# Homework 2 – Ταυτόχρονος Προγραμματισμός

Γαρυφαλλιά Αναστασία Παπαδούλη | 3533

Δημήτριος Τσαλαπατάς | 3246

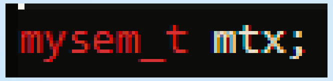
Νικόλαος Μπέτσος | 3267

Ιωάννης Ρείνος | 3390

09/12/2023

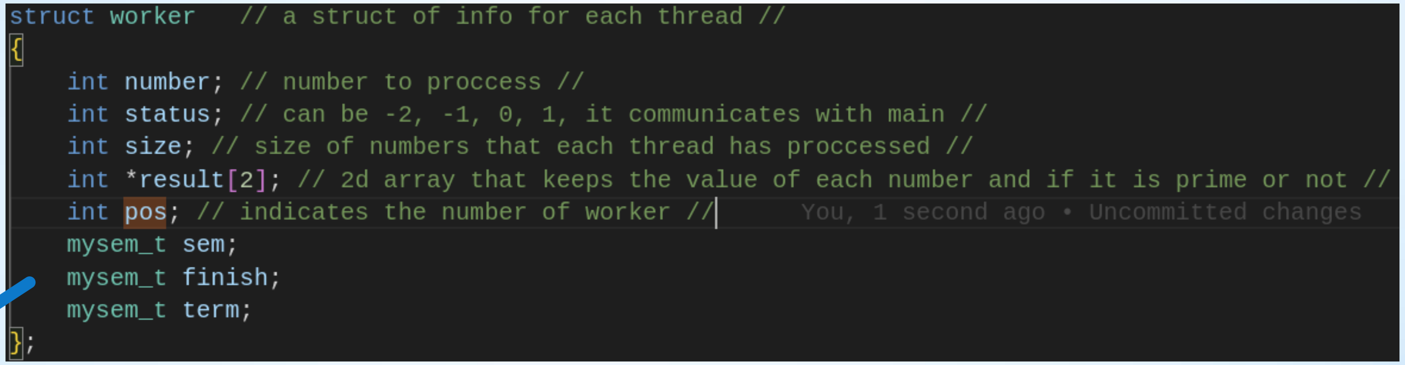
Ομάδα 4

# Assignment 2

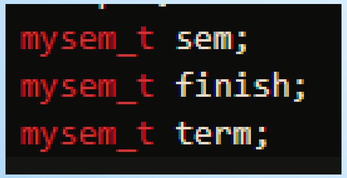


```
mysem_t mtx;
```

Semaphore mtx: Used for data protection between shared data



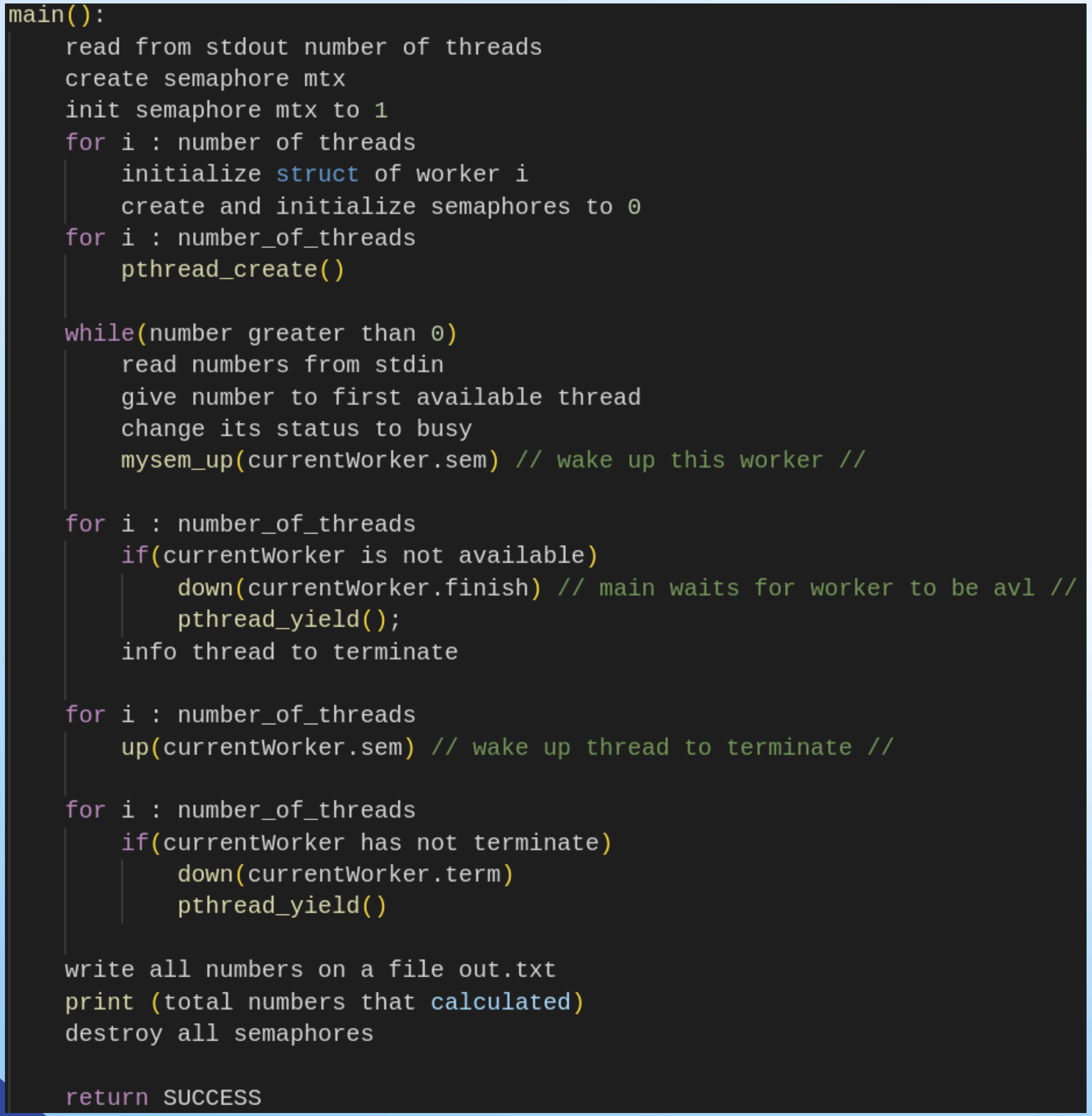
```
struct worker   // a struct of info for each thread //
{
    int number; // number to proccess //
    int status; // can be -2, -1, 0, 1, it communicates with main //
