**Name: Srinivas Sanjeevkumar Chenn**

**Division: A**

**Roll Number: 48**

**Subject: Operating System (OS) LAB**

**Lab Assignment 2**

**Producer-Consumer Problem**

1. **Producer consumer using mutex**

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

#define MaxItems 5

#define BufferSize 5

sem\_t empty;

sem\_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread\_mutex\_t mutex;

void \*producer(void \*pno)

{

int item;

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

item = rand();

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = item;

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

in = (in+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

}

void \*consumer(void \*cno)

{

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

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[out];

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

out = (out+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

}

int main()

{

pthread\_t pro[5],con[5];

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

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

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

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

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

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

}

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

pthread\_create(&con[i], NULL, (void \*)consumer, (void \*)&a[i]);

}

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

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

}

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

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

}

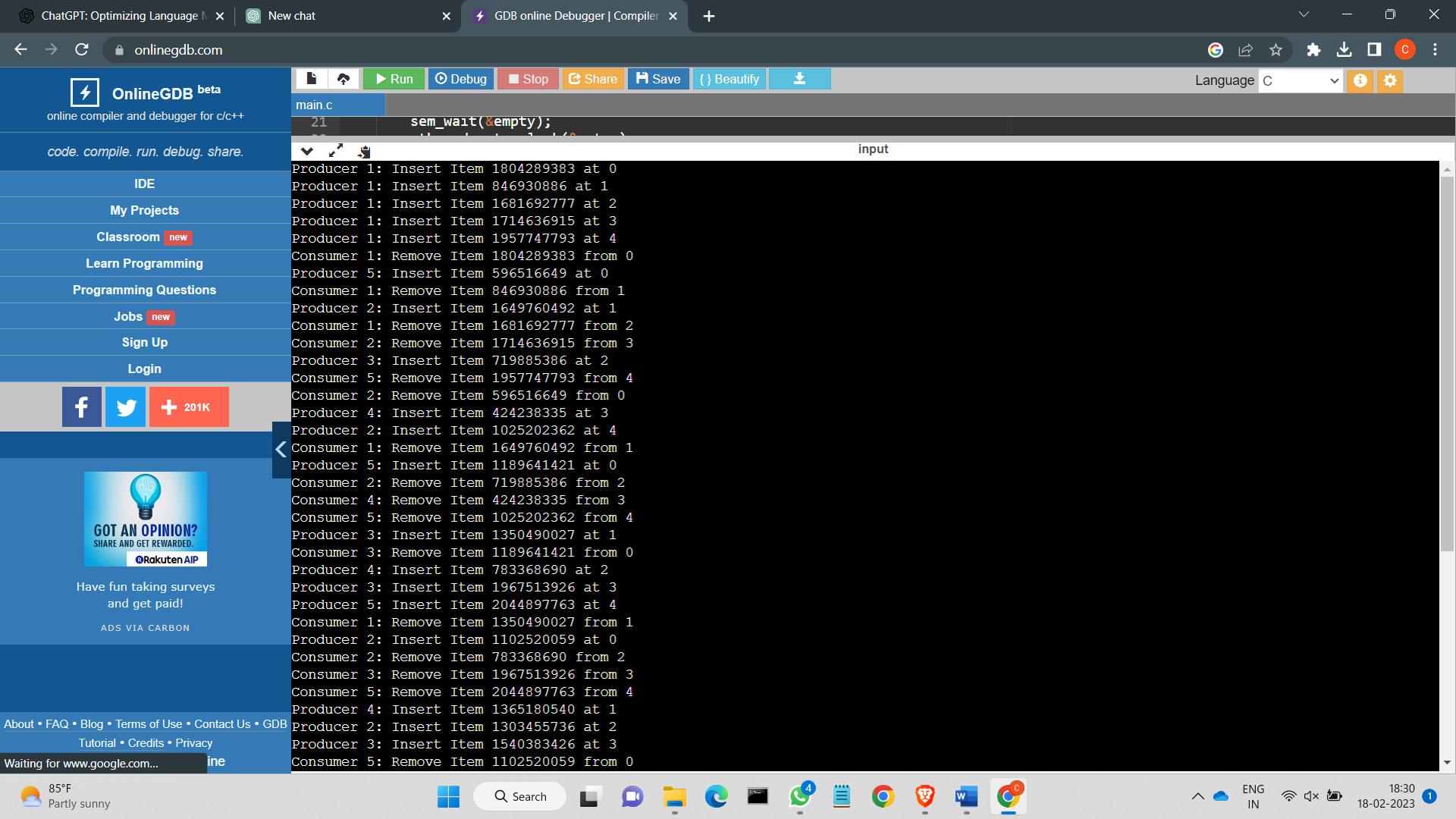
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}



