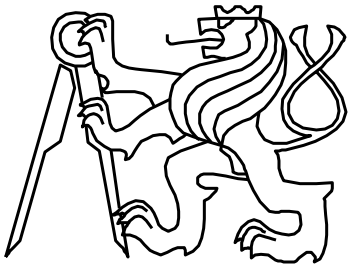
*Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.*
*Praha & EU: Investujeme do vaší budoucnosti*

# Rozklad problému na podproblémy, rekurze



**BI-PA1 Programování a algoritmizace 1, ZS 2012-2013**
**Katedra teoretické informatiky**
© Miroslav Balík
Fakulta informačních technologií
České vysoké učení technické

# Rozklad problému na podproblémy

- Postupný návrh programu rozkladem problému na podproblémy
  - zadaný problém rozložíme na podproblémy
  - pro řešení podproblémů zavedeme abstraktní příkazy
  - s pomocí abstraktních příkazů sestavíme hrubé řešení
  - abstraktní příkazy realizujeme pomocí metod
- Rozklad problému na podproblémy ilustrujme na příkladu hry NIM
- Pravidla:
  - hráč zadá počet zápalek ( např. od 15 do 35 )
  - pak se střídá se strojem v odebírání; odebrat lze 1, 2 nebo 3 zápalky,
  - prohraje ten, kdo odebere poslední zápalku.
- Dílčí podproblémy:
  - zadání počtu zápalek
  - odebrání zápalek hráčem
  - odebrání zápalek strojem

# Hra NIM

- Hrubé řešení:

```
int pocet;
bool stroj = false;
"zadání počtu zápalek"
do {
        if (stroj) "odebrání zápalek strojem"
        else "odebrání zápalek hráčem"
        stroj = !stroj;
} while (pocet > 0);
if (stroj) "vyhrál stroj"
else "vyhrál hráč"
```

v hrubém řešení použijeme typ *bool*, jehož hodnotami jsou *true* a *false*

- Podproblémy „zadání počtu zápalek“, „odebrání zápalek strojem“ a „odebrání zápalek hráčem“ budeme realizovat procedurami, proměnné *pocet* a *stroj* budou globálními (pro procedury nelokálními) proměnnými

# Hra NIM

```
/* prog6-nim.c */
#include <stdio.h>
#include <stdlib.h>

#define bool char
#define true 1
#define false 0

int pocet;
bool stroj = false; /* zacina hrac */
void zadaniPoctu(void) {
do {
    printf("zadejte pocet zapalek (od 15 do 35): ");
    scanf("%d", &pocet);
} while (pocet<15 || pocet>35); }

void bereHrac(void) {
    int x; bool chyba;
    do {
      chyba = false;
      printf("pocet zapalek je %d\n", pocet);
      printf("kolik odeberete: "); scanf("%d", &x);
```

co je typ *bool ?*

co je to *false ?*

Vysvětlení na dalším slajdu

# Hra NIM

```
      if (x<1) { printf("prilis malo\n"); chyba = true; }
      else if (x>3 || x>pocet) { printf("prilis mnoho\n"); chyba = true; }
    } while (chyba); pocet -= x;
}

void bereStroj(void) {
    printf("pocet zapalek je %d\n", pocet);
    int x = (pocet-1)%4;
    if (x==0) x = 1;
    printf("odebiram %d\n", x);
    pocet -= x;
}

int main(void) {
    zadaniPoctu();
    do {
      if (stroj) bereStroj();
      else bereHrac();
    stroj = !stroj;
    } while (pocet>0);
    if (stroj) printf("vyhral jsem\n");
    else printf("vyhral jste, gratuluji\n");
    system("PAUSE");
    return 0;
}
```

# Direktiva *define*

```
/* prog6-nim.c */

#define   bool   char
#define   true   1
#define   false   0

int pocet;
bool   stroj   =   false;
```


takže překladač zpracuje deklaraci proměnné *stroj* takto

```
char stroj = 0;
```

