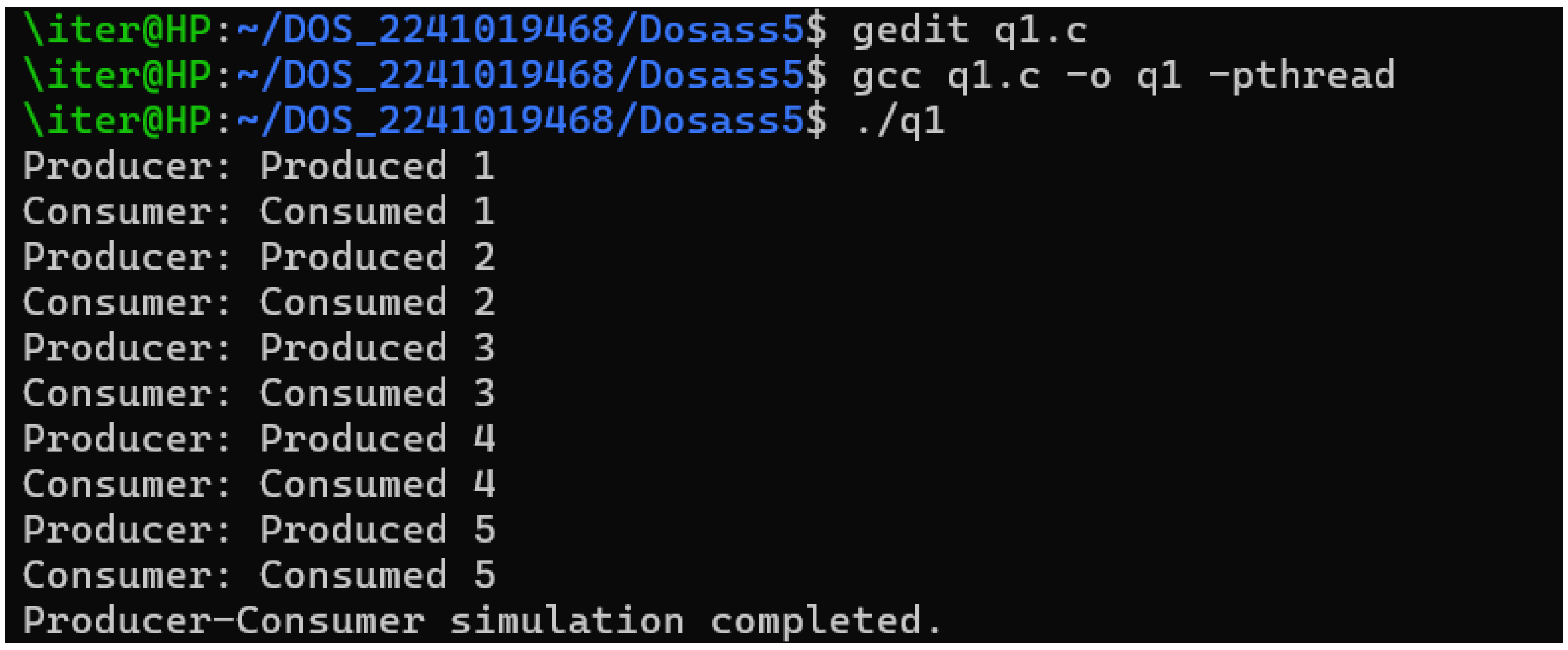
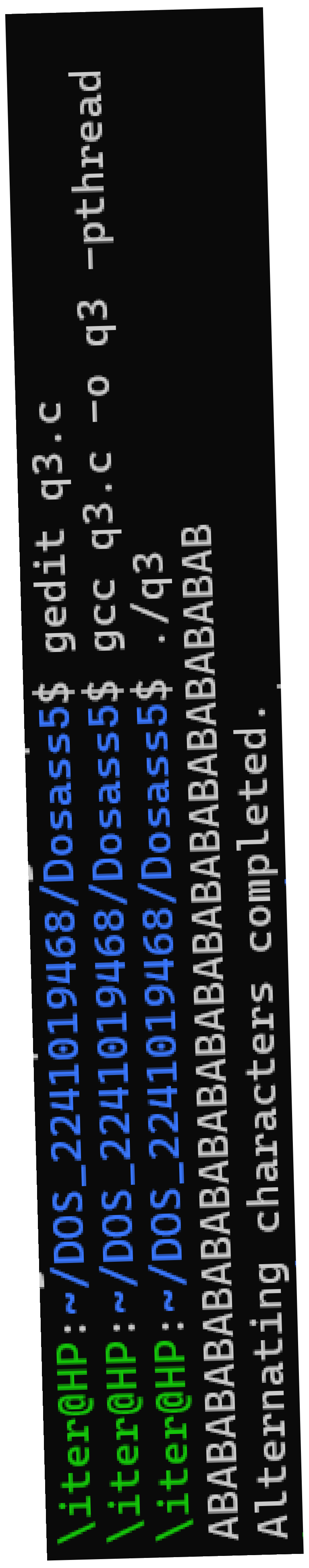
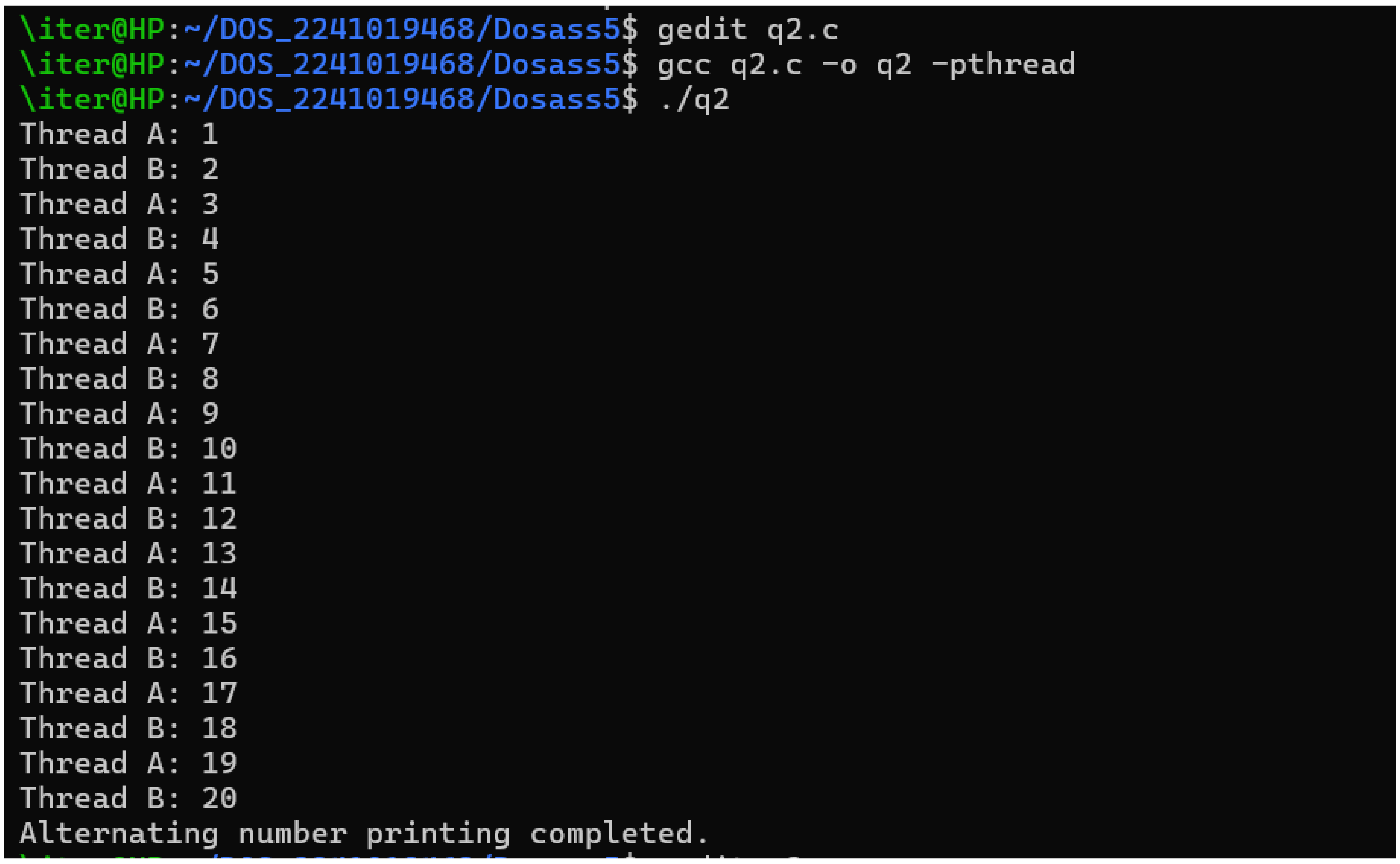


