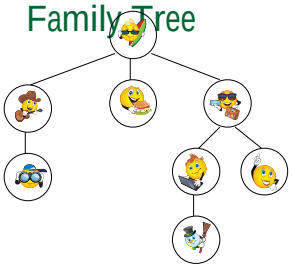
- The name of an important class of data objects
- Why is it called a tree?
  - Pieces of information are related by `branches'
- Example: Family tree

http://www.wpclipart.com

## Family Tree



- Relationships between individuals can be expressed using such a family tree
- Examples: Parent, child, sibling (i.e., brother/sister), ancestor, descendant

## Definition of Tree

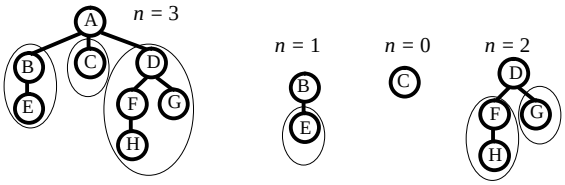
A tree is a finite set of one or more nodes such that

- There is a specially designated node called the root
- The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1,\ldots, T_n$, where each of these sets is a tree
- $T_1,\ldots, T_n$ are called the subtrees of the root

## Definition of Tree

A tree is a finite set of one or more nodes such that

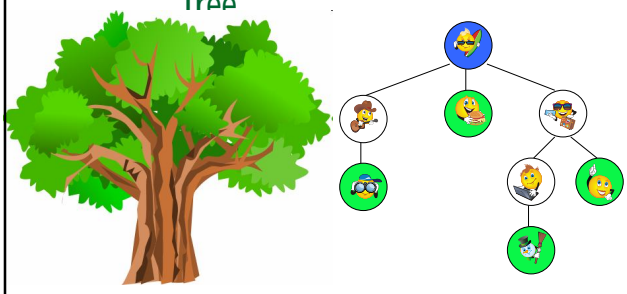
- There is a specially designated node called the root
- The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1,\ldots, T_n$, where each of these sets is a tree
- $T_1,\ldots, T_n$ are called the subtrees of the root



## Terminology

- Node: basic component of a tree
- Parent, child, sibling, ancestor, descendant: as in family tree
- Root or root node: The only node without a parent
- Every node (other than the root) has exactly one parent
- Leaf or Leaf node or Terminal node: Any node that does not have any children
- Other nodes are referred to as internal nodes

## Remember the picture of the tree



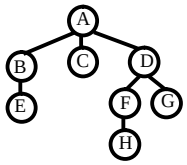
## Terminology ..

- Path: A sequence of nodes $n_1, n_2, \ldots, n_k$, such that $n_i$ is the parent of $n_{i+1}$ for $i = 1, 2, \ldots, k\text{-}1$
- Length of a path: 1 less than the number of nodes in the path
- Height of a node: Length of the longest path from the node to a leaf
- Height of a tree: Height of its root
- Similarly, depth of a node (length of unique path from root to the node)

## Paths

Path: A sequence of nodes $n_1, n_2, \ldots, n_k$, such that $n_i$ is the parent of $n_{i+1}$ for $i = 1, 2, \ldots, k\text{-}1$

Length of a path: 1 less than the number of nodes in the path




Paths of length 0: A, B, C, D, E, F, G,H

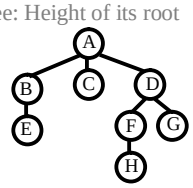
Paths of length 1: AB, BE, AC, AD, DF, DG, FH

Paths of length 2: ABE, ADF, ADG, DFH

Paths of length 3: ADFH

## Heights

- Height of a node: Length of the longest path from the node to a leaf
- Height of a tree: Height of its root




Nodes of height 0: E, C, H, G

Nodes of height 1: B, F

Nodes of height 2: D

Nodes of height 3: A

## Binary Tree

Definition: A binary tree is either empty or consists of a node called the root together with 2 binary trees called the left subtree and the right subtree.

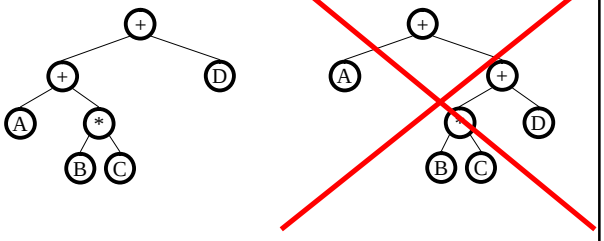
Notes

1. Unlike a general tree, binary tree can be empty
2. These 2 trees are not the same


Example: Expression tree

## Expression Tree

A + B * C + D




## Expressions

A + B * C + D

Infix Notation

( (A + (B * C) ) + D )

You may have heard of other notations for representing arithmetic expressions

Postfix Notation

Put the operator last

Also called Reverse Polish Notation

A B C * + D +

(( A ( B C *) +) D +)

## Expressions ..

A + B * C + D

Prefix notation

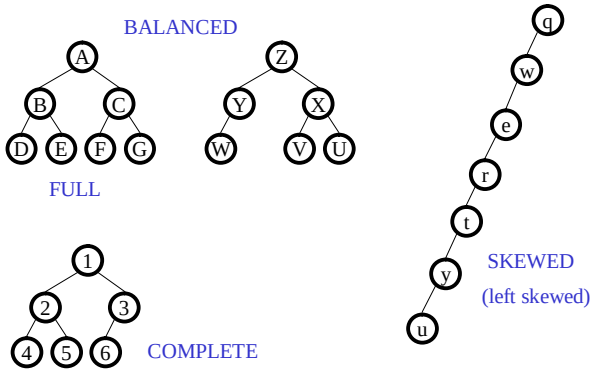
Put the operator first

Also called Polish notation

+ + A * B C D

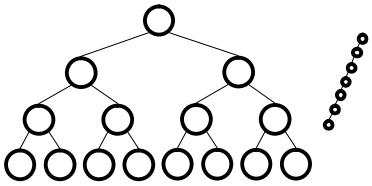
(+ (+ A ( * B C ) ) D)

## Some binary tree terminology




## More about binary trees

- Let *h* be the height of a given non-empty binary tree
- Question: At most how many nodes can it have?

