

## 19BIT0292

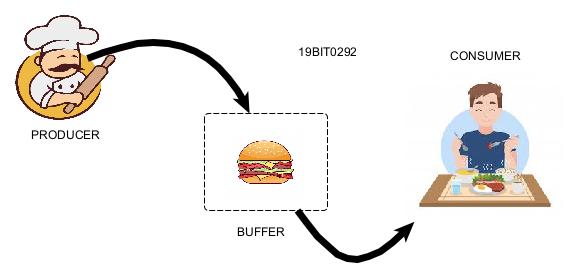
**Bhaumik Tandan**

ASSESMENT-2

OPERATING SYSTEM

Laboratory

## Q1. Write a program to implement the producer –consumer problem using semaphores.

****

**CODE**

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

int max;

int m, b;

sem\_t e;

sem\_t f;

int in = 0;

int out = 0;

int \*buffer;

pthread\_mutex\_t mx;

void \*pro(void \*pno)

{

int j;

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

{

j = rand() % 100;

sem\_wait(&e);

pthread\_mutex\_lock(&mx);

buffer[in] = j;

printf("Producer %d: Insert Item %d at %d\n", \*((int \*)pno), buffer[in], in);

in = (in + 1) % b;

pthread\_mutex\_unlock(&mx);

sem\_post(&f);

}

}

void \*con(void \*cno)

{

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

{

sem\_wait(&f);

pthread\_mutex\_lock(&mx);

int j = buffer[out];

printf("Consumer %d: Remove Item %d from %d\n", \*((int \*)cno), j, out);

out = (out + 1) % b;

pthread\_mutex\_unlock(&mx);

sem\_post(&e);

}

}

void main()

{

pthread\_t \*p, \*c;

int rn, wn, \*n;

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

printf("(19BIT0292)Enter the size of the buffer: ");

scanf("%d", &b);

buffer = (int \*)malloc(sizeof(int) \* b);

sem\_init(&e, 0, b);

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

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

scanf("%d", &rn);

p = (pthread\_t \*)malloc(sizeof(pthread\_t) \* rn);

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

scanf("%d", &wn);

c = (pthread\_t \*)malloc(sizeof(pthread\_t) \* wn);

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

{

n = malloc(sizeof(int));

\*n = i + 1;

pthread\_create(&p[i], NULL, pro, n);

n = NULL;

}

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

{

n = malloc(sizeof(int));

\*n = i + 1;

pthread\_create(&c[i], NULL, con, n);

n = NULL;

}

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

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

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

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

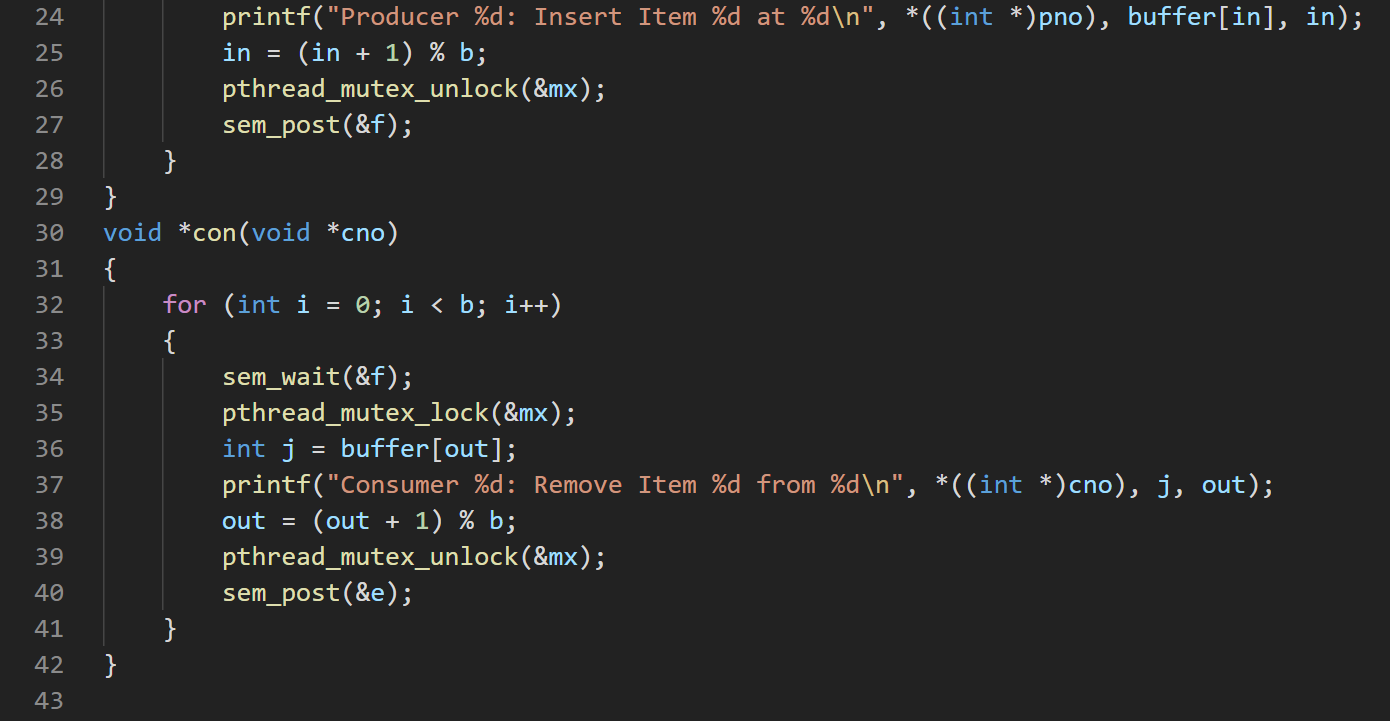
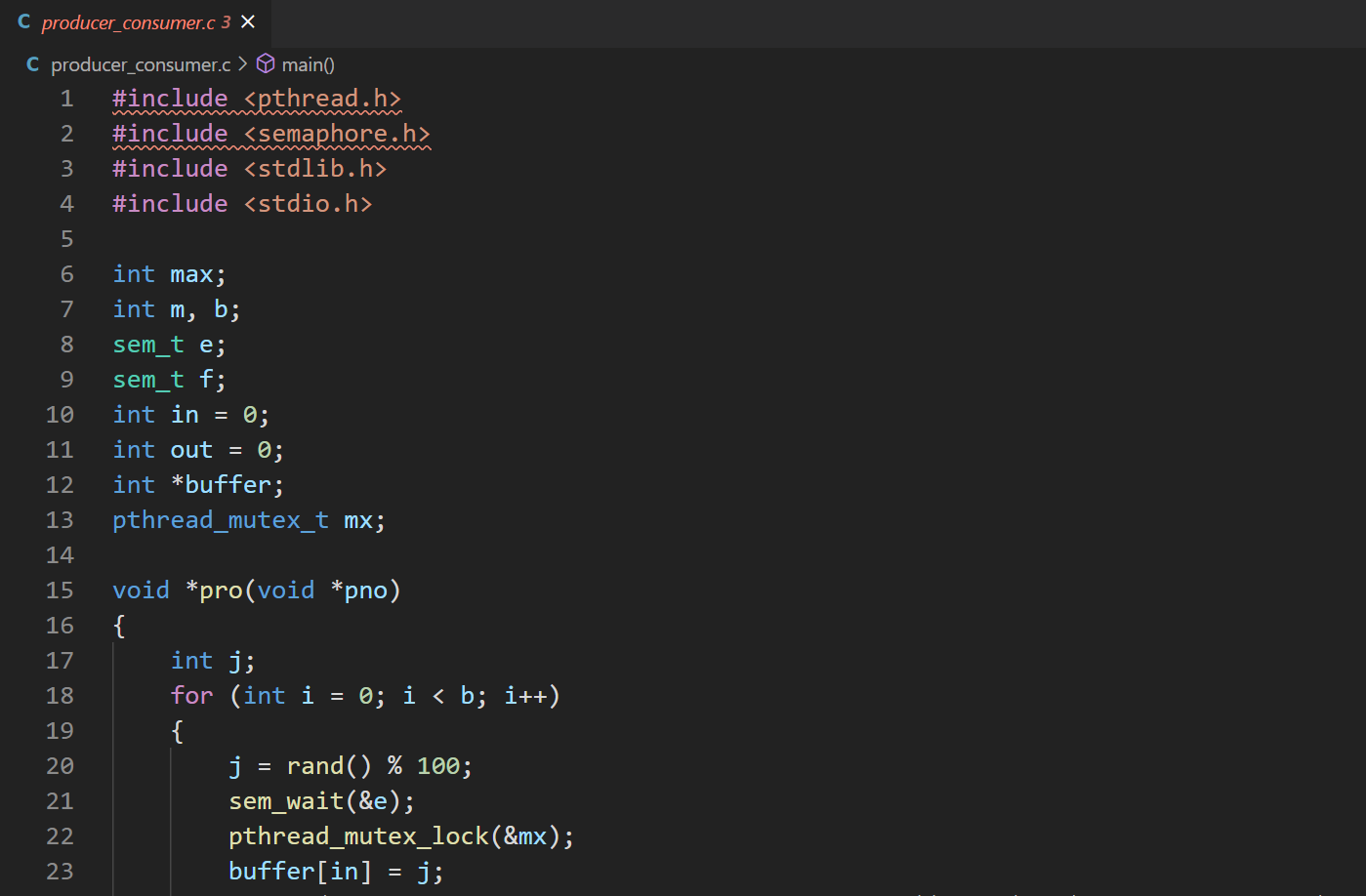
pthread\_mutex\_destroy(&mx);

