CHAPTER 6 TREES
PART 1

Terminology, Binary Tree, Expression Tree

Key Topics in Chapter 6

- Use a tree to represent a hierarchical organization of information
- Use recursion to process trees
- Different ways of traversing a tree
- Differences between binary trees, binary search trees, and heaps
- Implement binary trees, binary search trees, and heaps using linked data structures and arrays

Key Topics in Chapter 6

- Use binary search trees to store information for efficient retrieval
- Use a Huffman tree to encode characters efficiently
  - Used in compression

Key Topics in This PPT File

- Basic Terminologies about Trees
- Binary Tree
- Expression Tree

Trees - Introduction

- All previous data organizations we've studied
  - Are linear
  - Each element can have only one predecessor and successor
  - Accessing all elements in a linear sequence is $O(n)$
- Trees
  - Nonlinear and hierarchical
  - Each node can have multiple successors
    - But only one predecessor

Trees - Introduction (cont.)

- Examples of Tree applications
  - Store **hierarchical organizations** of information
    - class hierarchy
    - disk directory and subdirectories
    - family tree (single-parent)
  - Searching (via search tree)
  - Sorting (via heap)
  - Compression (via Huffman Tree)
- Trees are **recursive data structures**
  - Can be defined recursively
- Many methods to process trees are written recursively

Trees - Introduction (cont.)

Binary Trees

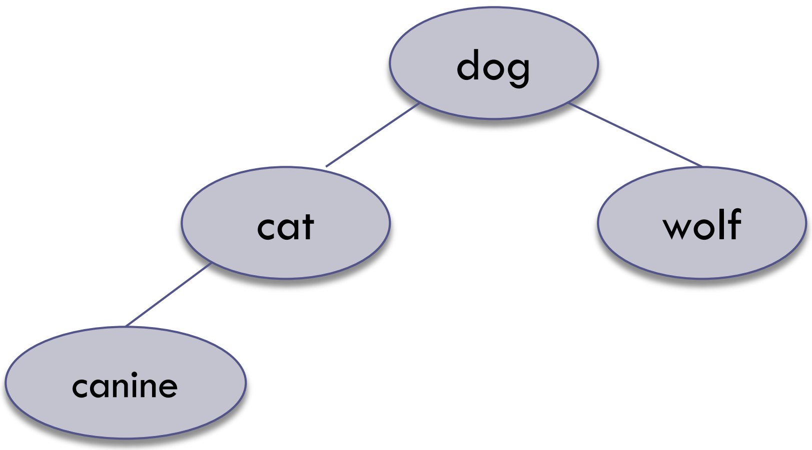
- Focus of Chapter 6
- **Each element has at most two successors**
- **Can be represented by arrays or by linked data structures**
- **Searching a binary search tree,** a sorted tree
  - Generally **more efficient than searching an unsorted list**
  - **O(log *n*) (if balanced) versus O(n)**

Tree Terminology and Applications

Section 6.1

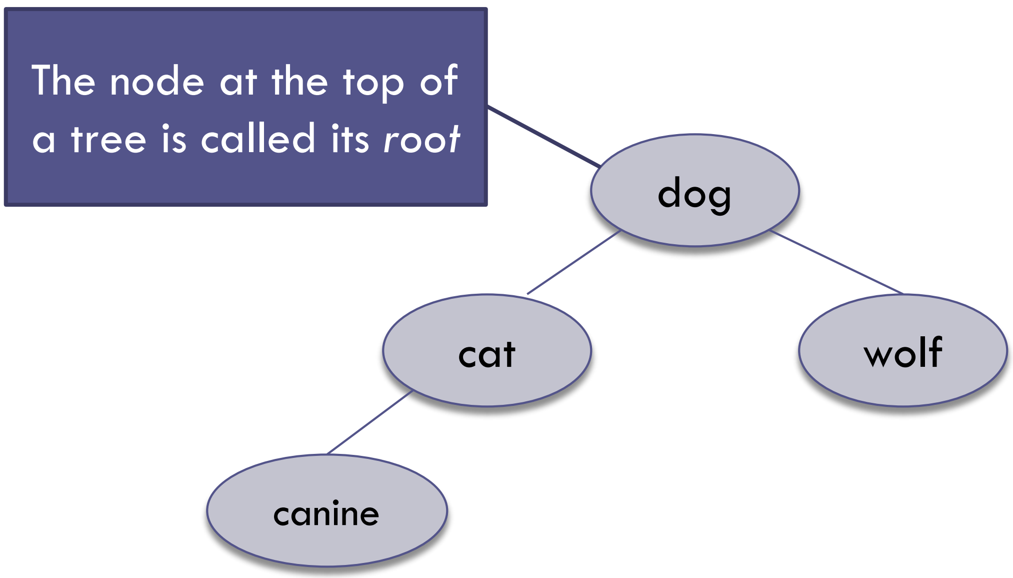
Tree Terminology: Tree

A tree consists of a collection of elements or nodes, with each node linked to its successors