    int size; // size of numbers that each thread has proccessed //
    int *result[2]; // 2d array that keeps the value of each number and if it is prime or not //
    int pos; // indicates the number of worker //      You, 1 second ago • Uncommitted changes
    mysem_t sem;
    mysem_t finish;
    mysem_t term;
};
```



```
mysem_t sem;
mysem_t finish;
mysem_t term;
```

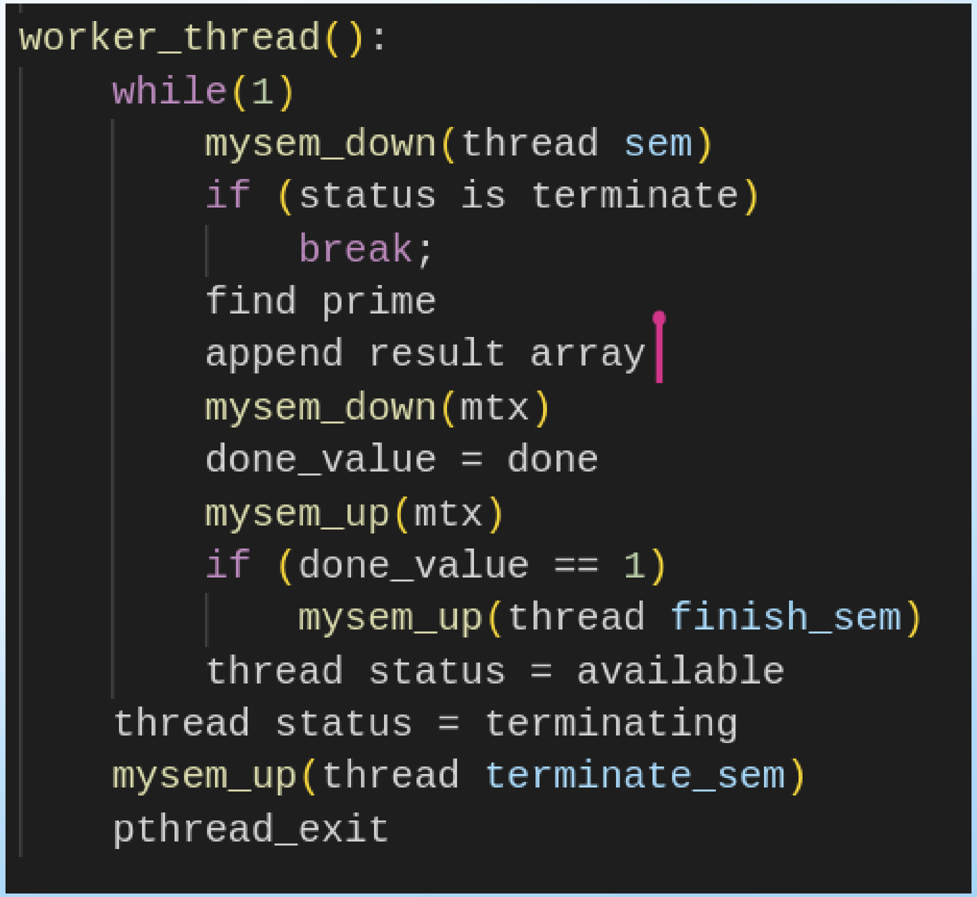
- Semaphore sem: 1 for every thread, used for sync between said thread and main.
- Semaphore finish: determines when threads are ready to exit
- Semaphore term: determines threads actually exit

