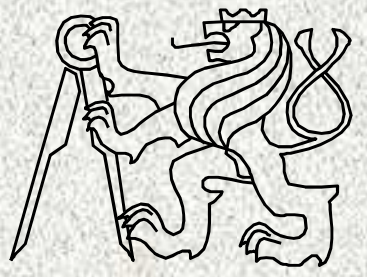
Rekurze versus iterace



A0B36PR1-Programování 1

Fakulta elektrotechnická
České vysoké učení technické

Rekurze

- Definice ze slovníku (pozor vtip)

Rekurze

viz „Rekurze“

- ….nekonečná rekurze, lépe:

pokud neznáte význam tohoto pojmu, pokračujte pojmem „**Rekurze“**

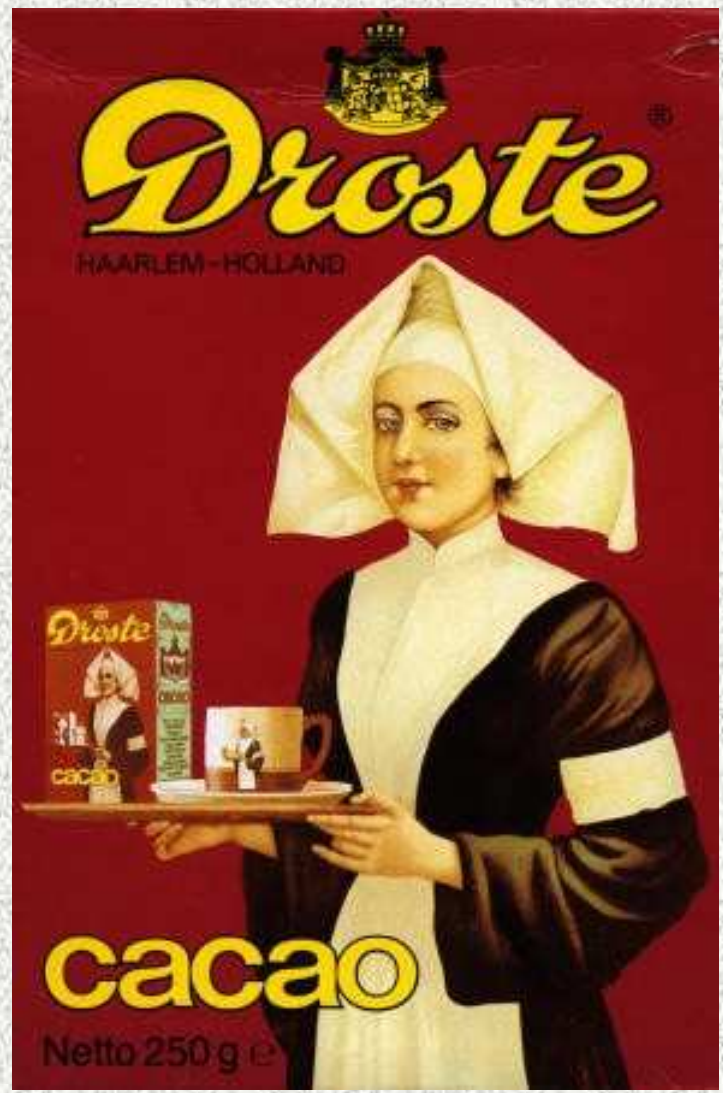
- Rekurze - algoritmus, který volá v průběhu svého běhu sama sebe

Příklad: výpočet faktoriálu: n!

0! = 1,

1! = 1, pro záporné číslo x budiž x! = 1

pro n>1 n! = n(n-1)!



