" << p[i].response_time << "\t" <<
endl;
        }
        cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
        cout << "Average Waiting Time = " << avg_waiting_time << endl;
        cout << "Average Response Time = " << avg_response_time << endl;
        cout << "CPU Utilization = " << cpu_utilisation << "%" << endl;
        cout << "Throughput = " << throughput << " process/unit time" << endl;


}

/*
```

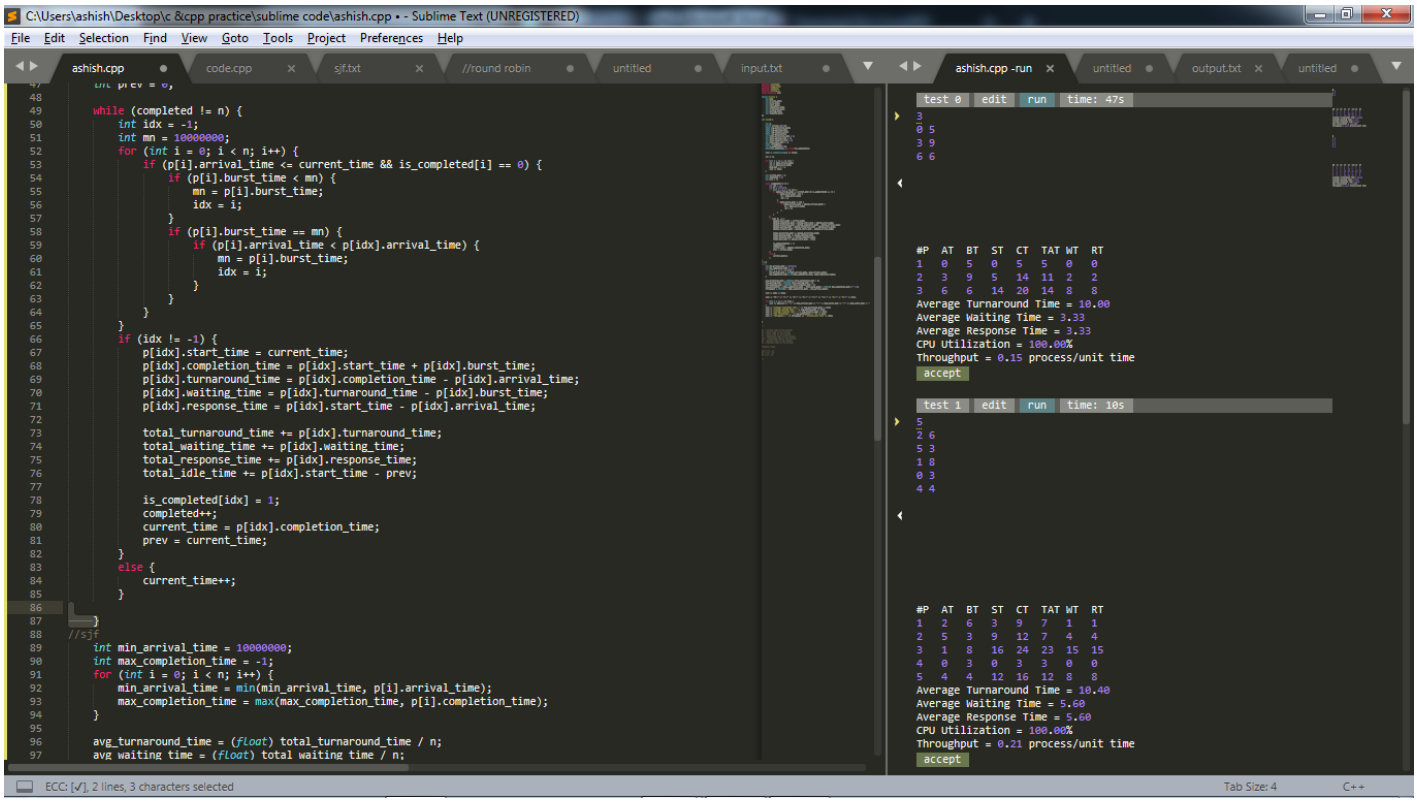
**AT - Arrival Time of the process**
**BT - Burst time of the process**
**ST - Start time of the process**
**CT - Completion time of the process**
**TAT - Turnaround time of the process**
**WT - Waiting time of the process**
**RT - Response time of the process**

**Formulas used:**

**TAT = CT - AT**
**WT = TAT - BT**
**RT = ST - AT**

***/**

**Output:**


C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
code.cpp
sjf.txt
//round robin
untitled
input.txt
ashish.cpp -run
untitled
output.txt
untitled
test 0 edit run time: 47s
3
0 5
3 9
6 6
#P AT BT ST CT TAT WT RT
1 0 5 0 5 5 0 0
2 3 9 5 14 11 2 2
3 6 6 14 20 14 8 8
Average Turnaround Time = 10.00
Average Waiting Time = 3.33
Average Response Time = 3.33
CPU Utilization = 100.00%
Throughput = 0.15 process/unit time
accept
test 1 edit run time: 10s
5
2 6
5 3
1 8
0 3
4 4
#P AT BT ST CT TAT WT RT
1 2 6 3 9 7 1 1
2 5 3 9 12 7 4 4
3 1 8 16 24 23 15 15
4 0 3 0 3 3 0 0
5 4 4 12 16 12 8 8
Average Turnaround Time = 10.40
Average Waiting Time = 5.60
Average Response Time = 5.60
CPU Utilization = 100.00%
Throughput = 0.21 process/unit time
accept
ECC: [✓], 2 lines, 3 characters selected
Tab Size: 4
C++

## PROGRAM 6 - CPU SCHEDULING ALGORITHMS – ROUND ROBIN

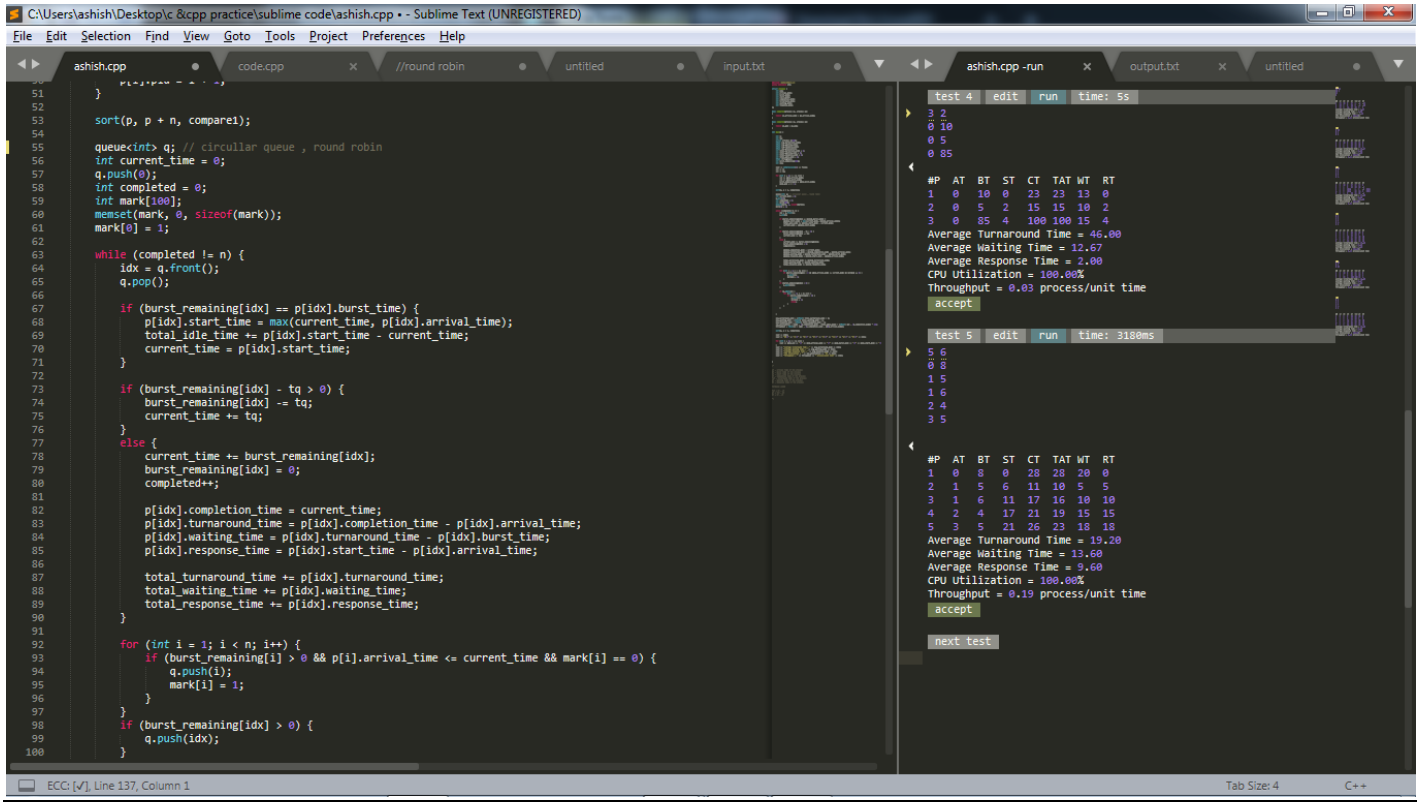
**Code:**

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

struct process {
	int pid;
	int arrival_time;
	int burst_time;
	int start_time;
	int completion_time;
	int turnaround_time;
	int waiting_time;
	int response_time;
};

bool compare1(process p1, process p2)
{
	return p1.arrival_time < p2.arrival_time;
}

bool compare2(process p1, process p2)
{
	return p1.pid < p2.pid;
}

int main() {

	int n;
	int tq;
	struct process p[100];
	float avg_turnaround_time;
	float avg_waiting_time;
	float avg_response_time;
	float cpu_utilisation;
	int total_turnaround_time = 0;
	int total_waiting_time = 0;
	int total_response_time = 0;
	int total_idle_time = 0;
	float throughput;
	int burst_remaining[100];
	int idx;

	cout << setprecision(2) << fixed;
	cin >> n;
	cin >> tq;

	for (int i = 0; i < n; i++) {
		cin >> p[i].arrival_time;
		cin >> p[i].burst_time;
		burst_remaining[i] = p[i].burst_time;
		p[i].pid = i + 1;
	}

	sort(p, p + n, compare1);

	queue<int> q; // circullar queue , round robin
	int current_time = 0;
	q.push(0);
	int completed = 0;
	int mark[100];
```

```
	memset(mark, 0, sizeof(mark));
	mark[0] = 1;

	while (completed != n) {
		idx = q.front();
		q.pop();

		if (burst_remaining[idx] == p[idx].burst_time) {
			p[idx].start_time = max(current_time, p[idx].arrival_time);
			total_idle_time += p[idx].start_time - current_time;
			current_time = p[idx].start_time;
		}

		if (burst_remaining[idx] - tq > 0) {
			burst_remaining[idx] -= tq;
			current_time += tq;
		}
else {
			current_time += burst_remaining[idx];
			burst_remaining[idx] = 0;
			completed++;

			p[idx].completion_time = current_time;
			p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
			p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
			p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

			total_turnaround_time += p[idx].turnaround_time;
			total_waiting_time += p[idx].waiting_time;
			total_response_time += p[idx].response_time;
		}

		for (int i = 1; i < n; i++) {
			if (burst_remaining[i] > 0 && p[i].arrival_time <= current_time && mark[i] == 0) {
				q.push(i);
				mark[i] = 1;
			}
		}
		if (burst_remaining[idx] > 0) {
			q.push(idx);
		}

		if (q.empty()) {
			for (int i = 1; i < n; i++) {
				if (burst_remaining[i] > 0) {
					q.push(i);
					mark[i] = 1;
					break;
				}
			}
		}


	}

	avg_turnaround_time = (float) total_turnaround_time / n;
	avg_waiting_time = (float) total_waiting_time / n;
	avg_response_time = (float) total_response_time / n;
	cpu_utilisation = ((p[n - 1].completion_time - total_idle_time) / (float) p[n - 1].completion_time) * 100;
	throughput = float(n) / (p[n - 1].completion_time - p[0].arrival_time);

	sort(p, p + n, compare2);
```

```
	cout << endl;
	cout << "#P\t" << "AT\t" << "BT\t" << "ST\t" << "CT\t" << "TAT\t" << "WT\t" << "RT\t" << endl;

	for (int i = 0; i < n; i++) {
		cout << p[i].pid << "\t" << p[i].arrival_time << "\t" << p[i].burst_time << "\t" << p[i].start_time << "\t" <<
p[i].completion_time << "\t" << p[i].turnaround_time << "\t" << p[i].waiting_time << "\t" << p[i].response_time << "\t" << endl;
	}
	cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
	cout << "Average Waiting Time = " << avg_waiting_time << endl;
	cout << "Average Response Time = " << avg_response_time << endl;
	cout << "CPU Utilization = " << cpu_utilisation << "%" << endl;
	cout << "Throughput = " << throughput << " process/unit time" << endl;


}
/*
AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:
TAT = CT - AT
WT = TAT - BT
RT = ST – AT*/
```

**Output:**


C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
code.cpp
//round robin
untitled
input.txt
ashish.cpp -run
output.txt
untitled
test 4 edit run time: 5s
3 2
0 10
0 5
0 85
#P AT BT ST CT TAT WT RT
1 0 10 0 23 23 13 0
2 0 5 2 15 15 10 2
3 0 85 4 100 100 15 4
Average Turnaround Time = 46.00
Average Waiting Time = 12.67
Average Response Time = 2.00
CPU Utilization = 100.00%
Throughput = 0.03 process/unit time
accept
test 5 edit run time: 3180ms
5 6
0 8
1 5
1 6
2 4
3 5
#P AT BT ST CT TAT WT RT
1 0 8 0 28 28 20 0
2 1 5 6 11 10 5 5
3 1 6 11 17 16 10 10
4 2 4 17 21 19 15 15
5 3 5 21 26 23 18 18
Average Turnaround Time = 19.20
Average Waiting Time = 13.60
Average Response Time = 9.60
CPU Utilization = 100.00%
Throughput = 0.19 process/unit time
accept
next test
ECC: [√], Line 137, Column 1
Tab Size: 4
C++

