# Task 1

**CASINO PARTY**


Tommy Shelby, the leader of the Peaky Blinders has come to know that his family members have been fighting over power. Now, Tommy who is deeply intrigued by betting against odds, wants to test out a new game. He doesn't like being disobeyed by anyone irrespective of their identity. He takes them to his brand new casino, The Birmingham Gamblers. He pulls out a cigarette, lights it and begins with a quote,

"**You can change what you do... but you can't change what you want. Let the game begin...**"

## Game Description

Welcome to the game world, a world where all the matters are sorted by games. Let's start building a simulated version of a betting game. The game revolves around betting your numbers to a casino.

There are 3 players A, B and C in the game, competing to win the grand betting tournament.

1. The game revolves around the server(**The casino**) distributing three cards to each of the players A, B and C in the range of 1 to 52. With cards 1-13 belonging to the suit **Diamond**, 14-26 belonging to **Hearts**, 27-39 belonging to **Spades**, 40-52 belonging to **Clubs**.

**Note:** The value of number 14 means that the card is "Ace of hearts" with value 1; similarly 15 in the array means the card is "2 of hearts" and 27 means "Ace of spades" and so on..

| CARD | VALUE |
|---|---|
| A | 1 |

***Note:*** *Don't Worry. You don't need to open NETFLIX*

| | |
|---|---|
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | 10 |
| J | 11 |
| Q | 12 |
| K | 13 |

2. In each round the server forms an array of 9 numbers,(You need to make sure that all random numbers in this array are distinct). The first 3 numbers of this array will be given to A, next 3 numbers to B and the last 3 numbers to C.
3. So as a player when the game starts you know the cards you have, you have to find the maximum number among all the 3 cards and return this value to the server.
   The player with the maximum value among all the returned values will win the game. You need to declare the winner of the round by sending a message to the casino declaring the winner. For e.g. “A is the winner of Round 1”.
4. The games ends when 4 rounds are played and after the 4th round the winner is declared shown to each player. For eg. “A wins the game”.

## Program files

You have to make two scripts in c++ or python.

1. **Casino:** sets up the network of players connected to it and assigns the random numbers to each player at the beginning of each round (file casino.py or casino.cpp)

After each round the casino declares the winner and displays it on the casino terminal.

2. **Player:** receives the 3 cards sent by the casino and returns the maximum of them to the casino.

***Note:-*** *all the players must run the same script.*

**Language: C++/Python**

**Suggested Reading: Socket Programming, Multithreading**

# Bonus

Make a separate client script which asks the user for the number of players and the number of rounds after which the score is displayed and also ensures that the number of rounds is not divisible by the number of players. This data is sent to the casino and the number of players is determined accordingly.

# Example

1. First run casino.cpp/casino.py and connect it to all the three players by running the player.py/player.cpp script in separate terminals.
2. The casino.cpp/casino.py shouldn't close the connection until the games is over ie. 4 rounds are played.
3. The distribution of the cards should be done simultaneously for each player using multithreading.
4. When a round is completed, the winner should be displayed on the casino terminal and when the score is displayed after 4 rounds, it should be visible only to every player terminal along with the casino terminal.

# Bonus Example

1. First connect client to casino
2. Client asks for the number of players and the number of rounds after which the score is displayed and sends it to casino which prints the number of players and displays it on the casino terminal
3. Then as many terminals are opened as the number of players and each player is connected to the server and the game runs.

# Task 2

## TRAFFIC JAM

What happens when we combine human consciousness with automobiles? You enter the fictitious world of cars. Welcome to the world of cars where automobiles have acquired consciousness and sadly inherited the stupidity which comes with that consciousness. So where there is stupidity, there needs to be rules! Law and order. Once a racecar, Hudson Hornet aka. Doc, takes care of the law in Radiator Springs and is faced with a problem of traffic jams and needs to devise a system to maintain the traffic of Radiator Springs. Help Doc design a traffic handling system so that the traffic problem in Radiator Springs is solved and everybody is happy.

## Task Description

In this task you are required to design a traffic light system for a cross road. Your design should be based on a finite state machine (FSM).

