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 e(int n)*** that computes the value ***e*** by the following formula:

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

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

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

```
float kahan(float num1, float num2, float& compensation) {
    float result;
    num2 = compensation;
    result = num1 + num2
compensation = (result - num1) - num2.
return result;
return num1 + num2 }
    float e(int n) {
for (n=0 n=j; n<0 . n++)
```

if we were asked to compute e. First of all we should compute the value of each numbers and then the sum of consequent numbers.

Use the backside, if needed

```
int t=1, z=1   double sum = 0;
for (k<n  k++)
    z = k*(k+1)
    B= (k+1)(k+2)
sum = sum + Kahan 1/4 , 1/z - 1/6
    return sum;
int main ()
float i = float e(sum)
    cout << sumi + 2
    return 0 }
```

O see AM, TA

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[](double[] data) {
we should made a function that will
compute
for int (i=0 ; i < data.lenght ; i++ )
    t = t + data.at(i) ; }
j = t/data lenght;
for int (z=0 , z<n , z++)
    k = k+(data at(j) - a)
double z = k(2)
           ---
            n
double sum = sqrt(2)
```

0
see AM

Use the backside, if needed

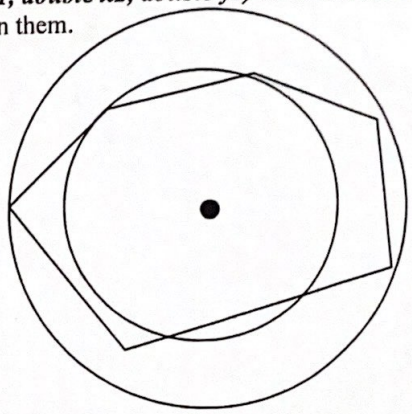
OOP.MT.17031.L036

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.



```
public static double thickness(double
[][] vertex)
for int(i = 0 , i < n , i++) {
mean_x = _x + vertex.a[[i][]
lenghs[]
double X = mean x
           vertex.lenght.
for int(j = 0 J < n J++)
mean_y = =y + vertex.af[i][] leeng hf[]
double y = mean y
           vertex.lenght.
```

1

Use the backside, if needed

Problem 3 of 4

OOP.MT.17031J.1036

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()***.

```
1. public static void swap(int[][] a, int[][] b, int row, int col){
   int t = [row][col]
   a[row][col] t = b[row][col]
   b[row][col] = z }
2. public static void swapCol(int[][] a, int[][] b,
int col) {
   for [i][col]
   a[i][col] = b[i][col]
   b[i][col] = z }

3. public static void swapRow(int[][] a, int[][] b, int row){
   for [row][col]
   a[row][i] = b[row][i]
   b[row][i] = z }
```

Good idea,
wrong implementation
2

Use the backside, if needed

OOP.MT.170317.L036