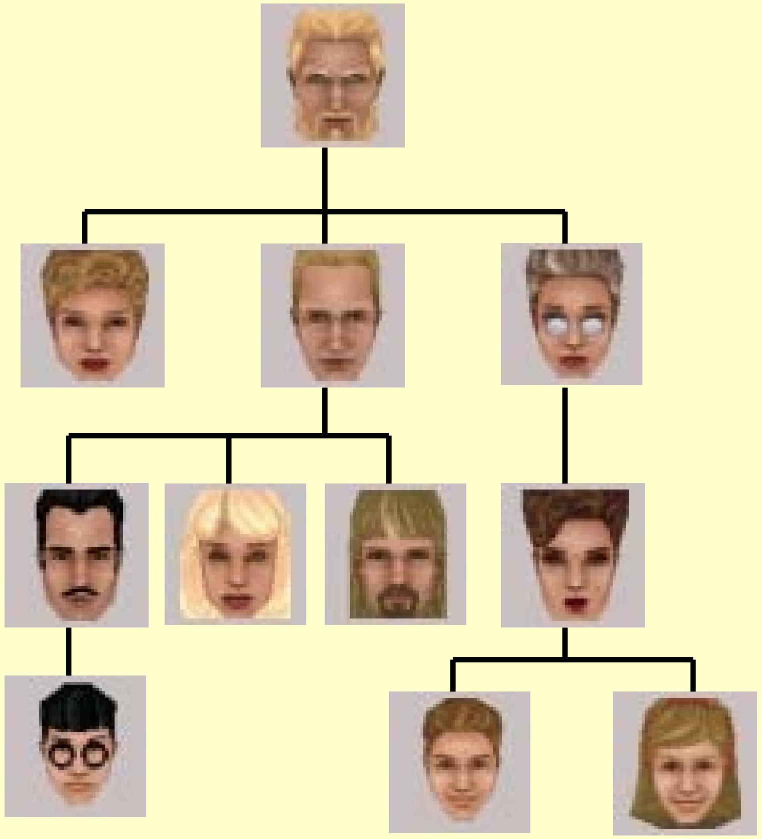
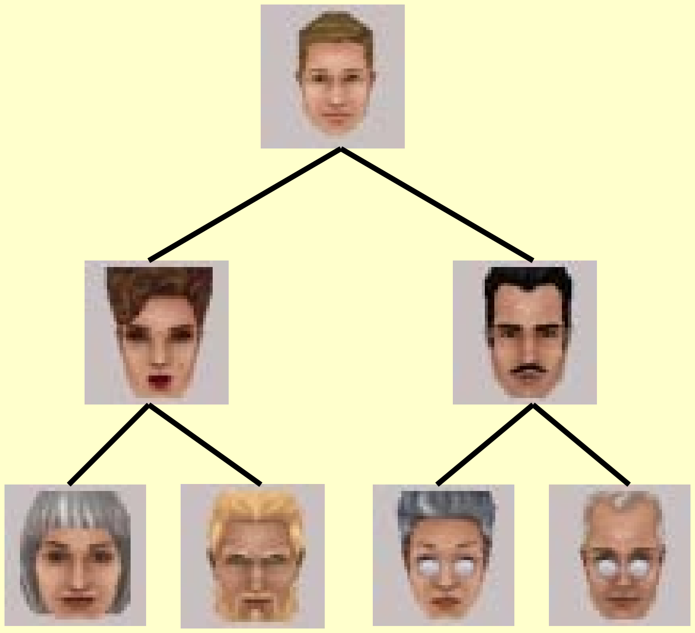
CHAPTER 4

TREES

§ 1 Preliminaries

1. Terminology


Lineal Tree


Pedigree Tree
(binary tree)

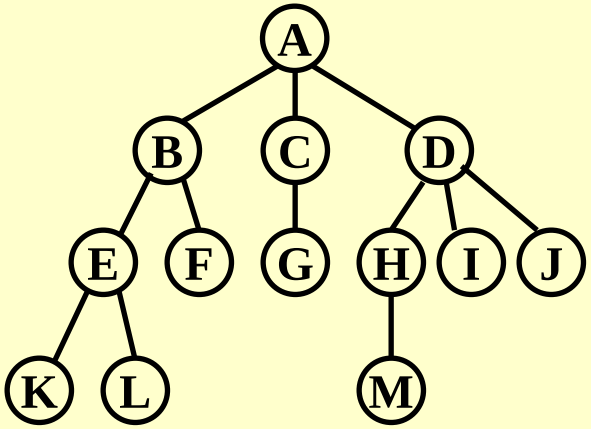
【Definition】A **tree** is a collection of nodes. The collection can be empty; otherwise, a tree consists of

(1) a distinguished node r, called the **root**;

(2) and zero or more nonempty **(sub)trees** $T_1, \cdots, T_k$, each of whose roots are connected by a directed **edge** from r.

Note:

- Subtrees must not connect together. Therefore every node in the tree is the root of some subtree.
- There are $N - 1$ edges in a tree with N nodes.
- Normally the root is drawn at the top.