At the cross road, cars can come from any of the four directions/sides and can either move straight or take a right turn. The traffic of each of the four sides is controlled using 4 sets of traffic lights. Each set of traffic lights has 3 lights - one red, one straight green, one right green. A red light indicates all cars from that road are at a halt. A straight green indicates cars which want to move straight are allowed to pass, and a right green indicates cars which want to move right are allowed to pass. It may so happen that both the straight and right green signals are active at the same time, meaning both, cars which want to go straight and those which want to go right are both allowed to pass.

## Assumptions:

1. It takes a car one time step to cross the junction.
2. For each side of the junction 2 new cars can appear at each time step. One which wants to go straight and one right. It may so happen that only one of them appears or none of them do. But it can't be the case that 2 cars appear both of which want to go straight (or right).
3. Cars line up in a line while waiting on the signal. Each side has two lines, one of the cars wanting to go straight and one for the cars going right.

## Input format:

1. Your code first reads an integer 't'. It denotes the number of time steps for which the cars keep entering the cross roads.

2. It is followed by t lines. Each line contains 8 space separated integers:
   A1 A2 B1 B2 C1 C2 D1 D2 (all will be either 0 or 1)

   A, B, C and D represent the four sides. x1=1 implies a car which wants to go straight has appeared on the side x. x1=0 implies no such car appeared. x2=1 implies a car which wants to go right has appeared on the side x. x2=0 implies no such car appeared.

   For example, 0 1 1 1 0 0 1 0 means a car which wants to go right appeared at side A, 2 cars appeared at side B one which wants to go right and one straight, no cars appeared at side C, a car which wants to go straight appeared at side D.

## Output format:

You can format the output as you wish but it should be clear and easy to understand and contain the following information:

1. The state of the lights at each time step.
2. The cars which cross the cross road at that time step. (No explicit identity to cars has been assigned to in the input format, you can do so for your convenience in a suitable format). You may also print the queue of the waiting cars at each time step.

## Constraints:

1. For each side in each of the 2 possible directions (straight and right) only 1 car can cross in a single time step.
2. The cars crossing the cross road at a time step must be in accordance with the state of the lights at that time step. A car from a side which has a red signal cannot cross or if only the

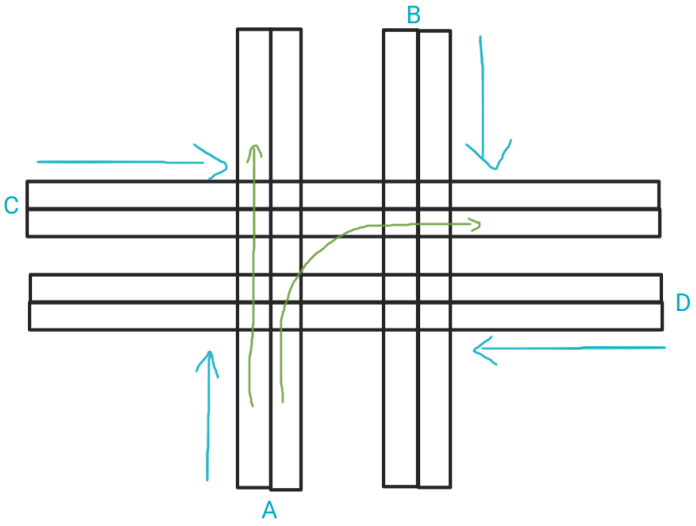
straight green signal is on for a side, then a car which wants to go right from that side cannot cross.

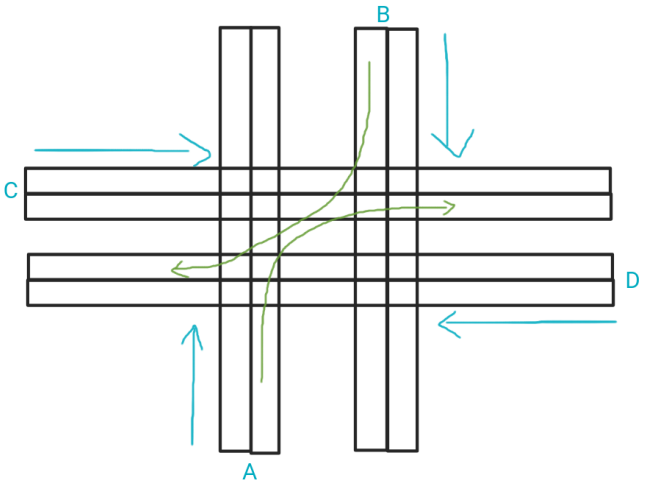
3. Note that there are 4 sets of 3 lights. So we have a total of 12 lights, each of which can be either on or off. Thus the total possible configurations are 2^12=4096. However, only a few of these configurations are actually valid. We define valid configurations below.

