**Registration Number: 19BCE2119**

**Name: Gaurav Kumar Singh**

**Cyclesheet-3**

**15. Write a program to provide a solution for reader- writer problem / producer consumer using semaphore.**

**CODE**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

sem\_t wrt;

pthread\_mutex\_t mutex;

int cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

sem\_wait(&wrt);

cnt = cnt\*2;

printf("Writer %d modified cnt to %d\n",(\*((int \*)wno)),cnt);

sem\_post(&wrt);

}

void \*reader(void \*rno)

{

pthread\_mutex\_lock(&mutex);

numreader++;

if(numreader == 1)

{

sem\_wait(&wrt);

}

pthread\_mutex\_unlock(&mutex);

printf("Reader %d: read cnt as %d\n",\*((int \*)rno),cnt);

pthread\_mutex\_lock(&mutex);

numreader--;

if(numreader == 0)

{

sem\_post(&wrt);

}

pthread\_mutex\_unlock(&mutex);

}

int main()

{

pthread\_t read[10],write[5];

pthread\_mutex\_init(&mutex, NULL);

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

int a[10] = {1,2,3,4,5,6,7,8,9,10};

for(int i = 0; i < 10; i++)

{

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

}

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

{

pthread\_create(&write[i], NULL, (void \*)writer, (void \*)&a[i]);

}

for(int i = 0; i < 10; i++)

{

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

}

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

{

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

}

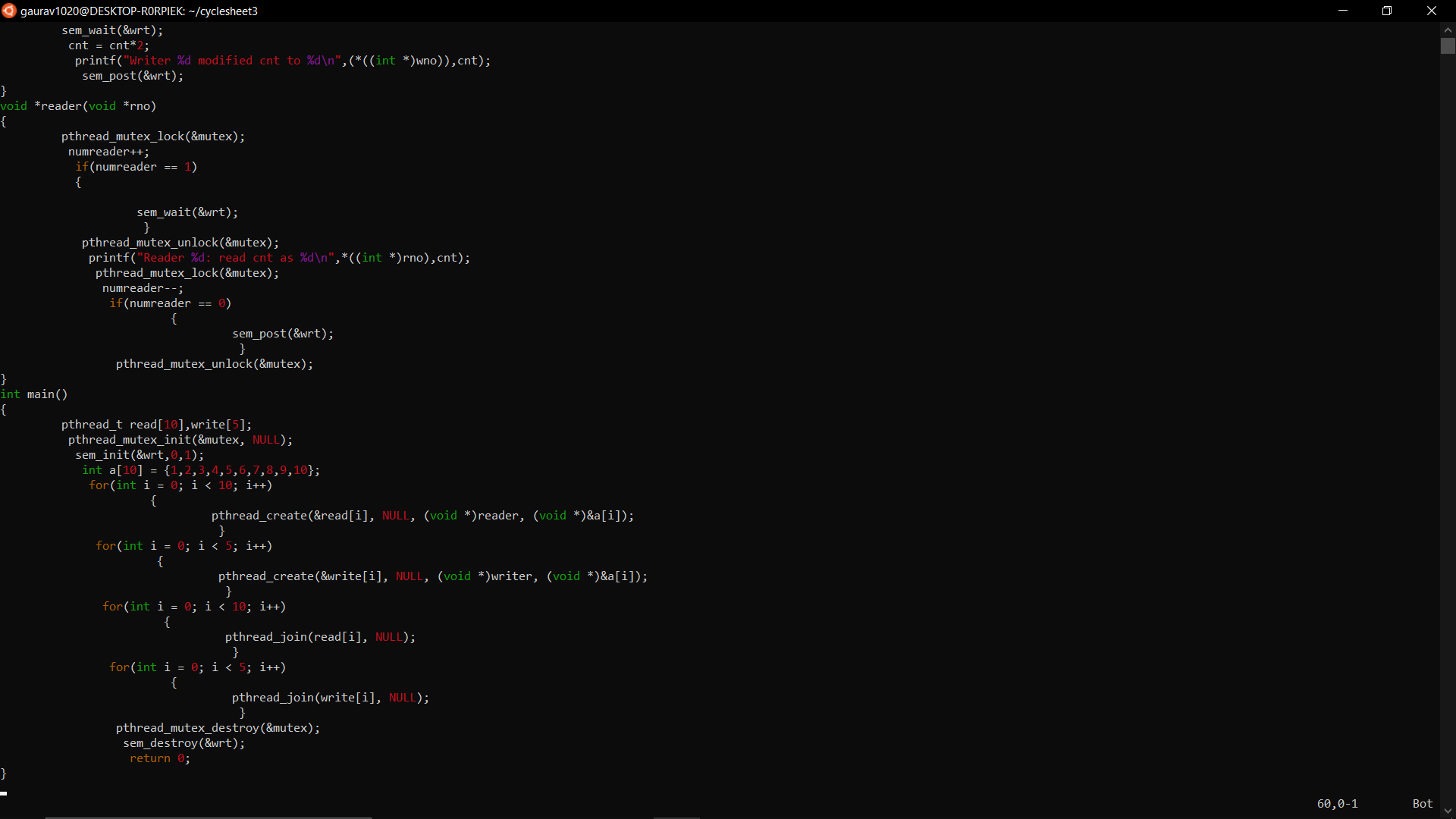
pthread\_mutex\_destroy(&mutex);

