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 9
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;
}
```

← did not use sorry :)

Using this function, write a C++ function ***float e(int n)*** that computes the value ***e*** by the following formula:

$$e = \sum_{k=0}^{n} \frac{1}{k!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{1*2} + \frac{1}{1*2*3} + \cdots$$

2

Recall that the factorial of non-positive numbers equals to *1* by definition.

The initial value of ***float compensation*** is ***0.0***.

```
int factorial (?) {
    int num, fact = 2;
    for ( int i = 1 ; i < num ; i++ ) {
        fact * = i ; }

float e(int n) {
    float result = 0.0 ;
    for ( int i = 0 ; i <= n ; i++ ) {
        result += 1. / factorial (i) ;
    }
```

/* first we create a function, which will compute the factorial of the given number then we create a float function, which by using for loop is going to sum up the numbers divide by each other & divided the factorials of the numbers */

OOP.MT.170317.M016

Section, Name and ID#:

Problem 2: Write a Java method ***public static double[] mean(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 mean value of the data points and the second element being the standard deviation. The standard deviation σ of n numbers a_i is computed as:

$$\sigma = \sqrt{\frac{\sum_{i=0}^{n-1}(a_i - mean)^2}{n}}$$

```
public static double [] mean (double [] data) {
    for (int i=0; i < data.length; i++) {
    double result = 0;
        result += data[i];
    }
    double meanVal = result / data.length;
    for (int i=0; i < data.length; i++ ) {
        double dev = 0;
        ~~dev = Math.sqrt((Math.power(data[i]-meanVal,2))/~~
        ~~dev += Math.sqrt((Math.power(data[i]-meanVal,2))/~~
        ? ~~data.length~~
        dev += Math.sqrt((Math.power(data[i]-meanVal,2))/
            data.length);
    }
```

by using for loop we compute the sum of the numbers in array, then when we divide it by the length of the array we get their mean value

Math.sqrt - will compute square root

math.power will compute the power of a number, where first argument will be the number and the second will be the power e.g. Math.power(4,2) will compute 4^2

to compute deviation of numbers we use for loop to pass every element of ~~the~~ array to the equation of deviation

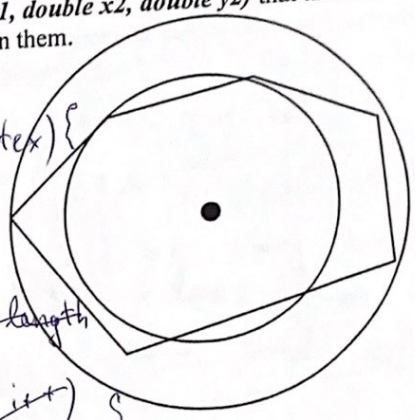
OOP.MT.170313.M016

Section, Name and ID#:

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

1. Computes the center – the mean ***x*** and ***y*** vertex coordinates;
2. Returns the difference between the maximal and minimal distances from the center to the vertices.

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.



x _____

y _____

```
public static double thickness(double[][] vertex){
    int sumX = 0;
    for (int i=0 ; i<vertex[0].length ; i++) {
        sumX += vertex[i];
    }
    int sumX = 0;
    for (int i=0 ; i<vertex[0][].length ; i++) {
        sumX += vertex[0][i] ; // the first for loop will compute the sum of all x coordinates
    }
    int sumY = 0;
    for (int i=0; i<vertex[1][].length; i++) {
        sumY += vertex[1][i] ; // the second for loop will compute the sum of all y coordinates
    }
    double mean = (sumX + sumY) / 2 ;
    // the mean value of all X's and y's
```

Separately ->

Use the backside, if needed

2

OOP. MT. 170317. M016

Section, Name and ID#:

Problem 4: Implement the following Java methods that swap element values between two 2D integer arrays of the same size ***int[][] a*** and ***int[][] b***:

1. ***public static void swap(int[][] a, int[][] b, int row, int col)*** – swaps element values from the specified row ***int row*** and column ***int col***;
2. ***public static void swapCol(int[][] a, int[][] b, int col)*** – swaps all element values from the specified column ***int col***;
3. ***public static void swapRow(int[][] a, int[][] b, int row)*** – swaps all element values from the specified row ***int row***. Get s bonus, if ***swapRow()*** performs faster than ***swapCol()***.

3

1. public static void swap (int[][]a, int[][]b, int row, int col) {

```
a[row][col] = a[row][col] ^ b[row][col];
a[row][col] = b[row][col] ^ a[row][col];
a[row][col] = a[row][col] ^ b[row][col];
}
```

/* by using ^ XOR we can swap elements values from specified row and column */

2. public static void swapCol (int[][]a, int[][]b, int col){

```
for (int i=0 ; i > a.length ; i++){
a[i][col] = a[i][col] ^ b[i][col];
a[i][col] = b[i][col] ^ a[i][col];
b[i][col] = a[i][col] ^ b[i][col];
}
```

/* we use for loop to get all the elements of the row from specified column, then by using ^ XOR operater we swap each element */

3. public static void swapRow (int[][]a, int[][]b, int row){

```
for (int i=0; i > a[].length ; i++) {
a[row][i] = a[row][i] ^ b[row][i];
a[row][i] = b[row][i] ^ b[row][i];
b[row][i] = a[row][i] ^ b[row][i];
}
```

/* in order to get all numbers from specified row we use for loop and by the operator ^ XOR we swap rows. */

Use the backside, if needed

OOP.MI.17032.M016