Recursion

The Power of Calling a Method from Itself

Svetlin Nakov

Telerik Corporation

www.telerik.com

Table of Contents

What is Recursion?

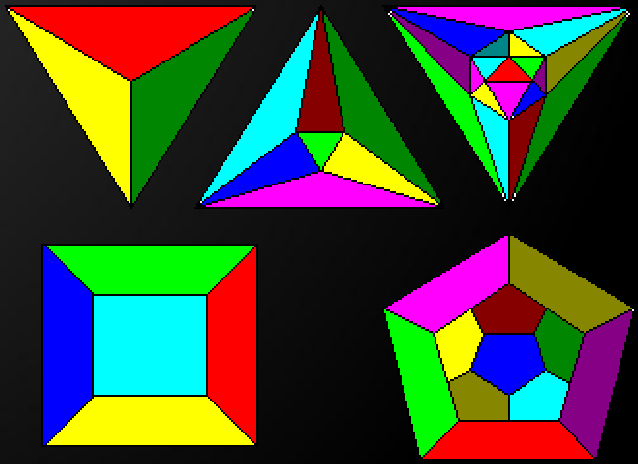
- Recursion is when a methods calls itself
  - Very powerful technique for implementing combinatorial and other algorithms
- Recursion should have
  - Direct or indirect recursive call
    - The method calls itself directly or through other methods
  - Exit criteria (bottom)
    - Prevents infinite recursion

Recursive Factorial – Example

- Recursive definition of n! (n factorial):

```
n! = n * (n-1)! for n >= 0
0! = 1
```

  - 5! = 5 * 4! = 5 * 4 * 3 * 2 * 1 * 1 = 120
  - 4! = 4 * 3! = 4 * 3 * 2 * 1 * 1 = 24
  - 3! = 3 * 2! = 3 * 2 * 1 * 1 = 6
  - 2! = 2 * 1! = 2 * 1 * 1 = 2
  - 1! = 1 * 0! = 1 * 1 = 1
  - 0! = 1



Recursive Factorial – Example

- Calculating factorial:
  - 0! = 1
  - n! = n* (n-1)!, n>0

```
static decimal Factorial(decimal num)
{
    if (num == 0)
        return 1;
    else
        return num * Factorial(num - 1);
}
```

