Section, Name and ID#: _____

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) {
    float result1 = 1, result2 = 1, result = 1, resultev = 1;
    for (int k = 0; k <= n; k++) {
        if (k % 2 == 1) {
            for(int i = 1; i <= 2k-1; i = i+2) {
                result1 = result1 * i; }
            for (int j = 2; j <= 2k; j = j+2) {
                result2 = result2 * j; }
            result = result * (result1 / (result2 * (2k+1)))
        } else if (k <= 1) {
            resultev = 1 / (1 * (2k+1));
        }
    result = 2 * result; ? kahan(result, resultev, compensation)
```

Use the backside, if needed

Problem 1 of 4

OOP.MT.170317.M025

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) {
    double[] lin = new
for (int x = 0, x < data.length; x++) {
    int[] xs = new int[];
    data[x] = x;
}
double meanofx = mean[int[] xs];
double meanofy = mean[double[] data];
double result = 0;
for (int x = 0; x < data.length; x++) {
    result = result + [double[x] * x;
}
result = result/data result : data.length;
double meanofx2 = 0;
for (int i=0; i < xs.length; i++) {
    meanofx2 = meanofx2 + i;
}
meanofx2 = meanofx2 : xs.length;
for(int i=0; i<2; i++) {
lin[0]
slope = ((result-(meanofx*meanofy)) : (meanofx2 - Math.pow(meanofx2))
slop
double B = new double,
B = meanofy - slope * meanofx
lin[1] = meanofy - lin[0] * meanofx.
}
return lin;
}
```

Use the backside, if needed

3

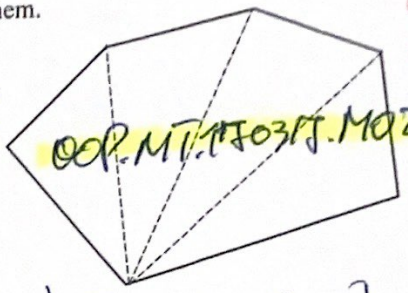
Section, Name and ID#: _____

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.

2



public static double area(double[] vertex){

for(int i=0; i<2; i++){

for(int j=0; j<vertex[i].length; j++)

$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}$

Let me explain how I will do it :)

So, we have 2 fors (I wrote it above), which read the elements of matrix.

We have x_0 and y_0 and other elements.

x_0 and y_0 are the first point.

So we have a function called distance. distance = sqrt($(x_1-x_2)^2 + (y_1-y_2)^2$)

We call this function and find all the distances from (x_0, y_0) to other points.

Then we also compute the distance from 2 successive points such as (x_1, y_1) and (x_2, y_2).

So, in case of 3 points we would have the first point with other two and the other tw with each other.

Then, we simply compute p, p = firstdist+sec+third and Area = $\sqrt{p(p-firstdist)(p-sec)(p-third)}$.

For other triangles we have the same thing.

First one is computed with the second successive points.

So, Area = ~~Area~~ the sum of areas of triangles.

Use the backside, if needed