# Rekurze

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava

Prezentace ke dni 12. února 2020



# Osnova přednášky

Rekurze

- Co si představit pod pojmem rekurze?
- Faktoriál
- Fibonacciho posloupnost

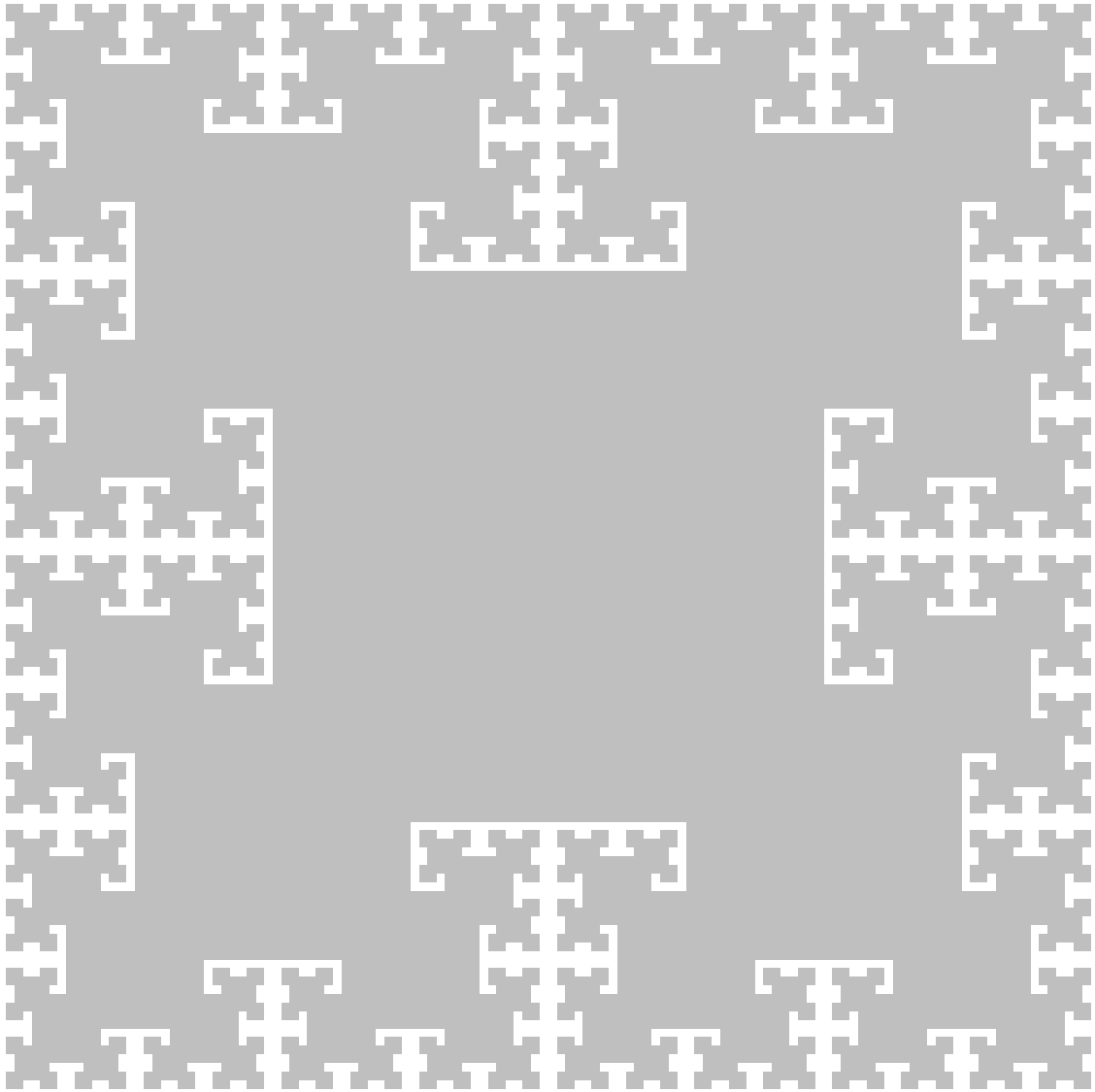
# Rekurze

Co si představit pod pojmem rekurze?

