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);
    for (int j=0; j<size; j++)
        reverse (&a2D[j], size);   // *(a2D + j) also possible
    transpose (a2D, size);
}
```

this is equally wrong

4

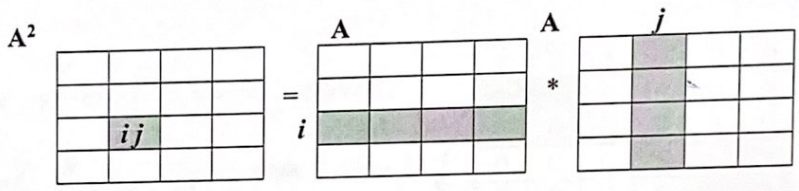
*Use the backside, if needed* OOP.MT2.240315.M105

**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 *a2p, int *product, int size) {
  int b2p = new int [size][size]; //copy of a2p
  for (int i=0; i<size; i++)
      for (int j=0; j<size; j++)
          b2p[i][j] = a2p[i][j];

  transpose (&b2p, size);

  for ( i=0; i<size; i++)
      for(j=0; j<size; j++)
          product[i][j] = scalar (a2p[i], b2p[j], size);
}
```

See Problem 1

2

OOP. MT2. 240315. M205

**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 |

// This is not computer science, this is madness.

~~this is a toy model of~~

```
void spiral2 (int *a2D, int even_size) {
~~int diff=2;~~
~~for(int diff=2; diff<(even_size)/2;~~
int *a;
a = segment (a2D, even_size/2-1, 1, 1);
a = segment (a, even_size/2-1, even_size, 1);
a = segment (a, even_size/2-1, -1, 1);
a = segment (a, even_size/2-2, even_size, 1)
a = segment (a, even_size/2-3, 1, 1)
}
```

This is a toy model of a routing algorithm and memory management – the perfect Computer Science

// we can use the same algorithm, coupled with rotate(), to fill up the bottom right segment. perfect!

// this code only applies to the case even_size = 6, unfortunately, I couldn't write the code for the general case.

It's OK

3