Problem 10: Flower Power

Difficulty: Hard

Author: Mike Trinka, Fort Worth, Texas, United States

Problem Background

Everyone likes numbers, right? They're so elegant and pretty. If you don't think so, maybe you could try looking at them using a different perspective. Maybe you could think of them as flowers!

Problem Description

In this problem you'll take ordinary numbers and turn them into flowers. Each flower will represent a digit of the number using its petals. Each flower can have a maximum of 8 petals, represented either by a pipe (|), dash (-), forward slash (/) or backslash (\) based on its location around the flower. Petals are added to the flower starting in the upper left corner, moving clockwise. Here's the full listing of all of the possible "digits":

0 * 	1 \ * 	2 | * 	3 |/ * 	4 |/ *-
5 |/ *- \	6 |/ *- |	7 |/ *- /|	8 |/ -*- /|	

Notice that there's no flower for nine; numbers greater than or equal to nine will need to be represented by multiple flowers. Moving from right to left, each flower will represent a value nine times greater than the one before it. For example, the number 42 would be represented as:

```
\|/ \|/
 *-  *-
     |\
```

The left flower's value is multiplied by 9, so represents 36 (4 * 9 = 36). The right flower is not multiplied at all, so represents 6. The two flowers added together represent our intended value of 42.

Sample Input

The first line of your program's input, received from the standard input channel, will contain a positive integer representing the number of test cases. Each test case will include a single line containing a positive integer to represent as one or more flowers.

```
3
12345
678
90
```

Sample Output

For each test case, your program must print three lines of text containing the flower representation of the provided number. Separate flowers with a single space. Remember, do not print any trailing whitespace on any line; the ← symbol in the sample below is used to indicate where line breaks occur. (Most languages provide a function called trim() or strip() which can be used to remove excess whitespace from the ends of a string; check your language's documentation for details.)

```
\   \|/ \|/ \|/ \|/←
 *   *- -*-  *   *-←
    /|\ /|\      |\←
\|/ \|/ \|/←
-*-  *   *←
/|\←
\   \←
 *   *   *←
←
```