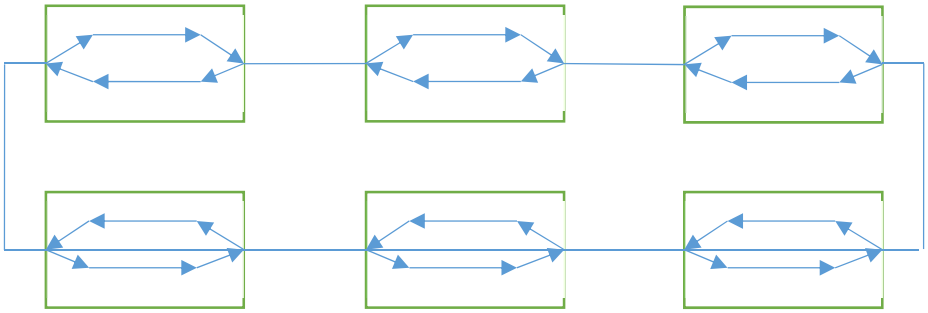
1)

A circle underground line has a number of stations connected by a single track. Every station has two tracks, track 0 devoted to the trains traveling clockwise, and track 1 devoted to the trains traveling counterclockwise, as shown in the figure.



Implement a concurrent C program that takes two command line arguments, the number of stations in the circle line, and the number of trains (less than the number of stations).

The main thread must randomly setup the initial condition, i.e., the station and track where every train is, and create a thread for every train. Notice that the direction of the train depends on the track it is resting. A train traveling clockwise will always use track 0 of any station, and train traveling counterclockwise will always use track 1 of any station.

When all the data structures representing trains, the stations, and the connection tracks between two stations have been set, the train thread can start. Every train stay at the station for a random number of seconds (max 5), then, if it is possible it goes to the next station, occupying the connection track for 10 seconds. The movement of the train (always in the same original direction) is possible if the destination station track is free and the connection track is free.

Every thread must print its current condition in a common log file.

See this example of log for 10 stations and 3 trains:

```
Train n. 0, in station 5 going COUNTERCLOCKWISE
Train n. 2, in station 6 going CLOCKWISE
Train n. 1, in station 2 going CLOCKWISE
Train n. 2, traveling toward station 7
Train n. 1, traveling toward station 3
Train n. 0, traveling toward station 4
Train n. 2, arrived at station 7
Train n. 2, in station 7 going CLOCKWISE
Train n. 2, traveling toward station 8
Train n. 1, arrived at station 3
Train n. 1, in station 3 going CLOCKWISE
Train n. 0, arrived at station 4
Train n. 0, in station 4 going COUNTERCLOCKWISE
```

.....

Suppose that you have available a function that returns a different pair (station, track) every time it is called: **`void select_station_and_track(int *station, int *track)`**

2)

Implement a program in concurrent C language using Pthreads and semaphores that creates **two** threads.

The first thread loops generating **k** times (**k** is given as argument in command line) a new thread that corresponds to an atom of Sodium (Na).

The creation requires a random times in the range [**0-4]** seconds. These threads are identified by numbers in range **0 to k-1.** The second thread has the same behaviour of the first one, but it generates atoms of Chlorine (Cl). These threads are identified by numbers in range **k to 2*k-1.**

When a Na atom (thread) is created, it must wait that at least one Cl atom exists. In this case, it can proceed to the combination of the two atoms, which simply consist in the thread printing its identifier and the identifier of the other combining thread, then the thread terminates.

A Cl thread has the same behaviour of a Na thread.

This is an example of the output of the program execution with **k=4.**

```
id: 0 - Na 0 Cl 6 id: 6 - Na 0 Cl 6
id: 1 - Na 1 Cl 7 id: 7 - Na 1 Cl 7
id: 2 - Na 2 Cl 5 id: 5 - Na 2 Cl 5
id: 3 - Na 3 Cl 4 id: 4 - Na 3 Cl 4
```

Please notice that both the Na and the Cl threads print their identifiers with the same format: the Na identifier first.

Hints: manage two parallel arrays of the identifier of the threads that have produced their atoms, and synchronize the print operations of each pair of Na and Cl threads.

Na	1	2	3
Cl	7	5	4