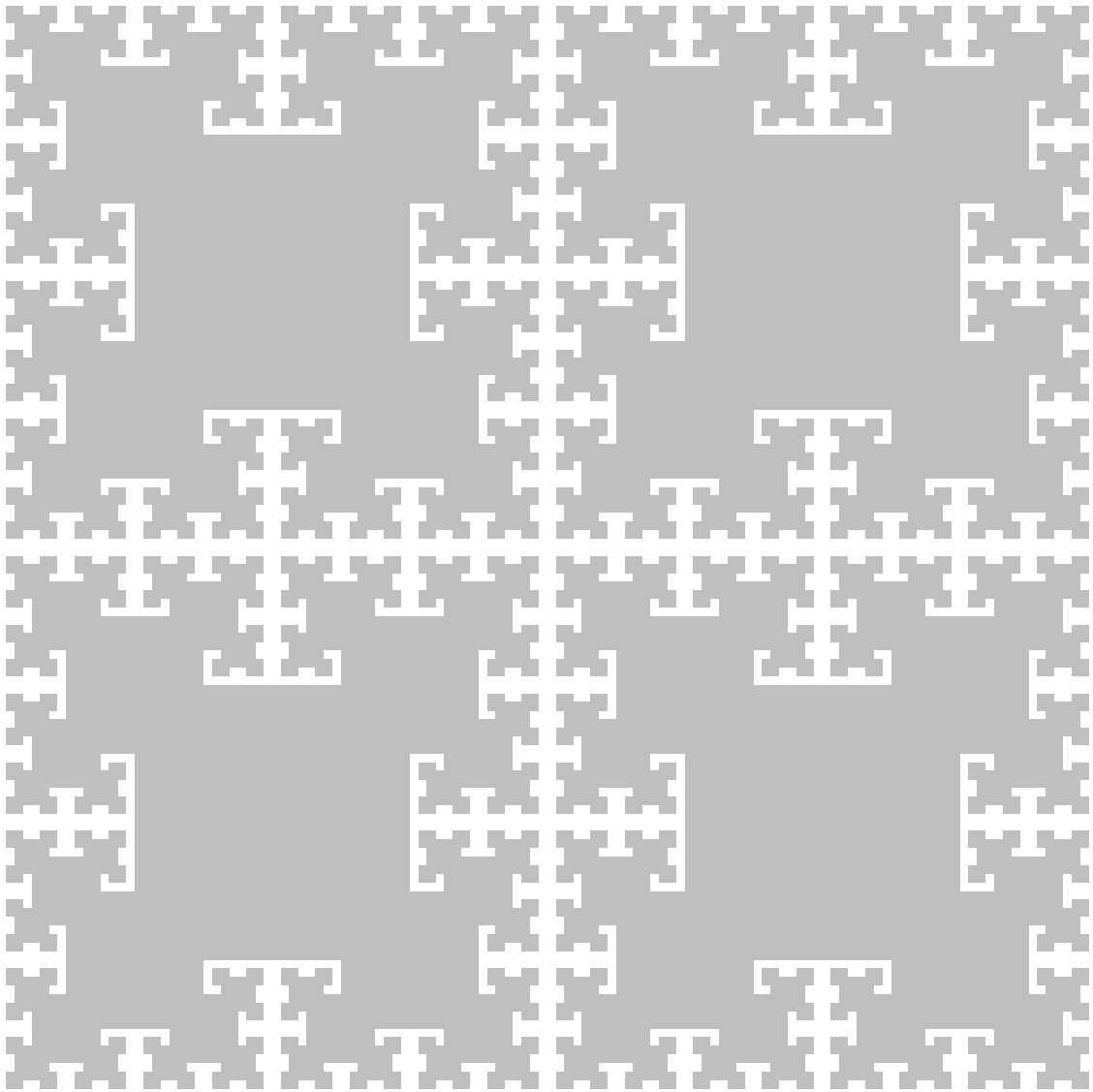
1. „seber všechny hračky!“,
2. soběpodobnost, fraktály,
3. z hlediska algoritmů, je to algoritmus, který v určitém okamžiku výpočtu volá sám sebe,
4. rozklad na menší podproblémy, princip **rozděl a panuj**, (divide et impera),
5. dělení končí u atomického problému – řešení je známo například z definice,
6. rekurze je jistým způsobem ekvivalentní iteraci (cyklu).

