C# Developer Exercise

# Train ticket machine

You are asked to write code to support the user interface of a train ticket machine.

You will not be writing any actual User Interface code.

But you should develop a search algorithm to help the user entering the name of a station.

The machine has a touchscreen display which works as follows.

As the user types each character of the station’s name on the touchscreen, the display should:

1. Update to show all valid choices for the next character
2. List of possible matching stations.

The illustration below shows what is needed when ‘D A R T’ has been entered.

User input: **D A R T \_\_**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E |  | **DARTFORD** |
| **F** | G | H | I | J |  | **DARTMOUTH** |
| K | L | **M** | N | O |  |  |
| P | Q | R | S | T |  |  |
| U | V | W | X | Y |  |  |
| Z |  |  |  |  |  |  |

## Requirements:

1. Typing a search string will return:
   1. All stations that start with the search string;
   2. All valid next characters for each matched station;
2. Runtime speed is very important;
3. A space is a valid character when returning a list of next characters;
4. You don’t need to go overboard with your station list in your tests. A small enough list of stations to adequately test each condition will suffice

## Not Required:

* A fast loading time is not required at start-up, runtime performance takes priority;
* This will be run on a dedicated machine designed for the purpose;
* The application will be used by a single user at a time. There’s no need to code for concurrency;
* No code is required for reading the stations from a data store;
* You may stub the station list or mock a station reader in your tests, whichever you feel represents the best real world solution;

## Expected Scenarios:

* **Given** a list of stations ‘DARTFORD’, ‘DARTMOUTH’, ‘TOWER HILL’, ‘DERBY’
  + **When** input ‘DART’
  + **Then** should return:
    1. The characters of ‘F’, ‘M’
    2. The stations ‘DARTFORD’, ‘DARTMOUTH’.
* **Given** a list of stations ‘LIVERPOOL’, ‘LIVERPOOL LIME STREET’, ‘PADDINGTON’
  + When input ‘LIVERPOOL’
  + Then should return:
    1. The characters of ‘ ‘
    2. The stations ‘LIVERPOOL’, ‘LIVERPOOL LIME STREET’
* **Given** a list of stations ‘EUSTON’, ‘LONDON BRIDGE’, ‘VICTORIA’
  + **When** input ‘KINGS CROSS’
  + **Then** the application will return:
    1. no next characters
    2. no stations

# Evaluation Guidelines:

1. **Delivery quality**
   1. Complete solution meeting all requirements;
   2. Project structure;
   3. Packaging and installation;
2. **Code readability**
   1. Assemblies organization;
   2. Class organization;
   3. Class and method and fields naming;
   4. Effective documentation;
3. **Code quality**
   1. Coding against tests;
   2. Code coverage;
   3. Code complexity;
4. **Solution quality**
   1. Use of O.O.P approach;
   2. Use of S.O.L.I.D. principles;
   3. Code against interfaces;
   4. K.I.S.S;
   5. D.R.Y;
   6. Correct use of proven Patterns;
5. **Use of framework features;**
   1. Correct use of frameworks features;
   2. Data structures fit for purpose;
   3. Use of the fast features;