Castle

Submission deadline:	**2023-11-20 11:59:59**	1024757.650 sec
Late submission with malus:	**2023-12-31 23:59:59** (Late submission malus: 100.0000 %)	
Evaluation:	**5.5000**	
Max. assessment:	**5.0000** (Without bonus points)	
Submissions:	1 / 20 Free retries + 10 Penalized retries (-10 % penalty each retry)	
Advices:	0 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)	

The task is to implement a function (not a whole program, just a function) which computes the area of a land controlled by a castle.

Assume a map of a square shape. The map is divided into `n x n` tiles, there is altitude known for each such tile. We assume the tiles are ideally flat, the altitude only changes on the edges of the tiles. A castle may be located on any such tile, the castle takes just and exactly one tile. A castle controls the surrounding land, the controlled land must be of a rectangular shape. Moreover, the following restrictions apply:

- the castle must be located somewhere inside the rectangle it controls,
- the castle is located on the highest tile in the controlled rectangle, i.e., all other tiles in the controlled rectangle must have altitude strictly lower than the altitude of the tile with the castle, and
- the castle controls land as big as possible.

The task is to develop function `castleArea`. The function is given the map with the altitudes (a 2D array) and the size of the map. For each tile of the map, the function computes the area of the biggest rectangular land controlled by a castle located on that tile.

```
void castleArea ( int altitude[][MAP_MAX], int size, int area[][MAP_MAX] )
```

The function computes the controlled land. The parameters are:

- `altitude` input parameter - 2D array with the altitude filled for each map tile. The array contains valid values in the range `[0][0]` to `[size-1][size-1]`. The values in this array are read-only, the function must not modify them.
- `size` - the size of the map. The map is of a rectangular shape, the dimensions are `size x size`.
- `area` output parameter - element `area[y][x]` will be filled with the area of the land that is controlled by a castle located on position `[y][x]`. The function is expected to fill only elements `[0][0]` to `[size-1][size-1]`.

```
bool identicalMap ( int a[][MAP_MAX], int b[][MAP_MAX], int size )
```

The function compares the contents of two maps. The function is not called from within the testing environment, however, you may want to implement the function when using the attached tests. The function compares two 2D arrays given by parameters a and b. The function only compares elements in the range `[0][0]` to `[size-1][size-1]` (the remaining elements are not tested). Return value is either `true` if the tested elements are identical, or `false` if there is a difference.

```
constexpr int MAP_MAX = 200;
```

a constant declared in the testing environment. The value is the maximum size of the map.

Example:

The example test case `alt1` contains a map 4x4 with altitudes:

```
alt1:
 2, 7, 1, 9
 3, 5, 0, 2
 1, 6, 3, 5
 1, 2, 2, 8
```

The function computes the following areas of the controlled land:

```
area1:
 1, 12, 2, 16
 4, 4, 1, 2
 1, 9, 4, 4
 1, 2, 1, 12
```

A castle located on position [0][0] controls only itself, thus area[0][0] = 1. A castle on position [0][1] (i.e., altitude = 7) controls rectangle [0][0] - [3][2], i.e., 4x3=12 tiles, thus area[0][1] = 12. A castle on position [0][2] controls rectangle [0][2] - [1][2], thus area[0][2] = 2.

Submit a source file with the implementation of the required function `castleArea`. Further, the source file must include your auxiliary functions which are called from the required function. The function will be called from the testing environment, thus, it is important to adhere to the required interface. Use the attached sample code as a basis for your development, complete the required function and add your required auxiliary functions. There is an example `main` with some test in the attached code. These values will be used in the basic test. Please note the header files as well as `main` is nested in a conditional compile block (`#ifdef/#endif`). Please keep these conditional compile blocks in place. They are present to simplify the development. When compiling on your computer, the headers and `main` will be present as usual. On the other hand, the header and `main` will "disappear" when compiled by Progtest. Thus, your testing `main` will not interfere with the testing environment's `main`.

Your function will be executed in a limited environment. There are limits on both time and memory. The exact limits are shown in the test log of the reference. The evaluation depends on the efficiency of the algorithm used by your function:

- A naive solution with time complexity n^8 passes the basic tests, however, it does not pass any speed test (n denotes the dimensions of the map). Such solution will be awarded less than 100% points.
- The first speed test requires an algorithm with time complexity n^6 or better and a reasonably efficient imlementation.
- The second speed test requires an algorithm with time complexity n^5 or better and a reasonably efficient imlementation.
- The third speed test requires an algorithm with time complexity n^4 or better and a reasonably efficient imlementation.

Sample data:		**Download**
Submit:	Browse... **No file selected.**	**Submit**

Reference

- **Evaluator: computer**
  - Program compiled
  - Test 'Zakladni test podle ukazky': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 1.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test meznich hodnot': success
    - result: 100.00 %, required: 25.00 %
    - Total run time: 0.000 s (limit: 2.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Test nahodnymi daty': success
    - result: 100.00 %, required: 25.00 %
    - Total run time: 0.012 s (limit: 2.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Test nahodnymi daty + mem debugger': success
    - result: 100.00 %, required: 25.00 %
    - Total run time: 0.067 s (limit: 4.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Test rychlosti #1': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.065 s (limit: 6.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Test rychlosti #2': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.191 s (limit: 3.000 s)
    - Bonus test - success, evaluation: 120.00 %
  - Test 'Test rychlosti #3': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.875 s (limit: 2.809 s)
    - Bonus test - success, evaluation: 120.00 %
  - Overall ratio: 144.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.20 * 1.20)
- Total percent: 144.00 %
- Early submission bonus: 0.50

- Total points: 1.44 * (5.00 + 0.50) = 7.92

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**1**	--	--	--
Lines of code:	**53**	**53.00 ± 0.00**	**53**	`castleArea`
Cyclomatic complexity:	**15**	**15.00 ± 0.00**	**15**	`castleArea`

1	**2023-11-08 14:51:07**	**Download**
Submission status:	Evaluated	
Evaluation:	5.5000	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 1.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test': success
    - result: 100.00 %, required: 25.00 %
    - Total run time: 0.005 s (limit: 2.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Random test': success
    - result: 100.00 %, required: 25.00 %
    - Total run time: 0.149 s (limit: 1.995 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Random test + mem debugger': success
    - result: 100.00 %, required: 25.00 %
    - Total run time: 0.879 s (limit: 4.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Speed test #1': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 1.595 s (limit: 6.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Speed test #2': Abnormal program termination (Time limit exceeded)
    - Cumulative test time exceeded, killed after:: 3.005 s (limit: 3.000 s)
    - Bonus test - failed, evaluation: No bonus awarded
  - Test 'Speed test #3': Not tested
    - Bonus test - failed, evaluation: No bonus awarded
  - Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.00)
- Total percent: 100.00 %
- Early submission bonus: 0.50
- Total points: 1.00 * (5.00 + 0.50) = 5.50

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**6**	--	--	--
Lines of code:	**165**	**27.50 ± 21.10**	**72**	`main`
Cyclomatic complexity:	**26**	**4.33 ± 1.80**	**7**	`kadane`