CAR PARKING SEMULATION

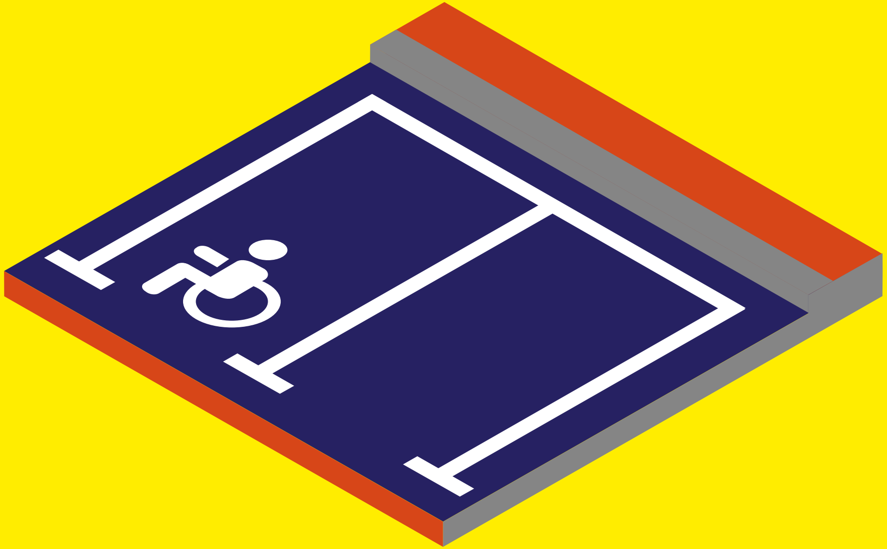
Main Algorithm Design
EE-463 Term Project

BG02
1- Hayan Al-Machnouk
2- Tamam Alahdal
3- Zyad Zamil

TABLE OF CONTENTS

01

INTRODUCTION

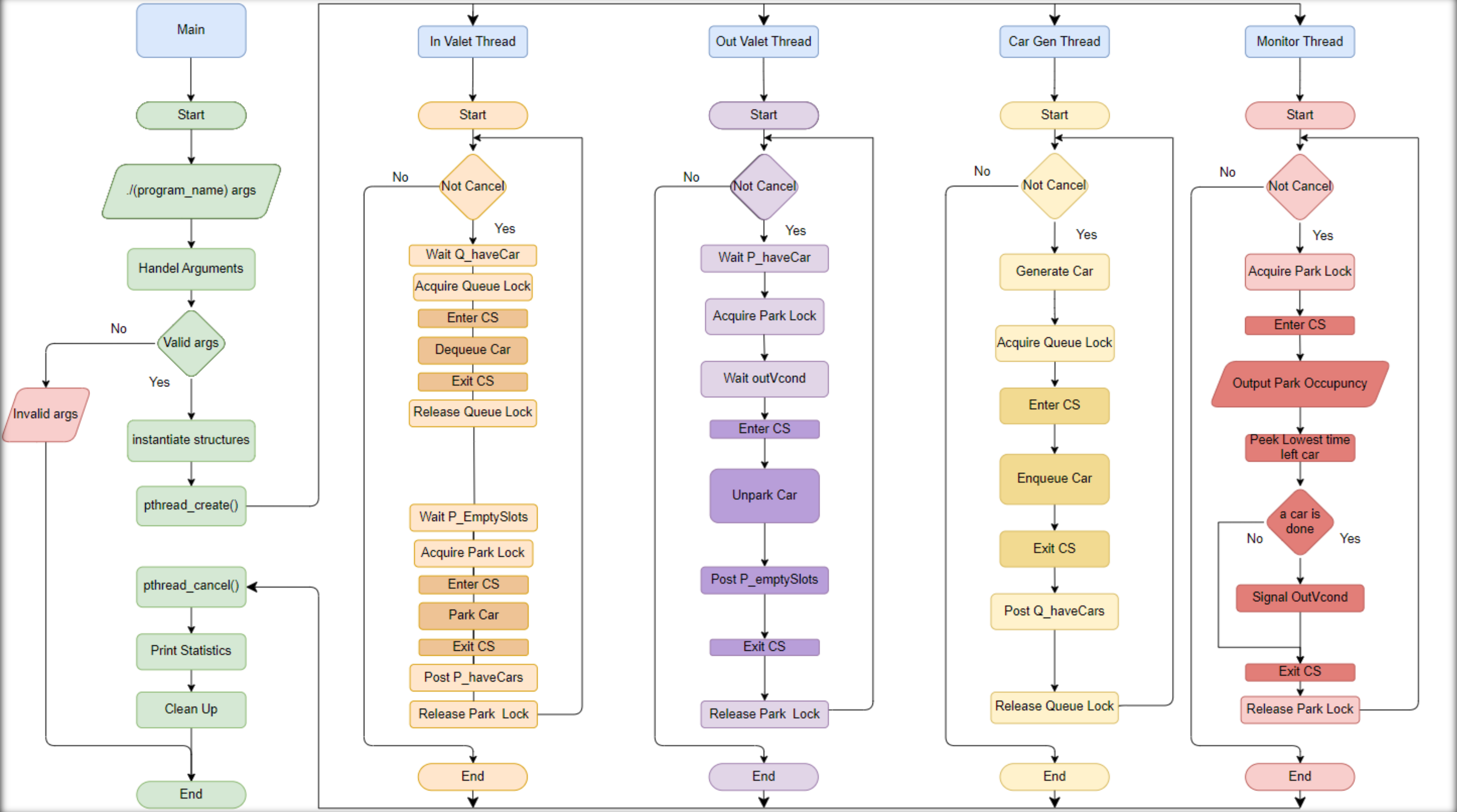


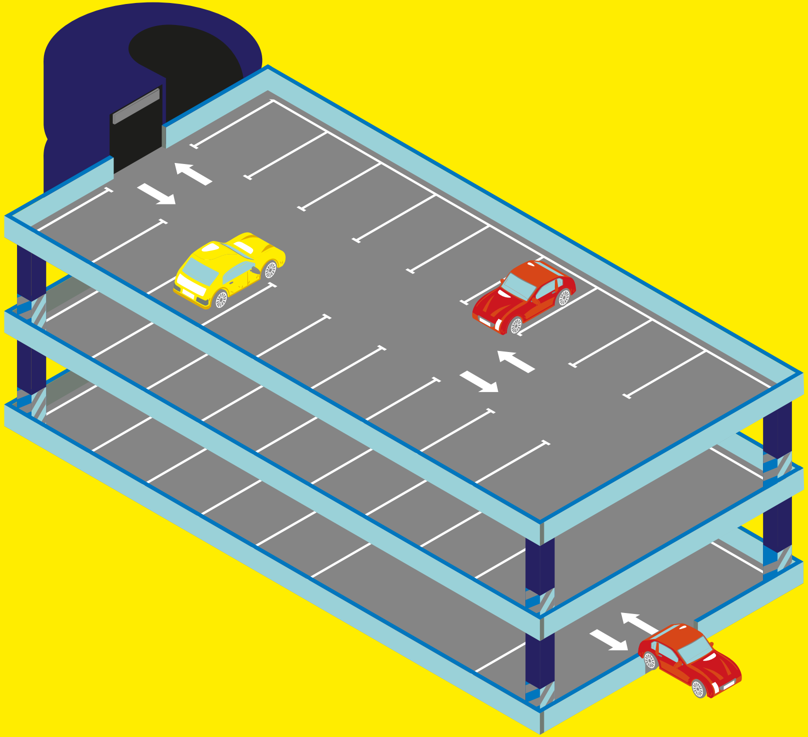
INTRODUCTION

- **Threads: the program will use a set of threads to perform certain functions.**
- **Mutex: a Mutex will be used to protect the shared resources in the program**
- **Semaphores: Semaphores will be used to avoid the busy-waiting**
- **Car park array: a park array will be used to represent the car park space.**
- **Car Queue: we implemented Queue data structure to hold cars.**

02

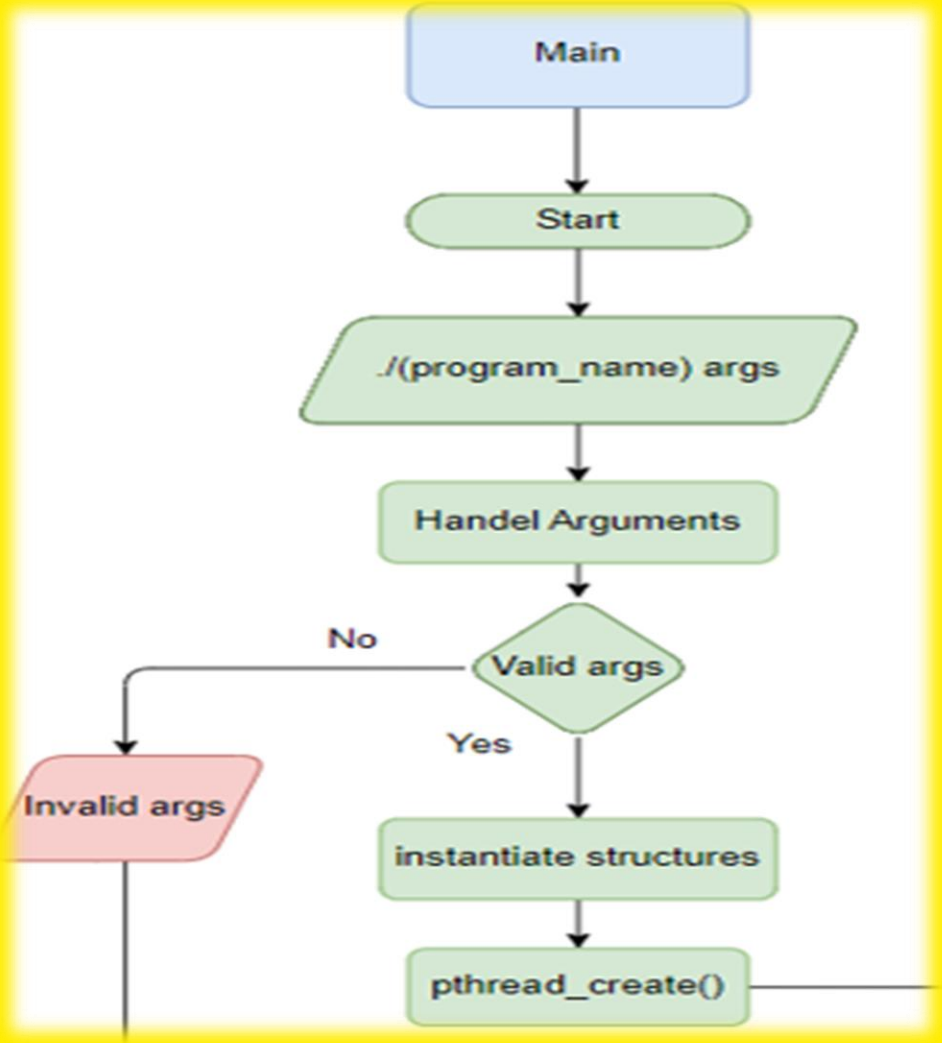
Flowchart


Main
Start
./(program_name) args
Handel Arguments
Valid args
No
Yes
Invalid args
instantiate structures
pthread_create()
pthread_cancel()
Print Statistics
Clean Up
End
In Valet Thread
Start
Not Cancel
No
Yes
Wait Q_haveCar
Acquire Queue Lock
Enter CS
Dequeue Car
Exit CS
Release Queue Lock
Wait P_EmptySlots
Acquire Park Lock
Enter CS
Park Car
Exit CS
Post P_haveCars
Release Park Lock
End
Out Valet Thread
Start
Not Cancel
No
Yes
Wait P_haveCar
Acquire Park Lock
Wait outVcond
Enter CS
Unpark Car
Post P_emptySlots
Exit CS
Release Park Lock
End
Car Gen Thread
Start
Not Cancel
No
Yes
Generate Car
Acquire Queue Lock
Enter CS
Enqueue Car
Exit CS
Post Q_haveCars
Release Queue Lock
End
Monitor Thread
Start
Not Cancel
No
Yes
Acquire Park Lock
Enter CS
Output Park Occupuncy
Peek Lowest time left car
a car is done
No
Yes
Signal OutVcond
Exit CS
Release Park Lock
End