- ✐ **degree of a node ::= number of subtrees of the node. For example, degree(A) = 3, degree(F) = 0.**

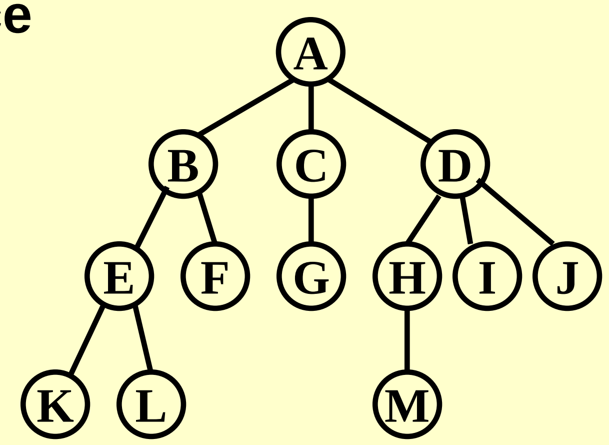
- ✐ **degree of a tree ::=** $\max_{node \in tree} \{ \text{degree}(node) \}$ **For example, degree of this tree = 3.**

- ✐ **parent ::= a node that has subtrees.**

- ✐ **children ::= the roots of the subtrees of a parent.**

- ✐ **siblings ::= children of the same parent.**

- ✐ **leaf (terminal node) ::= a node with degree 0 (no children).**



- **path from n_1 to n_k ::=** a (unique) sequence of nodes $n_1, n_2, \ldots, n_k$ such that n_i is the parent of n_{i+1} for $1 \le i < k$.

- **length of path ::=** number of edges on the path.

- **depth of n_i ::=** length of the unique path from the root to n_i. Depth(root) = 0.

- **height of n_i ::=** length of the longest path from n_i to a leaf. Height(leaf) = 0, and height(D) = 2.

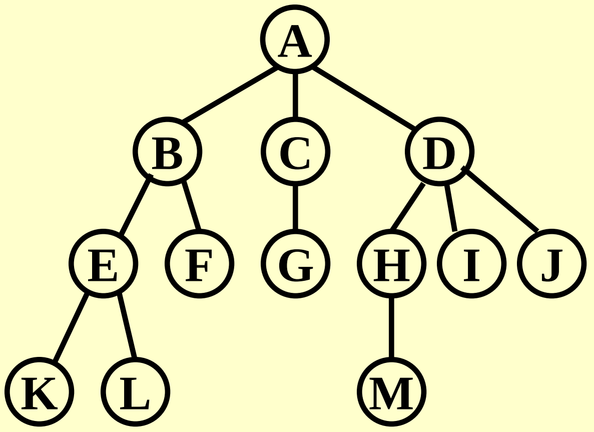
- **height (depth) of a tree ::=** height(root) = depth(deepest leaf).

- **ancestors of a node ::=** all the nodes along the path from the node up to the root.

- **descendants of a node ::=** all the nodes in its subtrees.