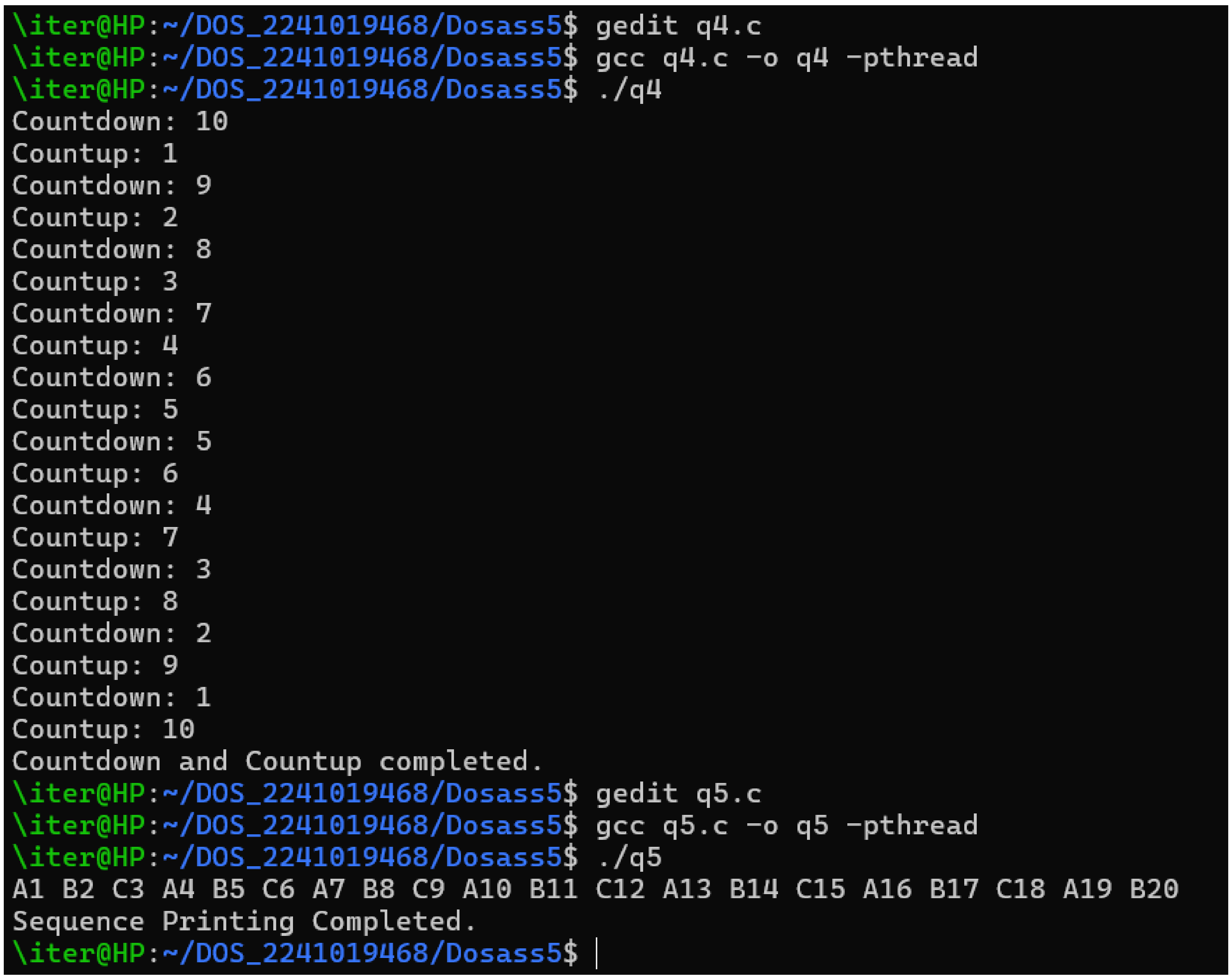
```
\iter@HP:~/DOS_2241019468/Dosass5$ gedit q1.c
\iter@HP:~/DOS_2241019468/Dosass5$ gcc q1.c -o q1 -pthread
\iter@HP:~/DOS_2241019468/Dosass5$ ./q1
Producer: Produced 1
Consumer: Consumed 1
Producer: Produced 2
Consumer: Consumed 2
Producer: Produced 3
Consumer: Consumed 3
Producer: Produced 4
Consumer: Consumed 4
Producer: Produced 5
Consumer: Consumed 5
Producer-Consumer simulation completed.
```



```
\iter@HP:~/DOS_2241019468/Dosass5$ gedit q3.c
\iter@HP:~/DOS_2241019468/Dosass5$ gcc q3.c -o q3 -pthread
\iter@HP:~/DOS_2241019468/Dosass5$ ./q3
ABABABABABABABABABABABABABABABABABABABAB
Alternating characters completed.
```

