



WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Featured content](#)
[Current events](#)
[Random article](#)
[Donate to Wikipedia](#)
[Wikipedia store](#)

Interaction

[Help](#)
[About Wikipedia](#)
[Community portal](#)
[Recent changes](#)
[Contact page](#)


Tools

[What links here](#)
[Related changes](#)
[Upload file](#)
[Special pages](#)
[Permanent link](#)
[Page information](#)
[Wikidata item](#)
[Cite this page](#)

Print/export

[Create a book](#)
[Download as PDF](#)
[Printable version](#)


Languages

 [Add links](#)

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)



Binary expression tree

From Wikipedia, the free encyclopedia
(Redirected from [Expression tree](#))

A **binary expression tree** is a specific application of a [binary tree](#) to evaluate certain expressions. Two common types of expressions that a binary expression tree can represent are [algebraic](#)^[1] and [boolean](#). These trees can represent expressions that contain both [unary](#) and [binary](#) operators.^[1]

In general, expression trees are a special kind of binary tree. A binary tree is a tree in which all nodes contain zero, one or two children. This restricted structure simplifies the programmatic processing of Expression trees.

Contents [\[hide\]](#)

- 1 Overview
 - 1.1 Traversal
 - 1.1.1 Infix Traversal
 - 1.1.2 Postfix Traversal
 - 1.1.3 Prefix Traversal
- 2 Construction of an Expression Tree
 - 2.1 Example
- 3 Algebraic expressions
- 4 Boolean expressions
- 5 See also
- 6 References

Overview [\[edit\]](#)

The leaves of a binary expression tree are operands, such as constants or variable names, and the other nodes contain operators. These particular trees happen to be binary, because all of the operations are binary, and 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. An expression tree, *T*, can be evaluated by applying the operator at the root to the values obtained by recursively evaluating the left and right subtrees.^[2]

Traversal [\[edit\]](#)

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](#). 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](#). A third strategy is to print out the operator first and then recursively print out the left and right subtree.^[2]

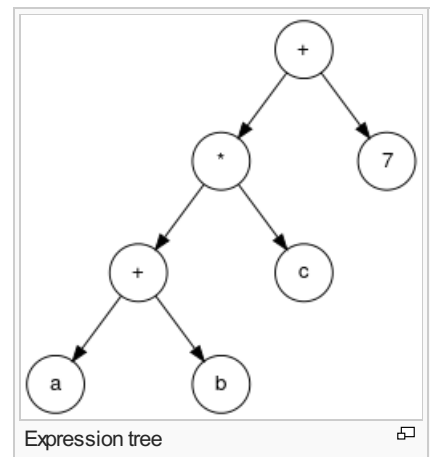
These three standard depth-first traversals are representations of the three different expression formats: infix, postfix, and prefix. An infix expression is produced by the inorder traversal, a postfix expression is produced by the post-order traversal, and a prefix expression is produced by the pre-order traversal.^[3]

Infix Traversal [\[edit\]](#)

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 [\[edit\]](#)

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 [\[edit\]](#)

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) and check if stack is not empty
end if
end prefix

