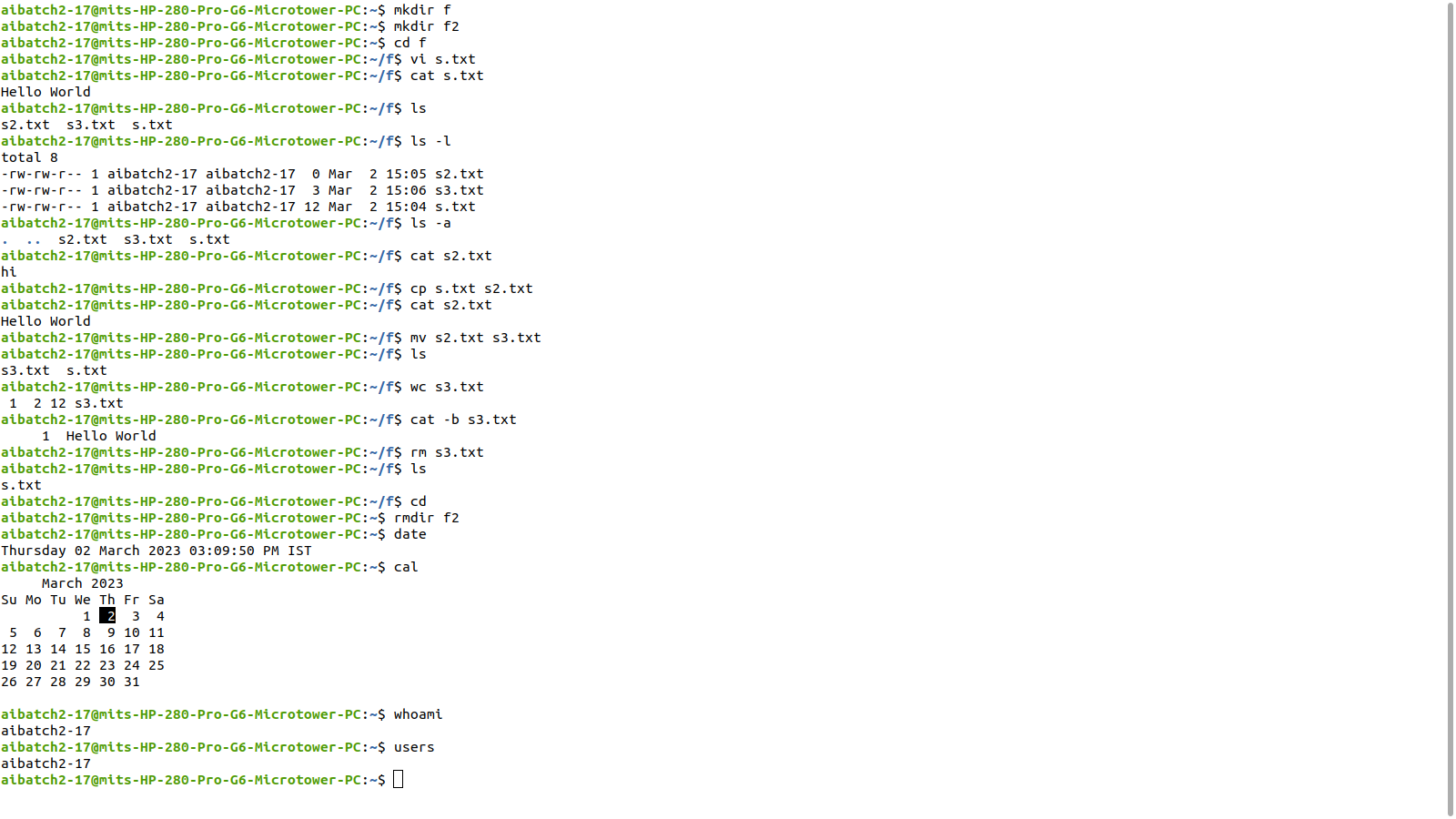
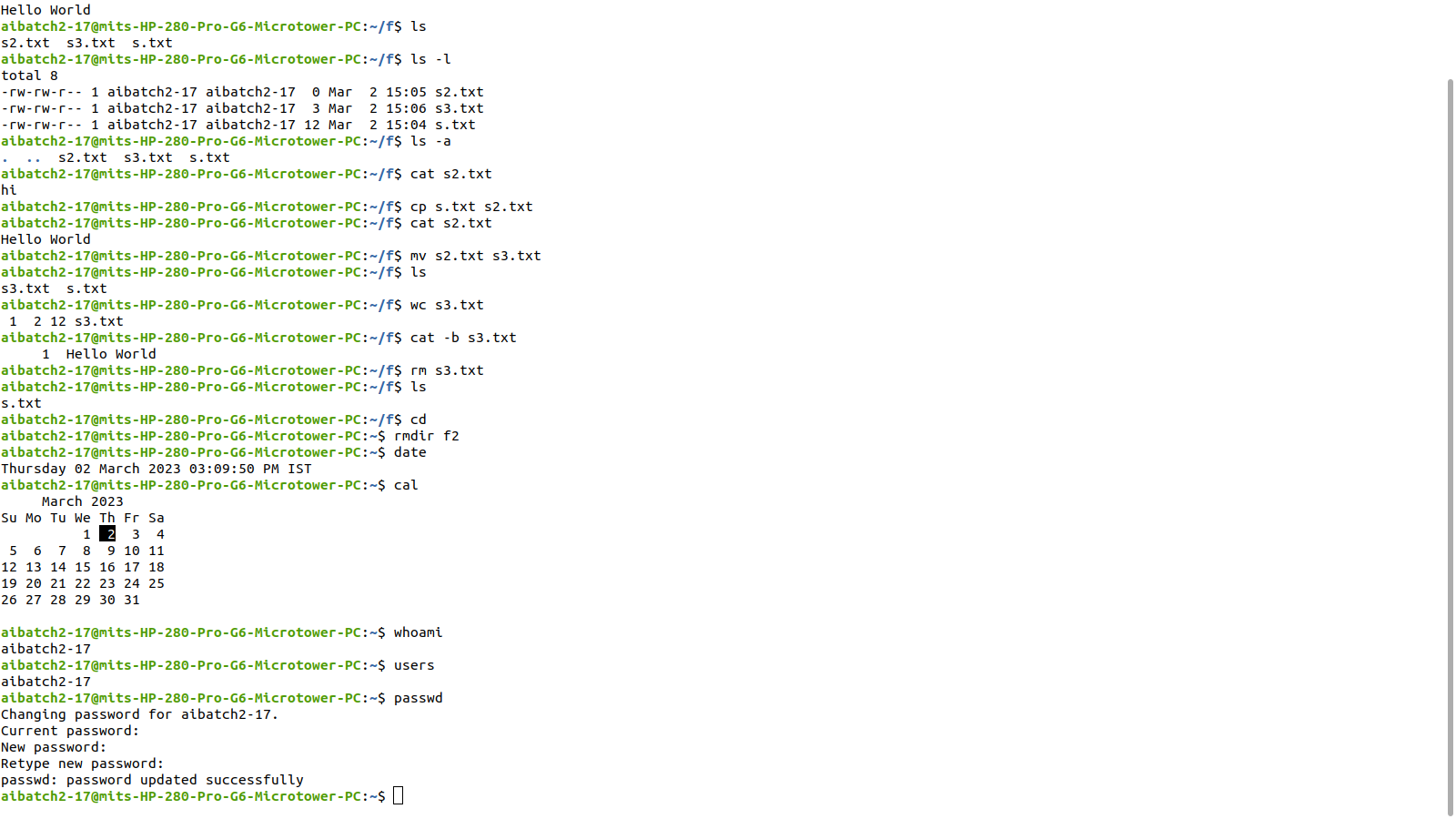
**EXP 1: BASIC LINUX COMMANDS  
**

**EXP 2: SHELL PROGRAMMING**

**PROGRAM**

#!/bin/sh

echo "Enter a number :"

read num

b=`expr $num % 2`

if [ $b -eq 0 ]

then

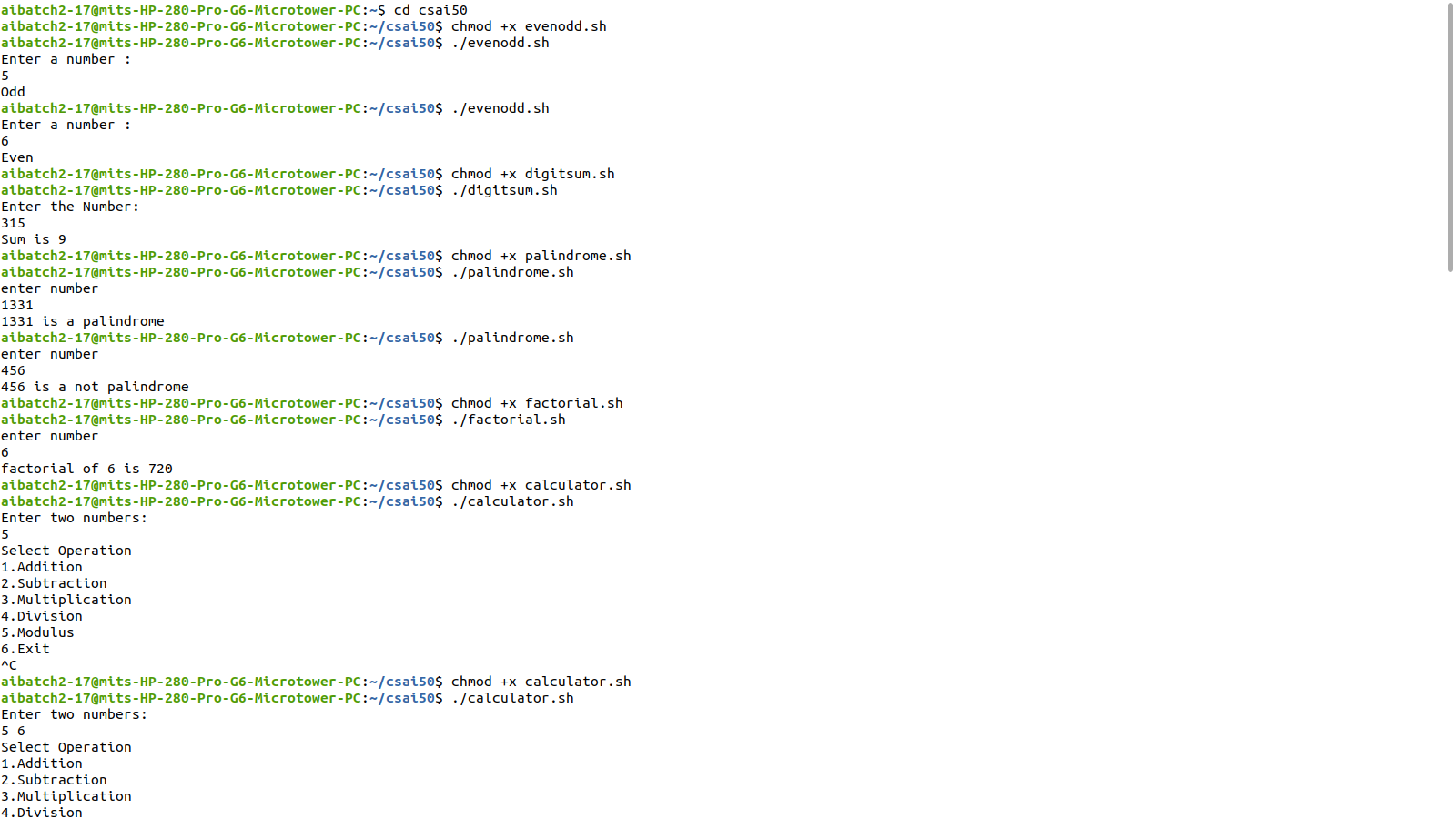
echo "Even"

else

echo "Odd"

fi

**OUTPUT**



**PROGRAM**

#!/bin/sh

echo "Enter the Number:"

read num

sum=0

while [ $num -ne 0 ]

do

digit=`expr $num % 10`

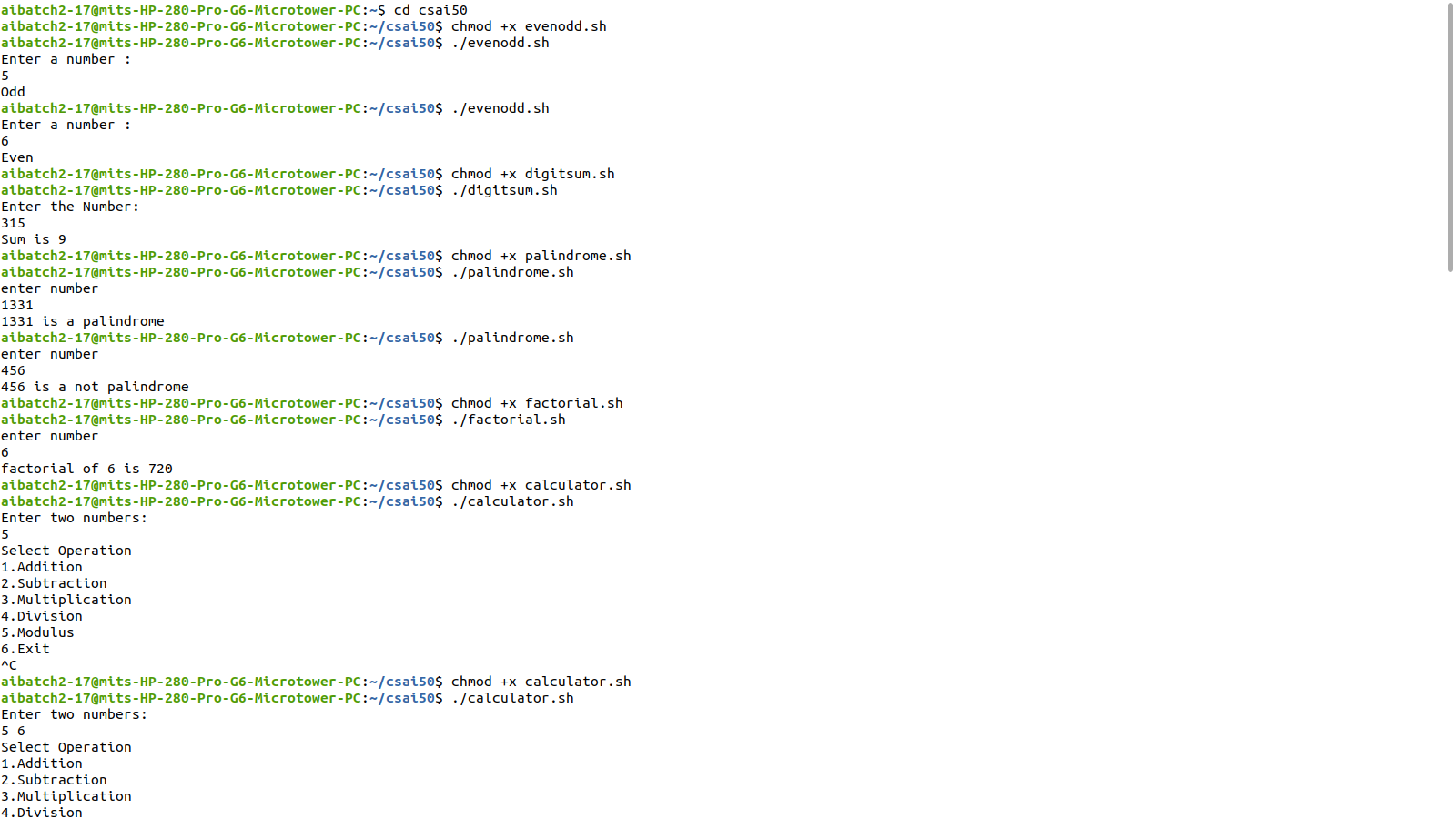
num=`expr $num / 10`

sum=`expr $sum + $digit`

done

echo "Sum is $sum"

