|  |  |  |
| --- | --- | --- |
| **S.No** | **NAME OF TE EXPERIMENT** | **PAGE NO** |
| 1 | Shell Program -Check the given number is Odd or Even | 2 |
| 2 | Shell Program - Check the given Year Leap Year or Not | 3 |
| 3 | Shell Program - Factorial of a Number | 4 |
| 4 | Shell Program - Sum of the first 100 numbers | 5 |
| 5 | Shell Program – Greatest among 3 numbers | 6 |
| 6 | System Calls | 7 |
| 7 | File Copy | 8-9 |
| 8 | Scheduling First Come First Serve (FCFS) | 10-11 |
| 9 | Scheduling Shortest Job First (SJF) | 12-13 |
| 10 | Scheduling Priority | 14-15 |
| 11 | Scheduling - Round Robin | 16-17 |
| 12 | Producer and Consumer Problem | 18-19 |
| 13 | Dining and Philosopher’s Problem | 20-22 |
| 14 | Banker’s algorithm | 23-25 |
| 15 | Deadlock Avoidance | 26-28 |
| 16 | Deadlock Prevention | 29-31 |
| 17 | Memory Allocation First-Fit | 32-34 |
| 18 | Memory Allocation Best-Fit | 35-36 |
| 19 | Memory Allocation Worst-Fit | 37-39 |
| 20 | Page Replacement -FIFO | 40-42 |
| 21 | Page Replacement -LRU | 43-45 |
| 22 | Page Replacement -Optimal | 46-49 |
| 23 | Disk Scheduling Algorithms-FCFS | 50-51 |
| 24 | Disk Scheduling Algorithms-Scan | 52-54 |
| 25 | Disk Scheduling Algorithms-Cscan | 55-57 |
| 26 | File Allocation-Sequential | 58-59 |
| 27 | File Allocation-Indexed | 60-61 |
| 28 | File Allocation-Linked | 62-63 |
| 29 | File Organization-single level | 64-66 |
| 30 | File Organization-two level | 67-70 |

**1. To check the given number is Odd or Even**

**Program:**

#!/bin/bash

echo "enter a number:"

read num

if((num%2==0));

then

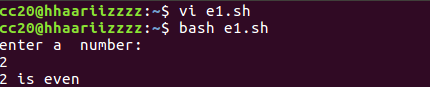
echo "$num is even"

else

echo "$num is odd"

fi

**Output:**



**2. To Check the given Year Leap Year or Not**

**Program:**

#!/bin/bash

echo "enter a year:"

read year

if((year%4==0));

then

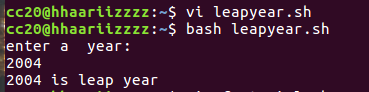
echo "$year is leap year"

else

echo "$year is not a leap year"

fi

**Output:**



**3. Factorial of a Number**

**Program:**

#!/bin/bash

echo "enter a number:"

read num

factorial=1

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

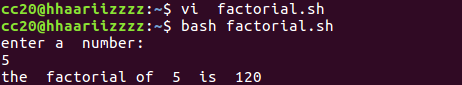
do

factorial=$((factorial\*i))

done

echo "the factorial of $num is $factorial"

output:



**4. Sum of numbers upto 100**

**Program:**

#!/bin/bash

sum=0

for((i=1;i<=100;i++))

do

sum=$((sum+i))

done

echo "the sum of the first 100 natural number is $sum"

Output:



**5.** G**reatest among 3 numbers**

**Program:**

#!/bin/bash

# Prompt the user to enter three numbers

echo "Enter three numbers:"

read num1 num2 num3

# Compare the numbers to find the greatest

if (( num1 > num2 && num1 > num3 ));

# num1 is the greatest

then echo "$num1 is the greatest number"

elif (( num2 > num1 && num2 > num3 ));

# num2 is the greatest

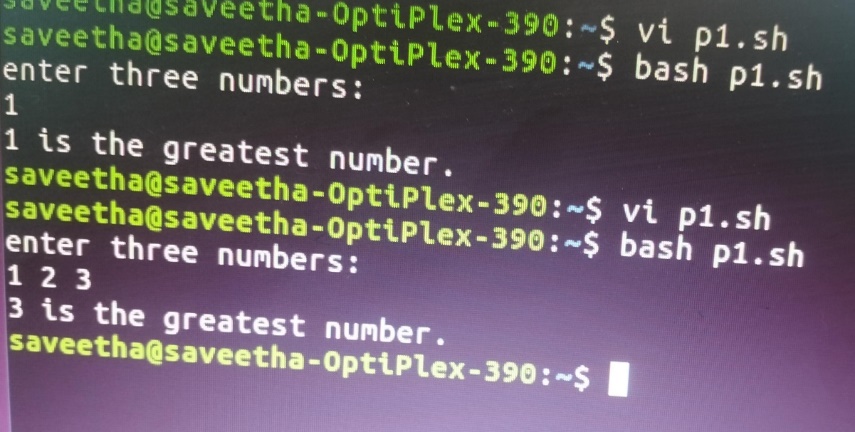
then echo "$num2 is the greatest number"

else

# num3 is the greatest

echo "$num3 is the greatest number"

fi

****

**6. System Calls**

**Program:**

#include <stdio.h>

#include <unistd.h>

int main()

{

int p\_id, p\_pid;

p\_id = getpid(); /\*process id\*/

p\_pid = getpid(); /\*parent process id\*/

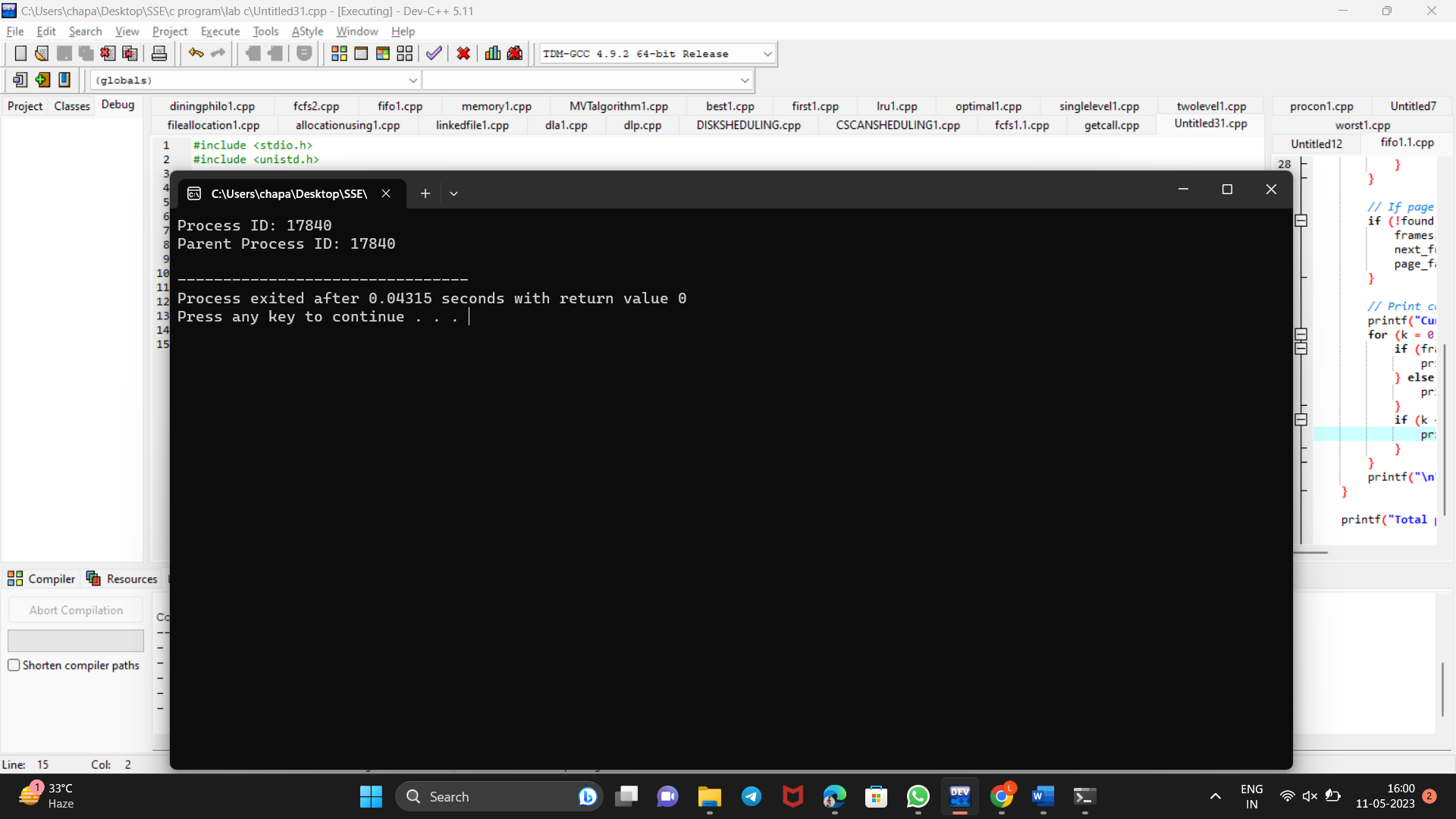
printf("Process ID: %d\n", p\_id);

printf("Parent Process ID: %d\n", p\_pid);

return 0;

}

**Output**



**7.File Copy**

**program**

#include <stdio.h>

#include <stdlib.h>

int main()

{

FILE \*fptr1, \*fptr2;

char filename[100], c;

printf("Enter the filename to open for reading \n");

scanf("%s", filename);

fptr1 = fopen(filename, "r");

if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

printf("Enter the filename to open for writing \n");

scanf("%s", filename);

fptr2 = fopen(filename, "w");

if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

c = fgetc(fptr1);

while (c != EOF)

{

fputc(c, fptr2);

c = fgetc(fptr1);

}

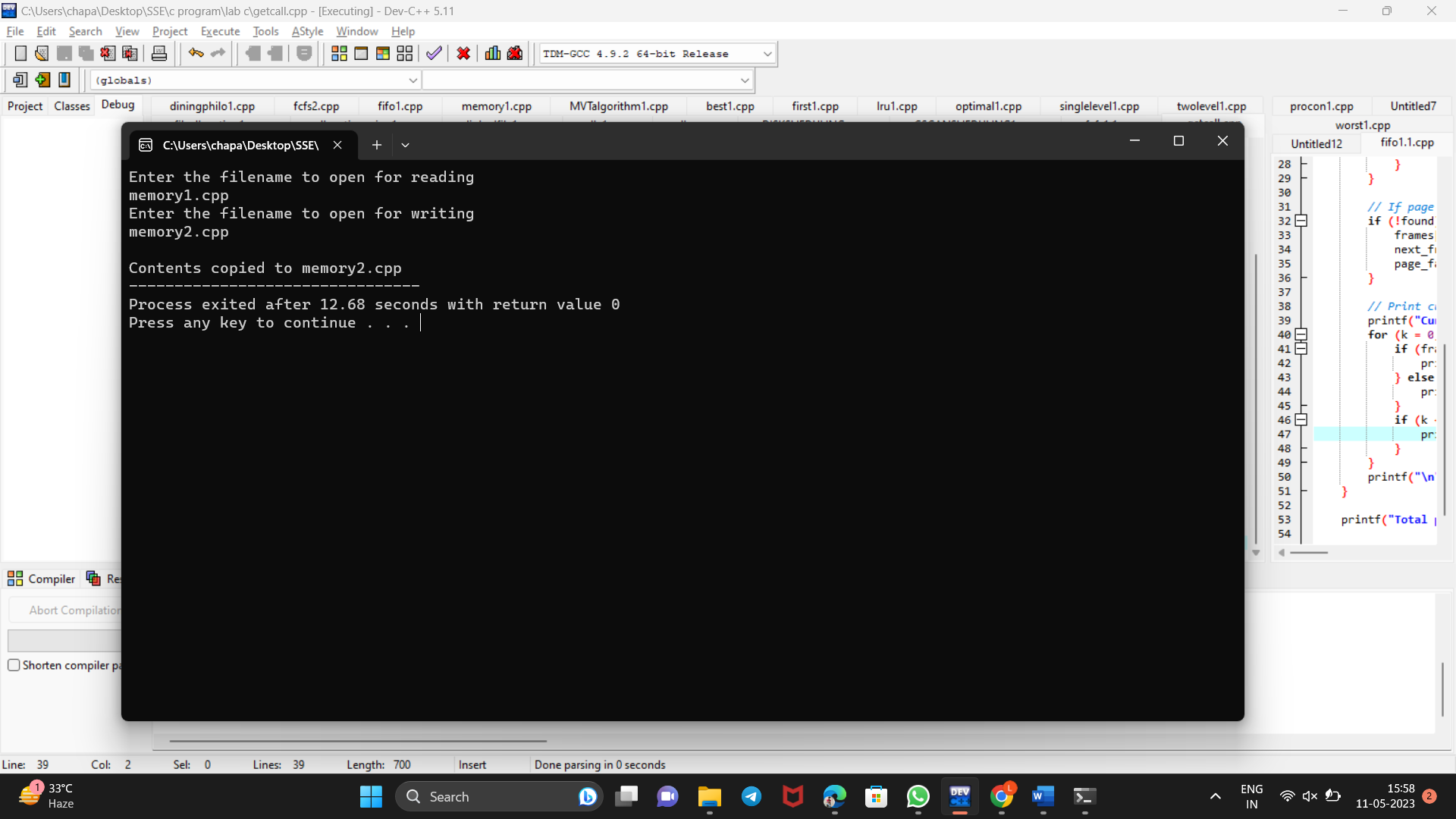
printf("\nContents copied to %s", filename);

fclose(fptr1);

fclose(fptr2);

return 0;

