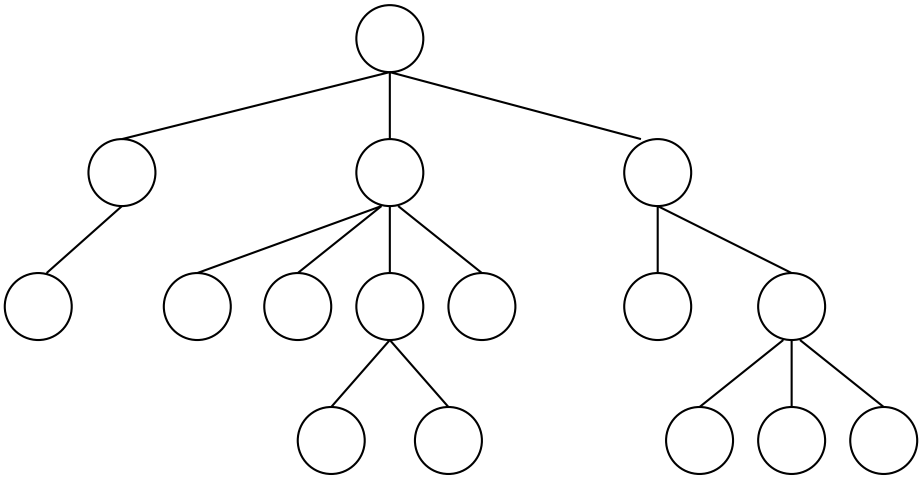
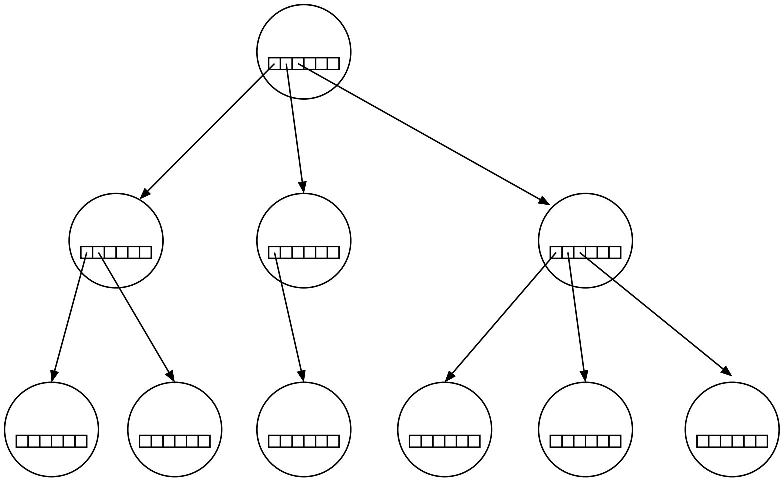
09-0: **Trees with > 2 children**

