Paradigmata programování 1

Přednáška 3. Rekurze 1

Michal Krupka

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Obsah

Příklad: procentuální podíl

Příklad: procentuální podíl

```
(define (percentage-1 part whole)
  (* (/ part whole) 100))
```

Příklad: procentuální podíl

```
(define (percentage-1 part whole)
  (* (/ part whole) 100))

(define the-whole 10625449)
```

Příklad: procentuální podíl

```
(define (percentage-1 part whole)
  (* (/ part whole) 100))

(define the-whole 10625449)

(define (percentage-2 part whole)
  (let ((whole (if (eq? whole #true)
                   the-whole
                   whole)))
    (* (/ part whole) 100)))
```

Příklad: procentuální podíl

```
(define (percentage-1 part whole)
  (* (/ part whole) 100))

(define the-whole 10625449)

(define (percentage-2 part whole)
  (let ((whole (if (eq? whole #true)
                   the-whole
                   whole)))
    (* (/ part whole) 100)))

(define (percentage-3 part whole)
  (percentage-1 part
                (if (eq? whole #true)
                    the-whole
                    whole)))
```

Příklad: procentuální podíl

```
(define (percentage-4 part whole)
```

Příklad: procentuální podíl

```
(define (percentage-4 part whole)
  (if (eq? whole #true)                    ;je-li whole rovno #true
```

Příklad: procentuální podíl

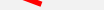
```
(define (percentage-4 part whole)
  (if (eq? whole #true)                    ;je-li whole rovno #true
      (percentage-4 part the-whole)        ;aplikujeme znovu percentage-4
```

Příklad: procentuální podíl

```
(define (percentage-4 part whole)
  (if (eq? whole #true)                    ;je-li whole rovno #true
      (percentage-4 part the-whole)        ;aplikujeme znovu percentage-4
      (* (/ part whole) 100)))             ;jinak výpočet
```

Příklad: procentuální podíl

```
(define (percentage-4 part whole)
  (if (eq? whole #true)
      (percentage-4 part the-whole)
      (* (/ part whole) 100)))
```



rekurzivní aplikace

Bude to fungovat?

Bude to fungovat?

Prostředí během aplikace `(percentage-4 100378 #true)`:

Bude to fungovat?

Prostředí během aplikace `(percentage-4 100378 #true)`:

Počáteční (globální) prostředí

Bude to fungovat?

Prostředí během aplikace `(percentage-4 100378 #true)`:

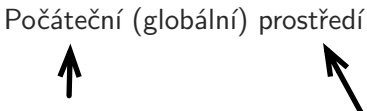
Počáteční (globální) prostředí

Prostředí procedury `percentage-4`
(první aplikace)

symbol	hodnota
`part`	`100378`
`whole`	`#true`

Bude to fungovat?

Prostředí během aplikace `(percentage-4 100378 #true)`:



Prostředí procedury `percentage-4` (první aplikace)

symbol	hodnota
part	100378
whole	#true

Prostředí procedury `percentage-4` (druhá aplikace)

symbol	hodnota
part	100378
whole	10625449

Příklad: prohledávání intervalu

Příklad: prohledávání intervalu

```
(define (square-1? n)
  (if (integer? (sqrt n))
      #true
      #false))
```

Příklad: prohledávání intervalu

```
(define (square-1? n)
  (if (integer? (sqrt n))
      #true
      #false))

(define (square-2? n)
  (integer? (sqrt n)))
```

Příklad: prohledávání intervalu

```
(define (square-1? n)
  (if (integer? (sqrt n))
      #true
      #false))

(define (square-2? n)
  (integer? (sqrt n)))
```

Přesnost?

Příklad: prohledávání intervalu

```
(define (square-1? n)
  (if (integer? (sqrt n))
      #true
      #false))

(define (square-2? n)
  (integer? (sqrt n)))
```

Přesnost?

```
(define (square-3? n)
  (= (sqr (round (sqrt n))) n))
```

Příklad: prohledávání intervalu

Příklad: prohledávání intervalu

```
(define (contains-square-1? a b)
```

Příklad: prohledávání intervalu

```
(define (contains-square-1? a b)
  (if (> a b)                                    ;když je interval prázdný
```

Příklad: prohledávání intervalu

```
(define (contains-square-1? a b)
  (if (> a b)                                ;když je interval prázdný
      #false                                 ;čtverec neobsahuje
```

Příklad: prohledávání intervalu

```
(define (contains-square-1? a b)
  (if (> a b)                                ;když je interval prázdný
      #false                                 ;čtverec neobsahuje
      (if (square-2? a)                      ;je-li dolní konec čtverec
```