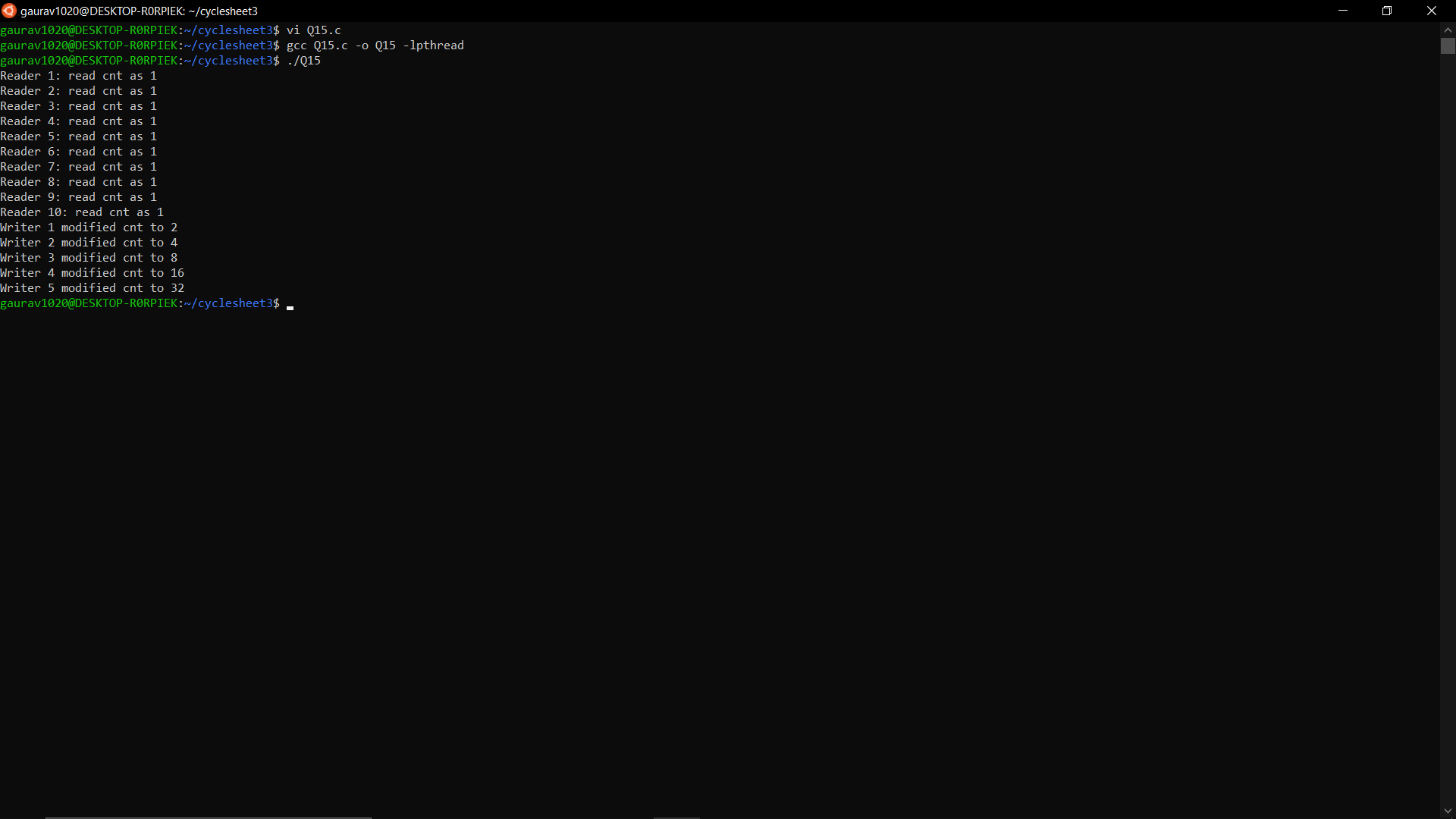
sem\_destroy(&wrt);

return 0;

}

**OUTPUT**





**16. Implement a solution for the classical synchronization problem: Dining Philosophers.**

**CODE**

#include<stdio.h>

#define n 5

int compltedPhilo = 0, i;

struct fork

{

int taken;

}

ForkAvil[n];

struct philosp

{

int left;

int right;

}

Philostatus[n];

void goForDinner(int philID)

{

if (Philostatus[philID].left == 10 && Philostatus[philID].right == 10)

printf("Philosopher %d completed his dinner\n", philID + 1);

else if (Philostatus[philID].left == 1 && Philostatus[philID].right == 1)

{

printf("Philosopher %d completed his dinner\n", philID + 1);

Philostatus[philID].left = Philostatus[philID].right = 10;

int otherFork = philID - 1;

if (otherFork == -1)

otherFork = (n - 1);

ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;

printf("Philosopher %d released fork %d and fork %d\n", philID + 1, philID + 1, otherFork + 1);

compltedPhilo++;

}

else if (Philostatus[philID].left == 1 && Philostatus[philID].right == 0)

{

if (philID == (n - 1))

{

if (ForkAvil[philID].taken == 0)

{

ForkAvil[philID].taken = Philostatus[philID].right = 1;

printf("Fork %d taken by philosopher %d\n", philID + 1, philID + 1);

}

else

{

printf("Philosopher %d is waiting for fork %d\n", philID+1, philID + 1);

}

}

else

{

int dupphilID = philID;

philID -= 1;

if (philID == -1)

philID = (n - 1);

if (ForkAvil[philID].taken == 0)

{

ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;

printf("Fork %d taken by Philosopher %d\n", philID + 1, dupphilID + 1);

}

else

{

printf("Philosopher %d is waiting for Fork %d\n", dupphilID + 1,

philID + 1);

}

}

}

else if (Philostatus[philID].left == 0)

{

if (philID == (n - 1))

{

if (ForkAvil[philID - 1].taken == 0)

{

ForkAvil[philID - 1].taken = Philostatus[philID].left = 1;

printf("Fork %d taken by philosopher %d\n", philID, philID + 1);

