Assignment 5:
Implementation of synchronization using semaphore:

Objective of this Assignment:

- To implement the concept of multi-threading in a process.
- To learn the use of semaphore i.e., to control access to shared resources.

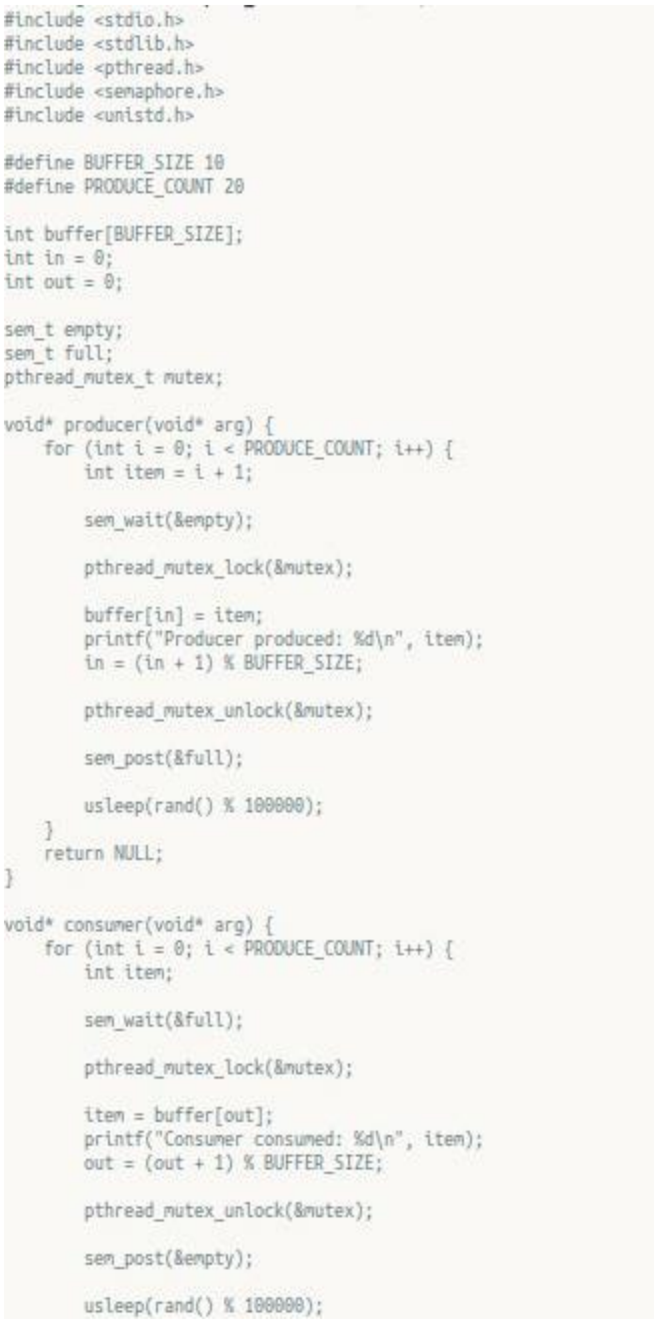
1. Producer-Consumer problem

Problem: Write a C program to implement the producer-consumer program where:

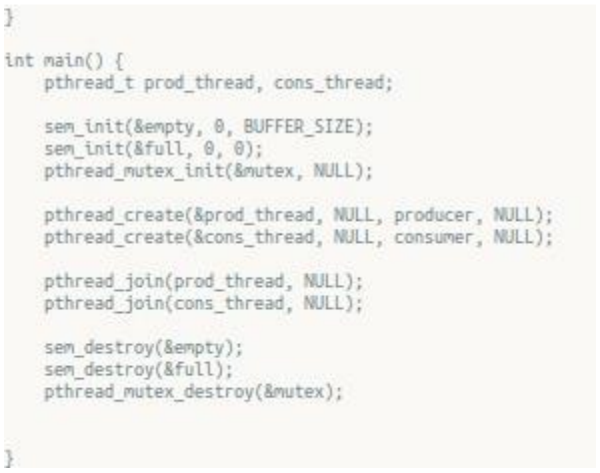
- Producer generates integers from 1 to 100.
- Consumer processes the numbers.

Requirements:

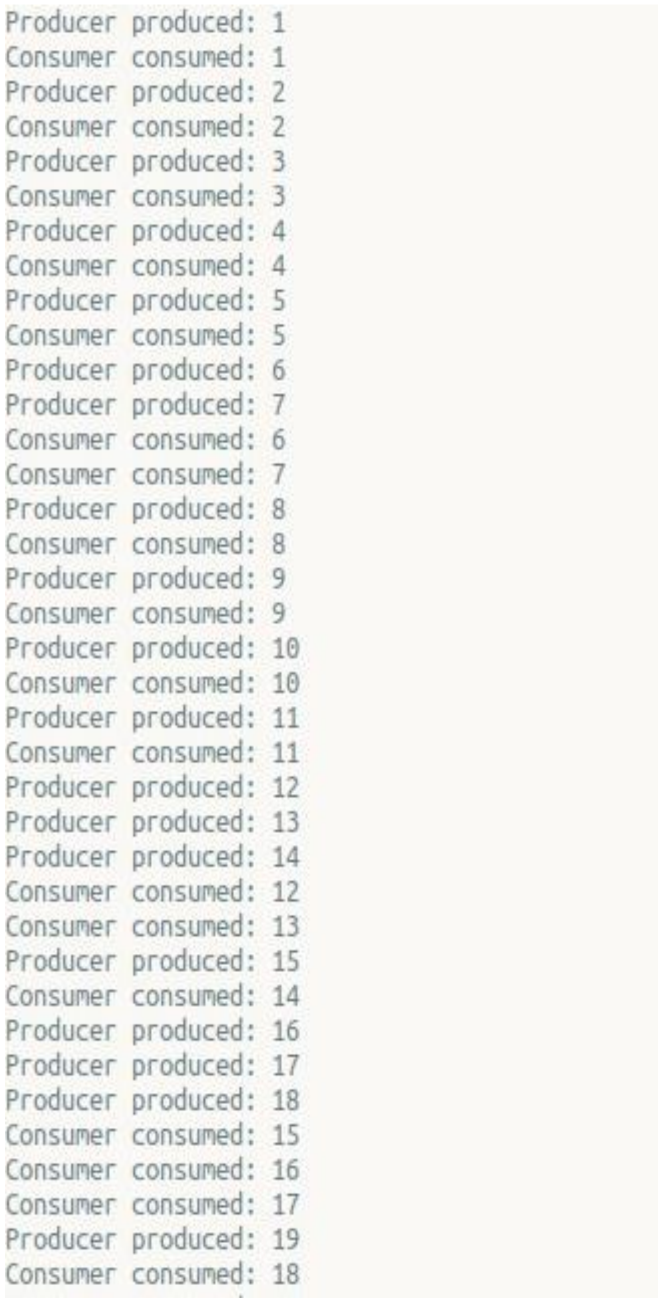
- Use a shared buffer with a maximum size of 10.
- Use semaphores and mutex to ensure thread-safe access to the buffer.
- Print the number that producer is producing and consumer is consuming.
- Both producer and consumer will continue for 20 iterations



```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define BUFFER_SIZE 10
#define PRODUCE_COUNT 20

int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;

sem_t empty;
sem_t full;
pthread_mutex_t mutex;

void* producer(void* arg) {
    for (int i = 0; i < PRODUCE_COUNT; i++) {
        int item = i + 1;

        sem_wait(&empty);

        pthread_mutex_lock(&mutex);

        buffer[in] = item;
        printf("Producer produced: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);

        sem_post(&full);

        usleep(rand() % 100000);
    }
    return NULL;
}

void* consumer(void* arg) {
    for (int i = 0; i < PRODUCE_COUNT; i++) {
        int item;

        sem_wait(&full);

        pthread_mutex_lock(&mutex);

        item = buffer[out];
        printf("Consumer consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);

        sem_post(&empty);

        usleep(rand() % 100000);
```



```c
}

int main() {
    pthread_t prod_thread, cons_thread;

    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&prod_thread, NULL, producer, NULL);
    pthread_create(&cons_thread, NULL, consumer, NULL);

    pthread_join(prod_thread, NULL);
    pthread_join(cons_thread, NULL);

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);


}
```



```
Producer produced: 1
Consumer consumed: 1
Producer produced: 2
Consumer consumed: 2
Producer produced: 3
Consumer consumed: 3
Producer produced: 4
Consumer consumed: 4
Producer produced: 5
Consumer consumed: 5
Producer produced: 6
Producer produced: 7
Consumer consumed: 6
Consumer consumed: 7
Producer produced: 8
Consumer consumed: 8
Producer produced: 9
Consumer consumed: 9
Producer produced: 10
Consumer consumed: 10
Producer produced: 11
Consumer consumed: 11
Producer produced: 12
Producer produced: 13
Producer produced: 14
Consumer consumed: 12
Consumer consumed: 13
Producer produced: 15
Consumer consumed: 14
Producer produced: 16
Producer produced: 17
Producer produced: 18
Consumer consumed: 15
Consumer consumed: 16
Consumer consumed: 17
Producer produced: 19
Consumer consumed: 18
```