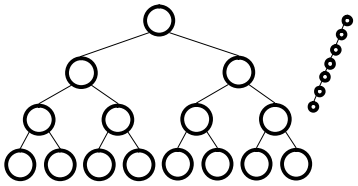


$h+1 \leq$ Number of nodes $\leq 2^{h+1}-1$

| *h* | Max nodes |
|---|---|
| 0 | 1 |
| 1 | 3 |
| 2 | 7 |
| 3 | 15 |

## More about binary trees ..

- Let *h* be the height of a given non-empty binary tree
- Question: At most how many leaves can it have?



$1 \leq$ Number of leaves $\leq 2^{h}$

| *h* | Max leaves |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

## Tree Traversal

The systematic enumeration of the nodes of a binary tree

1. In-order traversal

   Visit nodes of left subtree in in-order, then visit the root, then visit nodes of right subtree in in-order

2. Pre-order traversal

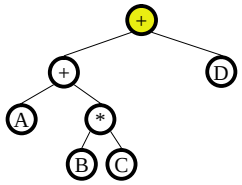
   Visit root, then visit the nodes of left subtree in pre-order, then visit nodes of right subtree in pre-order

3. Post-order traversal

   Visit nodes of left subtree in post-order, then visit nodes of right subtree in post-order, then visit the root

## Tree Traversal: Examples

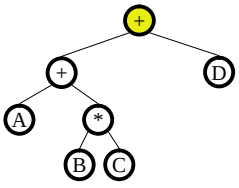
In-order traversal: Visit nodes of left subtree in in-order, then visit the root, then visit nodes of right subtree in in-order




Traversal of left subtree + Traversal of right subtree

## Tree Traversal: Examples

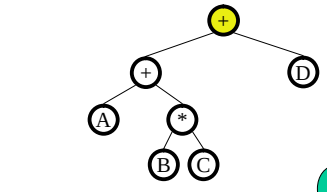
In-order traversal: Visit nodes of left subtree in in-order, then visit the root, then visit nodes of right subtree in in-order




Traversal of left subtree + D

## Tree Traversal: Examples

In-order traversal: Visit nodes of left subtree in in-order, then visit the root, then visit nodes of right subtree in in-order

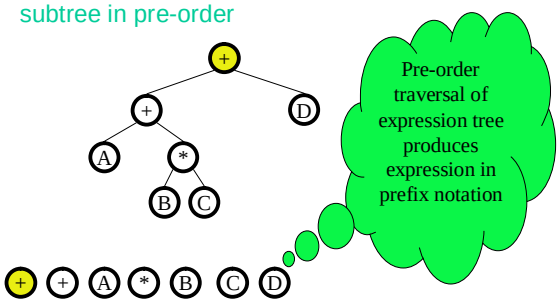



In-order traversal of expression tree produces expression in infix notation

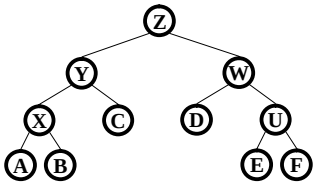
A + B * C + D

## Tree Traversal: Examples

Pre-order traversal: Visit root, then visit the nodes of left subtree in pre-order, then visit nodes of right subtree in pre-order




## Example




| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| In-order traversal | A | X | B | Y | C | Z | D | W | E | U | F |
| Pre-order traversal | Z | Y | X | A | B | C | W | D | U | E | F |
| Post-order traversal | A | B | X | C | Y | D | E | F | U | W | Z |

## Data Structures for Binary Trees

1. Pointer based
   - For every node, there is some associated information (e.g., 'A', 4, '*' in our examples)
   - In addition, for every node, there are 2 pointers
     i. One to the node of the root of its left subtree
     ii. One to the node of the root of its right subtree

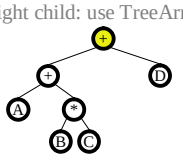
```
struct treeNode {
        char value;
        struct treeNode *left,
*right;
}
```

## Data Structures for Binary Trees ..

1. Pointer based
2. Using an array
   - char TreeArray [1000]
   - Use TreeArray [ 1 ] for the root node
   - For the node that is in TreeArray [ i ]
     - For its left child: use TreeArray [ 2i ]
     - For its right child: use TreeArray [ 2i + 1 ]

## Example of Using an array

- Use TreeArray [ 1 ] for the root node
- For the node that is in TreeArray [ i ]
  - For its left child: use TreeArray [ 2i ]
  - For its right child: use TreeArray [ 2i + 1 ]




| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | + | + | D | A | * | | | | | B | C |

## Tree Traversal

```
struct treeNode {
        char value;
        struct treeNode *left,
*right;
} *MyTree;
```

```
void inorder ( treeNode *root)
{
        if (root)  {
                inorder (root-
>left);
                visit (root);
                inorder (root-
        }
}
```