# Lab Assignment 1

**Problem Statement**- Basic Linux commands

**Code** – mkdir mandar ls

cd mandar

touch “mandar.txt” ls

pwd

rm “mandar.txt” ls

touch “mandar1.txt”

printf "\nHi I am Mandar">>mandar1.txt printf "\nThis is second line">>mandar1.txt cat “mandar1.txt”

echo “Hello World” man

uname whoami grep “Hii”

printf "\nThis is second line">>mandar1.txt printf "\nThis is third line">>mandar1.txt printf "\nHi I am Mandar1">>mandar1.txt head -3 "mandar1.txt"

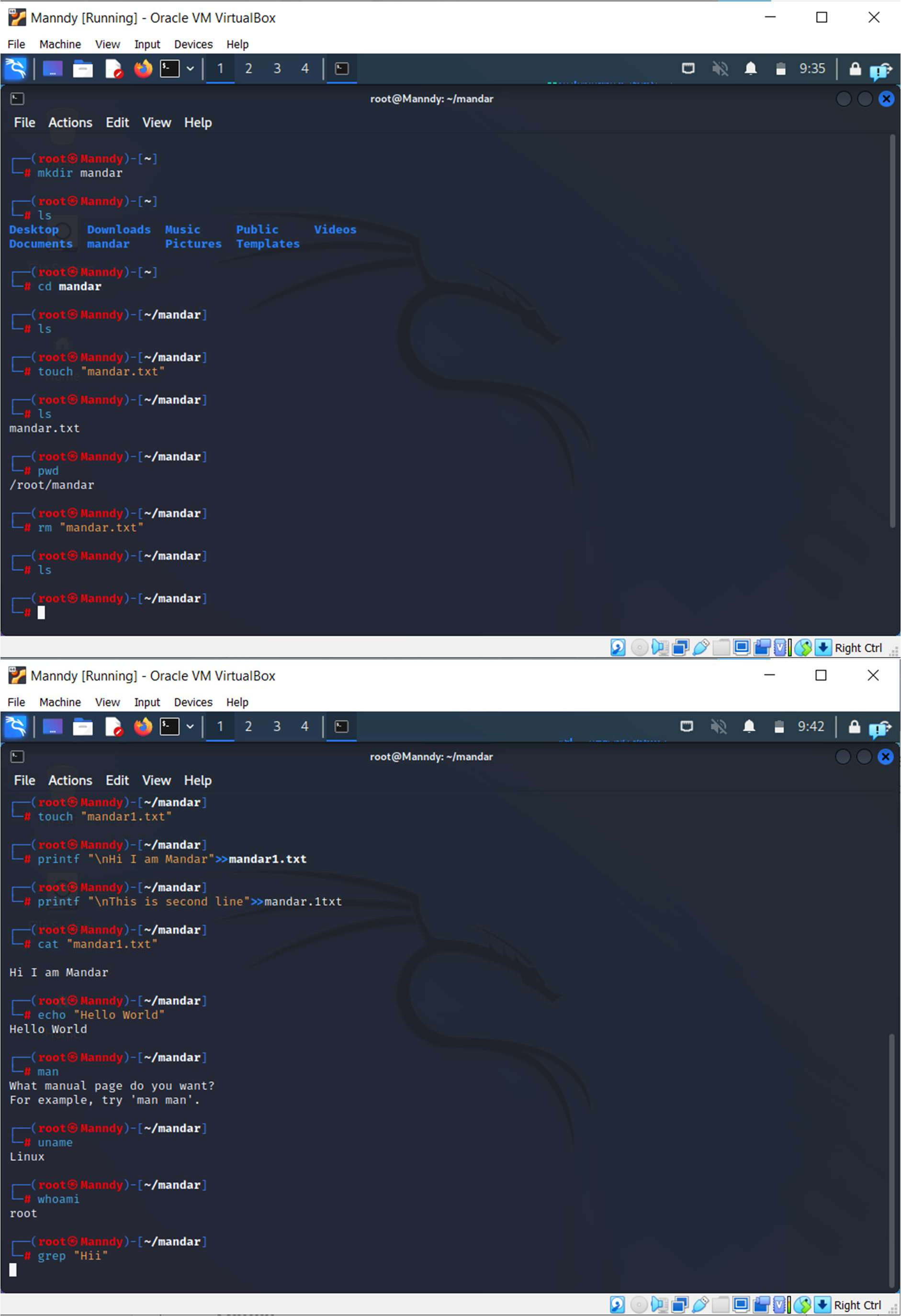
head -3 "mandar1.txt"

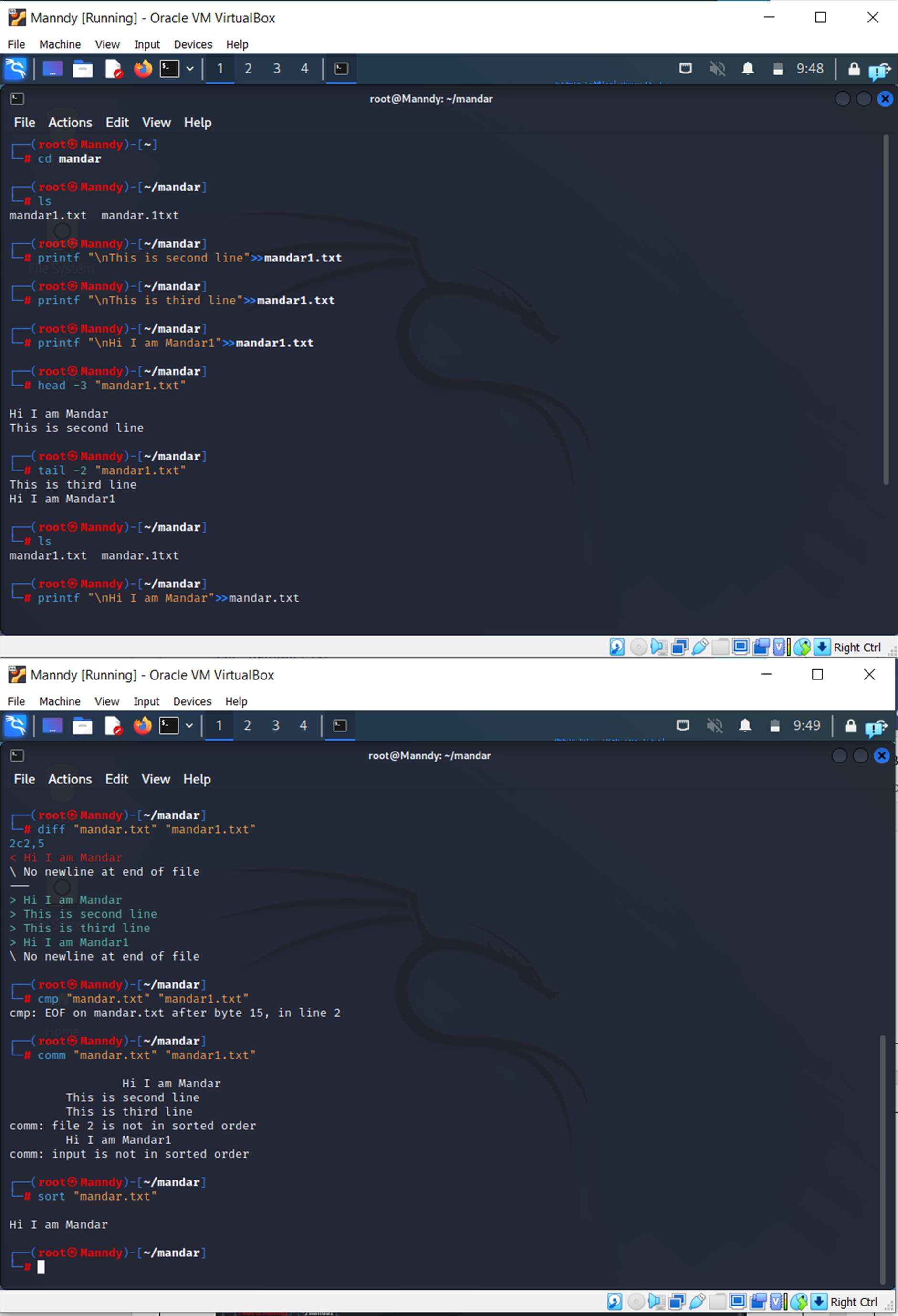
printf "\nHi I am Mandar">>mandar.txt diff "mandar.txt" "mandar1.txt"

cmp "mandar.txt" "mandar1.txt" comm "mandar.txt" "mandar1.txt"

sort "mandar.txt"