2. Alternating Numbers with Two Threads

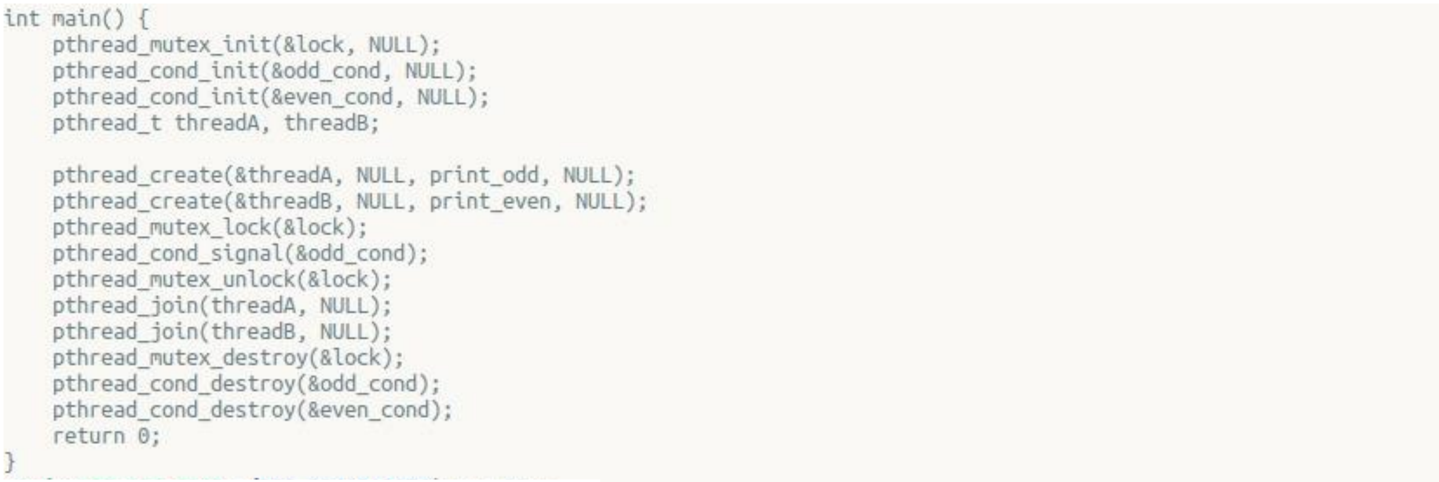
Problem: Write a program to print 1, 2, 3 … upto 20. Create threads where two threads print numbers alternately.

- **Thread A** prints odd numbers: 1, 3, 5 ...
- **Thread B** prints even numbers: 2, 4, 6 ...

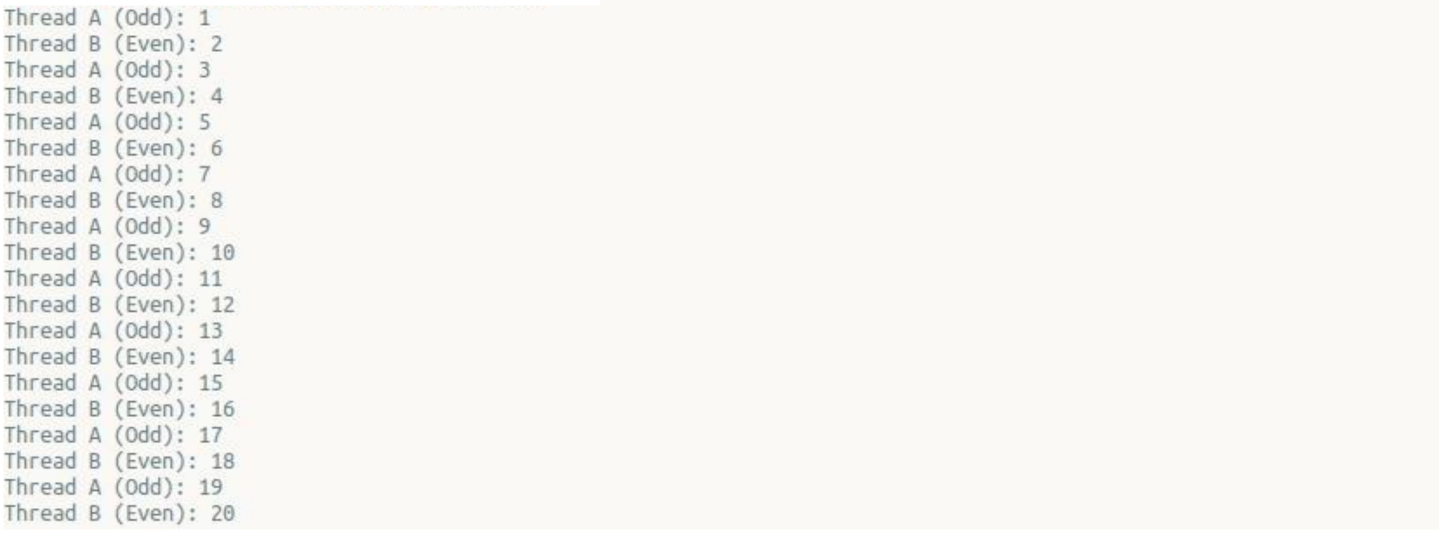
Requirements:

- Use semaphores to control the order of execution of the threads.
- Ensure no race conditions occur.

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define MAX_NUMBER 20
pthread_mutex_t lock;
pthread_cond_t odd_cond, even_cond;
int turn = 1;
void* print_odd(void* arg) {
    for (int i = 1; i <= MAX_NUMBER; i += 2) {
        pthread_mutex_lock(&lock);
        while (turn != 1) {
            pthread_cond_wait(&odd_cond, &lock);
        }
        printf("Thread A (Odd): %d\n", i);
        turn = 2;
        pthread_cond_signal(&even_cond);
        pthread_mutex_unlock(&lock);
        sleep(1);
    }
    return NULL;
}
void* print_even(void* arg) {
    for (int i = 2; i <= MAX_NUMBER; i += 2) {
        pthread_mutex_lock(&lock);
        while (turn != 2) {
            pthread_cond_wait(&even_cond, &lock);
        }
        printf("Thread B (Even): %d\n", i);
        turn = 1;
        pthread_cond_signal(&odd_cond);
        pthread_mutex_unlock(&lock);
```



