Section, Name and ID#:

AMERICAN UNIVERSITY OF ARMENIA
College of Science and Engineering
CS 120 Introduction to Object-Oriented Programming
MIDTERM EXAM

13

Date / Time: Friday, March 17 2017 at 17:30
Duration: 2 hours
Attention: **ANY TYPE OF COMMUNICATION IS STRICTLY PROHIBITED**
Write down your section, name and ID# at the top of all used pages

Participation:

Problem 1: Consider below a C++ function ***float kahan(float num1, float num2, float& compensation)*** that implements the ***Kahan Summation Algorithm*** for high-precision compensated summation of two float arguments ***float num1*** and ***float num2***:

```
float kahan(float num1, float num2, float &compensation)
{
        float result;
        num2 -= compensation;
        result = num1 + num2;
        compensation = (result - num1) - num2;
        return result;
}
```

Using this function, write a C++ function ***float pi(int n)*** that computes the value π by the following formula:

$$\pi = 2\sum_{k=0}^{n}\frac{(2k-1)!!}{(2k)!!(2k+1)} = \frac{2}{1*1}+\frac{1}{2}*\frac{2}{3}+\frac{1*3}{2*4}*\frac{2}{5}+\frac{1*3*5}{2*4*6}*\frac{2}{7}+\cdots$$

Recall that **n!!** is the product of odd numbers from *1* to *n*, if *n* is odd; and is the product of even numbers from *2* to *n*, if *n* is even. The double factorial of non-positive numbers equals to *1* by definition.

The initial value of ***float compensation*** is ***0.0***.

```
mul
int0;
int factorial (int k) {
for (int i=k; i>0; i-2)
mul *= i
}
```

```
int mul;
int factorial (int k) {
if (k>0) {
for (int i=k; i>0; i-2) {
mul *= i;
} return mul;
```



```
else {return 1)
```

OOP.MT.170317.H049

```
float pi(int n) {
    int pie;
for (int k=0; k≤n; k++) {

pie += factorial(2*k-1 /(factorial(2*k)*2*k+1);

}
return 2*pie;
```

3

Section, Name and ID#:______________________________

Problem 2: Write a Java method ***public static double[] lin(double[] data)*** that takes as its argument an array of data points ***double[] data***, and returns a two-element array – the first element being the slope of the linear regression and the second element being the intercept. The linear regression approximates the data points by the linear formula

$$y = k\,x + b,$$

where the slope k and the intercept b are computed as

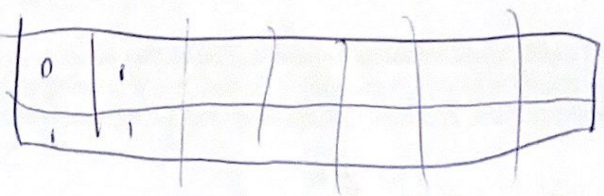
$$k = \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2}, b = \overline{y} - k\,\overline{x}$$

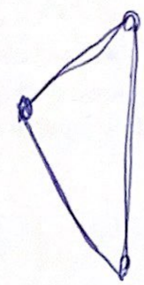
Here $\overline{x}$ is the mean of the x coordinates, $\overline{y}$ is the mean of the y coordinates, $\overline{x^2}$ is the mean of the squares of the x coordinates, and $\overline{xy}$ is the mean of the products of the x and y coordinates. Use the element indices of the array ***double[] data*** as x coordinates and the element values as y coordinates. You may assume and use the method double ***mean(double[] a)***.

```
public static double[]lin(double[] data){
double mean_x;
    for(int i=0; i<=data.length; i++) {
        mean_x +=i;
    }
double mean_y;
    for (i=0; i<data.length; i++) {
      mean_y +=data[i];
    }
~~double k=(mean_x*mean_y-~~
double mean_xy;
    for (i<0; i<data.length; i++) {
    mean_xy +=data[i]*(i+1)
    }
```

Use the backside, if needed

OOP.MT.170317.H049





problem 2. mean(x^2)

```
double k = (mean_xy - (mean_x * mean_y)/ (mean_x2*mean_x
- mean_x * mean_x);

double b = (mean_y* - k * mean_x; )*count
int[]linear = new int[2];
linear[0] = k ;
linear[1] = b;
return linear;
```

