Section, Name and ID#:_

12

AMERICAN UNIVERSITY OF ARMENIA

College of Science and Engineering

CS 120 Introduction to Object-Oriented Programming

MIDTERM EXAM

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***.

float pi(int n) { int a, b, c;

for (int i = 1; i < n; i+2) {

if (n % 2 != 0) {

a += 2*(((2*(i) - 1) (2·(i + 2)) / ((2(i) · 2(i+2))(2i+1)));

if (n % 2 = 0) {

a += 2*((2(i+1) - 1)(2·(i+3)) / (((2(i+1))*2(i+3))(·2(i+1)+1)) ; } }

3

return a

Kahan-?

OOP.MT.170317.H001

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

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

Here $\bar{x}$ is the mean of the x coordinates, $\bar{y}$ 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) {
 for (int i = 0; i < data.length(); i++) {
     a[i] = i; c[i] = i*i;
     b[i] = a[i] * data[i]; }
  d = (mean(b) - (mean(a) * mean(data))) /
      (mean(c) - (mean(a) * mean(a)))^2;

  e = mean(data) - (d * mean(a));

  f[0] = d; f[1] = e;

  return f;

  double d, e;
  double[] a, b, c    = new double[double.length()];
  double[] f = new double[2];
```

4

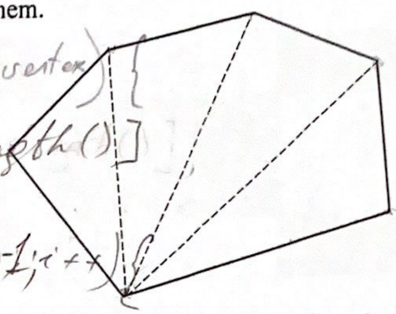
OOP.MT.170317.H001

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.



```
public static double area(double[][] vertex) {
    double[] p, a, b = new double [vertex.length()];
    for (int i = 0; i < vertex.length()-1; i++) {
        a[i] = dist(vertex[0][0], vertex[0][0], vertex[i+1][0],
               vertex[0][i+1]) }
    for (int i=0; i<vertex.length()-2; i++) {
        b[i] = dist(vertex0[i+1], vertex0[i+1], vertex0[i+2],
               vertex0[i+2]) }

    double area;
    for (int i=0 ; i< vertex.length() ; i++) {
        p[i] = (a[i] +a[i+1] +b[i]) /2 ;

        area += sqrt(p[i](p[i] - a[i])(p[i] - a[i+1])(
                p[i] - c[i])));

    return area;
```

3

OOP.MT.170317.H001

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; row
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.

row	1	2					7	8
row	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 magic 4N(int[][] square) {
int[][] a, 6 = new int [square[0].length()] [square.length()]; a[0][0] = 1;
for (int i = 1; i < square.length(); i++) {
a[i][0] = a[i-1][0] + square.length();
for (int j = 1; j < square[0].length(); j++) {
a[i][j] = a[i] [j-1] + 1; }}
for (int i = 0; i < square.length(); i++) {
for (int j = 0; j < square[0].length(); j++) {
6[i][j] = a[square.length() - i][square[0].length() - j]; }}
```

2

OOP.MT.170317.H001