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



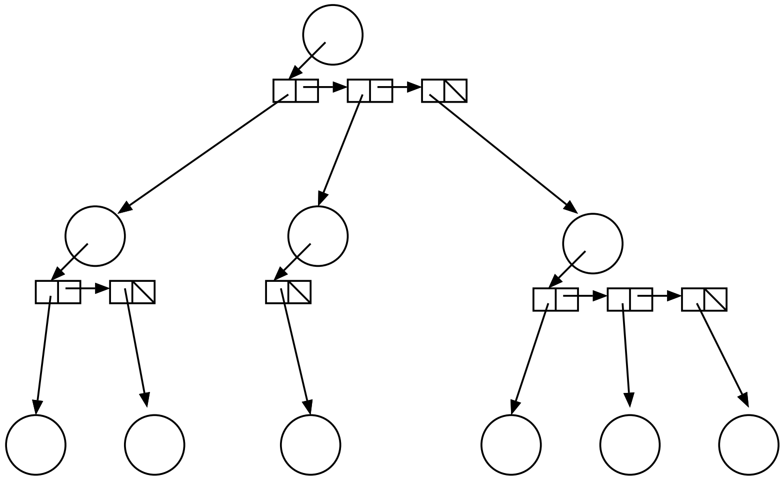
09-1: **Trees with > 2 children**

- Array of Children



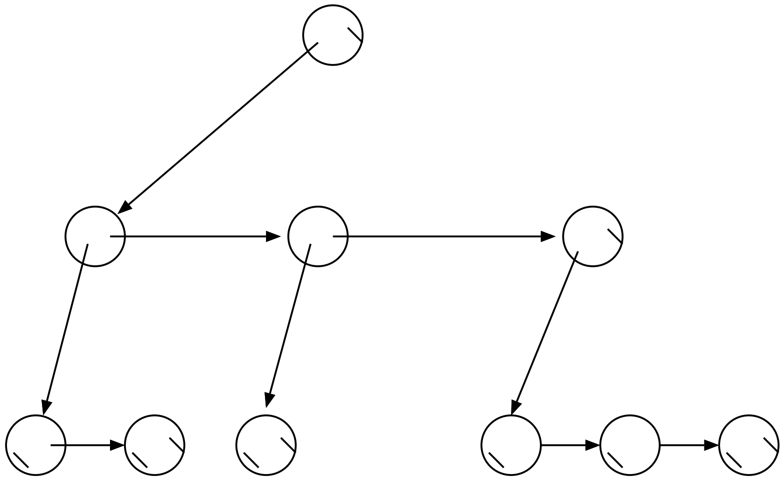
09-2: **Trees with > 2 children**

- Linked List of Children



09-3: **Left Child / Right Sibling**

- We can integrate the linked lists with the nodes themselves:

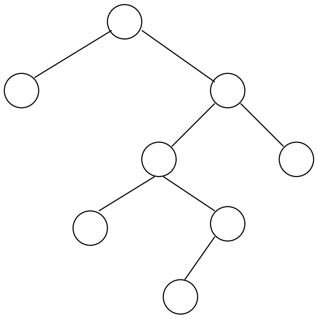


09-4: **Working with General Tree**

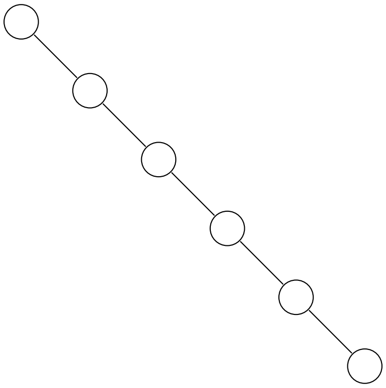
```
class Node  {
  private Node leftchild_;
  private Node rightsib_;
  private Object element_;

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

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

  Node element() {           void setElement(Object element) {
    return element_;           element_ = element;
  }                          }
}
```

09-5: **General Trees – NumNodes**

- Returns the number of nodes in a tree



Number of Nodes = 8



Number of Nodes = 6

09-6: **General Trees – NumNodes**

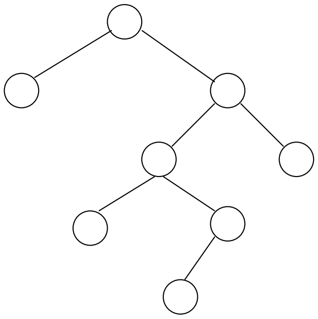
```
int numnodes(Node tree) {
  int descendants = 0;
  Node tmp;

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

  return descendants + 1;
}
```

09-7: **General Trees – NumNodes II**

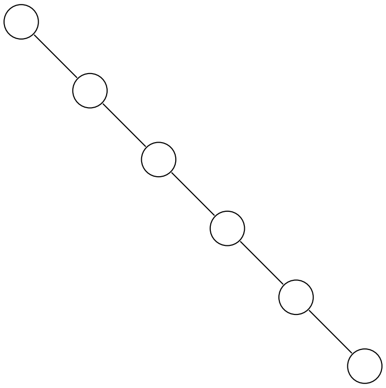
```
int numnodes(Node tree) {
  if (tree == null)
    return 0;
  return 1 + numnodes(tree.leftchild())
           + numnodes(tree.rightsib());
}
```

