

# Spojové seznamy: odstraňování potíží

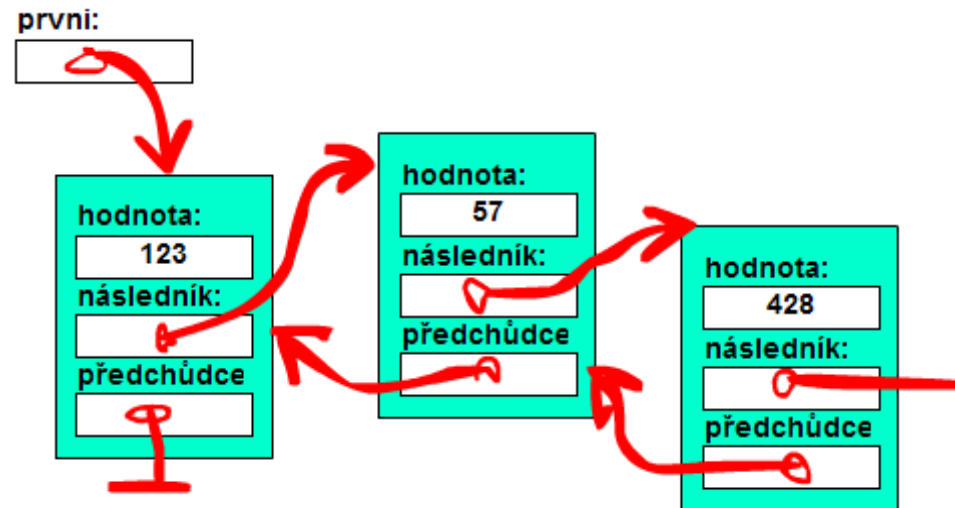
Potíž:

Nalezení předchozího prvku

Řešení:

Obousměrný seznam.

Kromě odkazu na následníka si ukládáme i odkaz na předchůdce.



Potíž:

Zvláštní zacházení s prvním prvkem

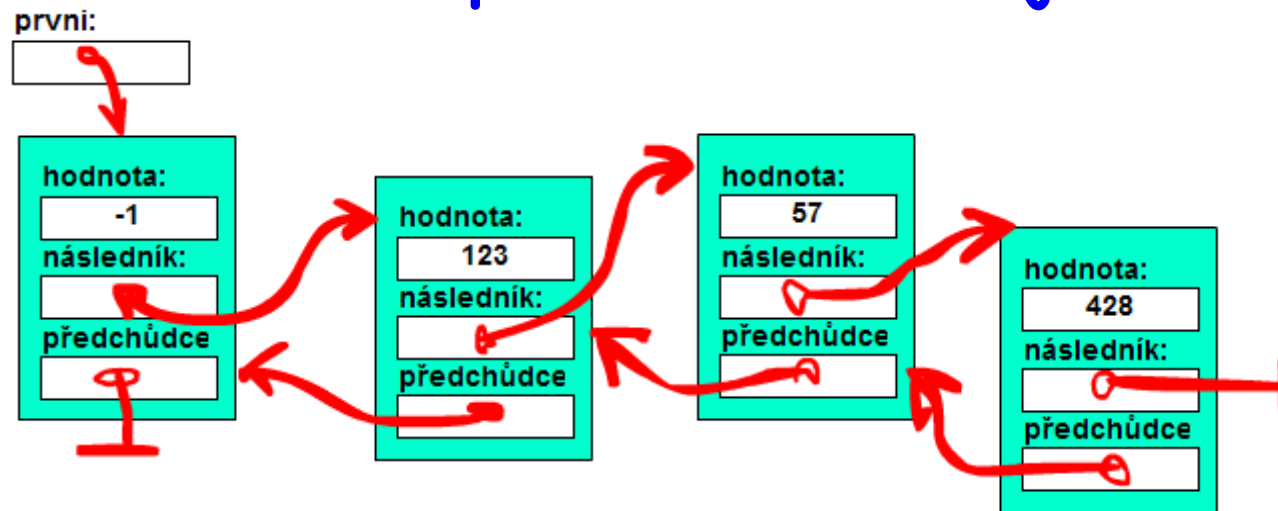
Řešení:

Seznam s hlavou.

Na začátek seznamu přidáme „nultý“ prvek HLAVU.

Každý (řádný) prvek má svého předchůdce.

Nikdy nemusíme měnit proměnnou ukazující na seznam.