## PROGRAM 7 - CPU SCHEDULING ALGORITHMS – PREEMPTITIVE PRIORITY SCHEDULING

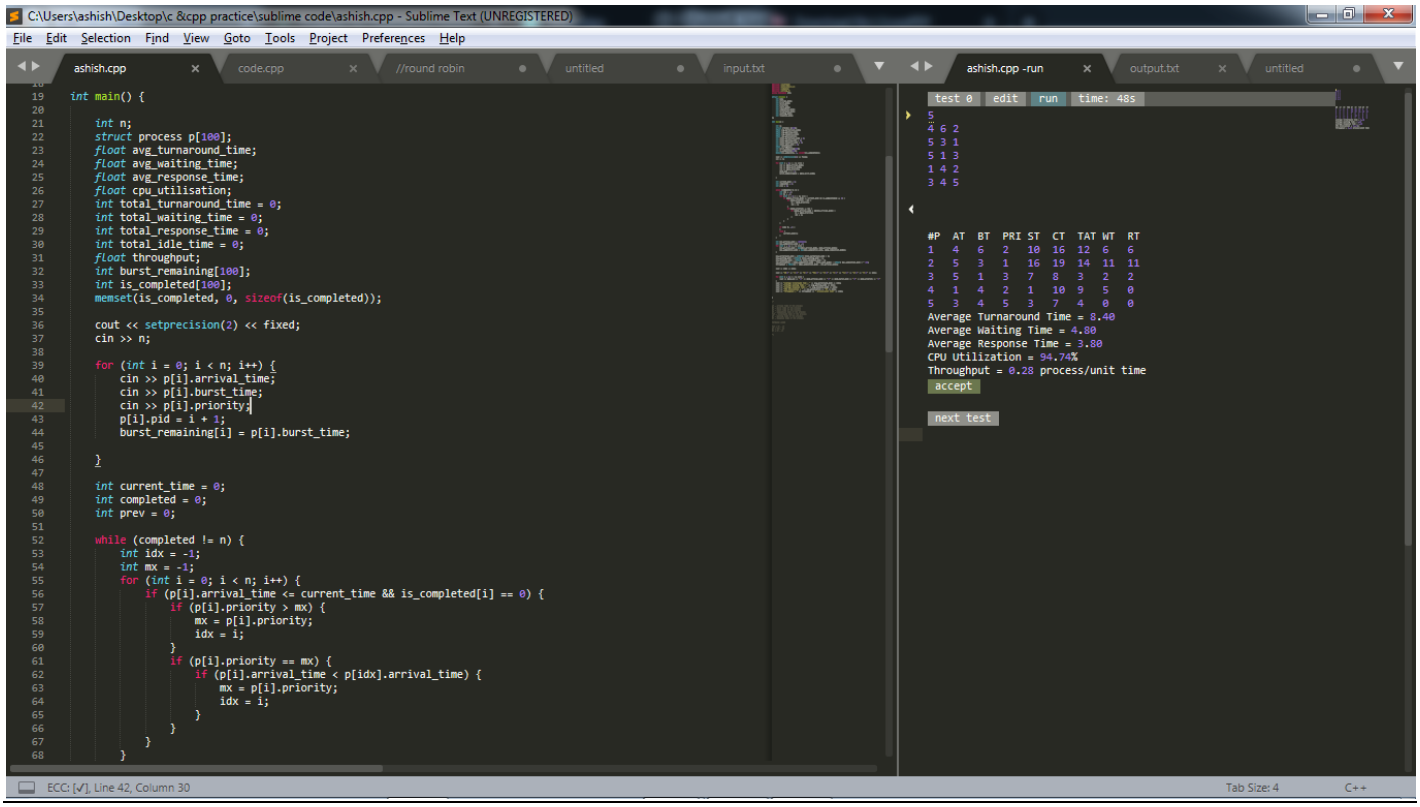
**Code:**

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

struct process {
	int pid;
	int arrival_time;
	int burst_time;
	int priority;
	int start_time;
	int completion_time;
	int turnaround_time;
	int waiting_time;
	int response_time;
};

int main() {

	int n;
	struct process p[100];
	float avg_turnaround_time;
	float avg_waiting_time;
	float avg_response_time;
	float cpu_utilisation;
	int total_turnaround_time = 0;
	int total_waiting_time = 0;
	int total_response_time = 0;
	int total_idle_time = 0;
	float throughput;
	int burst_remaining[100];
	int is_completed[100];
	memset(is_completed, 0, sizeof(is_completed));

	cout << setprecision(2) << fixed;
	cin >> n;

	for (int i = 0; i < n; i++) {
		cin >> p[i].arrival_time;
		cin >> p[i].burst_time;
		cin >> p[i].priority;
		p[i].pid = i + 1;
		burst_remaining[i] = p[i].burst_time;

	}

	int current_time = 0;
	int completed = 0;
	int prev = 0;

	while (completed != n) {
		int idx = -1;
		int mx = -1;
		for (int i = 0; i < n; i++) {
			if (p[i].arrival_time <= current_time && is_completed[i] == 0) {
				if (p[i].priority > mx) {
```

```
                                        mx = p[i].priority;
                                        idx = i;
                                }
                                if (p[i].priority == mx) {
                                        if (p[i].arrival_time < p[idx].arrival_time) {
                                                mx = p[i].priority;
                                                idx = i;
                                        }
                                }
                        }
                }
        if (idx != -1) {
                        if (burst_remaining[idx] == p[idx].burst_time) {
                                p[idx].start_time = current_time;
                                total_idle_time += p[idx].start_time - prev;
                        }
                        burst_remaining[idx] -= 1;
                        current_time++;
                        prev = current_time;

                        if (burst_remaining[idx] == 0) {
                                p[idx].completion_time = current_time;
                                p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
                                p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
                                p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

                                total_turnaround_time += p[idx].turnaround_time;
                                total_waiting_time += p[idx].waiting_time;
                                total_response_time += p[idx].response_time;

                                is_completed[idx] = 1;
                                completed++;
                        }
                }
                else {
                        current_time++;
                }
        }
        int min_arrival_time = 10000000;
        int max_completion_time = -1;
        for (int i = 0; i < n; i++) {
                min_arrival_time = min(min_arrival_time, p[i].arrival_time);
                max_completion_time = max(max_completion_time, p[i].completion_time);
        }
        avg_turnaround_time = (float) total_turnaround_time / n;
        avg_waiting_time = (float) total_waiting_time / n;
        avg_response_time = (float) total_response_time / n;
        cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_completion_time ) * 100;
        throughput = float(n) / (max_completion_time - min_arrival_time);

        cout << endl << endl;

        cout << "#P\t" << "AT\t" << "BT\t" << "PRI\t" << "ST\t" << "CT\t" << "TAT\t" << "WT\t" << "RT\t" << endl;

        for (int i = 0; i < n; i++) {
                cout << p[i].pid << "\t" << p[i].arrival_time << "\t" << p[i].burst_time << "\t" << p[i].priority << "\t" <<
p[i].start_time << "\t" << p[i].completion_time << "\t" << p[i].turnaround_time << "\t" << p[i].waiting_time << "\t" <<
p[i].response_time << "\t" << endl;
        }
        cout << "Average Turnaround Time = " << avg_turnaround_time << endl;
        cout << "Average Waiting Time = " << avg_waiting_time << endl;
        cout << "Average Response Time = " << avg_response_time << endl;
```

```
        cout << "CPU Utilization = " << cpu_utilisation << "%" << endl;
        cout << "Throughput = " << throughput << " process/unit time" << endl;

}
/*
AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:
TAT = CT - AT
WT = TAT - BT
RT = ST - AT
*/
```

**Output:**


C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
code.cpp
//round robin
untitled
input.txt
ashish.cpp -run
output.txt
untitled
int main() {
int n;
struct process p[100];
float avg_turnaround_time;
float avg_waiting_time;
float avg_response_time;
float cpu_utilisation;
int total_turnaround_time = 0;
int total_waiting_time = 0;
int total_response_time = 0;
int total_idle_time = 0;
float throughput;
int burst_remaining[100];
int is_completed[100];
memset(is_completed, 0, sizeof(is_completed));
cout << setprecision(2) << fixed;
cin >> n;
for (int i = 0; i < n; i++) {
cin >> p[i].arrival_time;
cin >> p[i].burst_time;
cin >> p[i].priority;
p[i].pid = i + 1;
burst_remaining[i] = p[i].burst_time;
}
int current_time = 0;
int completed = 0;
int prev = 0;
while (completed != n) {
int idx = -1;
int mx = -1;
for (int i = 0; i < n; i++) {
if (p[i].arrival_time <= current_time && is_completed[i] == 0) {
if (p[i].priority > mx) {
mx = p[i].priority;
idx = i;
}
if (p[i].priority == mx) {
if (p[i].arrival_time < p[idx].arrival_time) {
mx = p[i].priority;
idx = i;
}
}
}
}
test 0 edit run time: 48s
5
4 6 2
5 3 1
5 1 3
1 4 2
3 4 5
#P AT BT PRI ST CT TAT WT RT
1 4 6 2 10 16 12 6 6
2 5 3 1 16 19 14 11 11
3 5 1 3 7 8 3 2 2
4 1 4 2 1 10 9 5 0
5 3 4 5 3 7 4 0 0
Average Turnaround Time = 8.40
Average Waiting Time = 4.80
Average Response Time = 3.80
CPU Utilization = 94.74%
Throughput = 0.28 process/unit time
accept
next test
ECC: [✓], Line 42, Column 30
Tab Size: 4
C++

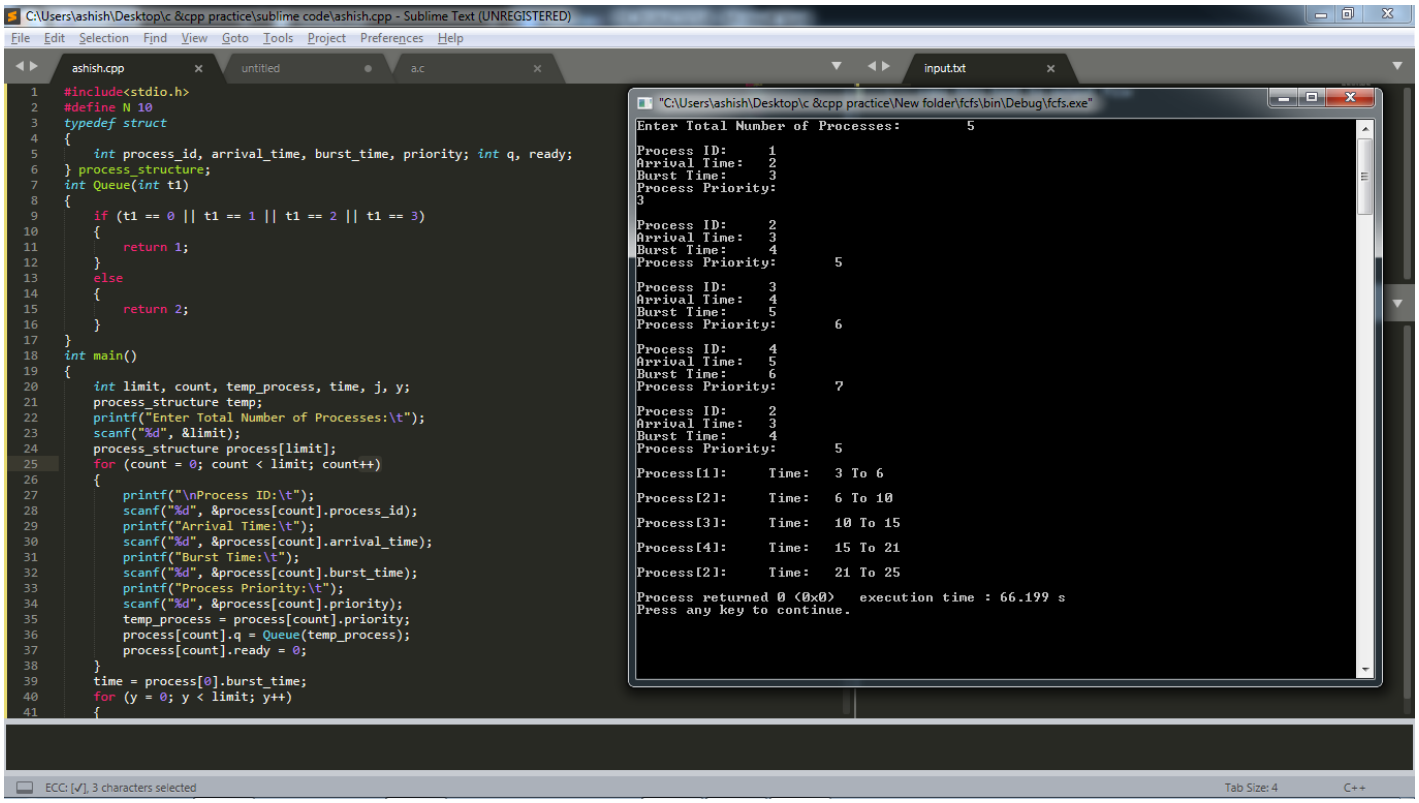
## PROGRAM: 8 simulate Multilevel Feedback Queue scheduling algorithm.

**Code :**