## MAIN



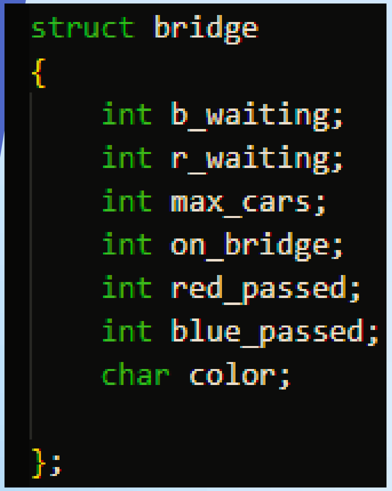
```
main():
    read from stdout number of threads
    create semaphore mtx
    init semaphore mtx to 1
    for i : number of threads
        initialize struct of worker i
        create and initialize semaphores to 0
    for i : number_of_threads
        pthread_create()

    while(number greater than 0)
        read numbers from stdin
        give number to first available thread
        change its status to busy
        mysem_up(currentWorker.sem) // wake up this worker //

    for i : number_of_threads
        if(currentWorker is not available)
            down(currentWorker.finish) // main waits for worker to be avl //
            pthread_yield();
        info thread to terminate

    for i : number_of_threads
        up(currentWorker.sem) // wake up thread to terminate //

    for i : number_of_threads
        if(currentWorker has not terminate)
            down(currentWorker.term)
            pthread_yield()

    write all numbers on a file out.txt
    print (total numbers that calculated)
    destroy all semaphores

    return SUCCESS
```

## THREAD



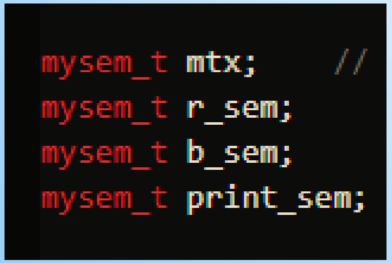
```
worker_thread():
    while(1)
        mysem_down(thread sem)
        if (status is terminate)
            break;
        find prime
        append result array
        mysem_down(mtx)
        done_value = done
        mysem_up(mtx)
        if (done_value == 1)
            mysem_up(thread finish_sem)
        thread status = available
    thread status = terminating
    mysem_up(thread terminate_sem)
    pthread_exit
```

# Assignment 3



```c
struct bridge
{
	int b_waiting;
	int r_waiting;
	int max_cars;
	int on_bridge;
	int red_passed;
	int blue_passed;
	char color;

};
```

