Homework: Class mat<T>

You should implement a template parameterized class mat (in case someone is not familiar with templates you can implement with only int type, e.g. without **`template<typename T>`**) which is going to be a 2d-array, allocated with operator new.
The mat class should have the following structure:

```
template <typename T>
class mat
{
      size_t M;
      size_t N;

      T** v;
public:
};
```

You have to implement the following methods:

- **`mat(const vector<vector<T>>&)`** - a constructor which uses a **`const vector<vector<T>>&`** parameter to initialize the mat object.
  Example:
  `vector<vector<int>> v1 = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };`
  `mat<int> m(v1);`
  The above line should be compiled and run correctly.
- **`mat(size_t m, size_t n)`** - a constructor which should initialize the mxn mat object with default values of type T.
  Example:
  `mat<int> m(10, 10); // each element of m[i][j] should have the default value zero.`
- **`mat(const mat<T>& m2)`** - copy constructor. You need to deep copy each value of `m2` into this object.
  Example:
  `vector<vector<int>> v1 = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };`
  `mat<int> m(v1);`
  `mat<int> m2(m);`
  `//m and m2 should have exactly the same values`
- **`~mat()`** - Destructor to correctly release (delete) all the allocated memory.
- Two subscript operators **`operator[]`** and **`const operator[]`**
- **`pair<size_t, size_t> size() const`** - to return M and N.
- **`vector<vector<T>> to_vec_of_vec() const`** - convert mat to **`vector<vector<T>>`**.
- **`friend ostream& operator<<(ostream& os, const mat<T>& m)`** - to be able to cout mat objects

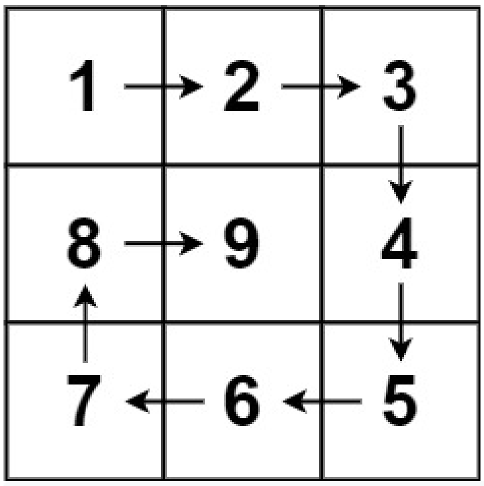
You also have to solve and implement the following problems in the most efficient way.

- **PROBLEM #1**

  ```
  template <typename T>
  mat<T> generate_spiral_mat(int n)
  ```

  Given a positive integer n, you need to generate an nxn mat object filled with elements from 1 to n^2 in spiral order.

  Example 1:

  ```
  Input: n = 3
  Output: [[1,2,3],[8,9,4],[7,6,5]]
  ```

  Example 2:

  ```
  Input: n = 1
  Output: [[1]]
  ```

  (Note that this function is not going to be a class member)

  Code example:

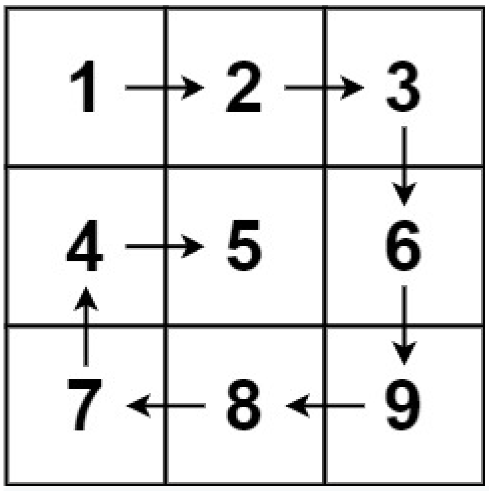
  ```
  mat<int> m = generate_spiral_mat<int>(3);
  vector<vector<int>> answ = m.to_vec_of_vec();
  answ == { {1, 2, 3}, {8, 9, 4}, {7, 6, 5} }
  ```

- **PROBLEM #2**

  ```
  vector<T> spiral_order() const
  ```

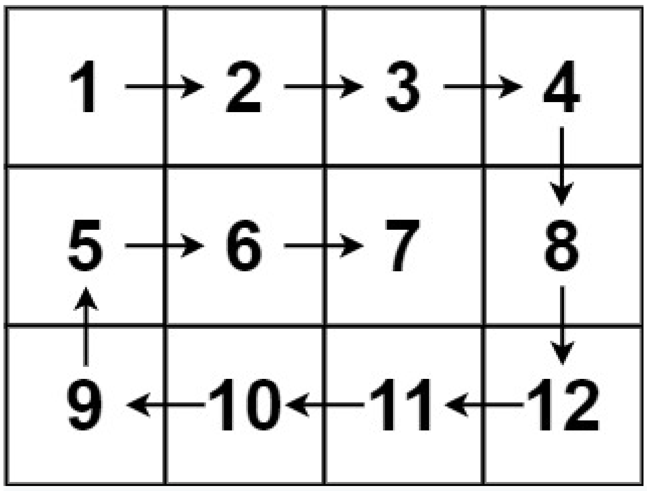
  Given an mxn `mat` object, return all the elements of the object in spiral order.