Output



**8. FIRST COME FIRST SERVE SCHEDULING ALGORITHM (FCFS)**

**PROGRAM**

#include<stdio.h>

#include<conio.h>

int main()

{

int bt[20], wt[20], tat[20], i, n; float wtavg, tatavg;

clrscr();

printf("\nEnter the number of processes -- ");

scanf("%d", &n);

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

{

printf("\nEnter Burst Time for Process %d -- ", i);

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

}

wt[0] = wtavg = 0;

tat[0] = tatavg = bt[0];

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

{

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

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

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n"); for(i=0;i<n;i++)

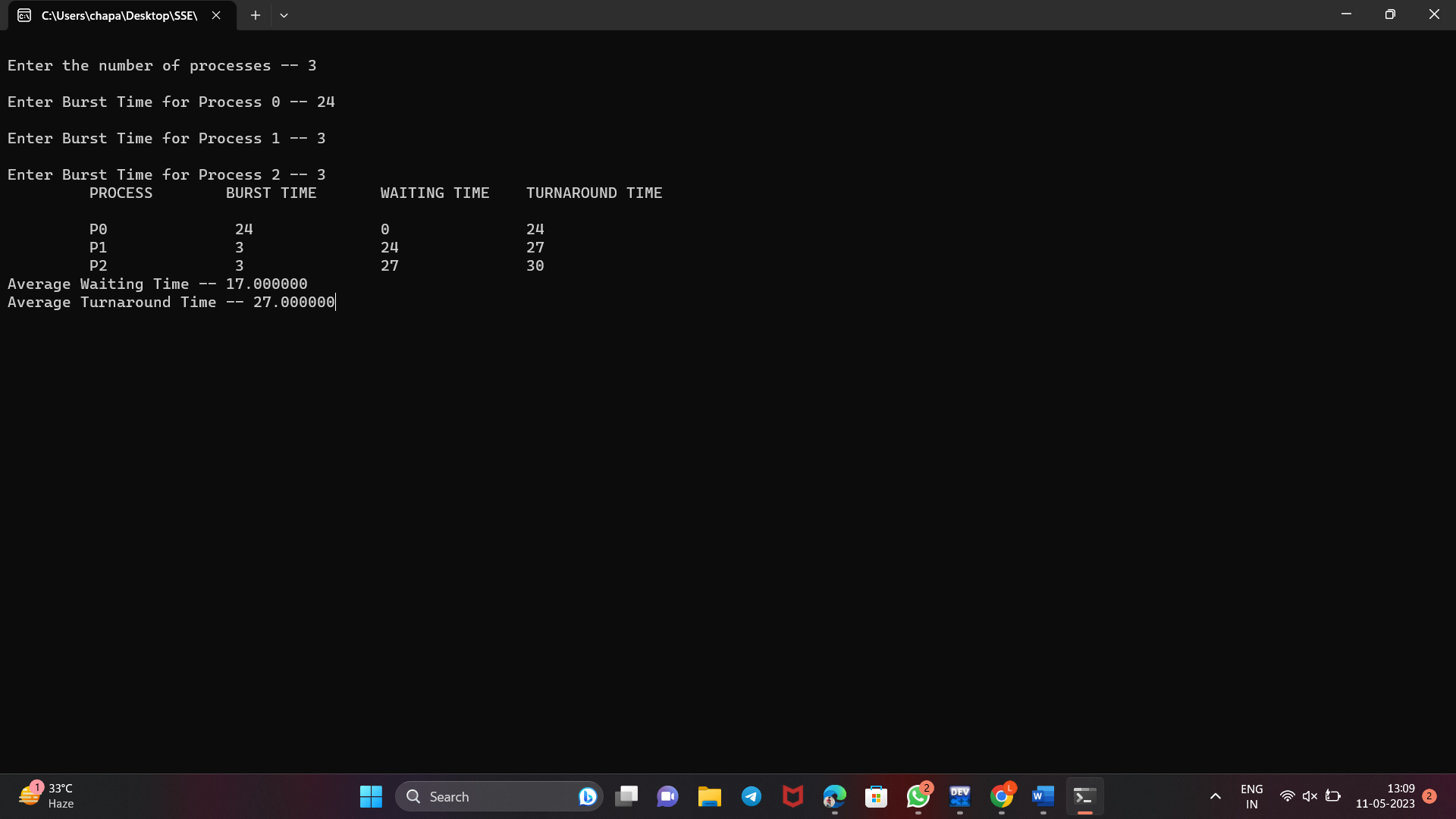
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);

printf("\nAverage Waiting Time -- %f", wtavg/n); printf("\nAverage Turnaround Time -- %f", tatavg/n);

getch();

}

Output



**9.SHORTEST JOB FIRST SCHEDULING ALGORITHM (SJF)**

**Program**

#include<stdio.h>

#include<conio.h>

Int main()

{

int p[20], bt[20], wt[20], tat[20], i, k, n, temp; float wtavg, tatavg;

//clrscr();

printf("\nEnter the number of processes -- ");

scanf("%d", &n);

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

{

p[i]=i;

printf("Enter Burst Time for Process %d -- ", i);

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

}

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

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

if(bt[i]>bt[k])

{

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=p[i];

p[i]=p[k];

p[k]=temp;

}

wt[0] = wtavg = 0;

tat[0] = tatavg = bt[0]; for(i=1;i<n;i++)

{

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

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

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

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

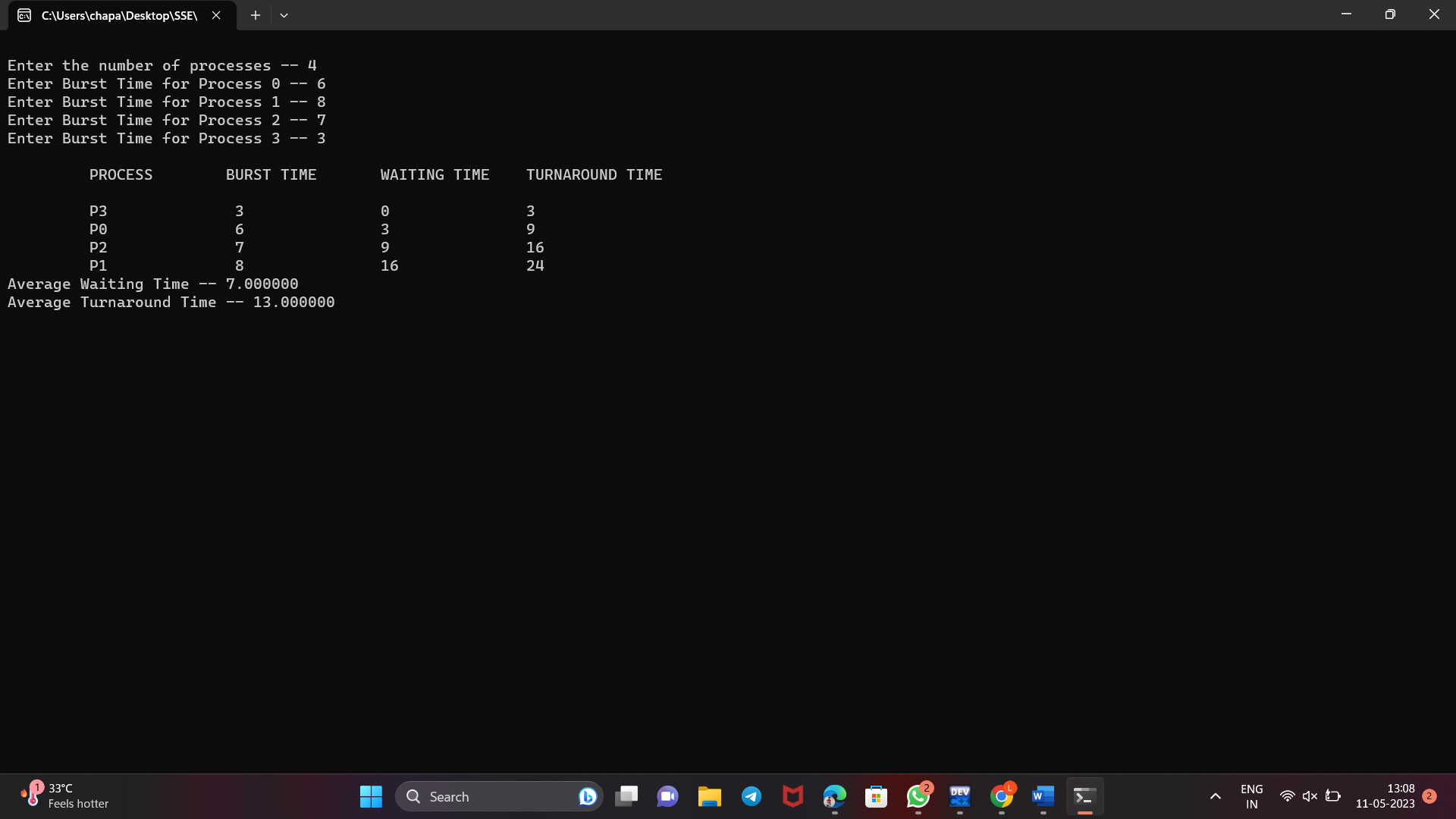
printf("\nAverage Waiting Time -- %f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n);

getch();

}

**OUTPUT**



**10.PRIORITY SCHEDULING ALGORITHM**

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp; float wtavg, tatavg;

//clrscr();

printf("Enter the number of processes --- ");

scanf("%d",&n);

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

p[i] = i;

printf("Enter the Burst Time & Priority of Process %d --- ",i); scanf("%d%d",&bt[i], &pri[i]);

}

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

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

if(pri[i] > pri[k]){

temp=p[i];

p[i]=p[k];

p[k]=temp;

temp=bt[i];

bt[i]=bt[k];

bt[k]=temp;

temp=pri[i];

pri[i]=pri[k];

pri[k]=temp;

}

wtavg = wt[0] = 0;

tatavg = tat[0] = bt[0];

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

{

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

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

wtavg = wtavg + wt[i];

tatavg = tatavg + tat[i];

}

printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");