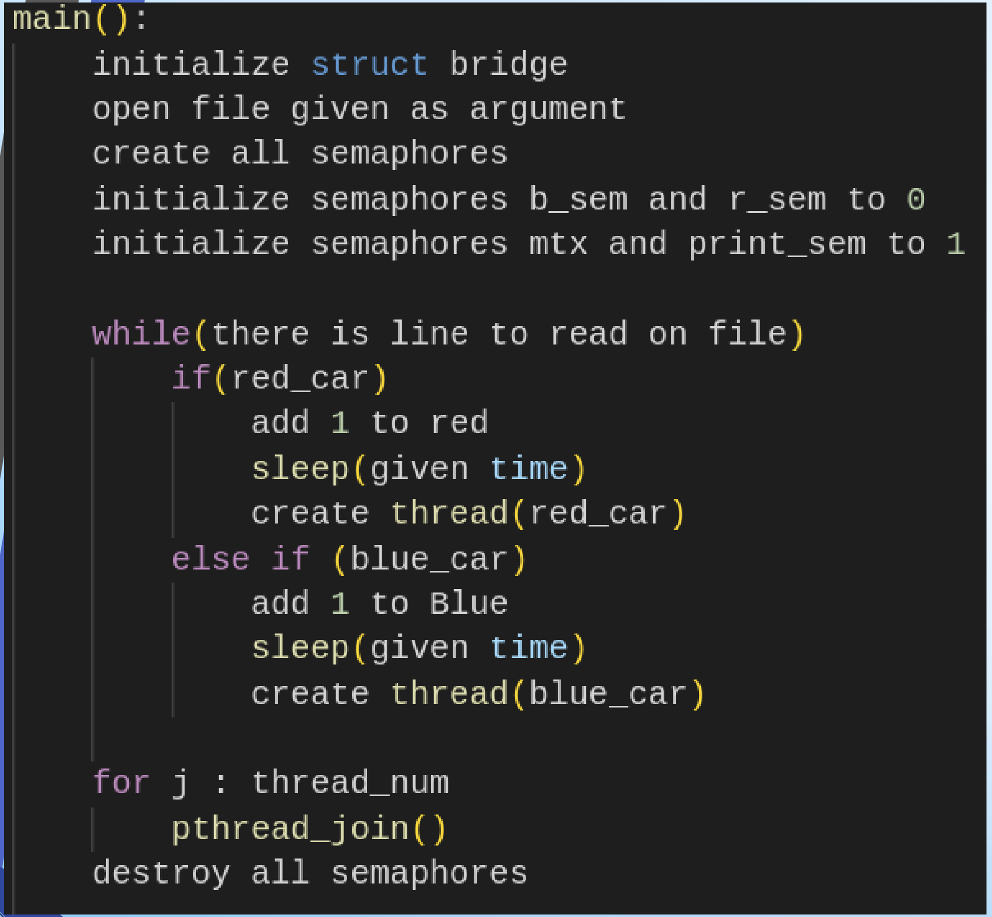
- b_waiting: num of blue cars waiting to pass
- r_waiting: num of red cars waiting to pass
- max_cars: limit of cars that can cross bridge simultaneously
- on_bridge: num of cars currently on bridge
- Red_passed: num of
- Blue_passed:
- Color: Color of cars currently in bridge – r,b or \o if empty



```c
mysem_t mtx;      //
mysem_t r_sem;
mysem_t b_sem;
mysem_t print_sem;
```

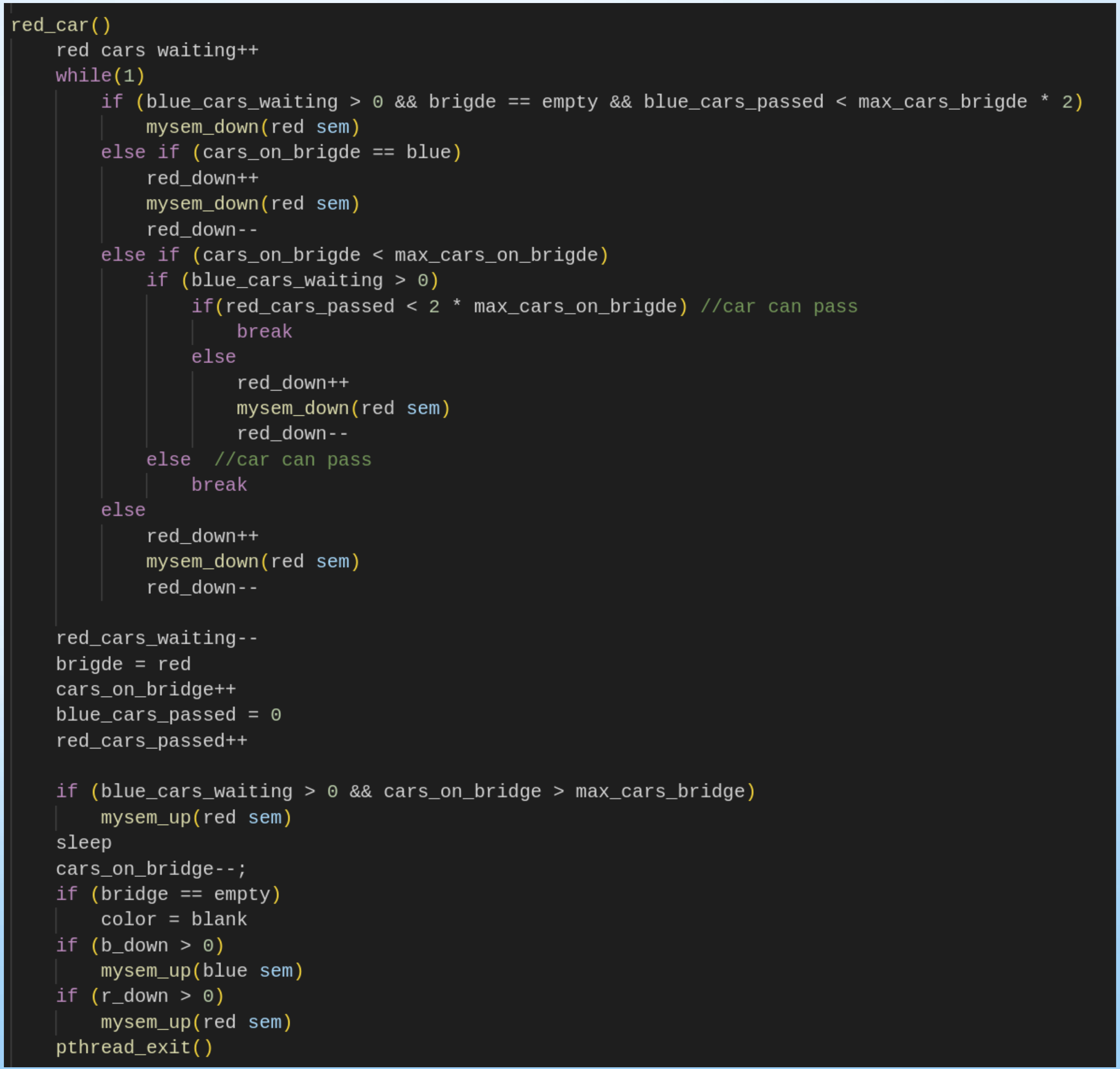
- Mtx: Protection of shared data
- R_sem: syncronisation of red semaphores
- B_sem: syncronisation of blue semaphores