1. **Producer consumer using semaphore**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define buffersize 20

sem\_t empty;

sem\_t full;

int in = 0, out = 0;

int buffer[buffersize];

void \*producer()

{

int data, temp;

temp = rand()%3;

while(1)

{

data = rand()%100;

sleep(temp);

sem\_wait(&empty);

buffer[in] = data;

printf("\n\nProducer Inserted Data %d at %d",data,in);

in = (in+1)%buffersize;

sem\_post(&full);

}

}

void \*consumer()

{

int temp, data;

temp = rand()%5;

while(1)

{

sleep(temp);

sem\_wait(&full);

data = buffer[out];

printf("\n\nConsumer Consumed Data %d from %d",data,out);

out = (out+1)%buffersize;

sem\_post(&empty);

}

}

int main()

{

pthread\_t prod, cons;

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

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

pthread\_create(&prod, NULL, &producer, NULL);

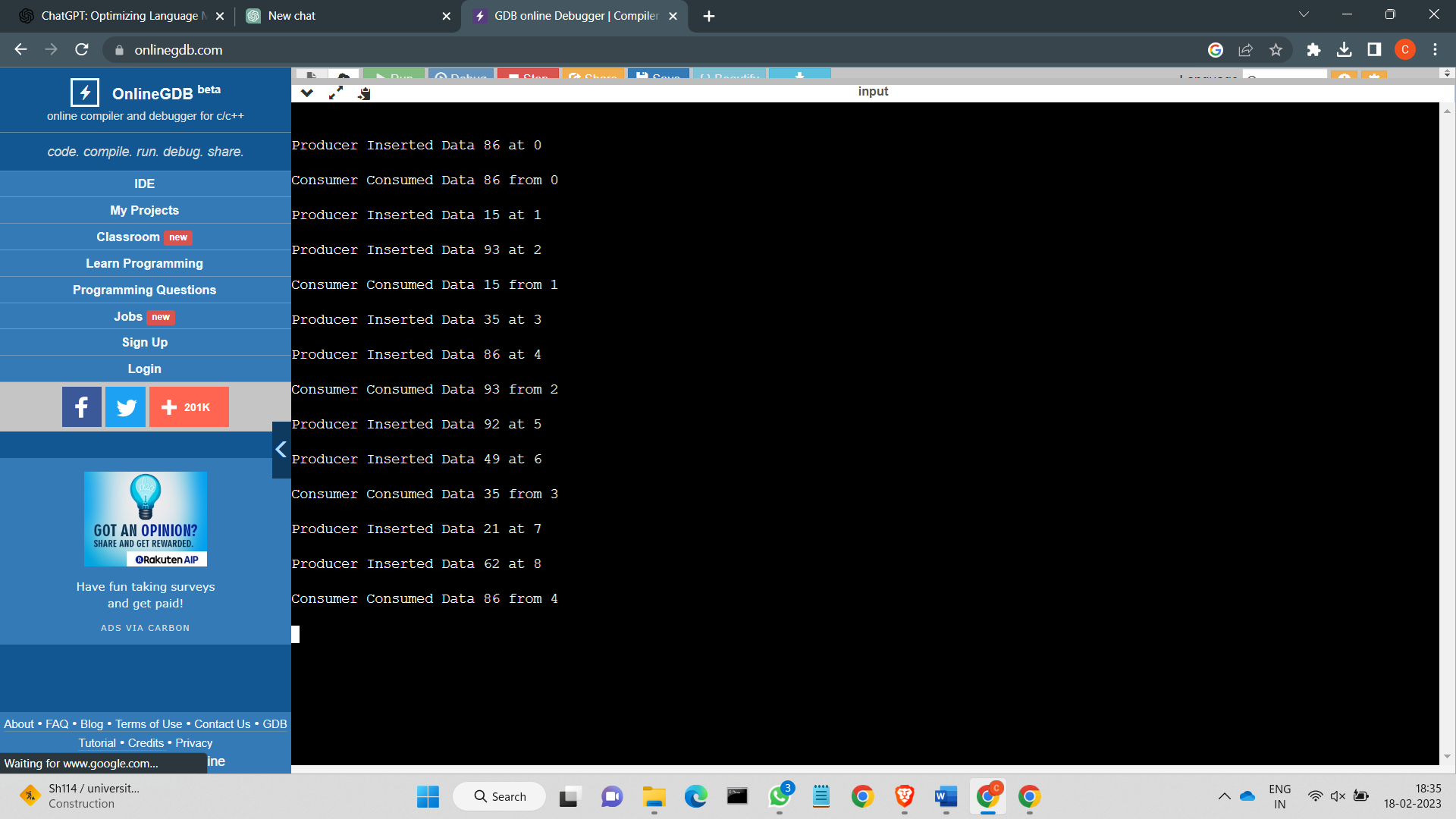
pthread\_create(&cons, NULL, &consumer, NULL);