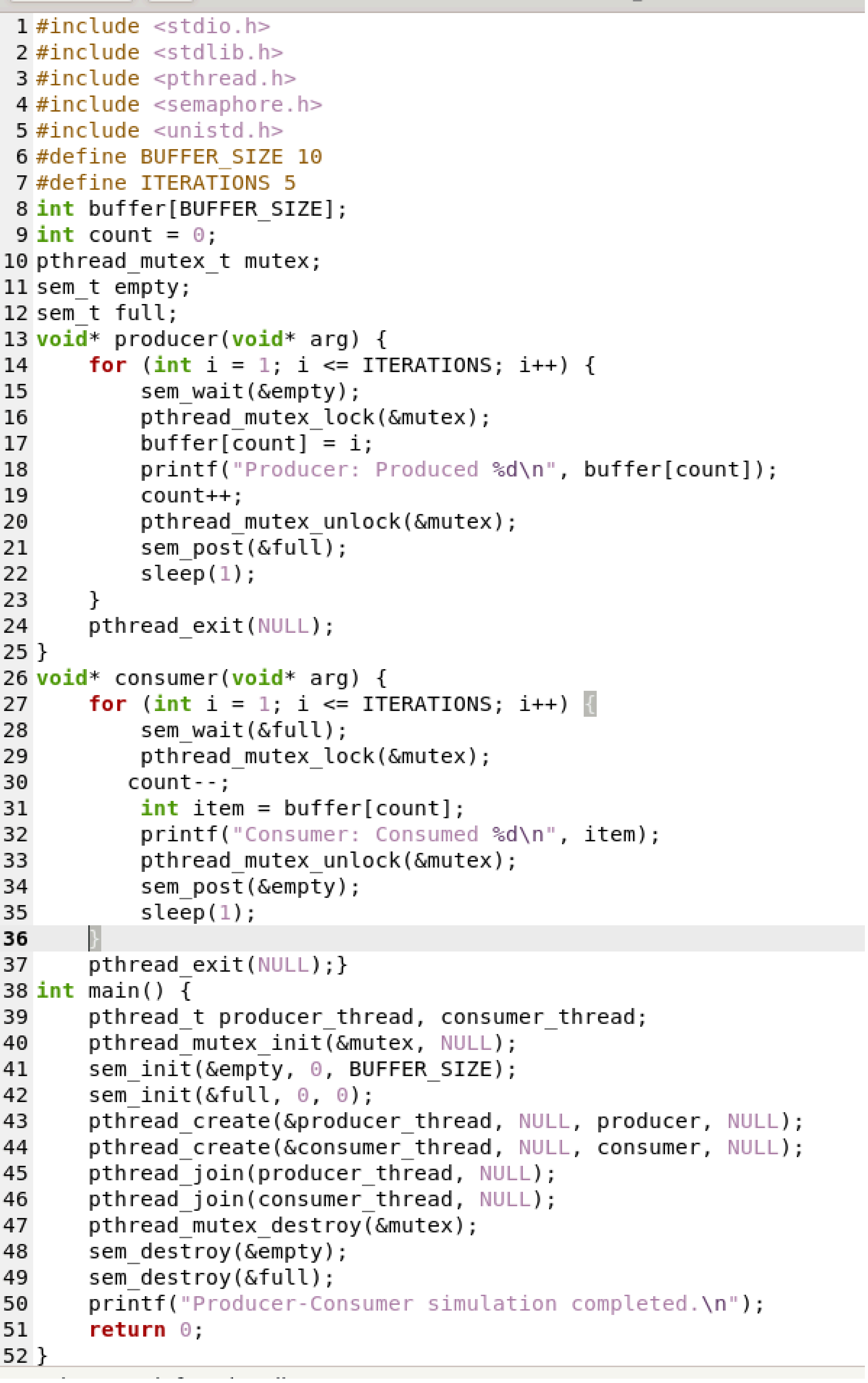


```
\iter@HP:~/DOS_2241019468/Dosass5$ gedit q2.c
\iter@HP:~/DOS_2241019468/Dosass5$ gcc q2.c -o q2 -pthread
\iter@HP:~/DOS_2241019468/Dosass5$ ./q2
Thread A: 1
Thread B: 2
Thread A: 3
Thread B: 4
Thread A: 5
Thread B: 6
Thread A: 7
Thread B: 8
Thread A: 9
Thread B: 10
Thread A: 11
Thread B: 12
Thread A: 13
Thread B: 14
Thread A: 15
Thread B: 16
Thread A: 17
Thread B: 18
Thread A: 19
Thread B: 20
Alternating number printing completed.
```



```
\iter@HP:~/DOS_2241019468/Dosass5$ gedit q4.c
\iter@HP:~/DOS_2241019468/Dosass5$ gcc q4.c -o q4 -pthread
\iter@HP:~/DOS_2241019468/Dosass5$ ./q4
Countdown: 10
Countup: 1
Countdown: 9
Countup: 2
Countdown: 8
Countup: 3
Countdown: 7
Countup: 4
Countdown: 6
Countup: 5
Countdown: 5
Countup: 6
Countdown: 4
Countup: 7
Countdown: 3
Countup: 8
Countdown: 2
Countup: 9
Countdown: 1
Countup: 10
Countdown and Countup completed.
\iter@HP:~/DOS_2241019468/Dosass5$ gedit q5.c
\iter@HP:~/DOS_2241019468/Dosass5$ gcc q5.c -o q5 -pthread
\iter@HP:~/DOS_2241019468/Dosass5$ ./q5
A1 B2 C3 A4 B5 C6 A7 B8 C9 A10 B11 C12 A13 B14 C15 A16 B17 C18 A19 B20
Sequence Printing Completed.
\iter@HP:~/DOS_2241019468/Dosass5$ 
```

