Section, Name and ID#: _____

AMERICAN UNIVERSITY OF ARMENIA
College of Science and Engineering
CS 120 Introduction to Object-Oriented Programming
MIDTERM EXAM

13

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:

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

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

```
float pi(int n) {
    float sum = 0;
    for(int i = 0; i <= n; i++) {
        float step = 16.0f / (i * pow(5.0f, i));
```


OOP.MT.170317.H050

```
float sum = 0
if
for(int i = 0; i <= n; i += 2){
    float step part1 = (16/(i.pow(5,i)));
    float part2 = (4/(i.pow(239,i)));
    float step = Kahan(part1, part2, 0.0f);
    sum = Kahan(sum, step, 0.0f);
}
return sum;
}
```

3

ction, 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.

find m, x

$\bar{y}$ = Math.log(arrayValue)

$\bar{x}$

$\bar{x} = \frac{sum(coords)}{n}$

```
float expReg(double[] dat){
  float xsum=0;
  float ysum=0;
  float ymean=0;
  float xsqsum=0;
  float xymean=0
  int n=data.length;
for(int i=0; i<n; i++){
   float dat=data[i];
   if(dat<0){
Use the backside, if needed  return new float[]{0,0};
}
```

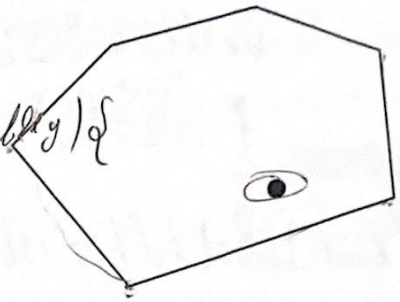

OOP.MT.170317.H050

```
        xsum += i;
        ysum += dat;
        ymean += Math.log(dat);
        xsqsum += Math.pow(dat, 2);
        xymean = i*Math.log(dat);
    }
    Xsum = Xsum/(float)n;
    ymean = ymean/(float)n;
    xsqsum = xsqsum/(float)n;
    xymean = xymean/(float)n;
    float m = (xymean - (ymean * xsum))/(xsqsum - Math.pow(xsum,2));
    float a = ymean - m*xsum;
    return new float[] {m, a};
}
```

ction, 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.



```
isInside(double[][]vertex, double x, double y){
~~float init~~
bool initialvalue=true;
for(int i=0, i<=~~vertex.length~~ vertex[0].length; i+=2){
~~if(i==0){~~
~~initialvalue=~~
~~true~~
~~init~~ if(i==0){
~~initialvalue~~ = isLeft(vertex[i][0], vertex[i][1];
~~vertex[i+1][0], vertex[i+1][1]~~, x, y);
continue;
}
~~if(is~~ bool isLeft = isLeft(vertex[i][0], vertex[i][1],
vertex[i+1][0], vertex[i+1][1], x, y);
if(isLeft != initialvalue){
return false;
}
}
```

Use the backside, if needed

OOP.MT.170315.H050

```
for(int i = vertexes[0].length-1; i >=1; i-=2){
    if(i == vertexes[0].length-1){
     initialvalue = isLeft(vertexes[i][0], vertexes[i][1],
     vertexes[i][0], vertexes[i-1][1], x, y);
     continue;
    }
bool isLeft = isLeft(vertexes[i][0], vertexes[i][1],
    vertexes[i-1][0], vertexes[i-1][1], x, y);
    if(isLeft = initialvalue){
        return false;
    }
}
return true;
}
```

3

ction, 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	

```
magicOdd(int[][] square){
int n=square[0].length;
int positionx=(n/2)+1;
int positiony=0;
for(int i=0; i<pow(n,2); i++){
  for(int i=1; i<=pow(n,2); i++){
    if(positionx==n){
      positionx=0;
    }
    if(positiony<0)
      positiony=n-1;
  }
  square[positiony][positionx]=i;
  positiony-=1;
  positionx+=1;
}
}
```

overwrite?

Use the backside, if needed

3

OOP-MT-1F03-FH050