

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**

Please write down your name at the top of all used pages

**Problem 1**

The easiest way to implement rotation by  $90^\circ$  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++)
        reverse(a2D[row], size);
    transpose(a2D, size);
}
```

OOP.MT2.240315.H090



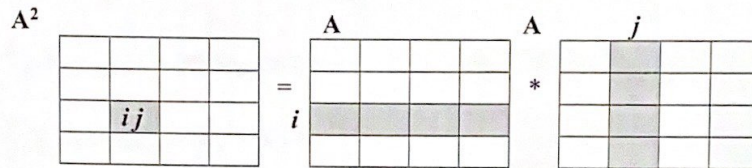
Name and, if possible, ID#:

## 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  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the array *\*product* is the scalar product of the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the array *\*a2D* and is calculated by the

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

```
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 *b2D = new int[size * size];
        for (int row = 0; row < size; row++)
            for (int col = 0; col < size; col++)
                b2D[row][col] = a2D[row][col];
        transpose(b2D, size);
        for (int i = 0; i < size; i++)
            for (int j = 0; j < size; j++)
                product[i][j] = scalar(a2D[i], b2D[j], size);
        delete [] b2D;
    }
}
```

see Problem 1

3

OOP.MT2.240315.H090



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 size size)
{
    int array1[4] = {1, size, -1, -size},
    int array2[4] = {1, -size, -1, size},
    int direction = 0;
    *a2D = 0;
    a2D = segment(a2D, even size, array1[direction], 0)
    for (int length = size - 2; length > 0; length -= 2)
    {
        a2D = segment(a2D, length, array1[(direction+1)%4], 0);
        a2D = segment(a2D, length, array1[(direction+2)%4], 0);
        direction = 2 - direction;
    }
    a2D = a2D + size - 1; perfect!
    direction = 0;
    *a2D = 1;
    a2D = segment(a2D, 1, array
    for (int length = 2; length < size - 1; length += 2)
    {
        a2D = segment(a2D, length, array2[direction], 0);
        a2D = segment(a2D, length, array2[(direction+1)%4], 0);
        direction = 2 - direction;
    }
    segment(a2D, size - 1, array2[0], 0);
}
```

Use the backside, if needed