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:

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

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

```
for(int k=0, k<n; ++k){ -> function A.
  float res = 0;
  float first = pow(-1, k);
  float pow1 = pow(5,(2*k+1));
  float second = pow((2*k+1), pow1);
  result = kahan(result, first/second, compensation)
}
return result.
```

function ~~with~~ analog to this is for (B)

our pi functions returns 16·A − 4·B

A and B are floats

2

Use the backside, if needed

Problem 1 of 4

OOP.MT.170317.M064

```
for (int k = 0; k <= 11; ++k) {
    float result = 0.0;
    float first = pow(-1, k);
    float pow1 = pow(1, 2*k+1);
    float second = pow((2*k+1), pow1);

    result = kohan(result, first/second, compensation);
}
```

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.

```
double xRes = 0
for (int i = 0; i < data.length; ++i) {   -> Function x̄ (double[] array)
    xRes += i;
}
return xRes / data.length;
double yRes = 0
for (int i = 0; i < data.length; ++i) {   -> Function ȳ (double[] array)
    yRes += Math.log(data[i])
}                                          if negative?
return yRes / data.length;
double xandy = 0
for (int i = 0; i < data.length; ++i) {   -> Function xy (double[] array)
    xandy += i * data[i] ;
return xandy / data.length
double powX = 0
for (. . . . . ) ;                         -> same -> Function x² (double[] array
    powX += i*i ;
return powX / data.length.
```

double m = xy(data) − x̄(data) * ȳ(data) / x²(data) − x̄(data) * x̄(data)

a = ȳ(data) − m * x̄(data)

```
int[] array = new int[2];
array[0] = a
array[1] = m
return array;
```

Use the backside, if needed

Problem 2 of 4

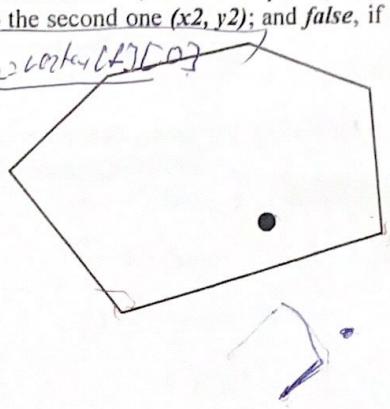
3

OOP.MT.180317.MC6

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.



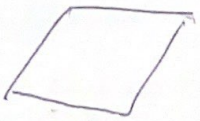
```
double x1 = vertex[0][0], x2 = vertex[1][0]
for (int col = 1; col < vertex.length; ++col)
  boolean isLeft = toLeft(vertex[0][col-1], vertex[1][col-1],
                          vertex[0][col], vertex[1][col],
                          x, y)
  if (!isLeft) return isLeft = false;
if (toLeft(vertex[0][vertex.length-1], vertex[1][length-1], x1, y1, x, y)
    == false)
  return false;
return true;
```

```
for (int col = 1; col < vertex.length; ++col)
  boolean left = isLeft(vertex[0][col-1], vertex[1][col-1],
                        vertex[0][col], vertex[1][col], x, y)
  return left == false;
return toLeft(vertex[0][vertex.length-1], vertex[1][length-1], x1, y1, x, y) == false
return true;
```

Use the backside, if needed

OOP. MT. 10318M064

[0,0] [0 1 5 7
[1,0] [1 2 6 8



2

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

~~int n = vertex[0].length;~~
~~for (int i=0; i < 2; ++i)~~
int col = 0;
int temp = 0
for (int i=

vertex[0][col], vertex[1][col], [0][col], [1,col],
x

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	

```
int[][] matrix = new int[m][m];
matrix[0][(m-1)/2] = 1;
for(int i=0, i < matrix.length; ++i) {
  for(int j = (m-1)/2; j < matrix.length; ++j) {
    if((i-1)<0 ) {
        matrix[m-1][j+1] = matrix[i][j] +1;
    }
else if((j+1) ≥ m) {
    matrix[i][0] = matrix[i][j] +1;
  }
else if((i-1)<0 && (j+1)≥m) {
    matrix[i+1][j] = matrix[i][j]+1;
  }
else {
    matrix[i+1][j+1] = matrix[i][j] + 1;
  }
```

2

Use the backside, if needed

Problem 4 of 4

OOP. MT. 170317. M064