2. Implementation

❖ List Representation

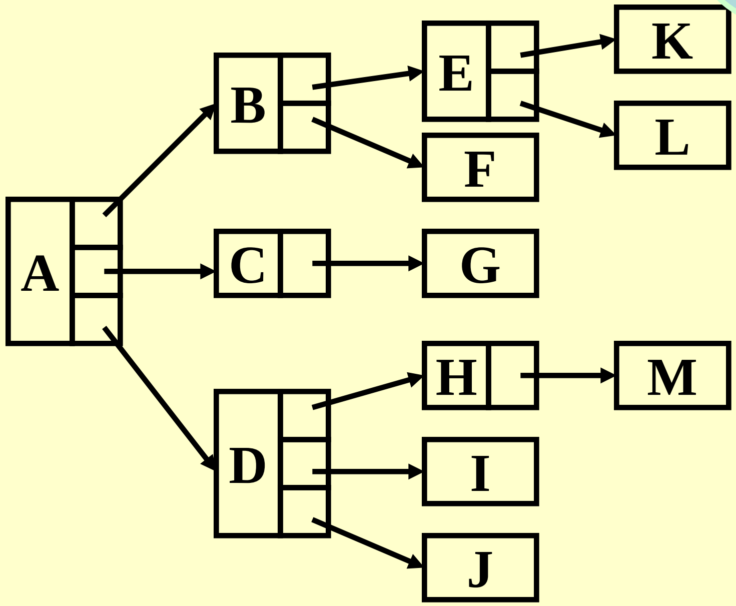


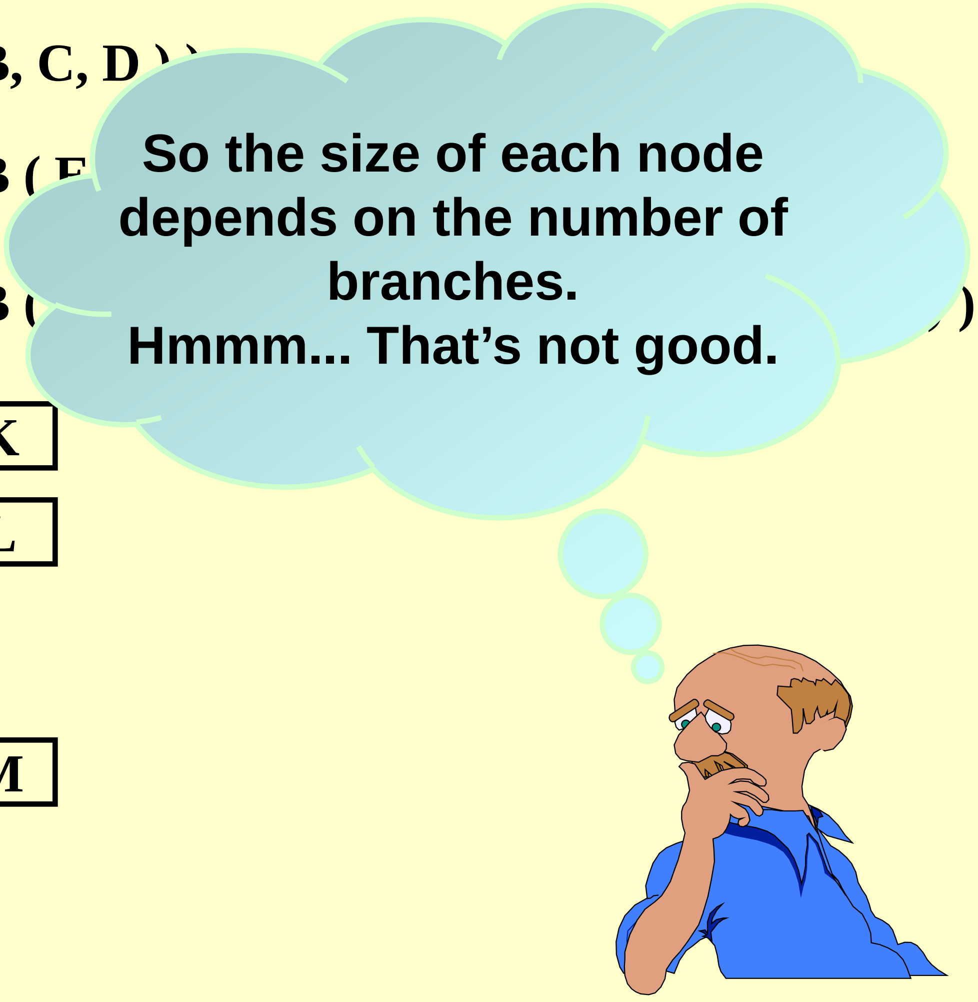
(A)

(A (B, C, D))

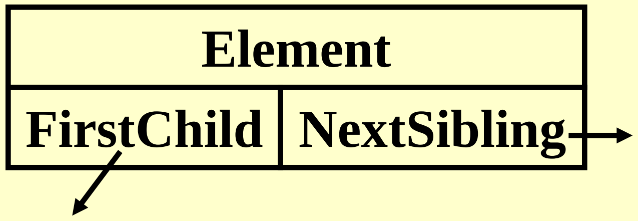
(A (B (E

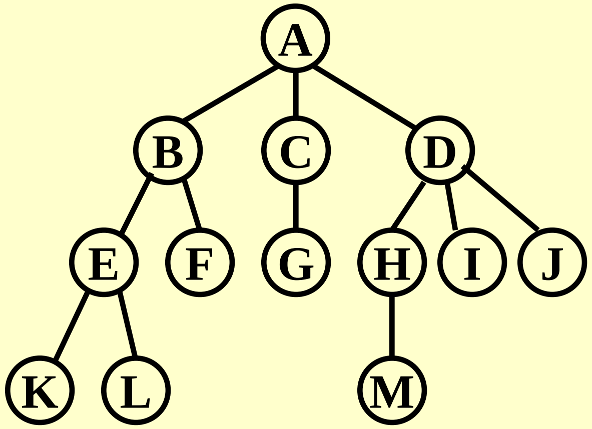
(A (B (   ))

