Programming 2 – Lab15 – AL

Your task is to implement hierarchy of classes representing terrain map and providing functionality of finding a route.

Class `MapInterface` is an abstract class defining an interface for every derived class representing a terrain map.

`TerrainMap` class which implements an interface defined by the `MapInterface` abstract class. The `TerrainMap` represents a 2-dimensional map divided into small squares. Each square is identified by a pair of coordinates.

Part 1 (2,5 point)

Class `TerrainMap` publicly inherited from `MapInterface` implements an interface defined by the `MapInterface.`

Additionally class `TerrainMap` possesses members:

```
enum class TerrainState{ Inaccessible, Accessible, Visited };
```

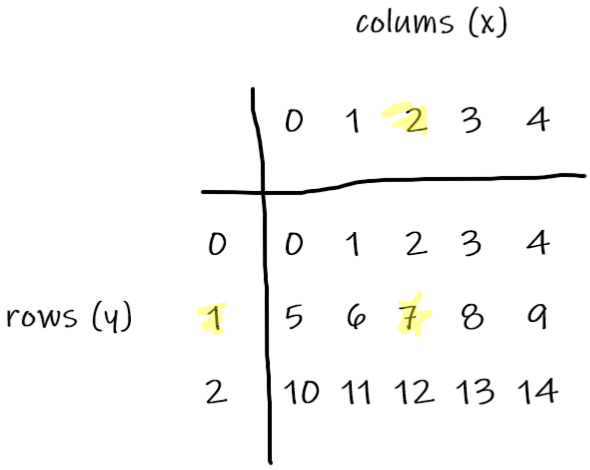
defining terrain types, which can be accessible (e.g. flat surface), inaccessible (e.g. swamps) or already visited (e.g. well-trodden path).

```
int cols; int rows;
```

defining the size of the map.

```
vector<TerrainState> positions;
```

keeping information about all positions on the map (what kind of terrain is formally at given position on the map). The `TerrainMap` represents a 2-dimensional map divided into small squares. Each square is identified by a pair of coordinates (x,y), where x indicate column and y indicate row. Formally map is stored in one-dimensional vector container. To convert between 2D coordinate (x,y) to proper 1D location inside vector, we use formula `y * cols + x`. E.g. coordinates (2,1) for a map of 5 columns and 3 rows are store in `positions` vector at location 1 * 5 + 2 = 7 (positions[7] – vector container can be accessed using classical indexing with []). We are indexing form 0. Relation between 2D and 1D coordinates are explained on Fig. 1.



	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14

Fig. 1. Relation between 2D coordinates (x,y) and 1D location in vector container.

Implement interface. Look also into main, for example usage of `TerrainMap` class.

`TerrainMap` constructor creates terrain map of given size, filling all elements in vector container as accessible (vector `positions` should be initialized to size `cols * rows` and filled with `Accessible` value).

Method `bool is_x_inRange(int x) const` checks whether given x coordinate is within range <0; cols). Method `bool is_y_inRange(int y) const` checks whether given y coordinate is within range <0; rows). For a map of 5 columns and 3 rows, x=5 is out of rang, because in range are only x coordinates in <0,4>.Also y=3 is out of range, because in range are only y coordinates in <0,2>. All negative coordinates are also out of range.

Methods `bool isAccessible(int x, int y) const` and `bool isInaccessible(int x, int y) const` check whether terrain at given position (x,y) is accessible or inaccessible.

Methods `void setAccessible(int x, int y)` and `void setInaccessible(int x, int y)` set terrain at given position (x,y) to accessible or inaccessible. All methods should throw standard `out_of_range` exception if position is incorrect (internally check whether x and y is in range).

When printing `TerrainMap` use “0” to indicate accessible position, “X” to indicate inaccessible positions and “#” to indicate visited positions.

Part 2 (1,5 point)

Method `void setVisited(int x, int y)` sets terrain at given position (x,y) to visited. The method should throw standard `out_of_range` exception if position is incorrect (internally check whether x and y is in range).

Method `int visitedCount() const` counts number of visited places on the map. In implementation use `count_if` algorithm with properly defined lambda expression.

Part 3 (1 point)

Overload `operator!(TerrainMap& operator!())`. The method should modify the internal representation of the map by changing accessible positions into inaccessible and inaccessible into accessible (visited are left unchanged). In implementation use `transform` algorithm with properly defined lambda expression.

Part 4 (1 point)

From `TerrainMap` inherit publicly class `Cartographer`. This class provides initially the same functionality as `TerrainMap` class. Implement necessary interface to create `Cartographer` class objects. Look also into main, for example usage of `Cartographer` class.

Part 5 (1 point)

Add method `void clearPath();`

The method should clear all existing paths (visited places on the map) by changing state of position form visited to accessible. In implementation use `for_each` algorithm with properly defined lambda expression.

Part 6 (1,5 point)

Add method `bool findRoute(int start_x, int start_y, int finish_x, int finish_y);`

The method checks whether it is possible to find path form starting position (`start_x, start_y`) to final position (`finish_x, finish_y`).

Hint 1: use `pair<int, int>` to represent coordinates of the positions.
Hint 2: use `queue<pair<int, int>> candidates`, to keep candidate positions for path;
Hint 3: the algorithm should be similar to:

```
-check if the start position and finish positions are in the map and are accessible
-put the start position into the queue (use push)
-while the queue is not empty:
    -take and remove the first position from the queue (use method front and
    poprespectively)
    -if the position is equal to the finish position return true
    -mark the corresponding position as visited
    -put into the queue the accessible neighbors of the current position (analyze
    four neighbors, checking if they are in range and accessible)
-clear already visited path (all visited positions)
-return false
```

Example program output

```
******************************** Part 1 (2,5 pts) ********************************
Position (5,3) is not in the range of map of 5 columns and 3 rows
Position (-1,-1) is not in the range of map of 5 columns and 3 rows
Position (1,1) is in the range of map of 5 columns and 3 rows

Position (2,0) is inaccessible
Position (2,1) is inaccessible
Position (0,0) is accessible
Position (2,2) is accessible

X is out of range
X is out of range
Y is out of range
Y is out of range
```

```
00X00
00XXX
00000

******************************** Part 2 (1,5 pt) ********************************
#0X00
#0XXX
##000
Number of visited elements: 4

******************************** Part 3 (1 pt) ********************************
#X0XX
#X000
##XXX
Number of visited elements: 4

******************************** Part 4 (1 pts) ********************************
#0X00
#0XXX
##000


******************************** Part 5 (0,5 pts) ********************************
00X00
00XXX
00000


******************************** Part 6 (1,5 pt) ********************************
Route not found!
00X00
00XXX
00000

Route found!
##X00
##XXX
####0
```