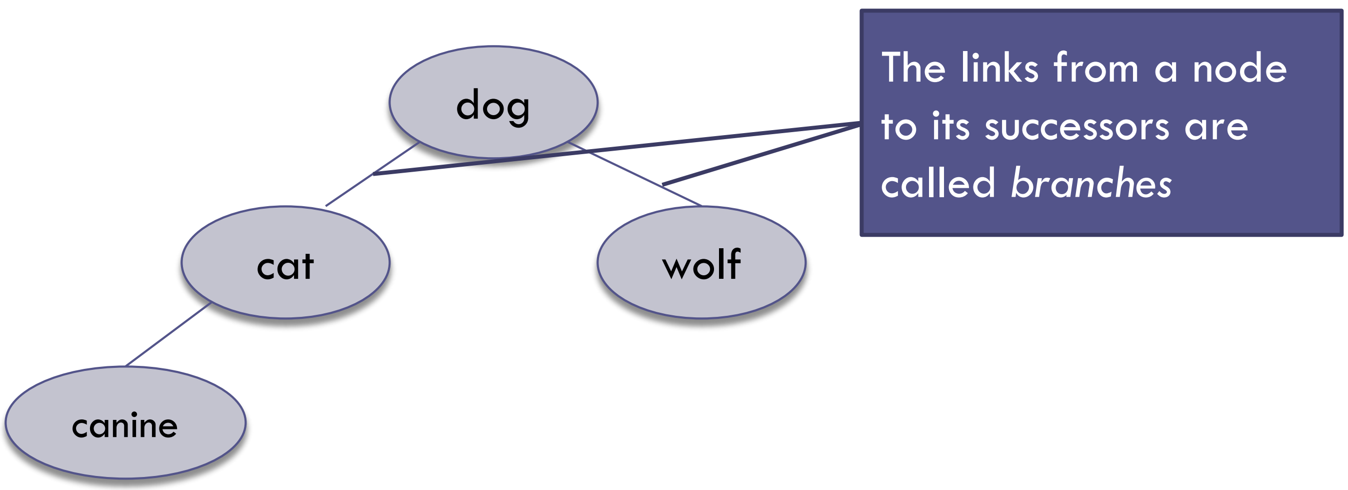
Tree Terminology: Root

The node at the top of a tree is called its *root*



Tree Terminology: Branch

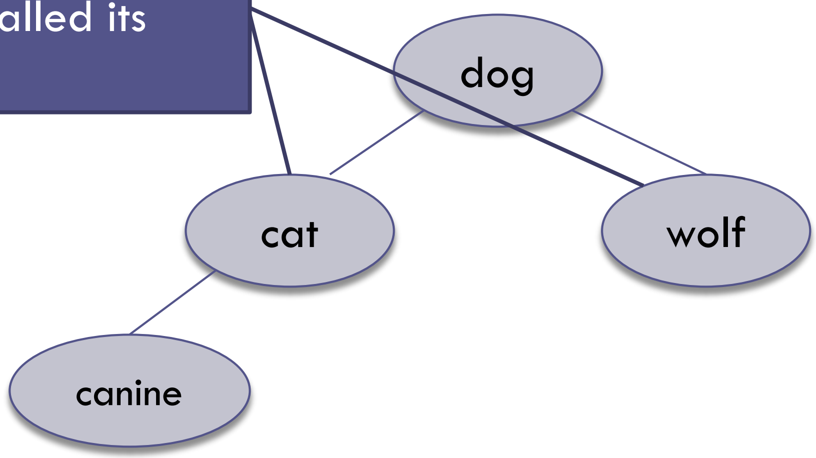
The links from a node to its successors are called *branches*



Tree Terminology: Children

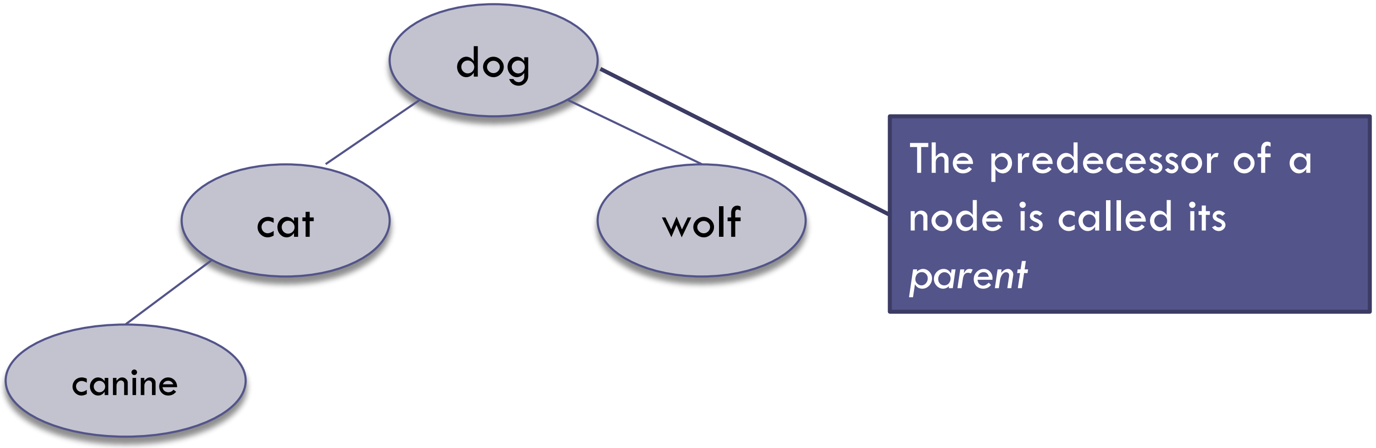
The successors of a node are called its *children*

The successors of a node are called its *children*



Tree Terminology: Parent

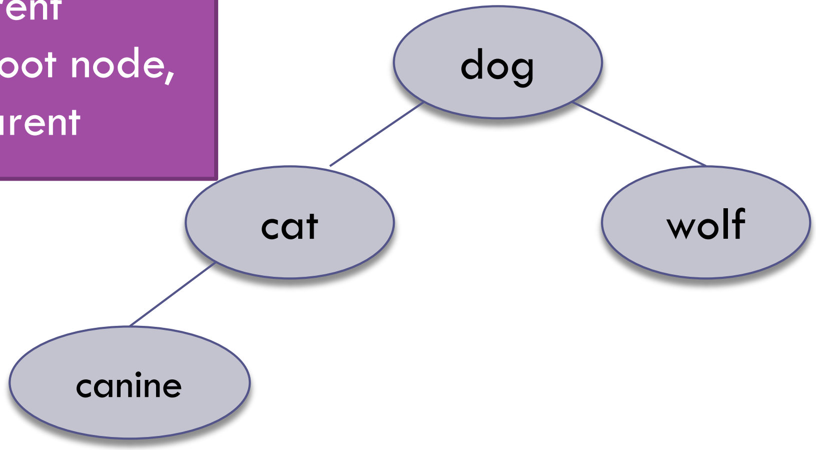
The predecessor of a node is called its *parent*



