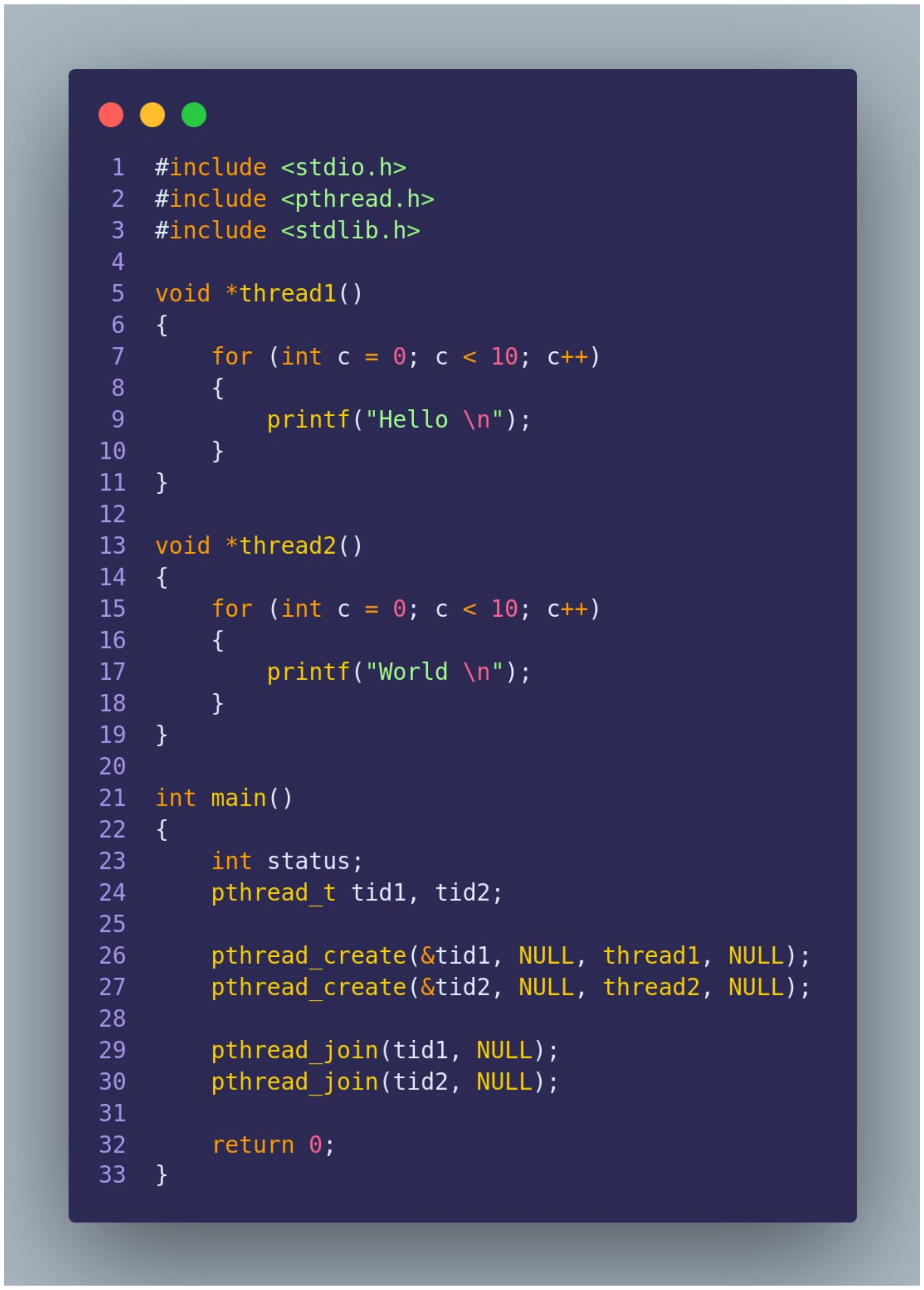
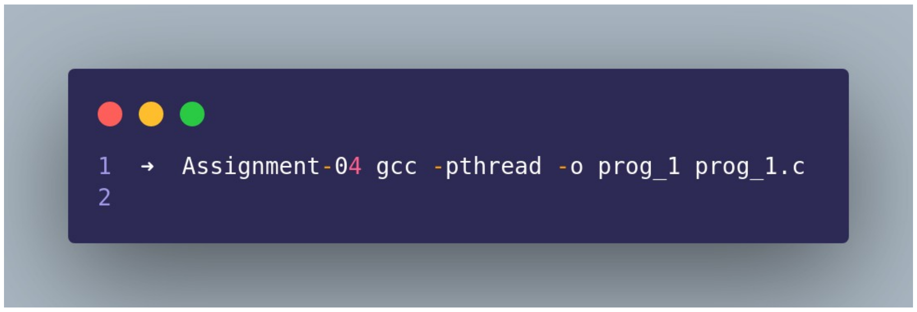
NAME : Jawad Ahmed
RollNO : 20P-0165
Section : BCS-4A

