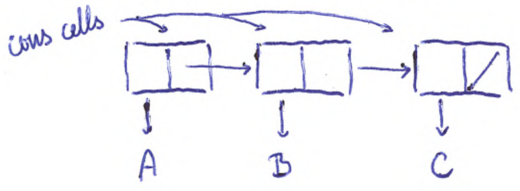
MI-SP-SP-9 (MI-FLP) (přednášky 2-7)

Lisp: atomy, seznamy, funkce, cons buňky, rekurze, iterační cykly, mapovací funkcionály, proměnné, strukturované datové typy, makra, realizace nedeterminismu

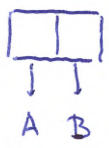
- všechno je atom nebo list

(čísla, proměnné, konstanty, stringy, ...)

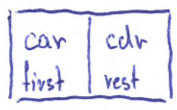
- listy jsou reprezentovány závorkami - prázdný list () = NIL



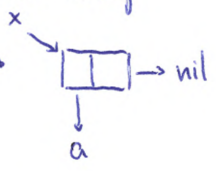
(A B C)

dotted pair 

(A . B)



- funkce cons - vytváří nové buňky

(setf x (cons 'a nil)) → 

funkce v lispu: (defun name (parameters) body)

(defun square (x) (* x x))

- všechny argumenty jsou evaluovány před zavoláním funkce
  → funkce quote nebo ' tomuto zabrání

podmínky:
```
(cond (p1, e1)          (if predicate
      (p2, e2)              consequent
      ...                   alternative)
      (pn, en))
```

predikáty:
```
(null s)  - T pro prázdný list      (length s), (nth s), (last s)
(number s)                          (butlast s)
(listp s) - T pro list s
```

rekurze:
```
(defun Fib (n)
  (if (< n 2) n
      (+ (Fib (- n 1)) (Fib (- n 2)))
  ))
```

- koncová rekurze - poslední operací ve funkci
  - výborná z pohledu implementace → není třeba další stack frame

```
(defun factorial (N)
  (if (= N 0) 1
  (* N (factorial (- N 1)))))
```
=>
```
(defun factorial (N)
   (factorial-aux N 1))
(defun factorial-aux (N ACC)
  (if (= N 0) ACC (factorial-aux (- N 1) (* N ACC))))
```

(defvar *global* 36) - globální proměnné

(defconstant big 100) - konstanty

<u>arrays a vectors</u> (vector 1 2 3)

(make-array 5 :initial-element nil)

<u>sequences</u> (length '(1 2 3)) → 3 (reverse '(1 2 3)) → (3 2 1)

(subseq '(1 2 3 4 5 6 7) 1 5) → (2 3 4 5)

copy-seq - kopíruje sekvenci (pole, vektory, stringy)

(concatenate a b)

<u>Hash tables</u>

make-hash-table

(gethash 'foo *h*) ≡ h['foo']

(setf (gethash 'foo *h*) 'abcd) ≡ h['foo'] = abcd

maphash

Stack - listy podporují push a pop operace

(push 'a x)

(pop x) → (a)

- stejně tak mohu vyrobit Set a Tree struktury

Mapovací funkcionály ← (funkce s funkcí jako argumentem)

(mapcar function list1 list2 ... listn)

- iteruje přes každý list a na všechny prvky volá danou funkci
  - všechny elementy na i-tých pozicích jsou zpracovávány najednou
  - končí když jeden list dojde (mapcar #'list '(1 2) '(5 8 10) '(9 9 9))
    → ((1 5 9)(2 8 9))

mapcan - spojuje výsledky do jediného listu
  → (1 5 9 2 8 9)

maplist - pracuje na sub listech
  → (((1 2)(5 8 10)(9 9 9)) ((2)(8 10)(9 9)))

mapc / mapcon / mapl - vrací první list (list 1)

Iterace (dolist (x '(1 2 3)) (print x))

(dotimes (i 4) (print i))

(do ((i 0 (1+ i))) ((>= i 4)) (print i))

(loop body-form*)

Macros - makra jsou kompilovaná a expandovaná

→ macro expansion ~~time~~ time - v tu chvíli není přístup k datům

- jsou napsaná v Lispu

- mnoho implicitních operátorů jsou macro (looping, when, unless, ...)

```
(defmacro unless (condition &rest body)
    `(if (not ,condition) (progn ,@body)))
```

debugování maker: macroexpand , macroexpand - n (expanduje)

- když jde obojí, je lepší funkce

```
(defun +1 (+1 x))

(defmacro +1 (x) `(+ 1 ,x))
```

- občas musíme použít makra
  - ← nechceme expandovat parametry
  - → chceme dynamicky vytvářet funkce s danými argumenty

- makro se vykonává před runtimem, může ale používat všechny funkce (i naše) → možné je toto:

```
(defmacro xreverse (x) (reverse x))

  (xreverse (3 8 -)) → 5
```

(choose '(1 2 3)) - random výběr

Funtional programing: - vracíme hodnoty, neměníme stav (pure functions)

→ nemáme žádné side effecty

→ neměníme přímo vstupní argumenty funkce

- Lisp obsahuje i destruktivní funkce