Haskell Platform, volání funkcí, if, definice podle vzoru, rekurze

IB015 Neimperativní programování

Kolektiv cvičících IB015

Fakulta informatiky, Masarykova univerzita

podzim 2018

Haskell v praxi

Haskell:

- kompilátor vs. interpret
- GHC, Haskell Platform: `ghc`, `ghci`, `runghc`
  (instalace viz osnova v ISu)
- API dokumentace na Hayoo

Referenční kompilátor pro podzim 2018:

- GHC-8.0.2
- existují i novější verze (nejnovější 8.4.3), klidně instalujte ty

Haskell, počítače na FI

Spouštení:

- všechny nástroje jsou spustitelné v shellu
- spuštění shellu např.: `Alt-F2`, `gnome-terminal`
- prompt shellu: `xlogin@nymfe12:/home/xlogin$`
- prompt interpretu: `Prelude>` nebo `Main>`

Stroje na FI (nymfeXX):

- Ubuntu 18.04 má lokálně instalovaný GHC-8.0.2

Práce s interpretem

Do interpretu (GHCi) lze zadávat příkazy a výrazy.
Příkazy:

- `:h[elp]` – nápověda
- `:t[ype] expr` – typ výrazu
- `:i[nfo] ident` – informace o objektech jazyka
- `:l[oad] file.hs` – načtení souboru s Haskellovým kódem
- `:r[eload]` – znovunačtení posledního souboru
- `:q[uit]` – ukončení práce s interpretem
- `:m[odule] Modul` – načtení modulu

Výrazy:

- vše ostatní...

Nekončící výpočty lze přerušit pomocí `Ctrl-C`.

Volání funkcí

Funkce se dají volat dvěma různými způsoby:

1. **prefixový zápis:** `mod 10 8`
   - funkce s alfanumerickým názvem před argumenty
   - výčet argumentů *není* vyčleněn závorkami
   - pozor na detaily: `f x y` vs. `f (x y)` vs. `f (x,y)`
2. **infixový zápis:** `3 + 4`
   - operátor (funkce) s nealfanumerickým názvem mezi argumenty
   - pouze pro *binární* funkce

Priorita a asociativita

Obvykle ve výrazech voláme mnoho funkcí oběma zápisy.

- **priorita:** co interpret uzávorkuje jako první?
  - `f 3 + 4` pro unární `f` chápeme jako `((f 3) + 4)`
  - prefixové volání má nejvyšší prioritu
  - pro infix můžeme prioritu nastavit při definici funkce
- **asociativita:** jak interpret uzávorkuje řadu infixových volání stejné funkce?
  - `3 * 4 * 5` $\rightsquigarrow$ `((3 * 4) * 5)` vs. `(3 * (4 * 5))` vs. chyba
  - asociativitu možno nastavit při definici operátoru (funkce)
  - pozor na neasociativní operátory: ==, /=, <, ...

Pokud to nemá operátor (funkce) explicitně definované, jeho priorita je 9 a je asociativní zleva.

Vlastnosti operátorů

Příklad 1.1.1: S využitím interního příkazu `:info` interpretu `ghci` zjistěte prioritu a asociativitu následujících operací:

```
^, *, /, +, -, ==, /=, >, <, >=, <=, &&, ||
```

Příklad 1.1.2: S použitím interpretu jazyka Haskell porovnejte vyhodnocení následujících dvojic výrazů a rozdíl vysvětlete.

a) `5 + 9 * 3` versus `(5 + 9) * 3`

b) `2 ^ 2 ^ 2 == (2 ^ 2) ^ 2` versus
   `3 ^ 3 ^ 3 == (3 ^ 3) ^ 3`

c) `3 + 3 + 3` versus `3 == 3 == 3`

d) `(3 == 3) == 3` versus `(4 == 4) == (4 == 4)`

Syntaktická struktura aneb závorkování výrazů

Příklad 1.1.4: Doplňte všechny implicitní závorky do následujících výrazů:

a) `recip 2 * 5`

b) `sin pi/2`

c) `mod 3 8 * 2`

d) `f g 3 + g 5`

e) `2 + div m 18 == m ^ 2 ^ n && m * n < 20`

f) `id id . flip const const`

Programy v souborech

Zapisování programů v souborech:

- soubory mají příponu `.hs`
- po spuštění `ghci` se načtou pomocí příkazu `:l filename`
- soubor primárně obsahuje definice uživatelských funkcí
- pro znovunačtení stejných souborů je možné použít příkaz `:r`
- komentáře v kódu:

```
-- line comment
function :: Int -> Int
function x = x + 1
{-
block comment
multiple lines
-}
```

První vlastní funkce v Haskellu

Příklad 1.1.5: Vytvořte funkci `circleArea`, která pro zadaný poloměr spočítá obsah kruhu o tomto poloměru. Přibližná hodnota konstanty π se dá v Haskellu získat pomocí `pi`.