**OUTPUT**



**PROGRAM**

#!/bin/sh

echo "Enter a number:"

read n

p=$n

q=0

rev=0

while [ $p -gt 0 ]

do

q=`expr $p % 10`

p=`expr $p / 10`

rev=`expr $rev \\* 10 + $q`

done

if [ $rev -eq $n ]

then

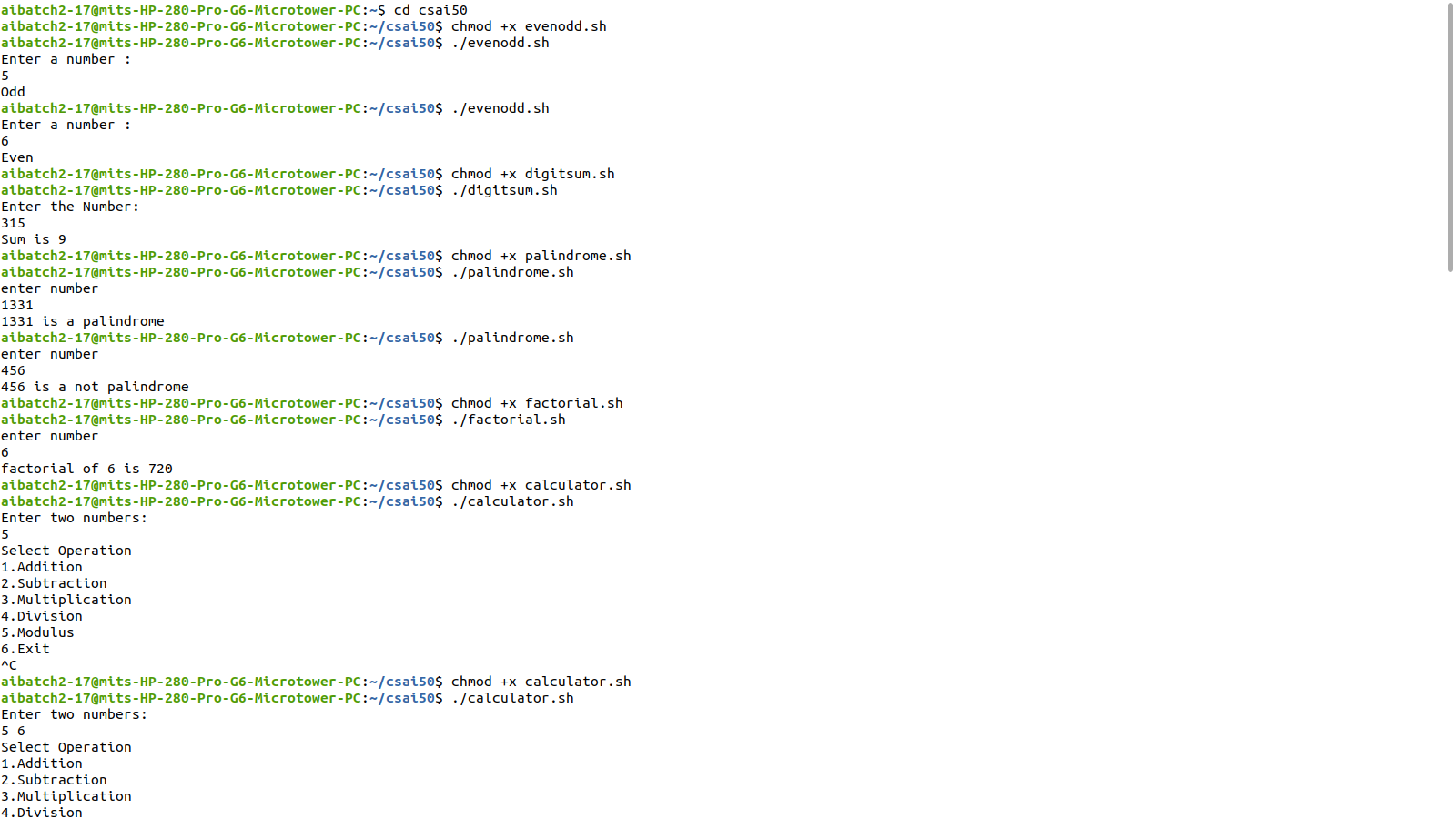
echo "$n is a Palindrome"

else

echo "$n is a not Palindrome"

fi

**OUTPUT**



**PROGRAM**

#!/bin/sh

echo "Enter a Number:"

read n

f=1

i=1

until [ $i -ge `expr $n + 1` ]

do

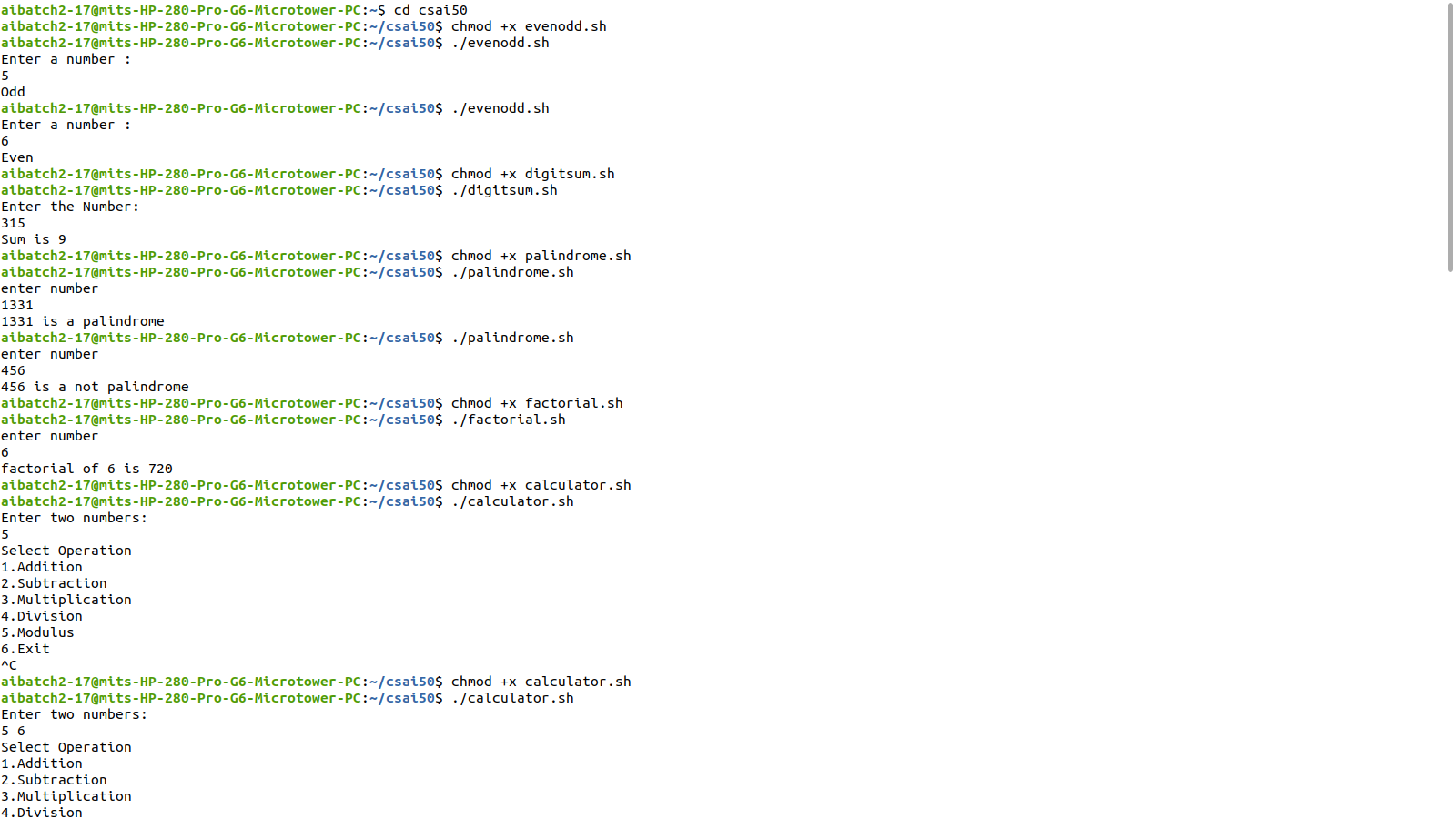
f=`expr $f \\* $i`

i=`expr $i + 1`

done

echo "Factorial of $n is $f"

**OUTPUT**



**PROGRAM**

#!/bin/sh

while [ 1 -eq 1 ]

do

echo "Enter two numbers:"

read a b

echo "Select Operation \n1.Addition"

echo "2.Subtraction\n3.Multiplication"

echo "4.Division\n5.Modulus\n6.Exit"

read c

case $c in

1)echo "$a+$b = `expr $a + $b`";;

2)echo "$a-$b = `expr $a - $b`";;

3)echo "$a\*$b = `expr $a \\* $b`";;

4)echo "$a/$b = `expr $a / $b`";;

5)echo "$a%$b = `expr $a % $b`";;

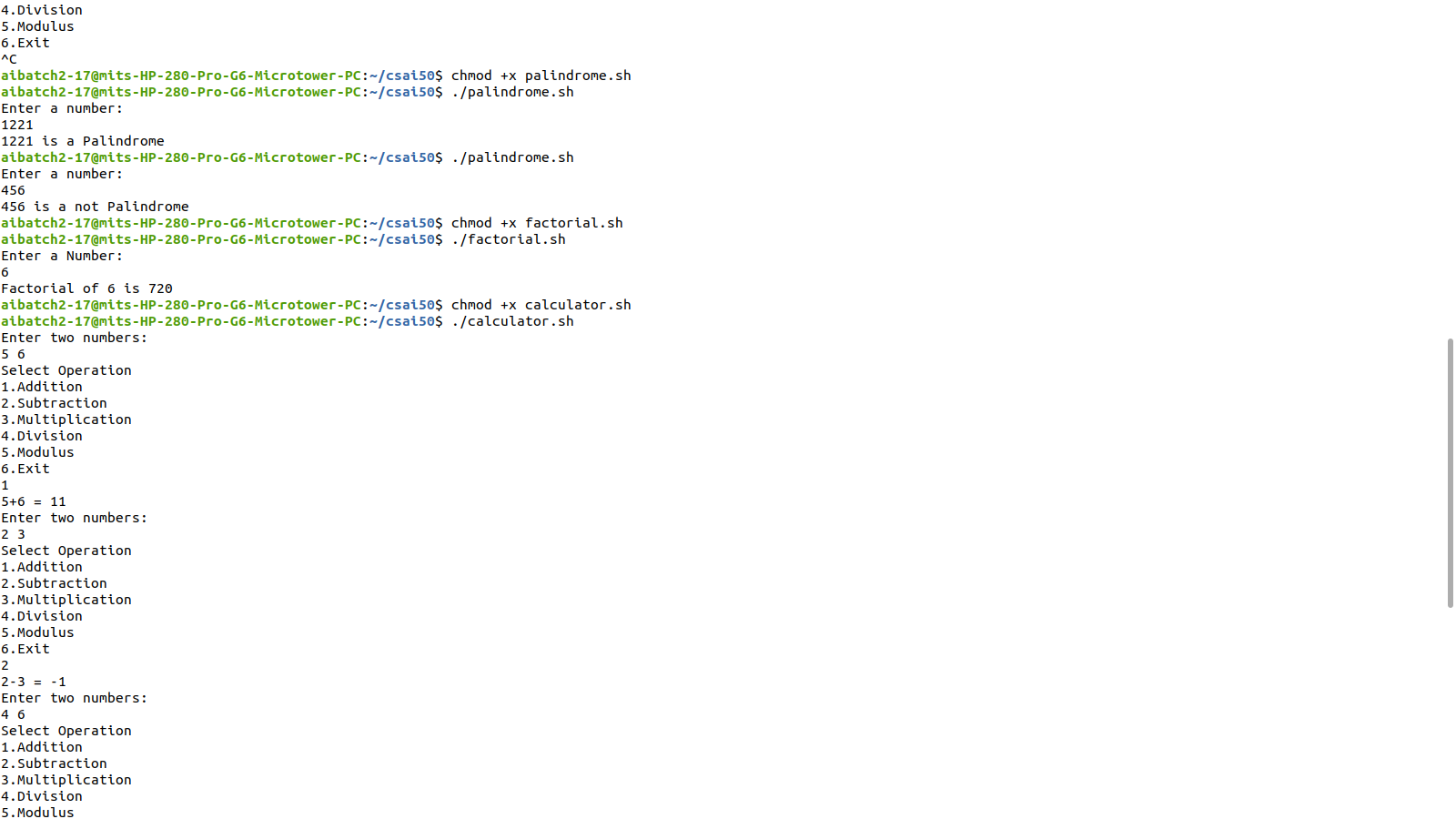
6)exit;;

\*)echo "Invalid Input"

esac

done

**OUTPUT**





**PROGRAM**

#!/bin/sh

while [ 1 -eq 1 ]

do

echo "Choose Option

echo "1.Current Working Directory"

echo "2.Today's Date"

echo "3.List of Users\n4.exit"

read n

case $n in

1)pwd;;

2)date;;

3)who;;

4)exit;;

\*)echo "Invalid Input"

esac

done

**OUTPUT**

****

**EXP 3: SYSTEM CALLS OF LINUX OS**

**PROGRAM**

#include<sys/types.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

void main() {oo

int status,pid,child\_pid;

pid=fork();

if(pid==-1) {

printf("Child Process Creation Failed!");

exit(0);

}

else if(pid==0) {

printf("Inside Child Process with process ID : %d \n",getpid());

execlp("/bin/date",”date”,NULL);

exit(0) ;

}

else {

child\_pid=wait(&status);

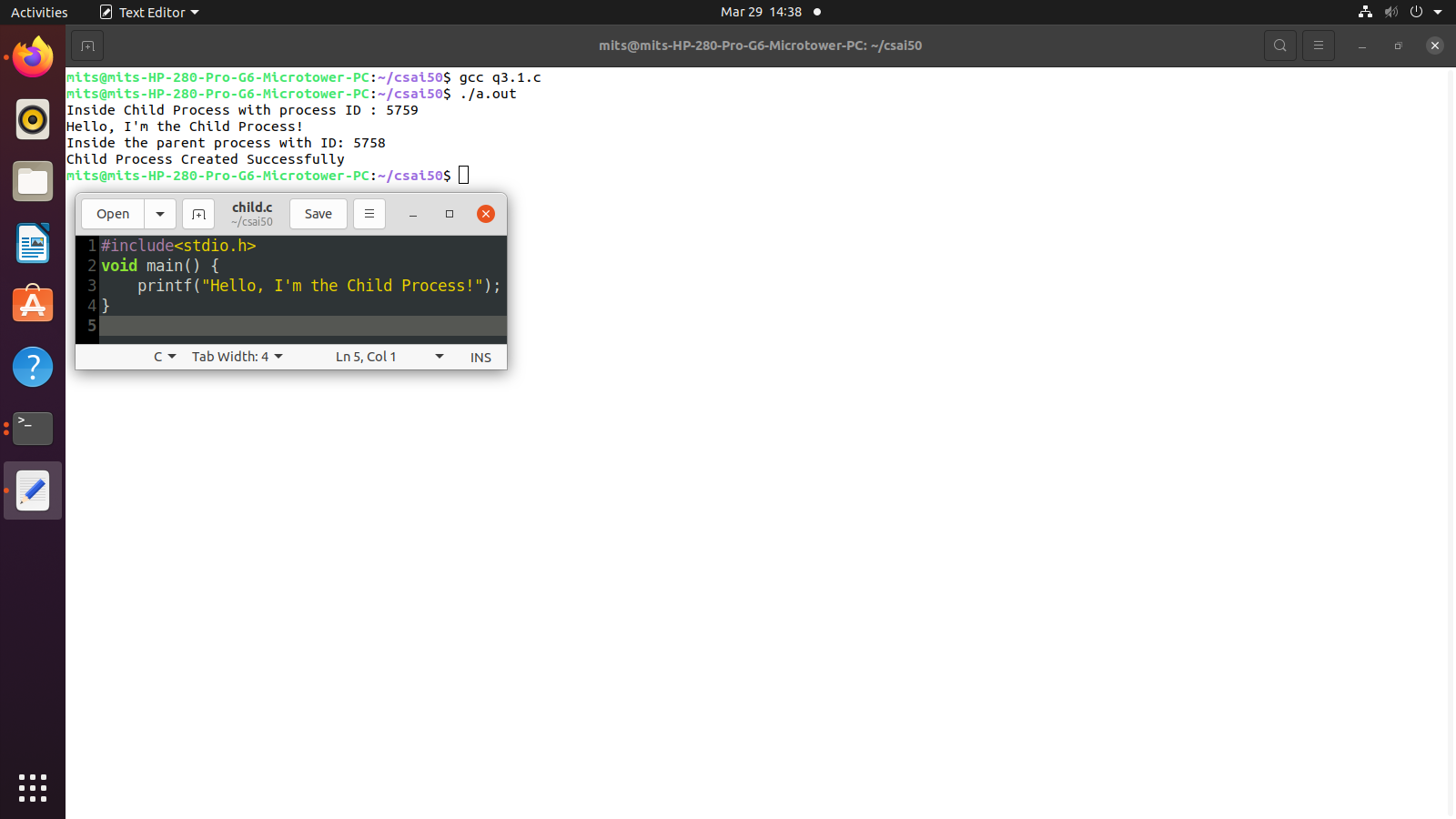
printf("\nInside the parent process with ID: %d \n",getpid());

printf("Child Process Created Successfully\n");

}

}

**OUTPUT**



**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

#include<sys/stat.h>

#include<time.h>

int main() {

char file[10];

struct stat \*node;

node= (struct stat\*) malloc (sizeof(struct stat));

printf("\nEnter Filename: ");

scanf("%s",file);

stat(file,node);

if(node->st\_ino==0)

printf("\nSuch a file does not exist!!");

else {

printf("\nInode/Serial Number: %ld",node->st\_ino);

printf("\nBlock Size: %ld",node->st\_blksize);

printf("\nAccess Time: %ld",node->st\_atime);

printf("\nLast modified time: %ld",node->st\_mtime);

printf("\nGroup ID: %d",node->st\_gid);

printf("\nSize of File: %ld",node->st\_size);

