Skratky v portacle

C -> CTRL, M -> ALT/Options

- C-x C-f -> otvoriť súbor
- C-x C-s -> uložiť súbor
- C-x k -> zavrieť súbor
- M-w -> Skopírovať text
- C-w -> vystrihnúť text
- C-y -> prilepiť text
- C - / -> Undo
- C - c C - c -> Skompilovanie funkcie
- C - c C - l -> load súboru
- C- up/down -> Prechadzanie historie v slime buffer
- C- left/right -> Presúvanie zatvoriek v texte (pozor dosť chaoticke)

Nastavenie hodnôt do symbolov

- (setf jano #'+) -> nastaví do function-value slotu symbolu funkciu + (pozor, musi byť quoted a s mriežkou)
- ak chceme zavolať funkciu z jana -> (funcall jano 3 4 5) -> vykona (+ 3 4 5) pretože vo function-value slote je funkcia +
- (setq jano 'hodnota) -> nastaví do value slotu hodnotu "hodnota"
- (defvar symbol 'hodnota) -> rovnako nastavi do value slotu hodnotu "hodnota"
- (symbol-value 'symbol) -> vráti hodnotu z value slotu (hodnota)
- (symbol-function 'symbol) -> vráti hodnotu z function-value slotu
- (setf (symbol-function 'symbol) #'+) -> nastaví do value slotu funkciu (aspon myslim) -> dá sa volať bez funcall : (symbol 10 12)

Funkcie

- (lambda (x) (print x)) -> definícia anonymnej funkcie
- ((lambda (x) (print x)) 10) -> Definuje anonymnú funkciu a zavolá ju s parametrom 10
- (funcall #'(lambda (x) (print x)) 10) -> definicia anonymnej funkcie so zavolaním pomocou funcall (treba kvotovať)
- (defun y (x) (print x)) -> definícia funkcie
- (symbol-function 'y) -> vráti funkciu
- (y 10) -> volanie funkcie
- (funcall #'y 12) -> volanie funkcie pomocou funcall (pozor treba kvotovať)
- (defun vonkajšia-funkcia (x1)

```
(let (( pomocna-funkcia (lambda (x) ( print x )))
    ( funcall pomocna-funkcia x1)
)
```

- -> definicia lokalnej funkcie “pomocna funkcia” pomocou let (do symbolu pomocna-funkcia sa uloži anonymna funkcia, ktorú potom voláme pomocou funcall (netreba kvotovanie pretože to je vo value slote?)

-
```
(defun vonkajšia-funkcia (x1)
    (let ((pomocna-funkcia))
        (setf (symbol-function ‘pomocna-funkcia) (lambda (x) (print x) ) ) )
        (pomocna-funkcia x1)
    )
)
```
    - -> rovnako ako predtým ale najprv sa definuje symbol pomocna-funkcia, nasledne sa pomocou setf nastavi anonym funkcia do symbolu (function value slotu). Tymto netreba funcall pri volaní funkcie v tele

-
```
(defun vonkajšia-funkcia (x1)
    (labels (( pomocna-funkcia (x) (print x) ) )
        (pomocna-funkcia x1)
        )
)
```
    - -> rovnako ako hned nad tým ale s využitím labels (nepoužívame anonymnú funkciu ale funkciu (bez lambda)

Control Flow

Sekvencie

- (progn 1 2 3 4) -> postupne vykoná kroky (prejde číslami 1 2 3 4 ale namiesto čísel môžu byť volania funkcii (napr printy) (sekvencia)
- (let () 1 2 3 4) -> vykoná to isté ako progn (tiež sekvencia)

Vetvenie

- (if (< 3 4) 4 3) -> if funguje ako ternarny operator, prvy param je predikat, druhy parameter je hodnota ktora sa vrati ak predikát platí tretí je hodnota ktorá sa vrati ak predikat neplatí
- (when t (print “ok”) 7) -> prvý parameter je predikát (t alebo NIL), ak je predikát rovný T tak vykoná telo (ostatné parametre) inak vráti NIL
- (unless (not t) (print “ok”) 7) -> rovnaké ako when ale negované (ak je predikát nil vtedy sa vykoná telo)
-
```
(cond ( (< -1 0) (print “menej”) (- -1))
        (nil (print “toto sa nikdy nevykona”))
        (t (print “toto je default vetva”))
)
```
    - -> cond funguje podobne ako switch postupne sa prechádzajú všetky podmienky a tá ktorá prvá platí tak ten riadok sa vykoná

Iterácia

-
```
(do ((x 0 (1+ x))
    (y 0 (1+ y))) ;;; -> definicia premenných (nazov init_hodnota aktualizacia_v_iter)
```

```
((= x 5) t)      ;;; -> ukončovacia podmienka cyklu
(print y))       ;;; -> telo cyklu
```