# Hra NIM

- Pravidla pro odebírání zápalek strojem, která vedou k vítězství (je-li to možné):
  - počet zápalek nevýhodných pro protihráče je 1, 5, 9, atd., obecně $4n+1$, kde $n >= 0$,
  - stroj musí z počtu $p$ zápalek odebrat $x$ zápalek tak, aby platilo $p - x = 4n + 1$
  - z tohoto vztahu po úpravě a s ohledem na omezení pro $x$ dostaneme $x = (p - 1) \% 4$
  - vyjde-li $x=0$, znamená to, že okamžitý počet zápalek je pro stroj nevýhodný a bude-li protihráč postupovat správně, stroj prohraje.

```
void bereStroj(void) {
    printf("pocet zapalek je %d\n", pocet);
    int x = (pocet-1)%4;
    if (x==0) x = 1;
    printf("odebiram %d\n", x);
    pocet -= x;
}
```

# Rekurze

- Definice ze slovníku (pozor vtip)

**Rekurze**

**viz „Rekurze“**

- ….nekonečná rekurze, lépe:

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

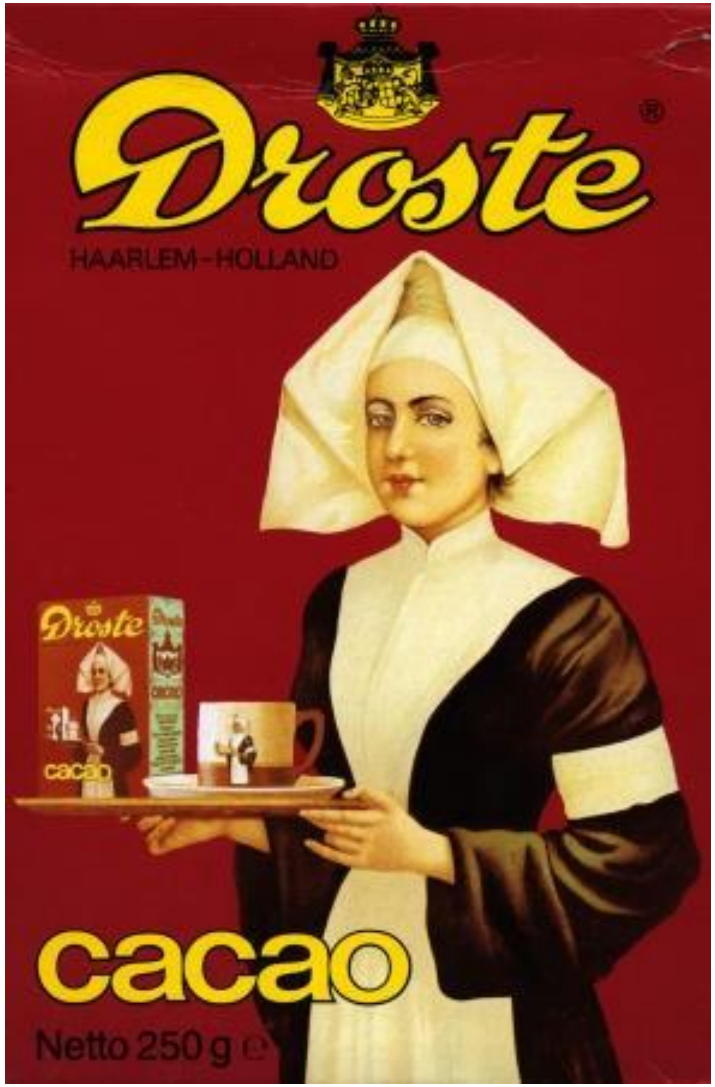
- Rekurze - algoritmus, který volá v průběhu svého běhu sama sebe

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

- $n! = 1$ pro $n \leq 1$
- $n! = n*(n-1)!$ pro $n>1$

```
int fakt(int n)  {
    if (n<=1) return
    return n * fakt(
}
```

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

```
1;
fakt(n-1);
```

```
int fakt(int n) {
    return  n<=1?1:n * fakt(n-1);   // ternární operátor
}
```

# Iterační alg. – NSD(), připomenutí

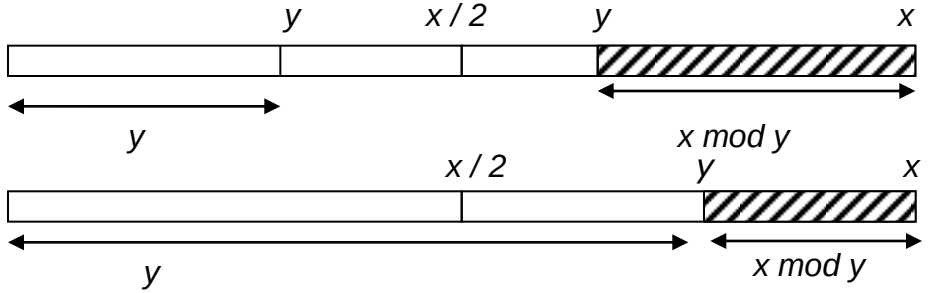
```
int nsd(int x, int y) {
    int zbytek;
    while ( y != 0 ) {
        zbytek = x % y; x = y; y = zbytek;
    }
    return x;
}
```

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.

