Domácí úkol 8: Jednoduchý imperativní jazyk

V tomto domácím úkolu si v Haskellu vyzkoušíte naprogramovat interpret jednoduchého imperativního jazyka. Odměnou vám za to mohou být až dva body do hodnocení a jeden bonusový bod, který se počítá do bodů za aktivitu nad rámec 5 bodů ze cvičení.

Jazyk *Imp*

Pro účely této úlohy si definujeme jednoduchý imperativní jazyk *Imp*, který se skládá z aritmetických výrazů na celých číslech, logických výrazů a imperativních příkazů, které se postupně vykonávají. Tyto tři složky nyní jednotlivě popíšeme.

Aritmetické výrazy

Aritmetické výrazy v jazyce *Imp* jsou několika druhů:

- celočíselná konstanta,
- proměnná,
- součet dvou aritmetických výrazů,
- součin dvou aritmetických výrazů.

V jazyce Haskell budeme aritmetické výrazy reprezentovat vlastním datovým typem `AExpr`:

```
type Varname = String

data AExpr = Con Integer
           | Var Varname
           | Add AExpr AExpr
           | Mul AExpr AExpr
           deriving ( Eq, Show, Read )
```

Můžete si všimnout, že jednotlivé proměnné jsou pojmenované textovými řetězci. Abychom od sebe viditelně odlišili obyčejné řetězce a řetězce reprezentující názvy proměnných, zavádíme typové synonymum `Varname` pro řetězce, které reprezentují názvy proměnných,

Logické výrazy

I logické výrazy jsou v jazyce *Imp* několika druhů:

- test na rovnost dvou aritmetických výrazů,
- test na neostrou nerovnost dvou aritmetických výrazů („menší nebo rovno“),
- konjunkce dvou logických výrazů,
- negace logického výrazu.

V jazyce Haskell budeme logické výrazy reprezentovat vlastním datovým typem `BExpr`:

```
data BExpr = Equal AExpr AExpr
           | LEQ AExpr AExpr
           | And BExpr BExpr
           | Not BExpr
           deriving ( Eq, Show, Read )
```

Příkazy

Jazyk *Imp* také obsahuje následující druhy příkazů:

- přiřazení hodnoty aritmetického výrazu do proměnné;
- výpis obsahu proměnné na standardní výstup (ukončený znakem nového řádku);

- klasický podmíněný příkaz *if*, který podle hodnoty logického výrazu spustí buď první příkaz, pokud se logický výraz vyhodnotí na `True`, nebo druhý příkaz v opačném případě;
- klasický cyklus *while*, který opakovaně spouští zadaný příkaz, dokud se zadaný logický výraz vyhodnocuje na `True`;
- sekvenční kompozici dvou příkazů, která odpovídá spuštění prvního příkazu a poté druhého příkazu.

V jazyce Haskell budeme příkazy reprezentovat vlastním datovým typem `Command`:

```
data Command = Assign Varname AExpr
             | Print Varname
             | If BExpr Command Command
             | While BExpr Command
             | Seq Command Command
             deriving ( Eq, Show, Read )
```

Zadání

Vaším úkolem je napsat funkci `eval :: Command -> IO ()`, která interpretuje zadaný program a vypisuje na standardní výstup všechny jeho výpisy, které vznikly pomocí příkazu `Print`. Předpokládejte, že na začátku vyhodnocování programu mají všechny proměnné hodnotu 0.

Příklady

Níže následuje několik příkladů programů a očekávaných výstupů funkce `eval`. Další programy, na kterých si můžete svůj interpret vyzkoušet, najdete ve studijních materiálech. Navíc doporučujeme zkusit si napsat i vlastní programy[1] a zkoušet si svůj interpret na nich.

```
program1 :: Command
program1 = Print "x" `Seq`
           Assign "x" (Con 42) `Seq`
           Print "x" `Seq`
           Print "y"

program2 :: Command
program2 = While (Var "x" `LEQ` Con 9)
           (
             Assign "x" (Var "x" `Add` Con 1) `Seq`
             Print "x"
           )

program3 :: Command
program3 = Assign "x" (Con 1) `Seq`
           While (Var "x" `LEQ` Con 100)
           (
             Print "x" `Seq`
             Assign "x" (Var "x" `Mul` Con 2)
           )
```

Očekávané výsledky volání funkce `eval` pro tyto programy jsou:

```
> eval program1
0
42
0

> eval program2
1
```