Tree Terminology: Parent

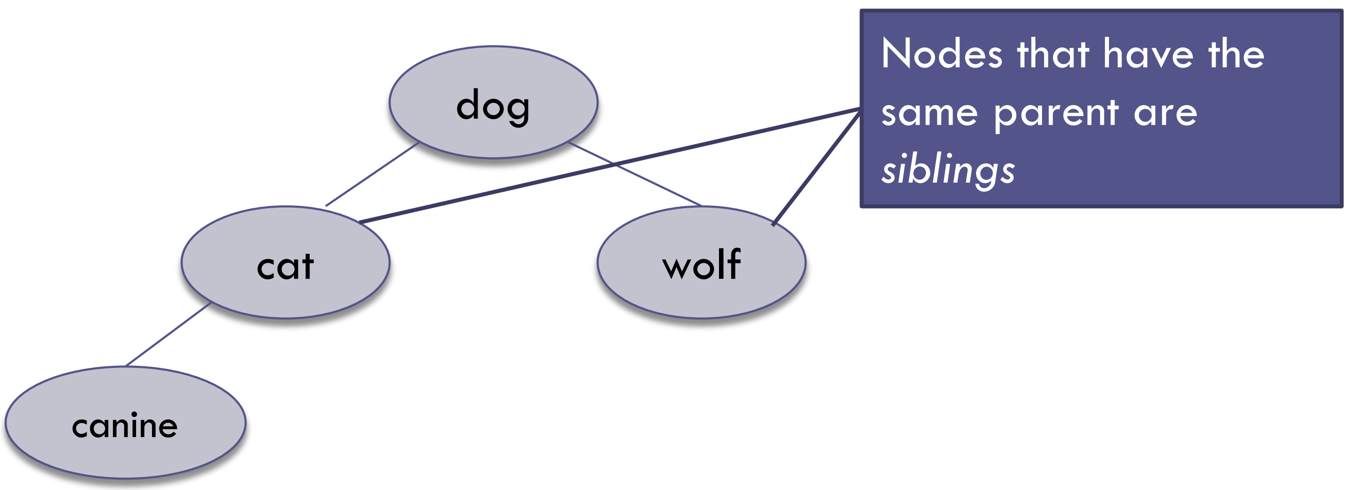
Each node in a tree has exactly one parent except for the root node, which has no parent

Each node in a tree has exactly one parent except for the root node, which has no parent



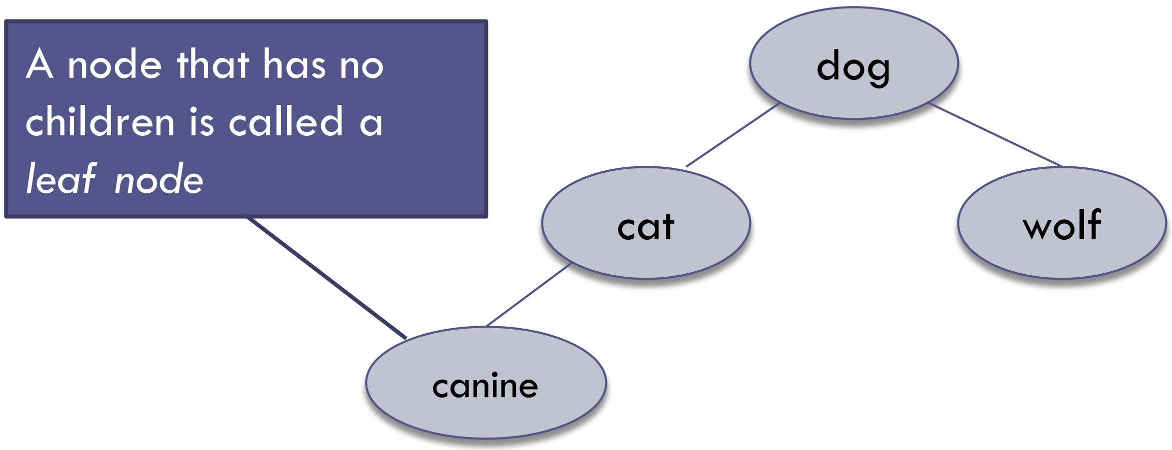
Tree Terminology: Sibling

Nodes that have the same parent are *siblings*



Tree Terminology: Leaf node

A node that has no children is called a *leaf node*

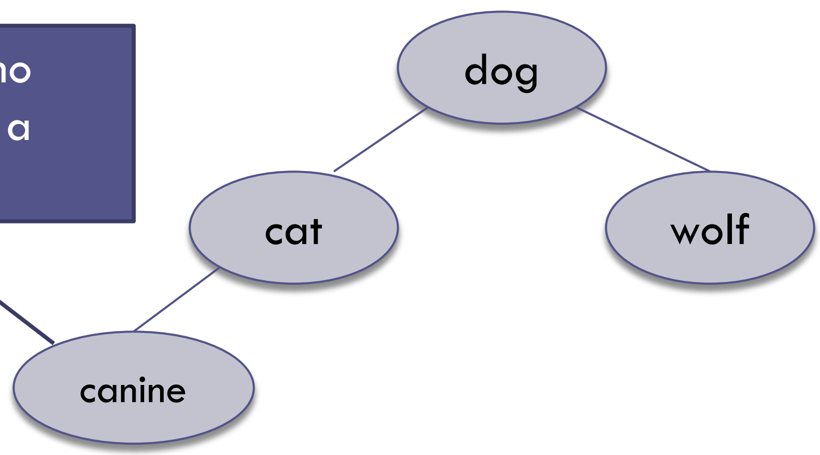


Tree Terminology: Leaf node

Leaf nodes - also called *external nodes*

Nonleaf nodes are called *internal nodes*

A node that has no children is called a *leaf node*

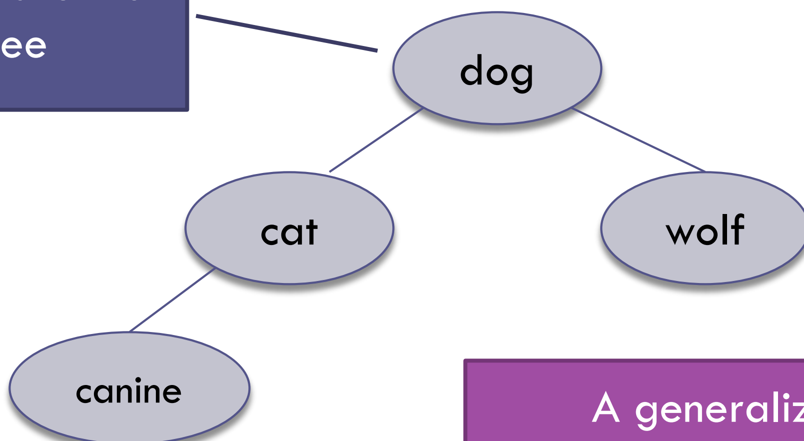


Leaf nodes also are known as *external nodes*, and nonleaf nodes are known as *internal nodes*