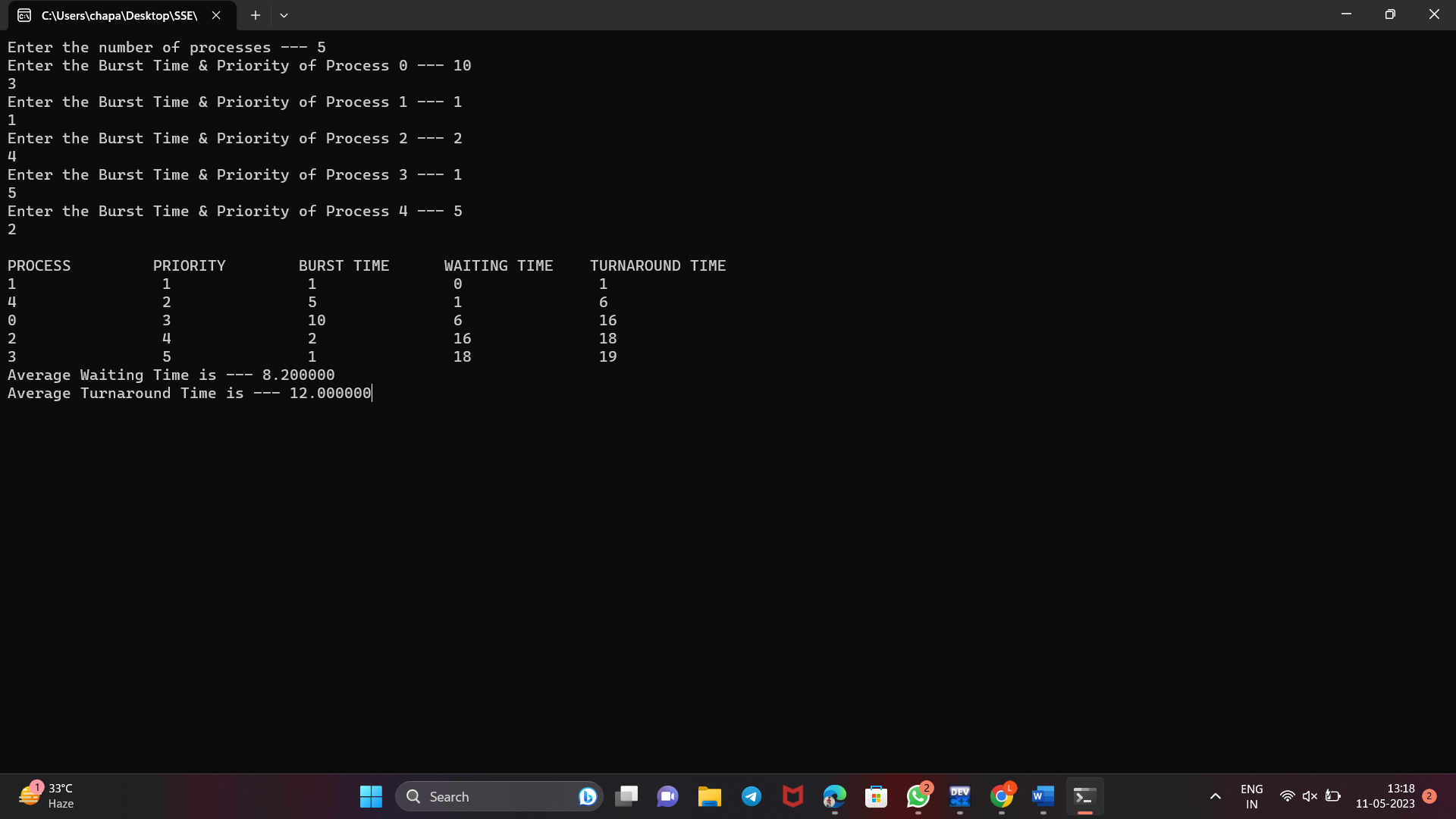
for(i=0;i<n;i++)

printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]); printf("\nAverage Waiting Time is --- %f",wtavg/n); printf("\nAverage Turnaround Time is --- %f",tatavg/n);

getch();

}

**OUTPUT**



**11.ROUND ROBIN SCHEDULING ALGORITHM**

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;

float awt=0,att=0,temp=0;

//clrscr();

printf("Enter the no of processes -- ");

scanf("%d",&n);

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

{

printf("\nEnter Burst Time for process %d -- ", i+1);

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

ct[i]=bu[i];

}

printf("\nEnter the size of time slice -- ");

scanf("%d",&t);

max=bu[0];

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

if(max<bu[i])

max=bu[i];

for(j=0;j<(max/t)+1;j++)

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

if(bu[i]!=0)

if(bu[i]<=t) {

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

temp=temp+bu[i];

bu[i]=0;

}

else {

bu[i]=bu[i]-t;

temp=temp+t;

}

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

wa[i]=tat[i]-

ct[i]; att+=tat[i];

awt+=wa[i];}

printf("\nThe Average Turnaround time is -- %f",att/n);

printf("\nThe Average Waiting time is -- %f ",awt/n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");

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

printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);

getch();

}**OUTPUT**



**12.simulate producer-consumer problem**

**Program**

#include<stdio.h>

main()

{

int buffer[10], bufsize, in, out, produce, consume, choice=0; in = 0,out = 0;

bufsize = 10;

while(choice !=3)

{

printf("\n1. Produce \t 2. Consume \t3. Exit");

printf( "\nEnter your choice:");

scanf("%d",&choice);

switch(choice) {

case 1: if((in+1)%bufsize==out)

printf("\nBuffer is Full");

else

{

}

break;;;

printf("\nEnter the value: ");

scanf("%d", &produce);

buffer[in] = produce;

in = (in+1)%bufsize;

case 2: if(in == out)

printf("\nBuffer is Empty");

else

{

consume = buffer[out];

printf("\nThe consumed value is %d”, consume");

out = (out+1)%bufsize;

}

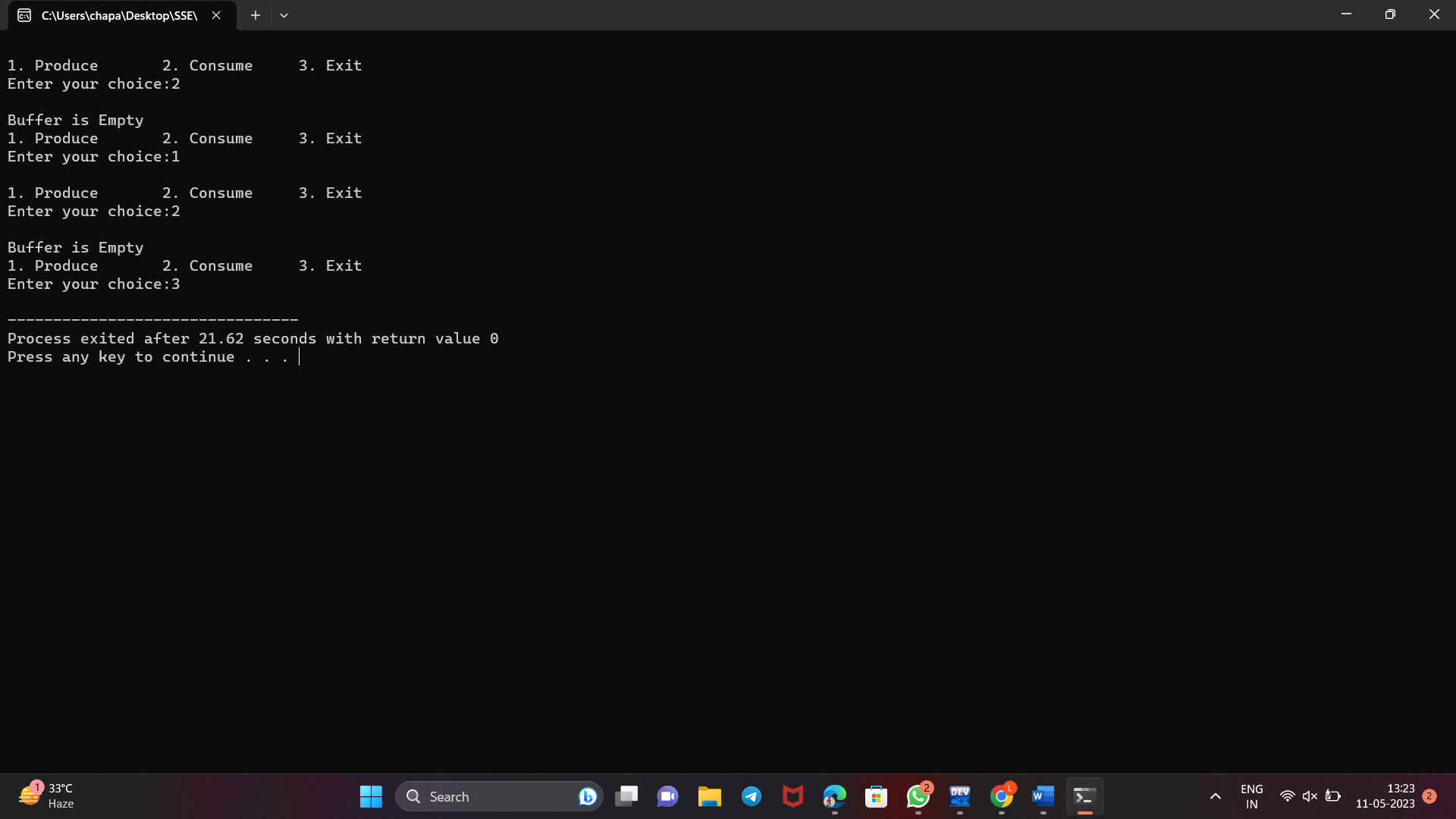
break;

}

}

}

**OUTPUT**



**13.Dining-Philosophers**

**Program**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t room;

sem\_t chopstick[5];

void \* philosopher(void \*);

void eat(int);

int main()

{

int i,a[5];

pthread\_t tid[5];

sem\_init(&room,0,4);

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

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

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

a[i]=i;

pthread\_create(&tid[i],NULL,philosopher,(void \*)&a[i]);

}

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

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

}

void \* philosopher(void \* num)

{

int phil=\*(int \*)num;

sem\_wait(&room);

printf("\nPhilosopher %d has entered room",phil);

sem\_wait(&chopstick[phil]);

sem\_wait(&chopstick[(phil+1)%5]);

eat(phil);

sleep(2);

printf("\nPhilosopher %d has finished eating",phil);

sem\_post(&chopstick[(phil+1)%5]);

sem\_post(&chopstick[phil]);

sem\_post(&room);

}

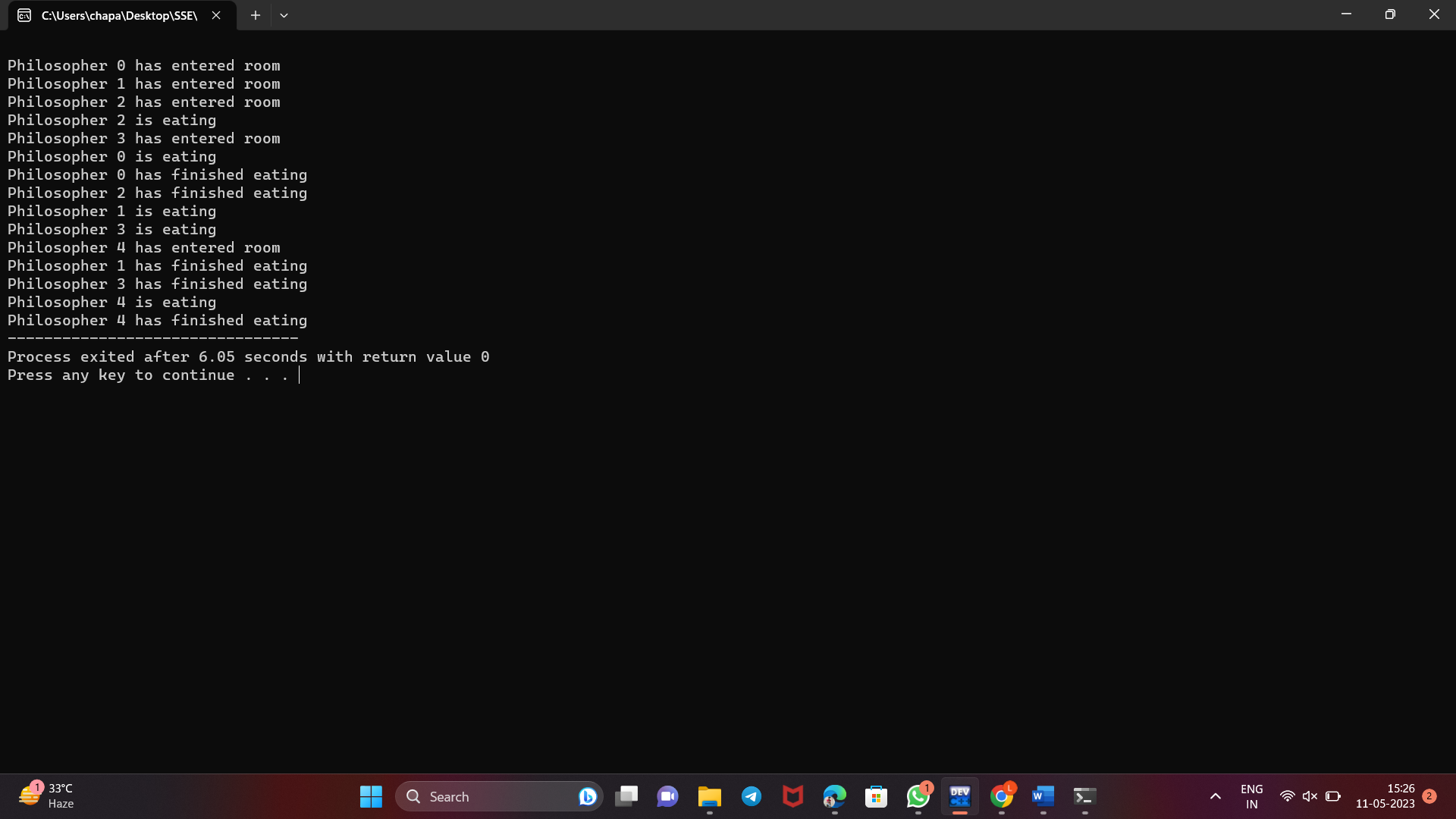
void eat(int phil)