```
#include<stdio.h>
#define N 10
typedef struct
{
	int process_id, arrival_time, burst_time, priority; int q, ready;
} process_structure;
int Queue(int t1)
{
	if (t1 == 0 || t1 == 1 || t1 == 2 || t1 == 3)
	{
		return 1;
	}
	else
	{
		return 2;
	}
}
int main()
{
	int limit, count, temp_process, time, j, y;
	process_structure temp;
	printf("Enter Total Number of Processes:\t");
	scanf("%d", &limit);
	process_structure process[limit];
	for (count = 0; count < limit; count++)
	{
		printf("\nProcess ID:\t");
		scanf("%d", &process[count].process_id);
		printf("Arrival Time:\t");
		scanf("%d", &process[count].arrival_time);
		printf("Burst Time:\t");
		scanf("%d", &process[count].burst_time);
		printf("Process Priority:\t");
		scanf("%d", &process[count].priority);
		temp_process = process[count].priority;
		process[count].q = Queue(temp_process);
		process[count].ready = 0;
	}
	time = process[0].burst_time;
	for (y = 0; y < limit; y++)
	{
		for (count = y; count < limit; count++)
		{
			if (process[count].arrival_time < time)
			{
				process[count].ready = 1;
			}
		}
		for (count = y; count < limit - 1; count++)
		{
			for (j = count + 1; j < limit; j++)
			{
				if (process[count].ready == 1 && process[j].ready == 1)
				{
					if (process[count].q == 2 && process[j].q == 1)
					{
						temp = process[count];
						process[count] = process[j];
						process[j] = temp;
```

```
                                        }
                                }
                        }
                }
                for (count = y; count < limit - 1; count++)
                {
                        for (j = count + 1; j < limit; j++)
                        {
                                if (process[count].ready == 1 && process[j].ready == 1)
                                {
                                        if (process[count].q == 1 && process[j].q == 1)
                                        {
                                                if (process[count].burst_time >
                                                    process[j].burst_time)
                                                {
                                                        temp = process[count];
                                                        process[count] = process[j];
                                                        process[j] = temp;
                                                }
                                                else
                                                {
                                                        break;
                                                }
                                        }
                                }
                        }
                }
                printf("\nProcess[%d]:\tTime:\t%d To %d\n", process[y].process_id, time, time +
                    process[y].burst_time);
                time = time + process[y].burst_time;
                for (count = y; count < limit; count++)
                {
                        if (process[count].ready == 1)
                        {
                                process[count].ready = 0;
                        }
                }
        }
        return 0;
}
```

## OUTPUT :

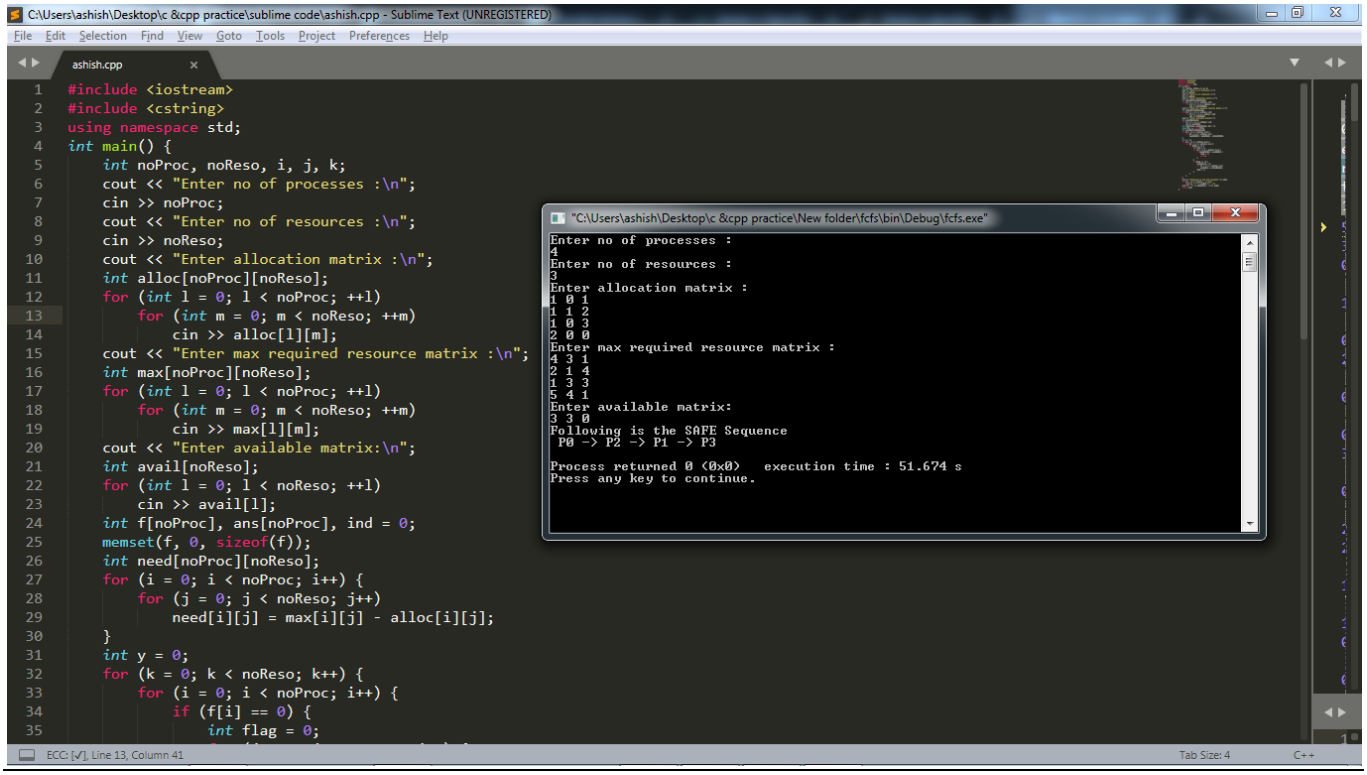

C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
untitled
a.c
input.txt
#include<stdio.h>
#define N 10
typedef struct
{
int process_id, arrival_time, burst_time, priority; int q, ready;
} process_structure;
int Queue(int t1)
{
if (t1 == 0 || t1 == 1 || t1 == 2 || t1 == 3)
{
return 1;
}
else
{
return 2;
}
}
int main()
{
int limit, count, temp_process, time, j, y;
process_structure temp;
printf("Enter Total Number of Processes:\t");
scanf("%d", &limit);
process_structure process[limit];
for (count = 0; count < limit; count++)
{
printf("\nProcess ID:\t");
scanf("%d", &process[count].process_id);
printf("Arrival Time:\t");
scanf("%d", &process[count].arrival_time);
printf("Burst Time:\t");
scanf("%d", &process[count].burst_time);
printf("Process Priority:\t");
scanf("%d", &process[count].priority);
temp_process = process[count].priority;
process[count].q = Queue(temp_process);
process[count].ready = 0;
}
time = process[0].burst_time;
for (y = 0; y < limit; y++)
{
"C:\Users\ashish\Desktop\c &cpp practice\New folder\fcfs\bin\Debug\fcfs.exe"
Enter Total Number of Processes: 5
Process ID: 1
Arrival Time: 2
Burst Time: 3
Process Priority:
3
Process ID: 2
Arrival Time: 3
Burst Time: 4
Process Priority: 5
Process ID: 3
Arrival Time: 4
Burst Time: 5
Process Priority: 6
Process ID: 4
Arrival Time: 5
Burst Time: 6
Process Priority: 7
Process ID: 2
Arrival Time: 3
Burst Time: 4
Process Priority: 5
Process[1]: Time: 3 To 6
Process[2]: Time: 6 To 10
Process[3]: Time: 10 To 15
Process[4]: Time: 15 To 21
Process[2]: Time: 21 To 25
Process returned 0 (0x0) execution time : 66.199 s
Press any key to continue.
ECC: [✓], 3 characters selected
Tab Size: 4
C++

## PROGRAM 9 - Program to stimulate deadlock detection

**Code:**

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
	int noProc, noReso, i, j, k;
	cout << "Enter no of processes :\n";
	cin >> noProc;
	cout << "Enter no of resources :\n";
	cin >> noReso;
	cout << "Enter allocation matrix :\n";
	int alloc[noProc][noReso];
	for (int l = 0; l < noProc; ++l)
		for (int m = 0; m < noReso; ++m)
			cin >> alloc[l][m];
	cout << "Enter max required resource matrix :\n";
	int max[noProc][noReso];
	for (int l = 0; l < noProc; ++l)
		for (int m = 0; m < noReso; ++m)
			cin >> max[l][m];
	cout << "Enter available matrix:\n";
	int avail[noReso];
	for (int l = 0; l < noReso; ++l)
		cin >> avail[l];
	int f[noProc], ans[noProc], ind = 0;
	memset(f, 0, sizeof(f));
	int need[noProc][noReso];
	for (i = 0; i < noProc; i++) {
		for (j = 0; j < noReso; j++)
			need[i][j] = max[i][j] - alloc[i][j];
	}
	int y = 0;
	for (k = 0; k < noReso; k++) {
		for (i = 0; i < noProc; i++) {
			if (f[i] == 0) {
				int flag = 0;
				for (j = 0; j < noReso; j++) {
					if (need[i][j] > avail[j]) {
						flag = 1;
						break;
					}
				}
				if (flag == 0) {
					ans[ind++] = i;
					for (y = 0; y < noReso; y++)
						avail[y] += alloc[i][y];
					f[i] = 1;
				}
			}
		}
	}
	cout << "Following is the SAFE Sequence" << endl;
	for (i = 0; i < noProc - 1; i++)
		cout << " P" << ans[i] << " ->";
	cout << " P" << ans[noProc - 1] << endl;
	return (0);
}
```

**Output:**

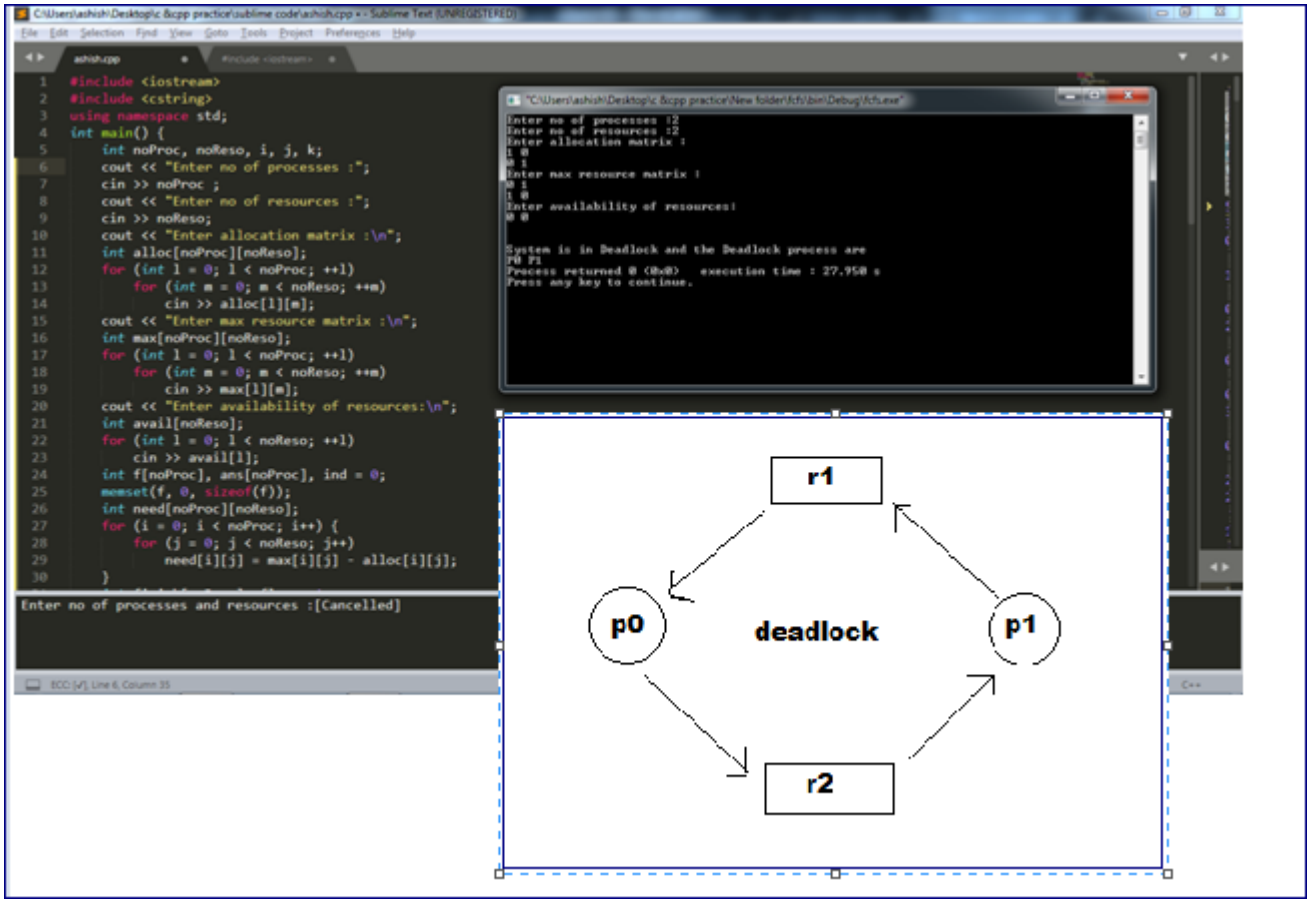

C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
#include <iostream>
#include <cstring>
using namespace std;
int main() {
int noProc, noReso, i, j, k;
cout << "Enter no of processes :\n";
cin >> noProc;
cout << "Enter no of resources :\n";
cin >> noReso;
cout << "Enter allocation matrix :\n";
int alloc[noProc][noReso];
for (int l = 0; l < noProc; ++l)
for (int m = 0; m < noReso; ++m)
cin >> alloc[l][m];
cout << "Enter max required resource matrix :\n";
int max[noProc][noReso];
for (int l = 0; l < noProc; ++l)
for (int m = 0; m < noReso; ++m)
cin >> max[l][m];
cout << "Enter available matrix:\n";
int avail[noReso];
for (int l = 0; l < noReso; ++l)
cin >> avail[l];
int f[noProc], ans[noProc], ind = 0;
memset(f, 0, sizeof(f));
int need[noProc][noReso];
for (i = 0; i < noProc; i++) {
for (j = 0; j < noReso; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < noReso; k++) {
for (i = 0; i < noProc; i++) {
if (f[i] == 0) {
int flag = 0;
"C:\Users\ashish\Desktop\c &cpp practice\New folder\fcfs\bin\Debug\fcfs.exe"
Enter no of processes :
4
Enter no of resources :
3
Enter allocation matrix :
1 0 1
1 1 2
1 0 3
2 0 0
Enter max required resource matrix :
4 3 1
2 1 4
1 3 3
5 4 1
Enter available matrix:
3 3 0
Following is the SAFE Sequence
P0 -> P2 -> P1 -> P3
Process returned 0 (0x0) execution time : 51.674 s
Press any key to continue.
ECC: [√], Line 13, Column 41
Tab Size: 4
C++

