Name and, if possible, ID#:

AMERICAN UNIVERSITY OF ARMENIA
College of Science and Engineering
COMP120 Introduction to Object-Oriented Programming
MIDTERM 2 EXAM

Date:	Tuesday, March 24 2015
Starting time:	10:30
Duration:	1 hour 20 minutes
Attention:	**ANY COMMUNICATION IS STRICTLY PROHIBITED**

9/15

Please write down your name at the top of all used pages

Problem 1

The easiest way to implement rotation by 90^0 of a square array is to transpose it and then reverse all its rows separately. Transposing once more after the rotation will result in vertical flip – the top row will appear at the bottom, the second row will become the last but one, etc. Write a C++ function ***void flip(int *a2D, int size)*** that takes as its argument a pointer to the first element of a square array ***int *a2D*** of the specified ***int size*** and flips it vertically. Use already implemented functions ***void reverse(int a1D[], int length)*** and ***void transpose(int *a2D, int size)***:

```
void reverse(int a1D[], int length)
{
      for (int i = 0; i < length / 2; i++)
            swap(a1D[i], a1D[length - 1 - i]);
}

void transpose(int *a2D, int size)
{
      for (int row = 0; row < size; row++)
            for (int col = row + 1; col < size; col++)
                  swap(a2D[row * size + col], a2D[col * size + row]);
}
```

```
void flip(int *a2D, int size)
{
  transpose(*a2D, size);
  reverse(a1D[], length);
  transpose(*a2D, size);
} // by calling these functions in this order we do a vertical flip;
// the problem is reverse function that get different values
from transpose function.
```

3

OOP.MT2.240315.1099

Use the backside, if needed

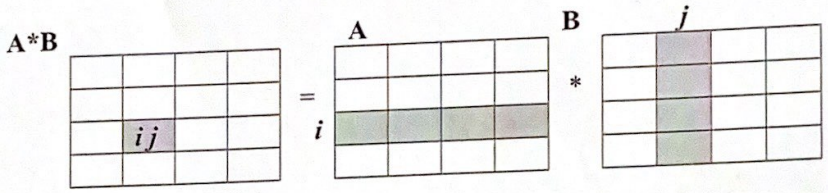
Name and, if possible, ID#:

Problem 2

Using functions ***transpose()*** from **Problem 1** and ***scalar()*** from below, write a C++ function ***void mult(int *a2D, int *b2D, int *product, int size)*** that takes as its arguments pointers to the first elements of square arrays ***int *a2D*** and ***int *b2D*** of the same specified ***int size***, computes their product and saves it in another square array of the same size, the pointer to the first element of which is given by ***int *product***. Each element p_{ij} in the i^{th} row and j^{th} column of the array ****product*** is the scalar product of the i^{th} row of ****a2D*** and j^{th} column of ****b2D*** and is calculated by the expression: $p_{ij} = \sum_{k=0}^{size-1} a_{ik} b_{kj}$

both of them should be row or column to calcalate the scalar;

```
int scalar(int a[], int b[], int length)
{
      int result = 0;
      for (int i = 0; i < length; i++)
            result += a[i] * b[i];
      return result;
}
```




```
void mult(int *a2D, int *b2D, int *product, int size)
{
    int array[size][size];   int *array = new int[size*size];

    transpose(*b2D, size); // to make it a row (B or J)
                           see Problem 1

    scalar(*a2D, *b2D, size);

    for(int i=0; i<size; i++){
        for(int j=0; j<size; j++){
            array[i][j] = scalar(*a2D, *b2D, size);
                                  index->
        }
    }
    delete[] array;
}

// function scalar count the result and write it into array[i][j];
```

3

Name and, if possible, ID#:_____

Problem 3

Using, if you wish, ***segment()*** and ***rotate()*** functions from the **C++ Reference Functions** section, write a C++ function ***void spiral2(int *a2D, int even_size)*** that takes as its argument a pointer to the first element of a square array ***int *a2D*** of the specified even size ***int even_size*** and fills its top-left and bottom-right quadrants with spirals of successive values from *1* to ***even_size*** ^{2} / *4* . The remaining two quadrants are filled with zeros. Each spiral propagates horizontally toward the array center, then vertically toward the center, then in opposite directions horizontally and vertically, and so on. Obviously, the spirals do not cross the central axes. A shaded example is shown below:

```
int* segment(int *start, int length, int direction, int increment)
{
      for (; length > 0; length--)
      {
            *(start + direction) = *start + increment;
            start += direction;
      }
      return start;
}
```

1	2	3	0	0	0
8	9	4	0	0	0
7	6	5	0	0	0
0	0	0	5	6	7
0	0	0	4	9	8
0	0	0	3	2	1

```
int array2D[even_size][even_size];
for(int i=0; i< even_size; i++){
    for(int j=0; j< even_size; j++){
        array2D[i][j]=0; // array full of 0;
    }
}
```

```
int *start = array2D[0][0];
*start = 1; // 1   start + 1 = *start += 1; // 2   start + 1 = *start += 1; // 3
start + even_size = *start += 1; // 4   start + even_size = *start += 1; // 5
start - 1 = *start += 1; // 6   start - 1 = *start += 1; // 7
start - even_size = *start += 1; // 8   start + 1 = *start += 1; // 9
```

up left corner

```
start + (4 * even_size) + 4 = *start = 1; // 1
start - 1 = *start += 1; // 2   start - 1 = *start += 1; // 3
start - even_size = *start += 1; // 4   start - even_size = *start += 1; // 5
start + 1 = *start += 1; // 6   start + 1 = *start += 1; // 7
start + even_size = *start += 1; // 8   start - 1 = *start += 1; // 9
```

down right corner

3

Use the backside, if needed

OOP.MT2.240315.H059