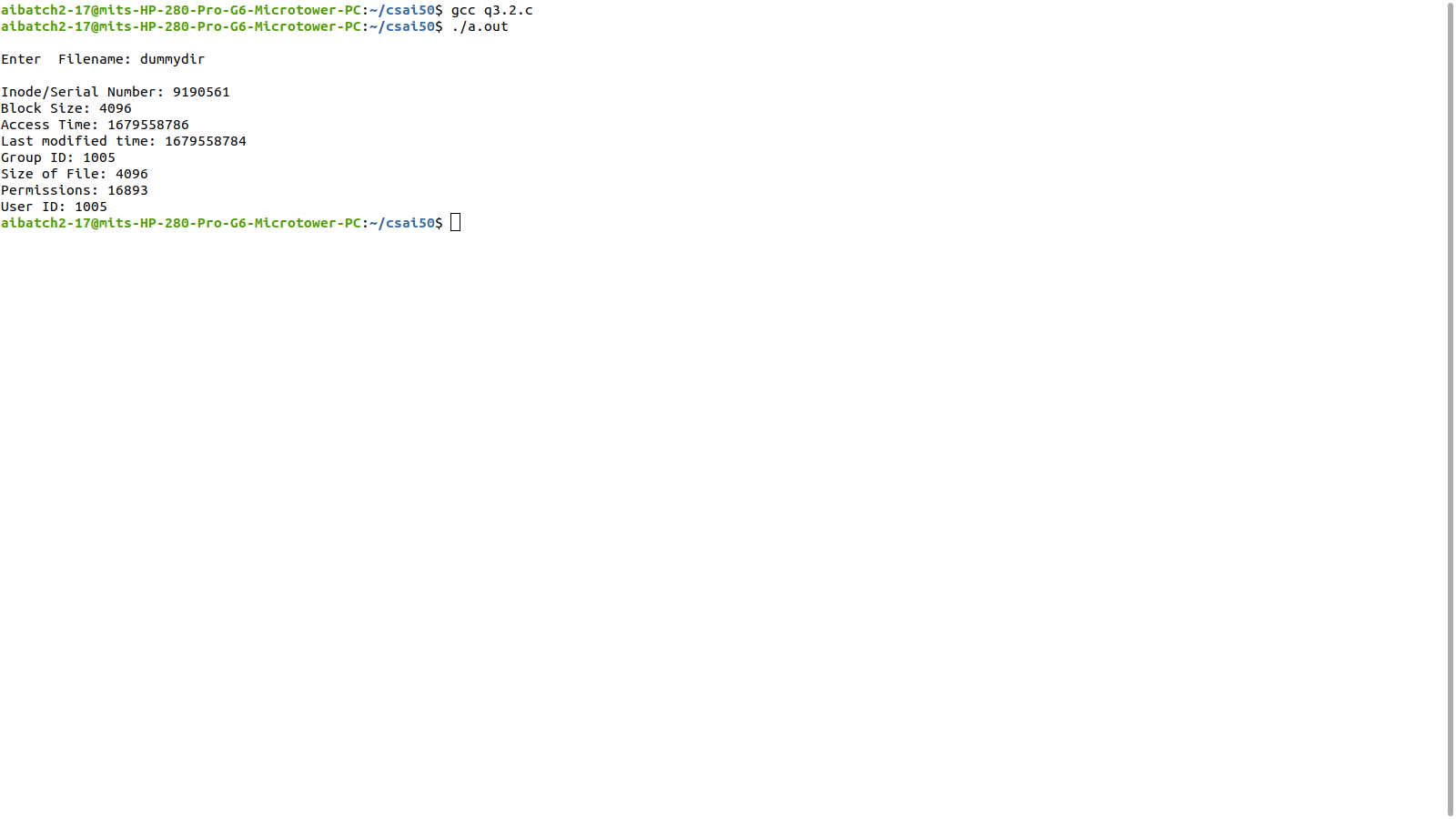
printf("\nPermissions: %d",node->st\_mode);

printf("\nUser ID: %d\n",node->st\_uid);

}

}

**OUTPUT**



**PROGRAM**

#include<sys/types.h>

#include<stdio.h>

#include<dirent.h>

void main() {

DIR \*dir;

struct dirent \*ptr2;

char dir\_name[50];

printf("\nEnter the Directory: ");

scanf("%s",dir\_name);

dir=opendir(dir\_name);

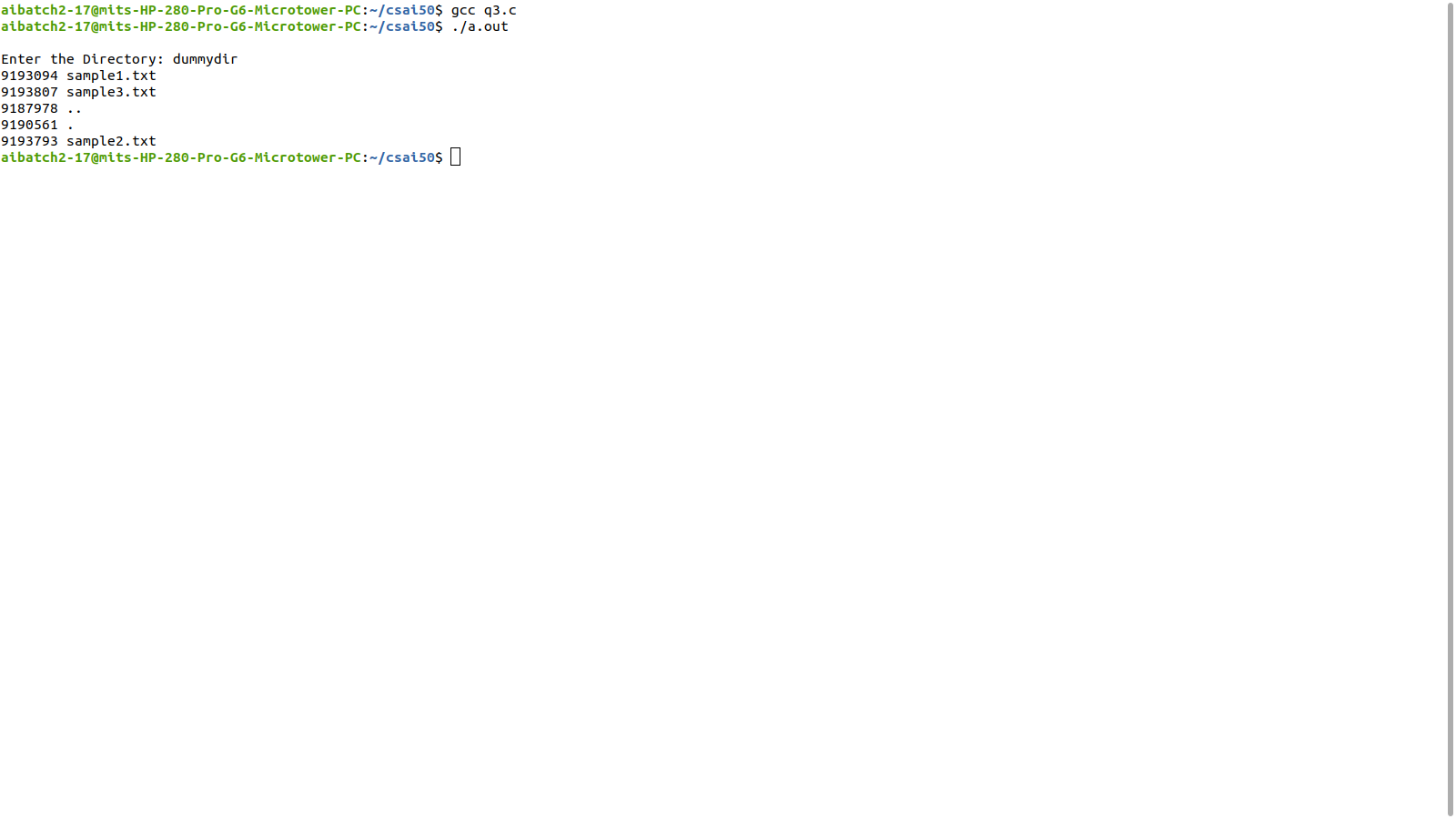
while((ptr2=readdir(dir))!=NULL)

printf("%ld\t%s\n",ptr2->d\_ino,ptr2->d\_name);

closedir(dir);

}

**OUTPUT**



**EXP 4: I/O SYSTEM CALLS**

**PROGRAM**

#include<sys/stat.h>

#include<sys/types.h>

#include<stdio.h>

#include<unistd.h>

#include<string.h>

#include<fcntl.h>

int main() {

int fd,fd2;

char wbuf[128];

char rbuf[128];

fd=open("file.txt",O\_RDWR);

printf("Enter the text to be written:\n");

scanf("%s",wbuf);

write(fd,wbuf,strlen(wbuf));

close(fd);

fd2=open("file.txt",O\_RDWR);

printf("\nThe Contents are:\n");

read(fd2,rbuf,100);

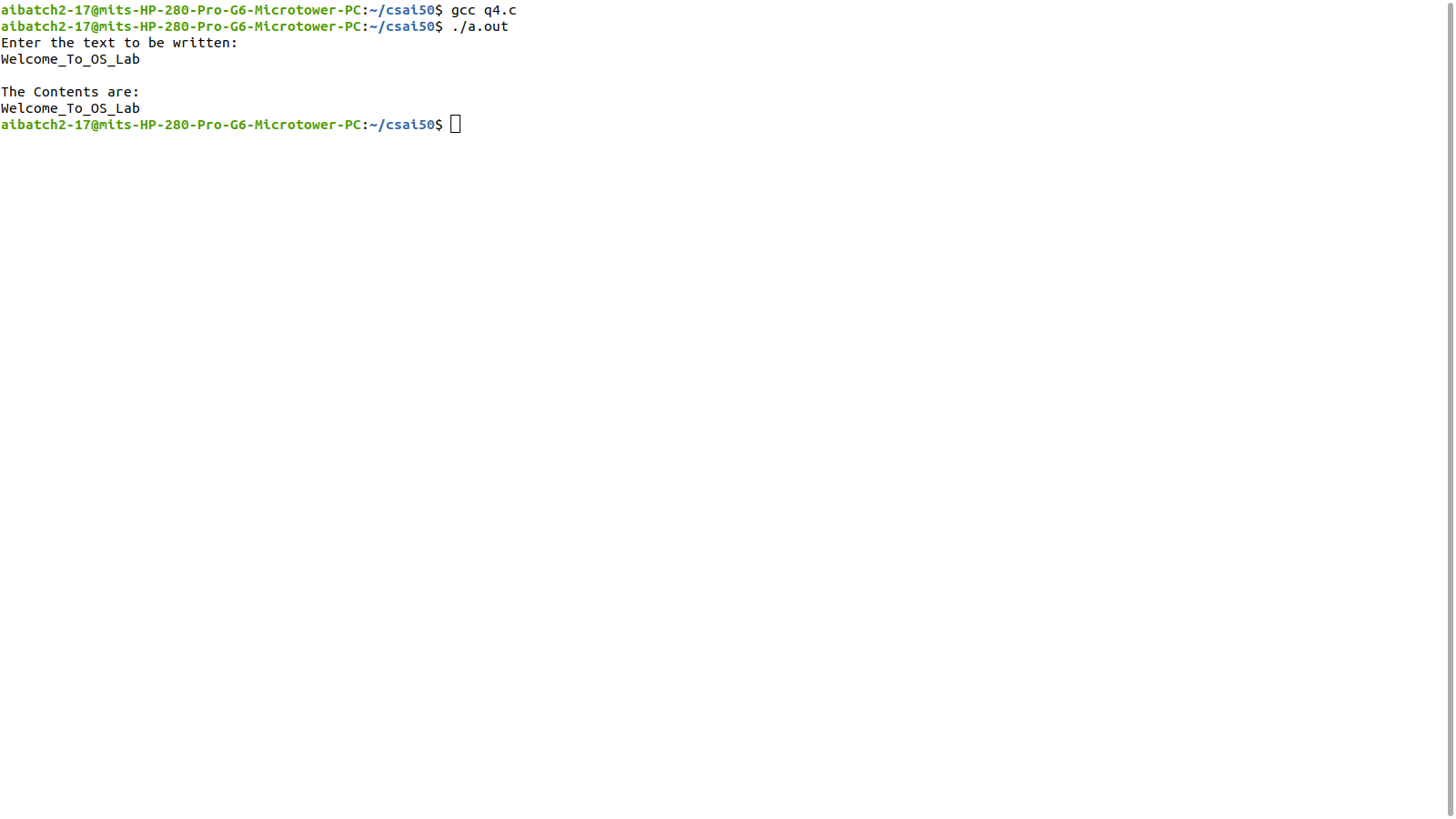
printf("%s\n",rbuf);

close(fd2);

return 0;

}

**OUTPUT**



**EXP 5: NON-PREEMPTIVE SCHEDULING ALGORITHMS**

**PROGRAM**

//SJF Scheduling

#include<stdio.h>

void main() {

int p[20],wt[20],bt[20],tt[20],n,temp;

float wt\_avg=0,tt\_avg=0;

printf("Enter the number of Processes : ");

scanf("%d",&n);

printf("\nEnter the Burst Time of Each Process :\n");

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

p[i]=i+1;

printf("P%d : ",i+1);

scanf("%d",&bt[i]);

}

for(int i=1;i<n;i++)

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

if(bt[j]>bt[j+1]) {

temp=p[j];

p[j]=p[j+1];

p[j+1]=temp;

temp=bt[j];

bt[j]=bt[j+1];

bt[j+1]=temp;

}

wt[0]=0;

for(int i=1;i<n;i++) {

wt[i]=bt[i-1]+wt[i-1];

wt\_avg+=wt[i];

}

wt\_avg/=n;

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

tt[i]=wt[i]+bt[i];

tt\_avg+=tt[i];

}

tt\_avg/=n;

printf("\nProcess\t\t Burst Time\t\t Waiting Time\t\t Turnaround Time\n");

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

printf("P%d\t\t\t%d\t\t\t%d\t\t\t%d\n",p[i],bt[i],wt[i],tt[i]);

printf("\nAverage Waiting Time : %.2f",wt\_avg);

printf("\nAverage Turnaround Time : %.2f\n",tt\_avg);

printf("\nGantt Chart\n\n");

printf("---------------------------------------------------------------------------------------

---------------------\n");

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

printf("|\tP%d\t|",p[i]);

printf("\n-------------------------------------------------------------------------------------

-----------------------\n");

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

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

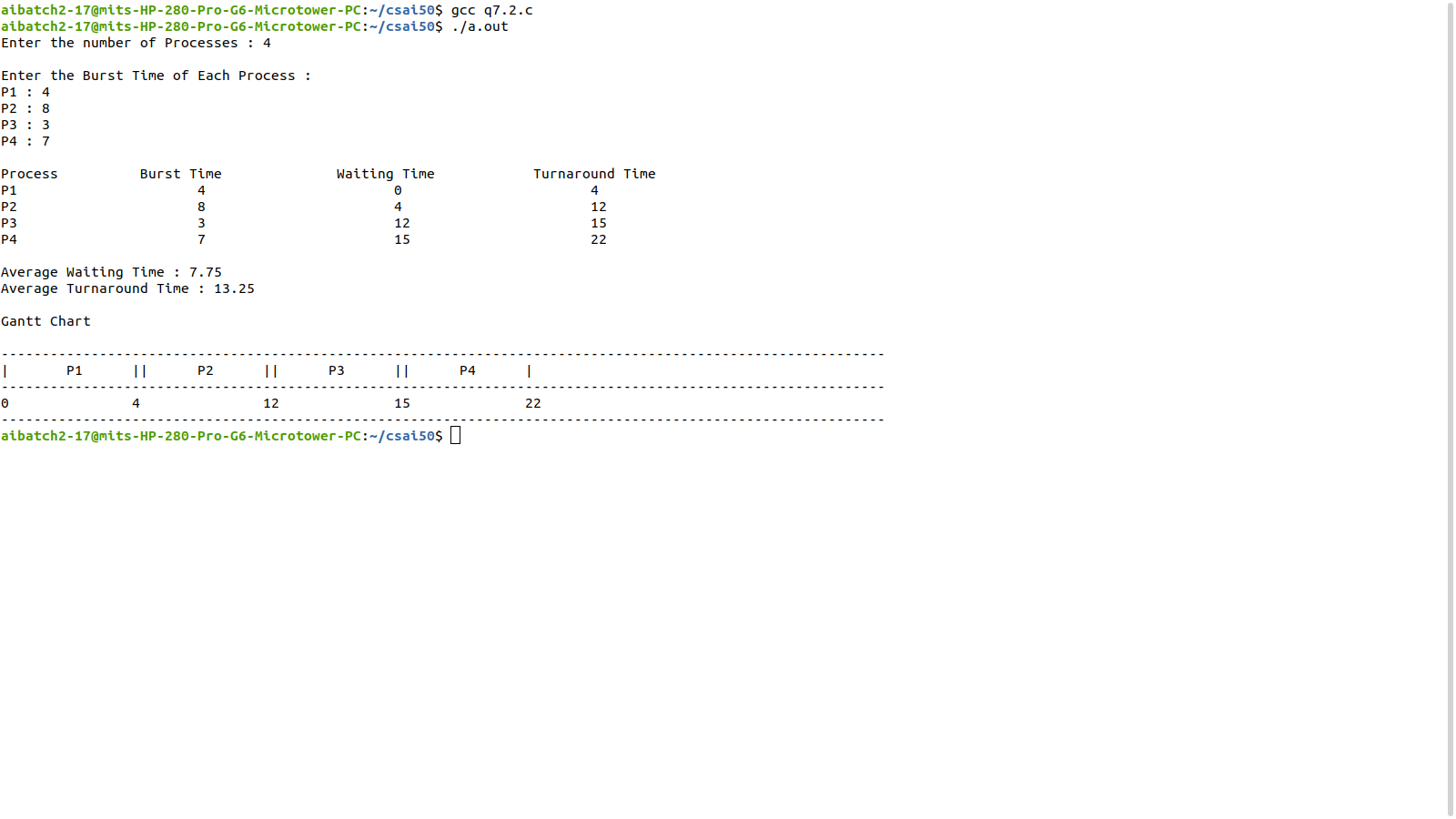
printf("%d",tt[n-1]);

printf("\n----------------------------------------------------------------

--------------------------------------------\n");

