Předdefinované formy jazyka Scheme

verze 1 z 24.11.2002

Rudolf KRYL

Součástí přednášek z neprocedurálního programování je i **krátké extempore o programování v jazyce LISP**. Používáme k tomu **dialektu LISP**, který se jmenuje **Scheme**.

Výklad na přednáškách **vychází z knihy** H.Abelsona a G.J.Sussmana Structure and Interpretation of Computer Programs (přesná citace je v seznamu literatury k příslušným předmětům), která byla na MIT používána v základním kursu programování.

V tomto materiálu se **popisují** jen ty **předdefinované formy jazyka Scheme**, které budeme na přednášce a na seminářích potřebovat. Z jazyka Scheme jsme **vynechali** *lokální formu define* a **nahradili** *apostrof* **obvyklejší** formou *quote*.

Tento text je pouze o něco rozšířeným přehledem standardních forem a nepokouší se být ucelenou učebnicí.

1. Forma define

Pomocí formy **define** můžeme **pojmenovat funkce** resp. **hodnoty** (ty můžeme považovat za nulární funkce).

Obecný tvar je

(define (<jméno> <formální parametry>) < tělo>)

Příklady použití :

```
( define dve 2 )
( define ( plus4 x ) ( + x 4 ) )
( define ( natreti x ) ( * x x x ) )
( define ( sectikrychle x y ) ( + ( natreti x ) ( natreti y ) ) )
```

2. Podmíněn výraz - forma cond

Pomocí formy **cond** můžeme **vytvářet podmíněné výrazy**.

Pokud máme nějaký výraz interpretovat jako **predikát** (nabývající hodnot pravda/nepravda), pak

- (konstatní) **výraz nil je** interpretován jako **nepravda** a
- **každý jiný výraz jako pravda**

(nil tedy hraje obdobnou roli jako 0 v jazyku C).

Obecný tvar výrazu definovaného pomocí formy cond má je

(cond (<p_1> <e_1>)
(<p_2> <e_2>)
...
(<p_n> <e_n>))

kde pro i=1..n jsou p_i výrazy, které interpretujeme jako predikáty a v_i výrazy. Postupně vyhodnocuje predikáty p_1, p_2, … až do doby, kdy první z nich – dejme tomu p_j - je pravdivý. Hodnotu výrazu pak určuje příslušný výraz e_j . Pokud žádný z predikátů není pravdivý, je hodnotou definovaného výrazu `nil` .

Příklady použití :

```
(define ( abs x )
  ( cond  (   ( > x 0 ) x )
              ( ( = x 0 ) 0 )
              ( ( < x 0 ) ( - x ) )    ) )
```

3. Podmíněn výraz - forma if

Forma **if** je funkcionální obdobou podmíněného příkazu. Obecný tvar je :

(if <podmínka> <důsledek> <alternativa>)

Vyhodnotí se výraz <podmínka> a interpretuje se jako predikát, pokud je pravdivý, určuje hodnotu výraz <důsledek>. Pokud je výraz <podmínka> nepravdivý, určuje hodnotu výrazu výraz <alternativa> .

Příklady použití :

```
(define ( abs x )
   ( if  ( < x 0 ) (- x)  x ) )
```

4. Forma lambda

Jedním ze zdrojů LISPu byl tzv. **λ-kalkul** . Odtud název formy, která umožňuje vyhnout se nutnosti pojmenovávat každou funkci, se kterou chceme pracovat. Lambda forma vyrábí z funkce „její jméno“. Oceníme to především chceme-li rozlišovat mezi parametry a proměnnými funkcí.

Obecný tvar je

(lambda (< formální parametry > < tělo >) .

Příklady použití :

```
( lambda ( x ) ( + x 5 ) )      „funkce plus 5“  jedné proměnné
( lambda ( x y ) ( + x y ) )    „funkce plus “    dvou proměnných
( lambda ( x ) ( + x y ) )      „funkce plus y“  jedné proměnné,
                                      která přičítá k proměnné hodnotu parametru y
```

hodnoty následujících výrazů

```
( ( lambda ( x ) ( + x 5 ) ) ( 13 )          „=“ 18
( lambda ( x y ) ( + x y ) ) ( 5 12 )        „=“ 17
( lambda ( x ) ( + x y ) ) ( 4 )             „=“ y  .
```

