KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Michal Krupka
krupka.inf.upol.cz
michal.krupka@upol.cz
17. listopadu 12, 771 46 Olomouc

Paradigmata programování 1 ⋄ poznámky k přednášce

1. Symbolické výrazy

verze z 25. září 2024

1 Úvod

Cílem čtyřsemestrálního kurzu Paradigmata programování je seznámit studenty s principy programování a základními přístupy a styly v programování používanými (tzv. *paradigmaty*). Budeme to dělat bez ohledu na to, jaké přístupy se v současné době zrovna používají v praxi — zkušenost říká, že ty se rychle mění a vyvíjejí, zatímco principy zůstávají mnoho desetiletí stále stejné.

Jedině programátor, který rozumí principům, na kterých je programování a programovací jazyky postaveno, je schopen programovat kvalitně.

K výukovým účelům budeme používat vlastní verzi jazyka Lisp, která je velmi zjednodušenou variantou jazyka Common Lisp. Programovat budeme v aplikaci ***PP Polyglot***, která je pro potřeby kurzu vyvíjena na naší katedře.

2 Lisp jako kalkulačka

Po spuštění *PP Polyglotu* se otevře okno, kterému říkáme **Listener** (*posluchač*). V něm vstupujeme do kontaktu s aplikací. (Často budu, i když nepřesně, říkat, že s jazykem Lisp.)

Okno můžeme používat třeba na počítání s čísly, podobně jako kalkulačku (ale i na další věci, o kterých se dozvíme později). Napíšeme do něj, co chceme vypočítat, a dostaneme výsledek:

```
> 1
1

> (+ 1 1)
2
```

Nejprve jsme zkusili vypočítat číslo 1. To můžeme chápat jako už vypočítanou hodnotu, takže není divu, že se nám jako výsledek vrátilo číslo 1.

Jako druhý pokus jsme vypočítali číslo $1+1$. Museli jsme ovšem výpočet zadat ve tvaru, kterému jazyk Lisp rozumí: nejprve napsat operaci, kterou chceme, aby provedl, a pak čísla, se kterými ji má provést. Vše uzavřít do kulatých závorek.

Další příklady:

```
> 1
1

> (+ 1 1)
2

> (/ 4 2)
2

> (- 4 2)
2

> (* 4 2)
8

> (+ 1 2 3 4)
10

> (* 1 2 3 4)
24

> (/ 24 6 2)
2

> (+ (/ -4 2) (* -4 2))
-10

> (- 5)
-5

> (- -5)
5

> (+)
0

> (*)
1
```

Vidíme, že kromě sčítání můžeme i odečítat, násobit a dělit. Můžeme to dělat i s více než dvěma čísly. A můžeme i dělat složitější výpočty s více operacemi.

Čtvrtý a třetí pokus od konce ukazují použití operace `-` k výpočtu opačného čísla k zadanému. Jde vlastně o zkratky k zadání `(- 0 5)` a `(- 0 -5)`.

Poslední dva pokusy ukazují, že Lisp umí sečíst a vynásobit i nulový počet čísel. Můžete se sami zamyslet, proč jako výsledek uvádí právě nulu a jedničku.

Kromě celých čísel můžeme pracovat i s čísly racionálními a obecně reálnými:

```
> (/ 4 6)
2/3

> (/ 189 216)
7/8

> (/ 123)
1/123

> (/ 1/123)
123

> pi
3.141592653589793D0
```

Poslední ukázka nás poučí, že v Lispu můžeme používat číslo π tak, že napíšeme jeho anglický název. Znaků `D0` na konci čísla si nemusíme všímat, za chvilku se dozvíme, co znamenají. Jinak, jak vidíme, v zápisu desetinného rozvoje se používá desetinná tečka místo čárky.

Takto bychom tedy mohli vypočítat obvod kružnice o poloměru `r`:

```
> (* 2 pi r)
```

... kdyby ovšem Listener věděl, kolik je `r`. To on neví, takže nám nahlásí chybu:

```
Error: Unbound variable R.
```

Chybová zpráva `Error: Unbound variable R` na prvním řádku znamená, že Lisp nezná hodnotu `R`. (Jak jste si teď mohli všimnout, Lisp nerozlišuje, zda je symbol `r` psán malými, nebo velkými písmeny, a sám je píše velkými.)