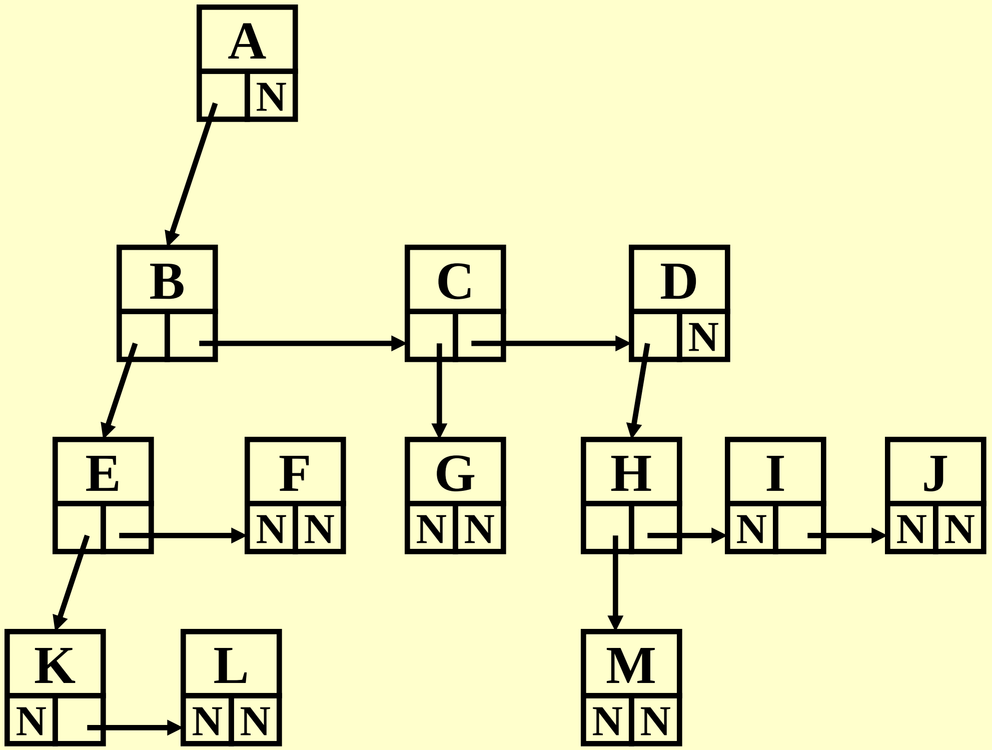




❖ FirstChild-NextSibling Representation





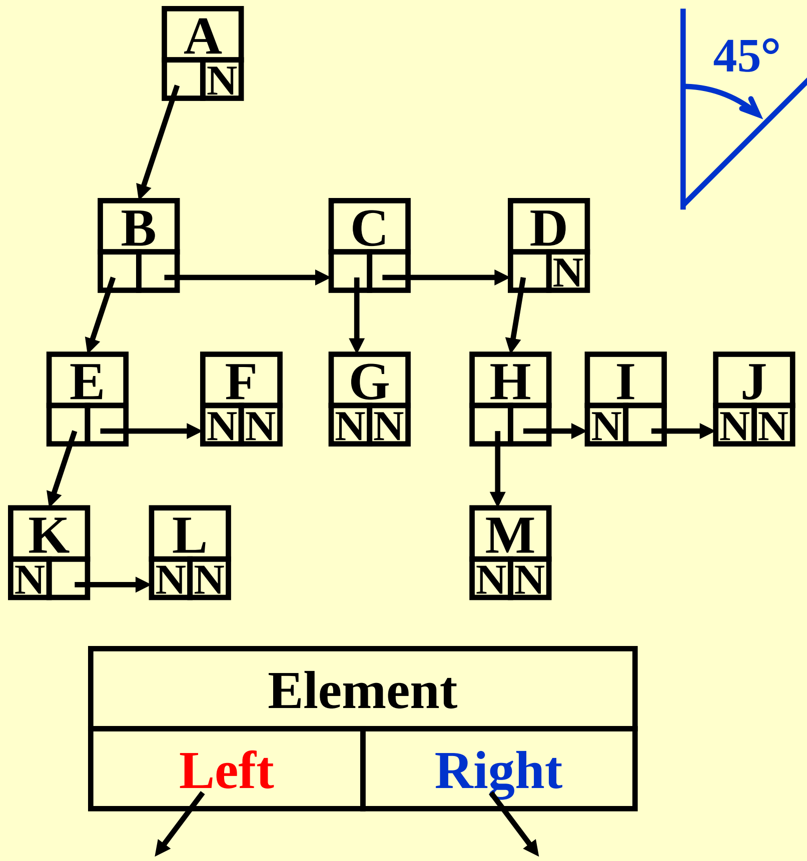


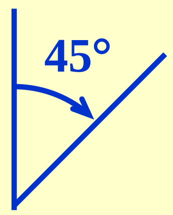
Note: The representation is **not unique** since the children in a tree can be of any order.

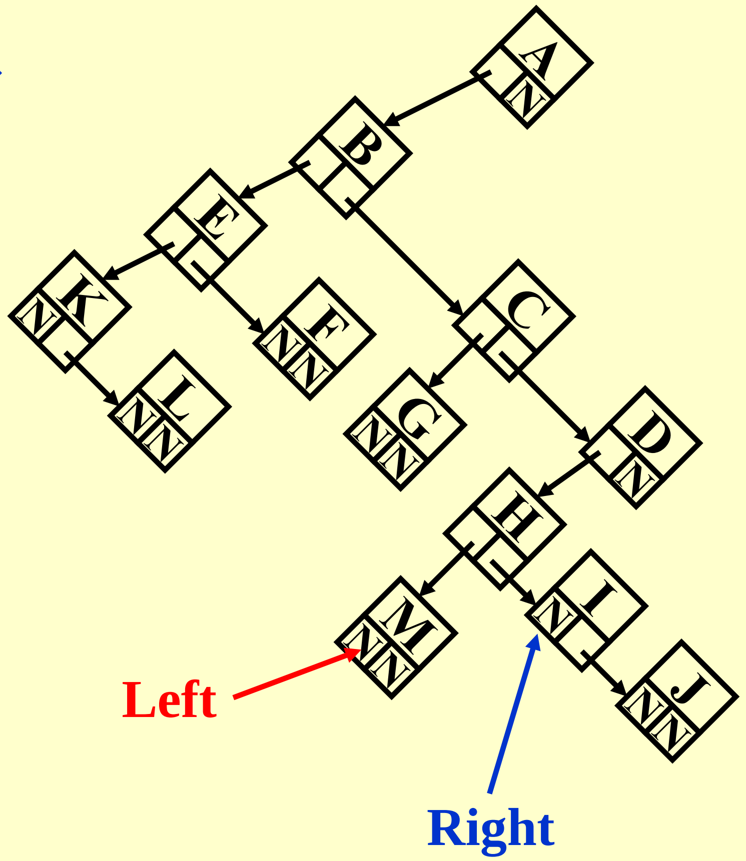
§ 2 Binary Trees

【Definition】A binary tree is a tree in which no node can have more than two children.

Rotate the FirstChild-NextSibling tree clockwise by 45°.