}

else {

printf("Philosopher %d is waiting for fork %d\n", philID + 1, philID);

}

}

else {

if (ForkAvil[philID].taken == 0)

{

ForkAvil[philID].taken = Philostatus[philID].left = 1;

printf("Fork %d taken by Philosopher %d\n", philID + 1, philID + 1);

}

else

{

printf("Philosopher %d is waiting for Fork %d\n", philID + 1, philID + 1);

}

}

}

}

int main()

{

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

ForkAvil[i].taken = Philostatus[i].left = Philostatus[i].right = 0;

int compltedPhilo=0;

while (compltedPhilo < n)

{

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

goForDinner(i);

printf("\nTill now num of philosophers completed dinner are %d\n\n", compltedPhilo);

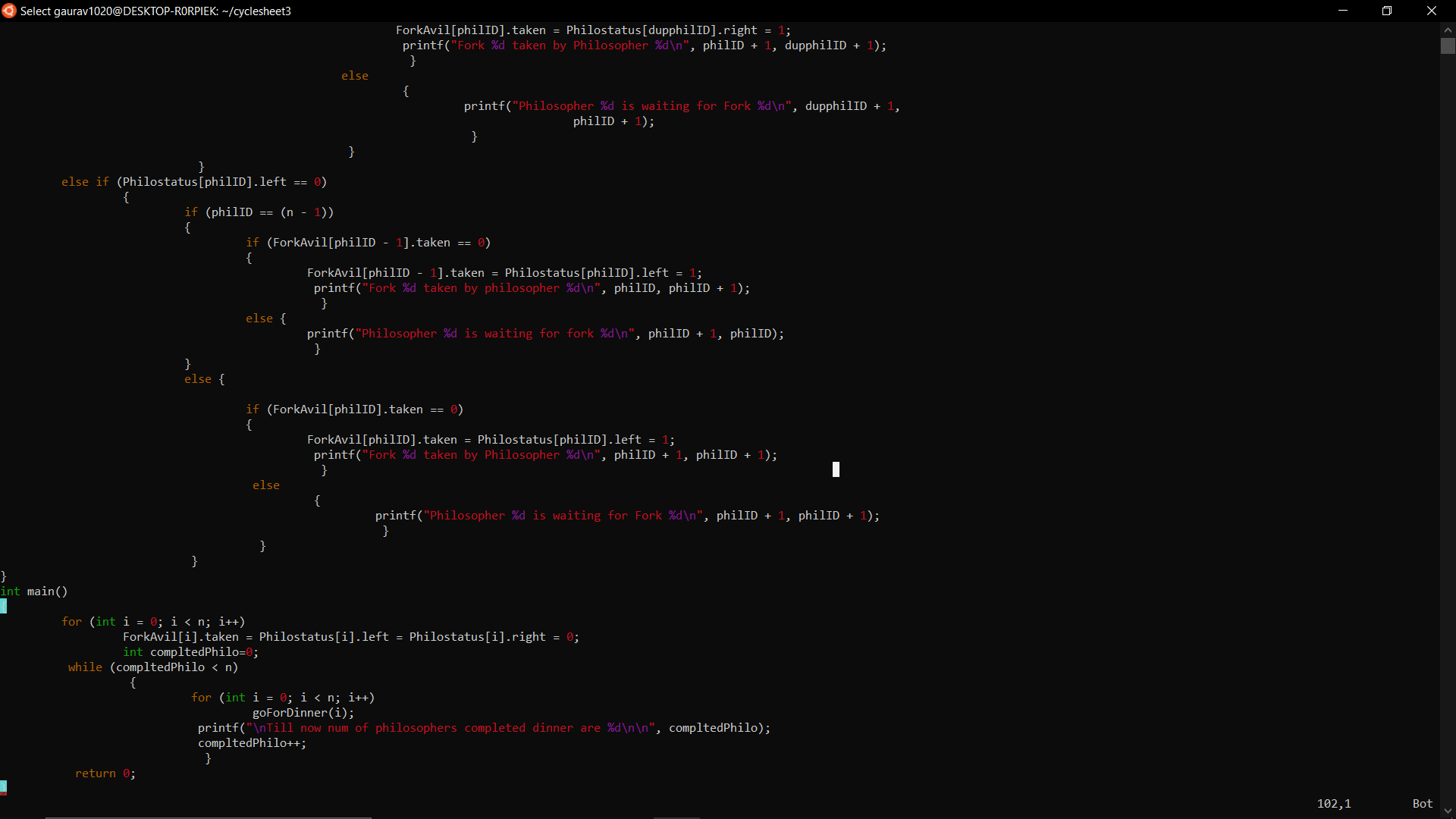
compltedPhilo++;