Faktoriál pomocí rekurze a iterace

- ***Rekurze***
  - $n! = 1$ pro $n \leq 1$
  - $n! = n*(n-1)!$ pro $n>1$
- ***Iterace***
  - $n! = n*(n-1)*(n-2)*\ldots*2*1$

```
static int fakt(int n) {
    if (n<=
     return
      }
```

```
 static int fakt(int n) {
) return 1;
turn n*fakt(n-1);
```

```
static int fakt(int n) {
  int f = 1;
  while (n>1){
       f *= n;
       n--;
  }
  return f;
}
```

```
static int
  return n
}
```

```
ní operátor
```

Rekurze a rozklad problému na podproblémy

- Příklad:

Program, který přečte posloupnost čísel zakončenou nulou a vypíše ji obráceně

- Rozklad problému:
  - zavedeme abstraktní příkaz „*obrať posloupnost*“
  - příkaz rozložíme do tří kroků:
    - *přečti číslo*

      (***“a ulož si ho”***)
    - if (*přečtené číslo není nula*) „*obrať posloupnost*“

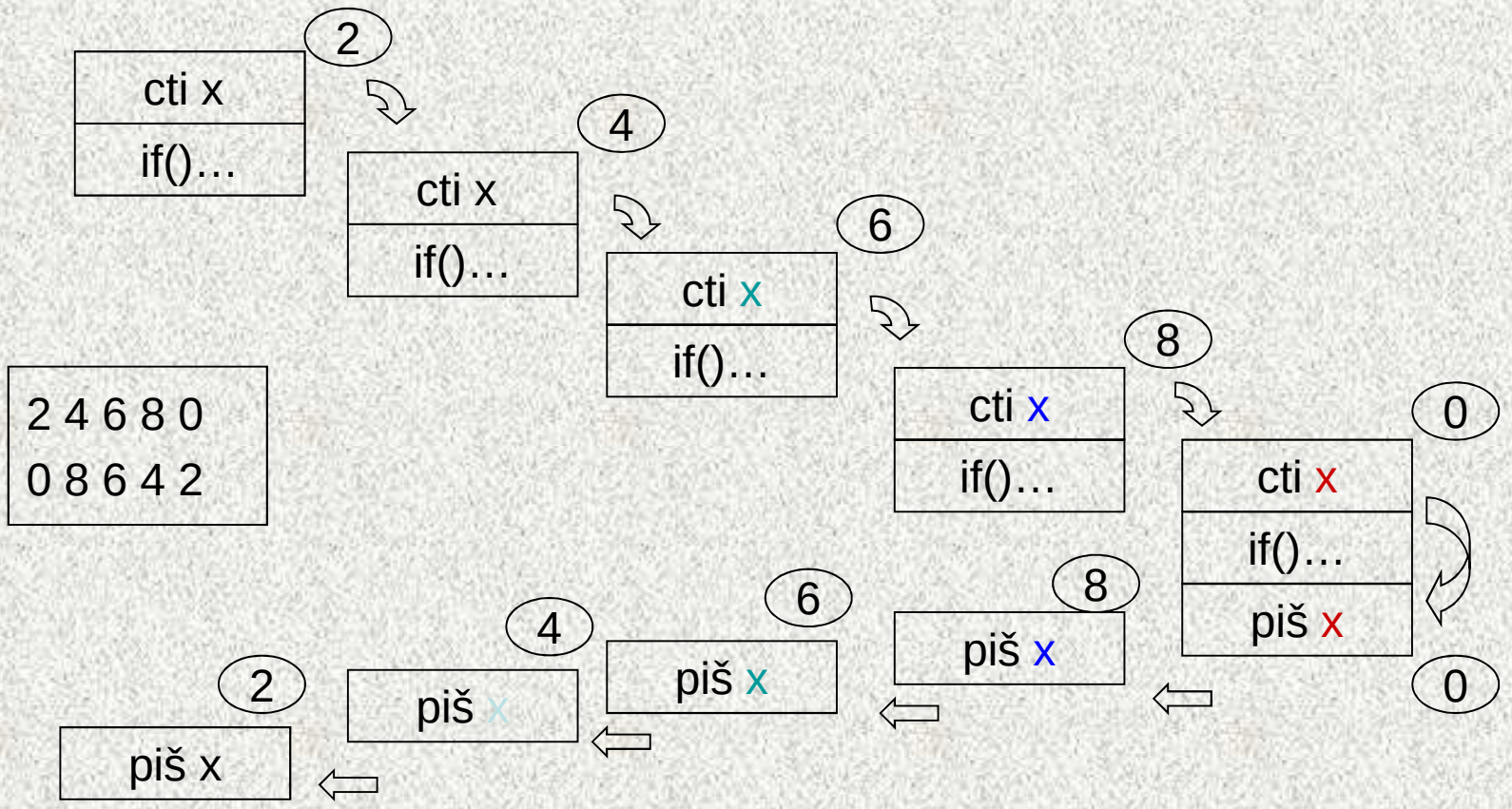
      (***“zbytek!!”***)
    - *vypiš číslo*