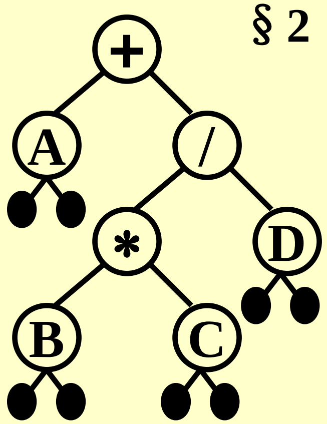






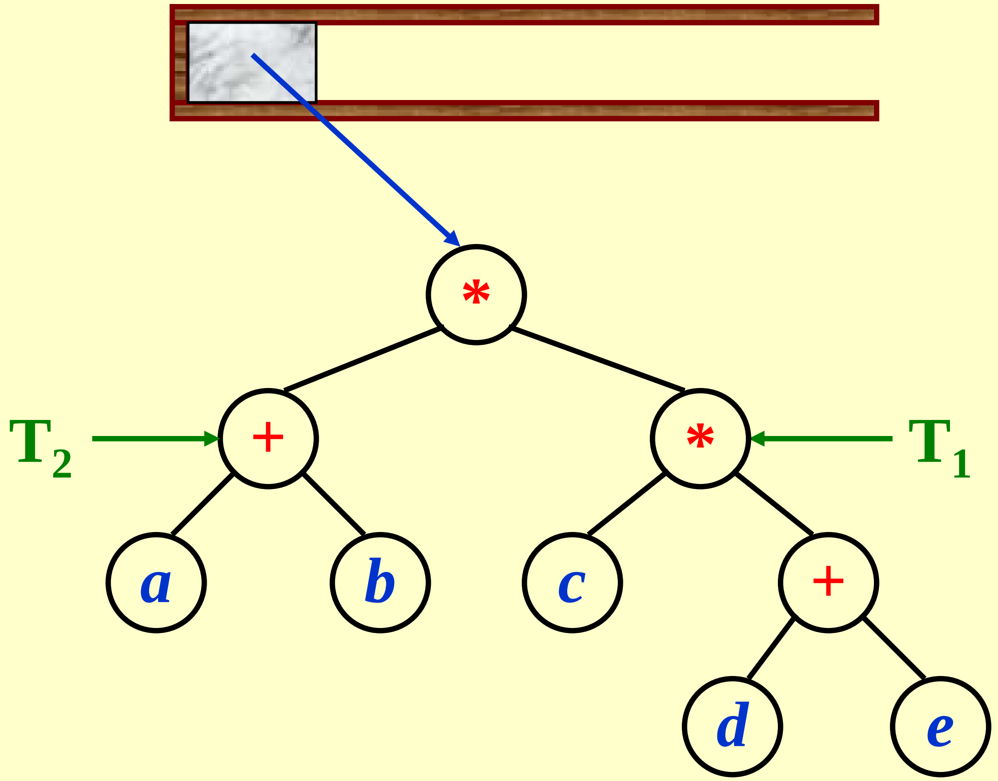
❖ Expression Trees (syntax trees)

〖Example〗 Given an infix expression:
$A + B * C / D$



☞ Constructing an Expression Tree
(from postfix expression)

〖Example〗 $(a + b) * (c * (d + e)) = a\ b + c\ d\ e + * *$



☞ Tree Traversals —— visit each node exactly once

❖ Preorder Traversal

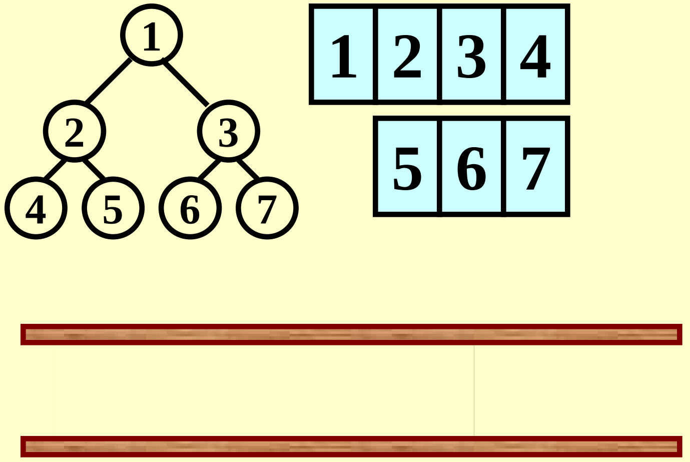
```
void  preorder ( tree_ptr  tree )
{  if  ( tree )   {
        visit ( tree );
        for (each child C of tree )
            preorder ( C );
    }
}
```

❖ Postorder Traversal

```
void  postorder ( tree_ptr  tree )
{  if  ( tree )   {
        for (each child C of tree )
            postorder ( C );
        visit ( tree );
    }
}
```

❖ Levelorder Traversal

```
void  levelorder ( tree_ptr  tree )
{   enqueue ( tree );
    while (queue is not empty) {
        visit ( T = dequeue ( ) );
        for (each child C of T )
            enqueue ( C );
    }
}
```



❖ Inorder Traversal

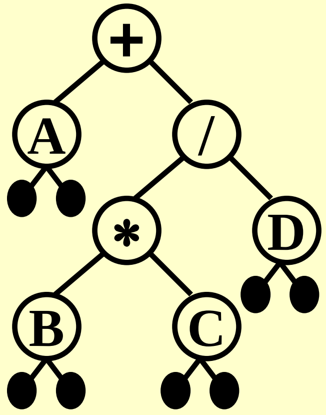
```
void  inorder ( tree_ptr  tree )
{  if  ( tree )   {
        inorder ( tree->Left );
        visit ( tree->Element );
        inorder ( tree->Right );
   }
}
```

Iterative Program

```
void  iter_inorder ( tree_ptr  tree )
{ Stack  S = CreateStack( MAX_SIZE );
  for ( ; ; )  {
     for ( ; tree; tree = tree->Left )
        Push ( tree, S ) ;
     tree = Top ( S );  Pop( S );
     if ( ! tree )  break;
     visit ( tree->Element );
     tree = tree->Right;   }
}
```