Tree Terminology: *Ancestor-Descendant*

A generalization of the parent-child relationship is the *ancestor-descendant relationship*

dog is the parent of cat in this tree

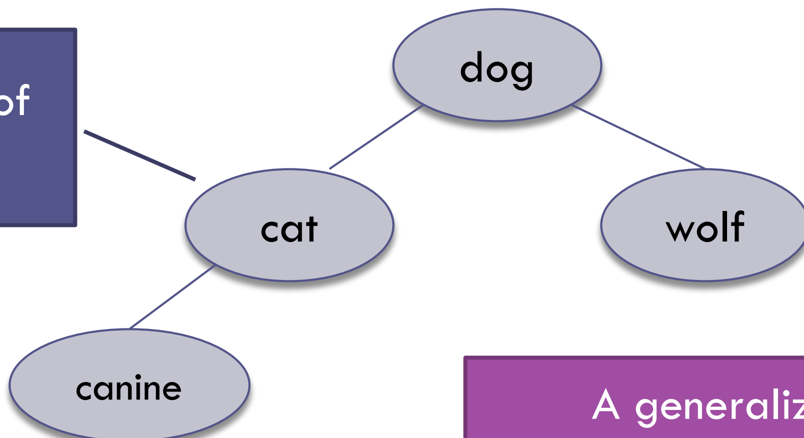


A generalization of the parent-child relationship is the *ancestor-descendant relationship*

Tree Terminology: *Ancestor-Descendant*

A generalization of the parent-child relationship is the *ancestor-descendant relationship*

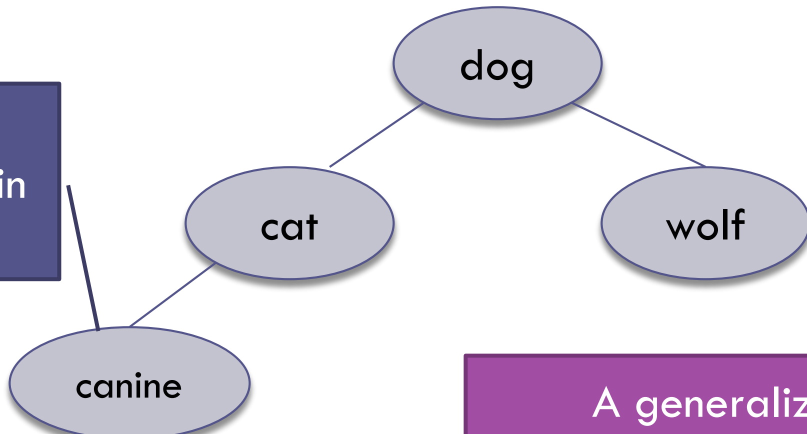
cat is the parent of canine in this tree



A generalization of the parent-child relationship is the *ancestor-descendant relationship*

Tree Terminology: *Ancestor-Descendant*

canine is a *descendant* of cat in this tree

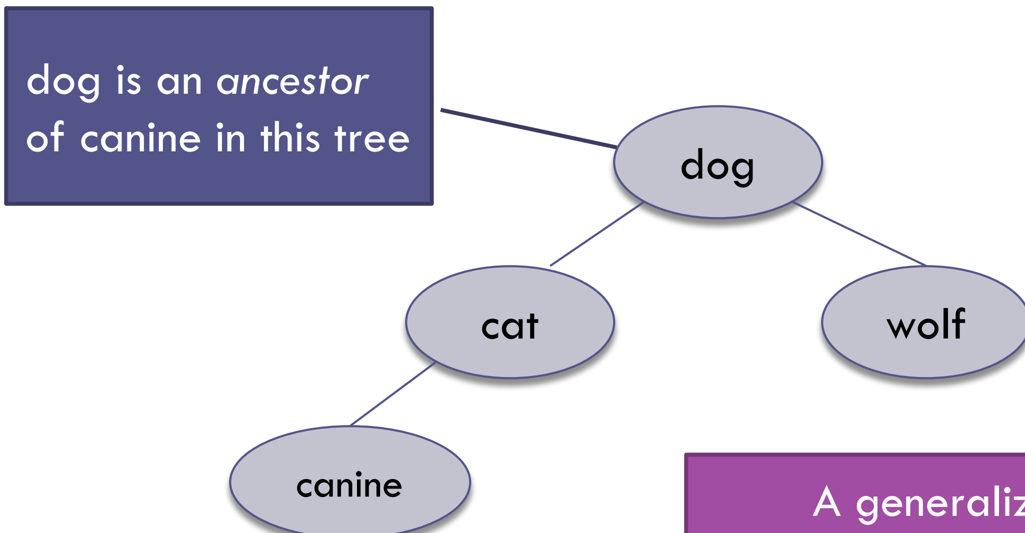


canine is a *descendant* of cat in this tree

A generalization of the parent-child relationship is the *ancestor-descendant relationship*

Tree Terminology: *Ancestor-Descendant*

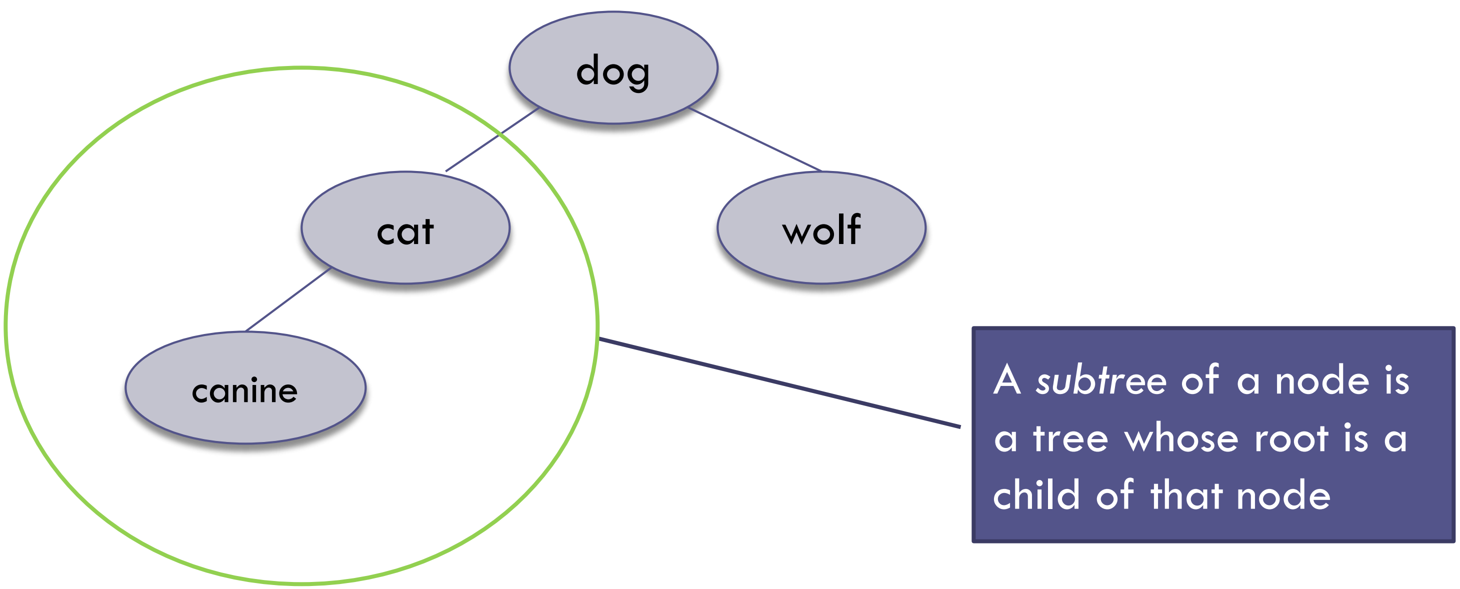
dog is an *ancestor* of canine in this tree



