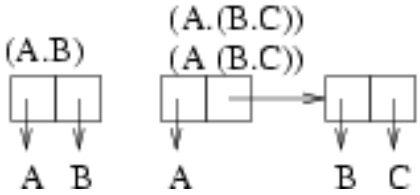
Lisp 6-1

Vnitřní implementace S-výrazů

základ: cons-buňka
- struktura o dvou ukazatelých a poli příznaků
- první ukazatel zpřístupněn funkcí **car**, druhý ukazatel funkcí **cdr**
- z cons-buněk se vytvářejí spojové zřetězené seznamy
- každá buňka se alokuje a obsadí příslušnými ukazateli jako výsledek prvodení operace cons
- při provedení cons není podstatné zda druhý argument je seznama nebo jiný obecný S-výraz

=> výsledkem nemusí být seznam, ale obecně tzv. **tečka-dvojice**

- tečka-dvojice je jediný neatomický S-výraz, seznam je jen speciální případ tečka-dvojice



(a b c d) = (a . (b . (c . (d . nil))))
= (a b . (c . (d . nil)))
= (a b c . (d . nil))
= (a b c d . nil)
= (a b c d)

(cons 'a 'b) -> (a . b)
(cons 'a (b)) -> (a b) = (a . (b . nil))
(cons 'a (cons 'b 'c)) -> (a (b . c)) = (a . (b . c))

Výpis seznamu v "tečkové" notaci:

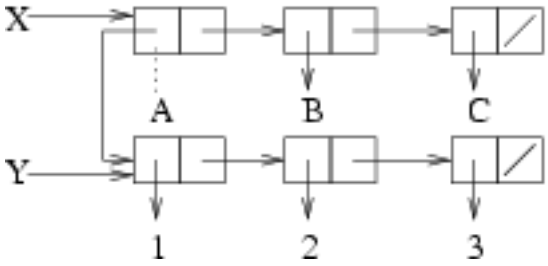
```
(defun vypis(s)
  (cond ((null s) (princ nil))
        ((atom s) (princ s))
        (t (princ "(")
           (vypis (car s)) ;první část tečka-dvojice
           (princ " . ")
           (vypis (cdr s)) ;druhá část tečka-dvojice
           (princ ")")
           nil ))) ;nil jenom kvůli rozumné návratové hodnotě funkce
```

příklad: (vypis '(a (b c) d)) -> (A . ((B . (C . NIL)) . (D . NIL)))

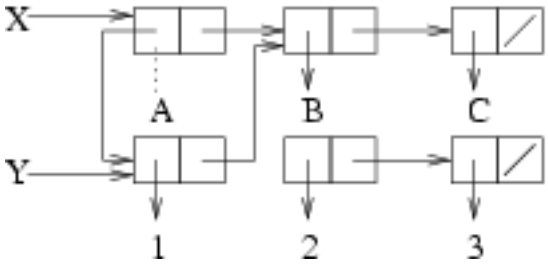
Modifikace struktur
- v čistém funkcionálním stylu struktury pouze vznikají a zanikají
- občas je to neefektivní extrémismus
- rplaca = replace car
- rplacd = replace cdr

Lisp 6-2

(setq x '(a b c))
(setq y '(1 2 3))
(rplaca x y) -> ((1 2 3) b c) ;car ukazatel na 'a jsme nahradili ukazatelem na '(1 2 3)
x -> ((1 2 3) b c)
y -> (1 2 3)



(rplacd y (cdr x)) -> (1 b c)
y -> (1 b c)
x -> ((1 b c) b c)



- očividně by nebyl žadný problém udělat cyklický seznam
- nebyl by ani problém si ale přidelat dost práce zkoumáním chybného kódu

Destruktivní spojení dvou seznamů:

```
(defun spoj (x y)
  (cond ((null x) y)
        ((atom x) (cons x y))
        (t (rplacd (last x) y)))
   x) ;návratová hodnota funkce
```

příklad: (setq a '(1 2 3))
(setq b '(4 5))
(spoj a b) -> (1 2 3 4 5)
a -> (1 2 3 4 5)
b -> (4 5)

Práce s binárním stromem:
reprezentace seznamem: (levy hodnota pravy)

*(defun **levy**(uzel) (car uzel))*

*(defun **hodnota**(uzel) (cadr uzel))*

*(defun **pravy**(uzel) (caddr uzel))*

Lisp 6-3

```
(defun vytvor(levy hodnota pravy)  (list levy hodnota pravy))
```

Přidá číslo do binárního stromu - nedestruktivně:

```
(defun pridat(cislo uzel)
  (cond ((null uzel) (vytvor nil cislo nil))
        ((< cislo (hodnota uzel)) (vytvor (pridat cislo (levy uzel))
                                          (hodnota uzel)
                                          (pravy uzel)))
        (t (vytvor (levy uzel)
                   (hodnota uzel)
                   (pridat cislo (pravy uzel))))
) )
```

příklad:
(setq strom nil)
(setq strom (pridat 5 strom)) -> (NIL 5 NIL)
(setq strom (pridat 1 strom)) -> ((NIL 1 NIL) 5 NIL)
(setq strom (pridat 3 strom)) -> ((NIL 1 (NIL 3 NIL)) 5 NIL)
(setq strom (pridat 6 strom)) -> ((NIL 1 (NIL 3 NIL)) 5 (NIL 6 NIL))
(setq strom (pridat 2 strom)) -> ((NIL 1 ((NIL 2 NIL) 3 NIL)) 5 (NIL 6 NIL))
(setq strom (pridat 0 strom)) -> (((NIL 0 NIL) 1 ((NIL 2 NIL) 3 NIL)) 5 (NIL 6 NIL))