      (***“uložené”***)

Příklad rekurze „*Obrať posloupnost*“

„*obrať posloupnost*“

- *přečti číslo*
- if (*přečtené číslo není nula*) „*obrať posloupnost, tj. zbytek*“
- *vypiš číslo*



Příklad rekurze - obrat()

- Řešení:

```
public class Obrat {
   static Scanner sc = new Scanner(System.in);
   public static void main(String[] args) {
    System.out.println("zadejte …… zakončenou nulou");
    obrat();
   }

 static void obrat() {
    int x = sc.nextInt();              // načtení
    if (x!=0) obrat();                 // otočení zbytku
    System.out.print(x + " ");         // výpis uloženého
   }
}
```

Příklad rekurze - Hanojské věže

- Úkol: přemístit disky na druhou jehlu s použitím třetí pomocné jehly, přičemž musíme dodržovat tato pravidla:
  - v každém kroku můžeme přemístit pouze jeden disk, a to vždy z jehly na jehlu (disky nelze odkládat mimo jehly),
  - není možné položit větší disk na menší.

Příklad rekurze - Hanojské věže

- Zavedeme abstraktní příkaz: *přenes_věž*(*n*,*1*,*2*,*3*), který interpretujeme jako "přenes *n* disků z jehly *1* na jehlu *2* s použitím jehly *3*".
- Pro *n*>0 lze příkaz rozložit na tři jednodušší příkazy
  - *přenes_věž*(*n*-1,*1*,*3*,*2*)
  - "*přenes disk z jehly 1 na jehlu 2*",
  - *přenes_věž*(*n*-1,*3*,*2*,*1*)

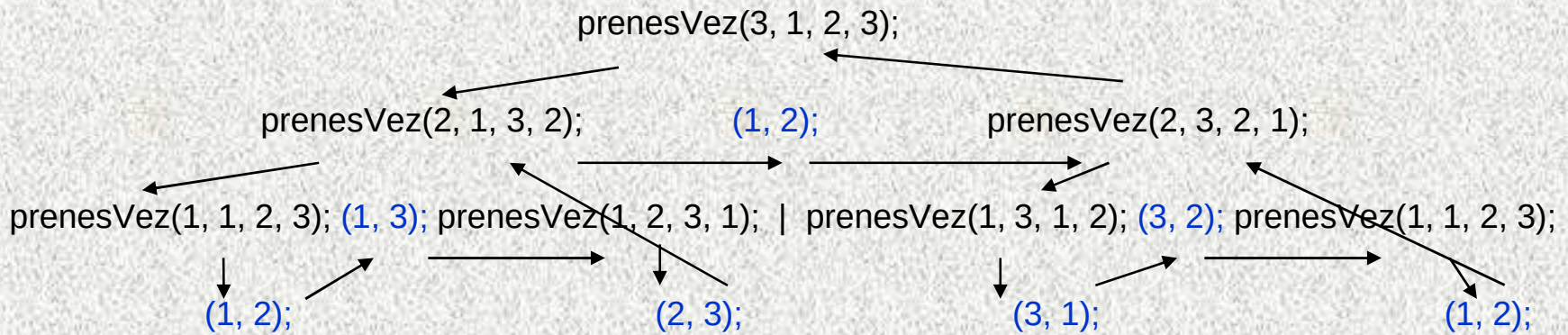
Příklad rekurze - Hanojské věže

```
public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);
   System.out.println("zadejte výšku věže");
   int pocetDisku = sc.nextInt();
   prenesVez(pocetDisku, 1, 2, 3);
}
```

```
3
přenes disk z 1 na 2
přenes disk z 1 na 3
přenes disk z 2 na 3
přenes disk z 1 na 2
přenes disk z 3 na 1
přenes disk z 3 na 2
přenes disk z 1 na 2
```

```
static void prenesVez(int vyska,int odkud,int kam,int pomoci)
   {
if (vyska>0) {
 prenesVez(vyska-1, odkud, pomoci, kam);
 System.out.println("přenes disk z "+odkud+" na "+kam);
 prenesVez(vyska-1, pomoci, kam, odkud);
   }
}
```