**Output –**





# Lab Assignment 2

**Problem Statement**- Bash programming programs

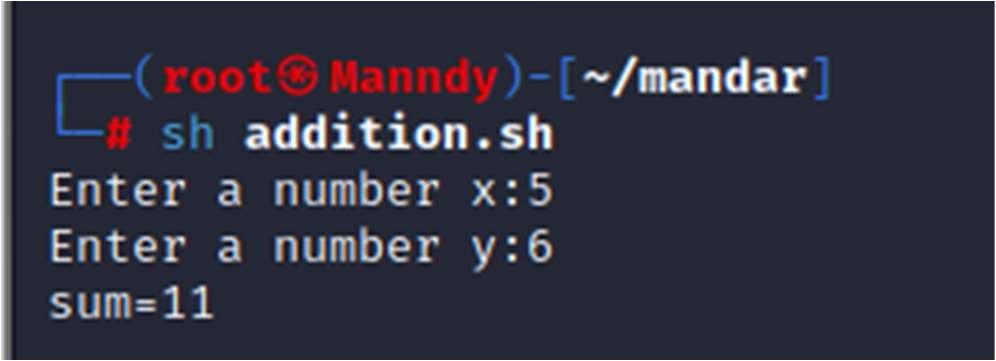
**Code and output** –

1. Addition:

echo -n "Enter a number x:" read x

echo -n "Enter a number y:" read y

sum=$(($x + $y)) echo "sum="$sum



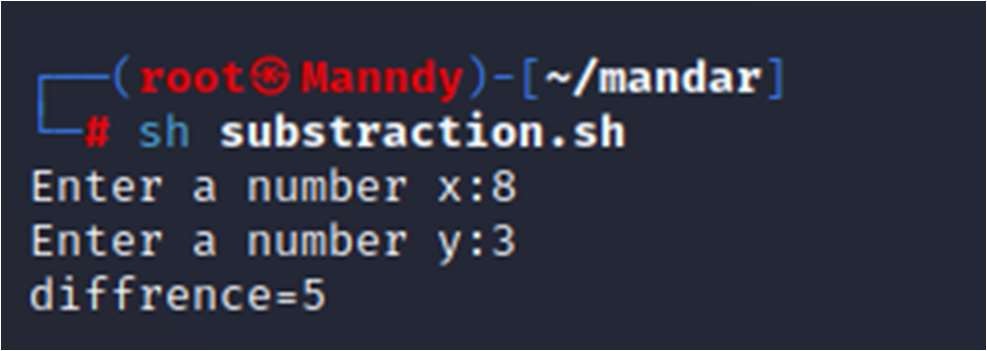
1. Substraction:

echo -n "Enter a number x:" read x

echo -n "Enter a number y:" read y

sum=$(($x - $y))

echo "difference="$sum



1. Even Odd:

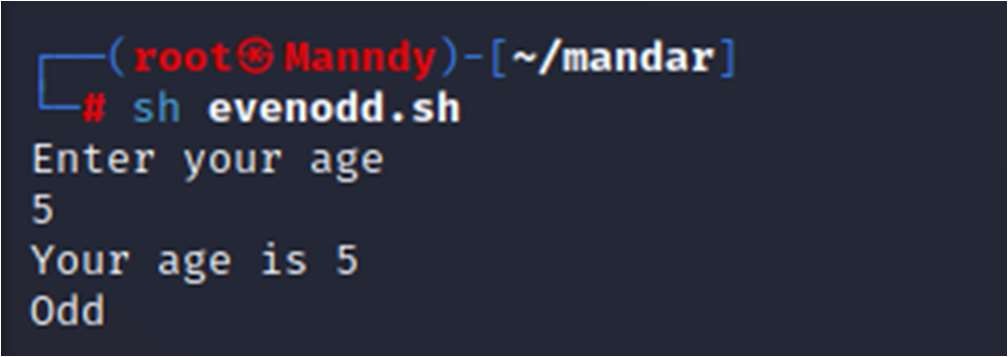
echo "Enter your age" read age

echo "Your age is $age"

if [ `expr $age % 2` -eq 0 ]; then

echo "Even" else

echo "Odd" fi



1. Reverse:

echo "Enter a number:" read num

reverse=0

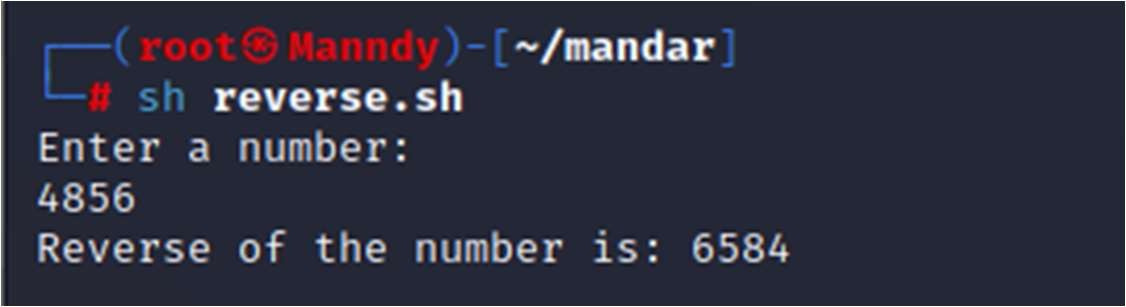
while [ $num -gt 0 ] do

digit=$(( $num % 10 ))

reverse=$(( $reverse \* 10 + $digit )) num=$(( $num/10))

done

echo "Reverse of the number is: $reverse "



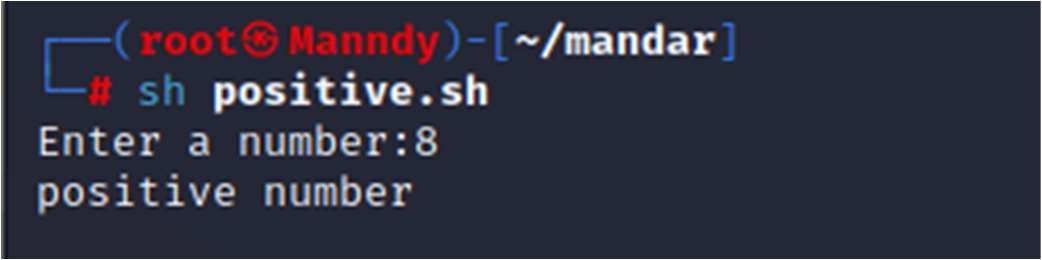
1. Positive negative:

echo -n "Enter a number:" read n

if [ $n -gt 0 ] then

echo "positive number" elif [$n -lt 0]

then

else fi

echo "negative Number"

echo "Neither Positive nor negative"

**Problem Statement**-

# Lab Assignment 3

## Process Synchronization problem: 1.Producer -consumer problem 2.Reader -writer problem 3.Dining -philosopher problem

1. Producer -consumer problem

**Code** –

#include <stdio.h> #include <pthread.h> #include <semaphore.h> #include <stdlib.h> #include <time.h> #include <unistd.h>

sem\_t empty, full; pthread\_mutex\_t mutex; int buffer[5];

int count = 0;

void \*producer(void \*arg)

{

long int num = (long int)arg; sem\_wait(&empty); pthread\_mutex\_lock(&mutex); buffer[count] = rand() % 10;

printf("\n Producer: %ld Produced:%d", num + 1, buffer[count]); count++;

sleep(1); pthread\_mutex\_unlock(&mutex); sem\_post(&full);

}

void \*consumer(void \*arg)

{

long int num = (long int)arg; sem\_wait(&full); pthread\_mutex\_lock(&mutex); count--;

printf("\n Consumer: %ld Consumed:%d", num + 1, buffer[count]); sleep(1);

pthread\_mutex\_unlock(&mutex); sem\_post(&empty);

}

int main()

{

int np, nc;

pthread\_t p[10], c[10]; unsigned long int i, j, k, l;

printf("Mandar Gatke's code:\n");

printf("Enter the number of producers and consumers\n "); scanf("%d %d", &np, &nc);

sem\_init(&empty, 0, 5);

sem\_init(&full, 0, 0); pthread\_mutex\_init(&mutex, NULL); for (i = 0; i < np; i++)