〖Example〗 Given an infix expression:

$A + B * C / D$

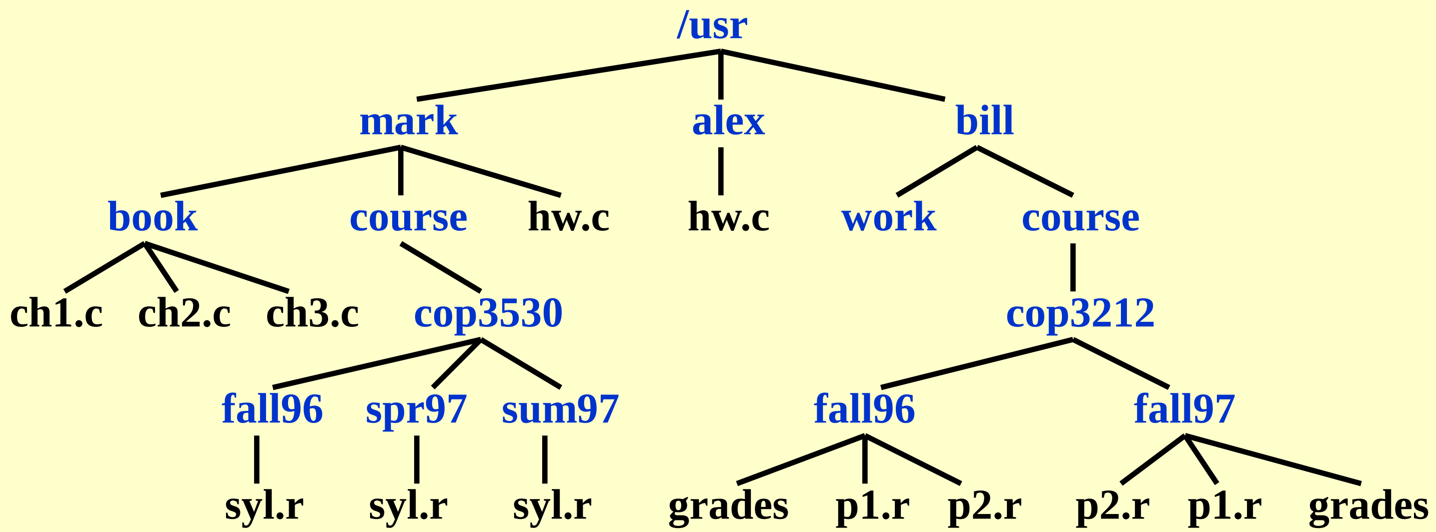


Then inorder traversal ⇒ $A + B * C / D$

postorder traversal ⇒ $A\ B\ C * D / +$

preorder traversal ⇒ $+ A / * B\ C\ D$

〖Example〗 Directory listing in a hierarchical file system.



Unix directory

Listing format: files that are of depth d_i will have their names indented by d_i tabs.

```
/usr
    mark
        book
            Ch1.c
            Ch2.c
            Ch3.c
        course
            cop3530
                fall96
                    syl.r
                spr97
                    syl.r
                sum97
                    syl.r
        hw.c
    alex
        hw.c
    bill
        work
        course
            cop3212
                fall96
                    grades
                    p1.r
                    p2.r
                fall97
                    p2.r
                    p1.r
                    grades
```

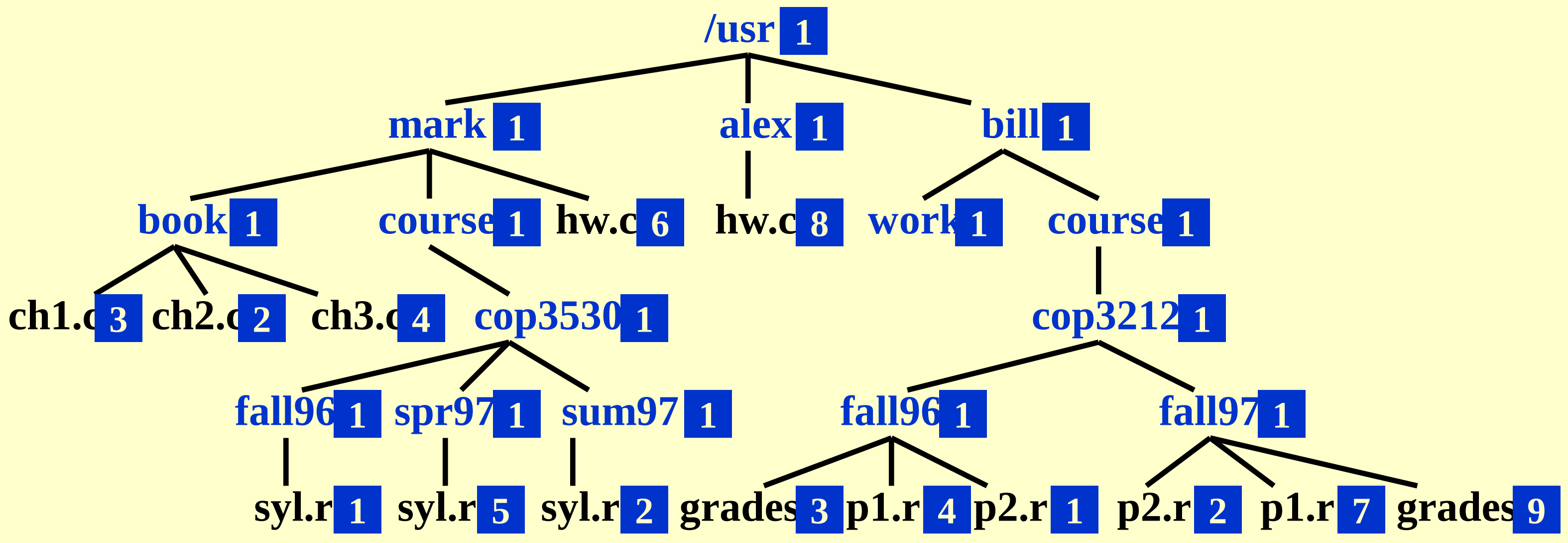
```
static void  ListDir ( DirOrFile D, int Depth )
{
   if  ( D is a legitimate entry )   {
      PrintName (D, Depth );
      if ( D is a directory )
         for (each child C of D )
            ListDir ( C, Depth + 1 );
   }
}
```