{

printf("\nPhilosopher %d is eating",phil);

}

**Output**



**14.Bankers Algorithm**

**Program**

#include<stdio.h>

int main() {

int p, c, count = 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5], available[3], done[5], terminate = 0;

printf("Enter the number of process and resources");

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

printf("enter allocation of resource of all process %dx%d matrix", p, c);

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

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

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

}

}

printf("enter the max resource process required %dx%d matrix", p, c);

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

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

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

}

}

printf("enter the available resource");

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

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

printf("\n need resources matrix are\n");

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

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

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

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

}

printf("\n");

}

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

done[i] = 0;

}

while (count < p) {

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

if (done[i] == 0) {

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

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

break;

}

if (j == c) {

safe[count] = i;

done[i] = 1;

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

available[j] += alc[i][j];

}

count++;

terminate = 0;

} else {

terminate++;

}

}

}

if (terminate == (p - 1)) {

printf("safe sequence does not exist");

break;

}

}

if (terminate != (p - 1)) {

printf("\n available resource after completion\n");

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

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

}

printf("\n safe sequence are\n");

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

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

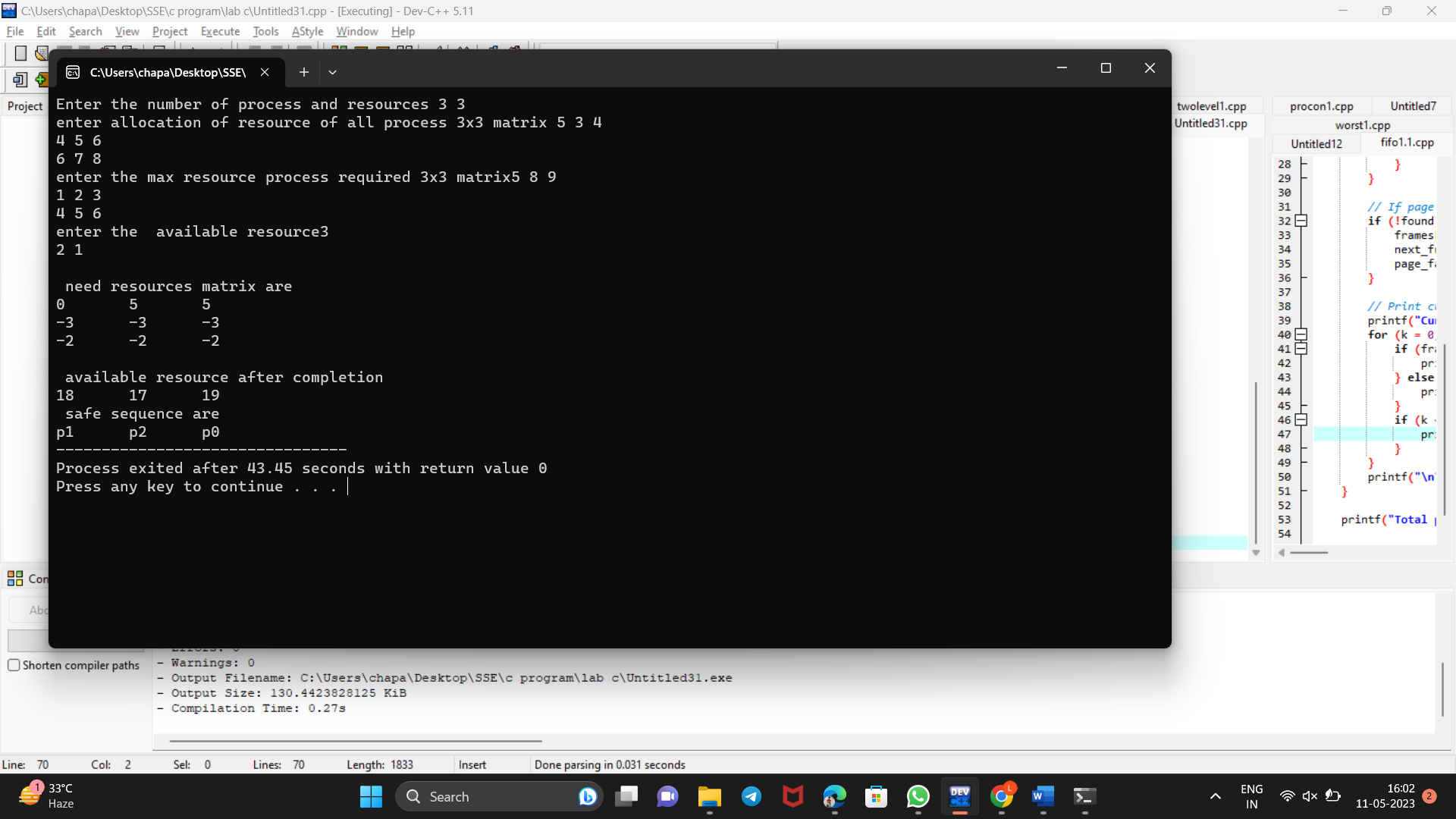
}

}

return 0;

}

**Output**



**15.Dead Lock Avoidance**

**Program**

#include<stdio.h>

#include<conio.h>

#include<string.h>

main()

{

int alloc[10][10],max[10][10];

int avail[10],work[10],total[10];

int i,j,k,n,need[10][10];

int m;

int count=0,c=0;

char finish[10];

//clrscr();

printf("Enter the no. of processes and resources:");

scanf("%d%d",&n,&m);

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

finish[i]='n';

printf("Enter the claim matrix:\n");

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

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

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

printf("Enter the allocation matrix:\n");

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

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

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

printf("Resource vector:");

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

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

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

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

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

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

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

work[i]=avail[i];

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

work[j]=total[j]-work[j];

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

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

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

A:

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

{

c=0;

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

if((need[i][j]<=work[j])&&(finish[i]=='n'))

c++;

if(c==m)

{

printf("All the resources can be allocated to Process %d", i+1);

printf("\n\nAvailable resources are:");

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

{

work[k]+=alloc[i][k];

printf("%4d",work[k]);

}

printf("\n");

finish[i]='y';

printf("\nProcess %d executed?:%c \n",i+1,finish[i]);

count++;

}

}

if(count!=n)

goto A;

else

printf("\n System is in safe mode");

printf("\n The given state is safe state");

getch();

}**OUTPUT**



**16.Deadlock prevention**

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

char job[10][10];

int time[10],avail,tem[10],temp[10]; int safe[10];

int ind=1,i,j,q,n,t;

//clrscr();

printf("Enter no of jobs: ");

scanf("%d",&n);

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

{

printf("Enter name and time: ");

scanf("%s%d",&job[i],&time[i]);

}

printf("Enter the available resources:");

scanf("%d",&avail);

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

{

temp[i]=time[i];

tem[i]=i;

}

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

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

{

if(temp[i]>temp[j])

{

t=temp[i];

temp[i]=temp[j];

temp[j]=t; t=tem[i];

tem[i]=tem[j];

tem[j]=t;

}

}

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

{

q=tem[i];

if(time[q]<=avail)

{

safe[ind]=tem[i];

avail=avail-tem[q];

printf("%s",job[safe[ind]]);

ind++;

}

else

{

printf("No safe sequence\n");

}

}

printf("Safe sequence is:");

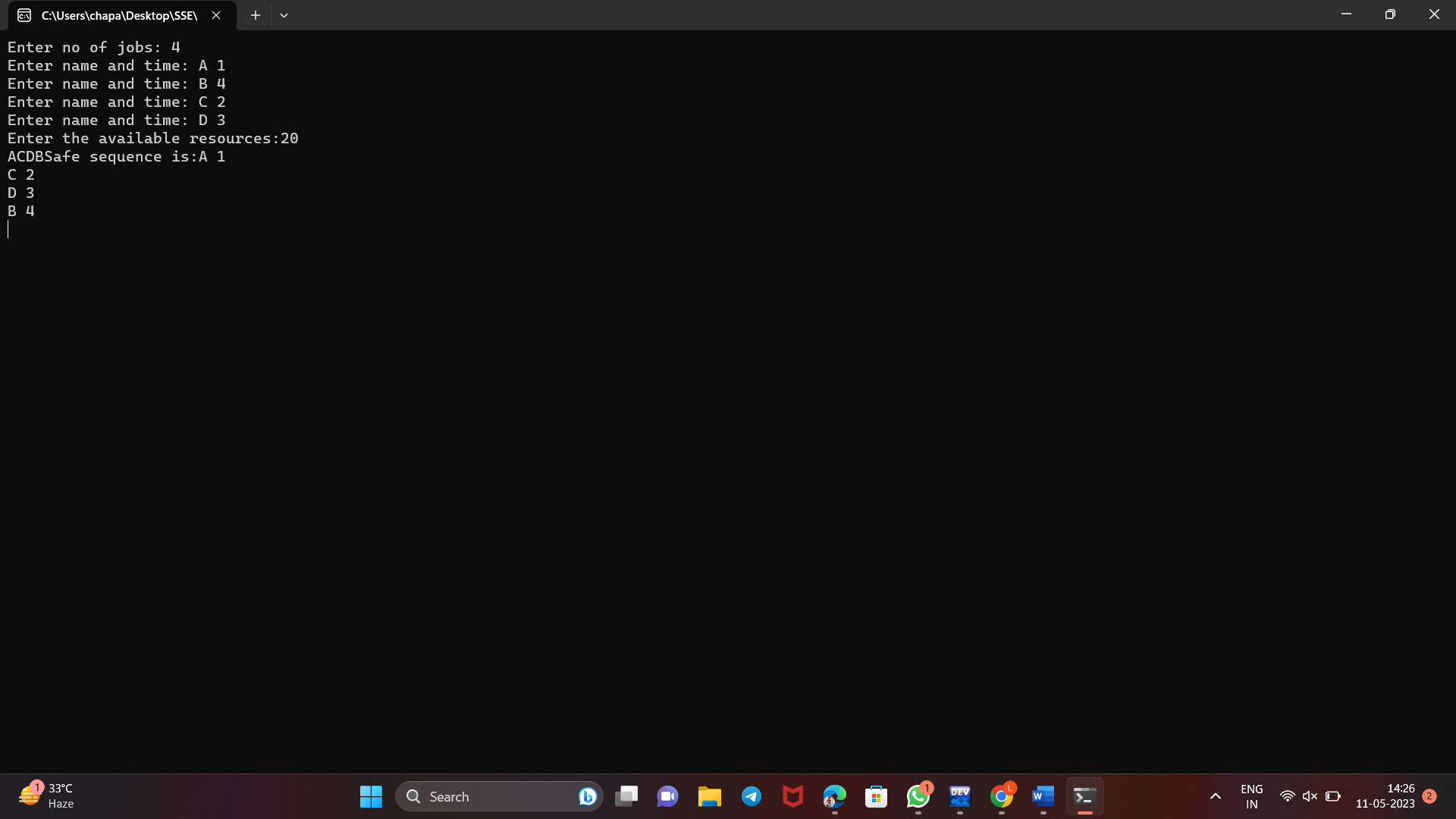
for(i=1;i<ind; i++)

printf("%s %d\n",job[safe[i]],time[safe[i]]);

getch();

}

**OUTPUT**



**17.FIRST-FIT**

**Program**

#include<stdio.h>

#include<conio.h>

#define max 25

main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;

static int bf[max],ff[max];

//clrscr();

printf("\n\tMemory Management Scheme - Worst Fit");

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

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

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

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

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

}

printf("Enter the size of the files :-\n");

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

{

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

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

}

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

{

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

{

if(bf[j]!=1) //if bf[j] is not allocated

{

temp=b[j]-f[i];

if(temp>=0)

if(highest<temp)

{

}

}

frag[i]=highest; bf[ff[i]]=1; highest=0;

}

ff[i]=j; highest=temp;

}

printf("\nFile\_no:\tFile\_size:\tBlock\_no:\tBlock\_size:\tFragement");