[1] Teoreticky je možné v jazyce *Imp* napsat libovolný algoritmus, který je možné napsat v jakémkoliv jiném programovacím jazyce. Jazyk *Imp* je totiž i přes svoji jednoduchost *Turingovsky úplný*.

```
2
3
4
5
6
7
8
9
10

> eval program3
1
2
4
8
16
32
64
```

Poznámky a tipy

- Ve svém programu můžete importovat a používat libovolné moduly ze standardní knihovny (balíček `base`).
- Abyste nemuseli definice datových typů `AExpr`, `BExpr` a `Command` kopírovat z tohoto zadání, najdete je v souboru ve studijních materiálech.
- Při vyhodnocování programu si budete potřebovat ukládat aktuální přiřazení do proměnných (tzv. *kontext*). K tomu se vám může hodit zavést si jedno z následujících typových synonym podle toho, jak chcete aktuální přiřazení ukládat:
  - `type Context = [(Varname, Integer)]`, nebo
  - `type Context = Varname -> Integer`.
- Doporučujeme si nejprve napsat pomocné funkce na vyhodnocení aritmetických a logických výrazů v zadaném kontextu.
- Kvůli omezení vyhodnocovací služby používejte pro práci s `IO` jen ty funkce, které znáte z přednášky nebo cvičení.
- Přebíráte-li kód odjinud, nezapomeňte uvést zdroj. V opačném případě bude na vaši práci pohlíženo jako na plagiát.

Odevzdání a bodování

Domácí úkol se odevzdává přes odpovědník v ISu. Odpovědník obsahuje dvě pole: do prvního vložte svou implementaci funkce `eval` a případně všech pomocných funkcí a vámi definovaných typů, do druhého okomentujte, jak jste při řešení postupovali. Definice typů `AExpr`, `BExpr` a `Command` do řešení nevkládejte. Máte **dvě možnosti odevzdání celého odpovědníku** (a samozřejmě kontrolu syntaxe před odevzdáním). Dokud nekliknete na „odevzdat“, můžete odpovědník zavřít a znovu otevřít bez penalizace. Po každém odevzdání uvidíte výsledky automatických testů, věnujte však čas dostatečnému vlastnímu testování. Po odevzdání budeme úkoly vyhodnocovat i ručně; bude se hodnotit i kvalita kódu a jeho popis, ne jen jeho funkčnost. Můžete získat až dva body, které se vám započítají do sumy bodů z domácích úkolů. Hodnocení není binární; i v případě jen částečné funkčnosti vašeho programu můžete nějaké body získat.

Pokud vám u vašeho řešení vyhodnocovací služba vrátí `SomeAsyncException`, znamená to, že vaše řešení neskončilo v zadaném časovém limitu. Vyhodnocovací služba vám v takovém případě vypíše program, u nějž k tomu došlo – zkuste tedy ověřit, jestli na něm vaše funkce `eval` necyklí a jestli její vyhodnocení netrvá příliš dlouho.

Bonus

V tomto úkolu máte možnost získat i jeden bonusový bod, který se počítá do bodů za aktivitu (i nad limit 5 bodů). Ten dostanete za to, když nějakým zajímavým a netriviálním způsobem rozšíříte zadaný jazyk *Imp*. Můžete například zkusit jazyk *Imp* rozšířit o *for* cykly, o čtení vstupu od uživatele nebo o podporu řetězců a práce s nimi. Naopak za netriviální rozšíření nepovažujeme například přidání příkazu `Skip`, jehož provedení neudělá nic.

Pokud budete měnit syntaxi jazyka *Imp*, nezapomeňte odpovídajícím způsobem změnit i jeho sémantiku. Tedy pokud budete například přidávat nový příkaz odpovídající *for* cyklu, nezapomeňte odpovídajícím způsobem upravit i funkci `eval`.

Bonusovou část tohoto úkolu odevzdávejte jako jeden soubor s příponou `.hs` do příslušné odevzdávárny v ISu. Před odevzdáváním si ověřte, že tento soubor jde samostatně zkompilovat. Tedy odevzdaný soubor nesmí záviset na žádných jiných souborech; zejména musí obsahovat všechny definice datových typů (např. `AExpr`, `BExpr` a `Command`) i vámi definovaných funkcí. Na začátek souboru také do komentáře napište, jak jste jazyk *Imp* rozšířili a jak jste při tom postupovali. Na bonusovou část úkolu máte více času než na základní část – bonusovou část můžete odevzdávat do konce neděle 26. listopadu 2017.