## MAIN

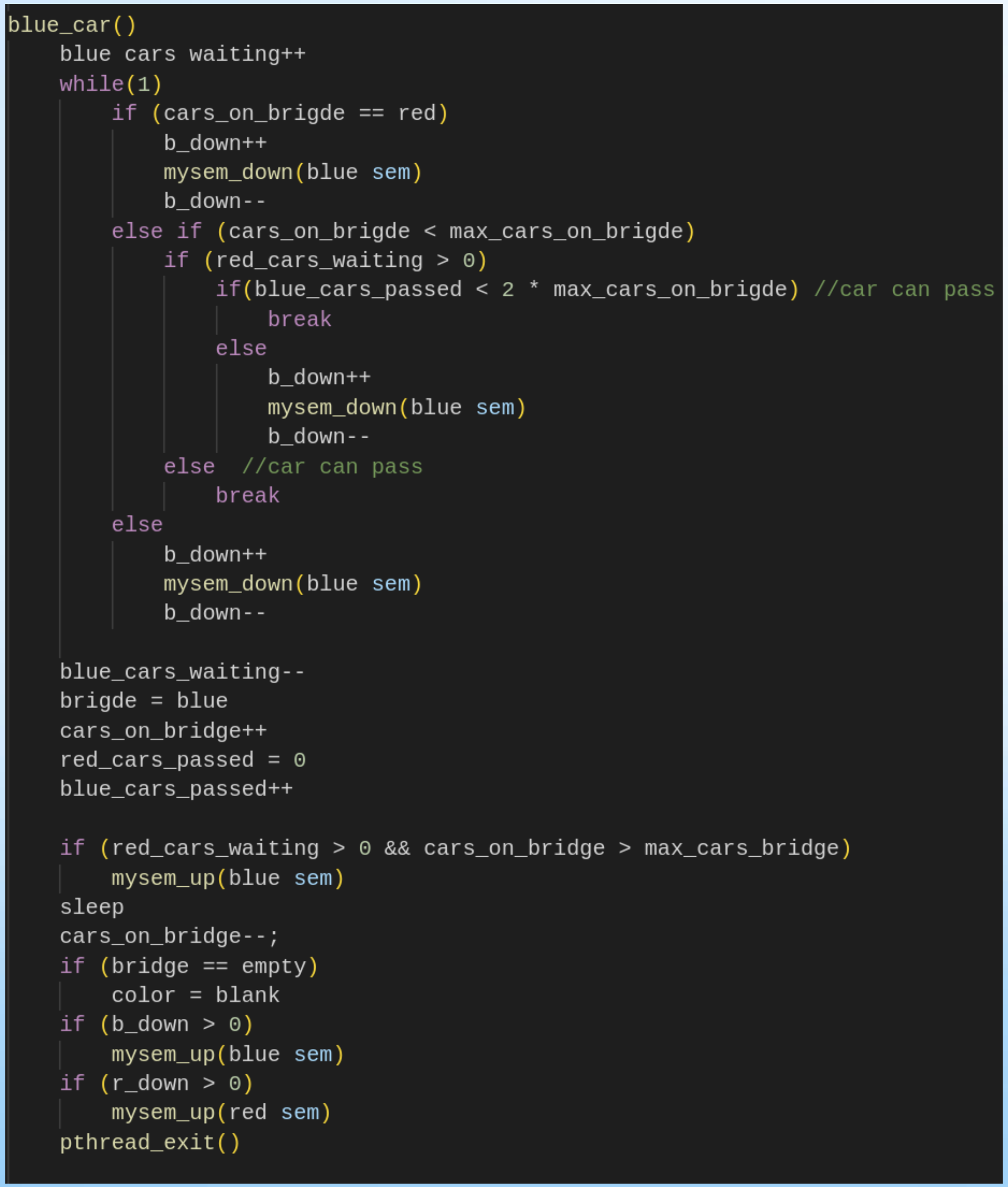


```
main():
    initialize struct bridge
    open file given as argument
    create all semaphores
    initialize semaphores b_sem and r_sem to 0
    initialize semaphores mtx and print_sem to 1

    while(there is line to read on file)
        if(red_car)
            add 1 to red
            sleep(given time)
            create thread(red_car)
        else if (blue_car)
            add 1 to Blue
            sleep(given time)
            create thread(blue_car)

    for j : thread_num
        pthread_join()
    destroy all semaphores
```