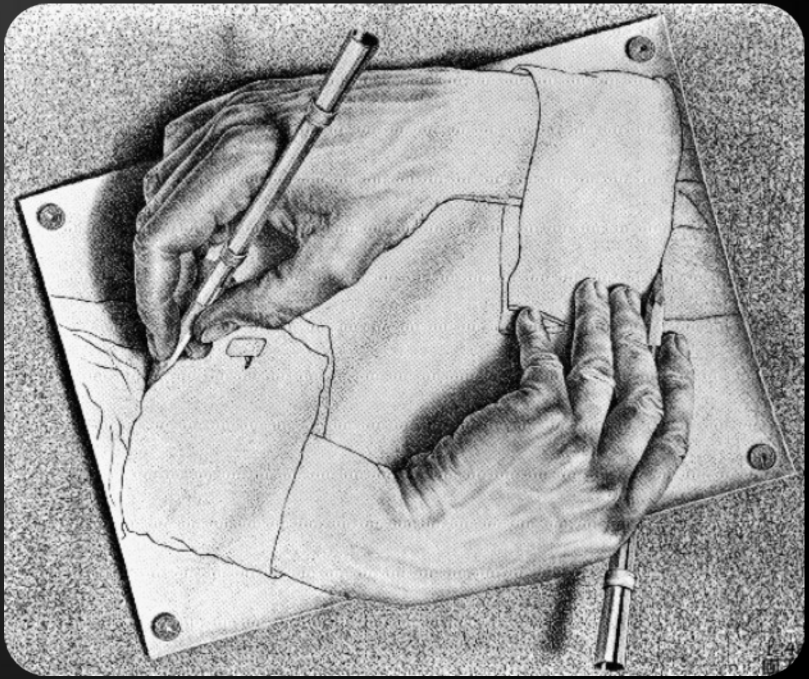
The bottom of the recursion

Recursive call: the method calls itself

- Don't try this at home!
  - Use iteration instead

Recursive Factorial

Live Demo



Generating 0/1 Vectors

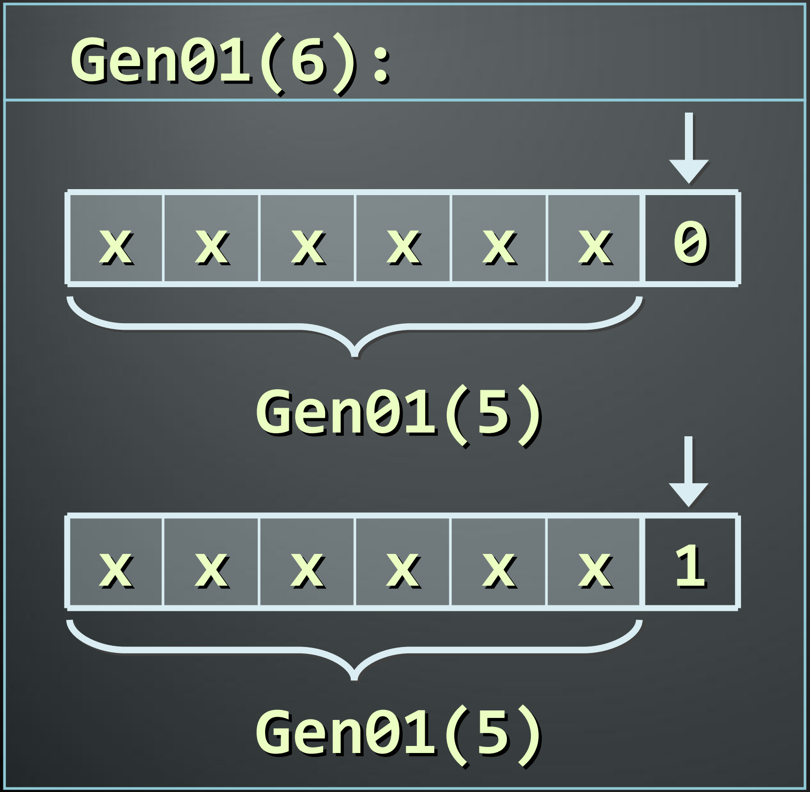
- How to generate all 8-bit vectors recursively?

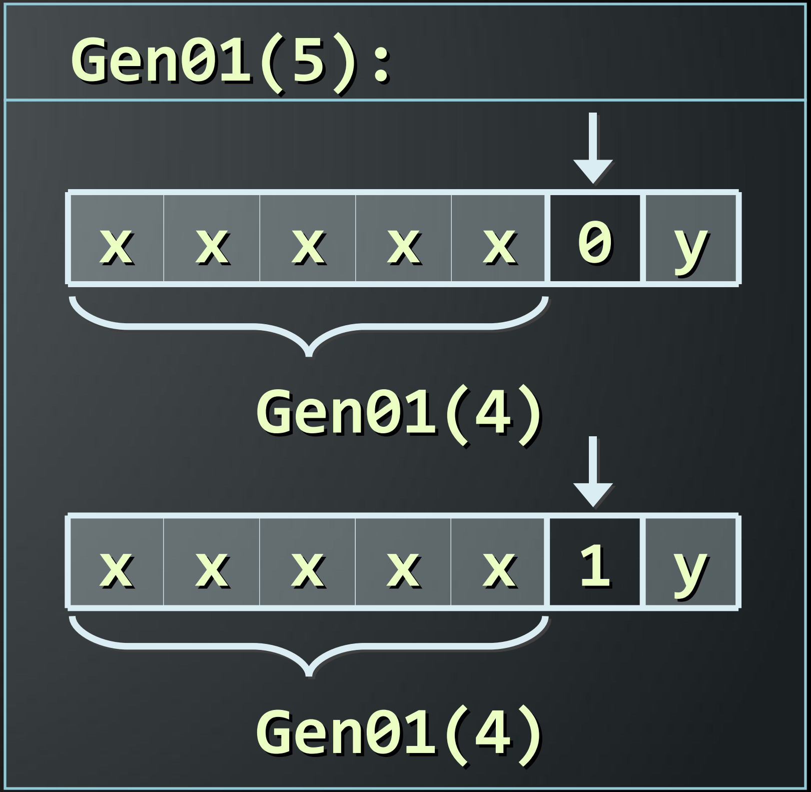
```
00000000
00000001
...
01111111
10000000
...
11111110
11111111
```