03

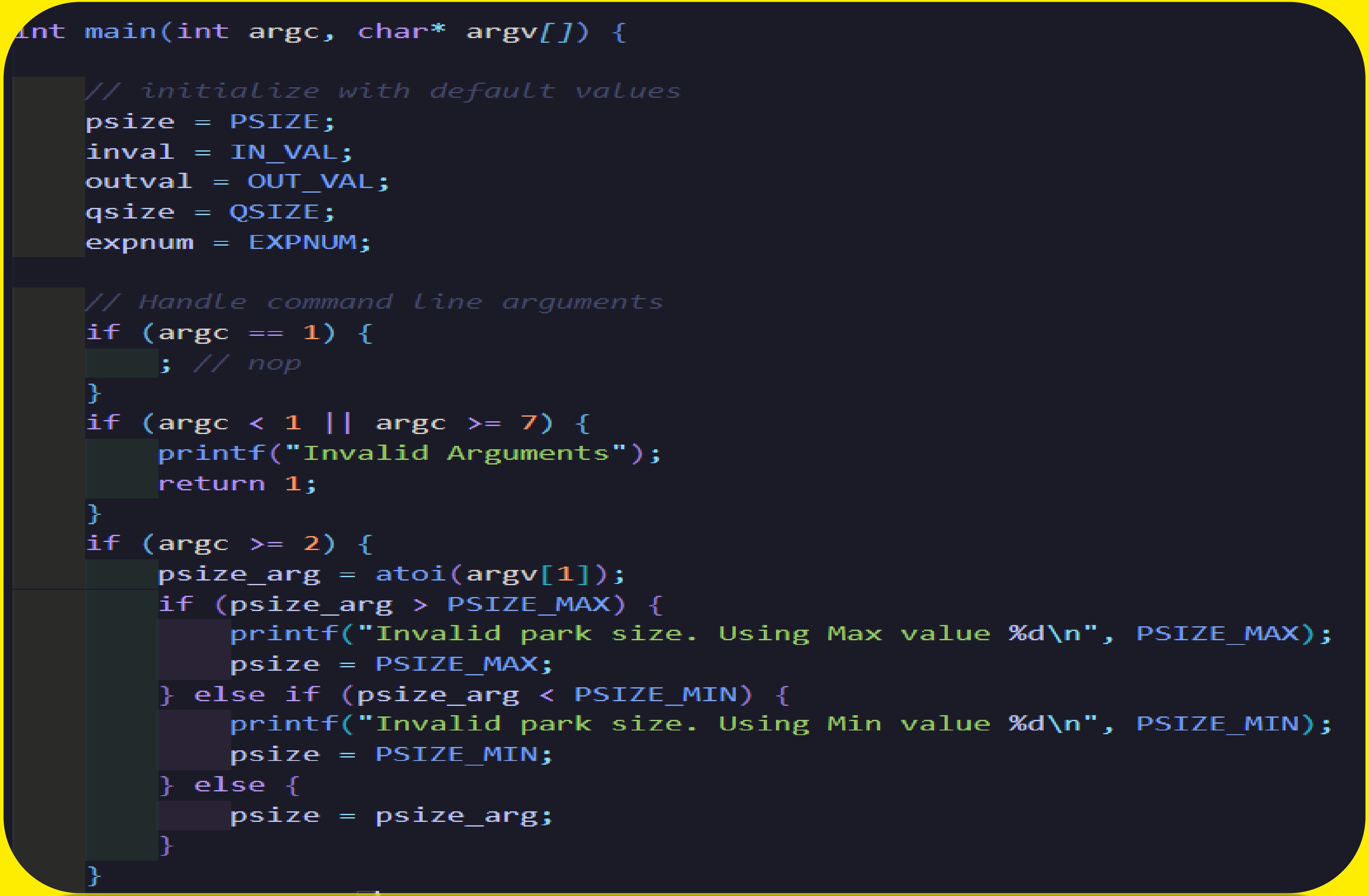
MAIN THREAD



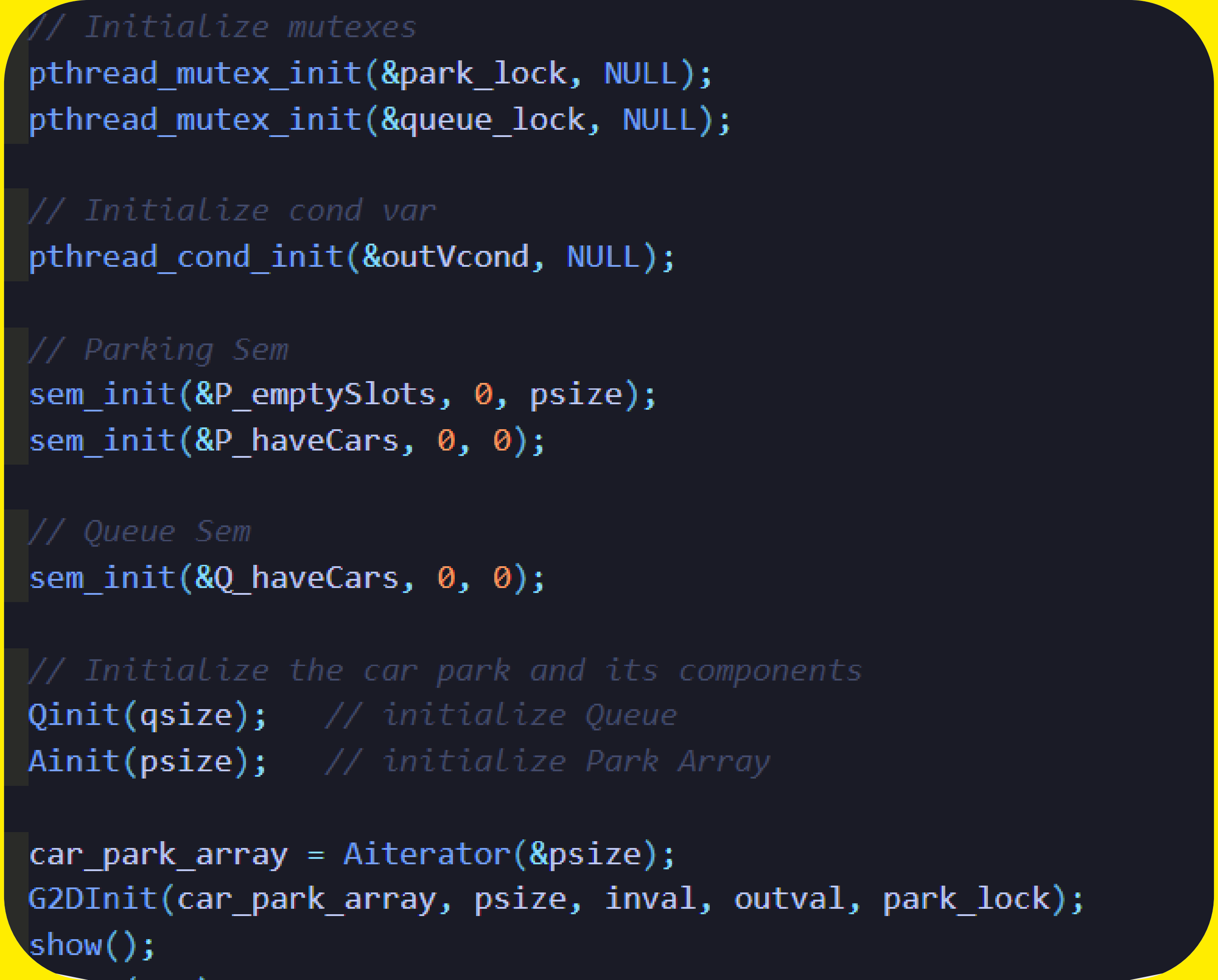
ARGUMENTS

INITIALIZE STRUCTURES

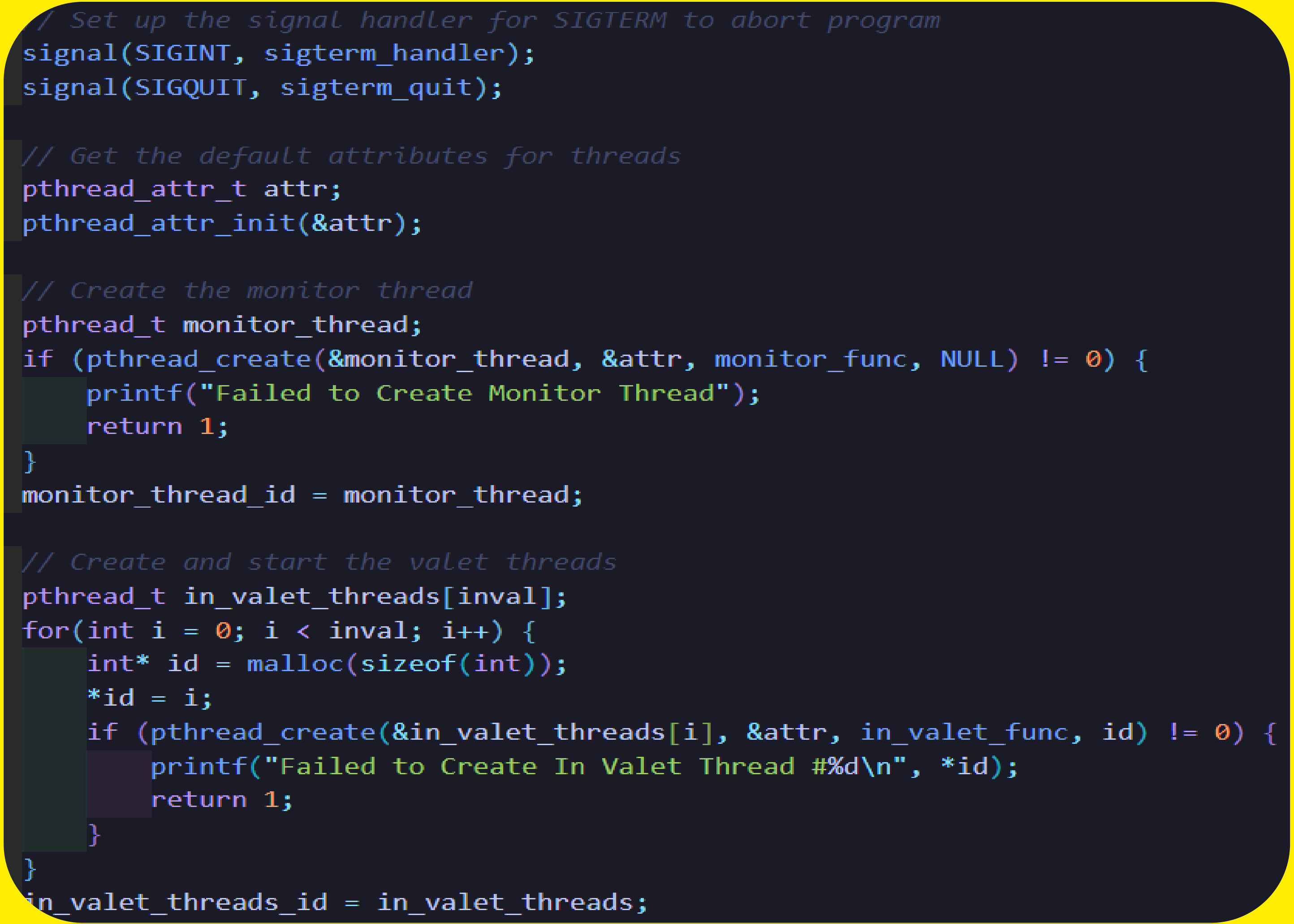
CREATE THREADS



```
int main(int argc, char* argv[]) {

    // initialize with default values
    psize = PSIZE;
    inval = IN_VAL;
    outval = OUT_VAL;
    qsize = QSIZE;
    expnum = EXPNUM;

    // Handle command line arguments
    if (argc == 1) {
        ; // nop
    }
    if (argc < 1 || argc >= 7) {
        printf("Invalid Arguments");
        return 1;
    }
    if (argc >= 2) {
        psize_arg = atoi(argv[1]);
        if (psize_arg > PSIZE_MAX) {
            printf("Invalid park size. Using Max value %d\n", PSIZE_MAX);
            psize = PSIZE_MAX;
        } else if (psize_arg < PSIZE_MIN) {
            printf("Invalid park size. Using Min value %d\n", PSIZE_MIN);
            psize = PSIZE_MIN;
        } else {
            psize = psize_arg;
        }
    }
```



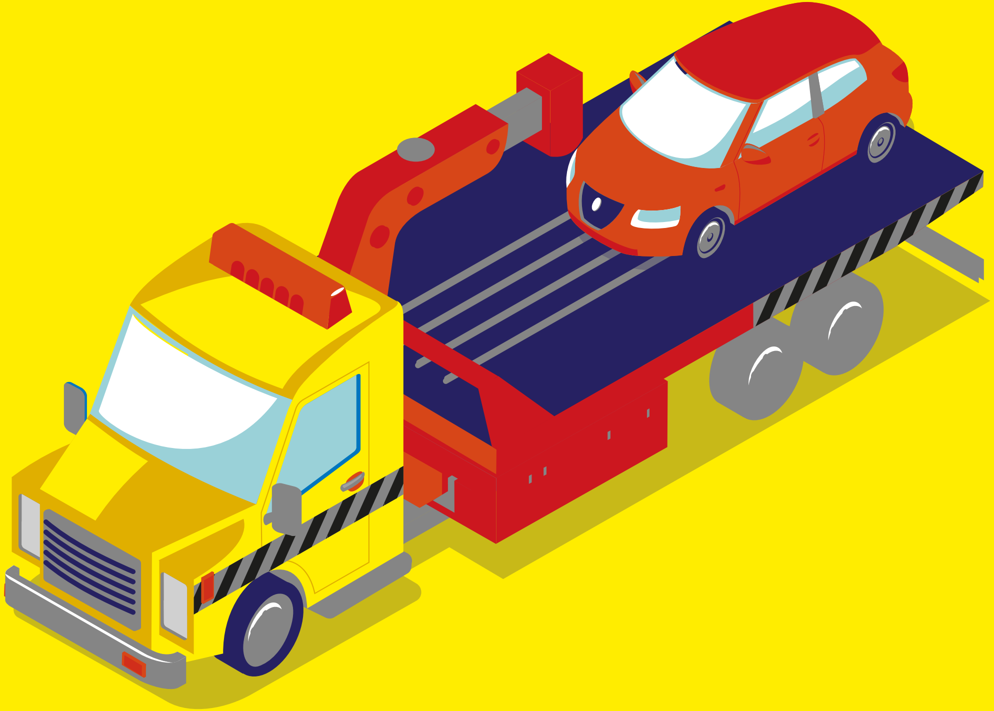
```
// Initialize mutexes
pthread_mutex_init(&park_lock, NULL);
pthread_mutex_init(&queue_lock, NULL);

// Initialize cond var
pthread_cond_init(&outVcond, NULL);

// Parking Sem
sem_init(&P_emptySlots, 0, psize);
sem_init(&P_haveCars, 0, 0);

// Queue Sem
sem_init(&Q_haveCars, 0, 0);

// Initialize the car park and its components
Qinit(qsize);   // initialize Queue
Ainit(psize);   // initialize Park Array

car_park_array = Aiterator(&psize);
G2DInit(car_park_array, psize, inval, outval, park_lock);
show();
```