}

**OUTPUT**



**PROGRAM**

//FCFS Scheduling

#include<stdio.h>

void main() {

int p[20],wt[20],bt[20],tt[20],n,temp;

float wt\_avg=0,tt\_avg=0;

printf("Enter the number of Processes : ");

scanf("%d",&n);

printf("\nEnter the Burst Time of Each Process :\n");

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

p[i]=i+1;

printf("P%d : ",i+1);

scanf("%d",&bt[i]);

}

wt[0]=0;

for(int i=1;i<n;i++) {

wt[i]=bt[i-1]+wt[i-1];

wt\_avg+=wt[i];

}

wt\_avg/=n;

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

tt[i]=wt[i]+bt[i];

tt\_avg+=tt[i];

}

tt\_avg/=n;

printf("\nProcess\t\t Burst Time\t\t Waiting Time\t\t Turnaround Time\n");

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

printf("P%d\t\t\t%d\t\t\t%d\t\t\t%d\n",p[i],bt[i],wt[i],tt[i]);

printf("\nAverage Waiting Time : %.2f",wt\_avg);

printf("\nAverage Turnaround Time : %.2f\n",tt\_avg);

printf("\nGantt Chart\n\n");

printf("\n----------------------------------------------------------------------------------

--------------------------\n");

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

printf("|\tP%d\t|",p[i]);

printf("\n----------------------------------------------------------------------------------

--------------------------\n");

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

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

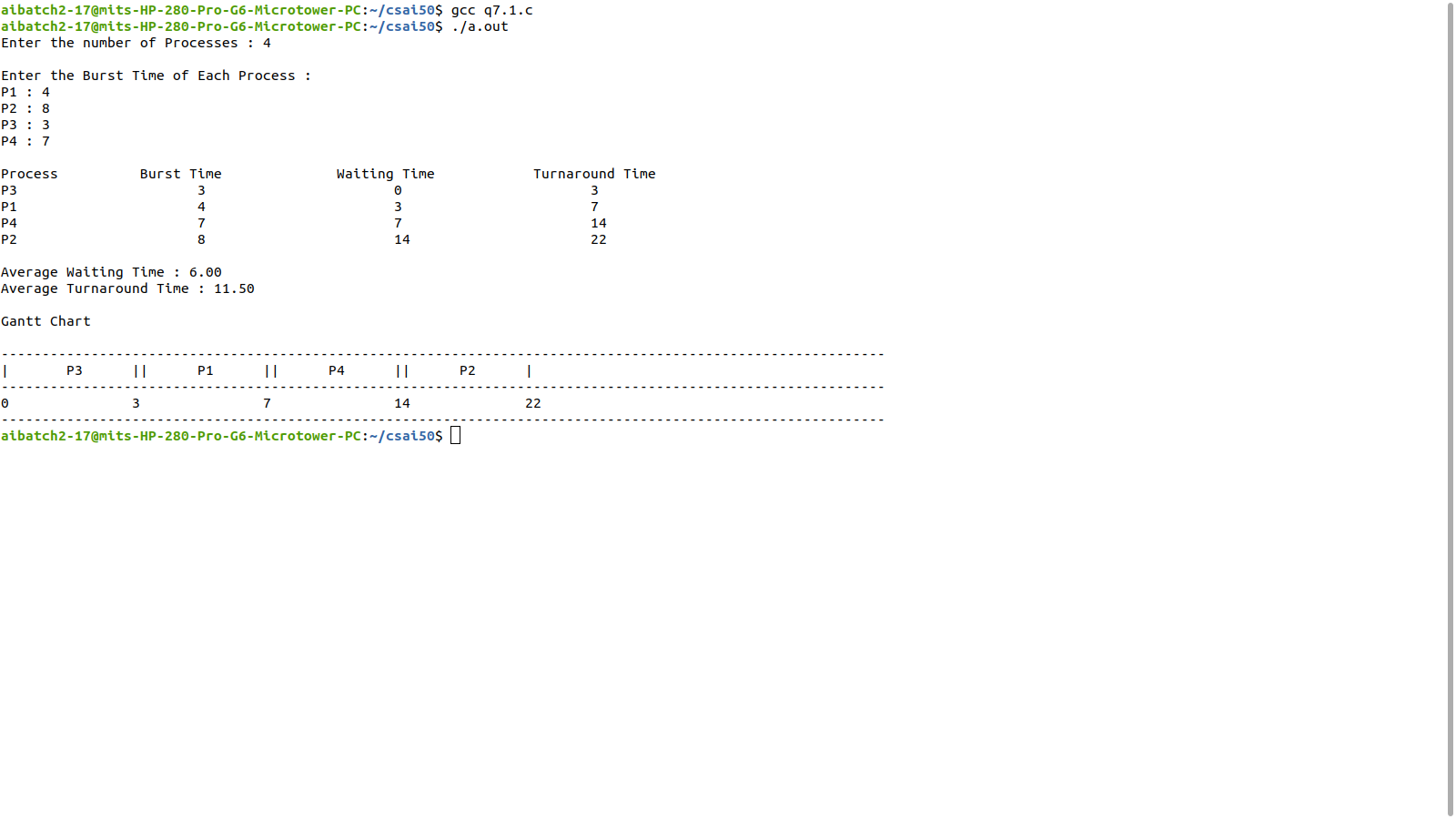
printf("%d",tt[n-1]);

printf("\n----------------------------------------------------------------------------------

--------------------------\n");

}

**OUTPUT**



**PROGRAM**

//Round Robin Scheduling

#include<stdio.h>

void main() {

int i,sum,gt[20],time[20],qt,n,bt[20],tt[20],wt[20],bt\_cp[20],p[20],temp,count=0,l=0,k=1;

float wt\_avg=0,tt\_avg=0;

printf("Enter the number of Processes(Max 20) : ");

scanf("%d",&n);

printf("\nEnter the Burst Time of Each Process :\n");

for(int j=0;j<n;j++) {

p[j]=j+1;

printf("P%d : ",j+1);

scanf("%d",&bt[j]);

bt\_cp[j]=bt[j];

}

time[0]=0;

printf("\nEnter the Time Slice : ");

scanf("%d",&qt);

while(count!=n) {

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

if(bt\_cp[i]==0) {

count++;

continue;

}

if(bt\_cp[i]>qt) {

bt\_cp[i]-=qt;

temp=qt;

}

else

if(bt\_cp[i]<=qt && bt\_cp[i]>0) {

temp=bt\_cp[i];

bt\_cp[i]=0;

}

sum+=temp;

tt[i]=sum;

gt[l++]=p[i];

time[k]=time[k-1]+temp;

k++;

}

}

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

wt[i]=tt[i]-bt[i];

wt\_avg+=wt[i];

tt\_avg+=tt[i];

}

wt\_avg/=n;

tt\_avg/=n;

printf("\nProcess\t\t Burst Time\t\t Waiting Time\t\t Turnaround Time\n");

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

printf("P%d\t\t\t%d\t\t\t%d\t\t\t%d\n",p[i],bt[i],wt[i],tt[i]);

printf("\nAverage Waiting Time : %.2f ms",wt\_avg);

printf("\nAverage Turnaround Time : %.2f ms\n",tt\_avg);

printf("\nGantt Chart\n\n");

printf("\n-----------------------------------------------------------------

---------------------------------------------------------------------------\n");

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

printf("|\tP%d\t|",gt[i]);

printf("\n-----------------------------------------------------------------

---------------------------------------------------------------------------\n");

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

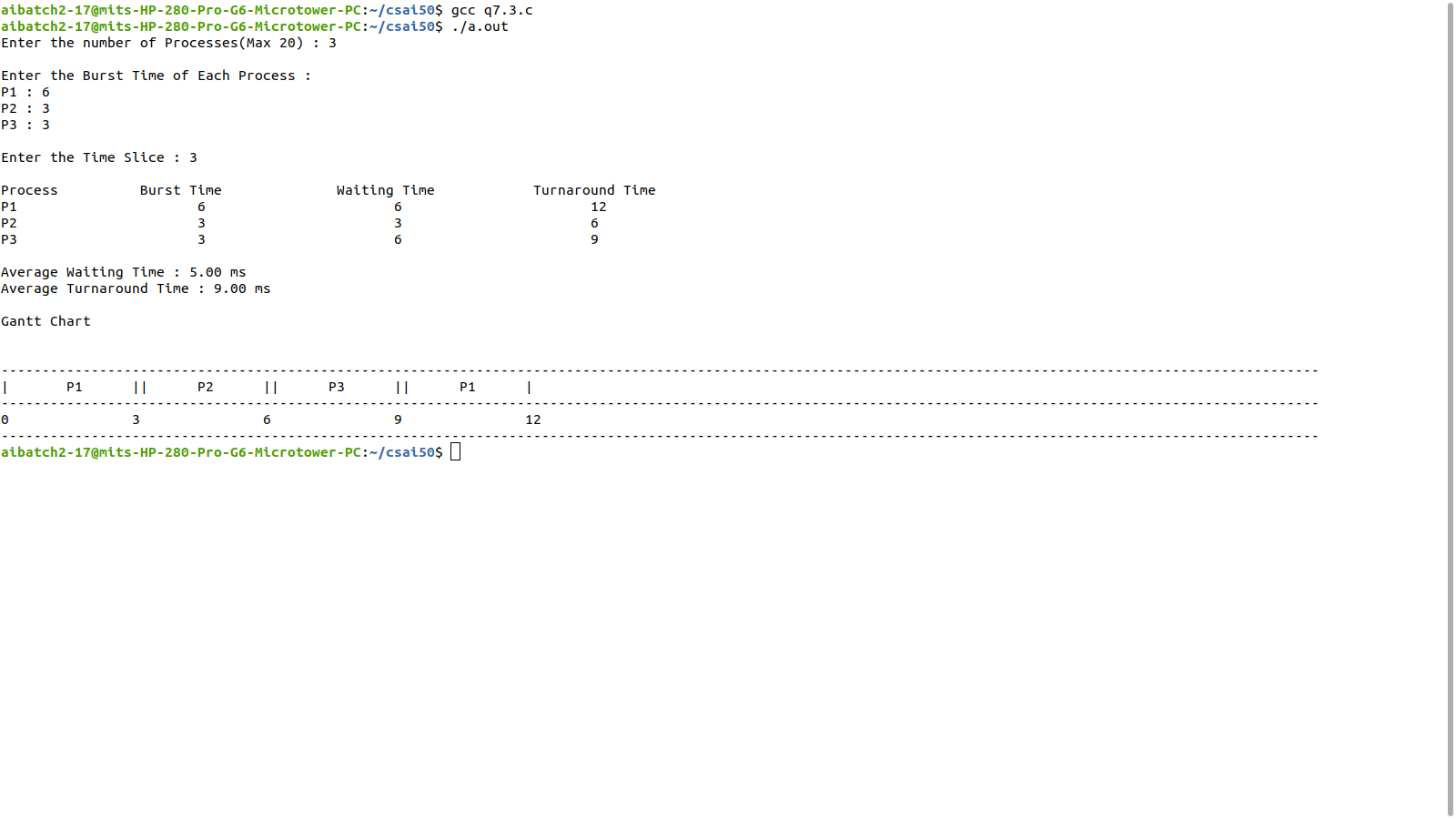
printf("%d\t \t",time[i]);

printf("\n-----------------------------------------------------------------

---------------------------------------------------------------------------\n");

}

**OUTPUT**



**PROGRAM**

//Priority Scheduling

#include<stdio.h>

void main() {

int p[20],wt[20],bt[20],tt[20],prty[20],n,temp;

float wt\_avg=0,tt\_avg=0;

printf("Enter the number of Processes : ");

scanf("%d",&n);

printf("\nEnter the Priority & Burst Time of Each Process :\n");

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

p[i]=i+1;

printf("P%d : ",i+1);

scanf("%d%d",&prty[i],&bt[i]);

}

for(int i=1;i<n;i++)

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

if(prty[j]>prty[j+1]) {

temp=p[j];

p[j]=p[j+1];

p[j+1]=temp;

temp=bt[j];

bt[j]=bt[j+1];

bt[j+1]=temp;

temp=prty[j];

prty[j]=prty[j+1];

prty[j+1]=temp;

}

wt[0]=0;

for(int i=1;i<n;i++) {

wt[i]=bt[i-1]+wt[i-1];

wt\_avg+=wt[i];

}

wt\_avg/=n;

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

tt[i]=wt[i]+bt[i];

tt\_avg+=tt[i];

}

tt\_avg/=n;

printf("\nProcess\t\t Burst Time\t\t Waiting Time\t\t

Turnaround Time\n");

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

printf("P%d\t\t\t%d\t\t\t%d\t\t\t%d\n",p[i],bt[i],wt[i],tt[i]);

printf("\nAverage Waiting Time : %.2f",wt\_avg);

printf("\nAverage Turnaround Time : %.2f\n",tt\_avg);

printf("\nGantt Chart\n\n");

printf("-------------------------------------------------------------------

------------------\n");

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

printf("|\tP%d\t|",p[i]);

printf("-------------------------------------------------------------------

------------------\n");

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

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

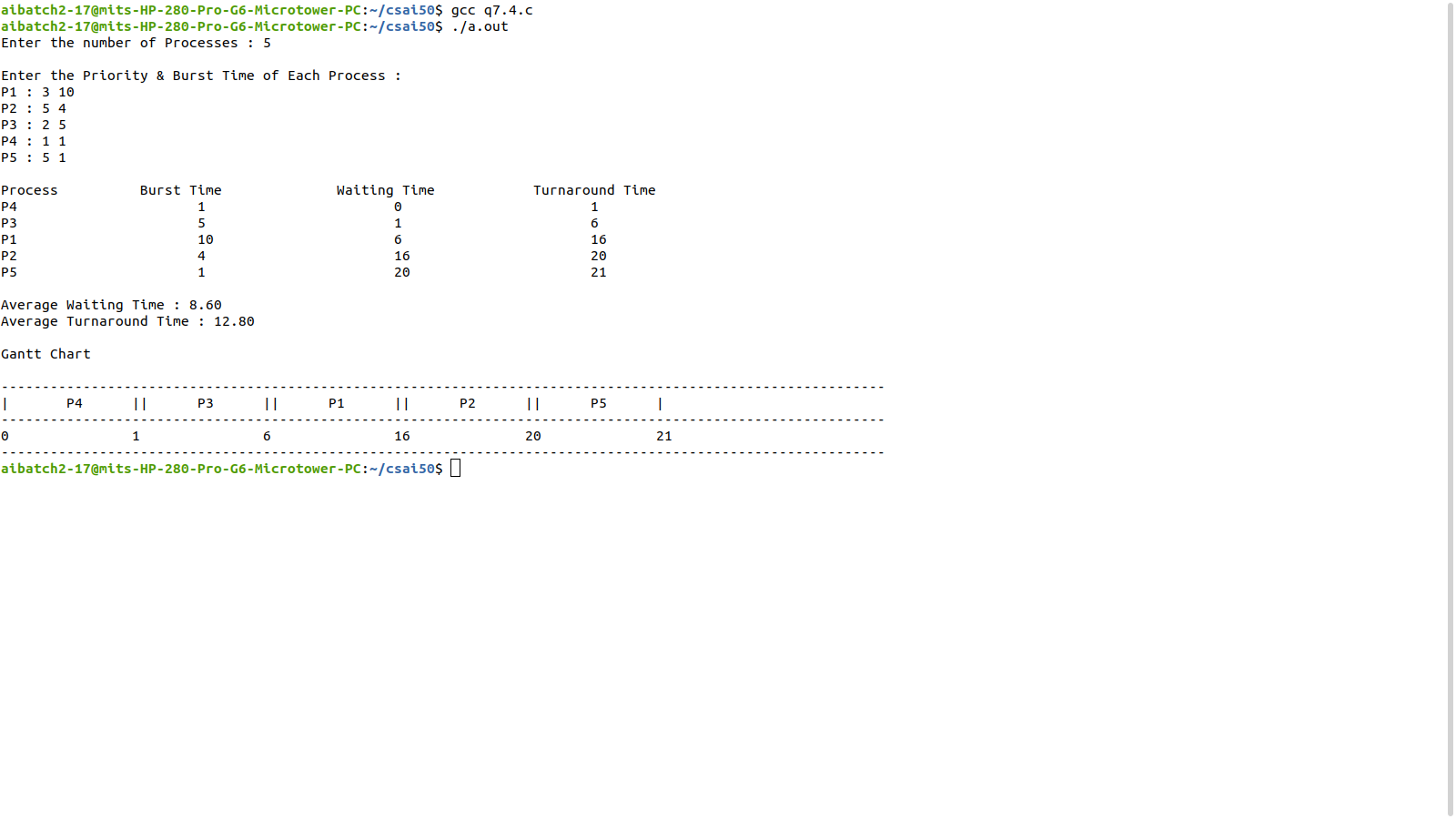
printf("%d",tt[n-1]);

printf("-------------------------------------------------------------------

------------------\n");

