

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° 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 (int *a2D, int size)
    for (j=0, j<sizelength, j++) {
        reverse (int a1D[], int length) {
    }
    transpose (a2D, size);
```

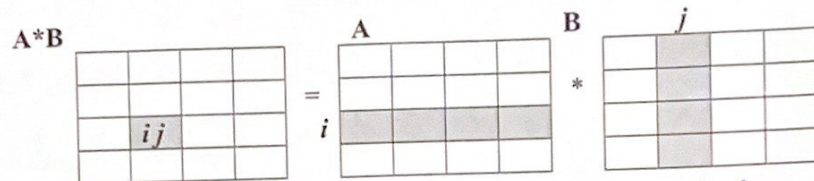
Name and, if possible, ID#:

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

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

```
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(~~a2D~~, size);

for (i=0; i < ~~length~~^{size}; i++){

for (j=0; j < ~~length~~^{size}; j++){

scalar(int a[i], int b[j], int length)

~~product[i][j]~~ product[i][j] = result;

}

transpose(~~b2D~~, size)

see Problem 1

2

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;
}
```

```
void spiral2(int *a2D, int even_size)
{
    a2D = segment(a2D, int length, int direction, int increment)
    rotate(a2D, size)
    rotate(a2D, size)
    a[0][0] = 1;

```

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	1	1	1	1	1	0
2	1	0	0	0	1	0
3	1	0	1	1	1	0
4	1	0	0	0	0	0
5	1	1	1	1	1	1

[0][0] [4][1]
[0][5] [2][1]
[4][5]

for (length = size; length > 0; length--) ?

```
{
    a2D = segment(a2D, length, direction, 0)
    a2D = segment(a2D, length, direction, 0)

```

OOP.MT2-240315-1131

```

{
    segment(a2D, 5, 1, 0)
    segment(a2D, 5, 1, 0)
    segment(a2D, 4, 6, 0)
    segment(a2D, 4, 6, 0)
    segment(a2D, 4, -1, 0)
    segment(a2D, 4, -1, 0)
    segment(a2D, 2, -6, 0)
    segment(a2D, 2, -6, 0)
    segment(a2D, 2, 1, 0)
    segment(a2D, 2, 1, 0)

```

3) a[0][5] = 1
the same again
with ones

Use the backside, if needed

```

2) rotate(a2D, size)
    rotate(a2D, size)

```