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

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 učí, že ty se rychle mění a vyvíjejí, zatímco principy zůstávají mnoho desetiletí stále stejné. A koneckonců, když se podíváme, jak mnohdy fungují softwarové produkty, které v *praxi* vznikají, nezískáme dojem, že by se zrovna z ní bylo dobré ve všem inspirovat.

V tomto semestru se budeme zabývat programovací jazykem Scheme. Je třeba, abyste si naistalovali a správně nastavili aplikaci DrRacket. (Podrobnosti jsou v jiném dokumentu.)

2 Scheme jako kalkulačka

Okno aplikace DrRacket se skládá ze dvou hlavních podoken. Horní je *okno definic*, dolní *okno interakcí* (také *příkazový řádek*, *Listener*). Budeme se nejprve věnovat oknu interakcí.

V okně interakcí vstupujeme do kontaktu (*interagujeme*) s programem DrRacket. (Často budu, i když nepřesně, říkat, že s jazykem Scheme.)

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 Scheme rozumí: nejprve napsat operaci, kterou chceme, aby provedl, a pak čísla, se kterými ji má provést. Vše uzavřít do kulatých závorek.

Další příklady:

```
> (/ 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
```

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.

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

```
> (/ 4 6)
2/3
> (/ 1234 2345)
1234/2345
> pi
#i3.141592653589793
```

Poslední vyhodnocení nás poučí, že ve Scheme můžeme používat číslo π tak, že napíšeme jeho anglický název. Znaků `#i` na začátku čísla si nemusíme všímat. Upozorňují nás jen, že hodnota nemusí být přesná, je *inexact*. Jinak, jak vidíme, v zápisu necelých čísel se používá desetinná tečka místo čárky.

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

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

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

```
r: this variable is not defined
```

Tak si aspoň vypočítáme obvod kružnice o poloměru 10:

```
> (* 2 pi 10)
#i62.83185307179586
```

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

```
> (/ 1 0)
/: division by zero
```

Kromě operací s čísly můžeme používat i další funkce, například odmocninu a goniometrické funkce:

```
> (sqrt 16)
4
> (sqrt 2)
#i1.4142135623730951
> (sqrt -1)
0+1i
> (sin (/ pi 2))
#i1.0
> (cos pi)
#i-1.0
```

Jak vidíte, Scheme umí vypočítat i odmocninu z -1! Zato kosinus π a sinus $\frac{\pi}{2}$ neumí vypočítat přesně. I když se s výsledkem trefil, dává nám znaky `#i` najevo, že si s přesností není jistý.

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

```
> (sin pi)
#i1.2246467991473532e-16
```

Znaky `e-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 okně interakcí vyjde stejně?

```
> (sin (/ pi 6))
#i0.49999999999999994
```

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)))
#i0.4999999999999999
```

3 Symbolické výrazy

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

Symbolický výraz je

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

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`, `0+1i`, `2/3`, `#i0.4999999999999999`, `#i1.2246467991473532e-16` a podobně.

Symbol je posloupnost písmen a případně čísel a dalších znaků, která neoznačuje číslo. Symboly neobsahují některé nepovolené znaky, například `#`, závorky, mezery a další prázdné znaky. Symboly jsou například `pi`, `r`, `a1`, `sqrt`, `+`, `/`, `1+1`. Například `-5` nebo `0+1i` nejsou symboly, protože to jsou čísla.

Seznam (složený výraz) je posloupnost jednoho nebo více výrazů. Prvky seznamu jsou odděleny mezerami nebo jinými prázdnými znaky. Seznam začíná levou a končí pravou kulatou závorkou.

Příklady seznamů:

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

Poslední seznam má tři prvky, některé z nichž jsou seznamy.

Toto *nejsou* symbolické výrazy:

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

Abychom mohli ve Scheme pracovat, musíme umět složené symbolické výrazy vytvářet.

Vytváření symbolických výrazů

První symbol v seznamu označuje **název** matematického výrazu (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:

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

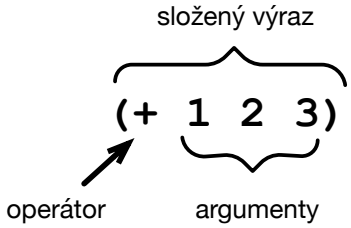
Výraz

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

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: *s čím* se to má udělat

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

Prefixová notace

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

prefix: $\sin x$, -5

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

postfix: $10!$

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

Na důsledně prefixovou notaci je třeba si ale zvyknout.

4 Vyhodnocování symbolických výrazů

Když napíšeme do okna interakcí 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 argumenty $a_1 \dots a_n$, pak

1. se zjistí hodnota p operátoru o (opět vyhodnocovacím procesem), p musí být **procedura**,
2. zjistí se hodnoty $v_1 \dots v_n$ argumentů $a_1 \dots a_n$ (opět vyhodnocovacím procesem).
3. Výsledkem je výsledek **aplikace** procedury p na hodnoty $v_1 \dots v_n$.

V popisu zjednodušeného vyhodnocovacího procesu jsme narazili na tři pojmy, kterým zatím nerozumíme: *hodnota symbolu*, *procedura* a *aplikace procedury*. Pojďme si je vysvětlit.

Hodnota symbolu

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

```
> pi
#i3.141592653589793
```

Hodnotou symbolu `pi` je číslo `#i3.141592653589793`. Proto:

```
> (* 2 pi)
#i6.283185307179586
```

Symbolu, který má hodnotu, také říkáme **proměnná**.