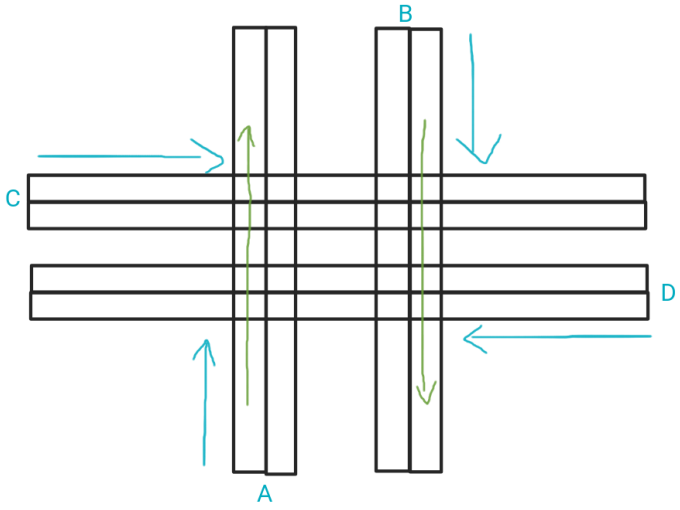
## Valid Configurations:

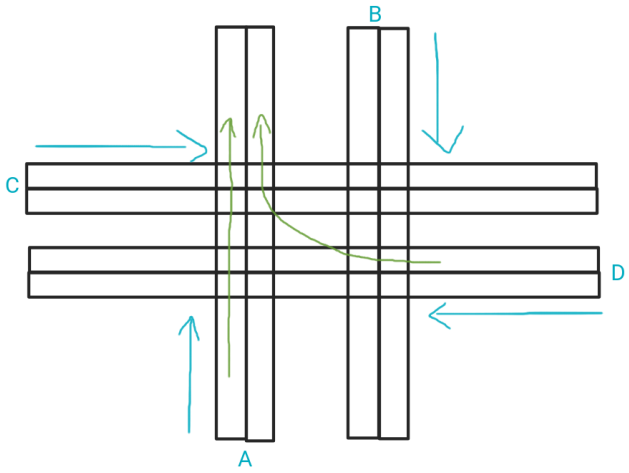
**STRICTLY FOLLOW THE ORIENTATIONS OF SIDES A, B, C and D AS SHOWN BELOW**

A valid configuration of traffic lights is one which allows smooth flow of traffic without causing any jam. A basic requirement is that for any flow of traffic from a particular side, it should not be the case that some other flow intersects with it. The valid configurations can be one of the following:

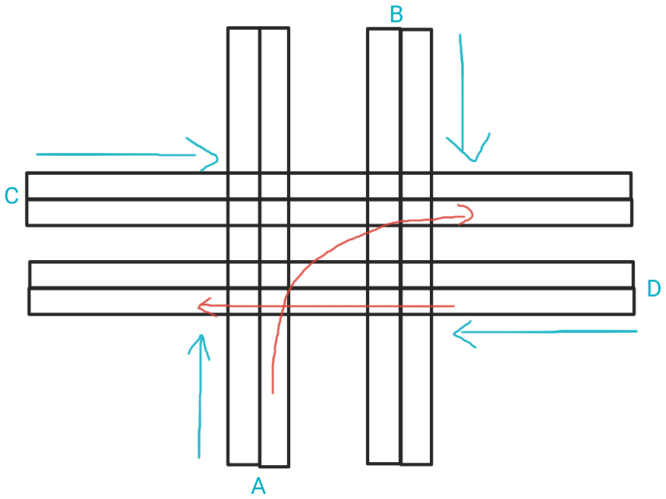


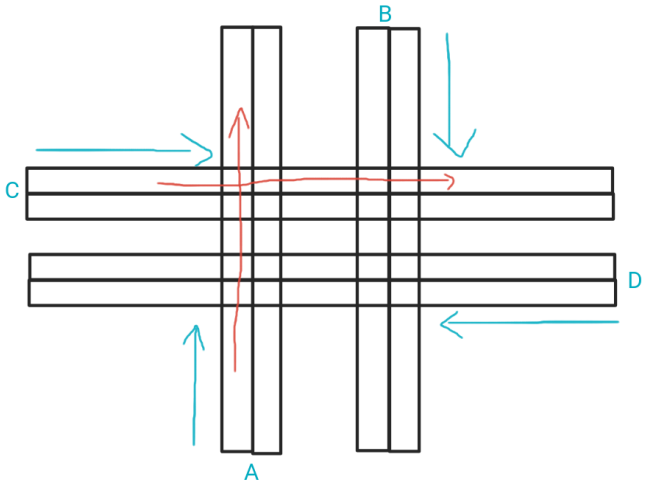



- **Note** that these are shown from the perspective of side A. Similar configurations are valid for side B, C and D as well. (Blue arrows show the direction in which traffic moves on each side, green arrows represent the allowed flow of traffic.)
- Below are examples of invalid configurations:




Implement a FSM based controller in Python or C/C++. Note the following points while implementing:

1. While we want the controller to be optimised, your primary focus should be to design a FSM with clearly defined transitions and states.
2. Optimise the transition function to ensure smooth flow of traffic on each of the four sides. Traffic should not be allowed to accumulate on any of the four sides. Your controller must automatically decide the state of the lights to ensure this.
3. Your code should process queries in an online manner, i.e., you must not read the next line of input until you have already given the output for the present time step.

## Bonus Task

Instead of reading from the standard input, now you have to send the data to your program using Socket Programming. You will have to design a client which will allow you to enter the information of cars appearing at each time step. The client will forward this to the server where the controller resides.

# EXAMPLE

t=4

Input line 1: 1 1 1 1 1 1 1 1

Time step 1: A - go straight
B - go straight
C - off
D - off
Initial queue - 1 1 1 1 1 1 1 1
Final queue - 0 1 0 1 1 1 1 1

Input line 2: 1 0 1 1 1 0 1 0

Time step 2: A - go straight
B - go straight
C - off
D - off
Initial queue - 1 1 1 2 2 1 2 1
Final queue - 0 1 0 2 2 1 2 1

Input line 3: 0 0 1 1 0 0 0 0

Time step 3: A - go right
B - go right
C - off
D - off
Initial queue - 0 1 1 3 2 1 2 1
Final queue - 0 0 1 2 2 1 2 1

Input line 4: 0 1 1 0 1 0 0 0

Time step 4: A - off
B - go straight, go right
C - off
D - off
Initial queue - 0 1 2 2 3 1 2 1
Final queue - 0 1 1 1 3 1 2 1

Time step 5: A - off
B - go straight, go right
C - off
D - off
Initial queue - 0 1 1 1 3 1 2 1
Final queue - 0 1 0 0 3 1 2 1

Time step 6: A - off
B - off
C - go straight
D - go straight
Initial queue - 0 1 0 0 3 1 2 1
Final queue - 0 1 0 0 2 1 1 1

Time step 7: A - off
B - off
C - go straight
D - go straight
Initial queue - 0 1 0 0 2 1 1 1
Final queue - 0 1 0 0 1 1 0 1

Time step 8: A - off

B - off

C - go right

D - go right

Initial queue - 0 1 0 0 1 1 0 1

Final queue - 0 1 0 0 1 0 0 0

Time step 9: A - go right

B - off

C - go straight

D - off

Initial queue - 0 1 0 0 1 0 0 0

Final queue - 0 0 0 0 0 0 0 0

NOTE:

1. The following output is given as an example. We do not expect your output to exactly match with the one given above. However, your output should adhere to the constraints given in the problem description.
2. Your code must terminate only after clearing all traffic.