1:



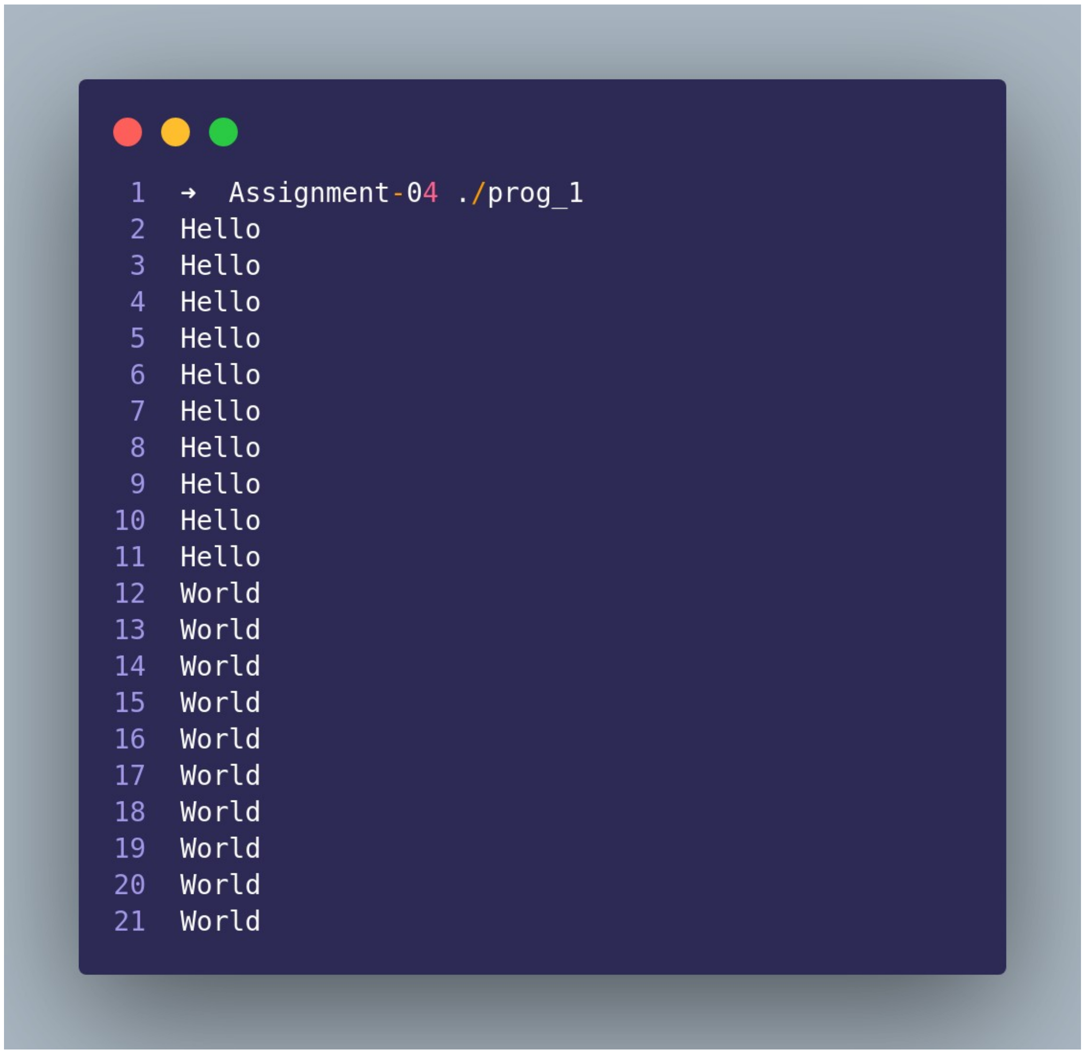
```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

void *thread1()
{
    for (int c = 0; c < 10; c++)
    {
        printf("Hello \n");
    }
}

void *thread2()
{
    for (int c = 0; c < 10; c++)
    {
        printf("World \n");
    }
}

int main()
{
    int status;
    pthread_t tid1, tid2;

    pthread_create(&tid1, NULL, thread1, NULL);
    pthread_create(&tid2, NULL, thread2, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    return 0;
}
```

