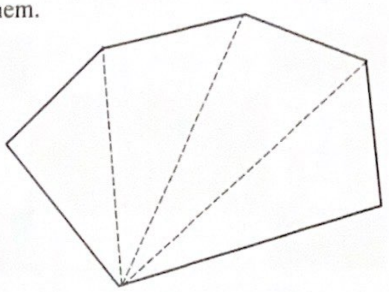
Section, Name and ID#:

Problem 3: Write a Java function ***public static double area(double[][] vertex)*** 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. It returns polygon's area as follows:

1. Divides the polygon into triangles by connecting the ***first*** vertex with the ***n***th and ***(n+1)***st vertices;
2. Adds the areas of the constructed triangles using the formula $area = \sqrt{p(p-a)(p-b)(p-c)}$, where ***a***, ***b*** and ***c*** are the sides and ***p = (a + b + c) / 2***.

You may assume and use a method ***double dist(double x1, double y1, double x2, double y2)*** that takes as its arguments coordinates of two points and returns the distance between them.



2

```
void space (int length) {
for (;length < 0;i--) {
  cout << " ";
 }
}

void symbol (char st, int lengt) {
   for (; lengt < 0;i--) {
     cout << st;
   }
  }

void pattern (int size) {
for (int ~~i~~ i <= size ; i++) {
 ~~[illegible]~~ s~~pace (size)~~;
       space (size-1)
       symbol (*; i)
       symbol (*; i-1)
       cout << endl;
 }
For (int i = size; i > 0; i--)
     space(size-i)
       symbol (*;i);
       symbol (*;i-1);
         cout endl;
     }
```

Use the backside, if needed

```
int main() {
   pattern ("enter size");
}
```

OOP.MT.170317.L045

Section, Name and ID#:

Problem 4: Write a Java method ***public static void magic4N(int[][] square)*** that creates a magic square of a ***4N-by-4N*** size using the following algorithm:

1. Creates an array of the same size as ***int[][] square*** and fills it forward with successive integers assigning *1* to the top-left element;
2. Creates anther array of the same size as ***int[][] square*** and fills it backward with successive integers assigning *1* to the bottom-right element;
3. Divides the original ***int[][] square*** into 16 blocks of the same size – 4 blocks per row and column. In the on-diagonal (shaded) blocks copies the elements from the first array, and in the off-diagonal blocks copies the elements from second array.

1	2					7	8
9	10					15	16
		19	20	21	22		
		27	28	29	30		
		35	36	37	38		
		43	44	45	46		
49	50					55	56
57	58					63	64

		62	61	60	59		
		54	53	52	51		
48	47					42	41
40	39					34	33
32	31					26	25
24	23					18	17
		14	13	12	11		
		6	5	4	3		

```
c i8stream>              sorting
~~[illegible]~~>

      int main () {
   int  a[99];
     for (int i=0 i >(sizeof(a)/ sizeof(a[0])); i++) {
     a[i] = rand()%100+1; // random number generator from 100 to 1

 int temp ;
 int b =  sizeof(a)/ sizeof(a[0])) ;
    }
   ~~for(int q=0 ;q<(sizeof(a) / sizeof(a[0])), q++) {~~
        for (

        ~~for(int q=(sizeof(a) / sizeof(a[0])) q>=0; q--) {~~
          ~~for(int i=0~~
   {
          for(int q = b; q>0; q--){

           for (int z=0; z<b; z++) {
        if (a[z] >a[z+1]) {
              temp = a[z+1];
       }       a[z+1] = a[z]
          }    a[z] = temp;
```

OOP.MT.170317.L045