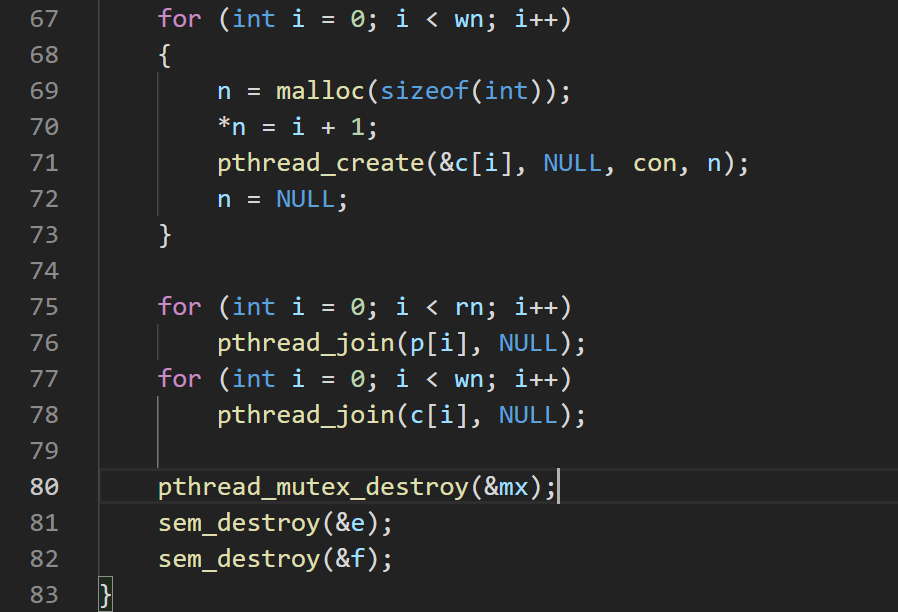
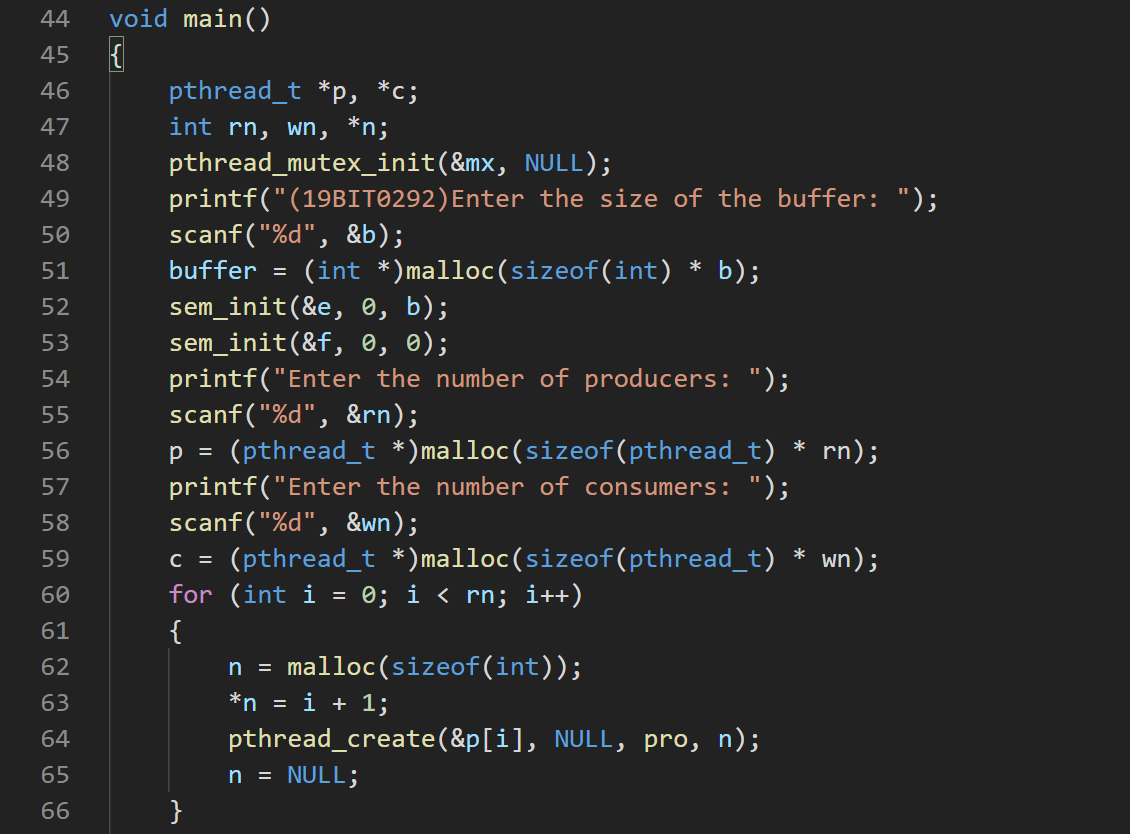
sem\_destroy(&e);

sem\_destroy(&f);

