# Computer Science 3A - CSC3A10

## Lecture 6: Trees

Academy of Computer Science and Software Engineering
University of Johannesburg

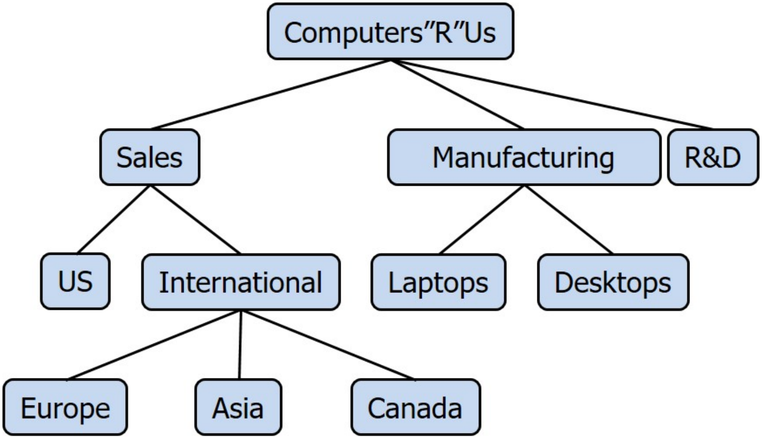
- Euler Tour Traversal of a Binary Tree
- Print Arithmetic Expressions
- Exercises

# Trees

# Tree Definition

- 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 charts
  - File systems
  - Programming environments

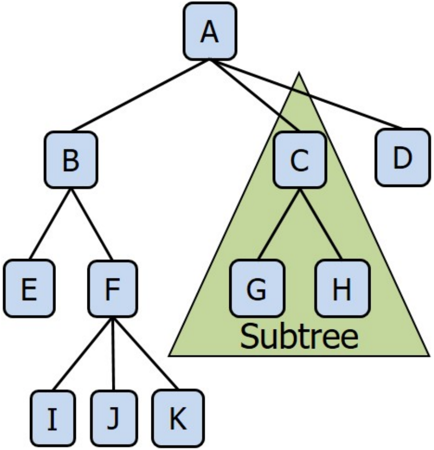
# Tree Definition II



# Tree Terminology

- **Root** - node without parent
- **Internal node** - node with at least one child
- **External node (a.k.a. leaf )** - node without children
- **Ancestors of a node** - parent, grandparent, grand-grandparent, etc.
- **Descendant of a node** - child, grandchild, grand-grandchild, etc.
- **Subtree** - tree consisting of a node and its descendants
- **Edge** - Pair of nodes (u,v) where u is the parent of v, or vice versa
- **Path** - Sequence of nodes such that any two consecutive nodes in the sequence form an edge

# Tree Terminology II



# Tree Terminology III

**Depth of node v** is the number of ancestors of *v* excluding *v*. Defined recursively as:

```
If v is root
  depth of v is 0
Else depth of v is 1+depth(parent(v))
```

Depth of node *v*

# Tree Terminology IV

**Height of node v** is defined recursively as:

```
If v is an external node
  height of v is 0
Else height of v is 1+max(height(child(v)))
```

Height of node v

# Tree ADT

We use positions to abstract nodes, where Positions are defined relative to neighbouring position.
Generic methods:

- *integer size()*
- *boolean isEmpty()*
- *Iterator elements()*
- *Iterator positions()*

Accessor methods:

- *position root()*
- *position parent(p)*
- *positionIterator children(p)*

# Tree ADT II

Query methods:

- *boolean isInternal(p)*
- *boolean isExternal(p)*
- *boolean isRoot(p)*

Update methods:

- *Object replace(p,o)*

Update methods may be defined by data structures implementing the Tree ADT

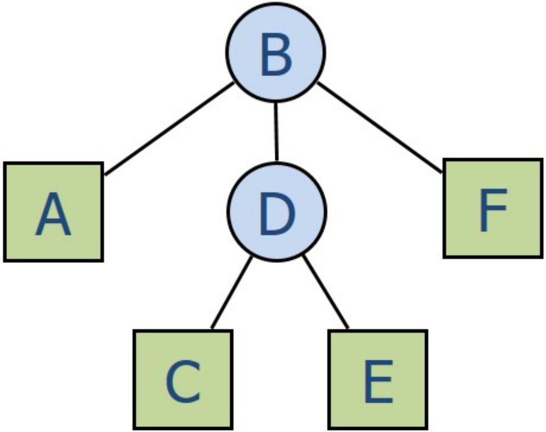
# Linked Structure for Trees

A node is represented by an object storing:

