Section, Name and ID#:_

AMERICAN UNIVERSITY OF ARMENIA

College of Science and Engineering

CS 120 Introduction to Object-Oriented Programming

MIDTERM EXAM

7

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***.

2

```
int result = 0;
addTop(n) {
if (n%2) for (i=1, i<=n, i+2)
result += float (i, 2i+1) 1+3
else   i= result 1
i=1, i<=n, i++
take = i
= float (0+2i+1) 1+3
result = i

int result = 0;
addTop(n, result) if (n < 0) return (result - 1);
else if (n % 2)
for (i = 0, i < n/2 i++)            something.
result += kahan (0, 2i+1)
return result; else. for (i=1, i<=n/2, i++
                  result += kahan (0, 2n)
                  return result;
add_Bottom (n, &result)
if (n<0)
return (result-1)
return result;
if (n > 0 &&
addOdd(

int main () float pi(int n))
add Top(        result+ int final result;
final result = addTop(2n-1) / (addTop(2n) * addTop(2n+1))
add

float pi (int n) {
for (int k=0, k<n, k++)
final result += float kahan (k)
resultTop += addTop(2k-1);
result Bottom += addTop(2k);
result Bottom right += kahan (0, 2k+1)
}
return?
```

need to add 3 results and return them.

Use the backside, if needed

OOP.MT.170317.M068

float pi (int n) {   1 2   3 4 5
if (n %2) (n is odd) -> true   int result)   1   3
float kahan (compute for (1 + n   5
for (i = 1; i < n/2+1 i += 2)
result += float (1, i+2)   1+3+5

if (n %2 for (i = 0, i < n/2 , i++
else   result += float (2, i)   + 2

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

[0 2]

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

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)***.

1

~~public static [] lin(double[] data) {~~

```
    public static double[] lin(double[] data) {
    int [ ] Lreturn = new it [2];
    ret

    return [0] =        return[1] =
    return Lreturn;
}
```

~~[up + figure]~~

mean[double lin]

Use the backside, if needed

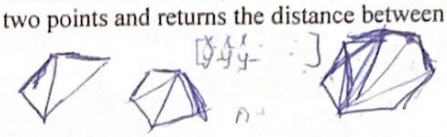
OOP.MT.110317.M068

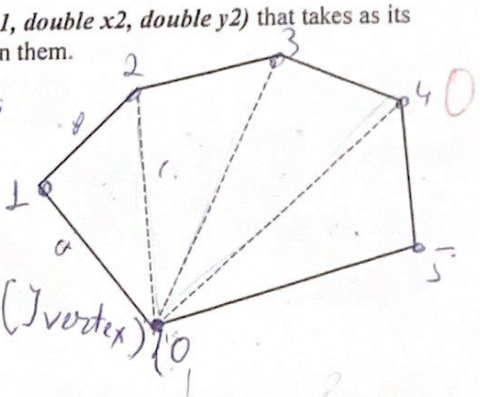
Section, Name and ID#: [redacted]

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){

num of triangle = (vertex - 2;

we call the double dist function by calling it each pair 2 of successive vertices, which means by successive indies of

wrong problem and hopefully right solution in the back

Use the backside, if needed

OOP. MT. 19-03[illegible]068

ices

of the same size and return a boolean array of boolean array where the index there is false when elements don't match and true otherwise.

```
public static boolean[][] matrixInter(int[][] matrix1, int[][] matrix2){
    for (int row = 0; row = matrix1.length; row++){
        for (int col = 0; col = matrix1.[row].length; col++)
boolean[][] boolMatrix = new [int [matrix.length][matrix.length];
boolMatrix[row][col] = (if (matrix1[row][col] == matrix2[row][col]);
    }
}
return boolMatrix;
}
```

since they have the same # of el.

1

Section, Name and ID#: [redacted]

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 int[][] createArray ( int size ) {
int [][] array = new int[size][size];
for (int row = 0; row < array.length; row++) {
    for (int col = 0; col < array.length; col++) {
if (col < array.length)
    array[row][col] = fill(array, row+1)
        ( then transpose.
}
```

at first

wrong problem hopefully right solution in the back of the paper.

Use the backside, if needed

OOP. MT. 1803HM068

any row is t row,
in other words a symmetric matrix, so that when created with specified size, it fills the matrix with odd numbers

```
public static Create int [][]matrix createMatrix(int size) {
        int[][]matrix = new[size][size]
    for(int row=0, row<matrix.length, row++){.
     for(int col=0, col<matrix[row].length, col++ {.
       matrix[row][col] = 2(row+col)+1}
    }}
  return matrix.
```

almost good for problem 4

```
Public static print_matrix ( int[][] matrix) {
  for (int row=0, row<matrix.lengt, row++ ){
      for(int col=0, col< matrix[row].length,col ++
    System.out.print ( matrix [row][col]
  }}
  System.out.print( )
}
```

2

```
2.  public static void main( ){.
        int size = 5 ;
int[]matrix= creatematrix (size).
    printMatrix(matrix);
```

2) Check whether a matrix is symmetric or not.

```
public static boolean checkMatrix ( int [][] matrix){
   int counter=0        int Tmatrix = transpose(matrix)
for ( int row=0, row< matrix.length, row ++){.
  for(int col=0, col < matrix[row].lengt, col ++ {.
     if(matrix[row][col] ==Tmatrix [row][col],
         counter ++
 }}
if (couter ==matrix.lenght * matrix[0].lengt)
  return true;
else
  return false;
}
```

suppose we have a transpose function that returns the transpose of a given matrix

→ for the second problem. :(

1

> adding this to function below before main and adding checkMatrix(matrix); we will have the result.