Příklad rekurze - Hanojské věže




```
prenesVez(int vyska, int odkud, int kam, int pomoci) {
     if (vyska>0) {
      prenesVez(vyska-1, odkud, pomoci, kam);
      System.out.println("přenes disk z "+odkud+" na "+kam);
      prenesVez(vyska-1, pomoci, kam, odkud);
       }
     }
   }
```

```
3
přenes disk z 1 na 2
přenes disk z 1 na 3
přenes disk z 2 na 3
přenes disk z 1 na 2
přenes disk z 3 na 1
přenes disk z 3 na 2
přenes disk z 1 na 2
```

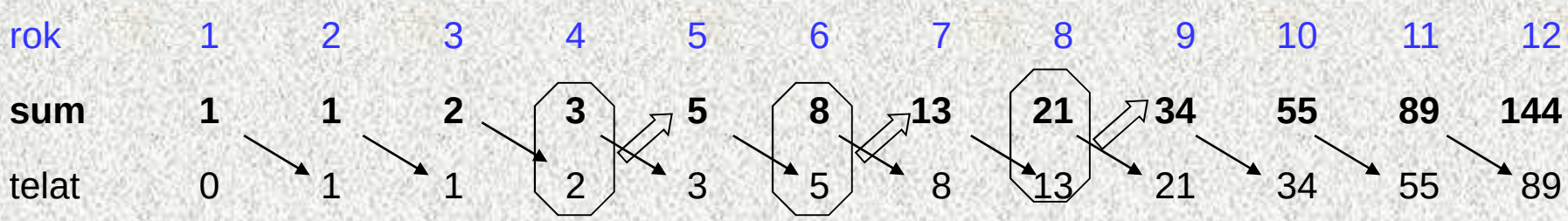
Obecně k rekurzivitě

- Rekurzivní funkce (procedury) jsou přímou realizací rekurzivních algoritmů
- Rekurzivní algoritmus předepisuje výpočet „shora dolů“ v závislosti na velikosti (složitosti) vstupních dat:
    - pro nejmenší (nejjednodušší) data je výpočet předepsán přímo
    - pro obecná data je výpočet předepsán s využitím téhož algoritmu pro menší (jednodušší) data
- Výhodou rekurzivních funkcí (procedur) je jednoduchost a přehlednost
- Nevýhodou může být časová náročnost způsobená např. zbytečným opakováním výpočtu
- Řadu rekurzívních algoritmů lze nahradit iteračními, které počítají výsledek „zdola nahoru“, tj, od menších (jednodušších) dat k větším (složitějším)
- Pokud algoritmus výpočtu „zdola nahoru“ nenajdeme (např. při řešení problému Hanojských věží), lze rekurzivitu odstranit pomocí tzv. zásobníku

Fibonacciho posloupnost - historie

- *Maatraameru* (Chhandah-shāstra, the Art of Prosody, 450 or 200 BC)
- **Leonardo Pisano** (Leonardo z Pisy), známý také jako Fibonacci (cca 1175–1250) - králíci
- Henry E. Dudeney (1857 - 1930) - krávy
- „Jestliže každá kráva vyprodukuje své první tele (jalovici) za rok a poté každý rok jednu další jalovici, kolik budete mít krav za 12 let, jestliže Vám žádná nezemře? Na počátku budete mít jednu krávu“
- Po 12 let je k dispozici jeden či více býků

Fibonacciho posloupnost - historie



- 50 20 365 011 074 (20 miliard)

„Jestliže každá kráva vyprodukuje své první tele (jalovici) za rok a poté každý rok jednu další jalovici, kolik budete mít krav za 12 let, jestliže Vám žádná nezemře? Na počátku budete mít jednu krávu“ (a jednoho býka!!)