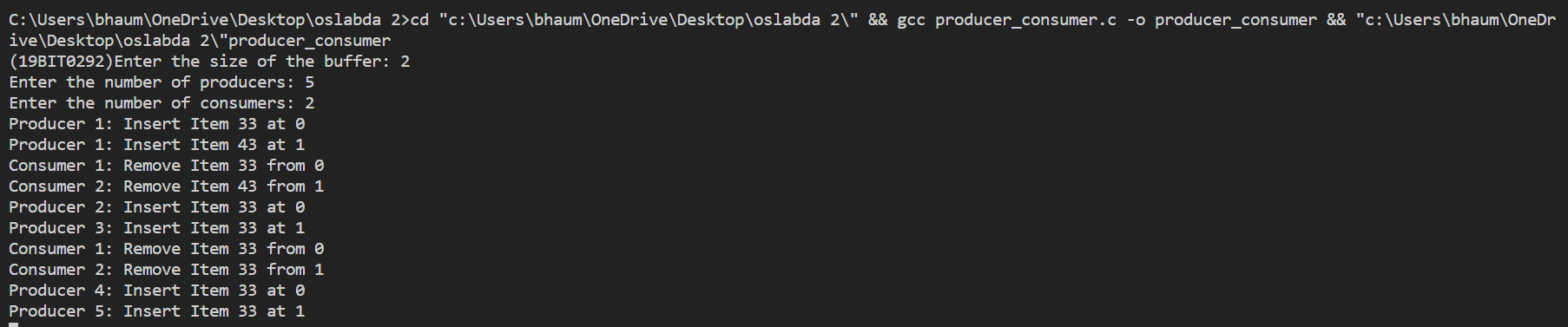
}

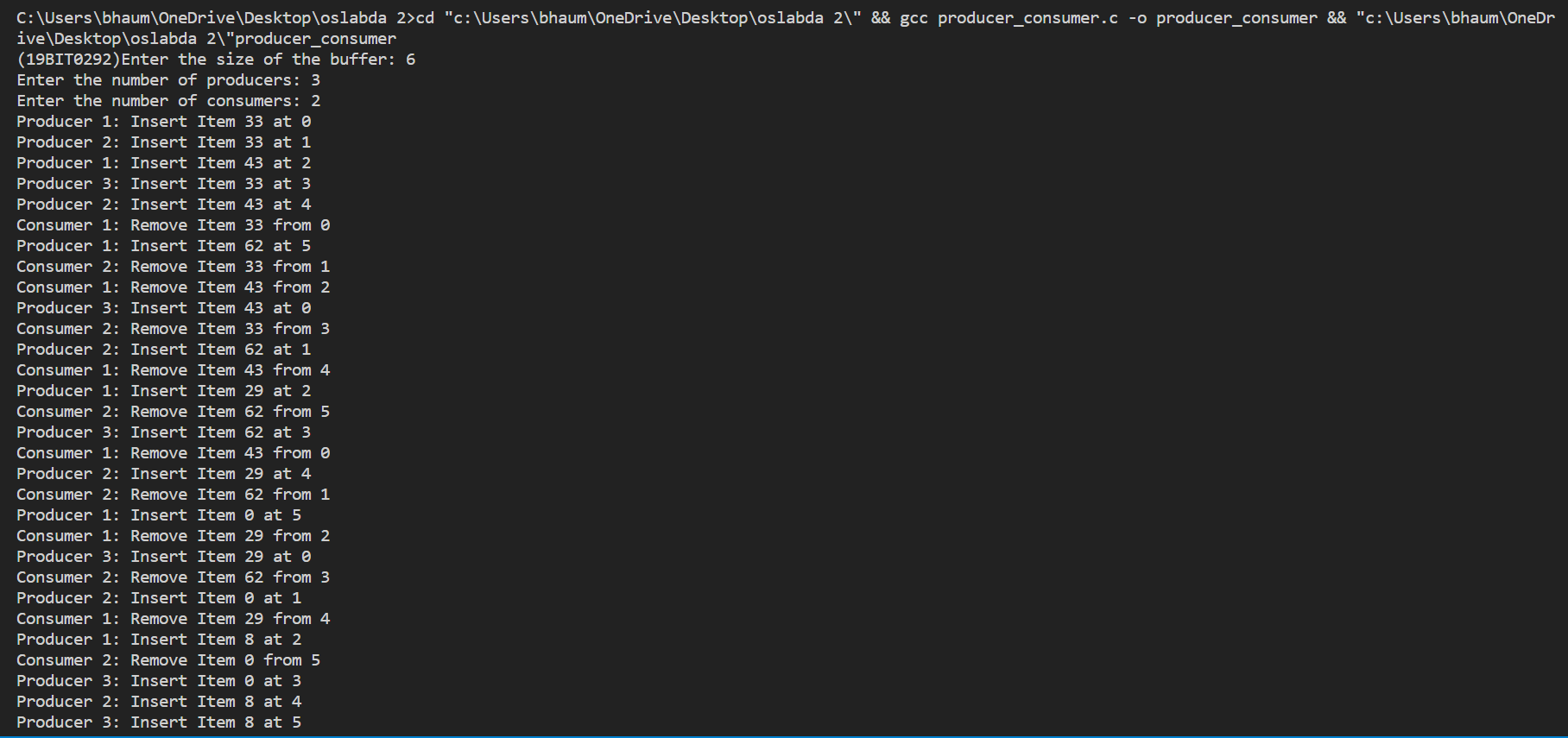
**SCREENSHOT OF CODE**

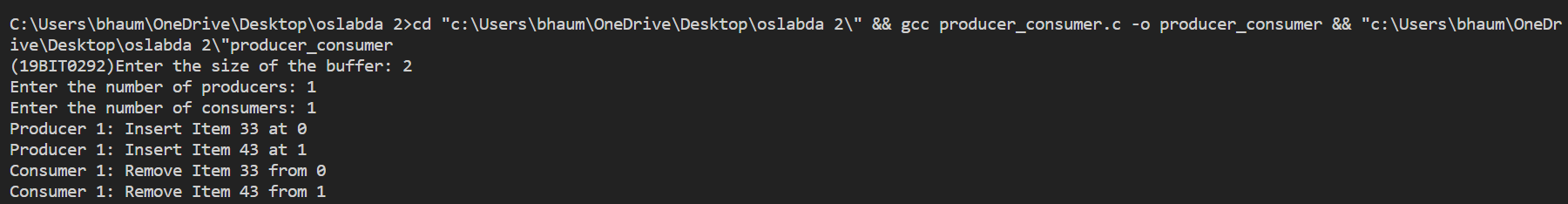
****

****

**OUTPUT**

****

****

****

## **Q2.** Write a Program to implement the solution for dining philosopher’s problem.

****

**CODE**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

int n;

int \*s;

sem\_t mutex;

sem\_t \*S;

void test(int phnum)

{

if (s[phnum] == 1 && s[(phnum + 4) % n] != 0 && s[(phnum + 1) % n] != 0)

{

s[phnum] = 0;

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, phnum + 1, (phnum + 4) % n + 1);

printf("Philosopher %d is eating\n", phnum + 1);

sem\_post(&S[phnum]);

}

}

void take\_fork(int phnum)

{

sem\_wait(&mutex);

s[phnum] = 1;

test(phnum);

sem\_post(&mutex);

sem\_wait(&S[phnum]);

}

void put\_fork(int phnum)

{

sem\_wait(&mutex);

s[phnum] = 2;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, (phnum + 4) % n + 1, phnum + 1);

printf("Philosopher %d has finished eating and now thinking again\n", phnum + 1);

test((phnum + 4) % n);

test((phnum + 1) % n);

sem\_post(&mutex);

}

void \*philospher(void \*num)

{

int \*i = num;

take\_fork(\*i);

put\_fork(\*i);

}

int main()

{

int i, \*p;

printf("(19BIT0292)Enter the total number of philosophers: ");

scanf("%d", &n);

pthread\_t thread\_id[n];

S = malloc(sizeof(sem\_t) \* n);

s = malloc(sizeof(int) \* n);

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

p[i] = i;

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

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

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

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

{

p = malloc(sizeof(int));

\*p = i + 1;

pthread\_create(&thread\_id[i], NULL,

philospher, p);

printf("Philosopher %d is thinking\n", i + 1);

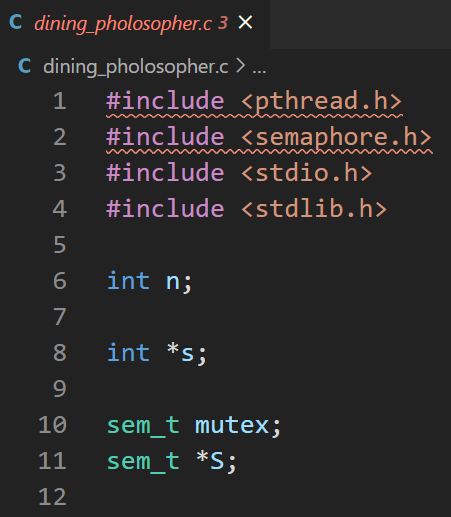
}

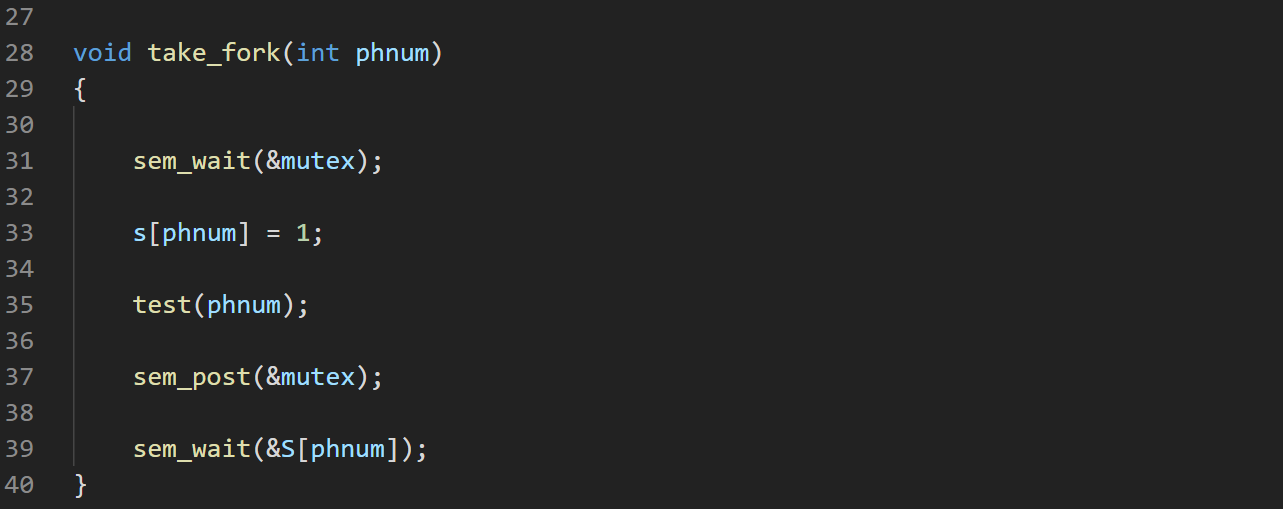
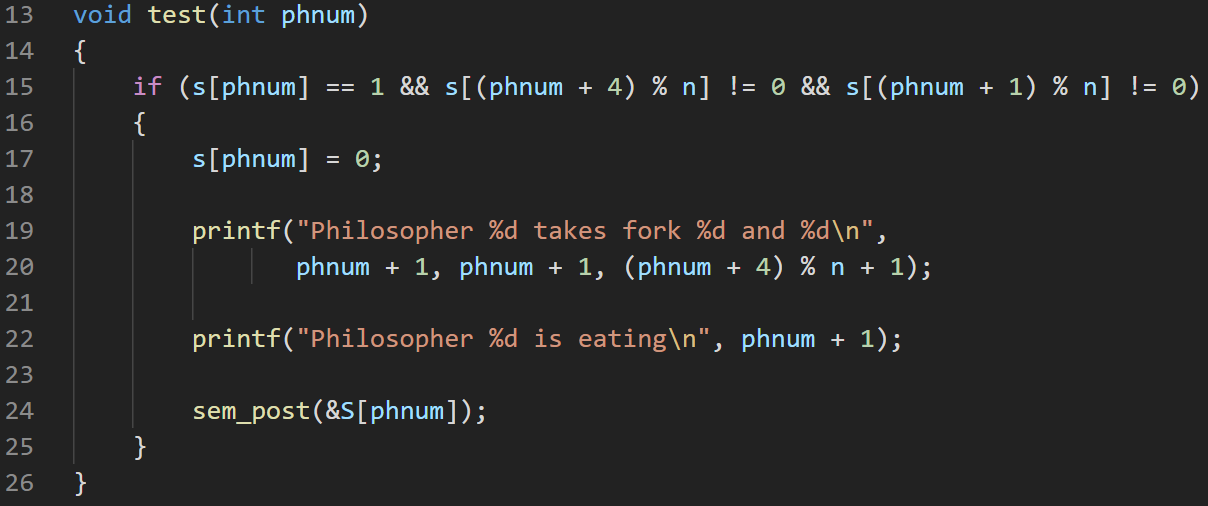
for (i = 0; i < n; i++)

