Paradigmata programování 1

Přednáška 2. Procedury a prostředí

Michal Krupka

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Outline

1 Abstrakce pomocí procedur

2 Vazby

3 Prostředí

4 Vytváření prostředí operátorem `let`

5 Závěr

Abstrakce

Jako činnost (proces)

Myšlenkový proces, kterým předmět zbavujeme zvláštností a odlišností a uvažujeme jen jeho obecné a (pro daný problém) podstatné vlastnosti.

Jako pojem

Výsledek procesu abstrakce, tzv. *abstraktní pojem*. Je dán souhrnem vlastností nějaké množiny objektů a má *název*. (Každý pojem je více nebo méně abstraktní.)

Programová abstrakce

Vytvoření nástroje (např. procedury), který je možné použít bez znalosti technických detailů jeho práce. (Víme *co* dělá, nemusíme vědět, *jak*.)

Abstrakce

Jako činnost (proces)

Myšlenkový proces, kterým předmět zbavujeme zvláštností a odlišností a uvažujeme jen jeho obecné a (pro daný problém) podstatné vlastnosti.

Jako pojem

Výsledek procesu abstrakce, tzv. *abstraktní pojem*. Je dán souhrnem vlastností nějaké množiny objektů a má *název*. (Každý pojem je více nebo méně abstraktní.)

Programová abstrakce

Vytvoření nástroje (např. procedury), který je možné použít bez znalosti technických detailů jeho práce. (Víme *co* dělá, nemusíme vědět, *jak*.)

Abstrakce

Jako činnost (proces)

Myšlenkový proces, kterým předmět zbavujeme zvláštností a odlišností a uvažujeme jen jeho obecné a (pro daný problém) podstatné vlastnosti.

Jako pojem

Výsledek procesu abstrakce, tzv. *abstraktní pojem*. Je dán souhrnem vlastností nějaké množiny objektů a má *název*. (Každý pojem je více nebo méně abstraktní.)

Programová abstrakce

Vytvoření nástroje (např. procedury), který je možné použít bez znalosti technických detailů jeho práce. (Víme *co* dělá, nemusíme vědět, *jak*.)

Abstrakce

Výhody programové abstrakce

- v daný moment řešíme pouze omezené množství problémů
- neřešíme problémy, které zrovna nejsou podstatné
- zvyšuje se možnost znovupoužitelnosti programu
- zvyšuje se čitelnost
- snadněji se dělají změny
- snižuje se chybovost

Abstrakce

Výhody programové abstrakce

- v daný moment řešíme pouze omezené množství problémů
- neřešíme problémy, které zrovna nejsou podstatné
- zvyšuje se možnost znovupoužitelnosti programu
- zvyšuje se čitelnost
- snadněji se dělají změny
- snižuje se chybovost

Abstrakce

Výhody programové abstrakce

- v daný moment řešíme pouze omezené množství problémů
- neřešíme problémy, které zrovna nejsou podstatné
- zvyšuje se možnost znovupoužitelnosti programu
- zvyšuje se čitelnost
- snadněji se dělají změny
- snižuje se chybovost

Abstrakce

Výhody programové abstrakce

- v daný moment řešíme pouze omezené množství problémů
- neřešíme problémy, které zrovna nejsou podstatné
- zvyšuje se možnost znovupoužitelnosti programu
- zvyšuje se čitelnost
- snadněji se dělají změny
- snižuje se chybovost

Abstrakce

Výhody programové abstrakce

- v daný moment řešíme pouze omezené množství problémů
- neřešíme problémy, které zrovna nejsou podstatné
- zvyšuje se možnost znovupoužitelnosti programu
- zvyšuje se čitelnost
- snadněji se dělají změny
- snižuje se chybovost

Abstrakce

Výhody programové abstrakce

- v daný moment řešíme pouze omezené množství problémů
- neřešíme problémy, které zrovna nejsou podstatné
- zvyšuje se možnost znovupoužitelnosti programu
- zvyšuje se čitelnost
- snadněji se dělají změny
- snižuje se chybovost

Definice procedury

```
(define (triangle-area b h)
  (* 1/2 b h))
```

Definice procedury

(*formální*) *parametry* procedury

```
(define (triangle-area b h)
  (* 1/2 b h))
```

tělo procedury

Každá procedura si pamatuje:

- své parametry,
- své tělo
- a ještě jednu informaci (zjistíme později).