A generalization of the parent-child relationship is the *ancestor-descendant relationship*

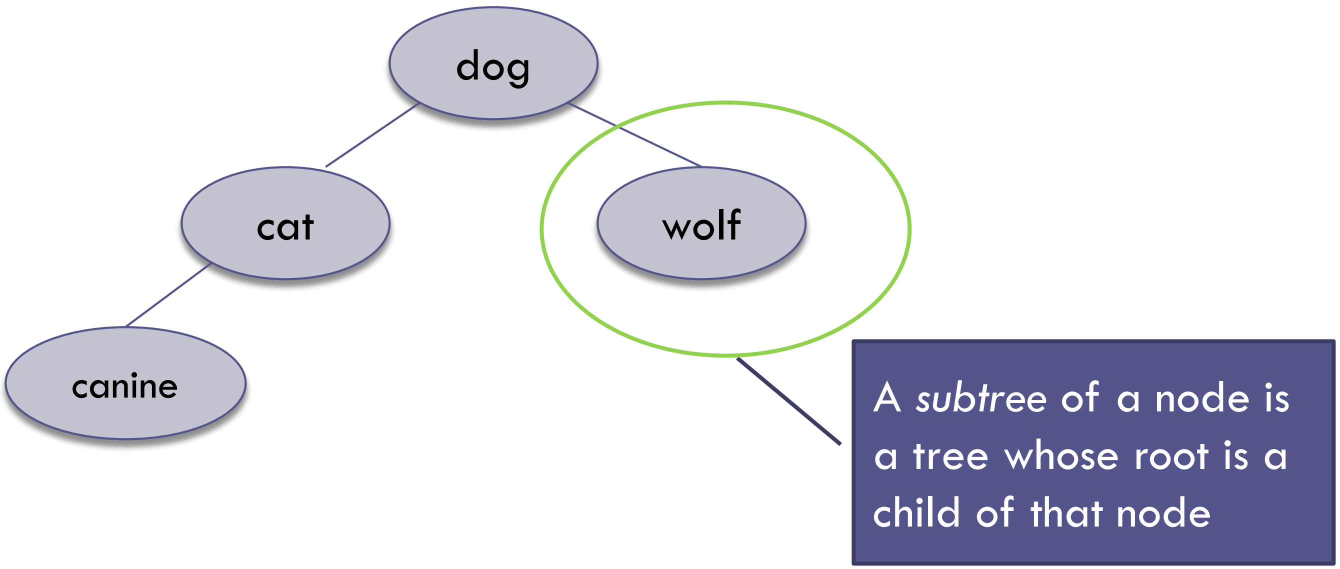
Tree Terminology: Subtree

A *subtree* of a node is a tree whose root is a child of that node



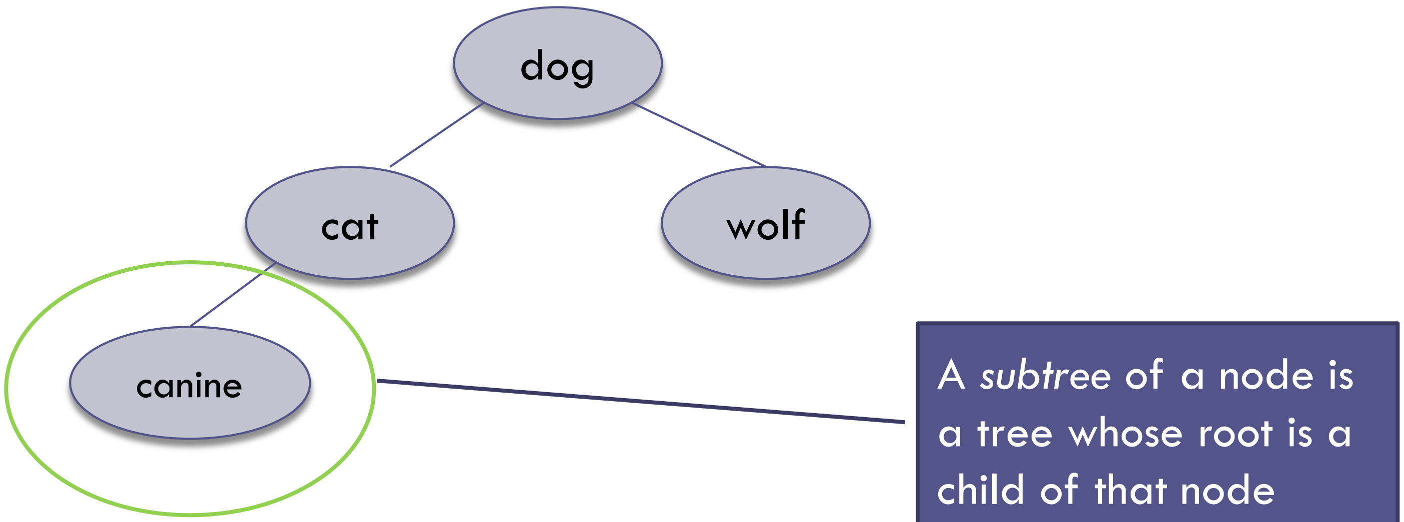
Tree Terminology: Subtree

A *subtree of a node* is a tree whose root is a child of that node



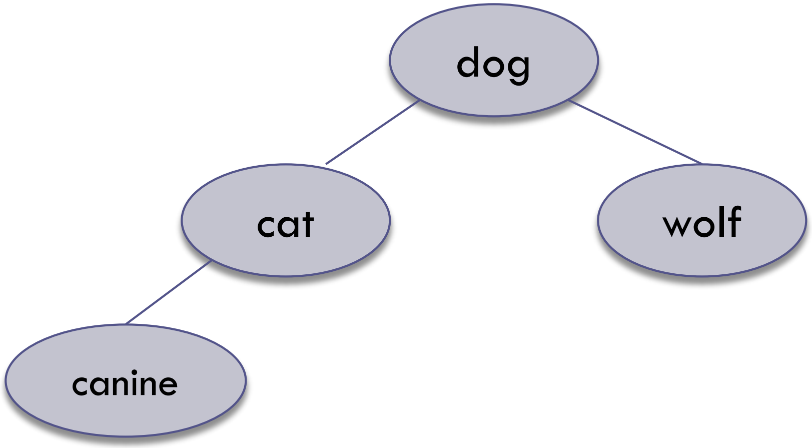
Tree Terminology: Subtree

A *subtree of a node* is a tree whose root is a child of that node