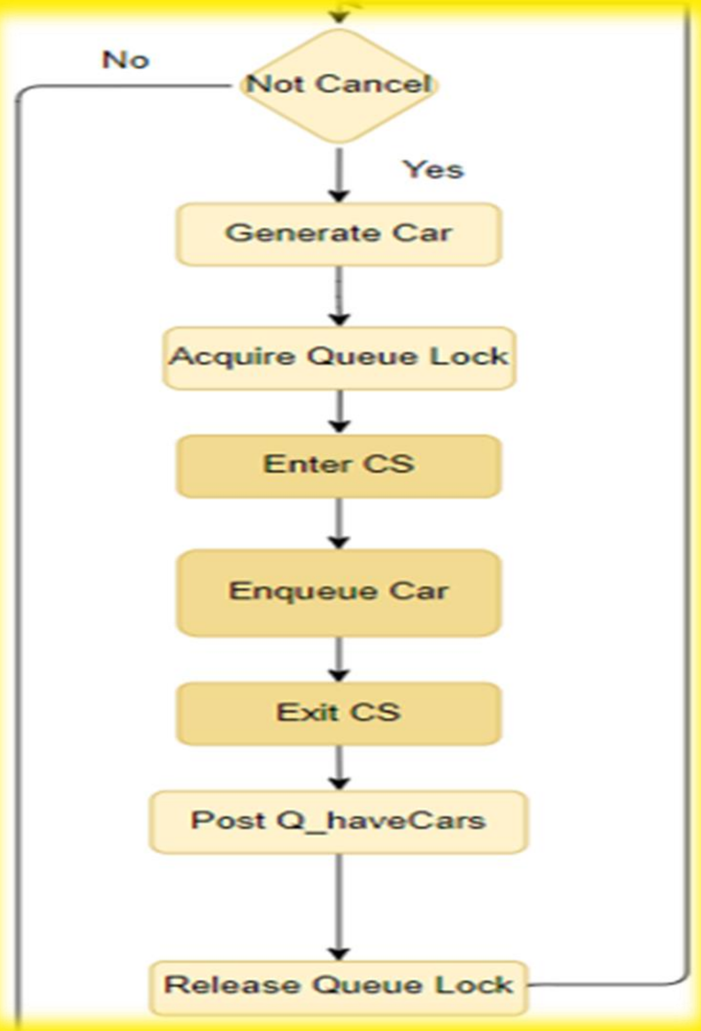


```
// Set up the signal handler for SIGTERM to abort program
signal(SIGINT, sigterm_handler);
signal(SIGQUIT, sigterm_quit);

// Get the default attributes for threads
pthread_attr_t attr;
pthread_attr_init(&attr);

// Create the monitor thread
pthread_t monitor_thread;
if (pthread_create(&monitor_thread, &attr, monitor_func, NULL) != 0) {
    printf("Failed to Create Monitor Thread");
    return 1;
}
monitor_thread_id = monitor_thread;

// Create and start the valet threads
pthread_t in_valet_threads[inval];
for(int i = 0; i < inval; i++) {
    int* id = malloc(sizeof(int));
    *id = i;
    if (pthread_create(&in_valet_threads[i], &attr, in_valet_func, id) != 0) {
        printf("Failed to Create In Valet Thread #%d\n", *id);
        return 1;
    }
}
in_valet_threads_id = in_valet_threads;
```

04

CAR GENERATION THREAD





No

Not Cancel

Yes

Generate Car

Acquire Queue Lock

Enter CS

Enqueue Car

Exit CS

Post Q_haveCars

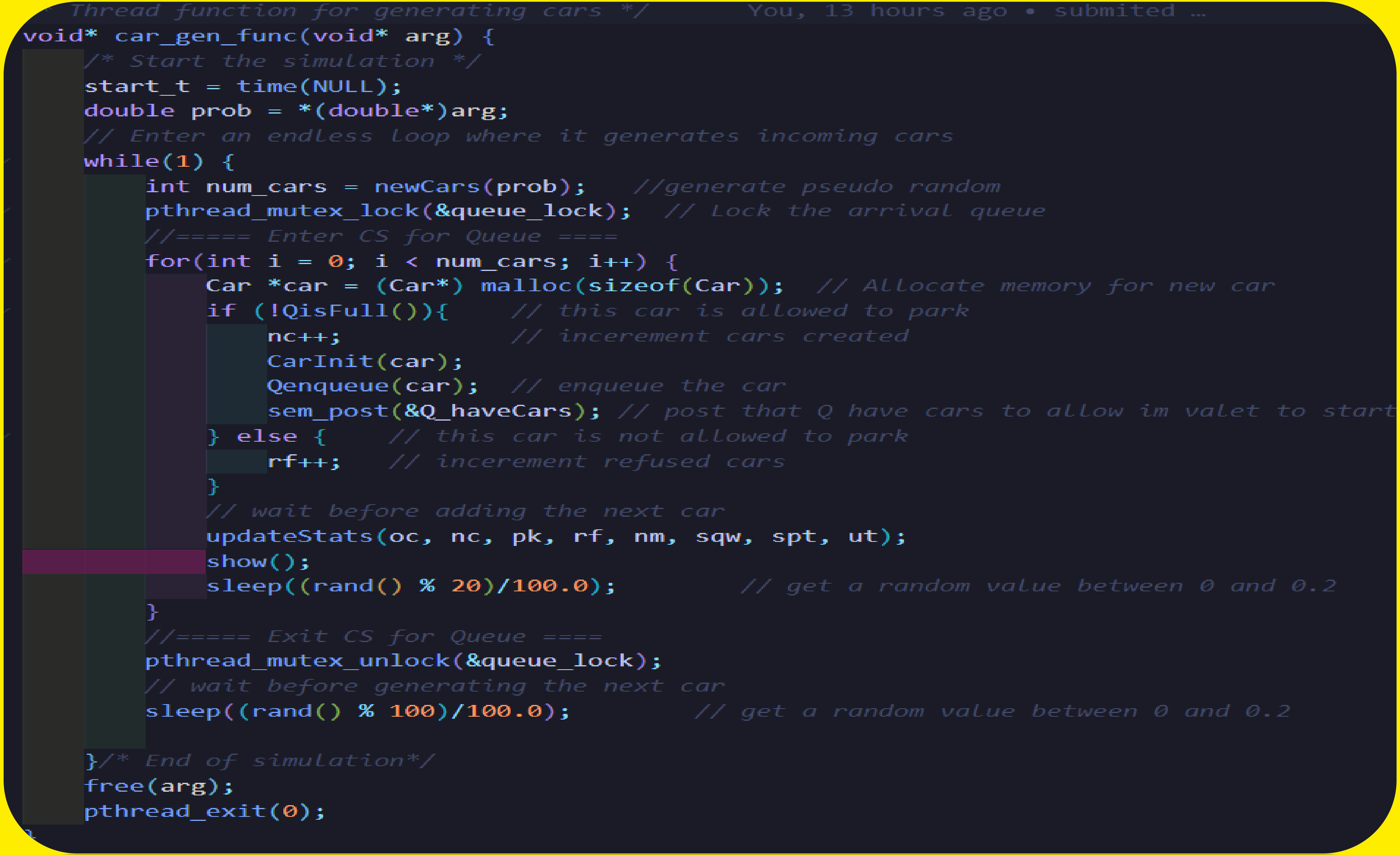
Release Queue Lock

ENDLESS LOOP
GENERATING CARS

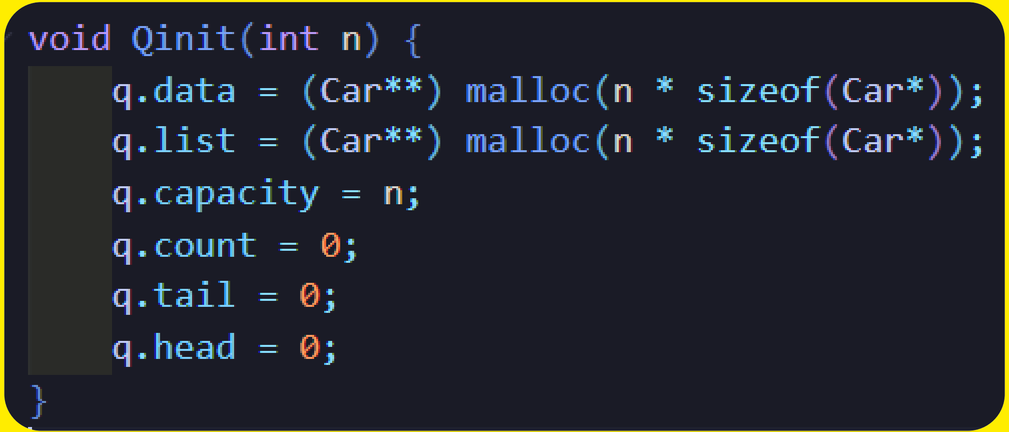
ENTER CS QUEUE

ADDING CARS

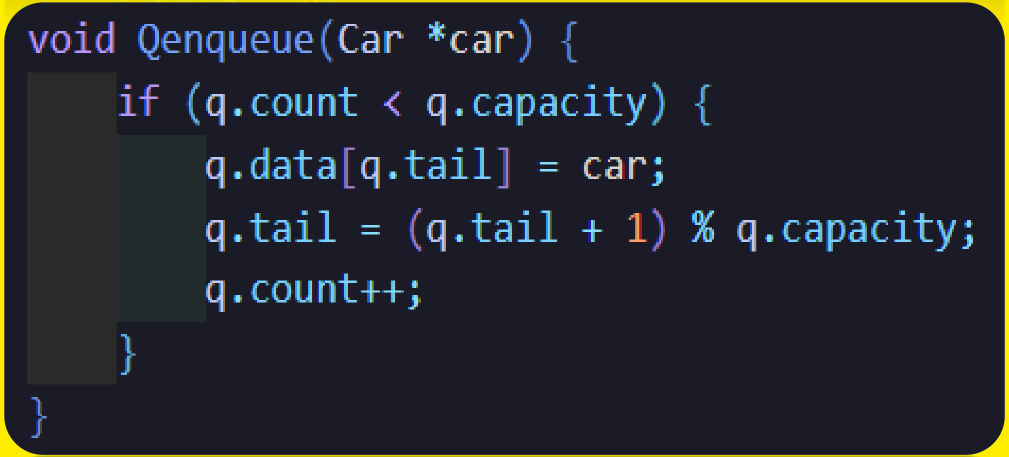
EXIT CS QUEUE



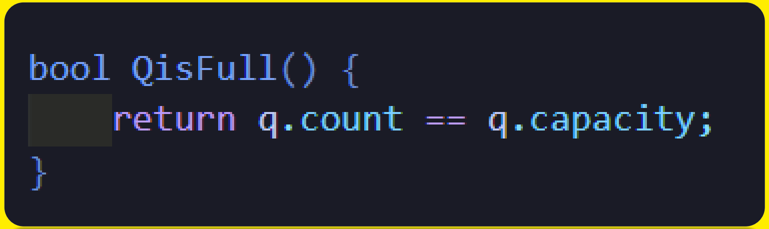
```c
/* Thread function for generating cars */        You, 13 hours ago • submited …
void* car_gen_func(void* arg) {
    /* Start the simulation */
    start_t = time(NULL);
    double prob = *(double*)arg;
    // Enter an endless loop where it generates incoming cars
    while(1) {
        int num_cars = newCars(prob);   //generate pseudo random
        pthread_mutex_lock(&queue_lock);  // Lock the arrival queue
        //===== Enter CS for Queue ====
        for(int i = 0; i < num_cars; i++) {
            Car *car = (Car*) malloc(sizeof(Car));  // Allocate memory for new car
            if (!QisFull()){    // this car is allowed to park
                nc++;           // incerement cars created
                CarInit(car);
                Qenqueue(car);  // enqueue the car
                sem_post(&Q_haveCars); // post that Q have cars to allow im valet to start
            } else {    // this car is not allowed to park
                rf++;   // incerement refused cars
            }
            // wait before adding the next car
            updateStats(oc, nc, pk, rf, nm, sqw, spt, ut);
            show();
            sleep((rand() % 20)/100.0);         // get a random value between 0 and 0.2
        }
        //===== Exit CS for Queue ====
        pthread_mutex_unlock(&queue_lock);
        // wait before generating the next car
        sleep((rand() % 100)/100.0);        // get a random value between 0 and 0.2

    }/* End of simulation*/
    free(arg);
    pthread_exit(0);
}
```



```
void Qinit(int n) {
    q.data = (Car**) malloc(n * sizeof(Car*));
    q.list = (Car**) malloc(n * sizeof(Car*));
    q.capacity = n;
    q.count = 0;
    q.tail = 0;
    q.head = 0;
}
```



```
void Qenqueue(Car *car) {
    if (q.count < q.capacity) {
        q.data[q.tail] = car;
        q.tail = (q.tail + 1) % q.capacity;
        q.count++;
    }
}
```



```
bool QisFull() {
    return q.count == q.capacity;
}
```