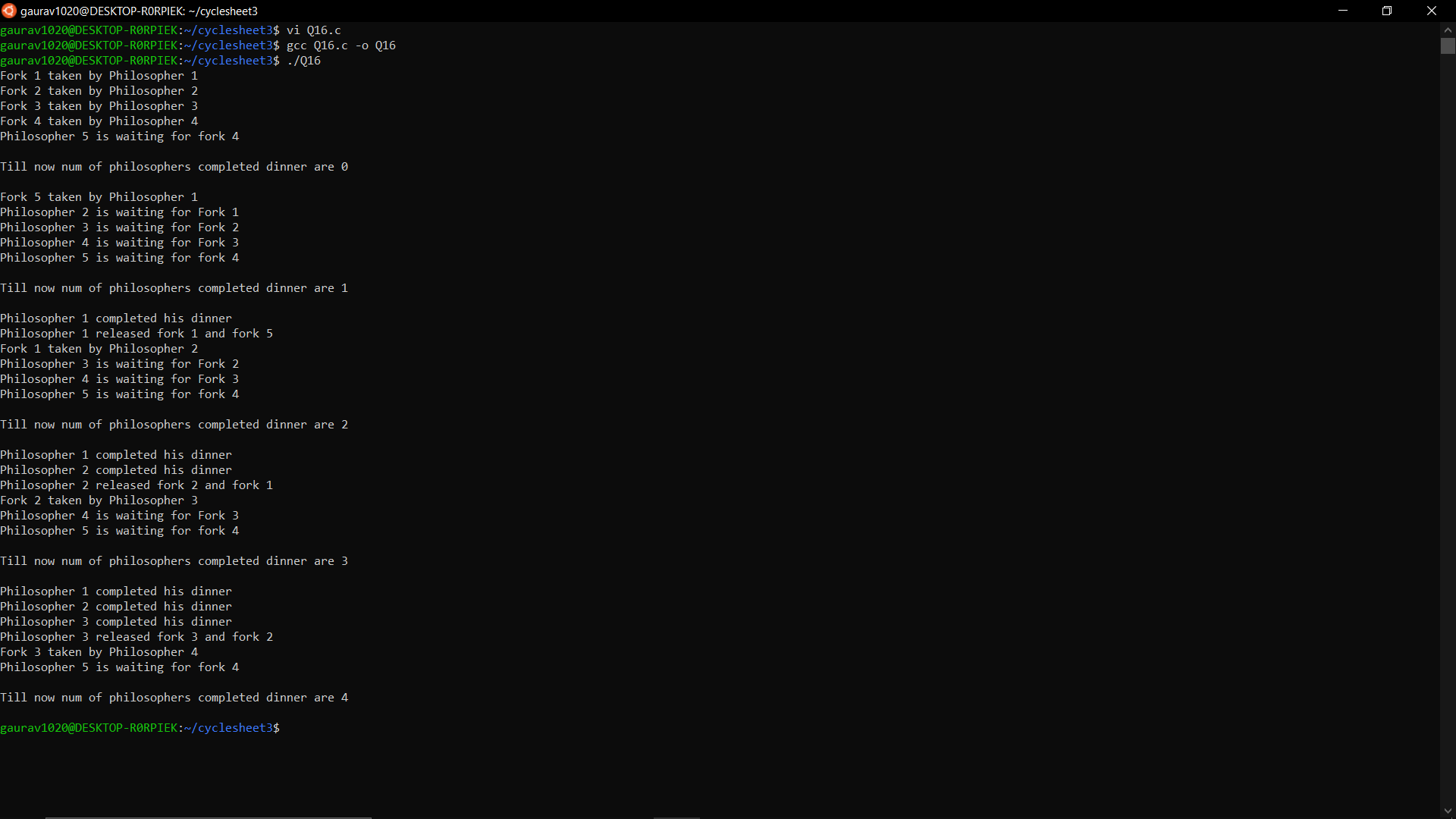
}

return 0;

}

**OUTPUT**





**17. Write a program to avoid deadlock using Banker’s algorithm.(Safety algorithm)**

**CODE**

#include <stdio.h>

int main()

{

int n, m, i, j, k;

n = 5;

m = 3;

int alloc[5][3] = { { 0, 1, 0 }, // P0

{ 2, 0, 0 }, // P1

{ 3, 0, 2 }, // P2

{ 2, 1, 1 }, // P3

{ 0, 0, 2 } }; // P4

int max[5][3] = { { 7, 5, 3 }, // P0

{ 3, 2, 2 }, // P1

{ 9, 0, 2 }, // P2

{ 2, 2, 2 }, // P3

{ 4, 3, 3 } }; // P4

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;

}

}

}

}

printf("Following is the SAFE Sequence\n");