```
int main() {
    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&odd_cond, NULL);
    pthread_cond_init(&even_cond, NULL);
    pthread_t threadA, threadB;

    pthread_create(&threadA, NULL, print_odd, NULL);
    pthread_create(&threadB, NULL, print_even, NULL);
    pthread_mutex_lock(&lock);
    pthread_cond_signal(&odd_cond);
    pthread_mutex_unlock(&lock);
    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&odd_cond);
    pthread_cond_destroy(&even_cond);
    return 0;
}
```



```
Thread A (Odd): 1
Thread B (Even): 2
Thread A (Odd): 3
Thread B (Even): 4
Thread A (Odd): 5
Thread B (Even): 6
Thread A (Odd): 7
Thread B (Even): 8
Thread A (Odd): 9
Thread B (Even): 10
Thread A (Odd): 11
Thread B (Even): 12
Thread A (Odd): 13
Thread B (Even): 14
Thread A (Odd): 15
Thread B (Even): 16
Thread A (Odd): 17
Thread B (Even): 18
Thread A (Odd): 19
Thread B (Even): 20
```

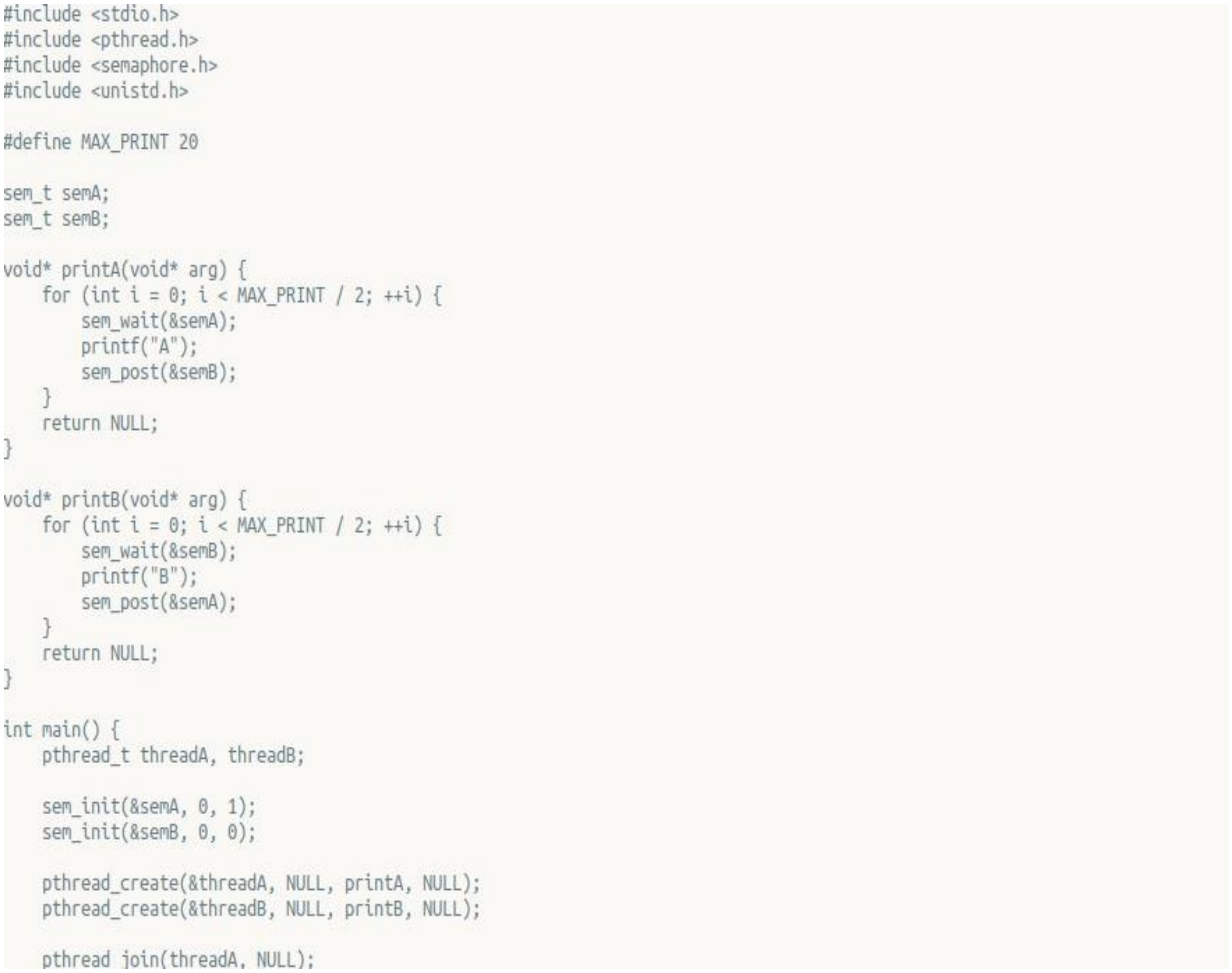
3. Alternating Characters

Problem: Write a program to create two threads that print characters (A and B) alternately such as ABABABABA…. upto 20. Use semaphores to synchronize the threads.

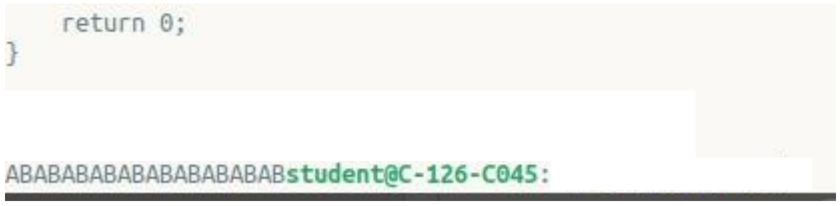
- **Thread A** prints A.
- **Thread B** prints B.

Requirements:

- Use semaphores to control the order of execution of the threads.
- Ensure no race conditions occur.