09-8: **Tree Operations – Height**

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



Height = 5



Height = 6

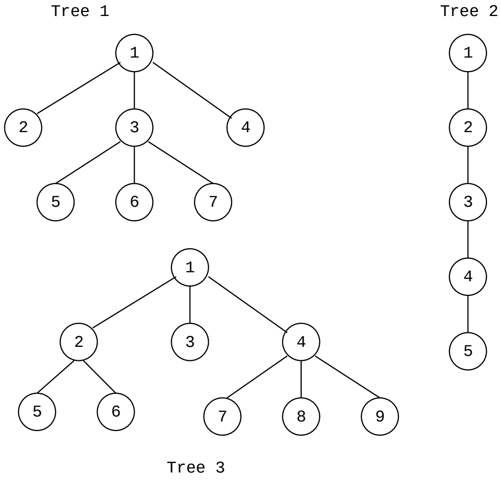
09-9: **General Trees – Height**

```
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**

```
int height(Node tree) {
  if (tree == null)
    return 0;
  return MAX((1 + height(tree.leftchild())),
             height(tree.rightsib()));
}
```

09-11: **General Trees**



Write `numLeaves` and `print` 09-12: **General Trees – numLeaves**

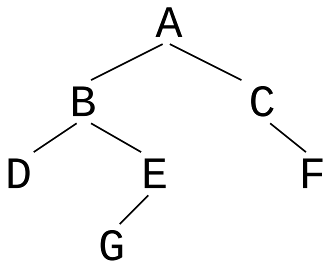
```
int numLeaves(Node tree) {

  if (tree == null)
    return 0;
  if (tree.leftchild() == null)
    return 1 + numLeaves(tree.rightsib());
  return numLeaves(tree.leftchild()) +
         numLeaves(tree.rightsib());
}
```

09-13: **General Trees – numLeaves**

```
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**

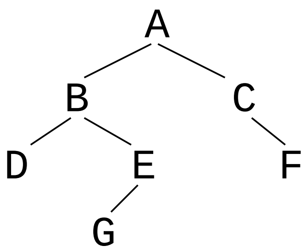
- Print a tree to a file, saving structure information
- First Try: Print out nodes, in order that they would appear in a PREORDER traversal.
  - Why doesn't this work?



$ABDEGCF$

09-15: **Serializing Binary Trees**

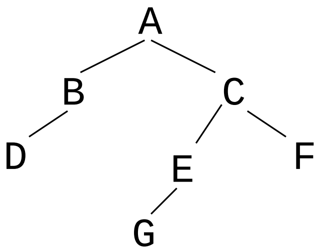
- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don't know when we've hit a null pointer
- Store null pointers, too!



$ABD//EG///C/F//$

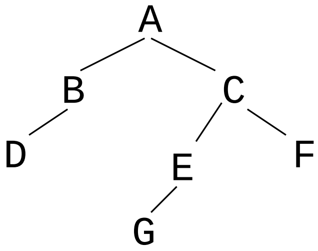
09-16: **Serializing Binary Trees**

- 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!



09-17: **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!



$ABD///CEG///F//$

09-18: **Serializing Binary Trees**

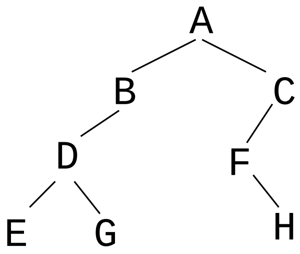
- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don't know when we've hit a null pointer
- Store null pointers, too!

$ABDE//G///CF/H///$

09-19: **Serializing Binary Trees**

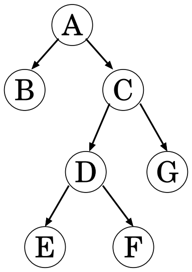
- Printing out nodes, in order that they would appear in a PREORDER traversal does not work, because we don't know when we've hit a null pointer
- Store null pointers, too!

$$ABDE//G///CF/H///$$



09-20: **Serializing Binary Trees**

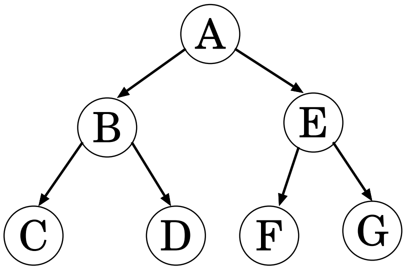
- If we are searializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:



$$A_0B_1C_0D_0E_1F_1G_1$$

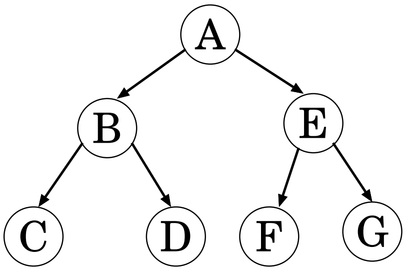
09-21: **Serializing Binary Trees**

- If we are searializing 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:



