Автор материала: Безух Владимир

Задача по работе с двумерным массивом.

Общая постановка задачи.

В двумерном массиве размерности n на n поменять местами элементы строки с номером a на элементы столбца с номером b (нумерация с нуля).

Особенности формулировки.

Следует строго определить:

— что нужно делать с общим элементом на пересечении;

— как формируется пара элементов для перестановки местами.

Рассмотрим следующий вариант:

— общий элемент заменяется на строку "AB" для наглядности;

— каждому элементу в строке и столбце (независимо друг от друга), исключая общий элемент, последовательно присваиваются номера от 1 до (n - 1), элементы с одинаковыми номерами образуют пары для перестановки местами, всего таких пар (n - 1).

Формат входных данных:

Предполагается, что все данные введены корректно.

n > 1, 0 <= a < n, 0 <= b < n.

Анализ задачи.

Количество перестановок элементов — (n - 1).

Для того, чтобы игнорировать общий элемент, можно в момент достижения общего элемента в строке или в ряду принудительно увеличивать на единицу соответствующий итератор. Назовём это действие «перескоком». Важно убедиться, чтобы «перескок» не вызывал ошибок.

Рассмотрим варианты входных данных a и b, чтобы иметь представление о возможном поведении при попытке обратиться к тому или иному элементу двумерного массива после «перескока» любого из итераторов.

0. Корректные входные a и b являются индексами строки и ряда двумерного массива соответственно.

1. 0 <= a, b < n - 1

Выбраны строка и ряд не на одной из границ матрицы:

X — возможные варианты расположения общего элемента

X	X	X	X	*
X	X	X	X	*
X	X	X	X	*
X	X	X	X	*
*	*	*	*	*

Если в момент достижения общего элемента мы захотим «перескочить» общий элемент, то останемся в пределах массива. Это не должно привести к ошибкам.

2. a == n - 1 или b == n - 1, a != b

Выбраны строка или ряд на конечной границе матрицы:

X — возможные варианты расположения общего элемента

*	*	*	*	X
*	*	*	*	X
*	*	*	*	X
*	*	*	*	X
X	X	X	X	*

Если задать цикл с конечным числом итераций (n - 1), к моменту, когда может возникнуть ошибка, все пары элементов уже будут переставлены местами и выход за пределы массива по одной из размерностей не произойдёт.

3. a == n - 1 и b == n - 1

Выбраны строка и ряд на конечной границе матрицы:

X — возможные варианты расположения общего элемента

*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	X

Сравнивать итераторы с соответствующим индексом размерности общего элемента следует перед проведением замены, т.к. если a или b заданы 0, «перескочить» общий элемент необходимо уже на первой итерации цикла.

Из этого следует, что проверки на «перескок» происходят перед непосредственной заменой пары элементов.

Если a и b одновременно заданы (n - 1), все перестановки элементов будут завершены до «перескока» любого из итераторов.

Например, пусть условие выхода из цикла выглядит так:

column_iterator != n && row_iterator != n.

Если проверка условия выхода из цикла происходит только после завершения тела цикла, т.е. for (... ; column_iterator != n && row_iterator != n;), произойдёт выход за пределы массива — такая проверка предполагает обязательный «перескок» любого из итераторов. Чтобы решить эту проблему, проверку условия выхода из цикла следует поместить перед непосредственной попыткой переставить пару элементов.

Исходный код.

```cpp
#include <string>
#include <iostream>

int main()
{
    // В рамках задачи входной интерфейс не рассматривается и может быть любым
    const size_t a = 0, b = 0, n = 5;
    std::string matrix[n][n] = {
        {"00", "01", "02", "03", "04"},
        {"10", "11", "12", "13", "14"},
        {"20", "21", "22", "23", "24"},
        {"30", "31", "32", "33", "34"},
        {"40", "41", "42", "43", "44"}
    };

    matrix[a][b] = "AB"; // Общий элемент

    // «Перескок» совершается лишь единожды
    bool flag_rows_shift = false,
         flag_columns_shift = false;

    size_t column_iterator = 0,
           row_iterator = 0;

    short int counter = n - 1;

    while (counter)
    {
        // Проверка «перескока» для итератора по строкам
        if (row_iterator == a && !flag_rows_shift) {
            ++row_iterator; flag_rows_shift = true;
        }

        // Проверка «перескока» для итератора по столбцам
        if (column_iterator == b && !flag_columns_shift) {
            ++column_iterator; flag_columns_shift = true;
        }

        std::swap(matrix[a][column_iterator], matrix[row_iterator][b]);
        ++column_iterator;
        ++row_iterator;
        --counter;
    }
```

```
    /* Вариант с for
    bool flag_rows_shift = false,
         flag_columns_shift = false;

    for (size_t column_iterator = 0, row_iterator = 0;;)
    {
        if (row_iterator == a && !flag_rows_shift) {
            ++row_iterator; flag_rows_shift = true;
        }

        if (column_iterator == b && !flag_columns_shift) {
            ++column_iterator; flag_columns_shift = true;
        }

        // Проверка выхода из цикла перед заменой
        if (column_iterator != n && row_iterator != n)
        {
            std::swap(matrix[a][column_iterator], matrix[row_iterator][b]);
            ++column_iterator;
            ++row_iterator;
        }
        else break;
    }
    */

    // Выходной интерфейс
    for (size_t i = 0; i != n; ++i)
    {
        for (size_t j = 0; j != n; ++j)
            std::cout << matrix[i][j] << " ";

        std::cout << "\n";
    }

    return 0;
}
```