Section, Name and ID#:

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

16

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***.

```
~~public static void main(String[] ...){~~
float pi(int n) {
float pi = 0.0, compensation = 0.0;
for(int i=0; i<=n; i++) {
  float a = fac(2*i-1) / (~~float~~ fac(2*i) * (2*i+1));
  pi = ~~float~~ kahan(pi, a, compensation);
}
pi = 2 * pi;
return pi;
}

public static float fac(float n) {
float result = 1;
if (n % 2 == 1) {
  for(i=1; i<=n; i+=2)
    result *= i;
}
```

Use the backside, if needed

(→ See next page)

OOP.MT.17031.K060

```
else if( n %2=0) {
   for ( i=2 ; i <=n ; i+=2 )
      result+ = i ;}
else
return result;
}
```

4

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

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

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[] xy = new double[data.length];
double[] x = new double[data.length];
double[] x2 = new double[data.length];
for(int i=0; i<data.length; i++){
  xy[i] = i*data[i];
  x[i] = i;
  x2[i] = i*i;
}
double meanx = mean(x);
double meany = mean(data);
result[0] = (mean(xy) - meanx*meany)/(mean(x2) - meanx*meanx);
result[1] = meany - result[0]*meanx;
return result;
}
```

Use the backside, if needed

Problem 2 of 4

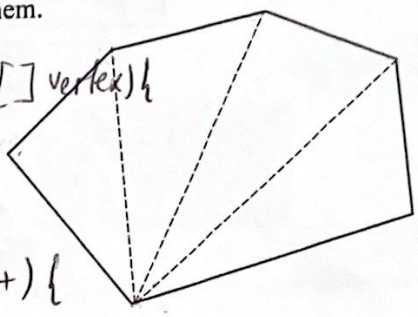
4

OOP.MT.17031.1060

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.

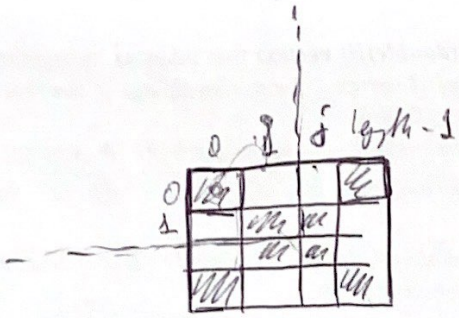


```
public static double area(double[][] vertex){
double result=0;
for(int i=1;i<vertex[1].length-1;i++){
double a = dist(vertex[0][0],vertex[1][0],vertex[0][i],vertex[1][i]);
double b = dist(vertex[0][0],vertex[1][0],vertex[0][i+1],vertex[1][i+1]);
double c = dist(vertex[0][i],vertex[1][i],vertex[0][i+1],vertex[1][i+1]);
double p = (a+b+c)/2;
result+=Math.sqrt(p*(p-a)*(p-b)*(p-c));
}
return result;
}
```

34

Use the backside, if needed

OOP.MT.170317.HOGO


0
1
log n - 1
0
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){
int[][] M1 = new int[square.length][square[0].length];
int[][] M2 = new int[square.length][square[0].length];
int k=1;
for(int i=0; i<square.length; i++)
   for(int j=0; j<square[0].length; j++){
    M1[i][j]=k;
    M2[M2.length-1-i][M2.length-1-j]=k;
    k++;
    }
}
for(int i=0; i<square.length; i=i+(square.length)/4)
   for(int j=0; j<square.length; j=j+(square.length)/4)
      for(int m=i; m<=square.length/4; m++)
        for(int n=j; n<=square.length/4; n++){
          if((j/(square.length/4))%2==0){
          square[m][n]=M1[m][n];}
          else{
          square[m][n]=M2[
```

Use the backside, if needed

Problem 4 of 4

See next page

OOP.M1.1703.4060

```
if ( i < length/4 || i >= length - square.length/4) {
  if ( j < square.length/4 || j >= square.length - square.length/4)
    square [m][n] = m1[m][n];
  else
    square [m][n] = m2[m][n];}

else {
  if ( j < square.length/4 || j >= square.length - square.length/4)
    square [m][n] = m2[m][n];
  else
    square [m][n] = m1[m][n];
  }
}
```

4