In computer science, a tree is an abstract model of a hierarchical structure.

A tree consists of nodes with a parent-child relation.

Applications:

Organization of a company
Another Example

- table of contents of a book
Another Example

- Unix or DOS/Windows file system
Terminology

- **A** is the *root* node.
- **B** is the *parent* of **D** and **E**.
- **C** is the *siblings* of **B**
- **D** and **E** are the *children* of **B**
- **D, E, F, G, I** are *external nodes*, or *leaves*
- **A, B, C, H** are *internal nodes*
- The *depth (level)* of **E** is 2
- The *height* of the tree is 3
- The *degree* of node **B** is 2

**Property:** $(\# \text{ edges}) = (\# \text{ nodes}) - 1$
Subtree: tree consisting of a node and its descendants

Ordered Tree: the children of a node are ordered
Tree ADT

- We use positions to abstract nodes

**Generic methods:**
- integer `size()`
- boolean `isEmpty()`
- Iterator `elements()`
- Iterator `positions()`

**Accessor methods:**
- position `root()`
- position `parent(p)`
- positionIterator `children(p)`

**Query methods:**
- boolean `isInternal(p)`
- boolean `isExternal(p)`
- boolean `isRoot(p)`

**Update method:**
- object `replace (p, o)`

Additional update methods may be defined by data structures implementing the Tree ADT
Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

Algorithm **preOrder**($v$)

```
visit($v$)
for each child $w$ of $v$
presOrder($w$)
```
Postorder Traversal

In a postorder traversal, a node is visited after its descendants.

Application: compute space used by files in a directory and its subdirectories.

Algorithm postOrder(v)
for each child w of v
postOrder(w)
visit(v)
Binary Trees

A binary tree is a tree with the following properties:
- Each internal node has at most two children (exactly two for proper binary trees)
- The children of a node are an ordered pair

We call the children of an internal node left child and right child

Alternative recursive definition:
a binary tree is either
- a tree consisting of a single node, or
- a tree whose root has an ordered pair of children, each of which is a binary tree.

```
A
/   \
B     C
/ \   / \ 
D   E F   G
/ \ /\  / \   / \  
H I I  H I
```
Examples of Binary Trees

Arithmetic Expression Tree

- Binary tree for an arithmetic expression
  - internal nodes: operators
  - external nodes: operands

Example: arithmetic expression tree for the expression \((2 \times (a - 1) + (3 \times b))\)
Examples of Binary Trees

Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions

Example: dining decision

Want a fast meal?
- Yes
  - How about coffee?
    - Yes: Tim Hortons
    - No: Spike’s
- No
  - On expense account?
    - Yes: Al Forno
    - No: Café Paragon
Properties of Binary Trees

- \((\# \text{ external nodes}) = (\# \text{ internal nodes}) + 1\)
- \((\# \text{ nodes at level } i) \leq 2^i\)
- \((\# \text{ external nodes}) \leq 2^{\text{height}}\)
- \((\text{height}) \geq \log_2 (\# \text{ external nodes})\)
- \((\text{height}) \geq \log_2 (\# \text{ nodes}) - 1\)
- \((\text{height}) \leq (\# \text{ internal nodes}) = ((\# \text{ nodes}) - 1)/2\)
BinaryTree ADT

The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT.

Additional methods:

- position left(p)
- position right(p)
- boolean hasLeft(p)
- boolean hasRight(p)

Update methods may be defined by data structures implementing the BinaryTree ADT.
Inorder Traversal

In an inorder traversal a node is visited after its left subtree and before its right subtree.

Algorithm *inOrder*(v)

if hasLeft (v)
    *inOrder* (left (v))
visit(v)
if hasRight (v)
    *inOrder* (right (v))
Print Arithmetic Expressions

Specialization of an inorder traversal
- print operand or operator when visiting node
- print "(" before traversing left subtree
- print ")" after traversing right subtree

Algorithm $printExpression(v)$

if $hasLeft(v)$ then {
    print("(")
    $printExpression(left(v))$
}

print($v.element$)

if $hasRight(v)$ then {
    $printExpression(right(v))$
    print (")")
}

Algorithm:

```
if hasLeft(v) then {
    print("("
    printExpression(left(v))
}
print(v.element())
if hasRight(v) then {
    printExpression(right(v))
    print (")")
}
```

```
((2 × (a − 1)) + (3 × b))
```
Evaluate Arithmetic Expressions

Specialization of a postorder traversal
- recursive method returning the value of a subtree
- when visiting an internal node, combine the values of the subtrees

Algorithm $evalExpr(v)$
if $isExternal(v)$
    return $v.element()$
else {
    $x ← evalExpr(leftChild(v))$
    $y ← evalExpr(rightChild(v))$
    $◊ ←$ operator stored at $v$
    return $x ◊ y$
}
Linked Structure for Trees

- A node is represented by an object storing:
  - Element
  - Parent node
  - Sequence of children nodes

- Node objects implement the Position ADT
 Linked Structure for Binary Trees

- A node is represented by an object storing:
  - Element
  - Parent node
  - Left child node
  - Right child node

- Node objects implement the Position ADT