05

MONITOR THREAD

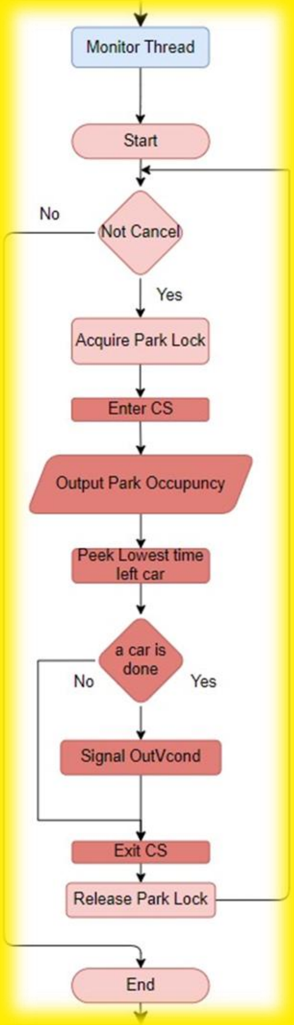
MONITOR THREAD

ENTER CS PARK

UPDATING STATE

EXIT CS PARK

CALCULATING



Monitor Thread

Start

Not Cancel

No

Yes

Acquire Park Lock

Enter CS

Output Park Occupuncy

Peek Lowest time left car

a car is done

No

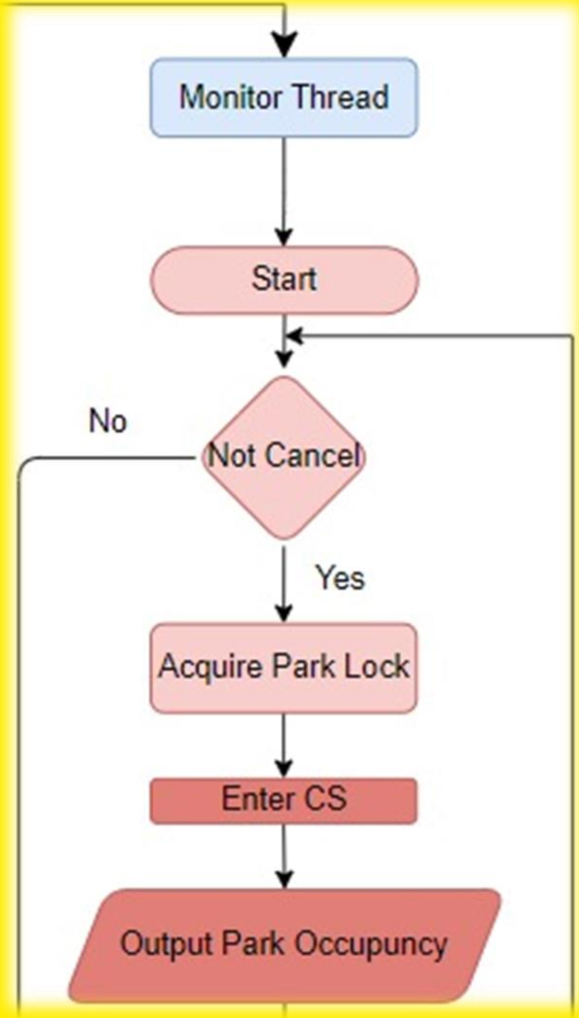
Yes

Signal OutVcond

Exit CS

Release Park Lock

End



Monitor Thread

Start

No

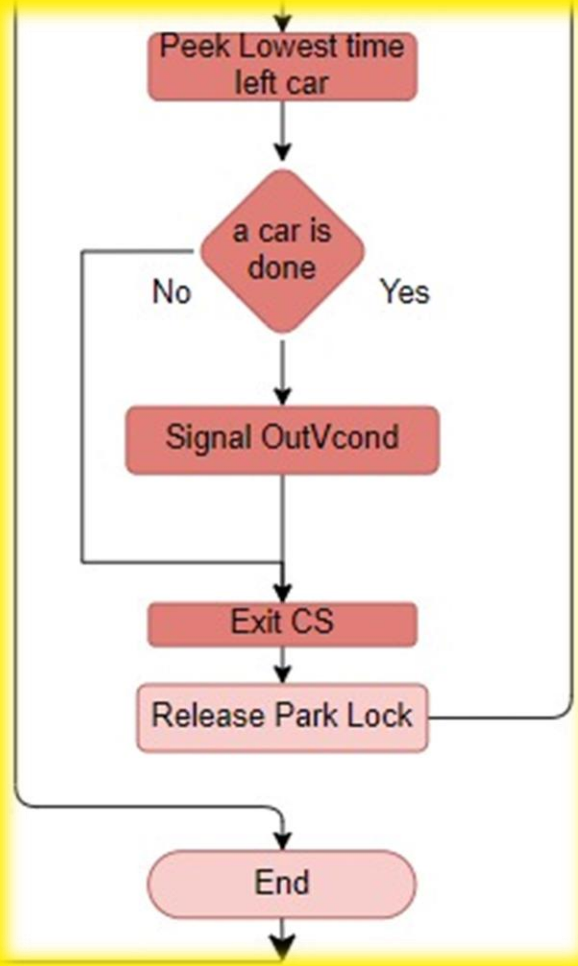
Not Cancel

Yes

Acquire Park Lock

Enter CS

Output Park Occupuncy



Peek Lowest time left car

a car is done

No

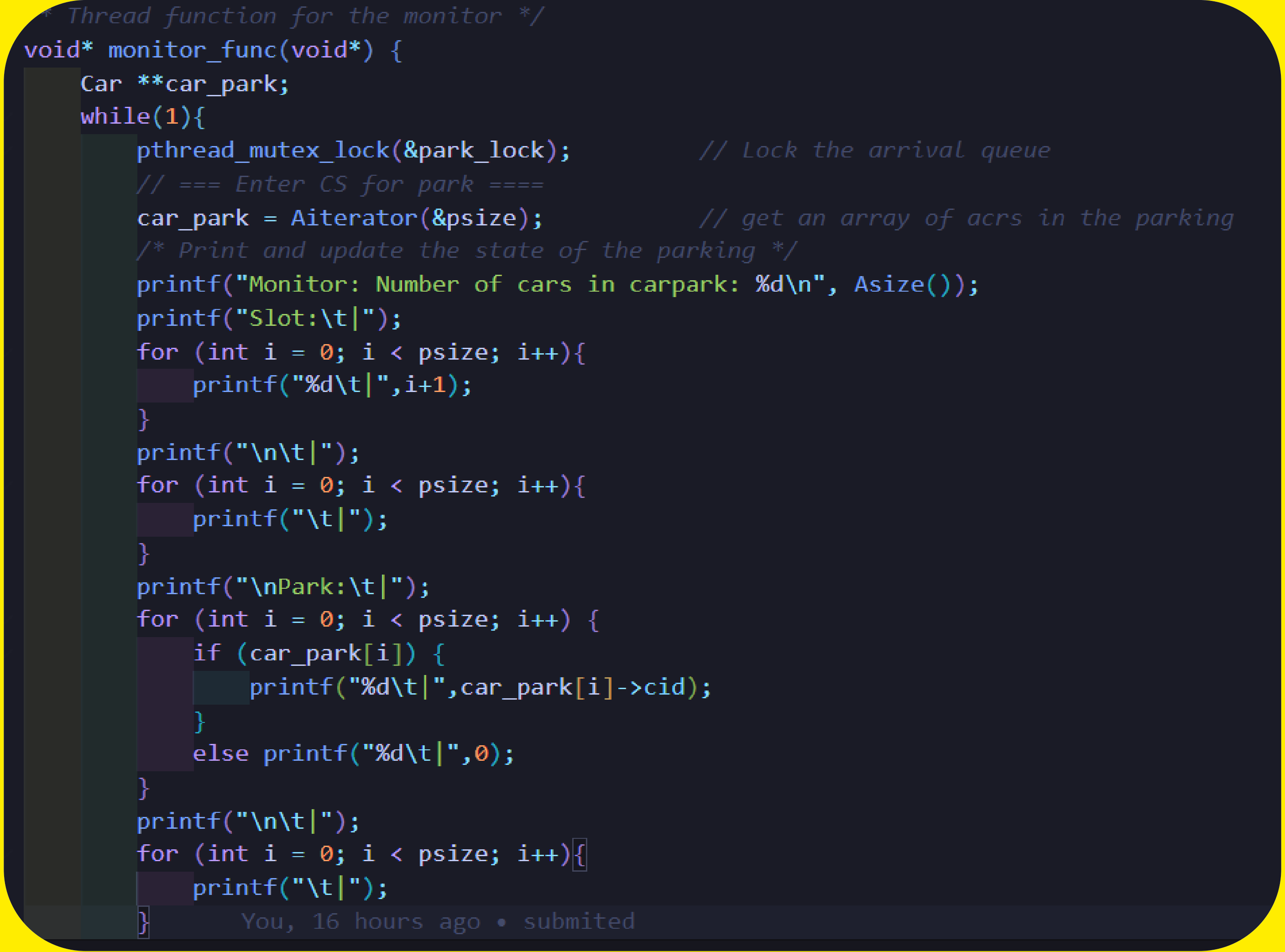
Yes

Signal OutVcond

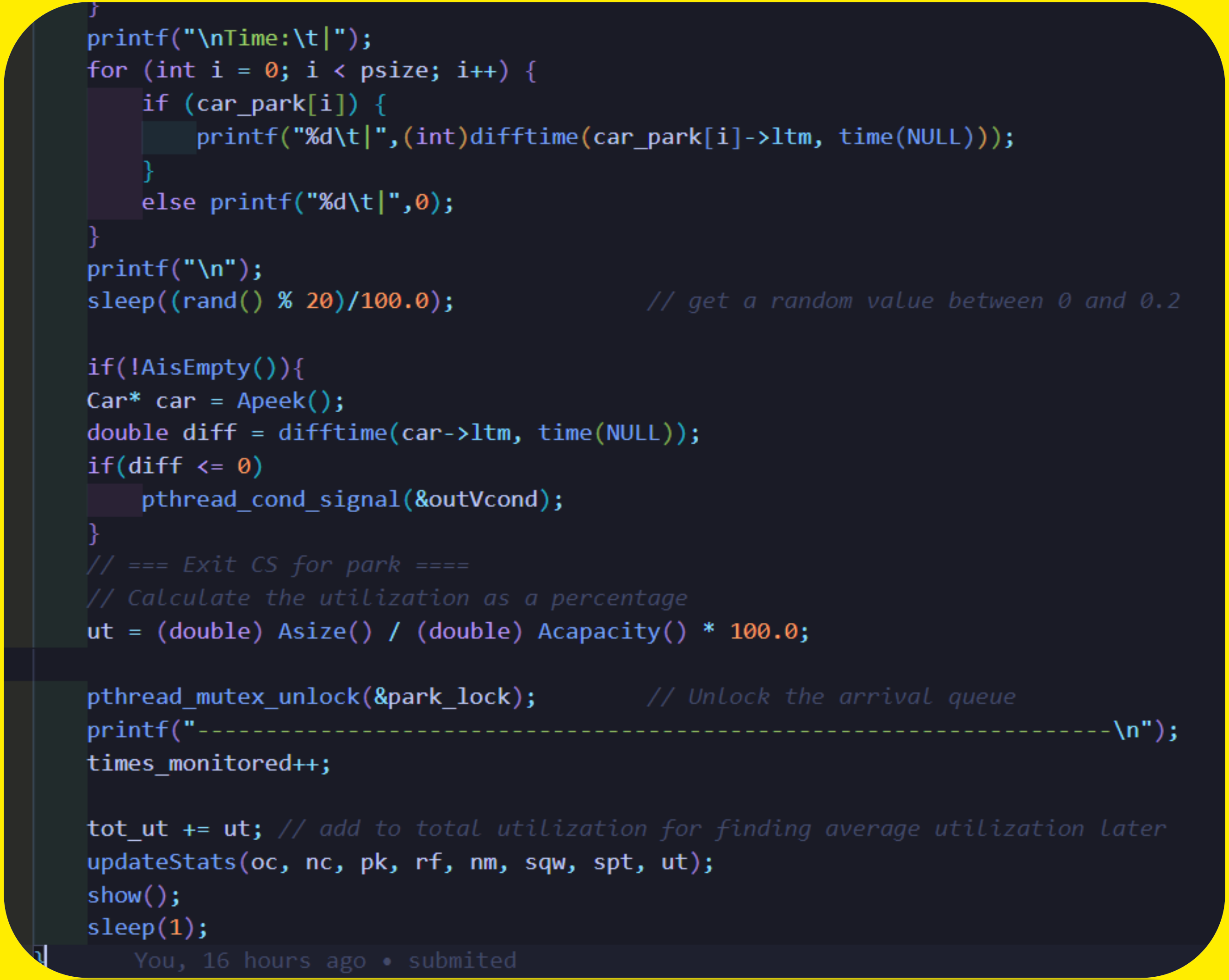
Exit CS

Release Park Lock

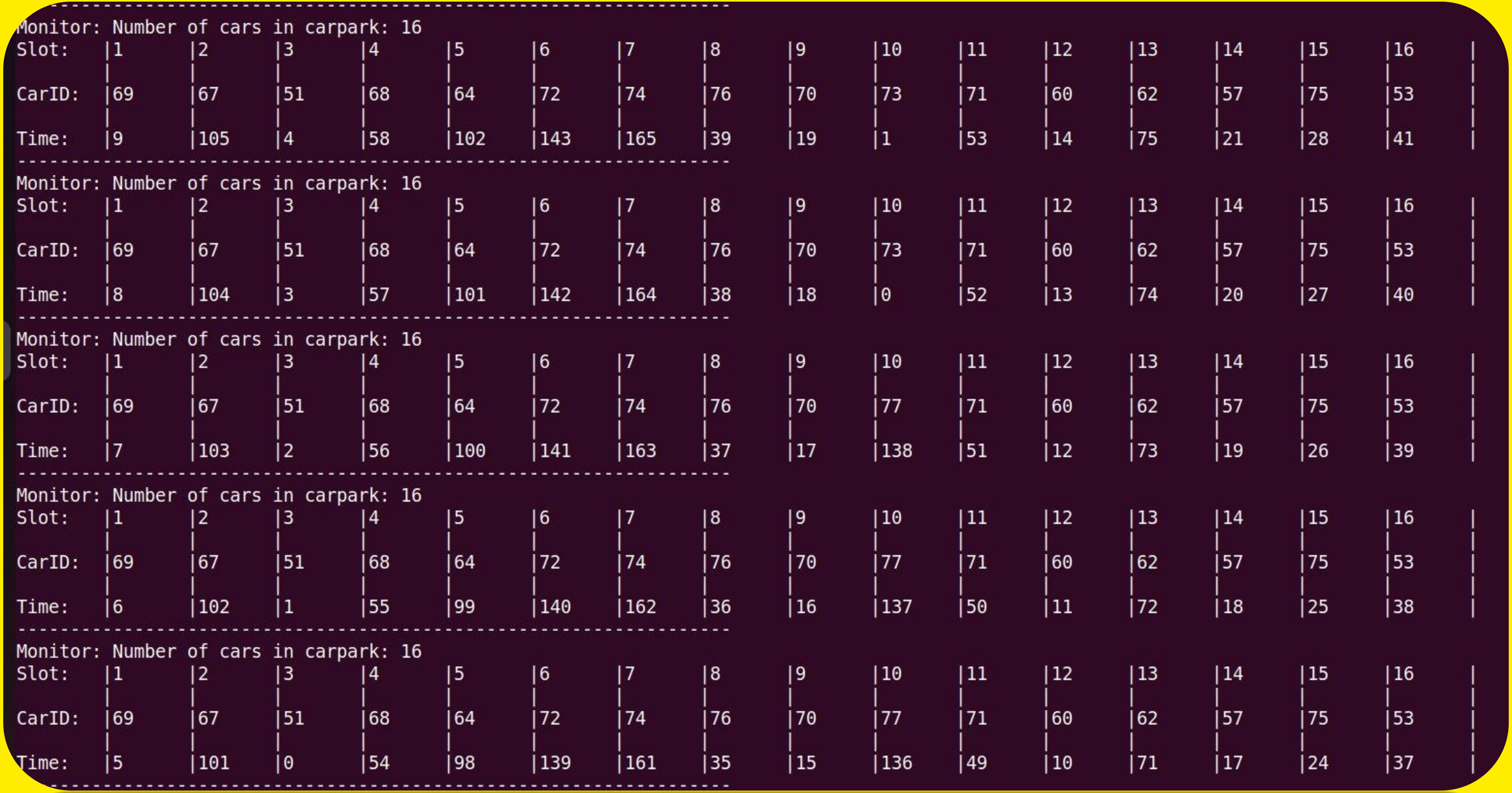
End



```c
* Thread function for the monitor */
void* monitor_func(void*) {
    Car **car_park;
    while(1){
        pthread_mutex_lock(&park_lock);         // Lock the arrival queue
        // === Enter CS for park ====
        car_park = Aiterator(&psize);           // get an array of acrs in the parking
        /* Print and update the state of the parking */
        printf("Monitor: Number of cars in carpark: %d\n", Asize());
        printf("Slot:\t|");
        for (int i = 0; i < psize; i++){
            printf("%d\t|",i+1);
        }
        printf("\n\t|");
        for (int i = 0; i < psize; i++){
            printf("\t|");
        }
        printf("\nPark:\t|");
        for (int i = 0; i < psize; i++) {
            if (car_park[i]) {
                printf("%d\t|",car_park[i]->cid);
            }
            else printf("%d\t|",0);
        }
        printf("\n\t|");
        for (int i = 0; i < psize; i++){
            printf("\t|");
        }      You, 16 hours ago • submited
```



```
}
printf("\nTime:\t|");
for (int i = 0; i < psize; i++) {
    if (car_park[i]) {
        printf("%d\t|",(int)difftime(car_park[i]->ltm, time(NULL)));
    }
    else printf("%d\t|",0);
}
printf("\n");
sleep((rand() % 20)/100.0);              // get a random value between 0 and 0.2

if(!AisEmpty()){
Car* car = Apeek();
double diff = difftime(car->ltm, time(NULL));
if(diff <= 0)
    pthread_cond_signal(&outVcond);
}
// === Exit CS for park ====
// Calculate the utilization as a percentage
ut = (double) Asize() / (double) Acapacity() * 100.0;

pthread_mutex_unlock(&park_lock);        // Unlock the arrival queue
printf("------------------------------------------------------------------\n");
times_monitored++;

tot_ut += ut; // add to total utilization for finding average utilization later
updateStats(oc, nc, pk, rf, nm, sqw, spt, ut);
