Problem 4 – Brackets! Again?!

Here is a picture of cat! Enjoy it while you can!

This cat wants you to solve some C# problems. Why? Because! It may give a couple of myaus if you manage to do it but that is not a promise! Let's call the cat "**Котангенс**". Pretty cat-ish name, right? No? ☹

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. Your task is to find where all method invokes happen. For each declared method, you need to tell all invoked methods in it. 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, whatsoever…
- 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 unnecessary line breaks and empty new lines. Unnecessary spaces may occur here and there though (Котангенс is playful).
- Brackets are your best friends. ;)

You will be given **N** lines with C# code. Find all method invokes in which particular method declaration are called. Order the method invokes in the order they are called in code. Print them in the following format: "{method declaration} -> {method1 call, method2 call, method 3 call}". If there are no method calls in certain declaration, print "{method declaration} -> None". See the example below.

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.

For each method declaration **print separate line**, in which you show all invokes in format described 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

```
170
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Digits
{
    class Program
    {
        static void InitPatterns(List<bool[,]> patterns)
        {
            // zero
            patterns.Add(new bool[,]
            {
                {true, true, true},
                {true, false, true},
                {true, false, true},
                {true, false, true},
                {true, true, true},
            });

            // one
            patterns.Add(new bool[,]
            {
                {false, false, true},
                {false, true, true},
                {true, false, true},
                {false, false, true},
                {false, false, true},
            });

            // two
            patterns.Add(new bool[,]
            {
                {false, true, false},
                {true, false, true},
                {false, false, true},
                {false, true, false},
                {true, true, true},
            });

            // three
            patterns.Add(new bool[,]
        {
            {true, true, true},
            {false, false, true},
            {false, true, true},
            {false, false, true},
            {true, true, true},
        });

        // four
        patterns.Add(new bool[,]
        {
            {true, false, true},
            {true, false, true},
            {true, true, true},
            {false, false, true},
            {false, false, true},
        });

        // five
        patterns.Add(new bool[,]
        {
            {true, true, true},
            {true, false, false},
            {true, true, true},
            {false, false, true},
            {true, true, true},
        });

        // six
        patterns.Add(new bool[,]
        {
            {true, true, true},
            {true, false, false},
            {true, true, true},
            {true, false, true},
            {true, true, true},
        });

        // seven
        patterns.Add(new bool[,]
        {
            {true, true, true},
            {false, false, true},
            {false, true, false},
            {false, true, false},
            {false, true, false},
        });

        // eight
        patterns.Add(new bool[,]
        {
            {true, true, true},
            {true, false, true},
            {false, true, false},
            {true, false, true},
            {true, true, true},
        });

        // nine
        patterns.Add(new bool[,]
        {
            {true, true, true},
            {true, false, true},
            {false, true, true},
            {false, false, true},
            {true, true, true},
        });
    }

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

        return true;
    }

    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];
                    if (CheckCurrentPattern(numbers, currentPattern, row, col, pattern))
                    {
                        sum += pattern;
                    }
                }
            }
        }
```

```
            Console.WriteLine(sum);
        }
    }
}
```

Example output

```
InitPatterns -> Add, Add, Add, Add, Add, Add, Add, Add, Add, Add
CheckCurrentPattern -> GetLength, GetLength
Main -> Parse, ReadLine, ReadLine, Split, Parse, InitPatterns, GetLength, GetLength, CheckCurrentPattern,
WriteLine
```