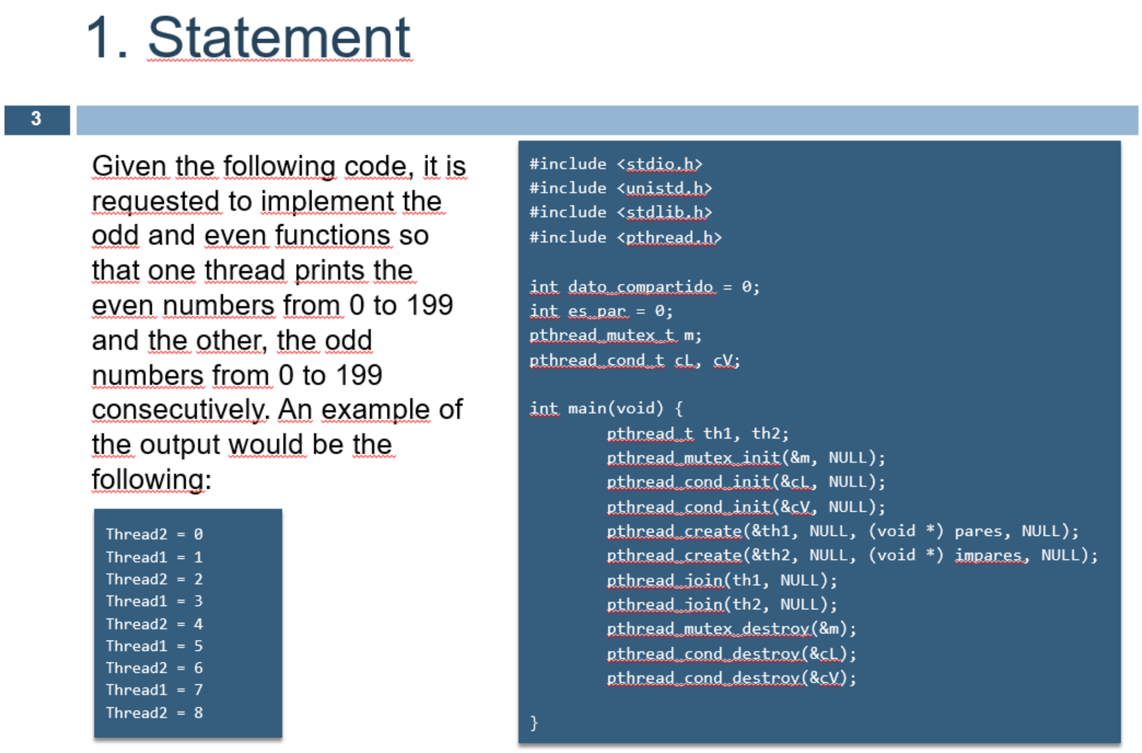

1. Statement

3

Given the following code, it is requested to implement the odd and even functions so that one thread prints the even numbers from 0 to 199 and the other, the odd numbers from 0 to 199 consecutively. An example of the output would be the following:

```
Thread2 = 0
Thread1 = 1
Thread2 = 2
Thread1 = 3
Thread2 = 4
Thread1 = 5
Thread2 = 6
Thread1 = 7
Thread2 = 8
```

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

int dato_compartido = 0;
int es_par = 0;
pthread_mutex_t m;
pthread_cond_t cL, cV;

int main(void) {
        pthread_t th1, th2;
        pthread_mutex_init(&m, NULL);
        pthread_cond_init(&cL, NULL);
        pthread_cond_init(&cV, NULL);
        pthread_create(&th1, NULL, (void *) pares, NULL);
        pthread_create(&th2, NULL, (void *) impares, NULL);
        pthread_join(th1, NULL);
        pthread_join(th2, NULL);
        pthread_mutex_destroy(&m);
        pthread_cond_destroy(&cL);
        pthread_cond_destroy(&cV);

}
```

SOLUTION:

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <pthread.h>


// Function declarations

void pares(void);

void impares(void);


int dato_compartido = 0;

int es_par = 0;

pthread_mutex_t m;

pthread_cond_t cL, cV;


void pares(void)

{
    int i;
    for(i=0; i < 100; i++ )
    {
        pthread_mutex_lock(&m);
        if (es_par==0)
        {
            pthread_cond_signal(&cL);
        }
        printf("Thread1 = %d \n", dato_compartido++);
        es_par=1;
        pthread_cond_wait(&cV,&m);
        pthread_mutex_unlock(&m);
    }
}

void impares(void)
{
   int i;
   for(i=0; i < 100; i++ )
   {
        pthread_mutex_lock(&m);
        if (es_par==1)
        {
            pthread_cond_signal(&cV);
        }
        printf("Thread2 = %d \n", dato_compartido++);
        es_par=0;
        pthread_cond_wait(&cL, &m);
        pthread_mutex_unlock(&m);
    }
}
```

```
int main(void) {
        pthread_t th1, th2;
        pthread_mutex_init(&m, NULL);
        pthread_cond_init(&cL, NULL);
        pthread_cond_init(&cV, NULL);
        pthread_create(&th1, NULL, (void *) pares, NULL);
        pthread_create(&th2, NULL, (void *) impares, NULL);
        pthread_join(th1, NULL);
        pthread_join(th2, NULL);
        pthread_mutex_destroy(&m);
        pthread_cond_destroy(&cL);
        pthread_cond_destroy(&cV);
        return 0;
}
```

When the even-numbered thread acquires the mutex, it first checks if the variable is_even is 0. If so, it sends a signal to the condition variable cL (which the odd-numbered thread is waiting for). It then prints the current number (shared_data) and updates the variable is_even to 1 to indicate that the next number should be odd.

Then, it waits on the condition variable cV (which is being waited on by the even-numbered thread). When the condition is met (ie, when the odd-numbered thread sends a signal to cV), the even-numbered thread will unlock the mutex and allow the odd-numbered thread to continue execution.

The odd-numbered thread performs a similar process, but this time it sends a signal to the condition variable cV before waiting on the condition variable cL. This way, it makes sure that the even numbered thread is notified to continue its execution and be able to print the next even number.

In short, condition variables allow threads to wait for a certain condition to be met before continuing their execution, which is useful for synchronizing the behavior of multiple threads.

In this case, they are used to ensure that the odd and even threads print the numbers in alternating order.