Aplikace uživatelsky definované procedury

Vyhodnocujeme například

```
> (triangle-area (+ 5 7) (- 4 3))
```

Co se bude dít při aplikaci procedury `triangle-area` na čísla 12 a 1?

Připomenutí:

```
(define (triangle-area b h)
  (* 1/2 b h))
```

1. Zařídí se, aby symbol `b` měl hodnotu 12 a symbol `h` měl hodnotu 1.
2. Vyhodnotí se tělo procedury.
3. Výsledkem aplikace bude výsledek tohoto vyhodnocení.

Aplikace uživatelsky definované procedury

Vyhodnocujeme například

```
> (triangle-area (+ 5 7) (- 4 3))
```

Co se bude dít při aplikaci procedury `triangle-area` na čísla 12 a 1?

Připomenutí:

```
(define (triangle-area b h)
  (* 1/2 b h))
```

1. Zařídí se, aby symbol `b` měl hodnotu 12 a symbol `h` měl hodnotu 1.
2. Vyhodnotí se tělo procedury.
3. Výsledkem aplikace bude výsledek tohoto vyhodnocení.

Aplikace uživatelsky definované procedury

Vyhodnocujeme například

```
> (triangle-area (+ 5 7) (- 4 3))
```

Co se bude dít při aplikaci procedury `triangle-area` na čísla 12 a 1?

Připomenutí:

```
(define (triangle-area b h)
  (* 1/2 b h))
```

1. Zařídí se, aby symbol `b` měl hodnotu 12 a symbol `h` měl hodnotu 1.
2. Vyhodnotí se tělo procedury.
3. Výsledkem aplikace bude výsledek tohoto vyhodnocení.

Aplikace uživatelsky definované procedury

Vyhodnocujeme například

```
> (triangle-area (+ 5 7) (- 4 3))
```

Co se bude dít při aplikaci procedury `triangle-area` na čísla 12 a 1?

Připomenutí:

```
(define (triangle-area b h)
  (* 1/2 b h))
```

1. Zařídí se, aby symbol b měl hodnotu 12 a symbol h měl hodnotu 1.
2. Vyhodnotí se tělo procedury.
3. Výsledkem aplikace bude výsledek tohoto vyhodnocení.

Aplikace uživatelsky definované procedury

Vyhodnocujeme například

```
> (triangle-area (+ 5 7) (- 4 3))
```

Co se bude dít při aplikaci procedury `triangle-area` na čísla 12 a 1?

Připomenutí:

```
(define (triangle-area b h)
  (* 1/2 b h))
```

1. Zařídí se, aby symbol `b` měl hodnotu `12` a symbol `h` měl hodnotu `1`.
2. Vyhodnotí se tělo procedury.
3. Výsledkem aplikace bude výsledek tohoto vyhodnocení.

Aplikace uživatelsky definované procedury

Vyhodnocujeme například

```
> (triangle-area (+ 5 7) (- 4 3))
```

Co se bude dít při aplikaci procedury `triangle-area` na čísla 12 a 1?

Připomenutí:

```
(define (triangle-area b h)
  (* 1/2 b h))
```

1. Zařídí se, aby symbol `b` měl hodnotu `12` a symbol `h` měl hodnotu `1`.
2. Vyhodnotí se tělo procedury.
3. Výsledkem aplikace bude výsledek tohoto vyhodnocení.

Aplikace uživatelsky definované procedury

Vyhodnocujeme například

```
> (triangle-area (+ 5 7) (- 4 3))
```

Co se bude dít při aplikaci procedury `triangle-area` na čísla 12 a 1?

Připomenutí:

```
(define (triangle-area b h)
  (* 1/2 b h))
```

1. Zařídí se, aby symbol `b` měl hodnotu `12` a symbol `h` měl hodnotu `1`.
2. Vyhodnotí se tělo procedury.
3. Výsledkem aplikace bude výsledek tohoto vyhodnocení.