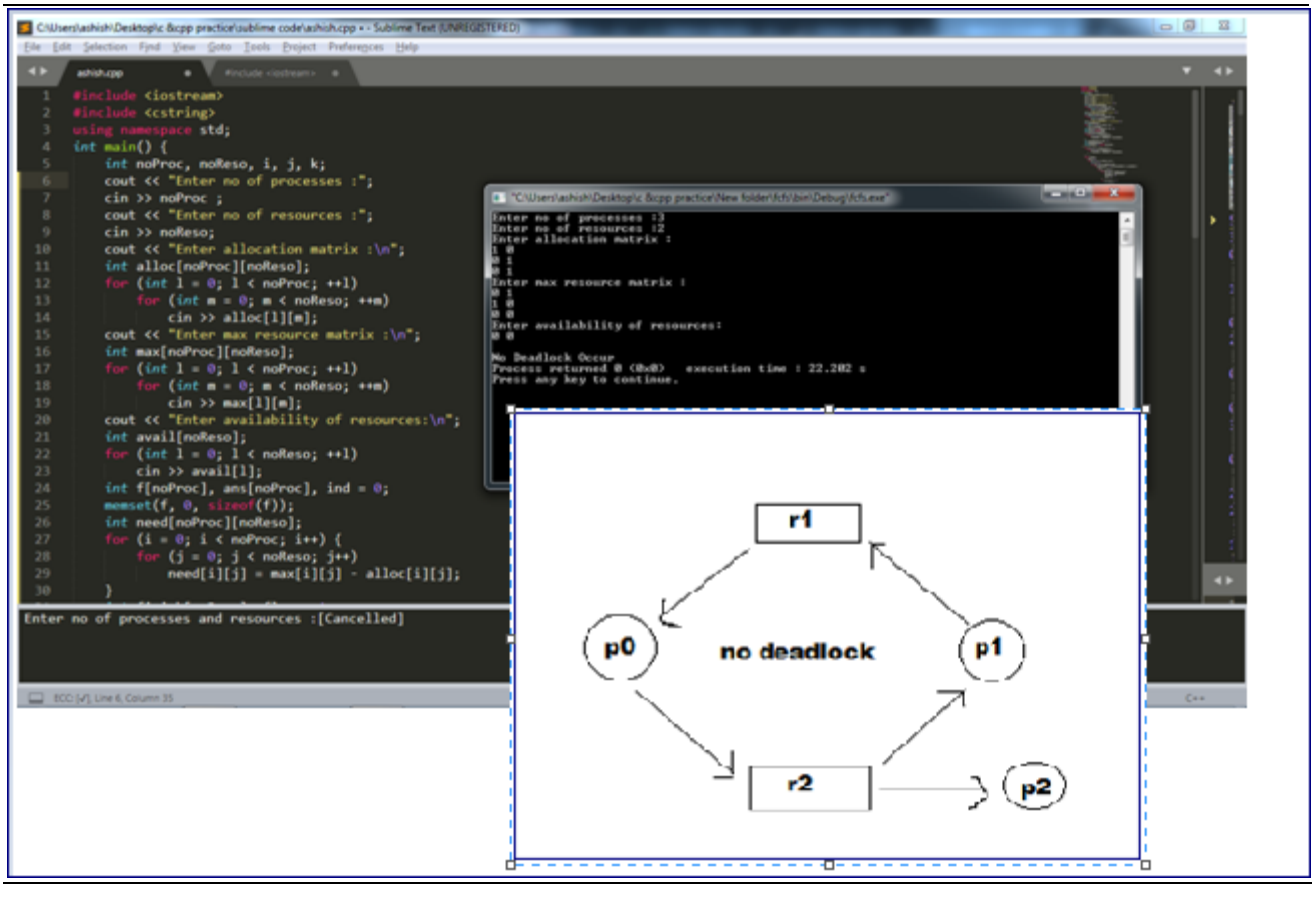
## PROGRAM 10 - Program to stimulate deadlock avoidance

**Code:**

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
	int noProc, noReso, i, j, k;
	cout << "Enter no of processes :";
	cin >> noProc ;
	cout << "Enter no of resources :";
	cin >> noReso;
	cout << "Enter allocation matrix :\n";
	int alloc[noProc][noReso];
	for (int l = 0; l < noProc; ++l)
		for (int m = 0; m < noReso; ++m)
			cin >> alloc[l][m];
	cout << "Enter max resource matrix :\n";
	int max[noProc][noReso];
	for (int l = 0; l < noProc; ++l)
		for (int m = 0; m < noReso; ++m)
			cin >> max[l][m];
	cout << "Enter availability of resources:\n";
	int avail[noReso];
	for (int l = 0; l < noReso; ++l)
		cin >> avail[l];
	int f[noProc], ans[noProc], ind = 0;
	memset(f, 0, sizeof(f));
	int need[noProc][noReso];
	for (i = 0; i < noProc; i++) {
		for (j = 0; j < noReso; j++)
			need[i][j] = max[i][j] - alloc[i][j];
	}
	int finish[noProc], flag = 1;
	int deadlock[noProc];
	memset(finish, 0, sizeof(finish));
//find need matrix
	for (i = 0; i < noProc; i++) {
		for (j = 0; j < noReso; j++) {
			need[i][j] = max[i][j] - alloc[i][j];
		}
	}
	while (flag) {
		flag = 0;
		for (i = 0; i < noProc; i++) {
			int c = 0;
			for (j = 0; j < noReso; j++)
				if ((finish[i] == 0) && (need[i][j] <= avail[j])) {
					c++;
					if (c == noReso) {
						for (k = 0; k < noReso; k++) {
							avail[k] +=
alloc[i][j];

							finish[i] = 1;
							flag = 1;
						}
						if (finish[i] == 1)
							i = noProc;
					}
				}
		}
	}
j = 0;
	flag = 0;
	for (i = 0; i < noProc; i++)
		if (finish[i] == 0) {
			deadlock[j++] = i;
			flag = 1;
		}
	if (flag == 1) {
		cout << "\n\nSystem is in
Deadlock and the Deadlock process are\n";
		for (i = 0; i < noProc; i++) {
			cout << "P" <<
deadlock[i] << " ";
		}
	} else {
		cout << "\nNo Deadlock
Occur";
	}
	return (0);
}
```

**Output:**


#include <iostream>
#include <cstring>
using namespace std;
int main() {
int noProc, noReso, i, j, k;
cout << "Enter no of processes :";
cin >> noProc ;
cout << "Enter no of resources :";
cin >> noReso;
cout << "Enter allocation matrix :\n";
int alloc[noProc][noReso];
for (int l = 0; l < noProc; ++l)
for (int m = 0; m < noReso; ++m)
cin >> alloc[l][m];
cout << "Enter max resource matrix :\n";
int max[noProc][noReso];
for (int l = 0; l < noProc; ++l)
for (int m = 0; m < noReso; ++m)
cin >> max[l][m];
cout << "Enter availability of resources:\n";
int avail[noReso];
for (int l = 0; l < noReso; ++l)
cin >> avail[l];
int f[noProc], ans[noProc], ind = 0;
memset(f, 0, sizeof(f));
int need[noProc][noReso];
for (i = 0; i < noProc; i++) {
for (j = 0; j < noReso; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
Enter no of processes and resources :[Cancelled]
Enter no of processes :2
Enter no of resources :2
Enter allocation matrix :
1 0
0 1
Enter max resource matrix :
0 1
1 0
Enter availability of resources:
0 0
System is in Deadlock and the Deadlock process are
P0 P1
Process returned 0 (0x0) execution time : 27.950 s
Press any key to continue.
r1
p0
deadlock
p1
r2

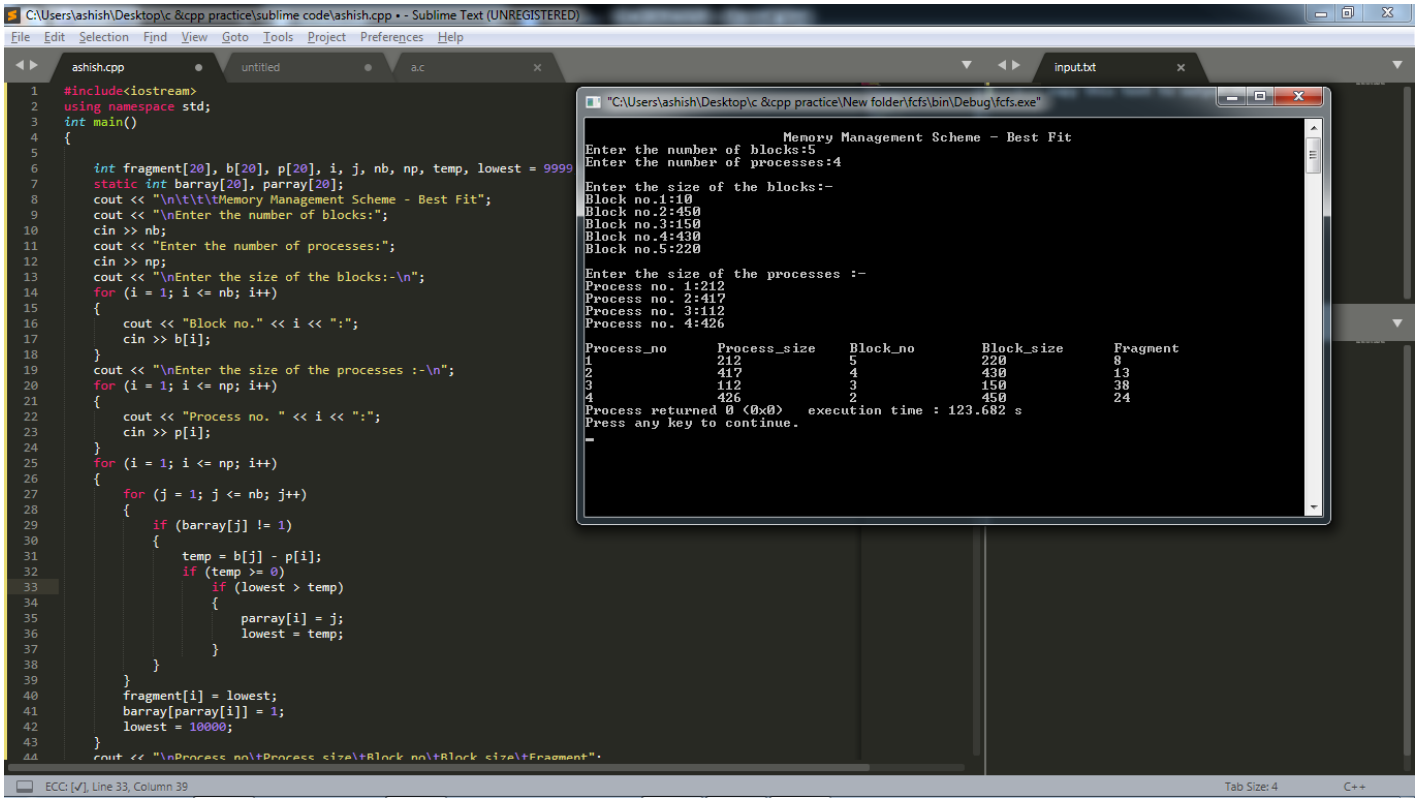

#include <iostream>
#include <cstring>
using namespace std;
int main() {
int noProc, noReso, i, j, k;
cout << "Enter no of processes :";
cin >> noProc ;
cout << "Enter no of resources :";
cin >> noReso;
cout << "Enter allocation matrix :\n";
int alloc[noProc][noReso];
for (int l = 0; l < noProc; ++l)
for (int m = 0; m < noReso; ++m)
cin >> alloc[l][m];
cout << "Enter max resource matrix :\n";
int max[noProc][noReso];
for (int l = 0; l < noProc; ++l)
for (int m = 0; m < noReso; ++m)
cin >> max[l][m];
cout << "Enter availability of resources:\n";
int avail[noReso];
for (int l = 0; l < noReso; ++l)
cin >> avail[l];
int f[noProc], ans[noProc], ind = 0;
memset(f, 0, sizeof(f));
int need[noProc][noReso];
for (i = 0; i < noProc; i++) {
for (j = 0; j < noReso; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
Enter no of processes and resources :[Cancelled]
Enter no of processes :3
Enter no of resources :2
Enter allocation matrix :
1 0
0 1
0 1
Enter max resource matrix :
0 1
1 0
0 0
Enter availability of resources:
0 0
No Deadlock Occur
Process returned 0 (0x0) execution time : 22.202 s
Press any key to continue.
r1
p0
no deadlock
p1
r2
p2

## PROGRAM 11 : best-fit contiguous memory allocation.

**Code :**