{

pthread\_create(&p[i], NULL, producer, (void \*)i);

}

for (j = 0; j < nc; j++)

{

pthread\_create(&c[j], NULL, consumer, (void \*)j);

}

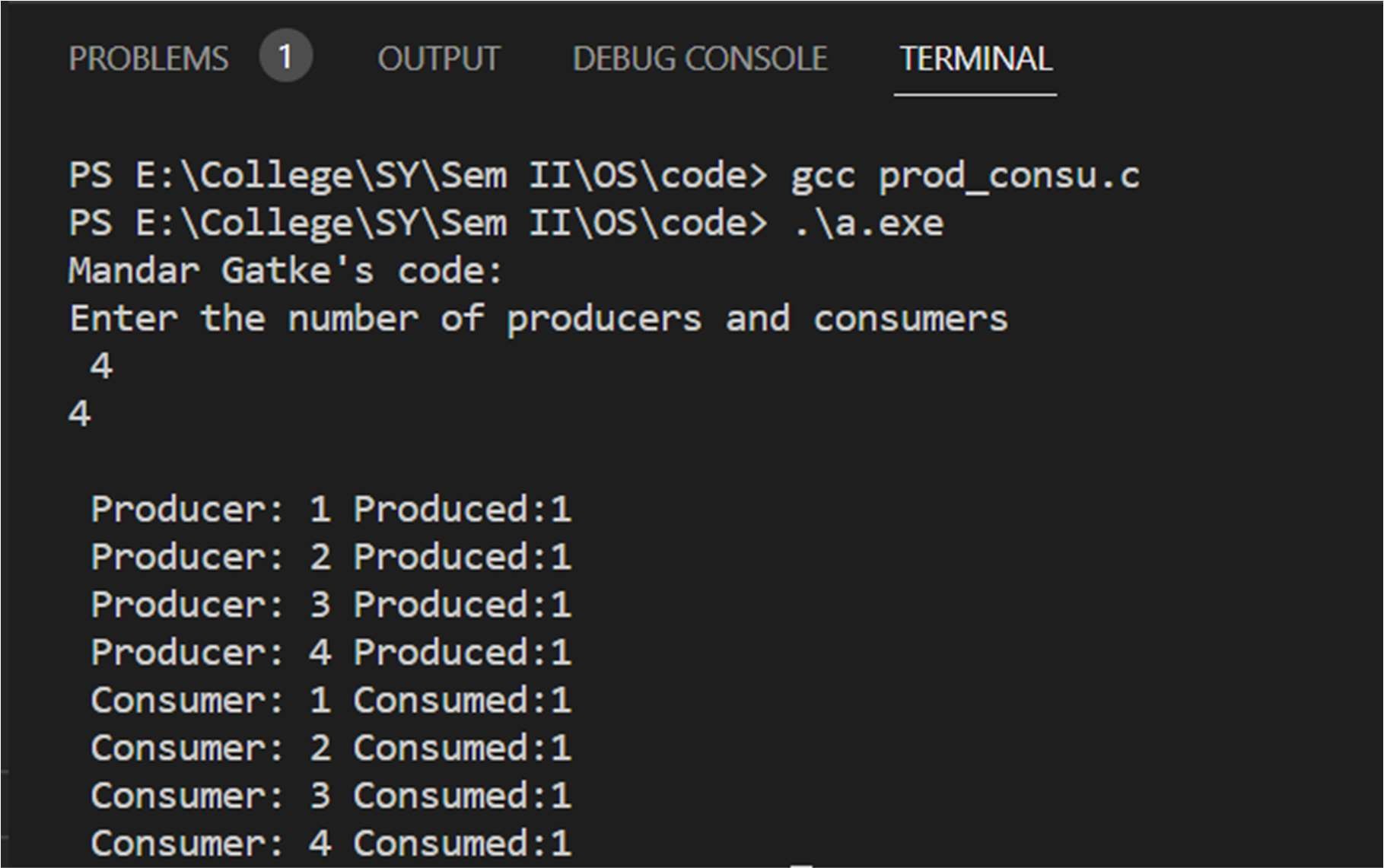
for (k = 0; k < np; k++) pthread\_join(p[k], NULL);

for (l = 0; l < nc; l++) pthread\_join(c[l], NULL);

return 0;

}

**Output –**



## Reader -writer problem

**Code** –

#include<stdio.h> #include<pthread.h> #include<unistd.h> #include<stdlib.h> pthread\_mutex\_t wr,mutex;

int a=10,readcount=0; void \*reader(void \*arg)

{

long int num; num=(long int)arg;

pthread\_mutex\_lock(&mutex); readcount++; pthread\_mutex\_unlock(&mutex); if(readcount==1)

pthread\_mutex\_lock(&wr);

printf("\n Reader %li is in critical section",num); printf("\n Reader %li is reading data %d",num,a); sleep(1);

pthread\_mutex\_lock(&mutex); readcount--; pthread\_mutex\_unlock(&mutex); if(readcount==0)

pthread\_mutex\_unlock(&wr);

printf("\n Reader %li left critical section",num);

}

void \*writer(void \*arg)

{

long int num; num=(long int)arg;

pthread\_mutex\_lock(&wr);

printf("\n Writer %li is in left critical section",num); printf("\n Writer %li have written data as %d ",num,++a); sleep(1);

pthread\_mutex\_unlock(&wr);

printf("\n Writer %li left critical section",num);

}

int main()

{

pthread\_t r[10],w[10]; long int i,j;

int nor,now; pthread\_mutex\_init(&wr,NULL); pthread\_mutex\_init(&mutex,NULL); printf("Mandar Gatke's code:\n");

printf("Enter number of reader and writers\n"); scanf("%d %d",&nor,&now);

for(i=0;i<nor;i++)

{

pthread\_create(&r[i],NULL,reader,(void \*)i);

}

for(j=0;j<now;j++)

{

pthread\_create(&w[j],NULL,writer,(void \*)j);

}

for(i=0;i<nor;i++)

{

pthread\_join(r[i],NULL);

}

for(j=0;j<now;j++)

{

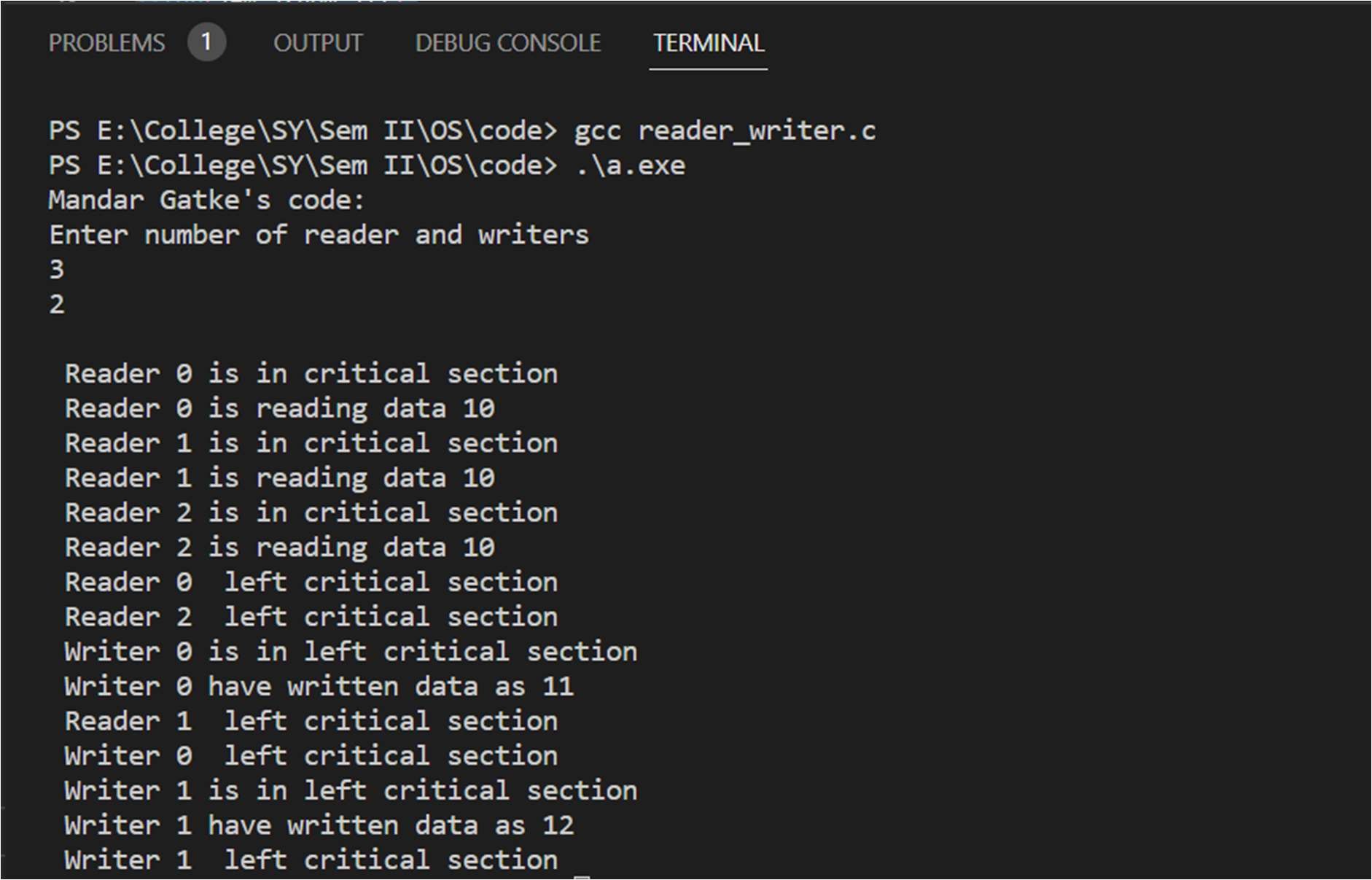
pthread\_join(w[j],NULL);