Následující dva výrazy jsou ekvivalentní :

```
( define ( plus4 x ) ( + x 4 ) )
( define  plus4  ( lambda (x ) ( + x 4 ) ) )
```

5. Forma let

Forma let umožňuje mimo jiné lokální označení nějaké hodnoty. Význam má především pro efektivitu programů. Obecný tvar vypadá následovně :

(let ((<var_1> <exp_1>)
(<var_2> <exp_2>)
……
(<var_n> <exp_n>))
< tělo >)

kde var_i jsou proměnné a exp_i výrazy. Význam formy spočívá v tom, že ve výrazu tělo označuje každá z proměnných var_i hodnotu výrazu exp_i .

Ekvivalentně lze tedy předcházející definici zapsat jako

```
(  (lambda ( (  <var1> <var2>  …  <varn>  )
               <  tělo > )
<exp1> <exp2> … <expn> )
```

Vidíme tedy, že bychom se bez formy let obešli. Je třeba si uvědomit, že „substituce“ definované v příkazu let platí až ve výrazu < tělo >, ne ve výrazech <exp_i> .

Příklady použití :

```
( define ( f x y )
    let (   ( a ( + 1 ( * x y ) ) )
            ( b ( - 1 y ) )          )
        ( +  ( * x ( square a ) )
             ( * y b )
             ( * a b )            ) ) )
```

6. Dvojice

Dvojice jsou zadány pomocí dvou selektorů *car* a *cdr ;a konstruktoru cons* .

Pokud definujeme

```
( define x ( cons 5 3 ) )
```

pak výraz

`( car x )` má hodnotu `5`
`( cdr x )` má hodnotu `3`

Téměř všechny dialekty mají definovány funkce jako *caddr, cdar, cddadr* následujícím způsobem :

```
( define ( caddr x ) ( car ( cdr ( cdr x ) ) ) )
( define ( cdar x )  ( cdr ( car) )            )
( define ( cddadr x ) ( cdr ( cdr ( car ( cdr x ) ) ) ) )
```

7. Seznamy

Obdobně jako v Prologu **jsou** i v Lispu **seznamy definovány** rekursivně **pomocí dvojic**.

Prázdný seznam je reprezentován konstantou nil ,
neprázdný seznam je dvojice hlava (první prvek) a ocas (seznam zbylých prvků).

Seznam obsahující prvky 1, 2 , 3 a 4 je tedy reprezentován výrazem

```
( cons 1 ( cons 2 ( cons 3 ( cons 4 nil ) ) ) )
```

Existuje však i **forma list**, která z výčtu prvků vyrábí příslušný seznam. V našem případě

```
( list 1 2 3 4 )
```
dá stejný výsledek

Příklady použití :

```
( cadr ( list 1 2 3 4 5 ) )        má hodnotu  2
( cdddr ( list 4 5 6 7 8 ) )       má hodnotu  ( list 7 8 )
```

8. Forma quote

V dětství jste se možná setkali s tím, že se na Vás někdo obrálil slovy: „Řekni jak se jmenuješ“ . Pravděpodobně jste mu to prozradili – např. „Pepík Vomáčka“. Byl-li ošklivý (nebo matfyzák – nebo dokonce obojí zároveň), pokáral Vás, že neumíte ani opakovat jednoduchou větu. Prý chtěl, abyste opakovali větu , kterou řekl. Očekával, že odpovíte „Jak se jmenuješ“ . Pokud byste odpověděli takto, pokáral by Vás také, že chtěl vědět, jak se jmenujete. To je samozřejmě lapálie.

V logice (kde sebereflexe je velmi důležitým postupem) je pochopitelně potřeba rozlišovat mezi těmito dvěma možnostmi. Jinak bychom se mohli dopouštět vážných chyb v důkazech.

Forma quote vrací „výraz sám“ aniž by došlo k jeho vyčíslení.

Příklady použití :

na	`( define a 1 )`	odpoví interpret	`a`
na	`( define b 2 )`	odpoví interpret	`b`
na	`( list a b )`	odpoví interpret	`( 1 2 )`
na	`( list ( quote a ) b )`	odpoví interpret	`( a 2 )`
na	`( list ( quote a ) ( quote b ) )`	odpoví interpret	`( a b )`
na	`( car ( quote ( a b c ) )`	odpoví interpret	`a`