# Rekurzivní algoritmus – NSD() I

- 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 $nsd(x,y) = nsd(x,y\%x)$

```
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**

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

**Iterace**

# Rekurze a rozklad problému na podproblémy

- Příklad:

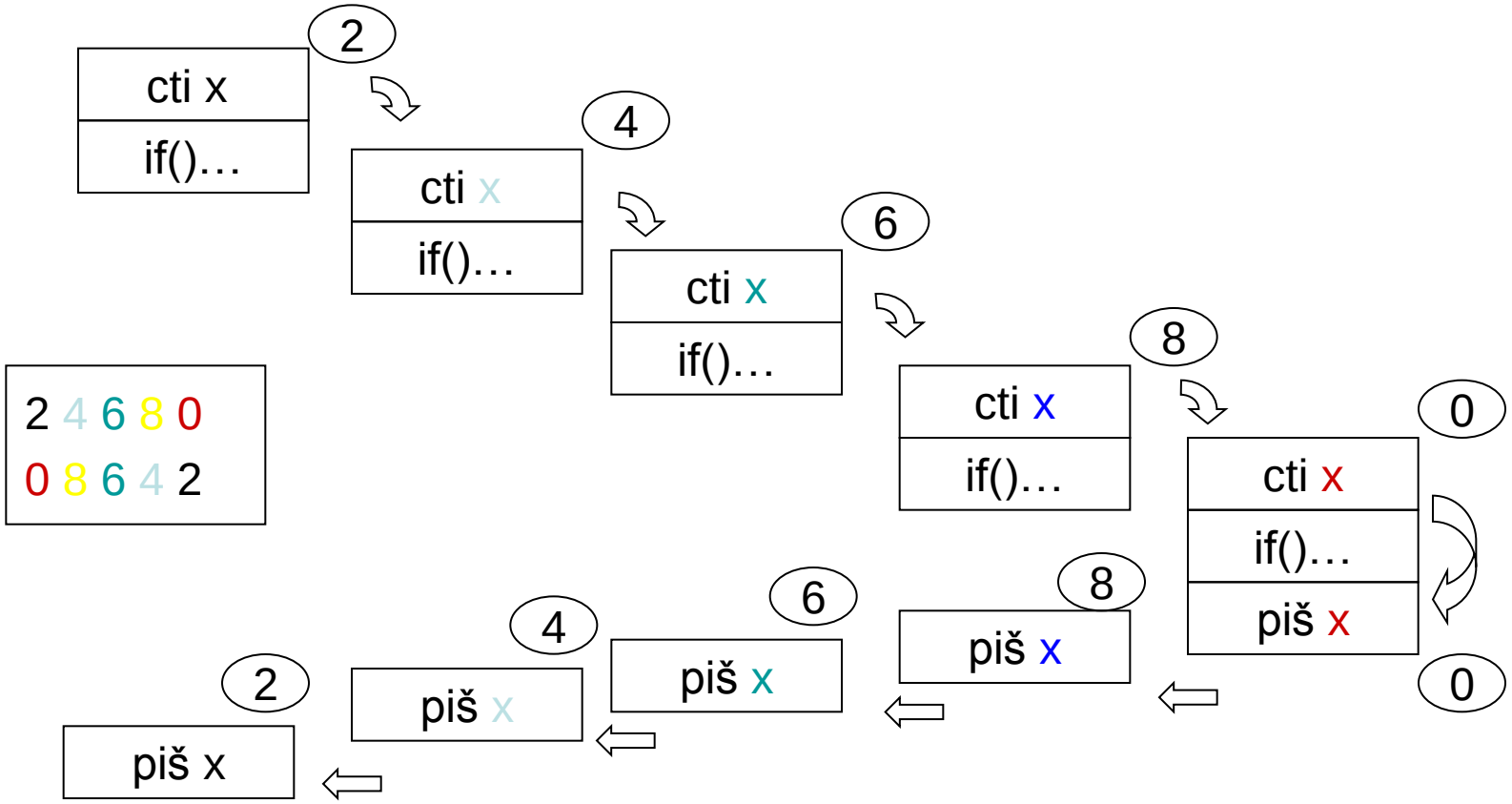
Program, který přečte posloupnost čísel zakončenou nulou a vypíše ji obráceně

- 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“*

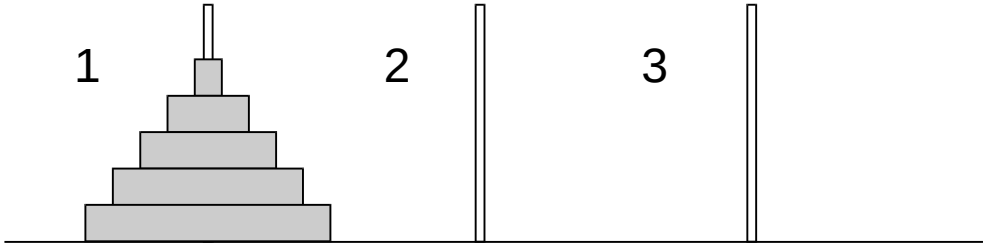
- *přečti číslo*
- if (*přečtené číslo není nula*) *„obrať posloupnost, tj. zbytek“*
- *vypiš číslo*



# Příklad rekurze - obrat()

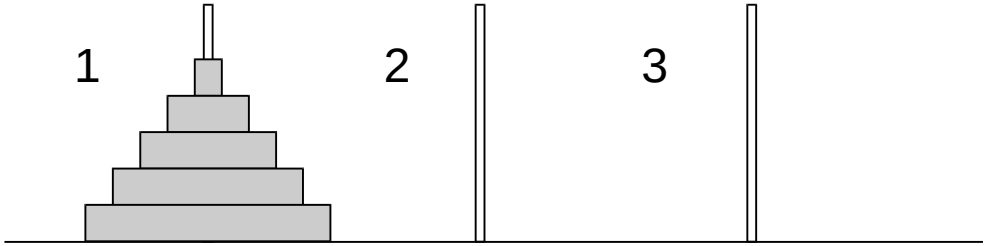
```
/* prog6-obrat.c */
#include <stdio.h>
#include <stdlib.h>

