Problem 4 – CATastrophe

The world is in big CATastrophe. Most people love cats and obey them! Look at the one on the right! So cute, so furry, you will be its slave forever! This cat's name is "**Котангенс**". Enjoy it while you can!

Your task is simple. Just pet the cat a bit and you are done. Actually, you need to solve one hard programming problem in addition to petting Котангенс.

You are given a C# code with valid syntax. You need to find all primitive types declaration in what scope are defined. Scopes are method scope, loop scope, conditional scope.

- Conditional statements scope – find all primitive data type variable names declared in "**if-else**" code blocks. Do not check for "**switch**" code blocks.
- Loops scope – find all primitive data type variables names declared in "**for**", "**while**" and "**foreach**" loops. Do not check for "**do-while**" loops.
- Method scope – find all primitive data type variable names declared in methods (and not declared in loops and conditional statements).

Primitive data types are: **sbyte**, **byte**, **short**, **ushort**, **int**, **uint**, **long**, **ulong**, **float**, **double**, **decimal**, **bool**, **char**, **string**. Nullable types should be counted.

See the example below for better clarification.

For your convenience (and because the cat is cute) the C# code have some limitations to make the task easier.

- The code will be fully understandable by you for your current level at C# Part 2. There will be no strange structures, object oriented programming, cats lurking around the text, unknown keywords, CATaclysms, whatsoever…
- There will not be any class level variables.
- All variables will be declared with their keywords.
- The code may or may not have the namespace declaration.
- All method declarations will be static without any access modifiers. Access modifiers are "public", "private", "internal" and "protected".
- The code will not be necessary compiling but will be with valid C# syntax.
- All curly brackets will be formatted and put on correct separate lines.
- All method names will be on the same line with the static keyword.
- There will not be any other static declarations except for the methods.
- There will not be any commented code or code in strings.
- The code will be easily readable. There will not be any unnecessary line breaks and empty new lines. Unnecessary spaces may occur here and there though (Котангенс is playful).

You will be given **N** lines with C# code. Find all declared variable names of primitive types in which scope they live. Order the variable names in the way they appear in the code. If there are no variables in certain scope, print "{scope} -> None". Print the result on three separate lines in the following format:

"Methods -> {number of variables} -> {var1, var2, var3}"

"Loops -> {number of variables} -> {var1, var2, var3}"

"Conditional Statements -> {number of variables} -> {var1, var2, var3}"

See the example below.

Mrrr (no CATastrophes around Telerik Academy though…)

Input

The input data should be read from the console.

On the first input line you will be given the number **N**.

On the next **N** lines you will read the C# code.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

Print three lines in the described format above.

Constraints

- **N** will be between **10** and **200**, inclusive.
- Each line will have at most **1 000** characters.
- The code will follow the rules described above.
- Allowed working time for your program: **0.1 seconds**.
- Allowed memory: **16 MB**.

Examples

Example input

```
79
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Digits
{
    class Program
    {
        static bool CheckCurrentPattern(int[,] numbers, bool[,] pattern, int row, int col, int digit)
        {
            for (int i = 0; i < pattern.GetLength(0); i++)
            {
                for (int j = 0; j < pattern.GetLength(1); j++)
                {
                    if (pattern[i, j])
```

```
                {
                                        string letter = "text";

                    if (numbers[row + i, col + j] != digit)
                    {
                        return false;
                    }
                }
                                else
                                {
                                        short someNumber = 12;
                                }
            }
        }

        return true;
    }

    static bool IsEven(int number)
    {
        return number % 2 == 0;
    }

    static void Main(string[] args)
    {
        int? n = int.Parse(Console.ReadLine());

        int[,] numbers = new int[n, n];

        for (int i = 0; i < n; i++)
        {
            var currentNumbers = Console.ReadLine().Split(new[] { ' ' },
StringSplitOptions.RemoveEmptyEntries);
            for (int j = 0; j < currentNumbers.Length; j++)
            {
                numbers[i, j] = int.Parse(currentNumbers[j]);
            }
        }

        List<bool[,]> patterns = new List<bool[,]>();
        InitPatterns(patterns);

        int sum = 0;
        for (int row = 0; row < numbers.GetLength(0) - 4; row++)
        {
            for (int col = 0; col < numbers.GetLength(1) - 2; col++)
            {
                for (int pattern = 0; pattern < patterns.Count; pattern++)
                {
                    var currentPattern = patterns[pattern];
                                        decimal anotherNumber = 1.2m;
                    if (CheckCurrentPattern(numbers, currentPattern, row, col, pattern))
                    {
                        sum += pattern;
                    }
                }
            }
        }

        Console.WriteLine(sum);
    }
```

```
    }
}
```

Example output

```
Methods -> 6 -> row, col, digit, number, n, sum
Loops -> 8 -> i, j, i, j, row, col, pattern, anotherNumber
Conditional Statements -> 2 -> letter, someNumber
```