Example 1:



```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [1,2,3,6,9,8,7,4,5]
```

Example 2:



```
Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

(Note that this function is going to be a member function and it's not mandatory to be a rectangle matrix: M can be different from N).
Code example:

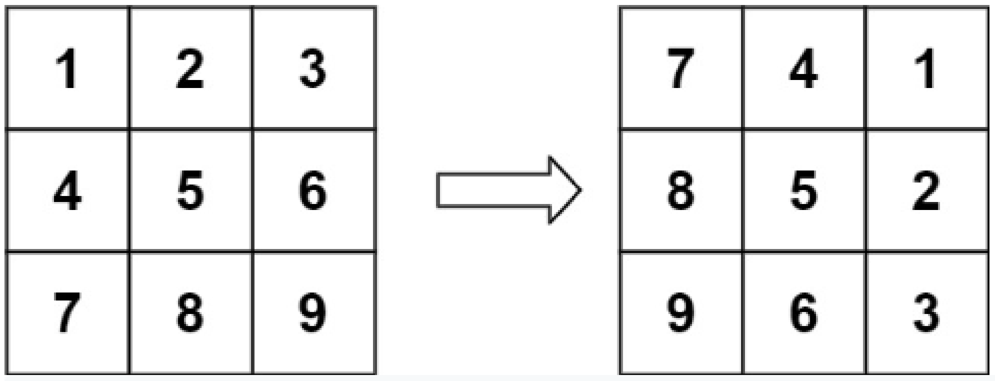
```
mat<int> m({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });
vector<int> answ = m.spiral_order();
answ == { 1, 2, 3, 6, 9, 8, 7, 4, 5 };
```

- **PROBLEM #3**

  `void rotate()`

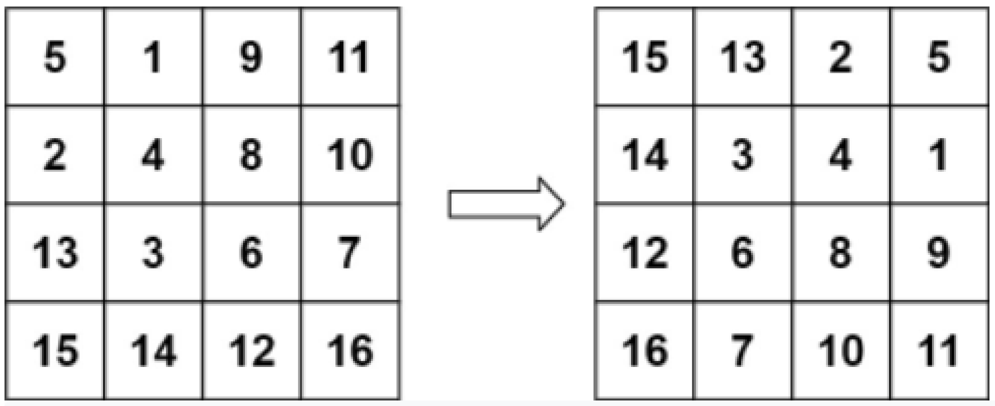
Add a rotate member function to the mat object and rotate the 2d- array by 90 degrees (clockwise). You have to rotate the object in-place, which means you have to modify the input mat directly. DO NOT allocate another 2D matrix and do the rotation.

Example 1:



```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

Example 2:



```
Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]
```

Code example:

```
mat<int> m({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });
m.rotate();
vector<vector<int>> answ = m.to_vec_of_vec();
answ == { {7, 4, 1}, {8, 5, 2}, {9, 6, 3} };
```

I implemented some tests to check the correctness of your implementation. You can use those tests to check yourself. If you implement everything right at the end you should see on the screen the following lines:

```
running spiral_order_tests
true
true

running spiral_mat_tests
true
true

running rotate_tests
true
true
true
```

This is going to be the first part of our `mat` class on which we're going to build a very huge class with rich functionality by adding more functions.