```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define MAX_PRINT 20

sem_t semA;
sem_t semB;

void* printA(void* arg) {
    for (int i = 0; i < MAX_PRINT / 2; ++i) {
        sem_wait(&semA);
        printf("A");
        sem_post(&semB);
    }
    return NULL;
}

void* printB(void* arg) {
    for (int i = 0; i < MAX_PRINT / 2; ++i) {
        sem_wait(&semB);
        printf("B");
        sem_post(&semA);
    }
    return NULL;
}

int main() {
    pthread_t threadA, threadB;

    sem_init(&semA, 0, 1);
    sem_init(&semB, 0, 0);

    pthread_create(&threadA, NULL, printA, NULL);
    pthread_create(&threadB, NULL, printB, NULL);

    pthread_join(threadA, NULL);
```



```
    return 0;
}
```

```
ABABABABABABABABABABstudent@C-126-C045:
```

4.Countdown and Countup

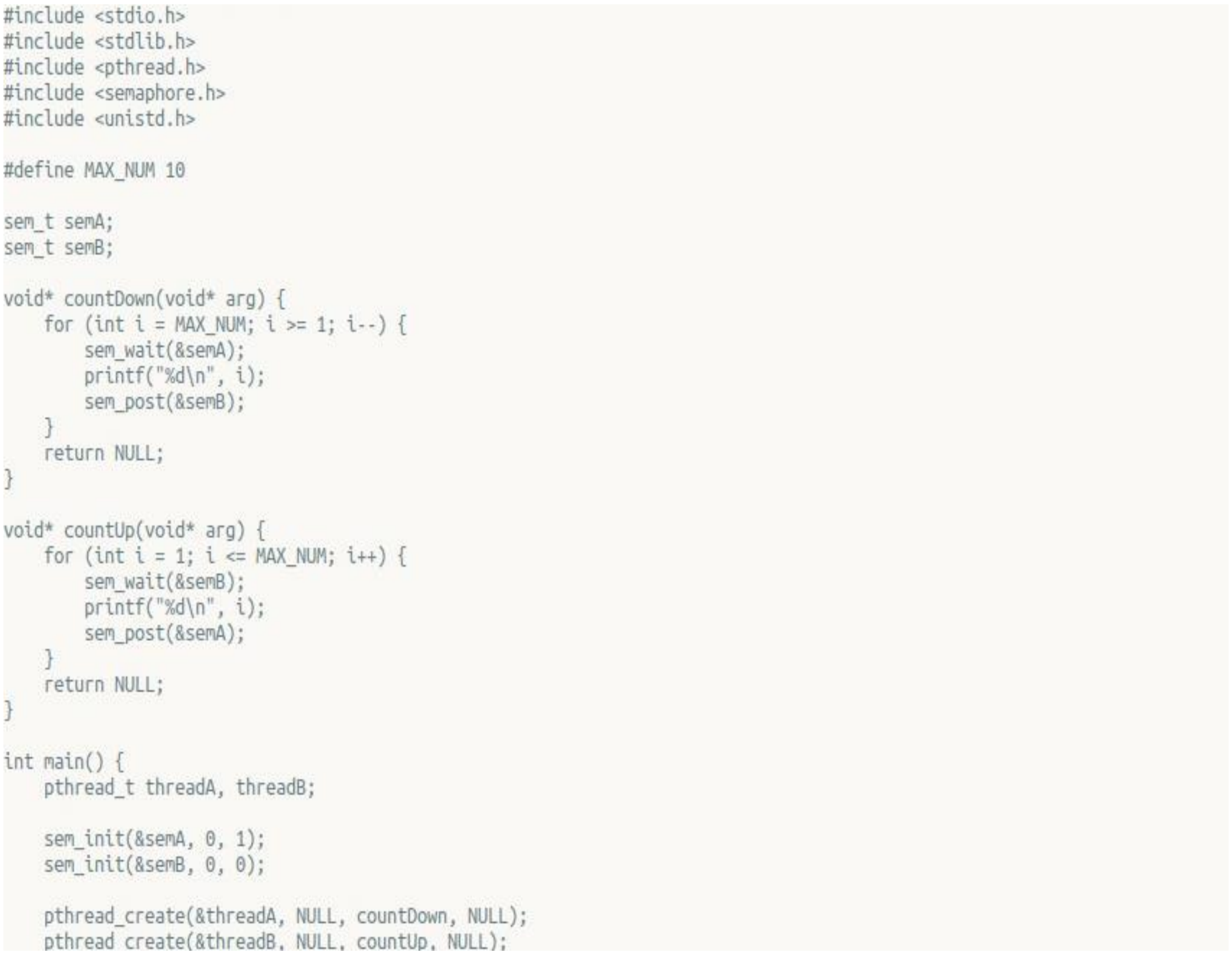
Problem: Write a program create two threads where:

- **Thread A** counts down from 10 to 1.
- **Thread B** counts up from 1 to 10.

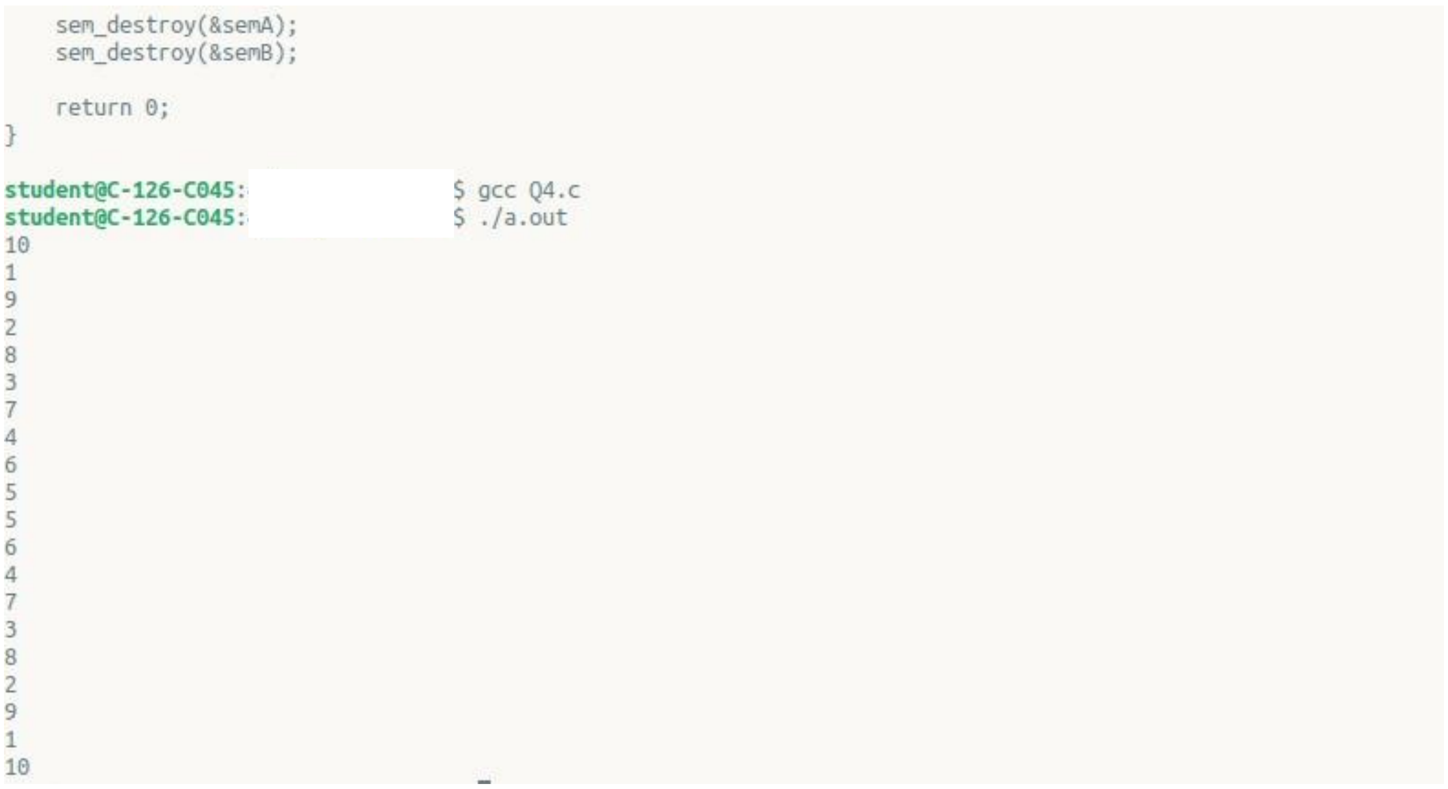
Both threads should alternate execution.

Requirements:

- Use semaphores to control the order of execution of the threads.
- Ensure no race conditions occur.