}

return 0;

}

**Output –**



## Dining -philosopher problem

**Code** –

#include<stdio.h> #include<stdlib.h> #include<pthread.h> #include<semaphore.h> sem\_t chopstick[5]; void \*philos(void \*); void eat(int);

int main()

{

int i,n[5]; pthread\_t T[5];

printf("Mandar Gatke's code:\n"); for(i=0;i<5;i++)

{

sem\_init(&chopstick[i],0,1); for(i=0;i<5;i++)

{

n[i]=i;

pthread\_create(&T[i],NULL,philos,(void \*)&n[i]);

}

for(int i=0;i<5;i++)

{

pthread\_join(T[i],NULL);

}

}

return 0;

}

void \*philos(void \*n)

{

int ph=\*(int \*)n;

printf("Philosopher %d wants to eat\n",ph); printf("Philosopher try to pick left chopstick \n",ph); sem\_wait(&chopstick[ph]);

printf("Philosopher %d pick left chopstick \n",ph); printf("Philosopher try to pick right chopstick \n",ph); sem\_wait(&chopstick[(ph+1)%5]);

printf("Philosopher %d pick right chopstick \n",ph); eat(ph);

sleep(2);

printf("Philosopher %d has finished eating\n",ph); sem\_post(&chopstick[ph]);

printf("Philosopher %d leaves the chopstick\n",ph);

}

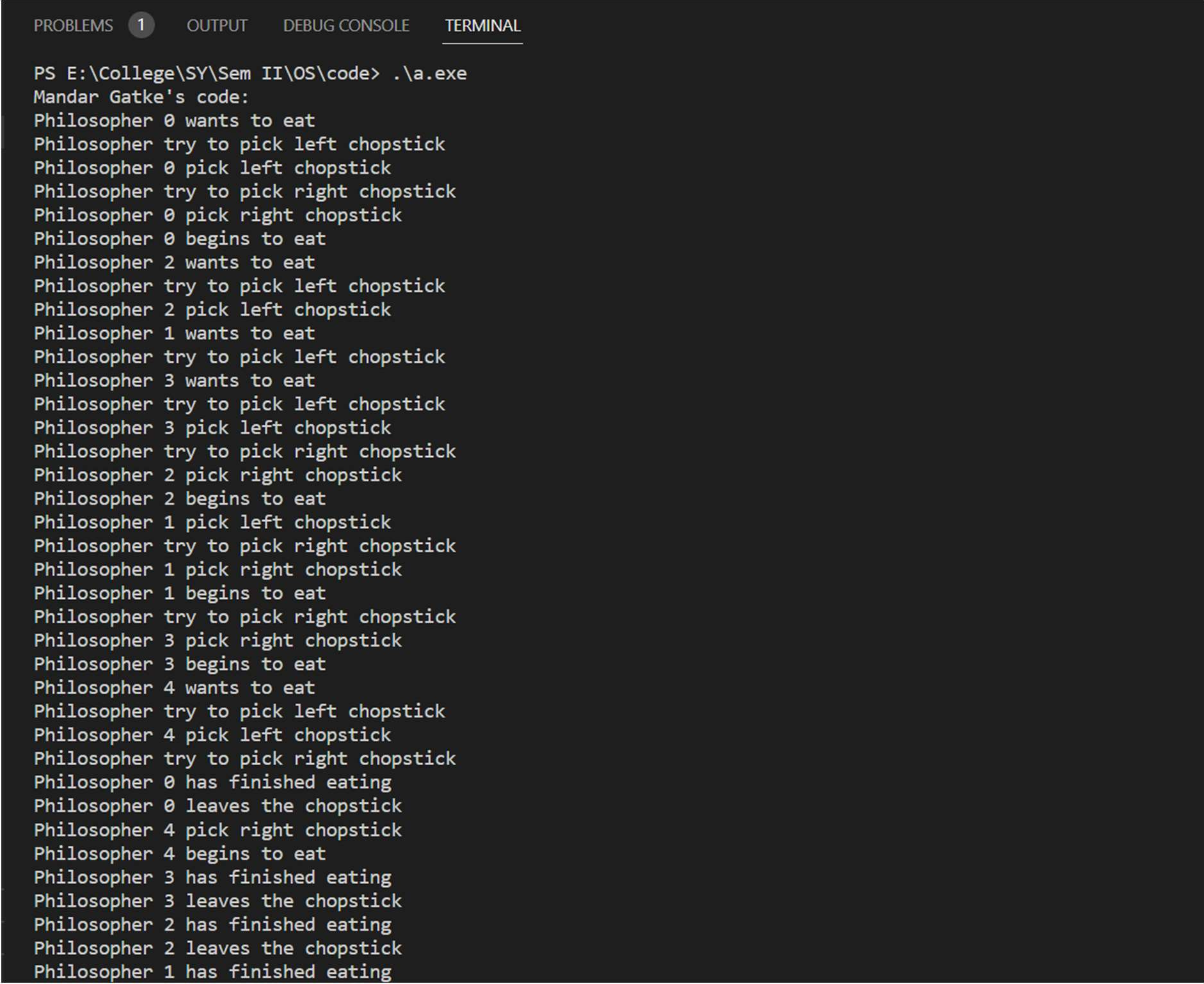
void eat(int ph)

{

printf("Philosopher %d begins to eat \n",ph);

}

**Output –**



**Problem Statement**-

# Lab Assignment 4

CPU scheduling using following – 1]FCFS

1. SJF
2. Priority based 4] Round Robin

1] FCFS:

**Code** –

// Mandar Gatke

#include <stdio.h> int main()

{

int A[100][4];

int i, j, n, total = 0, index, temp; float avg\_wt, avg\_tat; printf("Mandar Gatke's code:\n"); printf("Enter number of process: "); scanf("%d", &n);

printf("Enter Burst Time:\n");

for (i = 0; i < n; i++) { printf("P%d: ", i + 1); scanf("%d", &A[i][1]); A[i][0] = i + 1;

}

A[0][2] = 0;

for (i = 1; i < n; i++) { A[i][2] = 0;

for (j = 0; j < i; j++) A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n; total = 0;

printf("P BT WT TAT\n");

for (i = 0; i < n; i++) { A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0], A[i][1], A[i][2], A[i][3]);