```
#include<iostream>
using namespace std;
int main()
{

	int fragment[20], b[20], p[20], i, j, nb, np, temp, lowest = 9999;
	static int barray[20], parray[20];
	cout << "\n\t\t\tMemory Management Scheme - Best Fit";
	cout << "\nEnter the number of blocks:";
	cin >> nb;
	cout << "Enter the number of processes:";
	cin >> np;
	cout << "\nEnter the size of the blocks:-\n";
	for (i = 1; i <= nb; i++)
	{
		cout << "Block no." << i << ":";
		cin >> b[i];
	}
	cout << "\nEnter the size of the processes :-\n";
	for (i = 1; i <= np; i++)
	{
		cout << "Process no. " << i << ":";
		cin >> p[i];
	}
	for (i = 1; i <= np; i++)
	{
		for (j = 1; j <= nb; j++)
		{
			if (barray[j] != 1)
			{
				temp = b[j] - p[i];
				if (temp >= 0)
					if (lowest > temp)
					{
						parray[i] = j;
						lowest = temp;
					}
			}
		}
		fragment[i] = lowest;
		barray[parray[i]] = 1;
		lowest = 10000;
	}
	cout << "\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment";
	for (i = 1; i <= np && parray[i] != 0; i++)
		cout << "\n" << i << "\t\t" << p[i] << "\t\t" << parray[i] << "\t\t" << b[parray[i]] << "\t\t" << fragment[i];
	return 0;
}
```

**OUTPUT:**


C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
untitled
a.c
input.txt
#include<iostream>
using namespace std;
int main()
{
int fragment[20], b[20], p[20], i, j, nb, np, temp, lowest = 9999
static int barray[20], parray[20];
cout << "\n\t\t\tMemory Management Scheme - Best Fit";
cout << "\nEnter the number of blocks:";
cin >> nb;
cout << "Enter the number of processes:";
cin >> np;
cout << "\nEnter the size of the blocks:-\n";
for (i = 1; i <= nb; i++)
{
cout << "Block no." << i << ":";
cin >> b[i];
}
cout << "\nEnter the size of the processes :-\n";
for (i = 1; i <= np; i++)
{
cout << "Process no. " << i << ":";
cin >> p[i];
}
for (i = 1; i <= np; i++)
{
for (j = 1; j <= nb; j++)
{
if (barray[j] != 1)
{
temp = b[j] - p[i];
if (temp >= 0)
if (lowest > temp)
{
parray[i] = j;
lowest = temp;
}
}
}
fragment[i] = lowest;
barray[parray[i]] = 1;
lowest = 10000;
}
"C:\Users\ashish\Desktop\c &cpp practice\New folder\fcfs\bin\Debug\fcfs.exe"
Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4
Enter the size of the blocks:-
Block no.1:10
Block no.2:450
Block no.3:150
Block no.4:430
Block no.5:220
Enter the size of the processes :-
Process no. 1:212
Process no. 2:417
Process no. 3:112
Process no. 4:426
Process_no Process_size Block_no Block_size Fragment
1 212 5 220 8
2 417 4 430 13
3 112 3 150 38
4 426 2 450 24
Process returned 0 (0x0) execution time : 123.682 s
Press any key to continue.
ECC: [✓], Line 33, Column 39
Tab Size: 4
C++

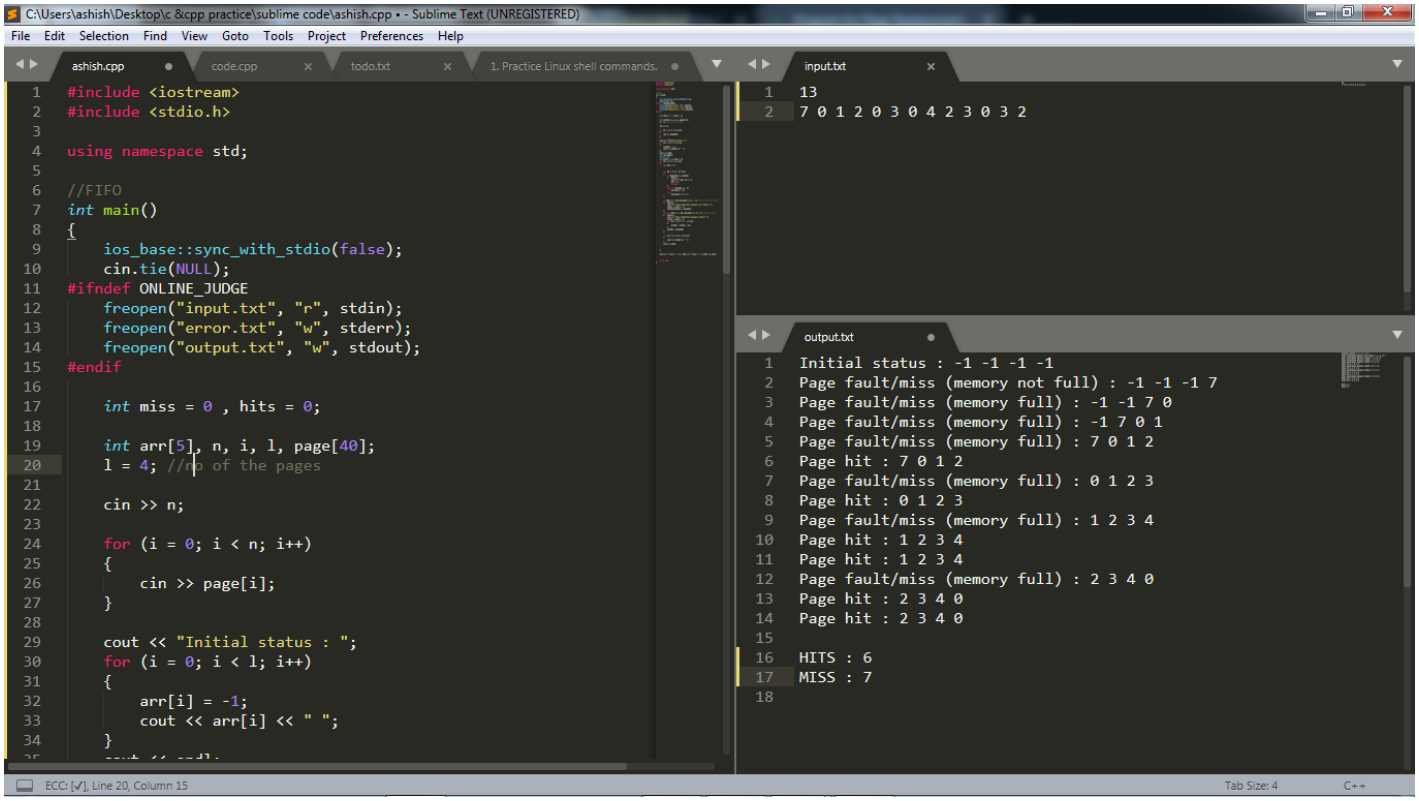
## PROGRAM 12 - Program to stimulate FIFO page replacement algorithm

**Code :**

```
#include <iostream>
#include <stdio.h>

using namespace std;

//only give positive nos. as input
int main()
{
	ios_base::sync_with_stdio(false);
	cin.tie(NULL);
#ifndef ONLINE_JUDGE
	freopen("input.txt", "r", stdin);
	freopen("error.txt", "w", stderr);
	freopen("output.txt", "w", stdout);
#endif

	int miss = 0 , hits = 0;

	int arr[5], n, i, l, page[40];
	l = 4; //no of the pages

	cin >> n;

	for (i = 0; i < n; i++)
	{
		cin >> page[i];
	}

	cout << "Initial status : ";
	for (i = 0; i < l; i++)
	{
		arr[i] = -1;
		cout << arr[i] << " ";
	}
	cout << endl;
	int end_index;
	int j = 0;
	int count = 0, flag = 0;
	for (i = 0; i < n; i++)
	{
		int hit = 0;


		for (j = 0; j < l; j++)
		{
			if (page[i] == arr[j])
			{	hits++;
				cout << "Page hit : ";
				hit = 1;
				break;
			}
			else if (arr[j] == -1)
				end_index = j;
			else
				end_index = l + 1;
		}

	if (hit == 0 && end_index <= l - 1) //only for the first element
	{	miss++;
```

```
                cout << "Page fault/miss (memory not full) : ";
                count = count + 1;
                arr[end_index] = page[i];
        }
        else if (hit == 0 && end_index == l + 1) //memory full
        {       miss++;
                cout << "Page fault/miss (memory full) : ";
                count = count + 1;
                for (j = 0; j < l - 1; j++)
                {
                        arr[j] = arr[j + 1];
                }
                arr[j] = page[i];
        }

        for (int k = 0; k < l; k++)
        {
                cout << arr[k] << " ";
        }
        cout << endl;


    }

    cout << "\nHITS : " <<  hits << " MISS : " << miss << endl;


    return 0;
}
```

## Output:




```cpp
#include <iostream>
#include <stdio.h>

using namespace std;

//FIFO
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("error.txt", "w", stderr);
    freopen("output.txt", "w", stdout);
#endif

    int miss = 0 , hits = 0;

    int arr[5], n, i, l, page[40];
    l = 4; //no of the pages

    cin >> n;

    for (i = 0; i < n; i++)
    {
        cin >> page[i];
    }

    cout << "Initial status : ";
    for (i = 0; i < l; i++)
    {
        arr[i] = -1;
        cout << arr[i] << " ";
    }
```

```
13
7 0 1 2 0 3 0 4 2 3 0 3 2
```

```
Initial status : -1 -1 -1 -1
Page fault/miss (memory not full) : -1 -1 -1 7
Page fault/miss (memory full) : -1 -1 7 0
Page fault/miss (memory full) : -1 7 0 1
Page fault/miss (memory full) : 7 0 1 2
Page hit : 7 0 1 2
Page fault/miss (memory full) : 0 1 2 3
Page hit : 0 1 2 3
Page fault/miss (memory full) : 1 2 3 4
Page hit : 1 2 3 4
Page hit : 1 2 3 4
Page fault/miss (memory full) : 2 3 4 0
Page hit : 2 3 4 0
Page hit : 2 3 4 0

HITS : 6
MISS : 7

```

## PROGRAM 13 - Program to stimulate LRU page replacement algorithm

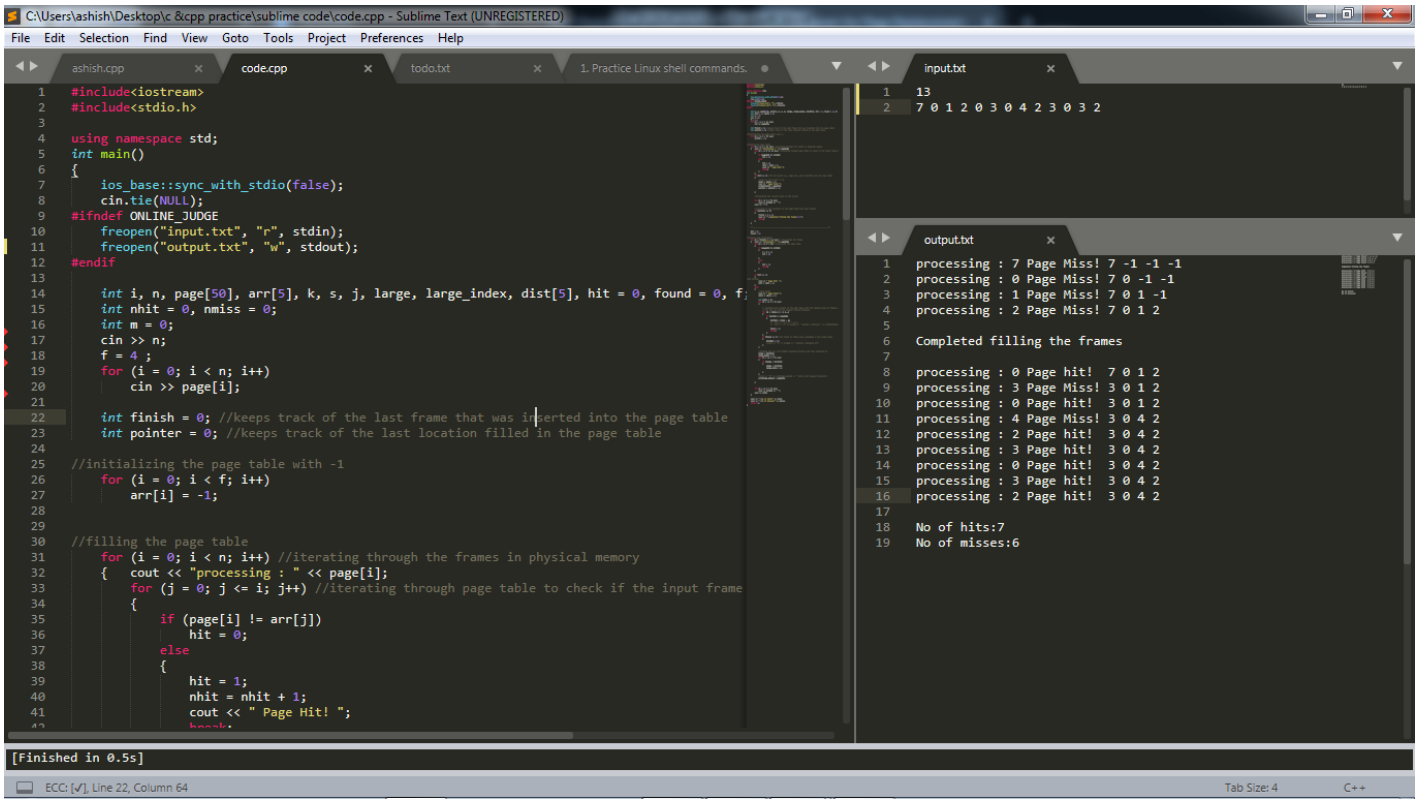
**Code:**