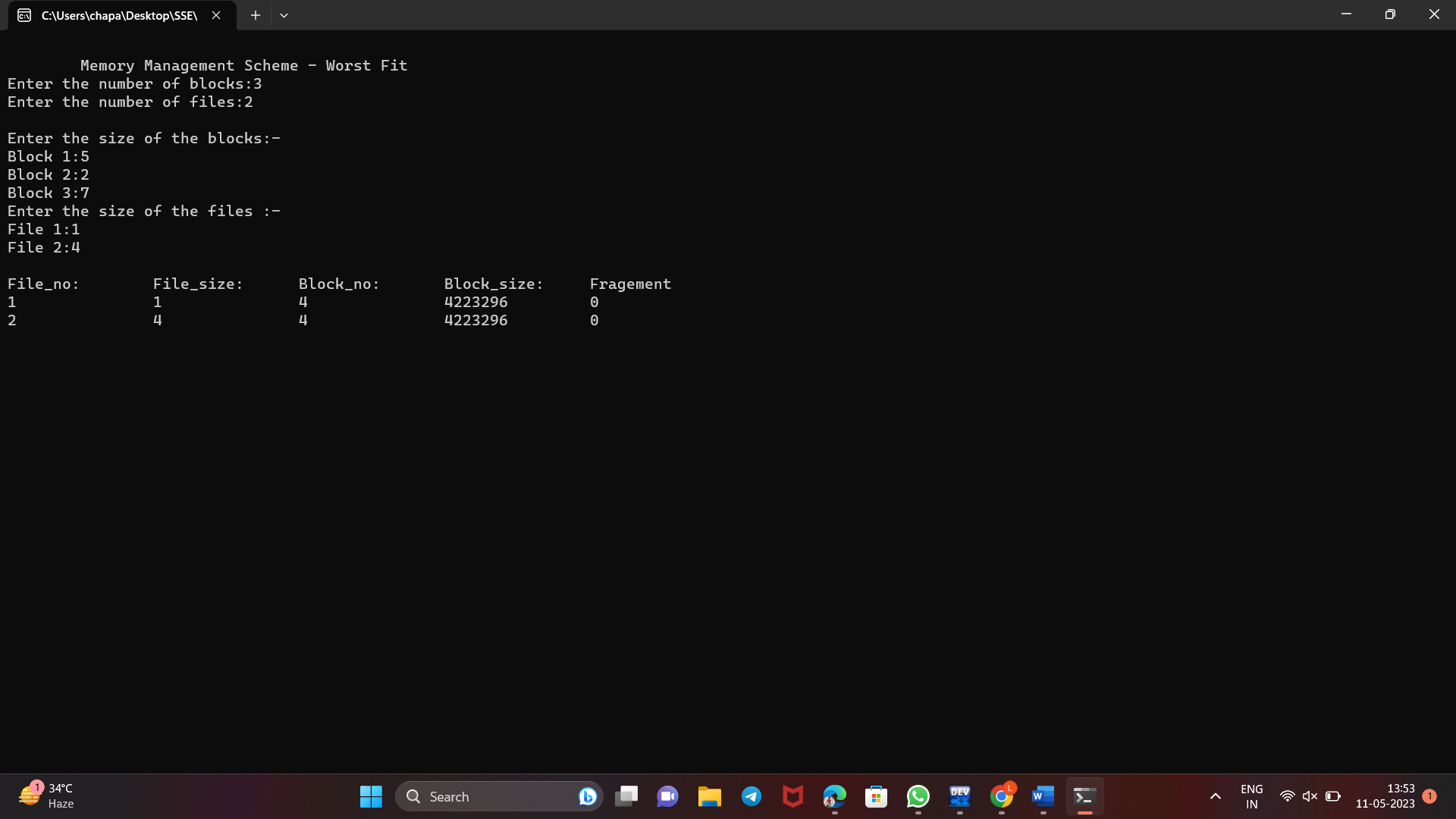
for(i=1;i<=nf;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}

**OUTPUT**



**18.BEST-FIT**

**Program**

#include<stdio.h>

#include<conio.h>

#define max 25

main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;

static int bf[max],ff[max];

//clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

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

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

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

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

}

printf("Enter the size of the files :-\n");

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

{

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

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

}

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

{

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

{

if(bf[j]!=1)

{

temp=b[j]-f[i];

if(temp>=0)

if(lowest>temp)

{

ff[i]=j;

lowest=temp;

}

}}

frag[i]=lowest; bf[ff[i]]=1; lowest=10000;

}

printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");

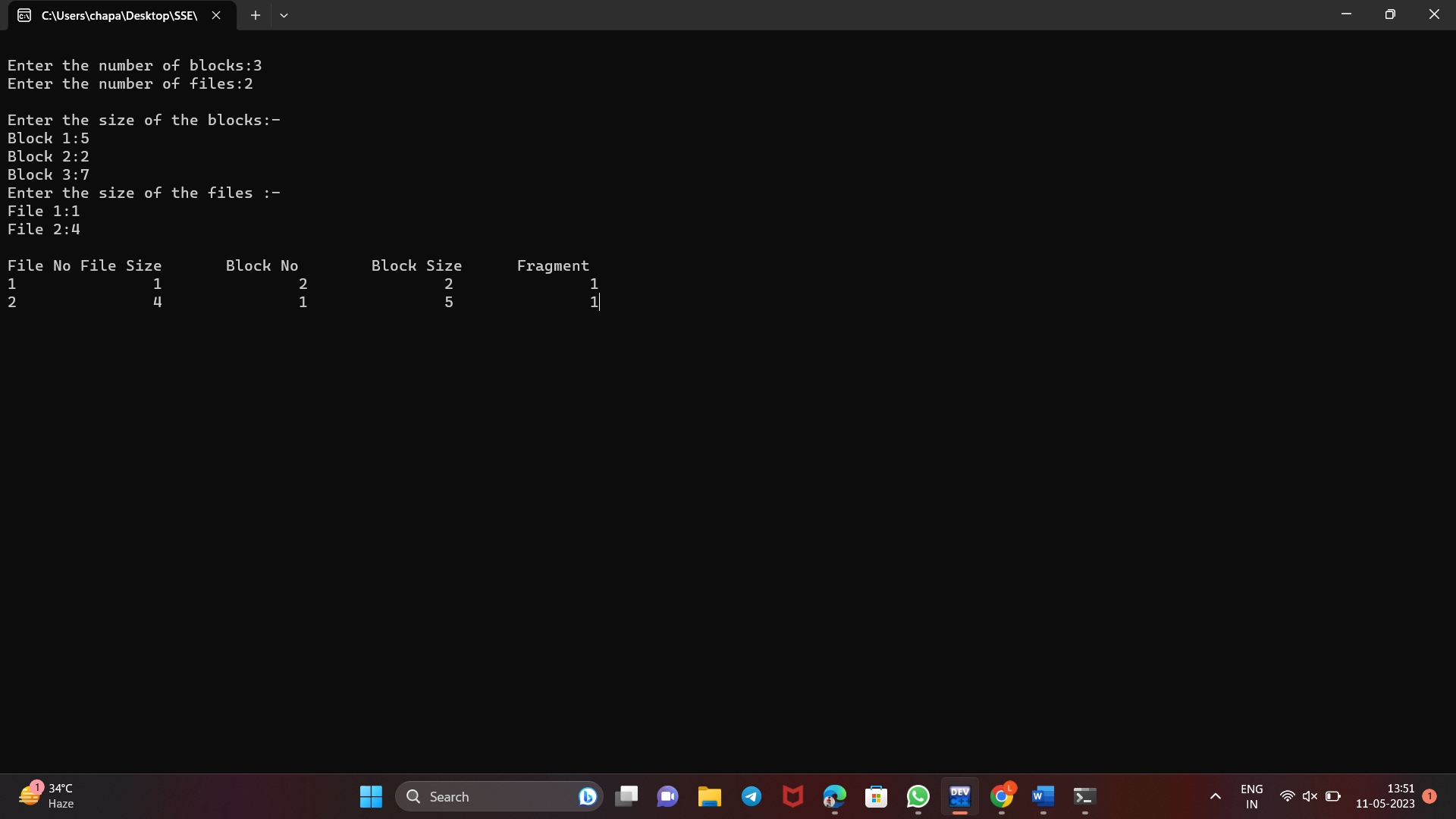
for(i=1;i<=nf && ff[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}

**OUTPUT**



**19.Worst-fit Algorithm**

**Program**

#include<stdio.h>

#include<conio.h>

#define max 25

main()

{

int

frag[max],b[max],f[max],i,j,nb,nf,temp;

static int bf[max],ff[max];

//clrscr();

printf("\n\tMemory Management Scheme - First Fit");

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

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

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

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

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

}

printf("Enter the size of the files :-\n");

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

{

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

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

}

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

{

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

{

if(bf[j]!=1)

{

temp=b[j]-f[i];

if(temp>=0)

{

ff[i]=j;

break;

}

}

}

frag[i]=temp;

bf[ff[i]]=1;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

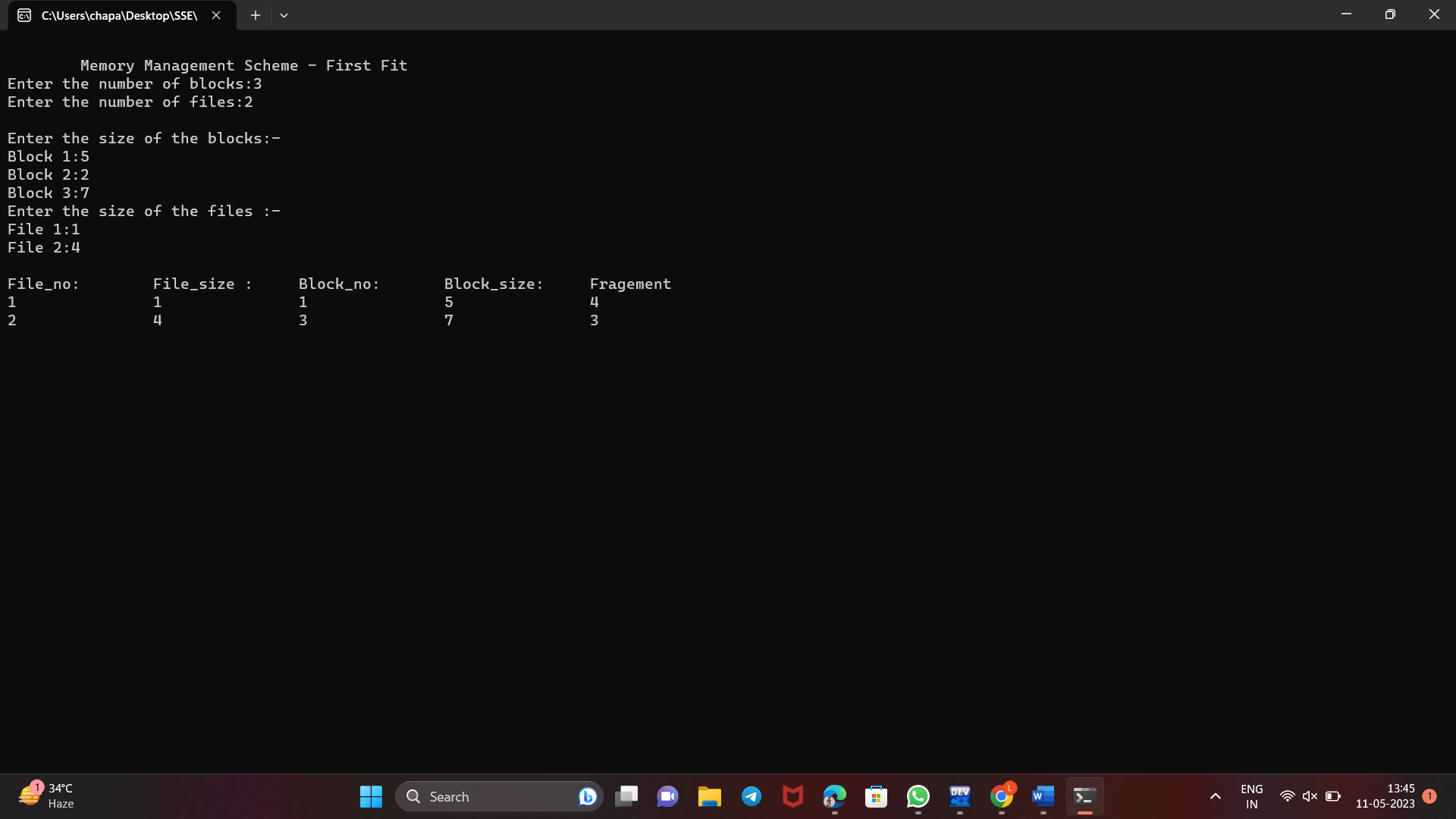
for(i=1;i<=nf;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}

**OUTPUT**



**20. Page Replacement Algorithm – First in First Out (FIFO)**

**Program**

#include <stdio.h>

#define MAX\_FRAMES 3

int main() {

int pages[] = {4, 1, 2, 4, 3, 2, 1, 5};

int n = sizeof(pages) / sizeof(int);

int frames[MAX\_FRAMES];

int page\_faults = 0;

int next\_frame = 0;

int i, j, k;

int found;

// Initialize frames to -1 to indicate empty

for (i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1;

}

// Loop through pages

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

found = 0;

// Check if page is already in a frame

for (j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] == pages[i]) {

found = 1;

break;

}

}

// If page is not in a frame, replace the oldest page

if (!found) {

frames[next\_frame] = pages[i];

next\_frame = (next\_frame + 1) % MAX\_FRAMES;

page\_faults++;

