```
ouzo2:code$ ghci
GHCi, version 8.2.2: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /Users/hallgren/.ghci
Prelude> :l MonadicEvaluators.hs
[1 of 2] Compiling Parsing          ( Parsing.hs, interpreted )
[2 of 2] Compiling MonadicEvaluators ( MonadicEvaluators.hs, interpreted )

MonadicEvaluators.hs:163:7: error:
    Not in scope: data constructor ‘E’
    |
163 | runE (E f) = f
    |       ^
Failed, one module loaded.
*Parsing> :r
[2 of 2] Compiling MonadicEvaluators ( MonadicEvaluators.hs, interpreted )
Ok, two modules loaded.
*MonadicEvaluators> eval
eval     evalA    evalDo   evalM    eval_v1
*MonadicEvaluators> ex2
2+2
*MonadicEvaluators> ex3
(1+2)*3
*MonadicEvaluators> ex4
1+2*3
*MonadicEvaluators> ex5
x/(2*y)
*MonadicEvaluators> eval_v1 [("x",10),("y",1)] ex5
5
*MonadicEvaluators> eval_v1 [("x",10),("y",0)] ex5
*** Exception: divide by zero
*MonadicEvaluators> eval_v1 [("x",10),("z",0)] ex5
*** Exception: undefined variable: y
CallStack (from HasCallStack):
  error, called at MonadicEvaluators.hs:114:33 in main:MonadicEvaluators
*MonadicEvaluators> :t lookup
lookup :: Eq a => a -> [(a, b)] -> Maybe b
*MonadicEvaluators> :i Maybe
data Maybe a = Nothing | Just a         -- Defined in ‘GHC.Base’
instance Applicative Maybe -- Defined in ‘GHC.Base’
instance Eq a => Eq (Maybe a) -- Defined in ‘GHC.Base’
instance Functor Maybe -- Defined in ‘GHC.Base’
instance Monad Maybe -- Defined in ‘GHC.Base’
instance Monoid a => Monoid (Maybe a) -- Defined in ‘GHC.Base’
instance Ord a => Ord (Maybe a) -- Defined in ‘GHC.Base’
instance Show a => Show (Maybe a) -- Defined in ‘GHC.Show’
instance Read a => Read (Maybe a) -- Defined in ‘GHC.Read’
instance Foldable Maybe -- Defined in ‘Data.Foldable’
instance Traversable Maybe -- Defined in ‘Data.Traversable’
*MonadicEvaluators> :t guard
guard :: GHC.Base.Alternative f => Bool -> f ()
*MonadicEvaluators> :t guard False
guard False :: GHC.Base.Alternative f => f ()
*MonadicEvaluators> :t guard False :: Maybe Integer
```

```
<interactive>:1:1: error:
    • Couldn't match type '()' with 'Integer'
      Expected type: Maybe Integer
        Actual type: Maybe ()
    • In the expression: guard False :: Maybe Integer
*MonadicEvaluators>  guard False :: Maybe Integer

<interactive>:15:2: error:
    • Couldn't match type '()' with 'Integer'
      Expected type: Maybe Integer
        Actual type: Maybe ()
    • In the expression: guard False :: Maybe Integer
      In an equation for 'it': it = guard False :: Maybe Integer
*MonadicEvaluators>  guard False :: Maybe )=

<interactive>:16:2: error:
    Invalid type signature: guard False :: ...
    Should be of form <variable> :: <type>
*MonadicEvaluators>  guard False :: Maybe ()
Nothing
*MonadicEvaluators> :t join

<interactive>:1:1: error: Variable not in scope: join
*MonadicEvaluators> :r
[2 of 2] Compiling MonadicEvaluators ( MonadicEvaluators.hs, interpreted )
Ok, two modules loaded.
