Section, Name and ID#: [redacted]

AMERICAN UNIVERSITY OF ARMENIA

College of Science and Engineering

CS 120 Introduction to Object-Oriented Programming

MIDTERM EXAM

10

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***:

```
                       5              7,6                  0,1
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:

$$\pi = 16\sum_{k=0}^{n}\frac{(-1)^k}{(2k+1)5^{2k+1}} - 4\sum_{k=0}^{n}\frac{(-1)^k}{(2k+1)239^{2k+1}} = \left(\frac{16}{1*5} - \frac{4}{1*239}\right) - \left(\frac{16}{3*5^3} - \frac{4}{3*239^3}\right) + \left(\frac{16}{5*5^5} - \frac{4}{5*239^5}\right) - \cdots$$

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

```
float pi(int n){
    ~~int~~ float sum1=0.0;    int pi1, pi2; babken = ?
            float sum2 =0.0.
    for ( int k=0; k <= n; k++){
  ?      pi1 = (16 * pow(-1, k)) / (2xk+1) x pow(5, 2xk+1);
Kahan    pi2 = (-4 x pow(-1, 4)) / (2xk+1) x pow (239, 2xk+1);
         sum1 += pi1;
         sum2 += pi2;
    }
return ~~floa~~ kahan( sum1, sum2, babken);
```

3

Use the backside, if needed

Section, Name and ID#: [redacted]

Problem 2: Write a Java method ***public static double[] expReg(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 exponent of an exponential regression and the second element being the amplitude. The exponential regression approximates the data points by a formula

$$y = a\, e^{mx},$$

where the exponent m and the amplitude a are computed as

$$m = \frac{\overline{xy} - \overline{x}\,\overline{y}}{\overline{x^2} - \overline{x}^2}, a = \overline{y} - m\overline{x}$$

Here $\overline{x}$ is the mean of the x coordinates, $\overline{y}$ is the mean of the natural logarithm of 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 natural logarithm of y coordinates. Use the element indices of the array ***double[] data*** as x coordinates and the element values as y coordinates. For natural logarithm, use the method ***double Math.log()***.

Both result elements are zeros, if at least one data element is non-positive.

```
public static double [] expReg (double [] data) {
    double me, a;
    double (Xym, Xm, ym, X2m, Xm2;) = 0
    ~~foreach (d: data)~~ { (int k=0; k< data.length(); k++)
        Xym+= k*data[k];
        Xm += k;
        ym += data [k];
        X2m += pow (K,2);
        ~~Xm2 +=~~.

    }Xm/= data.length();
    Xm2 = pow(Xm, 2)/ data.length();
    Xym/= data.length();
    ym/= data.length();

    me= (Xym - Xm*ym)/(X2m - Xm2);

    a= ym - me*Xm;
```

ln needed

3

Use the backside, if needed

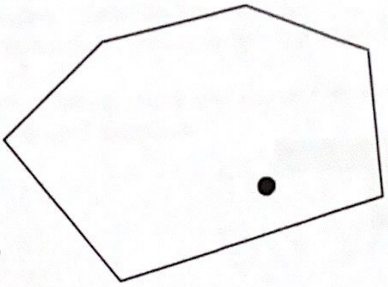
```
    double [] z ={ me, a};

    return z;
```

OOP.MT.170303.M019

Section, Name and ID#: [redacted]

Problem 3: Write a Java function ***public static boolean isInside(double[][] vertex, double x, double y)*** 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, and ***double x*** and ***double y*** coordinates of a point. It checks, if the point is inside the polygon.

Assume and use a method ***boolean toLeft(double x1, double y1, double x2, double y2, double x0, double y0)*** that takes as its arguments coordinates of three points and returns ***true***, if the third point ***(x0, y0)*** is in the left-hand side, when moving from the first point ***(x1, y1)*** to the second one ***(x2, y2)***; and ***false***, if it is in the right-hand side.



```
public static boolean... ( ) {
  ~~double~~ vertex. sort;
for (int k = 0; k < vertex. length(); k++) {
  if ~~boo~~(toLeft (vertex (k], vertex (k+1], X, y );~~{~~) {
      return false;
  }
}
  return true;
```

6 args needed

2

Use the backside, if needed

Problem 3 of 4

OOP.MT.170317.M019

Section, Name and ID#: [redacted]

Problem 4: Write a Java method ***public static void magicOdd(int[][] square)*** that creates a magic square of an *odd* size using the following algorithm:

1. The number ***1*** goes in the middle of the top row; square [0][1] =1;
2. All numbers are then placed one ***column to the right*** and ***one row up*** from the previous number;
3. Whenever the next number placement is above the top row, stay in the same column and place the number in the bottom row (note the place of ***2*** instead of the shaded location);
4. Whenever the next number placement is outside of the rightmost column, stay in the same row and place the number in the leftmost column (note the place of ***3*** instead of the shaded location);
5. When encountering an already filled-in square, place the next number directly below the previous number;
6. When the next number position is outside both a row and a column, place the number directly beneath the previous number (note the place of ***7*** instead of the shaded location).

	9	2	7
8	1	6	8
3	5	7	3
4	9	2	

```
public static .... (int[][] square){

  ~~square[0][1]=~~
                                 int r=0
  rl = square[0].length();  int c=(rl/2)+1
  rown =square.length();
  square[0][(rl/2)+1]=1;

for (int k=0; k≤ ~~dod~~(rl×rown); k++){
                <
      square[~~0~~ r-k][~~(rl/2)+1~~ c+k] = 1+k;
   if (~~0-k~~ r-k < 0){
         ~~k=~~ r = rown -1;
   }
   if ( c+k > rl){
        c = 0;
   }
                              if( r+k<0 && c+k>rl){
                                   r+=2; c-=1;
                              }
   return square.
```

overwrite?

2

Use the backside, if needed