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 (1 2 3)) představuje seznam (1 2 3)
  - zkratkou za quote je ' ... obvyklý zápis téhož je '(1 2 3)

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) — absolutní 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 – plocha kruhu

- 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))

Pokračování – prstenec

- definovanou funkci lze použít v definici dalších funkcí
- například plocha prstence:

```
(define (plocha-prstence vnejsi vnitrni)
   (- (plocha-kruhu vnejsi) (plocha-kruhu vnitrni)))
```

- (plocha-prstence 8 5)

```
(- (plocha-kruhu 8) (plocha-kruhu 5))
(- (* 3.14 (* 8 8)) (* 3.14 (* 5 5)))
```

Komentáře

- řádek začínající středníkem je komentářem
- doporučuje se popisovat v komentářích syntax a význam definovaných funkcí
  - první řádek – **kontrakt**: jméno, typy argumentů a výsledku
  - další řádky: slovní popis funkce
- ;;plocha-prstence: číslo číslo -> číslo
  ;;vypočte plochu prstence s poloměrem 'vnejsi'
  ;;a poloměrem otvoru 'vnitrni'
  (define (plocha-prstence vnejsi vnitrni) ...)

Program ve Scheme

- sada spolupracujících funkcí
- nedělejte velké kroky (dlouhé a složité funkce)
- vhodná je dekompozice – každá funkce vyřeší logicky uzavřenou část problému
- k vyřešení dílčích podproblémů volá další funkce (rozděl a panuj)
- snazší, srozumitelnější a lépe laditelné

Kvadratická rovnice (1)

```
;kvadrovnice: číslo číslo číslo -> seznam čísel
;vypočte kořeny kvadratické rovnice
;ax^2 + bx + c = 0
; vrací seznam kořenů
(define (kvadrovnice a b c)
   (list (koren1 a b c)
         (koren2 a b c)))
```

Kvadratická rovnice (2)

```
;koren1,2: číslo číslo číslo -> číslo
;vypočte kořen kvadratické rovnice podle vzorce
;(–b + sqrt(D)) / 2a  a  (–b – sqrt(D)) / 2a
(define (koren1 a b c)
   (/ (+ (– b) (sqrt (diskrim a b c))) (* 2 a)))
(define (koren2 a b c)
   (/ (– (– b) (sqrt (diskrim a b c))) (* 2 a)))

;diskrim: číslo číslo číslo -> číslo
;vypočte diskriminant kvadratické rovnice: b^2 – 4ac
(define (diskrim a b c)
   (– (* b b) (* 4 a c)))
```

Srozumitelnost programů

- komentáře a volné místo činí program snadněji srozumitelným
- je záhodno používat mnemotechnické identifikátory
- strukturu výrazu lze zvýraznit odsazením, které vyznačí úrovně vnoření jednotlivých konstrukcí

```
(define (koren1 a b c)
   (/ (+ (– b)
         (sqrt (diskrim a b c)))
      (* 2 a)))
```

vytvořeno s podporou
projektu ESF