In-place convert matrix in specific order

Write a code to convert a matrix in specific way without using extra space.

```
Input:
      1 2 3
      4 5 6
      7 8 9

Output:
      1 6 7
      2 5 8
      3 4 9
```

At first look, the problem seems similar to finding transpose of the matrix. But if you look carefully, you will notice that every even column in output matrix has elements of corresponding row in input matrix in opposite order.

We strongly recommend you to minimize your browser and try this yourself first.

The problem can be easily converted to transpose of the matrix by doing some modification to the input matrix. If we invert every even row present in input matrix, we can use solution given here to convert the matrix in desired order and that too without using any auxiliary memory.

Below is C++ implementation of the idea.

```
// Program for convert matrix in specific order
// using in-place matrix transpose
#include <bits/stdc++.h>
#define HASH_SIZE 128
using namespace std;

// Non-square matrix transpose of matrix of size r x c
// and base address A
void transformMatrix(int *A, int r, int c)
{
    // Invert even rows
    for (int i = 1; i < r; i = i + 2)
        for (int j1 = 0, j2 = c - 1; j1 < j2; j1++, j2--)
            swap(*(A + i*c + j1), *(A + i*c + j2));

    // Rest of the code is from below post
    // http://tinyurl.com/j79j445
    int size = r*c - 1;
    int t; // holds element to be replaced, eventually
           // becomes next element to move
    int next; // location of 't' to be moved
    int cycleBegin; // holds start of cycle

    bitset<HASH_SIZE> b; // hash to mark moved elements

    b.reset();
    b[0] = b[size] = 1;
    int i = 1; // Note that A[0] and A[size-1] won't move
    while (i < size)
    {
        cycleBegin = i;
        t = A[i];
        do
        {
            // Input matrix [r x c]
            // Output matrix 1
            // i_new = (i*r)%(N-1)
```

```
            next = (i*r)%size;
            swap(A[next], t);
            b[i] = 1;
            i = next;

        }  while (i != cycleBegin);

        // Get Next Move (what about querying
        // random location?)
        for (i = 1; i < size && b[i]; i++)
            ;
    }
}

// A utility function to print a 2D array of size
// nr x nc and base address A
void Print2DArray(int *A, int nr, int nc)
{
    for (int r = 0; r < nr; r++)
    {
        for (int c = 0; c < nc; c++)
            printf("%4d", *(A + r*nc + c));

        printf("\n");
    }

    printf("\n");
}

// Driver program to test above function
int main(void)
{
    int A[][4] = {{1, 2, 3, 4},
                  {5, 6, 7, 8},
                  {9, 10, 11, 12}};

    int r = 3, c = 4;

    cout << "Given Matrix:\n";
    Print2DArray((int *)A, r, c);

    transformMatrix((int *)A, r, c);

    cout << "Transformed Matrix:\n";
    Print2DArray((int *)A, c, r);

    return 0;
}
```

Output:

```
Given Matrix:
   1   2   3   4
   5   6   7   8
   9  10  11  12

Transformed Matrix:
   1   8   9
   2   7  10
   3   6  11
   4   5  12
```