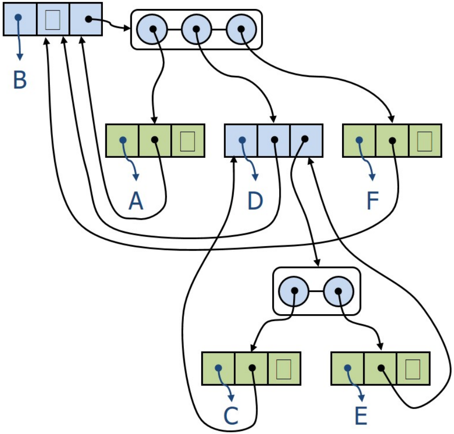
- Element
- Parent node
- Sequence of children nodes

Node objects implement the Position ADT

# Linked Structure for Trees II



# Linked Structure for Trees III



# Preorder Traversal

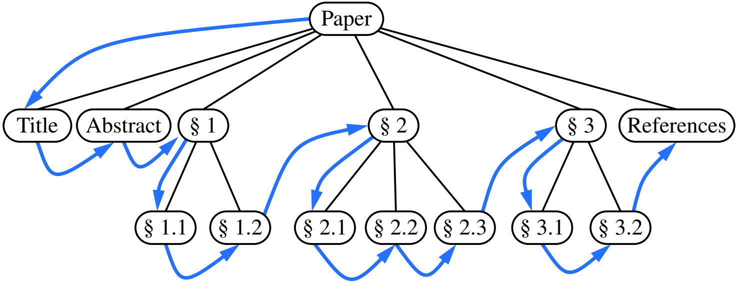
- A traversal is a systematic way of accessing or visiting all the nodes of a tree
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

# Preorder Traversal II

```
Algorithm preOrder(v)
  visit(v)
  for each child w of v
    preorder (w)
```

preOrder Traversal

# Preorder Traversal III



# Postorder Traversal

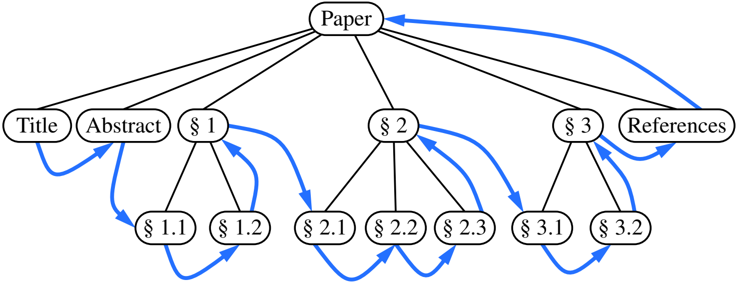
- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

# Postorder Traversal II

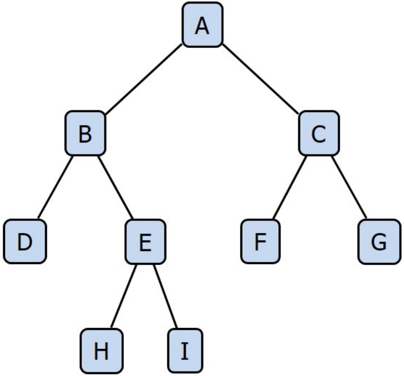
```
Algorithm postOrder(v)
for each child w of v
  postOrder (w)
visit(v)
```

preOrder Traversal

# Postorder Traversal III



# Binary Trees



# Binary Tree Defintion

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

# Binary Tree Defintion II

Alternative recursive definition:

- a tree consisting of a single node (root), or
- a tree whose root has an ordered pair of children, each of which is a binary tree

Applications include:

- arithmetic expressions
- decision processes
- searching

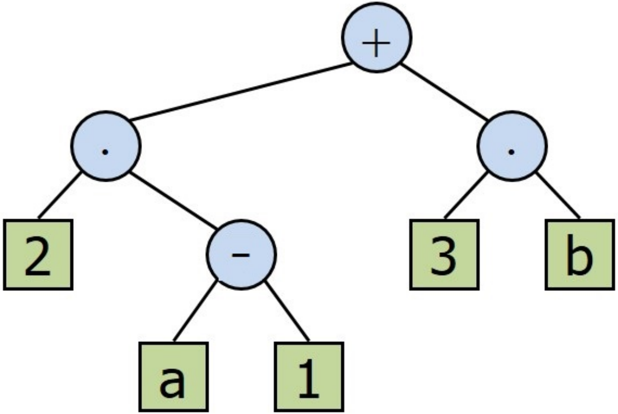
# Arithmetic Expression Tree

Binary tree associated with an arithmetic expression

- internal nodes: operators
- external nodes: operands

# Arithmetic Expression Tree II

Example: Arithmetic expression tree for the expression $(2 * (a - 1)) + (3 * b)$



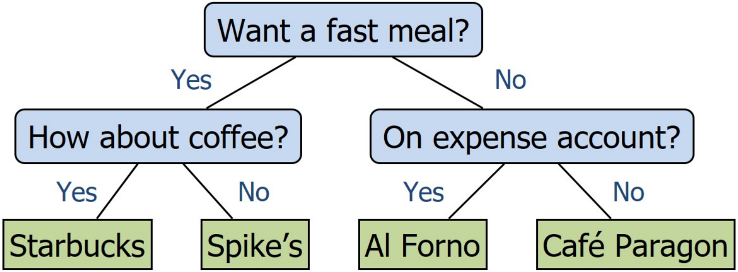
# Decision Tree

Binary tree associated with a decision process

- internal nodes: questions with yes/no answer
- external nodes: decisions

# Decision Tree II

Example: dining decision



# Binary Tree ADT

The BinaryTree ADT extends the Tree ADT i.e. it inherits all the methods of the Tree ADT. The additional methods include:

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

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

# Binary Tree Properties

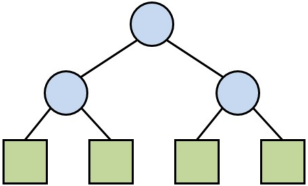
Notation:

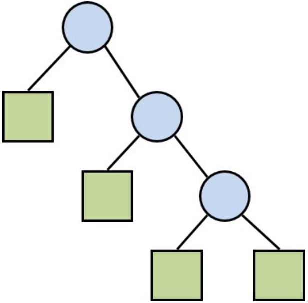
- $n$ - number of nodes
- $e$ - number of external nodes
- $i$ - number of internal nodes
- $h$ - height

Properties:

1. $e = i + 1$
2. $n = 2e - 1$
3. $h \leq i$
4. $h \leq n - 1$
5. $e \leq 2^h$
6. $h \geq log_2 e$
7. $h \geq log_2(n + 1) - 1$

# Binary Tree Properties II





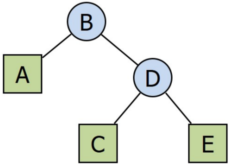
# Linked Structure for Binary Trees

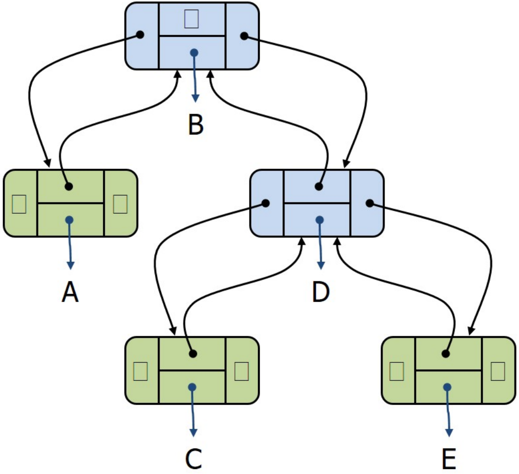
A node is represented by an object storing:

- Element
- Parent node
- Left child node
- Right child node

Node objects implement the Position ADT

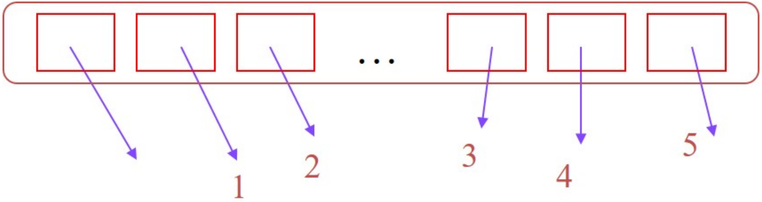
# Linked Structure for Binary Trees II



# Array-Based Implementation of Binary Trees

Nodes are stored in an array

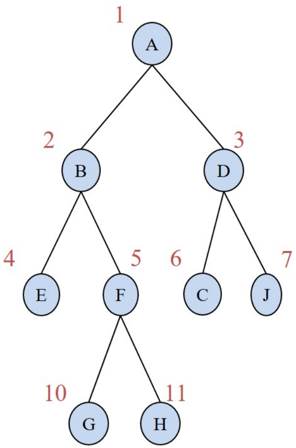


# Array-Based Implementation of Binary Trees II

Let rank(node) be defined as follwos:

- rank(root) $= 1$
- if node is the left child of parent(node)
  - rank(node) $=$ 2*rank(parent(node))
- if node is the right child of parent(node)
  - rank(node) $=$ 2*rank(parent(node))$+1$

# Array-Based Implementation of Binary Trees III



# Preorder Traversal of a Binary Tree

```
Algorithm binaryPreOrder(T,v)
visit(v)
if hasLeft(v)
  binaryPreOrder (T,left(v))
if hasRight(v)
      binaryPreOrder (T,right(v))
```

Preorder traversal of a binary tree

# Postorder Traversal of a Binary Tree

```
Algorithm binaryPostOrder(T,v)
if hasLeft(v)
  binaryPostOrder (T,left(v))
if hasRight(v)
  binaryPostOrder (T,right(v))
visit(v)
```

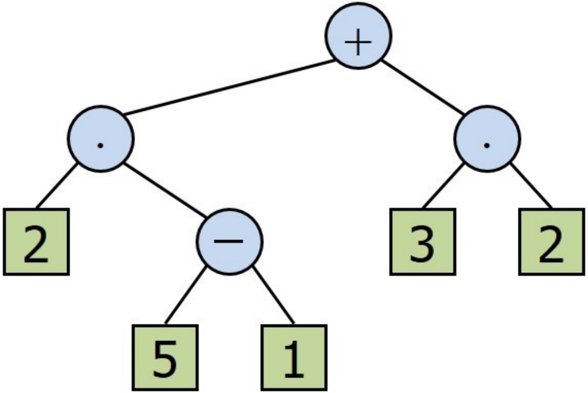
Postorder traversal of a binary tree

# 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

# Evaluate Arithmetic Expressions II



# Evaluate Arithmetic Expressions III

```
Algorithm evalExpr(v)
if isExternal (v)
  return v.element ()
else
  x = evalExpr(leftChild (v))
  y = evalExpr(rightChild (v))
  op = operator stored at v
return x op y
```

Evaluating expressions using a binary tree

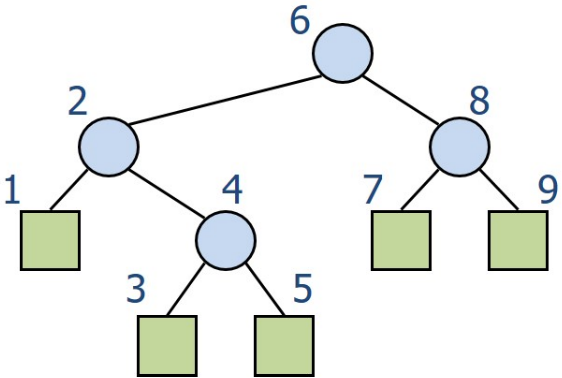
# Inorder Traversal of Binary Tree

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

Application: draw a binary tree:

- x(v) = inorder rank of v
- y(v) = depth of v

# Inorder Traversal of Binary Tree II



# Inorder Traversal of Binary Tree III

```
Algorithm inOrder(v)
if hasLeft (v)
  inOrder (left (v))
visit(v)
if hasRight (v)
  inOrder (right (v))
```

Inorder Traversal

# Euler Tour Traversal of a Binary Tree

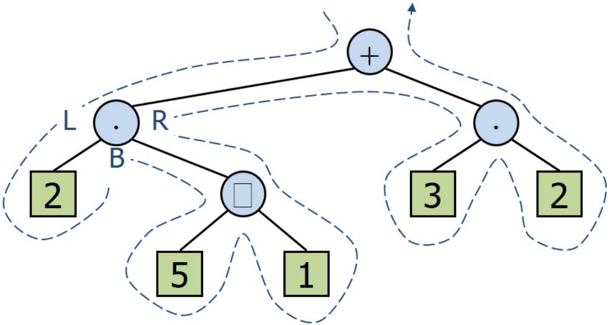
Generic traversal of a binary tree. Includes the special cases for the following traversals

- Preorder
- postorder
- Inorder

Walk around the tree and visit each node three times:

- on the left (preorder)
- from below (inorder)
- on the right (postorder)

# Euler Tour Traversal of a Binary Tree II



# Euler Tour Traversal of a Binary Tree III

```
Algorithm EulerTour(T,v)
  visitLeft(T,v)
if T.hasLeft(v)
  EulerTour(T,left(v))
visitBelow(T,v)
if T.hasRight(v)
  EulerTour(T,right(v))
visitRight(T,v)
```

Euler Tour Traversal

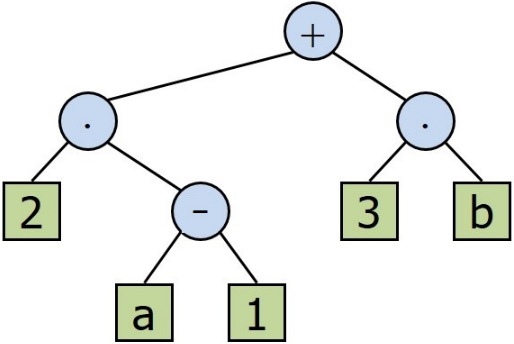
# Print Arithmetic Expressions

Specialization of Euler tour traversal

- “On the left” - if the node is internal then print “(“
- “From below” - print the value or operator stored in node
- “On the right” - if the node is internal then print “)”

# Euler Tour Traversal of a Binary Tree II

$$((2 \cdot (a - 1)) + (3 \cdot b)) \tag{1}$$



# Euler Tour Traversal of a Binary Tree III

```
Algorithm printExpression(T,v)
if T.isInternal(v)
  print('(')
if T.hasLeft (v)
  printExpression(T,T.left(v))
print the operator or value stored at v
if T.hasRight (v)
  printExpression(T,T.right(v))
if T.isInternal(v)
  print (')')
```

Euler Tour Print Expression

## Template Method Pattern

- Generic algorithm that can be specialized by redefining certain steps
- Implemented by means of an abstract Java class
- Visit methods that can be redefined by subclasses

Template method of Euler Tour:

- Recursively called on the left and right children
- A Result object with fields leftResult, rightResult and finalResult keeps track of the output of the recursive calls to eulerTour

# Template Method Pattern II

```
Algorithm templateEulerTour(T,v)
r = new object of type TourResult
visitLeft(T,v,r)
if T.hasLeft(v)
  r.left = templateEulerTour(T,T.left(v))
visitBelow(T,v,r)
if T.hasRight(v)
  r.right = templateEulerTour(T,T.right(v))
visitRight(T,v,r)
return r.out
```

Template method pattern

# Exercises

Reinforcement exercises:

- R-8.1
- R-8.4
- R-8.10
- R-8.11 - R-8.16

Creativity exercises:

- C8.27 - C8.29
- C8.30