09-22: **Serializing Binary Trees**

- If we are searializing 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:



$$A_0B_0C_1D_1E_0F_1G_1$$

09-23: **Serializing Binary Trees**

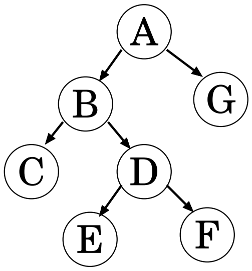
- If we are searializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:

$$A_0B_0C_1D_0E_1F_1G_1$$

09-24: **Serializing Binary Trees**

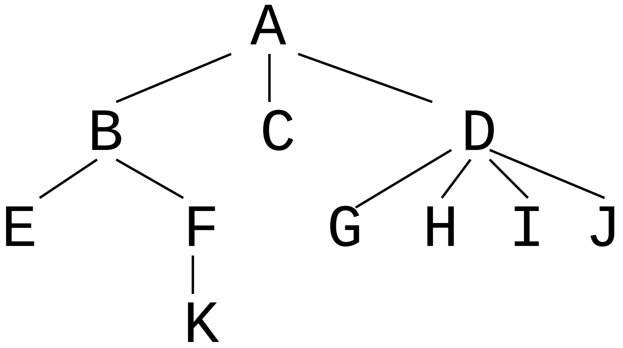
- If we are searializing a full binary tree (each node contains exactly 0 or 2 children), we can store a single extra bit for each node 0 for an internal node, 1 for a leaf:

$$A_0B_0C_1D_0E_1F_1G_1$$



09-25: **Serializing General Trees**

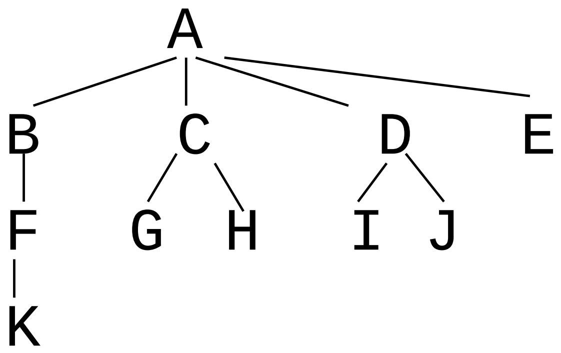
- Store an “end of children” marker



$$ABE)FK)))C)DG)H)I)J)))$$

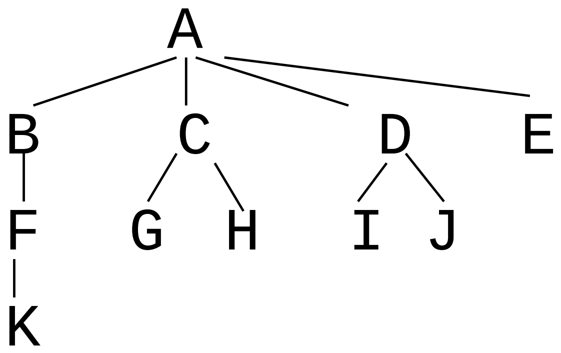
09-26: **Serializing General Trees**

- Store an "end of children" marker



09-27: **Serializing General Trees**

- Store an "end of children" marker



$$ABFK)))CG)H))DI)J))E))$$

09-28: **Serializing General Trees**

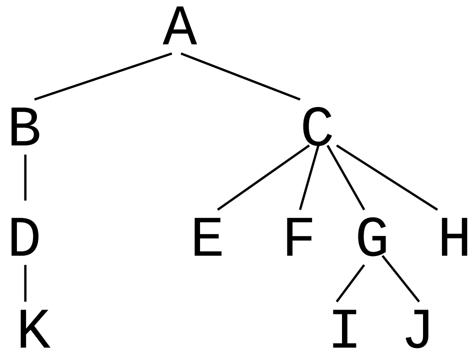
- Store an "end of children" marker

$$ABDK)))CE)F)GI)J))H)))$$

09-29: **Serializing General Trees**

- Store an "end of children" marker

$$ABDK)))CE)F)GI)J))H)))$$


A
B
C
D
E
F
G
H
K
I
J