$$T(N) = O(N)$$

Note: Depth is an internal variable and must not be seen by the user of this routine. One solution is to define another interface function as the following:

```
void ListDirectory ( DirOrFile  D )
{         ListDir( D, 0 );             }
```

〖Example〗 Calculating the size of a directory.

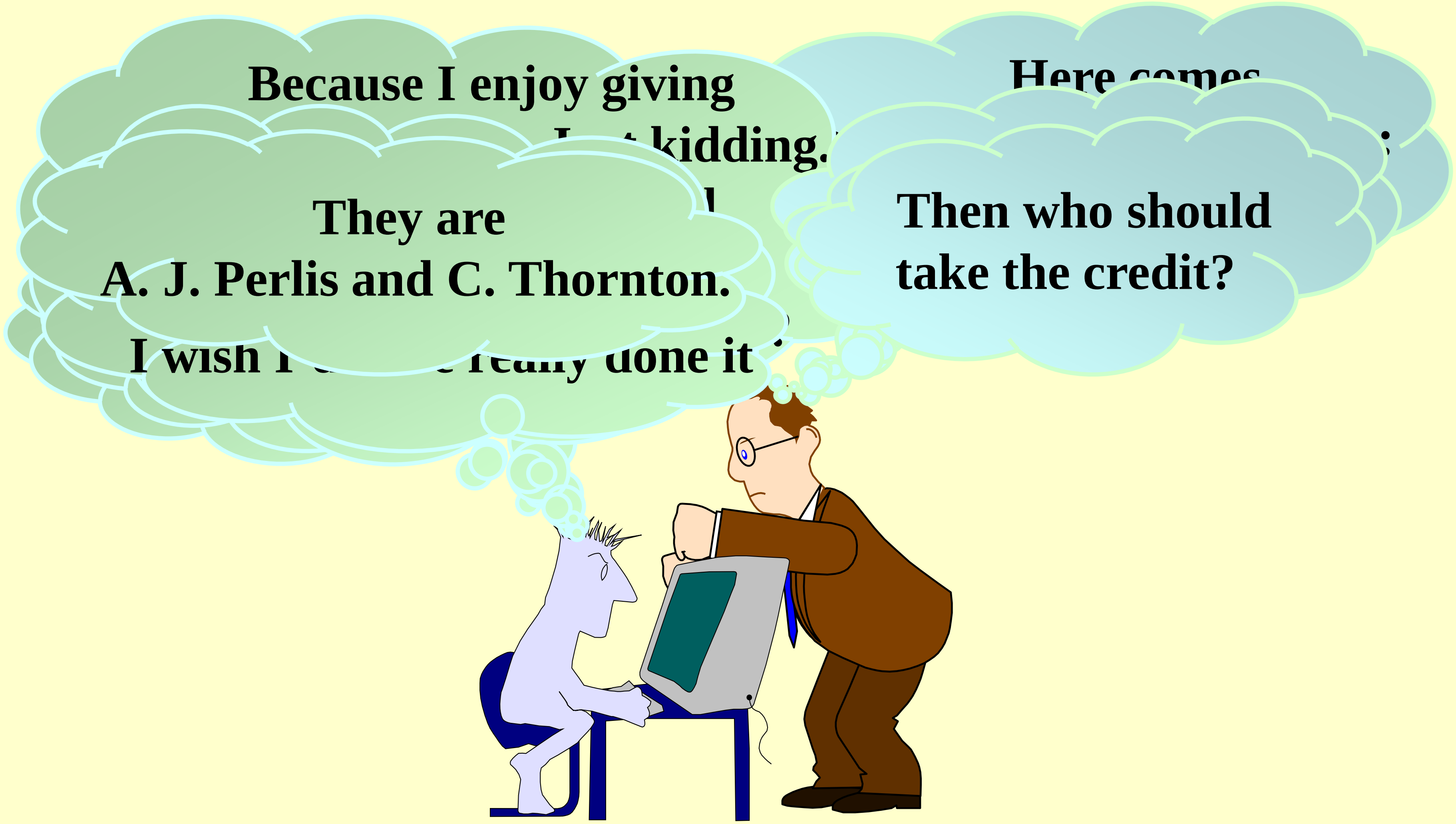


Unix directory with file sizes

```
static int  SizeDir ( DirOrFile D )
{
   int TotalSize;
   TotalSize = 0;
   if  ( D is a legitimate entry )   {
      TotalSize = FileSize( D );
         if ( D is a directory )
            for (each child C of D )
               TotalSize += SizeDir(C);
      } /* end if D is legal */
      return TotalSize;
}
```

$T(N) = O(N)$

❖ Threaded Binary Trees



Rule 1: If Tree->Left is null, replace it with a pointer to the inorder predecessor of Tree.