Potíž:

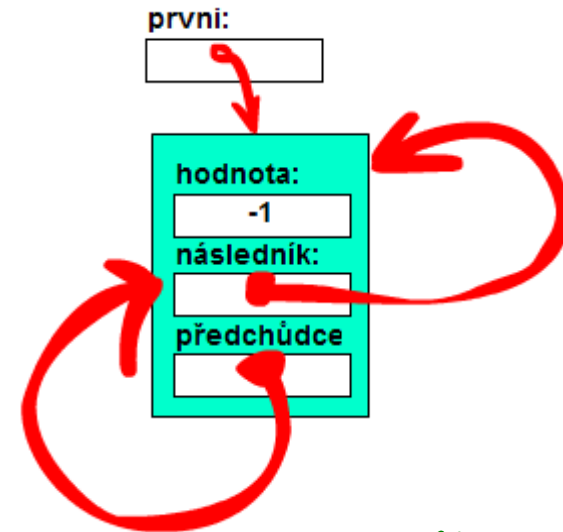
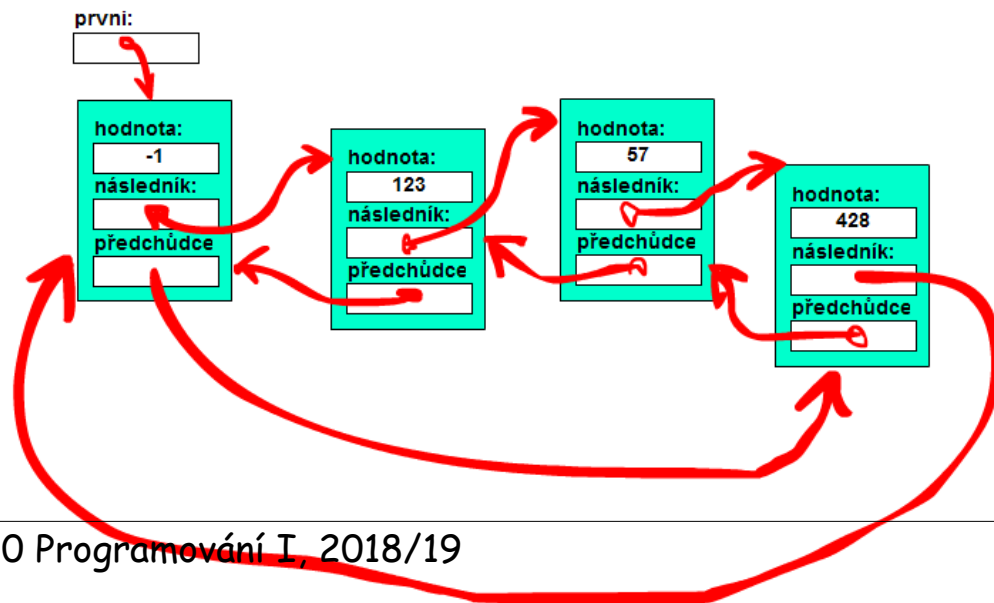
Zvláštní zacházení s posledním prvkem

Řešení:

Cyklický seznam.

Místo NIL ukazatel na první prvek  
(je jedno, s čím porovnáváme).

Každý prvek má svého následníka.



