Section, Name and ID#:_

AMERICAN UNIVERSITY OF ARMENIA

College of Science and Engineering

CS 120 Introduction to Object-Oriented Programming

MIDTERM EXAM

5

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 e (int n){ Float compensation =0,0;~~

```
float e(int n){ ~~int~~ z; ~~int k=0;~~
int, n;   int z = 1/1;  ~~int q;~~ int k=1;
cin>>n;
~~while(K<n){~~
~~cout<<z+1/K;~~
~~K++;~~
~~}~~
for(~~K=0;~~ int q=0; q<n; q++){
for(int K=0; K<n; K++){
cout << z + 1/(K.q) <<endl;
K++;
q++;
}
}
~~cout<<z+1/n;~~
}
}
```

1

Use the backside, if needed

Problem 1 of 4

OOP.MT.170317.L046

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

```
float e (int n){   int q;
                   int n;
                   int k = 0;
                   int z = 1/1;
      while(k<n) {
      ~~cout<< Z + 1~~  ~~for(q=1; q<=n; q++){~~
         ~~cout<< Z + 1/q;~~
~~}k++;}~~
          cout<< Z + 1/k;
          k++;
}   {
```

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){

~~double[]data;~~ ~~first~~

for(int i =0; i<double.length; i++{

double answer =double[i];

double mvt=answer/data.length; return mean;

}

~~data.length~~ / answer ~~/data~~ length;

for(int i =0; i <doubl.length; i++)

finalanswer { ~~double f~~

double answer =doubl[i]-~~mdata~~) * (~~data[i]-mvt~~ mvt);

* (data[i]-mvt)/n; return ~~answer~~ finalanswer;

}

}

1

Use the backside, if needed

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()***.

```
Scanner spec. =new Scanner(System.in);
 int column = next[int];
 int  row =next.int;
1) public static void swap(int[][]a, int[][]b, int row, int col)
                                                                  {
 a[row][column] =a[row][column]^b[row][column];
 b[row][column] = a[row][column]^b[row][column]; }


int  column=nextInt;
   public. . . . . . . . . . . . . . .
2) for (int  i=0; i<a.length; i++){
   a[col][i]=a[col ][i]^b[col][i];
   b[col][i]= a[col][i]^b[col][i];
   a[col][i] = a[col][i]^b[col][i]; }
3){ for(int i=0; i<a.ledth; i++) {
  { a[i][row] =a[i] [row]^b[i][row];
   b[i][row = a[i][row]^b[i][row];
   a[i][row] = a[i][row]^b[i][row]; }  }
```

Use the backside, if needed

3

Problem 4 of 4

OOP.MT.170317.L046