Section, Name and ID#: [redacted]

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

MIDTERM EXAM

9

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***:

```
                    1              2           3
float kahan(float num1, float num2, float &compensation)
{
        float result;
        num2 -= compensation;   2
        result = num1 + num2;   3
        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){
  int result =0, pi = 0
  int compensation = 0.0,
  for(int i=0; i<n; i++){
    if(i%2 = 1){
      for(j=0; j<n/2 ;j+=2){
        pi = pi + 2.




return result
}
```

1

Use the backside, if needed

OOP.MT.170317.K058

Section, Name and ID#:[redacted]

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

$$y = k\,x + b,$$

index - x
value - y

where the slope k and the intercept b are computed as

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

Here $\bar{x}$ is the mean of the x coordinates, $\bar{y}$ is the mean of the 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 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) {
  int ymean = mean(          data),
  int[]     xarr = new int[data length].
  for(i=0;i < data length,i++){
     xarr[i] = i
  };
  int  x mean = mean(xarr)
                    int
  int[]xyarr = new[data.length];
  for(j=0;i<data.length;i++){
    xyarr[i] = i* data.[i];
  }
  int xymean = mean(xyarr),
  int [] x2arr = new int [data.length];
  for(i=0;i< data.length;i++){
     x2arr[i] = i*i;
  int x2 mean = mean(x2arr);
  int[] result = new[2];
                0
     result[1] =(xymean - xmean* ymean)/(x2mean - xmean*
                                                   xmean)
           1
  result[2] = ymean-result[1]*xmean
```

Use the backside, if needed

```
      return result;
  }
```

3

OOP.MT.170317.H058

```
public static int[][] transpose(int[][] arr){
  int[][] trans = new int[arr.length][arr length];
  for(i=0; i<arr.length; i++){
    for(j=0; j<arr[i].length; j++){
     trans[j][i] = ~~{~~arr[j][i];
    }
  }
   return trans;
}
```

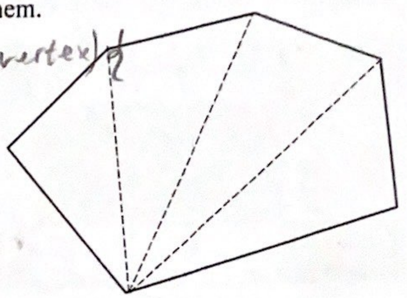
1

Section, Name and ID#:

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 double area(double[][] vertex){
```

0

Use the backside, if needed

```
public static int[][] multiply(int[][]mat1,int[][
                                                   mat2
  int[][] result = new int[mat1 length][mat1 length],  2
  transpose (mat 2);
  for(int i =0; i < result length, i++){
     for (int j =0 ; j< result.length; i++){
       result [i][j] = product (mat1[i]; mat2[j]),
       }
     }
   return result;
  }


public static int product(int[] arr1; int[]arr2){
   int result =0;
   for (i=0; i<arr1.length, i++){
    result +=arr1[i]* arr2[i];
   }
   return result
                                                       1
 }
```

Section, Name and ID#:

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		

```
public static void magic4N(int[][] square) {
1)  int[][] magict= new int[square.length][square length],
    for(i=0, i < square.length, i ++){
       for(j = 0, j < squarc [i].length, j ++){
         magic [i][j] = i * magict.length + j + 1
       } }
2)  int[][] magicb = new int [square length][square length],
    for(i=0;i< magicb length, i++){
       for(j=0,j< magicb[i] length, j++){
    magicb[j][i] = magicb.length * magic b. length - i *magict.length-
                                                                   -j
      }
    }
3) for(i=0, i < square length, i++){
    for(~~int~~j = 0, j < square. length, j ++){
     if( i=j|| i+j = square length-1 //(some cases I don't know))
      square[i][j] = magict[i][j];

   else
     square[i][j] = magicb[i][j],
```

3

Use the backside, if needed

Problem 4 of 4

OOP.MT.170317.H058