```
ouzo2:code$ ghci
GHCi, version 8.2.2: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /Users/hallgren/.ghci
Prelude> :l ParsingExamples.hs
[1 of 2] Compiling Parsing          ( Parsing.hs, interpreted )
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> takeWhile isDigit "123+456"
"123"
*ParsingExamples> dropWhile isDigit "123+456"
"+456"
*ParsingExamples> span isDigit "123+456"
("123","+456")
*ParsingExamples> span isDigit "a+123+456"
("","a+123+456")
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> number_v1 "12+34"
Just (12,"+34")
*ParsingExamples> number_v1 "12asdf"
Just (12,"asdf")
*ParsingExamples> number_v1 "12"
Just (12,"")
*ParsingExamples> number_v1 "x+12"
Nothing
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> addition_v1 "12+34"
Just (46,"")
*ParsingExamples> addition_v1 "12+34kasjhdf"
Just (46,"kasjhdf")
*ParsingExamples> addition_v1 "12+"
Nothing
*ParsingExamples> addition_v1 ""
Nothing
*ParsingExamples> addition_v1 "x+1"
Nothing
*ParsingExamples> addition_v1 "12"
*** Exception: ParsingExamples.hs:(32,17)-(36,36): Non-exhaustive patterns in ca
se

*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> addition_v1 "12"
Nothing
*ParsingExamples> addition_v1 "12+345"
Just (357,"")
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> calculation_v1 "12+34"
```

```
Just (46,"")
*ParsingExamples> calculation_v1 "3*4"
Just (12,"")
*ParsingExamples> calculation_v1 "3+4"
Just (7,"")
*ParsingExamples> calculation_v1 "3/4"
Nothing
*ParsingExamples> calculation_v1 "3*4+4"
Just (12,"+4")
*ParsingExamples> :t map
map :: (a -> b) -> [a] -> [b]
*ParsingExamples> :t fmap
fmap :: Functor f => (a -> b) -> f a -> f b
*ParsingExamples> :t (<$>)
(<$>) :: Functor f => (a -> b) -> f a -> f b
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> :t digit
digit :: Parser Char
*ParsingExamples> :t oneOrMore digit
oneOrMore digit :: Parser [Char]
*ParsingExamples> :t read <$> oneOrMore digit
read <$> oneOrMore digit :: Read b => Parser b
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> :t parse
parse :: Parser a -> String -> Maybe (a, String)
*ParsingExamples> parse number "12+34"
Just (12,"+34")
*ParsingExamples> parse addition "12+34"
Just (46,"")
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> parse calculation "3+4"
Just (7,"")
*ParsingExamples> parse calculation "3*4"
Just (12,"")
*ParsingExamples> parse calculation "3*4alksj"
Just (12,"alksj")
*ParsingExamples> parse calculation "3*"
Nothing
*ParsingExamples> parse calculation "3"
Nothing
*ParsingExamples> parse calculation ""
Nothing
*ParsingExamples> parse calculation "asdkfj"
Nothing
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> parse calculation "3*4"
```

```
Just (12,"")
*ParsingExamples> parse calculation "3+4"
Just (7,"")
*ParsingExamples> fold
foldMap  foldl    foldl1   foldr    foldr1
*ParsingExamples> :t foldr1
foldr1 :: Foldable t => (a -> a -> a) -> t a -> a
*ParsingExamples> foldr1 Add [Num 1,Num 2,Num 3]
Add (Num 1) (Add (Num 2) (Num 3))
*ParsingExamples> foldl1 Add [Num 1,Num 2,Num 3]
Add (Add (Num 1) (Num 2)) (Num 3)
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )

ParsingExamples.hs:127:31: error: Variable not in scope: t :: Expr
    |
127 |           return (foldl1 Mul (t:ts))
    |                               ^

ParsingExamples.hs:127:33: error:
    • Variable not in scope: ts :: [Expr]
    • Perhaps you meant ‘fs’ (line 126)
    |
127 |           return (foldl1 Mul (t:ts))
    |                                 ^^
Failed, one module loaded.
*Parsing> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> parse expr "1+2*3"
Just (Add (Num 1) (Mul (Num 2) (Num 3)),"")
*ParsingExamples> parse expr "(1+2)*3"
Just (Mul (Add (Num 1) (Num 2)) (Num 3),"")
*ParsingExamples> parse expr "(1+2)*x"
Just (Add (Num 1) (Num 2),"*x")
*ParsingExamples> parse expr "(1+2)*"
Just (Add (Num 1) (Num 2),"*")
*ParsingExamples> parse expr "(1+2"
Nothing
*ParsingExamples> parse expr "(1+2)*3+4+5+6"
Just (Add (Add (Add (Mul (Add (Num 1) (Num 2)) (Num 3)) (Num 4)) (Num 5)) (Num 6
),"")
*ParsingExamples> parse expr "(1+2)*3+4+5+6*4"
Just (Add (Add (Add (Mul (Add (Num 1) (Num 2)) (Num 3)) (Num 4)) (Num 5)) (Mul (
Num 6) (Num 4)),"")
*ParsingExamples> eval <$> parse expr "(1+2)*3+4+5+6*4"

<interactive>:61:10: error:
    • Couldn't match type ‘(Expr, String)’ with ‘Expr’
      Expected type: Maybe Expr
        Actual type: Maybe (Expr, String)
    • In the second argument of ‘(<$>)’, namely
        ‘parse expr "(1+2)*3+4+5+6*4"’
      In the expression: eval <$> parse expr "(1+2)*3+4+5+6*4"
```

```
      In an equation for ‘it’: it = eval <$> parse expr "(1+2)*3+4+5+6*4"
*ParsingExamples> (eval . fst) <$> parse expr "(1+2)*3+4+5+6*4"
Just 42
*ParsingExamples> (eval . fst) <$> parse expr "(1+2)*3"
Just 9
*ParsingExamples>  parse expr "(1+2)*3"
Just (Mul (Add (Num 1) (Num 2)) (Num 3),"")
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> parse expr "1+2*3"
Just (Add (Num 1) (Mul (Num 2) (Num 3)),"")
*ParsingExamples> parse expr "1+2*3+4"
Just (Add (Add (Num 1) (Mul (Num 2) (Num 3))) (Num 4),"")
*ParsingExamples> parse expr "(1+2)*3+4"
Just (Add (Mul (Add (Num 1) (Num 2)) (Num 3)) (Num 4),"")
*ParsingExamples> parse expr "(1+2)*(3+4)"
Just (Mul (Add (Num 1) (Num 2)) (Add (Num 3) (Num 4)),"")
*ParsingExamples> :r
[2 of 2] Compiling ParsingExamples  ( ParsingExamples.hs, interpreted )
Ok, two modules loaded.
*ParsingExamples> main
Welcome to the simple calculator!
Expression? 1+2+3
Value: 6
Expression? 1+2*3
Value: 7
Expression? (1+2)*3
Value: 9
Expression? (1+
Syntax error!
Expression? 1+2ajsd
Syntax error!
Expression? ^CInterrupted.
*ParsingExamples>
Leaving GHCi.
ouzo2:code$
```