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

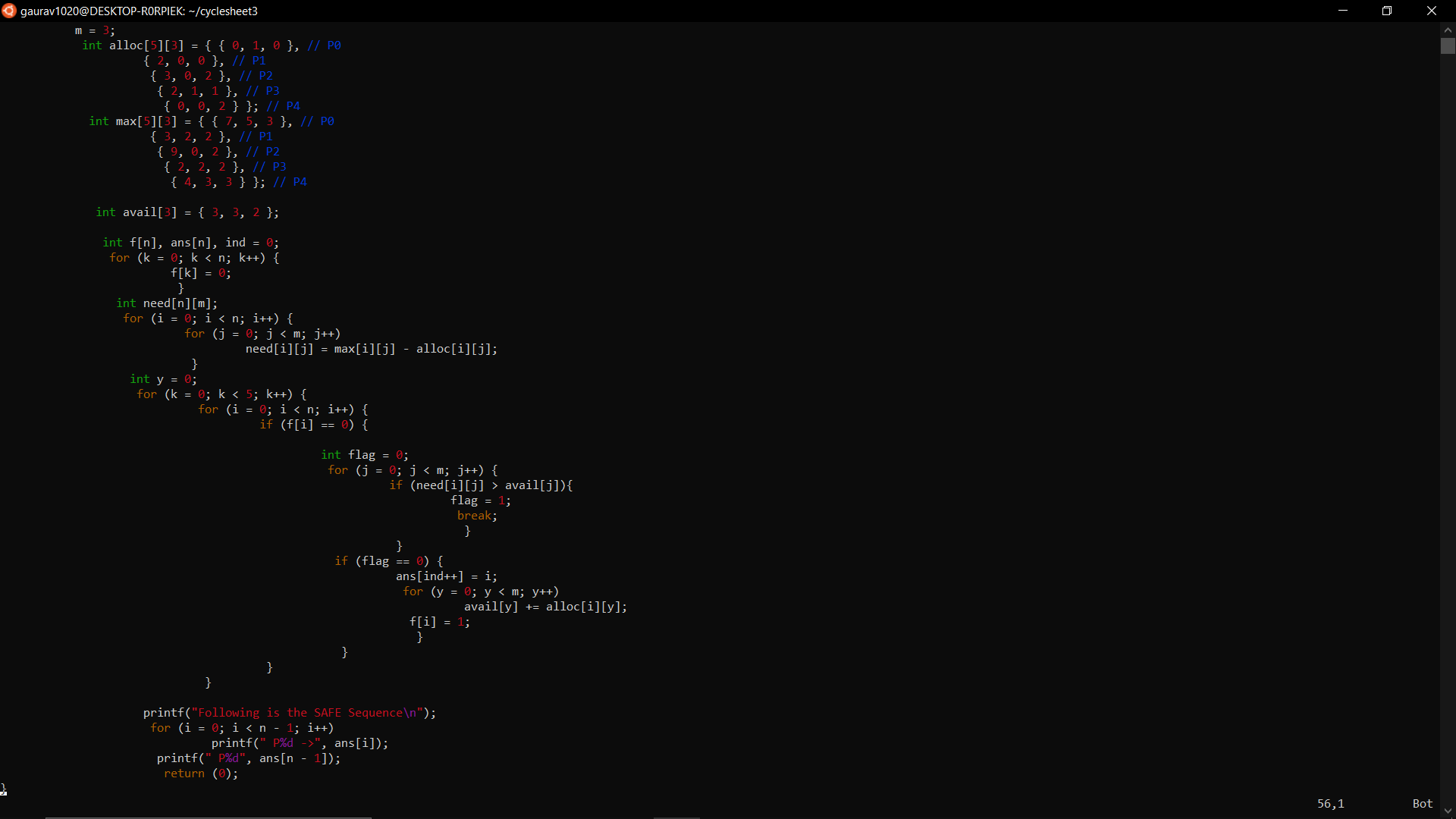
printf(" P%d ->", ans[i]);

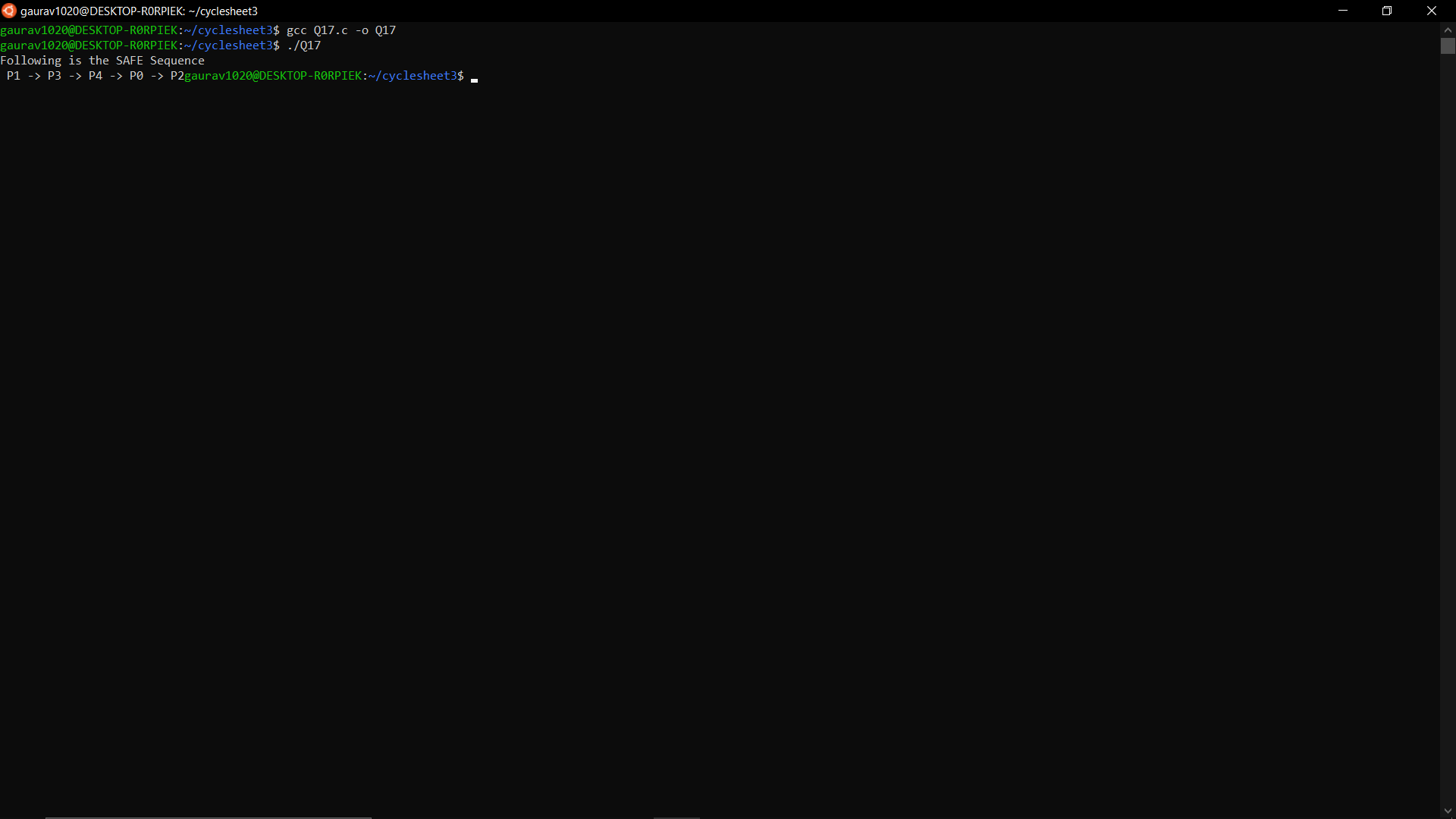
printf(" P%d", ans[n - 1]);

return (0);

