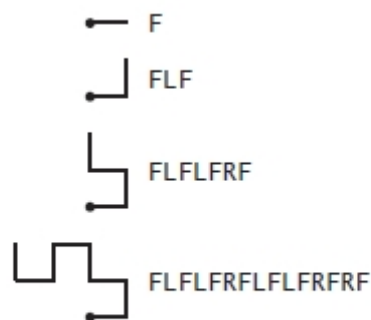


## WarmUp Prog. Exercise for SE (March 8)

Solve the following 4 problems by programming. Test all your programs in your own environment. Put all your code with input data and output data in separate sub-folders for each problem, zip them in a package like [2345AdamSmith-WarmUpProg.zip](#) and send it to [c.max@yeah.net](mailto:c.max@yeah.net) with an email subject like: **SE-2345AdamSmith: WarmUp Prog. Exercise.** before (including) March 14.

**P1 Dragon curves.** Write a program to get an integer N from command-line arguments and print the instructions for drawing the dragon curve of order N. The instructions are strings of F, L, and R characters, where F means “draw line while moving 1 unit forward,” L means “turn left,” and R means “turn right.” A dragon curve of order n is formed when you fold a strip of paper in half n times, then unfold to right angles. The key to solving this problem is to note that a curve of order n is a curve of order n-1 followed by an L followed by a curve of order n-1 traversed in reverse order, and then to figure out a similar description for the reverse curve.



*Dragon curves of order 0, 1, 2, and 3*

(Tips: You may search “**Dragon** curves” or “龙形曲线” online for help.

For example: <https://www.cnblogs.com/WhyEngine/p/4013245.html> )

**P2** Write a program to output the following digit diamond in the console.

```

      0
    1 0 1
  2 1 0 1 2
3 2 1 0 1 2 3
4 3 2 1 0 1 2 3 4
5 4 3 2 1 0 1 2 3 4 5
6 5 4 3 2 1 0 1 2 3 4 5 6
7 6 5 4 3 2 1 0 1 2 3 4 5 6 7
8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8
9 8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8 9
8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8
7 6 5 4 3 2 1 0 1 2 3 4 5 6 7
6 5 4 3 2 1 0 1 2 3 4 5 6
5 4 3 2 1 0 1 2 3 4 5
4 3 2 1 0 1 2 3 4
3 2 1 0 1 2 3
2 1 0 1 2
1 0 1
0

```

**P3** Write 2 overloading static methods to construct a String that compose a diamond:

```

public static String diamond ();
public static String diamond (int n, char... color);
System.out.print( diamond() ) will output as left below:

```

```

      *
     ***
    * * *
   * * *
  * * *
 * * *
* * *
*****
*****
*****
*****
****
***
**
*
*

```

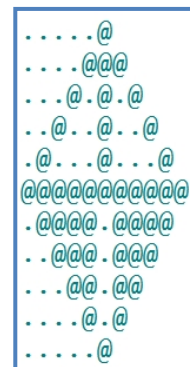
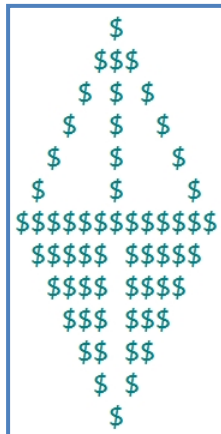
```

      *
     ***
    * * *
   * * *
  *****
 ***
**
*
*

```

System.out.print( diamond( 5 ) ) will output as right above.

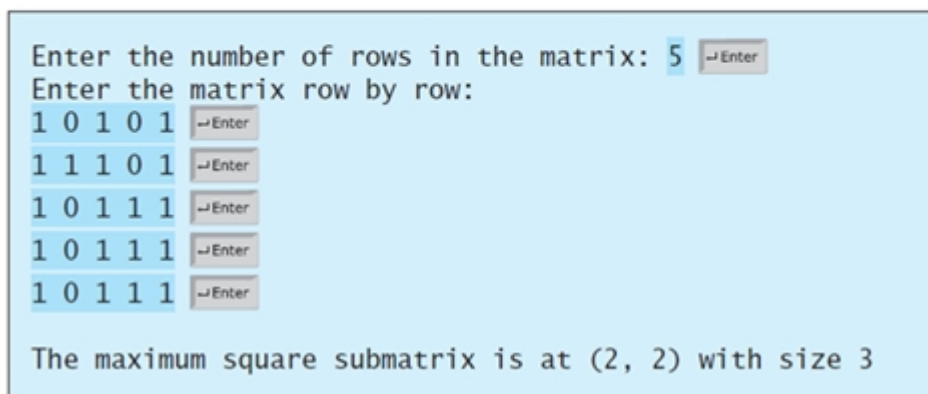
System.out.print( diamond( 7, '\$' ) ) will output as left below:



`System.out.print( diamond( 6, '@', '.' ) )` will output as right above.

(Tips: in `diamond(int n, char... color)` declaration, where declared a **variable-length argument (varargs)** `color` that works like a `char[]` in the method body and may have a `color` without any entry, i.e., `color.length` may be 0 and `color = {}` .

**P4 (Largest block)** Given a square matrix with the elements 0 or 1, write a program to find a maximum square submatrix whose elements are all 1s. Your program should prompt the user to enter the number of rows in the matrix. The program then displays the location of the first element in the maximum square submatrix and the number of the rows in the submatrix. Here is a sample run:



**Tips:** If `int[][] a = int[N][N]` with elements 0 or 1,

let `int[][] s = int[N][N]`, `s[r][c]` denotes the size of the maximum sub-matrix start from `a[r][c]` towards right-down direction.

Then we could compute `s[r][c]` in the following way (using pseudo-code):

```
s[N-1][N-1 .. 0] = a[N-1][N-1 .. 0]    // last row
s[N-2 .. 0][N-1] = a[N-2 .. 0][N-1]    // last column

for (r from N-2 down to 0)
  for (c from N-2 down to 0)
    s[r][c] = (a[r][c] == 0) ? 0 :
              1 + min(s[r][c+1], s[r+1][c], s[r+1][c+1])
```

The Output may be changed to the following example.

```
a[][]:
[1, 0, 1, 0, 1]
[1, 1, 1, 0, 1]
[1, 0, 1, 1, 1]
[1, 0, 1, 1, 1]
[1, 0, 1, 1, 1]
s[][]:
[1, 0, 1, 0, 1]
[1, 1, 1, 0, 1]
[1, 0, 3, 2, 1]
[1, 0, 2, 2, 1]
[1, 0, 1, 1, 1]
The maximum square submatrix is at (2, 2) with size 3
```