Tree Terminology: Level of Node

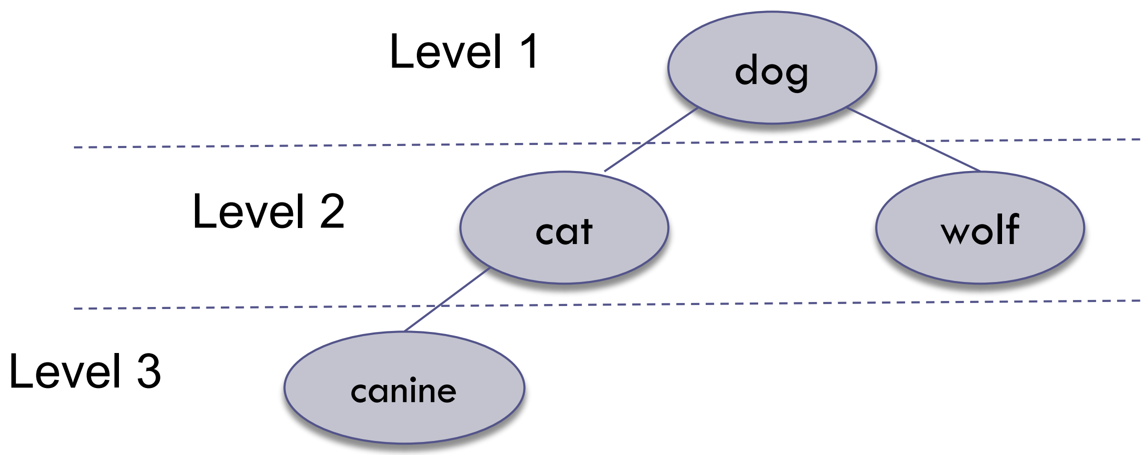
The *level of a node* is determined by its distance from the root



The *level of a node* is determined by its distance from the root

Tree Terminology: Level of Node

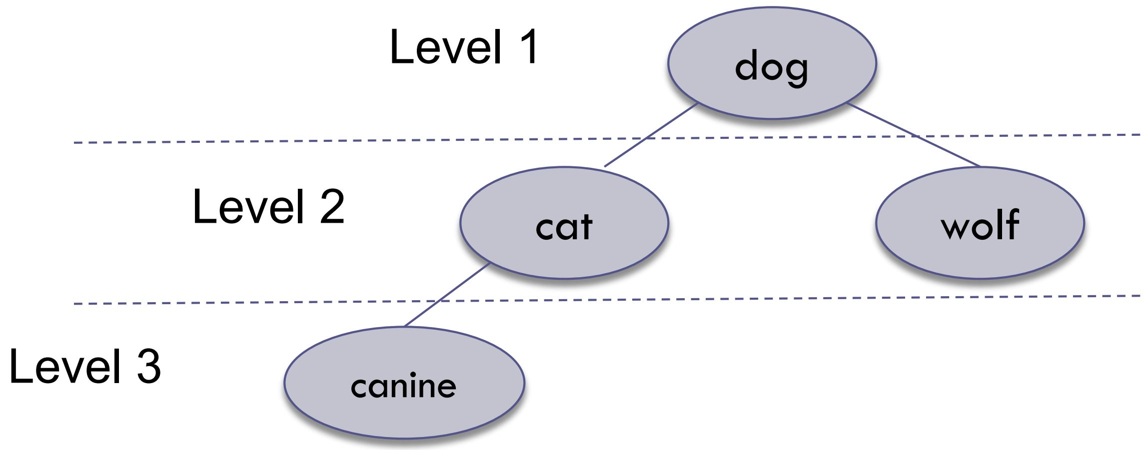
The *level of a node* is its distance from the root plus 1



The *level of a node* is its distance from the root plus 1

Tree Terminology: Level of Node

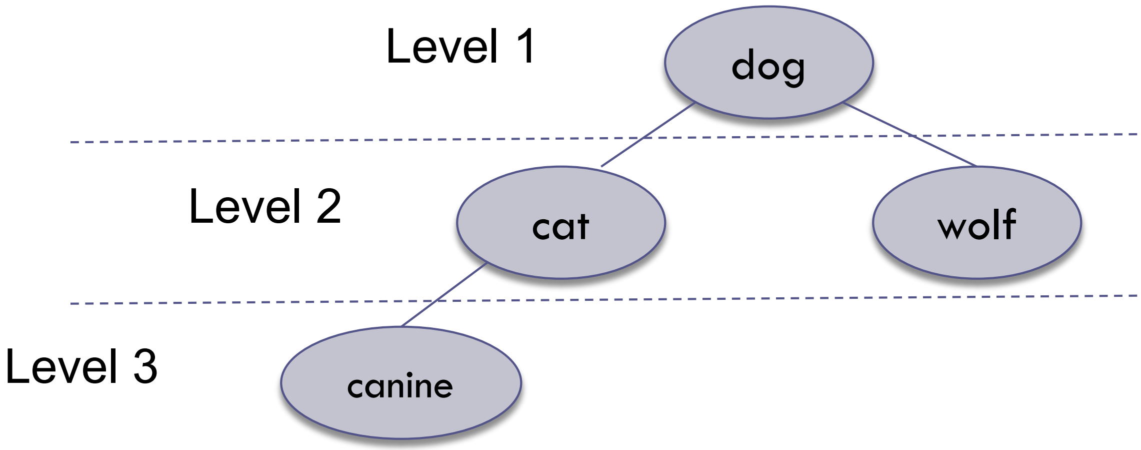
The *level of a node* is defined recursively