}

**OUTPUT**





**18. Implement a program to allocate memory by applying the following strategies.**

**a. FIRST FIT**

**CODE**

#include<stdio.h>

#include<unistd.h>

#define max 25

int main()

{

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

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

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:\tFragment");

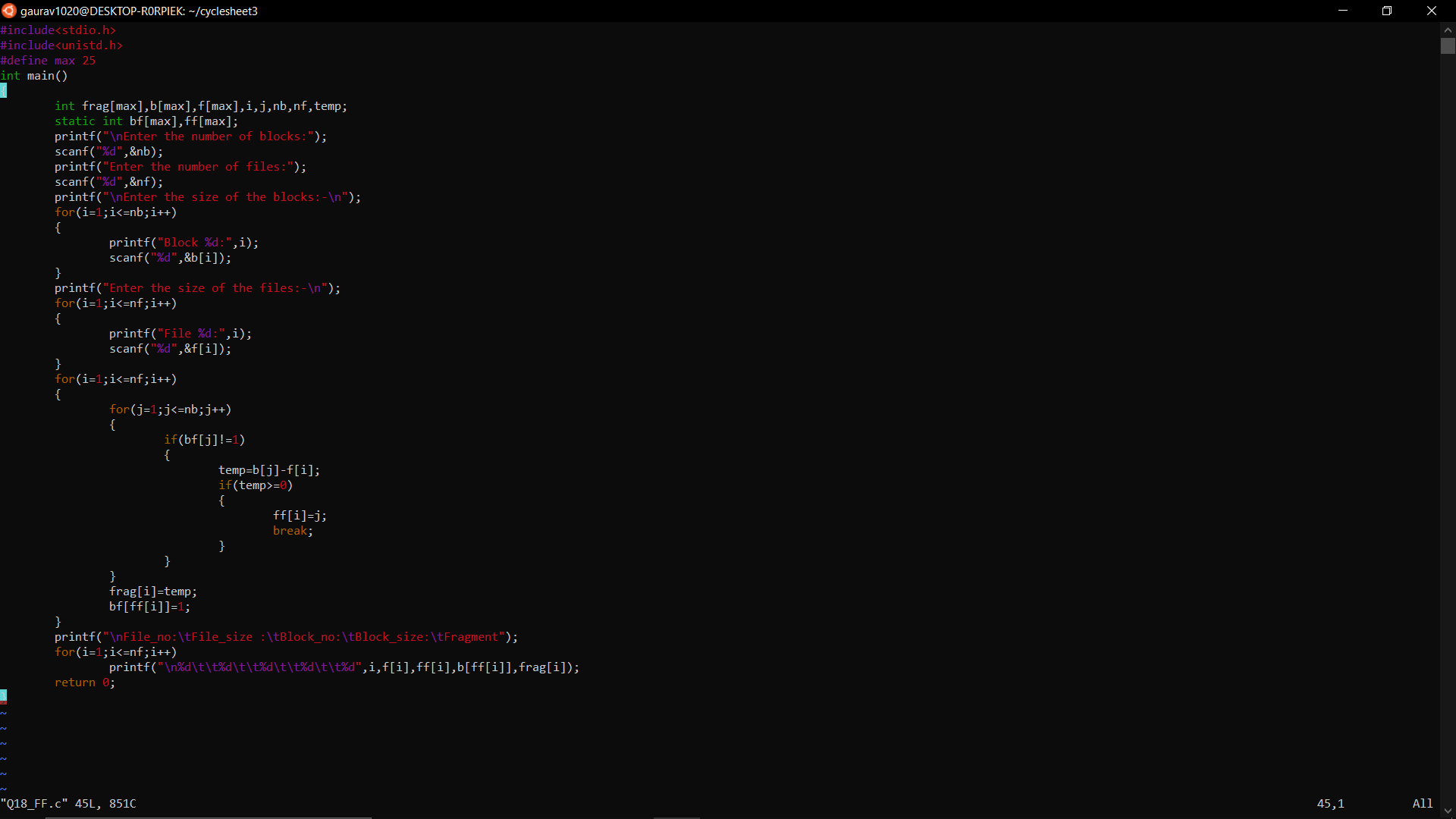
for(i=1;i<=nf;i++)

