Bubble Sort

```
// BubbleSort, Algoritmus na usporiadanie poľa.
public class BubbleSort
{
  public static void main (String[] args)
  {
    int[] a = new int[args.length];
    for ( int i = 0; i < args.length; i++ )
    { a[i] = Integer.parseInt(args[i]); }
```

Bubble Sort (Pokrač.)

```
boolean sorted;
do
{ sorted = true;
  for ( int i = 0; i < a.length - 1; i++ )
  { if ( a[i] > a[i+1] )
    { int h  = a[i];
      a[i]   = a[i+1];
      a[i+1] = h;
      sorted = false;
  } }
} while ( !sorted );
```

Pozor!

```
a[i] = a[i+1];

a[i+1] = a[i];
```

Nevymení prvky

```
a[i] a a[i+1]
```

Bubble Sort (Pokrač.)

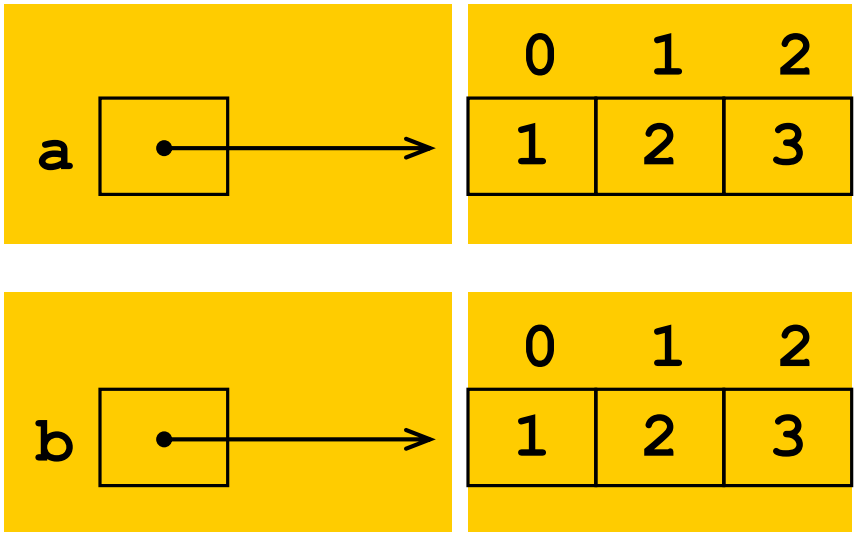
```
    System.out.println(
      "Cisla vo od najmensieho po najvacsie:");
    for ( int i = 0; i < a.length; i++ )
    { System.out.println(a[i]); }
  }
}
```

7.4 Polia a pointery

- S `new` vytvoríme nové pole daného typu
- Pri priradení takto vytvoreného poľa na premennú sa vloží do premennej adresa vytvoreného poľa, teda premenná bude ukazovať na pole.
- Pri priradení poľa na do premennej teda pole nie je kopírované!

Príklad 1

```
int[] a = new int[] {1, 2, 3};
int[] b = new int[] {1, 2, 3};
```

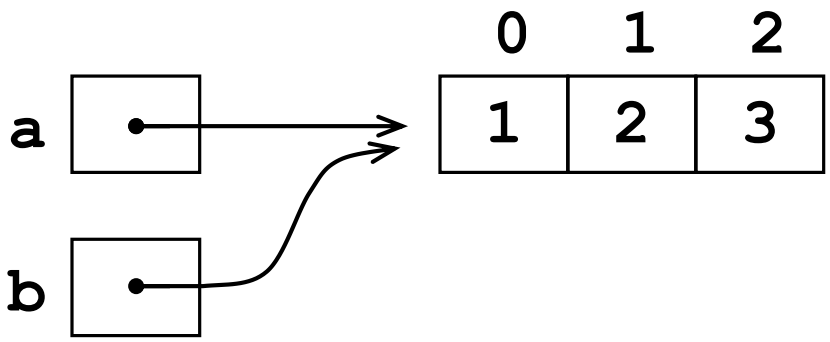


Príklad 2

```
int[] a, b;

a = new int[] {1, 2, 3};

b = a;
```



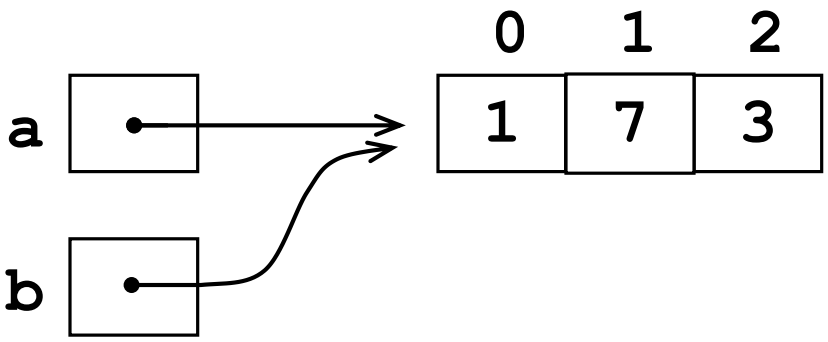
Pozor!

Pole nie je pri priradení kopírované!

Príklad 2 (Pokrač.)

```
a[1] = 7;

// Teraz platí b[1] = 7 !!!
```

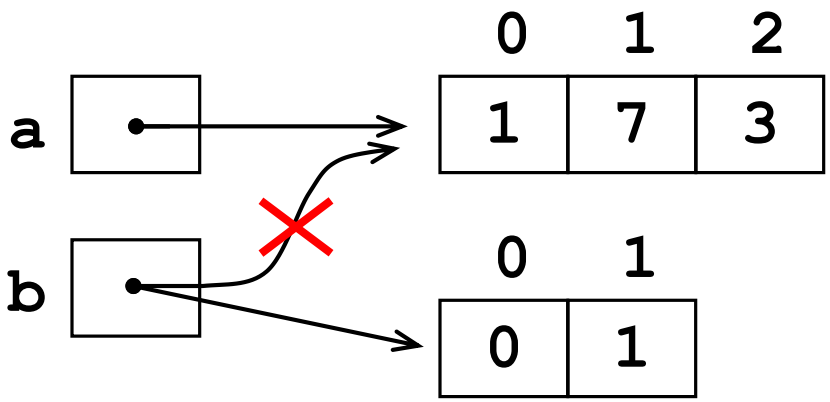


Pozor!

Priradenie hodnoty prvku a[1] zmenilo hodnotu prvku b[1], keďže obidve premenné a aj b ukazujú na to isté pole.

Príklad 2 (Pokrač.)

```
b = new int[] { 0, 1 };
```



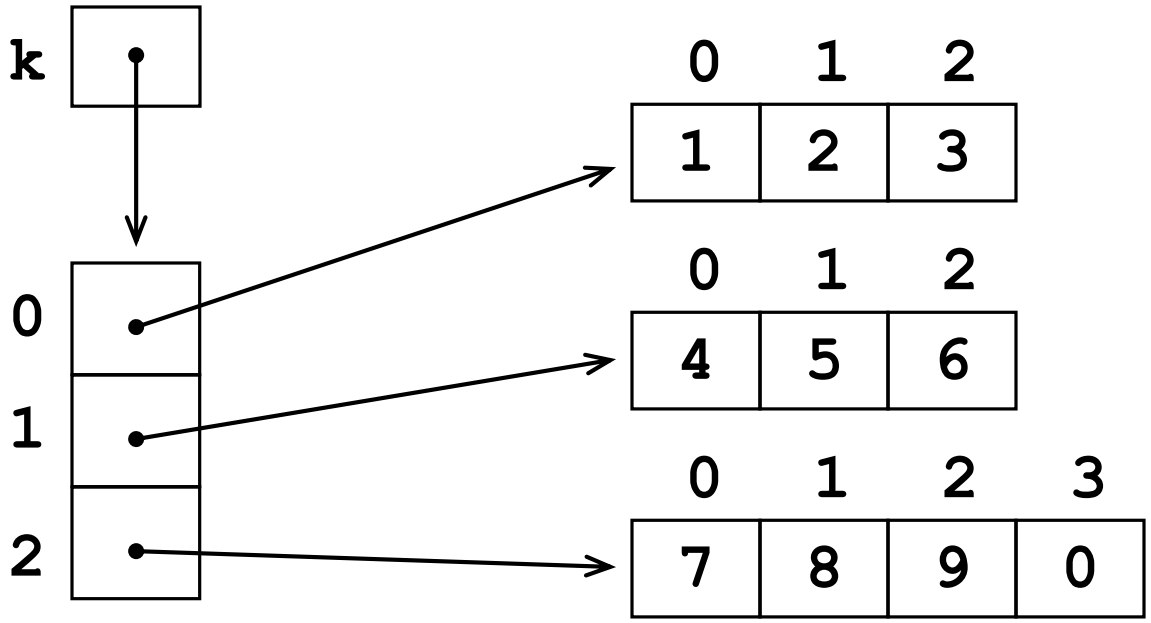
Pozor!

Pri priradení do premennej b sa samozrejme a nemení!

Premenná b bude totiž ukazovať na iné pole ako premenná a.

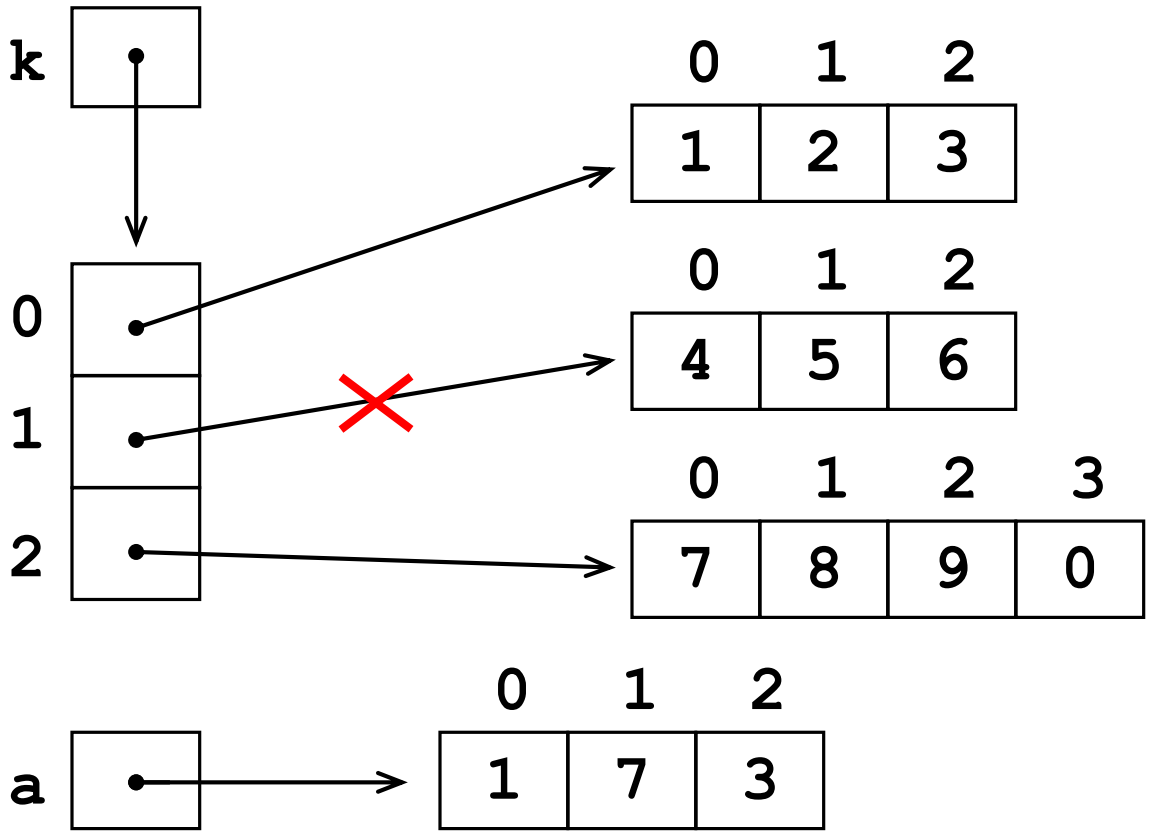
Príklad 3
Viacdimenzionálne polia

```
int[][] k = new int[][]
   { { 1, 2, 3 }, {4, 5, 6}, {7, 8, 9, 0} };
```



Príklad 3 (Pokrač.)

```
k[1] = a;
```



II. Základné elementy

8. Funkcie a procedúry

8.1 Motivácia

- Vo veľkých programoch sa často používa rovnaká časť programu viackrát (napr. výpočet odmocniny alebo usporiadanie poľa, ale aj výpis na obrazovku a pod.)
- **Funkcie** a **procedúry** umožňujú opakované použitie časti programu.

Príklad: Binomialkoeficient

```
public class Binomialkoeficient
{
   static int fak(int n)
   { int vysledok = 1;
     for (int i = 1; i <= n; i++)
     { vysledok = vysledok * i; }

     return vysledok;
   }
```

Definícia funkcie

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Príklad: Binomialkoeficient

```
  public static void main (String[] args)
  { int n = Integer.parseInt(args[0]);
    int k = Integer.parseInt(args[1]);

    int vysledok = fak(n) / ( fak(k) * fak(n - k) );

    System.out.println(„Vysledok: “ + vysledok);
  }
}
```

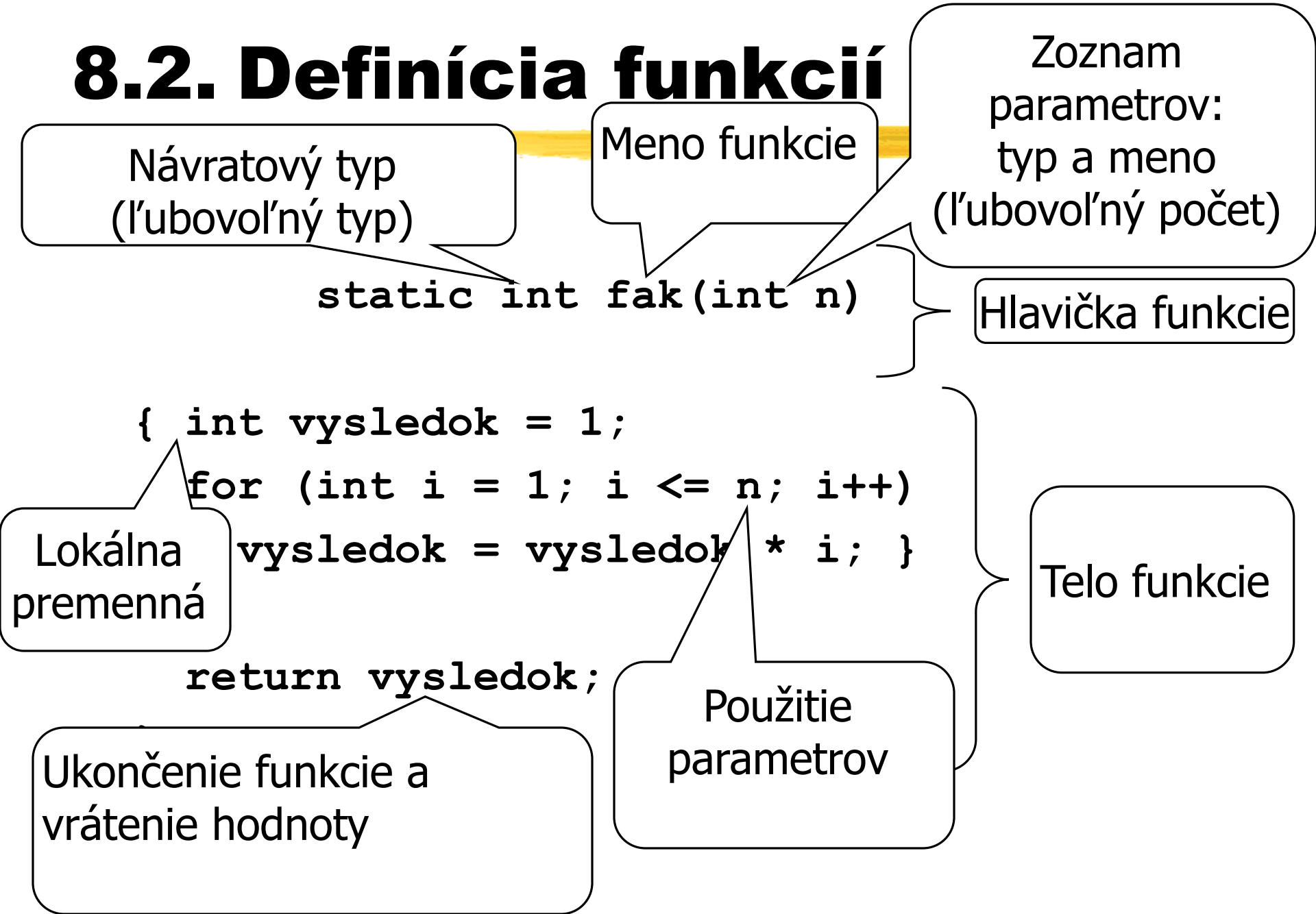
Volania funkcie

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Motivácia

- Vlastné funkcie môžeme používať rovnako ako preddefinované funkcie (napr. **`Math.abs`**, **`Math.sqrt`**, **`Math.max`**, ...).
- Použitím funkcií sa stáva program predľadnejším

8.2. Definícia funkcií



8.2. Definícia funkcií - funkčný prototyp v C

```
static int fak(int n);
```

- Navyše v C: ak funkcia je volaná pred jej definíciou, je potrebné použiť tzv. funkčný prototyp – hlavičku funkcie ukončenú bodkočiarkou

Hlavička funkcie

```
static int fak(int n)
```

- Definuje signatúru (Typy parametrov a typ návratovej hodnoty) funkcie. V našom príklade:

```
fak: int -> int
```

- **Typ návratovej hodnoty** a **typy parametrov** môžu byť ľubovoľné typy (napr. aj polia a pointery)
- Funkcia môže mať ľubovoľný počet parametrov, napr. aj žiadny: `f()`

Telo funkcie

- Telo funkcie je blok príkazov, ktorý sa vykoná pri zavolaní funkcie.
- V tele funkcie je možné použiť parametre funkcie. Parametre sa používajú rovnako ako premenné.
- Pomocou príkazu **`return`** sa ukončí funkcia a hodnota výrazu za kľúčovým slovom **`return`** je vrátená naspäť ako výsledná hodnota funkcie.

`return`-príkaz

- Výraz po kľučovom slove **`return`** musí mať rovnaký typ ako návratový typ funkcie daný v hlavičke.
- Príkaz **`return`** nemusí byť na konci tela funkcie.
- V tele funkcie môže byť príkaz **`return`** použitý viackrát.
- Akonáhle sa vykoná príkaz **`return`** je funkcia ukončená.

8.3 Volanie funkcie (Použitie funkcie)

- Funkcia môže byť použitá vo výrazoch. Pri volaní musí byť každému parametru funkcie (**formálne parametre**) priradený výraz zodpovedajúceho typu. Jeho hodnota sa stáva **aktuálnym parametrom**.
  Príklad: `fak(n-k)` volanie funkcie fak, kde formálny parameter n funkcie fak nadobudne hodnotu výrazu n-k (kde n a k sú premenné funkcie main)
- Zavolaná funkcia má vo výraze svoj návratový typ a hodnotu danú výrazom za príkazom return.

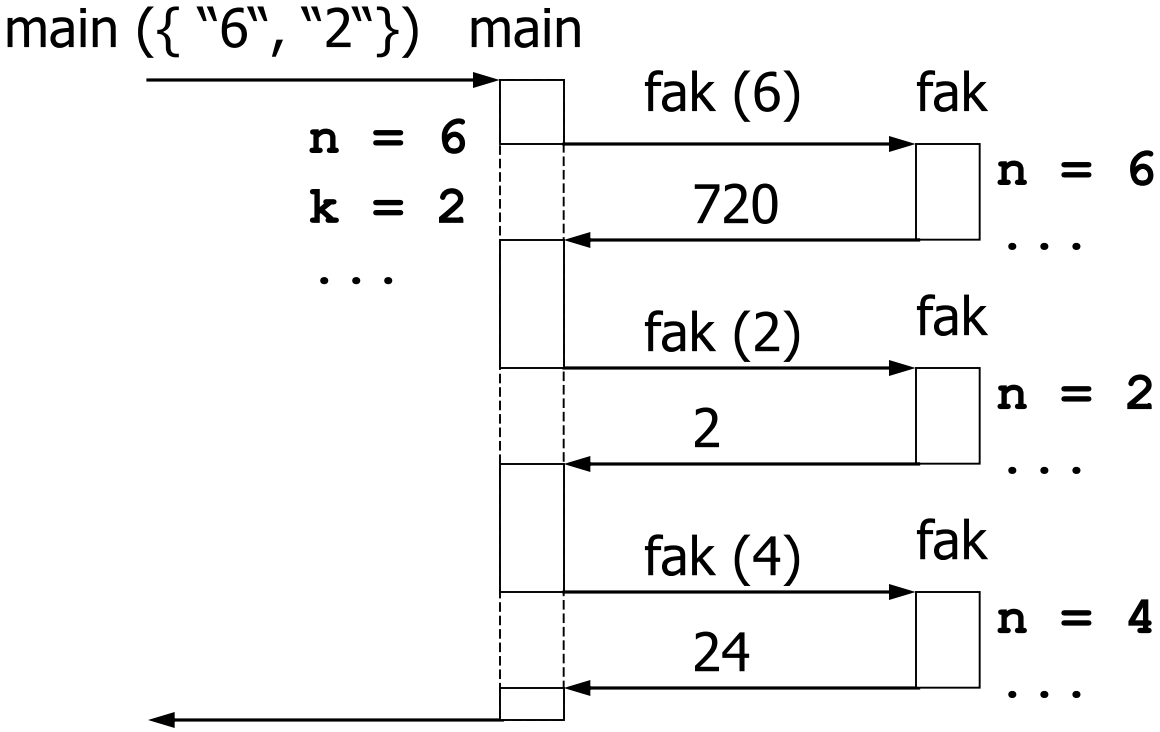
Príklad

```
static int fak(int n)
 { int vysledok = 1;
   for (int i = 1; i <= n; i++)
   { vysledok = vysledok * i; }
   return vysledok; }

 public static void main (String[] args)
 { int n = Integer.parseInt(args[0]);
   int k = Integer.parseInt(args[1]);
   int vysledok = fak(n) / ( fak(k) * fak(n - k) );
   System.out.println(„Vysledok: " + vysledok); }
```

Príklad (Pokrač.)

```
>Binomialkoeficient 6 2
```



8.4 Procedúry

- Procedúry sú funkcie, ktorých návratový typ je prázdny t.j. `void` (z angl. pre prázdny).
- Keďže procedúra nedáva žiadny výsledok, nemôže byť použitá vo výraze.
- Napriek tomu sú procedúry užitočné:
  `System.out.println("Bla");`