```
#include<iostream>
#include<stdio.h>

using namespace std;
int main()
{
	ios_base::sync_with_stdio(false);
	cin.tie(NULL);
#ifndef ONLINE_JUDGE
	freopen("input.txt", "r", stdin);
	freopen("output.txt", "w", stdout);
#endif

	int i, n, page[50], arr[5], k, s, j, large, large_index, dist[5], hit = 0, found = 0, f;
	int nhit = 0, nmiss = 0;
	int m = 0;
	cin >> n;
	f = 4 ;
	for (i = 0; i < n; i++)
		cin >> page[i];

	int finish = 0; //keeps track of the last frame that was inserted into the page table
	int pointer = 0; //keeps track of the last location filled in the page table

//initializing the page table with -1
	for (i = 0; i < f; i++)
		arr[i] = -1;


//filling the page table
	for (i = 0; i < n; i++) //iterating through the frames in physical memory
	{	cout << "processing : " << page[i];
		for (j = 0; j <= i; j++) //iterating through page table to check if the input frame is already present in the page
table or not
		{
			if (page[i] != arr[j])
				hit = 0;
			else
			{
				hit = 1;
				nhit = nhit + 1;
				cout << " Page Hit! ";
				break;
			}
		}
		if (hit == 0) //if not presnt i.e. page miss ,then inserted into the page table
		{
			//cout << "\n j is:" << j;
			nmiss = nmiss + 1;
			cout << " Page Miss! ";
			arr[pointer] = page[i];
			pointer = pointer + 1;

		}

		//Displaying the current state of the array!

		for (k = 0; k < f; k++)
			cout << arr[k] << " ";
```

```
                cout << "\n";

                //checking if all locations of the page table have been filled
                if (pointer == f)
                {
                        finish = i + 1;
                        cout << "\nCompleted filling the frames\n\n";
                        break;
                }
        }

        /*--------------------------------------------------------------------------*/

        hit = 0;
        found = 0;

//optimal page replacement
        for (i = finish; i < n; i++) // traversing the frames
        {       cout << "processing : " << page[i];
                for (j = 0; j < f;) //traversing the page table
                {
                        if (page[i] != arr[j])
                        {
                                j = j + 1;
                                hit = 0;

                        }
                        else
                        {
                                hit = 1;
                                break;
                        }
                }

                if (hit == 1)
                {
//do nothing
                        cout << " Page hit!  ";
                        nhit = nhit + 1;
                }
                else
                {
                        //Page fault
                        cout << " Page Miss! ";
                        nmiss = nmiss + 1;

                        int index = i;
                        for (k = 0; k < f; k++)
                        {

                                // checking each element of the page table with the remaing array of frames, to check
                                // for the one with the longest forward distance
                                for (m = index; m > 0; m--)
                                {
                                        if (arr[k] == page[m])
                                        {
                                                dist[k] = index - m;
                                                //      cout << "\n m is " << m;
                                                //      cout << "\n" << arr[k] << " alloted a distance " << int(dist[k]);

                                                found = 1;
                                                break;
```

```
                                        }
                                }
                                if (found == 0) //not found in frame array ,assigned a very large value
                                {
                                        dist[k] = 99;
                                        //cout << "\n" << arr[k] << " alloted a distance 99";
                                }
                        }

                        //finding the one with largest backward distance and then replacing it
                        large = dist[0];
                        large_index = 0;
                        for (s = 1; s < f; s++)
                        {
                                if (large < dist[s])
                                {
                                        large = dist[s];
                                        large_index = s;

                                }
                        }
                        //cout << "\n" << arr[large_index] << " stands with largest distance";
                        arr[large_index] = page[i];

                }


                for (k = 0; k < f; k++)
                        cout << arr[k] << " ";
                cout << endl;
        }

        cout << "\nNo of hits:" << nhit;
        cout << "\nNo of misses:" << nmiss;
        return 0;
}
```

## Output:




```
#include<iostream>
#include<stdio.h>

using namespace std;
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif

    int i, n, page[50], arr[5], k, s, j, large, large_index, dist[5], hit = 0, found = 0, f;
    int nhit = 0, nmiss = 0;
    int m = 0;
    cin >> n;
    f = 4 ;
    for (i = 0; i < n; i++)
        cin >> page[i];

    int finish = 0; //keeps track of the last frame that was inserted into the page table
    int pointer = 0; //keeps track of the last location filled in the page table

//initializing the page table with -1
    for (i = 0; i < f; i++)
        arr[i] = -1;


//filling the page table
    for (i = 0; i < n; i++) //iterating through the frames in physical memory
    {   cout << "processing : " << page[i];
        for (j = 0; j <= i; j++) //iterating through page table to check if the input frame
        {
            if (page[i] != arr[j])
                hit = 0;
            else
            {
                hit = 1;
                nhit = nhit + 1;
                cout << " Page Hit! ";
                break;
```

```
13
7 0 1 2 0 3 0 4 2 3 0 3 2
```

```
processing : 7 Page Miss! 7 -1 -1 -1
processing : 0 Page Miss! 7 0 -1 -1
processing : 1 Page Miss! 7 0 1 -1
processing : 2 Page Miss! 7 0 1 2

Completed filling the frames

processing : 0 Page hit!  7 0 1 2
processing : 3 Page Miss! 3 0 1 2
processing : 0 Page hit!  3 0 1 2
processing : 4 Page Miss! 3 0 4 2
processing : 2 Page hit!  3 0 4 2
processing : 3 Page hit!  3 0 4 2
processing : 0 Page hit!  3 0 4 2
processing : 3 Page hit!  3 0 4 2
processing : 2 Page hit!  3 0 4 2

No of hits:7
No of misses:6
```

## PROGRAM 14 - Second Chance page replacement algorithm.

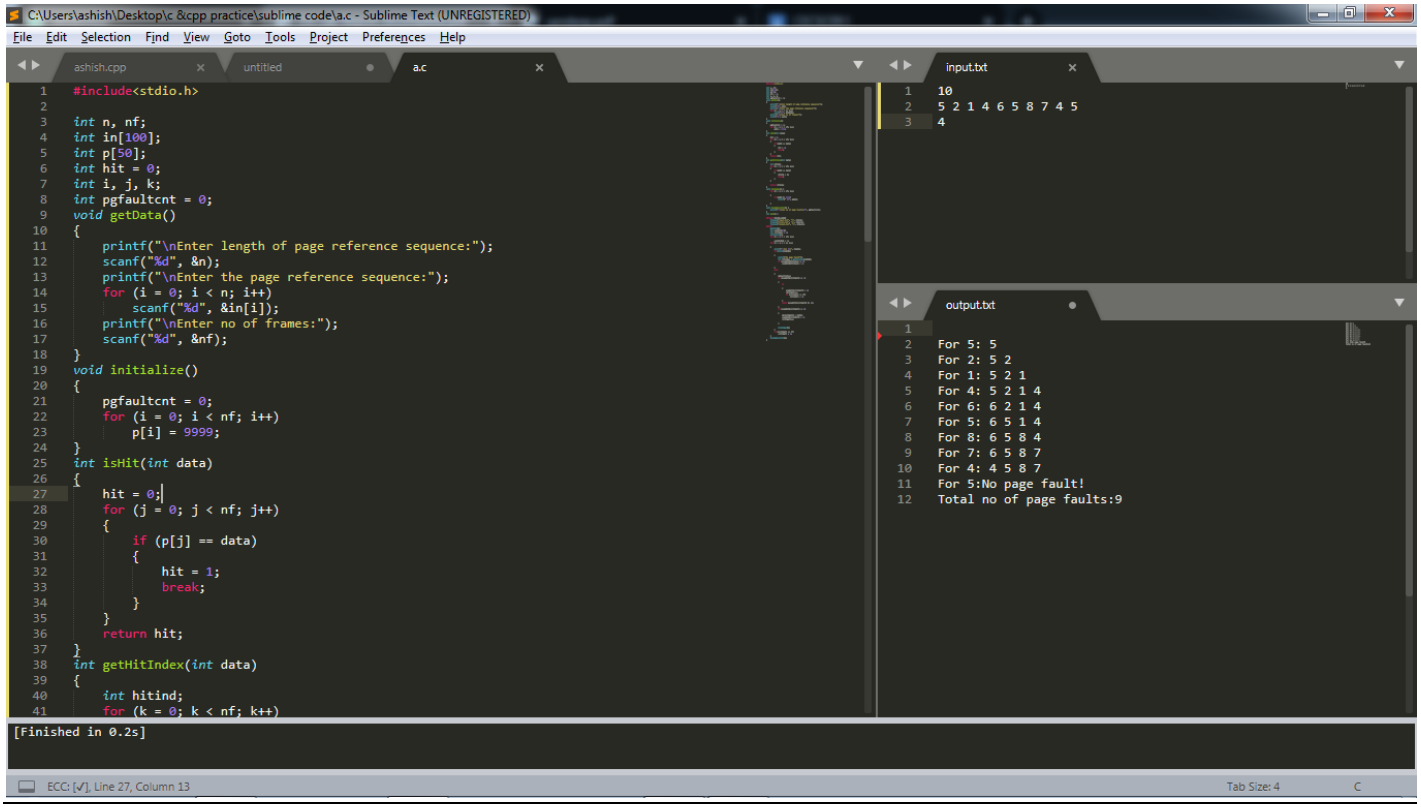
**CODE :**

```
#include<stdio.h>

int n, nf;
int in[100];
int p[50];
int hit = 0;
int i, j, k;
int pgfaultcnt = 0;
void getData()
{
	printf("\nEnter length of page reference sequence:");
	scanf("%d", &n);
	printf("\nEnter the page reference sequence:");
	for (i = 0; i < n; i++)
		scanf("%d", &in[i]);
	printf("\nEnter no of frames:");
	scanf("%d", &nf);
}
void initialize()
{
	pgfaultcnt = 0;
	for (i = 0; i < nf; i++)
		p[i] = 9999;
}
int isHit(int data)
{
	hit = 0;
	for (j = 0; j < nf; j++)
	{
		if (p[j] == data)
		{
			hit = 1;
			break;
		}
	}
	return hit;
}
int getHitIndex(int data)
{
	int hitind;
	for (k = 0; k < nf; k++)
	{
		if (p[k] == data)
		{
			hitind = k;
			break;
		}
	}

	return hitind;
}
void dispPages() {
	for (k = 0; k < nf; k++)

	{
		if (p[k] != 9999)
			printf(" %d", p[k]);

	}
```

```
}
void dispPgFaultCnt() {
	printf("\nTotal no of page faults:%d", pgfaultcnt);
}
int main() {

#ifndef ONLINE_JUDGE
	freopen("input.txt", "r", stdin);
	freopen("error.txt", "w", stderr);
	freopen("output.txt", "w", stdout);
#endif
	getData();
	int usedbit[50];
	int victimptr = 0;
	initialize();
	for (i = 0; i < nf; i++)

		usedbit[i] = 0;
	for (i = 0; i < n; i++)

	{
		printf("\nFor %d:", in[i]);
		if (isHit(in[i]))

		{
			printf("No page fault!");
			int hitindex = getHitIndex(in[i]);
			if (usedbit[hitindex] == 0)
				usedbit[hitindex] = 1;

		}
		else

		{
			pgfaultcnt++;
			if (usedbit[victimptr] == 1)

			{
				do

				{
					usedbit[victimptr] = 0;
					victimptr++;
					if (victimptr == nf)
						victimptr = 0;

				}
				while (usedbit[victimptr] != 0);

			}
			if (usedbit[victimptr] == 0)

			{
				p[victimptr] = in[i];
				usedbit[victimptr] = 1;
				victimptr++;

			}

			dispPages();
		}
		if (victimptr == nf)
```

```
                victimptr = 0;
        }
        dispPgFaultCnt();
}
```

## OUTPUT :



## PROGRAM : 15 Enhanced Second Chance page replacement algorithm.

**CODE :**

```
#include <iostream>
using namespace std;
void SecondChance_page_replacement(int n, int no_of_frames, int pages[], bool modified[])
{
	int pointer = 0;
	int page_faults = 0;
	int frames[no_of_frames];
	bool ref[no_of_frames] = {0};
	bool mod[no_of_frames] = {0};
	for (int i = 0; i < no_of_frames; ++i)
		frames[i] = -1;
	for (int i = 0; i < n; i++)
	{
		int j;
		for (j = 0; j < no_of_frames; j++)
		{
			if (frames[j] == -1)
			{
				page_faults++;
				frames[j] = pages[i];
				mod[j] = modified[i];
				cout << pages[i] << ": Page fault\t\t";
				break;
			}
			else if (frames[j] == pages[i])
			{
				ref[j] = 1;
				cout << pages[i] << ": Page hit\t\t";
				break;
			}
		}
		if (j == no_of_frames)
		{
			cout << "ENTERED FOR REPLACEMENT\n";
			int k = 0;


			while (true)
			{
				while (k < no_of_frames)
				{
					if (ref[pointer] == 0 && mod[pointer] == 0)
					{
						cout << "00 FOUND\n";
						frames[pointer] = pages[i];
						mod[pointer] = modified[i];
						pointer = (pointer + 1) % no_of_frames;
						cout << pages[i] << ": Page fault\t\t";
						break;
					}
					pointer = (pointer + 1) % no_of_frames;
					k++;
				}
				if (k == no_of_frames)
				{
					cout << "NO 00 FOUND\n";
					while (k--)
					{
						if (ref[pointer] == 0 && mod[pointer] == 1)
```