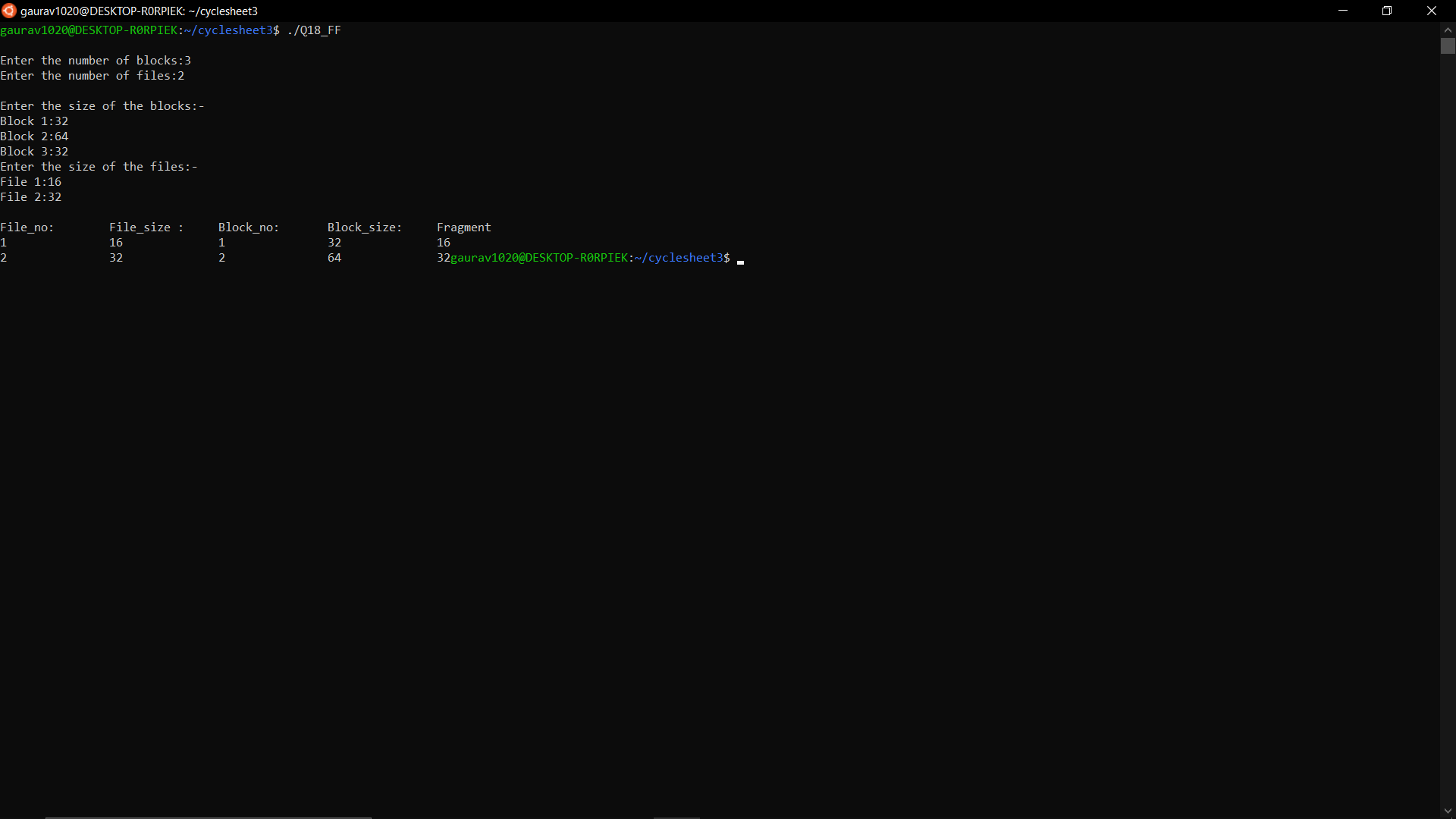
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

return 0;

}

**OUTPUT**





**b. BEST FIT**

**CODE**

#include<stdio.h>

#include<unistd.h>

#define max 25

int main()

{

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

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

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");

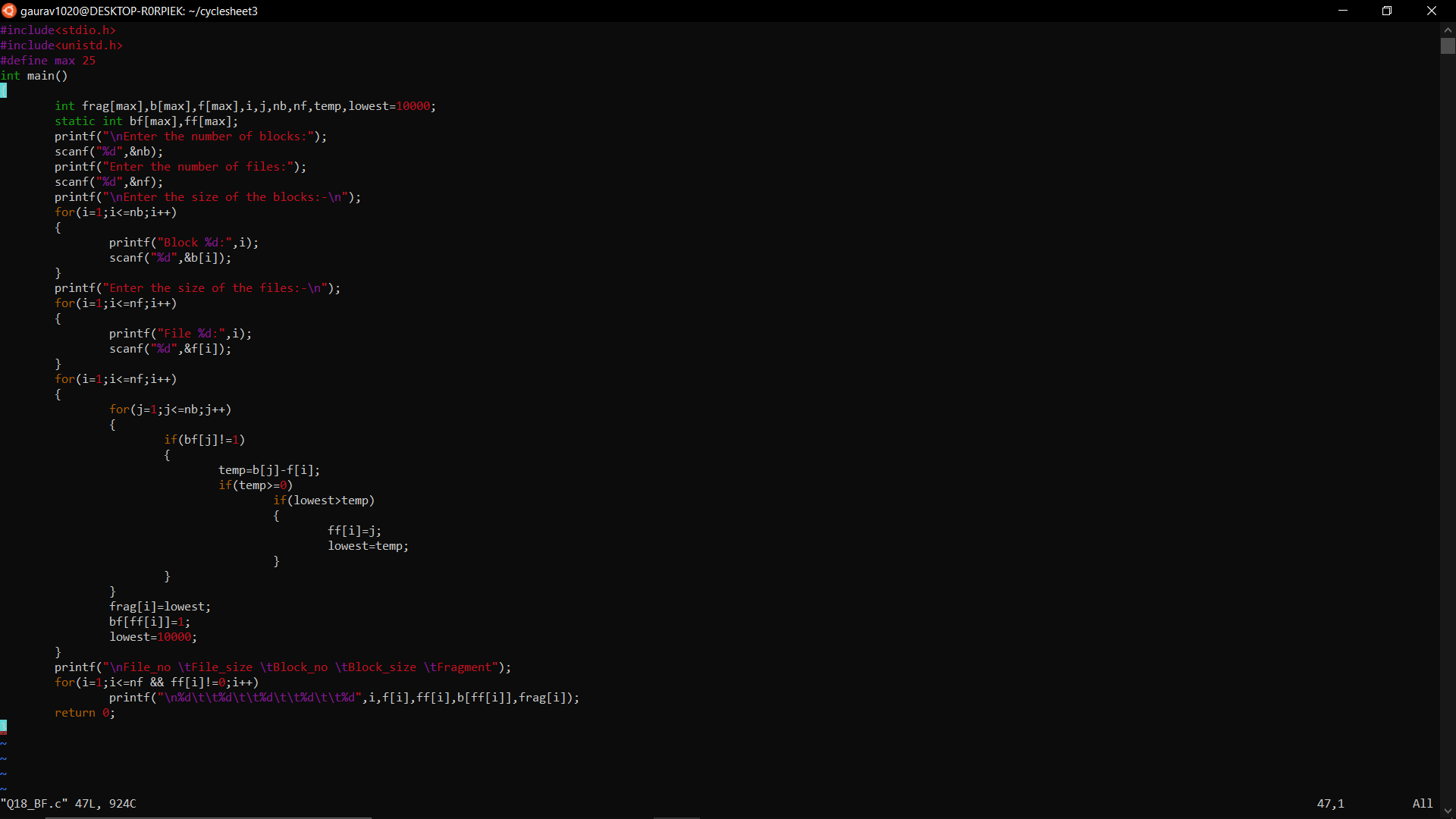
for(i=1;i<=nf && ff[i]!=0;i++)

