Lisp 4-1

Mapovací funkcionály - vyjádření určitého způsobu iterace

- aplikuje funkci na stejnolehlé prvky seznamů a vrací seznam výsledků
(mapcar funkce seznam1 seznam2 ...)
- aplikuje funkci na stejnolehlé prvky seznamů a vrací triviální hodnotu
(mapc funkce seznam1 seznam2 ...)
-aplikuje funkci na celé seznamy, pak na jejich zbytky, atd. a vrací seznam výsledků
(maplist funkce seznam1 seznam2 ...)
-aplikuje funkci na celé seznamy, pak na jejich zbytky, atd. a vrací triviální hodnotu
(map funkce seznam1 seznam2 ...)
- pracuje jako mapcar ale vrací seznam vniklý jako append dílčích výsledků
(mapcan funkce seznam1 seznam2 ...)
- pracuje jako maplist ale vrací seznam vniklý jako append dílčích výsledků
(mapcon funkce seznam1 seznam2 ...)

(mapcar 'list '(a b c) '(d e f) '(g h i)) -> ((a d g) (b e h) (c f i))
(maplist 'list '(a b c) '(d e f) '(g h i)) -> (((a b c) (d e f) (g h i)) ((b c) (e f) (h i)) ((c) (f) (i)))
(mapcan 'list '(a b c) '(d e f) '(g h i)) -> (a d g b e h c f i)
(mapcon 'list '(a b c) '(d e f) '(g h i)) -> ((a b c) (d e f) (g h i) (b c) (e f) (h i) (c) (f) (i))

Seznam druhých mocnin:
(defun mocnina(x) (x x))*

(defun mocnina-list(s)
(mapcar 'mocnina s))
příklad: (mocnina-list '(1 2 3)) -> (1 4 9)

Součet seznamů:
(defun suma(x y z) (+ x y z))

(defun suma-list(a b c)
(mapcar 'suma a b c))
příklad: (suma-list '(1 2 3) '(4 5 6) '(7 8 9)) -> (12 15 18)

Anonymní funkce - Lambda
prostředek pro zvýšení abstrakce

Seznam druhých mocnin:
(defun mocnina2-list(s)
(mapcar '(lambda (n) (n n)) s))*
příklad: (mocnina2-list '(1 2 3)) -> (1 4 9)

Součet seznamů:
(defun suma2-list (a b c)
(mapcar #'(lambda (x y z) (+ x y z)) a b c))
příklad: (suma2-list '(1 2 3) '(4 5 6) '(7 8 9)) -> (12 15 18)

Lisp 4-2

Vyhodnocování

(eval výraz)
- funkce eval je základ celého vyhodnocovacího systému Lispu

```
(setq x '(1 2))
(setq y '(3 4))
(setq xy '(x y))
(cons 'append xy) -> (append x y) ;vytvořili jsme výraz, který lze vyhodnotit
(eval (cons 'append xy)) -> (1 2 3 4)
```

(apply funkce (value1 value2 ....)
- část vyhodnocovacího systému Lispu
- při vyhodnocení zápisu funkce se standardně vyhodnotí její argumenty, z jejich hodnost se udělá seznam a ten se předává k aplikaci

(funkce arg1 arg2 ...) => (apply funkce (value1 value2 ...))

```
(apply '+ '(1 2 3 4 5)) -> 15
(apply 'list '(a b c)) -> (a b c)
```

(funcall funkce arg1 arg2 ...)
- obdobné jako apply, ale navíc zde dojde ještě k vyhodnocení argumentů
- použití - když jméno funkce, kterou chceme volat dostáváme jako parametr

```
(defun operace (fce a b) (list (funcall fce a b)))

(operace '+ '1 '2) -> (3)
(operace '-  '1 '2) -> (-1)
```

Member, který vrací t/nil:

```
(defun my_member(a s)
  (eval (cons 'or (mapcar '(lambda (x) (equal a x)) s)))))
```

příklad: (my_member 'a '(b d a)) -> t

Test na podmnožinu:

```
(defun podmnžina (a b)
   (eval (cons 'and
            (mapcar '(lambda (x) (my_member x b)) a)
 )))
```

příklad: (podmnozina '(a b) '(f b s a r)) -> t

Lisp 4-3

Počet atomů ve víceúrovňovém seznamu:

```
(defun atomcount(s)
  (cond ((null s) 0)
        ((atom s) 1)
        (t (apply '+ (mapcar '(lambda (x) (atomcount x)) s)))
        ; na každý prvek seznamu zavoláme lambda funkci
) )
```

příklad: (atomcount '((a (b c)) d ((e) f g) h)) -> 8

```
(defun atomcount2(s)
  (cond ((null s) 0)
        ((atom s) 1)
        (t (apply '+ (mapcar 'atomcount2 s)))
        ; na každý prvek seznamu zavoláme atomcount2
) )
```

příklad: (atomcount2 '((a (b c)) d ((e) f g) h)) -> 8