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. 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);
   for (int row=0; row<size; row++)
      ~~array~~[ int *ptr = new int[size]
      reverse(*ptr , size)
   transpose(a2D, size)
}  delete [] ptr;
```

elements of *ptr are not initialized

row 0 → 1 2 3 4
row 1 → 5 6 7 8
row 2 → 9 10 11 12
row 3 → 13 14 15 16

reverse each row

4 3 2 1
8 7 6 5
12 11 10 9
16 15 14 13

transpose

16 15 14 13
12 11 10 9
8 7 6 5
4 3 2 1

OOP.MT2.240315.PM117

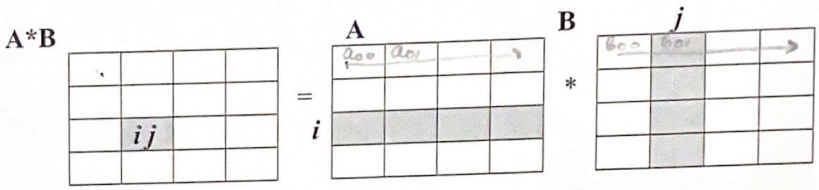
Use the backside, if needed

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: $p_{ij} = \sum_{k=0}^{size-1} a_{ik} b_{kj}$

```
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)
{   transpose ( b2D, size )
    *ptr = i;
    for ( int row = 0 ; row < size; row++)
        int sum = 0;
        scalar (*ptr, size) ;
}
```

($a_{00} * b_{00} + a_{01} * b_{01} + \ldots$)

good idea - wrong implementation;
~~row by row~~ assignment is needed

OOP.MT2-240315.M11J

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 *start = 1, direction = 1,
        increment = 1;
    for (int length=0; length < even_size; length++)
        segment (*start, length, direction, increment)
            cin >> "0";
    for (int length = even_size, length > 0; length --)
        int direction =-1, increment = -1;
        segment( *start, length, direction, increment)
            cin >> "1";
}
```

assignement is needed

too few explicit calls of segment()

2

OOP.MT2.240315.M117