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)
  {                    → nothing:)
    transpose(*a2D, size);
    for (int i=0, i< ~~a2D~~ size , i++) a2D+i*size
        { ~~a2D reverse[i][]~~, reverse(a2D[i][], size);
    }
  }
```

this method of solving problem is not so interesting, so I wrote on the next page my own solution.

this would not be interesting, of many errors

4

OOP.MT2.240315.M111

```
void flip(int *a2D, int size) {  int a2Dnew[size][size];
 for(int i=0, i<size, i++) {
     for(int j=0, j<size, j++) {
  a2Dnew[j][i] = a2D[i][j];
  }
 }
 int a2Dfinal[size][size];
  for(int a=0, a<size, a++) {
      for(int b=0, b<size, b++) {
      a2Dfinal[a][b] = a2Dnew[a][size-1->b];
    }
   }
 return a2Dfinal
  for(int  x=0, x<size, x++){
      for(int y=0, y<size, y++){
       a2D[x][y] = a2Dfinal[x][y];
     }
    }
  return  a2D;
}
```

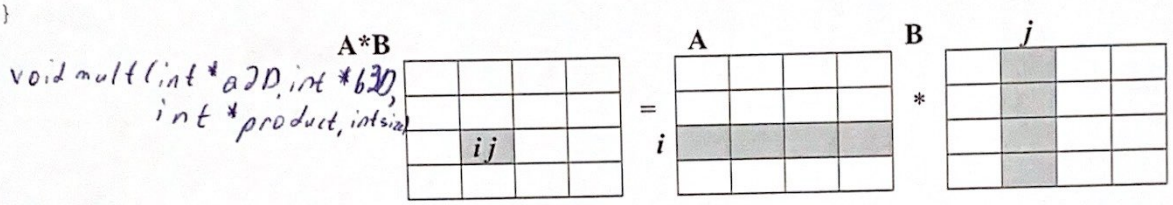
this method permitted so many mistakes — so, it's not so interesting

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)

{

```
for (int i=0, i<size; i++) {
    for (int j=0, j<size, j++) {
    product[i][j] = scalar[a2D[i], b2D[j], size];
    }
}
```

transpose(b2D, size) is required

~~return product~~

}

this one is also not interesting so I will not lose opportunity to solve it ~~in m~~ without any other function.

3

OOP.m12.240215.m1

2.

```
void mult(int *a2D, int *b2D, int *product, int size)
{   for(int i=0; i<size, i++){
        for(int j=0, j<size, j++){
            ~~product[i][j] =~~
            ~~int a = 0;~~
            int b = 0;
            for(int x=0; x<size, x++){
                ~~for(int y=0; y<size, y++){~~
                b += a2D[i][x] * b2D[x][j]
            }
            product[i][j] = b;
    }
}
```

do you ever seen more complicated solution? I don't think so :)

again, this complicated erroneous solution isn't graded :)

oop.m12. 2403 · m11

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;
}
```

00 01 02 12 22

00 01 02

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 spiral2(int *a2D, int even_size)
{  int i = ((even_size)*(even_size))/4;
   for (int a = 1; a <= i; a++)
      if (a <= even_size/2)
         a2D[0][a] = a;
      if ((a < even_size) && (a > even_size/2))
         a2D[even_size/2][a - even_size/2] = a;
      if ((a >= even_size) && (a <= even_size + 1))
         a2D[even_size/2][
```

see the last page

OOP MT2. 240315. M11

```
void spiral2(int *a2D, int even_size){
    int a = ((even_size)*(even_size))/4;
    int b = even_size; for(int q=0; q<a, q++){
      for(int j=0, j<even_size, j++){
        a2D[0][j] = q; }
```

I can't solve this problem by writing program that is my fault.

But I understand the logic of problem.

that firstly the array increase in → direction be even_size after that there is a symetry.

if we denote array by.

a2D[i][j]

Firslty I is 0 and j increase from 0 to even_size - 1

after i increases by even_size-1 then I remains the same and j decrease by even_size

it increase and decrease in this manner

a2D[i][j]

	+	even_size - 1
+		even_size - 1
	−	even_size - 1
−		even_size - 2
	+	even_size - 2
+		even_size - 3
	−	even_size - 3
−		even_size - 4
	+	
+		
	−	
−		

OOP.M12.240315.211111

there is a pattern

increase, increase

decrease, decrease

2