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**

14/15

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)***:

~~const 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);
    for (int i = 0; i < size; i++)
        reverse (a2D[i], size);
    transpose (a2D, size);
}

void main { const int size; cin>>size;
    int az2D [size; size], *a2D;
    a2D = &az2D[0][0];
    flip (a2D, size);
}
```

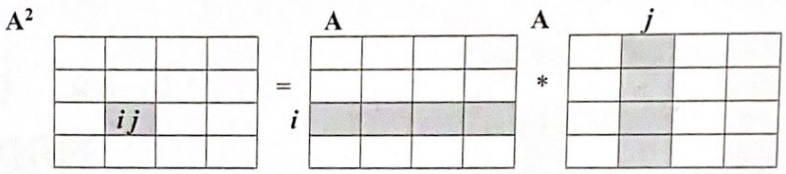
4

Name and, if possible, ID

Problem 2

Using functions ***transpose()*** from **Problem 1** and ***scalar()*** from below, write a C++ function ***void square(int *a2D, int *product, int size)*** that takes as its argument a pointer to the first element of a square array ***int *a2D*** of the specified ***int size***, computes its square (multiplies it by itself) 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 and j^{th} column of the array ****a2D*** and is calculated by the expression: $p_{ij} = \sum_{k=0}^{size-1} a_{ik} a_{kj}$ ~~const int size = ?;~~

```
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 square (int *a2D, int *product, int size) {
    int transposedA2[size][size], *transposeA;
    transposeA = &transposedA2 [0][0];
    transpose(transposeA, size);
    for (int i=0; i<size; i++)
        for (int j=0; j<size; j++)
            transposedA2[i][j] = a2D[i][j];
    transpose (transposeA, size);
    for (int m=0; m<size, m++)
        for (int n=0; n<size, n++)
            product[m][n] = scalar(a2D[i], transpose[j], size);
}

void main { const int size; cin>>size;
    int array2D[size][size], *a2D, product_array[size][size], *product;
    a2D = & array2D[0,0];
    product = & product_array[0][0];
    square (a2D, product, size);
}
```

5

Use the backside, if needed

OOP.MT2.240315.H092

Name and, if possible, ID [illegible]

Problem 3

Using functions ***segment()*** from below and ***rotate()*** from **Problem 1**, 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 it with two spirals of ***zeros*** and ***ones***. The entire first row starting from the first element is filled with ***zeros*** and, symmetrically, entire last row starting from the last element is filled with ***ones***. Then, the entire last column, except the last element, is filled with ***zeros*** and, symmetrically, the entire first column, except the first element – with ***ones***. And so on, until the central elements are reached. 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;
}
```

0	0	0	0	0	0
1	1	1	1	1	0
1	0	0	0	1	0
1	0	1	1	1	0
1	0	0	0	0	0
1	1	1	1	1	1

```
~~segment{~~        int K = 1;
} start = ~~...~~ 1;
segment (start, size·size -1, 1, 0);
a2D[0][0] = 0;
start = & a2D[0][0];
~~while ((size-2)/2 >= 1)~~
segment (start, size -1, 1, 0);
while((size-2)/2 >= 1) {

    ~~seg~~ ~~..~~ segment (start, size - 2, K·size, 0);
    segment (start, size-2, K·(-1), 0);
    K *= -1; //K=-K;

}
```

5

OOP.MT2.240315.H092

Use the backside, if needed