Tak si aspoň vypočítáme obvod kružnice o poloměru 10:

```
> (* 2 pi 10)
62.83185307179586D0
```

Když už jsme u chyb, ukážeme si další pokus o výpočet, který vyvolá chybu:

```
> (/ (+ 1 1) (- 1 1))
Error in function /: Division by zero with argument 2.
```

Chybové hlášení říká: došlo k chybě, protože Listener měl dělit nulou (Division by zero). Stalo se to při pokusu o dělení čísla 2.

A můžeme pokračovat. Kromě operací s čísly můžeme používat i matematické funkce, například odmocninu (*square root*, náš název je `sqrt`) a goniometrické funkce:

```
> (sqrt 16)
4.0

> (sqrt 2)
1.4142135

> (sqrt -1)
#C(0.0 1.0)

> (sin (/ pi 2))
1.0D0
```

Jak vidíte, Lisp umí vypočítat i odmocninu z -1. Komplexní číslo i zapsal jako `#C(0.0 1.0)`. Některé výsledky ale samozřejmě nevypočítal přesně, jak se můžeme snadno přesvědčit:

```
> (* (sqrt 2) (sqrt 2))
1.9999999
```

Číslo $\sin \pi$ (které je, jak víme, rovno nule) Lisp také nevypočte přesně:

```
> (sin pi)
1.2246467991473532D-16
```

Znaky `D-16` na konci ukazují, že jde o zápis čísla v *exponenciálním tvaru*. Výsledek je zhruba $1{,}22 \cdot 10^{-16}$, tedy 0,0000000000000000122.

Ve škole jsme se učili, že $\sin \frac{\pi}{6} = \frac{1}{2}$. Jestlipak to v našem Listeneru vyjde stejně?

```
> (sin (/ pi 6))
0.49999999999999995D0
```

Vidíme, že (téměř) ano. A poslední příklad: Víme, že $\sin \frac{\pi}{4} = \frac{\sqrt{2}}{2}$, takže když ho umocníme na 2 (neboli vynásobíme sebou samým), měli bychom dostat $\frac{1}{2}$:

```
> (* (sin (/ pi 4)) (sin (/ pi 4)))
0.4999999999999999D0
```

Obecně platí, že pokud je číslo zapsáno s desetinnou tečkou (ať už s koncovým „`D`“, nebo ne), nesmíme se spoléhat, že jde o přesný výsledek. Celá čísla zapsaná bez tečky a zlomky jsou přesné.[1]

Seznam základních číselných funkcí, které *PP Polyglot* zná, najdete na konci textu. Pokus o použití funkce, kterou nezná, vede k chybě:

```
(funkce 10)
Error: Undefined function FUNKCE called with arguments (10).
```

3 Symbolické výrazy

Do okna Listeneru píšeme tzv. symbolické výrazy. *Symbolický výraz* je, jednoduše řečeno, text, kterému Listener rozumí a který pochopí jako zadání výpočtu. Nyní si řekneme, jak správně utvořený symbolický výraz vypadá.

Symbolický výraz (expression) je

- jednoduchý výraz, neboli *atom*, nebo
- složený výraz, neboli *seznam*.

Jednoduchý výraz (atom) je

- číslo nebo
- symbol nebo
- ... (o dalších možnostech si řekneme později)

Číslo je zapsané jedním ze způsobů, které jsme již uvedli, např. `10`, `-5`, `2/3`, `0.0`, `0.4999999999999999D0`, `1.2246063538223773D-16`, `#C(0.0 1.0)` a podobně.

Symbol je posloupnost písmen a případně čísel a dalších znaků, která neoznačuje číslo. Symboly neobsahují některé nepovolené znaky, například dvojtečku, `#`, kulaté závorky, uvozovky, mezery a další prázdné znaky. Symboly jsou například `pi`, `r`, `a1`, `sqrt`, `+`, `/`, `1+1`. Například `-5` symbol není, protože to je číslo.

Složený výraz, seznam je posloupnost jednoho nebo více výrazů. Těmto výrazům říkáme **prvky složeného výrazu (seznamu)** Jsou odděleny mezerami

[1] Různé zápisy čísel v Lispu (a v ostatních programovacích jazycích) souvisejí s tím, že počítač si čísla v paměti ukládá různým způsobem a různě s nimi pracuje. Podrobnosti o tom se dozvíte v jiném předmětu.

nebo jinými prázdnými znaky. Složený výraz začíná levou a končí pravou kulatou závorkou.

Příklady složených výrazů:

- `(+ 1 2)`
- `(+)`
- `(1 2 3)`
- `(1 + 2 + 3)`
- ```
  (* (sin (/ pi 4))
     (sin (/ pi 4)))
  ```

Předposlední výraz je správně utvořený složený symbolický výraz, přestože nejde o správně utvořené zadání výpočtu. Poslední složený výraz má tři prvky, některé z nich jsou složené výrazy.

