

Name and, if possible, ID#:

AMERICAN UNIVERSITY OF ARMENIA
College of Science and Engineering
COMP120 Introduction to Object-Oriented Programming

MIDTERM 1 EXAM

Date: Tuesday, February 17 2015
Starting time: 10:30
Duration: 1 hour 20 minutes
Attention: **ANY TYPE OF COMMUNICATION IS STRICTLY PROHIBITED**
Please write down your name at the top of all used pages

Problem 1

Square arrays can be rotated by 90° , say, in clock-wise direction. For example:

| | | | | | | | | | | |
|----|----|----|----|----|---|----|----|----|----|---|
| 1 | 2 | 3 | 4 | 5 | → | 21 | 16 | 11 | 6 | 1 |
| 6 | 7 | 8 | 9 | 10 | | 22 | 17 | 12 | 7 | 2 |
| 11 | 12 | 13 | 14 | 15 | | 23 | 18 | 13 | 8 | 3 |
| 16 | 17 | 18 | 19 | 20 | | 24 | 19 | 14 | 9 | 4 |
| 21 | 22 | 23 | 24 | 25 | | 25 | 20 | 15 | 10 | 5 |

The easiest way to implement the rotation by 90° is to transpose the initial square array and then to reverse all its rows separately. Write a Java method `void rotate(int[][] array2D)` that takes as its argument a square `int[][] array2D` and rotates its. Use already implemented methods `void reverse(int[] array1D)` and `void transpose(int[][] array2D)`:

```
public static void reverse(int[] array1D) {
    for (int i = 0; i < array1D.length / 2; i++) {
        array1D[array1D.length - 1 - i] += array1D[i];
        array1D[i] = array1D[array1D.length - 1 - i] - array1D[i];
        array1D[array1D.length - 1 - i] -= array1D[i];
    }
}

public static void transpose(int[][] array2D) {
    for (int row = 0; row < array2D.length; row++)
        for (int col = row + 1; col < array2D.length; col++) {
            array2D[row][col] += array2D[col][row];
            array2D[col][row] = array2D[row][col] - array2D[col][row];
            array2D[row][col] -= array2D[col][row];
        }
}
```

```
public static void rotate(int[][] array2D) {
    reverse transpose (array2D); reverse (array2D);
    for (i=0; i < array2D.length; i++)
        for (j=0; j < array2D.length; j++)
            array2D[i][j] = - array2D[j][i];
}
```

00P.MT1, 170215.H094

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++)
                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*.

5

```
Color[][] g;

public int getRed() {
    return c.getRed();
}

public int getGreen() {
    return c.getGreen();
}

public int getBlue() {
    return c.getBlue();
}

int colorGray = new Color(
    (c.getRed() + c.getGreen() + c.getBlue()) / 3,
    (c.getRed() + c.getGreen() + c.getBlue()) / 3,
    (c.getRed() + c.getGreen() + c.getBlue()) / 3);

Color[][] g = new Color[row][col];

for (int row = 0; row < g.length; row++)
    for (int col = 0; col < g[0].length; col++)
        g[row][col] = new Color(colorGray, colorGray, colorGray);
```

Use the backside, if needed

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 c) {

string string;
~~double sum = 0;~~ double sum = 0; ~~sum + c < sum;~~

char string;
while (c <> '\0'; c <> '\n') {

cout << sum << sum + c << endl; Why concatenating?
return -?

QEP.MT1-130215.1094