## Red car



```
red_car()
    red cars waiting++
    while(1)
        if (blue_cars_waiting > 0 && brigde == empty && blue_cars_passed < max_cars_brigde * 2)
            mysem_down(red sem)
        else if (cars_on_brigde == blue)
            red_down++
            mysem_down(red sem)
            red_down--
        else if (cars_on_brigde < max_cars_on_brigde)
            if (blue_cars_waiting > 0)
                if(red_cars_passed < 2 * max_cars_on_brigde) //car can pass
                    break
                else
                    red_down++
                    mysem_down(red sem)
                    red_down--
            else  //car can pass
                break
        else
            red_down++
            mysem_down(red sem)
            red_down--

    red_cars_waiting--
    brigde = red
    cars_on_bridge++
    blue_cars_passed = 0
    red_cars_passed++

    if (blue_cars_waiting > 0 && cars_on_bridge > max_cars_bridge)
        mysem_up(red sem)
    sleep
    cars_on_bridge--;
    if (bridge == empty)
        color = blank
    if (b_down > 0)
        mysem_up(blue sem)
    if (r_down > 0)
        mysem_up(red sem)
    pthread_exit()
```

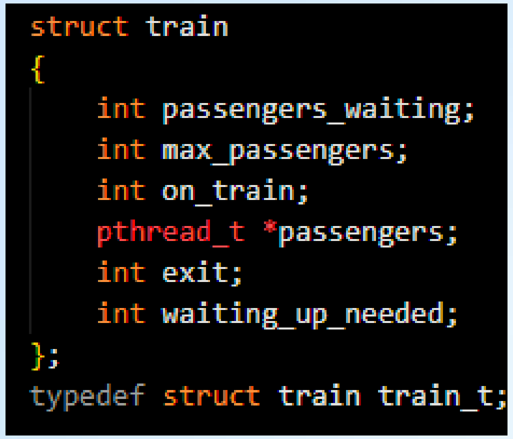


```
blue_car()
    blue cars waiting++
    while(1)
        if (cars_on_brigde == red)
            b_down++
            mysem_down(blue sem)
            b_down--
        else if (cars_on_brigde < max_cars_on_brigde)
            if (red_cars_waiting > 0)
                if(blue_cars_passed < 2 * max_cars_on_brigde) //car can pass
                    break
                else
                    b_down++
                    mysem_down(blue sem)
                    b_down--
            else  //car can pass
                break
        else
            b_down++
            mysem_down(blue sem)
            b_down--

    blue_cars_waiting--
    brigde = blue
    cars_on_bridge++
    red_cars_passed = 0
    blue_cars_passed++

    if (red_cars_waiting > 0 && cars_on_bridge > max_cars_bridge)
        mysem_up(blue sem)
    sleep
    cars_on_bridge--;
    if (bridge == empty)
        color = blank
    if (b_down > 0)
        mysem_up(blue sem)
    if (r_down > 0)
        mysem_up(red sem)
    pthread_exit()
```

## ***<u>Blue car:</u>***

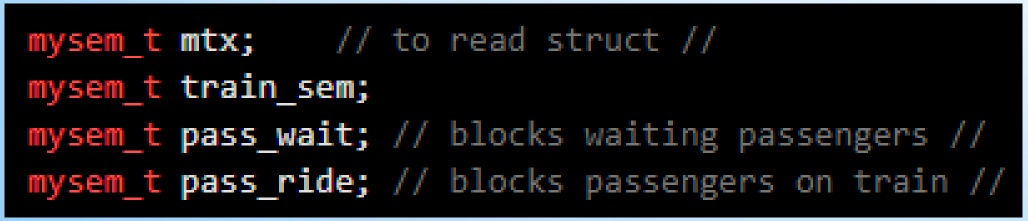
Basically, a mirror of red car without the first condition.