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

return 0;

}



**Reader-Writer Problem**

1. **Reader writer using mutex**

#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);

}

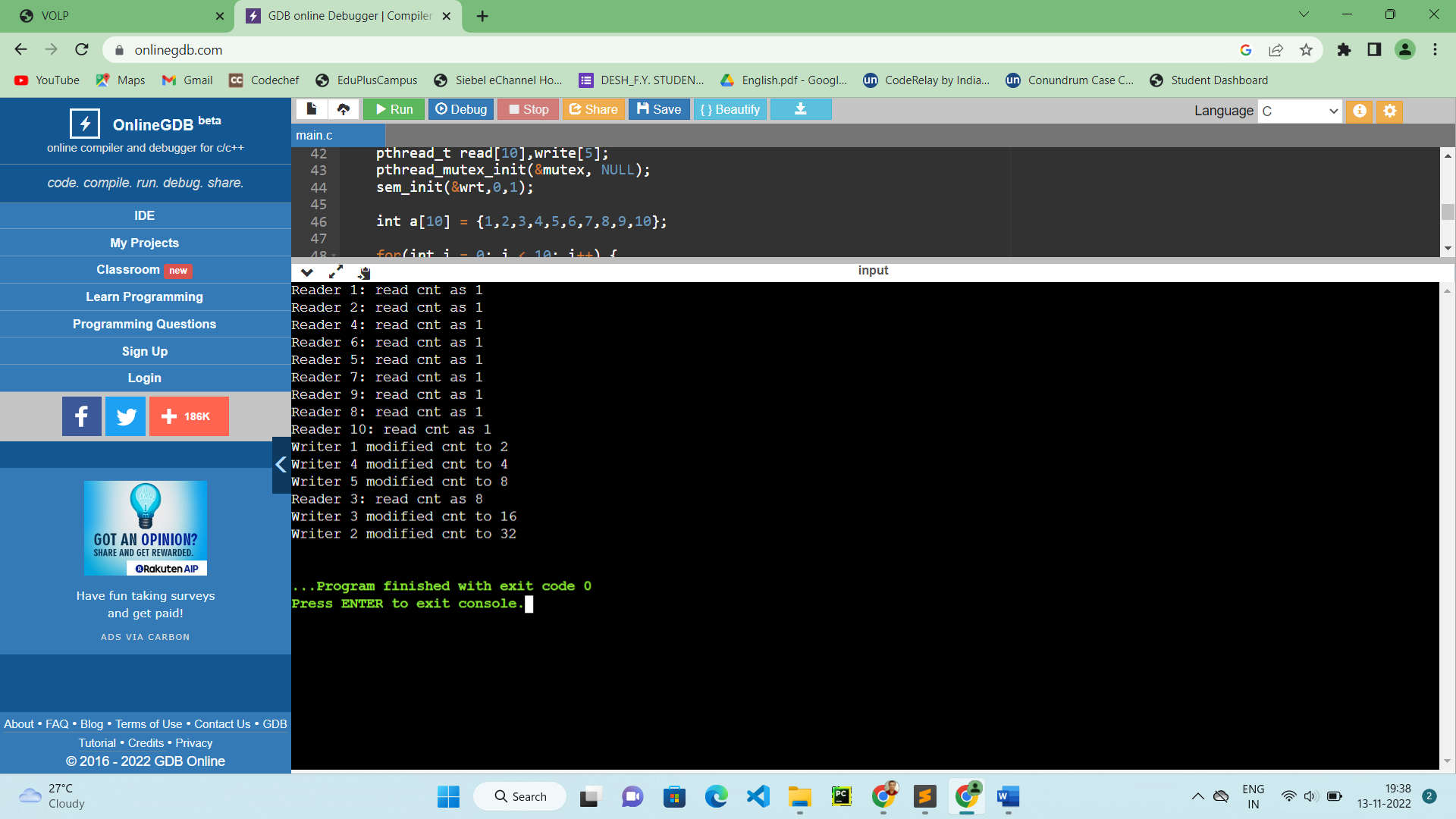
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