- How to generate all n-bit vectors?

Generating 0/1 Vectors (2)

- Algorithm `Gen01(n)`: put `0` and `1` at the last position n and call `Gen01(n-1)` for the rest:





```
...
Gen01(-1) → Stop!
```

Generating 0/1 Vectors (3)

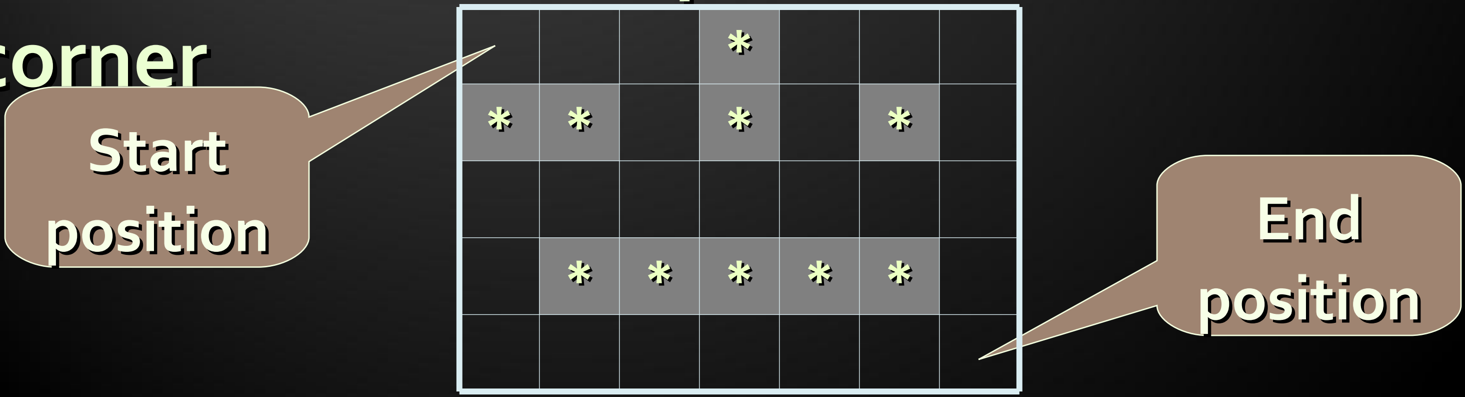
```
static void Gen01(int index, int[] vector)
{
    if (index == -1)
        Print(vector);
    else
        for (int i=0; i<=1; i++)
        {
            vector[index] = i;
            Gen01(index-1, vector);
        }
}
static void Main()
{
    int size = 8;
    int[] vector = new int[size];
    Gen01(size-1, vector);
}
```

Generating 0/1 Vectors

Live Demo

Finding All Paths in a Labyrinth

- We are given a labyrinth
  - Represented as matrix of cells of size M x N
  - Empty cells are passable, the others (*) are not
- We start from the top left corner and can move in the all 4 directions: left, right, up, down
- We need to find all paths to the bottom right corner