2:



```
→ Assignment-04 gcc -pthread -o prog_1 prog_1.c

```

3: OUTPUT

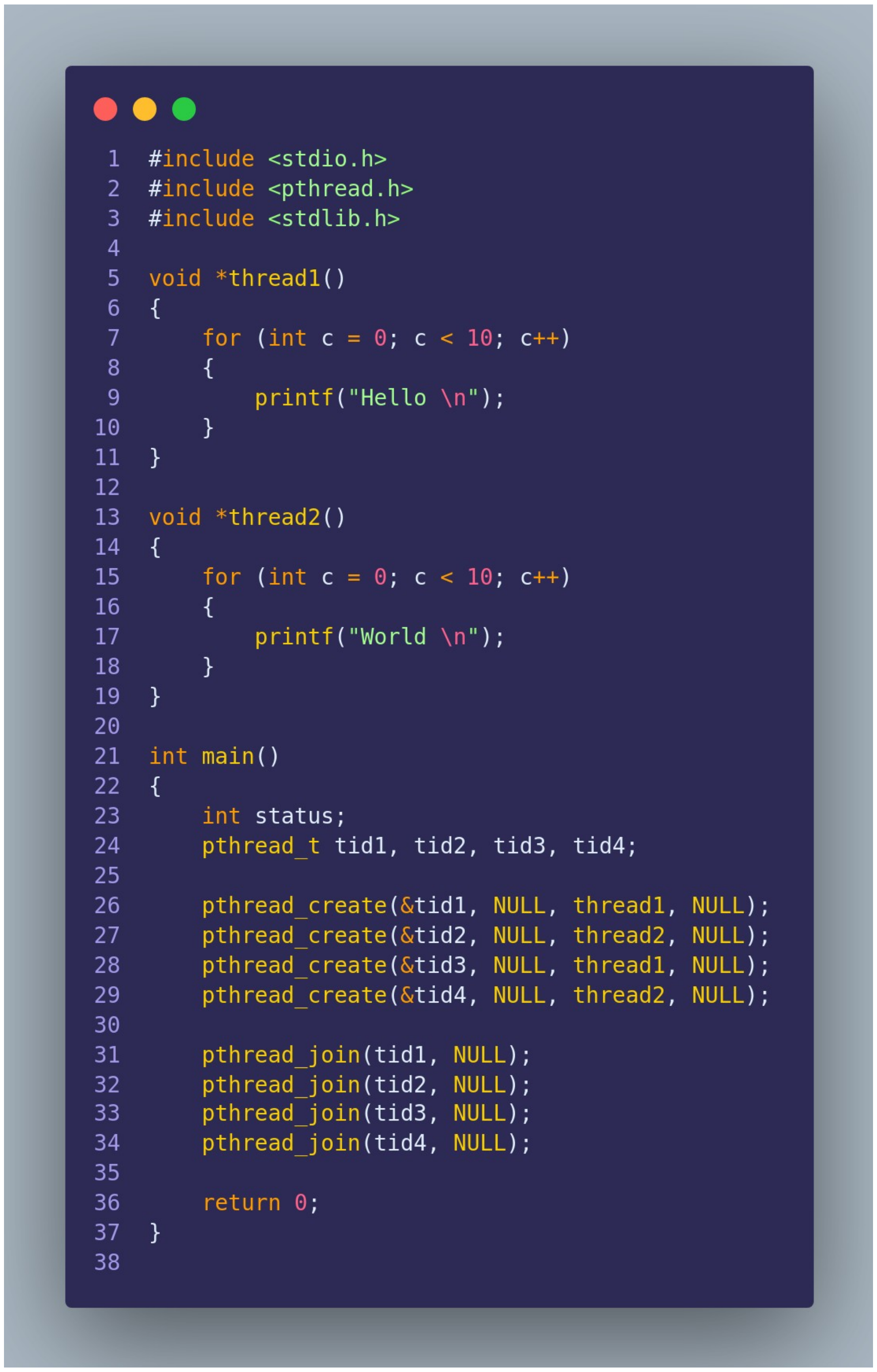


```
➜  Assignment-04 ./prog_1
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
World
World
World
World
World
World
World
World
World
World
```