**Output –**



1. **Reader writer using semaphore**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t read\_sem, write\_sem;

int readcount = 0;

int data = 0;

void \*reader(void \*arg) {

int reader\_id = \*((int \*)arg);

while (1) {

sem\_wait(&read\_sem);

readcount++;

if (readcount == 1) {

sem\_wait(&write\_sem);

}

sem\_post(&read\_sem);

printf("Reader %d read data: %d\n", reader\_id, data);

sem\_wait(&read\_sem);

readcount--;

if (readcount == 0) {

sem\_post(&write\_sem);

}

sem\_post(&read\_sem);

sleep(1);

}

return NULL;

}

void \*writer(void \*arg) {

int writer\_id = \*((int \*)arg);

while (1) {

sem\_wait(&write\_sem);

data++;

printf("Writer %d wrote data: %d\n", writer\_id, data);

sem\_post(&write\_sem);

sleep(1);

}

return NULL;

}

int main() {

pthread\_t reader\_threads[5], writer\_threads[5];

int i;

sem\_init(&read\_sem, 0, 1);

sem\_init(&write\_sem, 0, 1);

int reader\_ids[5];

int writer\_ids[5];

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

reader\_ids[i] = i + 1;

writer\_ids[i] = i + 1;

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

pthread\_create(&writer\_threads[i], NULL, writer, (void \*)&writer\_ids[i]);

}

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

pthread\_join(reader\_threads[i], NULL);

pthread\_join(writer\_threads[i], NULL);