void obrat(void) {
    int x;
    scanf("%d", &x);
    if (x!=0) obrat();
    printf("%d ", x);
}

int main(void) {
    printf("zadejte posloupnost celych cisel zakoncenou nulou\n");
    obrat();
    printf("\n");
return 0;
}
```

# 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

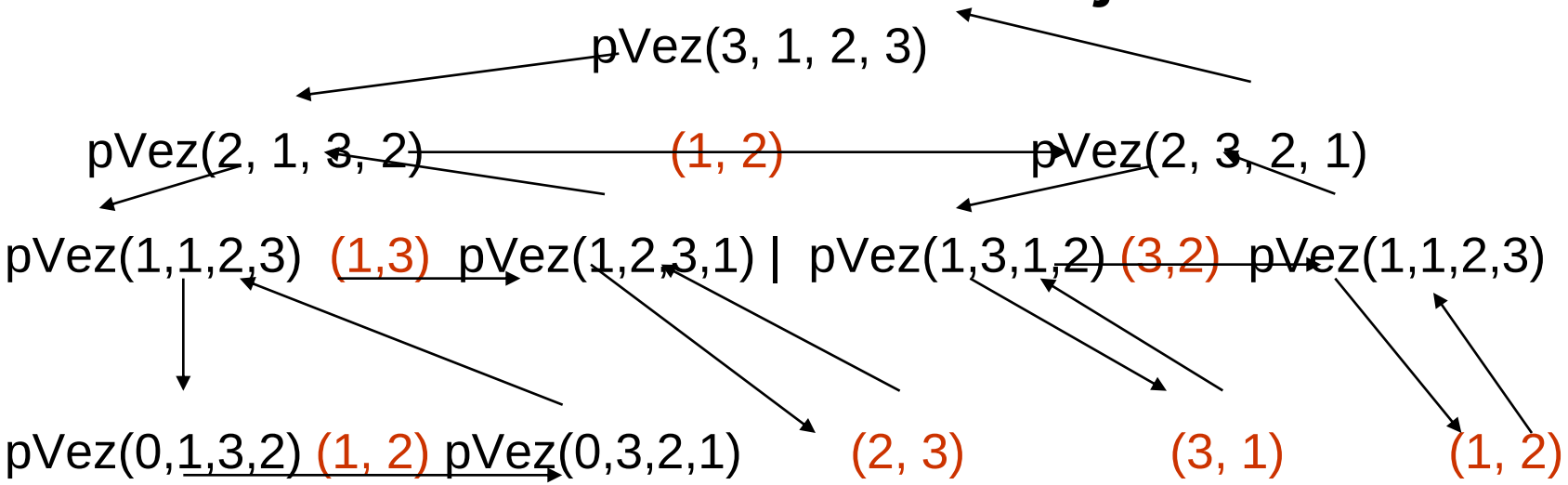



- Zavedeme abstraktní příkaz: *přenes_věž*(*n*,*1*,*2*,*3*), který interpretujeme jako "přenes *n* disků z jehly *1* na jehlu *2* s použitím jehly *3*".
- Pro *n*>0 lze příkaz rozložit na tři jednodušší příkazy
  - *přenes_věž*(*n*-1,*1*,*3*,*2*)
  - "*přenes disk z jehly 1 na jehlu 2*",
  - *přenes_věž*(*n*-1,*3*,*2*,*1*)

# Hanojské věže