Outline

1 Abstrakce pomocí procedur

2 Vazby

3 Prostředí

4 Vytváření prostředí operátorem `let`

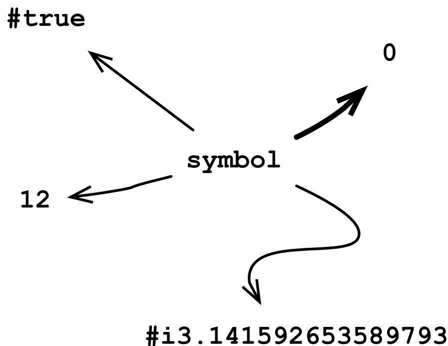
5 Závěr

Zajímavá otázka

```
> (define r 0)
> (define (circle-area r)
    (* pi r r))
> (circle-area 10)
#i314.159265358979
> r
???
```

Vazby

- Každý symbol může mít více *vazeb*.
- Jedna vazba může být *aktuální*. Ta *zastiňuje* ostatní (na obr. tučně).
- Každá vazba má *hodnotu*.
- Hodnotou symbolu je vždy hodnota jeho aktuální vazby.

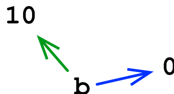


Vazby

Zpět k zajímavé otázce:

```
> (define r 0)
> (define (circle-area r)
    (* pi r r))
> (circle-area 10)
#i314.1592653589793
> r
???
```

Symbol r má dvě vazby:



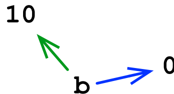
V jazyce Scheme je zařízeno, že v těle procedury circle-area je aktuální první vazba, v okně interakcí druhá.

Vazby

Zpět k zajímavé otázce:

```
> (define r 0)
> (define (circle-area r)
    (* pi r r))
> (circle-area 10)
#i314.159265358979 3
> r
???
```

Symbol r má dvě vazby:



V jazyce Scheme je zařízeno, že v těle procedury circle-area je aktuální první vazba, v okně interakcí druhá.

Vazby

```
> (define (circle-area r)
    (* 2 pi r))
> (circle-area 10)
#i62.83185307179586
```

Při aplikaci procedury `circle-area`:

1. se vytvoří nová vazba na symbol `r`,
2. učiní se aktuální (tím zastíní původní vazbu),
3. nastaví se jí hodnota 10.
4. Vyhodnotí se tělo procedury.
5. Po vyhodnocení vazba přestává být aktuální.

Vazba na symbol `pi` se nevytváří a nenastavuje, aktuální zůstává původní vazba. Totéž platí ještě pro jiný symbol (který?).

Vazby

```
> (define (circle-area r)
    (* 2 pi r))
> (circle-area 10)
#i62.83185307179586
```

Při aplikaci procedury `circle-area`:

1. se vytvoří nová vazba na symbol `r`,
2. učiní se aktuální (tím zastíní původní vazbu),
3. nastaví se jí hodnota 10.
4. Vyhodnotí se tělo procedury.
5. Po vyhodnocení vazba přestává být aktuální.

Vazba na symbol `pi` se nevytváří a nenastavuje, aktuální zůstává původní vazba. Totéž platí ještě pro jiný symbol (který?).

Vazby

```
> (define (circle-area r)
    (* 2 pi r))
> (circle-area 10)
#i62.83185307179586
```

Při aplikaci procedury `circle-area`:

1. se vytvoří nová vazba na symbol `r`,
2. učiní se aktuální (tím zastíní původní vazbu),
3. nastaví se jí hodnota 10.
4. Vyhodnotí se tělo procedury.
5. Po vyhodnocení vazba přestává být aktuální.

Vazba na symbol `pi` se nevytváří a nenastavuje, aktuální zůstává původní vazba. Totéž platí ještě pro jiný symbol (který?).

Vazby

```
> (define (circle-area r)
    (* 2 pi r))
> (circle-area 10)
#i62.83185307179586
```

Při aplikaci procedury `circle-area`:

1. se vytvoří nová vazba na symbol `r`,
2. učiní se aktuální (tím zastíní původní vazbu),
3. nastaví se jí hodnota 10.
4. Vyhodnotí se tělo procedury.
5. Po vyhodnocení vazba přestává být aktuální.

Vazba na symbol `pi` se nevytváří a nenastavuje, aktuální zůstává původní vazba. Totéž platí ještě pro jiný symbol (který?).