Zoznamy

- (list 1 2 3 4) -> vytvorí zoznam (1 2 3 4)
- (list '(1 2 3 4)) -> vytvorí zoznam v zozname ((1 2 3 4))
- (list 1 (list 2 3) (list) (list 4)) -> (1 (2 3) NIL (4))
- (make-list 4) -> vytvorí (NIL NIL NIL NIL)
- (make-list 4 :initial-element 2) -> vytvorí (2 2 2 2)

Testovanie zoznamov

- (listp '(1 2 3)) -> listp je predikát na testovanie či parameter je zoznam (tu vráti t)
- (null '()) -> testovanie či je zoznam prázdny (vráti T v tomto prípade)
- (equal '(1 2) (list 1 2)) -> otestovanie či sa dva zoznamy rovnaju !!! pozor používať equal a nie rovná sa alebo "eq"

Rozoberanie zoznamu

- (car '(1 2 3 4)) -> vráti prvý prvok zoznamu 1
- (cdr '(1 2 3 4)) -> vráti "tail" zoznamu bez prvého prvku (2 3 4)
- (cadadr '(1 (2 3) NIL (4))) -> sekvenčne zavolané car a cdr za sebou v jednej funkcii (tu to vráti hodnotu 3 lebo najprv sa vykoná cdr = ((2 3) NIL (4)) -> car = (2 3) -> cdr = (3) -> car = 3
- (first '(1 2 3 4)) -> prvý prvok "1"
- (second '(1 2 3 4)) -> druhý prvok "2"
- …
- (rest '(1 2 3 4)) -> ako cdr (2 3 4)
- (nth 0 '(1 2 3)) -> výber n-tého prvku = 1
- (nth 3 '(1 2 3)) -> ak je index väčší ako dlžka vráti NIL
- (nthcdr 0 '(1 2 3)) -> vráti zbytok zoznamu - n prvkov = (1 2 3)
- (nthcdr 1 '(1 2 3)) -> (2 3)
- (last '(1 2 3)) -> (nthcdr (dlz-1)) -> vráti posledných n prvkov (tu je bez param vráti 3)
- (last '(1 2 3) 2) -> (2 3)
- (last '(1 2 3) 0) -> () / NIL
- (butlast '(1 2 3)) -> vráti zoznam okrem posledný n prvkov (bez param vráti (1 2))
- (butlast '(1 2 3) 2) -> vráti (1)

Rozširovanie zoznamu

- (cons 1 '(2 3)) -> pridá prvý parameter k zoznamu -> (1 2 3)
- (cons '(1 2) '(3 4)) -> ((1 2) 3 4) !!! Pozor, pridáva celý zoznam ako jeden prvok !!
- (append '(1 2) '(3 4) '(5 6)) -> spojí hodnoty zo zoznamov do jedného -> (1 2 3 4 5 6)
- (list-length '(1 2 3)) -> vráti dĺžku zoznamu

Hľadanie prvku

- (member 2 '(1 2 3)) -> vráti T pretože 2ka sa nachádza v zozname
- (member '(2) '((1) (2) (3))) -> vráti N pretože member používa eq na testovanie (to nefunguje pri porovnavani zoznamov
- (member '(2) '((1) (2) (3)) :test #'equal) -> vráti T pretože nastavíme funkciu na testovanie "equal" ktorá porovnáva zoznamy

- (member 2 '((a 1) (b 2) (c 3)) :key #'cadr) -> použitie funkcie cadr pre vybratie hodnoty (tým, že hladame iba value v zozname zoznamov musime vybrať hodnotu z kazdeho podzoznamu pomocou cadr pred testovanim

Rôzne užitočné funkcie

- (remove 2 '(1 2 3 2 1)) -> odstráni všetky výskyty hodnoty 2 v zozname
- (remove 2 '(1 2 3 2 1) :count 1 :from-end t) -> odstráni iba jeden výskyt od konca
- (remove 2 '(1 2 3 2 1) :start 1 :end 2) -> odstráni prvok medzi prvým a -2 od konca (myslim)
- (remove 2 '((1 3) (3 1)) :test #'< :key #'car) -> odstráni prvky menšie ako 2 (podla zadaneho testu) a vyberá hodnoty pomocou car pretože zoznam v zozname
- (reverse '(1 2 3)) -> otočí zoznam
- (length '(1 2 3)) -> dĺžka zoznamu
- (subseq '(1 2 3 4) 1 3)-> vráti podzoznam ktory bude začínať v prvom prvku a končiť v druhom (index 1 a index 2), (prvé čislo je inkluzívne druhé nie)