Finding All Paths in a Labyrinth (2)

- There are 3 different paths from the top left corner to the bottom right corner:

1)

0	1	2	*			
*	*	3	*		*	
6	5	4				
7	*	*	*	*	*	
8	9	10	11	12	13	14

2)

0	1	2	*	8	9	10
*	*	3	*	7	*	11
		4	5	6		12
	*	*	*	*	*	13
						14

3)

0	1	2	*			
*	*	3	*		*	
		4	5	6	7	8
	*	*	*	*	*	9
						10

telerik

Finding All Paths in a Labyrinth (2)

- Suppose we have an algorithm `FindExit(x,y)` that finds and prints all paths to the exit (bottom right corner) starting from position `(x,y)`
- If (x,y) is not passable, no paths are found
- If (x,y) is already visited, no paths are found
- Otherwise:
  - Mark position (x,y) as visited (to avoid cycles)
  - Find all paths to the exit from all neighbor cells: (x-1,y) , (x+1,y) , (x,y+1) , (x,y-1)
  - Mark position (x,y) as free (can be visited again)

Find All Paths: Algorithm

- Representing the labyrinth as matrix of characters (in this example 5 rows and 7 columns):

```
static char[,] lab =
{
    {' ', ' ', ' ', '*', ' ', ' ', ' '},
    {'*', '*', ' ', '*', ' ', '*', ' '},
    {' ', ' ', ' ', ' ', ' ', ' ', ' '},
    {' ', '*', '*', '*', '*', '*', ' '},
    {' ', ' ', ' ', ' ', ' ', ' ', 'e'},
};
```

  - Spaces (' ') are passable cells
  - Asterisks ('*') are not passable cells
  - The symbol 'e' is the exit (can be multiple)

Find All Paths: Algorithm (2)

```
static void FindExit(int row, int col)
{
    if ((col < 0) || (row < 0) || (col >= lab.GetLength(1))
        || (row >= lab.GetLength(0)))
    {
        // We are out of the labyrinth -> can't find a path
        return;
    }

    // Check if we have found the exit
    if (lab[row, col] == 'e')
    {
        Console.WriteLine("Found the exit!");
    }

    if (lab[row, col] != ' ')
    {
        // The current cell is not free -> can't find a path
        return;
    }
```

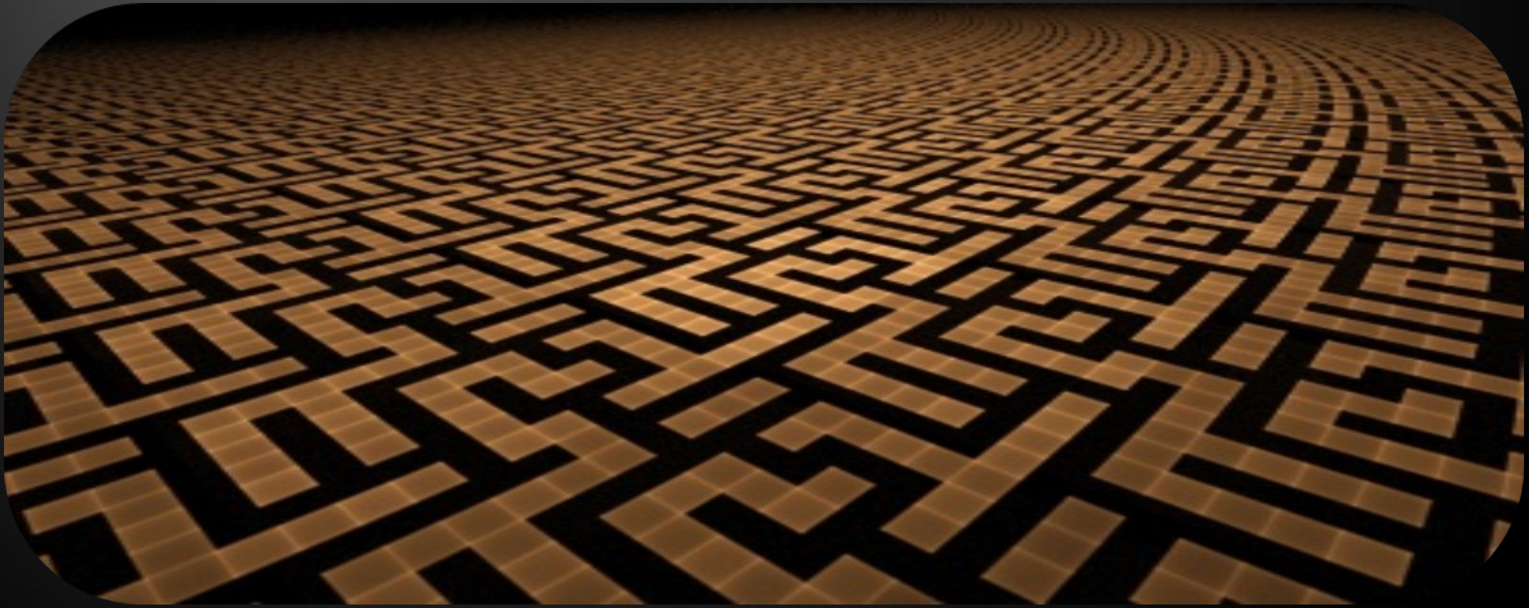
(example continues)