Přidá číslo do binárního stromu - destruktivně:

```
(defun pridat2(cislo uzel)
  (cond ((null uzel) (vytvor nil cislo nil))
        ((< cislo (hodnota uzel)) (rplaca uzel (pridat2 cislo (levy uzel))) uzel)
        (t (rplaca (cddr uzel) (pridat2 cislo (pravy uzel))) uzel)
) )
```

příklad:
(setq strom (vytvor nil 5 nil)) -> (NIL 5 NIL)
(pridat2 1 strom) strom -> ((NIL 1 NIL) 5 NIL)
(pridat2 3 strom) strom -> ((NIL 1 (NIL 3 NIL)) 5 NIL)
(pridat2 6 strom) strom -> ((NIL 1 (NIL 3 NIL)) 5 (NIL 6 NIL))
(pridat2 2 strom) strom -> ((NIL 1 ((NIL 2 NIL) 3 NIL)) 5 (NIL 6 NIL))
(pridat2 0 strom) strom -> (((NIL 0 NIL) 1 ((NIL 2 NIL) 3 NIL)) 5 (NIL 6 NIL))
(pridat2 4 strom) strom -> (((NIL 0 NIL) 1 ((NIL 2 NIL) 3 (NIL 4 NIL))) 5 (NIL 6 NIL))

<u>Setf</u>

- setf je makro a lze jej považovat za zobecnění funkcí rplaca, rplacd, set, setq
  (rplaca x y) = (setf (car x) y)
  (rplacd x y) = (setf (cdr x) y)
- opakování:

rozlišujeme setq, set, setf

- (setq promena_ktera_uz_se_nevyhodnocuje argument)
- (set promena_na_jejiz_hodnotu_probehne_navazani argument)
- (setf místo_kam_dáme_hodnotu argument)

Přidá číslo do binárního stromu - destruktivně pomocí setf:

```
(defun pridat2(cislo uzel)
  (cond ((null uzel) (vytvor nil cislo nil))
        ((< cislo (hodnota uzel)) (setf (car uzel) (pridat2 cislo (levy uzel))) uzel)
        (t (setf (caddr uzel) (pridat2 cislo (pravy uzel))) uzel)
) )
```

příklad:
(setq strom (vytvor nil 5 nil)) -> (NIL 5 NIL)
(pridat2 1 strom) strom -> ((NIL 1 NIL) 5 NIL)
(pridat2 3 strom) strom -> ((NIL 1 (NIL 3 NIL)) 5 NIL)
(pridat2 6 strom) strom -> ((NIL 1 (NIL 3 NIL)) 5 (NIL 6 NIL))
(pridat2 2 strom) strom -> ((NIL 1 ((NIL 2 NIL) 3 NIL)) 5 (NIL 6 NIL))
(pridat2 0 strom) strom -> (((NIL 0 NIL) 1 ((NIL 2 NIL) 3 NIL)) 5 (NIL 6 NIL))

Zásobník pomocí proměnné: (str. 93-94 ve skriptu)

```
(defun pridat (prvek zasobnik)
  (set zasobnik (cons prvek (eval zasobnik))))
 ;(setq (eval zasobnik) .....) nelze, očekává se symbol jako první parametr

(defun odebrat (zasobnik)
  (let ((prvni (car (eval zasobnik))))
       (set zasobnik (cdr (eval zasobnik)))
       prvni)) ;návratová hodnota funkce
```

příklad:
(setq zasobnik1 nil)
(pridat 'a 'zasobnik1) ;předává se nevyhodnocený zasobnik1
(pridat 'b 'zasobnik1)
(pridat 'c 'zasobnik1)
;zasobnik1 -> (c b a)
(odebrat 'zasobnik1) -> c
;zasobnik1 -> (b a)

špatné varianty:

```
(defun pridat_spatne (prvek zasobnik)
  (format t "pred:~% zasobnik = ~S~%" zasobnik)
  (format t " zasobnik1 = ~S~%" zasobnik1)
  (setq zasobnik (cons prvek (eval zasobnik)))   ;přepíše se jen obsah lokální proměnné
  (format t "po:~% zasobnik = ~S~%" zasobnik)
  (format t " zasobnik1 = ~S~%" zasobnik1))
```

(pridat_spatne 'c 'zasobnik1) ;nebude fungovat

```
(defun pridat_spatne2 (prvek zasobnik)
  (format t "pred:~%  zasobnik = ~S~%" zasobnik)
  (format t "  zasobnik1 = ~S~%" zasobnik1)
  (setq zasobnik (cons prvek zasobnik))        ;přepíše se jen obsah lokální proměnné
  (format t "po:~%  zasobnik = ~S~%" zasobnik)
  (format t "  zasobnik1 = ~S~%" zasobnik1))
```

(pridat_spatne2 'c zasobnik1) ;taky nebude fungovat