Section, Name and ID#:________

AMERICAN UNIVERSITY OF ARMENIA

College of Science and Engineering

CS 120 Introduction to Object-Oriented Programming

MIDTERM EXAM

10

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:

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

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

```
float pi(int n) {
        float sum1 = 0;
        float sum2 = 0;
        float result;


    for(int k=0; k<n; k++) {

?       sum1=sum1+(pow(-1,k)/(2k+1)·pow(5,2k+1));
        sum2 = sum2 +(pow(-1,k)/(2k+1)· pow(239,2k+1));
    }

      sum1 = 16·sum1;
      sum2 = -4 ·sum2;
     result = kahan(sum1,sum2,0.0);


return result;
```

Kahan?

3

Use the backside, if needed

Problem 1 of 4

OOP.MT.170317.H057

Section, Name and ID#:[redacted]

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) {
    double[] result = new double[2];
    ~~float~~ double meanx = 0;
    double ~~q~~ meany = 0;
    double meanx2 = 0;
    double prod = 0;
   for(int i=0; i< data length; i++) {
        meanx= meanx+i;                          If negative?
        meany = ~~t~~ meany + Math.log(data[i]);
       meanx2 = meanx2 +(i·i);
       prod = prod + meanx·meany
   }
   meanx= meanx/data.length;
   meany = meany/data.length;
   meanx2 = meanx2/data.length;
   prod = prod/data.length;
                                      multiply          meanx
   double m =(prod - meanx*meany)/(meanx2 -(meanx*mean)
   double a = meany - m*meanx;                          ;
   result[0]=m;
   result[1]=a;

                                                   3

 return result; }
```

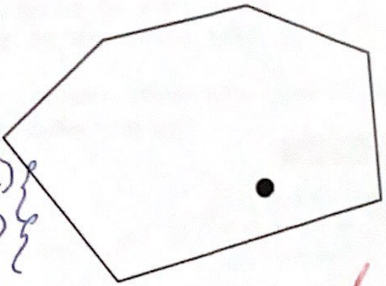
Use the backside, if needed

OOP.MT.170317.H057

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.



```
        import java.util.;
my problem. reverse array
    *→swap goes here **print goes here
  public static void reverse(int[] arr){
    for(int i=0; i<arr.length/2;i++){
      int j=arr.length-1-i;
        swap(arr,i,j);


    }

}
```

1

```
  swap
* public static void swap(int[]arr, int i, int j) {
        arr[i]=arr[i]^arr[j];
        arr[j]=arr[i]^arr[j];
        arr[i]=arr[i]^arr[j];
  }
public static void main(String[] args) {
      Scanner input =new Scanne(System.in);
        int size = input.nextInt();
        int[] arr =new int[size];
        reverse(arr);
        print(arr);

}
 ** public static void print(int[]arr){
          for(int i=0; i<arr.length;i++){
             System.out.print(arr[i]);
          }
   }
```

ection, 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 magiOdd(int [][] square) {
    ~~m square[0][square[row].length+1]=1;~~
    ~~for(int row=0;~~
      square[0][square[0].length/2 +1] = 1;
Երևի if-եր պետք է գրել nested for loop-երում
      ~~if(ra~~
    for(int row = 0; row < square.length; row++){
      for( int col = ~~0~~ squar[~~row~~ 0] length/2+1; col<
                   < square[row].length; col++)
                                              {

        if(row - 1 < 0) {
          row = square.length - 1;

        }
        else if( col
```

1

}

Use the backside, if needed

Problem 4 of 4

my problem on the next page!

```
public static void forward( int[][] arr) {
    for (~~ra~~ int row = 0; row < arr. length; row++) {
        for( int col = 0; col < arr[row]. length; col++) {
                                  (multiply)
            arr[row][col] = col + row * arr[row].length + 1;
                                                          1
        }
    }
}

~~public static void print2D (int[][] arr) {~~
    ~~for (int row = 0; row < arr. length ; row++) {~~
        ~~for (int col = 0; col <arr[row]. length; col++) {~~
            ~~arr[ System. out. print~~
public static void main (String[] args) {
    Scanner input = new Scanner(System.in);
    int ~~size~~ rows = input. nextInt();
    int cols = input. nextInt();
    int [] []. array1= new int [rows][cols];
    DecimalFormat tab = new DecimalFormat(" 00");
    forward(array1);
    for(int row = 0; row <arr. length; row++) {
        for (int col=0; col<arr[row]. length; col++) {
            System. out. print( tab. format (array1[row][col]));
        }
        System. out. println();
                                                    1
    }
}
```