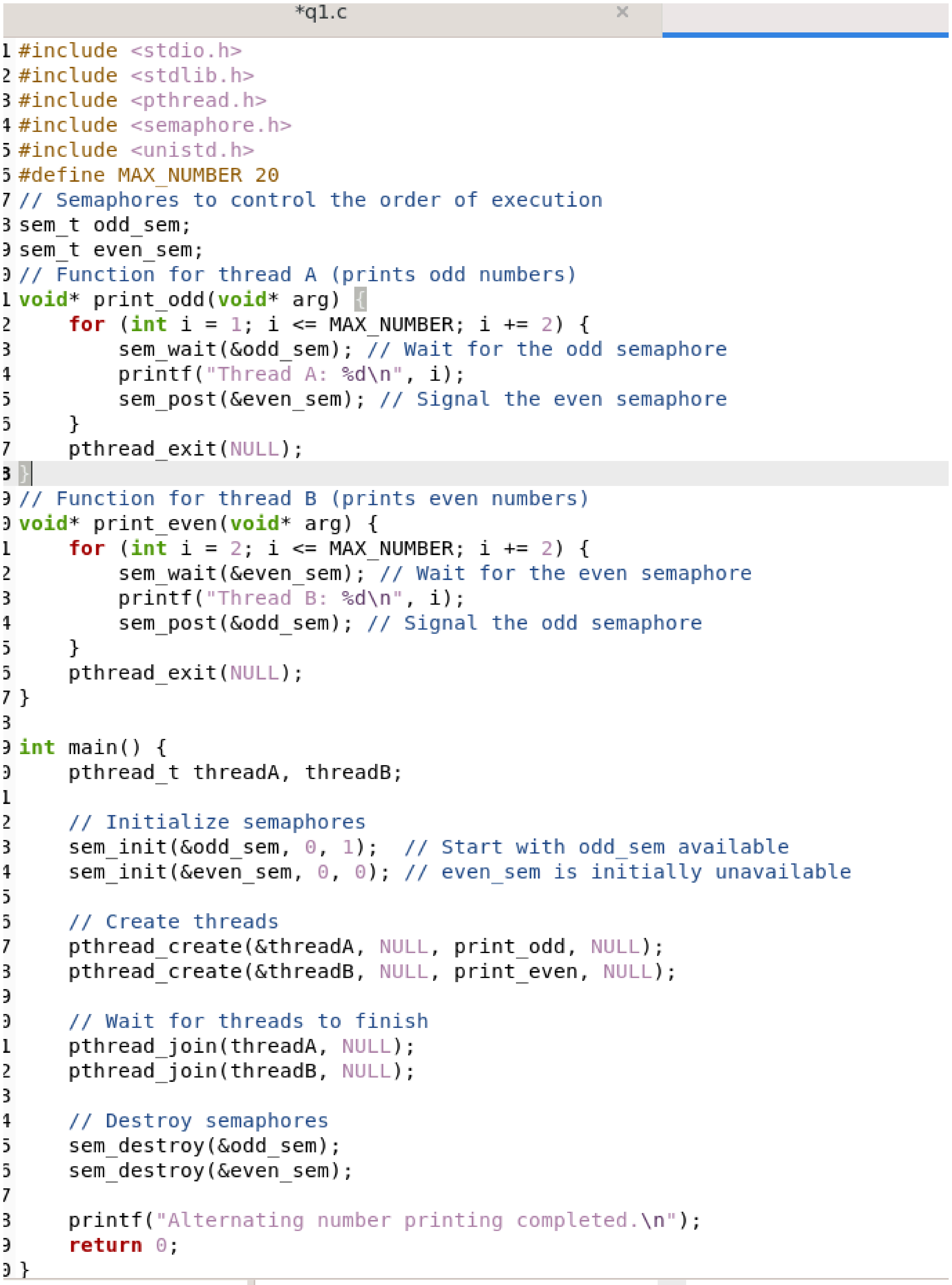
4 5



```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 10
#define ITERATIONS 5
int buffer[BUFFER_SIZE];
int count = 0;
pthread_mutex_t mutex;
sem_t empty;
sem_t full;
void* producer(void* arg) {
    for (int i = 1; i <= ITERATIONS; i++) {
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[count] = i;
        printf("Producer: Produced %d\n", buffer[count]);
        count++;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        sleep(1);
    }
    pthread_exit(NULL);
}
void* consumer(void* arg) {
    for (int i = 1; i <= ITERATIONS; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
       count--;
        int item = buffer[count];
        printf("Consumer: Consumed %d\n", item);
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        sleep(1);
    }
    pthread_exit(NULL);}
int main() {
    pthread_t producer_thread, consumer_thread;
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);
    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);
    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);
    printf("Producer-Consumer simulation completed.\n");
    return 0;
}
```