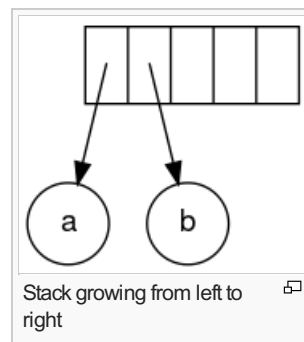
```

Construction of an Expression Tree [\[edit\]](#)

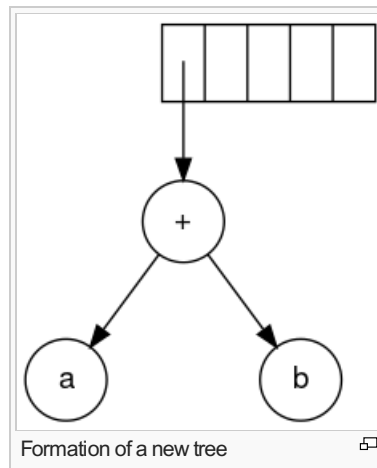
The evaluation of the tree takes place by reading the postfix expression one symbol at a time. If the symbol is an operand, one-node tree is created and a pointer is pushed onto a [stack](#). If the symbol is an operator, the pointers are popped to two trees T_1 and T_2 from the stack and a new tree whose root is the operator and whose left and right children point to T_2 and T_1 respectively is formed. A pointer to this new tree is then pushed to the Stack.^[4]

Example [\[edit\]](#)

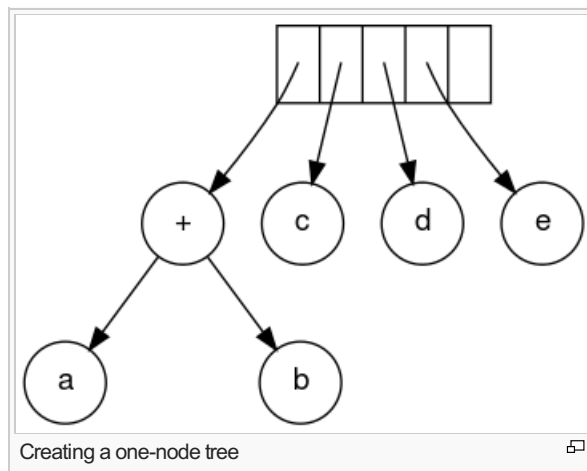
The input is: a b + c d e + * * Since the first two symbols are operands, one-node trees are created and pointers are pushed to them onto a stack. For convenience the stack will grow from left to right.



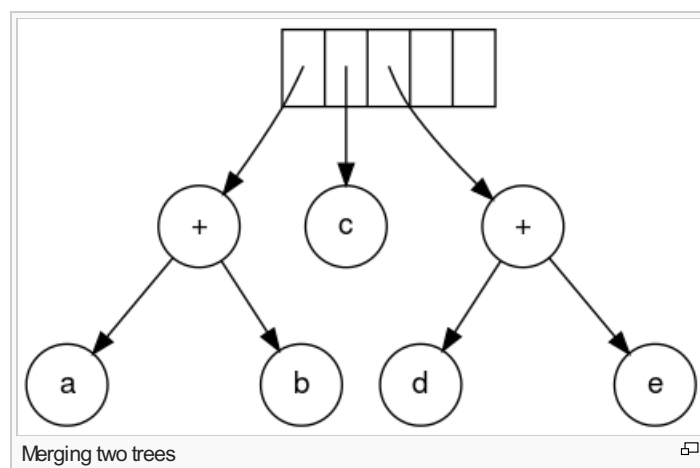
The next symbol is a '+'. It pops the two pointers to the trees, a new tree is formed, and a pointer to it is pushed onto the stack.



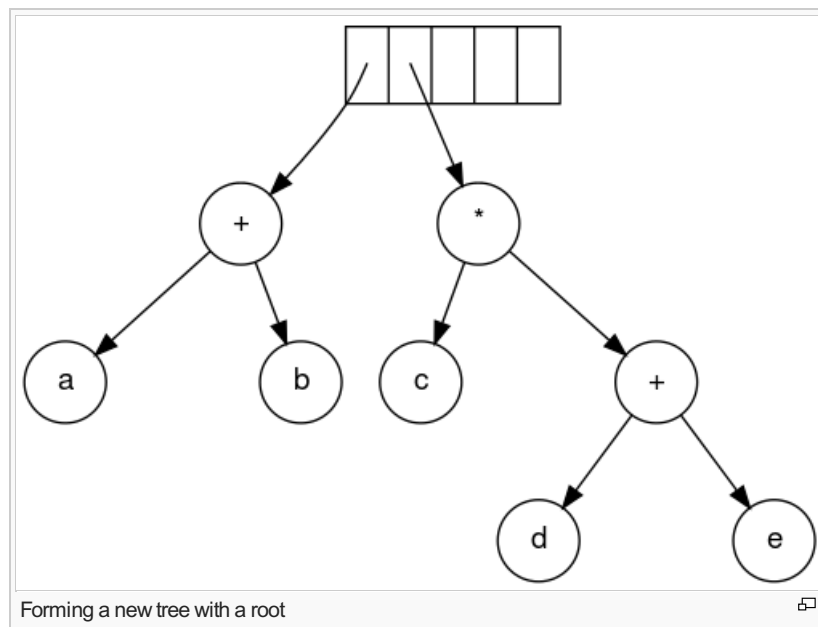
Next, c, d, and e are read. A one-node tree is created for each and a pointer to the corresponding tree is pushed onto the stack.



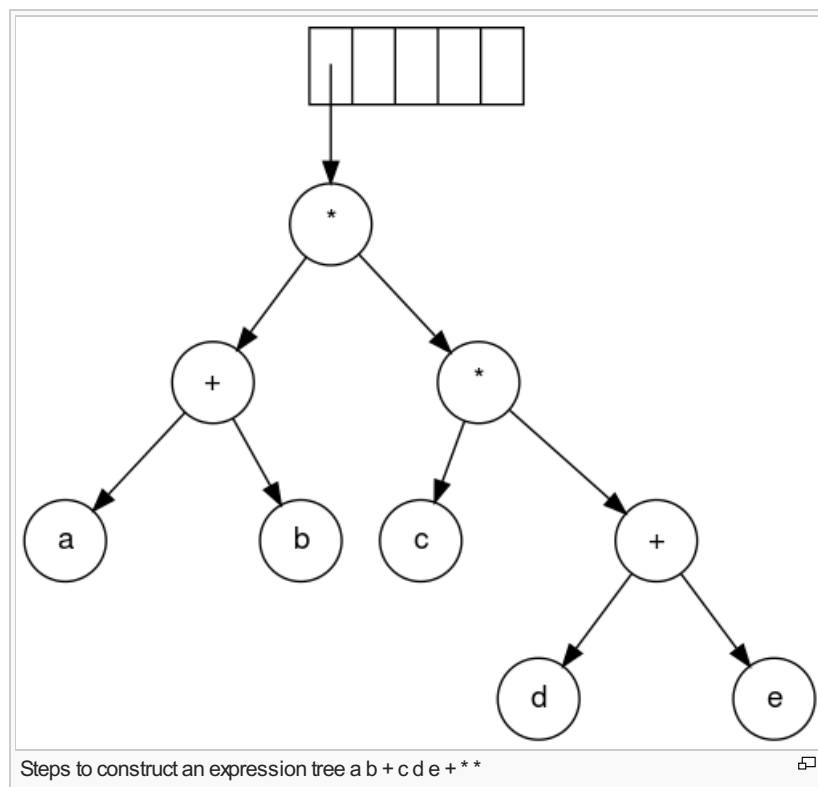
Continuing, a '+' is read, and it merges the last two trees.



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

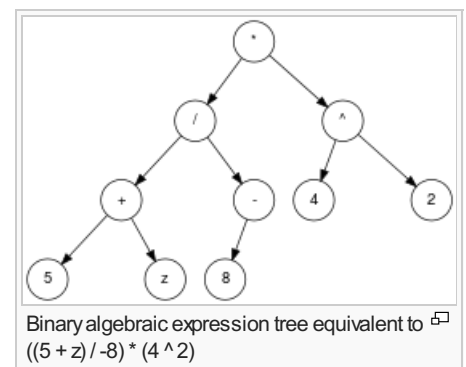


Finally, the last symbol is read. The two trees are merged and a pointer to the final tree remains on the stack.^[5]



Algebraic expressions [\[edit\]](#)

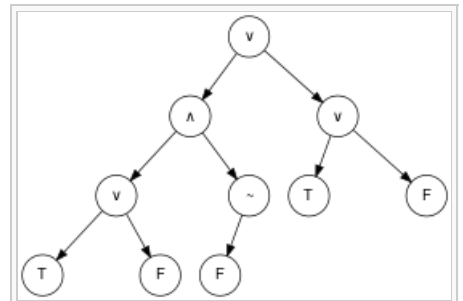