}

// Print current frame contents

printf("Current frames: ");

for (k = 0; k < MAX\_FRAMES; k++) {

if (frames[k] == -1) {

printf("-");

} else {

printf("%d", frames[k]);

}

if (k < MAX\_FRAMES - 1) {

printf(", ");

}

}

printf("\n");

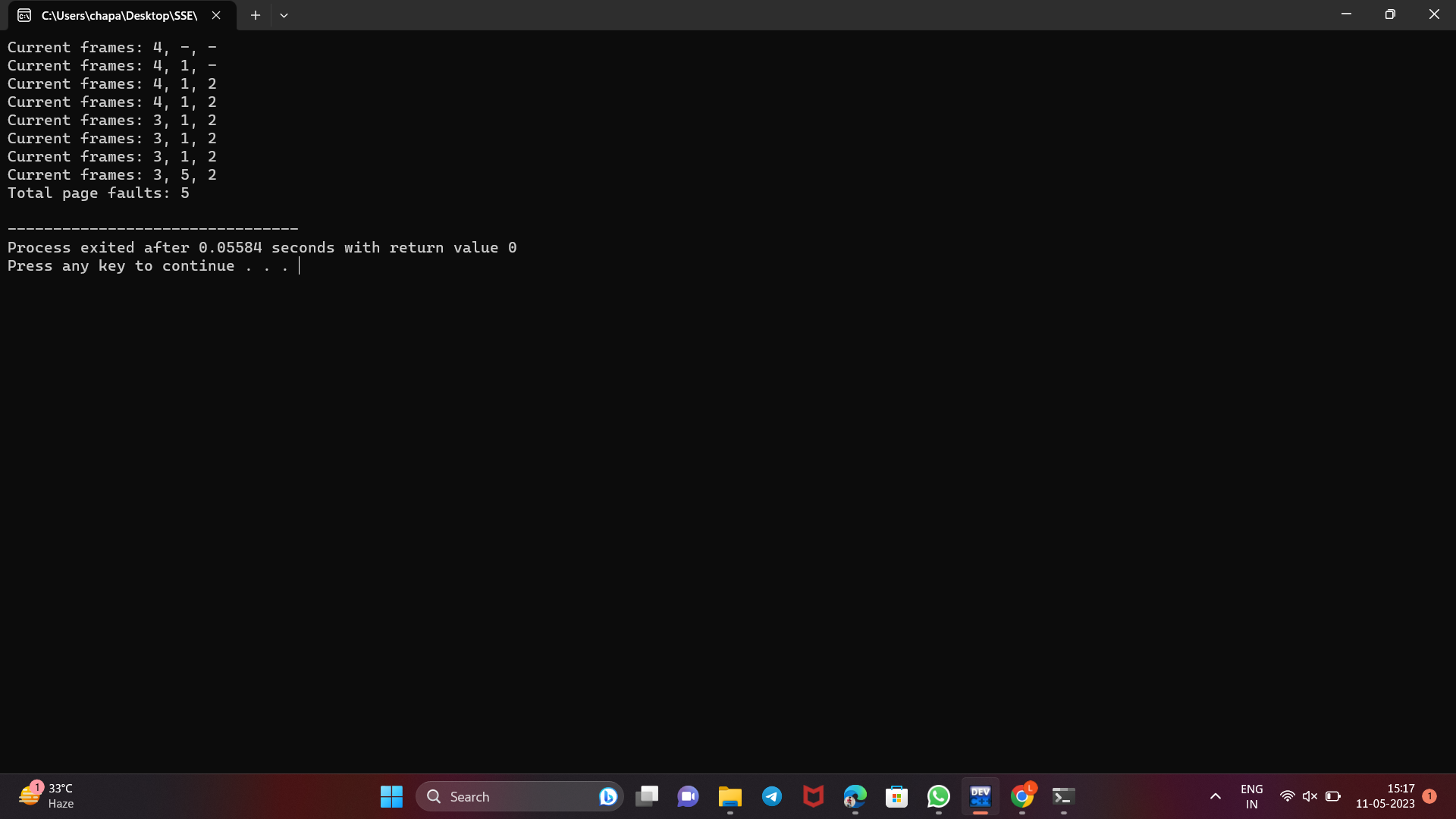
}

printf("Total page faults: %d\n", page\_faults);

return 0;

}

**Output**



**21. Page Replacement Algorithm -Least Recently Used (LRU)**

**Program**

#include<stdio.h>

#include<conio.h>

int fr[3];

main()

{

void display();

int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];

int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;

//clrscr();

for(i=0;i<3;i++)

{

fr[i]=-1;

}

for(j=0;j<12;j++)

{

flag1=0,flag2=0;

for(i=0;i<3;i++)

{

if(fr[i]==p[j])

{

flag1=1;

flag2=1; break;

}

}

if(flag1==0)

{

for(i=0;i<3;i++) {

if(fr[i]==

-1)

{

fr[i]=p[j]; flag2=1;

break; }}}

if(flag2==0) {

for(i=0;i<3;i++)

fs[i]=0;

for(k=j

-1,l=1;l<=frsize

-1;l++,k--

)

{

for(i=0;i<3;i++) {

if(fr[i]==p[k]) fs[i]=1;

}}

for(i=0;i<3;i++) {

if(fs[i]==0)

index=i; }

fr[index]=p[j];

pf++; }

display(); }

printf("\n no of page faults :%d",pf+frsize);

getch(); }

void display() {

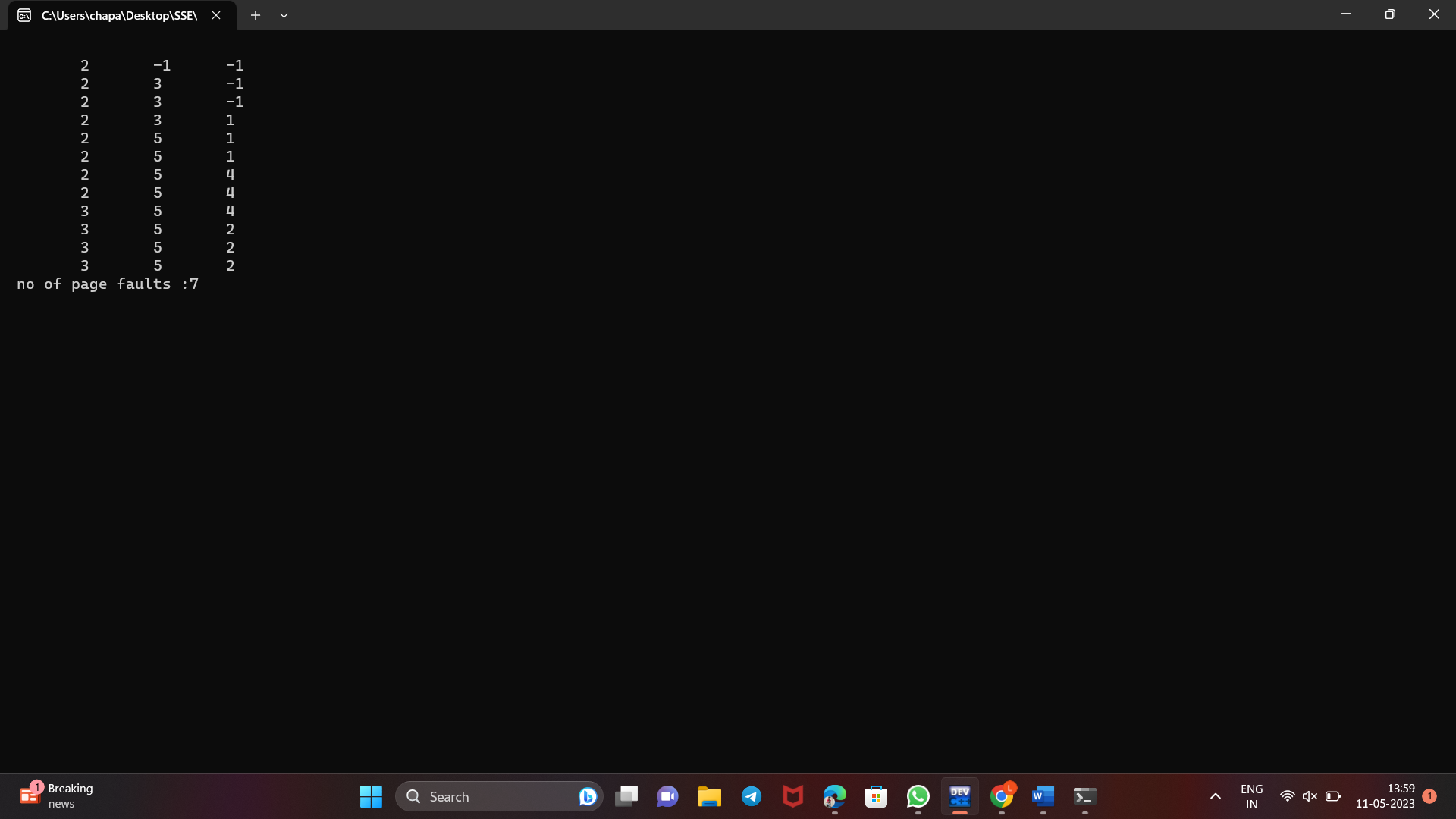
int i; printf("\n");

for(i=0;i<3;i++)

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

}

**OUTPUT**



**22. Page Replacement Algorithm -OPTIMAL**

**Program**

#include<stdio.h>

#include<conio.h>

int fr[3], n, m;

void

display();

main()

{

int i,j,page[20],fs[10];

int

max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0;

float pr;

//clrscr();

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

scanf("%d",&n);

printf("Enter the reference string: ");

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

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

printf("Enter no of frames: ");

scanf("%d",&m);

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

fr[i]=-1; pf=m;

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

flag1=0; flag2=0;

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

if(fr[i]==page[j]) {

flag1=1; flag2=1;

break; }}

if(flag1==0) {

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

if(fr[i]==

-1)

{

fr[i]=page[j]; flag2=1;

break; }}}

if(flag2==0) {

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

lg[i]=0;

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

for(k=j+1;k<=n;k++) {

if(fr[i]==page[k]) {

lg[i]=k

-j;

break; }}}

found=0;

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

if(lg[i]==0) {

index=i;

found = 1;

break;

}

}

if(found==0)

{

max=lg[0]; index=0;

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

{

if(max<lg[i])

{

max=lg[i];

index=i;

}

}

}

fr[index]=page[j];

pf++;

}

display();

}

printf("Number of page faults : %d\n", pf);

pr=(float)pf/n\*100;

printf("Page fault rate = %f \n", pr); getch();

}

void display()

{

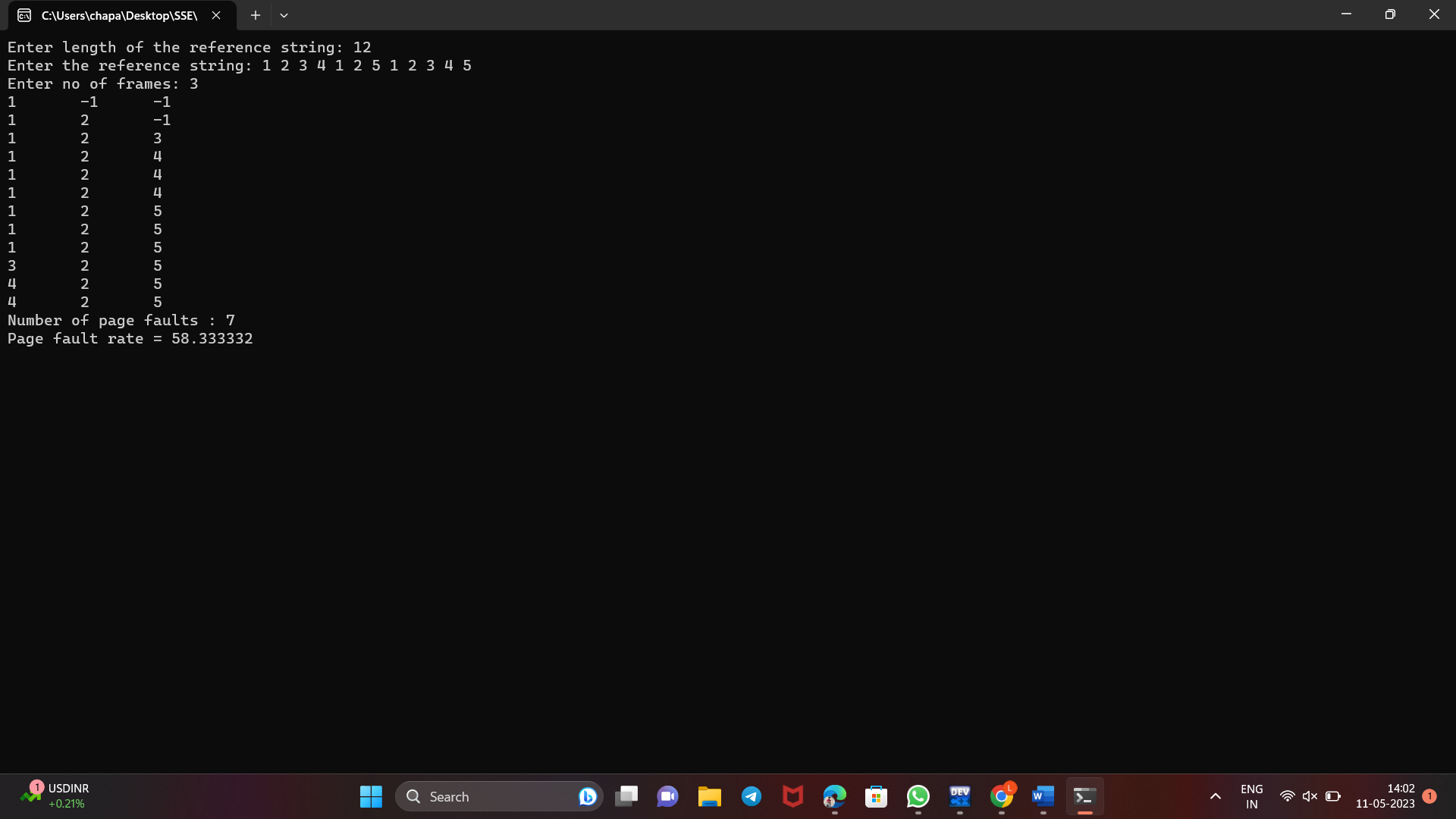
int i; for(i=0;i<m;i++)