# Assignment 4



```c
struct train
{
    int passengers_waiting;
    int max_passengers;
    int on_train;
    pthread_t *passengers;
    int exit;
    int waiting_up_needed;
};
typedef struct train train_t;
```

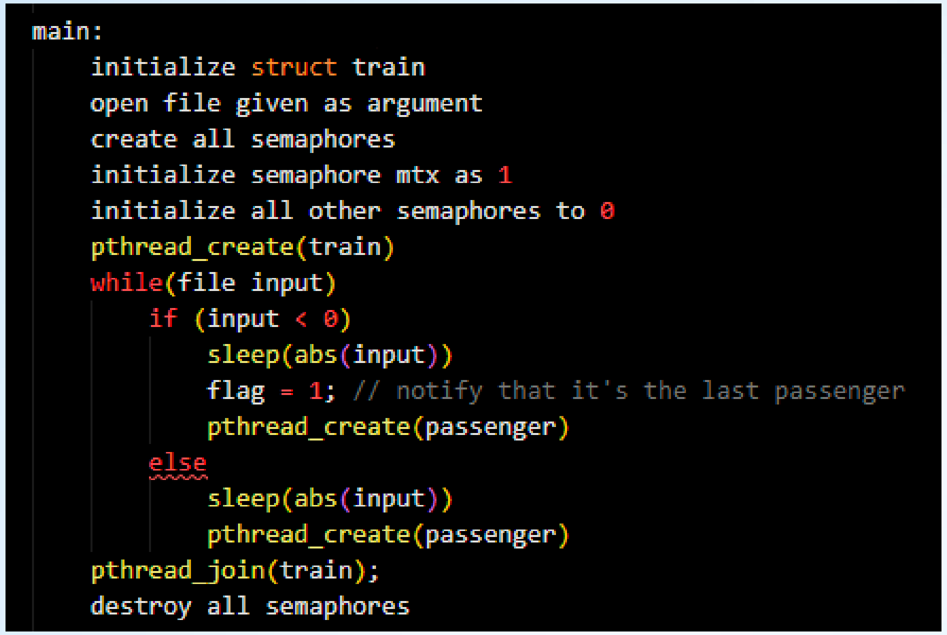
- Passengers_waiting: num of passengers waiting for a ride
- Max_passengers: limit of passengers per ride
- on_train: passengers currently on bridge
- Passengers: handles of threads corresponding to passengers [for debug purposes]
- Exit: signifies no more rides should be performed
- Waiting_up_needed: Used to regulate how many threads to wake up



```c
mysem_t mtx;     // to read struct //
mysem_t train_sem;
mysem_t pass_wait; // blocks waiting passengers //
mysem_t pass_ride; // blocks passengers on train //
```

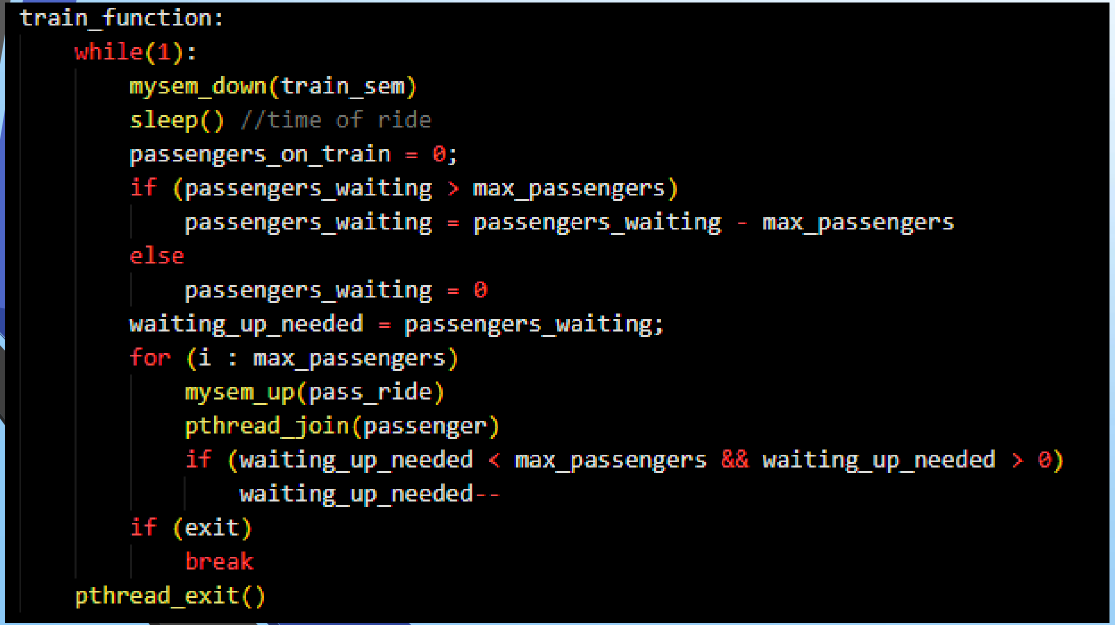
- Mtx: Protection of shared data
- Train_sem: sync between train and coming passengers
- Pass_wait: blocks waiting passengers outside of train
- Pass_ride: blocks passengers already on train

