

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:

$$\pi = 2 \sum_{k=0}^n \frac{(2k-1)!!}{(2k)!!(2k+1)} = \frac{2}{1 \cdot 1} + \frac{1}{2} \cdot \frac{2}{3} + \frac{1 \cdot 3}{2 \cdot 4} \cdot \frac{2}{5} + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} \cdot \frac{2}{7} + \dots$$

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(1);
    int result2;
```

```
    for (int i=0; i <= n; i++)
    {
        if (i % 2 == 0)
            result2 = 2 * n;
```

if $n \% 2 = 0 \Rightarrow$ $\frac{1}{2} \cdot \frac{2}{3} + \dots$

if $n \% 2 = 1 \Rightarrow$ $\frac{1}{2} \cdot \frac{2}{3} + \dots$

if $n \% 2 = 1 \Rightarrow$ $\frac{1}{2} \cdot \frac{2}{3} + \dots$

Use the backside, if needed

Problem 1 of 4

000.MT.120317.L027

Section, Name and ID#:

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

$$y = kx + b,$$

where the slope k and the intercept b are computed as

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

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)
```

```
double[] a = {x, y}
```

```
double[] sum = {0, 0}
```

```
double[] x = new double[data.length]
```

```
for (int i = 0; i < data.length; i++)
```

```
    x[i] = data[i]
```

```
public static array(int[][] square)
{
    int n = row length
    int sum = 0
    int m = col length
```

```
    for (int i = 0; i < n; i++)
```

```
        for (int j = 0; j < m; j++)
```

```
            sum += square[i][j]
```

```
    return sum;
```

```
}
```

→

$$y = kx + b$$

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

$$b = \bar{y} - k\bar{x}$$

Sum = the sum of the elements of the rows of the array

also we can calculate the sum of the elements of columns by adding

$$sum += square[i][0]$$

1

Use the backside, if needed

Problem 2 of 4

DDP.MT.1303/7 L047

Section, Name and ID#:

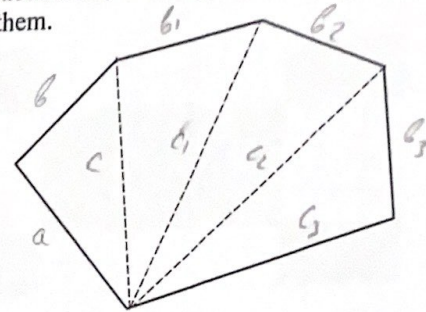
Problem 3: Write a Java function `public static double area(double[][] vertex)` that takes as its argument a 2-by-n array of a convex polygon's vertex coordinates `double[][] vertex` - the x coordinates in the first row and y coordinates in the second row. It returns polygon's area as follows:

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)
 double area = 0;
 // polygon is not self-intersecting
 // vertices are a, b, c, ...
 // hence area

(0,0)
 x1 x2 x3 x4
 (1,0)
 y1 y2 y3 y4



$$\sqrt{p(p-a)(p-b)(p-c)}$$

$$p = \frac{(a+b+c)}{2}$$

double dist (double x1, double y1, double x2, double y2)
 int a=0, b=0, c=0, p=0;

for (int i=0, i<vertex[0].length, i++)
 for (int j=0, j<vertex[i].length, j++)

not needed

array of x1, x2, x3, x4, ... xn
 same row of y1, y2, y3, y4, ... yn

double dist (double x1, double y1, double x2, double y2) that y1, y2, y3, y4, ... yn

original array of x and y coordinates
 x1, x2, x3, x4, ... xn
 y1, y2, y3, y4, ... yn
 hence area
 a, b, c, ...
 p = $\frac{(a+b+c)}{2}$

2

Use the backside, if needed

Problem 3 of 4

OOP.MT. 170317. LOJ

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 another 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	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	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)
{
    int[][] square = new int[4N][4N];
    for (int row = 0; row < square.length; row++)
        for (int col = 0; col < square[row].length; col++)
            square[row][col] = square[4N][4N];
}
```

As I couldn't solve this problem I solved the wrong problem.

```
public static void array (int[][] square)
{
    int sum = 0;
    int[][] square = new int[n][n]
    for (int i = 0; i < square.length; i++)
        for (int j = 0; j < square[i].length; j++)
            sum += square[i][j];
}
```

not needed
the places of the elements of the diagonals are square[i][i]

In this problem I calculate the sum of the elements of diagonals of the array