How can we implement trees with nodes that have > 2 children?
09-1: Trees with > 2 children

- Array of Children
Trees with > 2 children

- Linked List of Children
We can integrate the linked lists with the nodes themselves:
09-4: Working with General Tree

class Node {
    private Node leftchild_;  
    private Node rightsib_;  
    private Object element_;  

    Node leftchild() {   
        return leftchild_;  
    }  

    void setLeftchild(Node leftchild) {  
        leftchild_ = leftchild;  
    }  

    Node rightsib() {  
        return rightsib_;  
    }  

    void setRightsib(Node leftchild) {  
        rightsib_ = rightsib;  
    }  

    Node element() {  
        return element_;  
    }  

    void setElement(Object element) {  
        element_ = element;  
    }  
}
Returns the number of nodes in a tree

Number of Nodes = 8

Number of Nodes = 6
int numnodes(Node tree) {
    int descendants = 0;
    Node tmp;

    if (tree == null)
        return 0;
    for (tmp = tree.leftchild(); tmp != null;
         tmp = tmp.rightsib())
        descendants = descendants + numnodes(tmp);

    return descendants + 1;
}
int numnodes(Node tree) {
    if (tree == null)
        return 0;
    return 1 + numnodes(tree.leftchild())
        + numnodes(tree.rightsib());
}
09-8: Tree Operations – Height

- Returns the height of the tree
  - (Length of the path to the deepest leaf) + 1

Height = 5

Height = 6
```java
int height(Node tree) {
    if (tree == null)
        return 0;
    int childHeight = 0;
    for (Node tmp = tree.leftchild(); tmp != null;
        tmp = tmp.rightsib())
    {
        childHeight = MAX(childHeight, height(tmp));
    }
    return childHeight + 1;
}
```
09-10: General Trees – Height

```java
int height(Node tree) {
    if (tree == null)
        return 0;
    return MAX((1 + height(tree.leftchild())),
                height(tree.rightsib()));
}
```
Write `numLeaves` and print
```java
int numLeaves(Node tree) {
    if (tree == null)
        return 0;
    if (tree.leftchild() == null)
        return 1 + numLeaves(tree.rightsib());
    return numLeaves(tree.leftchild()) +
          numLeaves(tree.rightsib());
}
```
void print(Node tree, int offset) {
    if (tree != null) {
        for (int i = 0; i < offset; i++)
            System.out.print("\t");
        System.out.println(tree.element());
        print(tree.leftchild(), offset+1);
        print(tree.rightsib(), offset);
    }
}
09-14: Serializing Binary Trees

- Print a tree to a file, saving structure information
- First Try: Print out nodes, in order that they would appear in a PREORDER traversal.
  - Why doesn’t this work?
09-15: Serializing Binary Trees

- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don’t know when we’ve hit a null pointer
- Store null pointers, too!
09-16: Serializing Binary Trees

- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don’t know when we’ve hit a null pointer
- Store null pointers, too!
• Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don’t know when we’ve hit a null pointer

• Store null pointers, too!
• Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don’t know when we’ve hit a null pointer

• Store null pointers, too!

\[ABDE/G/CF/H/\]
Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don’t know when we’ve hit a null pointer.

Store null pointers, too!

```
ABDE/G/CF/H/
```
If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf: $A_0B_1C_0D_0E_1F_1G_1$
If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node: 0 for an internal node, 1 for a leaf:
If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:

\[ A_0 B_0 C_1 D_1 E_0 F_1 G_1 \]
If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:

\[ A_0 B_0 C_1 D_0 E_1 F_1 G_1 \]
Serializing Binary Trees

If we are serializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:

$$A_0B_0C_1D_0E_1F_1G_1$$

```
  A
 / \  \
B   G
/ \  / \
C   D
  \ / \
   E F
```
• Store an “end of children” marker

```plaintext
```
09-26: Serializing General Trees

- Store an “end of children” marker

```
          A
         / \  /
        B   C
       / \  / \  /
      F   G H  I  J K
     / \  / \  /   
    E   D
```
09-27: Serializing General Trees

- Store an “end of children” marker

A
  /   \
B      C
  |     |
F      G
  |     |   |
K      H
        |
          D
            |
              E

ABFK))))CG)H))))DI)J))))E)))
09-28: Serializing General Trees

- Store an “end of children” marker

\[ABDK)||(CE)F)(GI)J)||(H)\]
09-29: Serializing General Trees

- Store an “end of children” marker

\[ ABDK)()(CE)(F)(GI)(J)(H)\]

![Diagram of a tree with nodes and labels, illustrating the structure of a serializing process.](image-url)