Find All Paths: Algorithm (3)

```
    // Temporary mark the current cell as visited
    lab[row, col] = 's';

    // Invoke recursion the explore all possible directions
    FindExit(row, col-1); // left
    FindExit(row-1, col); // up
    FindExit(row, col+1); // right
    FindExit(row+1, col); // down

    // Mark back the current cell as free
    lab[row, col] = ' ';
}

static void Main()
{
    FindExit(0, 0);
}
```

Find All Paths in a Labyrinth

Live Demo



Find All Paths and Print Them

- How to print all paths found by our recursive algorithm?
  - Each move's direction can be stored in array

```
static char[] path =
    new char[lab.GetLength(0) * lab.GetLength(1)];
static int position = 0;
```

  - Need to pass the movement direction at each recursive call
  - At the start of each recursive call the current direction is appended to the array
  - At the end of each recursive call the last direction is removed form the array

Find All Paths and Print Them (2)

```
static void FindPathToExit(int row, int col, char direction)
{
    ...
    // Append the current direction to the path
    path[position++] = direction;

    if (lab[row, col] == 'e')
    {
        // The exit is found -> print the current path
    }
    ...

    // Recursively explore all possible directions
    FindPathToExit(row, col - 1, 'L'); // left
    FindPathToExit(row - 1, col, 'U'); // up
    FindPathToExit(row, col + 1, 'R'); // right
    FindPathToExit(row + 1, col, 'D'); // down
    ...

    // Remove the last direction from the path
    position--;
}
```

Find and Print All Paths in a Labyrinth

Live Demo

Recursion or Iteration?

When to Use and When to Avoid Recursion?

Recursion Can be Harmful!