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

printf("\n");

}

**OUTPUT**



**23. Disk Scheduling Algorithm – FCFS**

**Program**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int tracks[] = {55, 58, 60, 70, 18, 90, 150, 160, 184};

int n = sizeof(tracks) / sizeof(tracks[0]);

int head = 50; // start head position

int movement = 0;

printf("Sequence of tracks visited:\n%d", head);

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

{

movement += abs(tracks[i] - head);

head = tracks[i];

printf(" -> %d", head);

}

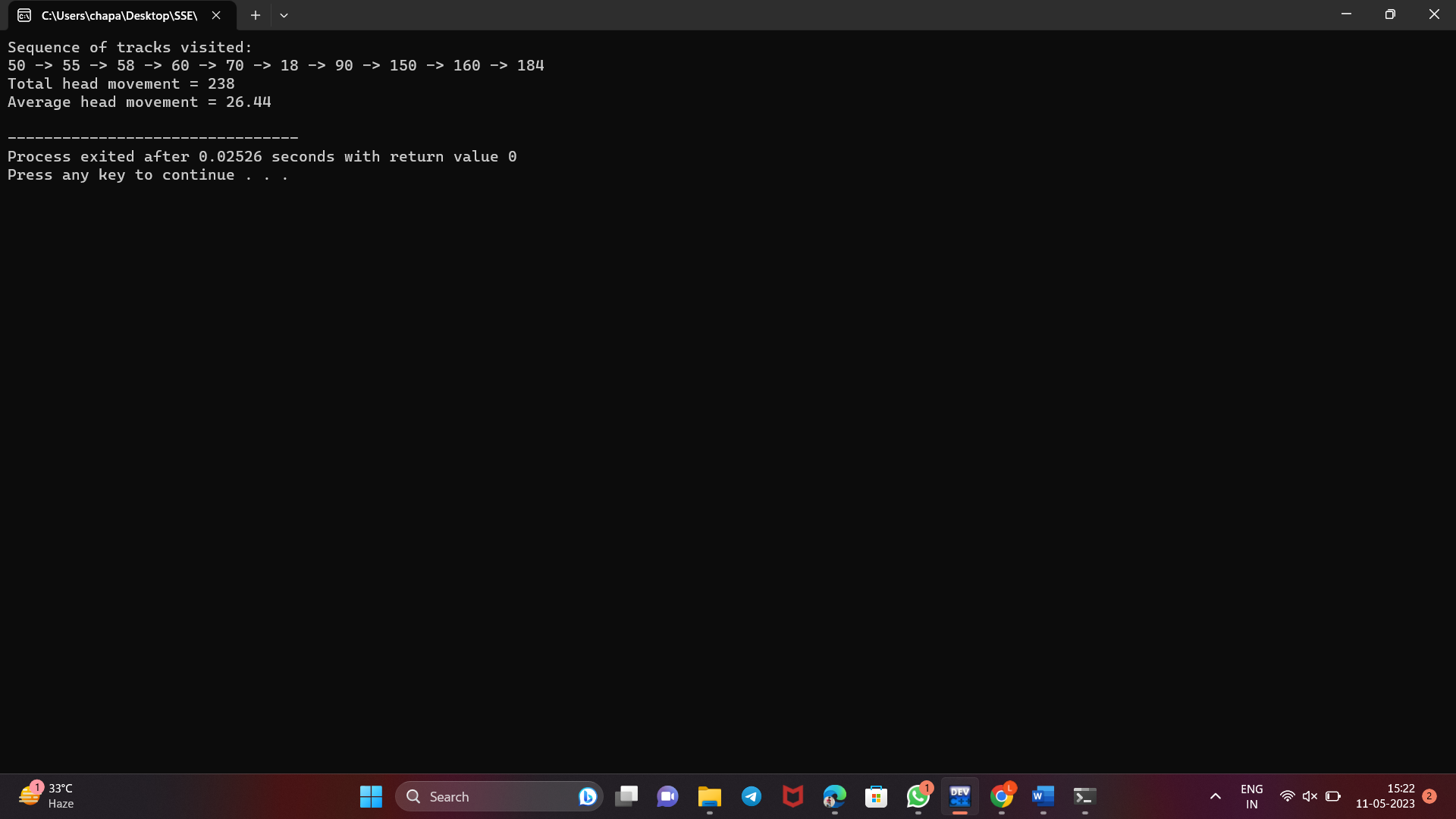
printf("\nTotal head movement = %d\n", movement);

printf("Average head movement = %.2f\n", (float)movement/n);

return 0;

}

**Output**



**24.SCAN DISK SCHEDULING ALGORITHM**

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;

//clrscr();

printf("enter the no of tracks to be traveresed");

scanf("%d'",&n);

printf("enter the position of head");

scanf("%d",&h);

t[0]=0;t[1]=h;

printf("enter the tracks");

for(i=2;i<n+2;i++)

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

for(i=0;i<n+2;i++)

{

for(j=0;j<(n+2)-i-1;j++)

{

if(t[j]>t[j+1])

{

temp=t[j];

t[j]=t[j+1];

t[j+1]=temp;

} } }

for(i=0;i<n+2;i++)

if(t[i]==h)

j=i;k=i;

p=0;

while(t[j]!=0)

{

atr[p]=t[j]; j--;

p++;

}

atr[p]=t[j];

for(p=k+1;p<n+2;p++,k++)

atr[p]=t[k+1];

for(j=0;j<n+1;j++)

{

if(atr[j]>atr[j+1])

d[j]=atr[j]-atr[j+1];

else

d[j]=atr[j+1]-atr[j];

sum+=d[j];

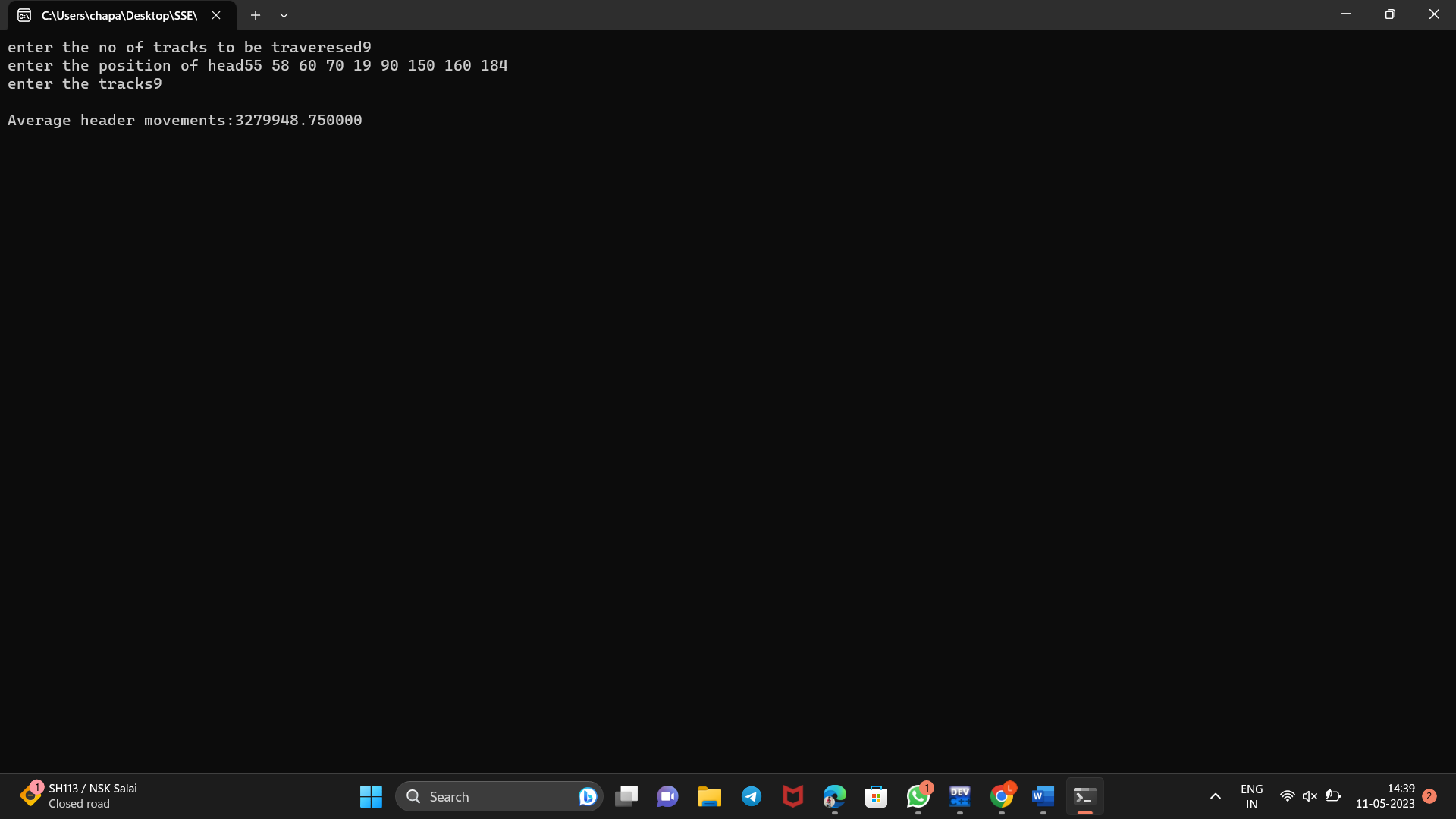
}

printf("\nAverage header movements:%f",(float)sum/n);

getch();

}

**OUTPUT**



**25.C-SCAN DISK SCHEDULING**

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;

//clrscr();

printf("enter the no of tracks to be traveresed");

scanf("%d'",&n);

printf("enter the position of head");

scanf("%d",&h);

t[0]=0;t[1]=h;

printf("enter total tracks");

scanf("%d",&tot);

t[2]=tot-1;

printf("enter the tracks");

for(i=3;i<=n+2;i++)

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

for(i=0;i<=n+2;i++)

for(j=0;j<=(n+2)-i-1;j++)

if(t[j]>t[j+1])

{

temp=t[j];

t[j]=t[j+1];

t[j+1]=temp;

}

for(i=0;i<=n+2;i++)

if(t[i]==h);

j=i;

//break;

p=0;

while(t[j]!=tot-1)

{

atr[p]=t[j];

j++;

p++;

}

atr[p]=t[j];

p++;

i=0;

while(p!=(n+3) && t[i]!=t[h])

{

atr[p]=t[i]; i++;

p++;

}

for(j=0;j<n+2;j++)

{

if(atr[j]>atr[j+1])

d[j]=atr[j]-atr[j+1];

else

d[j]=atr[j+1]-atr[j];

sum+=d[j];

}

printf("total header movements%d",sum);

printf("avg is %f",(float)sum/n);

getch();

}

**OUTPUT**



**26.File allocation-sequential**

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int f[50],i,st,j,len,c,k;

//clrscr();

for(i=0;i<50;i++)

f[i]=0;

X:

printf("\n Enter the starting block & length of file");

scanf("%d%d",&st,&len);

for(j=st;j<(st+len);j++)

if(f[j]==0)

