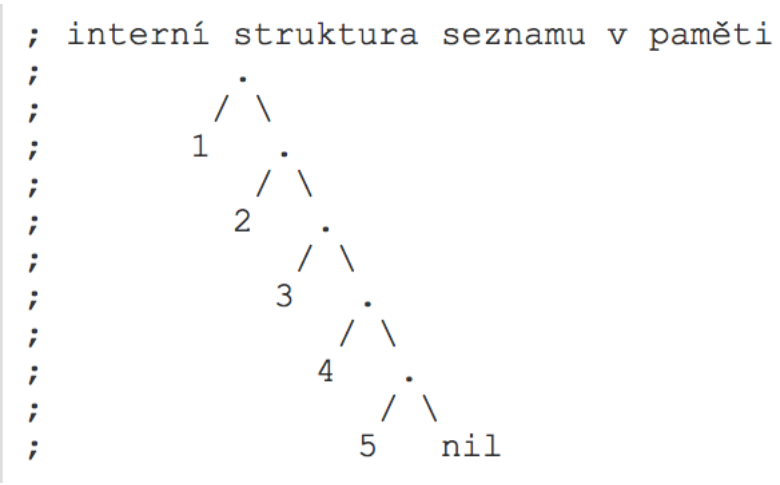
1 Základní vyhodnocovací cyklus Lispu. Práce se seznamy v Lispu. Destruktivní a nedestruktivní konstrukty. Lambda kalkulus, funkce vyššího řádu, lokální proměnná, kombinace iterativních konstruktů a rekurze. Modely pravidlového progamování. (A4B33FLP)

1.1 Vyhodnocování v Lispu

- vyhodnocování probíhá způsobem REPL: Read-Eval-Print-Loop
- často se u Lispu používá překladač, který zdrojový kód přeloží do optimalizovaného kódu podobného výsledkům Céčkového překladače
- zapisujeme v prefixové notaci (+ 1 2)
- pokud součástí seznamu je další podseznam, tak se nejprve vyhodnocuje ten
  - dochází k rekurzivnímu zanořování
  - teoreticky není omezená hloubka zanoření, záleží spíš na kapacitě fyzických zdrojů

1.2 Seznam v Lispu

- všechno v lispu je reprezentováno jako seznam



- zápis seznamu ve zdrojovém kódu: (prvek1 prvek2 prvek3 ...)
- obecně se dá říct, že první prvek seznamu je chápaný jako název funkce a této funkci se předá následně zbytek seznamu jako seznam parametrů
- základní funkce pro práci se seznamem:
  - **car -** vrátí první prvek seznamu
  - **cdr -** vrací podseznam seznamu bez prvního prvku
  - cons - spojuje dva zadané seznamy
  - append - přidá nakonec seznamu další prvek

1.3 Destruktivní a nedestruktivní konstruktory

- nedestruktivní konstrukce
  - v těchto konstrukcích funkcionálního programování funkce nemění hodnotu proměnné a pouze vrací novou hodnotu
- destruktivní konstrukce
  - pro odlišení se názvy funkcí zapisují s vykřičníkem
  - při jejich použití se například zapisuje a tím se mění hodnoty proměnných

1.4 Lambda kalkulus

- lambda kalkulus je konstrukce funkcionálního programování, pomocí níž jsme schopni uvnitř funkce definovat jednorázovou lambda funkci a rovnou jí uvnitř využít
- chovají se jako funkce
- mezivýpočty

1.5 Funkce vyššího řádu

- funkce, které jsou schopné v parametru přijmout jiné funkce nebo vrátit funkci
- apply
  - jako první argument očekává funkci a následně seznam prvků
  - funkci postupně aplikuje na všechny prvky předaného seznamu
  - (apply + '(1 2 3)) vrací tedy 6
- map
  - jako první argument opět očekává funkci a následně tuto funkci aplikuje na každý prvek seznamu separátně
  - (define (mocnina a) (* a 2))
  - (map mocnina '(1 2 3))
  - >> (2 4 6)

1.6 Lokální proměnné

- lokální proměnnou v Lispu můžeme definovat pomocí funkce "let"
- (let (x data))
- tento způsob však ve funkcionálním programování není obvyklou konstrukcí a vše co je zapsáno pomocí lokální proměnné jde přepsat do rekurzivní formy
- globální proměnné vůbec neexistují

1.7 Iterativní konstrukce vs rekurze

- rekurze
  - základní přístup funkcionálního programování
  - funkce volá sebe sama
  - je nutné definovat ukončovací podmínku
  - tail recursion optimalizace
    - ve chvíli kdy dochází k rekurzi v posledním volání, tak se šetří místo na zásobníku
    - není nutné ukládat starý rámec a rekurze se vrací pouze skokem
- iterace
  - využívá akumulátor
  - částečné výsledky jsou odesílány pomocí akumulátoru jako parametr funkce

1.8 Modely pravidlového programování

- pravidlové programování pouze definuje pravidla - predikáty
- pomocí resoluční metody se hledá řešení zadané úlohy
- neobsahuje přesný popis řešení - algoritmus