Příklad 1.1.6: Definujte funkci `isSucc`, která pro dvě přirozená čísla `n1`, `n2` rozhodne, jestli `n2` je následníkem `n1`.

Příklad 1.1.7: Pomocí funkce `max` definujte funkci `max3`, která pro tři celá čísla `z1`, `z2` a `z3` vrátí to největší z nich.

Příklad 1.1.8: Naprogramujte funkci `mid`, která pro tři celá čísla `z1`, `z2` a `z3` vrátí to *prostřední* z nich (t.j. to druhé v jejich uspořádané trojici podle $\leq$).

Podmíněný výraz

Syntax: `if bool_expr then expr1 else expr2`
Výrazy *expr1* a *expr2* musí být stejného typu.

Příklad 1.1.9: Pomocí `if` a funkce `mod` definujte funkci `tell`, která bere jeden argument `n` a vrací:

a) `"one"` pro `n = 1`
b) `"two"` pro `n = 2`
c) `"(even)"` pro sudé `n > 2`
d) `"(odd)"` pro liché `n > 2`

Definice funkce podle vzoru

Definice funkce podle vzoru obsahuje:

- název funkce (`tell`),
- hodnoty argumentů (`1`, `2`), formální parametry (`x`), anonymní parametry (`_`),
- rovnítko oddělující pravou a levou stranu definice (=),
- pravou stranu/tělo funkce.

```
tell 1 = "one"
tell 2 = "two"
tell x = if even x then "(even)" else "(odd)"
```

- Všechny řádky definující funkci musí být spolu.
- Není možné použít stejný parametr na levé straně víckrát.

Funkce na procvičení

Příklad 1.2.1: Definujte funkci `isWeekendDay`, která rozhodne, jestli daný řetězec obsahuje právě a jenom název víkendového dne.

Příklad 1.2.2: Definujte funkci `isSmallVowel`, která rozhodne, jestli je dané ASCII písmeno malou samohláskou (např. literál znaku *a* se v Haskellu zapíše jako `'a'`).

Rekurze

Příklad 1.2.4: Definujte rekurzivní funkci pro výpočet faktoriálu.

Příklad 1.2.5: Naprogramujte rekurzivně funkci, která pro dané nezáporné celé číslo vypočítá jeho zbytek po dělení 3. Nesmíte použít funkci `mod`.

Příklad 1.2.6: Definujte funkci `power`, která bude brát dva argumenty `z` a `n` a vrátí n-tou mocninu čísla `z`. Následně si zkontrolujte, jak se vaše řešení chová na záporných n a pokuste se zajistit, aby pro tyto případy `power` vracela 0.

Příklad 1.2.7: Napište funkci, která o zadaném přirozeném čísle rozhodne, jestli je mocninou dvojky.

Rekurze

Příklad 1.2.8: Definujte v Haskellu funkci `dfct` (v kombinatorice někdy značenou !!), kde

$$\begin{aligned} 0!! &= 1, \\ (2n)!! &= 2 \cdot 4 \cdots (2n), \\ (2n+1)!! &= 1 \cdot 3 \cdots (2n+1) \end{aligned}$$

Zkuste funkci naprogramovat vícero způsoby.

Rekurze

Příklad 1.2.8: Definujte v Haskellu funkci `dfct` (v kombinatorice někdy značenou !!), kde

$$\begin{aligned} 0!! &= 1, \\ (2n)!! &= 2 \cdot 4 \cdots (2n), \\ (2n+1)!! &= 1 \cdot 3 \cdots (2n+1) \end{aligned}$$

Zkuste funkci naprogramovat vícero způsoby.

Příklad 1.2.15: Co počítá následující funkce? Jak se chová na argumentech, kterými jsou nezáporná čísla? Jak se chová na záporných argumentech?

```
fun 0 = 0
fun n = fun (n - 1) + 2 * n - 1
```

(bonus pro rychlé)