How can we implement trees with nodes that have > 2 children?
Trees with > 2 children

- Array of Children
09-2: **Trees with > 2 children**

- Linked List of Children
We can integrate the linked lists with the nodes themselves:
class Node {
    private Node leftchild_; 
    private Node rightsib_; 
    private Object element_;  
    
    Node leftchild() {
        return leftchild_;  
    } 
    
    Node rightsib() {
        return rightsib_;  
    } 
    
    Node element() {
        return element_; 
    }  
    
    void setLeftchild(Node leftchild) {
        leftchild_ = leftchild;  
    }  
    
    void setRightsib(Node rightsib) {
        rightsib_ = rightsib; 
    }  
    
    void setElement(Object element) {
        element_ = element; 
    }  
}
09-5: General Trees – NumNodes

- 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());
}
Tree Operations – Height

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

Height = 5
Height = 6
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;
}
int height(Node tree) {
    if (tree == null)
        return 0;
    return MAX((1 + height(tree.leftchild())),
                height(tree.rightsib()));
}
Write `numLeaves` and print
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);
    }
}
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?

```
A
  B  C
  D  E  F
    G
ABDEGCF
```
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!

```
ABD///EG///C/F///
```
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!
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!

\[ABDE//G///CF/H///\]
09-19: 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!

```
ABDE\G\CF\H\\
```

```
A
  /\  \\
 B   C
  /\  /\
 D   F
  /\  /\
 E   G 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
/   \
B     C
|     |
D-----E
      /\  \
     /   \
    F-----G
```

$A_0 B_1 C_0 D_0 E_1 F_1 G_1$
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:
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:

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

- Store an “end of children” marker

$$ABE(FK)))C(DG)H(I)J)))$$
09-26: Serializing General Trees

- Store an “end of children” marker
09-27: Serializing General Trees

- Store an “end of children” marker

\[ ABFK\text{
\{\text{end of children marker}\}CG\text{
\{\text{end of children marker}\}D}\text{
\{\text{end of children marker}\}I\text{
\{\text{end of children marker}\}J\text{
\{\text{end of children marker}\}E\}}} \]
• Store an “end of children” marker

\[ A B D K)))(C E)(F)(G I)(J)))(H))) \]
Serializing General Trees

- Store an “end of children” marker

\[ A\overline{B\overline{D\overline{K}}}\overline{C\overline{E\overline{F\overline{G\overline{I\overline{J}}}H}}}} \]