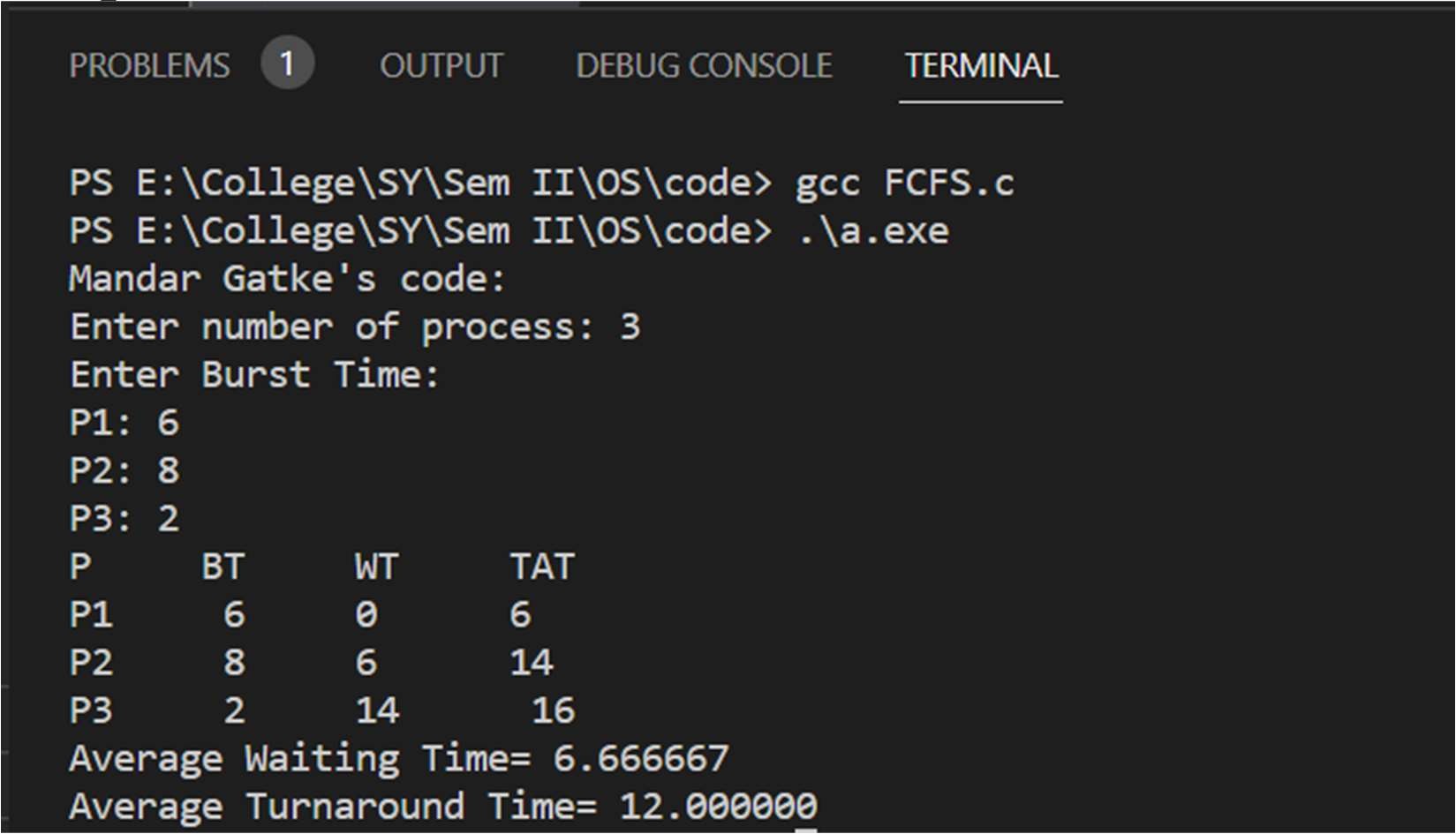
}

avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt); printf("\nAverage Turnaround Time= %f", avg\_tat);

}

**Output –**



1] SJF:

**Code** –

#include <stdio.h> int main()

{

int A[100][4];

int i, j, n, total = 0, index, temp; float avg\_wt, avg\_tat; printf("Mandar Gatke's code:\n"); printf("Enter number of process: "); scanf("%d", &n);

printf("Enter Burst Time:\n");

for (i = 0; i < n; i++) { printf("P%d: ", i + 1); scanf("%d", &A[i][1]); A[i][0] = i + 1;

}

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

for (j = i + 1; j < n; j++) if (A[j][1] < A[index][1])

index = j; temp = A[i][1]; A[i][1] = A[index][1]; A[index][1] = temp;

temp = A[i][0]; A[i][0] = A[index][0]; A[index][0] = temp;

}

A[0][2] = 0;

for (i = 1; i < n; i++) { A[i][2] = 0;

for (j = 0; j < i; j++) A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n; total = 0;

printf("P BT WT TAT\n");

for (i = 0; i < n; i++) { A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0], A[i][1], A[i][2], A[i][3]);

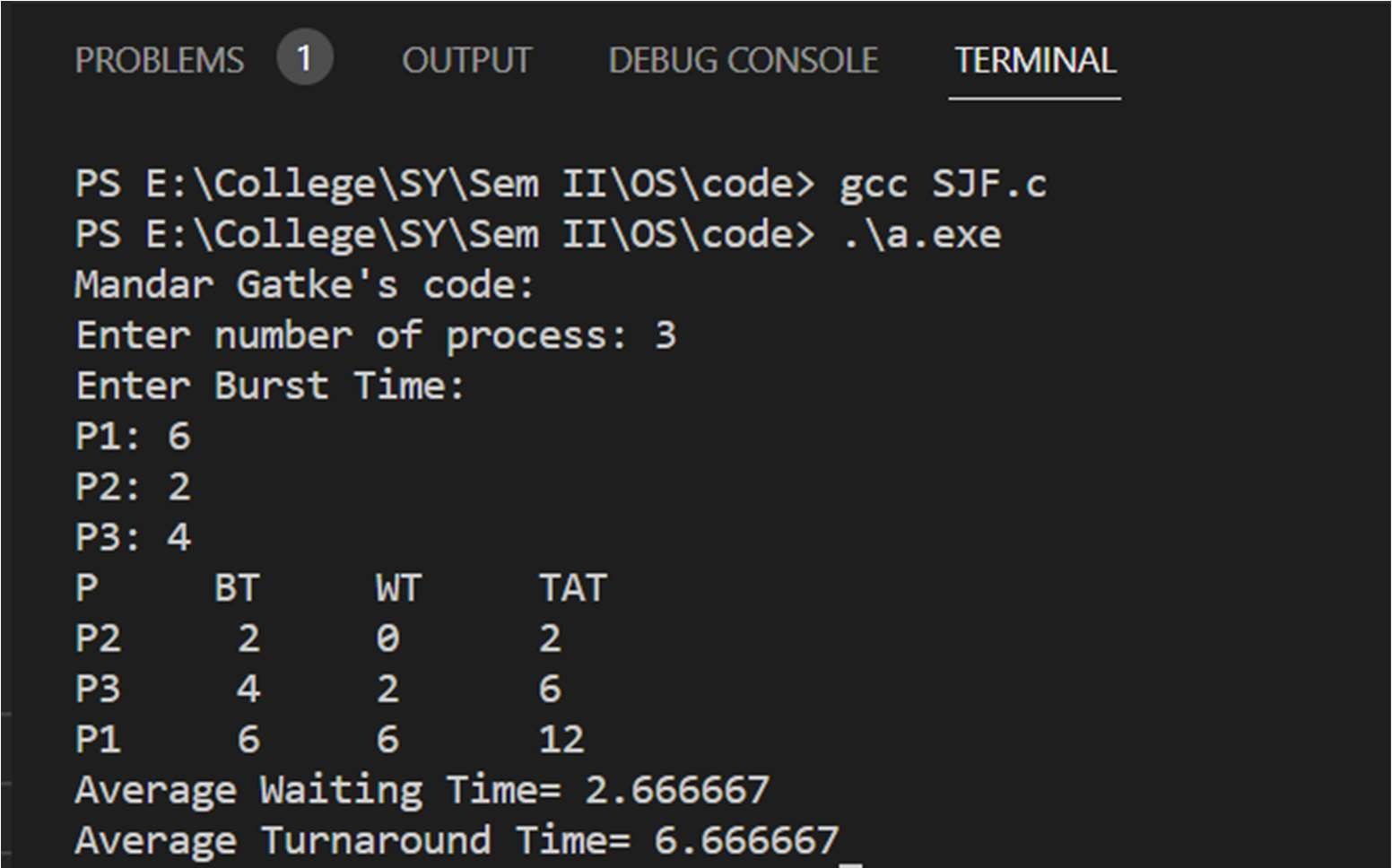
}

avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt); printf("\nAverage Turnaround Time= %f", avg\_tat);

