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**

8/15

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. Write a C++ function ***void rotate(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 rotates its. 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 rotate(int *a2D, int size)
{ int * A = new int [size];
  transpose(a2D, size);
   for (int row=0; row< size; row++) {
        for (int col = 0 ; col < size ; col++){
        A[col] = a2D [row][col];
        }
        reverse (A[], size);
        }
        transpose (a2D, size);
} delete [] A;
```

OOP.MT2.240315.M113

3

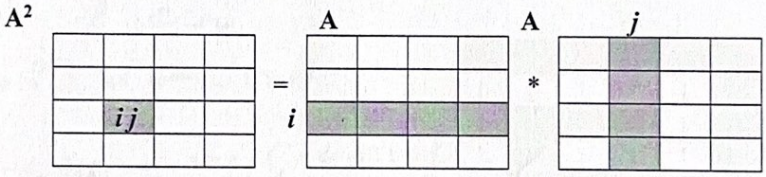
Use the backside, if needed

Name and, if possible, ID#:____

Problem 2

Using functions ***transpose()*** from **Problem 1** and ***scalar()*** from below, write a C++ function ***void 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: $p_{ij} = \sum_{k=0}^{size-1} a_{ik} a_{kj}$

```
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 * = new int [size x size]

transpose(for(int i=1, i=j, i++)

elements are not initialized

transpose (a2D, size)

for (int row = 0; row < size; row++) {

for(int col=0; col< size; col++){

[col] = a2D[row][col];

}

transpose (a2D, size);

}

OOP.MTZ.240315.M#13

← row-by-row loop for elements of product as needed

int scalar (int a[]; int b[], int length)

int result = 0

for (int i=0 ; i<length; i++)

result += a[i] * b[i];

P[i][j] = result return result; }

delete [] a2D;

Use the backside, if needed

Name and, if possible, ID#:

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

```
void spiral2(int *a2D, int even_size)
{ int along[4] = {-1, even_size, 1, -even_size};
  direction = 0;

  *a2D = 1
  for (int length = 1; length < even_size; length++)
  {
    for(int step = 0; step < length; step++)
    *(a2D + along[direction]) = *a2D + 1;
    a2D += along[direction];
  }
  for (int step = 0; step < length; step++)
  { *(a2D + along[direction+1]) = *a2D + 1;
    a2D += along[direction+1];
    direction = 2 - direction;
  }
  for(int step = 1; step < even_size; step++)
  { *(a2D + along[direction]) = *a2D + 1;
    a2D += along[direction];
  }
}
```

OOP.MT2.240315.mm

3

Use the backside, if needed