show();
sleep(1);
    You, 16 hours ago • submited
```

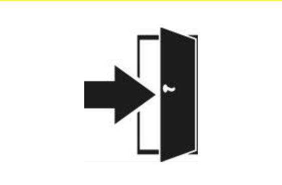


```
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11     |12     |13     |14     |15     |16     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
CarID:  |69     |67     |51     |68     |64     |72     |74     |76     |70     |73     |71     |60     |62     |57     |75     |53     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
Time:   |9      |105    |4      |58     |102    |143    |165    |39     |19     |1      |53     |14     |75     |21     |28     |41     |
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11     |12     |13     |14     |15     |16     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
CarID:  |69     |67     |51     |68     |64     |72     |74     |76     |70     |73     |71     |60     |62     |57     |75     |53     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
Time:   |8      |104    |3      |57     |101    |142    |164    |38     |18     |0      |52     |13     |74     |20     |27     |40     |
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11     |12     |13     |14     |15     |16     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
CarID:  |69     |67     |51     |68     |64     |72     |74     |76     |70     |77     |71     |60     |62     |57     |75     |53     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
Time:   |7      |103    |2      |56     |100    |141    |163    |37     |17     |138    |51     |12     |73     |19     |26     |39     |
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11     |12     |13     |14     |15     |16     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
CarID:  |69     |67     |51     |68     |64     |72     |74     |76     |70     |77     |71     |60     |62     |57     |75     |53     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
Time:   |6      |102    |1      |55     |99     |140    |162    |36     |16     |137    |50     |11     |72     |18     |25     |38     |
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11     |12     |13     |14     |15     |16     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
CarID:  |69     |67     |51     |68     |64     |72     |74     |76     |70     |77     |71     |60     |62     |57     |75     |53     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
Time:   |5      |101    |0      |54     |98     |139    |161    |35     |15     |136    |49     |10     |71     |17     |24     |37     |
-------------------------------------------------------------------
```

06

IN-VALET THREAD

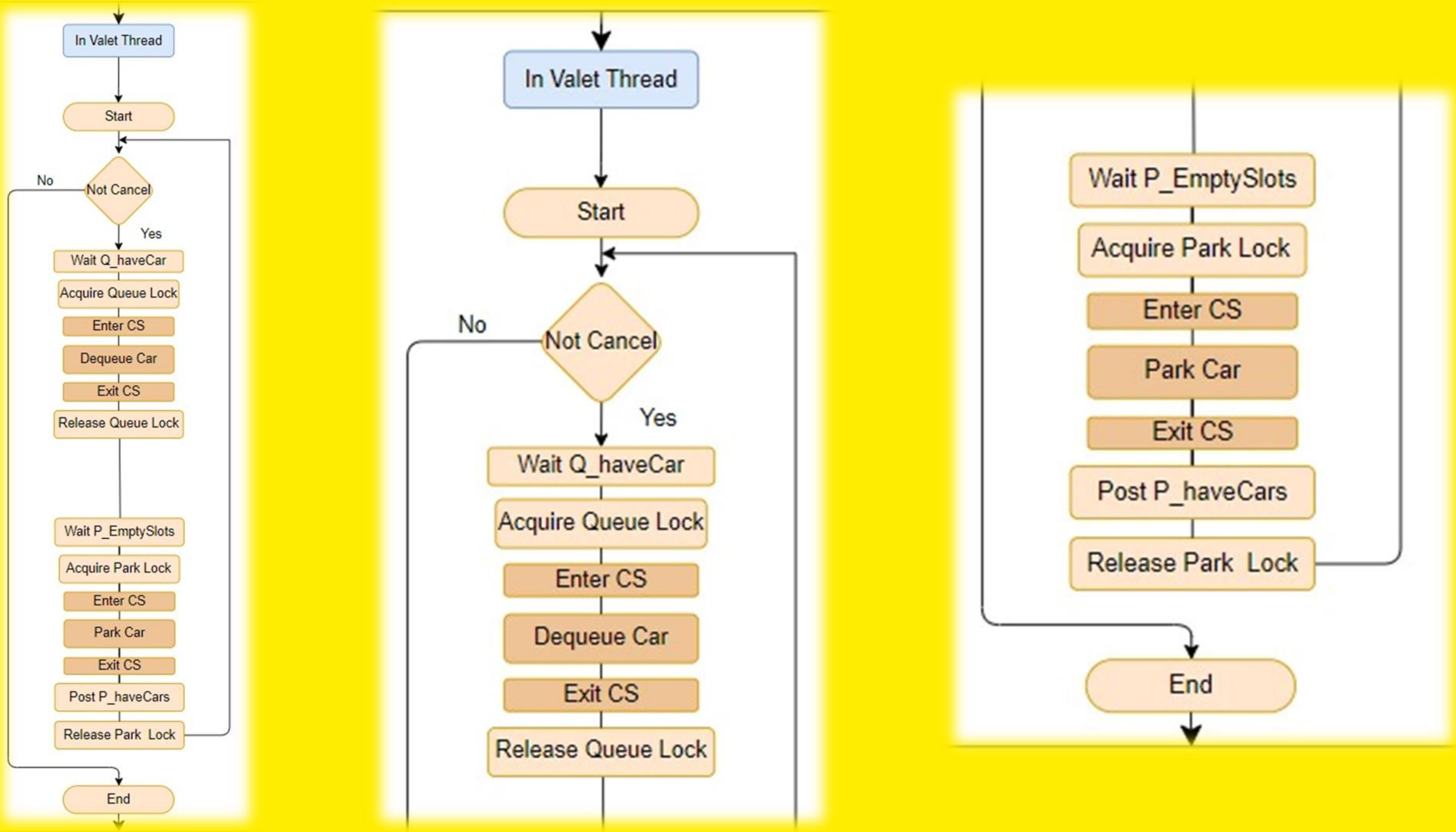
IN-VALET THREAD



ENTRY SECTION

CRITICAL SECTION

EXIT SECTION



In Valet Thread

Start

Not Cancel

No

Yes

Wait Q_haveCar

Acquire Queue Lock

Enter CS

Dequeue Car

Exit CS

Release Queue Lock

Wait P_EmptySlots

Acquire Park Lock

Enter CS

Park Car

Exit CS

Post P_haveCars

Release Park Lock

End

In Valet Thread

Start

No

Not Cancel

Yes

Wait Q_haveCar

Acquire Queue Lock

Enter CS

Dequeue Car

Exit CS

Release Queue Lock

Wait P_EmptySlots

Acquire Park Lock

Enter CS

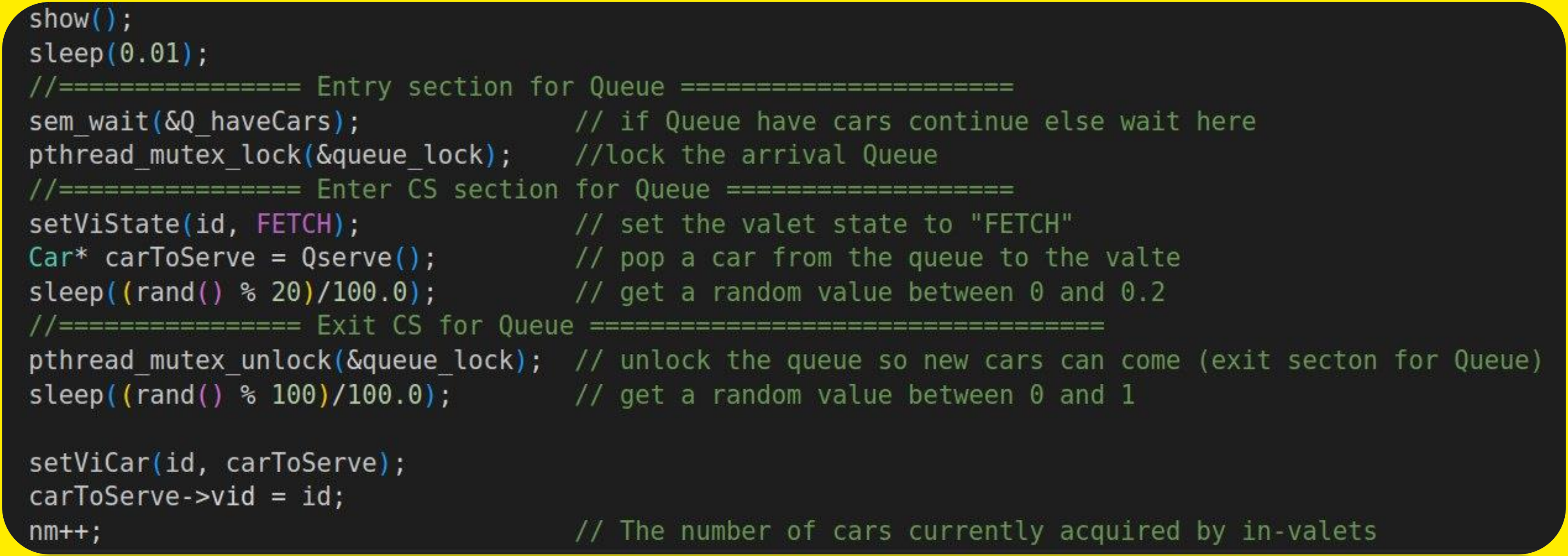
Park Car

Exit CS

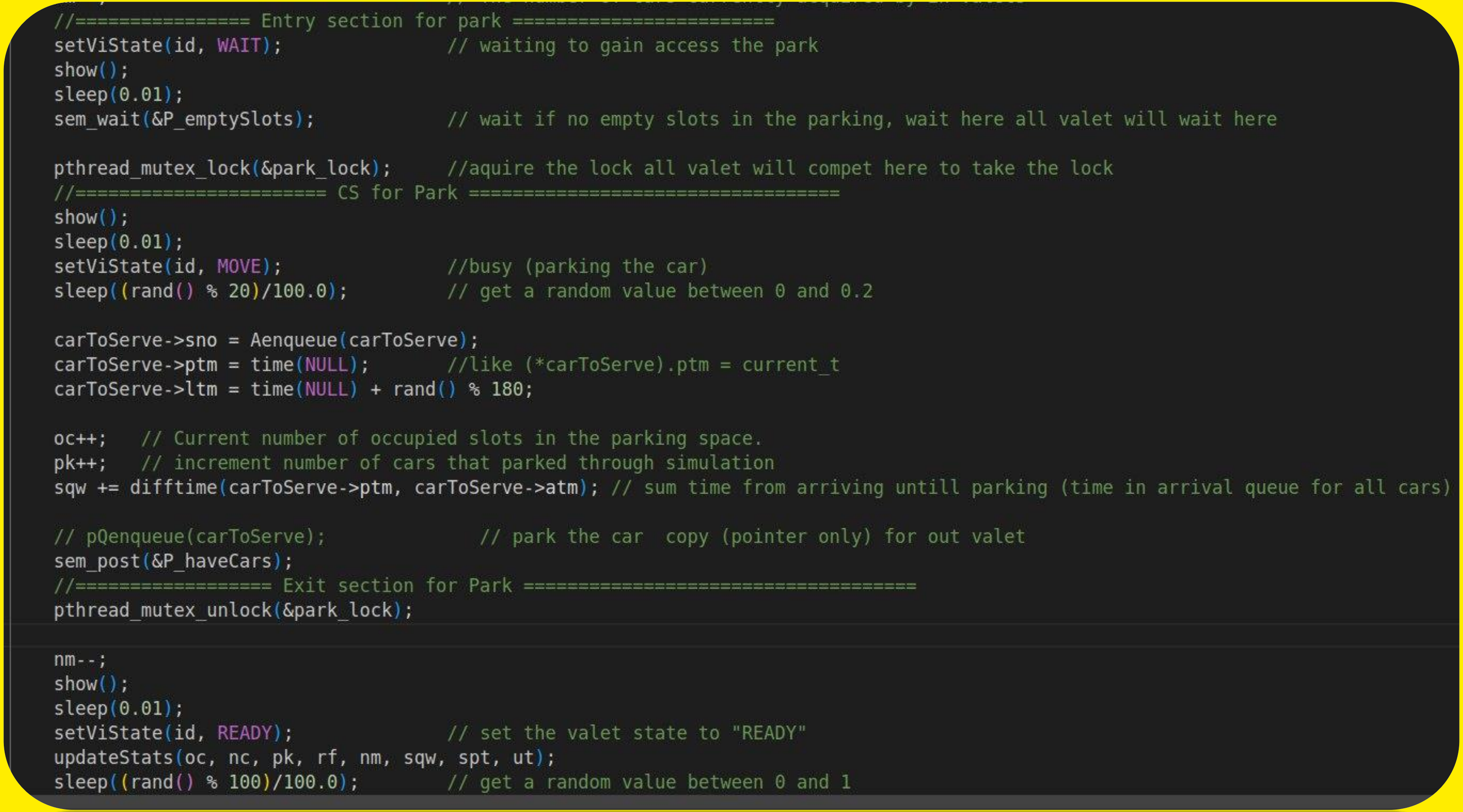
Post P_haveCars

Release Park Lock

End



```
show();
sleep(0.01);
//================ Entry section for Queue ======================
sem_wait(&Q_haveCars);              // if Queue have cars continue else wait here
pthread_mutex_lock(&queue_lock);    //lock the arrival Queue
//================ Enter CS section for Queue ===================
setViState(id, FETCH);              // set the valet state to "FETCH"
Car* carToServe = Qserve();         // pop a car from the queue to the valte
sleep((rand() % 20)/100.0);         // get a random value between 0 and 0.2
//================ Exit CS for Queue ==================================
pthread_mutex_unlock(&queue_lock);  // unlock the queue so new cars can come (exit secton for Queue)
sleep((rand() % 100)/100.0);        // get a random value between 0 and 1

setViCar(id, carToServe);
carToServe->vid = id;
nm++;                               // The number of cars currently acquired by in-valets
```

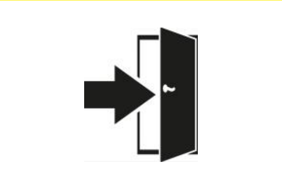


```
    //================ Entry section for park =======================
    setViState(id, WAIT);               // waiting to gain access the park
    show();
    sleep(0.01);
    sem_wait(&P_emptySlots);            // wait if no empty slots in the parking, wait here all valet will wait here

    pthread_mutex_lock(&park_lock);     //aquire the lock all valet will compet here to take the lock
    //====================== CS for Park ==================================
    show();
    sleep(0.01);
    setViState(id, MOVE);               //busy (parking the car)
    sleep((rand() % 20)/100.0);         // get a random value between 0 and 0.2

    carToServe->sno = Aenqueue(carToServe);
    carToServe->ptm = time(NULL);       //like (*carToServe).ptm = current_t
    carToServe->ltm = time(NULL) + rand() % 180;

    oc++;   // Current number of occupied slots in the parking space.
    pk++;   // increment number of cars that parked through simulation
    sqw += difftime(carToServe->ptm, carToServe->atm); // sum time from arriving untill parking (time in arrival queue for all cars)

    // pQenqueue(carToServe);              // park the car  copy (pointer only) for out valet
    sem_post(&P_haveCars);
    //================== Exit section for Park ====================================
    pthread_mutex_unlock(&park_lock);

    nm--;
    show();
    sleep(0.01);
    setViState(id, READY);              // set the valet state to "READY"
    updateStats(oc, nc, pk, rf, nm, sqw, spt, ut);
    sleep((rand() % 100)/100.0);        // get a random value between 0 and 1
```

07

OUT-VALET THREAD

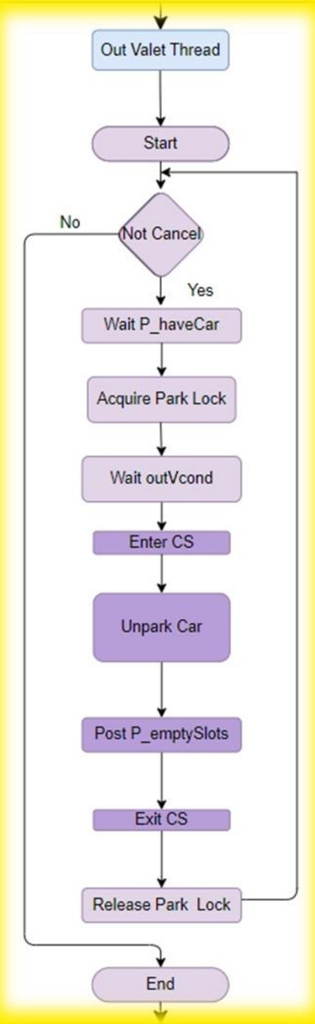
OUT-VALET THREAD

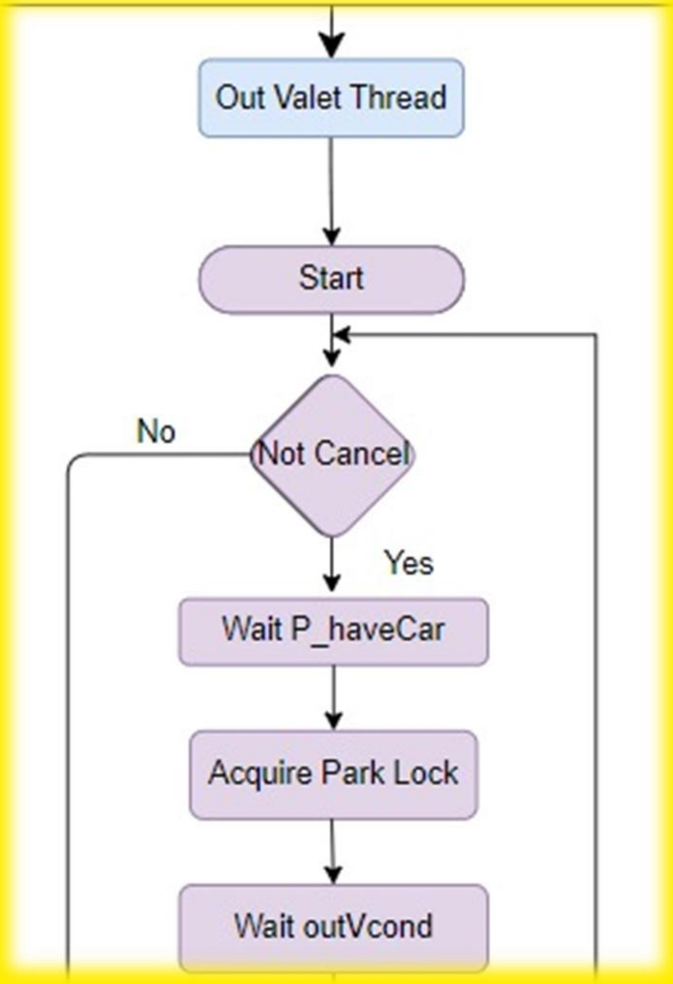


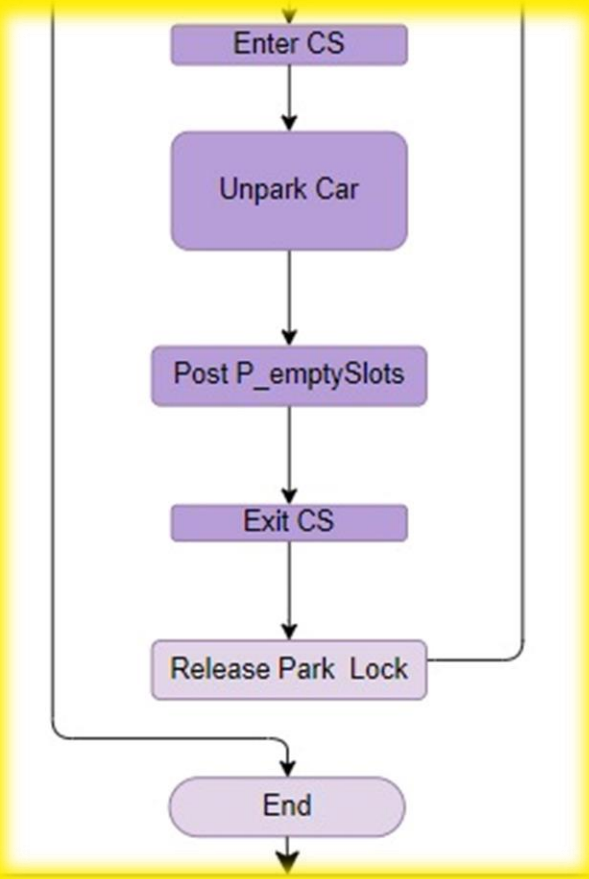
ENTRY SECTION

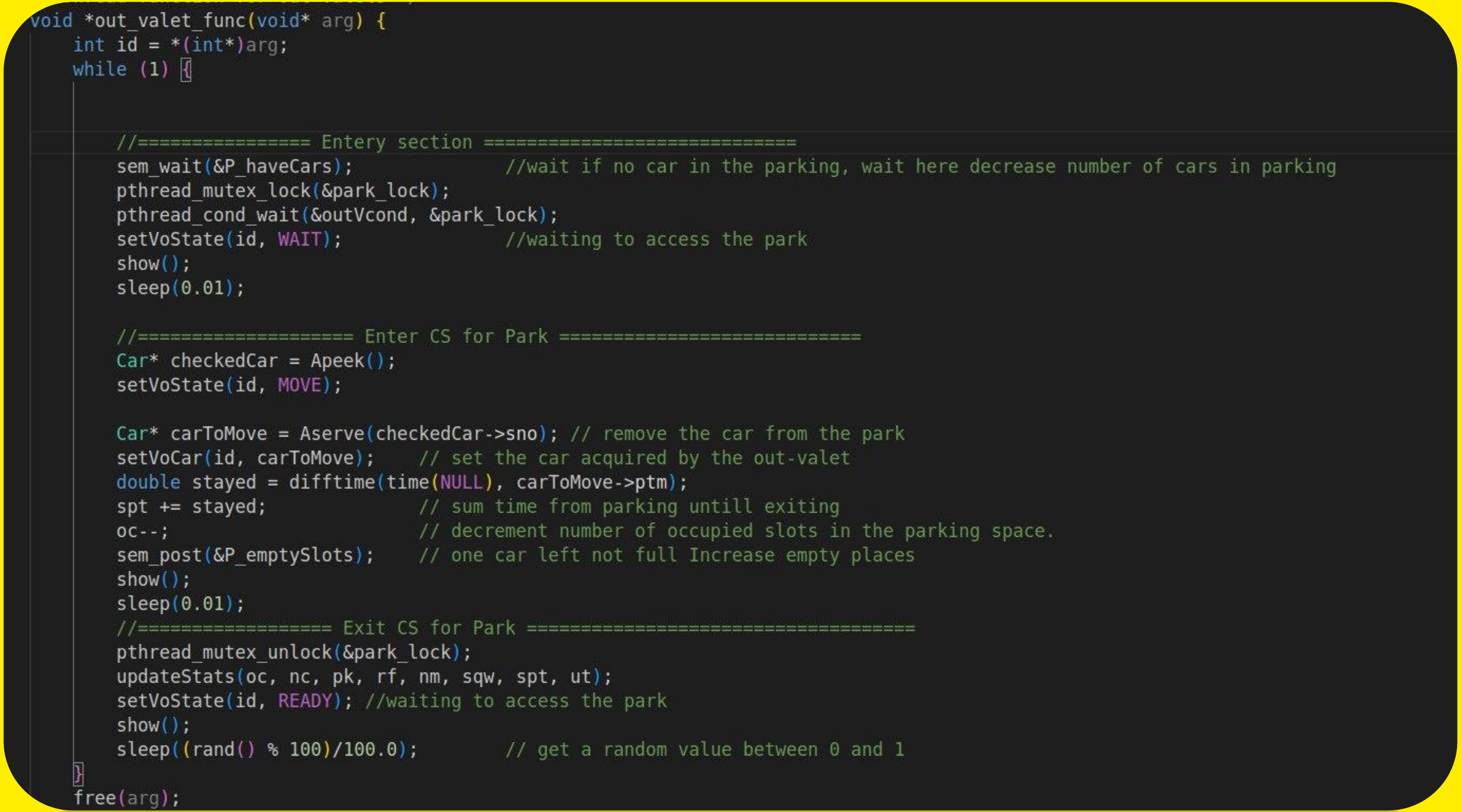
CRITICAL SECTION

EXIT SECTION

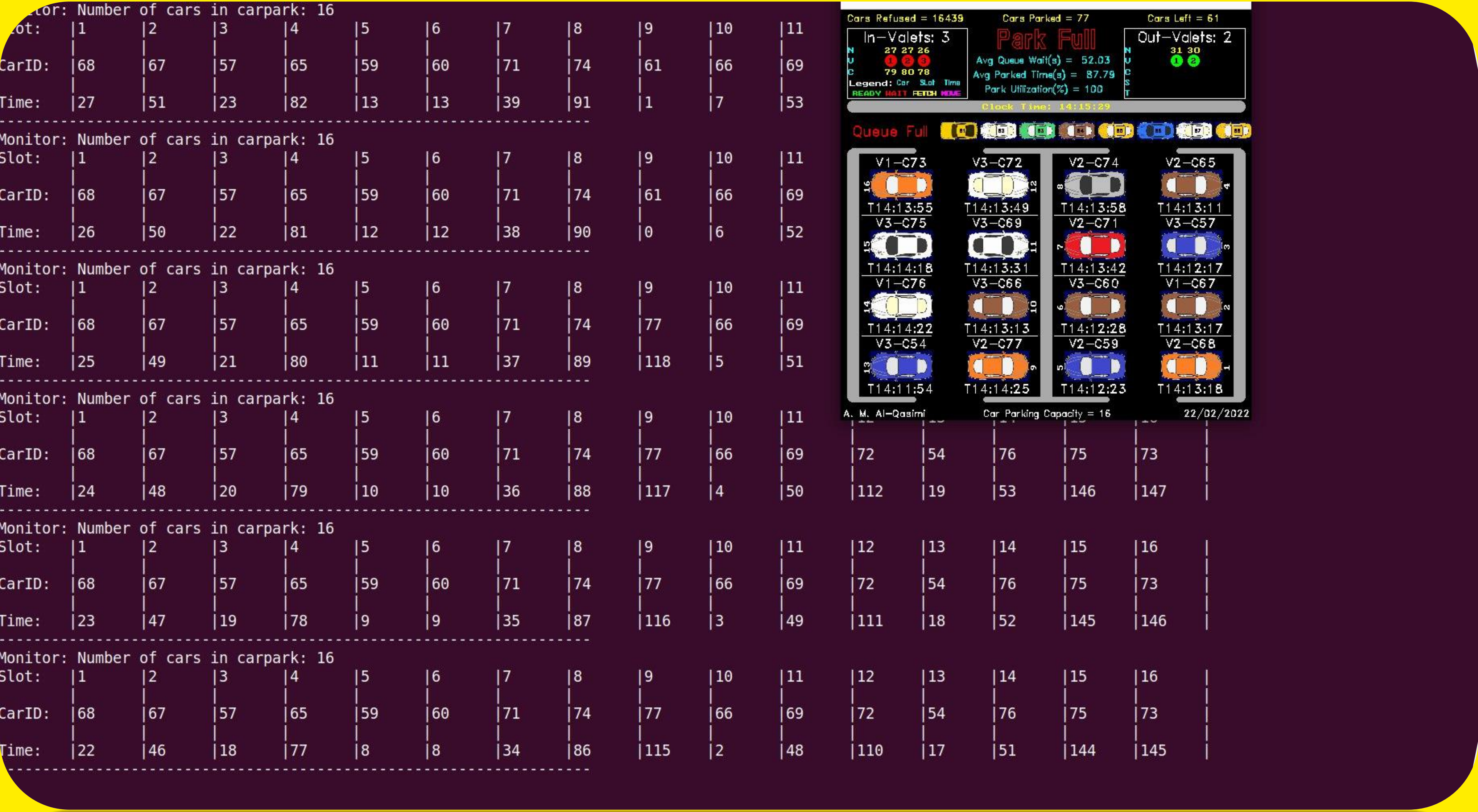

Out Valet Thread
Start
Not Cancel
No
Yes
Wait P_haveCar
Acquire Park Lock
Wait outVcond
Enter CS
Unpark Car
Post P_emptySlots
Exit CS
Release Park Lock
End


Out Valet Thread
Start
No
Not Cancel
Yes
Wait P_haveCar
Acquire Park Lock
Wait outVcond


Enter CS
Unpark Car
Post P_emptySlots
Exit CS
Release Park Lock
End



```
void *out_valet_func(void* arg) {
    int id = *(int*)arg;
    while (1) {


        //================ Entery section ==============================
        sem_wait(&P_haveCars);              //wait if no car in the parking, wait here decrease number of cars in parking
        pthread_mutex_lock(&park_lock);
        pthread_cond_wait(&outVcond, &park_lock);
        setVoState(id, WAIT);               //waiting to access the park
        show();
        sleep(0.01);

        //==================== Enter CS for Park ===========================
        Car* checkedCar = Apeek();
        setVoState(id, MOVE);

        Car* carToMove = Aserve(checkedCar->sno); // remove the car from the park
        setVoCar(id, carToMove);    // set the car acquired by the out-valet
        double stayed = difftime(time(NULL), carToMove->ptm);
        spt += stayed;              // sum time from parking untill exiting
        oc--;                       // decrement number of occupied slots in the parking space.
        sem_post(&P_emptySlots);    // one car left not full Increase empty places
        show();
        sleep(0.01);
        //================== Exit CS for Park ====================================
        pthread_mutex_unlock(&park_lock);
        updateStats(oc, nc, pk, rf, nm, sqw, spt, ut);
        setVoState(id, READY); //waiting to access the park
        show();
        sleep((rand() % 100)/100.0);        // get a random value between 0 and 1
    }
    free(arg);
```



```
tor: Number of cars in carpark: 16
lot:    |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11
        |       |       |       |       |       |       |       |       |       |       |
CarID:  |68     |67     |57     |65     |59     |60     |71     |74     |61     |66     |69
        |       |       |       |       |       |       |       |       |       |       |
Time:   |27     |51     |23     |82     |13     |13     |39     |91     |1      |7      |53
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11
        |       |       |       |       |       |       |       |       |       |       |
CarID:  |68     |67     |57     |65     |59     |60     |71     |74     |61     |66     |69
        |       |       |       |       |       |       |       |       |       |       |
Time:   |26     |50     |22     |81     |12     |12     |38     |90     |0      |6      |52
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11
        |       |       |       |       |       |       |       |       |       |       |
CarID:  |68     |67     |57     |65     |59     |60     |71     |74     |77     |66     |69
        |       |       |       |       |       |       |       |       |       |       |
Time:   |25     |49     |21     |80     |11     |11     |37     |89     |118    |5      |51
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
CarID:  |68     |67     |57     |65     |59     |60     |71     |74     |77     |66     |69     |72     |54     |76     |75     |73     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
Time:   |24     |48     |20     |79     |10     |10     |36     |88     |117    |4      |50     |112    |19     |53     |146    |147    |
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11     |12     |13     |14     |15     |16     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
CarID:  |68     |67     |57     |65     |59     |60     |71     |74     |77     |66     |69     |72     |54     |76     |75     |73     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
Time:   |23     |47     |19     |78     |9      |9      |35     |87     |116    |3      |49     |111    |18     |52     |145    |146    |
-------------------------------------------------------------------
Monitor: Number of cars in carpark: 16
Slot:   |1      |2      |3      |4      |5      |6      |7      |8      |9      |10     |11     |12     |13     |14     |15     |16     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
CarID:  |68     |67     |57     |65     |59     |60     |71     |74     |77     |66     |69     |72     |54     |76     |75     |73     |
        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
Time:   |22     |46     |18     |77     |8      |8      |34     |86     |115    |2      |48     |110    |17     |51     |144    |145    |
-------------------------------------------------------------------
```

Cars Refused = 16439   Cars Parked = 77   Cars Left = 61

In-Valets: 3
N 27 27 26
V 1 2 3
C 79 80 78
Legend: Car Slot Time
READY WAIT FETCH MOVE

Park Full
Avg Queue Wait(s) = 52.03
Avg Parked Time(s) = 87.79
Park Utilization(%) = 100

Out-Valets: 2
N 31 30
V 1 2
C
S
T

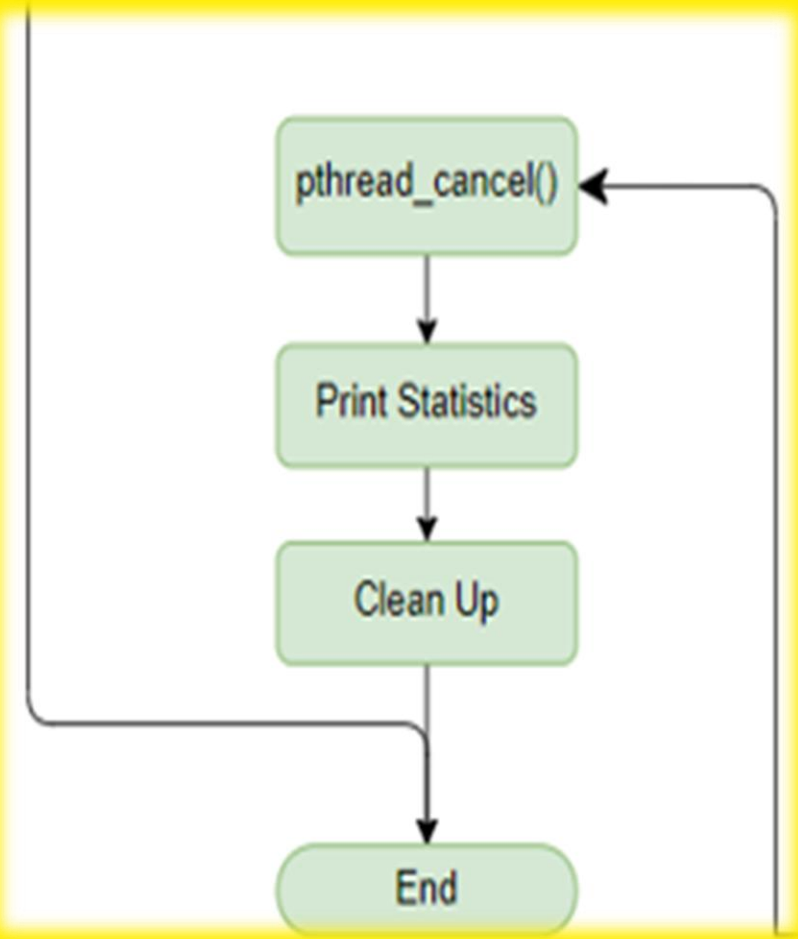
Clock Time: 14:15:29

Queue Full

A. M. Al-Qasimi   Car Parking Capacity = 16   22/02/2022

08

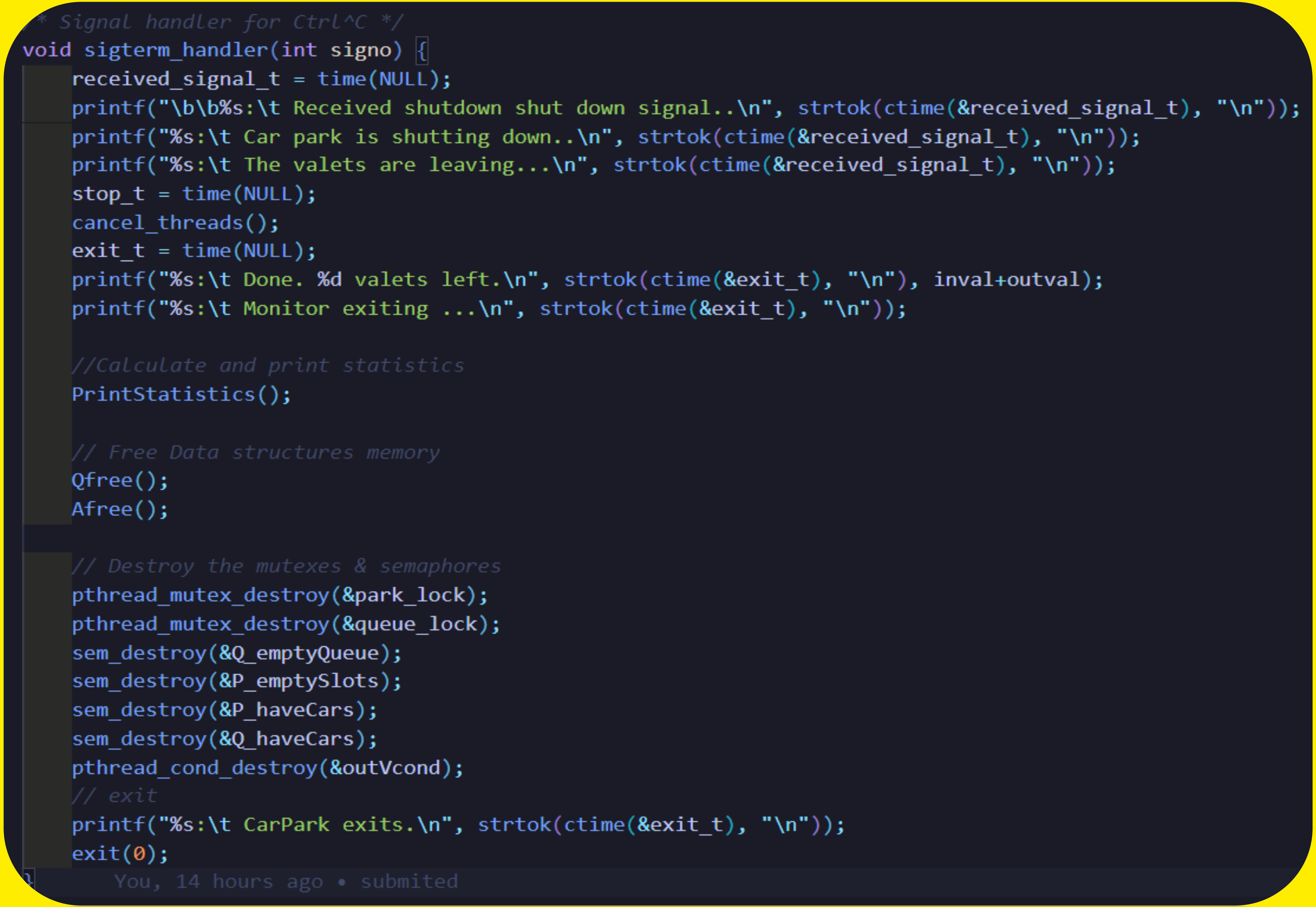
Exiting



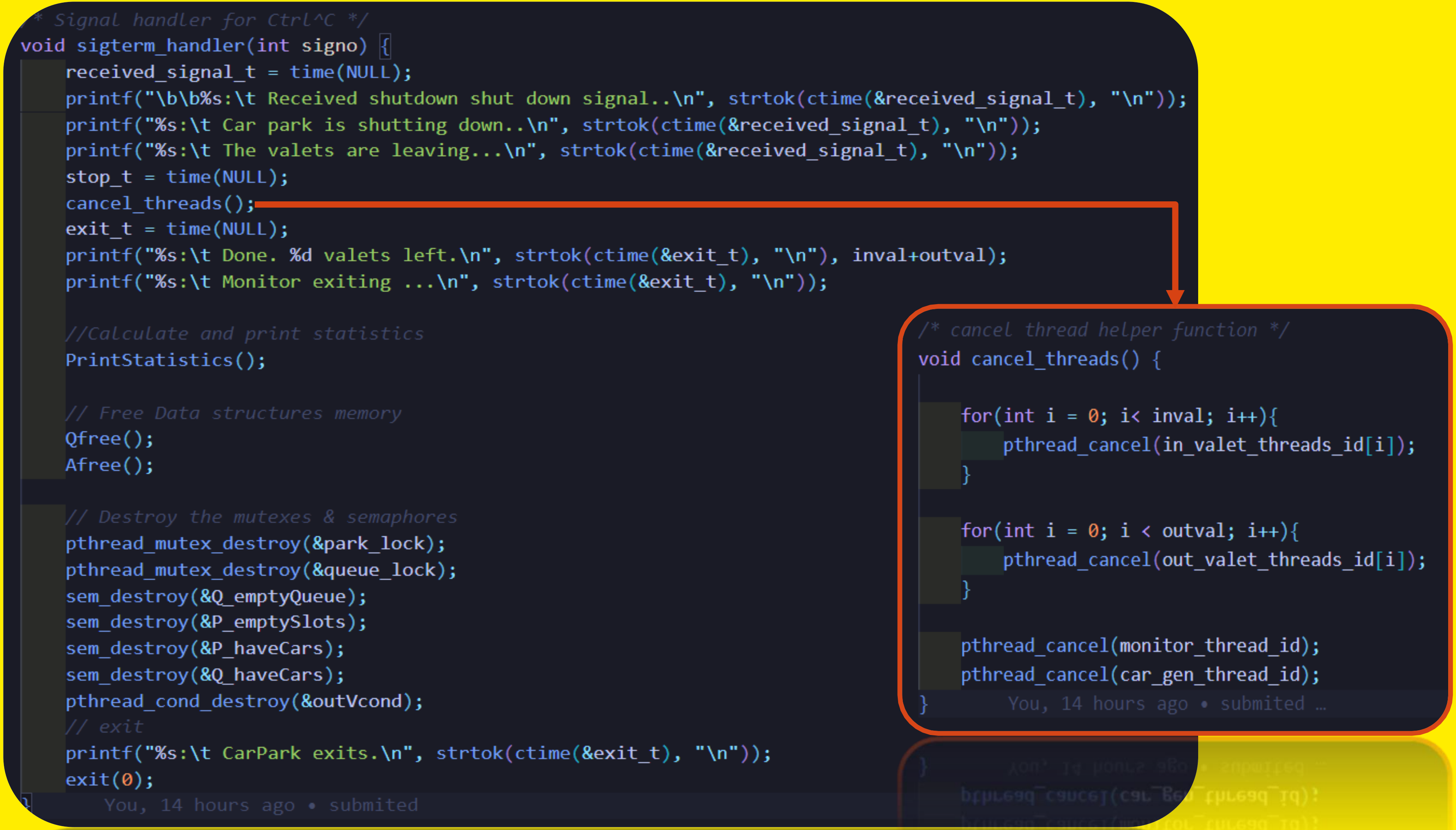
STOP SIMULATION

PRINT STATISTICS

CLEAN UP

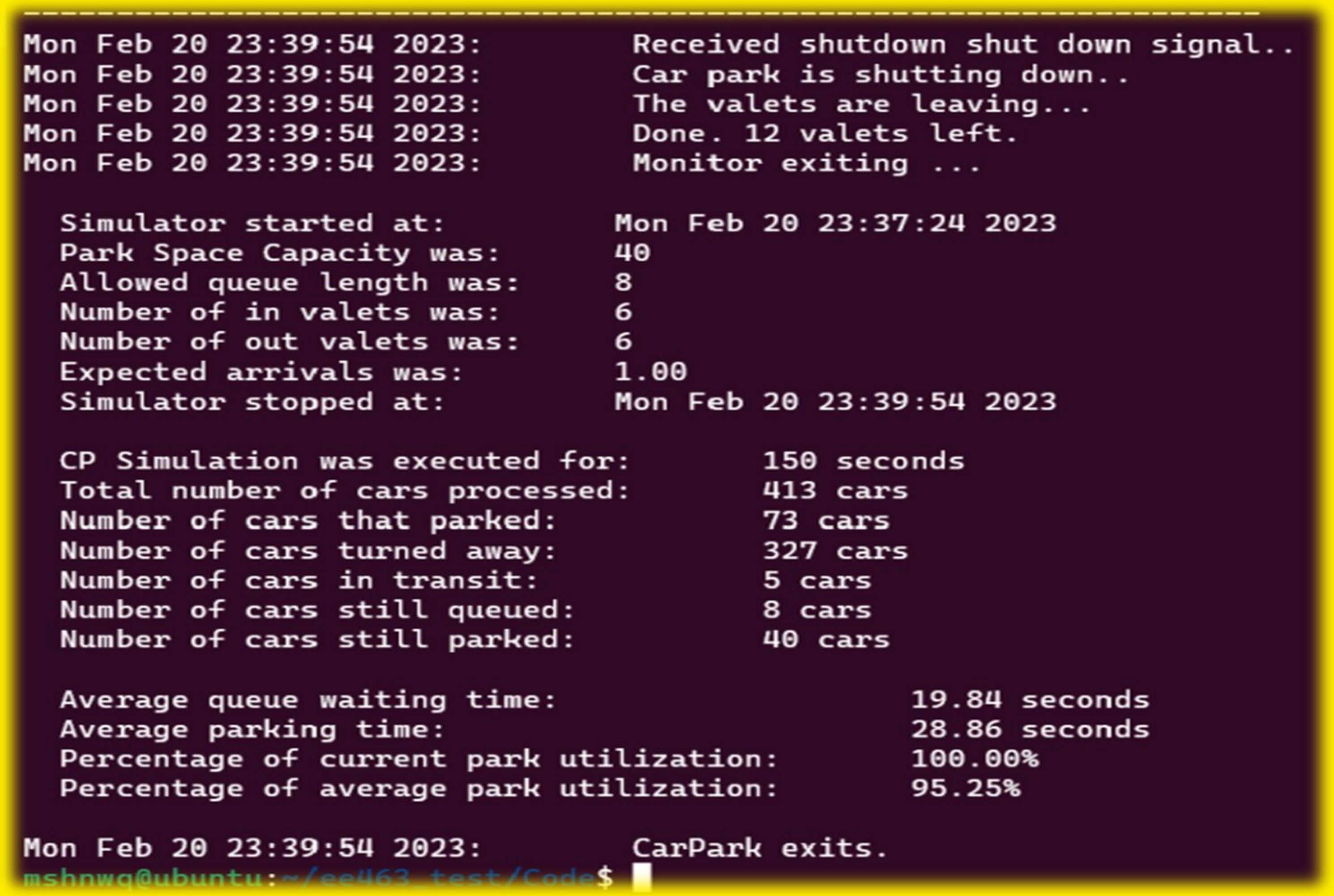


```
/* Signal handler for Ctrl^C */
void sigterm_handler(int signo) {
    received_signal_t = time(NULL);
    printf("\b\b%s:\t Received shutdown shut down signal..\n", strtok(ctime(&received_signal_t), "\n"));
    printf("%s:\t Car park is shutting down..\n", strtok(ctime(&received_signal_t), "\n"));
    printf("%s:\t The valets are leaving...\n", strtok(ctime(&received_signal_t), "\n"));
    stop_t = time(NULL);
    cancel_threads();
    exit_t = time(NULL);
    printf("%s:\t Done. %d valets left.\n", strtok(ctime(&exit_t), "\n"), inval+outval);
    printf("%s:\t Monitor exiting ...\n", strtok(ctime(&exit_t), "\n"));

    //Calculate and print statistics
    PrintStatistics();

    // Free Data structures memory
    Qfree();
    Afree();

    // Destroy the mutexes & semaphores
    pthread_mutex_destroy(&park_lock);
    pthread_mutex_destroy(&queue_lock);
    sem_destroy(&Q_emptyQueue);
    sem_destroy(&P_emptySlots);
    sem_destroy(&P_haveCars);
    sem_destroy(&Q_haveCars);
    pthread_cond_destroy(&outVcond);
    // exit
    printf("%s:\t CarPark exits.\n", strtok(ctime(&exit_t), "\n"));
    exit(0);
}      You, 14 hours ago • submited
```



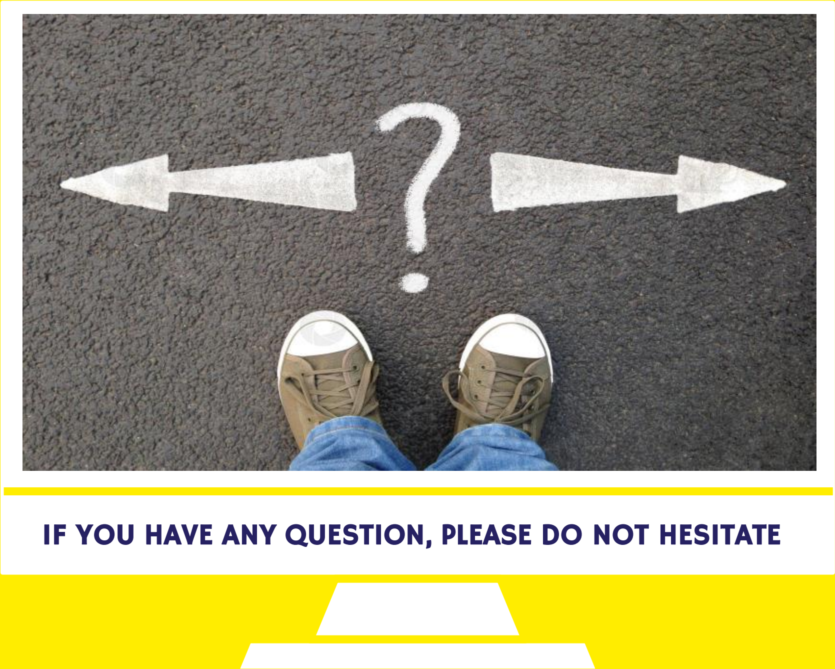
```
/* Signal handler for Ctrl^C */
void sigterm_handler(int signo) {
    received_signal_t = time(NULL);
    printf("\b\b%s:\t Received shutdown shut down signal..\n", strtok(ctime(&received_signal_t), "\n"));
    printf("%s:\t Car park is shutting down..\n", strtok(ctime(&received_signal_t), "\n"));
    printf("%s:\t The valets are leaving...\n", strtok(ctime(&received_signal_t), "\n"));
    stop_t = time(NULL);
    cancel_threads();
    exit_t = time(NULL);
    printf("%s:\t Done. %d valets left.\n", strtok(ctime(&exit_t), "\n"), inval+outval);
    printf("%s:\t Monitor exiting ...\n", strtok(ctime(&exit_t), "\n"));

    //Calculate and print statistics
    PrintStatistics();

    // Free Data structures memory
    Qfree();
    Afree();

    // Destroy the mutexes & semaphores
    pthread_mutex_destroy(&park_lock);
    pthread_mutex_destroy(&queue_lock);
    sem_destroy(&Q_emptyQueue);
    sem_destroy(&P_emptySlots);
    sem_destroy(&P_haveCars);
    sem_destroy(&Q_haveCars);
    pthread_cond_destroy(&outVcond);
    // exit
    printf("%s:\t CarPark exits.\n", strtok(ctime(&exit_t), "\n"));
    exit(0);
}       You, 14 hours ago • submited
```

```
/* cancel thread helper function */
void cancel_threads() {

    for(int i = 0; i< inval; i++){
        pthread_cancel(in_valet_threads_id[i]);
    }

    for(int i = 0; i < outval; i++){
        pthread_cancel(out_valet_threads_id[i]);
    }

    pthread_cancel(monitor_thread_id);
    pthread_cancel(car_gen_thread_id);
}       You, 14 hours ago • submited …
```



```
-------------------------------------------------------------------
Mon Feb 20 23:39:54 2023:        Received shutdown shut down signal..
Mon Feb 20 23:39:54 2023:        Car park is shutting down..
Mon Feb 20 23:39:54 2023:        The valets are leaving...
Mon Feb 20 23:39:54 2023:        Done. 12 valets left.
Mon Feb 20 23:39:54 2023:        Monitor exiting ...

  Simulator started at:         Mon Feb 20 23:37:24 2023
  Park Space Capacity was:      40
  Allowed queue length was:     8
  Number of in valets was:      6
  Number of out valets was:     6
  Expected arrivals was:        1.00
  Simulator stopped at:         Mon Feb 20 23:39:54 2023

  CP Simulation was executed for:       150 seconds
  Total number of cars processed:       413 cars
  Number of cars that parked:           73 cars
  Number of cars turned away:           327 cars
  Number of cars in transit:            5 cars
  Number of cars still queued:          8 cars
  Number of cars still parked:          40 cars

  Average queue waiting time:                   19.84 seconds
  Average parking time:                         28.86 seconds
  Percentage of current park utilization:       100.00%
  Percentage of average park utilization:       95.25%

Mon Feb 20 23:39:54 2023:        CarPark exits.
mshnwq@ubuntu:~/ee463_test/Code$ 
```

THANKS!



IF YOU HAVE ANY QUESTION, PLEASE DO NOT HESITATE