ADT Binary Tree

CreateBinaryTree ()
CreateBinaryTree (RootItem), ...

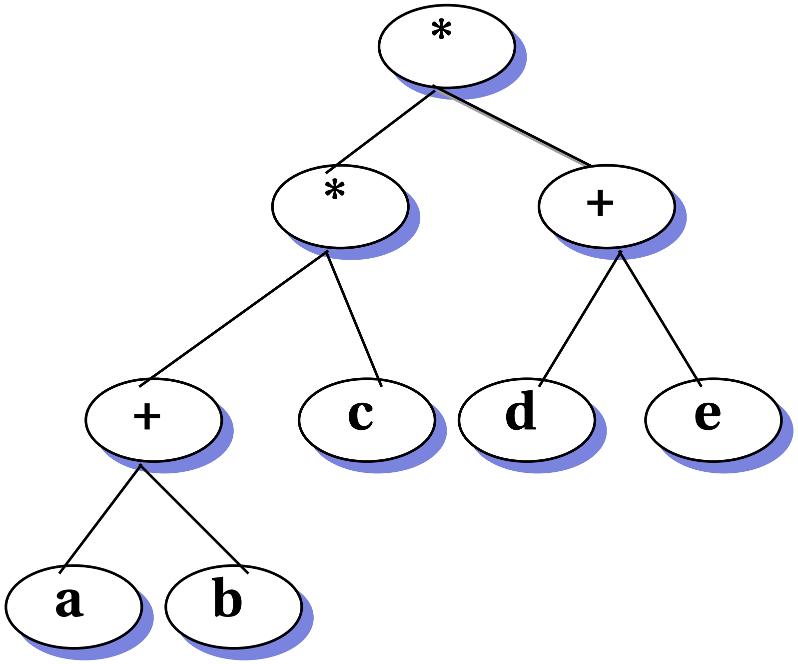
DestroyBinaryTree ()
GetRoot (), SetRootData ()
LeftSubtree (), RightSubtree ()
AttachLeft (NewItem), AttachRight (NewItem)

Traversals:
- **Preorder (Visit)**
- **Inorder (Visit)**
- **Postorder (Visit)**

page 447

Expression Trees

(a+b)*c*(d+e)

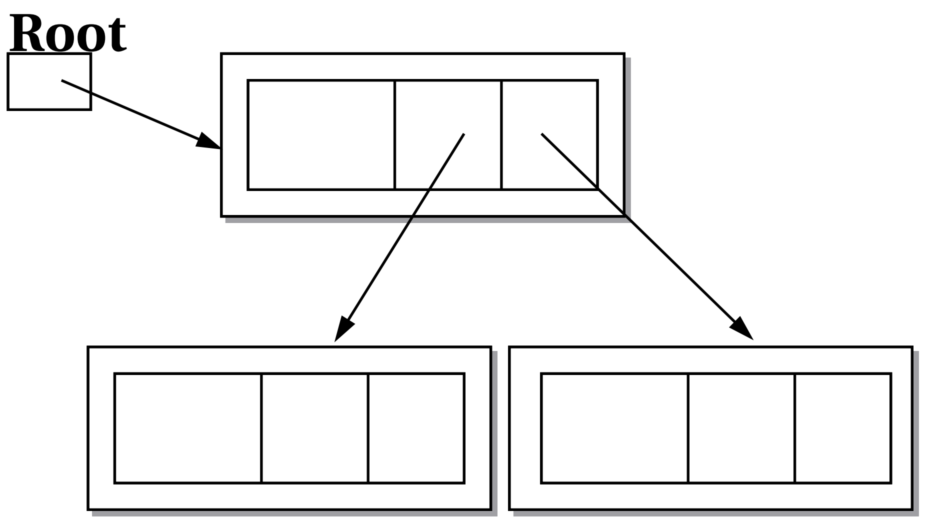


Implementations

Powerful ADT Node

ADT Node

```
template<class T>
class Tree {
   T key;
   Tree *left, *right;
public:
   Tree (T k, Tree *l, Tree *r);
   ~Tree ();
   int Size ();
   int Depth ();
   ...
};
```



Implementation (array)

```
template <class T>
class Tree {
   class TreeNode {
   public:
      T key;
      int left, right;
   };
    int Root, Free;
    TreeNode  Nodes[MSIZE];
public:
    Tree ();
    ~Tree ();
    int Depth ();
    bool AddNode ();
    ...
};
```

