

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 ( a2D, size);
    for (int row=0; row<size; row=row+size)
        reverse ( a2D, size)
        reverse (a2D [row], size);
    transpose ( a2D, size);
}
```

OOP.M12. 240315.4088

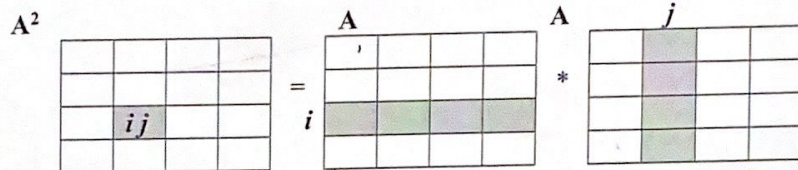
Name and, if possible, ID#:

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

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

```
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 *a2D2 = new int[size * size];
    transpose(a2D2, size);
    for (int i = 0; i < size * size; i++)
    *product[i] = scalar(a2D[i * size], a2D2[i * size], size);
}
```

```
int *a2D2 = new int[size * size];
for (int m = 0; m < size * size; m++)
    *a2D2[m] = *a2D[m];
transpose(a2D2, size);
```

```
for (int i = 0; i < size; i++)
    for (int j = 0; j < size; j++)
        *product[i * size + j] = scalar(a2D[i * size], a2D2[j * size], size);
delete [] a2D2;
perfect!
```

see Problem 1

Use the backside, if needed

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

Void spiral (int *a2D, int even_size)

```
{ a2D for (int e=0; e < even_size * even_size; e++)
```

~~*a2D~~ [e] = 0;

*a2D = 1;

int Dir [4] = {Size, -1, -Size, +1};

id m = 0; *p = a2D + even_size * even_size - 1;

a2D = segment (a2D, even_size / 2, +1, +1);

*p = 1; p = segment (p, even_size / 2 - 1, -1, +1);

for (int i = 0; i < even_size / 2; i++)

{ a2D = segment (a2D, even_size / 2 - i - 1, Dir [m], 1);

a2D = segment (a2D, even_size / 2 - i - 1, Dir [m+1], 1);

~~m = 2 - m;~~ p = segment (p, even_size / 2 - i - 1, Dir [2 - m], 1);

p = segment (p, even_size / 2 - i - 1, Dir [2 - m + 1], 1);

m = 2 - m;

}

Use the backside, if needed