Příklad: prohledávání intervalu

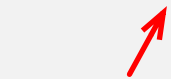
```
(define (contains-square-1? a b)
  (if (> a b)                                ;když je interval prázdný
      #false                                 ;čtverec neobsahuje
      (if (square-2? a)                      ;je-li dolní konec čtverec
          #true                              ;interval čtverec obsahuje
```

Příklad: prohledávání intervalu

```
(define (contains-square-1? a b)
  (if (> a b)                                  ;když je interval prázdný
      #false                                   ;čtverec neobsahuje
      (if (square-2? a)                        ;je-li dolní konec čtverec
          #true                                ;interval čtverec obsahuje
          (contains-square-1? (+ a 1) b))))    ;jinak zkoumáme
                                               ;interval [a + 1, b]
```

Příklad: prohledávání intervalu

```
(define (contains-square-1? a b)
  (if (> a b)                                  ;když je interval prázdný
      #false                                   ;čtverec neobsahuje
      (if (square-2? a)                        ;je-li dolní konec čtverec
          #true                                ;interval čtverec obsahuje
          (contains-square-1? (+ a 1) b))))    ;jinak zkoumáme
                                               ;interval [a + 1, b]
```

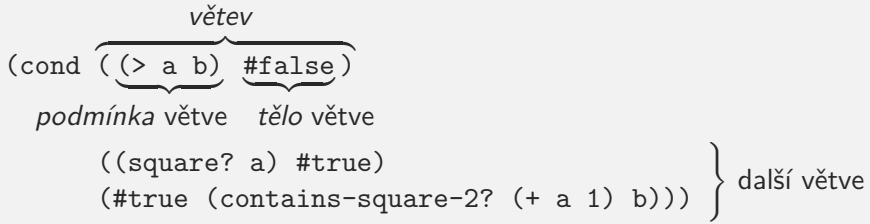


Příklad: prohledávání intervalu

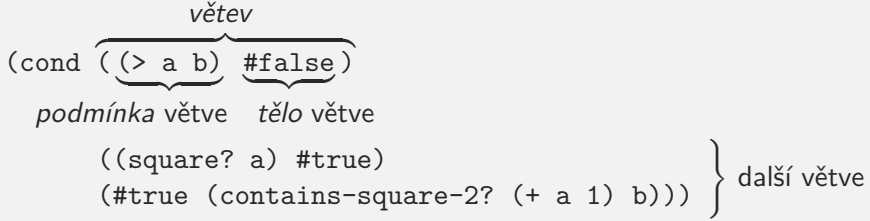
Příklad: prohledávání intervalu

```
(define (contains-square-2? a b)
  (cond ((> a b) #false)
        ((square? a) #true)
        (#true (contains-square-2? (+ a 1) b))))
```

Odbočka: speciální operátor `cond`



Odbočka: speciální operátor `cond`



1. Postupně se vyhodnocují podmínky větví.
2. Jakmile je nějaká splněna, vyhodnotí se tělo příslušné větve.
3. Další podmínky se nevyhodnocují.
4. Vrátí se výsledek vyhodnoceného těla, pokud žádná podmínka nebyla splněna, dojde k chybě.

Prohledávání intervalu

```
(define (contains-square-3? a b)
  (and (<= a b)
       (or (square? a)
           (contains-square-3? (+ a 1) b))))
```

Odbočka: speciální operátory and a or, procedura not

Operátor and

```
(and e1 e2 ... en)
```

Vrací #true, pokud se všechny e_i vyhodnotí na #true, jinak vrací #false. Používá *zkrácené vyhodnocování*.

Operátor or

```
(or e1 e2 ... en)
```

Vrací #true, pokud se některé e_i vyhodnotí na #true, jinak #false. Používá *zkrácené vyhodnocování*.

Procedura not

Procedura not počítá *logickou negaci*.

Pevný bod funkce cos

Pevný bod funkce $\cos$

Hledáme přibližnou hodnotu čísla x takového, že $\cos x = x$.

