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 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***.

```
float pi(int n){
    float pi;
    for(int z=0; z<=n; z++){
        pi += kahan(sumEvenOdd(z)/(sumEv(z)*(2n+1)),
                    sumOdd(z+1)/(sumEv(z+1)*(2n+1)), 0)
    }
    return pi;
}
```

```
float sumEven(float f){
    float d;
    for(int i=0; i<=f; i+=2){
        d += kahan(d, i+2, 0.0)
    }return d;
}
float sumOdd(float n){
    float c;
    for(int j=1; j<=n; j+=2){
        c += kahan(j, j+2, 0.0)
    }
    return c;
}
```

product needed

Use the backside, if needed

Problem 1 of 4

OOP.MT.17031.M018

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

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

sqrX1 sqrX2

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[] arr = new double[2];
double X = mean(arrCr(data.length));
double y = mean(data);
double Xy = mean(arrMul(data));
double sqrX1 = mean(arrCr(data.length), 2);
double sqrX2 = X*X;
arr[0] = (Xy - X*y)/(sqrX1 - sqrX2);
arr[1] = y - k*X;
return arr;
}
public static double[] arrCr(int c, int d) {
double[] arr1 = new double[c];
for(int i=0; i<=c; i++){
if(d==1){
arr1[i]=i;
}else if(d==2){
arr1[i]=i*i;
}
return arr1;
}
public static double[] arrMul(double[] s) {
double arr2 = new double[s.length];
for(int j=0; j<=s.length; j++) {
arr2[j] = arr2[j]*j;
}
return arr2;
}
```

Use the backside, if needed

3

OOP.MT.IT0305.Mo18