```
data Shape = Circle Float
           | Rect Float Float
           deriving Show
```

Add this

```
square :: Float -> Shape
square n = Rect n n

area :: Shape -> Float
area (Circle r) = pi * r^2
area (Rect x y) = x * y
```

```
ghci> square 5

<interactive>:51:1: error:
    * No instance for (Show Shape) arising from a use of `print'
    * In a stmt of an interactive GHCi command: print it
ghci>
```

```
ghci>
ghci>
ghci> square 5
Rect 5.0 5.0
ghci>
ghci>
ghci> area (Circle 3)
28.274334
ghci>
ghci>
ghci> area (Rect 3 4)
12.0
ghci>
ghci>
```

```
data Maybe' a = Nothing' | Just' a
```

Add this → `deriving Show`

```
safediv :: Int -> Int -> Maybe' Int
safediv _ 0 = Nothing'
safediv m n = Just' (m `div` n)

safehead :: [a] -> Maybe' a
safehead [] = Nothing'
safehead xs = Just' (head xs)
```

```
ghci>
ghci> safediv 5 0
Nothing'
ghci>
ghci> safediv 5 2
Just' 2
ghci>
ghci> safehead []
Nothing'
ghci>
ghci> safehead [1,2,3,4,5]
Just' 1
ghci>
ghci> safehead ['a', 'b', 'c', 'd', 'e']
Just' 'a'
ghci>
```

```
data Nat = Zero | Succ Nat
           deriving Show
```

```
nat2int :: Nat -> Int
nat2int Zero = 0
nat2int (Succ n) = 1 + nat2int n
```

```
int2nat :: Int -> Nat
int2nat 0 = Zero
int2nat n = Succ (int2nat (n-1))
```

```
ghci>
ghci> nat2int Zero
0
ghci>
ghci> nat2int (Succ Zero)
1
ghci>
ghci> nat2int (Succ(Succ Zero))
2
ghci>
ghci> nat2int (Succ(Succ(Succ Zero)))
3
```

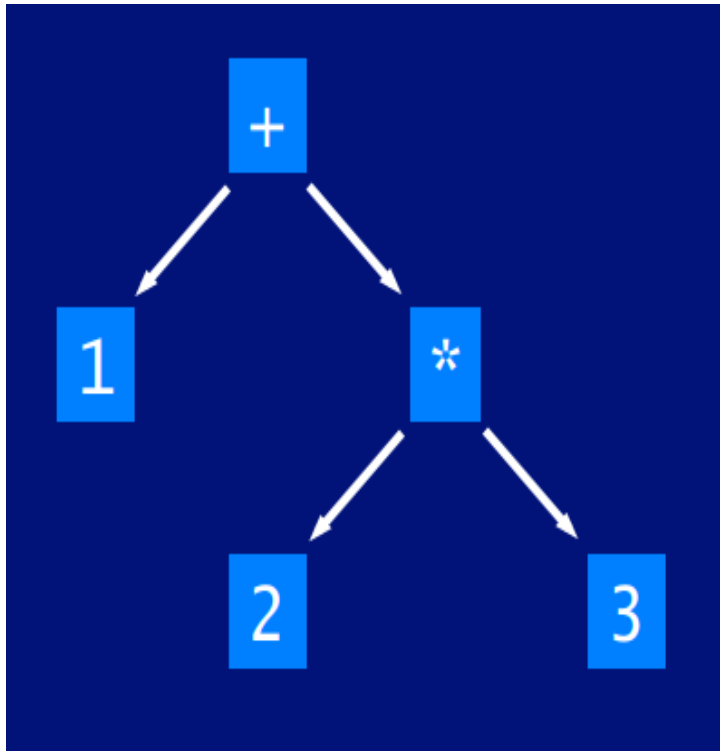
```
data Nat = Zero | Succ Nat
        deriving Show

nat2int :: Nat -> Int
nat2int Zero = 0
nat2int (Succ n) = 1 + nat2int n

int2nat :: Int -> Nat
int2nat 0 = Zero
int2nat n = Succ (int2nat (n-1))
```

```
ghci>
ghci> int2nat 0
Zero
ghci>
ghci> int2nat 1
Succ Zero
ghci>
ghci> int2nat 2
Succ (Succ Zero)
ghci>
ghci> int2nat 3
Succ (Succ (Succ Zero))
ghci>
```

```
add1 :: Nat -> Nat -> Nat
add1 m n = int2nat (nat2int m + nat2int n)

add2 :: Nat -> Nat -> Nat
add2 Zero n = n
add2 (Succ m) n = Succ (add2 m n)
```