}

**Output –**



1] Priority based:

**Code** –

#include <stdio.h> int main()

{

int A[100][5];

int i, j, n, total = 0, index, temp; float avg\_wt, avg\_tat; printf("Mandar Gatke's code:\n"); printf("Enter number of process: "); scanf("%d", &n);

printf("Enter Burst Time:\n");

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

printf("P%d Burst Time: ", i + 1); scanf("%d", &A[i][1]);

printf("P%d Priority: ", i + 1); scanf("%d", &A[i][5]);

A[i][0] = i + 1;

}

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

for (j = i + 1; j < n; j++) if (A[j][5] < A[index][5])

index = j; temp = A[i][1]; A[i][1] = A[index][1]; A[index][1] = temp;

temp = A[i][5]; A[i][5] = A[index][5]; A[index][5] = temp;

temp = A[i][0]; A[i][0] = A[index][0]; A[index][0] = temp;

}

A[0][2] = 0;

for (i = 1; i < n; i++) { A[i][2] = 0;

for (j = 0; j < i; j++) A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n; total = 0;

printf("P BT WT TAT\n");

for (i = 0; i < n; i++) { A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0], A[i][1], A[i][2], A[i][3]);

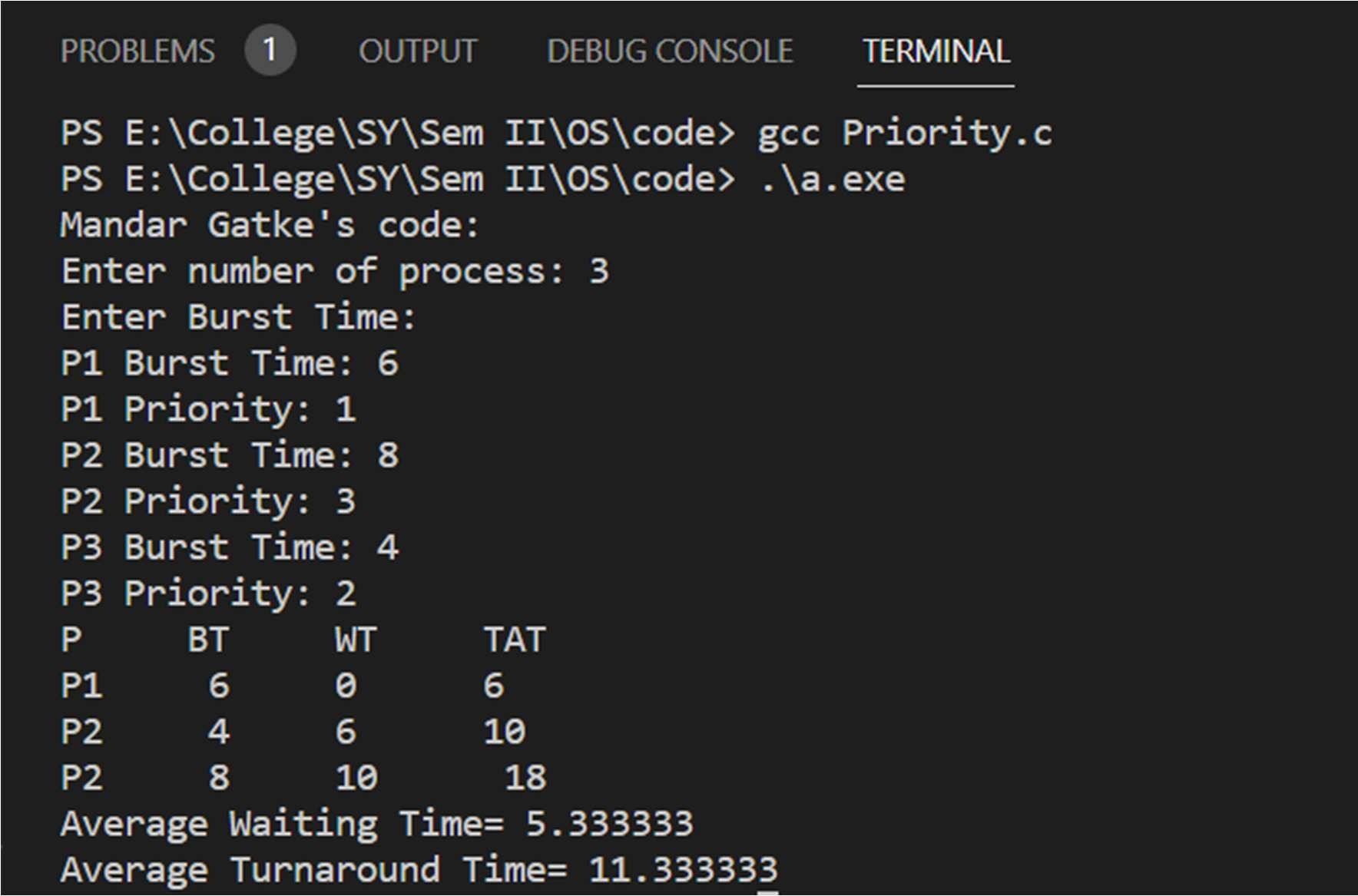
}

avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt); printf("\nAverage Turnaround Time= %f", avg\_tat);

}

**Output –**



1. Round Robbin:

**Code** –

import java.util.Scanner;

public class roundRobbin {

public static void main(String args[]) {

int n, i, qt, count = 0, temp, sq = 0, bt[], wt[], tat[], rem\_bt[]; float awt = 0, atat = 0;

bt = new int[15]; wt = new int[15]; tat = new int[15];

rem\_bt = new int[15];

Scanner sc = new Scanner(System.in); System.out.println("Mandar Gatke's code:");

System.out.print("Enter the number of process (maximum 15) = "); n = sc.nextInt();

System.out.print("Enter the burst time of the process\n"); for (i = 0; i < n; i++) {

System.out.print("P" + i + " = "); bt[i] = sc.nextInt();

rem\_bt[i] = bt[i];

}

System.out.print("Enter the quantum time: "); qt = sc.nextInt();

while (true) {

for (i = 0, count = 0; i < n; i++) { temp = qt;

if (rem\_bt[i] == 0) { count++; continue;

}

if (rem\_bt[i] > qt)

rem\_bt[i] = rem\_bt[i] - qt; else if (rem\_bt[i] >= 0) {

temp = rem\_bt[i];

rem\_bt[i] = 0;

}

sq = sq + temp; tat[i] = sq;

}

if (n == count) break;

}

System.out.print("\nProcess for (i = 0; i < n; i++) {

wt[i] = tat[i] - bt[i]; awt = awt + wt[i];

atat = atat + tat[i];

Burst Time

Turnaround Time

Waiting Time\n");

System.out.print("\n " + (i + 1) + "\t\t " + bt[i] + "\t\t " + tat[i] + "\t\t " +

wt[i] );