```
> ahoj
ahoj: this variable is not defined
```

Symbol `ahoj` nemá hodnotu. (Není to proměnná.)

Procedury

Už jsme se setkali i s jinými symboly, které mají hodnotu:

```
> +
#<procedure:+>
> /
#<procedure:/>
```

Jejich hodnotou jsou **procedury**.

Procedura je prvek programu, který umožňuje provádět nějaké (například matematické) výpočty se *vstupními hodnotami*. Výpočet se spustí tzv. *aplikací procedury na vstupní hodnoty*. O výsledku výpočtu někdy říkáme, že ho procedura *vrací*.

Vyhodnocovací proces: příklad

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

Vyhodnocení výrazu `(/ (- 5 3) (+ 5 3))`
Je to seznam, vyhodnotí se nejprve operátor:

Vyhodnocení výrazu `/`
Je to symbol, výsledkem je jeho hodnota `#<procedure:/>`

pak se vyhodnotí argumenty:

Vyhodnocení výrazu `(- 5 3)`
Je to seznam, vyhodnotí se nejprve operátor:

Vyhodnocení výrazu `-`
Je to symbol, výsledkem je jeho hodnota `#<procedure:->`

pak se vyhodnotí argumenty:

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

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

Výsledkem je aplikace procedury `#<procedure:->` na `5` a `3`, tedy číslo `2`

Vyhodnocení výrazu `(+ 5 3)`
Je to seznam, vyhodnotí se nejprve operátor:

Vyhodnocení výrazu `+`
Je to symbol, výsledkem je jeho hodnota `#<procedure:+>`

pak se vyhodnotí argumenty:

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

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

Výsledkem je aplikace procedury `#<procedure:+>` na hodnoty `5` a `3`, tedy `8`

Výsledkem je aplikace procedury `#<procedure:/>` na hodnoty `2` a `8`, tedy `1/4`

5 Logické hodnoty a podmíněné výrazy

Ukázali jsme si několik procedur jazyka Scheme, které realizují výpočet matematických operací a funkcí. Konkrétně procedury `+` `-`, `*`, `/` počítají hodnoty operací, procedury `sin`, `cos`, `sqrt` hodnoty funkcí. Je jistě výhoda, že procedury lze použít jak na implementaci operací, tak 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, *jestli* jsou dvě čísla v určitém vztahu[1]. 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 jazyce Scheme pomocí procedur. Jazyk nám poskytuje dvě tzv. *pravdivostní* (*logické*) *hodnoty* `#true` a `#false`, které symbolizují pravdu a nepravdu. Matematické relace jsou v jazyce realizovány pomocí procedur, které logické hodnoty vracejí jako výsledek:

```
> (= 1 1)
#true
> (= 1 2)
#false
> (< 1 2)
#true
> (>= 1 2)
#false
```

Hodnoty `#true` a `#false` jsou kromě čísel a symbolů další typ atomů a vyhodnocují se na sebe:

```
> #true
#true
> #false
#false
```

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

```
> (if (<= 2 1) 10 11)
11
> (if (< 1 2) 1 (/ 1 0))
1
```

Druhý příklad nás vede k pojmu *speciálního operátoru*.

6 Speciální operátory

Zopakujme si poslední příklad:

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

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

Je vidět, že vyhodnocené 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.

Speciální operátory `if` a `define`

Podmíněný výraz je složený výraz s operátorem `if`. Takový výraz musí mít přesně tři argumenty (jinak dojde k chybě). Vyhodnocuje se takto:

Vyhodnocení výrazu `(if` a b c`)`

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

Operátor `if` je prvním příkladem **speciálního operátoru**. Výrazy se speciálními operátory mají svá vlastní pravidla vyhodnocování.

Dalším příkladem speciálního operátoru je symbol `define`. Výraz se speciálním operátorem `define` musí mít dva argumenty, z nichž první musí být symbol, který zatím není definován jako jméno hodnoty (proměnná). Operátor tuto definici provede a hodnotu nastaví.

Vyhodnocení výrazu `(define` a b`)`

1. Vyhodnotí se b.
2. symbol a se učiní jménem této hodnoty.

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

Příklad:

```
> r
r: this variable is not defined
> (define r 10)
> r
10
> (* 2 pi r)
```

```
#i62.83185307179586
> (define r 11)
r: this name was defined previously and cannot be re-defined
```

Takto tedy vypadá obecný popis vyhodnocovacího procesu v jazyce Scheme:

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 argumenty** $a_1 \ldots a_n$**,** pak

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

Jinak

1. se zjistí hodnota p operátoru o (opět vyhodnocovacím procesem), p musí být procedura,
2. zjistí se hodnoty $v_1 \ldots v_n$ argumentů $a_1 \ldots a_n$ (opět vyhodnocovacím procesem).
3. Výsledkem je výsledek aplikace procedury p na hodnoty $v_1 \ldots v_n$.

7 Závěr

Pojmy k zapamatování

Symbolický výraz, atom, seznam, číslo, symbol, logická (pravdivostní) hodnota. Symbol jako jméno hodnoty, procedura. Operátor, argumenty. Vyhodnocovací proces. Speciální operátor, speciální operátory `if` a `define`.

Kontrolní otázky

1. Jaký je rozdíl mezi procedurou a speciálním operátorem?
2. Proč musí být `define` speciální operátor?

Otázky a úkoly na cvičení

1. Kolik prvků mají tyto seznamy `((((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:

```
(+ 1 1 1), (+ 1 1), (+ 1), (+), +, (/ 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), (+ 2 + 3),
((1 + 2) (1 + 3)), ((+ 1 2) (+ 1 3)),
(+ +), (+ (+)), ((+) (+))
```

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

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

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

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

5. Co bude výsledkem vyhodnocení druhého výrazu, když před ním vyhodnotíme první výraz? Odhadněte, zdůvodněte a pak vyzkoušejte.

```
(define minus +)
(minus 2 2)
```