}

**OUTPUT**



**EXP 6: PREEMPTIVE SCHEDULING ALGORITHMS**

**PROGRAM**

//SRTF Scheduling

#include <stdio.h>

int main() {

int n,at[10],bt[10],ct[10],wt[10],temp[10],tat[10],p[10],smallest,count=0,time;

double avg\_wt=0,avg\_tt=0,end=0;

printf("Enter the no. of Process: ");

scanf("%d",&n);

printf("Enter the AT and BT of processes\n");

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

printf("P%d:", i + 1);

scanf("%d %d",&at[i],&bt[i]);

temp[i]=bt[i];

}

bt[9] = 9876;

for(time=0;count!=n;time++) {

smallest=9;

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

if(at[i]<=time&&bt[i]<bt[smallest]&&bt[i]>0)

smallest=i;

bt[smallest]--;

if(bt[smallest]==0) {

count++;

end=time+1;

ct[smallest]=end;

wt[smallest]=end-at[smallest]-temp[smallest];

tat[smallest]=end-at[smallest];

}

}

printf("\n-------------------------------------------");

printf("\n Prcs\tAT\tBT\tCT\tTAT\tWT\n");

printf("-------------------------------------------\n");

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

printf(" P%d\t%d\t%d\t%d\t%d\t%d\n",i+1,at[i],temp[i],ct[i],tat[i],wt[i]);

avg\_tt+=tat[i];

avg\_wt+=wt[i];

}

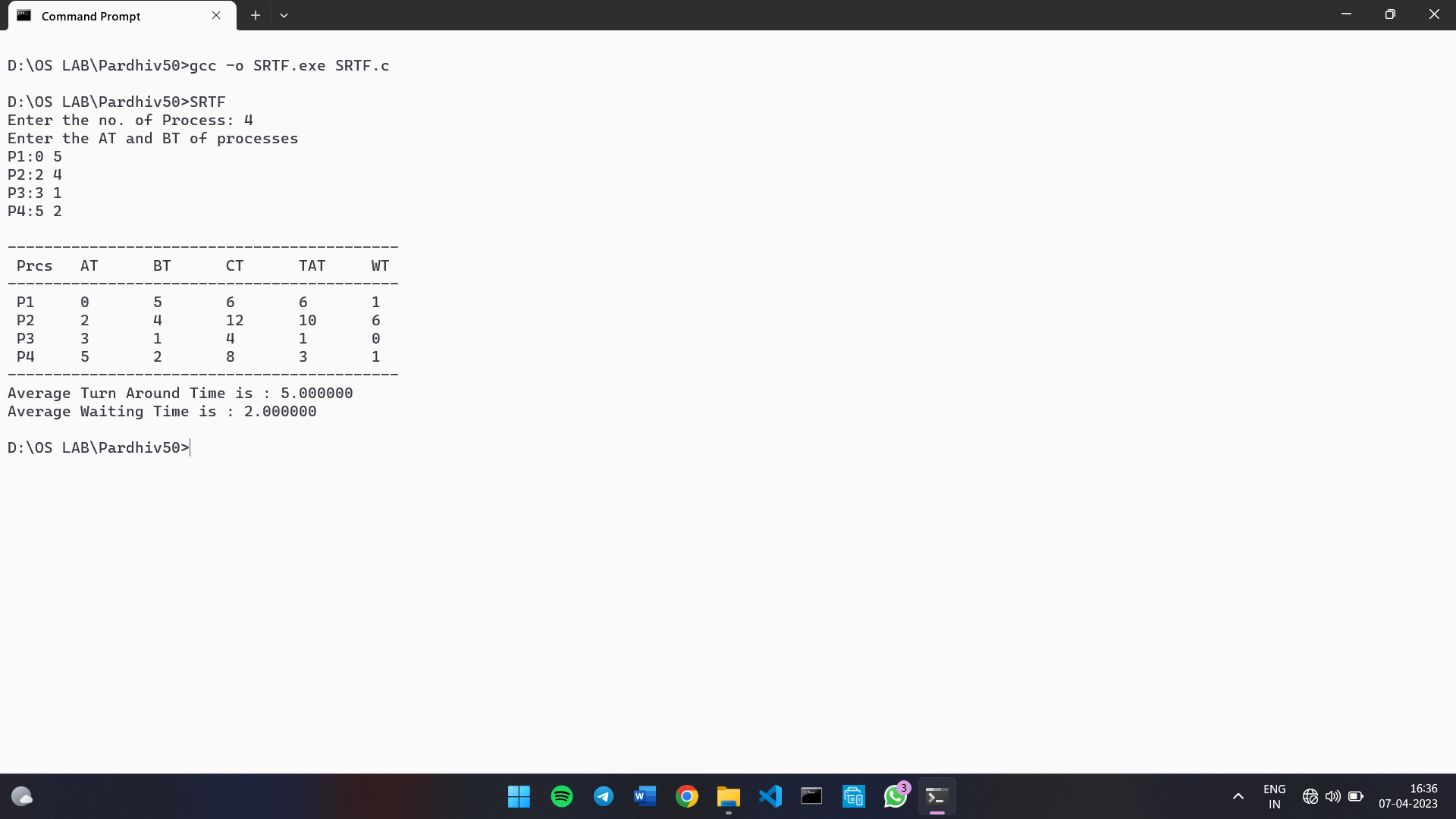
printf("-------------------------------------------");

printf("\nAverage Turn Around Time is : %lf\n",avg\_tt/n);

printf("Average Waiting Time is : %lf\n",avg\_wt/n);

}

**OUTPUT**

****

**PROGRAM**

//Pre-Emptive Priority Scheduling

#include <stdio.h>

int main() {

int n,at[10],bt[10],ct[10],wt[10],temp[10],tat[10],p[10],pr[10],smallest,count=0,time;

double avg\_wt=0,avg\_tt=0,end=0;

printf("Enter the no. of Process: ");

scanf("%d",&n);

printf("Enter the AT, BT and Priority of processes\n");

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

printf("P%d:", i + 1);

scanf("%d%d%d",&at[i],&bt[i],&pr[i]);

temp[i]=bt[i];

}

bt[9] = 9876;

for(time=0;count!=n;time++) {

smallest=9;

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

if(at[i]<=time&&pr[i]<pr[smallest]&&bt[i]>0)

smallest=i;

bt[smallest]--;

if(bt[smallest]==0) {

count++;

end=time+1;

ct[smallest]=end;

wt[smallest]=end-at[smallest]-temp[smallest];

tat[smallest]=end-at[smallest];

}

}

printf("\n---------------------------------------------------");

printf("\n Prcs\tPrty\tAT\tBT\tCT\tTAT\tWT\n");

printf("---------------------------------------------------\n");

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

printf(" P%d\t%d\t%d\t%d\t%d\t%d\t%d\n",i+1,pr[i],at[i],temp[i],ct[i],tat[i],wt[i]);

avg\_tt+=tat[i];

avg\_wt+=wt[i];

}

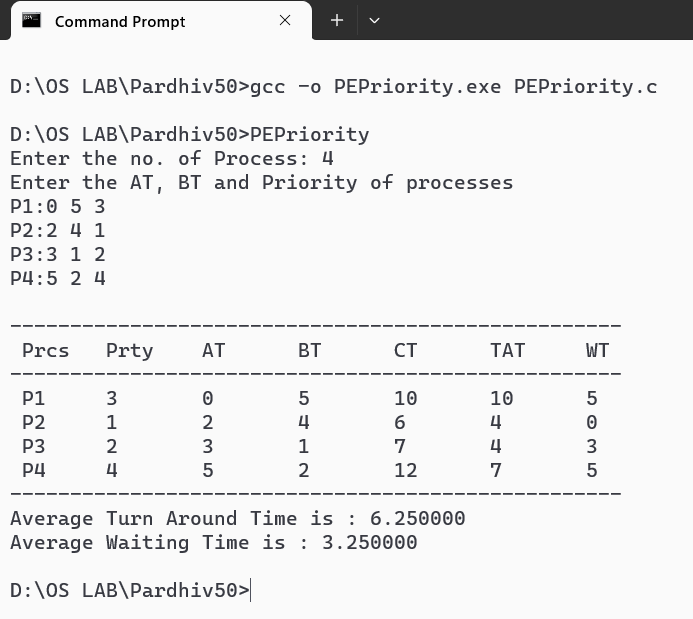
printf("---------------------------------------------------");

printf("\nAverage Turn Around Time is : %lf\n",avg\_tt/n);

printf("Average Waiting Time is : %lf\n",avg\_wt/n);

}

**OUTPUT**

****

**EXP 7: IPC USING SHARED MEMORY**

**PROGRAM: WRITER PROCESS**

#include<sys/ipc.h>

#include<sys/shm.h>

#include<unistd.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void main() {

int id;

void \*sm;

char buf[100];

id=shmget((key\_t)1222,1024,0666|IPC\_CREAT);

printf("Key of Shared Memory is %d\n",id);

sm=shmat(id,NULL,0);

printf("Process attached at %p\n",sm);

printf("Enter the data to be written:\n");

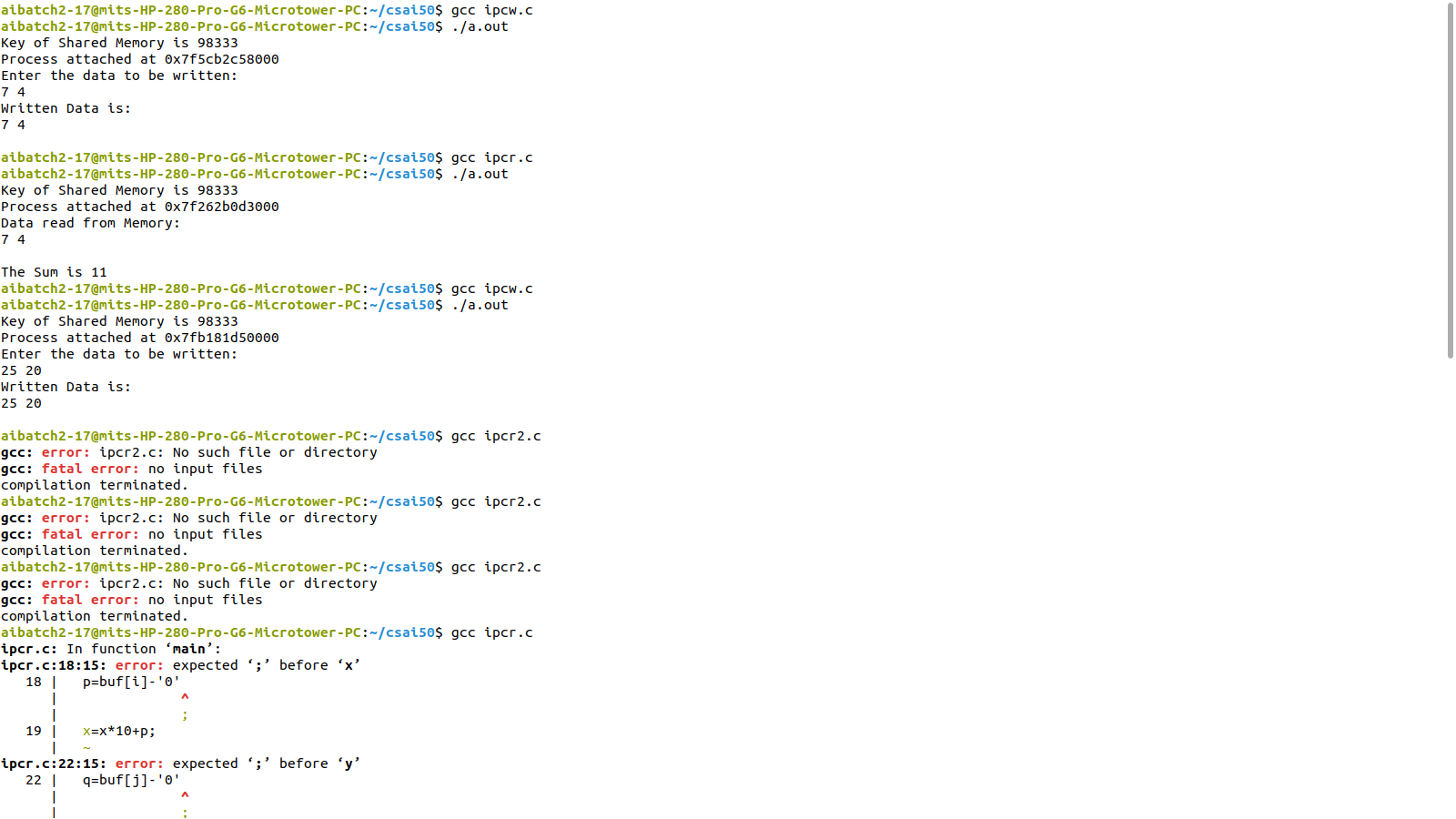
read(0,buf,100);

strcpy(sm,buf);

printf("Written Data is:\n%s\n",(char \*)sm);

}

**OUTPUT**

****

**PROGRAM: READER PROCESS**

#include<sys/ipc.h>

#include<sys/shm.h>

#include<unistd.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void main() {

int id;

void \*sm;

char buf[100];

id=shmget((key\_t)1222,1024,0666);

printf("Key of Shared Memory is %d\n",id);

sm=shmat(id,NULL,0);

printf("Process attached at %p\n",sm);

printf("Data read from Memory:\n%s\n",(char \*)sm);

strcpy(buf,sm);

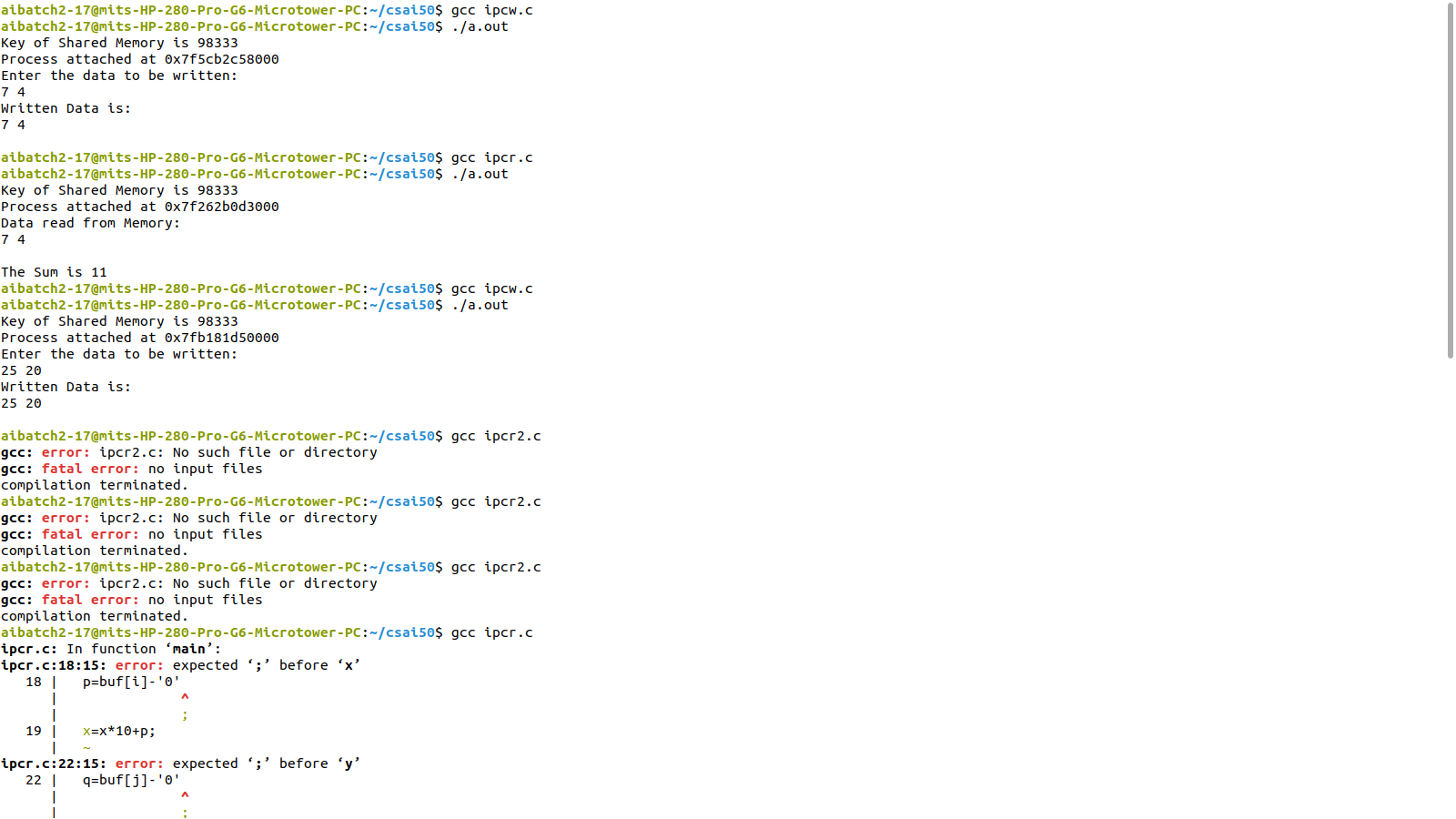
int a=buf[0]-'0';

int b=buf[2]-'0';

printf("The Sum is %d\n",a+b);

