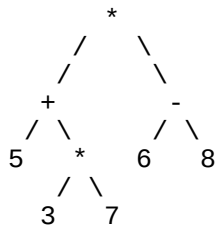
PROBLEM SHEET 6.

A tree is a data structure made up of a hierarchy of things called nodes. The example below is of a tree which holds an arithmetic expression:



ie: (5 + 3 * 7) * (6 - 8)

Each node has a piece of data attached to it as well as a number of branches. Data in the above tree is either an arithmetic operator or a number. Generally the node at the top of a tree is known as the 'root node', nodes at the bottom of a tree have no branches and are called 'terminal nodes' or 'leaf nodes' (get it? root, branch, leaf & tree). In an expression tree like the one above, all terminal nodes are numbers.

In list form the above tree would be represented as:

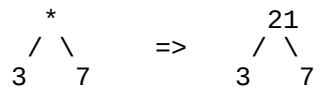
```
'(* (+ 5 (* 3 7)) (- 6 8))
```

where each node is of the form (data left-tree right-tree).

PROBLEM 6.1

Produce a function which takes an expression tree as its argument and returns that tree with the relevant calculated values in place of the operators.

Eg:



So:
```
(evaltree '(* (+ 5 (* 3 7)) (- 6 8)) )
==> (-52 (26 5 (21 3 7)) (-2 6 8))
```

HINT : apply is a CL function which applies other functions, eg:
```
(apply '* '(3 4))  ==>  12
```

PROBLEM 6.2

Produce a function which takes an expression tree as its argument and returns the expression which it contains in infix form containing brackets only where necessary.

So:
```
(rewrite '(* (+ 5 (* 3 7)) (- 6 8)) )
==> ((5 + 3 * 7) * (6 - 8))
```

For both presentations you may assume that each arithmetic operator will always have two operands and that the data you are given is error-free.