Section, Name and ID#: _____

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

12

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***.

Another problem.
create a function that takes as an argument a matrix of any size and returns the (row, col) of the left upper corner of the region 5×5 that has the greatest number of 1s.

```
public static int getNum(int[][] mat, int sc, int sr){ // sc-starting column
                                                        // sr-start. row
    int count=0;
    for(int row=sr; row<sr+4; row++){
        for(int col=sc; col<sc+4; col++){
            if( mat[row][col]=1){
                count++;
    } } }
    return count;
    if(sc<0 || sr<0 || sc+4>mat.length || sr+4>mat.length){
        return 0
    }
```

Use the backside, if needed

OOP.MT.170317.HOG1

```
int maximum=0;
int mr=0;          // mr, mc - the coord. of the upper left region
int mc=0;             that has the greatest num of ones

for(int r=0; r< mat.length; r++){
   for(int c=0; c< mat.length; c++){
int count = getNum(mat, r, c);
      if(count >= maximum){
         maximum = count;
         mr = r;
         mc = c;
      }
   }
}

System.out.print(mr+" "+mc);
```

2

ection, 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

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

where the slope ***k*** and the intercept ***b*** are computed as

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

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[] result = new double[2];
    double[] xs = new double[data.length];
    for(i=0; i<data.length; i++){
        xs[i] = i; }
    double meanx, meany, meanpr, meanxsq, k, b;
    meanx = mean(xs);
    meany = mean(data);
for(int i=0; i<data.length; i++){
        productxy[i] = i*data[i];
        xsq[i] = i*i;

}
    meanpr = mean(productxy);
    meanxsq = mean(xsq);

k = (meanpr - meanx*meany)/(meanxsq - meanx*meanx);
b = meany - k*meanx;
result[0] = k;
result[1] = b;
return result;

}
```

[2, 4, 7, 6]

double[] xsq = new double[data.length];
double[] productxy = new double[data.length];

Use the backside, if needed

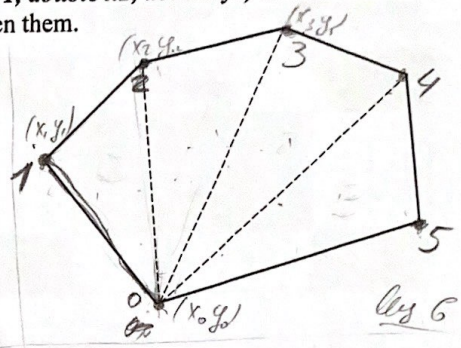
OOP.MT.17031.1061

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*** and ***(n+1)*** 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;
double a=0;
double b=0;
double c=0;

for(int i=0; i<vertex.length-2; i++){
    a=dist(vertex[0][0], vertex[1][0], vertex[0][i+1], vertex[1][i+1]);
    b=dist(vertex[0][0], vertex[1][0], vertex[0][i+2], vertex[1][i+2]);
    c=dist(vertex[0][i+1], vertex[1][i+1], vertex[0][i+2], vertex[1][i+2]);
    double p=(a+b+c)/2;
    area+=sqrt(p*(p-a)*(p-b)*(p-c)); //use library for computing
                                     the square root or creat
                                     a function called sqrt
}
return area;
}
```

Use the backside, if needed

OOP.MT.170317.4061

ection, 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){
    int[][] arrf= new int[square.length][square.length]; int value = 1;
while(value <= value1)  for(int row =0; row < arrf.length; row++){
                          for(int col=0; col < arrf[0].length; col++){
                              arrf[row][col] = value;
                              value ++;}
                          }
    int[][] arrb = new int [square.length][square.length];
                  int value1 = square.length * square.length;
while(value1>0){  for(int row=0; row < arrb.length; row++){
                      for(int col=0; col<arrb[0].length; col++){
                          arrb[row][col] = value1;
                          value1--;
                      }
                  }
}
```

I did not notice 4by4, so I wrote for general case. :)

2

Use the backside, if needed

OOP.MT.Fall17.Model