The *level of a node* is defined recursively

Tree Terminology: Level of Node

The *level of a node* is defined recursively



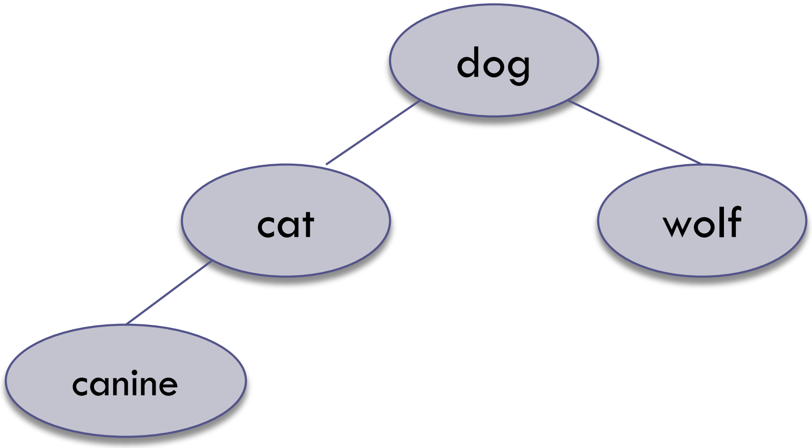
The *level of a node* is defined recursively

- If node *n* is the root of tree T, its level is 1
- If node *n* is not the root of tree T, its level is 1 + the level of its **parent**

Tree Terminology: Height of Tree

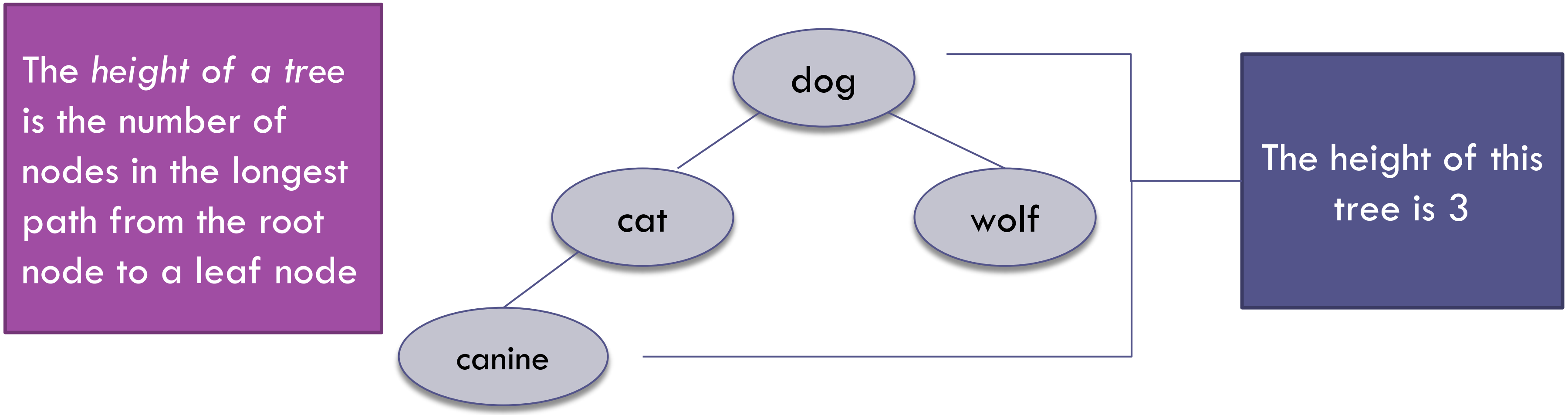
The *height of a tree* is the number of nodes in the longest path from the root node to a leaf node

The *height of a tree* is the number of nodes in the longest path from the root node to a leaf node



Tree Terminology: Height of Tree

The *height of a tree* is the number of nodes in the longest path from the root node to a leaf node