Zarážka, NIL

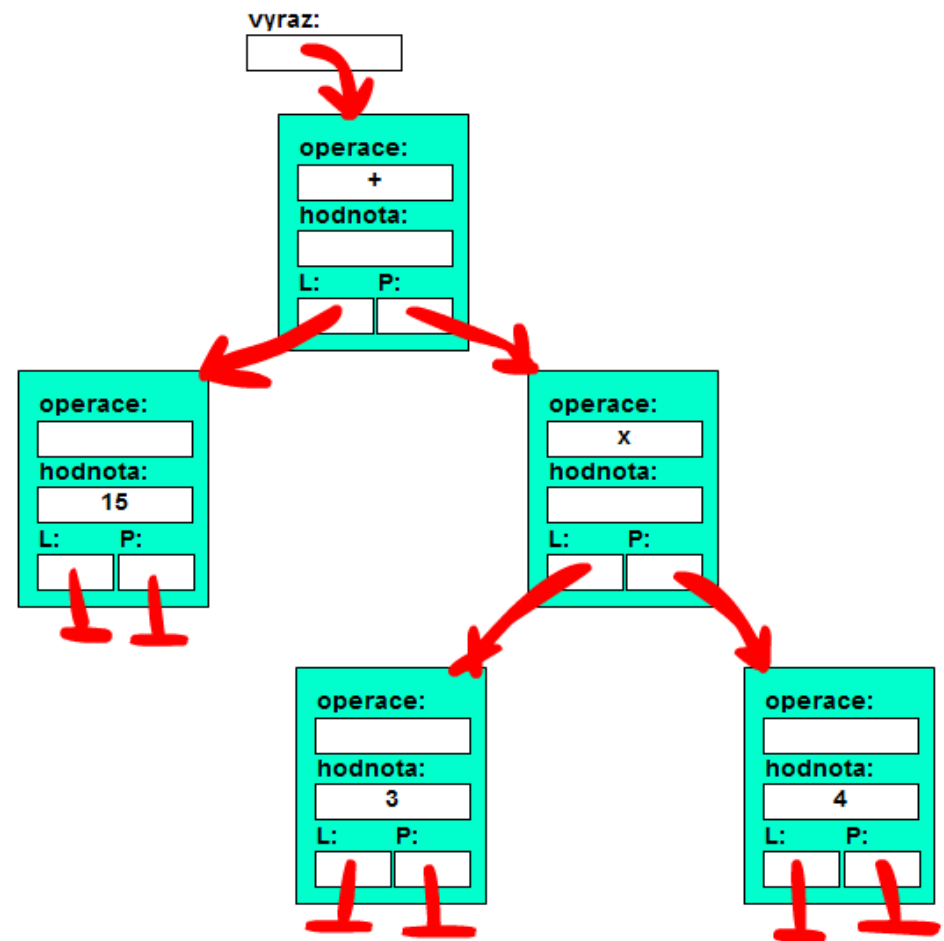
# Složitější dynamické struktury stromy

## Aritmetický výraz

```
type PVyraz = ^TVyraz;  
TVyraz = record  
    operace: char;  
    hodnota: integer;  
    L, P: PVyraz  
end;
```

vyhodnocení výrazu

výpis / průchod



# Průchody a algebraické notace

```
procedure Projdi ( V: PVyraz );  
begin  
    if V^.L=NIL then write( V^.hodnota )  
        else  
            begin  
                write( V^.operace );  
                Projdi( V^.L );  
                write( V^.operace );  
                Projdi( V^.P );  
                write( V^.operace );  
            end  
        end;  
end;
```

**PREORDER/PREFIX**

**INORDER/INFIX**

**POSTORDER/POSTFIX**

# Vyhodnocení výrazu v notaci POSTFIX

## Zásobník

Když přečteš...

...číslo: ulož do zásobníku

...operaci:

- 1) vyber ze zásobníku operandy
- 2) proved' na ně operaci
- 3) výsledek ulož do zásobníku

# Vyhodnocení výrazu v notaci PREFIX

## Rekurse

Když přečteš...

...číslo: vrat' jeho hodnotu

...operaci: proved' ji na dvě volání  
sama sebe

```
if zn='+' then
```

```
    Hodnota := Hodnota + Hodnota
```