Funkcie vyššieho rádu

8 tyzden

štvrtok 16. novembra 2023 13:33

Function – vybera hodnotu z function-value slotu
Setf – nastavuje value slot
Funcall - volá funkciu premennej z value slotu (prvý parameter), dalšie parametre sú vložene do param funkcii
Apply - podobne ako funcall ale berie parametre ktore maju isť do funkcie z listu

Testovanie s predikátom

(every #'(lambda (x) (evenp (print x))) '(2 4 6)) -> funkcia ktora je v druhom parametri musi vratiť nil/t

-> predikaty skončia pri prvej hodnote ktora neplati

(some) -> aspon jeden platí
(notevery) -> aspon jeden neplatí
(notany) -> ani jeden neplatí

Spracovanie zoznamu/sekvencie

!!! Member, member-if member-if-not atd vracajú prvý prvok a vsetky dalšie kde predikat platí
(member-if #'evenp '(1 2 3))
(member-if-not)
(member-if #'evenp '(1 2 3) :key #'1+)

Member-if -> vrati zoznam prvkov pre ktore platí predikát
Member-if-not -> nemalo by sa už používať, treba použiť
complement pre predikát

(remove-if)
(remove-if-not)

Redukcia zoznamu

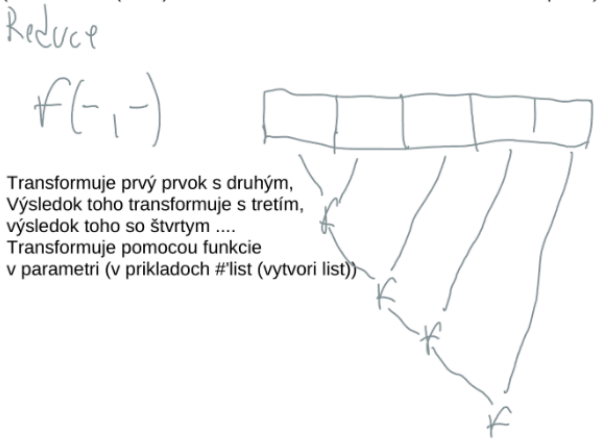
Reduce -> transformácia
(reduce #'list '(1 2 3))

(reduce #'list '(1 2 3) :initial-value 0) -> nezačína sa prvy prvok s druhym ale initial value s prvým

(reduce #'list '(1 2 3) :from-end t -> začína sa od konca)

(reduce #'list '(1 2 3) :start 1 -> posunieme začiatok)

(reduce #'list '(1 2 3) :end 2 -> nebudeme končiť na konci ale na 2 prvku)



Transformuje prvý prvok s druhým,
Výsledok toho transformuje s tretím,
výsledok toho so štvrtym ....
Transformuje pomocou funkcie
v parametri (v prikladoch #'list (vytvori list))

Mapovanie zoznamu

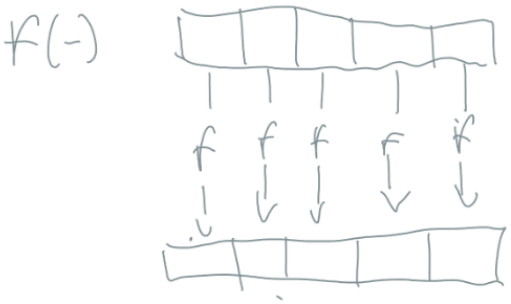
(mapcar #'- '(1 2 3))
(mapcar #'+ '(1 2 3) '(12 12))

(mapc #'(lambda (x) (print (- x))) '(1 2 3))

(maplist #'append '(1 2 3) '(a b))

(mapl #'(lambda (x) (print (length x))) '(1 2 3))

Mapcar – každá hodnota sa vpustí funkciou a vráti sa zoznam nových hodnôt

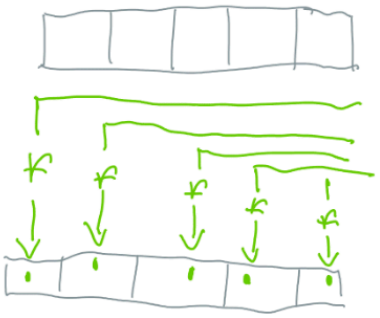


– Ak bude viac parametrov v f ta potrebujeme viac zoznamov
– ak sú zoznamy iného dĺžky/výsledný zoznam má veľkosť najmenšieho

MapC -> podobne ako mapcar akurát vráti pôvodný zoznam

Mapcan -> rovnako ako mapcar ale ak je vystup funkcie zoznam tak spojí zoznam zoznamov do jedného zoznamu

Maplist -> Aplikuje funkciu na podzoznam



Mapl -> Rovnako ako mapcar a mapc, vykoná funkcie na podzoznamy ale vráti pôvodný zoznam

Mapcon -> rovnako ako mapcar a mapcan -> ak sú vystupom funkcie zoznamy, tak sa zoznamy spoja do jedného