}

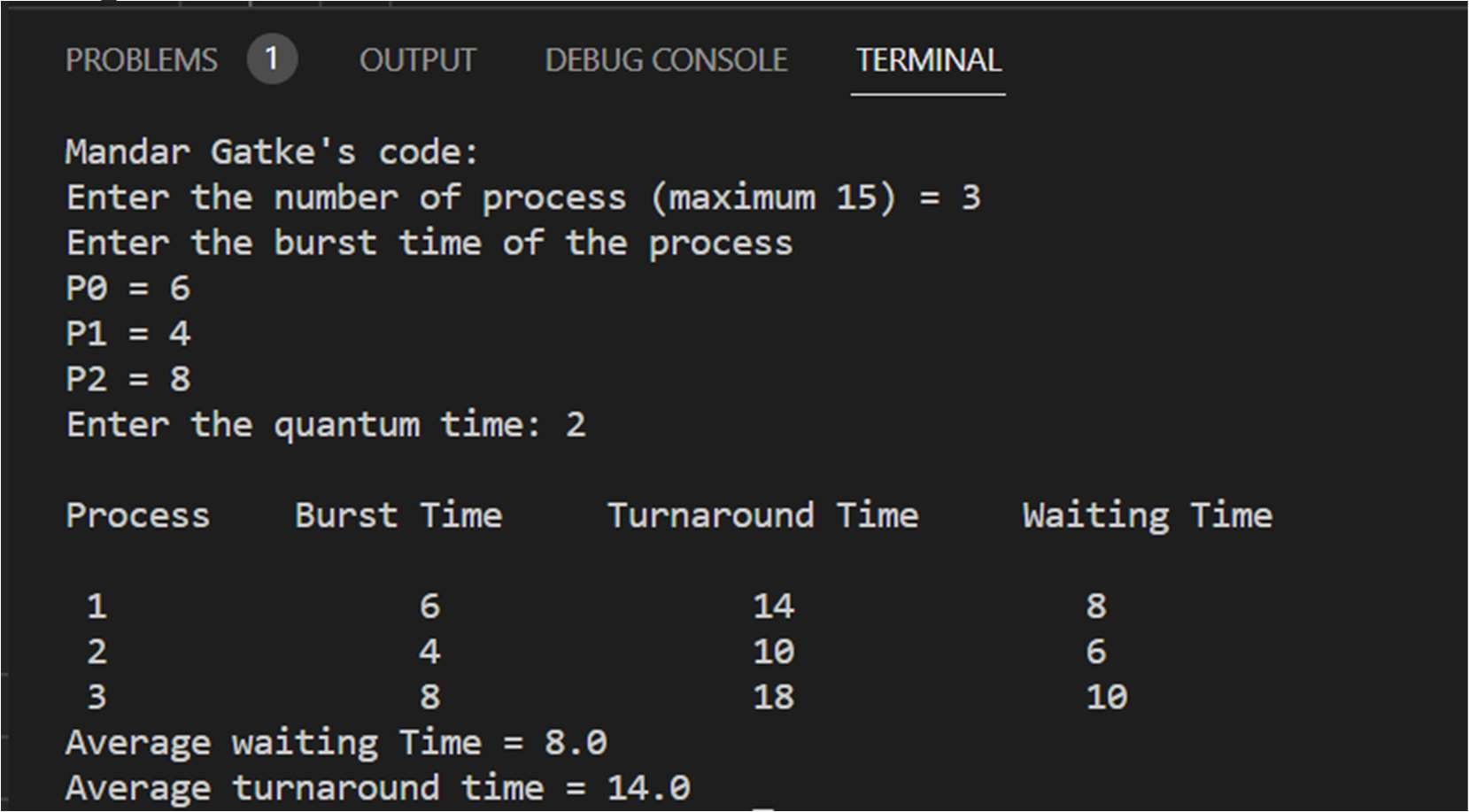
awt = awt / n; atat = atat / n;

System.out.println("\nAverage waiting Time = " + awt ); System.out.println("Average turnaround time = " + atat);

}

}

**Output –**



# Lab Assignment 5

**Problem Statement**- Deadlock detection and safe sequence using banker’s algorithm.

**Code:**

#include <stdio.h> int main()

{

int n, m, i, j, k; n = 5;

m = 3;

int alloc[5][3] = {{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}};

int max[5][3] = {{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}};

int avail[3] = {3, 3, 2};

int f[n], ans[n], ind = 0; for (k = 0; k < n; k++)

{

f[k] = 0;

}

int need[n][m];

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

{

for (j = 0; j < m; j++)

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

for (k = 0; k < 5; k++)

{

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

{

if (f[i] == 0)

{

int flag = 0;

for (j = 0; j < m; j++)

{

if (need[i][j] > avail[j])

{

flag = 1; break;

}

}

if (flag == 0)

{

ans[ind++] = i;

for (y = 0; y < m; y++) avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

int flag = 1;

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

{

if (f[i] == 0)

{

flag = 0;

printf("The given processes can not be arranged in safe

sequence");

break;

}

}

if (flag == 1)

{

printf("Following is the SAFE Sequence for the given processes\n"); for (i = 0; i < n - 1; i++)

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

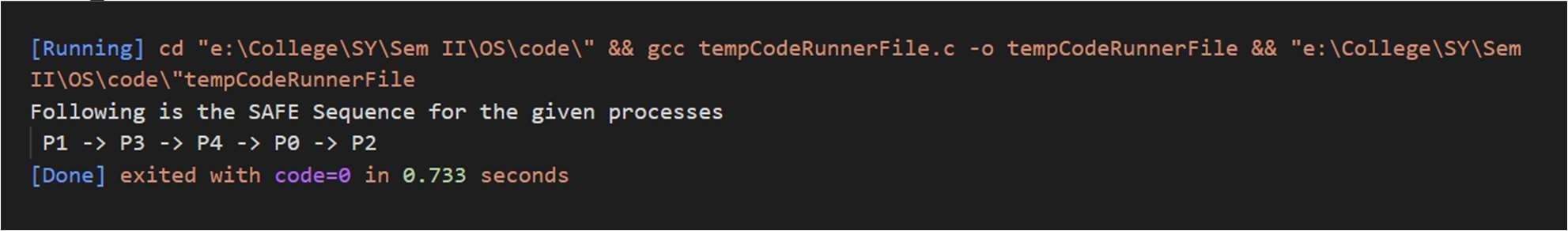
printf(" P%d", ans[n - 1]);

}

return (0);

}

**Output:**



# Lab Assignment 6

**Problem Statement**- Disk scheduling algorithms.

* 1. **FIFO Code:**

#include <stdio.h> int main()

{

int incomingStream[] = {4 , 1 , 2 , 4 , 5}; int pageFaults = 0;

int frames = 3; int m, n, s, pages;

pages = sizeof(incomingStream)/sizeof(incomingStream[0]); printf(" Incoming \t Frame 1 \t Frame 2 \t Frame 3 "); int temp[ frames ];

for(m = 0; m < frames; m++)

{

temp[m] = -1;

}

for(m = 0; m < pages; m++)

{

s = 0;

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

{

if(incomingStream[m] == temp[n])

{ s++;

pageFaults--;

}

}

pageFaults++;

if((pageFaults <= frames) && (s == 0))

{

temp[m] = incomingStream[m];

}

else if(s == 0)

{

temp[(pageFaults - 1) % frames] = incomingStream[m];

}

printf("\n"); printf("%d\t\t\t",incomingStream[m]); for(n = 0; n < frames; n++)

{

if(temp[n] != -1)

printf(" %d\t\t\t", temp[n]);

else

printf(" - \t\t\t");