```
/* prog6-hanoj.c */ /* hanojske veze */
#include <stdio.h>
#include <stdlib.h>
void prenesVez(int vyska, int odkud, int kam, int pomoci) {
    if (vyska>0) {
        prenesVez(vyska-1, odkud, pomoci, kam);
        printf("prenes disk z %d na %d\n", odkud, kam);
        prenesVez(vyska-1, pomoci, kam, odkud);
    }
}
int main(void) {
    int pocetDisku;
    printf("zadejte pocet disku: ");
    scanf("%d", &pocetDisku);
    if (pocetDisku<=0) printf("pocet disku musi byt kladne cislo\n");
    else prenesVez(pocetDisku, 1, 2, 3);
return 0;
}
```

# Příklad rekurze - Hanojské věže




```
void prenesVez(int vyska, int odkud, int kam, int pomoci) {
    if (vyska>0) {
        prenesVez(vyska-1, odkud, pomoci, kam);
        printf("prenes disk z %d na %d\n", odkud, 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

- Pingala (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) ve věku dvou let a poté každý rok jednu další jalovici, kolik budete mít krav za 12 let, jestliže Vám žádná nezemře?“
- rok počet krav (jalovic)
- 1 1
- 2 1
- 3 2
- 4 3
- 5 5
- 6 8

- …..
- 12 144
- …..
- 50 20 365 011 074 (20 miliard)

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

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

# Fibonacciho posloupnost

- Platí:

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

Rekurzivní funkce:

```
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?

# Fibonacciho posloupnost - iteračně

- Platí:

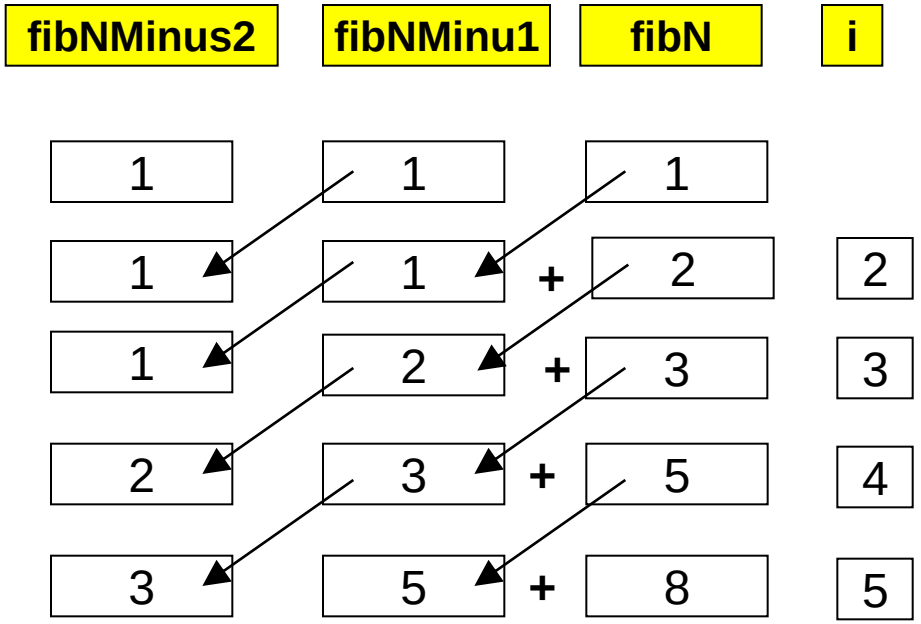
$f_0 = 1$

$f_1 = 1$

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

Iteračně:

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



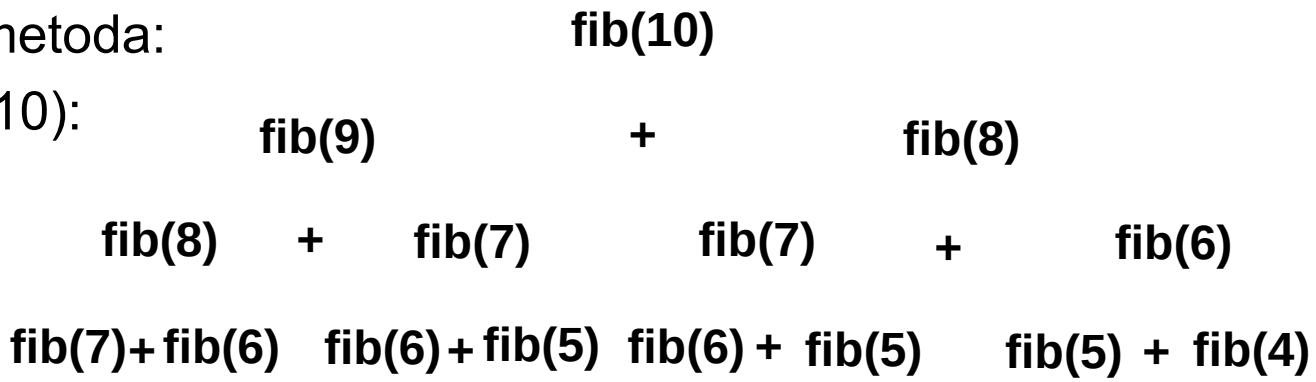

Složitost:

3*n

# Složitost výpočtu Fibonacciho čísla

- Iterační metoda: 3*n
- Rekurzivní metoda:

příklad pro fib(10):




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

**Složitost je exponenciální!!!!**

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

# Složitost výpočtu Fibonacciho čísla 2

- **Iterační metoda: 3*n**
- **Rekurzivní výpočet ~ $2^n$**

- **Přímý výpočet** (Johannes Kepler)
- zlatý řez (golden ratio)
  ≈1,61803398874989484820458683436563

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

# Příklad rekurze, základní schéma – součin

```
void main ( void )  {
    int x, y;
    ……;
    printf (”%d %d”, soul(x, y), souR(x,y));
}

int soul ( int s, int t ) {
    int i, sI=0;
    for (i = 0; i < s; i++)
        sI = sI + t;
        return sI;
  }
```

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



pro zájemce

# 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



pro zájemce

# Rozklad na prvočinitele - iterací

```
int rozklad (int x, int d) {
    while (d < x && x % d != 0) d++;
    printf ("%d \n",d);
    return d;
}
void main (void ) {
    int x;
    int d = 2;
    printf (“zadejte prirozene cislo:\n");
    scanf ("%d ",&x);
    if (x < 1) {
        printf (“cislo neni prirozene\n");
        return;
    }
    while (d < x) {
        d = rozklad(x, d);
        x = x/d;
    }
}
```

zadejte přirozené číslo: 144
2 2 2 2 3



pro zájemce

# Rozklad na prvočinitele - rekurzí

```
void rozklad (int x, int d ) {
    if (d < x) {
       while (d < x && x % d != 0) d++;
       printf ("%d ",d);
       rozklad (x / d, d);
    }
}
void main(void) {
    int x;
    printf (“zadejte prirozene cislo:\n");
    scanf ("%d ",&x)
    if (x < 1) {
       printf (“cislo neni prirozene\n");
       return;
    }
    rozklad(x, 2);
}
```

zadejte přirozené číslo: 144
2 2 2 2 3



pro zájemce