Haskell — Expressions P70540_en

We want to have a module to mainpulate and evaluate integer expressions with addition, substraction, multiplication and division operations. In order to do so, the following type is declared:

data *Expr* = *Val* **Int** | *Add Expr Expr* | *Sub Expr Expr* | *Mul Expr Expr* | *Div Expr Expr*

For instance, *Add* (*Val* 3) (*Div* (*Val* 4) (*Val* 2)) represents $3 + 4/2$, which evaluates to 5.

1. Evaluation without errors (20 points)

Using the *Expr* type, define a function *eval1* :: *Expr* → **Int** that, given an expression, returns its evaluation. You can assume there will never be divisions by zero.

2. Evaluation with possible error (30 points)

Using the *Expr* type, define a function *eval2* :: *Expr* → **Maybe Int** that, given an expression, returns its evaluationn as a **Just** value. In the case that some division by zero occurs, the result must be **Nothing**. You probably want to use the **do** notation over the **Maybe** *a* monad.

3. Evaluation with error report (30 points)

Using the *Expr* type, define a function *eval3* :: *Expr* → **Either String Int** that, given an expression, returns its evaluation as **Right** value. In the case that some division by zero occurs, the result must be **Left** `"div0"`. You probably want to use the **do** notation over the **Either** *a b* monad.

Sample input 1

```
eval1 (Val 2)
eval1 (Add (Val 2) (Val 3))
eval1 (Sub (Val 2) (Val 3))
eval1 (Div (Val 4) (Val 2))
eval1 (Mul (Add (Val 2) (Val 3)) (Sub (Val 2) (Val 3)))
```

Sample output 1

```
2
5
-1
2
-5
```

Sample input 2

```
eval2 (Val 2)
eval2 (Add (Val 2) (Val 3))
eval2 (Sub (Val 2) (Val 3))
eval2 (Div (Val 4) (Val 2))
eval2 (Mul (Add (Val 2) (Val 3)) (Sub (Val 2) (Val 3)))
eval2 (Div (Val 4) (Val 0))
```

```
eval2 (Add (Div (Val 4) (Val 0)) (Val 3))
eval2 (Add (Val 3) (Div (Val 4) (Val 0)))
```

Sample output 2

```
Just 2
Just 5
Just (-1)
Just 2
Just (-5)
Nothing
Nothing
Nothing
```

Sample input 3

```
eval3 (Val 2)
eval3 (Add (Val 2) (Val 3))
eval3 (Sub (Val 2) (Val 3))
eval3 (Div (Val 4) (Val 2))
eval3 (Mul (Add (Val 2) (Val 3)) (Sub (Val 2) (Val 3)))
eval3 (Div (Val 4) (Val 0))
eval3 (Add (Div (Val 4) (Val 0)) (Val 3))
eval3 (Add (Val 3) (Div (Val 4) (Val 0)))
```

Sample output 3

```
Right 2
Right 5
Right (-1)
Right 2
Right (-5)
Left "div0"
Left "div0"
Left "div0"
```

Problem information

Author : Jordi Petit
Translator : Jordi Petit
Generation : 2024-05-02 22:23:05