# Vyhodnocení výrazu v notaci INFIX[1]

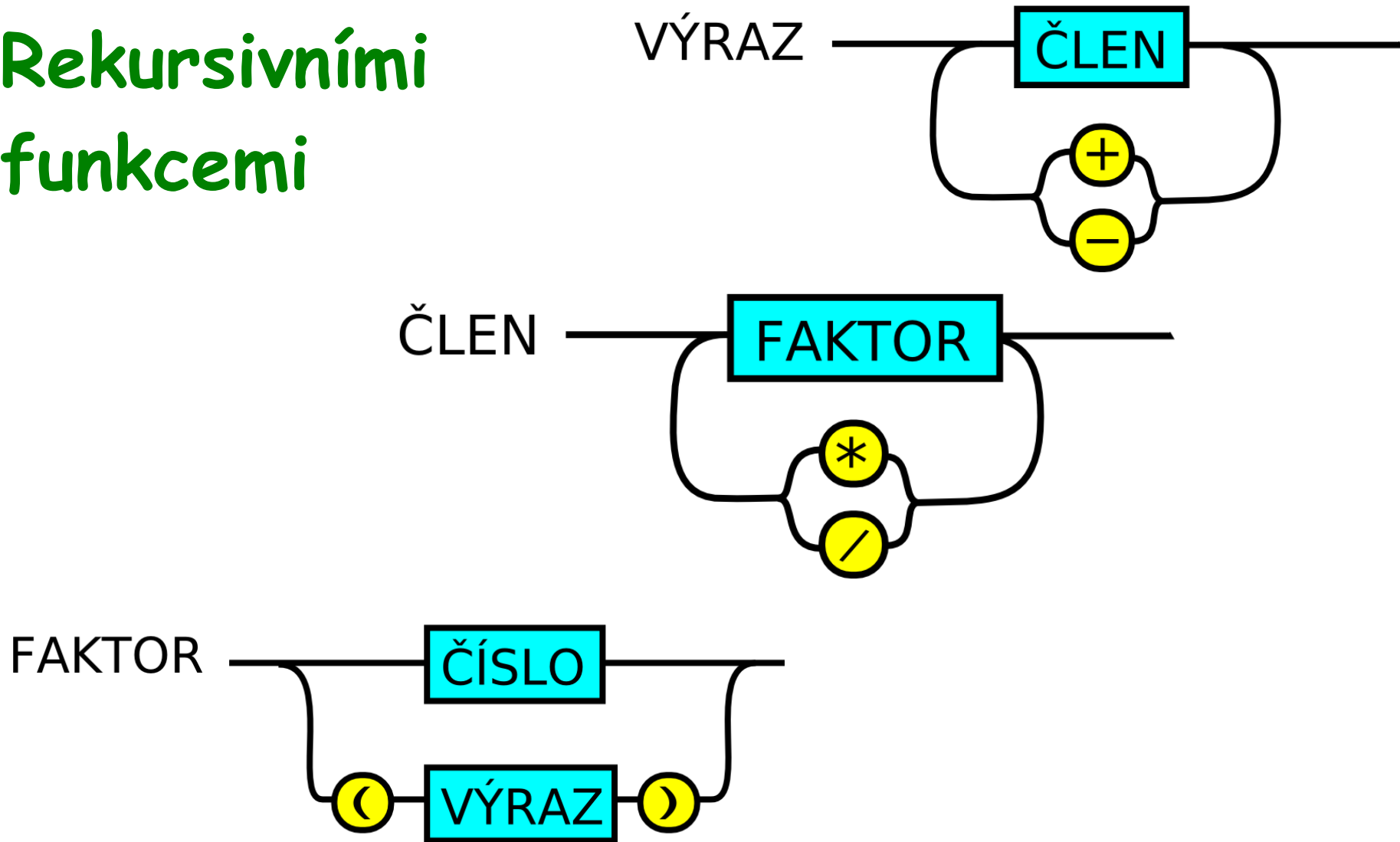
Rozkladem na podvýrazy.

Najít operaci s nejmenší prioritou  
a tu provést na části výrazu:

$$\begin{array}{l} 12 * (7+3) - (2*4-5) / 8 + 19*7 \\ 12 * (7+3) - (2*4-5) / 8 + \underline{19} * \underline{7} \\ \underline{12} * (7+3) - (2*4-5) / \underline{8} + 19*7 \\ 12 * (\underline{7+3}) - (2*4-5) / 8 + 19*7 \\ 12 * (\underline{7} + \underline{3}) - (\underline{2*4} - \underline{5}) / 8 + 19*7 \end{array}$$

# Vyhodnocení výrazu v notaci INFIX[2]

Rekursivními  
funkcemi



# Abstraktní datový typ

je definován rozhraním (interface)

rozhraní může zahrnovat

- data
- procedury a funkce (souhrnně „metody“)

## Příklad

typ Seznam:

- function JePrazdny: boolean
- procedure Pridej( prvek: TPrvek )
- procedure Vyber( var prvek: TPrvek )

Zajímá nás rozhraní a ne to,  
jak je to rozhraní realizováno uvnitř  
(pole, spojový seznam, soubor...).

# Datová struktura FRONTA

FIFO - First In First Out

# Datová struktura ZÁSObNÍK

LIFO - Last In First Out

...jsou zvláštní typy Seznamu

(stejně jako jezevčík je pes je savec je obratlovec je živočich).

=> mezi abstraktními datovými typy může být vztah „A je zvláštním případem (specializací) B“. („ISA“)

A.D.T. v různých jazycích různě, zatím neřešíme.

# Realizace zásobníku a fronty

Pomocí pole

- zásobník
- fronta
  - kruhová fronta

Pomocí spojových seznamů

Pomocí souborů...

Prioritní fronta.

# Obecný algoritmus prohledávání stavů

```
1. SeznamNeprozkoumanýchStavů  
   := { PočátečníStav }
```

```
2. while not KONEC do  
   if Seznam.JePrázdný then  
     => KONEC, řešení neexistuje
```

```
   s := Seznam.Vyber;
```

```
   pro každý t dostupný jedním krokem z s:
```

```
     if t je cílový stav then  
       => KONEC, řešení nalezeno
```

```
     else
```

```
       Seznam.Přidej( t )
```

# Zajímavé pozorování

Když Seznam je FRONTA:

Prohledávání do šířky.

(vlna)

Když Seznam je ZÁSOBNÍK:

Prohledávání do hloubky.

(backtracking)

Příklad: program zásobník, fronta, na šachovnici

Kdy prohledávat do šířky a kdy do hloubky

Prohledávání do rostoucí hloubky