Toto *nejsou* symbolické výrazy:

- `)`
- `(+ (- 10 11)`
- `ah oj`

Abychom mohli v Lispu pracovat, musíme umět složené symbolické výrazy vytvářet. Vysvětlím to na příkladu výrazů vyjadřujících matematický výpočet.

Vytváření složených výrazů

První symbol v seznamu označuje **název** výrazu (např. matematického: součet, rozdíl, součin, podíl). Označuje tedy **poslední** operaci prováděnou ve výpočtu.

Například výraz

$$\frac{5-3}{5+3}$$

je podíl rozdílu a součtu:

```
(/ (- 5 3) (+ 5 3))
```

Při výpočtu se tedy podíl provede jako poslední operace.

Pro přehlednost můžeme výraz rozdělit na více řádků (znak pro ukončení řádku je stejně jako mezera prázdný znak):
```

```
(/ (- 5 3)
   (+ 5 3))
```

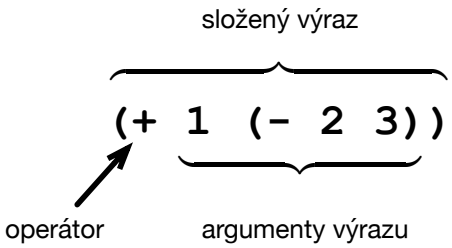
Výraz

$$\sin\left(\frac{5-3}{5+3}\cdot\pi\right)$$

je sinus součinu dříve uvedeného podílu a čísla π

```
(sin (* (/ (- 5 3) (+ 5 3))
        pi))
```

Terminologie pro složené výrazy (seznamy)



Operátor: *co* se má udělat

Argumenty výrazu: *s čím* se to má udělat

V tomto výrazu je tedy operátorem symbol `+` a argumenty číslo `1` a výraz `(- 2 3)`.

Terminologii byste měli dobře ovládat, abyste chápali, o čem se hovoří nebo píše, a abyste se sami o programování v Lispu mohli vyjadřovat.

Prefixová notace

Lisp důsledně používá tzv. **prefixovou notaci**: operátor je vždy uveden *před* ostatními výrazy. To je velké zjednodušení proti ostatním programovacím jazykům i matematice:

prefix: $\sin x$, -5

infix: $x+y\cdot z-3$, operátory $+, \cdot, -$ jsou uprostřed (komplikace: přednost operací)

postfix: $10\,!$, operátor ! je na konci

a další: x^y, $\sqrt[3]{10}$, $\frac{5}{6}$, $\bar{z}$, $|z|$, $\int_a^b 2x\,\mathrm{d}x$

4 Vyhodnocování symbolických výrazů

Když napíšeme do Listeneru symbolický výraz (a stiskneme Enter), dostaneme výsledek (samozřejmě pokud nedojde k chybě). Tomuto výsledku říkáme *hodnota symbolického výrazu*. Například

Číslo $\frac{1}{4}$ je **hodnotou symbolického výrazu** `(/ (- 5 3) (+ 5 3))`.

Počítač k hodnotě výrazu dojde přesně popsaným procesem, kterému se říká **vyhodnocovací proces**. Následuje jeho (zatím zjednodušený) popis:

Vyhodnocení výrazu E

Je-li E **symbol,** výsledkem je **hodnota** symbolu E.

Je-li E **jiný atom než symbol,** výsledkem je E.

Je-li E **seznam s operátorem** o **a dalšími výrazy** $a_1 \dots a_n$**,** pak o musí být symbol a

1. zjistí se **funkce** f, kterou operátor o označuje,
2. zjistí se hodnoty $v_1 \dots v_n$ výrazů $a_1 \dots a_n$ (opět vyhodnocovacím procesem).
3. Výsledkem je výsledek **aplikace** funkce f na hodnoty $v_1 \dots v_n$.

V popisu zjednodušeného vyhodnocovacího procesu jsme narazili na tři pojmy, kterým zatím nerozumíme: *hodnota symbolu*, *funkce* a *aplikace funkce*. Dalším používaným termínem je *argument*. Pojďme si je vysvětlit.

Hodnota symbolu

Symbol může sloužit jako **jméno** jiné hodnoty. S takovým symbolem jsme se už i setkali:

```
> pi
3.141592653589793D0
```

Hodnotou symbolu `pi` je číslo `3.141592653589793D0`. Pokud symbol slouží jako jméno hodnoty, říkáme mu **proměnná (variable)**. Také říkáme, že je symbol na svou hodnotu *navázán* (*bound*). Symbol `r` nemá hodnotu (*není navázán* na žádnou hodnotu; *is unbound*):

```
> r
Error: Unbound variable R.
```

Poznámky:

- Chybová zpráva mluví o symbolu `r` jako o proměnné, protože jsme s ním tak chtěli pracovat.
- Příkladem symbolů, se kterými nepracujeme jako s proměnnými, jsou některé názvy funkcí: `sin`, `sqrt` atd.
- Symboly `*`, `+` a další (především `**` a `***`), mají v Listeneru hodnotu, kterou můžeme využívat. Můžete si sami zjistit, o jakou hodnotu jde. (Symboly `*` a `+` jsou současně i názvy funkcí.)
- Symbol `pi` je také proměnná, přestože jeho hodnota se nikdy nemění. Takovým symbolům říkáme (poněkud paradoxně) **konstantní proměnná**.

Funkce

Funkci můžeme chápat jako ucelenou část programu obsahující nějaký výpočet, který počítač umí vykonávat. Takovým výpočtem může být třeba vykonání aritmetické operace (jako je sčítání nebo rozdíl čísel) nebo výpočet hodnoty matematické funkce (jako je kosinus nebo odmocnina) a mnoho dalších věcí.

Funkce v Lispu jsou označovány symboly. Například symbol `+` označuje funkci, která umí sčítat čísla (vykonávat aritmetickou operaci sčítání), symbol `sin` označuje funkci na výpočet sinu (tj. funkci, která umí počítat hodnoty matematické funkce sinus). Symbol označující funkci se také nazývá její *název*.

Argumenty aplikace funkce

V bodu 2. popisu vyhodnocovacího procesu se píše, že výrazy $a_1 \ldots a_n$ se vyhodnotí. Hodnoty, které vzniknou (označené $v_1 \ldots v_n$) se pak v bodu 3. použijí při aplikaci funkce. Těmto hodnotám říkáme *argumenty aplikace funkce*.

Aplikace funkce

Jak je napsáno výše, výpočet výsledku funkce se spustí tak, že se funkce *aplikuje na hodnoty*. Teď už víme, že těmto hodnotám říkáme *argumenty*. Samotnou aplikací rozumíme spuštění kódu, kterým funkce počítá požadovaný výsledek. Například u funkce `+` jde o výpočet součtu zadaných čísel.

Vyhodnocovací proces: příklad

Ukážeme si krok po kroku proces vyhodnocování výrazu `(/ (- 5 3) (+ 5 3))`.

Vyhodnocení výrazu `(/ (- 5 3) (+ 5 3))`
Je to složený výraz, jeho operátor `/` označuje funkci dělení.

Vyhodnotí se další výrazy:

Vyhodnocení výrazu `(- 5 3)`
Je to složený výraz, operátor `-` označuje funkci rozdílu:

Vyhodnotí se další výrazy:

Vyhodnocení výrazu `5`
Je to číslo, výsledkem je číslo `5`

Vyhodnocení výrazu `3`
Je to číslo, výsledkem je číslo `3`

Výsledkem je výsledek aplikace funkce `-` na argumenty `5` a `3`, tedy `2`

Vyhodnocení výrazu `(+ 5 3)`
Je to složený výraz, jeho operátor `+` označuje funkci sčítání

pak se vyhodnotí další výrazy:

Vyhodnocení výrazu `5`
Je to číslo, výsledkem je číslo `5`

Vyhodnocení výrazu `3`
Je to číslo, výsledkem je číslo `3`

Výsledkem je aplikace funkce `+` na argumenty `5` a `3`, tedy `8`

Výsledkem je aplikace funkce `/` na argumenty `2` a `8`, tedy `1/4`

5 Logické hodnoty a predikáty

Ukázali jsme si několik funkcí, které realizují výpočet matematických operací a matematických funkcí. Například funkce `+` `-`, `*`, `/` počítají hodnoty operací, funkce `sin`, `cos`, `sqrt` hodnoty matematických funkcí. Je jistě výhoda, že funkce lze použít jak na označení operací, tak matematických funkcí (a pomáhá nám v tom prefixová notace).

Existuje ještě jeden základní matematický pojem, a to pojem **relace**. Rozdíl mezi operacemi a funkcemi na jedné straně a relacemi na druhé je, že zatímco operace a funkce mají jako výsledek hodnotu (v našem případě číslo), relace popisují *vztah* mezi hodnotami. Například základní relace mezi čísly: $=$, $<$, $>$, $\leq$, $\geq$. Relace nám říkají, *zda* jsou dvě čísla v určitém vztahu[2]. Výraz, ve kterém vystupuje relace, se dá chápat, jako *tvrzení*, které platí nebo neplatí. Například $1 < 2$ je pravda, zatímco $1 \geq 2$ není pravda.

Z tohoto přístupu k relacím vychází možnost realizovat relace v Lispu pomocí funkcí. Jazyk nám poskytuje dvě tzv. *logické* (*pravdivostní*) *hodnoty* `t` a `nil`, které symbolizují pravdu a nepravdu. Matematické relace jsou v jazyce realizovány pomocí funkcí, které logické hodnoty vracejí jako výsledek:

```
> (= 1 1)
T

> (= 1 2)
NIL
```

[2]Přesnou a matematicky správnou definici relace se dozvíte v jiném předmětu.

```
> (< (/ 2 2) 2)
T

> (< (+ 1 2 3) 6)
NIL
```

Funkce, které vracejí jako výsledek logické hodnoty (a svou hodnotou tedy odpovídají na otázku, zda je něco pravda, nebo ne), se nazývají *predikáty*.

Hodnoty `t` a `nil` jsou symboly, které mají tu zvláštnost, že se stejně jako čísla vyhodnocují na sebe:

```
> t
T

> nil
NIL
```

Jsou to konstantní proměnné.

Symboly ovšem nejsou čísla. Není je proto možné používat s funkcemi pracujícími s čísly:

```
> (+ 1 nil)
Error in function +: Elements of (1 NIL) are not NUMBERs.

> (= 0 nil)
Error in function =: Elements of (0 NIL) are not NUMBERs.
```

Někdy se hodí porovnávat dvě hodnoty, které nejsou obě čísly. K tomu slouží funkce `eql`:

```
> (eql 1 1)
T

> (eql 1 2)
NIL

> (eql t t)
T

> (eql t nil)
NIL

> (eql 1 nil)
NIL
```

Pomocí logických hodnot můžeme pracovat s tzv. *podmíněnými výrazy*:

```
> (if (= 1 2) 10 11)
11

> (if (< 1 2) (/ 0 1) (/ 1 0))
0
```

Druhý příklad nás vede k pojmu *speciálního operátoru*.

6 Speciální operátory

Zopakujme si poslední příklad:

```
> (if (< 1 2) (/ 0 1) (/ 1 0))
0
```

Je vidět, že vyhodnocení tohoto výrazu se neřídí vyhodnocovacím procesem popsaným dříve. Jinak by totiž došlo k chybě (jaké a proč?).

Operátor `if` ve vyhodnocovaném výrazu je totiž vyjímečný (**speciální**) a klasický vyhodnocovací proces pro něj neplatí.

Operátory `if` a `bind`

Podmíněný výraz je složený výraz s operátorem `if`. Takový výraz musí mít za operátorem tři podvýrazy. Vyhodnocuje se takto:

Vyhodnocení výrazu `(if` a b c`)`

1. Vyhodnotí se a na hodnotu u.
2. Pokud je u rovno `NIL`, vyhodnotí se c a vrátí jeho hodnota.
3. Pokud není, vyhodnotí se b a vrátí jeho hodnota.

Uvědomte si přesný význam bodu 3. Plyne z něj, že hodnotou prvního podvýrazu za operátorem `if` může být i jiná hodnota než `t` nebo `nil`.

Operátor `if` je prvním příkladem operátoru, který neoznačuje funkci. Takovým operátorům říkáme **speciální operátory**. Výrazy se speciálními operátory mají svá vlastní pravidla vyhodnocování.

Dalším příkladem speciálního operátoru je symbol `bind`.

Výraz s operátorem `bind` musí mít za operátorem dva podvýrazy, z nichž první musí být symbol. Operátor nastaví jeho hodnotu na hodnotu posledního podvýrazu.

Vyhodnocení výrazu `(bind` a b`)`

1. Vyhodnotí se b.
2. Hodnota symbolu a se nastaví na tuto hodnotu.

Operátor `bind` udělá ze symbolu a tak zvanou *proměnnou listeneru* (*Listener Variable*), které se od ostatních proměnných významně liší. Podrobnosti si vysvětlíme později.

Všimněte si, že výraz a se **nevyhodnocuje** (co by se stalo, kdyby se vyhodnocoval?).

Příklady:

```
> (bind r 10)
10

> (* 2 pi r)
62.83185307179586D0

> (bind r 11)
11

> (* 2 pi r)
69.11503837897544D0

> (bind 1 1)
Error: First argument to BIND is not a symbol. Instead: 1.

