Please answer the below question to the best of your ability. If you have any questions do not hesitate to reach out to us for clarification. You may use any libraries you feel comfortable with to implement your solution.

Depth Charts

Most team sports have a depth chart (a ranking of each player) for each position they have. For example in NFL:

- Ben Roethlisberger is listed as the starting QB and first on the QB depth chart.
- Landry Jones, his backup is listed as the 2nd person on that depth chart.

We want to implement functionality that will manage these depth charts.

Data Model

Assume player objects look like this. Note that players can be on the depth chart for positions that are not their own.

```
{
      "player_id": 1,
      "name": "Bob",
      "position": "WR"
}
```

Use Cases to Implement

- `addPlayerToDepthChart(player, position, position_depth)` Adds a player to a depth chart for a given position (at a specific spot). If no `position_depth` is provided, then add them to the end of the depth chart for that position. If you are entering two players into the same slot, the last player entered gets priority and bumps the existing player down a depth spot.
- `removePlayerFromDepthChart(player, position)` Removes a player from the depth chart for a position
- `getFullDepthChart()` Prints out all depth chart positions
- `getPlayersUnderPlayerInDepthChart(player, position)` For a given player find all players below them on the depth chart.

Example

```
var bob = { "player_id": 1, "name": "Bob" }
var alice = { "player_id": 2, "name": "Alice" }
var charlie = { "player_id": 3, "name": "Charlie"}

addPlayerToDepthChart(bob, "WR", 0);
addPlayerToDepthChart(alice, "WR", 0);
addPlayerToDepthChart(charlie, "WR", 2);
addPlayerToDepthChart(bob, "KR");
getFullDepthChart();

/*
Output:
WR: [2, 1, 3],
KR: [1]
*/

getPlayersUnderPlayerInDepthChart(alice, "WR");

/*
Output:
      [1,3]
*/
```

Please implement the 4 use cases above for:

1. NFL supporting positions (QB, WR, RB, TE, K, P, KR, PR)
2. MLB supporting positions (SP, RP, C, 1B, 2B, 3B, SS, LF, RF, CF, DH).

Important

1. The logic should be able to scale. Keep in mind we potentially might add more sports in the future and we want to make it as easy as possible to add new ones.
2. You do not need to use these exact method signatures, but you need to have methods that satisfy the described use cases.
3. Include any appropriate automated tests
4. Code organisation and maintainability is an important consideration. Please factor this into your solution.
5. We're only looking for appropriate use of algorithms, data structures and dom ain modelling in the solution.
   We don't want you to expend effort on building a user interface, API endpoint s or a database-backed solution.

Submission

Please upload solution and instructions needed to build and/or run your code on your private Github account and share the link.

Instructions and assumptions should be placed inside a `README.md` file at the root level of the solution.