```
                                        {
                                                cout << "01 FOUND\n";
                                                frames[pointer] = pages[i];
                                                mod[pointer] = modified[i];
                                                pointer = (pointer + 1) % no_of_frames;
                                                cout << pages[i] << ": Page fault\t\t";
                                                page_faults++;
                                                break;
                                        }
                                        else
                                        {
                                                ref[pointer] = 0;
                                                pointer = (pointer + 1) % no_of_frames;
                                        }
                                }
                                if (k != 0)
                                        break;
                        }
                        else
                                break;


                }
        }
}
}
```

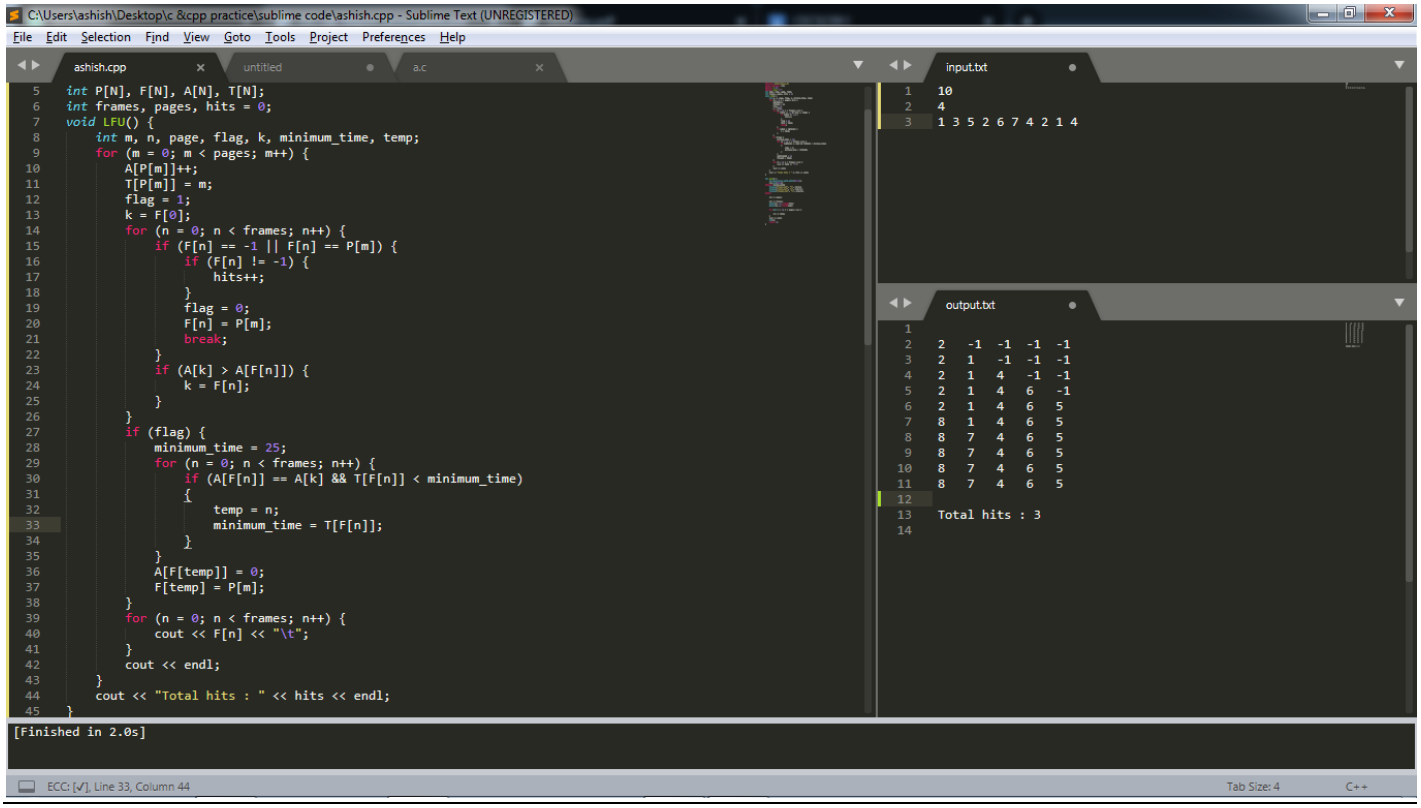
## PROGRAM : 16 : LFU page replacement algorithm.

**CODE :**

```
#include <bits/stdc++.h>
using namespace std;
#define N 100
#define endl "\n"
int P[N], F[N], A[N], T[N];
int frames, pages, hits = 0;
void LFU() {
	int m, n, page, flag, k, minimum_time, temp;
	for (m = 0; m < pages; m++) {
		A[P[m]]++;
		T[P[m]] = m;
		flag = 1;
		k = F[0];
		for (n = 0; n < frames; n++) {
			if (F[n] == -1 || F[n] == P[m]) {
				if (F[n] != -1) {
					hits++;
				}
				flag = 0;
				F[n] = P[m];
				break;
			}
			if (A[k] > A[F[n]]) {
				k = F[n];
			}
		}
		if (flag) {
			minimum_time = 25;
			for (n = 0; n < frames; n++) {
				if (A[F[n]] == A[k] && T[F[n]] < minimum_time)
				{
					temp = n;
					minimum_time = T[F[n]];
				}
			}
			A[F[temp]] = 0;
			F[temp] = P[m];
		}
		for (n = 0; n < frames; n++) {
			cout << F[n] << "\t";
		}
		cout << endl;
	}
	cout << "Total hits : " << hits << endl;
}

int main() {
	ios_base::sync_with_stdio(false);
	cin.tie(NULL);
#ifndef ONLINE_JUDGE
	freopen("input.txt", "r", stdin);
	freopen("error.txt", "w", stderr);
	freopen("output.txt", "w", stdout);
#endif

	cin >> pages;

	cin >> frames;
	memset(F, -1, sizeof(F));
```

```
        memset(A, 0, sizeof(A));

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

                cin >> P[i];
        }
        cout << endl;
        LFU();
        return 0;
}
```

## OUTPUT :

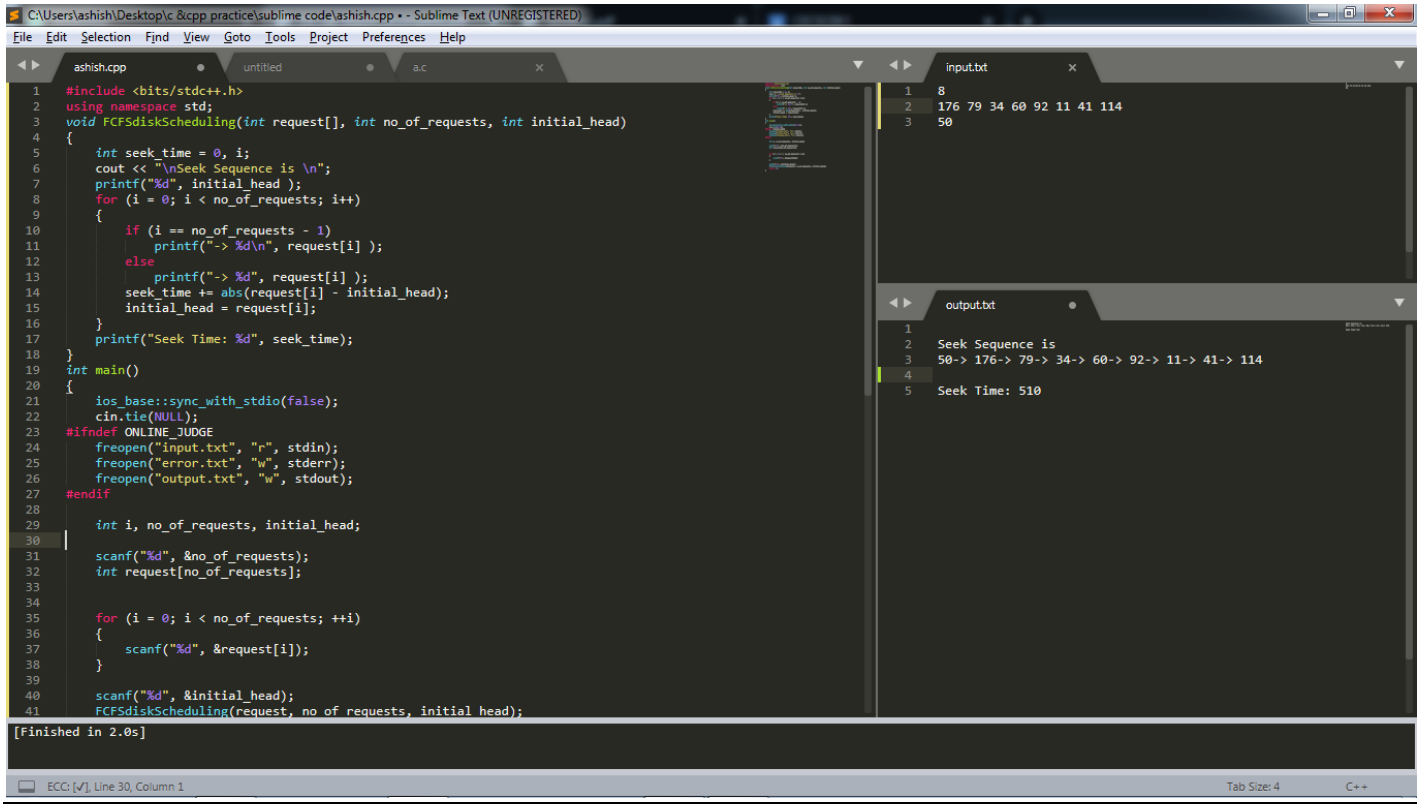


## PROGRAM 17 : FCFS disk scheduling algorithm.

**CODE :**

```
#include <bits/stdc++.h>
using namespace std;
void FCFSdiskScheduling(int request[], int no_of_requests, int initial_head)
{
	int seek_time = 0, i;
	cout << "\nSeek Sequence is \n";
	printf("%d", initial_head );
	for (i = 0; i < no_of_requests; i++)
	{
		if (i == no_of_requests - 1)
			printf("-> %d\n", request[i] );
		else
			printf("-> %d", request[i] );
		seek_time += abs(request[i] - initial_head);
		initial_head = request[i];
	}
	printf("Seek Time: %d", seek_time);
}
int main()
{
	ios_base::sync_with_stdio(false);
	cin.tie(NULL);
#ifndef ONLINE_JUDGE
	freopen("input.txt", "r", stdin);
	freopen("error.txt", "w", stderr);
	freopen("output.txt", "w", stdout);
#endif

	int i, no_of_requests, initial_head;

	scanf("%d", &no_of_requests);
	int request[no_of_requests];


	for (i = 0; i < no_of_requests; ++i)
	{
		scanf("%d", &request[i]);
	}

	scanf("%d", &initial_head);
	FCFSdiskScheduling(request, no_of_requests, initial_head);
	return 0;
}
```

## OUTPUT :

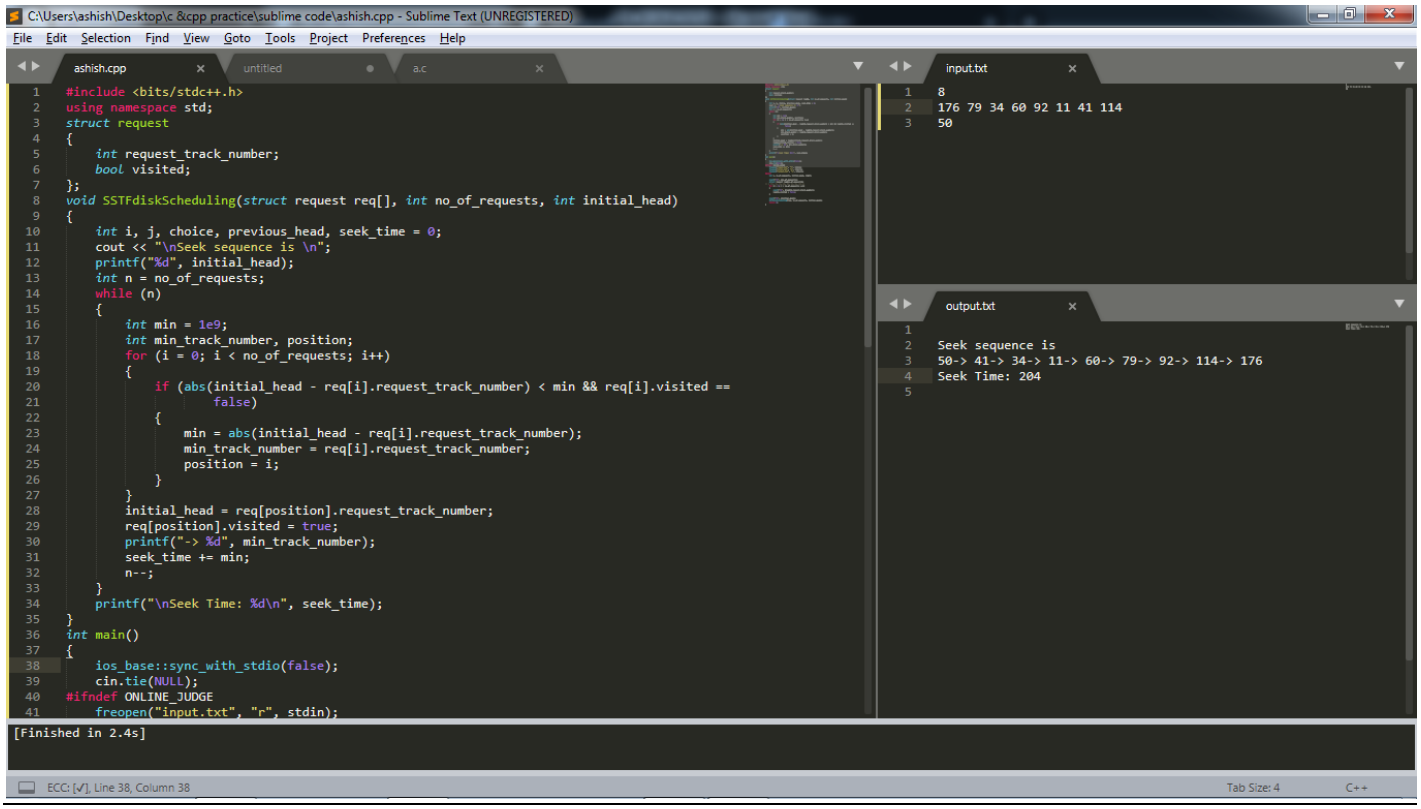

C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
untitled
a.c
#include <bits/stdc++.h>
using namespace std;
void FCFSdiskScheduling(int request[], int no_of_requests, int initial_head)
{
int seek_time = 0, i;
cout << "\nSeek Sequence is \n";
printf("%d", initial_head );
for (i = 0; i < no_of_requests; i++)
{
if (i == no_of_requests - 1)
printf("-> %d\n", request[i] );
else
printf("-> %d", request[i] );
seek_time += abs(request[i] - initial_head);
initial_head = request[i];
}
printf("Seek Time: %d", seek_time);
}
int main()
{
ios_base::sync_with_stdio(false);
cin.tie(NULL);
#ifndef ONLINE_JUDGE
freopen("input.txt", "r", stdin);
freopen("error.txt", "w", stderr);
freopen("output.txt", "w", stdout);
#endif
int i, no_of_requests, initial_head;
scanf("%d", &no_of_requests);
int request[no_of_requests];
for (i = 0; i < no_of_requests; ++i)
{
scanf("%d", &request[i]);
}
scanf("%d", &initial_head);
FCFSdiskScheduling(request, no_of_requests, initial_head);
input.txt
8
176 79 34 60 92 11 41 114
50
output.txt
Seek Sequence is
50-> 176-> 79-> 34-> 60-> 92-> 11-> 41-> 114
Seek Time: 510
[Finished in 2.0s]
ECC: [✓], Line 30, Column 1
Tab Size: 4
C++

## PROGRAM 18 : SSTF disk scheduling algorithm.

**CODE :**

