Domácí úkol 4: vyhodnocování aritmetických výrazů

V tomto domácím úkolu si vyzkoušíte vytvořit o něco delší kód v Haskellu, odměnou vám za to mohou být až dva body do hodnocení. Vaším úkolem je naprogramovat funkci pro vyhodnocování jednoduchých aritmetických výrazů zadaných v textovém řetězci. Výsledná funkce by měla fungovat nějak takto:

```
> eval "1 + 2 * 4"
9
> eval "2 - 2 - 2"
-2
```

Vaším úkolem je tedy naprogramovat funkci `eval :: String -> Int`, která vyhodnocuje aritmetické výrazy vytvořené podle následujících pravidel:

- aritmetický výraz může obsahovat jen **nezáporná dekadická čísla** a **operátory** `+`, `-`, `*`, `/`, `%` (modulo) a `^` (umocňování na nezáporný exponent), které jsou vždy zapsány jako infixové operátory,
- okolo čísel a operátorů může být libovolný nenulový počet mezer (tedy na každé straně operátoru musí být alespoň jedna mezera a může jich tam být i více),
- aritmetický výraz neobsahuje závorky,
- ve validním aritmetickém výrazu nikdy nedochází k dělení nulou, počítání zbytku po dělení nulou ani k umocňování na záporný exponent,
- aritmetický výraz nemůže být prázdný,
- výrazy, které obdrží váš program na vstup, budou vždy validní, nemusíte tedy ošetřovat žádné chybové případy.

Podobně jako v Haskellu mají i v těchto aritmetických výrazech operátory priority a asociativitu:

- nejmenší prioritu mají operátory `+` a `-`, které asociují zleva,
- dále pak operátory `*`, `/` a `%`, které také asociují zleva,
- nejvyšší prioritu má operátor `^`, který asociuje zprava.

Všechny operace jsou celočíselné a fungují tak, jak je provádějí Haskellové funkce `(+)`, `(-)`, `(*)`, `div`, `mod` a `(^)` nad typem `Int`.

Příklady výrazů, které může vaše funkce dostat na vstupu, jsou tedy například následující:

- `"4"`, samotné číslo je také validní výraz,
- `"2 + 0 - 18"`, odpovídá plně uzávorkovanému výrazu $(2 + 0) - 18$, a tedy se vyhodnotí na -16 (tedy, ačkoli ve výrazu nemůžou být záporná čísla, mohou vyjít v průběhu výpočtu),
- `"2 + 3 * 3 / 2"`, odpovídá $2 + \lfloor(3 * 3)/2\rfloor$, výsledek je tedy 6 (celočíselné dělení pomocí `div` zaokrouhluje dolů),
- `"3 * 2 + 2 ^ 3 ^ 2"` odpovídá $(3 * 2) + \left(2^{(3^2)}\right)$, výsledek je tedy 518.

Příklady výrazů, které jsou nevalidní (a nemusíte je řešit, protože vstupy jsou garantovaně validní):

- `"2 / 0"` dělení 0,
- `"3 % 0 ^ 8"` odpovídá $3 \bmod 0^8$, došlo by tedy k počítání zbytku po dělení 0,
- `"3 ^ 5 ^ 50"` není validní, protože dojde k přetečení datového typu `Int` (`5 ^ 50 :: Int` má hodnotu `-6776596920136667815`), a tedy k umocňování na záporný exponent.

Při tvorbě vašeho řešení by se vám mohly hodit některé funkce z modulu `Data.List`. Můžete rovněž používat další moduly ze standardních balíků.[1]

Způsob odevzdání: domácí úkol se odevzdává přes odpovědník v ISu. Odpovědník obsahuje dvě pole, do prvního vložte vaši implementaci funkce `eval :: String -> Int`, do druhého okomentujte, jak jste při řešení postupovali. Máte **dvě možnosti odevzdání** (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. Následně 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.

[1] tyto moduly najdete na https://hackage.haskell.org/package/base-4.7.0.0