}

**OUTPUT**

****

**EXP 8: PRODUCER-CONSUMER PROBLEM**

**PROGRAM**

#include<stdio.h>

#include<semaphore.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

sem\_t mutex;

sem\_t empty;

sem\_t full;

int buffer[8];

pthread\_t p[5];

pthread\_t c[5];

void producer(int \*p) {

int a[10],i=0,n=\*(int\*)p;

while(i<=5) {

sem\_wait(&empty);

sem\_wait(&mutex);

a[i]=3;

printf("Producer %d Produced Item %d\n",n,i);

sleep(1);

i++;

buffer[i]=a[i];

sem\_post(&mutex);

sem\_post(&full);

}

}

void consumer(void \*p) {

int b[10],i=0,n=\*(int\*)p;

while(i<=5) {

sem\_wait(&full);

sem\_wait(&mutex);

printf("Consumer %d Consumes Item %d\n",n,i);

sleep(1);

b[i]=buffer[i];

i++;

sem\_post(&mutex);

sem\_post(&empty);

}

}

void main()

{

int n;

sem\_init(&mutex,0,1);

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

sem\_init(&full,0,0);

for(n=0;n<5;n++) {

pthread\_create(&p[n],0,(void \*)producer,(void \*)&n);

pthread\_create(&c[n],0,(void \*)consumer,(void \*)&n);

}while(1);

}

**OUTPUT**



**EXP 9: DINING PHILOSOPHER PROBLEM**

**PROGRAM**

#include<stdio.h>

#include<semaphore.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

#define N 5

#define LEFT (i+4)%5

#define RIGHT (i)%5

#define THINKING 0

#define HUNGRY 1

#define EATING 2

int state[N];

pthread\_t t[N];

sem\_t s[N];

sem\_t mutex;

void think(int n) {

printf("The Philosopher %d is thinking \n",n);

sleep(1);

}

void eat(int n) {

printf("Philosopher %d is eating\n",n);

sleep(1);

printf("Philosopher %d finished eating\n",n);

}

void take\_fork(int i) {

sem\_wait(&mutex);

state[i]=HUNGRY;

if(state[i]==HUNGRY && state[LEFT]!=EATING && state[RIGHT]!=EATING) {

state[i]=EATING;

sem\_wait(&s[LEFT]);

sem\_wait(&s[RIGHT]);

}

sem\_post(&mutex);

}

void putforks(int i) {

state[i]=THINKING;

sem\_post(&s[LEFT]);

sem\_post(&s[RIGHT]);

}

void \*philo(int n) {

while(1) {

think(n);

take\_fork(n);

if(state[n]==EATING) {

eat(n);

putforks(n);

}

}

}

void main() {

int i;

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

sem\_init(&s[i],0,1);

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

sem\_init(&mutex,0,1);

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

pthread\_create(&t[i],0,(void \*)philo,(void \*)i);

}while(1);

}

**OUTPUT**



**EXP 10: BANKER’S ALGORITHM FOR DEADLOCK AVOIDANCE**

**PROGRAM**

#include<stdio.h>

#include<stdbool.h>

#include<stdlib.h>

void exit(int status);

int i,j,no,res;

int safety(int[][10],int[],int[][10],int[]);

void output(int [][10]);

void request(int[][10],int[][10],int[],int[],int);

void main(){

int ans,id,seq[10],R[10],A[10][10],C[10][10],N[10][10],W[10]={0},req[10],AV[10];

printf("\nEnter the number of Processes (less than 10): ");

scanf("%d",&no);

printf("Enter the number of resources (less than 10): ");

scanf("%d",&res);

printf("Enter the max available instances of each resource : \n");

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

printf("R%d:",i);

scanf("%d",&R[i]);

}

printf("\nEnter the Allocated Resource Table : \n");

for(i=0;i<no;i++)

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

scanf("%d",&A[i][j]);

printf("\nEnter the Maximum Claim Table : \n");

for(i=0;i<no;i++)

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

scanf("%d",&C[i][j]);

for(i=0;i<no;i++)

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

N[i][j]=C[i][j]-A[i][j];

printf("\nAllocated Resource Table:\n\t");

output(A);

printf("\n\nMaximum Claim Table:\n\t");

output(C);

printf("\n\nNeed Matrix:\n\t");

output(N);

for(j=0;j<res;j++){

for(i=0;i<no;i++)

W[j]+=A[i][j];

W[j]=R[j]-W[j];

}

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

AV[j]=W[j];

int ch=safety(A,W,N,seq);

if(ch==1){

printf("\n\nThe System is in SAFE STATE :)\n");

printf("\nThe Safe Sequence : ");

for(i=0;i<no;i++)

printf("|P%d|",seq[i]);

printf("\nResource Request for a Process Needed? (1=YES,0=NO): ");

scanf("%d",&ans);

if(ans==1){

printf("Enter the Process ID for initiating request: ");

scanf("%d",&id);

printf("Enter the Request Vector for P%d : ",id);

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

scanf("%d",&req[j]);

request(A,N,req,AV,id);

}

}

}

void output(int arr[10][10]){

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

printf("R%d\t",i);

for(i=0;i<no;i++){

printf("\nP%d\t",i);

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

printf("%d\t",arr[i][j]);

}

}

int safety(int A[10][10],int W[10],int N[10][10],int seq[]){

int x=0,flg=0,target=0;

int finish[10];

for(i=0;i<no;i++)

finish[i]=0;

for(int w=0;w<no;w++){

label:

for(int i=0;i<no;i++){

flg=0;

for(int j=0;j<res;j++)

if(N[i][j] > W[j])

flg++;

if(flg==0 && finish[i]==0){

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

W[j]+=A[i][j];

finish[i]=1;

target++;

seq[x++]=i;

goto label;

}

continue;

}

}

if(target==no)

return 1;

else

printf("\nThe System is in UNSAFE STATE :(\n");

}

void request(int A[10][10],int N[10][10],int req[10],int AV[10],int ID){

int seq[10];

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

if(req[i] > N[ID][i]){

printf("\nNOT POSSIBLE as the process P%d has exceeded its max claim !!",ID);

exit(0);

}

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

if(req[i] > AV[i]){

printf("\nThe process P%d has to wait since resouces are not

available yet!!",ID);

exit(0);

}

for(j=0;j<res;j++){

AV[j] -= req[j];

A[ID][j] += req[j];

N[ID][j] -= req[j];

}

int ch=safety(A,AV,N,seq);

if(ch==1){

printf("\nThe System is in SAFE state. Hence, the resources can be allocated\n");

printf("\nThe Safe Sequence : ");

for(i=0;i<no;i++)

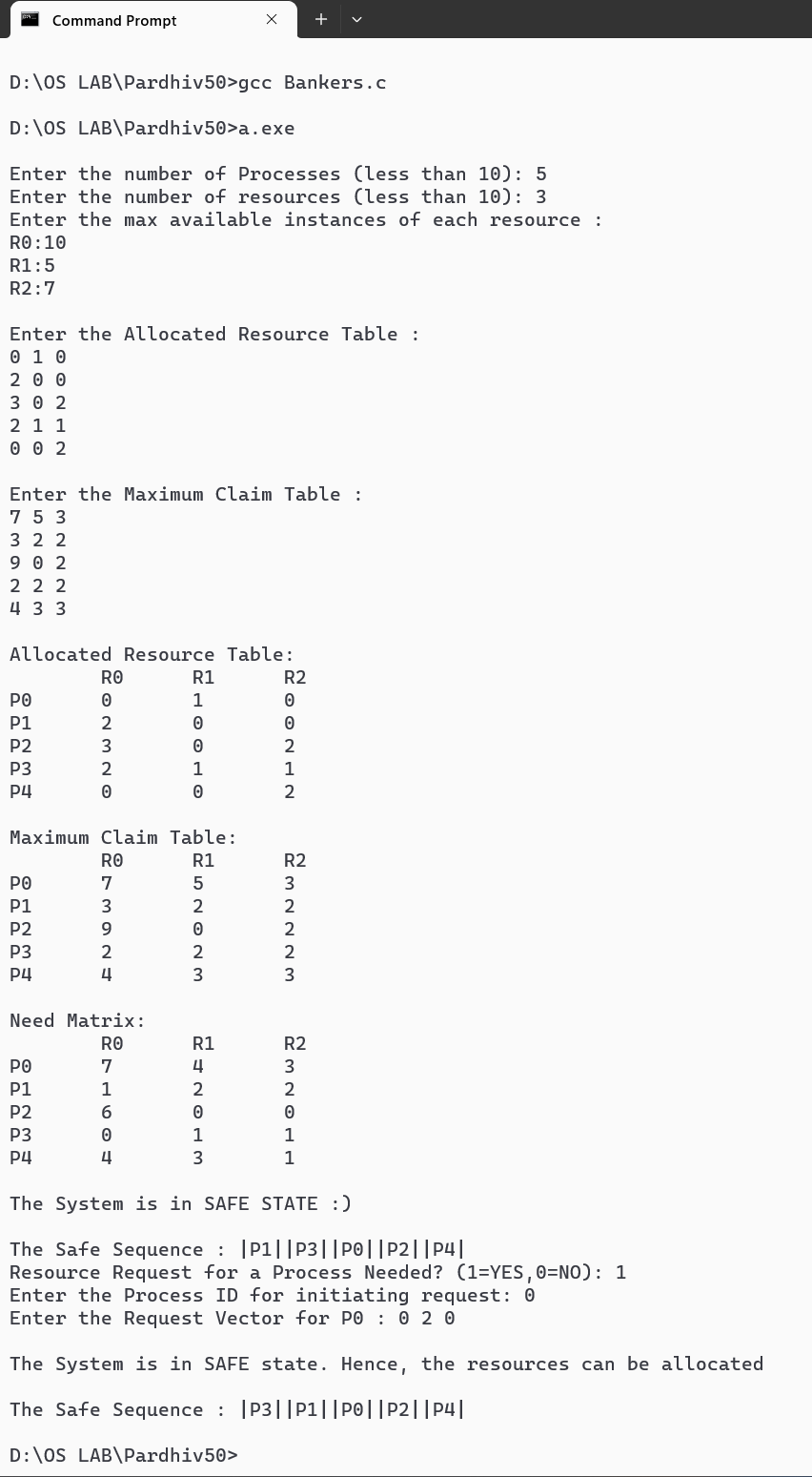
printf("|P%d|",seq[i]);

}

printf("\n");

}

**OUTPUT**

****

**EXP 11: DEADLOCK DETECTION ALGORITHM**

**PROGRAM**

#include<stdio.h>

#include<conio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n,r;

void input();

void show();

void cal();

int main(){

int i,j;

input();

show();

cal();

getch();

return 0;

}

void input(){

int i,j;

printf("Enter the no of Processes:");

scanf("%d",&n);

printf("Enter the no of resource instances:");

scanf("%d",&r);

printf("Enter the Max Matrix:\n");

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

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

scanf("%d",&max[i][j]);

printf("\nEnter the Allocation Matrix:\n");

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

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

scanf("%d",&alloc[i][j]);

printf("\nEnter the available Resources:\n");

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

scanf("%d",&avail[j]);

}

void show(){

int i,j;

printf("\nProcess\t Allocation\t Max\t\t Available");

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

printf("\nP%d\t ",i+1);

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

printf("%d ",alloc[i][j]);

printf("\t ");

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

printf("%d ",max[i][j]);

printf("\t ");

if(i==0)

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

printf("%d ",avail[j]);

}

}

void cal(){

int finish[100],temp,need[100][100],flag=1,k,c1=0;

int dead[100],safe[100],i,j;

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

finish[i]=0;

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

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

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

while(flag){

flag=0;

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

int c=0;

for(j=0;j<r;j++){

if((finish[i]==0)&&(need[i][j]<=avail[j])){

c++;

if(c==r){

for(k=0;k<r;k++){

avail[k]+=alloc[i][j];

finish[i]=1;

flag=1;

}

if(finish[i]==1){

i=n;

}

}

}

}

}

}

j=0;

flag=0;

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

if(finish[i]==0){

dead[j]=i;

j++;

flag=1;

}

}

if(flag==1){

printf("\n\nSystem is in Deadlock and the Deadlock process are\n");

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

printf("P%d\t",dead[i]);

}

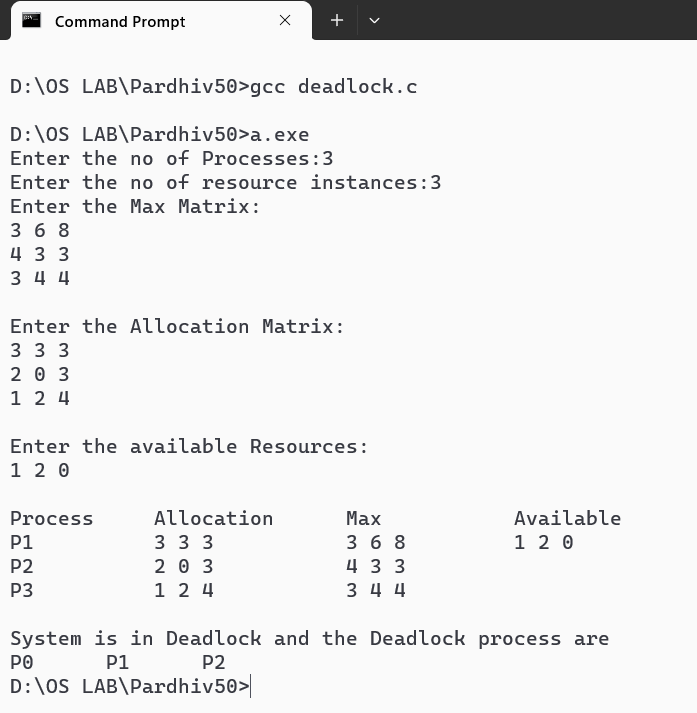
}

else

printf("\nNo Deadlock Occur");

}

**OUTPUT**

****

**EXP 12: MEMORY ALLOCATION**

**PROGRAM**

#include<stdio.h>

int p,b,block[10],process[10],b\_copy[10],remain[10];

void first\_fit();

void best\_fit();

void worst\_fit();

void main() {

int i;

printf("How many process & blocks ? :");

scanf("%d%d",&p,&b);

printf("\nProcess sizes ?\n");

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

printf("P%d: ",i+1);

scanf("%d",&process[i]);

}

printf("\nBlock sizes ?\n");

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

printf("B%d: ",i+1);

scanf("%d",&block[i]);

}

printf("\nFIRST FIT:\n");

first\_fit();

printf("\nBEST FIT:\n");

best\_fit();

printf("\nWORST FIT:\n");

worst\_fit();

}

void first\_fit() {

int i,j,p\_flg[10]={0},b\_flg[10]={0};

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

remain[i]=b\_copy[i]=block[i];

printf("Process Name\tProcess Size\tBlock name\tTotal

Space\tWastage Space\n");

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

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

if(process[i]<=block[j] && b\_flg[j]==0) {

remain[j]=block[j]-process[i];

printf("P%d\t\t%d\t\tB%d\t\t%d\t\t%d\n"

,i+1,process[i],j+1,block[j],remain[j]);

p\_flg[i]=b\_flg[j]=1;break;

}

if(p\_flg[i]==0)

printf("P%d\t\t%d\t\t-\t\t-\t\t-\n",i+1,process[i]);

}

}

void best\_fit() {

int i,j,p\_flg[10]={0},b\_flg[10]={0},b\_index[10];

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

b\_index[i]=i+1;

remain[i]=b\_copy[i]=block[i];

}

for(i=1;i<b;i++)

for(j=0;j<b-i;j++)

if(block[j]>block[j+1]) {

int temp=block[j];

block[j]=block[j+1];

block[j+1]=temp;

temp=b\_index[j];

b\_index[j]=b\_index[j+1];

b\_index[j+1]=temp;

}

printf("Process Name\tProcess Size\tBlock name

\tTotal Space\tWastage Space\n");

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

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

if(process[i]<=block[j] && b\_flg[j]==0) {

remain[j]=block[j]-process[i];

printf("P%d\t\t%d\t\tB%d\t\t%d\t\t%d\n",

i+1,process[i],b\_index[j],block[j],remain[j]);

p\_flg[i]=b\_flg[j]=1;break;

}

if(p\_flg[i]==0)

printf("P%d\t\t%d\t\t-\t\t-\t\t-\n",i+1,process[i]);

}

}

void worst\_fit() {

int i,j,p\_flg[10]={0},b\_flg[10]={0},b\_index[10];

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

b\_index[i]=i+1;

remain[i]=b\_copy[i];

}

for(i=1;i<b;i++)

for(j=0;j<b-i;j++)

if(b\_copy[j]<b\_copy[j+1]) {

int temp=b\_copy[j];

b\_copy[j]=b\_copy[j+1];

b\_copy[j+1]=temp;

temp=b\_index[j];

b\_index[j]=b\_index[j+1];

b\_index[j+1]=temp;

}

printf("Process Name\tProcess Size\tBlock name

\tTotal Space\tWastage Space\n");