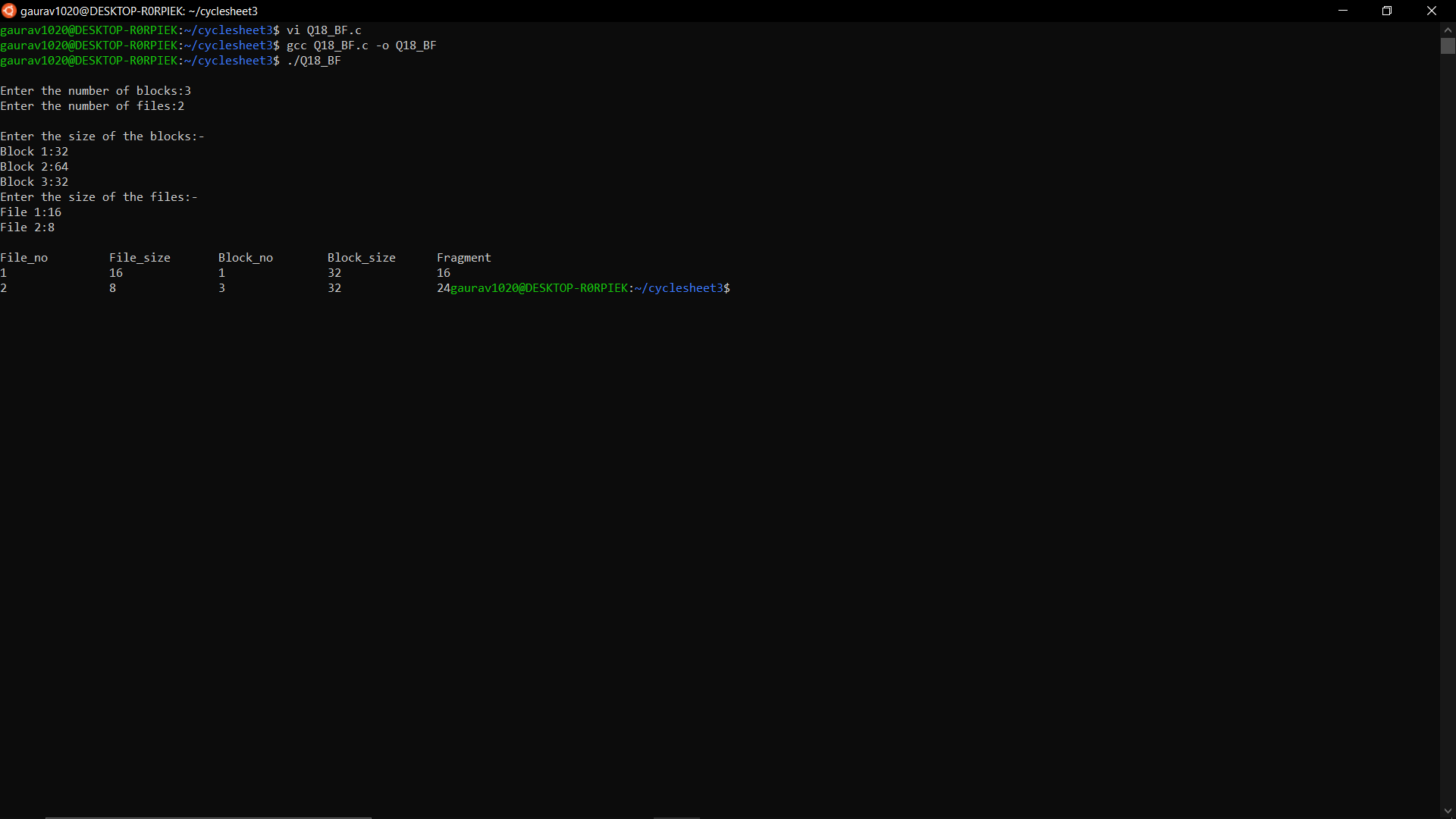
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

return 0;

}

**OUTPUT**





**c. WORST FIT**

**CODE**

#include<stdio.h>

#include<unistd.h>

#define max 25

int main()

{

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

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

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)

{

ff[i]=j;

highest=temp;

}

}

}

frag[i]=highest;

bf[ff[i]]=1;

highest=0;

}

printf("\nFile\_no \tFile\_size \tBlock\_no \tBlock\_size \tFragment");

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]);

return 0;

}

**OUTPUT**