Vazby

```
> (define (circle-area r)
    (* 2 pi r))
> (circle-area 10)
#i62.83185307179586
```

Při aplikaci procedury `circle-area`:

1. se vytvoří nová vazba na symbol `r`,
2. učiní se aktuální (tím zastíní původní vazbu),
3. nastaví se jí hodnota 10.
4. Vyhodnotí se tělo procedury.
5. Po vyhodnocení vazba přestává být aktuální.

Vazba na symbol `pi` se nevytváří a nenastavuje, aktuální zůstává původní vazba. Totéž platí ještě pro jiný symbol (který?).

Vazby

```
> (define (circle-area r)
    (* 2 pi r))
> (circle-area 10)
#i62.83185307179586
```

Při aplikaci procedury `circle-area`:

1. se vytvoří nová vazba na symbol `r`,
2. učiní se aktuální (tím zastíní původní vazbu),
3. nastaví se jí hodnota 10.
4. Vyhodnotí se tělo procedury.
5. Po vyhodnocení vazba přestává být aktuální.

Vazba na symbol `pi` se nevytváří a nenastavuje, aktuální zůstává původní vazba. Totéž platí ještě pro jiný symbol (který?).

Vazby

```
> (define (circle-area r)
    (* 2 pi r))
> (circle-area 10)
#i62.83185307179586
```

Při aplikaci procedury `circle-area`:

1. se vytvoří nová vazba na symbol `r`,
2. učiní se aktuální (tím zastíní původní vazbu),
3. nastaví se jí hodnota 10.
4. Vyhodnotí se tělo procedury.
5. Po vyhodnocení vazba přestává být aktuální.

Vazba na symbol `pi` se nevytváří a nenastavuje, aktuální zůstává původní vazba. Totéž platí ještě pro jiný symbol (který?).

Vazby

```
> (define (circle-area r)
    (* 2 pi r))
> (circle-area 10)
#i62.83185307179586
```

Při aplikaci procedury `circle-area`:

1. se vytvoří nová vazba na symbol `r`,
2. učiní se aktuální (tím zastíní původní vazbu),
3. nastaví se jí hodnota `10`.
4. Vyhodnotí se tělo procedury.
5. Po vyhodnocení vazba přestává být aktuální.

Vazba na symbol `pi` se nevytváří a nenastavuje, aktuální zůstává původní vazba. Totéž platí ještě pro jiný symbol (který?).

Vazby

```
> (define (circle-area r)
    (* 2 pi r))
> (circle-area 10)
#i62.83185307179586
```

Při aplikaci procedury `circle-area`:

1. se vytvoří nová vazba na symbol `r`,
2. učiní se aktuální (tím zastíní původní vazbu),
3. nastaví se jí hodnota 10.
4. Vyhodnotí se tělo procedury.
5. Po vyhodnocení vazba přestává být aktuální.

Vazba na symbol `pi` se nevytváří a nenastavuje, aktuální zůstává původní vazba. Totéž platí ještě pro jiný symbol (který?).

Outline

1 Abstrakce pomocí procedur

2 Vazby

3 Prostředí

4 Vytváření prostředí operátorem `let`

5 Závěr

Prostředí

Vazby jsou organizovány v *prostředí*. To chápeme jako tabulku, ze které Scheme zjišťuje vazby symbolů a jejich hodnoty. Řádky v tabulce jsou vazby. Například:

Globální prostředí

symbol	hodnota
pi	#i3.141592653589793
+	#<procedure:+>
-	#<procedure:->
*	#<procedure:*>
/	#<procedure:/>
sqrt	#<procedure:sqrt>
⋮	⋮

Prostředí

Vazby jsou organizovány v *prostředí*. To chápeme jako tabulku, ze které Scheme zjišťuje vazby symbolů a jejich hodnoty. Řádky v tabulce jsou vazby. Například:

Globální prostředí

symbol	**hodnota**
`pi`	`#i3.141592653589793`
`+`	`#<procedure:+>`
`-`	`#<procedure:->`
`*`	`#<procedure:*>`
`/`	`#<procedure:/>`
`sqrt`	`#<procedure:sqrt>`
⋮	⋮

Prostředí

Po vyhodnocení

```
> (define r 0)
> (define (circle-area r)
    (* pi r r))
```

Globální prostředí

symbol	hodnota
circle-area	#<procedure:circle-area>
r	0
pi	#i3.141592653589793
+	#<procedure:+>
-	#<procedure:->
*	#<procedure:*>
/	#<procedure:/>
sqrt	#<procedure:sqrt>
⋮	⋮

Speciální operátor `define` (znovu)

Vyhodnocení výrazu (`define` a b)

1. Vyhodnotí se b.
2. V globálním prostředí se vytvoří nová vazba na symbol a.
3. Získaná hodnota výrazu b se učiní hodnotou této vazby.