Počet krav = počet krav vloni +
počet narozených (odpovídá počtu krav předloni)

$$f_n = f_{n-1} + f_{n-2}$$

Fibonacciho posloupnost - rekurzivně

- Platí:

$f_0 = 1$
$f_1 = 1$
$f_n = f_{n-1} + f_{n-2}$ pro $n > 1$

Rekurzivní funkce:

```
static int fib(int i) {
    if (i<2) return 1;
    return fib(i-1)+fib(i-2);
}
```

Rekurze je hezká - zápis „odpovídá“ rekurentní definici. Je ale i efektivní?

Složitost výpočtu Fibonacciho čísla - rekurzivně

Příklad pro fib(10):

f_{50} 20 365 011 074 (20 miliard)

Složitost je exponenciální!!!!

```
static int fib(int i) {
  if (i<2) return 1;
  return fib(i-1)+fib(i-2);
 }
```

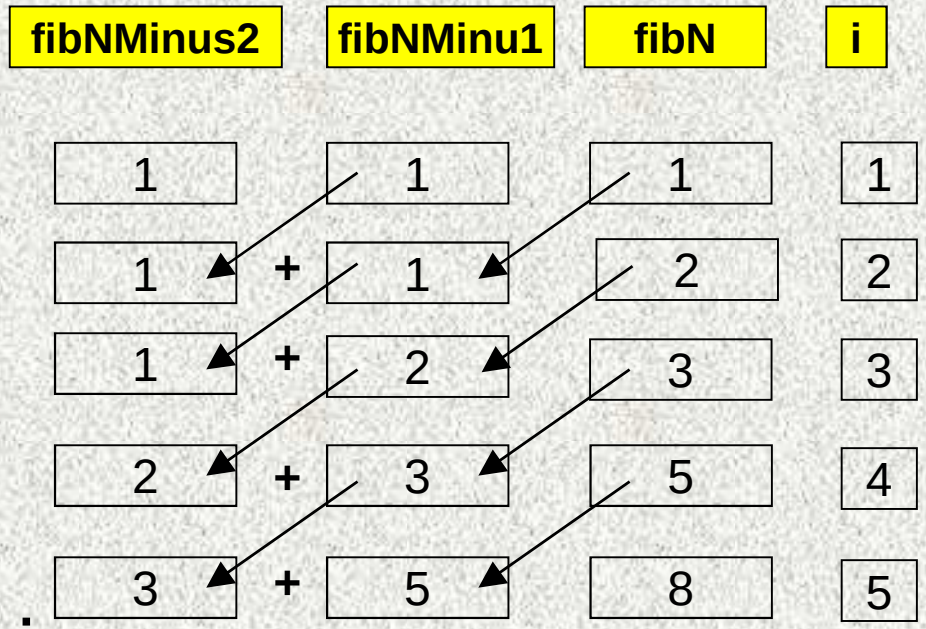
Fibonacciho posloupnost - iteračně

- Platí:

$f_0 = 1$

$f_1 = 1$

$f_n = f_{n-1} + f_{n-2}$,pro $n > 1$



Iteračně:

```
static int fib(int n) {
   int i,  fibNMinus2=1;
    int  fibNMinus1=1, fibN=1;
      for (i=2; i<=n; i++) {
       fibNMinus2 = fibNMinus1;
       fibNMinus1 = fibN;
       fibN = fibNMinus1 + fibNMinus2;
 }
 return fibN;
}
```

Složitost:

3*n

Složitost výpočtu Fibonacciho čísla 2

- **Iterační metoda: 3*n**
- **Rekurzivní výpočet ~ 2^n**
- **Podíl dvou po sobě následujících členů konverguje k hodnotě „zlatého řezu“** (Johannes Kepler - golden ratio):

  φ ≈1,618033988749894848204586834365 6

$$\varphi = \frac{1+\sqrt{5}}{2}, \qquad F(n) = \frac{\varphi^n}{\sqrt{5}} - \frac{(1-\varphi)^n}{\sqrt{5}}$$

Zlatý řez byl pokládán za středověku za ideální proporci
mezi různými délkami
Zlatý řez vznikne rozdělením úsečky na dvě části tak,
že poměr větší části k menší je stejný jako poměr celé úsečky k větší části

Rekurzivní algoritmus – NSD()