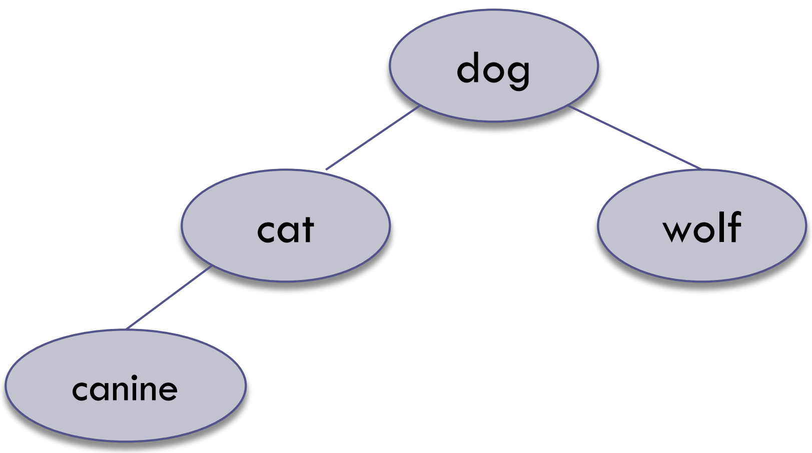


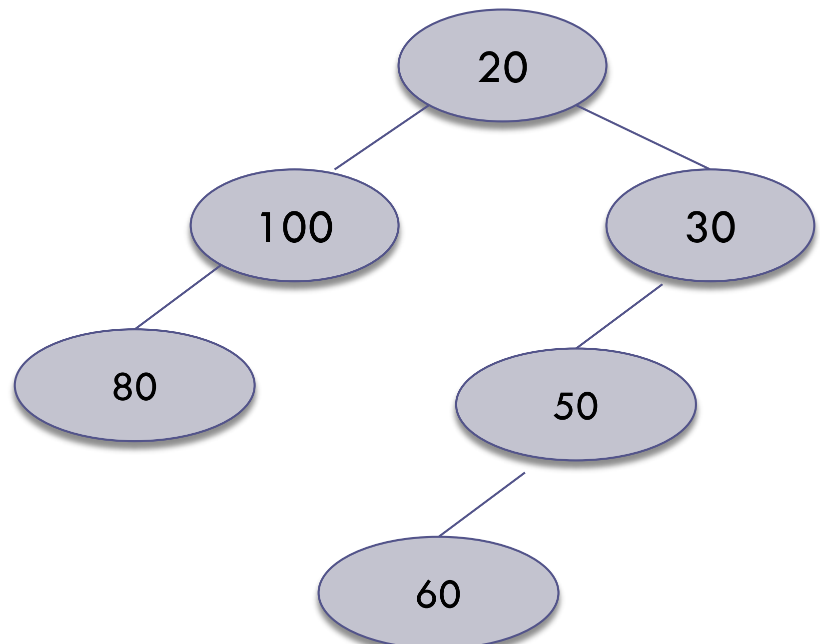
Binary Trees

- Each node has two subtrees
- A set of nodes T is a binary tree if either of the following is true
  - T is empty
  - Its root node has two subtrees, T_L and T_R, such that T_L and T_R are binary trees

 (T_L = left subtree; T_R = right subtree)

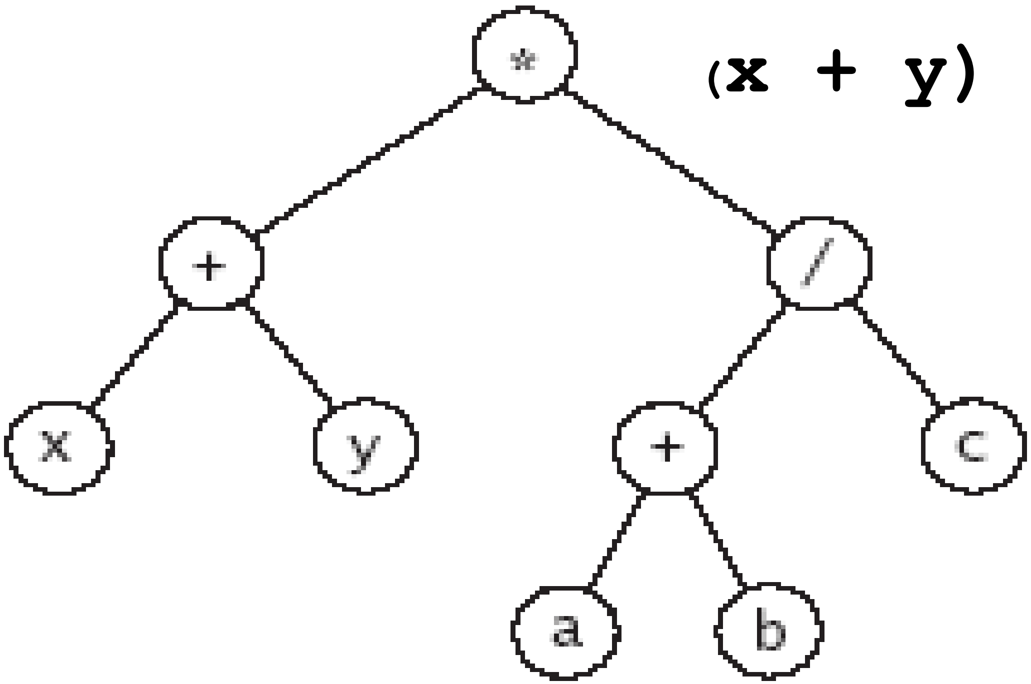
Binary Trees - Examples





Expression Tree

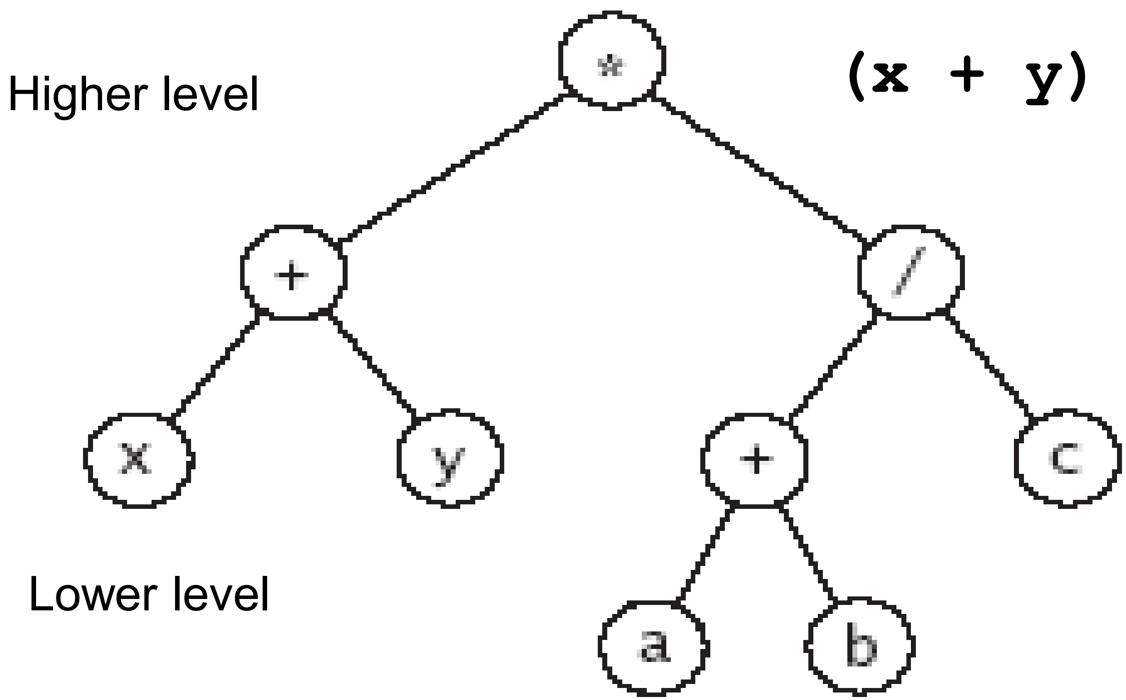
- Each node contains an operator or an operand
- **Operands** are stored in **leaf nodes**



`(x + y) * ((a + b) / c)`

Expression Tree

- **Parentheses** are **not stored** in the tree
  - **Tree structure dictates the order of operand evaluation**
- **Operators** in nodes at **higher tree levels** are **evaluated after** operators in nodes at **lower** tree **levels**




```
(x + y) * ((a + b) / c)
```