## ***MAIN:***



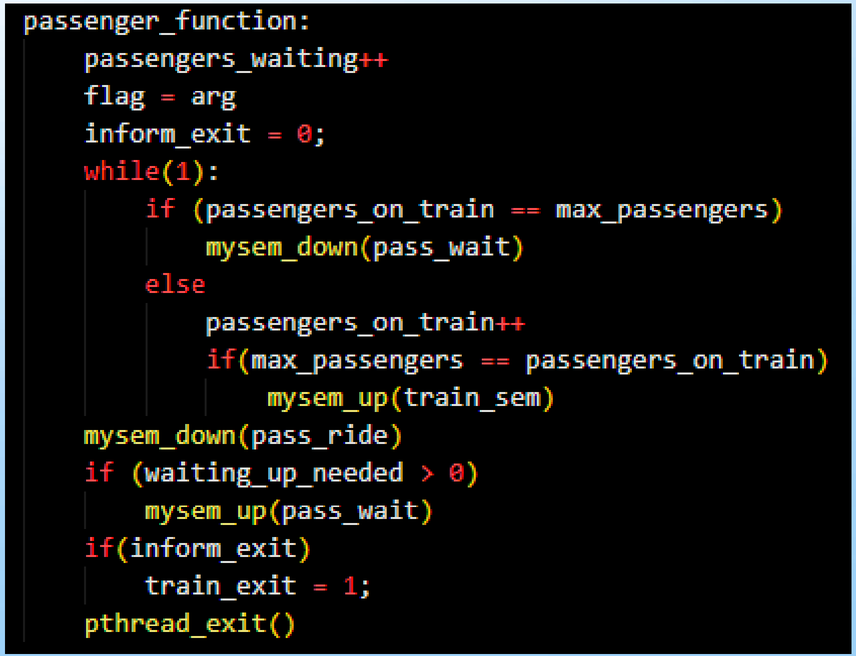
```
main:
    initialize struct train
    open file given as argument
    create all semaphores
    initialize semaphore mtx as 1
    initialize all other semaphores to 0
    pthread_create(train)
    while(file input)
        if (input < 0)
            sleep(abs(input))
            flag = 1; // notify that it's the last passenger
            pthread_create(passenger)
        else
            sleep(abs(input))
            pthread_create(passenger)
    pthread_join(train);
    destroy all semaphores
```

## ***Train_function:***



```
train_function:
    while(1):
        mysem_down(train_sem)
        sleep() //time of ride
        passengers_on_train = 0;
        if (passengers_waiting > max_passengers)
            passengers_waiting = passengers_waiting - max_passengers
        else
            passengers_waiting = 0
        waiting_up_needed = passengers_waiting;
        for (i : max_passengers)
            mysem_up(pass_ride)
            pthread_join(passenger)
            if (waiting_up_needed < max_passengers && waiting_up_needed > 0)
                waiting_up_needed--
        if (exit)
            break
    pthread_exit()
```

## ***Passenger_function:***



```
passenger_function:
    passengers_waiting++
    flag = arg
    inform_exit = 0;
    while(1):
        if (passengers_on_train == max_passengers)
            mysem_down(pass_wait)
        else
            passengers_on_train++
            if(max_passengers == passengers_on_train)
                mysem_up(train_sem)
    mysem_down(pass_ride)
    if (waiting_up_needed > 0)
        mysem_up(pass_wait)
    if(inform_exit)
        train_exit = 1;
    pthread_exit()
```

# Thanks for your time!

Team 4