> (bind nil 1)
Error: Cannot BIND a constant variable.
```

Takto vypadá obecnější popis vyhodnocovacího procesu v jazyce Common Lisp:

Vyhodnocení výrazu E

Je-li E **symbol,** výsledkem je hodnota symbolu E.

Je-li E **jiný atom než symbol,** výsledkem je E.

Je-li E **seznam s operátorem** o **a dalšími prvky** $a_1 \ldots a_n$**,** pak operátor o musí být symbol a vykoná se následující:

Jestliže o **je speciální operátor,** seznam se vyhodnotí podle pravidel tohoto operátoru.

Jinak
1. se zjistí funkce f, kterou symbol o označuje,
2. zjistí se hodnoty $v_1 \dots v_n$ výrazů $a_1 \dots a_n$ (opět vyhodnocovacím procesem).
3. Výsledkem je výsledek aplikace funkce f na hodnoty $v_1 \dots v_n$.

Nové symboly

Na konci každého textu budu uvádět seznam symbolů, které mají nějaký význam, ať už jako proměnné, funkce nebo speciální operátory a které se vztahují k tématu textu. O některých (ne o všech) se v textu přímo mluvilo. Dokumentaci k symbolům si můžete přečíst v aplikaci.

proměnné: `t`, `nil`, `pi`, `e`, `*`, `**`, `***`

speciální operátory: `if`, `bind`, `and`, `or`

funkce: `eql`, `numberp`, `=`, `<`, `+`, `-`, `*`, `/`, `round`, `floor`, `sqrt`, `expt`, `log`, `sin`, `cos`, `tan`

Otázky a úkoly na cvičení

1. Kolik prvků mají tyto složené výrazy?

```
((((1)))), (1 1), ((1 1)), (((1 1))), ((3 3 3) (1))
```

2. Pokuste se odhadnout, co bude výsledkem vyhodnocení následujících výrazů. Pokud by mělo dojít k chybě, řekněte i, k jaké. Pak ověřte na počítači:

```
(* 2 2 2), (* 2 2), (* 2), (*), (/ 0 1), (/ 1 0),
(/ (/ (/ (/ 16 2) 2) 2) 2), (1 2 3 4), 1, 2,
(if (= 0 (+ (- (+ 10 20) 30))) 7 8),
(+ 1 2), (+1 2), (1 + 2 + 3),
((1 + 2) (1 + 3)), ((+ 1 2) (+ 1 3))
```

3. Přepište do prefixové notace jazyka Lisp a pak vyhodnoťte:

$$\frac{7(1+5.8)}{4(2.51-2.34)}$$

4. Přepište do prefixové notace jazyka Lisp a pak vyhodnoťte:

$$\frac{5 + \frac{14}{3} + (2 - (3 - (6 - \frac{4}{3})))}{(1 - \frac{2}{3})(2 - 6)}$$

5. Co je to proměnná? Co je to funkce?
6. Co je v Lispu predikát? Uveďte příklad.
7. Jaký je rozdíl mezi funkcí a speciálním operátorem?
8. Proč musí být `bind` speciální operátor?
9. Popište všechny kroky vyhodnocení výrazu `(bind a (+ 1 pi))`.
10. Napište výraz, jehož vyhodnocením ověříme, že `if` není funkce.
11. Co je hodnotou následujících výrazů?

```
t, nil, (if 1 2 3), (if nil (+ 1 nil) nil)
```

12. Popište všechny kroky vyhodnocení výrazů z předchozí úlohy.
13. Napište výraz, jehož hodnotou bude `nil`, právě když hodnota proměnné `x` nebude `nil`. Zkuste to pomocí operátoru `if` a pak bez.
14. Napište výraz, jehož hodnota nebude `nil`, právě když číslo v proměnné `a` je menší nebo rovno číslu v proměnné `b`.
15. Napište výraz, jehož hodnota nebude `nil`, právě když celé číslo v proměnné `a` je dělitelné celým číslem v proměnné `b`.
16. Napište výraz, jehož hodnota nebude `nil`, právě když nezáporné celé číslo v proměnné `a` je tzv. *čtverec*, tj. je druhou mocninou nezáporného celého čísla.