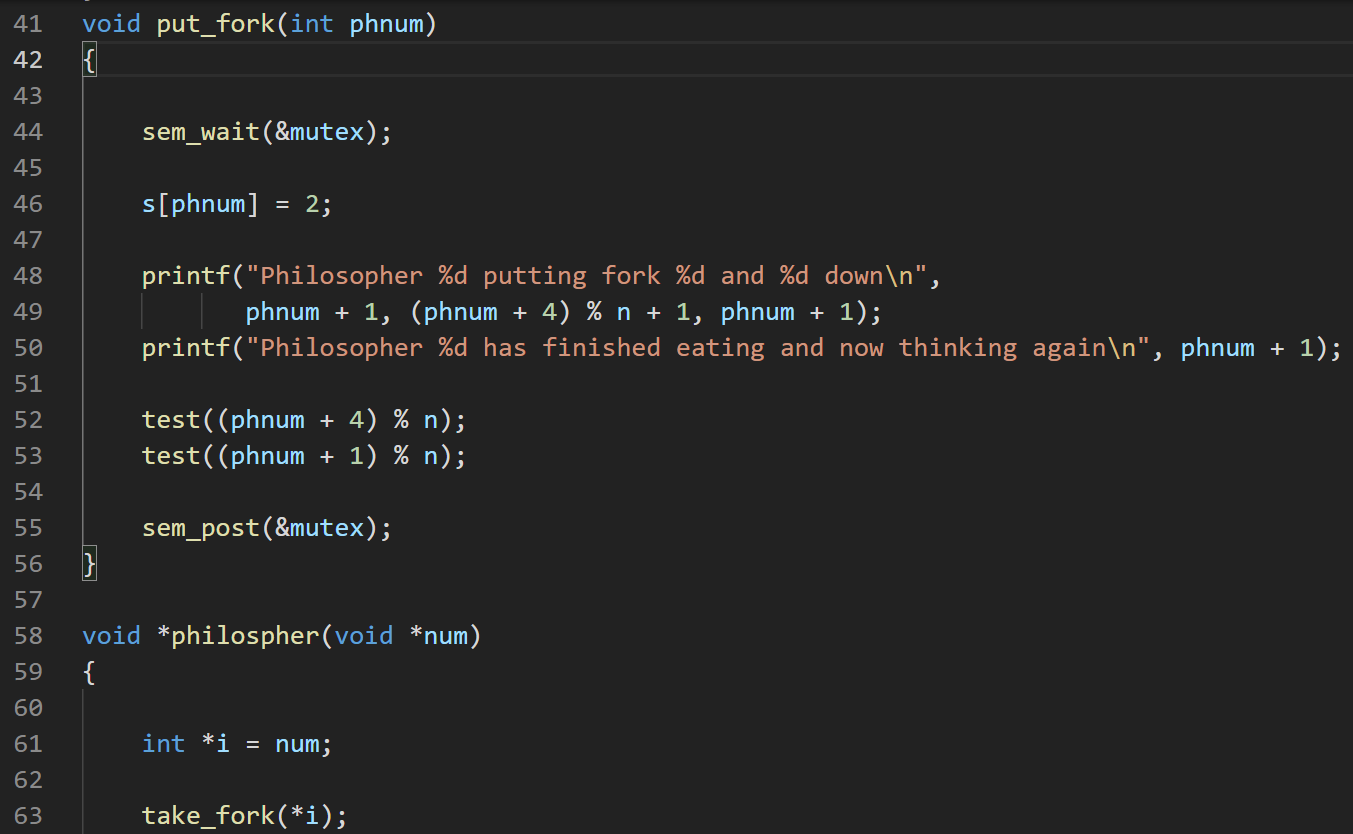
pthread\_join(thread\_id[i], NULL);

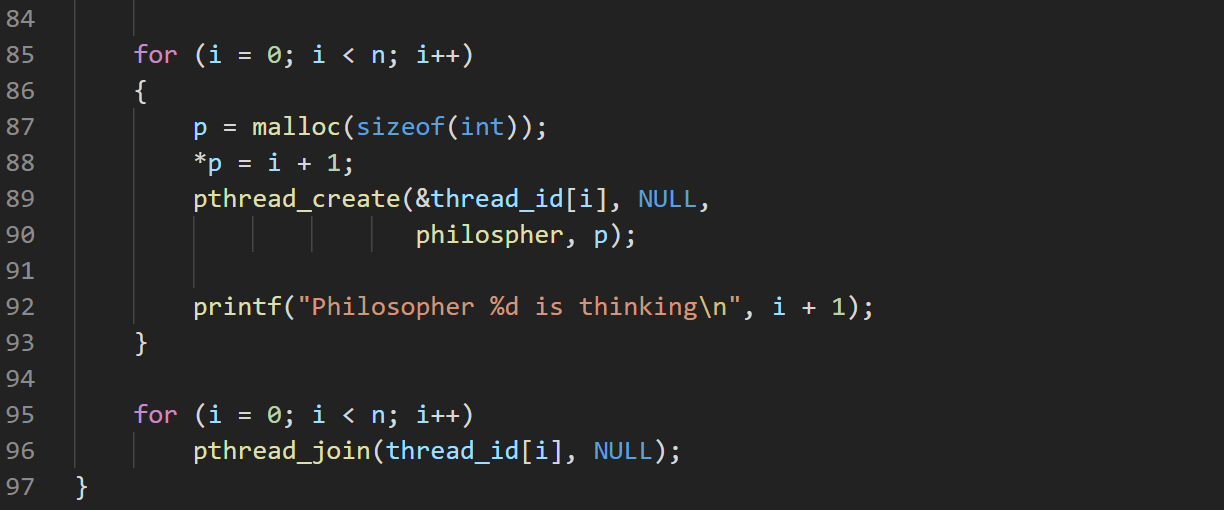
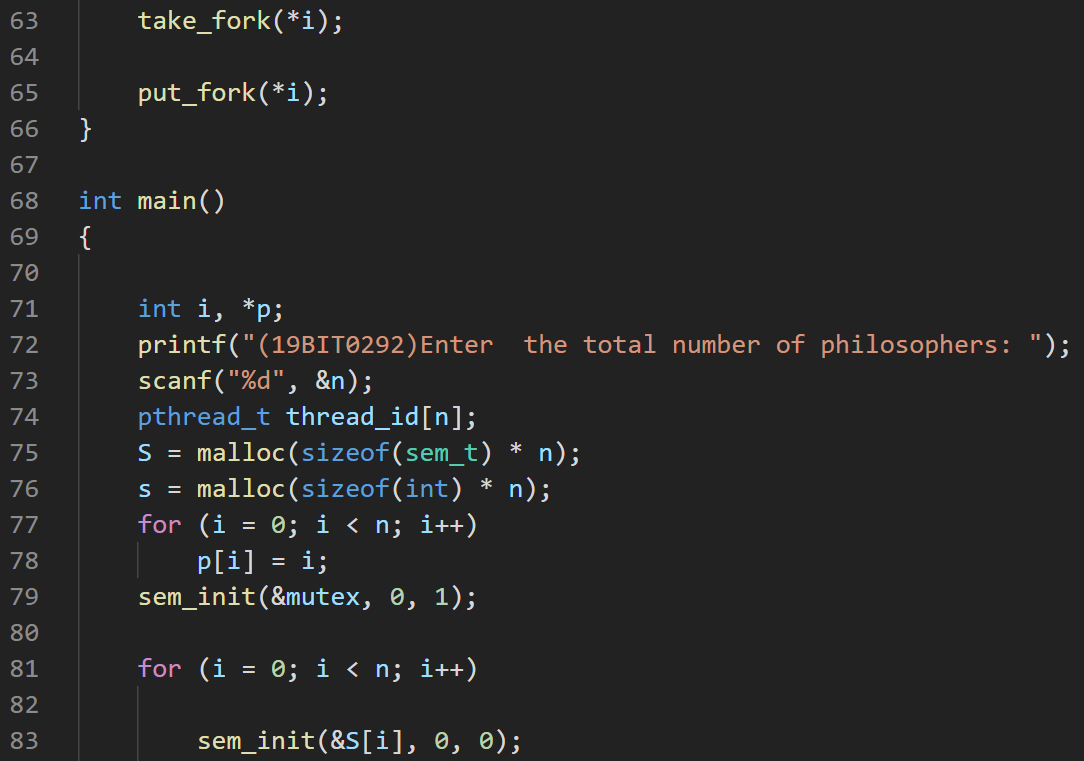
}

