Section, Name and ID#

AMERICAN UNIVERSITY OF ARMENIA

College of Science and Engineering

CS 120 Introduction to Object-Oriented Programming

MIDTERM EXAM

14

Date / Time: Friday, March 17 2017 at 17:30

Duration: 2 hours

Attention: <u>**ANY TYPE OF COMMUNICATION IS STRICTLY PROHIBITED**</u>

<u>***Write down your section, name and ID# at the top of all used pages***</u>

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:

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

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

```
float pi(int n){
  float p = 0;
  for(int i=0; i< n; i++){        pow(-1, i)              pow(5, (2*i + 1))
    p = kahan(p, kahan(16*      )/((2*i + 1)*            ),
                  (-4)*     ) / ((2*i + 1)*
                       pow(-1, i)             pow(239, (2*i + 1))), 0.0), 0.0);
  }
  return p;
}
```

3

Use the backside, if needed

OOP.MT.170317.H006

Section, Name and ID#

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

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

Here $\bar{x}$ is the mean of the ***x*** coordinates, $\bar{y}$ 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[] result = new double[2];
   double  mx;
   double  my;
   double  mxsq;
   double  mprod;

   for ( int i=0; i < data.length; i++){
      mx += i;
      my += Math.log( data[i]);
      mxsq += i*i;
      mprod += i * Math.log( data[i]);
      if( data[i] < 0.0)
         return result;
   }
   mx /= data.length;
   my /= data.length;
   mxsq /= data.length;
   mprod /= data.length;
   result[0] = (mprod - mx * my)/(mxsq - mx * mx);
   result[1] = my - result[0] * mx;

   return result;
}
```

3

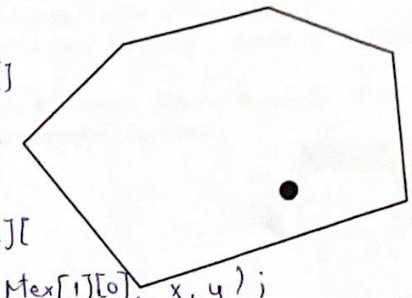
Use the backside, if needed

OOP.MT.170317.H006

Section, Name and ID#

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 isInside(double[][]
                    vertex, double x, double y){

  boolean k;
  k = toLeft(vertex[0][vertex.length-1], vertex[1][
              vertex.length-1], vertex[0][0], vertex[1][0], x, y);


  for (int i=0; i< vertex.length-1; i++)
     if( toLeft(vertex[0][i], vertex[1][i], vertex[0][i+1], vertex[1][i+1], x, y)
        != k)
        return false;


  return true;

}
```

4

OOP.MT.170317.H006

Section, Name and ID:

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;
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 void magicOdd (int[][] square){
int n = square.length;
int x = n/2;
int y = 0;

for (int i=1; i<=n*n ;i++){
  square[y][x]= i;

  X++;
  y--;
  if( x==n  && || y==-1) {
     x--;
     y+=2;}
  else if (n==x)
     x = (x + n) % n;   X=0
  else if (y==-1)
     y =(y + n) %n;   y=n-1
  else if ( square[y][x] != 0) {
     x--;
     y+= 2;

  }
}

}
```

4

Use the backside, if needed

OOP.MT.1J031J.H006