**Traversal**[[edit](https://en.wikipedia.org/w/index.php?title=Binary_expression_tree&action=edit&section=2" \o "Edit section: Traversal)]

An algebraic expression can be produced from a binary expression tree by recursively producing a parenthesized left expression, then printing out the operator at the root, and finally recursively producing a parenthesized right expression. This general strategy (left, node, right) is known as an [in-order traversal](https://en.wikipedia.org/wiki/Tree_traversal). An alternate traversal strategy is to recursively print out the left subtree, the right subtree, and then the operator. This traversal strategy is generally known as [post-order traversal](https://en.wikipedia.org/wiki/Tree_traversal). A third strategy is to print out the operator first and then recursively print out the left and right subtree.[[2]](https://en.wikipedia.org/wiki/Binary_expression_tree#cite_note-Gopal2010-2)

These three standard depth-first traversals are representations of the three different expression formats: infix, postfix, and prefix. An infix expression is produced by the inorder traversal, a postfix expression is produced by the post-order traversal, and a prefix expression is produced by the pre-order traversal.[[3]](https://en.wikipedia.org/wiki/Binary_expression_tree#cite_note-Gilberg-3)

**Infix traversal**[[edit](https://en.wikipedia.org/w/index.php?title=Binary_expression_tree&action=edit&section=3" \o "Edit section: Infix traversal)]

When an infix expression is printed, an opening and closing parenthesis must be added at the beginning and ending of each expression. As every subtree represents a subexpression, an opening parenthesis is printed at its start and the closing parenthesis is printed after processing all of its children.

Pseudocode:

Algorithm infix (tree)

*/\*Print the infix expression for an expression tree.*

*Pre : tree is a pointer to an expression tree*

*Post: the infix expression has been printed\*/*

**if** (tree not empty)

**if** (tree token is operator)

print (open parenthesis)

end **if**

infix (tree left subtree)

print (tree token)

infix (tree right subtree)

**if** (tree token is operator)

print (close parenthesis)

end **if**

end **if**

end infix

**Postfix traversal**[[edit](https://en.wikipedia.org/w/index.php?title=Binary_expression_tree&action=edit&section=4" \o "Edit section: Postfix traversal)]

The postfix expression is formed by the basic postorder traversal of any binary tree. It does not require parentheses.

Pseudocode:

Algorithm postfix (tree)

*/\*Print the postfix expression for an expression tree.*

*Pre : tree is a pointer to an expression tree*

*Post: the postfix expression has been printed\*/*

**if** (tree not empty)

postfix (tree left subtree)

postfix (tree right subtree)

print (tree token)

end **if**

end postfix

**Prefix traversal**[[edit](https://en.wikipedia.org/w/index.php?title=Binary_expression_tree&action=edit&section=5" \o "Edit section: Prefix traversal)]

The prefix expression formed by prefix traversal uses the standard pre-order tree traversal. No parentheses are necessary.

Pseudocode:

Algorithm prefix (tree)

*/\*Print the prefix expression for an expression tree.*

*Pre : tree is a pointer to an expression tree*

*Post: the prefix expression has been printed\*/*

**if** (tree not empty)

print (tree token)

prefix (tree left subtree)

prefix (tree right subtree)

end **if**

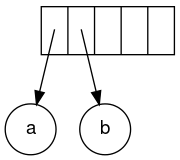
end prefix

Construction of an expression tree[[edit](https://en.wikipedia.org/w/index.php?title=Binary_expression_tree&action=edit&section=6" \o "Edit section: Construction of an expression tree)]

The evaluation of the tree takes place by reading the postfix expression one symbol at a time. If the symbol is an operand, one-node tree is created and a pointer is pushed onto a [stack](https://en.wikipedia.org/wiki/Stack_(abstract_data_type)). If the symbol is an operator, the pointers are popped to two trees *T1* and *T2* from the stack and a new tree whose root is the operator and whose left and right children point to *T2* and *T1* respectively is formed . A pointer to this new tree is then pushed to the Stack.[[4]](https://en.wikipedia.org/wiki/Binary_expression_tree#cite_note-4)

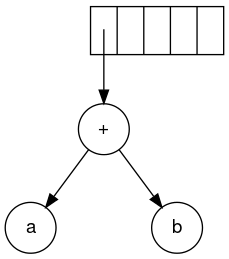
**Example**[[edit](https://en.wikipedia.org/w/index.php?title=Binary_expression_tree&action=edit&section=7" \o "Edit section: Example)]

The input is: a b + c d e + \* \* Since the first two symbols are operands, one-node trees are created and pointers are pushed to them onto a stack. For convenience the stack will grow from left to right.

