Name and, if possible, ID#: Aus

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 90^0 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 i=0; i<size; i++){
       reverse(a2D[0], size);
    }
    transpose(a2D, size);
}
```

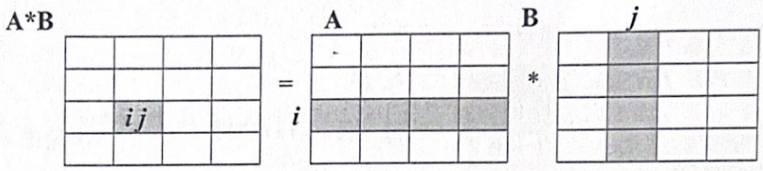
OOP.MT2.240315.M110

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){
  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]);
    }
  }

  transpose(b2D, size);

}
```

OOP.MT2.240315.MHO

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 spiral2(int *a2D, int even_size){
  for(int i=0; i<even_size; i++){
    for(int j=0; j< even_size; j++){
      *a2D[i][j]=0;
    }
  }
int increment = even_size * even_size / 4;
segment.
segment(&a2D[even_size/4+1][even_size/4+1], even_size/2-1,
for(int i=0; i<even_size/2; i++){
  segment(&a2D[even_size/4+1][even_size/4+1], even_size/2, 1,
                                  even_size*even_size/4);
}
for(int j=0; j< even_size/2; j++){
  segment(&a2D[even_size/4+1][even_size/4+1], even_size/2, 1,
                                  even_size*even_size/4);
}
}
```

See Problem 2

Too few explicit calls of segment()

3

OOP-MT2.240315.M100

Name and, if possible, ID#:_____

Problem 2

Colors in Java can be represented by objects of type ***Color***. Each such object contains the ***red***, ***green*** and ***blue*** components of the corresponding color as integer values from **0** to **255**. Consider below a Java code that creates and initializes a rectangular array of ***Color*** type:

```
import java.util.Scanner;
import java.awt.Color;

public class Colors {

public static void main(String args[]) {
      Scanner in = new Scanner(System.in);

// Read number of rows and columns and create a Color array of such size
      Color[][] c = new Color[in.nextInt()][in.nextInt()];

// For each element read the red, green and blue components as integers and
// create a Color object by calling Color(int, int, int) constructor
      for (int row = 0; row < c.length; row++)
            for (int col = 0; col < c[0].length; col++)
int z;              c[row][col] = new Color(in.nextInt(), in.nextInt(), in.nextInt());
// TO BE CONTINUED
}
}
```

Continue with a Java code that creates another array ***Color[][] g*** of the same size and fills it with gray equivalents of the colors from the array ***Color[][] c***. To get a grey equivalent of a given color ***c[i][j]***, it is enough to construct a ***Color*** object, whose red, green and blue components all are equal to the calculated average of red, green and blue components of the initial ***c[i][j]***. Use ***int getRed()***, ***int getGreen()*** and ***int getBlue()*** methods of class ***Color***.

```
Color[][] g = new Color[c.length][c[0].length];
for (int x = 0; x < c.length; x++){
      for(int y=0; y<c[0].length; y++){
int z= c[x][y].getRed() + c[x][y].getGreen() + c[x][y].getBlue())/3;
g[x][y] = new Color(z, z, z);
}
}
```

6

OOP.MT1-130215.M111

Name and, if possible, ID#:_____

Problem 3

Similar to files, strings also can be related to streams in C++, this time using ***stringstream*** objects. Particularly, it is enough to create an object of type ***istringstream*** to organize formatted reading from a string. Consider, for example, a C++ code below:

```
#include <string>
#include <sstream>
#include <iostream>
using namespace std;

void main()
{
	string text = "Before_increment: 199999999", word;
	int num;
	istringstream tokens(text);

	tokens >> word >> num;
	cout << "After " << word.substr(7) << num + 1 << endl;
}
// After increment:200000000
```

Write a C++ function ***double value(string expression)*** that takes as its argument a string representing an arithmetic expression, evaluates it and returns its value. The expression includes only '+' and '–' operations and double operands, both positive and negative. The operands and operations are delimited by spaces.

For example, ***value("5.1 – -0.7 + 1.2")*** results in ***7.0***.

```
double value(string expression) {
istringstream tokens(stringexpression)
int spaces=0;
for(int x=0; x<stringexpression.length; x++)
  { if (stringexpression.charAt(x) == " ")
        spaces++;
string words[spaces];
for (int x=0; x<spaces; x++)
   { tokens >> words[x];
   }
For (int x=0; x<spaces; x++)
  { if (words[x] == "+")
    {words[x] = words[x-1] + words[x+1];
     words[x-1] = " ";
     words[x+1] = " ";
    }
     if (words[x] == "-")
    { words[x] = words[x-1] - words[x+1];
      words[x-1] = " ";
      words[x+1] = " ";
    }
}
```

4

OOP.MT1.180215.M111