Section, Name and ID#:_

8

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***.

2

```
float pi (int n) {
int upperFact=1, lowerFact=1, lowerSum=0;
/* (2k-1) is less than 0 and odd, when k=0, that's why we skip non-positive
numbers, as their product is 1. So, to calculate upperFact,
we will start from k=1 */
for (int i = 1; i<(2n-1); i++) {
upperFact = upperFact * kahan((2i-1), 2, 0.0);
}
/* As double factorial of 0 is 1, and 2k is even,
we start calculating it from k=1 */
for (int j=1; j<2n; j++) {
lowerFact = lowerFact *kahan (2j, 2, 0.0);       wrong logic
}
for (int k=0; k <= (2n+1); k++) {
lowerSum += kahan((2k+1), 1, 0.0);
}
return 2* upperFact/(lowerFact * lowerSum);
}
```

Use the backside, if needed

OOP.MT.17O317.MOGs

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)***.

3

a[0] - y   a[1] - y
0 - x   1 - x

```
public static double [] lin (double[] data) {
int   k, b, meanOfY=0, meanOfX, meanOfProd = 0, meanOfXSq = 0;
for (int i = 0; i <=(ArraySize.data() - 1); i++) {
meanOfX+= i;
}
meanOfX = meanOfX : ArraySize.data();
meanOfY = mean (double[] data);
for (int j = 0; j <=(ArraySize.data()-1) ; j++) {
meanOfProd+=(j * data[j]);
}
meanOfProd= meanOfProd : (ArraySize.data());
for (int k=0; k <=(ArraySize.data() -1) ; k++) {
meanOfXSq +=((meanOfXSq*1)*(meanOfXSq+1));
}
meanOfXSq = meanOfXSq : (ArraySize.data());
k = (meanOfProd - meanOfX*meanOfY) : (meanOfXSq -
  - meanOfX*meanOfX);
b = meanOfY - k * meanOfX;
return  {{k, b}};
}
```

?

Use the backside, if needed

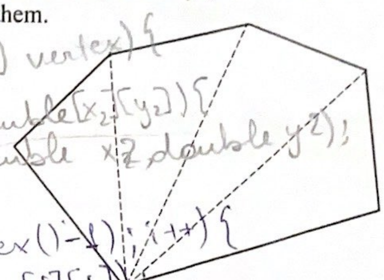
OOP.MT.120317.M085

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 a,b,c,area=0,p;
double distOfPoints(double[x1][y1], double[x2][y2]){
return double dist(double x1, double y1, double x2, double y2);
}
for (int i=1; i<=(ArraySize.vertex()-1); i++){
a=distOfPoints(vertex[0][0], vertex[i][i]);
b=distOfPoints(vertex[0][0], vertex[i+1][i+1]);
c=distOfPoints(vertex[i][i], vertex[i+1][i+1]);
p=(a+b+c):2;
area += math.squere(p*(p-a)*(p-b)*(p-c));
}
return area;
}
```

?

3

OOP.MT.170317.M8CS