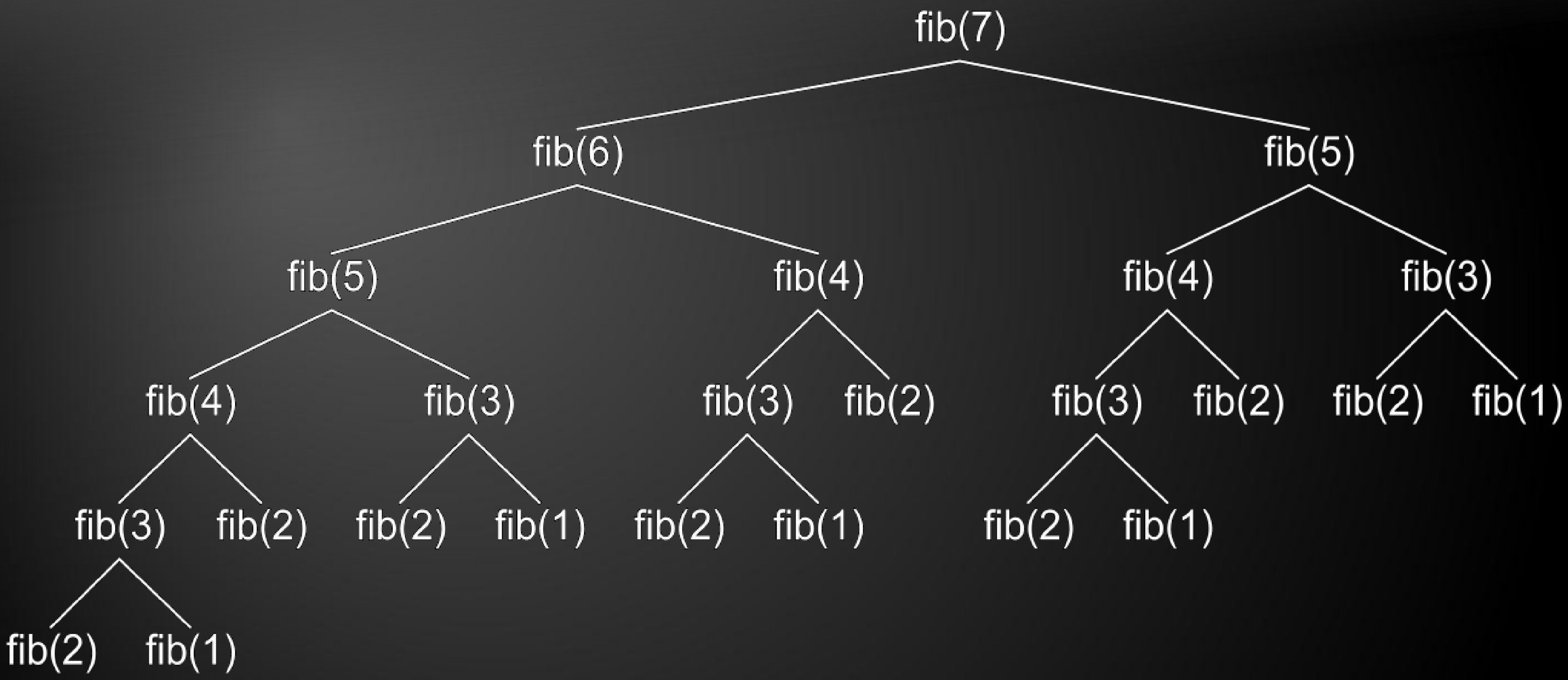
- When used incorrectly the recursion could take too much memory and computing power
- Example:

```
static decimal Fibonacci(int n)
{
    if ((n == 1) || (n == 2))
        return 1;
    else
        return Fibonacci(n - 1) + Fibonacci(n - 2);
}

static void Main()
{
    Console.WriteLine(Fibonacci(10)); // 89
    Console.WriteLine(Fibonacci(50)); // This will hang!
}
```

Harmful Recursion

Live Demo

How the Recursive Fibonacci Calculation Works?



- fib(n) makes about fib(n) recursive calls
- The same value is calculated many, many times!

telerik

Fast Recursive Fibonacci

- Each Fibonacci sequence member can be remembered once it is calculated
  - Can be returned directly when needed again

```
static decimal[] fib = new decimal[MAX_FIB];
static decimal Fibonacci(int n)
{
    if (fib[n] == 0)
    {
        // The value of fib[n] is still not calculated
        if ((n == 1) || (n == 2))
            fib[n] = 1;
        else
            fib[n] = Fibonacci(n - 1) + Fibonacci(n - 2);
    }
    return fib[n];
}
```