- Platí: je-li x, y > 0, pak nsd(x, y):

  je-li $x = y$, pak $nsd(x,y) = x$

  je-li $x > y$, pak $nsd(x,y) = nsd(x\%y,y)$

  je-li $x < y$, pak $nsd(x,y) = nsd(x,y\%x)$

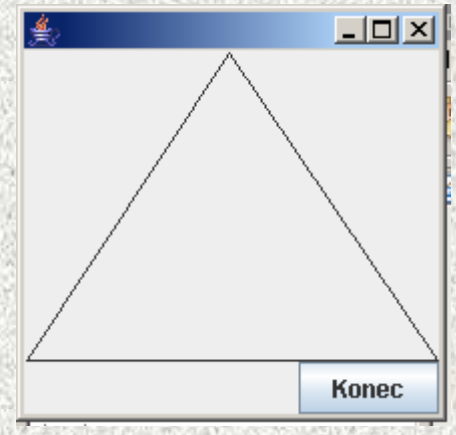
```
static int nsd(int x, int y) {
    if (x==y) return x;
    else if (x>y) return nsd(x%y, y);
              else return nsd(x, y%x);
  }
```

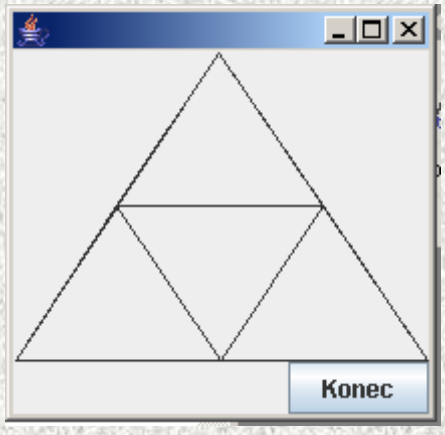
Rekurze

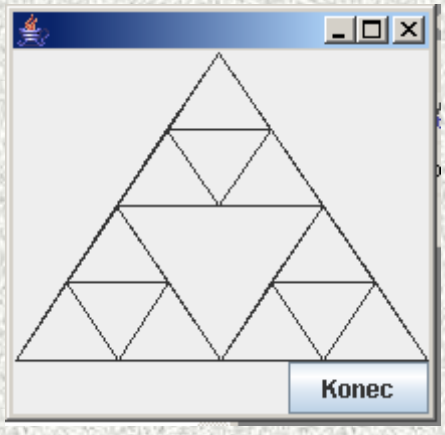
```
static int nsd(int x, int y) {
  int zbytek;
  while( y != 0 ) {
    zbytek = x % y; x = y; y = zbytek;
  }
  return x;
}
```