[](https://en.wikipedia.org/wiki/File:Exp-tree-ex-2.svg)

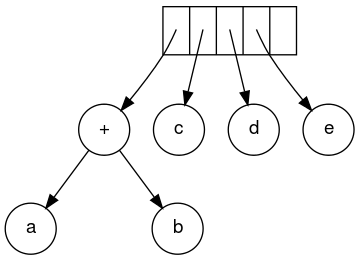
Stack growing from left to right

The next symbol is a '+'. It pops the two pointers to the trees, a new tree is formed, and a pointer to it is pushed onto to the stack.

[](https://en.wikipedia.org/wiki/File:Exp-tree-ex-3.svg)

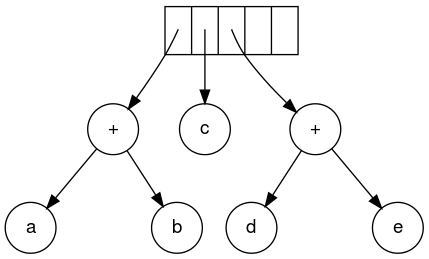
Formation of a new tree

Next, c, d, and e are read. A one-node tree is created for each and a pointer to the corresponding tree is pushed onto the stack.

[](https://en.wikipedia.org/wiki/File:Exp-tree-ex-6.svg)

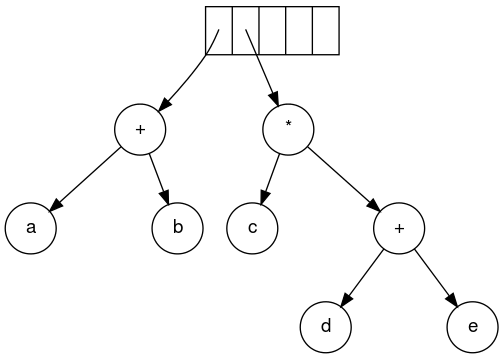
Creating a one-node tree

Continuing, a '+' is read, and it merges the last two trees.

[](https://en.wikipedia.org/wiki/File:Exp-tree-ex-7.svg)

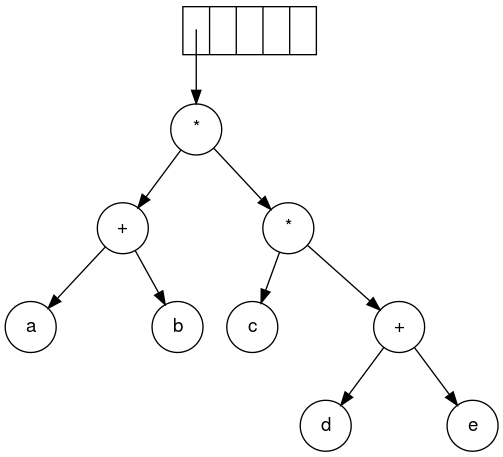
Merging two trees

Now, a '\*' is read. The last two tree pointers are popped and a new tree is formed with a '\*' as the root.

[](https://en.wikipedia.org/wiki/File:Exp-tree-ex-8.svg)

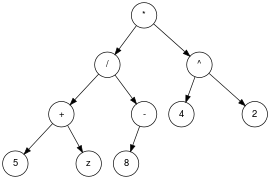
Forming a new tree with a root

Finally, the last symbol is read. The two trees are merged and a pointer to the final tree remains on the stack.[[5]](https://en.wikipedia.org/wiki/Binary_expression_tree#cite_note-Gopal2010.353-5)

[](https://en.wikipedia.org/wiki/File:Exp-tree-ex-9.svg)

Steps to construct an expression tree a b + c d e + \* \*

Algebraic expressions[[edit](https://en.wikipedia.org/w/index.php?title=Binary_expression_tree&action=edit&section=8)]

[](https://en.wikipedia.org/wiki/File:Exp-tree-ex-12.svg)

Binary algebraic expression tree equivalent to ((5 + z) / -8) \* (4 ^ 2)

Algebraic expression trees represent expressions that contain [numbers](https://en.wikipedia.org/wiki/Number), [variables](https://en.wikipedia.org/wiki/Variable_(mathematics)), and unary and binary operators. Some of the common operators are × ([multiplication](https://en.wikipedia.org/wiki/Multiplication)), ÷ ([division](https://en.wikipedia.org/wiki/Division_(mathematics))), + ([addition](https://en.wikipedia.org/wiki/Addition)), − ([subtraction](https://en.wikipedia.org/wiki/Subtraction)), ^ ([exponentiation](https://en.wikipedia.org/wiki/Exponentiation)), and - ([negation](https://en.wikipedia.org/wiki/Negation)). The operators are contained in the [internal nodes](https://en.wikipedia.org/wiki/Internal_node) of the tree, with the numbers and variables in the [leaf nodes](https://en.wikipedia.org/wiki/Leaf_nodes).[[1]](https://en.wikipedia.org/wiki/Binary_expression_tree#cite_note-brpreiss-1) The nodes of binary operators have two [child nodes](https://en.wikipedia.org/wiki/Child_nodes), and the unary operators have one child node.