Objective:

- The following problems are taken from a Programming Contests in order to give you exposure and confidence to solve such open and unconventional problems.

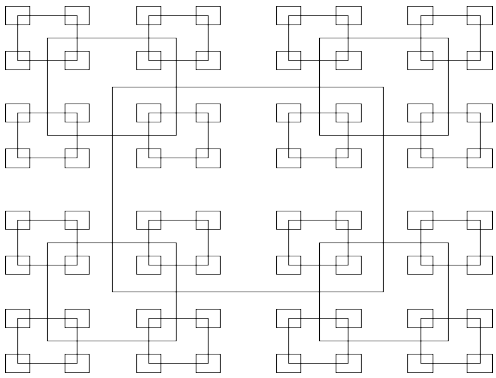
Task-1: **All Squares**

Time Limit: 3 Seconds

Geometrically, any square has a unique, well-defined centre point. On a grid this is only true if the sides of the square are an odd number of points long. Since any odd number can be written in the form 2k+1, we can characterize any such square by specifying k, that is we can say that a square whose sides are of length 2k+1 has size k. Now define a pattern of squares as follows.

1. The largest square is of size k (that is sides are of length 2k+1) and is centred in a grid of size 1024 (that is the grid sides are of length 2049).
2. The smallest permissible square is of size 1 and the largest is of size 512, thus $1 \le 512 \le k$.
3. All squares of size $k > 1$ have a square of size k div 2 centred on each of their 4 corners. (Div implies integer division, thus 9 div 2 = 4).
4. The top left corner of the screen has coordinates (0,0), the bottom right has coordinates (2048, 2048).

Hence, given a value of k, we can draw a unique pattern of squares according to the above rules. Furthermore any point on the screen will be surrounded by zero or more squares. (If the point is on the border of a square, it is considered to be surrounded by that square). Thus if the size of the largest square is given as 15, then the following pattern would be produced.



Write a program that will read in a value of k and the coordinates of a point, and will determine how many squares surround the point.

Input and Output

Input will consist of a series of lines. Each line will consist of a value of k and the coordinates of a point. The file will be terminated by a line consisting of three zeroes (0 0 0).

Output will consist of a series of lines, one for each line of the input. Each line will consist of the number of squares containing the specified point, right justified in a field of width 3.

Sample input

```
500 113 941
300 100 200
300 1024 1024
0 0 0
```

Sample output

5
0
1

Task-2:

On the standard Touch-ToneTM telephone dial, the digits are mapped onto the alphabet (minus the letters *Q* and *Z*) as shown in the diagram below:

In order to make their phone numbers more memorable, service providers like to find numbers that spell out some word (called a **mnemonic**) appropriate to their business that makes that phone number easier to remember. For example, the phone number for a recorded time-of-day message in some localities is 637-8687 (NERVOUS).

Imagine that you have just been hired by a local telephone company to write a function **ListMnemonics** that will generate all possible letter combinations that correspond to a given number, represented as a string of digits. For example, if you call ***ListMnemonics("723")*** your program should generate the following 27 possible letter combinations that correspond to that prefix:

PAD	PBD	PCD	RAD	RBD	RCD	SAD	SBD	SCD
PAE	PBE	PCE	RAE	RBE	RCE	SAE	SBE	SCE
PAF	PBF	PCF	RAF	RBF	RCF	SAF	SBF	SCF

The function declaration is listed below:

void ListMnemonics(char* str);

Task-3:

You're standing at the base of a staircase and are heading to the top. A small stride will move up one stair, a large stride advances two. You want to count the number of ways to climb the entire staircase based on different combinations of large and small strides. For example, a staircase of three steps can be climbed in three different ways: via three small strides or one small stride followed by one large stride or one large followed by one small. A staircase of four steps can be climbed in five different ways (enumerating them is an exercise left to reader :-).

Write the recursive function **int countWays(int numStairs)** that takes a positive **numStairs** value and returns the number of different ways to climb a staircase of that height taking strides of one or two stairs at a time.

Here's a hint about the recursive structure of the problem: consider the options you have at each stair. You must either take a small stride or a large stride; either will take you closer to the goal and therefore represents a simpler instance of the same problem that can be handled recursively. What is the simplest possible situation and how is it handled?