Algebraic expression trees represent expressions that contain [numbers](#), [variables](#), and unary and binary operators. Some of the common operators are \times ([multiplication](#)), \div ([division](#)), $+$ ([addition](#)), $-$ ([subtraction](#)), $^$ ([exponentiation](#)), and $-$ ([negation](#)). The operators are contained in the [internal nodes](#) of the tree, with the numbers and variables in the [leaf nodes](#).^[1] The nodes of binary operators have two [child nodes](#), and the unary operators have one child node.



Boolean expressions [\[edit\]](#)

Boolean expressions are represented very similarly to algebraic expressions, the only difference being the

specific values and operators used. Boolean expressions use *true* and *false* as constant values, and the operators include \wedge (AND), \vee (OR), \neg (NOT).



Binary boolean expression tree equivalent to $((\text{true} \vee \text{false}) \wedge \neg \text{false}) \vee (\text{true} \vee \text{false})$

See also [[edit](#)]

- [Expression \(coding\)](#)
- [Expression \(mathematics\)](#)
- [Term \(logic\)](#)
- [Context-free grammar](#)
- [Parse tree](#)

References [[edit](#)]

- ↑ ***a b c*** Bruno R. Preiss (1998). "Expression Trees" . Retrieved December 20, 2010.
- ↑ ***a b*** Gopal, Arpita. *Magnifying Data Structures*. PHI Learning, 2010, p. 352.
- ↑ Richard F. Gilberg & Behrouz A. Forouzan. *Data Structures: A Pseudocode Approach with C*. Thomson Course Technology, 2005, p. 280.
- ↑ Mark Allen Weiss, *Data Structures and Algorithm Analysis in C, 2nd edition*, Addison Wesley publications
- ↑ Gopal, Arpita. *Magnifying Data Structures*. PHI Learning, 2010, p. 353.

Categories: [Binary trees](#)

This page was last modified on 1 July 2015, at 09:33.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