**SCREENSHOT OF CODE**

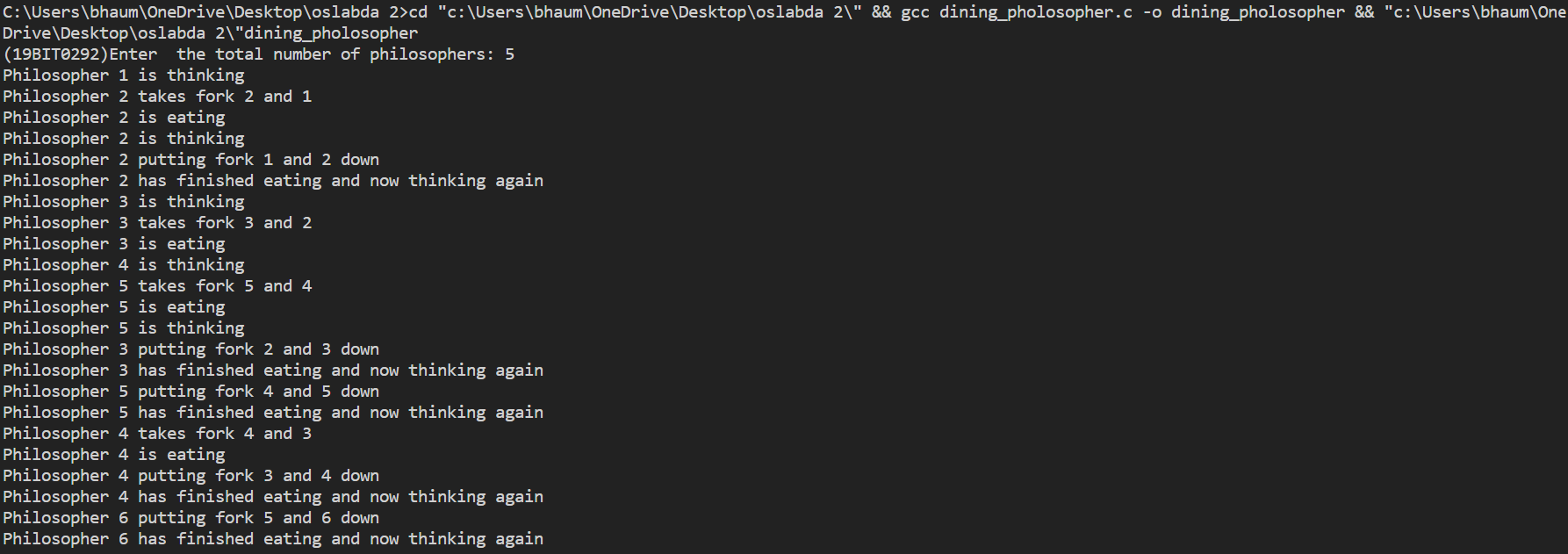


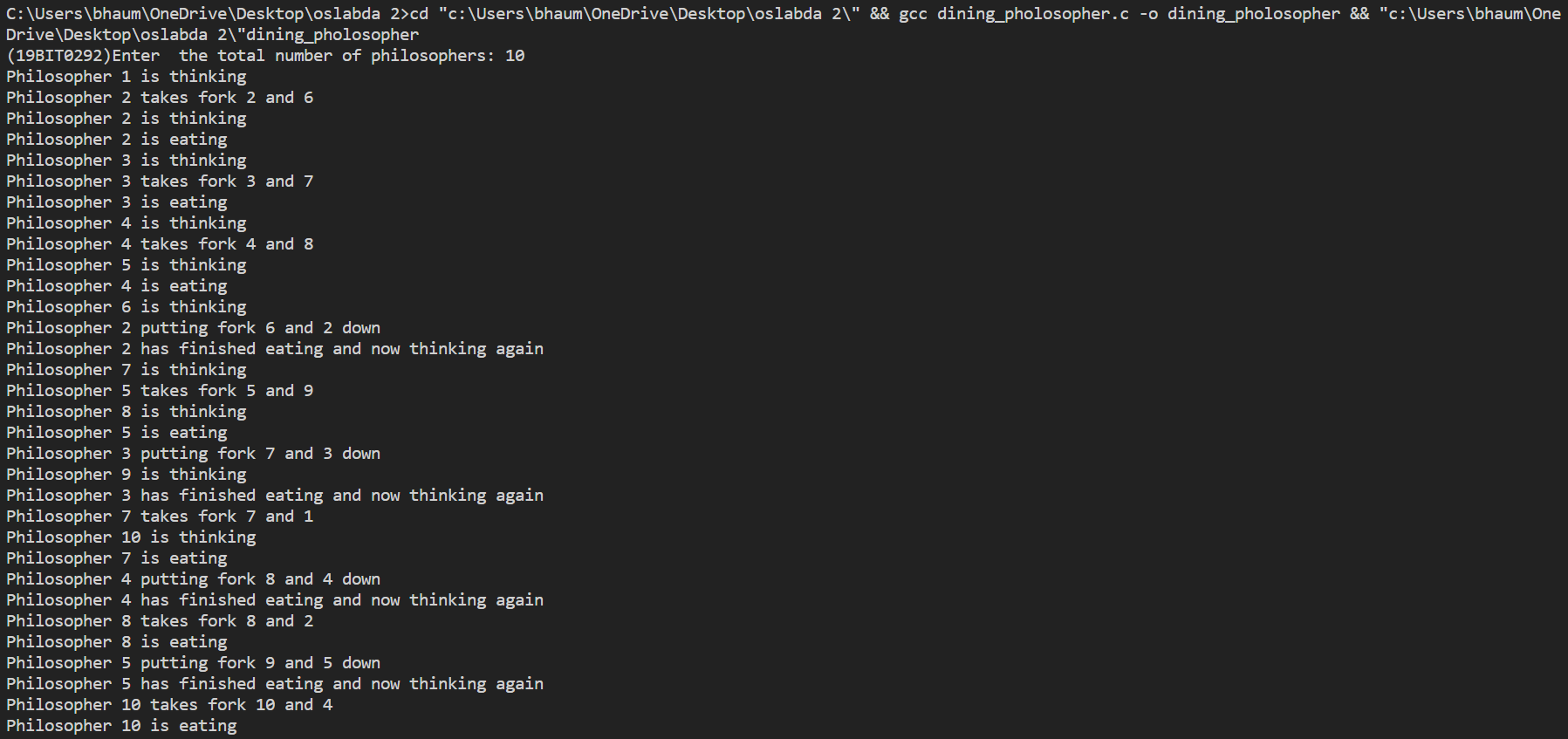






**OUTPUT**





## **Q3. Write a program to implement the solution for Readers Writers Problem using semaphores.**

**CODE**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

sem\_t wrt;

pthread\_mutex\_t mutex;

int cnt = 1;

int numreader = 0;

void \*wr(void \*wno)

{

sem\_wait(&wrt);

cnt = cnt \* 2;

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

sem\_post(&wrt);

}

void \*rd(void \*rno)

{

pthread\_mutex\_lock(&mutex);

numreader++;

if (numreader == 1)

{

sem\_wait(&wrt);

}

pthread\_mutex\_unlock(&mutex);

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

pthread\_mutex\_lock(&mutex);

numreader--;

if (numreader == 0)

{

sem\_post(&wrt);

}

pthread\_mutex\_unlock(&mutex);

}

void main()

{

pthread\_t \*r, \*w;

int rn, wn, \*n;

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

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

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

scanf("%d", &rn);

r = (pthread\_t \*)malloc(sizeof(pthread\_t) \* rn);

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

scanf("%d", &wn);

w = (pthread\_t \*)malloc(sizeof(pthread\_t) \* wn);

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

{

n = malloc(sizeof(int));

\*n = i + 1;

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

}

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

{

n = malloc(sizeof(int));

\*n = i + 1;

pthread\_create(&w[i], NULL, (void \*)wr, (void \*)n);

}

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

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

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

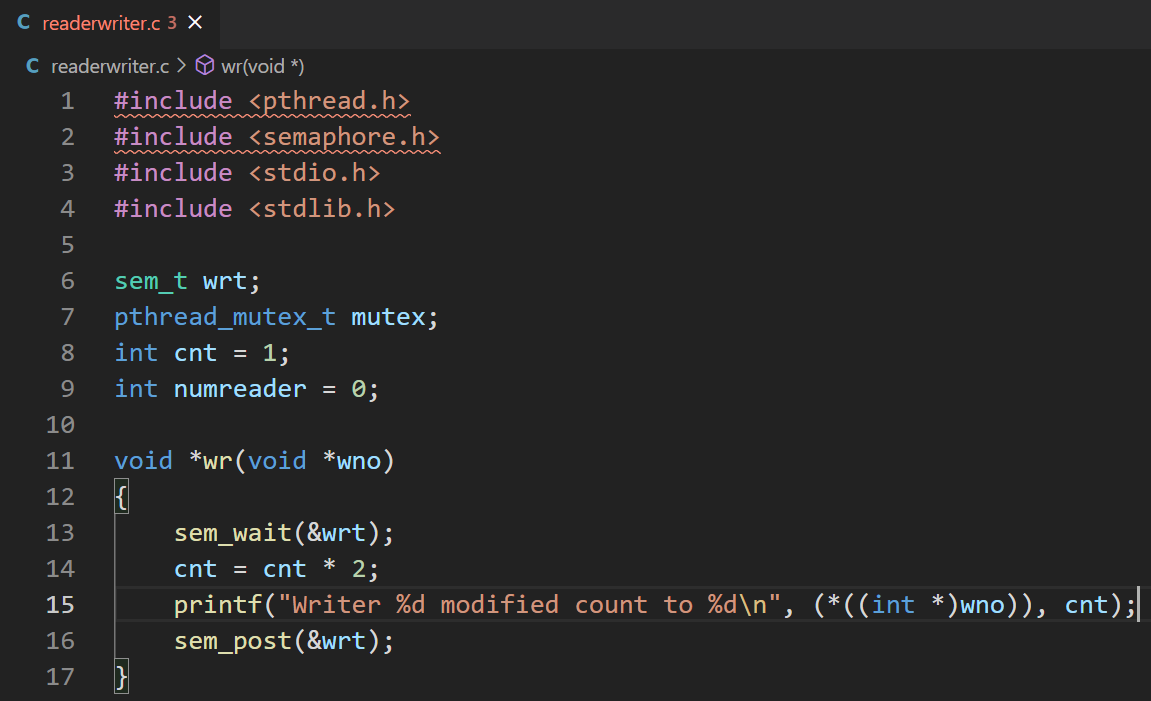
pthread\_join(w[i], NULL);