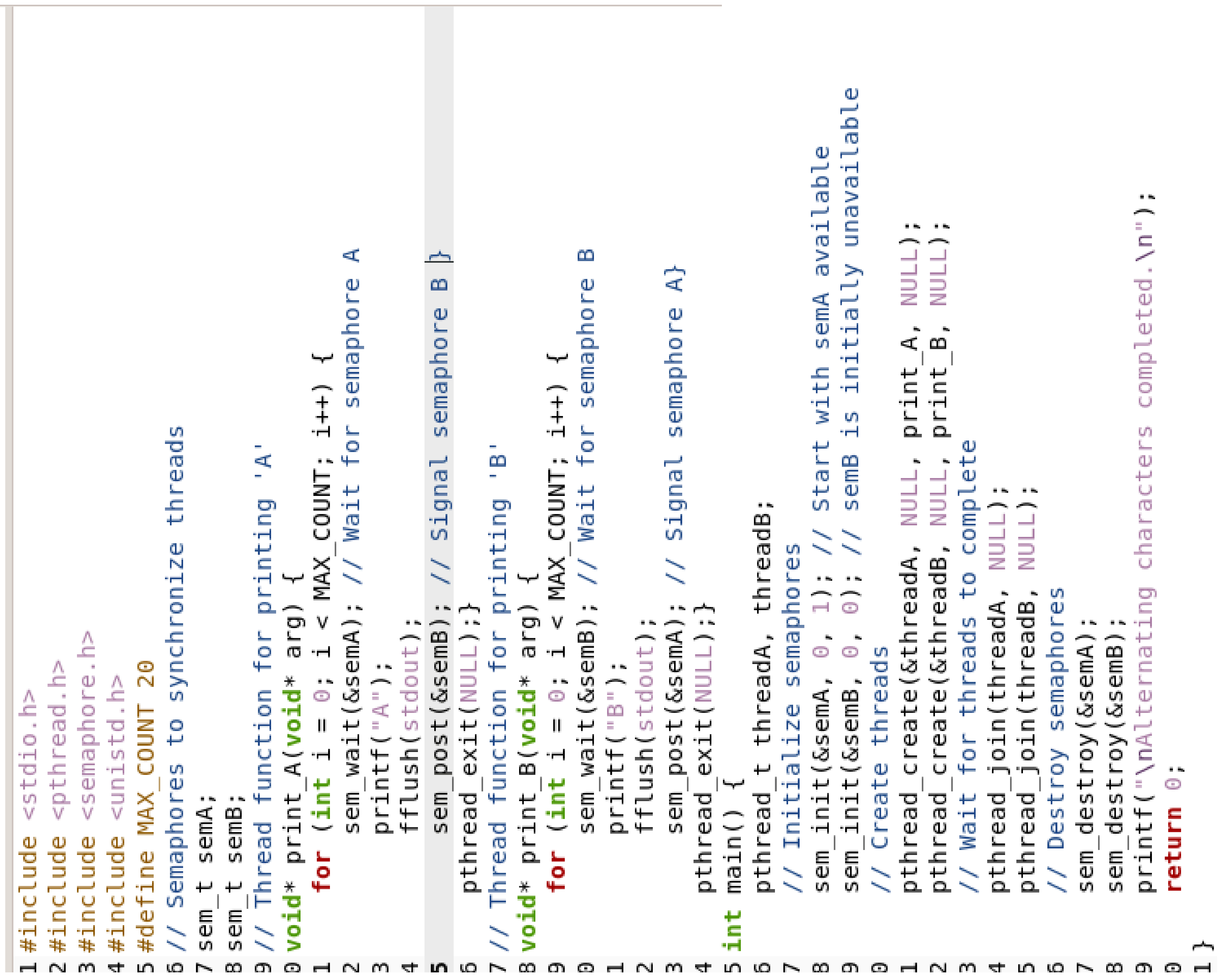
1

2



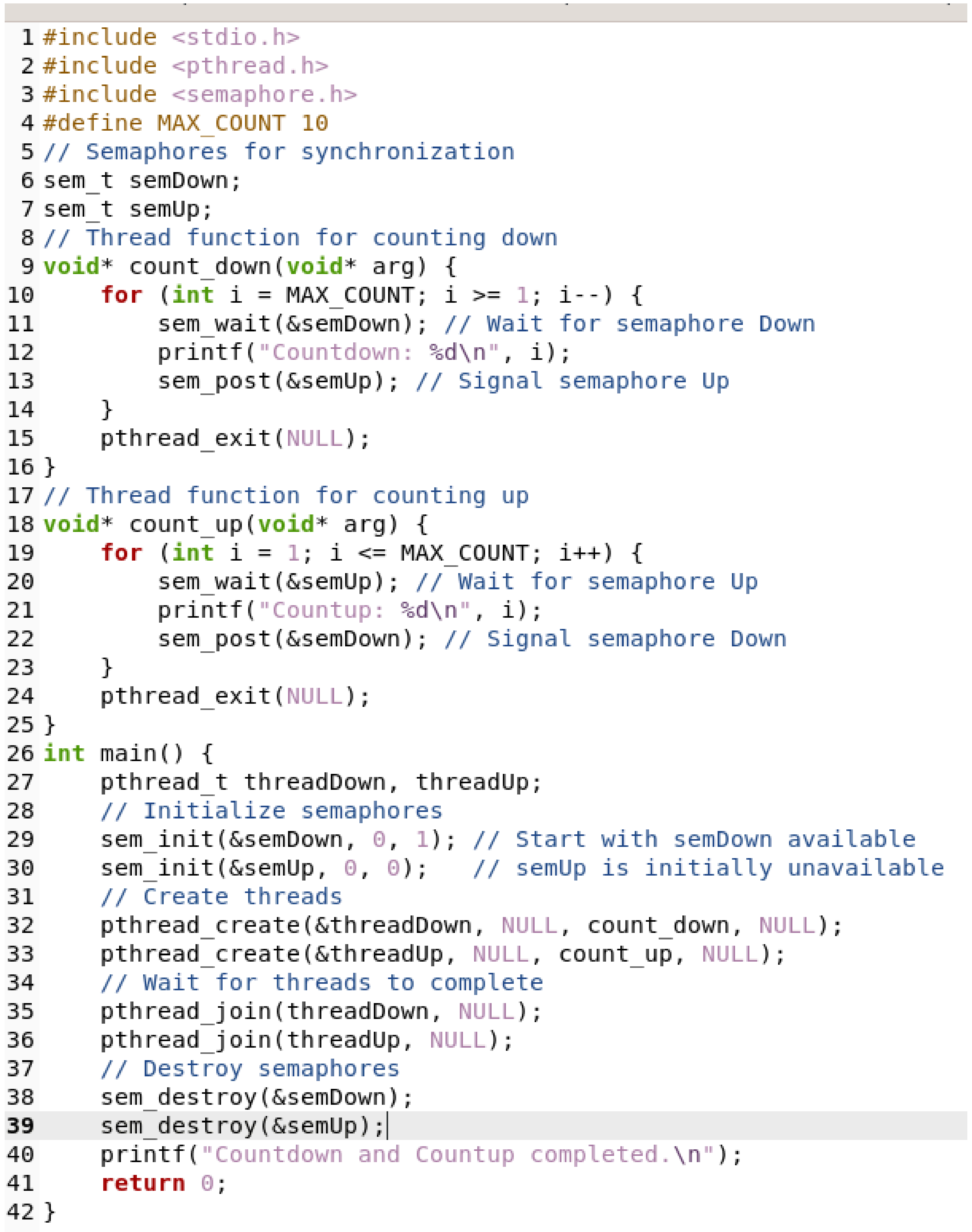
```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define MAX_NUMBER 20
// Semaphores to control the order of execution
sem_t odd_sem;
sem_t even_sem;
// Function for thread A (prints odd numbers)
void* print_odd(void* arg) {
    for (int i = 1; i <= MAX_NUMBER; i += 2) {
        sem_wait(&odd_sem); // Wait for the odd semaphore
        printf("Thread A: %d\n", i);
        sem_post(&even_sem); // Signal the even semaphore
    }
    pthread_exit(NULL);
}
// Function for thread B (prints even numbers)
void* print_even(void* arg) {
    for (int i = 2; i <= MAX_NUMBER; i += 2) {
        sem_wait(&even_sem); // Wait for the even semaphore
        printf("Thread B: %d\n", i);
        sem_post(&odd_sem); // Signal the odd semaphore
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t threadA, threadB;

    // Initialize semaphores
    sem_init(&odd_sem, 0, 1);  // Start with odd_sem available
    sem_init(&even_sem, 0, 0); // even_sem is initially unavailable

    // Create threads
    pthread_create(&threadA, NULL, print_odd, NULL);
    pthread_create(&threadB, NULL, print_even, NULL);

    // Wait for threads to finish
    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);

    // Destroy semaphores
    sem_destroy(&odd_sem);
    sem_destroy(&even_sem);

    printf("Alternating number printing completed.\n");
    return 0;
}
```