Prostředí

```
> (define (circle-area r)
    (* pi r r))
> (circle-area 10)
#i314.1592653589793
```

Prostředí
procedury `circle-area`

symbol	hodnota
`r`	10

To nestačí. Proto: předek prostředí.

Prostředí

Globální prostředí

symbol	hodnota
`circle-area`	`#<procedure:circle-area>`
`pi`	`#i3.141592653589793`
`+`	`#<procedure:+>`
`-`	`#<procedure:->`
`*`	`#<procedure:*>`
⋮	⋮

Prostředí
procedury `circle-area`

symbol	hodnota
`r`	`10`

Globální prostředí je předkem prostředí procedury `circle-area`. Každé prostředí kromě globálního má předka.

Znovu o vyhodnocování

Výrazy se vždy vyhodnocují v prostředí.
Aktuální prostředí je prostředí, ve kterém je výraz vyhodnocován.

Prvky složeného výrazu se (až na stanovené výjimky) vyhodnocují ve stejném prostředí jako původní složený výraz.

Vyhodnocování symbolu

Při vyhodnocování symbolu se nejprve hledá jeho vazba v aktuálním prostředí. Pokud je nalezena, hodnotou symbolu je hodnota vazby. Pokud není nalezena, vazba se hledá v předkovi aktuálního prostředí. Tak se pokračuje tak dlouho, dokud se neskončí v globálním prostředí. Pokud ani tam není vazba nalezena, dojde k chybě.

Aplikace uživatelské procedury

Aplikace uživatelské procedury

Při aplikaci uživatelské procedury se vytvoří nové prostředí s vazbami parametrů na hodnoty argumentů. Předkem tohoto prostředí se stane globální prostředí (toto později zobecníme). V tomto prostředí se pak vyhodnotí tělo procedury. Nové prostředí se vytváří při každé aplikaci znovu.

Outline

1 Abstrakce pomocí procedur

2 Vazby

3 Prostředí

4 Vytváření prostředí operátorem `let`

5 Závěr

Speciální operátor `let`

```
> (let ((a 2)
        (b (+ 1 2)))
    (* a b))
> 6
```

let: terminologie

popis prostředí

```
(let ((a 2) (b (+ 1 2)))
  (* a b))
```

popis vazby — `(a 2)`; *popis vazby* — `(b (+ 1 2))`; *tělo* — `(* a b)`

Popis prostředí Seznam libovolné délky. Jeho prvky jsou *popisy vazeb*.

Popis vazby Dvouprvkový seznam. Na prvním místě má symbol, na druhém libovolný výraz.

Tělo Libovolný výraz.

let: vyhodnocení

```
           popis prostředí
(let ( (a 2)  (b (+ 1 2) ))
     popis vazby  popis vazby
  (* a b))
    tělo
```

1. V aktuálním prostředí se vyhodnotí všechny druhé položky popisů vazeb.
2. Vytvoří se nové prostředí a v něm vazby tak, že každá první položka popisu vazby (která musí být symbolem) se naváže na hodnotu druhé položky.
3. Předkem nového prostředí se učiní aktuální prostředí.
4. Tělo se vyhodnotí v tomto novém prostředí. Výsledek se vrátí jako hodnota celého výrazu.

Outline

1 Abstrakce pomocí procedur

2 Vazby

3 Prostředí

4 Vytváření prostředí operátorem `let`

5 Závěr

Pojmy k zapamatování

Primitivní a uživatelsky definovaná procedura, parametry a tělo procedury, vazba, aktuální vazba, zastínění vazby, hodnota vazby, prostředí, globální (počáteční) prostředí, aktuální prostředí, vyhodnocení výrazu v prostředí, předek prostředí; popis prostředí, popis vazby a tělo pro operátor `let`.