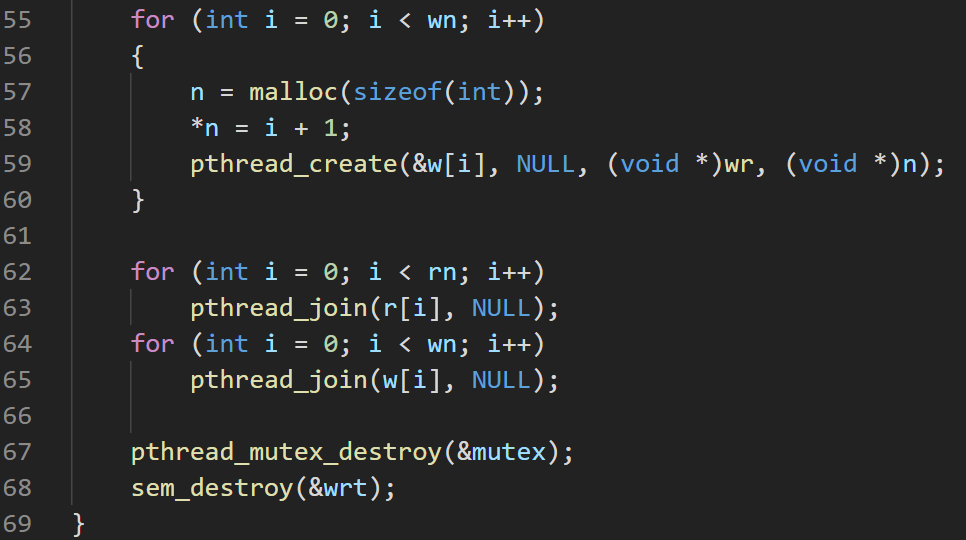
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

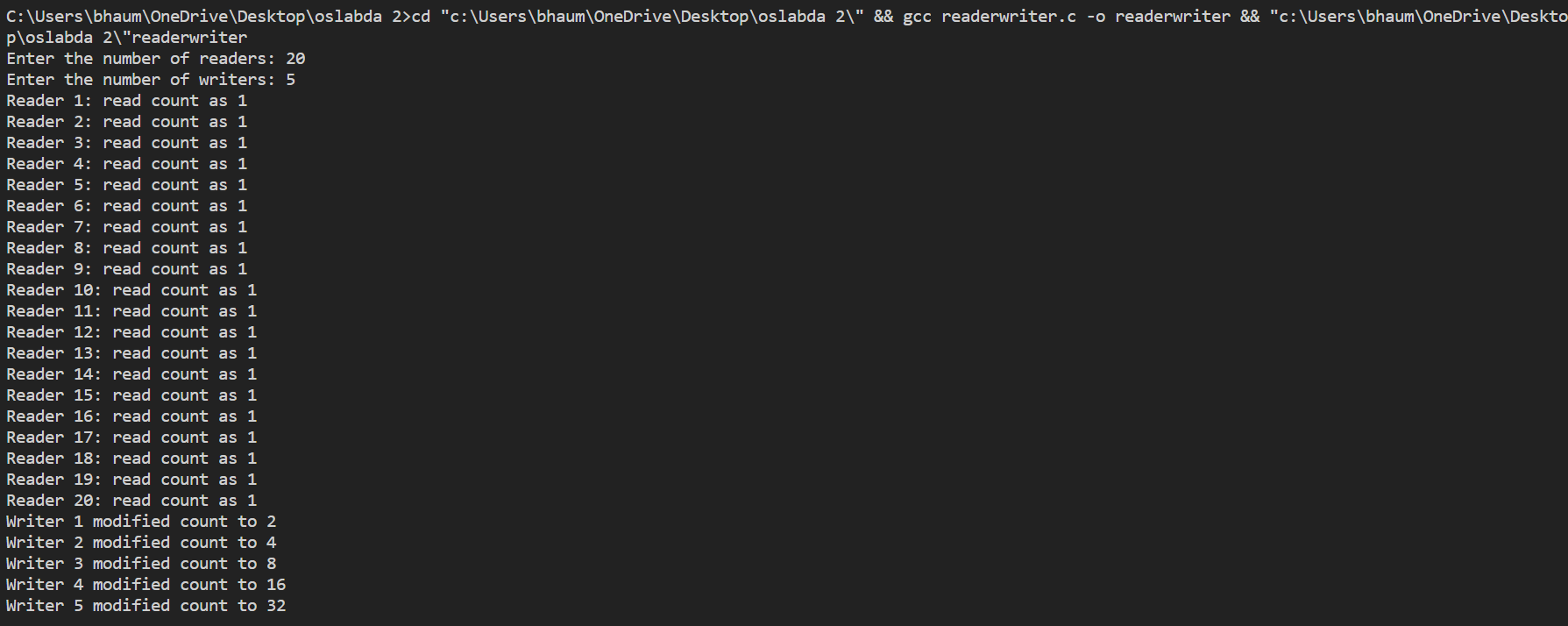
}

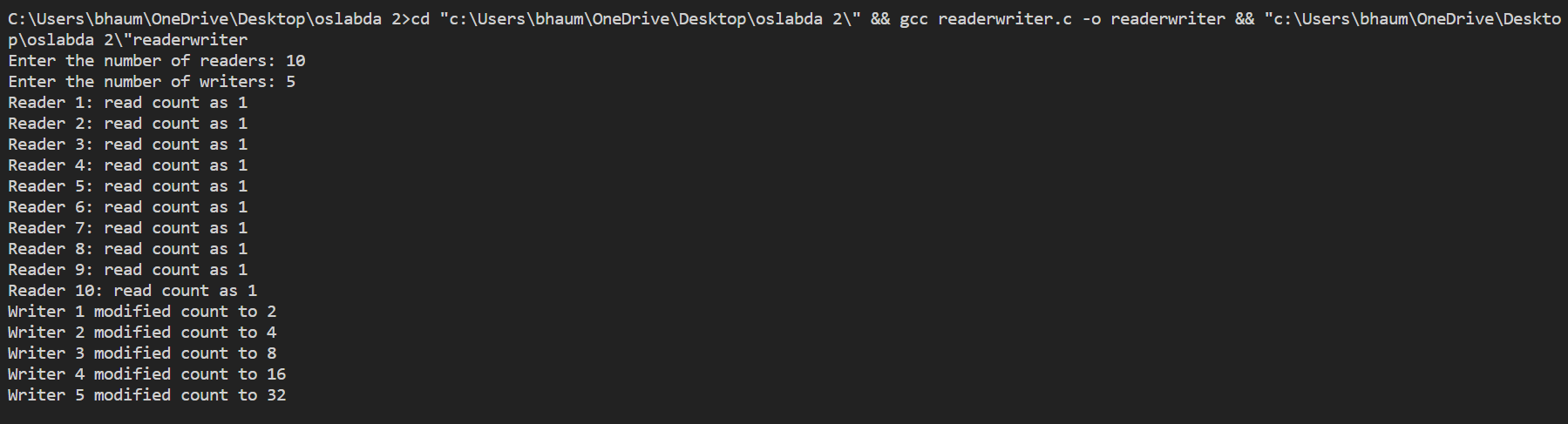
**SCREENSHOT OF CODE**

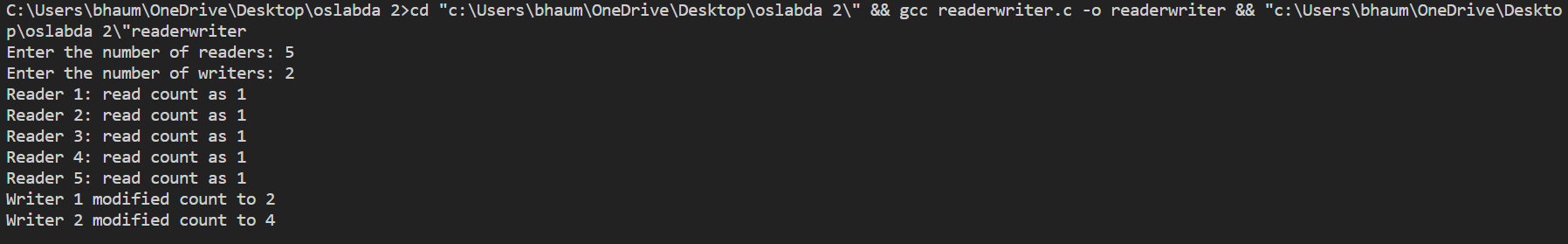




**OUTPUT**







## **Q4. Write a Program to implement banker’s algorithm for Deadlock avoidance.**

**CODE**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int n, r, i, j, f, fl, k = -1, \*re, \*\*p, \*\*mx, \*\*ne, \*s, \*jk;

printf("(19BIT0292)Enter the number of processes: ");

scanf("%d", &n);

p = (int \*\*)malloc(sizeof(int \*) \* n);

mx = (int \*\*)malloc(sizeof(int \*) \* n);

ne = (int \*\*)malloc(sizeof(int \*) \* n);

jk = (int \*)malloc(sizeof(int) \* n);

printf("(19BIT0292)Number of types of resources: ");

scanf("%d", &r);

re = (int \*)malloc(sizeof(int) \* r);

s = (int \*)calloc(r, sizeof(int));

printf("\n(19BIT0292)Enter the process allocation matrix\n");

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

{

p[i] = (int \*)malloc(sizeof(int) \* r);

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

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

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

}

printf("\n(19BIT0292)Enter the process maximum matrix\n");

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

{

mx[i] = (int \*)malloc(sizeof(int) \* r);

ne[i] = (int \*)malloc(sizeof(int) \* r);

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

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

{

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

ne[i][j] = mx[i][j] - p[i][j];

}

}

printf("\n(19BIT0292)Need Matrix\n");

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

{

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

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

{

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

s[j] += ne[i][j];

}

}

printf("\n(19BIT0292)Total number of resources used uptil now(in sequece): ");

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

{

printf("%d ", s[i]);

}

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

{

printf("\n(19BIT0292)Enter the remaing instance of resouces number %d: ", i + 1);

scanf("%d", re + i);

}

printf("(19BIT0292)Sequence: \n");

while (1)

{

fl = 0;

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

{

if (p[i] == NULL)

continue;

else

{

f = 0;

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

if (ne[i][j] > re[j])

{

f = 1;

break;

}

if (f == 0)

{

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

for (j = 0; j < r - 1; j++)