3



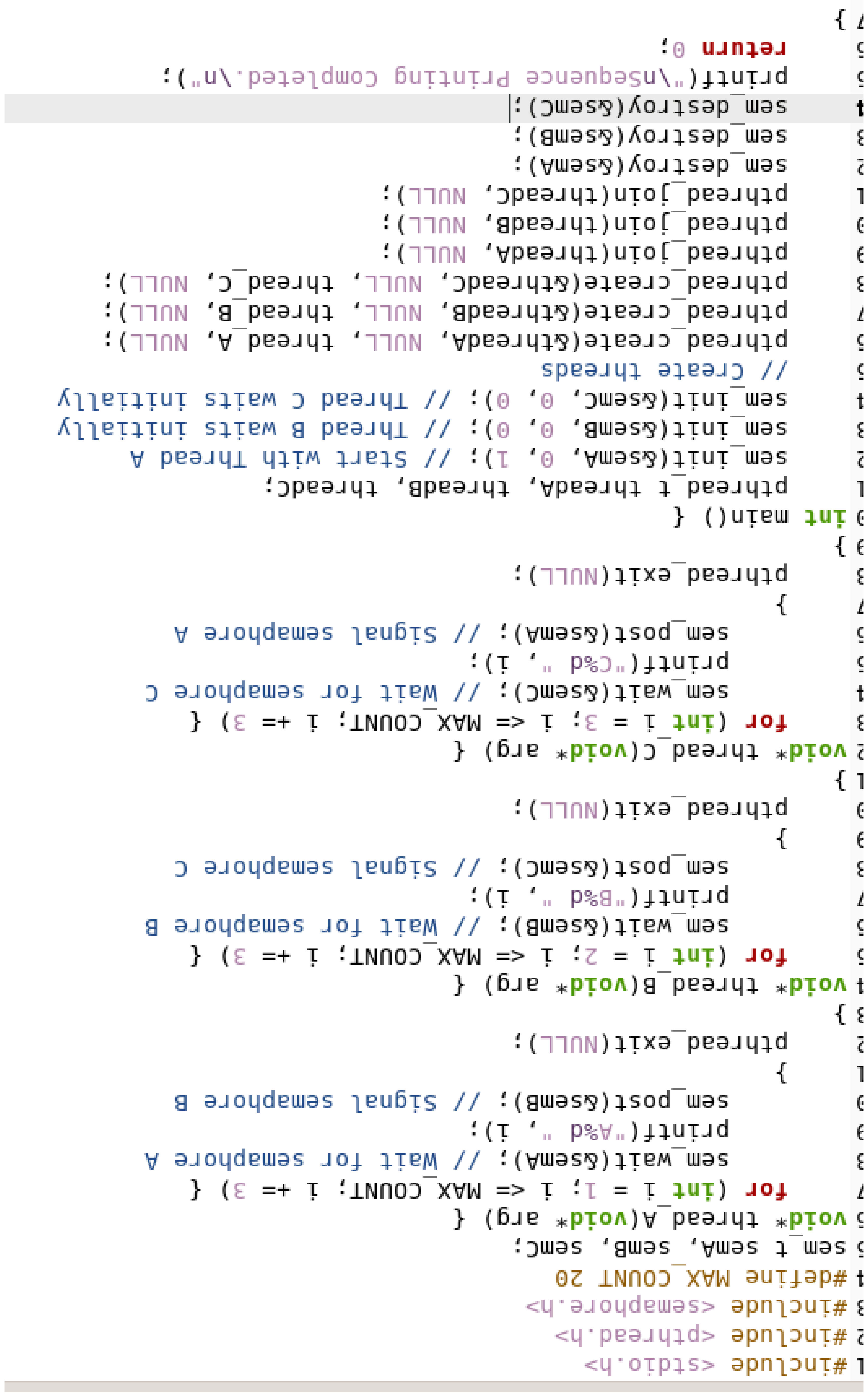
```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define MAX_COUNT 20
// Semaphores to synchronize threads
sem_t semA;
sem_t semB;
// Thread function for printing 'A'
void* print_A(void* arg) {
    for (int i = 0; i < MAX_COUNT; i++) {
        sem_wait(&semA); // Wait for semaphore A
        printf("A");
        fflush(stdout);
        sem_post(&semB); // Signal semaphore B }
    pthread_exit(NULL);}
// Thread function for printing 'B'
void* print_B(void* arg) {
    for (int i = 0; i < MAX_COUNT; i++) {
        sem_wait(&semB); // Wait for semaphore B
        printf("B");
        fflush(stdout);
        sem_post(&semA); // Signal semaphore A}
    pthread_exit(NULL);}
int main() {
    pthread_t threadA, threadB;
    // Initialize semaphores
    sem_init(&semA, 0, 1); // Start with semA available
    sem_init(&semB, 0, 0); // semB is initially unavailable
    // Create threads
    pthread_create(&threadA, NULL, print_A, NULL);
    pthread_create(&threadB, NULL, print_B, NULL);
    // Wait for threads to complete
    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    // Destroy semaphores
    sem_destroy(&semA);
    sem_destroy(&semB);
    printf("\nAlternating characters completed.\n");
    return 0;
}
```

