Rekurzivní programování

Rekurze a rekurzivní podprogram

- **Rekurze**
  - znamená definici objektu nebo procesu pomocí jeho samého

- **Rekurzivní podprogram**
  - znamená to, že podprogram volá sám sebe
  - existují dva typy rekurze
    - **rekurze přímá** – podprogram skutečně volá sám sebe
    - **rekurze nepřímá** – podprogram **A** volá podprogram **B** a ten volá zase **A**
  - v jednom okamžiku je rozpracován více než jeden exemplář stejného podprogramu

Pravidla rekurze

- Rekurzivní konstrukce vedou snadno k nekonečnosti, je proto dodržovat **pravidla rekurze**:
  - Musí být **definován konec** KO
    - situace, kdy je problém natolik zjednodušen, že lze stanovit jeho řešení
  - **V každém kroku** musí dojít ke **zjednodušení problému** ZP
  - **Nejprve prověřit**, zda nenastala **koncová situace** KS
    - rekurzivní krok se provede až po testu
    - při opačném pořadí by se vždy uplatnil rekurzivní krok, k posouzení koncové podmínky by nedošlo → **nekonečný cyklus**

Příklad – výpočet faktoriálu

- **Výpočet faktoriálu** **n!**

  - **iterativní řešení**
    - **0! = 1**
    - ***n*! = *n* . (*n* – 1) . (*n* – 2) … 2 . 1** **pro *n* > 0**

  řešení

```
function I_Faktorial(N:word):word;
var Fakt, I: word;
begin
     Fakt:=1;
     for I:=2 to N do Fakt:=Fakt * I;
     I_Faktorial:=Fakt;
end;
```

- **rekurzivní řešení**
  - **0! = 1**
  - ***n*! = *n* . (*n* – 1)! pro *n* > 0**

řešení

```
function R_Faktorial(N:word):word;
begin
     if N=0 then R_Faktorial:=1
     else R_Faktorial:= N * R_Faktorial(N-1);
end;
```

KO KS ZP

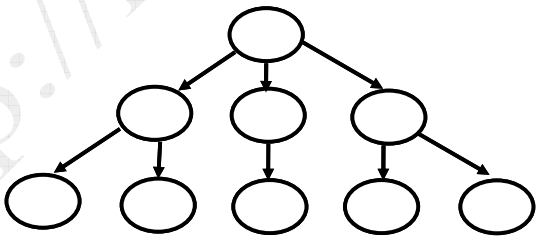
Operační a prostorová složitost rekurze

- **Operační složitost rekurze**
  - vycházíme ze skutečnosti, že v každé nižší hladině rekurze se rozsah problému snižuje
  - pro odhad složitosti lze použít **strom rekurze**
    - celková složitost ***T*(*n*)** je dána součtem složitostí všech uzlů $T_i(n)$

$$T(n) = \sum_{i=0}^{h} T_i(n)$$

    - výška stromu ***h*** pro typické úlohy roste **lineárně** nebo **logaritmicky** s ***n***
    - složitost $T_i(n)$ bývá konstantní (**O(1)**) nebo lineární (**O(*n*)**)
    - rozlišujeme případy, kdy $T_i(n)$ i**=1,2,…*h*** tvoří geometrickou řadu s kvocientem ***q*<1**, ***q*=1** nebo ***q*>1**

    - **Konstrukce stromu rekurze**



$T_1 = n$

$T_2 = q \,.\, T_1$

$T_3 = q \,.\, T_2$

    - **Složitost podle stromu rekurze**

$T_i(n)$	O(1)	$T_i(n)$ = O(*n*), $T_{i+1} = q.T_i$, $T_1 = n$		
		$q < 1$	$q = 1$	$q > 1$
log n	O(log *n*)	O(*n*.log *n*)	O(*n*.log *n*)	O(*n*.log *n*)
n	O(*n*)	O(n^2)	O(n^2)	O($n.q^n$)

- **Prostorová složitost rekurze**
  - prostorová složitost $S_p(n)$ je dána
    - délkou segmentu L_S ukládaného do zásobníku
      - obsahuje datový segment procedury (lokální proměnné), parametry procedury, návratovou adresu a adresu vrcholu zásobníku nižší hladiny rekurze
    - hloubkou rekurze h
    - délkou globálních datových objektů L_G

$$S_p(n) = h \,.\, L_S + L_G$$

Výhody a nevýhody rekurze

- **Výhody**
  - charakter programování „shora dolů“ – postupné zjednodušování problému
  - obyčejně jednodušší a přehlednější

- **Nevýhody**
  - vícenásobné zbytečné vyhodnocování argumentů funkcí – zvyšuje operační složitost
  - pro každou hladinu rekurze preventivně zakládá nové exempláře všech datových objektů – zvyšuje prostorovou složitost