# Ukázka rekurze graficky

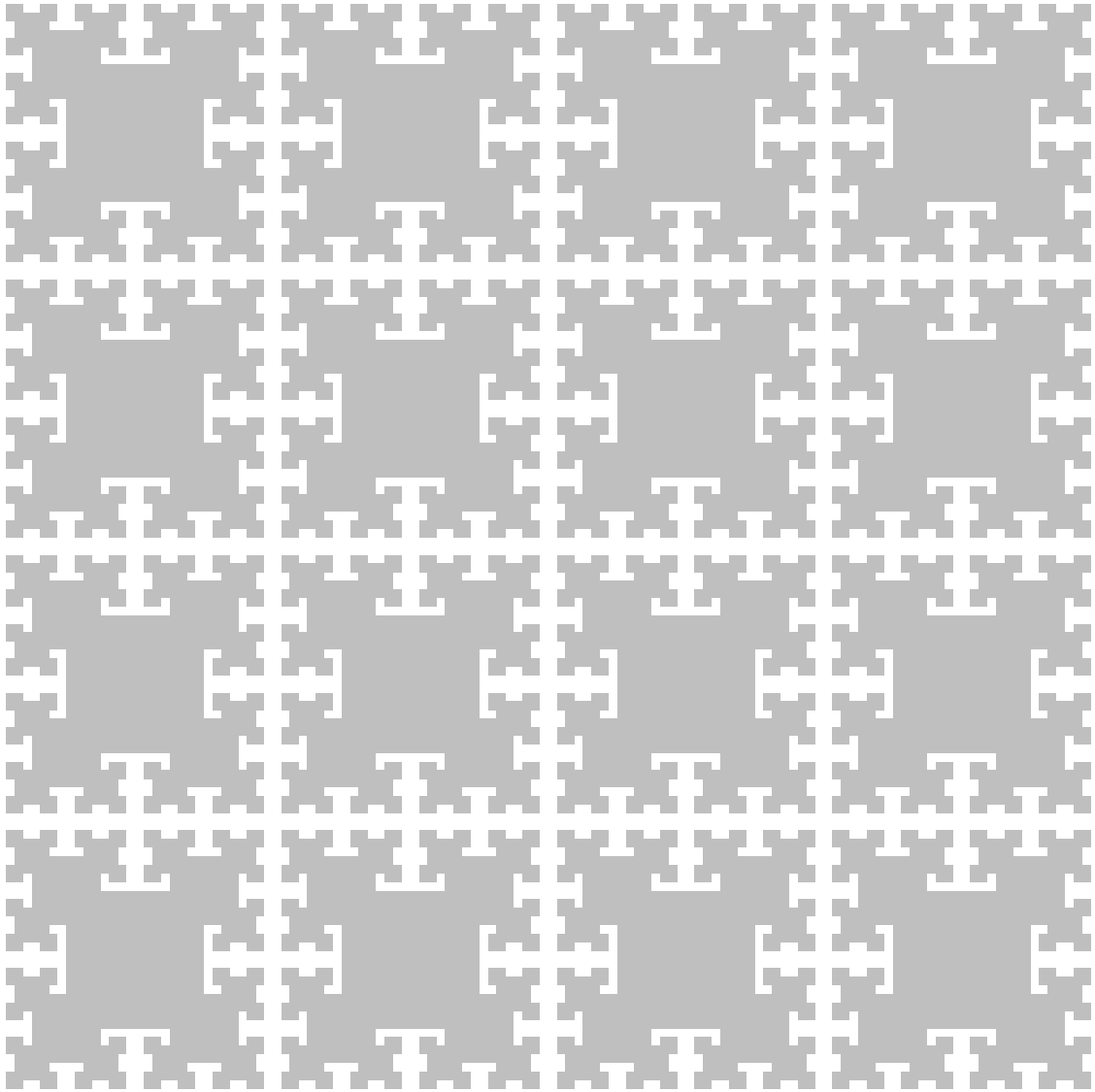
# Ukázka rekurze graficky



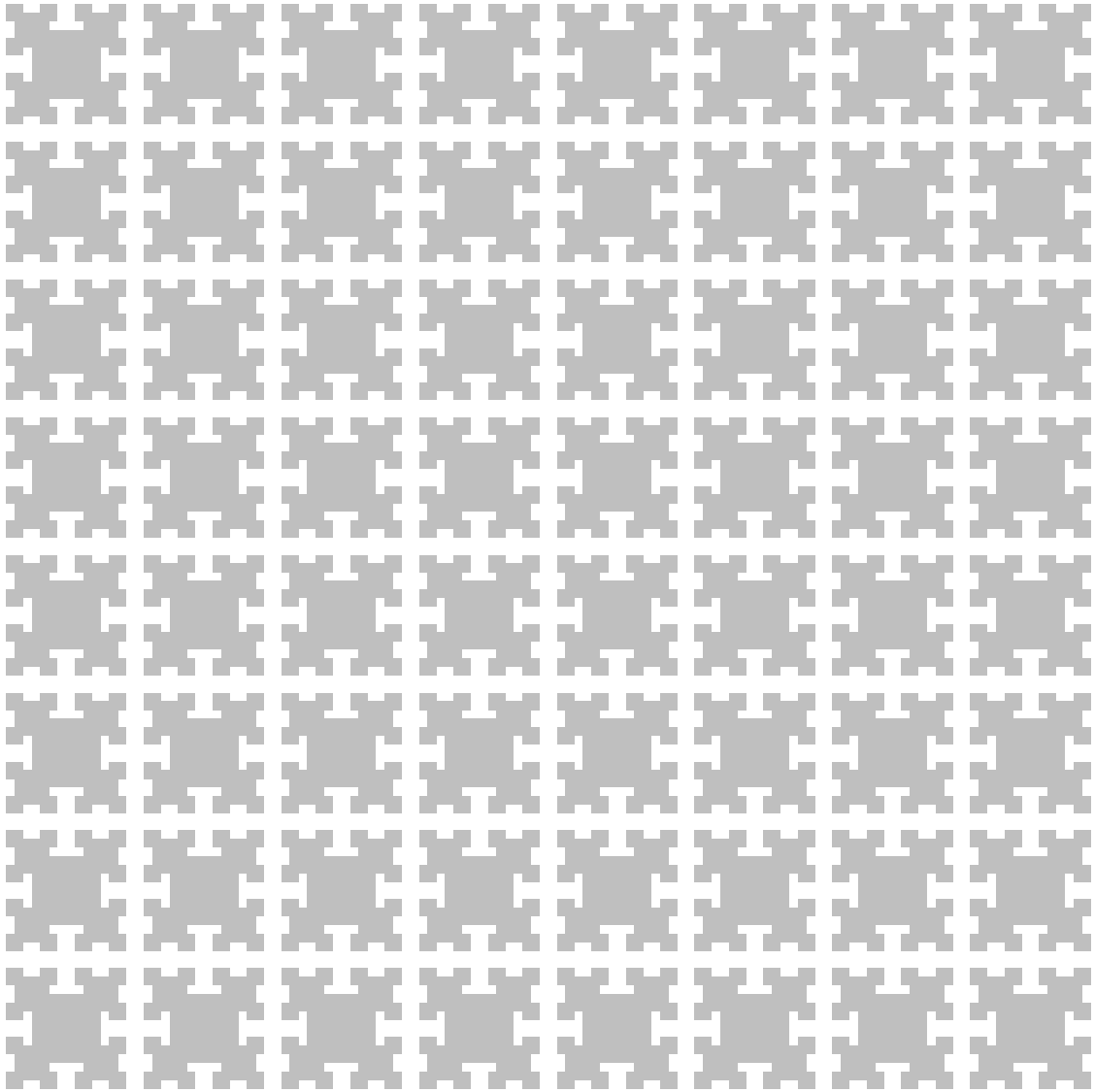
# Ukázka rekurze graficky



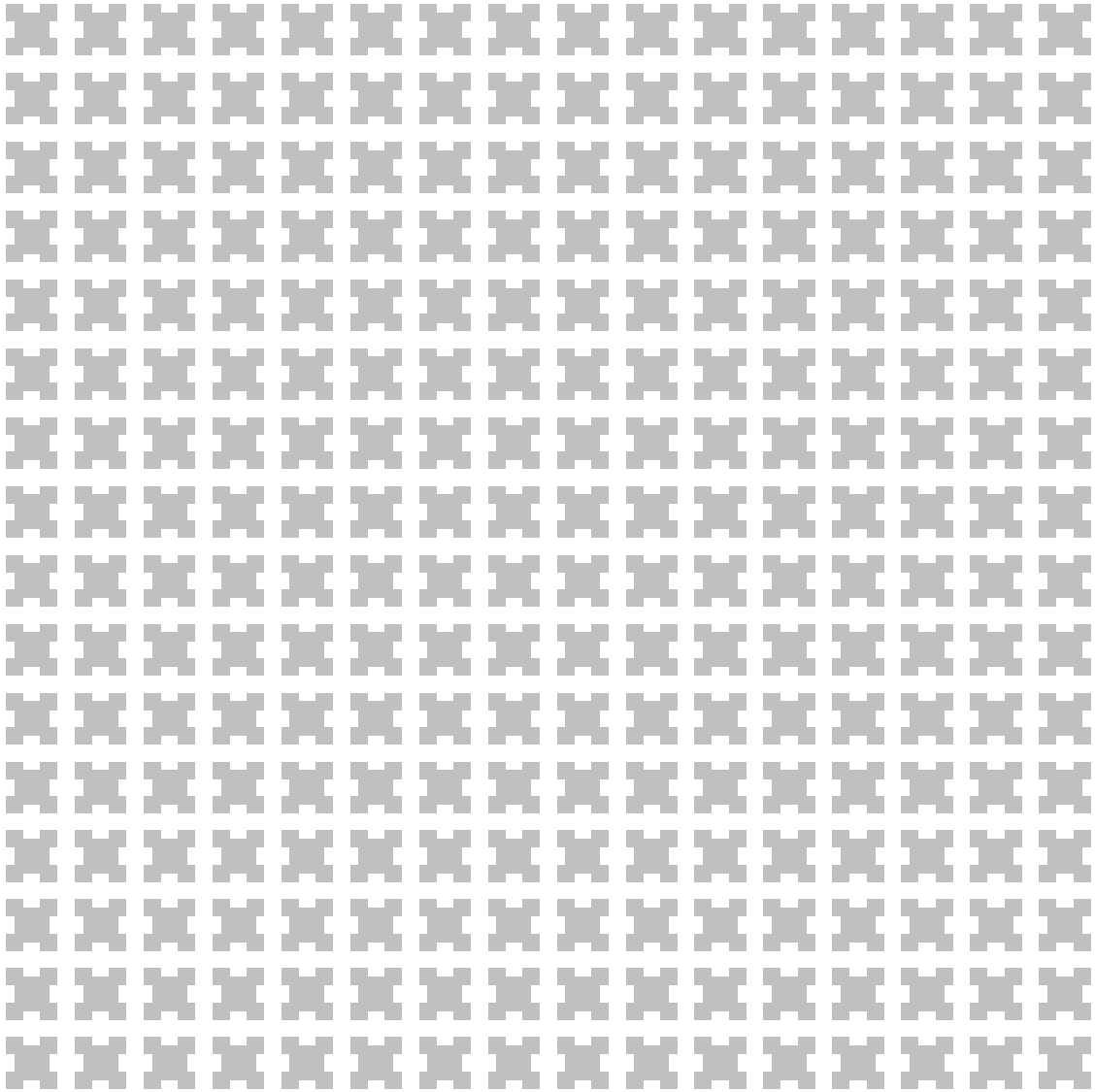
# Ukázka rekurze graficky



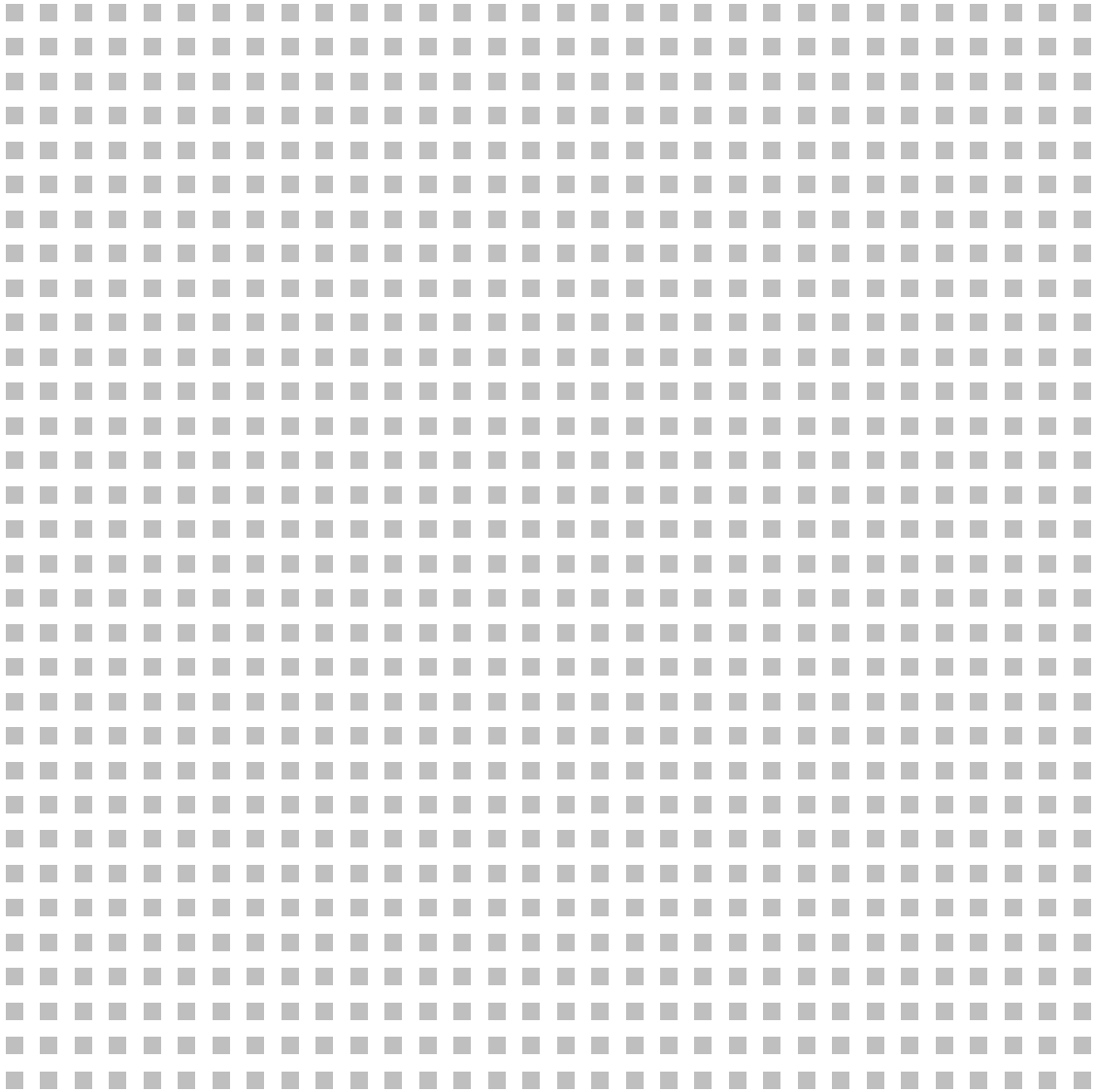
# Ukázka rekurze graficky



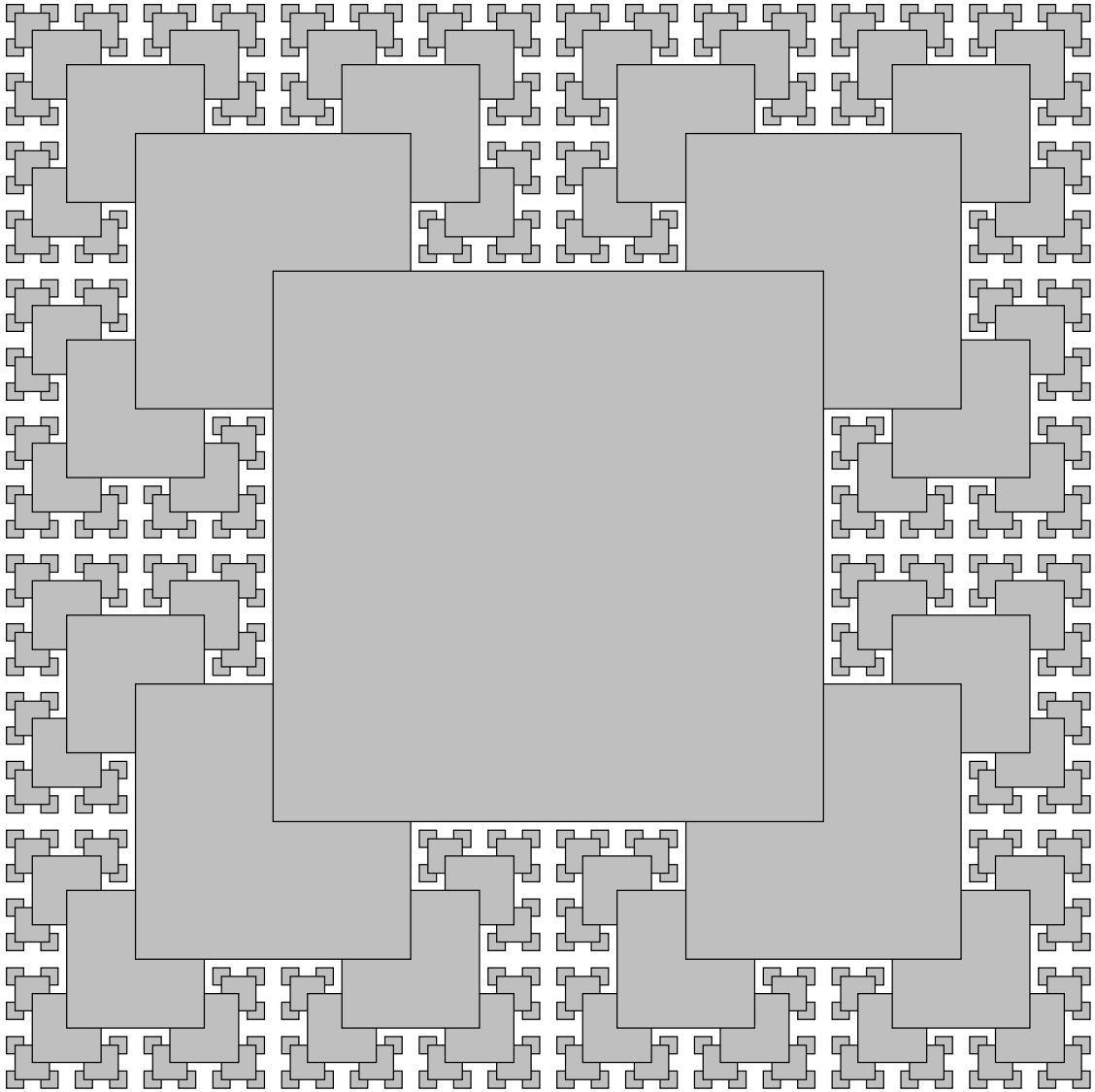
# Ukázka rekurze graficky



# Ukázka rekurze graficky



# Ukázka rekurze graficky



# Funkce vykreslující ukázkový fraktál

```
1 void Draw(const int Level, const double CenterX, const double CenterY,
      const double A)
2 {
3    if (Level > MaxLevel)
4       return;
5    double A2 = A / 2;
6    Draw(Level + 1, CenterX - A2, CenterY - A2, A2);
7    Draw(Level + 1, CenterX + A2, CenterY - A2, A2);
8    Draw(Level + 1, CenterX + A2, CenterY + A2, A2);
9    Draw(Level + 1, CenterX - A2, CenterY + A2, A2);
10   DrawSquare(CenterX - A2, CenterY - A2, A);
11 }
```

## Ukázka volání

```
Draw(0, 0, 0, 50);
```

# Některé pojmy

- přímá rekurze – funkce volá sama sebe.
- nepřímá rekurze – funkce **A** volá funkci **B** a ta opět volá funkce **A**.
- ukončovací podmínka – jistá část řešení problému je atomická, řešení je známo např. z definice problému.

# Faktoriál

Princip rekurze si ukážeme na výpočtu faktoriálu čísla $n$. Funkce faktoriál je definována jako

$$n! = \begin{cases} 1 & pro\ n = 0 \\ n * (n-1)! & pro\ n \geq 1 \end{cases}$$

Pokud funkci faktoriál místo $n!$ označíme jako $f(n)$ můžeme faktoriál zapsat jako

$$f(n) = \begin{cases} 1 & pro\ n = 0 \\ n * f(n-1) & pro\ n \geq 1 \end{cases}$$

Což vede ke známému vyjádření

$$n! = n*(n-1)! = n*(n-1)(n-2)! = \cdots = n*(n-1)*(n-2)*\cdots*1*0!$$

# Výpočet hodnot funkce faktoriál

| Vstupní hodnota | Výpočet hodnoty | Hodnota faktoriálu |
|---:|---:|---:|
| 0 | podle definice | 1 |
| 1 | $1 * 1$ | 1 |
| 2 | $2 * 1$ | 2 |
| 3 | $3 * 2$ | 6 |
| 4 | $4 * 6$ | 24 |
| 5 | $5 * 24$ | 120 |
| 6 | $6 * 120$ | 720 |
| 7 | $7 * 720$ | 5 040 |
| 8 | $8 * 5\,040$ | 40 320 |
| 9 | $9 * 40\,320$ | 362 880 |
| 10 | $10 * 362\,880$ | 3 628 800 |

# Faktoriál – ukázka kódu

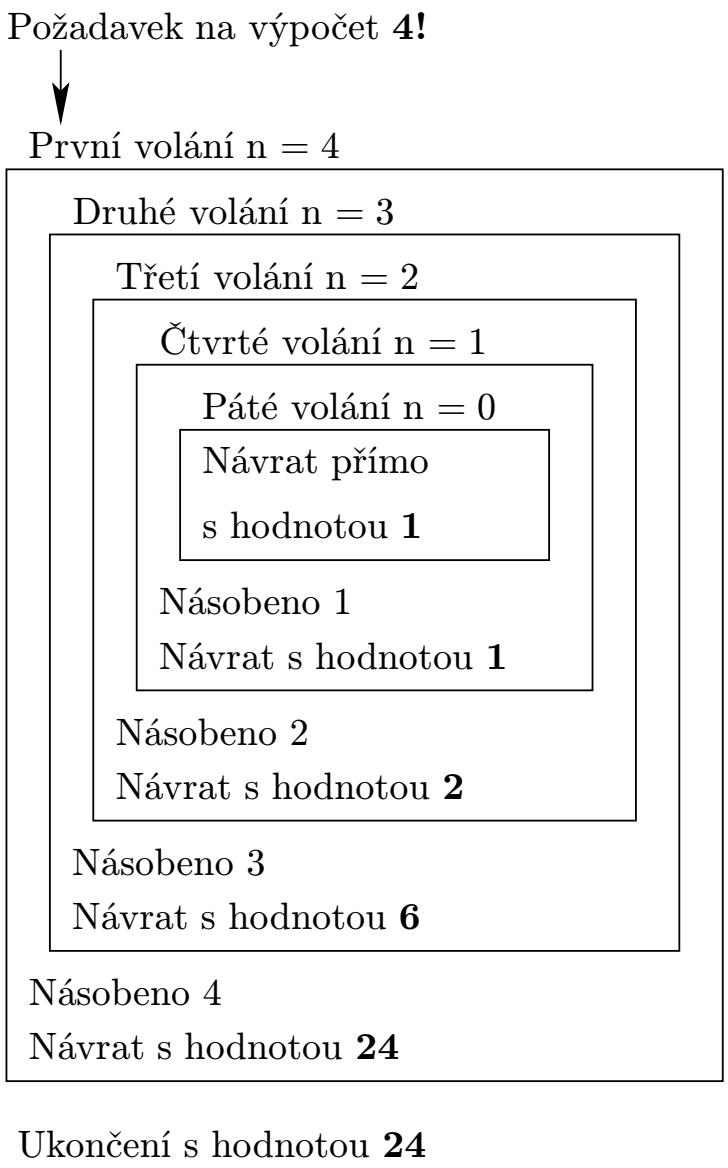
```
1 int factorial(const int n)
2 {
3    if(n == 0)
4       return 1;
5    return n * factorial(n - 1);
6 }
```

# Faktoriál – výpočet

Co se děje při rekurzívním výpočtu? Jak je zajištěno, aby se uchovaly všechny hodnoty proměnných v okamžiku rekurentního volání dílčí úlohy a při jejím ukončení, tj.návratu z poslední rekurentní funkce či procedury byly předány správné hodnoty?

- jednotlivé úlohy seřadíme do řady, aktuální úloha je na konci
- v okamžiku dokončení výpočtu, je výsledek předán volající metodě, ruší se lokální proměnné
- je to **zásobník**, LIFO
- každým rekurzivním voláním metody vzniká nová množina všech parametrů a lokálních proměnných
- mají sice stejné identifikátory jako při prvním volání, ale jejich hodnoty jsou jiné

# Faktoriál – schéma rekurze



# Fibonacciho posloupnost

aneb *Kolik potomků – párů králíků bude mít po roce jeden původní králičí pár?*

1. Na pole vypustíme pár králíků.
2. Králíci dospívají po jednom měsíci, březost trvá jeden měsíc. To znamená, že po dvou měsících samice vyprodukuje nový pár králíků.
3. Králíci neumírají a nejsou nemocní.
4. Každý pár králíků vždy vyprodukuje jeden nový pár samec/samice.

Úloha byla poprvé publikována roku 1202 Leonardem Pisano (Leonardo Fibonacci) v knize *Liber Abacci*

# Fibonacciho posloupnost

**Řešení**:

1. Po prvním měsíci bude na poli stále jeden pár králíků. Na poli je 1 pár.
2. Na konci druhého měsíce vznikne jeden nový pár králíků. Na poli jsou 2 páry.
3. Na konci třetího měsíce původní pár vyprodukuje nový pár, druhý pár dospívá. Na poli jsou 3 páry.
4. Na konci čtvrtého měsíce, původní pár vyprodukoval další pár, pár narozený před dvěma měsíci má svoje první potomstvo. Na poli je 5 párů.

**Na konci n-tého měsíce je počet párů králíků roven počtu nových párů králíků (což je počet párů v měsíci n-2) plus počet párů žijících v minulém měsíci.**

# Fibonacciho posloupnost

## Matematické vyjádření:

$$F(n) = \begin{cases} 0 & pro\ n = 0 \\ 1 & pro\ n = 1 \\ F(n-1) + F(n-2) & pro\ n > 1 \end{cases}$$

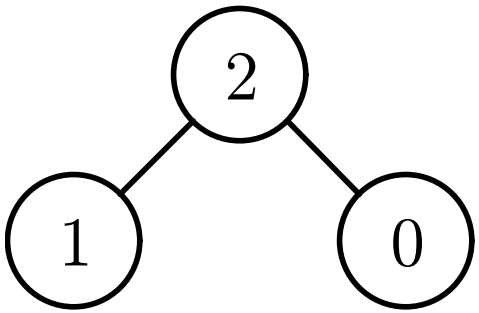
$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, \ldots$$

$$F_0 = 0,\ F_1 = 1,\ F_2 = 1,\ F_3 = 2,\ F_4 = 3,\ F_5 = 5,\ F_6 = 8,\ F_7 = 13,\ \ldots$$

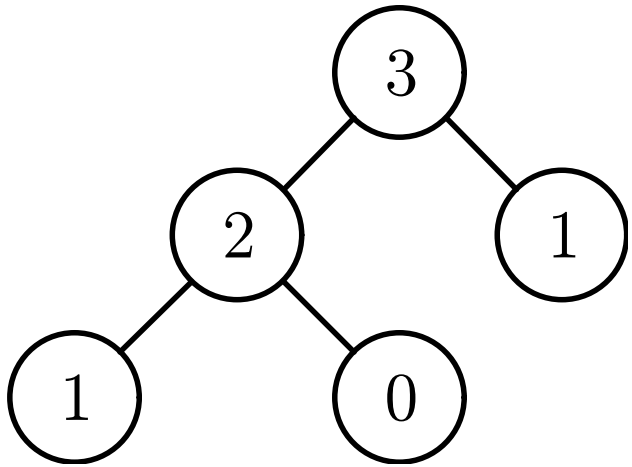
# Fibonacciho posloupnost – ukázka kódu

```
1 int F(const int n)
2 {
3    if (n == 0)
4       return 0;
5    if (n == 1)
6       return 1;
7    return F(n - 1) + F(n - 2);
8 }
```

# Fibonacciho posloupnost – ukázka výpočtu

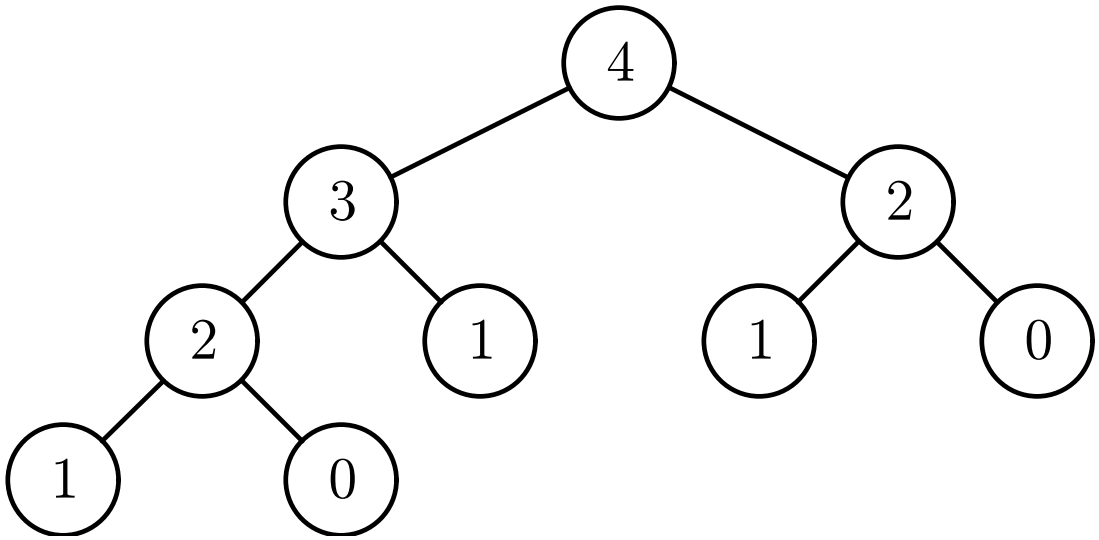



$n = 2$



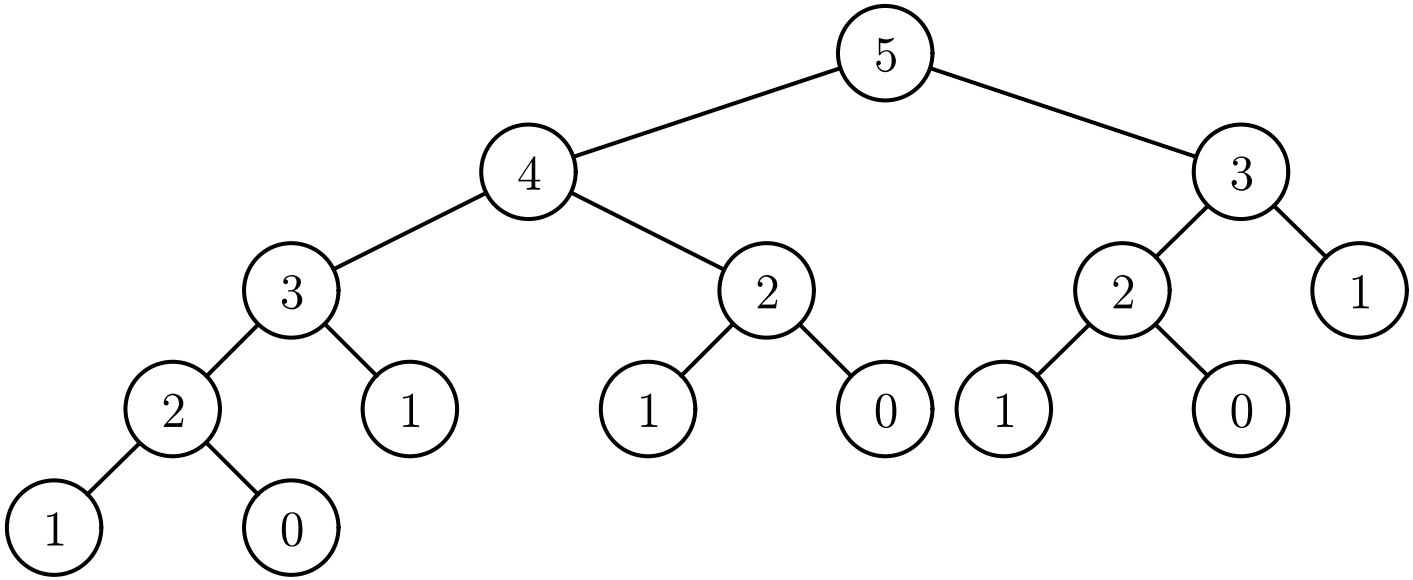

$n = 3$

# Fibonacciho posloupnost – ukázka výpočtu




$n = 4$

# Fibonacciho posloupnost – ukázka výpočtu




$n = 5$

# Vyhledání maxima hodnot v poli iterativně

```
1 int maximumIter(int[] a, const int N)
2 {
3    int max = a[0];
4    for(int i = 0; i < N; i++)
5    {
6       if (a[i] > max)
7          max = a[i];
8    }
9    return max;
10 }
```

# Vyhledání maxima hodnot v poli rekurzivně

```
1 int maxRecursive(int[] a, const int left, const int right)
2 {
3    if (left == right)
4       return a[left];
5    int m = (left + right) / 2;
6    int lm = maxRecursive(a, left, m);
7    int rm = maxRecursive(a, m+1, right);
8    return lm > rm ? lm : rm;
9 }
```

# Rekurze – shrnutí

- užitečná technika jak psát programy,
- založena na rozdělení problému na menší podproblémy,
- z vyřešených dílčích částí skládá celek,
- spíše deklarujeme co chceme řešit, než jak to máme řešit,
- používána pro vyhledávání řešení.

# Děkuji za pozornost