{

f[j]=1

;

printf("\n%d->%d",j,f[j]);

}

else

{

printf("Block already allocated");

break;

}

if(j==(st+len))

printf("\n the file is allocated to disk");

printf("\n if u want to enter more files?(y-1/n-0)");

scanf("%d",&c);

if(c==1)

goto X;

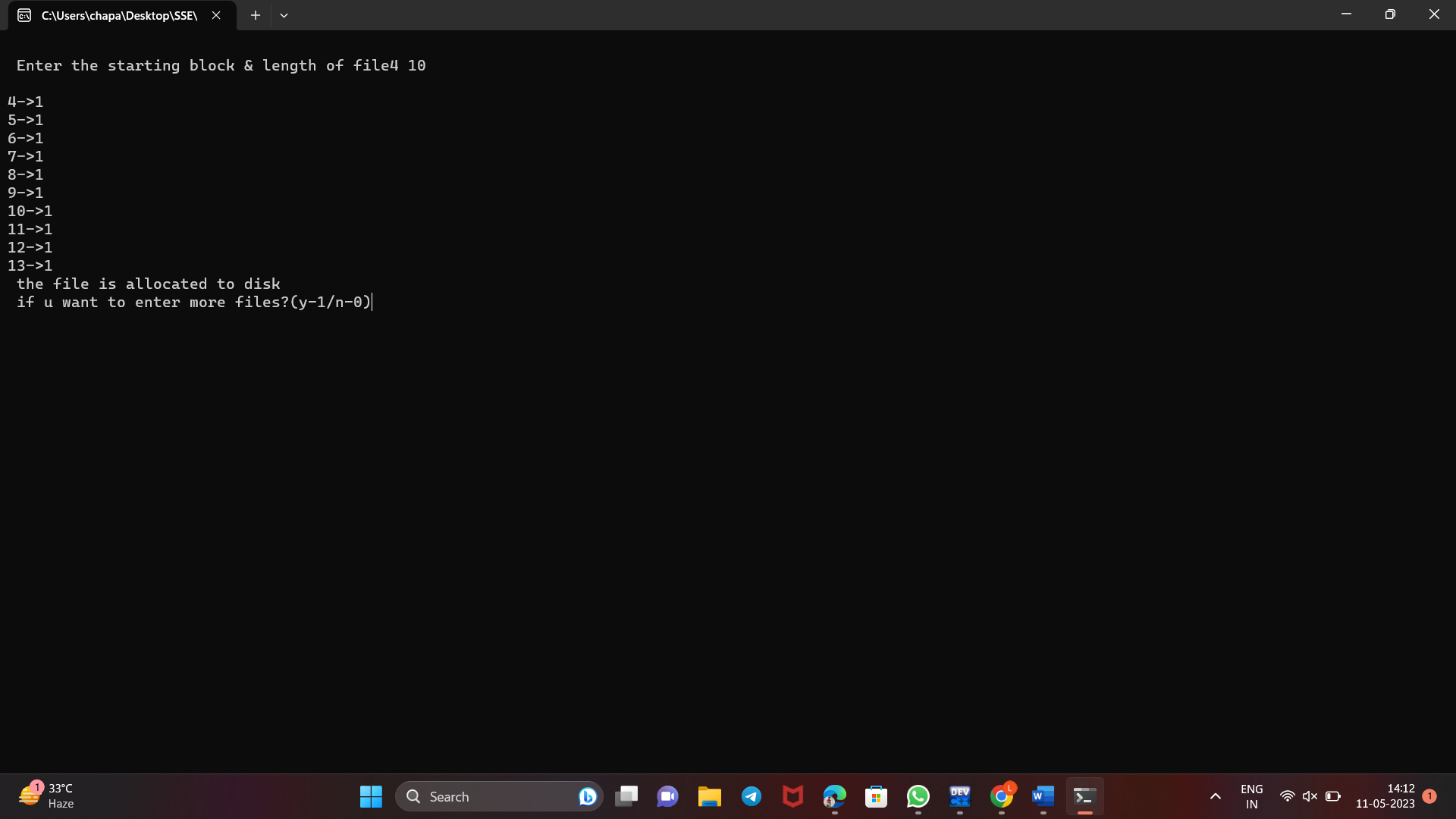
else

//exit();

getch();

}

**OUTPUT**



**27.linked file allocation technique**.

**Program**

#include<stdio.h>

#include<conio.h>

main()

{

int f[50],p,i,j,k,a,st,len,n,c;

//clrscr();

for(i=0;i<50;i++) f[i]=0;

printf("Enter how many blocks that are already allocated");

scanf("%d",&p);

printf("\nEnter the blocks no.s that are already allocated");

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

{

scanf("%d",&a);

f[a]=1;

}

X:

printf("Enter the starting index block & length");

scanf("%d%d",&st,&len); k=len;

for(j=st;j<(k+st);j++)

{

if(f[j]==0)

{ f[j]=1;

printf("\n%d->%d",j,f[j]);

}

else

{

printf("\n %d->file is already allocated",j);

k++;

}

}

printf("\n If u want to enter one more file? (yes-1/no-0)");

scanf("%d",&c);

if(c==1)

goto

X;

else

//exit();

getch( );

}

**output**



**28.indexed file allocation technique**.

**Program**

#include<stdio.h>

#include<conio.h>

int f[50],i,k,j,inde[50],n,c,count=0,p;

main() {

//clrscr();

for(i=0;i<50;i++)

f[i]=0;

x: printf("enter index block\t");

scanf("%d",&p);

if(f[p]==0)

{

f[p]=1;

printf("enter no of files on index\t");

scanf("%d",&n); }

else {

printf("Block already allocated\n");

goto x; }

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

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

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

if(f[inde[i]]==1) {

printf("Block already allocated");

goto x; }

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

f[inde[j]]=1;

printf("\n allocated");

printf("\n file indexed");

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

printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);

printf(" Enter 1 to enter more files and 0 to exit\t");

scanf("%d",&c);

if(c==1)

goto x;

else

//exit();

getch();

}

**OUTPUT**



**29.Single level directory**

**Program**

#include<stdio.h>

#include<string.h>

#include<conio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir;

main()

{

int i,ch; char

f[30]; //clrscr();

dir.fcnt = 0;

printf("\nEnter name of directory -- ");

scanf("%s", dir.dname);

while(1)

{

printf("\n\n1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice -- ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter the name of the file -- ");

scanf("%s",dir.fname[dir.fcnt]);

dir.fcnt++; break;

case 2: printf("\nEnter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is deleted ",f); strcpy(dir.fname[i],dir.fname[dir.fcnt-1]); break;

}

}

if(i==dir.fcnt)

printf("File %s not found",f);

else

dir.fcnt--;

break;

case 3: printf("\nEnter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is found ", f);

break;

}

}

if(i==dir.fcnt)

printf("File %s not found",f);

break;

case 4: if(dir.fcnt==0)

printf("\nDirectory Empty");

else

{

printf("\nThe Files are -- ");

for(i=0;i<dir.fcnt;i++)

printf("\t%s",dir.fname[i]);

}

break;

//default: //exit(0);

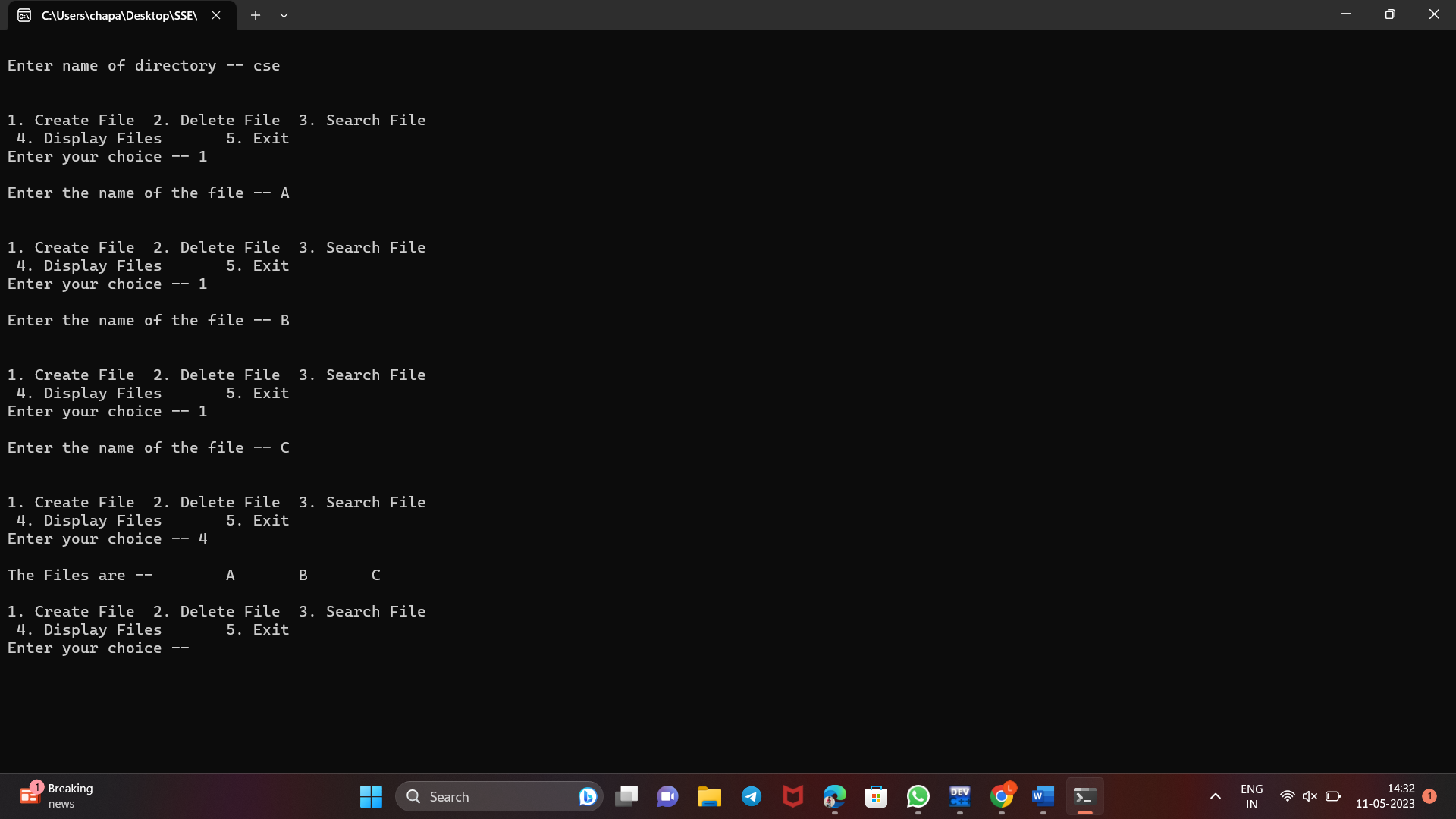
}

}

getch();

}

**OUTPUT**



**30.Two level file organization**

**Program**

#include<stdio.h>

#include<string.h>

#include<conio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir[10];

main()

{

int i,ch,dcnt,k; char

f[30], d[30]; //clrscr();

dcnt=0;

while(1)

{

printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");

printf("\n4. Search File\t\t5. Display\t6. Exit\t Enter your choice --");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter name of directory -- ");

scanf("%s", dir[dcnt].dname);

dir[dcnt].fcnt=0;

dcnt++;

printf("Directory created"); break;

case 2: printf("\nEnter name of the directory -- ");

scanf("%s",d);

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

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- ");

scanf("%s",dir[i].fname[dir[i].fcnt]);

dir[i].fcnt++;

printf("File created");

}

if(i==dcnt)

printf("Directory %s not found",d);

break;

case 3: printf("\nEnter name of the directory -- ");

scanf("%s",d);

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

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

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- ");

scanf("%s",f);

for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is deleted ",f);

dir[i].fcnt--;

strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);

goto jmp;

}

}

printf("File %s not found",f); goto jmp;

}

}

printf("Directory %s not found",d);

jmp : break;

case 4: printf("\nEnter name of the directory -- ");

scanf("%s",d);

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

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter the name of the file -- ");

scanf("%s",f);

for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is found ",f); goto jmp1;

}

}

printf("File %s not found",f); goto jmp1;

}

}

printf("Directory %s not found",d); jmp1: break;

case 5: if(dcnt==0)

printf("\nNo Directory's ");

else

{

printf("\nDirectory\tFiles");

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

{

printf("\n%s\t\t",dir[i].dname);

for(k=0;k<dir[i].fcnt;k++)

printf("\t%s",dir[i].fname[k]);

}

}

break;

//default:exit(0);

}

}

getch();

}

**OUTPUT**