```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define MAX_NUM 10

sem_t semA;
sem_t semB;

void* countDown(void* arg) {
    for (int i = MAX_NUM; i >= 1; i--) {
        sem_wait(&semA);
        printf("%d\n", i);
        sem_post(&semB);
    }
    return NULL;
}

void* countUp(void* arg) {
    for (int i = 1; i <= MAX_NUM; i++) {
        sem_wait(&semB);
        printf("%d\n", i);
        sem_post(&semA);
    }
    return NULL;
}

int main() {
    pthread_t threadA, threadB;

    sem_init(&semA, 0, 1);
    sem_init(&semB, 0, 0);

    pthread_create(&threadA, NULL, countDown, NULL);
    pthread_create(&threadB, NULL, countUp, NULL);
```



```c
    sem_destroy(&semA);
    sem_destroy(&semB);

    return 0;
}
```

```
student@C-126-C045:                 $ gcc Q4.c
student@C-126-C045:                 $ ./a.out
10
1
9
2
8
3
7
4
6
5
5
6
4
7
3
8
2
9
1
10
```

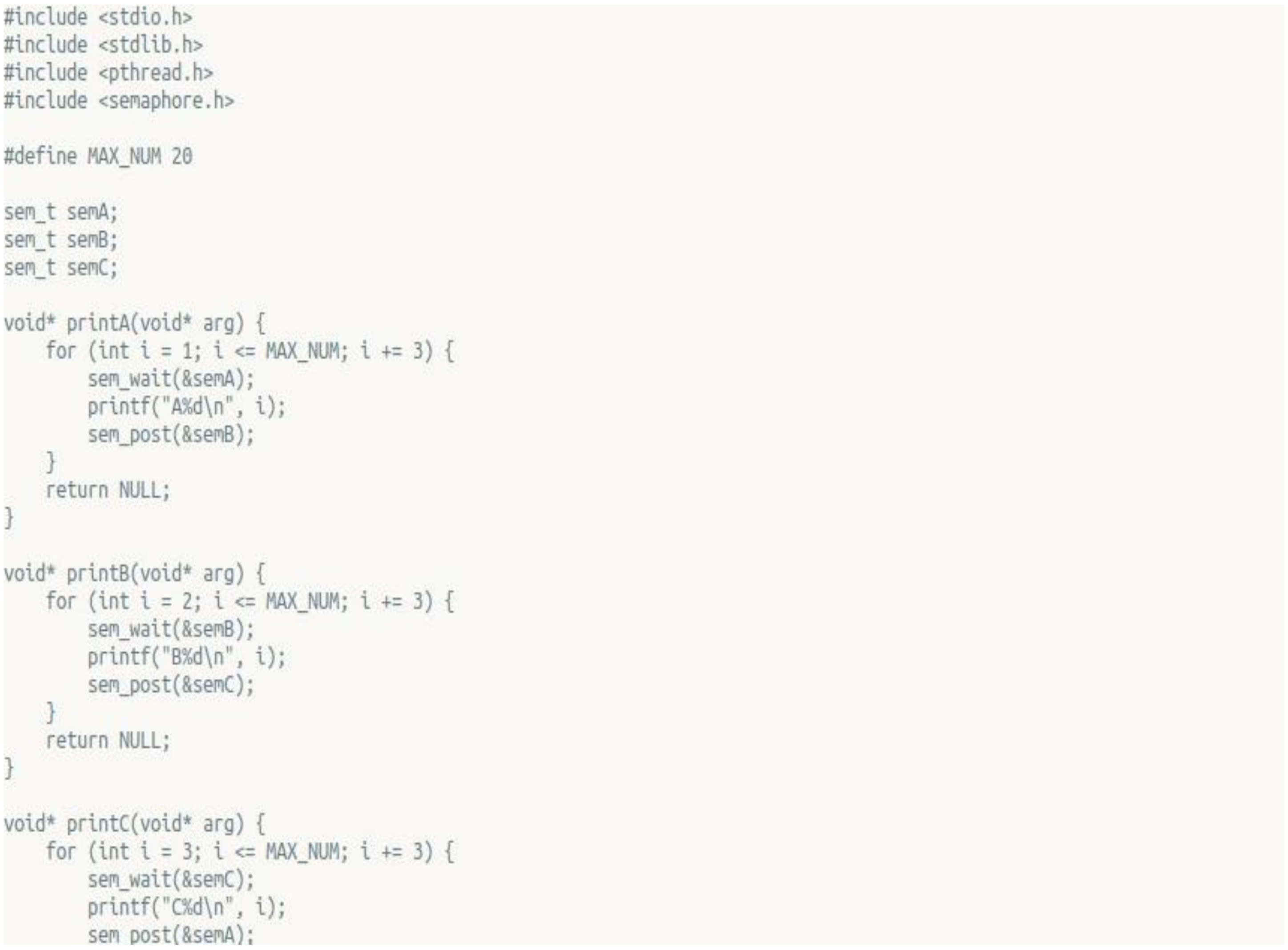
5.Sequence Printing using Threads

Problem: Write a program that creates three threads: Thread A, Thread B, and Thread C. The threads must print numbers in the following sequence: A1, B2, C3, A4, B5, C6 … upto 20 numbers.

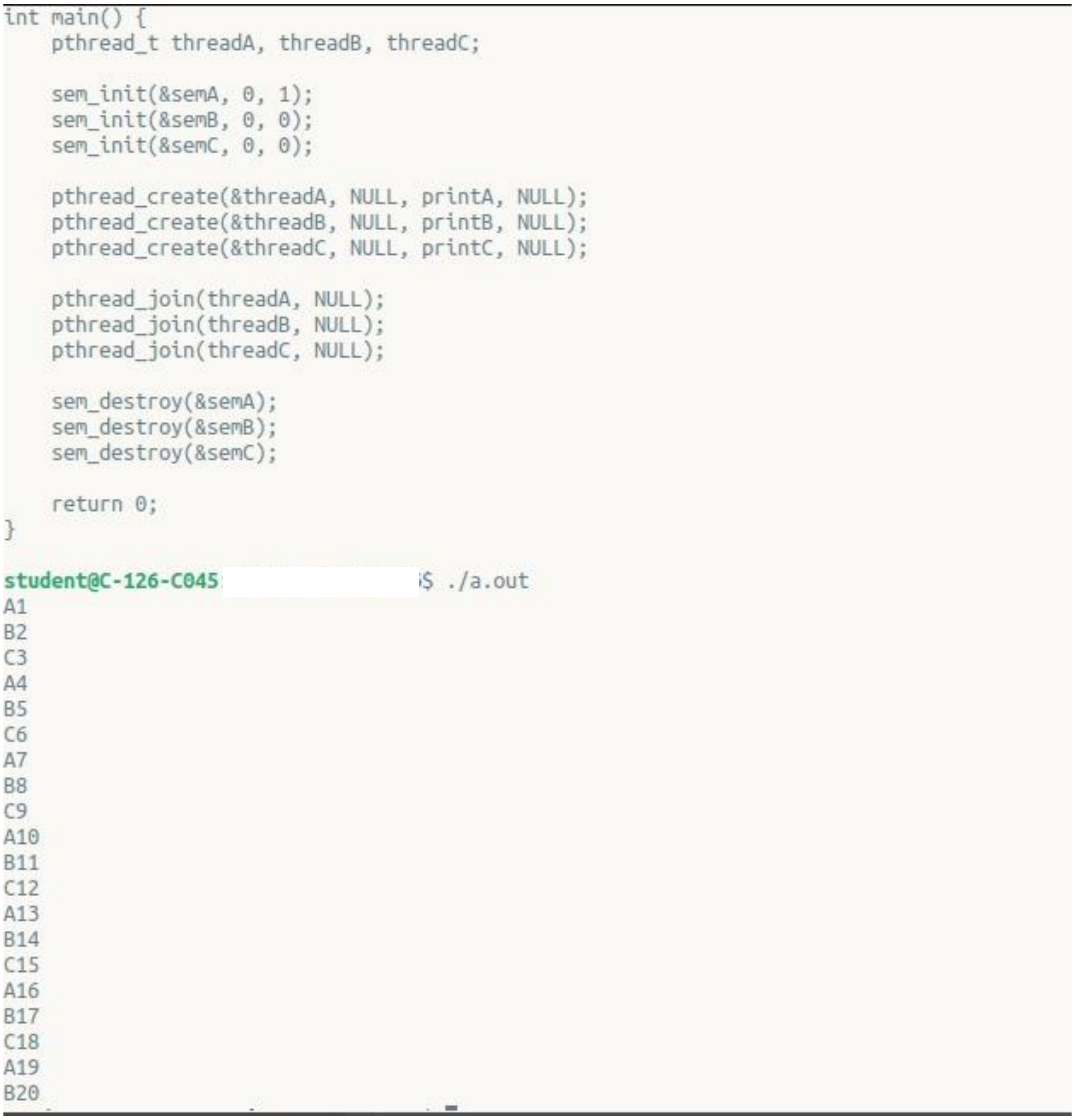
- **Thread A** prints A1, A4, A7, …
- **Thread B** prints B2, B5, B8, …
- **Thread C** prints C3, C6, C9, ...

Requirements:

- Use semaphores to control the order of execution of the threads.
- Ensure no race conditions occur.