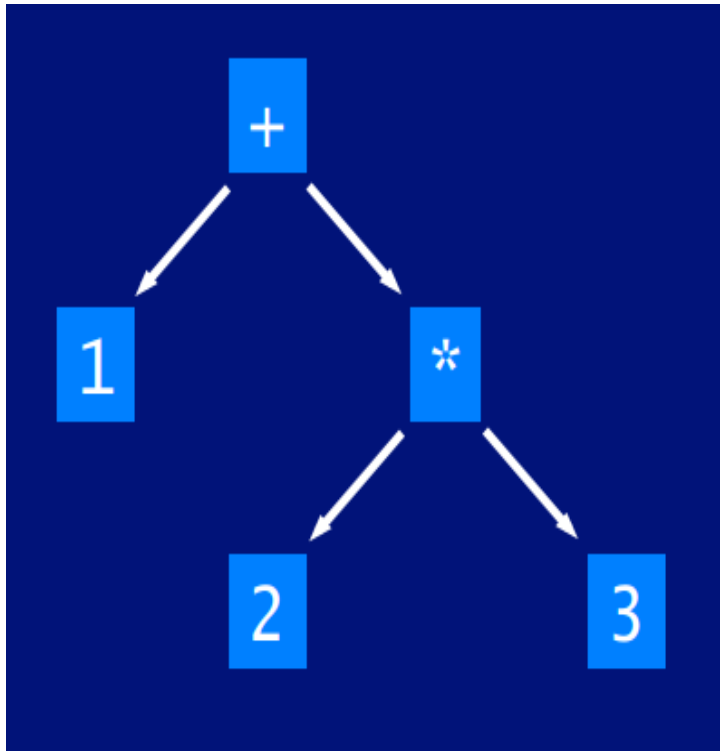
```
ghci>
ghci> add1 Zero (Succ Zero)
Succ Zero
ghci>
ghci> add1 (Succ Zero) (Succ(Succ Zero))
Succ (Succ (Succ Zero))
ghci>
ghci> add2 Zero (Succ Zero)
Succ Zero
ghci>
ghci> add2 (Succ Zero) (Succ(Succ Zero))
Succ (Succ (Succ Zero))
ghci>
```

```
data Expr = Val Int
  | Add Expr Expr
  | Mul Expr Expr

size :: Expr -> Int
size (Val n) = 1
size (Add x y) = size x + size y
size (Mul x y) = size x + size y

eval :: Expr -> Int
eval (Val n) = n
eval (Add x y) = eval x + eval y
eval (Mul x y) = eval x * eval y
```



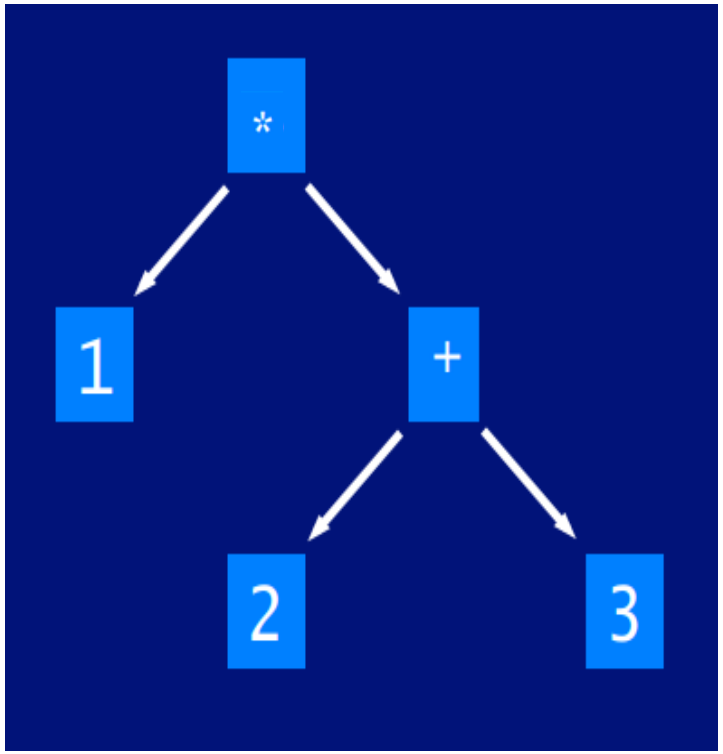
Add (Val 1) (Mul (Val 2) (Val 3))

```
data Expr = Val Int
  | Add Expr Expr
  | Mul Expr Expr

size :: Expr -> Int
size (Val n) = 1
size (Add x y) = size x + size y
size (Mul x y) = size x + size y

eval :: Expr -> Int
eval (Val n) = n
eval (Add x y) = eval x + eval y
eval (Mul x y) = eval x * eval y
```




```
ghci> size (Add (Val 1) (Mul (Val 2) (Val 3)))
3

ghci> eval (Add (Val 1) (Mul (Val 2) (Val 3)))
7
```

```
data Expr = Val Int
  | Add Expr Expr
  | Mul Expr Expr

size :: Expr -> Int
size (Val n) = 1
size (Add x y) = size x + size y
size (Mul x y) = size x + size y

eval :: Expr -> Int
eval (Val n) = n
eval (Add x y) = eval x + eval y
eval (Mul x y) = eval x * eval y
```




```
ghci> size (Mul (Val 1) (Add (Val 2) (Val 3)))
3

ghci> eval (Mul (Val 1) (Add (Val 2) (Val 3)))
5
```