*MonadicEvaluators> :t join
join :: Monad m => m (m a) -> m a
*MonadicEvaluators> eval_v1 [("x",10),("y",0)] ex5
*** Exception: divide by zero
*MonadicEvaluators> evalA [("x",10),("y",0)] ex5
Nothing
*MonadicEvaluators> evalA [("x",10),("y",1)] ex5
Just 5
*MonadicEvaluators> ex5
x/(2*y)
*MonadicEvaluators> :r
[2 of 2] Compiling MonadicEvaluators ( MonadicEvaluators.hs, interpreted )
Ok, two modules loaded.
*MonadicEvaluators> :r
[2 of 2] Compiling MonadicEvaluators ( MonadicEvaluators.hs, interpreted )

MonadicEvaluators.hs:210:40: error:
    • Couldn't match type 'Char' with '(Env, Integer)'
      Expected type: [(Env, Integer)]
        Actual type: Name
    • In the second argument of 'lookup', namely 'x'
      In the expression: lookup env x
      In the expression:
        case lookup env x of
          Nothing -> Left ("Undefined variable: " ++ x)
          Just n -> Right n
    |
210 | lookupVar x = E (\env->case lookup env x of
    |                                        ^
Failed, one module loaded.
*Parsing> :r
[2 of 2] Compiling MonadicEvaluators ( MonadicEvaluators.hs, interpreted )
Ok, two modules loaded.
*MonadicEvaluators> eval
eval     evalA    evalDo   evalM    eval_v1
*MonadicEvaluators> ex5
x/(2*y)
*MonadicEvaluators> :t evalM ex5
evalM ex5 :: Eval Integer
*MonadicEvaluators> :t runE (evalM ex5)
runE (evalM ex5) :: Env -> Either ErrorMessage Integer
*MonadicEvaluators> runE (evalM ex5) [("x",10),("y",1)]
Right 5
*MonadicEvaluators> runE (evalM ex5) [("x",10),("y",0)]
Left "divide by zero"
*MonadicEvaluators> :t join
join :: Monad m => m (m a) -> m a
*MonadicEvaluators> :t \e1 e2-> safeDiv <$> evalM e1 <*> evalM e2
\e1 e2-> safeDiv <$> evalM e1 <*> evalM e2
  :: Expr -> Expr -> Eval (Eval Integer)
*MonadicEvaluators> :R
unknown command ':R'
use :? for help.
*MonadicEvaluators> :r
[2 of 2] Compiling MonadicEvaluators ( MonadicEvaluators.hs, interpreted )
Ok, two modules loaded.
*MonadicEvaluators> runE (evalM ex5) [("x",10),("y",0)]
Left "divide by zero"
*MonadicEvaluators> runE (evalM ex5) [("x",10),("y",1)]
Right 5
*MonadicEvaluators> runE (evalM ex5) [("x",10),("z",1)]
Left "Undefined variable: y"
*MonadicEvaluators> :r
[2 of 2] Compiling MonadicEvaluators ( MonadicEvaluators.hs, interpreted )
Ok, two modules loaded.
*MonadicEvaluators> main
Welcome to the simple calculator!
> 1+2+3
6
> x=10
> x*x
100
> x*y
Undefined variable: y
> y=1
> x/(y-1)
*** Exception: MonadicEvaluators.hs:(248,6)-(259,63): Non-exhaustive patterns in
 case

*MonadicEvaluators> :r
[2 of 2] Compiling MonadicEvaluators ( MonadicEvaluators.hs, interpreted )
Ok, two modules loaded.
```

```
*MonadicEvaluators> main
Welcome to the simple calculator!
> x=10
> y=0
> x/(2*y)
divide by zero
> x/y
divide by zero
> z+1
Undefined variable: z
> y=1
> x/y
10
> x/(y-1)
Syntax error!
> <*>^?^CInterrupted.
*MonadicEvaluators> :t (<*)
(<*) :: Applicative f => f a -> f b -> f a
*MonadicEvaluators>
Leaving GHCi.
ouzo2:code$
```