Fast Recursive Fibonacci

Live Demo

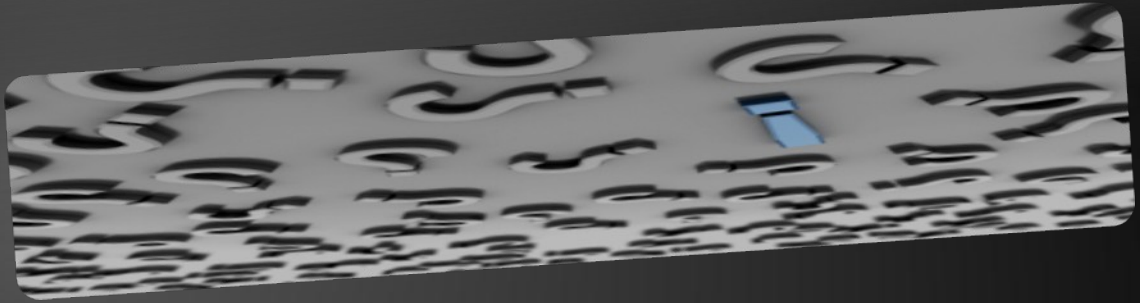
telerik

When to Use Recursion?

- Avoid recursion when an obvious iterative algorithm exists
  - Examples: factorial, Fibonacci numbers
- Use recursion for combinatorial algorithm where at each step you need to recursively explore more than one possible continuation
  - Examples: permutations, all paths in labyrinth
  - If you have only one recursive call in the body of a recursive method, it can directly become iterative (like calculating factorial)

Summary

- Recursion means to call a method from itself
  - It should always have a bottom at which recursive calls stop
- Very powerful technique for implementing combinatorial algorithms
  - Examples: generating combinatorial configurations like permutations, combinations, variations, etc.
- Recursion can be harmful when not used correctly



Questions?

Exercises

1. Write a recursive program that simulates execution of n nested loops from 1 to n. Examples:

```
                                  1 1 1
                                  1 1 2
                                  1 1 3
                1 1               1 2 1
n=2    ->       1 2    n=3    ->  ...
                2 1               3 2 3
                2 2               3 3 1
                                  3 3 2
                                  3 3 3
```

telerik

Exercises (2)

1. Write a recursive program for generating and printing all the combinations with duplicates of k elements from n-element set. Example:

   n=3, k=2 → (1 1), (1 2), (1 3), (2 2), (2 3), (3 3)

2. Write a recursive program for generating and printing all permutations of the numbers 1, 2, ..., n for given integer number n. Example:

   n=3 → {1, 2, 3}, {1, 3, 2}, {2, 1, 3},
   {2, 3, 1}, {3, 1, 2},{3, 2, 1}

Exercises (3)

1. Write a recursive program for generating and printing all ordered k-element subsets from n-element set (variations V^k_n).

   Example: n=3, k=2

   (1 1), (1 2), (1 3), (2 1), (2 2), (2 3), (3 1), (3 2), (3 3)

2. Write a program for generating and printing all subsets of k strings from given set of strings.

   Example: s = {test, rock, fun}, k=2

   (test rock), (test fun), (rock fun)

Exercises (4)

1. We are given a matrix of passable and non-passable cells. Write a recursive program for finding all paths between two cells in the matrix.
2. Modify the above program to check whether a path exists between two cells without finding all possible paths. Test it over an empty 100 x 100 matrix.
3. Write a program to find the largest connected area of adjacent empty cells in a matrix.
4. Implement the BFS algorithm to find the shortest path between two cells in a matrix (read about Breath-First Search in Wikipedia).

Exercises (5)

1. We are given a matrix of passable and non-passable cells. Write a recursive program for finding all areas of passable cells in the matrix.
2. Write a recursive program that traverses the entire hard disk drive C:\ and displays all folders recursively and all files inside each of them.