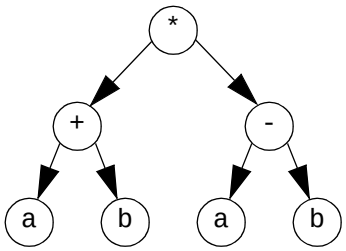

Figure 11.22 Expression tree for `(a+b)*(a-b)`

Similarly, if a `^` is on the operator stack and is also the input symbol, it will appear that the operator that is on the top of the stack has lower precedence. Thus it will not be popped, which is correct for right-associative operators. The token `VALUE` never gets placed on the stack, so its precedence is meaningless. The end of line token is given lowest precedence because it will be placed on the stack for use as a sentinel (this is done in the constructor). If we treat it as a right-associative operator, then it is covered under the operator case.

The remaining method is `processToken`, which is shown in Figure 11.20 (page 324). When we see an operand, it is pushed onto the postfix stack. If we see a close parenthesis, we repeatedly pop and process the top operator on the operator stack until the opening parenthesis is seen (lines 19 to 21). The open parenthesis is then popped at line 23. (The test at line 22 is used to avoid popping the sentinel in the event of a missing open parenthesis.) Otherwise, we have the general operator case, which is succinctly described by the code in lines 29 to 34. A simple `main` routine is given in Figure 11.21 (page 325). It repeatedly reads a line of input, instantiates an `Evaluator` object, and computes its value.

11.2.4 Expression Trees

In an *expression tree*, the leaves contain operands and the other nodes contain operators.

Figure 11.22 shows an example of an *expression tree*. The leaves of an expression tree are operands such as constants or variable names, and the other nodes contain operators. This particular tree happens to be binary because all of the operations are binary. Although this is the simplest case, it is possible for nodes to have more than two children. It is also possible for a node to have only one child, as is the case with the unary minus operator.

We evaluate an expression tree *T* by applying the operator at the root to the values obtained by recursively evaluating the left and right subtrees. In this example, the left subtree evaluates to `(a+b)` and the right subtree evaluates to `(a-b)`. The entire tree therefore represents `((a+b)*(a-b))`. It is evident that we can produce an (overly parenthesized) infix expression by recursively producing a parenthesized left expression, then printing out the operator at the root, and