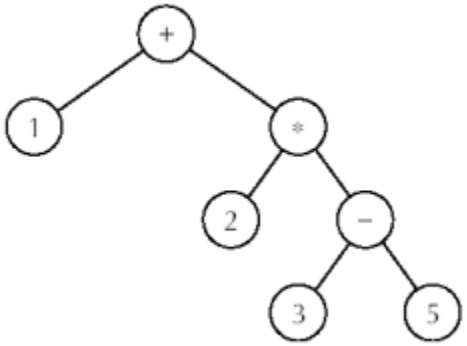
Trees

Not all data can or should be expressed in a linear way (as in a list). A very common data type which represents data hierarchically is a tree. For example, consider the arithmetic expression `1+2*(3-5)`. We can think of this expression as an *expression tree*;



We can represent this data in Haskell as:

```
data ExprTree = Leaf Float | Add ExprTree ExprTree | Sub ExprTree ExprTree |
                Mul ExprTree ExprTree | Div ExprTree ExprTree
                deriving Show
```

The internal nodes of the tree are represented as either `Add`, `Sub`, `Mul` or `Div` with two `ExprTree` values for the two branches. The *leaves* of the tree are represented with the `Leaf` and a floating point numeric value. The expression tree above would be written as:

```
(Add (Leaf 1) (Mul (Leaf 2) (Sub (Leaf 3) (Leaf 5))))
```

Given an expression tree, we might want to compute its value:

```
evaluate :: ExprTree -> Float
evaluate (Leaf x) = x
evaluate (Add e1 e2) = evaluate e1 + evaluate e2
evaluate (Sub e1 e2) = evaluate e1 - evaluate e2
evaluate (Mul e1 e2) = evaluate e1 * evaluate e2
evaluate (Div e1 e2) = evaluate e1 / evaluate e2
```

Notice that the function `evaluate` use a type of recursion called *tree recursion*. For each node, we recursively call the function on each of the two branches and combine the result. The base case of the recursion is when we get to a leaf.

The result of using `evaluate` would look like:

```
> evaluate (Add (Leaf 1) (Mul (Leaf 2) (Sub (Leaf 3) (Leaf 5))))
-3
```

Let's look at some more general trees and some operations upon them. We will have a tree where each node contains an `Int` value.

```
data IntTree = Leaf Int | Branch Int IntTree IntTree deriving Show
```

We can perform an operation to each of the elements in the tree:

```
mapTree :: (Int->Int) -> IntTree -> IntTree
mapTree f (Leaf x) = Leaf (f x)
mapTree f (Branch x t1 t2) = Branch (f x) (mapTree f t1) (mapTree f t2)

> mapTree (*2) (Branch 1 (Leaf 2) (Branch 3 (Leaf 4) (Leaf 5)))
Branch 2 (Leaf 4) (Branch 6 (Leaf 8) (Leaf 10))
```

We can count the number of leaves in the tree:

```
numLeaves :: IntTree -> Int
numLeaves (Leaf x) = 1
numLeaves (Branch x t1 t2) = numLeaves t1 + numLeaves t2
```

Notice the tree recursion in each of these examples.