4: Explanation of the OUTPUT:

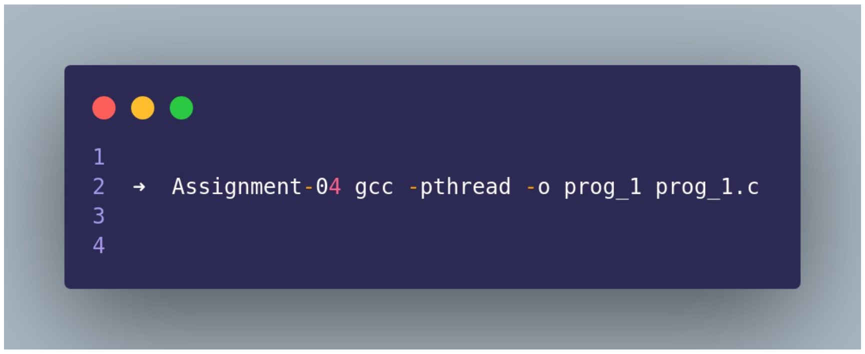
In this program what we have did is that we have created the two threads that will run two functions one function name is **thread1** and the other function name is **thread2** .T*he first thread will be going to run the thread1 function now it can happen that during the running of the thread1 the preemption can come and the control pass to second thread but in my case the first thread executed completely and after that the second thread runs and it completed it's execution and program exits. We have used the pthread_join that mean that don't exit the program until the given thread complete it's execution. If you don't add that the main exits and we all know that when main exits our program also exits. Also in the parameters when calling the function we passed* ***NULL*** *that mean that we are not passing that argument.*

5:



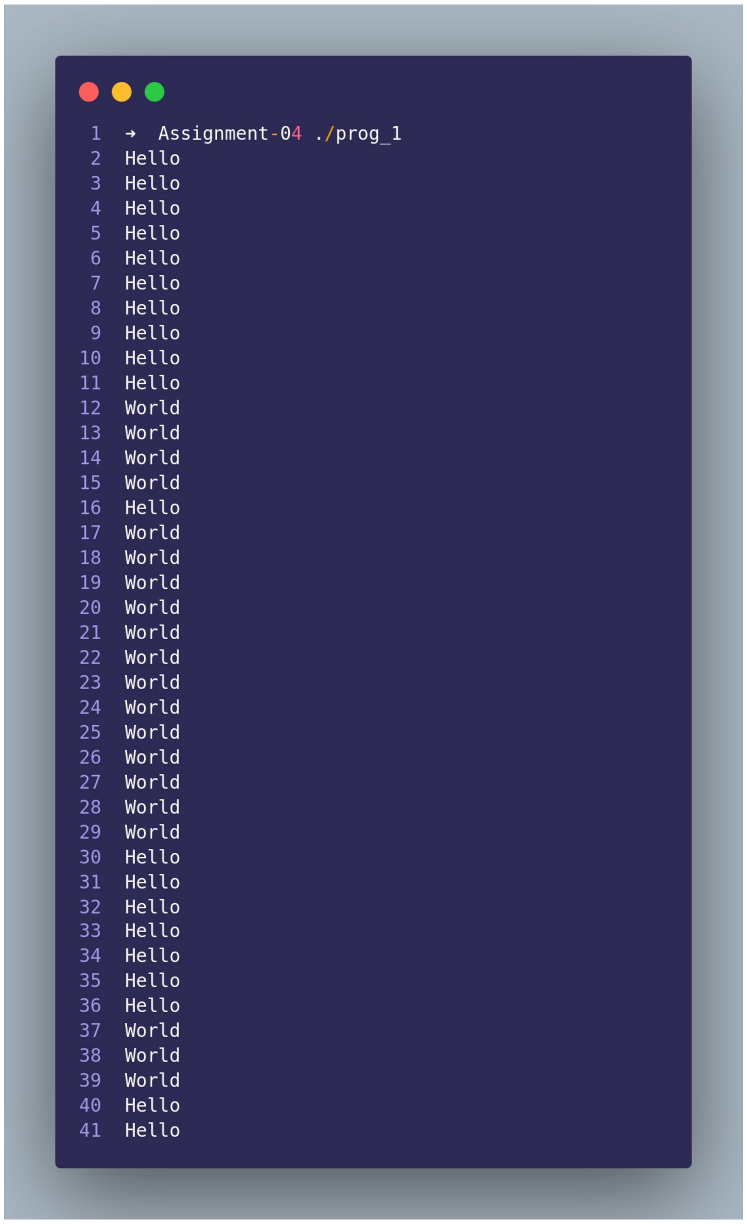
```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

void *thread1()
{
    for (int c = 0; c < 10; c++)
    {
        printf("Hello \n");
    }
}

void *thread2()
{
    for (int c = 0; c < 10; c++)
    {
        printf("World \n");
    }
}

int main()
{
    int status;
    pthread_t tid1, tid2, tid3, tid4;

    pthread_create(&tid1, NULL, thread1, NULL);
    pthread_create(&tid2, NULL, thread2, NULL);
    pthread_create(&tid3, NULL, thread1, NULL);
    pthread_create(&tid4, NULL, thread2, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    pthread_join(tid3, NULL);
    pthread_join(tid4, NULL);

    return 0;
}

```

6:



```

➜  Assignment-04 gcc -pthread -o prog_1 prog_1.c


```

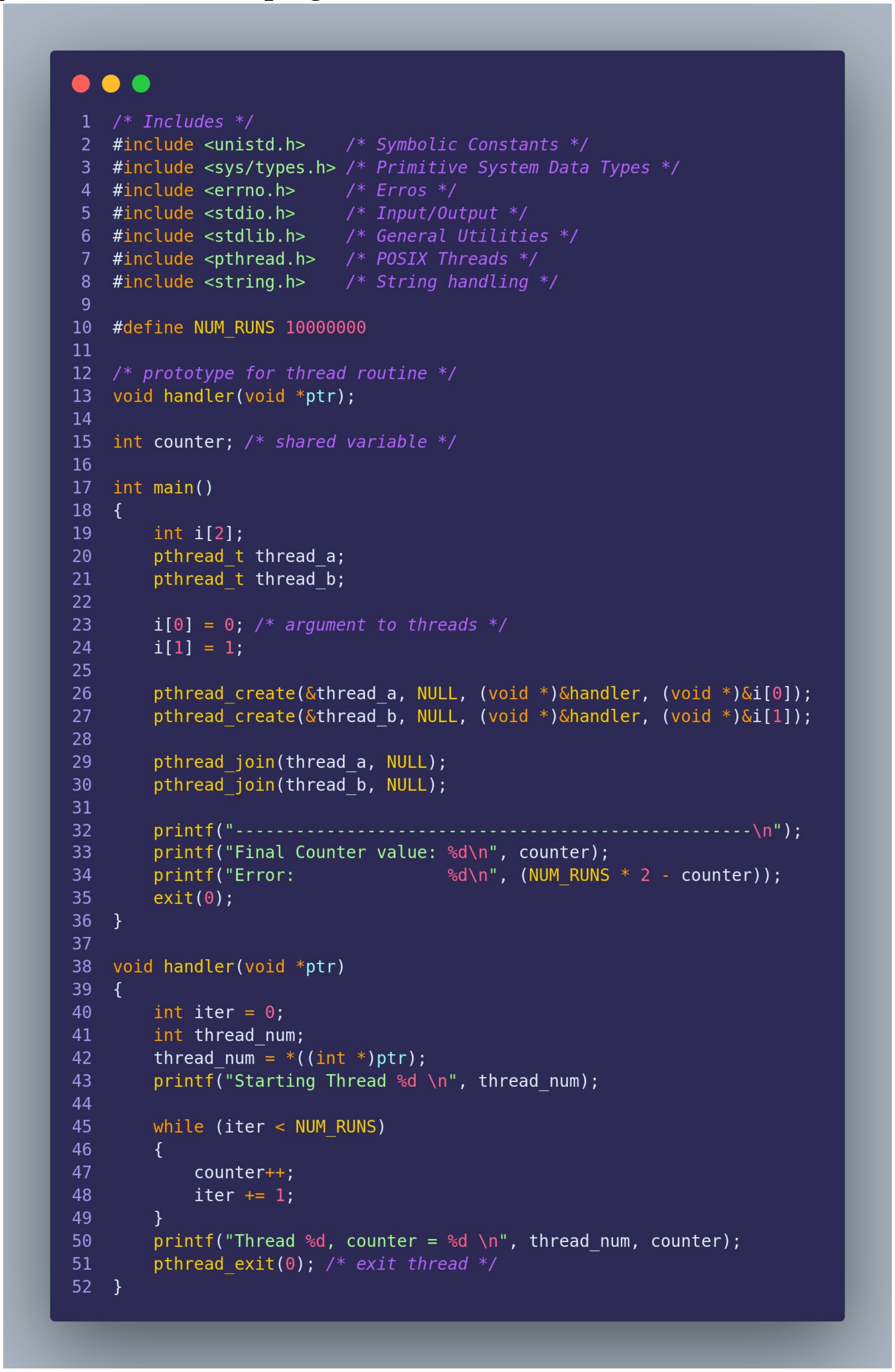
7:



```
➜  Assignment-04 ./prog_1
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
World
World
World
World
Hello
World
World
World
World
World
World
World
World
World
World
World
World
World
Hello
Hello
Hello
Hello
Hello
Hello
Hello
World
World
World
Hello
Hello
```