{

re[j] += p[i][j];

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

}

re[r - 1] += p[i][r - 1];

printf("%d)\n", re[r - 1]);

p[i] = NULL;

fl = 1;

jk[++k] = i + 1;

continue;

}

}

}

if (fl == 0)

break;

}

if (k == -1)

printf("\n\n(19BIT0292)Unsafe State");

else

{

printf("\n(19BIT0292)Safe State sequence <P%d", jk[0]);

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

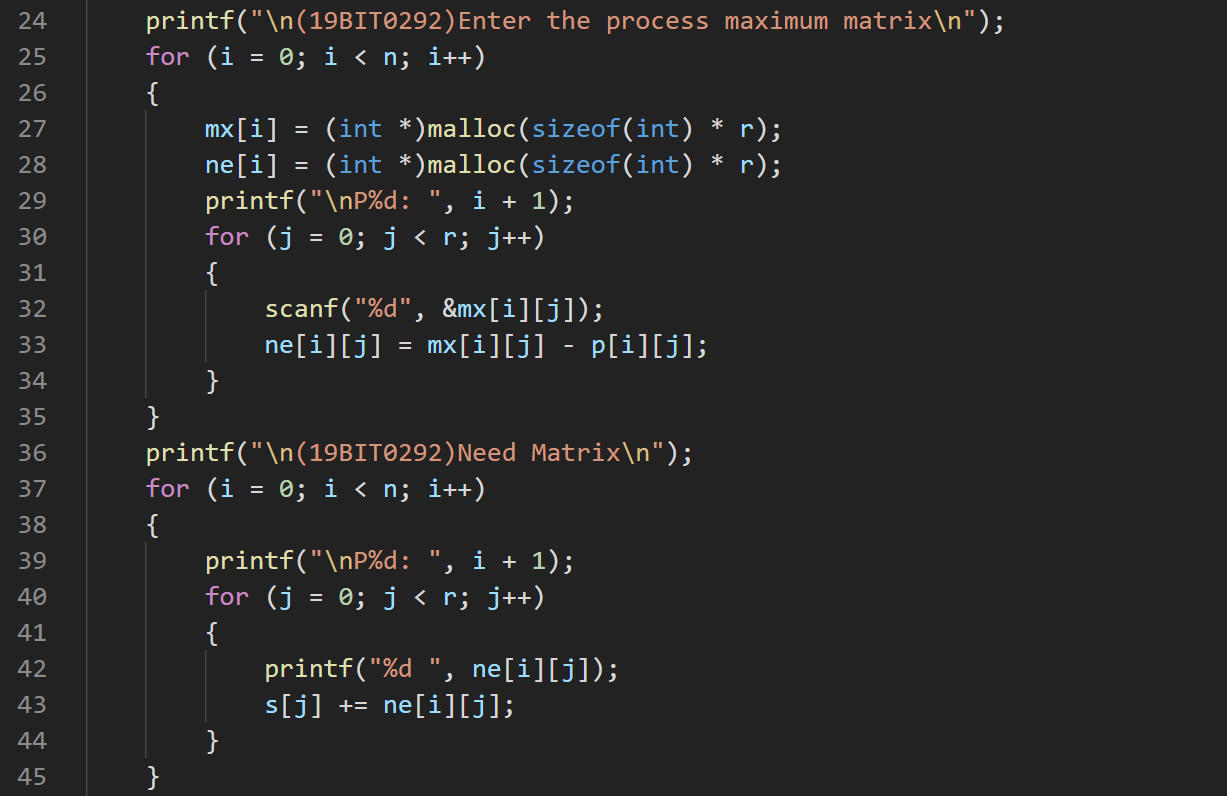
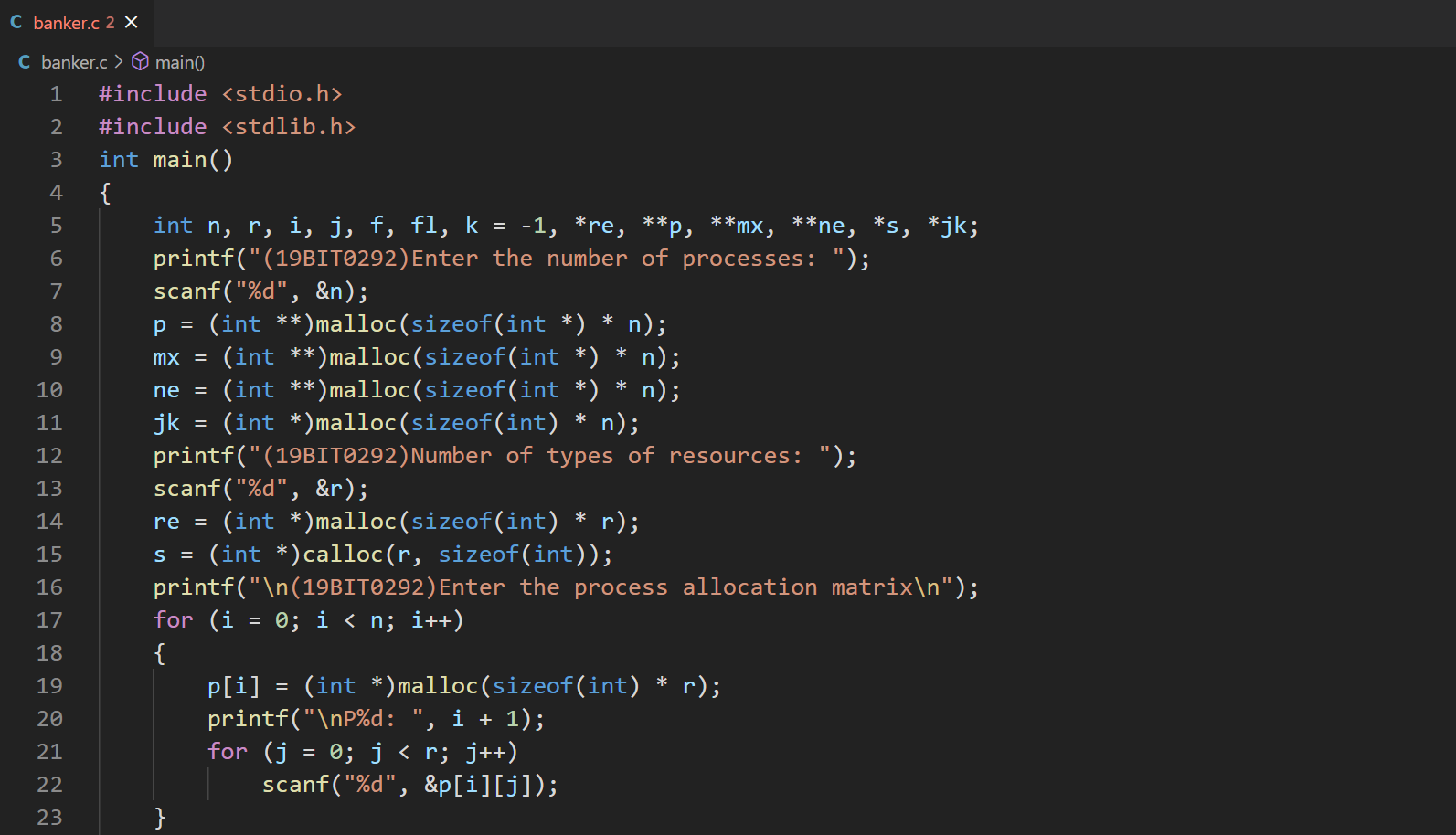
printf(",P%d", jk[k]);