Pevný bod funkce $\cos$

Hledáme přibližnou hodnotu čísla x takového, že $\cos x = x$.

```
(define (approx-= a b prec)
  (<= (abs (- a b)) prec))
```

Pevný bod funkce $\cos$

Hledáme přibližnou hodnotu čísla x takového, že $\cos x = x$.

```
(define (approx-= a b prec)
  (<= (abs (- a b)) prec))

(define (cos-fixpoint-iter x prec)
  (let ((y (cos x)))
    (if (approx-= x y prec)
        y
        (cos-fixpoint-iter y prec))))
```

Pevný bod funkce cos

Hledáme přibližnou hodnotu čísla x takového, že $\cos x = x$.

```
(define (approx-= a b prec)
  (<= (abs (- a b)) prec))

(define (cos-fixpoint-iter x prec)
  (let ((y (cos x)))
    (if (approx-= x y prec)
        y
        (cos-fixpoint-iter y prec))))

(define (cos-fixpoint prec)
  (cos-fixpoint-iter 0 prec))
```

Obsah

Rekurzivní procedury

Definition (rekurzivní procedura)

Procedura je *rekurzivní*, když ve svém těle obsahuje aplikaci sebe sama.

- je poznat ze zdrojového kódu procedury
- procedury `percentage-4`, `contains-square-1?`, `contains-square-2?`, `contains-square-3?`, `cos-fixpoint-iter` jsou rekurzivní

Rekurzivní výpočetní proces

Definition (rekurzivní výpočetní proces)

Výpočetní proces je *rekurzivní*, když během aplikace procedury dojde znovu k aplikaci téže procedury.

Rekurzivní výpočetní proces

Definition (rekurzivní výpočetní proces)

Výpočetní proces je *rekurzivní*, když během aplikace procedury dojde znovu k aplikaci téže procedury.

- je poznat, když program běží

Rekurzivní výpočetní proces

Definition (rekurzivní výpočetní proces)

Výpočetní proces je *rekurzivní*, když během aplikace procedury dojde znovu k aplikaci téže procedury.

- je poznat, když program běží
- některá aplikace procedury by k aplikaci téže procedury vést neměla (*ukončovací podmínka*)

Rekurzivní výpočetní proces

Definition (rekurzivní výpočetní proces)

Výpočetní proces je *rekurzivní*, když během aplikace procedury dojde znovu k aplikaci téže procedury.

- je poznat, když program běží
- některá aplikace procedury by k aplikaci téže procedury vést neměla (*ukončovací podmínka*)

Speciální případ:

Definition (iterativní výpočetní proces)

Výpočetní proces je *iterativní*, když na konci aplikace procedury dojde opět k aplikaci téže procedury.

Rekurzivní výpočetní proces

Definition (rekurzivní výpočetní proces)

Výpočetní proces je *rekurzivní*, když během aplikace procedury dojde znovu k aplikaci téže procedury.

- je poznat, když program běží
- některá aplikace procedury by k aplikaci téže procedury vést neměla (*ukončovací podmínka*)

Speciální případ:

Definition (iterativní výpočetní proces)

Výpočetní proces je *iterativní*, když na konci aplikace procedury dojde opět k aplikaci téže procedury.

- uvedené procedury generují iterativní výpočetní proces

Obsah

Obecná mocnina

Obecná mocnina

```
(define (power2 a)
  (* a a))
```

Obecná mocnina

```
(define (power2 a)
  (* a a))

(define (power3 a)
  (* a (power2 a)))
```

Obecná mocnina

```
(define (power2 a)
  (* a a))

(define (power3 a)
  (* a (power2 a)))

(define (power4 a)
  (* a (power3 a)))
```

Obecná mocnina

```
(define (power2 a)
  (* a a))

(define (power3 a)
  (* a (power2 a)))

(define (power4 a)
  (* a (power3 a)))

(define (power5 a)
  (* a (power4 a)))
```

Obecná mocnina

```
(define (power2 a)
  (* a a))

(define (power3 a)
  (* a (power2 a)))

(define (power4 a)
  (* a (power3 a)))

(define (power5 a)
  (* a (power4 a)))

(define (power a n)
  (if (= n 0)
      1
      (* a (power a (- n 1)))))
```

Faktoriál

$$n! = \begin{cases} 1 & \text{když } n = 0 \\ n \cdot (n-1)! & \text{když } n > 0 \end{cases}$$

Faktoriál

$$n! = \begin{cases} 1 & \text{když } n = 0 \\ n \cdot (n-1)! & \text{když } n > 0 \end{cases}$$

Napsáno do procedury:

```
(define (fact-1 n)
  (if (= n 0)
      1
      (* n (fact-1 (- n 1)))))
```

Faktoriál — iterativní verze

Faktoriál — iterativní verze

```
(define (fact-2-iter n ir)
  (if (= n 0)
      ir
      (fact-2-iter (- n 1) (* ir n))))
```

Faktoriál — iterativní verze

```
(define (fact-2-iter n ir)
  (if (= n 0)
      ir
      (fact-2-iter (- n 1) (* ir n))))

(define (fact-2 n)
  (fact-2-iter n 1))
```