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:

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

#include math.h;

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

```
float pi(int n){
    float result = 0.0; float compensation = 0.0;
    for(int c=0; c<=n, c++){
        float arg1 = (math.pow(-1,c)) / (2c+1) * math.pow(5,2c+1);
        float arg2 = (math.pow(-1,c)) / (2c+1) * math.pow(239,2c+1);
        float step = 16 * arg1 - 4 * arg2;
        result = kahan(result, step, compensation)
    }
    return result;
}
```

Use the backside, if needed

Problem 1 of 4

OOP.MT.17031.H011

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.

$\bar{X}$ ✓
$\bar{y}$ = math. log (dy)
$\bar{xy}$ = ✓

```
public static double[] expReg(double[] data) {
~~double expReg~~ double[] [X] answer = new double[2] ~~[double.length]~~;
~~for (int x=0, x< double.length, x++){~~
~~double expReg~~
for (int x=0, x< double.length, x++){
    ~~doub~~ double dy = Math.log(data[x]);   if negative?
      double ny = math.log(ny);
~~for(...) double m =((x*ny)/2) - (x*ny)) / (x~~
    ~~double xM = [~~
    double m = (((x*ny)/2) [-] (x*ny))/ x*x - dy*dy
    double a = dy - m[~~dx~~] x;
  answer[0][X] = m;
} answer[1][X] = a;

return answer;
}
```

3

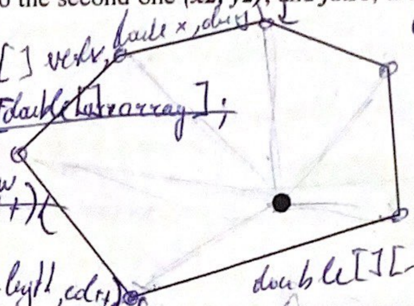
Use the backside, if needed

OOP.MT.170317.H011

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.



X	0	1	2	3	4
y	7	8	9	10	11

```
public static boolean isInside(double[][] vertex, double x, double y){
bool result; ~~bool[] checkArray = new bool[double[]array];~~
bool[] check = new bool[double[].length];
~~for (int row=0, x < double[row].length; ++){~~
    for (col int col = 0; col < double[].length, col++){
if(col != double[].length){
    state = toLeft(vertex[0][col], vertex[1][col],
            vertex[0][col+1], [1][col+1],
            x, y)
} else { state = toLeft( vertex[0][col], vertex[1][col],
            vertex[0][0], vertex[1][1], x, y)

    check[col] = state;

}
/* Assume there is a function checking if all the values in
the array are true or false. If one of the value not
the same ~~that~~ as other, it returns false */

result = isSame(check[])

}
return result;
}
```

4

Use the backside, if needed

OOP.MT.170317.H011

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){
  row = 0
  ~~int k, l, m;~~ int k, l, m   int firstC = math.round(square.length/2)
  ~~int midFirst = square[0][...]~~
  for(int c = 1, c ≤ square[].length * square.length; c++){
    ~~row = 0~~
    ~~firstC = c;~~
    ~~/* square[row][firstC] = c */~~
    if (firstC > square[].length){
       firstC = 0
    }
  else if (row < 0){
       row = square.length -1;
    }
  else if (square[row][firstC] ≥ c){
       row = row + 2;
       firstC = firstC - 1;
  } else if (row ≤ 0 && || col > square[].length){
       row = row + 2;
       firstC = firstC - 1;
    }
  } square[row][firstC] = c;
    row = row - 1;
    firstC = firstC + 1;
  }
}
```

clear (int[][] arr) - clears the array by

3

Use the backside, if needed

OOP.MT.170315.Ilcou