4

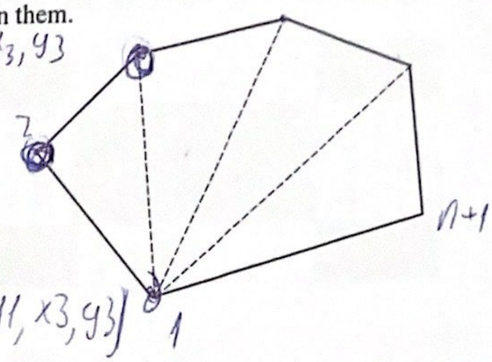
Section, Name and ID#: ______________________________

Problem 3: Write a Java function ***public static double area(double[][] vertex)*** that takes as its argument a ***2-by-n*** array of a convex polygon's vertex coordinates ***double[][] vertex*** – the ***x*** coordinates in the first row and ***y*** coordinates in the second row. It returns polygon's area as follows:

1. Divides the polygon into triangles by connecting the ***first*** vertex with the ***n***th and ***(n+1)***st vertices;
2. Adds the areas of the constructed triangles using the formula $area = \sqrt{p(p-a)(p-b)(p-c)}$, where ***a, b*** and ***c*** are the sides and $p = (a + b + c) / 2$.

You may assume and use a method ***double dist(double x1, double y1, double x2, double y2)*** that takes as its arguments coordinates of two points and returns the distance between them.

x1,y1,x2,y2 -> x3,y3




```
public static double p( ) {
int p;
P= dist(x1, y1, x2, y2) +
dist(x2, y2, x3, y3) + dist(x1,y1,x3,y3)

public static double area (double[][] vertex){
a = vertex[1].length
double P; double area;
 for(i=1; i<a+1; i++) {
  P= dist(vertex[0][0], vertex[1][0], vertex[0][i]
vertex[1][i], vertex[0][i+1] vertex[1][i+1]

area+= sqrt(p*(p-dist(vertex[0][0], vertex[1][0]
vertex[0][i] vertex[1][i])) *(p-dist(vertex[0][0],
vertex[1][0], vertex[0][i+1], vertex[1][i+1]) * p-
```



```
dist(vertex[0][i], vertex[1][i], vertex[0][i+1]
 vertex[1][i+1])) }
```

OOP.MT.170317.H019 4

```
return area;
}
```

Section, Name and ID#:____

Problem 4: Write a Java method ***public static void magic4N(int[][] square)*** that creates a magic square of a *4N-by-4N* size using the following algorithm:

1. Creates an array of the same size as ***int[][] square*** and fills it forward with successive integers assigning ***1*** to the top-left element;
2. Creates anther array of the same size as ***int[][] square*** and fills it backward with successive integers assigning ***1*** to the bottom-right element;
3. Divides the original ***int[][] square*** into 16 blocks of the same size – 4 blocks per row and column. In the on-diagonal (shaded) blocks copies the elements from the first array, and in the off-diagonal blocks copies the elements from second array.

1	2					7	8
9	10					15	16
		19	20	21	22		
		27	28	29	30		
		35	36	37	38		
		43	44	45	46		
49	50					55	56
57	58					63	64

		62	61	60	59		
		54	53	52	51		
48	47					42	41
40	39					34	33
32	31					26	25
24	23					18	17
		14	13	12	11		
		6	5	4	3		

```
public static void magic4N (int [][] square) {
int [][] square1 = new int [square[0].length][square[0].length]
int[][] square2 = new int [square[0].length][square[0].length]

for (int i=0; i<square[0].length; i++) {
    for(int j=0; j<square[0].length; j++)
        square1[i][j] = square[0].length * i + j+1
}
for (int i = square[0].length; i>=0; i--) {
    for (int j = square[0].length; j>=0; j--)
    square2[square.[0]length-i][square.length -j] =
```


Use the backside, if needed

```
= square.[0].length · i + j-1.
```

OOP.MT.170317.H049

problem 4

the function by understanding the column chooses the element sort places them in square 2D array

2