Section, Name and ID#:

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

See HG
$\cancel{7.5}$

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:

$$\pi = 16\sum_{k=0}^{n}\frac{(-1)^k}{(2k+1)5^{2k+1}} - 4\sum_{k=0}^{n}\frac{(-1)^k}{(2k+1)239^{2k+1}} = \left(\frac{16}{1*5}-\frac{4}{1*239}\right)-\left(\frac{16}{3*5^3}-\frac{4}{3*239^3}\right)+\left(\frac{16}{5*5^5}-\frac{4}{5*239^5}\right)-\dots$$

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

float pi (int n)
int sum, int m , int d,
{ for (ind k=0, k<n, k++)
m= 16·(pow(-1)k)/(2k+1) ·pow(5,2k+1) -
-4(pow(-1)k)/(2k+1)·pow(239,2k+1)
d=(16·pow(-1)(k+1))/2(k+1)+1)·pow(5,2(k+1)+1) -
-4(pow(-1)(k+1))/(2(k+1)+1)·pow(239,2(k+1)+1)
sum = sum + ~~[illegible]~~ kahan(m,d,0.0),
return sum;
}

3/12=
1.5
see HG

Use the backside, if needed

OOP.MT.170317.M067

Section, Name and ID#:

Problem 2: Write a Java method ***public static double[] expReg(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 exponent of an exponential regression and the second element being the amplitude. The exponential regression approximates the data points by a formula

$$y = a\, e^{mx},$$

where the exponent m and the amplitude a are computed as

$$m = \frac{\overline{xy} - \bar{x}\,\bar{y}}{\overline{x^2} - \bar{x}^2}, a = \bar{y} - m\bar{x}$$

Here $\bar{x}$ is the mean of the x coordinates, $\bar{y}$ is the mean of the natural logarithm of 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 natural logarithm of y coordinates. Use the element indices of the array ***double[] data*** as x coordinates and the element values as y coordinates. For natural logarithm, use the method ***double Math.log()***.

Both result elements are zeros, if at least one data element is non-positive.

```
public static double[] expReg(double[]data) {
 int x1,y1,xy, m,a,x2
 for (int x=0, x<data.length, x++)          negative?
 { xy = xy + x. (double) Math.log(data[x]);
   x1 = x1+x
   y1 = double Math.log(data[x])+y1
   x2 = x2 + pow(x,2);  }
 m = (xy/data.length)-(x/data.length).(y/data.length)
     ------------------------------------------------
     (x2/data length) - pow(x1/data.length,2)
 a = (y1/data length)-m.(x1/data length)
 double[] meh = new double[2];
   double[0]=m;
   double[1]=a;
   return meh;
```

3/2
=1.5

Use the backside, if needed

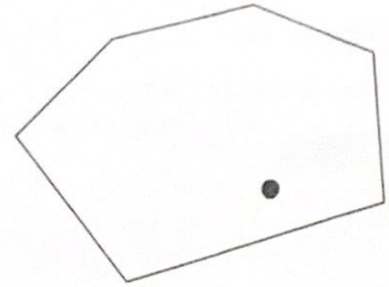
OOP.MT.170317.M067

see HG

Section, Name and ID#: ______________________________

Problem 3: Write a Java function ***public static boolean isInside(double[][] vertex, double x, double y)*** 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, and ***double x*** and ***double y*** coordinates of a point. It checks, if the point is inside the polygon.

Assume and use a method ***boolean toLeft(double x1, double y1, double x2, double y2, double x0, double y0)*** that takes as its arguments coordinates of three points and returns ***true***, if the third point *(x0, y0)* is in the left-hand side, when moving from the first point *(x1, y1)* to the second one *(x2, y2)*; and ***false***, if it is in the right-hand side.



public static boolean isInside
(double[][] vertex, double x, double y)
{ for(int i=0; i<2; i++)
{
for(int j=0; j<vertex[i].length; j++)
{ we should have an if statement
in order to state our conditions
of returning false or true

~~0~~ 1

OOP.MT.180317.m067

Use the backside, if needed

Section, Name and ID#: ______________________________

Problem 4: Write a Java method ***public static void magicOdd(int[][] square)*** that creates a magic square of an ***odd*** size using the following algorithm:

1. The number ***1*** goes in the middle of the top row;
2. All numbers are then placed one ***column to the right*** and ***one row up*** from the previous number;
3. Whenever the next number placement is above the top row, stay in the same column and place the number in the bottom row (note the place of ***2*** instead of the shaded location);
4. Whenever the next number placement is outside of the rightmost column, stay in the same row and place the number in the leftmost column (note the place of ***3*** instead of the shaded location);
5. When encountering an already filled-in square, place the next number directly below the previous number;
6. When the next number position is outside both a row and a column, place the number directly beneath the previous number (note the place of ***7*** instead of the shaded location).

	9	2	7
8	1	6	8
3	5	7	3
4	9	2	

public static void magicOdd(int[][] square)

{ int [][] magic = new int[n][n]

n % 2 = 1

for (ind j = 0, j < square.length, j++)
↳ index of column

} for (ind i = 0, i < square.length, i++)
↳ index of row

we state the odd size

~~row[i+1] = n~~
square [i~~..~~] [j+1] = 1
1st row 2nd column is equal to 1

1

OOP.MT.170317.M067

Use the backside, if needed