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)
{
  int * b = new int [size];
  transpose (a2D,size);
   for (int i=0; i<size; i++)
   {
    for (int j=0; j<size; j++)
    { b[j]=a2D[i][j];
    }
    reverse ( b[], size);
   }
   transpose(a2D,size);
}
```

OOP.MT2.240315.L123

← delete [] b;

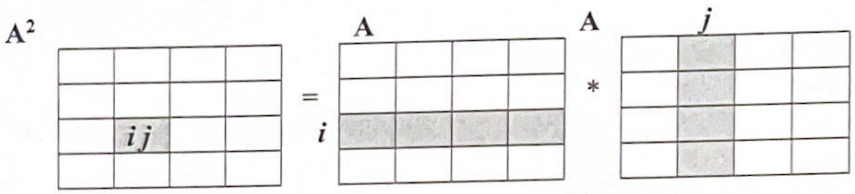
3

Use the backside, if needed

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 *a = new int [size];
int *b = new int [size];
   for(int i=0; i<size; i++)
  {
    for(int j=0; j<size; j++)
   {
    a[j] = *(a2D + i*size + j);
    transpose(a2D,size)-?
    b[j] = *(a2D + i + j*size);
    *(product + i*size + j) = scalar(a[j], b[j], size
   }
  }
  delete [] a;
  delete [] b; - see Problem 1
```

OOP.MPZ-240315.1123

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 a[6]; - not initialized
   for (int length = 0; length < even_size; length++)
   {
      for (int step = 0; step < length; step++)
      {
         *(start + a[direction+1]) = *start + 1;
         start += a[direction+0];
      }
      direction = 2 - direction;
   }
   for (int step = 1; step < even_size; step++)
   {
      *(start + a[direction]) = *start + 1;
      center += a[direction];
   }
}
```

3

OOP.MI2_2403-15.4123