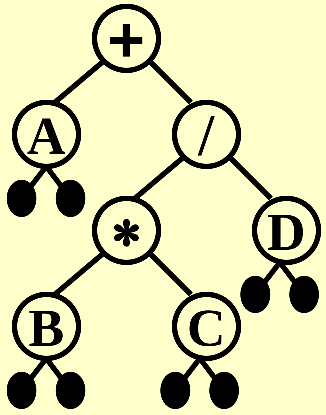
Rule 2: If Tree->Right is null, replace it with a pointer to the inorder successor of Tree.

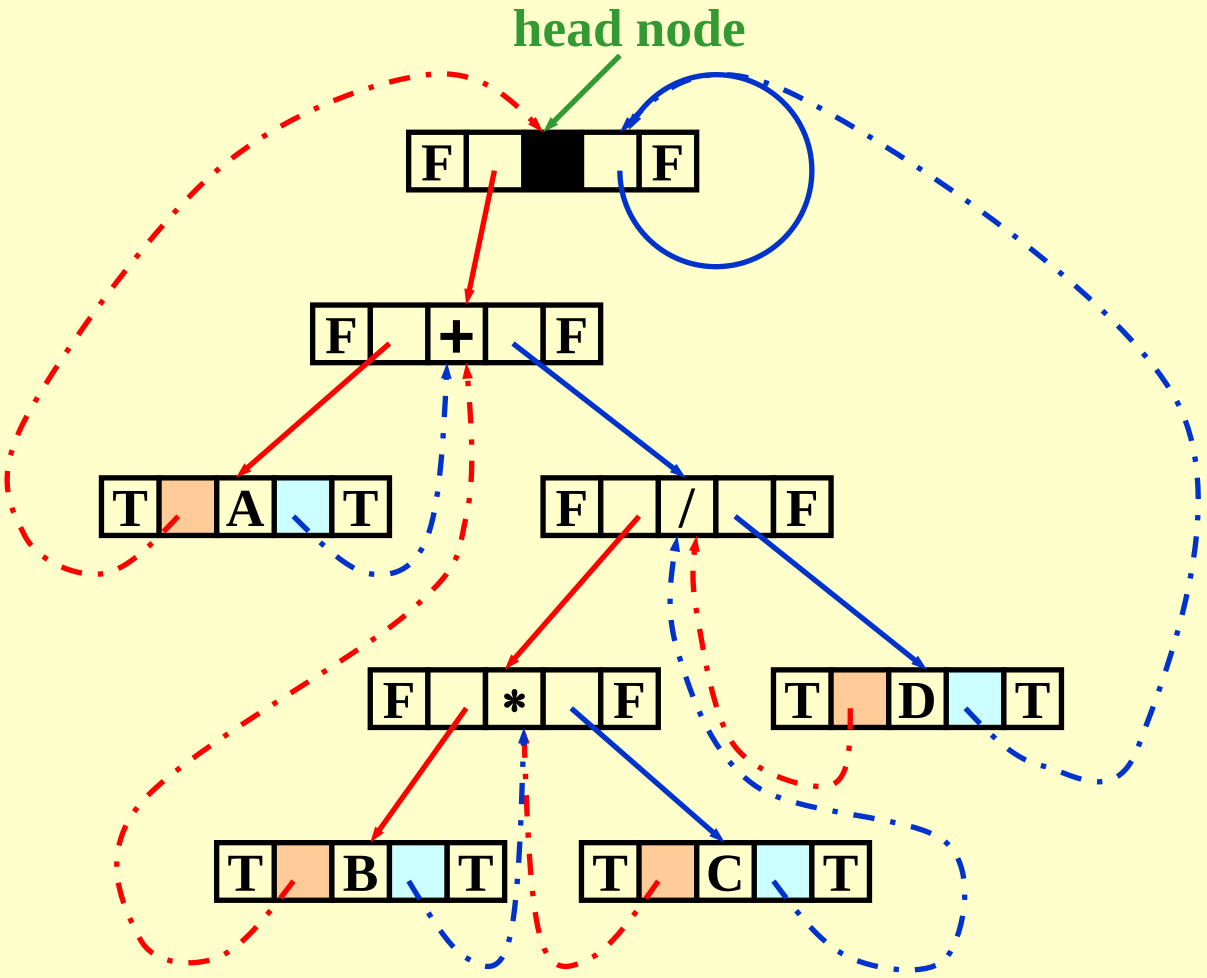
Rule 3: There must not be any loose threads. Therefore a threaded binary tree must have a head node of which the left child points to the first node.

```
typedef  struct  ThreadedTreeNode  *PtrTo  ThreadedNode;
typedef  struct  PtrToThreadedNode  ThreadedTree;
typedef  struct  ThreadedTreeNode {
      int                      LeftThread;   /* if it is TRUE, then Left */
      ThreadedTree             Left;      /* is a thread, not a child ptr.   */
      ElementType              Element;
      int                      RightThread; /* if it is TRUE, then Right */
      ThreadedTree             Right;     /* is a thread, not a child ptr.   */
}
```

〖Example〗 Given the syntax tree of an expression (infix)

$$A + B * C / D$$




head node
F
F
F
+
F
T
A
T
F
/
F
F
*
F
T
D
T
T
B
T
T
C
T