```
fun total [] = 0.0
  | total h::t = amount(h) + total(t): h::t
total [(coins 37), (notes 6), cheque("cibc", 107.36)]
 => 0.37 + 6.0 + 107.36
```

```
[1, 2.0, 3.5, 4]
datatype number = i of int
                | r of real;
```

sum[r(1.0), i(7), r(3.4), i(5)]: number list => 1.0 + 7.0 + 3.4 + 5.0

sum: number list ⟶ real

```
fun sum [] = 0.0
  | sum (r(x) :: Tail) = x + sum(tail)
  | sum (i(N):: Tail) = real(N)+ sum(Tail)
```

```
Sum [i(1), r(7.3), i(2)]
    i(1):: [r(7.5), i(2)]
     1        tail
     n
 => real + sum[r(7.3), i(2)]
           ⋮
=> 1.0 + 7.3+ 2.0 + sum [] => 1.0+ 7.3+2.0 + 0.0 => 10.3
```

Recursive Datatypes

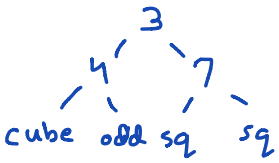

```
datatype tree = leaf of int
              | node of tree * tree
```

```
node(leaf(3), node(leaf (2), leaf(5))
```

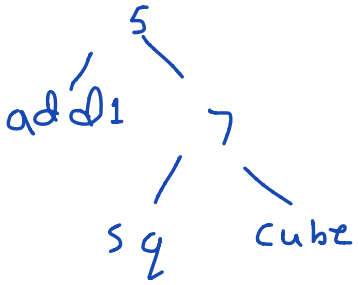

```
fun sum leaf(N) = N
  | sum node (T1, T2) = sum(T1) + sum(T2)
```

```
Sum: tree → int
```

higher-order recursive type




```
datatype tree = leaf of (int → int)
              | node of int · tree · tree
```




```
node(5, leaf(add1), node(7, leaf (sq), leaf(cube)))
```

```
gather: tree → (int → int) list
fun gather leaf(F) = [F]
  | gather node (N, T1, T2)
        = append (gather(T1), gather(T2));
```

```
sum: tree · int → int

fun sum (leaf (f), N) =    F(N)
  | sum (node (M, T1, T2), N) =
        M + sum(T1, N) + sum (T2, N);

sum: tree · int → int
```

Polymorphic recursive datatypes

```
node (leaf(1), node(leaf (2), leaf (3)): int tree

node (leaf (2), leaf ("abc")): error
```

```
datatype 'a tree = leaf of 'a
                 | node of ('a tree) * ('a tree)
                 | datatype intstrtree
                       = leaf1 of int
                       | leaf 2 of string
                       | node of (intstrtree) * (intstrtree)
```

```
fun count leaf(x) = 1
  | count node (T1, T2) = count (T1) + count (T2);

count : 'a tree → int
```