Funkcionální programování

- co dělá program?
  - dostane vstupní hodnoty, zpracuje je a na jejich základě stanoví hodnoty výstupní
- matematická funkce dělá přibližně totéž, liší se jen vnitřní mechanismus (postup výpočtu)
- funkcionální programování vychází z představy, že program je vlastně (potenciálně dost složitá) funkce

Typické rysy

- čisté funkce – výsledek závisí jen na argumentech, žádné vedlejší efekty
- funkce 1. třídy – funkce je hodnota jako každá jiná
- stavové informace představují riziko – stav výpočtu v argumentech a návratových hodnotách, ne v proměnných
- vše je výraz – cílem je ho vyhodnotit a získat výsledek

LISP (LISt Processor)

- klasický funkcionální programovací jazyk
- vytvořen ve 2. polovině 50. let
- původně jazyk pro umělou inteligenci
- prefixová notace s hojným používáním závorek
- volání funkce: (*funkce operandy*)
  např. (+ 19 4)
- vznikaly hardwarové implementace – LISP procesory a počítače

Scheme

- modernější dialekt LISPu (2. pol. 70. let)
- jednodušší
- principy stejné, liší se názvy některých konstrukcí
- Scheme na webu:
  - www.schemers.org
  - racket-lang.org – volná implementace Racket, GUI pro vývoj programů

Identifikátory

- extrémně volná pravidla
- mohou obsahovat
  - písmena a–z, A–Z (nerozlišuje malá/velká)
  - číslice
  - znaky ? ! . + – / < = > : $ % ^ & _ ~
  - nesmí začínat číslicí, +, – ani . (tak začínají čísla)
- příklady:
  - x3, Maximum, prumerny-plat, ?$*

Konvence pro identifikátory

- jména predikátů (vracejí pravdivostní hodnotu) končí otazníkem, např. number?
- jména procedur s vedlejšími efekty (např. změna hodnoty nějaké proměnné) končí vykřičníkem, např. pridej-prvek!
- procedury konvertující jeden typ na druhý bývají pojmenovány typ1->typ2

Čísla

- celá: 160, +24, −78
- racionální (zlomky): 1/2, −29/30
- s plovoucí řádovou čárkou: 3.14, .33, −7.689e4
- komplexní: 1+2i
- nemá přísnou typovou kontrolu ani typované proměnné, výpočet může kombinovat několik typů

Řetězce, pravdivostní hodnoty

- **řetězce** jsou uzavřeny do dvojitých uvozovek
  - "ahoj" "to je ale věc"
- **pravdivostní hodnoty**
  - true a false zapisovány jako #t a #f
  - revidovaná specifikace umožňuje místo #f používat ve stejném významu i prázdný seznam (), používá řada implementací

Seznamy

- lineární seznam **základní datovou strukturou**
- uzavřen do závorek, prvky oddělovány mezerami
  - (a b c), (12 "šroubek" .8), ()
- lze vnořovat
  - ((1) (2 (3 4) 5 6) 7 ((8)))
- prefixová notace – první prvek je chápán jako název procedury, další jako její argumenty
  - (* 4 9), (+ 15 27 58)

Výrazy

- nejjednodušším výrazem je konstanta
  - 26 nebo "raz dva"
- seznam je interpretován jako volání procedury
  - (/ 2 3)
- pokud má seznam sloužit jako data, je třeba použít quote
  - (quote (a b c)) představuje seznam (a b c)
  - zkratkou za quote je ' ... obvyklý zápis téhož je '(a b c)

Základní aritmetické operace

- `+` sčítání `(+ 8 7)` → 15
- `–` odčítání `(– 8 7)` → 1
- `*` násobení `(* 8 7)` → 56
- `/` dělení `(/ 8 7)` → 8/7
- nejsou žádné priority, je třeba je vyjádřit závorkami
  - 5 * (10 + 12) `(* 5 (+ 10 12))`
  - (8 – 3) / (2 + 5) `(/ (– 8 3) (+ 2 5))`

Testování čísel

- number? je daná hodnota číslem?
- testy číselných typů: integer?, rational?, real?, complex?
- =, <, <=, >, >= porovnání
  test na nerovnost: (not (= a b))
- testy hodnot: zero?, positive?, negative?, odd?, even?

Aritmetické funkce (1)

(abs x)	absoluní hodnota
(quotient m n)	celočíselné dělení
(modulo m n)	zbytek po celočíselném dělení
(max m n ...)	maximum
(min m n ...)	minimum
(round x)	nejbližší celé číslo
(numerator x)	čitatel zlomku
(denominator x)	jmenovatel zlomku

Aritmetické funkce (2)

- (sqrt x) odmocnina
- (exp x) e^x
- (expt x y) x^y
- (log x) $\log_e x$ (přirozený logaritmus)
- (sin x), (cos x), (tan x) goniometrické funkce
- (asin x), (acos x), (atan x) inverzní

Číselné speciality

- **nekonečné opakování** za desetinnou čárkou
  - zobrazuje klasicky čarou nad číslicemi
  - (/ 1 3) vydá $0.\overline{3}$
- upozornění na **zaokrouhlovací chyby**
  - prefix #i
  - (sqrt 2) vydá **#i**1.4142135623730951

Definice funkce

- **(define (*jméno-funkce argumenty*) *výraz*)**
- funkce má dané *jméno*
- identifikátory tvořící *argumenty* jsou při volání nahrazeny hodnotami získanými vyhodnocením skutečných argumentů funkce
- o přiřazení hodnot argumentům rozhoduje pořadí
- následně se vyhodnotí *výraz* a výsledná hodnota je výsledkem volání funkce

Příklad funkce

- definujme funkci plocha-kruhu; argumentem je poloměr r, počítá Pi * r^2
- (define (**plocha-kruhu** r) (* 3.14 (* r r)))
- nyní lze použít např.

(plocha-kruhu 5)

(define (plocha-kruhu r) (* 3.14 (* r r)))

vypočte se (* 3.14 (* 5 5))

vytvořeno s podporou
projektu ESF