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

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

if(process[i]<=b\_copy[j] && b\_flg[j]==0) {

remain[j]=b\_copy[j]-process[i];

printf("P%d\t\t%d\t\tB%d\t\t%d\t\t%d\n",

i+1,process[i],b\_index[j],b\_copy[j],remain[j]);

p\_flg[i]=b\_flg[j]=1;break;

}

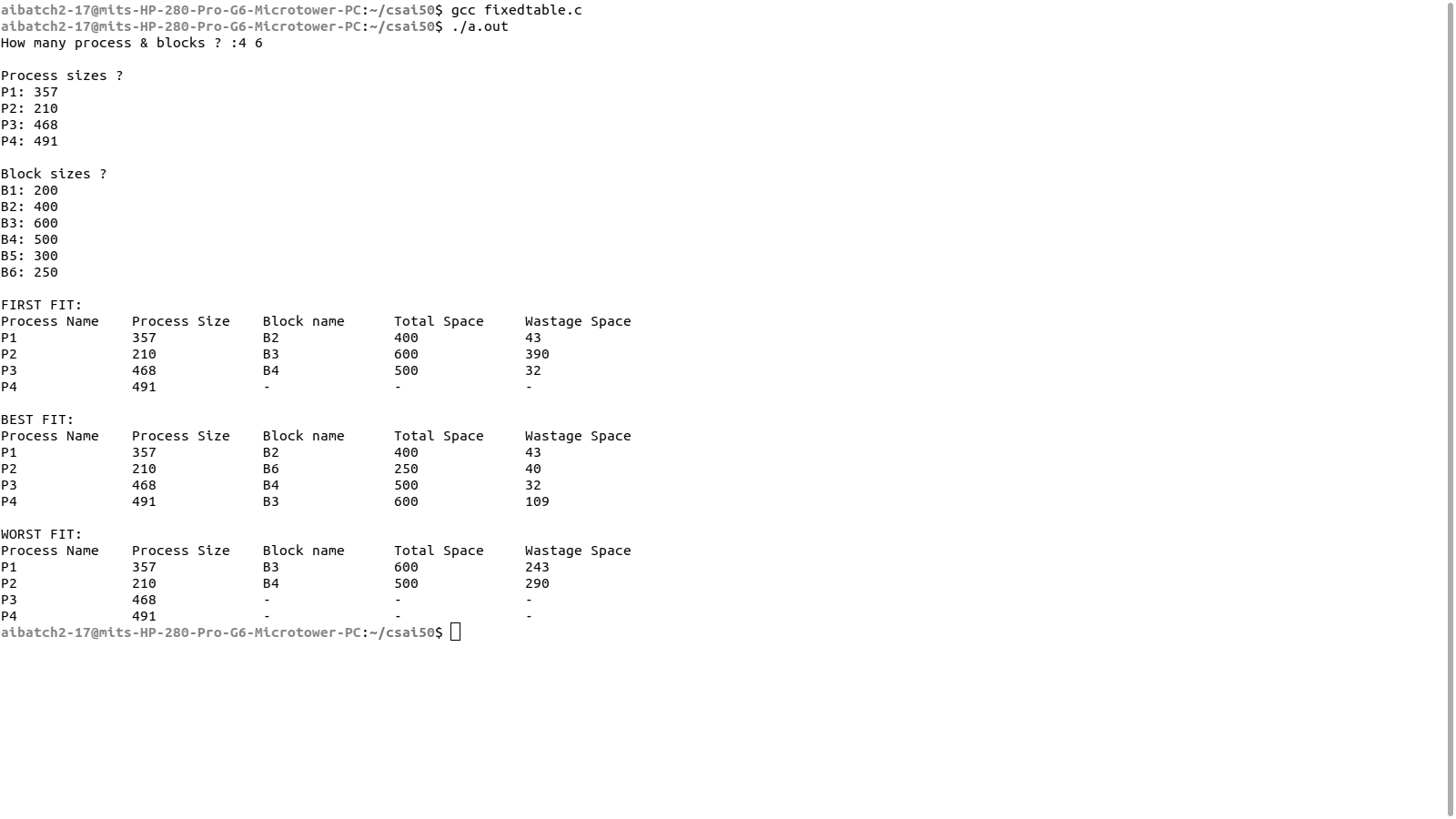
if(p\_flg[i]==0)

printf("P%d\t\t%d\t\t-\t\t-\t\t-\n",i+1,process[i]);

}

}

**OUTPUT**

****

**EXP 13: PAGE REPLACEMENT ALGORITHMS**

**PROGRAM: FIRST IN FIRST OUT (FIFO)**

#include<stdio.h>

int main() {

int frames,l,i,j,k,exist,m[10],str[100],count=0;

printf("Enter the length of the reference string: ");

scanf("%d",&l);

printf("Enter the reference string: ");

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

scanf("%d",&str[i]);

printf("Enter the no. of partitions: ");

scanf("%d",&frames);

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

m[i]=-1;

printf("\nThe Page Replacement Process is....\n");

for(i=0;i<l;i++,exist=0) {

for(j=0;j<fra

mes;j++)

if(m[j]==str[i])

exist=1;

if(exist==0) {

m[count%frames]=str[i];

count++;

}

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

printf("%d\t",m[k]);

if(exist==0)

printf("Page Fault: %d\n",count);

else

printf("HIT!\n");

}

printf("\nTotal Page Fault = %d\n",count);

printf("Total Hits = %d\n",l-count);

printf("Miss Ratio = %d%%\n",(count\*100)/l);

printf("Hit Ratio = %d%%\n\n",((l-count)\*100)/l);

}

**OUTPUT**

****

**PROGRAM: LEAST RECENTLY USED (LRU)**

#include <stdio.h>

void main() {

int str[20],flg[20],i,j,f,len,count[20],m[10],next=0,min,pf=0;

printf("Enter the length of Reference String:");

scanf("%d",&len);

printf("Enter the Reference String");

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

scanf("%d",&str[i]);

flg[i]=0;

}

printf("Enter the no. of Frames:");

scanf("%d",&f);

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

count[i]=0;

m[i]=-1;

}

printf("\nTHE PAGE REPLACEMENT PROCESS IS...\n");

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

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

if(m[j]==str[i]) {

flg[i]=1;

count[j]=next++;

}

if(flg[i]==0) {

if(i<f) {

m[i]=str[i];

count[i]=next++;

}

else {

min=0;

for(j=1;j<f;j++)

if(count[min]>count[j])

min=j;

m[min]=str[i];

count[min]=next++;

}

pf++;

}

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

printf("%d\t",m[j]);

if(flg[i]!=1)

printf("Page Fault:%d\n",pf);

else

printf("HIT!\n");

}

printf("\nTotal Page Fault = %d\n",pf);

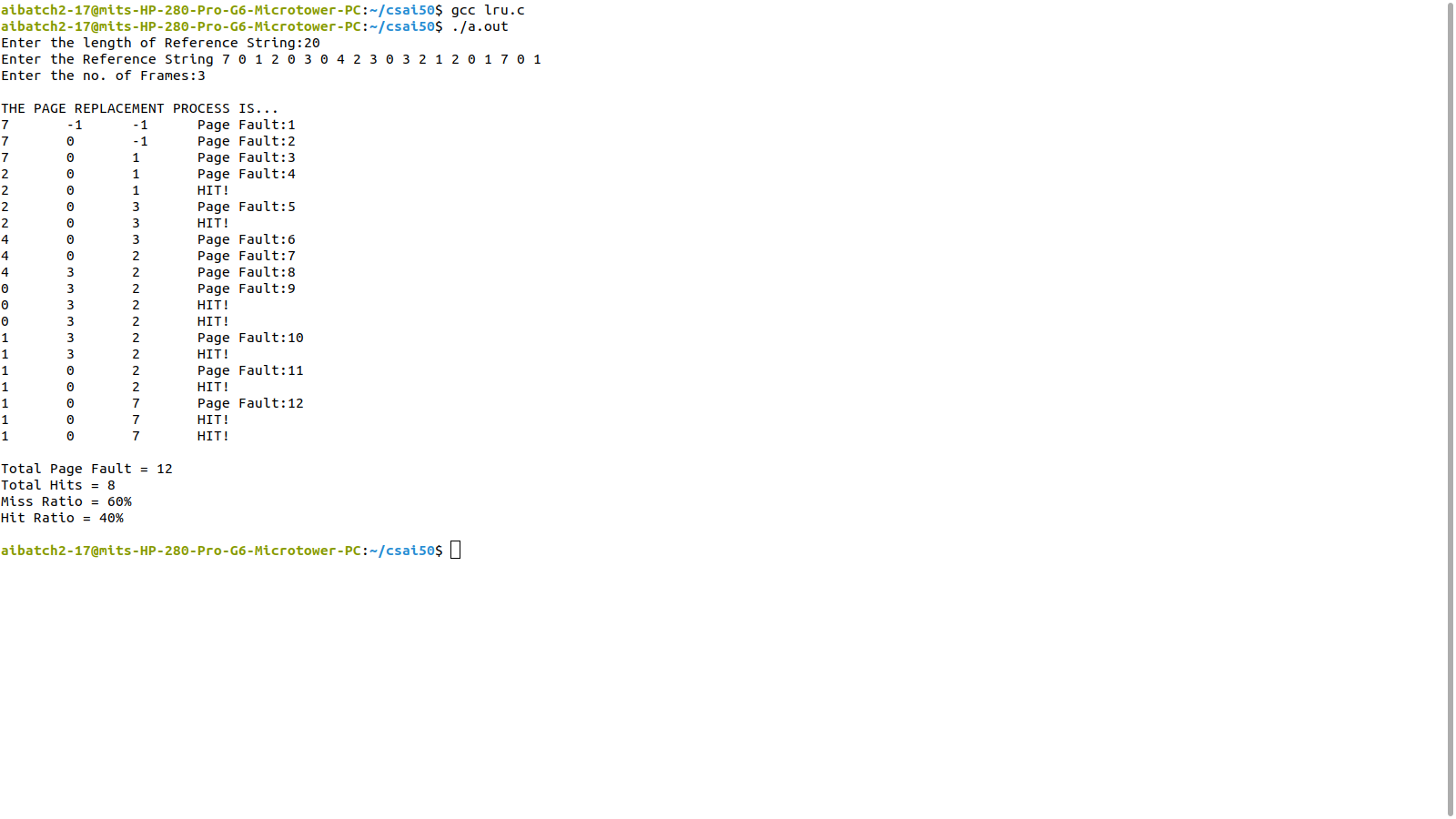
printf("Total Hits = %d\n",len-pf);

printf("Miss Ratio = %d%%\n",(pf\*100)/len);

printf("Hit Ratio = %d%%\n\n",((len-pf)\*100)/len);

}

**OUTPUT**

****

**PROGRAM: LEAST FREQUENTLY USED (LFU)**

#include <stdio.h>

void main() {

int str[20],flg[20]={0},i,j,f,len,count[20]={0};

int m[10],next=0,min,pf=0,freq[20]={0};

printf("Enter the length of Reference String:");

scanf("%d",&len);

printf("Enter the Reference String");

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

scanf("%d",&str[i]);

printf("Enter the no. of Frames:");

scanf("%d",&f);

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

m[i]=-1;

printf("\nTHE PAGE REPLACEMENT PROCESS IS...\n");

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

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

if(m[j]==str[i]) {

flg[i]=1;

count[j]=next++;

freq[j]++;

}

if(flg[i]==0) {

if(i<f) {

m[i]=str[i];

count[i]=next++;

freq[i]=1;

}

else {

min=0;

for(j=1;j<f;j++)

if(freq[min]>=freq[j])

if(count[min]>count[j])

min=j;

m[min]=str[i];

count[min]=next++;

freq[min]=1;

}

pf++;

}

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

printf("%d\t",m[j]);

if(flg[i]!=1)

printf("Page Fault:%d\n",pf);

else

printf("HIT!\n");

}

printf("\nTotal Page Fault = %d\n",pf);

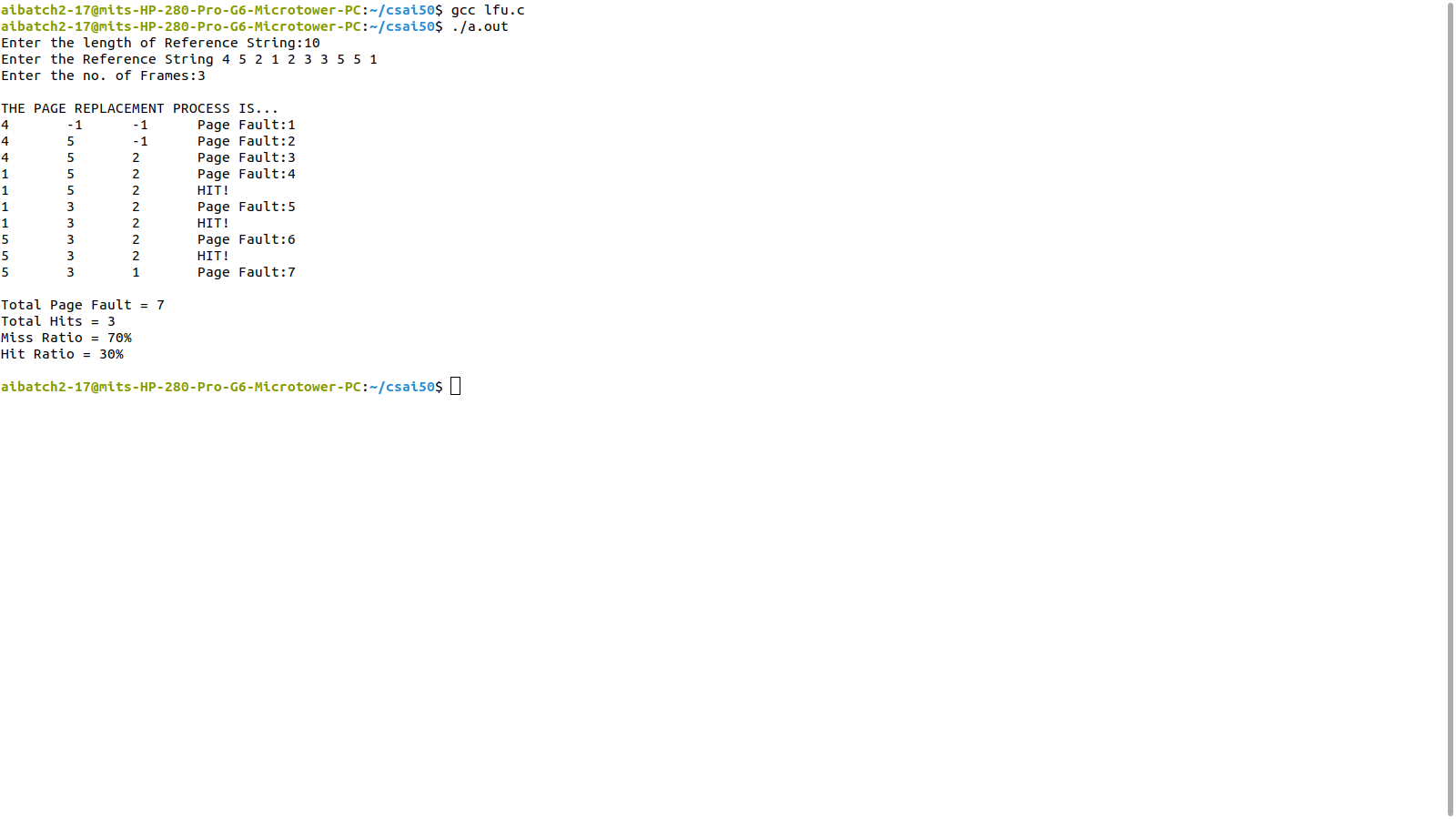
printf("Total Hits = %d\n",len-pf);

printf("Miss Ratio = %d%%\n",(pf\*100)/len);

printf("Hit Ratio = %d%%\n\n",((len-pf)\*100)/len);

}

**OUTPUT**

****

# PROGRAM: FCFS DISC SCHEDULING

#include<stdio.h> #include<stdlib.h> void main() { int n,q[100],i,diff,seek=0; printf("Enter the size of Queue: "); scanf("%d",&n); printf("Enter the Queue: "); for(i=1;i<=n;i++) {

scanf("%d",&q[i]);

}

printf("Enter the intial head position: "); scanf("%d",&q[0]); for(i=0;i<n;i++) { diff=abs(q[i]-q[i+1]);

seek+=diff;

printf("\nMove from %d to %d and the seek is %d",q[i],q[i+1],diff);

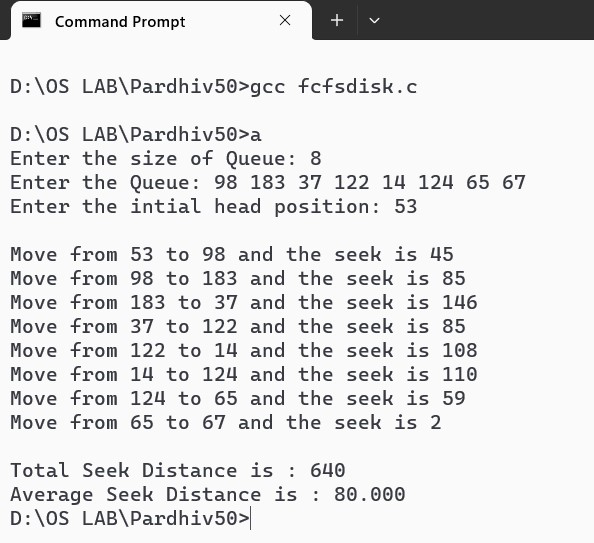
}

printf("\n\nTotal Seek Distance is : %d",seek); float avg=seek/n;

printf("\nAverage Seek Distance is : %.3f",avg);