Now In this case the preemption occurs and the output of different threads shown in the way CPU schedule the threads or Light Weight Process.

(a) Compile and execute the program.



```c
/* Includes */
#include <unistd.h>    /* Symbolic Constants */
#include <sys/types.h> /* Primitive System Data Types */
#include <errno.h>     /* Erros */
#include <stdio.h>     /* Input/Output */
#include <stdlib.h>    /* General Utilities */
#include <pthread.h>   /* POSIX Threads */
#include <string.h>    /* String handling */

#define NUM_RUNS 10000000

/* prototype for thread routine */
void handler(void *ptr);

int counter; /* shared variable */

int main()
{
    int i[2];
    pthread_t thread_a;
    pthread_t thread_b;

    i[0] = 0; /* argument to threads */
    i[1] = 1;

    pthread_create(&thread_a, NULL, (void *)&handler, (void *)&i[0]);
    pthread_create(&thread_b, NULL, (void *)&handler, (void *)&i[1]);

    pthread_join(thread_a, NULL);
    pthread_join(thread_b, NULL);

    printf("---------------------------------------------------\n");
    printf("Final Counter value: %d\n", counter);
    printf("Error:               %d\n", (NUM_RUNS * 2 - counter));
    exit(0);
}

void handler(void *ptr)
{
    int iter = 0;
    int thread_num;
    thread_num = *((int *)ptr);
    printf("Starting Thread %d \n", thread_num);

    while (iter < NUM_RUNS)
    {
        counter++;
        iter += 1;
    }
    printf("Thread %d, counter = %d \n", thread_num, counter);
    pthread_exit(0); /* exit thread */
}
```

Answer the following questions:
1: What should be the value of the counter variable at the end?
Ans: *2 * NUM_RUNS or 2 * 10000000 =>* ***20000000***
2: What is the value you get?
Ans: Counter Value => *10357845*
3: How large is the error and how much does it vary on different runs?
Ans: Error => 9642155 , *The Value of the Error Lies between 8000000 and 10000000 and it not greater then this range nor less than.*
4: How much user time (rougly) does the program take to run on your system?
Ans: By Taking the Average User Time => 0.80sec