```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

#define MAX_NUM 20

sem_t semA;
sem_t semB;
sem_t semC;

void* printA(void* arg) {
    for (int i = 1; i <= MAX_NUM; i += 3) {
        sem_wait(&semA);
        printf("A%d\n", i);
        sem_post(&semB);
    }
    return NULL;
}

void* printB(void* arg) {
    for (int i = 2; i <= MAX_NUM; i += 3) {
        sem_wait(&semB);
        printf("B%d\n", i);
        sem_post(&semC);
    }
    return NULL;
}

void* printC(void* arg) {
    for (int i = 3; i <= MAX_NUM; i += 3) {
        sem_wait(&semC);
        printf("C%d\n", i);
        sem_post(&semA);
```



```
int main() {
    pthread_t threadA, threadB, threadC;

    sem_init(&semA, 0, 1);
    sem_init(&semB, 0, 0);
    sem_init(&semC, 0, 0);

    pthread_create(&threadA, NULL, printA, NULL);
    pthread_create(&threadB, NULL, printB, NULL);
    pthread_create(&threadC, NULL, printC, NULL);

    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    pthread_join(threadC, NULL);

    sem_destroy(&semA);
    sem_destroy(&semB);
    sem_destroy(&semC);

    return 0;
}
```

```
student@C-126-C045[redacted]$ ./a.out
A1
B2
C3
A4
B5
C6
A7
B8
C9
A10
B11
C12
A13
B14
C15
A16
B17
C18
A19
B20
```