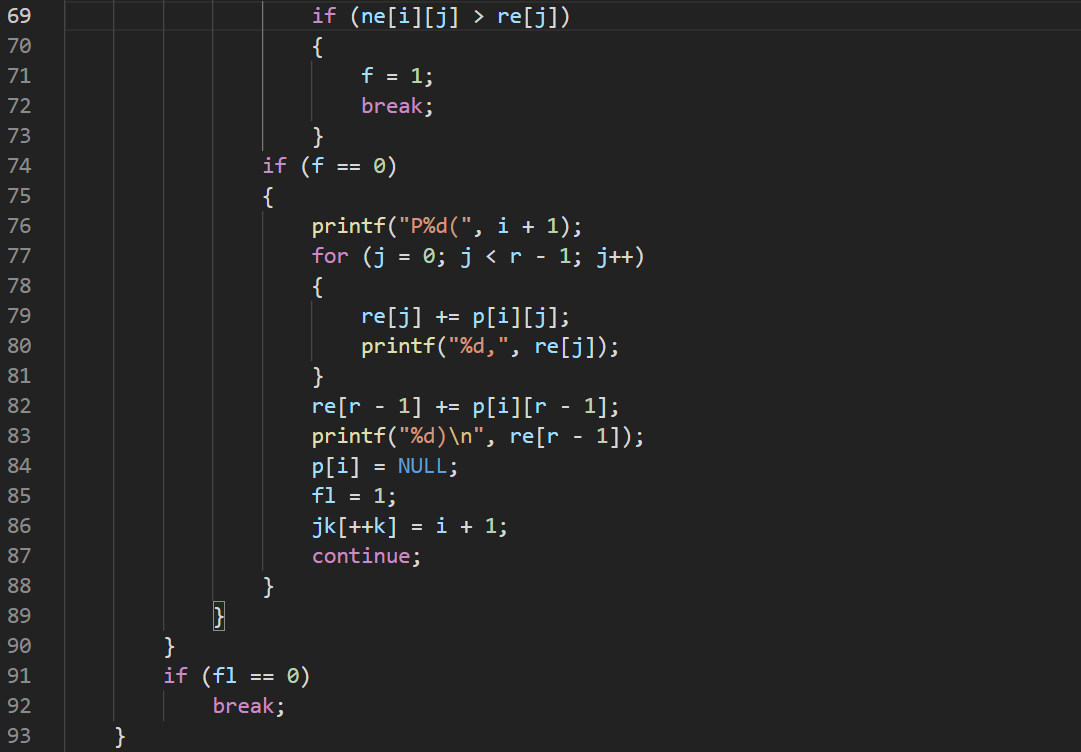
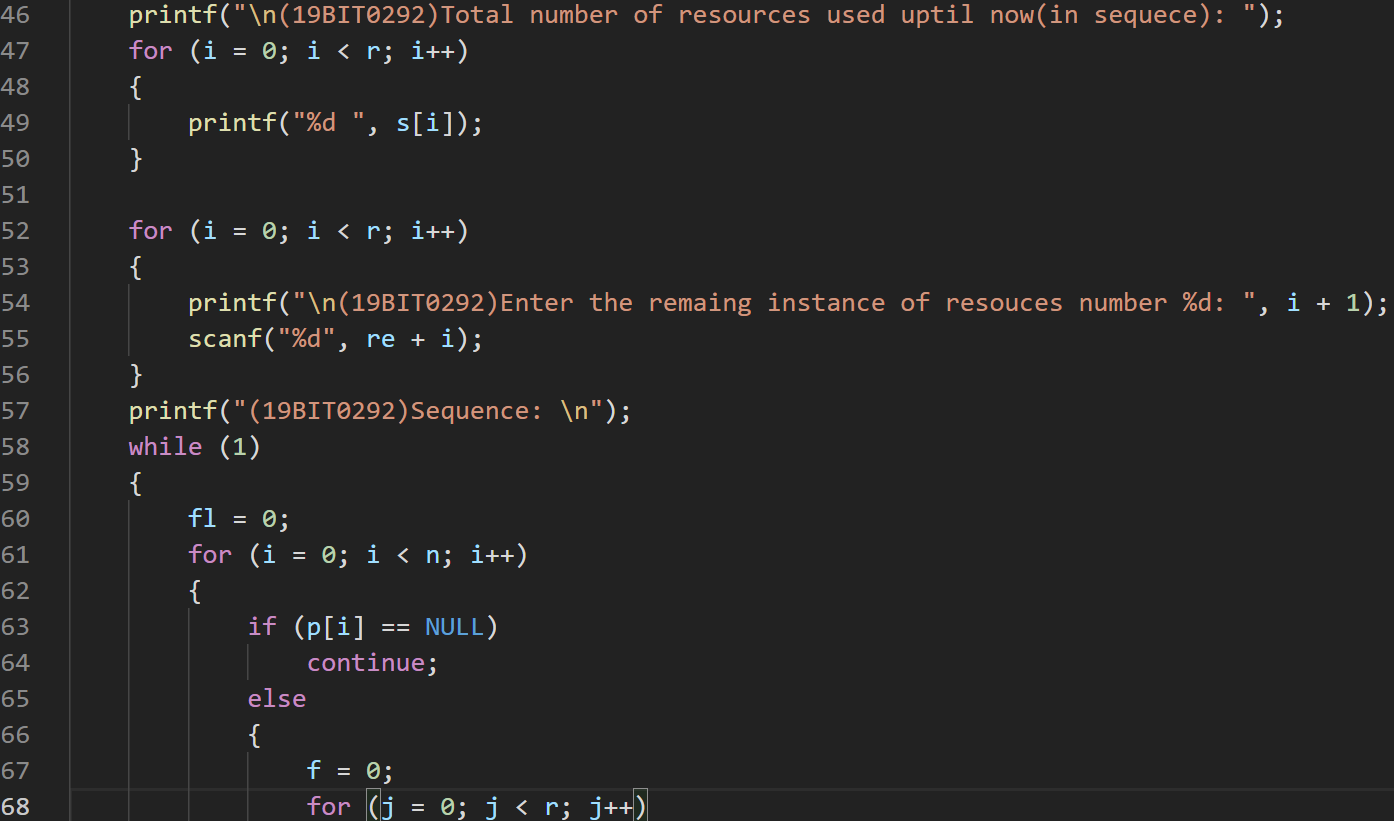
printf(">");

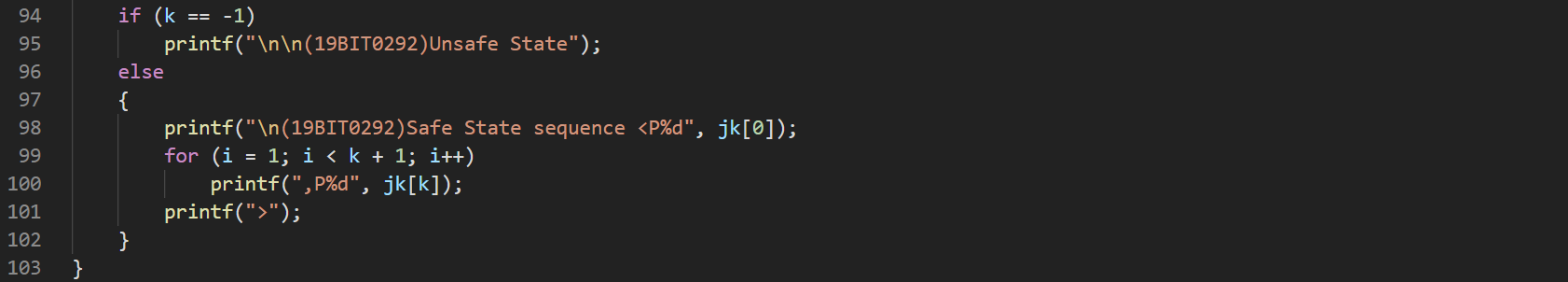
}

}

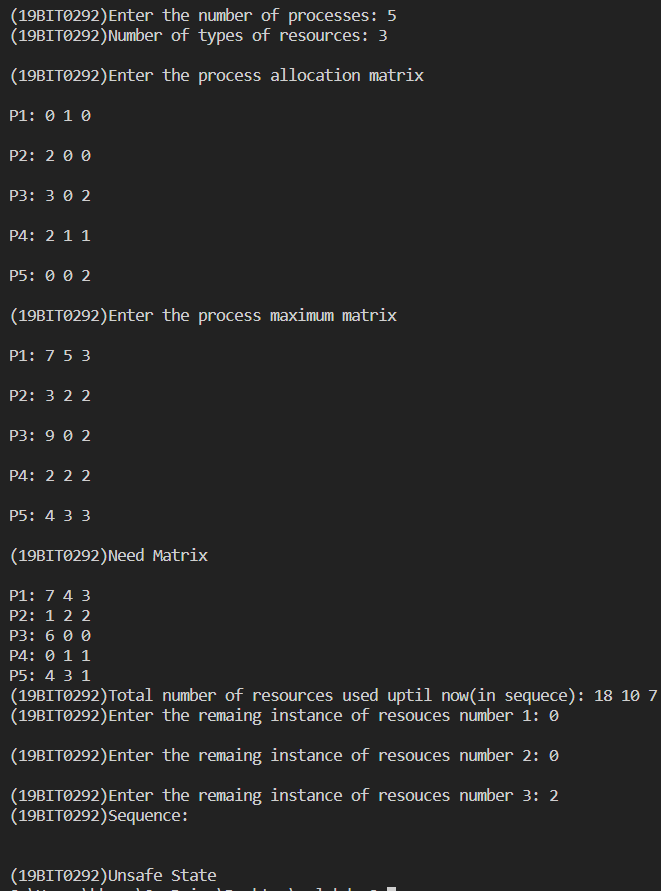
**SCREENSHOT OF CODE**

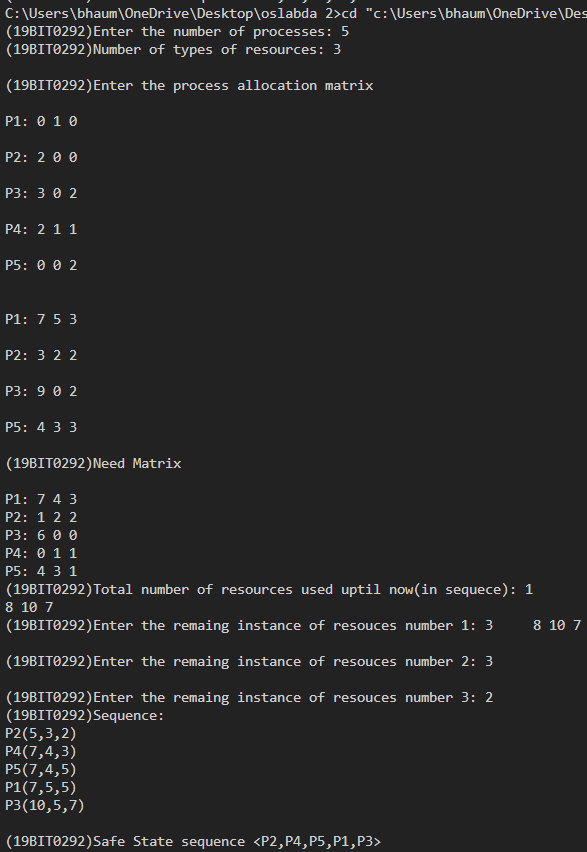






**OUTPUT**





[**CLICK HERE FOR GITHUBB LINK**](https://github.com/Bhaumik-Tandan/OS-LAB-DA-2-SYCHRONISATION-.git)