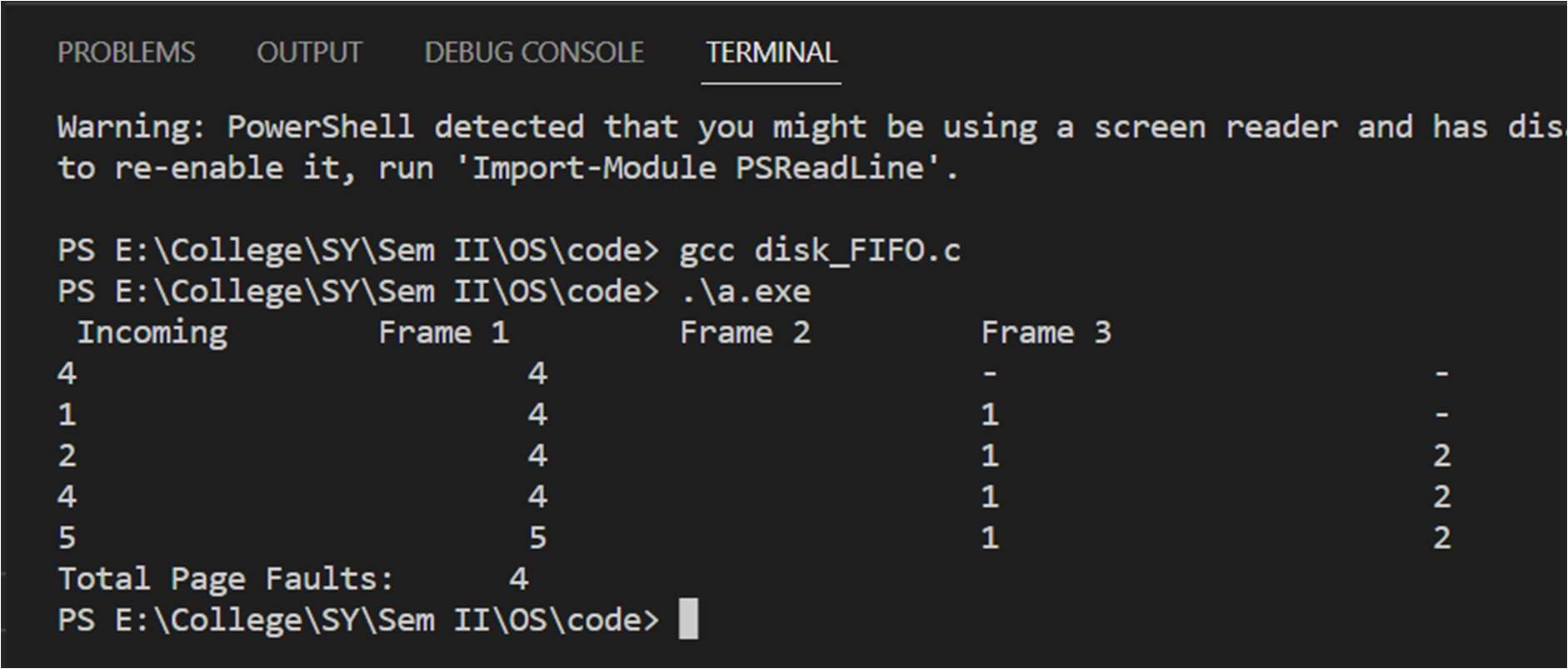
}

}

printf("\nTotal Page Faults:\t%d\n", pageFaults); return 0;

}

**Output:**



* 1. **LRU Code:**

#include<stdio.h> #include<limits.h>

int checkHit(int incomingPage, int queue[], int occupied){ for(int i = 0; i < occupied; i++){

if(incomingPage == queue[i])

return 1;

}

return 0;

}

void printFrame(int queue[], int occupied)

{

for(int i = 0; i < occupied; i++) printf("%d\t\t\t",queue[i]);

}

int main()

{

#### int incomingStream[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};

int n = sizeof(incomingStream)/sizeof(incomingStream[0]); int frames = 3;

#### int queue[n]; int distance[n];

int occupied = 0; int pagefault = 0;

#### printf("Page\t Frame1 \t Frame2 \t Frame3\n");

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

#### {

printf("%d: \t\t",incomingStream[i]);

#### if(checkHit(incomingStream[i], queue, occupied)){ printFrame(queue, occupied);

}

#### else if(occupied < frames){ queue[occupied] = incomingStream[i]; pagefault++;

occupied++;

#### printFrame(queue, occupied);

}

#### else{

int max = INT\_MIN; int index;

#### for (int j = 0; j < frames; j++)

{

#### distance[j] = 0;

for(int k = i - 1; k >= 0; k--)

#### {

++distance[j];

#### if(queue[j] == incomingStream[k]) break;

}

#### if(distance[j] > max){ max = distance[j]; index = j;

}

}

queue[index] = incomingStream[i]; printFrame(queue, occupied); pagefault++;

}

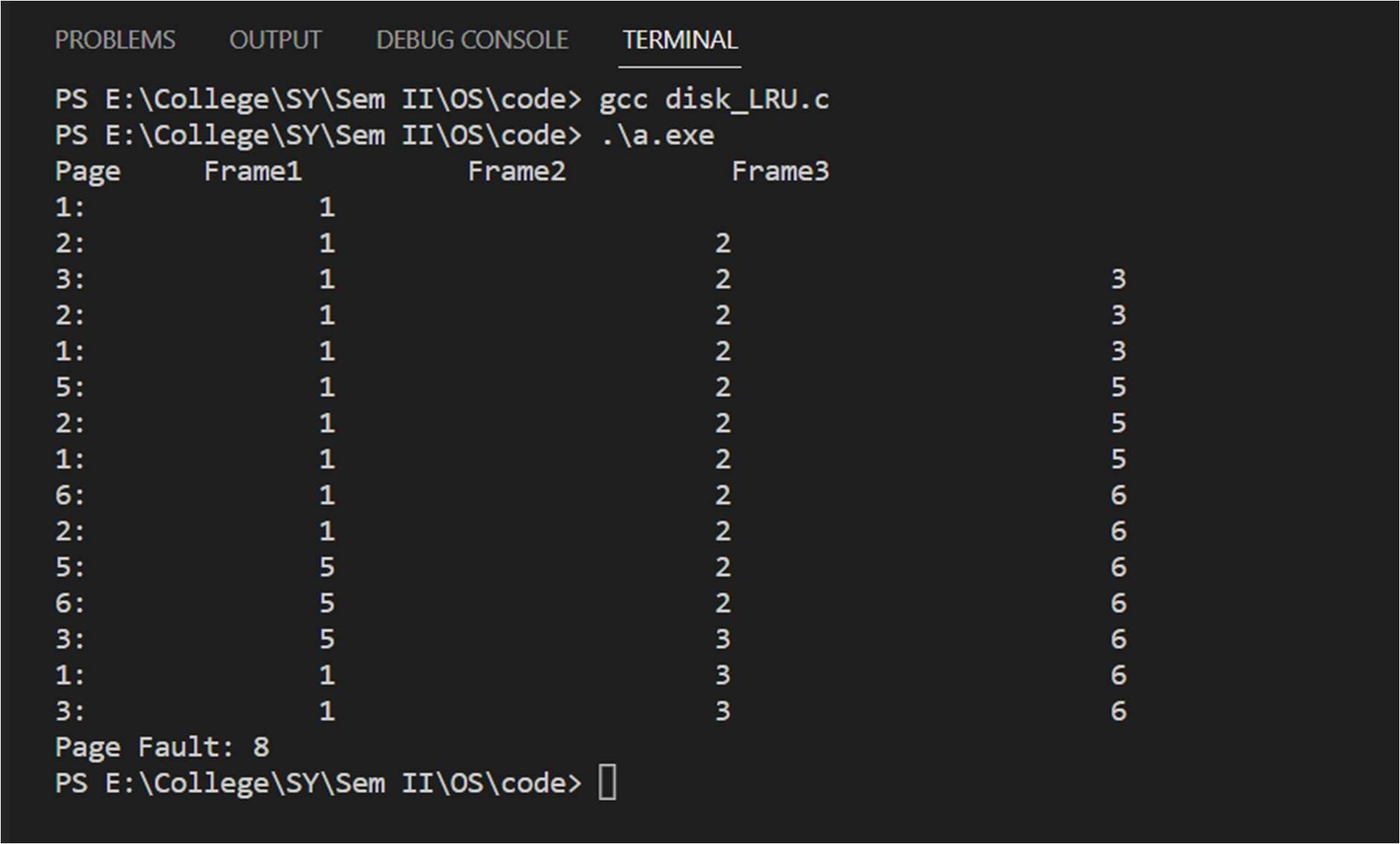
printf("\n");

}

printf("Page Fault: %d",pagefault); return 0;

}

**Output:**



* 1. **Optimal Page replacement algorithm: Code:**

#include <stdio.h> #include <stdbool.h>

bool search(int key, int fr[], int frSize) { for (int i = 0; i < frSize; i++) {

if (fr[i] == key) {

return true;

### }

}

### return false;

}

### int predict(int pg[], int pgSize, int fr[], int frSize, int index) {

int res = -1, farthest = index; for (int i = 0; i < frSize; i++) {

### int j;

for (j = index; j < pgSize; j++) { if (fr[i] == pg[j]) {

### if (j > farthest) { farthest = j; res = i;

}

### break;

}

### }

if (j == pgSize) { return i;

### }

}

### return (res == -1) ? 0 : res;

}

### void optimalPage(int pg[], int pn, int fn) { int fr[fn];

int hit = 0;

### for (int i = 0; i < fn; i++) { fr[i] = -1;

}

### for (int i = 0; i < pn; i++) {

// Page found in a frame: HIT if (search(pg[i], fr, fn)) {

### hit++; continue;

}

### // Page not found in a frame: MISS

// If there is space available in frames. if (fn > i) {

fr[i] = pg[i];

} else {

int j = predict(pg, pn, fr, fn, i + 1); fr[j] = pg[i];

}

}

printf("No. of hits = %d\n", hit); printf("No. of misses = %d\n", pn - hit);

}

int main() {

int pg[] = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };

int pn = sizeof(pg) / sizeof(pg[0]); int fn = 4;

optimalPage(pg, pn, fn); return 0;

}

**Output:**