}

# OUTPUT



# PROGRAM: SCAN DISC SCHEDULING

#include<stdio.h> #include<stdlib.h> void main() {

int q[100],n,seek=0,i,cur,prev,j,m,cyl,loc; float avg;

printf("Enter the no. of Cylinders: "); scanf("%d",&cyl); printf("Cylinders: 0 to %d\n",cyl-1); printf("Enter the Queue Size: "); scanf("%d",&m); n=m+1; printf("Enter the Queue: "); for(i=1;i<n;i++) scanf("%d",&q[i]); printf("Enter Current Head Position: "); scanf("%d",&cur); q[0]=cur;

printf("Enter Previous Head Position: "); scanf("%d",&prev); for(i=1;i<n;i++) for(j=0;j<n-i;j++) if(q[j]>q[j+1]) { int temp=q[j]; q[j]=q[j+1];

q[j+1]=temp;

}

printf("Displaying Requests in Order...\n");

for(i=0;i<n;i++) printf("%d\t",q[i]); for(i=0;i<n;i++) if(q[i]==cur) { loc=i;

break;

}

if(cur<prev) {

printf("\n\nScanning towards left...then right\n");

for(i=loc;i>=0;i--)

printf("%d --> ",q[i]); printf("0 --> "); for(i=loc+1;i<n;i++)

printf("%d --> ",q[i]); seek=cur+q[n-1];

} else {

printf("\n\nScanning towards right...then left\n");

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

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

printf("%d --> ",cyl-1);

for(i=loc-1;i>=0;i--)

printf("%d --> ",q[i]); seek=2\*(cyl-1)-cur-q[0];

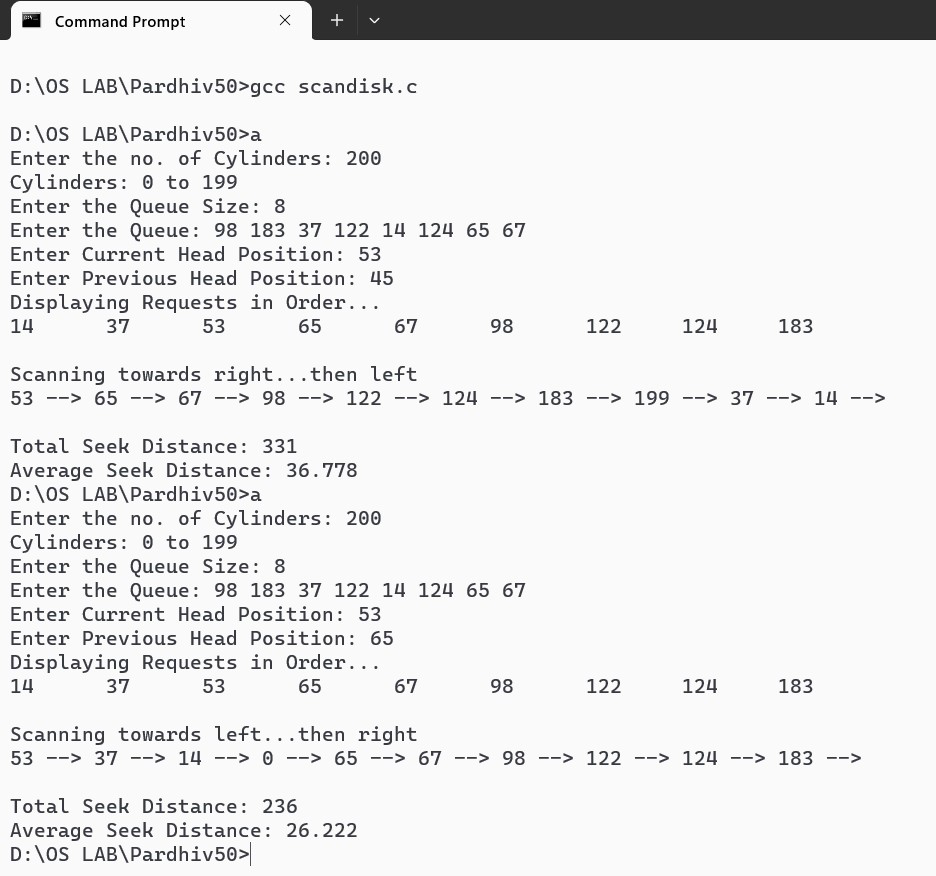
}

printf("\n\nTotal Seek Distance: %d\t",seek); avg=(float)seek/n;

printf("\nAverage Seek Distance: %.3f\t",avg);

}

# OUTPUT



# PROGRAM: CSCAN DISC SCHEDULING

#include<stdio.h> #include<stdlib.h> void main() {

int q[100],n,seek=0,i,cur,prev,j,m,cyl,loc; float avg;

printf("Enter the no. of Cylinders: "); scanf("%d",&cyl); printf("Cylinders: 0 to %d\n",cyl-1); printf("Enter the Queue Size: "); scanf("%d",&m); n=m+1;

printf("Enter the Queue: ");

for(i=1;i<n;i++) scanf("%d",&q[i]); printf("Enter Current Head Position: "); scanf("%d",&cur); q[0]=cur;

printf("Enter Previous Head Position: "); scanf("%d",&prev); for(i=1;i<n;i++) for(j=0;j<n-i;j++) if(q[j]>q[j+1]) { int temp=q[j]; q[j]=q[j+1];

q[j+1]=temp;

}

printf("Displaying Requests in Order...\n");

for(i=0;i<n;i++) printf("%d\t",q[i]); for(i=0;i<n;i++) if(q[i]==cur) { loc=i;

break;

}

if(cur<prev) {

printf("\n\nScanning towards left...then restart at right end\n"); for(i=loc;i>=0;i--)

printf("%d --> ",q[i]); printf("0 --> "); printf("%d --> ",cyl-1);

for(i=n-1;i>loc;i--)

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

seek=cur+2\*(cyl-1)-q[loc+1];

} else {

printf("\n\nScanning towards right...then restart at left end\n"); for(i=loc;i<n;i++)

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

printf("%d --> ",cyl-1);

printf("0 --> ");

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

printf("%d --> ",q[i]); seek=2\*(cyl-1)-cur+q[loc-1];

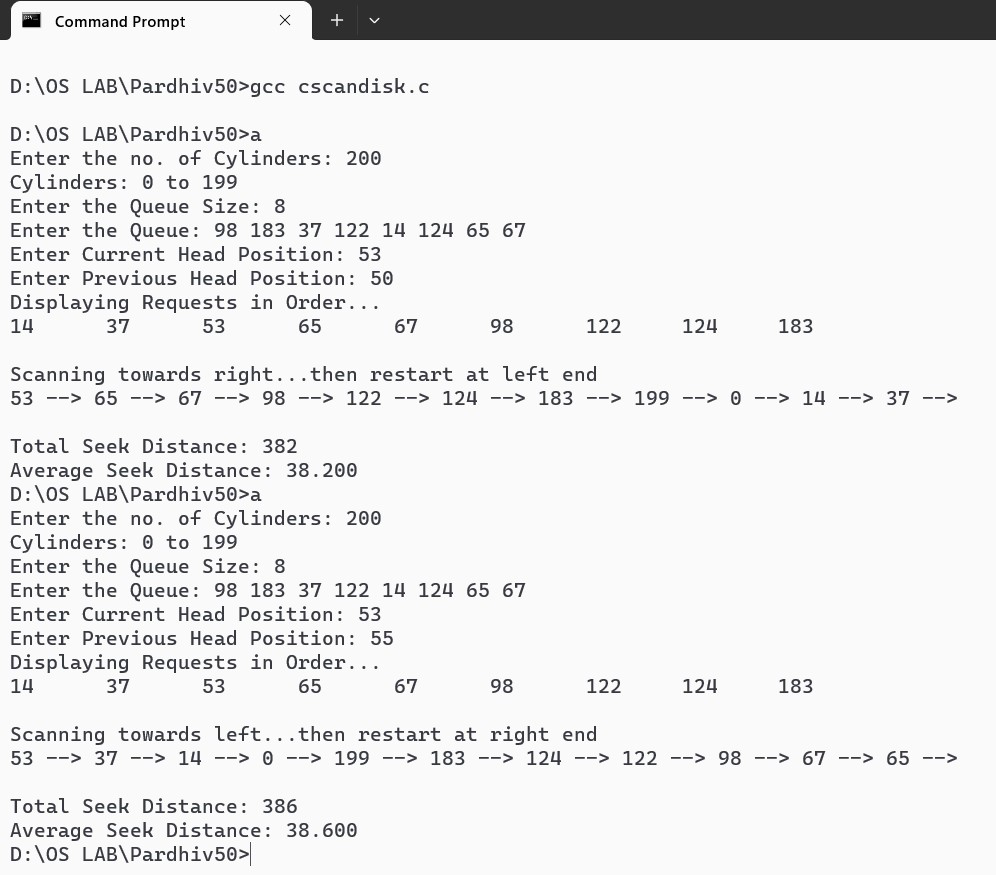
}

printf("\n\nTotal Seek Distance: %d\t",seek); avg=(float)seek/(n+1);

printf("\nAverage Seek Distance: %.3f\t",avg);

}

# OUTPUT



**PROGRAM: SEQUENTIAL FILE ALLOCATION**

#include<stdio.h>

#include<string.h>

struct file {

char name[10];

int num,start;

} a[20];

void main() {

int n,i,j;

char nam[10];

printf("Enter the no. of files: ");

scanf("%d",&n);

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

printf("\nEnter the name of file %d: ",i+1);

scanf("%s",a[i].name);

printf("Enter the starting block of file %s: ",a[i].name);

scanf("%d",&a[i].start);

printf("Enter the no. of blocks in file %s: ",a[i].name);

scanf("%d",&a[i].num);

}

printf("\nEnter the name of file to be searched: ");

scanf("%s",nam);

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

if(strcmp(nam,a[i].name)==0)

break;

printf("\nFile Name\tStart Block\tNo. of Blocks\tBlocks Occupied\n");

printf("%s\t\t%d\t\t%d\t\t",a[i].name,a[i].start,a[i].num);

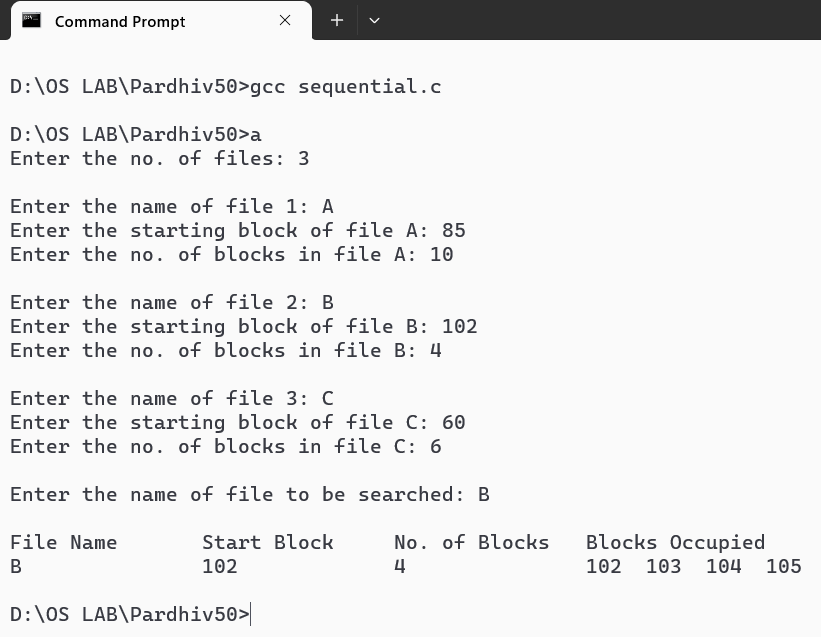
for(j=0;j<a[i].num;j++)

printf("%d ",a[i].start++);

printf("\n");

}

**OUTPUT**

****

**PROGRAM: INDEXED FILE ALLOCATION**

#include<stdio.h>

#include<string.h>

struct file {

char name[10];

int num,block[20];

} a[20];

void main() {

int n,i,j;

char nam[10];

printf("Enter the no. of files: ");

scanf("%d",&n);

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

printf("\nEnter the name of file %d: ",i+1);

scanf("%s",a[i].name);

printf("Enter the no. of blocks in file %s: ",a[i].name);

scanf("%d",&a[i].num);

printf("Enter the blocks in file %s: ",a[i].name);

for(j=0;j<a[i].num;j++)

scanf("%d",&a[i].block[j]);

}

printf("\nEnter the name of file to be searched: ");

scanf("%s",nam);

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

if(strcmp(nam,a[i].name)==0)

break;

printf("\nFile Name\tNo. of Blocks\tBlocks Occupied\n");

printf("%s\t\t%d\t\t",a[i].name,a[i].num);

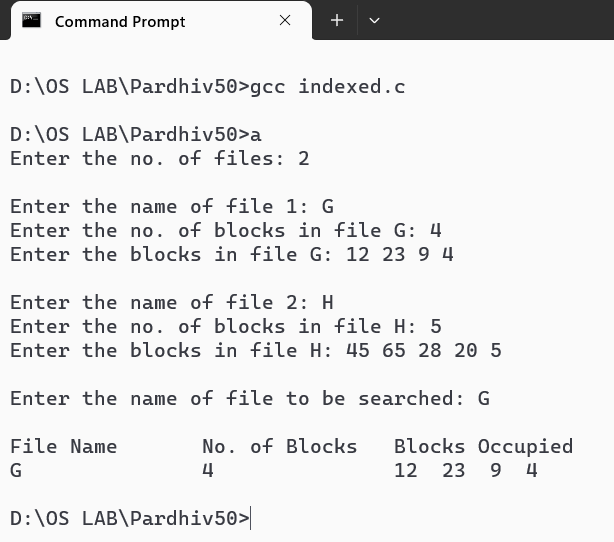
for(j=0;j<a[i].num;j++)

printf("%d ",a[i].block[j]);

printf("\n");

}

**OUTPUT**

****

**PROGRAM: LINKED FILE ALLOCATION**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

struct block {

int n;

struct block \*next;

};

struct file {

char name[10];

int num;

struct block \*temp,\*node;

} a[20];

void main() {

struct block \*head[20];

int n,i,j;

char nam[10];

printf("Enter the no. of files: ");

scanf("%d",&n);

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

head[i]=NULL;

printf("\nEnter the name of file %d: ",i+1);

scanf("%s",a[i].name);

printf("Enter the no. of blocks in file %s: ",a[i].name);

scanf("%d",&a[i].num);

printf("Enter the blocks in file %s: ",a[i].name);

for(j=0;j<a[i].num;j++) {

a[i].node=(struct block\*)malloc(sizeof(struct block));

scanf("%d",&a[i].node->n);

if(head[i]==NULL) {

head[i]=a[i].temp=a[i].node;

a[i].temp->next=NULL;

}

else {

a[i].temp->next=a[i].node;

a[i].temp=a[i].temp->next;

a[i].temp->next=NULL;

}

}

}

printf("\nEnter the name of file to be searched: ");

scanf("%s",nam);

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

if(strcmp(nam,a[i].name)==0)

break;

printf("\nFile Name\tNo. of Blocks\tBlocks Occupied\n");

printf("%s\t\t%d\t\t",a[i].name,a[i].num);

a[i].temp=head[i];

while(a[i].temp!=NULL) {

printf("%d -> ",a[i].temp->n);

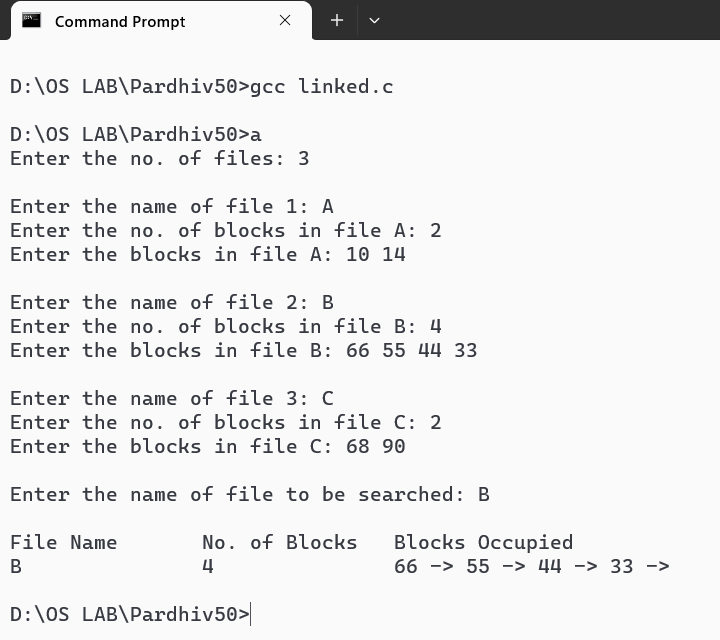
a[i].temp=a[i].temp->next;

}

printf("\n");

}

**OUTPUT**

****