4



```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define MAX_COUNT 10
// Semaphores for synchronization
sem_t semDown;
sem_t semUp;
// Thread function for counting down
void* count_down(void* arg) {
    for (int i = MAX_COUNT; i >= 1; i--) {
        sem_wait(&semDown); // Wait for semaphore Down
        printf("Countdown: %d\n", i);
        sem_post(&semUp); // Signal semaphore Up
    }
    pthread_exit(NULL);
}
// Thread function for counting up
void* count_up(void* arg) {
    for (int i = 1; i <= MAX_COUNT; i++) {
        sem_wait(&semUp); // Wait for semaphore Up
        printf("Countup: %d\n", i);
        sem_post(&semDown); // Signal semaphore Down
    }
    pthread_exit(NULL);
}
int main() {
    pthread_t threadDown, threadUp;
    // Initialize semaphores
    sem_init(&semDown, 0, 1); // Start with semDown available
    sem_init(&semUp, 0, 0);   // semUp is initially unavailable
    // Create threads
    pthread_create(&threadDown, NULL, count_down, NULL);
    pthread_create(&threadUp, NULL, count_up, NULL);
    // Wait for threads to complete
    pthread_join(threadDown, NULL);
    pthread_join(threadUp, NULL);
    // Destroy semaphores
    sem_destroy(&semDown);
    sem_destroy(&semUp);
    printf("Countdown and Countup completed.\n");
    return 0;
}
```

5



```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define MAX_COUNT 20
sem_t semA, semB, semC;
void* thread_A(void* arg) {
    for (int i = 1; i <= MAX_COUNT; i += 3) {
        sem_wait(&semA); // Wait for semaphore A
        printf("A%d ", i);
        sem_post(&semB); // Signal semaphore B
    }
    pthread_exit(NULL);
}
void* thread_B(void* arg) {
    for (int i = 2; i <= MAX_COUNT; i += 3) {
        sem_wait(&semB); // Wait for semaphore B
        printf("B%d ", i);
        sem_post(&semC); // Signal semaphore C
    }
    pthread_exit(NULL);
}
void* thread_C(void* arg) {
    for (int i = 3; i <= MAX_COUNT; i += 3) {
        sem_wait(&semC); // Wait for semaphore C
        printf("C%d ", i);
        sem_post(&semA); // Signal semaphore A
    }
    pthread_exit(NULL);
}
int main() {
    pthread_t threadA, threadB, threadC;
    sem_init(&semA, 0, 1); // Start with Thread A
    sem_init(&semB, 0, 0); // Thread B waits initially
    sem_init(&semC, 0, 0); // Thread C waits initially
    // Create threads
    pthread_create(&threadA, NULL, thread_A, NULL);
    pthread_create(&threadB, NULL, thread_B, NULL);
    pthread_create(&threadC, NULL, thread_C, NULL);
    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    pthread_join(threadC, NULL);
    sem_destroy(&semA);
    sem_destroy(&semB);
    sem_destroy(&semC);
    printf("\nSequence Printing Completed.\n");
    return 0;
}
```

Theory Assignment 2

On

Design Principles of Operating System

CSE 3249)

Submitted by

Name	**: Arman Singh**
Reg. No.	**: 2241019468**
Semester	**: 5th**
Branch	**: CSE**
Section	**: 016**
Session	**: 2024-2025**
Admission Batch	**: 2022**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
FACULTY OF ENGINEERING & TECHNOLOGY (ITER)
SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY