# ECEN/CSCE 689 PROJECT

# MODEL CHECKING OF TRAFFIC CONTROL SYSTEM WITH VEHICLE SIMULATION

**TEAM – 10**

**V-GROUP**

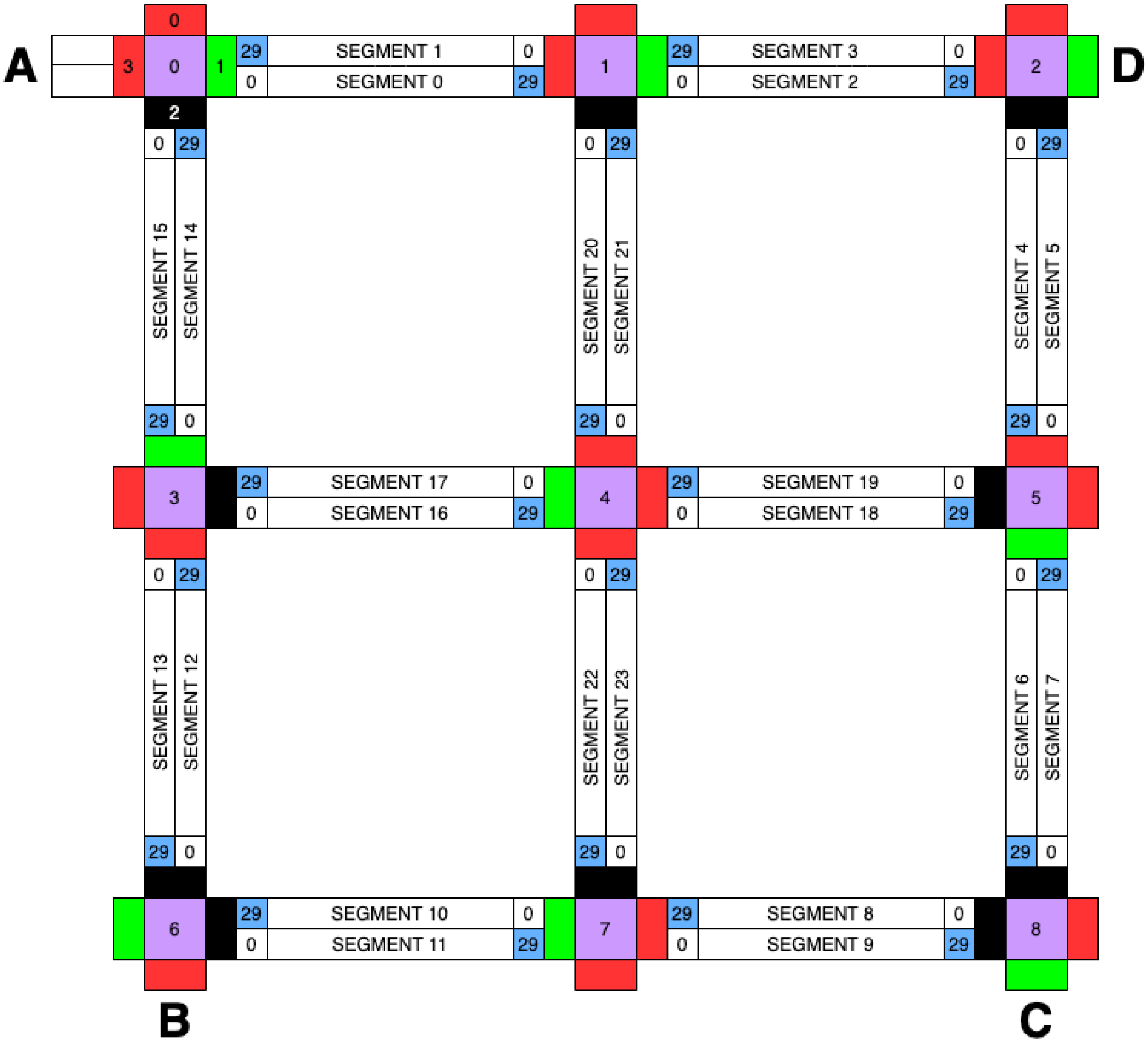
SUBMITTED BY

**MEGHANA JAYSING AMUP (632000331)**

**RISHYA SANKAR KUMARAN (931002051)**

# Index

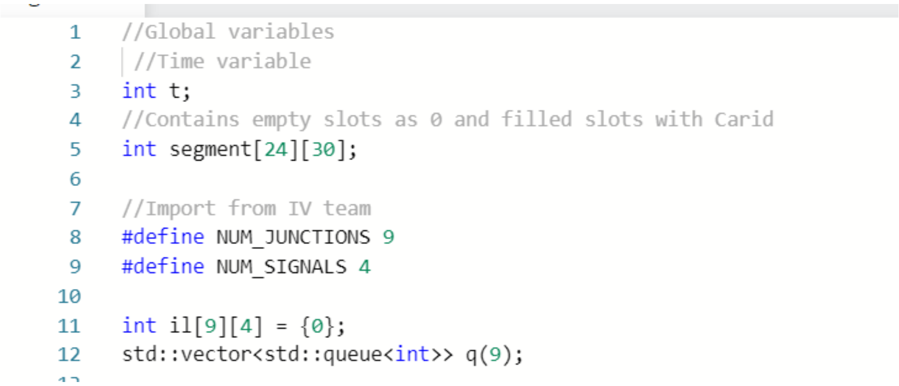
# ARCHITECTURE



The road system described in the problem statement is visualized as shown in the Figure above. The road system has four nodes A,B,C and D. A is the start point for all the cars. There are 9 intersections indicated by the purple boxes in the figure. There are 2 segments between any 2 intersections. Since there are 12 lines between any 2 intersections, and each line is divided into 2 segments, there are a total of 24 segments in the road system. Each intersection has 4 signal lights for each way. The lights can be black, red or green as shown in the figure at any intersection. The black light indicates that the light is not used/no road which follows that light. Red and green lights indicate whether the cars from road opposite to it should stop or can go. For example, at intersection 0, the green light box with index 1 indicates whether cars coming from A can move or should stop. Similarly, the red light box with index 3 indicates the cars from segment 1 should go or stop and the red light box with index 0 indicates whether the car from segment 14 can go or stop. The road in between 2 intersections (purple boxes) are divided into two segments with 30 small segments numbered from 0-29. The index in every segment is always incrementing. So we can easily say the direction of the cars in that particular segment. For example, between intersection 0 and 1, there are 2 segments – segment 0 and segment 1. It is observed that the indices of segment 0 is from 0 to 29 and the indices of segment 1 is from 29 to 0. So segment 0 is for cars moving from intersection 0 to 1 and segment 1 is for cars moving from intersection 1 to 0. This will give us a sense of direction of cars and direction in which the car is supposed to go in that particular segment. Based on these architectural features of the system, the design of system in software code is implemented by the following mapping. We have implemented and tested our design in C++.

## Modeling and Mapping:

*Global variables :*



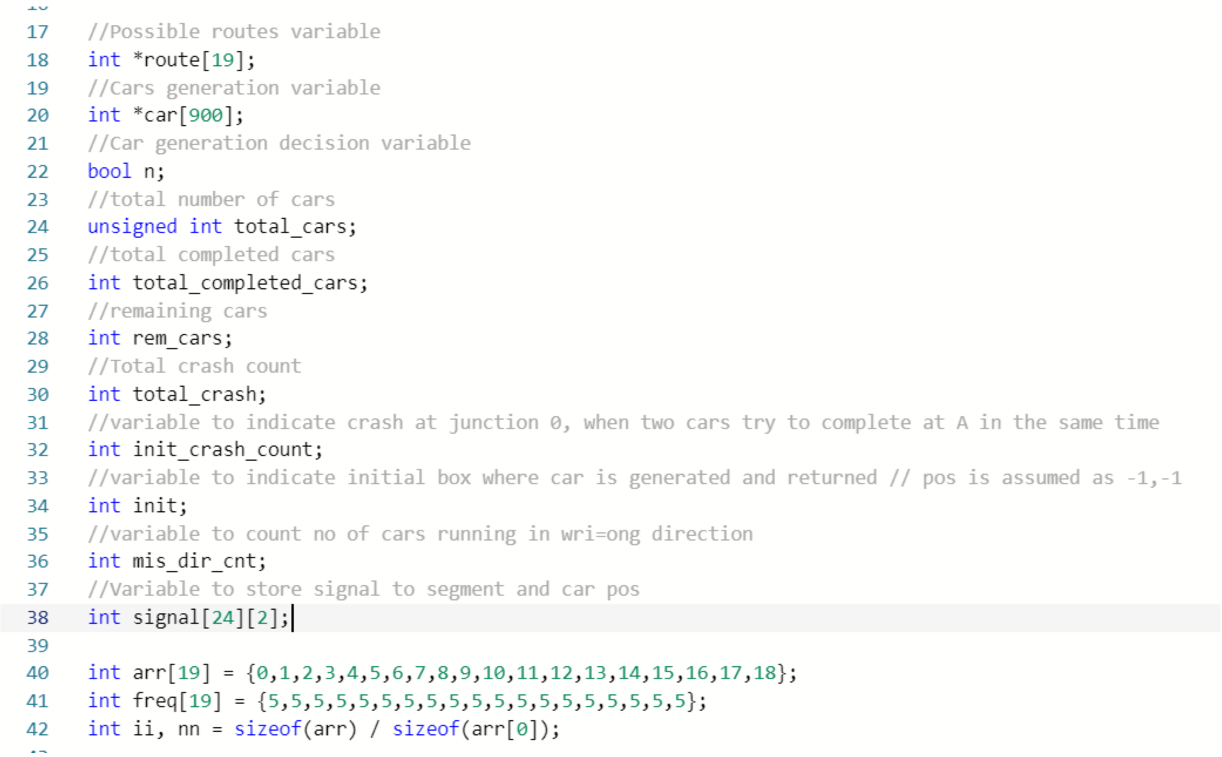
```
//Global variables
 //Time variable
int t;
//Contains empty slots as 0 and filled slots with Carid
int segment[24][30];

//Import from IV team
#define NUM_JUNCTIONS 9
#define NUM_SIGNALS 4

int il[9][4] = {0};
std::vector<std::queue<int>> q(9);
```

The main variables that are common between the I-group and V-group are shown in the Figure above.

t – time variable – As described in the problem statement, the time variable is considered as an integer. T = 1 implies 1 second. The time increments are done in steps of 2 i.e., each time step is 2 seconds. Each iteration t  will be increased by 2. The entire simulation should be run for 60 mins. Hence the loop is ran till t > 3600 and terminated after that.

Segment[24][30] – the segment variable is set as a 2-D array, where the first index indicates the segment number and the second index indicates the index of the element of inside each segment. This 2-D array is initialized to all zeros initially, which represents there are no cars in the road system. A car in the road system is represented by that car's id in one of the segment and one of the element sin the segment. For example if a car with id '1' is generated (will be covered later on the in functions), and the car moves from A to segment 0 first slot, then segment[0][0] will be set to 1. And when the car moves to the next slot, segment[0][0] is set back to 0 and segment [0][1] is set to 1. Hence, the 2-D segment array can give the congestion information (how many cars are there in the road system, how many cars in each segment etc) at any point of time in the simulation.

il[9][4]  - traffic signal lights – The il variable is also a 2-d array. The first index represents which intersection the light belongs to and the second index indicates the value of the light {0-not used,1-red,2-green}. The order of lights index at every intersection is fixed. For instance, at any intersection, the top light is indexed 0, left is indexed 3, right is indexed 1 and bottom is indexed 2. This is a shared variable and size was obtained from the I-group. For now, we are manipulating this variable to demonstrate different scenarios of the V-group.

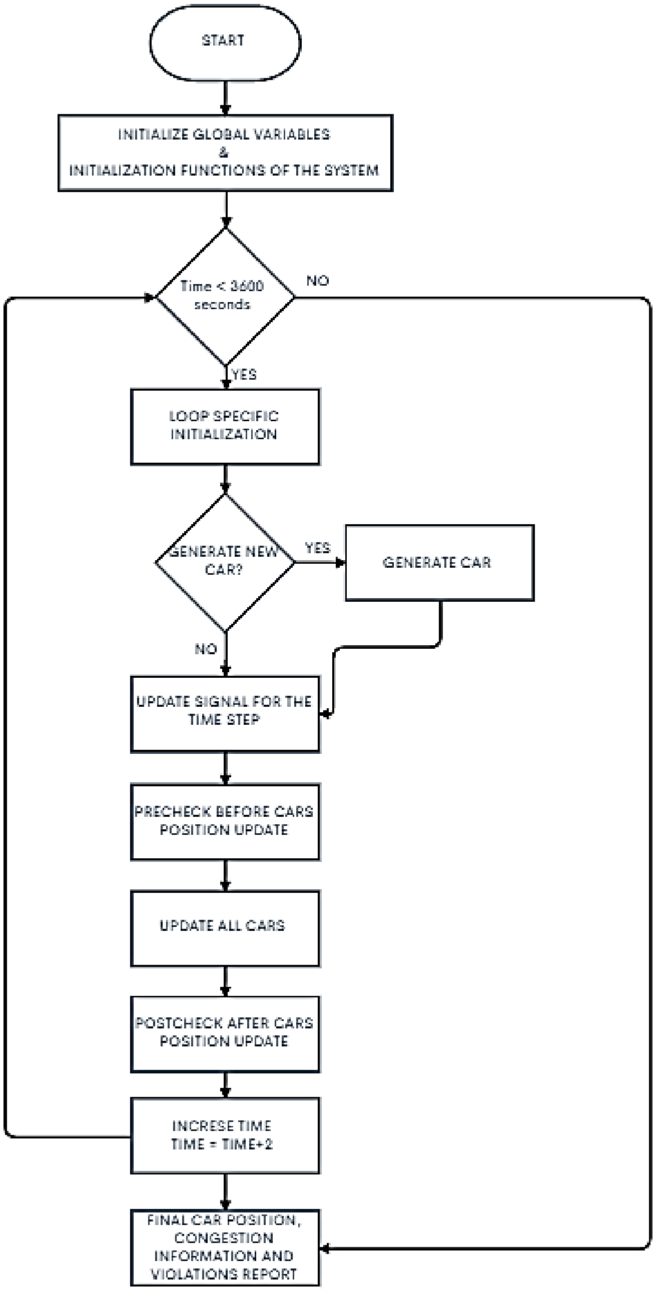


```
//Possible routes variable
int *route[19];
//Cars generation variable
int *car[900];
//Car generation decision variable
bool n;
//total number of cars
unsigned int total_cars;
//total completed cars
int total_completed_cars;
//remaining cars
int rem_cars;
//Total crash count
int total_crash;
//variable to indicate crash at junction 0, when two cars try to complete at A in the same time
int init_crash_count;
//variable to indicate initial box where car is generated and returned // pos is assumed as -1,-1
int init;
//variable to count no of cars running in wri=ong direction
int mis_dir_cnt;
//Variable to store signal to segment and car pos
int signal[24][2];

int arr[19] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18};
int freq[19] = {5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5};
int ii, nn = sizeof(arr) / sizeof(arr[0]);
```

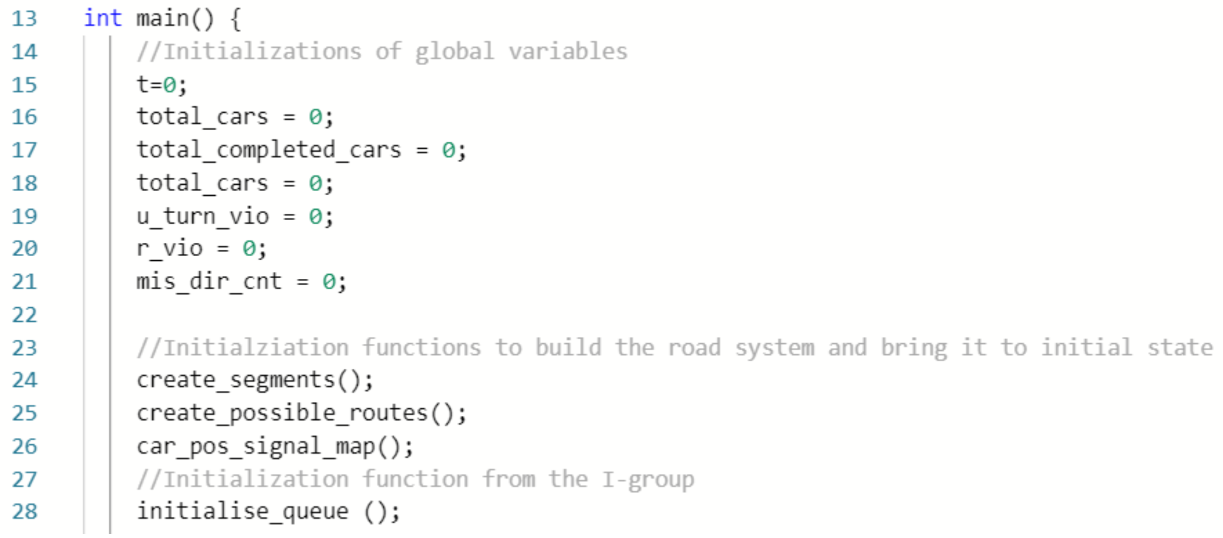
The list of other global variables used by the V-group is shown in the Figure above. The explanation of what the variable stores is given in the comments above each variable and will be discussed in detail in the upcoming sections. The arr,freq and nn variables are used for weighted random selection of the routes for newly generated cars based on the current congestion information. The freq is filled with same number to indicate that this is a simple scenario where all the possible routes have equal probability to be selected by the car (will be discussed in detail in later sections).

# ALGORITHM

The overall flow of the road system simulation for the V-group is shown in the flowchart in Figure below. Each section of the flowchart makes up the complete algorithm and functions performing each section are explained.


START
INITIALIZE GLOBAL VARIABLES
&
INITIALIZATION FUNCTIONS OF THE SYSTEM
Time < 3600 seconds
NO
YES
LOOP SPECIFIC INITIALIZATION
GENERATE NEW CAR?
YES
GENERATE CAR
NO
UPDATE SIGNAL FOR THE TIME STEP
PRECHECK BEFORE CARS POSITION UPDATE
UPDATE ALL CARS
POSTCHECK AFTER CARS POSITION UPDATE
INCRESE TIME
TIME = TIME+2
FINAL CAR POSITION, CONGESTION INFORMATION AND VIOLATIONS REPORT

*INTIALIZE GLOBAL VARIABLES AND INITIALIZATION FUNCTIONS:*



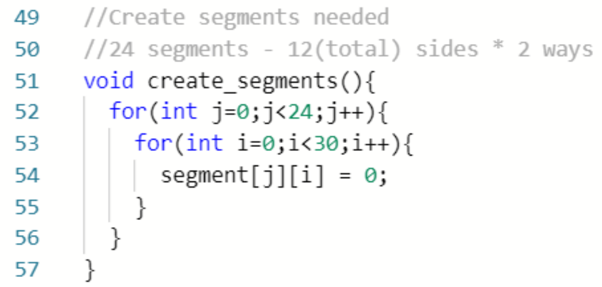
```
int main() {
    //Initializations of global variables
    t=0;
    total_cars = 0;
    total_completed_cars = 0;
    total_cars = 0;
    u_turn_vio = 0;
    r_vio = 0;
    mis_dir_cnt = 0;

    //Initialziation functions to build the road system and bring it to initial state
    create_segments();
    create_possible_routes();
    car_pos_signal_map();
    //Initialization function from the I-group
    initialise_queue ();
```

The time t is set to 0.

All other counters like total cars, total completed cars, different types of violations are set to 0.

The initialization functions are :

i. Create_segments()



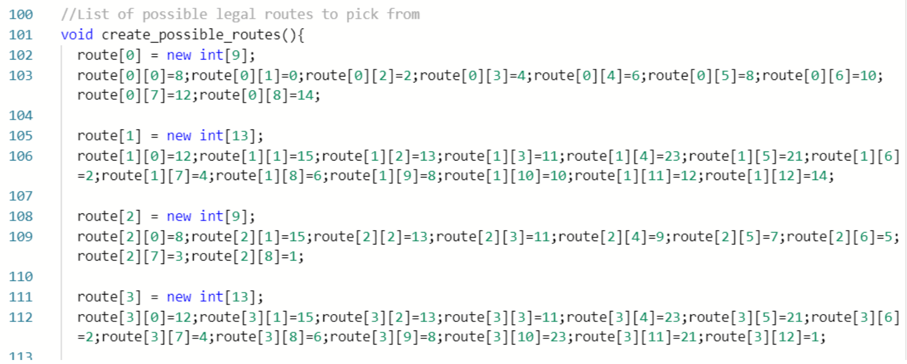
```
//Create segments needed
//24 segments - 12(total) sides * 2 ways
void create_segments(){
  for(int j=0;j<24;j++){
    for(int i=0;i<30;i++){
      segment[j][i] = 0;
    }
  }
}
```

There are 24 segments and initially there is going to be no cars in the system. Hence, all zeros are filled in every slot of the segments for all the 24 segments.

i. Create possible routes():

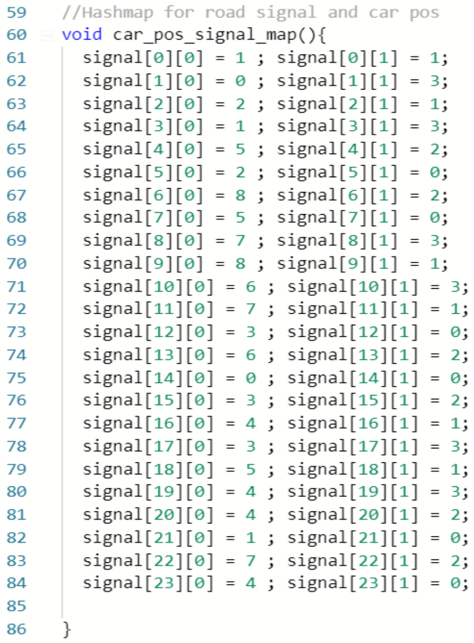
This function creates all possible routes from A to B,C and D in any order. The route variable is created as an array of dynamic size. In this 2-D array, the first index represents the route number and the second index first element is the no of hops between the segments in the route and rest of the elements in the second index is the segment numbers that needs to be followed to complete the tour and reach back to A. For example, from the Figure below, route 0 is dynamically assigned with size 9, first element i.e., route [0][0] = 8 is the number of hops between the segments and from route[0][1] to route[0][8], the numbers indicate that the car must follow segment 0,2,4,6,8,10,12 and 14 in order to complete the tour and reach back to A.

Similarly a number of possible routes are created and the new car will pick one from this list of routes when it is generated.



```
//List of possible legal routes to pick from
void create_possible_routes(){
  route[0] = new int[9];
  route[0][0]=8;route[0][1]=0;route[0][2]=2;route[0][3]=4;route[0][4]=6;route[0][5]=8;route[0][6]=10;
  route[0][7]=12;route[0][8]=14;

  route[1] = new int[13];
  route[1][0]=12;route[1][1]=15;route[1][2]=13;route[1][3]=11;route[1][4]=23;route[1][5]=21;route[1][6]
  =2;route[1][7]=4;route[1][8]=6;route[1][9]=8;route[1][10]=10;route[1][11]=12;route[1][12]=14;

  route[2] = new int[9];
  route[2][0]=8;route[2][1]=15;route[2][2]=13;route[2][3]=11;route[2][4]=9;route[2][5]=7;route[2][6]=5;
  route[2][7]=3;route[2][8]=1;

  route[3] = new int[13];
  route[3][0]=12;route[3][1]=15;route[3][2]=13;route[3][3]=11;route[3][4]=23;route[3][5]=21;route[3][6]
  =2;route[3][7]=4;route[3][8]=6;route[3][9]=8;route[3][10]=23;route[3][11]=21;route[3][12]=1;
```

ii. Car_pos_signal_map():



```
//Hashmap for road signal and car pos
void car_pos_signal_map(){
  signal[0][0] = 1 ; signal[0][1] = 1;
  signal[1][0] = 0 ; signal[1][1] = 3;
  signal[2][0] = 2 ; signal[2][1] = 1;
  signal[3][0] = 1 ; signal[3][1] = 3;
  signal[4][0] = 5 ; signal[4][1] = 2;
  signal[5][0] = 2 ; signal[5][1] = 0;
  signal[6][0] = 8 ; signal[6][1] = 2;
  signal[7][0] = 5 ; signal[7][1] = 0;
  signal[8][0] = 7 ; signal[8][1] = 3;
  signal[9][0] = 8 ; signal[9][1] = 1;
  signal[10][0] = 6 ; signal[10][1] = 3;
  signal[11][0] = 7 ; signal[11][1] = 1;
  signal[12][0] = 3 ; signal[12][1] = 0;
  signal[13][0] = 6 ; signal[13][1] = 2;
  signal[14][0] = 0 ; signal[14][1] = 0;
  signal[15][0] = 3 ; signal[15][1] = 2;
  signal[16][0] = 4 ; signal[16][1] = 1;
  signal[17][0] = 3 ; signal[17][1] = 3;
  signal[18][0] = 5 ; signal[18][1] = 1;
  signal[19][0] = 4 ; signal[19][1] = 3;
  signal[20][0] = 4 ; signal[20][1] = 2;
  signal[21][0] = 1 ; signal[21][1] = 0;
  signal[22][0] = 7 ; signal[22][1] = 2;
  signal[23][0] = 4 ; signal[23][1] = 0;

}
```

This function creates a predefined map (hash table) between each of the segment (each road) and corresponding signal it has to follow or depend on. The first index indicates the segment number. Since there are 24 segments, first index ranges from 0-23. The second index is fixed and ranges from 0-1. The second index, [0] represents the intersection number and [1] represents the light index from the I-group. Depending on these values from the hash map, whenever a car is crossing an intersection, the corresponding lights will be checked and the car position will be updated (covered in detail in update_car()).

iii. Initialize_queue():

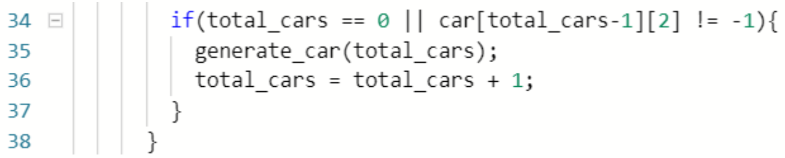
This function is inherited from the I-group. It initializes all the values of the signal lights at all the intersections.

***LOOP SPECIFIC INITIALIZATIONS:***

The init_crash_count variable is reset every time step. This variable will be reporting the number of crashes at all 9 intersections at each time instance. This will be accumulated in a global variable total_crash over the entire simulation.

***GENERATE CAR:***

The generation of car is done every 2 seconds (every time step). A new car will be generated only when the time step t=0 or the initial slot at A is not occupied by any other car generated before which has not updated and entered the road system due to a red signal. This is shown in the Figure below.

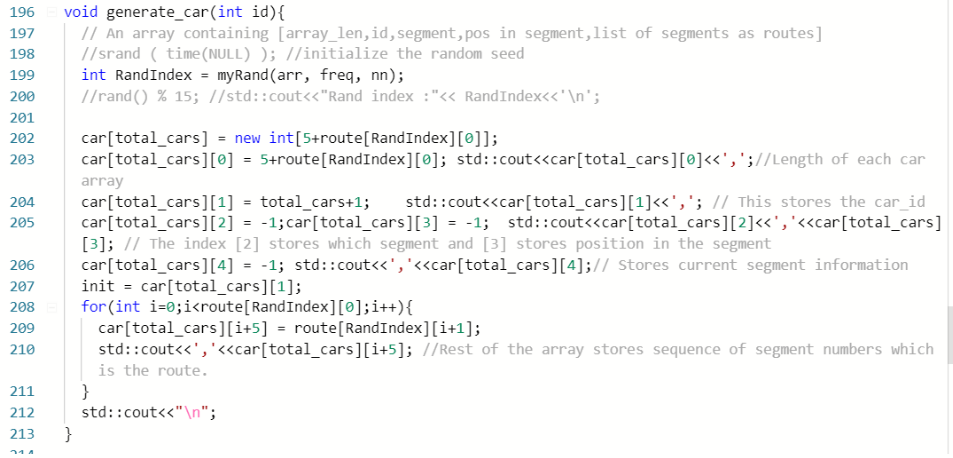


```
            if(total_cars == 0 || car[total_cars-1][2] != -1){
              generate_car(total_cars);
              total_cars = total_cars + 1;
            }
          }
```

Whenever a new car is generated, the function generate_car() is invoked and the total_number_of_cars in the system is incremented.

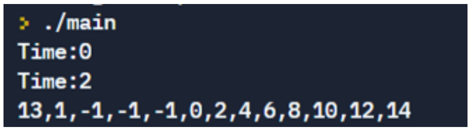
i.generate_car()

For each car that is generated, an array of dynamic size is created. The route that the car has to follow is randomly selected initially. We are using weighted random selection where the routes with the shortest distance are given more weights and routes with longer distance are given less weights. As seen from the Figure below line 199, the weighted random function is called to decide the route. The rest of the contents of the car array are explained below:



```
196  void generate_car(int id){
197    // An array containing [array_len,id,segment,pos in segment,list of segments as routes]
198    //srand ( time(NULL) ); //initialize the random seed
199    int RandIndex = myRand(arr, freq, nn);
200    //rand() % 15; //std::cout<<"Rand index :"<< RandIndex<<'\n';
201
202    car[total_cars] = new int[5+route[RandIndex][0]];
203    car[total_cars][0] = 5+route[RandIndex][0]; std::cout<<car[total_cars][0]<<',';//Length of each car array
204    car[total_cars][1] = total_cars+1;    std::cout<<car[total_cars][1]<<','; // This stores the car_id
205    car[total_cars][2] = -1;car[total_cars][3] = -1;  std::cout<<car[total_cars][2]<<','<<car[total_cars][3]; // The index [2] stores which segment and [3] stores position in the segment
206    car[total_cars][4] = -1; std::cout<<','<<car[total_cars][4];// Stores current segment information
207    init = car[total_cars][1];
208    for(int i=0;i<route[RandIndex][0];i++){
209      car[total_cars][i+5] = route[RandIndex][i+1];
210      std::cout<<','<<car[total_cars][i+5]; //Rest of the array stores sequence of segment numbers which is the route.
211    }
212    std::cout<<"\n";
213  }
```

For example, when a new car is created, the dynamic array for the corresponding car will look like this.



```
> ./main
Time:0
Time:2
13,1,-1,-1,-1,0,2,4,6,8,10,12,14
```

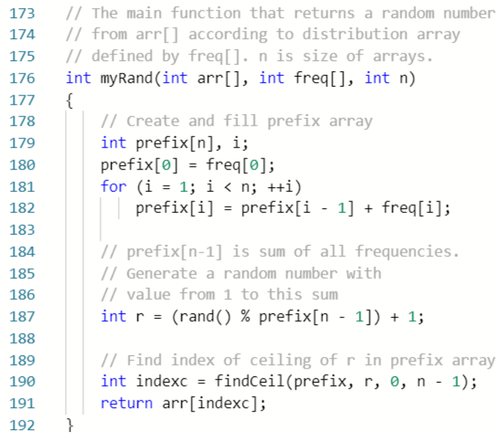
The first element indicates the size of the array (this can change depending on the length of route it selects)

The second element indicates the car-id (for the first car – id=1)

The third, fourth and fifth element indicates the segment number, slot number (position of the car) and index of current segment number in route. Initially all are set to -1 indicating the car is at A and hasn't entered the road system yet.

The rest of the elements is the sequence of segments that the car has to follow in order to visit B,C,D and return to A. (This is dynamic and appended from the route dynamic array based on the route picked by weighted random selection) [0,2,4,6,8,10,12,14 – car follows this sequence of segments to finish its tour]

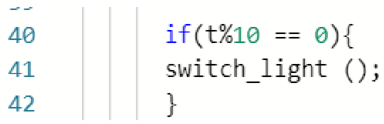
Route for each new car is picked according to the weights set for the route and that weight are dynamically changed based on the traffic congestion information from the segment variable.



```
// The main function that returns a random number
// from arr[] according to distribution array
// defined by freq[]. n is size of arrays.
int myRand(int arr[], int freq[], int n)
{
    // Create and fill prefix array
    int prefix[n], i;
    prefix[0] = freq[0];
    for (i = 1; i < n; ++i)
        prefix[i] = prefix[i - 1] + freq[i];

    // prefix[n-1] is sum of all frequencies.
    // Generate a random number with
    // value from 1 to this sum
    int r = (rand() % prefix[n - 1]) + 1;

    // Find index of ceiling of r in prefix array
    int indexc = findCeil(prefix, r, 0, n - 1);
    return arr[indexc];
}
```

### *UPDATE SIGNAL:*

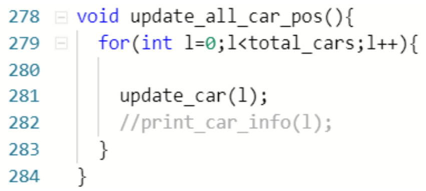
This part is supplied by the I-group. For now we have assumed a simple round robin that sets one green light at each junction and the light switches every 10 s (5 time steps) as shown in Figure below.



```
            if(t%10 == 0){
            switch_light ();
            }
```

### *UPDATE ALL CARS:*

After all the precheck functions (discussed later in precheck, poscheck and violations section) are completed, the update_all_cars function is invoked. This update_all_car_pos() will in turn invoke update_car() function for each of the car in the road system currently present at that time step.

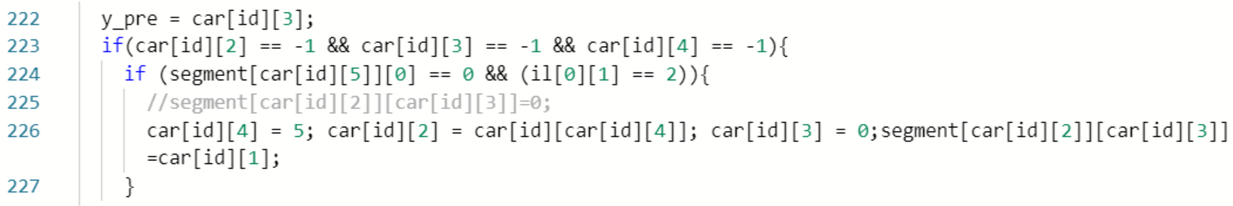


```
278 void update_all_car_pos(){
279   for(int l=0;l<total_cars;l++){
280 
281     update_car(l);
282     //print_car_info(l);
283   }
284 }
```

Update_car():

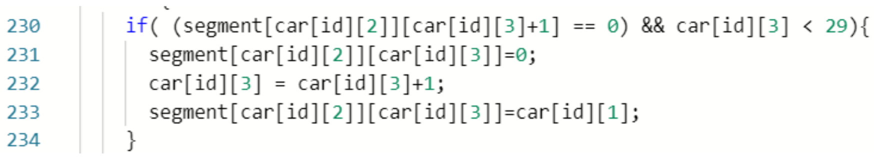
The different cases addressed for updating  the car position are as follows:

i. When the car is in initial position (line 223), only if the next segment that it wants to enter is free and the corresponding light is green (line 224), the new position of the car is updated in the segment variable (car-id is updated at that segment and slot – line 226) and car array.



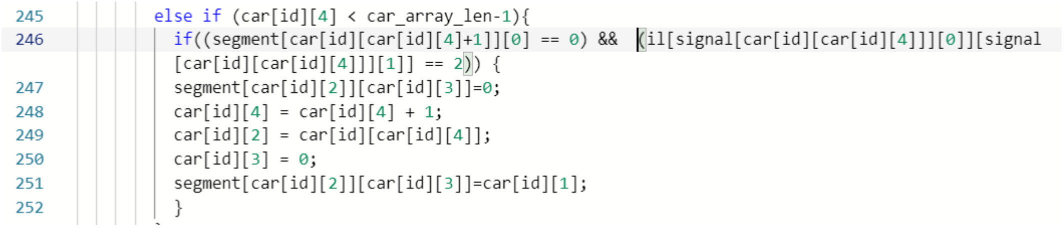
```
222   y_pre = car[id][3];
223   if(car[id][2] == -1 && car[id][3] == -1 && car[id][4] == -1){
224     if (segment[car[id][5]][0] == 0 && (il[0][1] == 2)){
225       //segment[car[id][2]][car[id][3]]=0;
226       car[id][4] = 5; car[id][2] = car[id][car[id][4]]; car[id][3] = 0;segment[car[id][2]][car[id][3]]
          =car[id][1];
227     }
```

ii. When the car is in any segment and the slot value is less than 29 (line 230), it can just update its position to the next slot (line 232) if that is free (line 230). It doesn't need to check for the lights. Old position in segment variable is reset to zero and new position I segment is updated with car id (line 231 and 233) and in car_array only the slot number is updated (line 232).



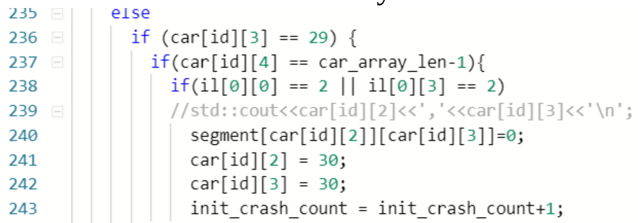
```
230     if( (segment[car[id][2]][car[id][3]+1] == 0) && car[id][3] < 29){
231       segment[car[id][2]][car[id][3]]=0;
232       car[id][3] = car[id][3]+1;
233       segment[car[id][2]][car[id][3]]=car[id][1];
234     }
```

iii. When the car is at the end of the segment, in order to move to the new segment, it should check the light and also if the next slot in the new segment is free (line 246). The new segment is updated, the slot is by default 0 and the current index of segment in the route is also updated in the car variable (line 248-250). Old position in segment variable is reset to zero and new position I segment is updated with car id (line 247 and 251)

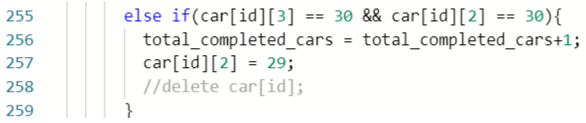


```
245     else if (car[id][4] < car_array_len-1){
246       if((segment[car[id][car[id][4]+1]][0] == 0) && (il[signal[car[id][car[id][4]]][0]][signal
          [car[id][car[id][4]]][1]] == 2)) {
247       segment[car[id][2]][car[id][3]]=0;
248       car[id][4] = car[id][4] + 1;
249       car[id][2] = car[id][car[id][4]];
250       car[id][3] = 0;
251       segment[car[id][2]][car[id][3]]=car[id][1];
252       }
```

iv. When the car is in the last segment of the route and the next step is to return back to A, it just has to check for the lights. The final position is indicated by 30,30 and next time for a finished car the final position becomes 29,30 which is out of bounds for the segment variable. Hence this can be used to represent cars that have finished and at the same time out of the road system.



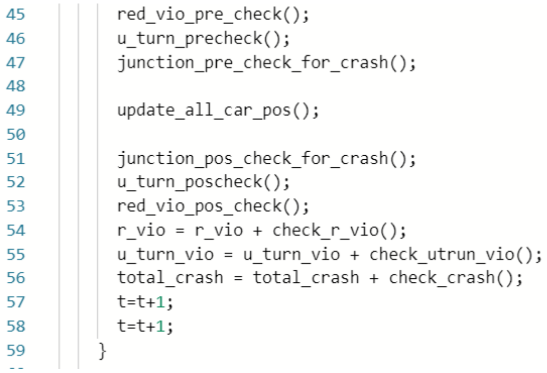
```
else
  if (car[id][3] == 29) {
    if(car[id][4] == car_array_len-1){
      if(il[0][0] == 2 || il[0][3] == 2)
      //std::cout<<car[id][2]<<','<<car[id][3]<<'\n';
        segment[car[id][2]][car[id][3]]=0;
        car[id][2] = 30;
        car[id][3] = 30;
        init_crash_count = init_crash_count+1;
```



```
else if(car[id][3] == 30 && car[id][2] == 30){
  total_completed_cars = total_completed_cars+1;
  car[id][2] = 29;
  //delete car[id];
}
```

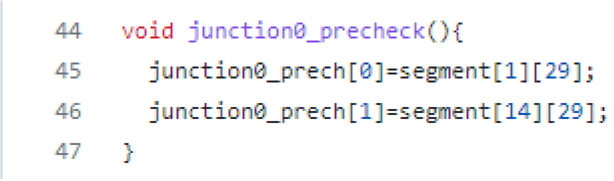
### *PRECHECK, POSCHECK AND VIOLATIONS:*

Intersections are also referred to as junctions and are used interchangeably. The functions in this section are used in the main function as shown in the Figure below. The necessary values are captured before and after updating the car positions, compared to identify violations and accumulated over the entire each time step of the simulation to report the total number at the end of simulation.



```
    red_vio_pre_check();
    u_turn_precheck();
    junction_pre_check_for_crash();

    update_all_car_pos();

    junction_pos_check_for_crash();
    u_turn_poscheck();
    red_vio_pos_check();
    r_vio = r_vio + check_r_vio();
    u_turn_vio = u_turn_vio + check_utrun_vio();
    total_crash = total_crash + check_crash();
    t=t+1;
    t=t+1;
  }
```

i. void junction0_precheck():

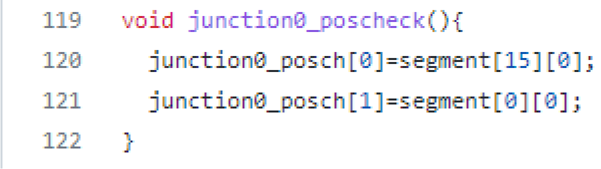


```
void junction0_precheck(){
  junction0_prech[0]=segment[1][29];
  junction0_prech[1]=segment[14][29];
}
```

The function junction0_precheck() initializes the array of slots to be checked at the junction 0. These are the slots at the junctions before car moves to the next segment. For instance, junction0_prech[0] indicates the last slot in segment 1 i.e. segment[1][29]. Similarly, junction0_prech[1] indicates the last slot in segment 14 i.e. segment[14][29]. If car is in these slots at current time, for the next time step the cars will move to another segment if signal is green.

On the similar lines, arrays are initialized for each junction. Since, there are 9 junctions, 9 arrays junction1_precheck(), junction2_precheck(), junction3_precheck() up to junction8_precheck() are initialized.

i. void junction0_poscheck():

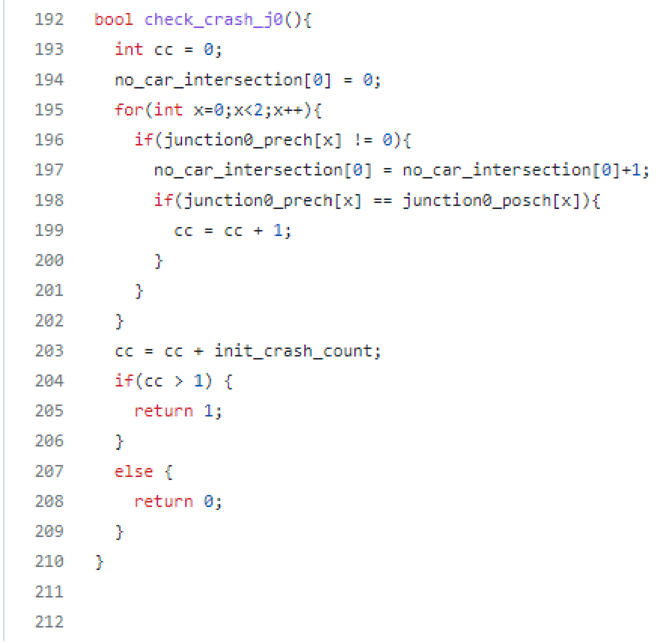


```
void junction0_poscheck(){
  junction0_posch[0]=segment[15][0];
  junction0_posch[1]=segment[0][0];
}
```

The function junction0_ poscheck () initializes the array of slots to be checked at the junction 0 after next time step. These are the slots at the junctions where car will move to the next segment in next time step. For instance, junction0_ poscheck[0] indicates that the car in slot segment[1][29] will move to segment[15][0] in next time step if signal is green. Similarly, junction0_ poscheck[0] indicates that the car in slot segment[41][29] will move to segment[0][0] in next time step if signal is green. The junction*_posch array has list of all the possible slots where can move in next time step at a junction if signal is green and next slot is empty.

On the similar lines, arrays are initialized for each junction. Since, there are 9 junctions, 9 arrays junction1_ poscheck (), junction2_ poscheck (), junction3_ poscheck () upto junction8_ poscheck () are initialized.

ii. bool check_crash_j0():



```
bool check_crash_j0(){
  int cc = 0;
  no_car_intersection[0] = 0;
  for(int x=0;x<2;x++){
    if(junction0_prech[x] != 0){
      no_car_intersection[0] = no_car_intersection[0]+1;
      if(junction0_prech[x] == junction0_posch[x]){
        cc = cc + 1;
      }
    }
  }
  cc = cc + init_crash_count;
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}
```

This function returns true if there is crash at junction 0. A crash will occur if two or more cars cross the intersection at the same time. To identify a crash junction0_ precheck and junction0_ poscheck arrays are used. If a car has moved to next segment by crossing intersection/junction then the value of junction0_ precheck will be equal to junction0_ poscheck. Here, a non-zero check was added as 0 indicates no car. If more than two values in arrays match then it can be interpreted as a crash. Only for junction 0, the initial crashes are also counted when two cars try to return at A at the same time.

Similarly, a crash check is implemented from all the 9 junctions.

iii. void junction_pre_check_for_crash():

Calling all the junction_pre_check functions for 9 junctions and initializing junction*_prech[] arrays.

iv. void junction_pos_check_for_crash()

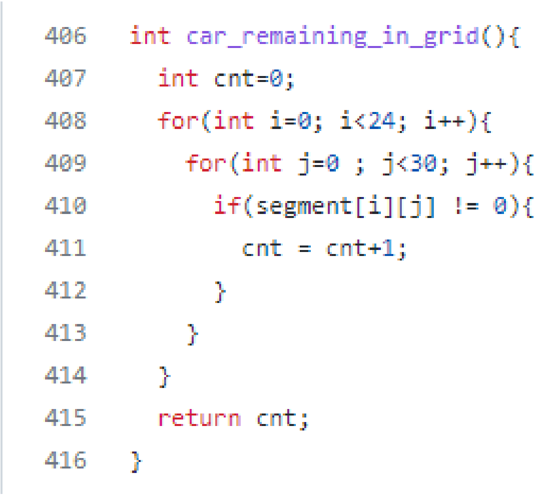
Calling all the junction_pos_check functions for 9 junctions and initializing junction*_posch[] arrays.

v. int check_crash():

The function returns the total number of junctions suffered by crashes.

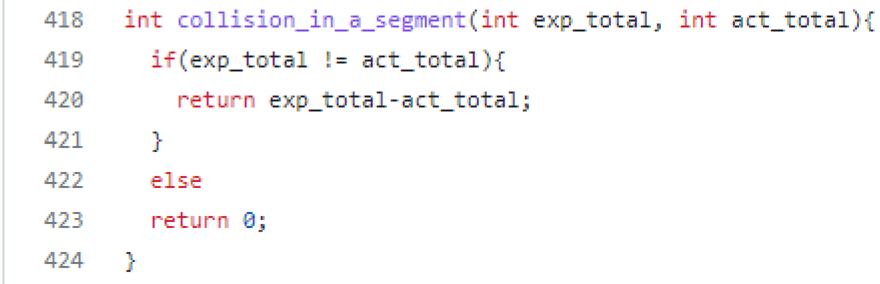
vi. car_remaining_in_grid():

This function returns total number of cars present in the grid at a time. The segment array is traversed and non-zero values are counted as the non-zero values indicate the car IDs.



```
int car_remaining_in_grid(){
  int cnt=0;
  for(int i=0; i<24; i++){
    for(int j=0 ; j<30; j++){
      if(segment[i][j] != 0){
        cnt = cnt+1;
      }
    }
  }
  return cnt;
}
```

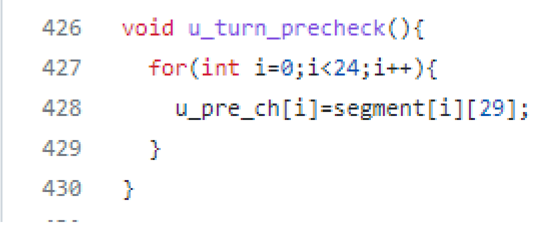
vii. int collision_in_a_segment(int exp_total, int act_total):



```
int collision_in_a_segment(int exp_total, int act_total){
  if(exp_total != act_total){
    return exp_total-act_total;
  }
  else
  return 0;
}
```

The function returns the collision in a segment. If Car 1 moves to next slot in the same segment even though there is Car 2 in next slot, the collision in segment will occur. If such collision happens, the ID of Car 2 in the next segment will be overwritten by Car 1's ID. And the value of slot where Car 1 was present will be assigned the value zero. Hence, Car 2 will disappear from the grid. The difference between sum of cars present on a grid & cars returned and total generated cars is used to identify the collisions in a segment.

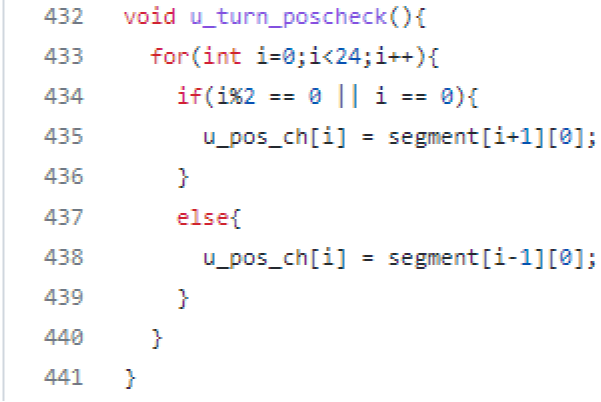
viii. u_turn_precheck():



```
void u_turn_precheck(){
  for(int i=0;i<24;i++){
    u_pre_ch[i]=segment[i][29];
  }
}
```

The function u_turn_precheck() initializes an array which stores the car ID present in end of each segment. For  example, the first element of array u_pos_ch will store the car ID present in segment[0][29]. The segment[0][29] is the last slot of segment 0.

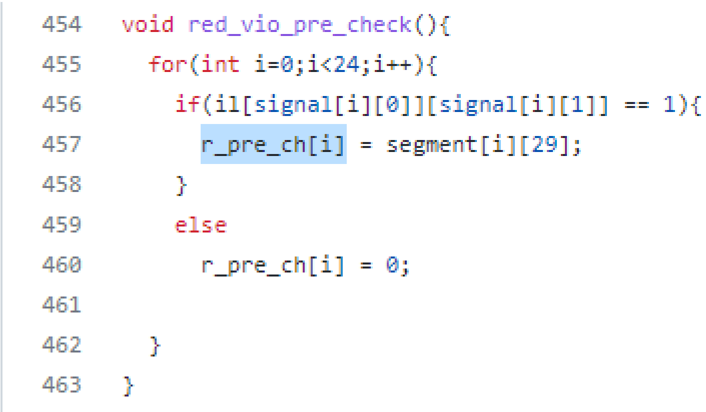
ix. void u_turn_poscheck()



```
void u_turn_poscheck(){
  for(int i=0;i<24;i++){
    if(i%2 == 0 || i == 0){
      u_pos_ch[i] = segment[i+1][0];
    }
    else{
      u_pos_ch[i] = segment[i-1][0];
    }
  }
}
```

This function also in initializes an array which stores the car ID present in start of next segment if a car takes U-turn in next time step. The first element of array u_pos_ch[] will store the car ID present in segment[0][29]. The segment[0][29] is the last slot of segment 0. Hence, if the car is present at segment[0][29] at current time stamp and it takes a U-turn in next time step, it will go to segment[1][0]. If a car present at segment[1][29] at current time stamp and it takes a U-turn in next time step, it will go to segment[0][0]. Hence, the values of segment[0][0] and segment[1][0] are stored in the array u_pos_ch[]. Similarly values at stored for all 24 segments.

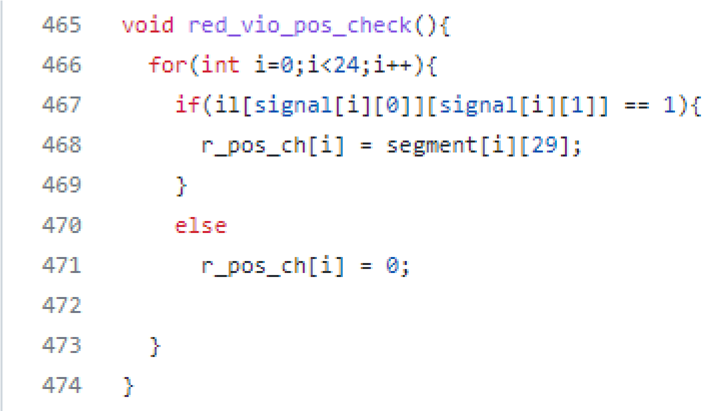
x. void red_vio_pre_check():



```
void red_vio_pre_check(){
  for(int i=0;i<24;i++){
    if(il[signal[i][0]][signal[i][1]] == 1){
      r_pre_ch[i] = segment[i][29];
    }
    else
      r_pre_ch[i] = 0;

  }
}
```

In this function, an array r_pre_ch[] stores the values of car IDs present in the last slot of each segment i.e. segment[i][29] where i = 0, 1, …23 if the signal for that car is RED at current time. For other signals GREEN or NO Signal, a zero is stored in the array. This is used to calculate red signal violations.

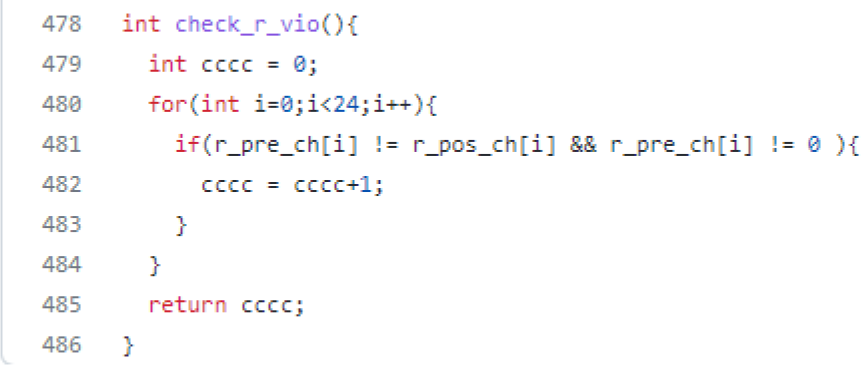
xi. void red_vio_pos_check():



```
void red_vio_pos_check(){
  for(int i=0;i<24;i++){
    if(il[signal[i][0]][signal[i][1]] == 1){
      r_pos_ch[i] = segment[i][29];
    }
    else
      r_pos_ch[i] = 0;

  }
}
```

In this function, an array r_pos_ch[] stores the values of car IDs present in the last slot of each segment i.e. segment[i][29] where i = 0, 1, ..., 23 for next time step. For other signals GREEN or NO Signal, a zero is stored in the array. This is used to calculate red signal violations.

xii. int check_r_vio():

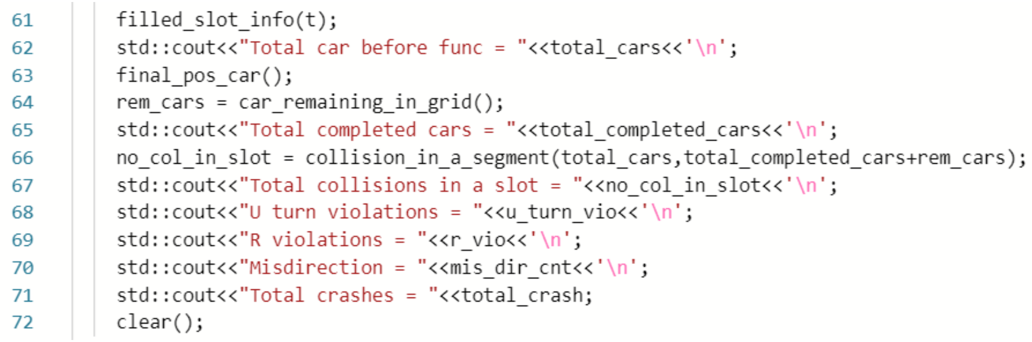


```
int check_r_vio(){
  int cccc = 0;
  for(int i=0;i<24;i++){
    if(r_pre_ch[i] != r_pos_ch[i] && r_pre_ch[i] != 0 ){
      cccc = cccc+1;
    }
  }
  return cccc;
}
```

The function check_r_vio() returns the number of red signal violations occurred in one time step at the junction. The arrays r_pre_ch[] and r_pos_ch[] are compared. If the car has moved although the signal is RED at time t = t + 2, then the for that car index the value in r_pos_ch[] will become zero. Hence, if r_pre_ch[i] is not equal to r_pos_ch[i], then it is interpreted as red signal violation by a car. The zeros are excluded from the check as the zero in r_pre_ch[] indicates that there is no car at the junction at time = t.

***FINAL OUTPUTS:***

As shown in the Figure below, the final car positions, congestion information and also all the different types are violations are given as outputs after the entire simulation.



```
filled_slot_info(t);
std::cout<<"Total car before func = "<<total_cars<<'\n';
final_pos_car();
rem_cars = car_remaining_in_grid();
std::cout<<"Total completed cars = "<<total_completed_cars<<'\n';
no_col_in_slot = collision_in_a_segment(total_cars,total_completed_cars+rem_cars);
std::cout<<"Total collisions in a slot = "<<no_col_in_slot<<'\n';
std::cout<<"U turn violations = "<<u_turn_vio<<'\n';
std::cout<<"R violations = "<<r_vio<<'\n';
std::cout<<"Misdirection = "<<mis_dir_cnt<<'\n';
std::cout<<"Total crashes = "<<total_crash;
clear();
```

## TESTCASES AND RESULTS

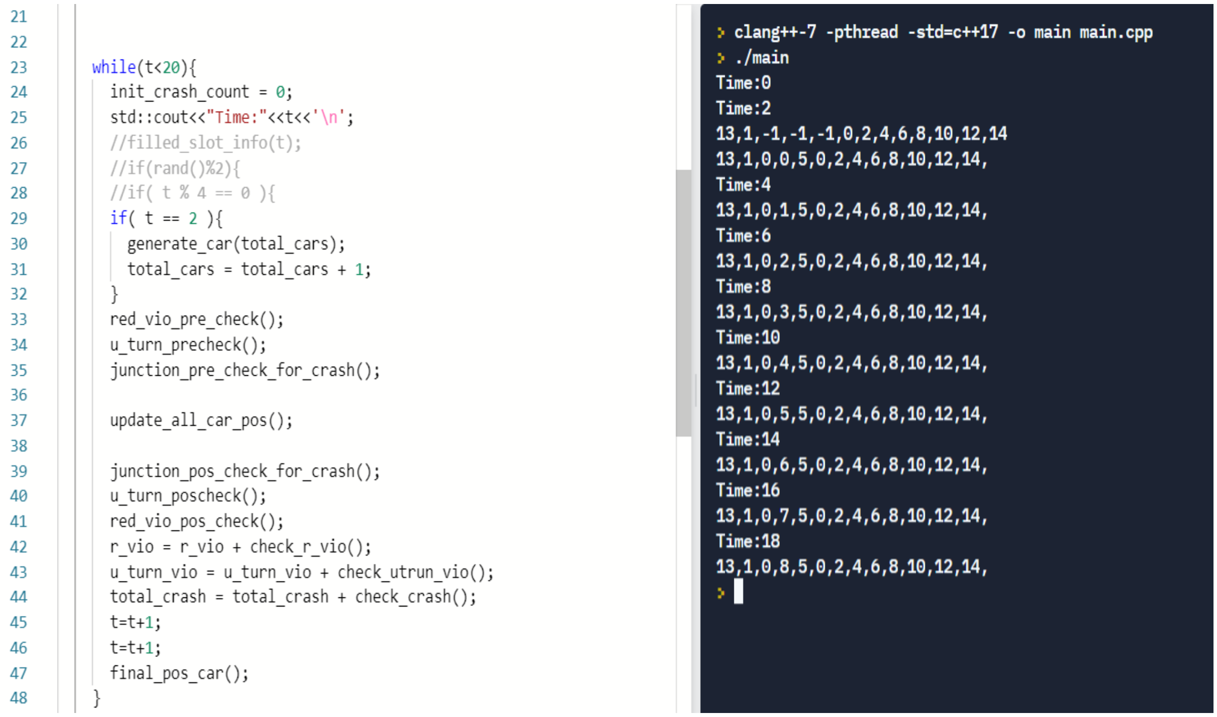
We have used print statements and mapped our output results using python to plot the visual representation of the testcases. Please find the GitHub link where all the outputs are documented.

https://github.com/rishyasankar/Formal_verification_project/tree/main/Phase1-V-submission

### Test Cases:

1. ***A car can run***
2. ***A car is present in a slot and its position is updated after time increment.***

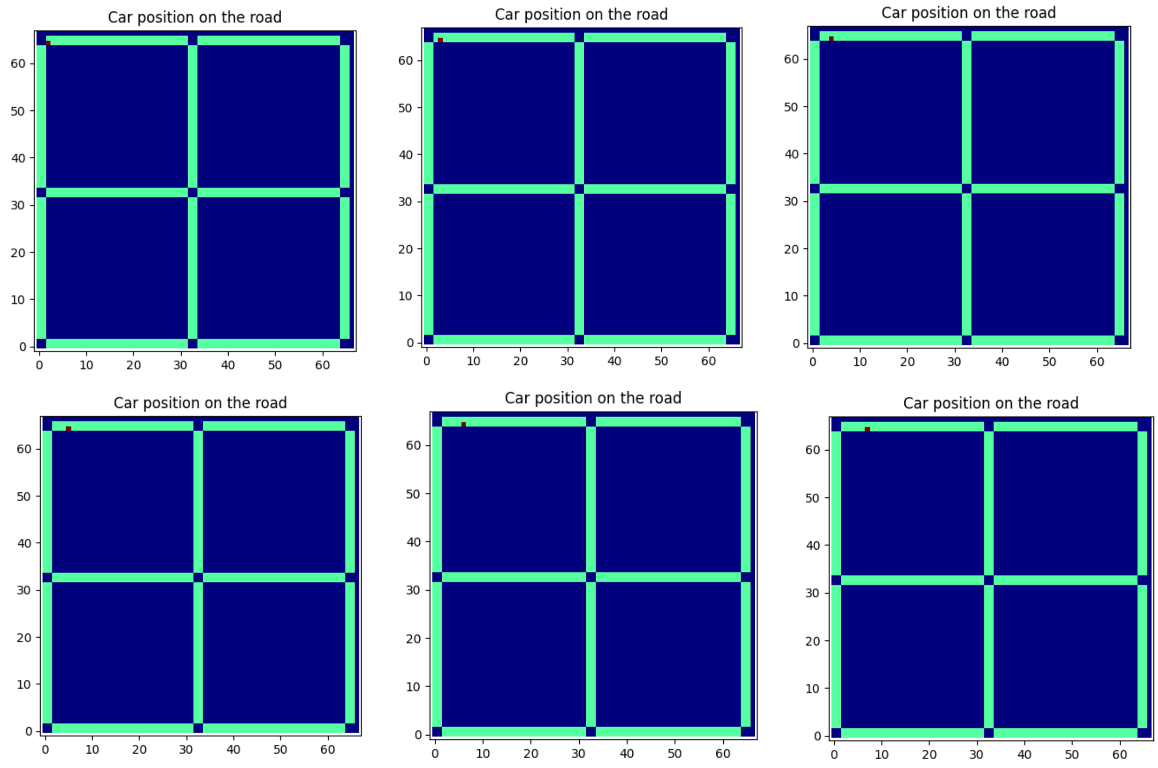
Below is the output for above test case 1 and test case 2:



```


while(t<20){
  init_crash_count = 0;
  std::cout<<"Time:"<<t<<'\n';
  //filled_slot_info(t);
  //if(rand()%2){
  //if( t % 4 == 0 ){
  if( t == 2 ){
    generate_car(total_cars);
    total_cars = total_cars + 1;
  }
  red_vio_pre_check();
  u_turn_precheck();
  junction_pre_check_for_crash();

  update_all_car_pos();

  junction_pos_check_for_crash();
  u_turn_poscheck();
  red_vio_pos_check();
  r_vio = r_vio + check_r_vio();
  u_turn_vio = u_turn_vio + check_utrun_vio();
  total_crash = total_crash + check_crash();
  t=t+1;
  t=t+1;
  final_pos_car();
}
```

```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
Time:0
Time:2
13,1,-1,-1,-1,0,2,4,6,8,10,12,14
13,1,0,0,5,0,2,4,6,8,10,12,14,
Time:4
13,1,0,1,5,0,2,4,6,8,10,12,14,
Time:6
13,1,0,2,5,0,2,4,6,8,10,12,14,
Time:8
13,1,0,3,5,0,2,4,6,8,10,12,14,
Time:10
13,1,0,4,5,0,2,4,6,8,10,12,14,
Time:12
13,1,0,5,5,0,2,4,6,8,10,12,14,
Time:14
13,1,0,6,5,0,2,4,6,8,10,12,14,
Time:16
13,1,0,7,5,0,2,4,6,8,10,12,14,
Time:18
13,1,0,8,5,0,2,4,6,8,10,12,14,
> 
```

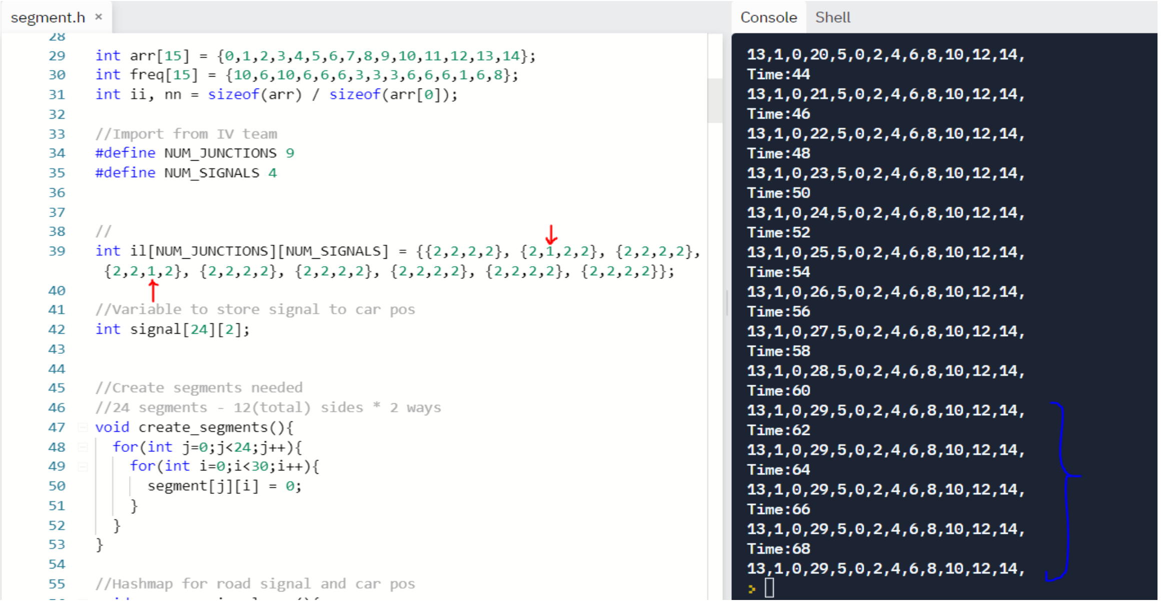
At time t = 2, car with ID 1 is generated. For each time step increment of 2 seconds, the car I smovinf forward in segment 0. This is indicated in the console by index 2 and 3 in printed lines. At time t = 2 , the car is generated and its initial position is -1, -1. At T= 2, the car has moved to first slot in segment 0. This is indicated by position 0,0. At time t = 4, car's position is 0, 1 i.e. segment zero slot one.



Above is the visual representation of the the console output. In the above images the red dot indicates the car’s position. For the first row, time increments from left to right. The first plot indicate the car’s position at time t = 0. The first picture in second row indicates car’s position at time t = 8 and time increments from left to right.

3. ***Car can stop***
4. ***Car stop at incremental time when signal is RED***

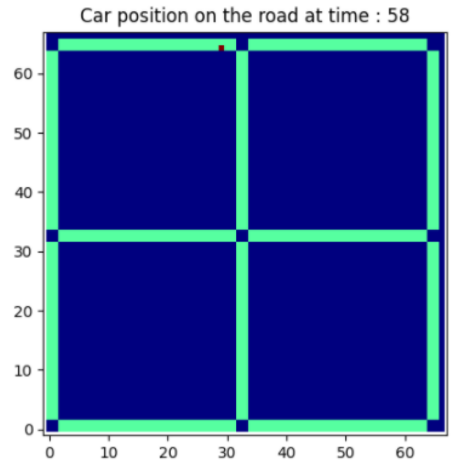
Below is the output for above test case 3 and test case 4:

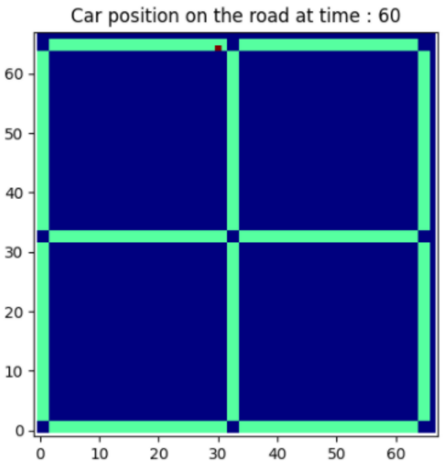


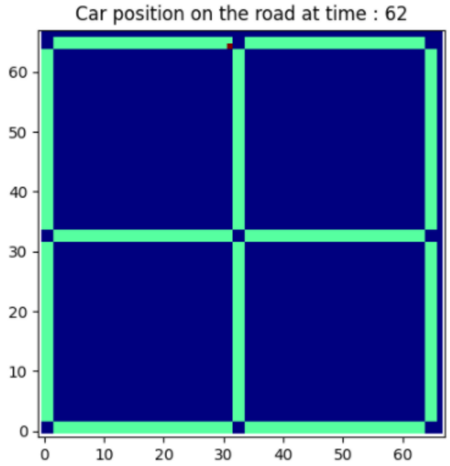
```c
int arr[15] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14};
int freq[15] = {10,6,10,6,6,6,3,3,3,6,6,6,1,6,8};
int ii, nn = sizeof(arr) / sizeof(arr[0]);

//Import from IV team
#define NUM_JUNCTIONS 9
#define NUM_SIGNALS 4


//
int il[NUM_JUNCTIONS][NUM_SIGNALS] = {{2,2,2,2}, {2,1,2,2}, {2,2,2,2},
 {2,2,1,2}, {2,2,2,2}, {2,2,2,2}, {2,2,2,2}, {2,2,2,2}, {2,2,2,2}};

//Variable to store signal to car pos
int signal[24][2];


//Create segments needed
//24 segments - 12(total) sides * 2 ways
void create_segments(){
  for(int j=0;j<24;j++){
    for(int i=0;i<30;i++){
      segment[j][i] = 0;
    }
  }
}

//Hashmap for road signal and car pos
```

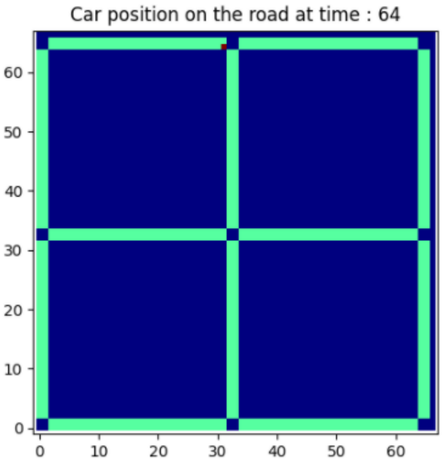
```
13,1,0,20,5,0,2,4,6,8,10,12,14,
Time:44
13,1,0,21,5,0,2,4,6,8,10,12,14,
Time:46
13,1,0,22,5,0,2,4,6,8,10,12,14,
Time:48
13,1,0,23,5,0,2,4,6,8,10,12,14,
Time:50
13,1,0,24,5,0,2,4,6,8,10,12,14,
Time:52
13,1,0,25,5,0,2,4,6,8,10,12,14,
Time:54
13,1,0,26,5,0,2,4,6,8,10,12,14,
Time:56
13,1,0,27,5,0,2,4,6,8,10,12,14,
Time:58
13,1,0,28,5,0,2,4,6,8,10,12,14,
Time:60
13,1,0,29,5,0,2,4,6,8,10,12,14,
Time:62
13,1,0,29,5,0,2,4,6,8,10,12,14,
Time:64
13,1,0,29,5,0,2,4,6,8,10,12,14,
Time:66
13,1,0,29,5,0,2,4,6,8,10,12,14,
Time:68
13,1,0,29,5,0,2,4,6,8,10,12,14,
>
```

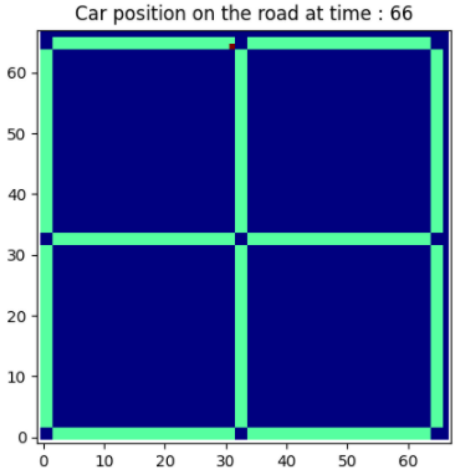
The signal 1 at juction 1 and signal 2 at juction 3 are set as RED. Refer architecture diagram for naming conventions. At time t = 44, car starts moving in next slots. At time t = 60, it reaches at the juction i.e. last slot of the segment 0 but since signal is RED, it won’t be able move to next segment. As indicated in the console output the car position is not updated at time t = 62 and onwards.

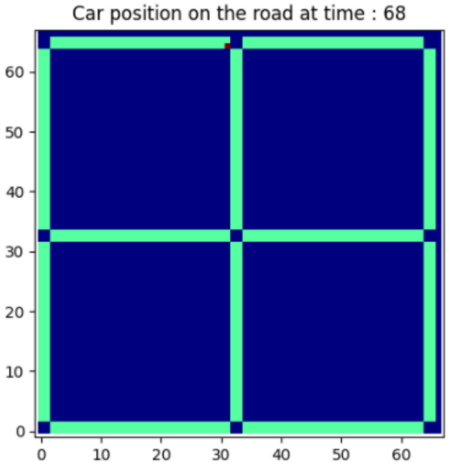






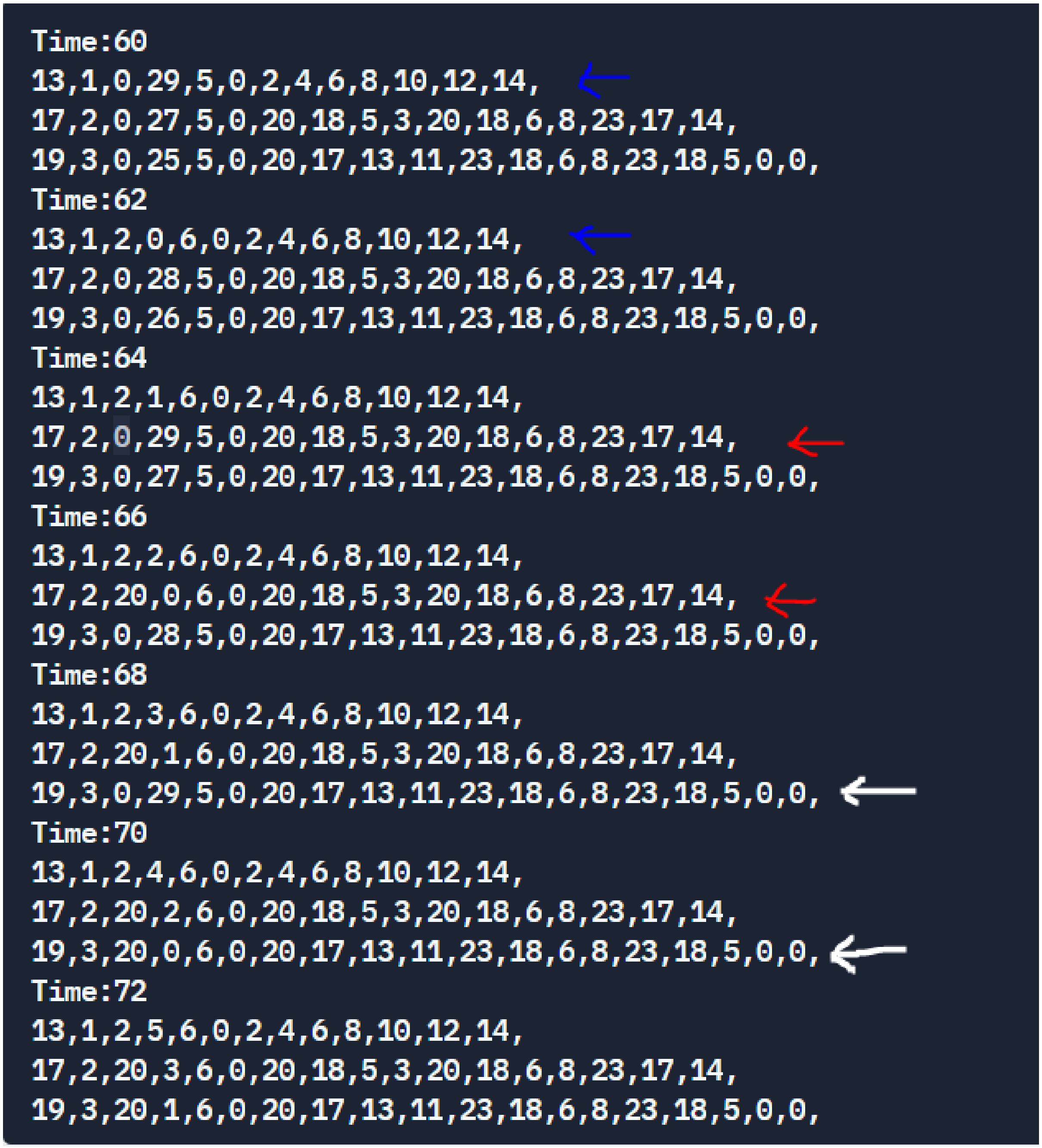




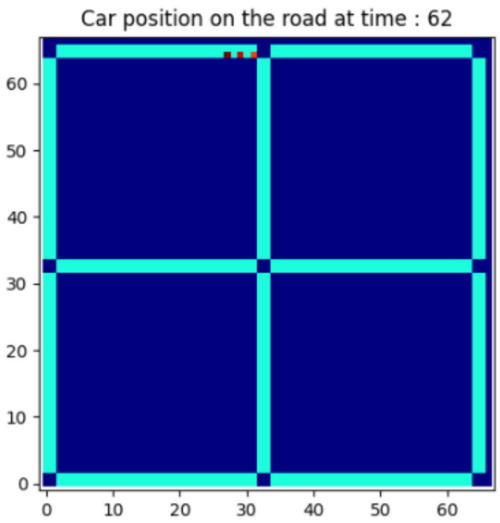


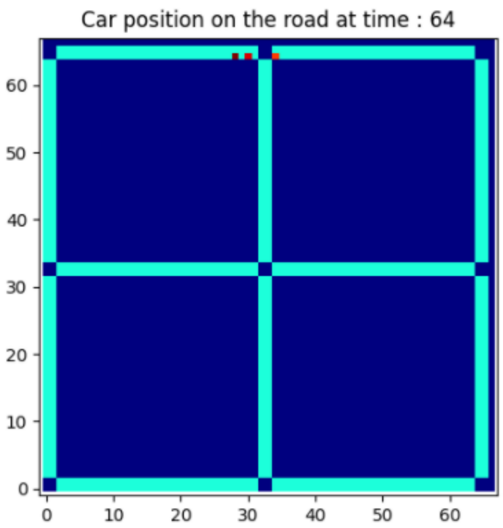
5. ***Car move straight at juction***
6. ***Car can turn left at juction***
7. ***Car can turn right at juction***

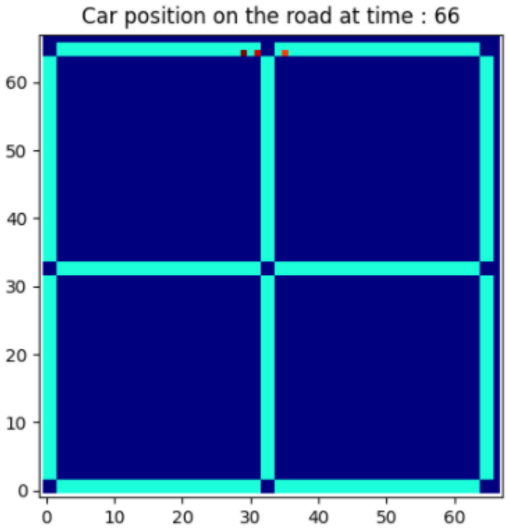
Below is the output for above test case 5, test case 6, and test case 7:

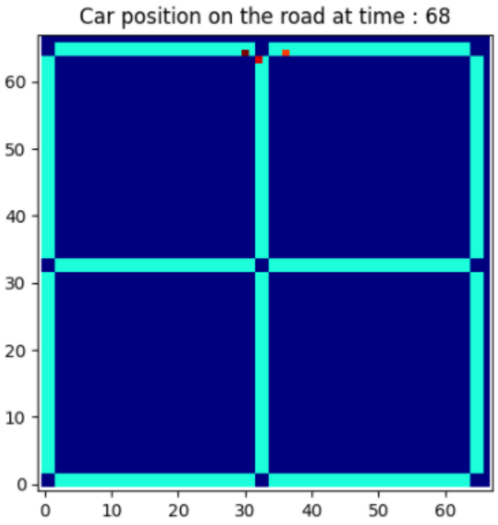


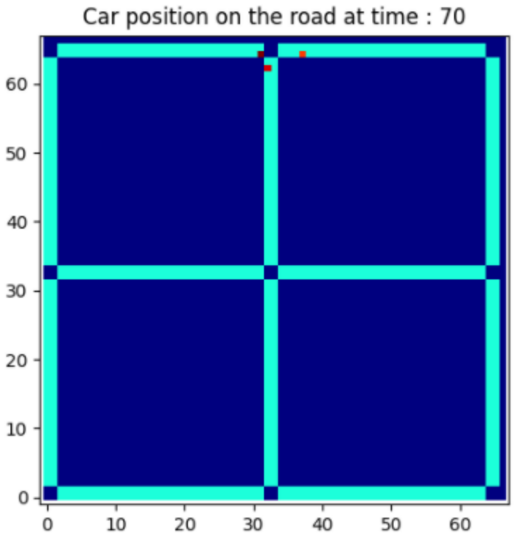
```
Time:60
13,1,0,29,5,0,2,4,6,8,10,12,14,
17,2,0,27,5,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,0,25,5,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:62
13,1,2,0,6,0,2,4,6,8,10,12,14,
17,2,0,28,5,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,0,26,5,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:64
13,1,2,1,6,0,2,4,6,8,10,12,14,
17,2,0,29,5,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,0,27,5,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:66
13,1,2,2,6,0,2,4,6,8,10,12,14,
17,2,20,0,6,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,0,28,5,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:68
13,1,2,3,6,0,2,4,6,8,10,12,14,
17,2,20,1,6,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,0,29,5,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:70
13,1,2,4,6,0,2,4,6,8,10,12,14,
17,2,20,2,6,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,20,0,6,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:72
13,1,2,5,6,0,2,4,6,8,10,12,14,
17,2,20,3,6,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,20,1,6,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
```

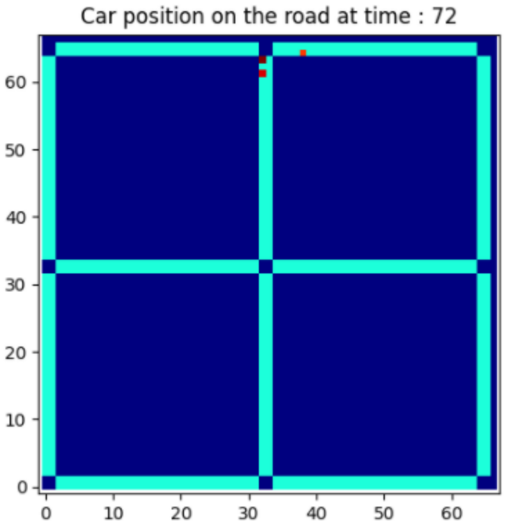

Car position on the road at time : 62
0
10
20
30
40
50
60


Car position on the road at time : 64
0
10
20
30
40
50
60

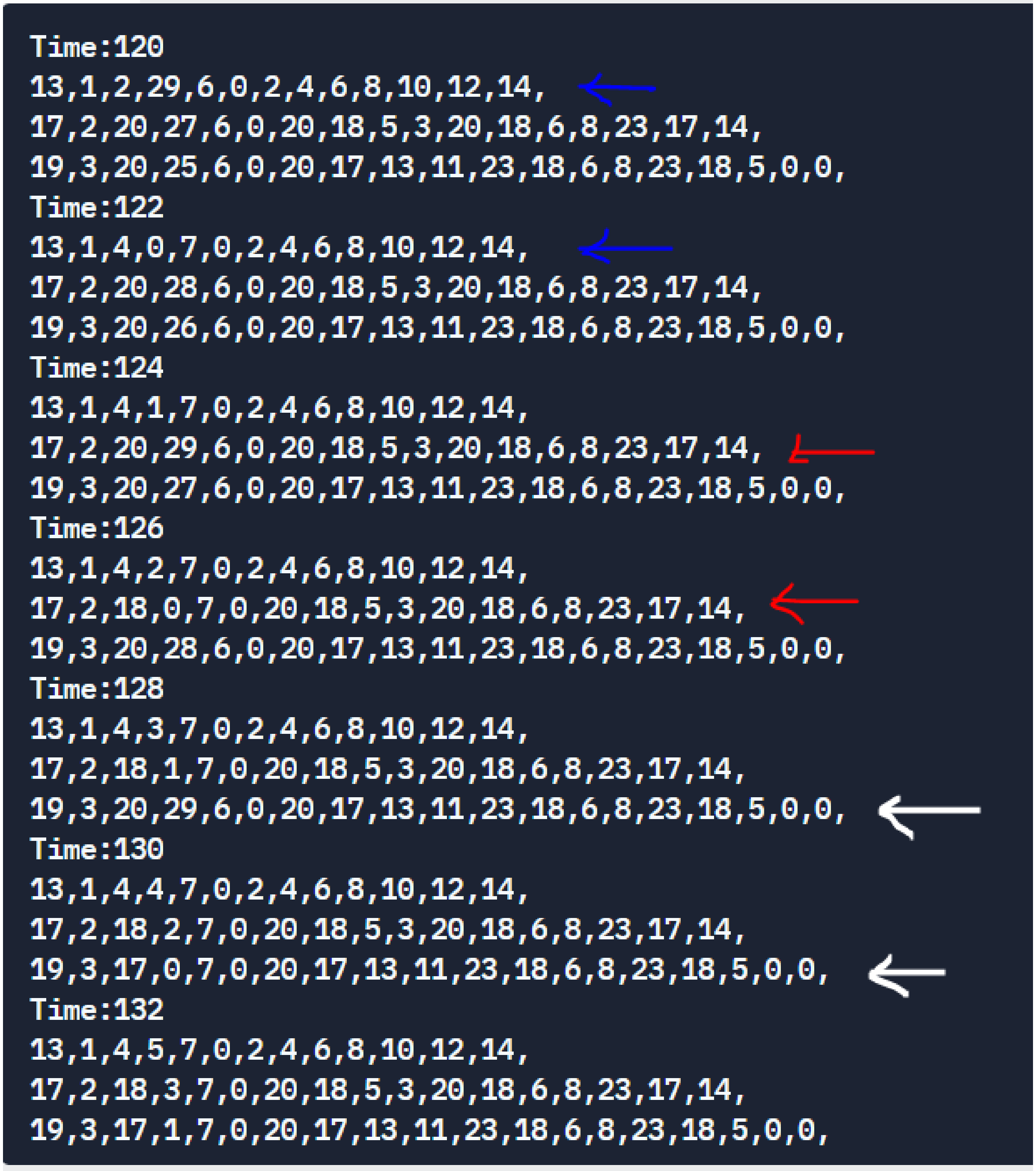

Car position on the road at time : 66
0
10
20
30
40
50
60


Car position on the road at time : 68
0
10
20
30
40
50
60

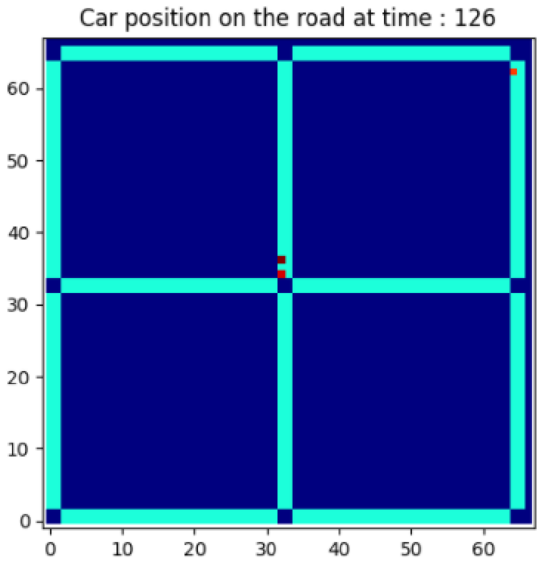

Car position on the road at time : 70
0
10
20
30
40
50
60

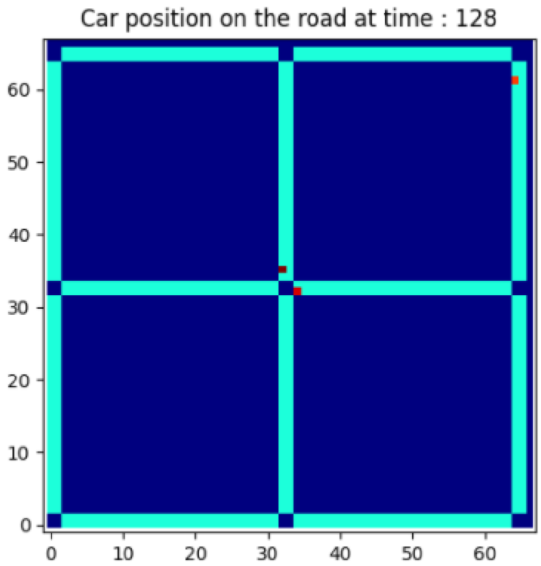

Car position on the road at time : 72
0
10
20
30
40
50
60

At time t = 62,the car 1 has reached at the junction. At t = 4, car 1 has moved straight to next segment. This indicated by blue arrows in console output. Similarly, at t = 68, car 2 has turned right and this is indicated by red arrows. At t = 72, car 3 has turned right, indicated by white arrows in console.



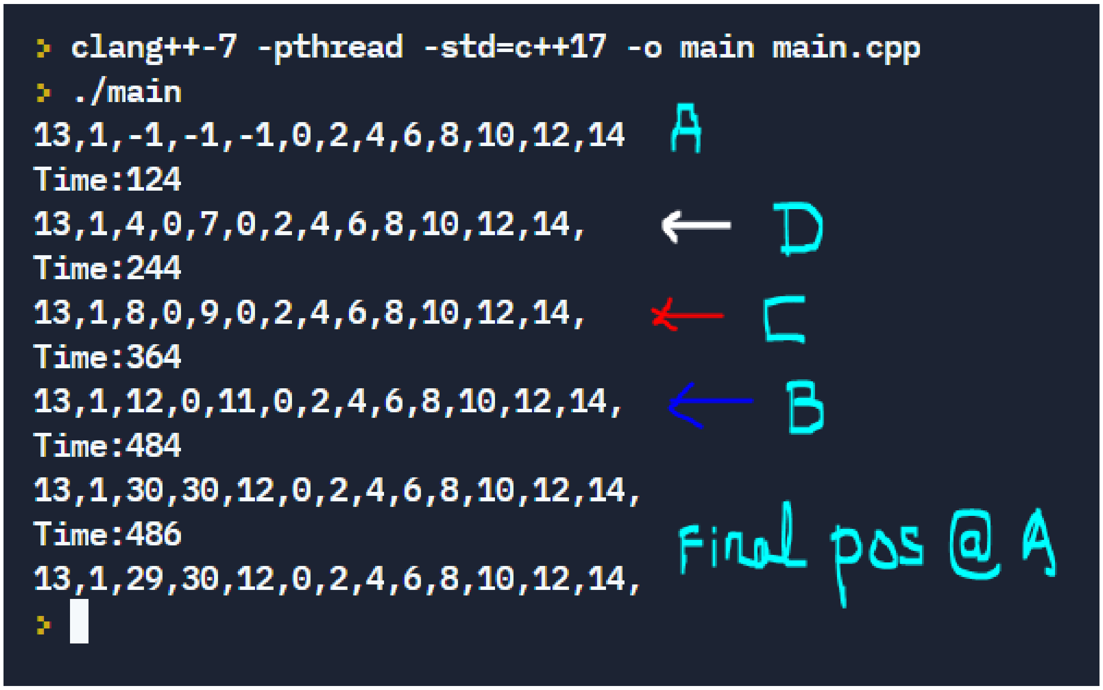
```
Time:120
13,1,2,29,6,0,2,4,6,8,10,12,14,
17,2,20,27,6,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,20,25,6,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:122
13,1,4,0,7,0,2,4,6,8,10,12,14,
17,2,20,28,6,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,20,26,6,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:124
13,1,4,1,7,0,2,4,6,8,10,12,14,
17,2,20,29,6,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,20,27,6,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:126
13,1,4,2,7,0,2,4,6,8,10,12,14,
17,2,18,0,7,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,20,28,6,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:128
13,1,4,3,7,0,2,4,6,8,10,12,14,
17,2,18,1,7,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,20,29,6,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:130
13,1,4,4,7,0,2,4,6,8,10,12,14,
17,2,18,2,7,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,17,0,7,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
Time:132
13,1,4,5,7,0,2,4,6,8,10,12,14,
17,2,18,3,7,0,20,18,5,3,20,18,6,8,23,17,14,
19,3,17,1,7,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
```


Car position on the road at time : 126

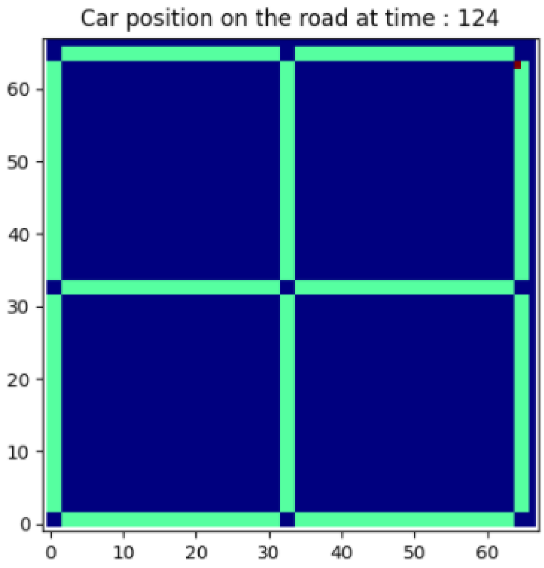

Car position on the road at time : 128

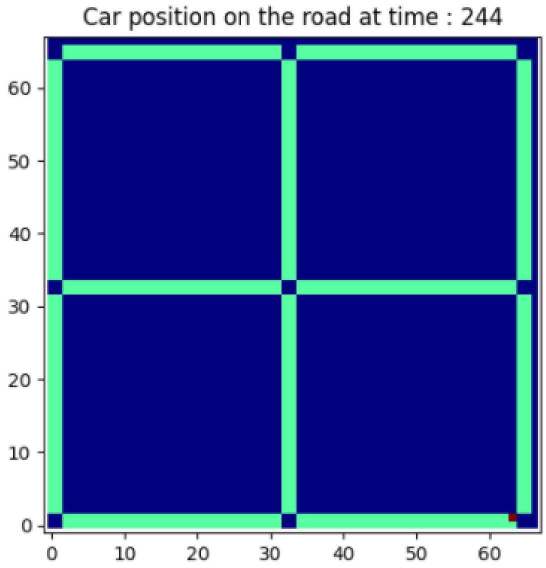
At time t = 126, the car 2 has reached at the junction. At t = 128, car 1 has turned left. This indicated by white arrows in console output.

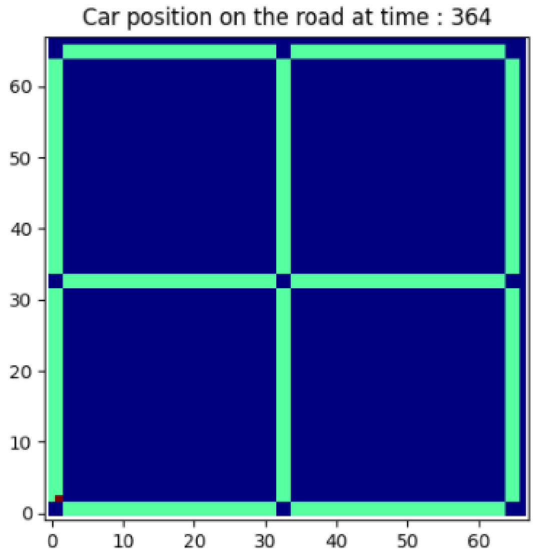
### 8. *Verify the car is goes to each node A, B, C and D and returns*

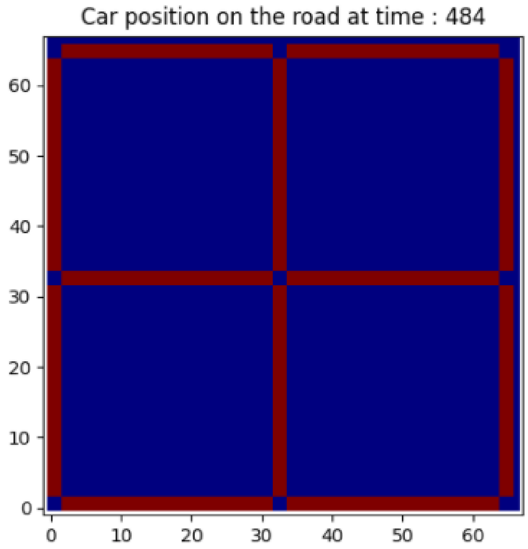


```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
13,1,-1,-1,-1,0,2,4,6,8,10,12,14
Time:124
13,1,4,0,7,0,2,4,6,8,10,12,14,
Time:244
13,1,8,0,9,0,2,4,6,8,10,12,14,
Time:364
13,1,12,0,11,0,2,4,6,8,10,12,14,
Time:484
13,1,30,30,12,0,2,4,6,8,10,12,14,
Time:486
13,1,29,30,12,0,2,4,6,8,10,12,14,
>
```





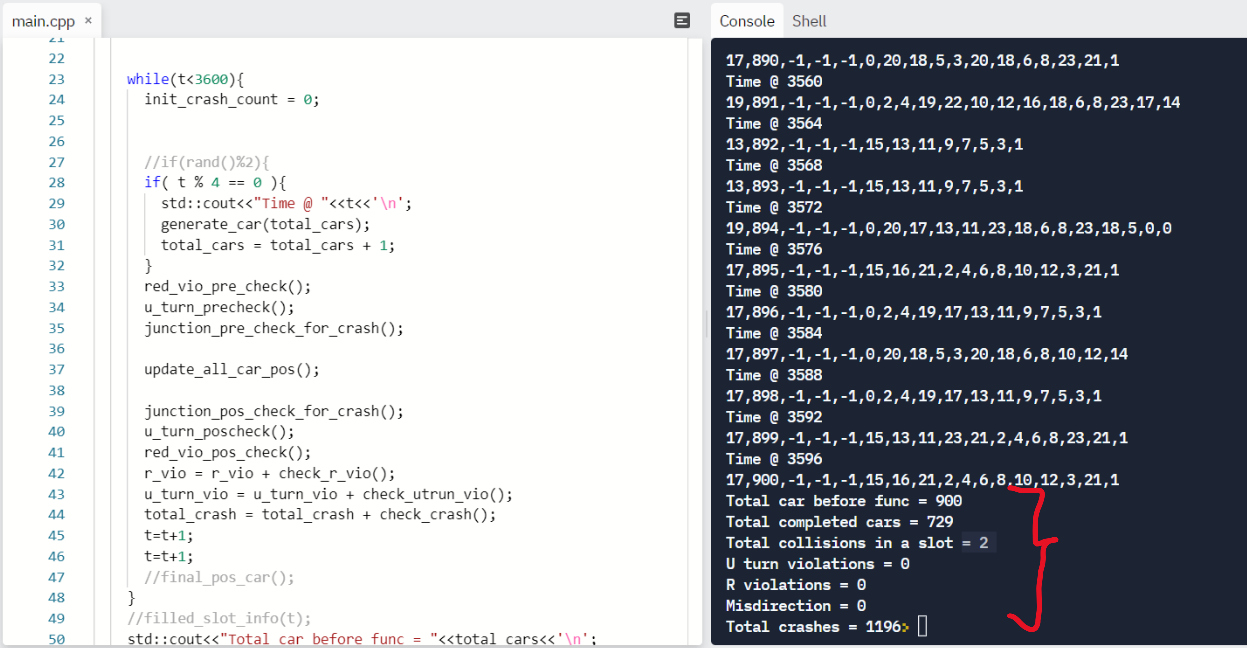




The car 1 has started from node A, then visits node D at t = 124, then visits node C at t = 244 and node B at t = 364. After visiting all the nodes, the car returns again to A at t = 484. In the last plot, there is no car as car has returned.

9. ***Please refer the gif created for demonstration of this testcase:*** https://github.com/rishyasankar/Formal_verification_project/tree/main/Phase1-V-submission

The entire simulation was run keeping all the lights green, to check if our violations are being detected correctly. As seen from the snapshot below (indicated by the red annotation), the total number of cars generated were 900 and 729 of the cars returned back to A. The total collisions within a slot has happened 2 times and it is due to 2 cars completing/returning to A at the same time instance. The U-turn violations were 0 as all the cars were only moving in specified routes as indicated in the problem statement and not taking a U-turn. Since all the signals were set to green, the number of red light violations is as expected – 0. Similarly, the number of cars running in misdirection is also 0. But since all the lights are green, crashes are expected to happen at intersections at different time instances and we can see that there are 1196 crashes reported. By this we can confirm that the number of crashes detecting function is working as expected.

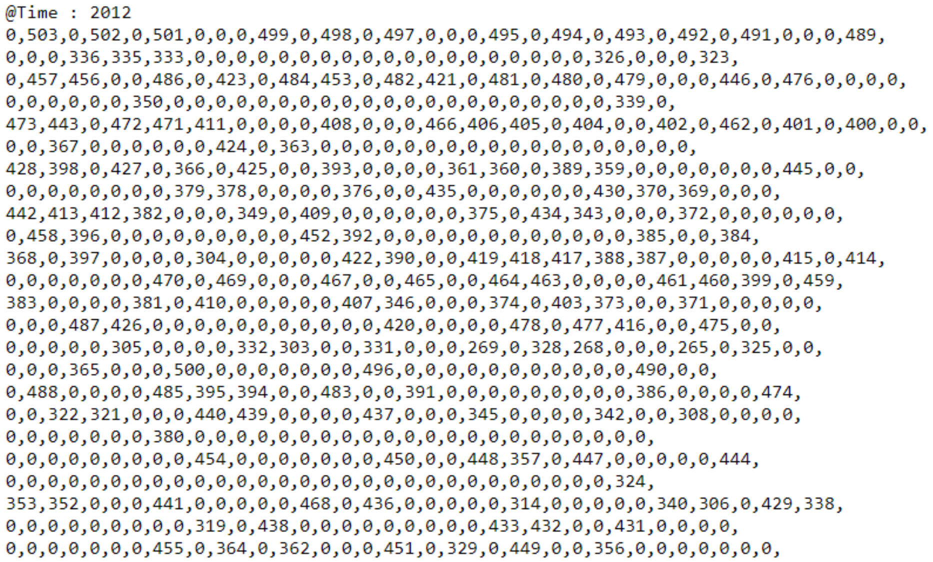


```cpp

    while(t<3600){
      init_crash_count = 0;


      //if(rand()%2){
      if( t % 4 == 0 ){
        std::cout<<"Time @ "<<t<<'\n';
        generate_car(total_cars);
        total_cars = total_cars + 1;
      }
      red_vio_pre_check();
      u_turn_precheck();
      junction_pre_check_for_crash();

      update_all_car_pos();

      junction_pos_check_for_crash();
      u_turn_poscheck();
      red_vio_pos_check();
      r_vio = r_vio + check_r_vio();
      u_turn_vio = u_turn_vio + check_utrun_vio();
      total_crash = total_crash + check_crash();
      t=t+1;
      t=t+1;
      //final_pos_car();
    }
    //filled_slot_info(t);
    std::cout<<"Total car before func = "<<total_cars<<'\n';
```

```
17,890,-1,-1,-1,0,20,18,5,3,20,18,6,8,23,21,1
Time @ 3560
19,891,-1,-1,-1,0,2,4,19,22,10,12,16,18,6,8,23,17,14
Time @ 3564
13,892,-1,-1,-1,15,13,11,9,7,5,3,1
Time @ 3568
13,893,-1,-1,-1,15,13,11,9,7,5,3,1
Time @ 3572
19,894,-1,-1,-1,0,20,17,13,11,23,18,6,8,23,18,5,0,0
Time @ 3576
17,895,-1,-1,-1,15,16,21,2,4,6,8,10,12,3,21,1
Time @ 3580
17,896,-1,-1,-1,0,2,4,19,17,13,11,9,7,5,3,1
Time @ 3584
17,897,-1,-1,-1,0,20,18,5,3,20,18,6,8,10,12,14
Time @ 3588
17,898,-1,-1,-1,0,2,4,19,17,13,11,9,7,5,3,1
Time @ 3592
17,899,-1,-1,-1,15,13,11,23,21,2,4,6,8,23,21,1
Time @ 3596
17,900,-1,-1,-1,15,16,21,2,4,6,8,10,12,3,21,1
Total car before func = 900
Total completed cars = 729
Total collisions in a slot = 2
U turn violations = 0
R violations = 0
Misdirection = 0
Total crashes = 1196>
```

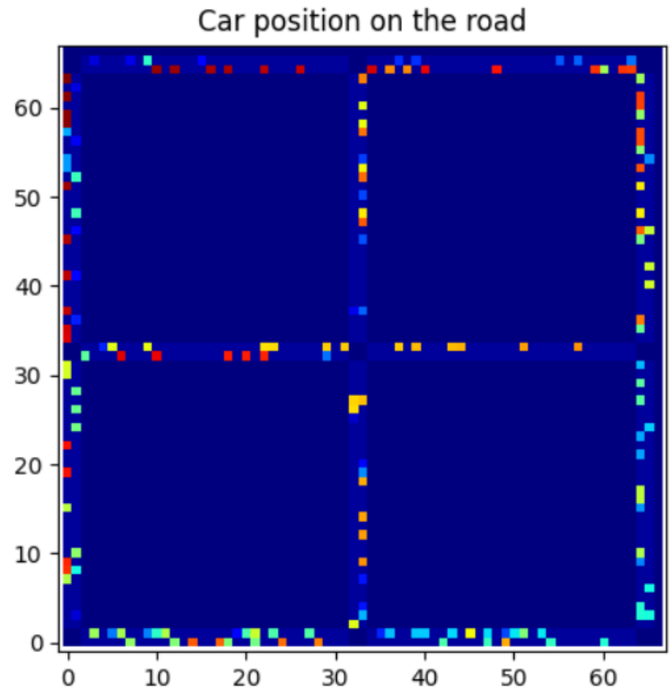
### *10. Traffic Congestion*

The traffic congestion information is printed out as follows: Each line represents a segment and there are 30 elements. A non zero element indicates that the slot in the segment is occupied by that particular car with id = non zero number.

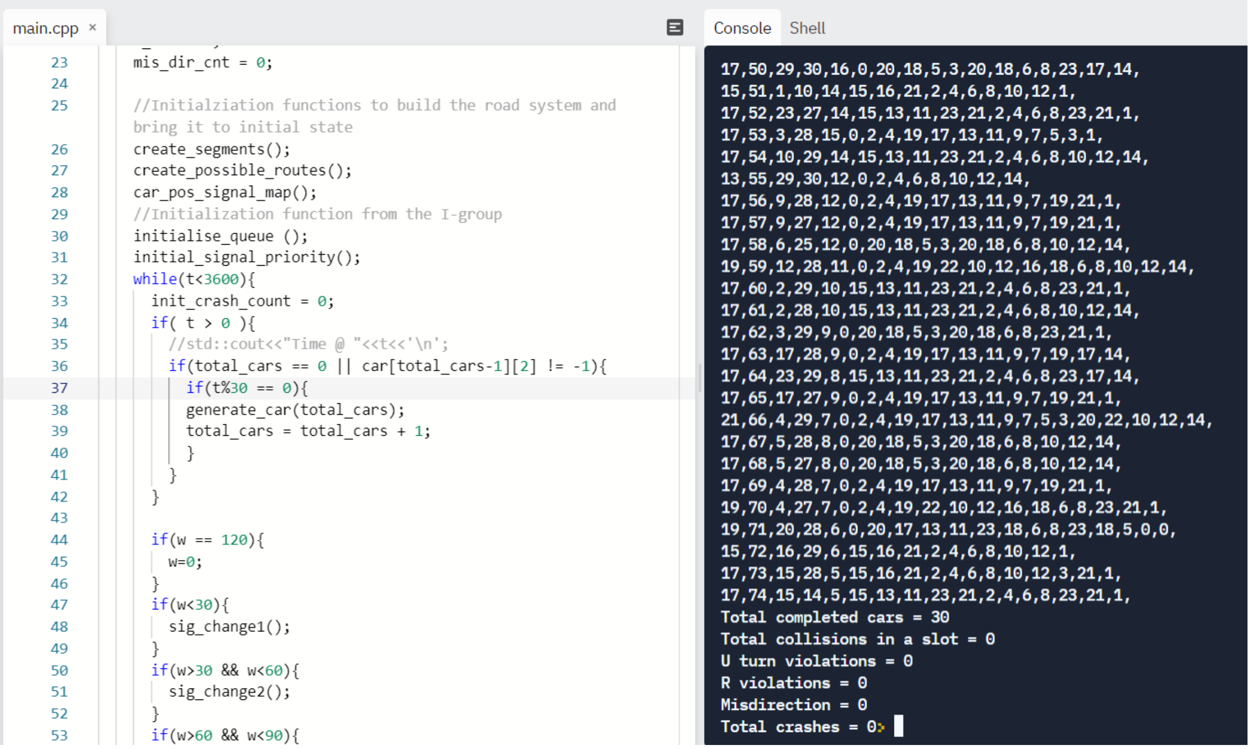


```
@Time : 2012
0,503,0,502,0,501,0,0,0,499,0,498,0,497,0,0,0,495,0,494,0,493,0,492,0,491,0,0,0,489,
0,0,0,336,335,333,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,326,0,0,0,323,
0,457,456,0,0,486,0,423,0,484,453,0,482,421,0,481,0,480,0,479,0,0,0,446,0,476,0,0,0,0,
0,0,0,0,0,0,350,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,339,0,
473,443,0,472,471,411,0,0,0,0,408,0,0,0,466,406,405,0,404,0,0,402,0,462,0,401,0,400,0,0,
0,0,367,0,0,0,0,0,0,424,0,363,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
428,398,0,427,0,366,0,425,0,0,393,0,0,0,0,361,360,0,389,359,0,0,0,0,0,0,0,445,0,0,
0,0,0,0,0,0,0,0,379,378,0,0,0,0,376,0,0,435,0,0,0,0,0,0,430,370,369,0,0,0,
442,413,412,382,0,0,0,349,0,409,0,0,0,0,0,0,375,0,434,343,0,0,0,372,0,0,0,0,0,0,
0,458,396,0,0,0,0,0,0,0,0,0,452,392,0,0,0,0,0,0,0,0,0,0,0,0,385,0,0,384,
368,0,397,0,0,0,0,304,0,0,0,0,0,422,390,0,0,419,418,417,388,387,0,0,0,0,0,415,0,414,
0,0,0,0,0,0,0,470,0,469,0,0,0,467,0,0,465,0,0,464,463,0,0,0,0,461,460,399,0,459,
383,0,0,0,0,381,0,410,0,0,0,0,0,407,346,0,0,0,374,0,403,373,0,0,371,0,0,0,0,0,
0,0,0,487,426,0,0,0,0,0,0,0,0,0,0,0,420,0,0,0,0,478,0,477,416,0,0,475,0,0,
0,0,0,0,0,305,0,0,0,0,332,303,0,0,331,0,0,0,269,0,328,268,0,0,0,265,0,325,0,0,
0,0,0,365,0,0,0,500,0,0,0,0,0,0,0,496,0,0,0,0,0,0,0,0,0,0,0,490,0,0,
0,488,0,0,0,0,485,395,394,0,0,483,0,0,391,0,0,0,0,0,0,0,0,0,386,0,0,0,0,474,
0,0,322,321,0,0,0,440,439,0,0,0,0,437,0,0,0,345,0,0,0,0,342,0,0,308,0,0,0,0,
0,0,0,0,0,0,0,380,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,454,0,0,0,0,0,0,0,450,0,0,448,357,0,447,0,0,0,0,0,444,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,324,
353,352,0,0,0,441,0,0,0,0,0,468,0,436,0,0,0,0,0,314,0,0,0,0,0,340,306,0,429,338,
0,0,0,0,0,0,0,0,0,319,0,438,0,0,0,0,0,0,0,0,0,433,432,0,0,431,0,0,0,0,
0,0,0,0,0,0,0,455,0,364,0,362,0,0,0,451,0,329,0,449,0,0,356,0,0,0,0,0,0,0,
```

This output was processed and mapped into a matrix for a plot using python and the congestion map looked like the figure shown below:

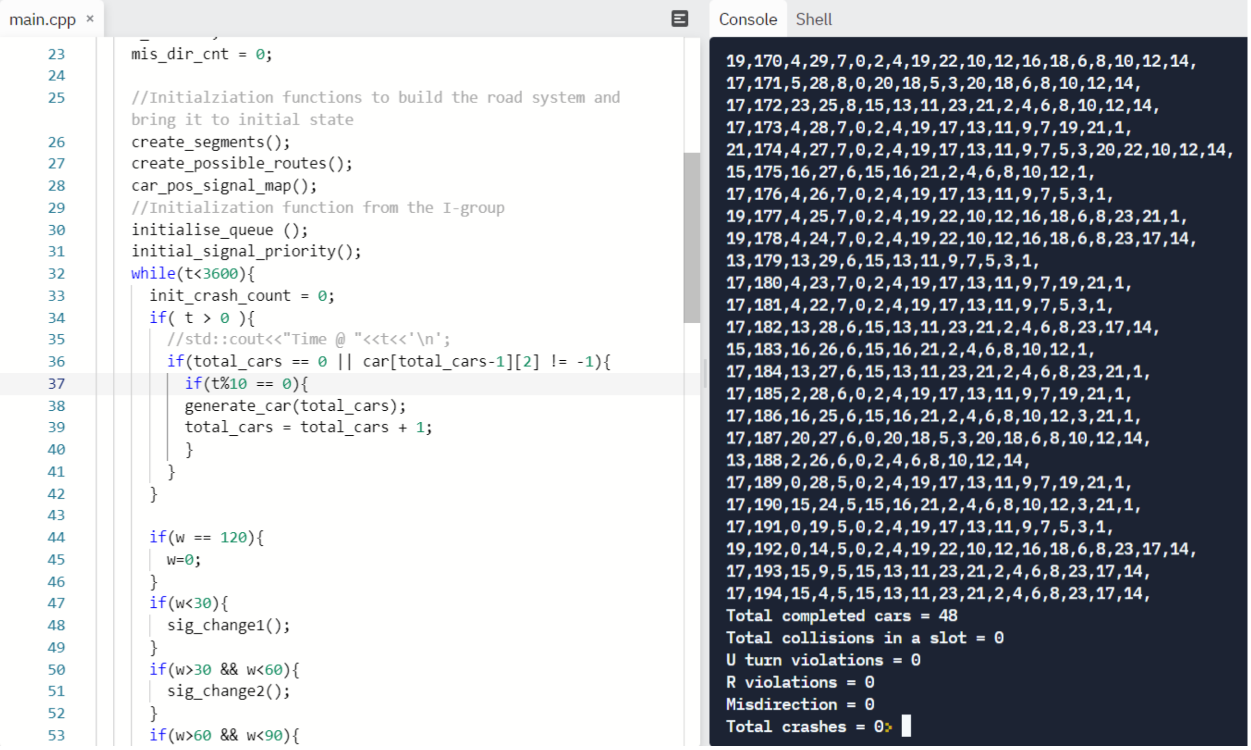


## 11. *Verification Results*



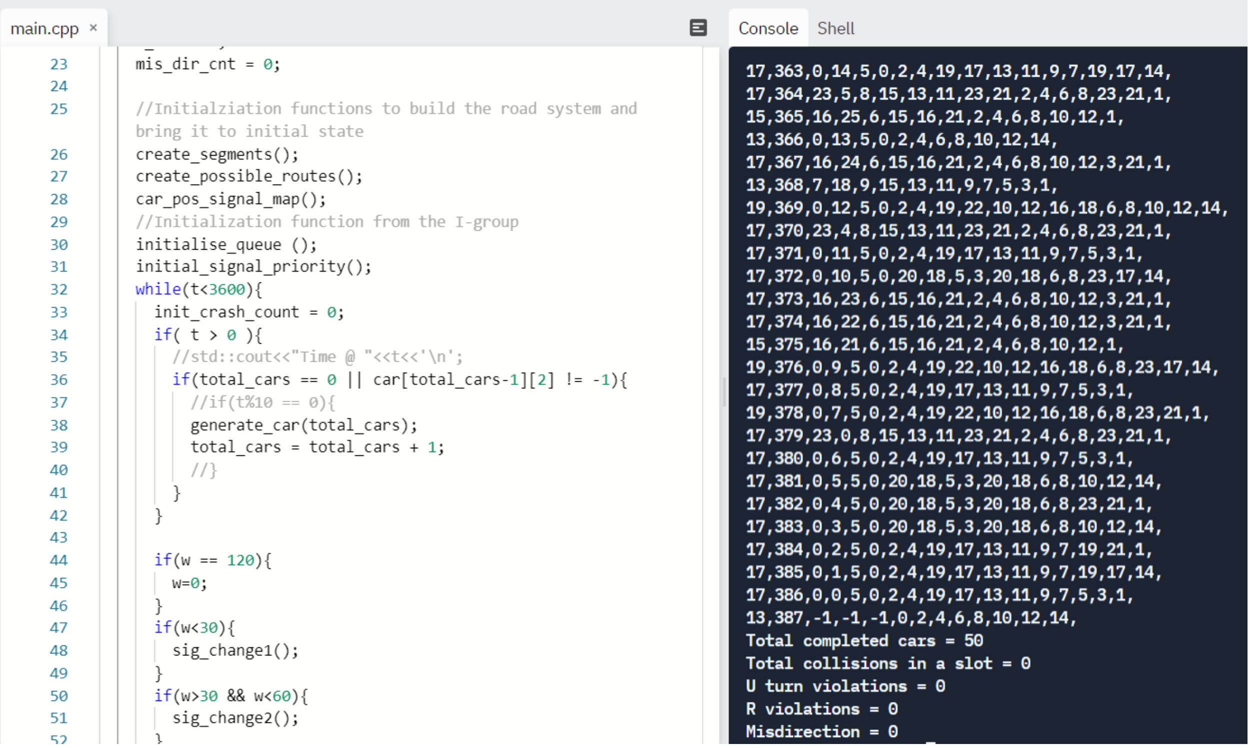
```cpp
mis_dir_cnt = 0;

//Initialzation functions to build the road system and bring it to initial state
create_segments();
create_possible_routes();
car_pos_signal_map();
//Initialization function from the I-group
initialise_queue ();
initial_signal_priority();
while(t<3600){
  init_crash_count = 0;
  if( t > 0 ){
    //std::cout<<"Time @ "<<t<<'\n';
    if(total_cars == 0 || car[total_cars-1][2] != -1){
      if(t%30 == 0){
      generate_car(total_cars);
      total_cars = total_cars + 1;
      }
    }
  }

  if(w == 120){
    w=0;
  }
  if(w<30){
    sig_change1();
  }
  if(w>30 && w<60){
    sig_change2();
  }
  if(w>60 && w<90){
```

```
17,50,29,30,16,0,20,18,5,3,20,18,6,8,23,17,14,
15,51,1,10,14,15,16,21,2,4,6,8,10,12,1,
17,52,23,27,14,15,13,11,23,21,2,4,6,8,23,21,1,
17,53,3,28,15,0,2,4,19,17,13,11,9,7,5,3,1,
17,54,10,29,14,15,13,11,23,21,2,4,6,8,10,12,14,
13,55,29,30,12,0,2,4,6,8,10,12,14,
17,56,9,28,12,0,2,4,19,17,13,11,9,7,19,21,1,
17,57,9,27,12,0,2,4,19,17,13,11,9,7,19,21,1,
17,58,6,25,12,0,20,18,5,3,20,18,6,8,10,12,14,
19,59,12,28,11,0,2,4,19,22,10,12,16,18,6,8,10,12,14,
17,60,2,29,10,15,13,11,23,21,2,4,6,8,23,21,1,
17,61,2,28,10,15,13,11,23,21,2,4,6,8,10,12,14,
17,62,3,29,9,0,20,18,5,3,20,18,6,8,23,21,1,
17,63,17,28,9,0,2,4,19,17,13,11,9,7,19,17,14,
17,64,23,29,8,15,13,11,23,21,2,4,6,8,23,17,14,
17,65,17,27,9,0,2,4,19,17,13,11,9,7,19,21,1,
21,66,4,29,7,0,2,4,19,17,13,11,9,7,5,3,20,22,10,12,14,
17,67,5,28,8,0,20,18,5,3,20,18,6,8,10,12,14,
17,68,5,27,8,0,20,18,5,3,20,18,6,8,10,12,14,
17,69,4,28,7,0,2,4,19,17,13,11,9,7,19,21,1,
19,70,4,27,7,0,2,4,19,22,10,12,16,18,6,8,23,21,1,
19,71,20,28,6,0,20,17,13,11,23,18,6,8,23,18,5,0,0,
15,72,16,29,6,15,16,21,2,4,6,8,10,12,1,
17,73,15,28,5,15,16,21,2,4,6,8,10,12,3,21,1,
17,74,15,14,5,15,13,11,23,21,2,4,6,8,23,21,1,
Total completed cars = 30
Total collisions in a slot = 0
U turn violations = 0
R violations = 0
Misdirection = 0
Total crashes = 0> 
```



```cpp
mis_dir_cnt = 0;

//Initialzation functions to build the road system and bring it to initial state
create_segments();
create_possible_routes();
car_pos_signal_map();
//Initialization function from the I-group
initialise_queue ();
initial_signal_priority();
while(t<3600){
  init_crash_count = 0;
  if( t > 0 ){
    //std::cout<<"Time @ "<<t<<'\n';
    if(total_cars == 0 || car[total_cars-1][2] != -1){
      if(t%10 == 0){
      generate_car(total_cars);
      total_cars = total_cars + 1;
      }
    }
  }

  if(w == 120){
    w=0;
  }
  if(w<30){
    sig_change1();
  }
  if(w>30 && w<60){
    sig_change2();
  }
  if(w>60 && w<90){
```

```
19,170,4,29,7,0,2,4,19,22,10,12,16,18,6,8,10,12,14,
17,171,5,28,8,0,20,18,5,3,20,18,6,8,10,12,14,
17,172,23,25,8,15,13,11,23,21,2,4,6,8,10,12,14,
17,173,4,28,7,0,2,4,19,17,13,11,9,7,19,21,1,
21,174,4,27,7,0,2,4,19,17,13,11,9,7,5,3,20,22,10,12,14,
15,175,16,27,6,15,16,21,2,4,6,8,10,12,1,
17,176,4,26,7,0,2,4,19,17,13,11,9,7,5,3,1,
19,177,4,25,7,0,2,4,19,22,10,12,16,18,6,8,23,21,1,
19,178,4,24,7,0,2,4,19,22,10,12,16,18,6,8,23,17,14,
13,179,13,29,6,15,13,11,9,7,5,3,1,
17,180,4,23,7,0,2,4,19,17,13,11,9,7,19,21,1,
17,181,4,22,7,0,2,4,19,17,13,11,9,7,5,3,1,
17,182,13,28,6,15,13,11,23,21,2,4,6,8,23,17,14,
15,183,16,26,6,15,16,21,2,4,6,8,10,12,1,
17,184,13,27,6,15,13,11,23,21,2,4,6,8,23,21,1,
17,185,2,28,6,0,2,4,19,17,13,11,9,7,19,21,1,
17,186,16,25,6,15,16,21,2,4,6,8,10,12,3,21,1,
17,187,20,27,6,0,20,18,5,3,20,18,6,8,10,12,14,
13,188,2,26,6,0,2,4,6,8,10,12,14,
17,189,0,28,5,0,2,4,19,17,13,11,9,7,19,21,1,
17,190,15,24,5,15,16,21,2,4,6,8,10,12,3,21,1,
17,191,0,19,5,0,2,4,19,17,13,11,9,7,5,3,1,
19,192,0,14,5,0,2,4,19,22,10,12,16,18,6,8,23,17,14,
17,193,15,9,5,15,13,11,23,21,2,4,6,8,23,17,14,
17,194,15,4,5,15,13,11,23,21,2,4,6,8,23,17,14,
Total completed cars = 48
Total collisions in a slot = 0
U turn violations = 0
R violations = 0
Misdirection = 0
Total crashes = 0> 
```



```cpp
mis_dir_cnt = 0;

//Initialziation functions to build the road system and bring it to initial state
create_segments();
create_possible_routes();
car_pos_signal_map();
//Initialization function from the I-group
initialise_queue ();
initial_signal_priority();
while(t<3600){
  init_crash_count = 0;
  if( t > 0 ){
    //std::cout<<"Time @ "<<t<<'\n';
    if(total_cars == 0 || car[total_cars-1][2] != -1){
      //if(t%10 == 0){
      generate_car(total_cars);
      total_cars = total_cars + 1;
      //}
    }
  }

  if(w == 120){
    w=0;
  }
  if(w<30){
    sig_change1();
  }
  if(w>30 && w<60){
    sig_change2();
  }
```

```
17,363,0,14,5,0,2,4,19,17,13,11,9,7,19,17,14,
17,364,23,5,8,15,13,11,23,21,2,4,6,8,23,21,1,
15,365,16,25,6,15,16,21,2,4,6,8,10,12,1,
13,366,0,13,5,0,2,4,6,8,10,12,14,
17,367,16,24,6,15,16,21,2,4,6,8,10,12,3,21,1,
13,368,7,18,9,15,13,11,9,7,5,3,1,
19,369,0,12,5,0,2,4,19,22,10,12,16,18,6,8,10,12,14,
17,370,23,4,8,15,13,11,23,21,2,4,6,8,23,21,1,
17,371,0,11,5,0,2,4,19,17,13,11,9,7,5,3,1,
17,372,0,10,5,0,20,18,5,3,20,18,6,8,23,17,14,
17,373,16,23,6,15,16,21,2,4,6,8,10,12,3,21,1,
17,374,16,22,6,15,16,21,2,4,6,8,10,12,3,21,1,
15,375,16,21,6,15,16,21,2,4,6,8,10,12,1,
19,376,0,9,5,0,2,4,19,22,10,12,16,18,6,8,23,17,14,
17,377,0,8,5,0,2,4,19,17,13,11,9,7,5,3,1,
19,378,0,7,5,0,2,4,19,22,10,12,16,18,6,8,23,21,1,
17,379,23,0,8,15,13,11,23,21,2,4,6,8,23,21,1,
17,380,0,6,5,0,2,4,19,17,13,11,9,7,5,3,1,
17,381,0,5,5,0,20,18,5,3,20,18,6,8,10,12,14,
17,382,0,4,5,0,20,18,5,3,20,18,6,8,23,21,1,
17,383,0,3,5,0,20,18,5,3,20,18,6,8,10,12,14,
17,384,0,2,5,0,2,4,19,17,13,11,9,7,19,21,1,
17,385,0,1,5,0,2,4,19,17,13,11,9,7,19,17,14,
17,386,0,0,5,0,2,4,19,17,13,11,9,7,5,3,1,
13,387,-1,-1,-1,0,2,4,6,8,10,12,14,
Total completed cars = 50
Total collisions in a slot = 0
U turn violations = 0
R violations = 0
Misdirection = 0
```

Based on random signal switching implemented by V-group (not efficient), we generated different number of cars and ran the entire simulation for 60 mins. From the screenshots above, we can see that we were getting 30-50% of the cars returning back when the number of cars generated is below 200. Since we are not changing the switching mechanism, even after increasing the number of cars generated, the number of cars reaching the destination is saturated.

# LINK TO SOURCE CODE, OUTPUTS, PLOTS AND GIFS

https://github.com/rishyasankar/Formal_verification_project/tree/main/Phase1-V-submission

# APPENDIX

# SOURCE CODE

*Segment.h:*

```
//Global variables
 //Time variable
int t;
//Contains empty slots as 0 and filled slots with Carid
int segment[24][30];

//Import from IV team
#define NUM_JUNCTIONS 9
#define NUM_SIGNALS 4

int il[9][4] = {0};
std::vector<std::queue<int>> q(9);




//Possible routes variable
int *route[19];
//Cars generation variable
int *car[900];
//Car generation decision variable
bool n;
//total number of cars
unsigned int total_cars;
//total completed cars
int total_completed_cars;
//remaining cars
int rem_cars;
//Total crash count
int total_crash;
//variable to indicate crash at junction 0, when two cars try to complete at A in the same time
int init_crash_count;
//variable to indicate initial box where car is generated and returned // pos is assumed as -1,-1
int init;
//variable to count no of cars running in wri=ong direction
int mis_dir_cnt;
//Variable to store signal to segment and car pos
int signal[24][2];
int arr[19] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18};
int freq[19] = {5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5};
int ii, nn = sizeof(arr) / sizeof(arr[0]);




//Create segments needed
//24 segments - 12(total) sides * 2 ways
void create_segments(){
  for(int j=0;j<24;j++){
    for(int i=0;i<30;i++){
      segment[j][i] = 0;
    }
  }
}

//Hashmap for road signal and car pos
void car_pos_signal_map(){
  signal[0][0] = 1 ; signal[0][1] = 1;
  signal[1][0] = 0 ; signal[1][1] = 3;
  signal[2][0] = 2 ; signal[2][1] = 1;
  signal[3][0] = 1 ; signal[3][1] = 3;
  signal[4][0] = 5 ; signal[4][1] = 2;
  signal[5][0] = 2 ; signal[5][1] = 0;
  signal[6][0] = 8 ; signal[6][1] = 2;
  signal[7][0] = 5 ; signal[7][1] = 0;
  signal[8][0] = 7 ; signal[8][1] = 3;
  signal[9][0] = 8 ; signal[9][1] = 1;
  signal[10][0] = 6 ; signal[10][1] = 3;
  signal[11][0] = 7 ; signal[11][1] = 1;
  signal[12][0] = 3 ; signal[12][1] = 0;
  signal[13][0] = 6 ; signal[13][1] = 2;
  signal[14][0] = 0 ; signal[14][1] = 0;
  signal[15][0] = 3 ; signal[15][1] = 2;
  signal[16][0] = 4 ; signal[16][1] = 1;
  signal[17][0] = 3 ; signal[17][1] = 3;
  signal[18][0] = 5 ; signal[18][1] = 1;
  signal[19][0] = 4 ; signal[19][1] = 3;
  signal[20][0] = 4 ; signal[20][1] = 2;
  signal[21][0] = 1 ; signal[21][1] = 0;
  signal[22][0] = 7 ; signal[22][1] = 2;
  signal[23][0] = 4 ; signal[23][1] = 0;

}

//Print out the filled slot info
void filled_slot_info (int time){
  std::cout<<"@Time : "<<time<<"\n";
  for(int j=0;j<24;j++){
    for(int i=0;i<30;i++){
      std::cout<<segment[j][i]<<",";
    }
    std::cout<<"\n";
  }

}

//List of possible legal routes to pick from
void create_possible_routes(){
  route[0] = new int[9];

route[0][0]=8;route[0][1]=0;route[0][2]=2;route[0][3]=4;route[0][4]=6;route[0][5]
=8;route[0][6]=10;route[0][7]=12;route[0][8]=14;

  route[1] = new int[13];

route[1][0]=12;route[1][1]=15;route[1][2]=13;route[1][3]=11;route[1][4]=23;route[
1][5]=21;route[1][6]=2;route[1][7]=4;route[1][8]=6;route[1][9]=8;route[1][10]=10;
route[1][11]=12;route[1][12]=14;

  route[2] = new int[9];

route[2][0]=8;route[2][1]=15;route[2][2]=13;route[2][3]=11;route[2][4]=9;route[2]
[5]=7;route[2][6]=5;route[2][7]=3;route[2][8]=1;

  route[3] = new int[13];

route[3][0]=12;route[3][1]=15;route[3][2]=13;route[3][3]=11;route[3][4]=23;route[
3][5]=21;route[3][6]=2;route[3][7]=4;route[3][8]=6;route[3][9]=8;route[3][10]=23;
route[3][11]=21;route[3][12]=1;

  route[4] = new int[13];

route[4][0]=12;route[4][1]=15;route[4][2]=13;route[4][3]=11;route[4][4]=23;route[
4][5]=21;route[4][6]=2;route[4][7]=4;route[4][8]=6;route[4][9]=8;route[4][10]=23;
route[4][11]=17;route[4][12]=14;
```

```
  route[5] = new int[13];

route[5][0]=12;route[5][1]=15;route[5][2]=13;route[5][3]=11;route[5][4]=23;route[
5][5]=21;route[5][6]=2;route[5][7]=4;route[5][8]=6;route[5][9]=8;route[5][10]=23;
route[5][11]=21;route[5][12]=1;

  route[6] = new int[15];

route[6][0]=14;route[6][1]=0;route[6][2]=2;route[6][3]=4;route[6][4]=19;route[6][
5]=22;route[6][6]=10;route[6][7]=12;route[6][8]=16;route[6][9]=18;route[6][10]=6;
route[6][11]=8;route[6][12]=23;route[6][13]=17;route[6][14]=14;

  route[7] = new int[15];

route[7][0]=14;route[7][1]=0;route[7][2]=2;route[7][3]=4;route[7][4]=19;route[7][
5]=22;route[7][6]=10;route[7][7]=12;route[7][8]=16;route[7][9]=18;route[7][10]=6;
route[7][11]=8;route[7][12]=23;route[7][13]=21;route[7][14]=1;

  route[8] = new int[15];

route[8][0]=14;route[8][1]=0;route[8][2]=2;route[8][3]=4;route[8][4]=19;route[8][
5]=22;route[8][6]=10;route[8][7]=12;route[8][8]=16;route[8][9]=18;route[8][10]=6;
route[8][11]=8;route[8][12]=10;route[8][13]=12;route[8][14]=14;

  route[9] = new int[13];

route[9][0]=12;route[9][1]=0;route[9][2]=2;route[9][3]=4;route[9][4]=19;route[9][
5]=17;route[9][6]=13;route[9][7]=11;route[9][8]=9;route[9][9]=7;route[9][10]=5;ro
ute[9][11]=3;route[9][12]=1;

  route[10] = new int[13];

route[10][0]=12;route[10][1]=0;route[10][2]=2;route[10][3]=4;route[10][4]=19;rout
e[10][5]=17;route[10][6]=13;route[10][7]=11;route[10][8]=9;route[10][9]=7;route[1
0][10]=19;route[10][11]=21;route[10][12]=1;

  route[11] = new int[13];

route[11][0]=12;route[11][1]=0;route[11][2]=2;route[11][3]=4;route[11][4]=19;rout
e[11][5]=17;route[11][6]=13;route[11][7]=11;route[11][8]=9;route[11][9]=7;route[1
1][10]=19;route[11][11]=17;route[11][12]=14;

  route[12] = new int[17];
```

```
route[12][0]=16;route[12][1]=0;route[12][2]=2;route[12][3]=4;route[12][4]=19;rout
e[12][5]=17;route[12][6]=13;route[12][7]=11;route[12][8]=9;route[12][9]=7;route[1
2][10]=5;route[12][11]=3;route[12][12]=20;route[12][13]=22;route[12][14]=10;route
[12][15]=12;route[12][16]=14;

  route[13] = new int[13];

route[13][0]=12;route[13][1]=15;route[13][2]=16;route[13][3]=21;route[13][4]=2;ro
ute[13][5]=4;route[13][6]=6;route[13][7]=8;route[13][8]=10;route[13][9]=12;route[
13][10]=16;route[13][11]=21;route[13][12]=1;

  route[14] = new int[11];

route[14][0]=10;route[14][1]=15;route[14][2]=16;route[14][3]=21;route[14][4]=2;ro
ute[14][5]=4;route[14][6]=6;route[14][7]=8;route[14][8]=10;route[14][9]=12;route[
14][10]=15;

  route[15] = new int[15];

route[15][0]=14;route[15][1]=0;route[15][2]=20;route[15][3]=17;route[15][4]=13;ro
ute[15][5]=11;route[15][6]=23;route[15][7]=18;route[15][8]=6;route[15][9]=8;route
[15][10]=23;route[15][11]=18;route[15][12]=5;route[13][10]=3;route[14][10]=1;

  route[16] = new int[13];

route[16][0]=12;route[16][1]=0;route[16][2]=20;route[16][3]=18;route[16][4]=5;rou
te[16][5]=3;route[16][6]=20;route[16][7]=18;route[16][8]=6;route[16][9]=8;route[1
6][10]=10;route[16][11]=12;route[16][12]=14;

  route[17] = new int[13];

route[17][0]=12;route[17][1]=0;route[17][2]=20;route[17][3]=18;route[17][4]=5;rou
te[17][5]=3;route[17][6]=20;route[17][7]=18;route[17][8]=6;route[17][9]=8;route[1
7][10]=23;route[17][11]=21;route[17][12]=1;

  route[18] = new int[13];

route[18][0]=12;route[18][1]=0;route[18][2]=20;route[18][3]=18;route[18][4]=5;rou
te[18][5]=3;route[18][6]=20;route[18][7]=18;route[18][8]=6;route[18][9]=8;route[1
8][10]=23;route[18][11]=17;route[18][12]=14;

}

int findCeil(int arr[], int r, int l, int h)
```

```
{
    int mid;
    while (l < h)
    {
        mid = l + ((h - l) >> 1); // Same as mid = (l+h)/2
        (r > arr[mid]) ? (l = mid + 1) : (h = mid);
    }
    return (arr[l] >= r) ? l : -1;
}


// The main function that returns a random number
// from arr[] according to distribution array
// defined by freq[]. n is size of arrays.
int myRand(int arr[], int freq[], int n)
{
    // Create and fill prefix array
    int prefix[n], i;
    prefix[0] = freq[0];
    for (i = 1; i < n; ++i)
        prefix[i] = prefix[i - 1] + freq[i];

    // prefix[n-1] is sum of all frequencies.
    // Generate a random number with
    // value from 1 to this sum
    int r = (rand() % prefix[n - 1]) + 1;

    // Find index of ceiling of r in prefix array
    int indexc = findCeil(prefix, r, 0, n - 1);
    return arr[indexc];
}



void generate_car(int id){
  // An array containing [array_len,id,segment,pos in segment,list of segments as
routes]
  //srand ( time(NULL) ); //initialize the random seed
  int RandIndex = myRand(arr, freq, nn);
  //rand() % 15; //std::cout<<"Rand index :"<< RandIndex<<'\n';

  car[total_cars] = new int[5+route[RandIndex][0]];
  car[total_cars][0] = 5+route[RandIndex][0];
std::cout<<car[total_cars][0]<<',';//Length of each car array
```

```
  car[total_cars][1] = total_cars+1;    std::cout<<car[total_cars][1]<<','; //
This stores the car_id
  car[total_cars][2] = -1;car[total_cars][3] = -1;
std::cout<<car[total_cars][2]<<','<<car[total_cars][3]; // The index [2] stores
which segment and [3] stores position in the segment
  car[total_cars][4] = -1; std::cout<<','<<car[total_cars][4];// Stores current
segment information
  init = car[total_cars][1];
  for(int i=0;i<route[RandIndex][0];i++){
    car[total_cars][i+5] = route[RandIndex][i+1];
    std::cout<<','<<car[total_cars][i+5]; //Rest of the array stores sequence of
segment numbers which is the route.
  }
  std::cout<<"\n";
}


void update_car(int id){

  int car_array_len = car[id][0];
  int car_route_len = car[id][0]-5;
  int y_pre,y_pos;

  y_pre = car[id][3];
  if(car[id][2] == -1 && car[id][3] == -1 && car[id][4] == -1){
    if (segment[car[id][5]][0] == 0 && (il[0][1] == 2)){
      //segment[car[id][2]][car[id][3]]=0;
      car[id][4] = 5; car[id][2] = car[id][car[id][4]]; car[id][3] =
0;segment[car[id][2]][car[id][3]]=car[id][1];
    }
  }
  else {
    if( (segment[car[id][2]][car[id][3]+1] == 0) && car[id][3] < 29){
      segment[car[id][2]][car[id][3]]=0;
      car[id][3] = car[id][3]+1;
      segment[car[id][2]][car[id][3]]=car[id][1];
    }
    else
      if (car[id][3] == 29) {
        if(car[id][4] == car_array_len-1){
          if(il[0][0] == 2 || il[0][3] == 2)
          //std::cout<<car[id][2]<<','<<car[id][3]<<'\n';
            segment[car[id][2]][car[id][3]]=0;
            car[id][2] = 30;
            car[id][3] = 30;
            init_crash_count = init_crash_count+1;
        }
        else if (car[id][4] < car_array_len-1){
          if((segment[car[id][car[id][4]+1]][0] == 0) &&
(il[signal[car[id][car[id][4]]][0]][signal[car[id][car[id][4]]][1]] == 2)) {
          segment[car[id][2]][car[id][3]]=0;
          car[id][4] = car[id][4] + 1;
          car[id][2] = car[id][car[id][4]];
          car[id][3] = 0;
          segment[car[id][2]][car[id][3]]=car[id][1];
          }
        }
      }
      else if(car[id][3] == 30 && car[id][2] == 30){
        total_completed_cars = total_completed_cars+1;
        car[id][2] = 29;
        //delete car[id];
      }
    }

    y_pos = car[id][3];
    if(y_pre > y_pos && y_pre != 29){
      mis_dir_cnt = mis_dir_cnt + 1;
    }
}

void print_car_info(int b){
  if(car[b][2] != 29 && car[b][3] != 30){
    for(int z=0;z<car[b][0];z++){
      //std::cout<<car[b][z]<<",";
    }
  //std::cout<<'\n';
  }
}


void update_all_car_pos(){
  for(int l=0;l<total_cars;l++){

    update_car(l);
    //print_car_info(l);
  }
}
```

```
void clear(){

  for(int j=0;j<19;j++){
    delete route[j];
  }

  for(int i=0; i< total_cars ; i++){
    //if(car[i][2] != 29 && car[i][3] == 30){
      delete car[i];
    //}
  }

}

void final_pos_car(){
  //std::cout<<"Total cars = "<<total_cars<<'\n';
  for(int l=0;l<total_cars;l++){
    //if(car[l][2] != 29 && car[l][3] == 30){
      int length_car = car[l][0];
      for(int m=0;m<length_car;m++){
        std::cout<<car[l][m]<<',';
      }
      std::cout<<'\n';
   //}
  }
}
```

*Crash.h:*

```
//Junction check for crash variables
int junction0_prech[2];
int junction0_posch[2];
int junction1_prech[6];
int junction1_posch[6];
int junction2_prech[2];
int junction2_posch[2];
int junction3_prech[6];
int junction3_posch[6];
int junction4_prech[12];
int junction4_posch[12];
int junction5_prech[6];
int junction5_posch[6];
int junction6_prech[2];
int junction6_posch[2];
int junction7_prech[6];
int junction7_posch[6];
int junction8_prech[2];
int junction8_posch[2];


//No. of collisions in a slot
int no_col_in_slot;
//no of cars at each intersection
int no_car_intersection[9];

//u-turn precheck
int u_pre_ch[24];
//u-turn poscheck
int u_pos_ch[24];
//no of u turn violations
int u_turn_vio;


//red violation precheck
int r_pre_ch[24];
//red violation poscheck
int r_pos_ch[24];
//number of red violations
int r_vio;


void junction0_precheck(){
  junction0_prech[0]=segment[1][29];
  junction0_prech[1]=segment[14][29];
}

void junction1_precheck(){
  junction1_prech[0]=segment[0][29];
  junction1_prech[1]=segment[0][29];
  junction1_prech[2]=segment[21][29];
  junction1_prech[3]=segment[21][29];
  junction1_prech[4]=segment[3][29];
  junction1_prech[5]=segment[3][29];
}

void junction2_precheck(){
  junction2_prech[0]=segment[2][29];
  junction2_prech[1]=segment[2][29];
}

void junction3_precheck(){
  junction3_prech[0]=segment[15][29];
  junction3_prech[1]=segment[15][29];
  junction3_prech[2]=segment[17][29];
  junction3_prech[3]=segment[17][29];
  junction3_prech[4]=segment[12][29];
  junction3_prech[5]=segment[12][29];
}

void junction4_precheck(){
  junction4_prech[0]=segment[20][29];
  junction4_prech[1]=segment[20][29];
  junction4_prech[2]=segment[20][29];
  junction4_prech[3]=segment[19][29];
  junction4_prech[4]=segment[19][29];
  junction4_prech[5]=segment[19][29];
  junction4_prech[6]=segment[23][29];
  junction4_prech[7]=segment[23][29];
  junction4_prech[8]=segment[23][29];
  junction4_prech[9]=segment[16][29];
  junction4_prech[10]=segment[16][29];
  junction4_prech[11]=segment[16][29];
}



void junction5_precheck(){
  junction5_prech[0]=segment[4][29];
  junction5_prech[1]=segment[4][29];
  junction5_prech[2]=segment[18][29];
  junction5_prech[3]=segment[18][29];
  junction5_prech[4]=segment[7][29];
  junction5_prech[5]=segment[7][29];
}

void junction6_precheck(){
  junction6_prech[0]=segment[13][29];
  junction6_prech[1]=segment[10][29];
}

void junction7_precheck(){
  junction7_prech[0]=segment[22][29];
  junction7_prech[1]=segment[22][29];
  junction7_prech[2]=segment[11][29];
  junction7_prech[3]=segment[11][29];
  junction7_prech[4]=segment[8][29];
  junction7_prech[5]=segment[8][29];
}

void junction8_precheck(){
  junction8_prech[0]=segment[6][29];
  junction8_prech[1]=segment[9][29];
}



void junction0_poscheck(){
  junction0_posch[0]=segment[15][0];
  junction0_posch[1]=segment[0][0];
}

void junction1_poscheck(){
  junction1_posch[0]=segment[2][0];
  junction1_posch[1]=segment[20][0];
  junction1_posch[2]=segment[1][0];
  junction1_posch[3]=segment[2][0];
  junction1_posch[4]=segment[1][0];
  junction1_posch[5]=segment[20][0];
}

void junction2_poscheck(){
  junction2_posch[0]=segment[4][0];
  junction2_posch[1]=segment[3][0];
}

void junction3_poscheck(){
  junction3_posch[0]=segment[16][0];
  junction3_posch[1]=segment[13][0];
  junction3_posch[2]=segment[14][0];
  junction3_posch[3]=segment[13][0];
  junction3_posch[4]=segment[16][0];
  junction3_posch[5]=segment[14][0];
}

void junction4_poscheck(){
  junction4_posch[0]=segment[17][0];
  junction4_posch[1]=segment[18][0];
  junction4_posch[2]=segment[22][0];
  junction4_posch[3]=segment[21][0];
  junction4_posch[4]=segment[17][0];
  junction4_posch[5]=segment[22][0];
  junction4_posch[6]=segment[18][0];
  junction4_posch[7]=segment[17][0];
  junction4_posch[8]=segment[21][0];
  junction4_posch[9]=segment[22][0];
  junction4_posch[10]=segment[18][0];
  junction4_posch[11]=segment[21][0];
}


void junction5_poscheck(){
  junction5_posch[0]=segment[19][0];
  junction5_posch[1]=segment[6][0];
  junction5_posch[2]=segment[6][0];
  junction5_posch[3]=segment[5][0];
  junction5_posch[4]=segment[19][0];
  junction5_posch[5]=segment[5][0];
}

void junction6_poscheck(){
  junction6_posch[0]=segment[11][0];
  junction6_posch[1]=segment[12][0];
}

void junction7_poscheck(){
  junction7_posch[0]=segment[10][0];
  junction7_posch[1]=segment[9][0];
  junction7_posch[2]=segment[9][0];
  junction7_posch[3]=segment[23][0];
  junction7_posch[4]=segment[23][0];
  junction7_posch[5]=segment[10][0];
}

void junction8_poscheck(){
  junction8_posch[0]=segment[8][0];
  junction8_posch[1]=segment[7][0];
}


bool check_crash_j0(){
  int cc = 0;
  no_car_intersection[0] = 0;
  for(int x=0;x<2;x++){
    if(junction0_prech[x] != 0){
      no_car_intersection[0] = no_car_intersection[0]+1;
      if(junction0_prech[x] == junction0_posch[x]){
        cc = cc + 1;
      }
    }
  }
  cc = cc + init_crash_count;
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}


bool check_crash_j1(){
  int cc = 0;
  no_car_intersection[1] = 0;
  for(int x=0;x<6;x++){
    if(junction1_prech[x] != 0){
      no_car_intersection[1] = no_car_intersection[1]+1;
      if(junction1_prech[x] == junction1_posch[x]){
        cc = cc + 1;
      }
    }
  }
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}

bool check_crash_j2(){
  int cc = 0;
  no_car_intersection[2] = 0;
  for(int x=0;x<2;x++){
    if(junction2_prech[x] != 0){
      no_car_intersection[2] = no_car_intersection[2]+1;
      if(junction2_prech[x] == junction2_posch[x]){
        cc = cc + 1;
      }
    }
  }
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}

bool check_crash_j3(){
  int cc = 0;
  no_car_intersection[3] = 0;
  for(int x=0;x<6;x++){
    if(junction3_prech[x] != 0){
      no_car_intersection[3] = no_car_intersection[3]+1;
      if(junction3_prech[x] == junction3_posch[x]){
        cc = cc + 1;
      }
    }
  }
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}
```

```
bool check_crash_j4(){
  int cc = 0;
  no_car_intersection[4] = 0;
  for(int x=0;x<12;x++){
    if(junction4_prech[x] != 0){
      no_car_intersection[4] = no_car_intersection[4] +1;
      if(junction4_prech[x] == junction4_posch[x]){
        cc = cc + 1;
      }
    }
  }
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}

bool check_crash_j5(){
  int cc = 0;
  no_car_intersection[5] = 0;
  for(int x=0;x<6;x++){
    if(junction5_prech[x] != 0){
      no_car_intersection[5] = no_car_intersection[5] + 1;
      if(junction5_prech[x] == junction5_posch[x]){
        cc = cc + 1;
      }
    }
  }
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}

bool check_crash_j6(){
  int cc = 0;
  no_car_intersection[6] = 0;
  for(int x=0;x<2;x++){
    if(junction6_prech[x] != 0){
      no_car_intersection[6] = no_car_intersection[6] +1;
      if(junction6_prech[x] == junction6_posch[x]){
        cc = cc + 1;
      }
    }
  }
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}
bool check_crash_j7(){
  int cc = 0;
  no_car_intersection[7] = 0;
  for(int x=0;x<6;x++){
    if(junction7_prech[x] != 0){
      no_car_intersection[7] = no_car_intersection[7]+1;
      if(junction7_prech[x] == junction7_posch[x]){
        cc = cc + 1;
      }
    }
  }
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}

bool check_crash_j8(){
  int cc = 0;
  no_car_intersection[8] = 0;
  for(int x=0;x<2;x++){
    if(junction8_prech[x] != 0){
      no_car_intersection[8] = no_car_intersection[8] +1;
      if(junction8_prech[x] == junction8_posch[x]){
        cc = cc + 1;
      }
    }
  }
  if(cc > 1) {
    return 1;
  }
  else {
    return 0;
  }
}


void junction_pre_check_for_crash(){

  junction0_precheck();
  junction1_precheck();
  junction2_precheck();
  junction3_precheck();
  junction4_precheck();
  junction5_precheck();
  junction6_precheck();
  junction7_precheck();
  junction8_precheck();

}
void junction_pos_check_for_crash(){
  junction0_poscheck();
  junction1_poscheck();

  junction2_poscheck();
  junction3_poscheck();
  junction4_poscheck();
  junction5_poscheck();
  junction6_poscheck();
  junction7_poscheck();
  junction8_poscheck();
}

int check_crash(){
  int tc = 0;
  tc = tc + check_crash_j0();
  tc = tc + check_crash_j1();
  tc = tc + check_crash_j2();
  tc = tc + check_crash_j3();
  tc = tc + check_crash_j4();
  tc = tc + check_crash_j5();
  tc = tc + check_crash_j6();
  tc = tc + check_crash_j7();
  tc = tc + check_crash_j8();
  return tc;
}
```

```
int car_remaining_in_grid(){
  int cnt=0;
  for(int i=0; i<24; i++){
    for(int j=0 ; j<30; j++){
      if(segment[i][j] != 0){
        cnt = cnt+1;
      }
    }
  }
  return cnt;
}

int collision_in_a_segment(int exp_total, int act_total){
  if(exp_total != act_total){
    return exp_total-act_total;
  }
  else
  return 0;
}

void u_turn_precheck(){
  for(int i=0;i<24;i++){
    u_pre_ch[i]=segment[i][29];
  }
}

void u_turn_poscheck(){
  for(int i=0;i<24;i++){
    if(i%2 == 0 || i == 0){
      u_pos_ch[i] = segment[i+1][0];
    }
    else{
      u_pos_ch[i] = segment[i-1][0];
    }
  }
}

int check_utrun_vio(){
  int ccc = 0;
  for(int i=0;i<24;i++){
    if(u_pre_ch[i] == u_pos_ch[i] && u_pre_ch[i]!= 0 ){
      ccc = ccc+1;
    }
  }
  return ccc;
}


void red_vio_pre_check(){
  for(int i=0;i<24;i++){
    if(il[signal[i][0]][signal[i][1]] == 1){
      r_pre_ch[i] = segment[i][29];
    }
    else
      r_pre_ch[i] = 0;

  }
}

void red_vio_pos_check(){
  for(int i=0;i<24;i++){
    if(il[signal[i][0]][signal[i][1]] == 1){
      r_pos_ch[i] = segment[i][29];
    }
    else
      r_pos_ch[i] = 0;

  }
}



int check_r_vio(){
  int cccc = 0;
  for(int i=0;i<24;i++){
    if(r_pre_ch[i] != r_pos_ch[i] && r_pre_ch[i] != 0 ){
      cccc = cccc+1;
    }
  }
  return cccc;
}
```

*Main.cpp: [Note : traffic.h will be supplied by the I-group]*

```
#include <iostream>
#include <time.h>
#include <queue>
#include <vector>
#include <map>
#include <string>
#include "segment.h"
#include "crash.h"
#include "traffic.h"

using namespace std;

int main() {
    //Initializations of global variables
    t=0;
    total_cars = 0;
    total_completed_cars = 0;
    total_cars = 0;
    u_turn_vio = 0;
    r_vio = 0;
    mis_dir_cnt = 0;

    //Initialziation functions to build the road system and bring it to initial
state
    create_segments();
    create_possible_routes();
    car_pos_signal_map();
    //Initialization function from the I-group
    initialise_queue ();

    while(t<3600){
      init_crash_count = 0;
      if( t > 0 ){
        //std::cout<<"Time @ "<<t<<'\n';
        if(total_cars == 0 || car[total_cars-1][2] != -1){
          generate_car(total_cars);
          total_cars = total_cars + 1;
        }
      }

      if(t%10 == 0){
      switch_light ();
      }
      std::cout<<il[0][0]<<','<<il[0][1]<<','<<il[0][3]<<'\n';
      red_vio_pre_check();
      u_turn_precheck();
      junction_pre_check_for_crash();

      update_all_car_pos();

      junction_pos_check_for_crash();
      u_turn_poscheck();
      red_vio_pos_check();
      r_vio = r_vio + check_r_vio();
      u_turn_vio = u_turn_vio + check_utrun_vio();
      total_crash = total_crash + check_crash();
      t=t+1;
      t=t+1;
    }

    filled_slot_info(t);
    std::cout<<"Total car before func = "<<total_cars<<'\n';
    final_pos_car();
    rem_cars = car_remaining_in_grid();
    std::cout<<"Total completed cars = "<<total_completed_cars<<'\n';
    no_col_in_slot =
collision_in_a_segment(total_cars,total_completed_cars+rem_cars);
    std::cout<<"Total collisions in a slot = "<<no_col_in_slot<<'\n';
    std::cout<<"U turn violations = "<<u_turn_vio<<'\n';
    std::cout<<"R violations = "<<r_vio<<'\n';
    std::cout<<"Misdirection = "<<mis_dir_cnt<<'\n';
    std::cout<<"Total crashes = "<<total_crash;
    clear();

}
```