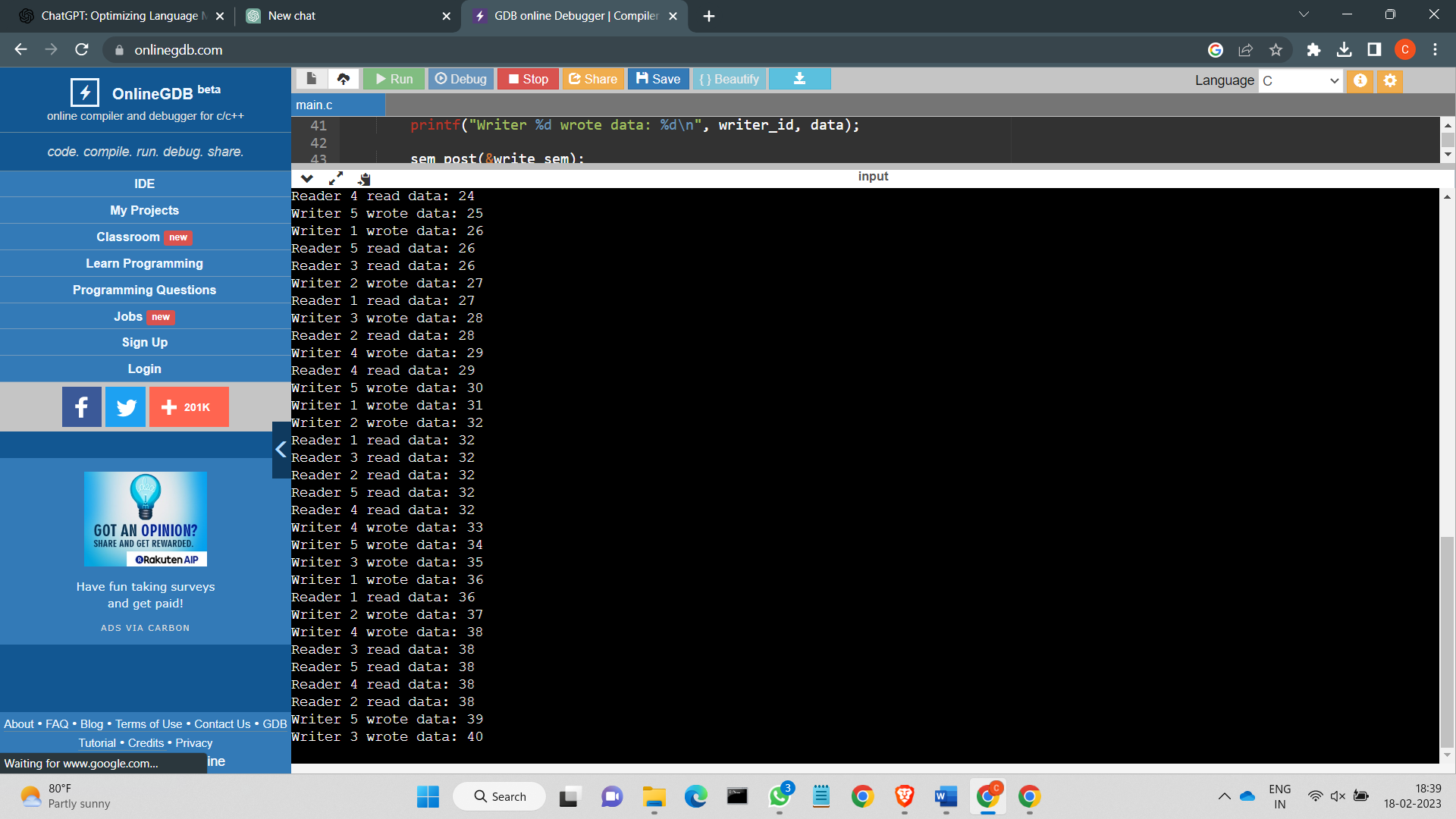
}

sem\_destroy(&read\_sem);

sem\_destroy(&write\_sem);

return 0;

}



**Dining Philosopher Problem**

#include <pthread.h>

#include <semaphore.h>

#include <bits/stdc++.h>

#include <iostrem.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING)

{

state[phnum] = EATING;

sleep(2);

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

phnum + 1, LEFT + 1, phnum + 1);

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

sem\_post(&S[phnum]);

}

}

void take\_fork(int phnum)

{

sem\_wait(&mutex);

state[phnum] = HUNGRY;

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

test(phnum);

sem\_post(&mutex);

sem\_wait(&S[phnum]);

sleep(1);

}

void put\_fork(int phnum)

{

sem\_wait(&mutex);

state[phnum] = THINKING;

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

phnum + 1, LEFT + 1, phnum + 1);

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

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philosopher(void\* num)

{

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

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

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

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

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

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

philosopher, &phil[i]);

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

}

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

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

}

**Output –**

