Section, Name and ID#: OOP.MT.170317.H047

AMERICAN UNIVERSITY OF ARMENIA OOP.MT.170317.H047

College of Science and Engineering

CS 120 Introduction to Object-Oriented Programming

MIDTERM EXAM

11

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***.

```
float pi(int n)
{ faloat result = 0.0, float odd, even;
  if (n%2!=0) {
    for(int i= 1; i<=n; i+=2)
      { odd *= i;
      }
  }
  else { for(int i=2; i<=n; i+=2)
           { even *= i;
           }
  }
  result = odd/even * (2*n+1);
} return 2*result;
```

loop needed

2

Use the backside, if needed

Section, Name and ID#: OOP.MT.17031?.H047

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

$$k = \frac{\overline{xy} - \bar{x}\,\bar{y}}{\overline{x^2} - \bar{x}^2}, b = \bar{y} - k\,\bar{x}$$

[k, b]

Here $\bar{x}$ (xm) is the mean of the ***x*** coordinates, $\bar{y}$ (ym) is the mean of the ***y*** coordinates, $\overline{x^2}$ (x2m) is the mean of the squares of the ***x*** coordinates, and $\overline{xy}$ (xym) 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)***.

```
public static double[] lin(double[] data) {
    double[] result;
double[] xm = new double[data.length]; for( int i=0; i< data.length; i++)
    { xm[i] = i;
    int x = mean(double[] xm)
  double [] ym = new double [data.length];
    for(int i=0; i< data.length; i++)
      { ym[i]= data[i];
      }
    int y= mean((double[] ym);
  double[] x2m = new double [data.length];
    for( int i=0, i< data.length; i++)
      { x2m[i]= i^2;
      }
    int x2 = mean(double[] x2m);
  double [] xym = new double [data.length],
    for(int i=0; i< data.length, i++)
      { xym[i]= i·data[i];
      }
    int xy = mean(double [] xym),
    result[0]=(xy - (x·y))/(x2 - x·x);
    result[1]= y - result[0]· x;
  } return result;
```

3

```
public static int scalar( int[] mat1, int[]
                                          mat2
                                          ){
    int result =0;
      for (int i=0; i< mat1.length; i++)
          { result +=mat1[i] * mat2[i];
          }
      return result;
   }


public static void print1D (int[] arr) {
      for(int i=0; i< arr.length; i++){

   System.out.print ( a[i] + " ");
   }

}
```

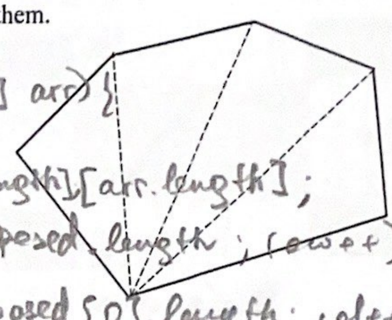
1

Section, Name and ID#: OOP-MI-170315H047

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 void transpose (int[][] arr){
  int [][] transposed = new int[arr[0].length][arr.length];
    for( int row = 0; row < transposed.length ; row++) {
        for(int col = 0; col < transposed[0].length; col++) {
            transposed[ row][col] = arr[col] [row];
        } }
(already created) print2d (transposed);
}
```

change in arr -?

1

```
public static void shift L( int[] arr){
    int keeper = arr[0];
    for(int i = 0; i< arr.length /2; i++)
    {
        arr[i] = arr[i+1];
    }
    arr [a.length -1] = keeper;
}   print 1d( arr);
```

not needed

1

1

```
public static int[][] Multiply (int[][]mat1 , int[][]mat2 )
{   int [][] result = new int[ mat1.length ][mat1.length];
  transpose (mat 2);
  for( int row = 0 ; row < result.length ; row ++) {
        for ( int col = 0; col < result.length; col++) {
            result [row][col] = scalar (mat1[row],
                                        mat2[col]);
  transpose (mat 2);
  return result;
}
```

previous page.

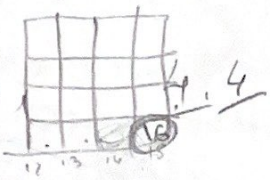
Use the backside, if needed

row · arr[row].length + col + 1

```
for( int row = 0; row < arr.length; row++)
{ for(int col = 0; col < arr[0].length ; col++)
  {
    arr[row][col] = row · arr[row].length + col + 1
  }
}
```

1 2 3 4

row 0
col 0
row 1
col 0
0 · 4 + 0 + 1 = 1
1 · 4 + 1 = 5



4 · 4 + 4 − 4

4 · 4 − (4 · 4 + 4 + 1) =
= 16 − ( )
4 · 4 + 3 + 4 = 15
16 + 2 − 4 = 14

arr.length · arr[row].length − (row · arr[row].length + col + 1)

```
for(int row = square.length-1, row > 0; row--) {
    for( int col = square[0].length ; col > 0; col--) {
        square[row][col] = arr.length · arr[row].length −
```

$\boxed{\text{arr.length} \cdot \text{row} + \text{col} - \text{arr.leng} + \text{col} + \text{square.leng}}$

row 3

4 · 3 + 4 − 4 = 12.

Section, Name and ID#: OOP.MT.1703(J.H047

Problem 4: Write a Java method ***public static void magic4N(int[][] square)*** that creates a magic square of a *4N-by-4N* size using the following algorithm:

1. Creates an array of the same size as ***int[][] square*** and fills it forward with successive integers assigning *1* to the top-left element;
2. Creates anther array of the same size as ***int[][] square*** and fills it backward with successive integers assigning *1* to the bottom-right element;
3. Divides the original ***int[][] square*** into 16 blocks of the same size – 4 blocks per row and column. In the on-diagonal (shaded) blocks copies the elements from the first array, and in the off-diagonal blocks copies the elements from second array.

1	2					7	8
9	10					15	16
		19	20	21	22		
		27	28	29	30		
		35	36	37	38		
		43	44	45	46		
49	50					55	56
57	58					63	64

		62	61	60	59		
		54	53	52	51		
48	47					42	41
40	39					34	33
32	31					26	25
24	23					18	17
		14	13	12	11		
		6	5	4	3		

in main
Scanner input
= new Scanner
(System.in);

2

```
① public static void magic4N(int[][] square) {
        int[][] arr = new int[square.length][square.length];
    for( int row = 0; row < square.length ; row ++) {
        for( int col = 0; col < square[0].length ; col ++) {
            square[row][col] = row * square[row].length + col + 1;
        }
    System.out.println();
    }
② //    int[][] arr2 = new int [square.length][square.length],
  //  for( int row = arr2.length , row > 0; row - - ) {
  //        for( int col = arr[2].length, col > 0; col - -) {
  //        arr2[row][col] = arr2.length • row + col - arr.length;
  ///       }
  //      System.out.println();
  //  }

for( int row = 0; row > arr2.length , row ++) {
        for(int col = 0; col > arr2.length ; col ++) {
            arr2.length * arr2.length - arr[row][col];
        }   forward [row][col];
arr2.length * length
    system.out.println();
}
```

Use the backside, if needed