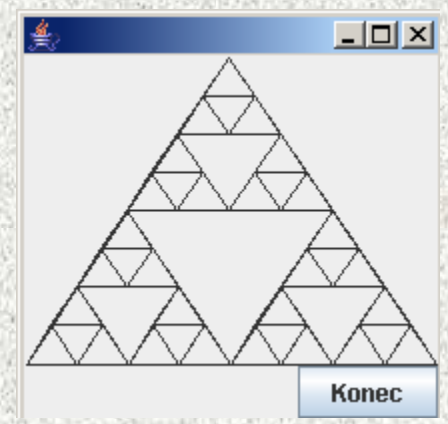
Iterace

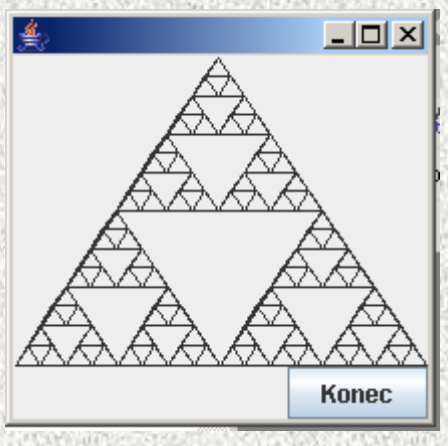
Příklad rekurze - fraktály

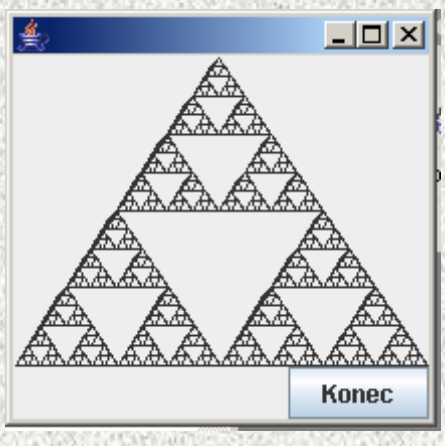












Příklad rekurze, základní schéma – součin

```
public static void main(String[] args) {
    int x,y;
    ……;
    System.out.println(" " + souI(x, y) + souR(x,y));
}
static int souI(int s, int t){
    int souI=0;
    for (int i = 0; i < s; i++)
        souI=souI+t;
    return souI;
  }
```

```
static int souR(int s, int t) {
int souR;
if (s > 0)souR=souR(s - 1,t)+t;
      else souR = 0;
    return souR;}
```

Rozklad na prvočinitele

- Rozklad přirozeného čísla n na součin prvočísel
- Řešení:
  - dělit 2, pak 3, atd. , a dalšími prvočísly, … n-1
  - každé dělení beze zbytku dodá jednoho prvočinitele

Příklad:

60/2=>30/2=>15/3=>5/5

60 má prvočinitele 2, 2, 3, 5

Rozklad na prvočinitele - iterací

```
public class PrvociniteleIter {
  static int rozklad(int x, int d) {
      while (d < x && x % d != 0) d++;
      System.out.print(d + " ");
      return d;
  }
  public static void main(String[] args) {
    System.out.print("zadejte přirozené číslo: ");
    int x = (new Scanner(System.in)).nextInt();
    if (x < 1) {
      System.out.print ("číslo není přirozené");
      System.exit(0);
    }
    int d = 2;
    while (d < x) {
      d = rozklad(x, d);
      x = x/d;
    }
  }}
```

```
zadejte přirozené číslo: 144
2 2 2 2 3
```

Rozklad na prvočinitele - rekurzí

```
public class Prvocinitele {
  static void rozklad(int x, int d) {
    if (d < x) {
      while (d < x && x % d != 0) d++;
      System.out.print(d + " ");
      rozklad(x / d, d);
    } }

  public static void main(String[] args) {
   System.out.print("zadejte přirozené číslo: ");
     int x = (new Scanner(System.in)).nextInt();
    if (x < 1) {
      System.out.println("číslo není přirozené");
      System.exit(0);
    }
    rozklad(x, 2);
  }}
```

```
zadejte přirozené číslo: 144
2 2 2 2 3
```

Iterační alg. – NSD(), připomenutí

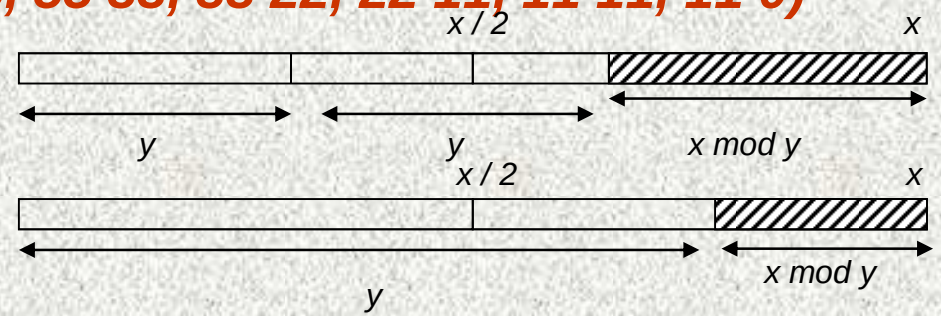
```
static int nsd(int x, int y) {
    int zbytek;
    while( y != 0 ) {
        zbytek = x % y; x = y; y = zbytek;
    }
    return x;
}
```

Kolikrát se provede tělo cyklu while ?

Platí: ***Je-li x ≥ y (> 0), pak x mod y < x/2***

(55 88, 88 55, 55 33, 33 22, 22 11, 11 11, 11 0)

- Důkaz:
  - buď je y ≤ x/2
  - nebo je y > x/2



- Nechť *n* je počáteční hodnota *y*. Každé dva průchody cyklem se *y* zmenší na polovinu, takže na hodnotu 0 dospěje nejpozději po

  $2.\log_2(n)$ průchodech.

K zamyšlení