```
#include <bits/stdc++.h>
using namespace std;
struct request
{
	int request_track_number;
	bool visited;
};
void SSTFdiskScheduling(struct request req[], int no_of_requests, int initial_head)
{
	int i, j, choice, previous_head, seek_time = 0;
	cout << "\nSeek sequence is \n";
	printf("%d", initial_head);
	int n = no_of_requests;
	while (n)
	{
		int min = 1e9;
		int min_track_number, position;
		for (i = 0; i < no_of_requests; i++)
		{
			if (abs(initial_head - req[i].request_track_number) < min && req[i].visited ==
			    false)
			{
				min = abs(initial_head - req[i].request_track_number);
				min_track_number = req[i].request_track_number;
				position = i;
			}
		}
		initial_head = req[position].request_track_number;
		req[position].visited = true;
		printf("-> %d", min_track_number);
		seek_time += min;
		n--;
	}
	printf("\nSeek Time: %d\n", seek_time);
}
int main()
{
	ios_base::sync_with_stdio(false);
	cin.tie(NULL);
#ifndef ONLINE_JUDGE
	freopen("input.txt", "r", stdin);
	freopen("error.txt", "w", stderr);
	freopen("output.txt", "w", stdout);
#endif
	int i, no_of_requests, initial_head, limit;

	scanf("%d", &no_of_requests);
	struct request req[no_of_requests];
// { 176, 79, 34, 60, 92, 11, 41, 114 };
	for (i = 0; i < no_of_requests; ++i)
	{
		scanf("%d", &req[i].request_track_number);
		req[i].visited = false;
	}

	scanf("%d", &initial_head);
	SSTFdiskScheduling(req, no_of_requests, initial_head);
	return 0;
}
```

## OUTPUT :


C:\Users\ashish\Desktop\c &cpp practice\sublime code\ashish.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
ashish.cpp
untitled
a.c
input.txt
output.txt
#include <bits/stdc++.h>
using namespace std;
struct request
{
int request_track_number;
bool visited;
};
void SSTFdiskScheduling(struct request req[], int no_of_requests, int initial_head)
{
int i, j, choice, previous_head, seek_time = 0;
cout << "\nSeek sequence is \n";
printf("%d", initial_head);
int n = no_of_requests;
while (n)
{
int min = 1e9;
int min_track_number, position;
for (i = 0; i < no_of_requests; i++)
{
if (abs(initial_head - req[i].request_track_number) < min && req[i].visited ==
false)
{
min = abs(initial_head - req[i].request_track_number);
min_track_number = req[i].request_track_number;
position = i;
}
}
initial_head = req[position].request_track_number;
req[position].visited = true;
printf("-> %d", min_track_number);
seek_time += min;
n--;
}
printf("\nSeek Time: %d\n", seek_time);
}
int main()
{
ios_base::sync_with_stdio(false);
cin.tie(NULL);
#ifndef ONLINE_JUDGE
freopen("input.txt", "r", stdin);
8
176 79 34 60 92 11 41 114
50
Seek sequence is
50-> 41-> 34-> 11-> 60-> 79-> 92-> 114-> 176
Seek Time: 204
[Finished in 2.4s]
ECC: [✓], Line 38, Column 38
Tab Size: 4
C++

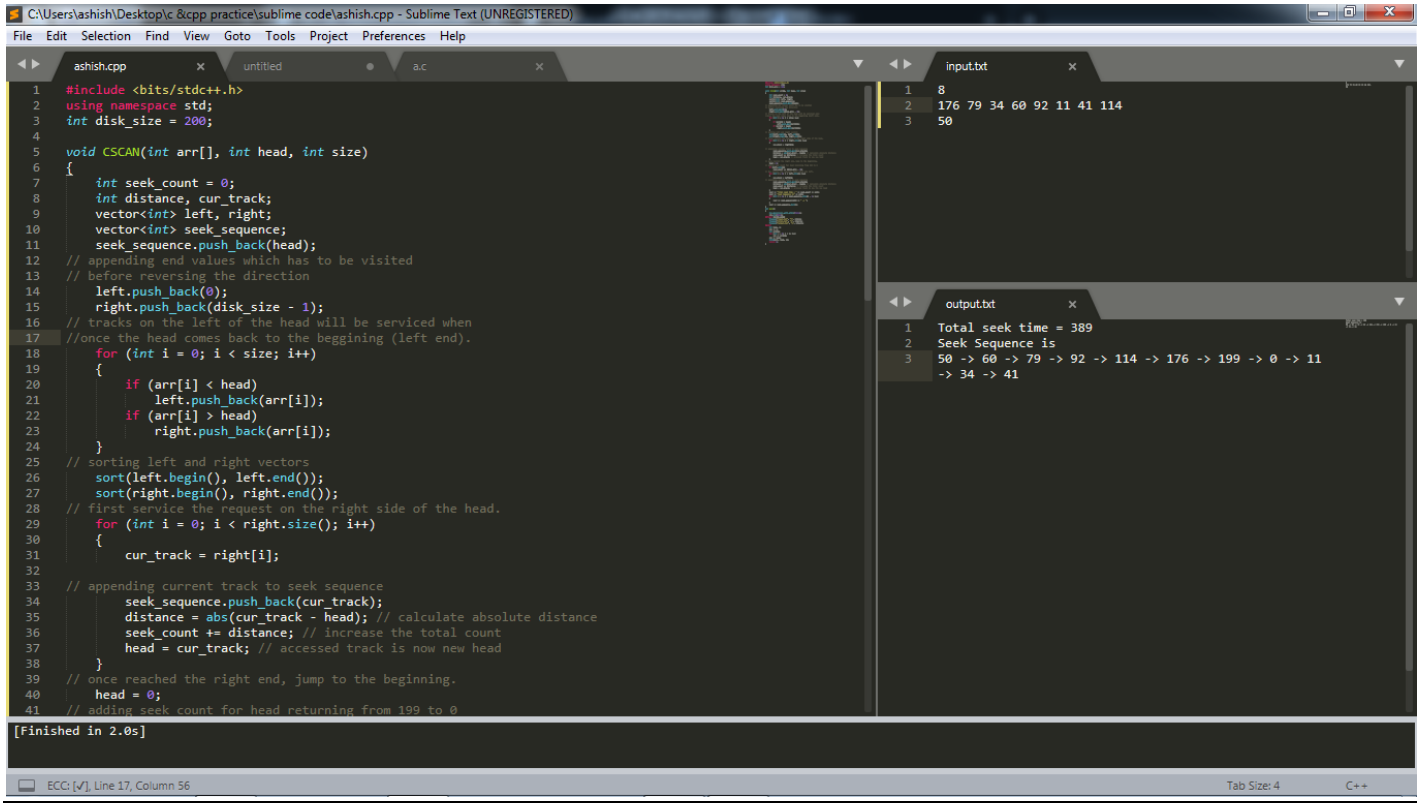
## PROGRAM 19 : C-SCAN disk scheduling algorithm.

**CODE :**

```
#include <bits/stdc++.h>
using namespace std;
int disk_size = 200;

void CSCAN(int arr[], int head, int size)
{
	int seek_count = 0;
	int distance, cur_track;
	vector<int> left, right;
	vector<int> seek_sequence;
	seek_sequence.push_back(head);
// appending end values which has to be visited
// before reversing the direction
	left.push_back(0);
	right.push_back(disk_size - 1);
// tracks on the left of the head will be serviced when
//once the head comes back to the beggining (left end).
	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]);
	}
// sorting left and right vectors
	sort(left.begin(), left.end());
	sort(right.begin(), right.end());
// first service the request on the right side of the head.
	for (int i = 0; i < right.size(); i++)
	{
		cur_track = right[i];

// appending current track to seek sequence
		seek_sequence.push_back(cur_track);
		distance = abs(cur_track - head); // calculate absolute distance
		seek_count += distance; // increase the total count
		head = cur_track; // accessed track is now new head
	}
// once reached the right end, jump to the beginning.
	head = 0;
// adding seek count for head returning from 199 to 0
	if (left.size())
		seek_count += (disk_size - 1);
// Now service the requests again which are left.
	for (int i = 0; i < left.size(); i++)
	{
		cur_track = left[i];
// appending current track to seek sequence
		seek_sequence.push_back(cur_track);
		distance = abs(cur_track - head); // calculate absolute distance
		seek_count += distance; // increase the total count
		head = cur_track; // accessed track is now the new head
	}
	cout << "Total seek time = " << seek_count << endl;
	cout << "Seek Sequence is" << endl;
	for (int i = 0; i < seek_sequence.size() - 1; i++)
	{
```

```
                cout << seek_sequence[i] << " -> ";
        }
        cout << seek_sequence.back();
}
int main()
{
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
#ifndef ONLINE_JUDGE
        freopen("input.txt", "r", stdin);
        freopen("error.txt", "w", stderr);
        freopen("output.txt", "w", stdout);
#endif
        int head, n;
        cin >> n;
        int arr[n];
        for (int i = 0; i < n; i++)
                cin >> arr[i];
        cin >> head;
        CSCAN(arr, head, n);
        return 0;
}
```

**OUTPUT :**



## PROGRAM : 20 LOOK disk scheduling algorithm.

**CODE :**

```
#include <bits/stdc++.h>
using namespace std;
int disk_size = 200;
void input(int arr[], int n)
{
	for (int i = 0; i < n; i++)
		cin >> arr[i];
}
void LOOK(int arr[], int head, int size, string direction)
{
	int seek_count = 0;
	int distance, cur_track;
	vector<int> left, right;
	vector<int> seek_sequence;
	seek_sequence.push_back(head);
// appending values which are currently at left
//and right direction from the head.
	for (int i = 0; i < size; i++)
	{
		if (arr[i] < head)
			left.push_back(arr[i]);
		else
			right.push_back(arr[i]);
	}
// sorting left and right vectors for
// servicing tracks in the correct sequence.
	std::sort(left.begin(), left.end());
	std::sort(right.begin(), right.end());
// run the while loop two times.
// to look in both direction
	int run = 2;
	while (run--)
	{
		if (direction == "left")
		{
			for (int i = left.size() - 1; i >= 0; i--)
			{

				cur_track = left[i];
				// appending current track to seek sequence
				seek_sequence.push_back(cur_track);
				distance = abs(cur_track - head); // calculate absolute distance
				seek_count += distance;// increase the total count
				head = cur_track;// accessed track is now the new head
			}
			direction = "right";// reversing the direction
		}
		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";// reversing the direction
                        }
        }
        cout << "Total seek time = " << seek_count << endl;
        cout << "Seek Sequence is" << endl;
        for (int i = 0; i < seek_sequence.size() - 1; i++)
        {
                cout << seek_sequence[i] << " -> ";
        }
        cout << seek_sequence.back();
}
int main()
{
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
#ifndef ONLINE_JUDGE
        freopen("input.txt", "r", stdin);
        freopen("error.txt", "w", stderr);
        freopen("output.txt", "w", stdout);
#endif
        int head, n;
        cin >> n;
        int arr[n];
        input(arr, n);
// { 176, 79, 34, 60, 92, 11, 41, 114 };
        cin >> head;
        string direction = "right";

        cin >> direction;
        LOOK(arr, head, n, direction);
        return 0;
}
```

**OUTPUT :**