Interface

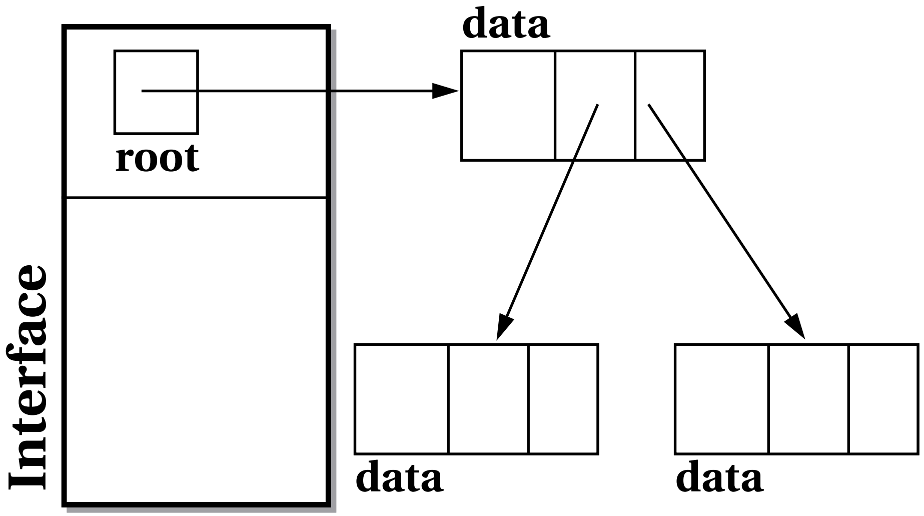
0

5

	1	4	0
	2	-1	1
	3	-1	2
	-1	-1	3
	-1	-1	4
	-1	6	5
	-1	7	6
	-1	8	7
	-1	9	8
	-1	-1	9

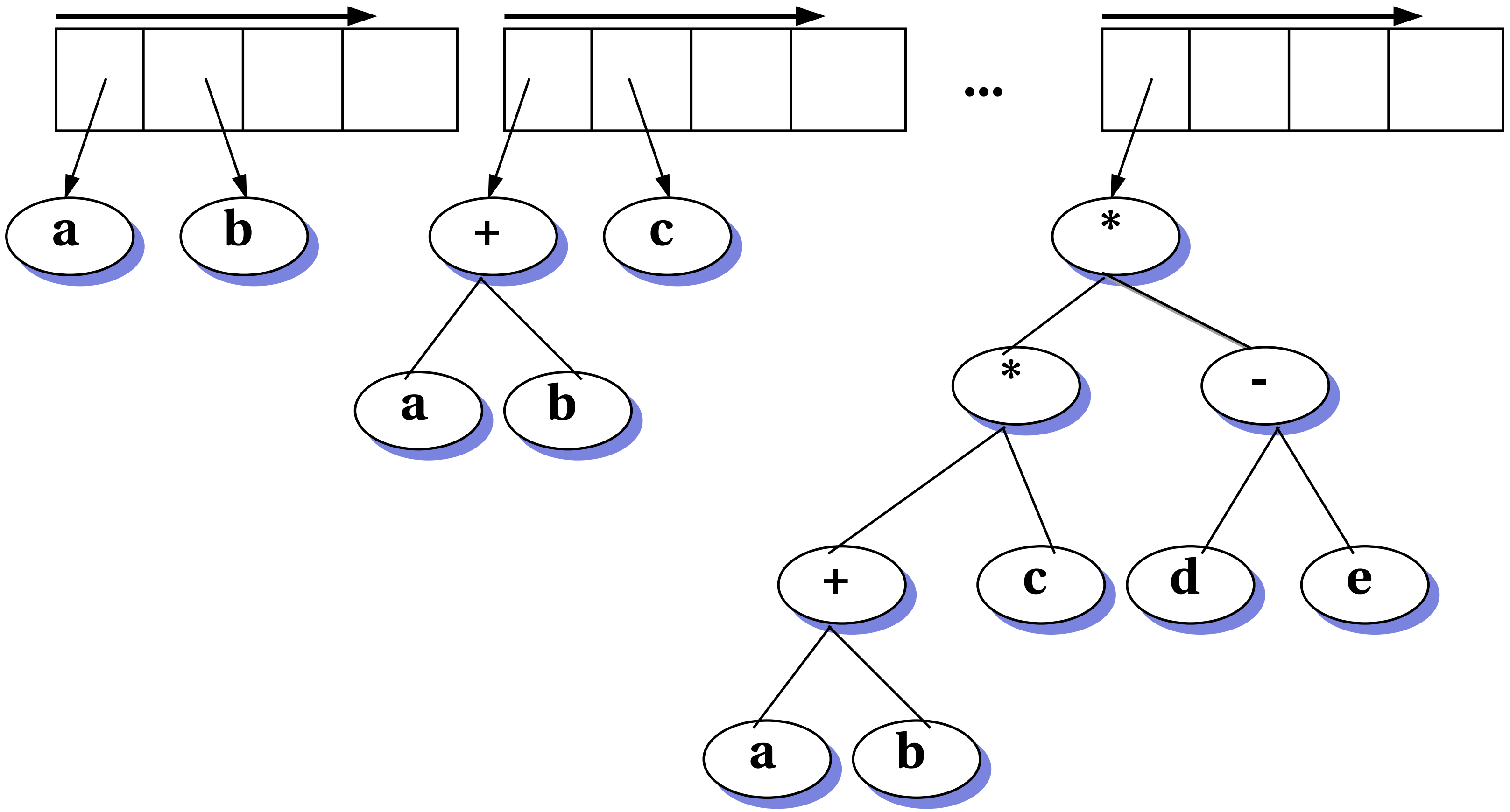
Implementation (ADT)

```
template <class T>
class Tree {
  class TreeNode {
  public:
    T key;
    TreeNode *left, *right;
  };
   Three *Root;
public:
   Tree ();
   ~Tree ();
   int Depth ();
   bool AddLeft (T);
   ...
};
```



Example - Expressions

ab+c*de-*



Example

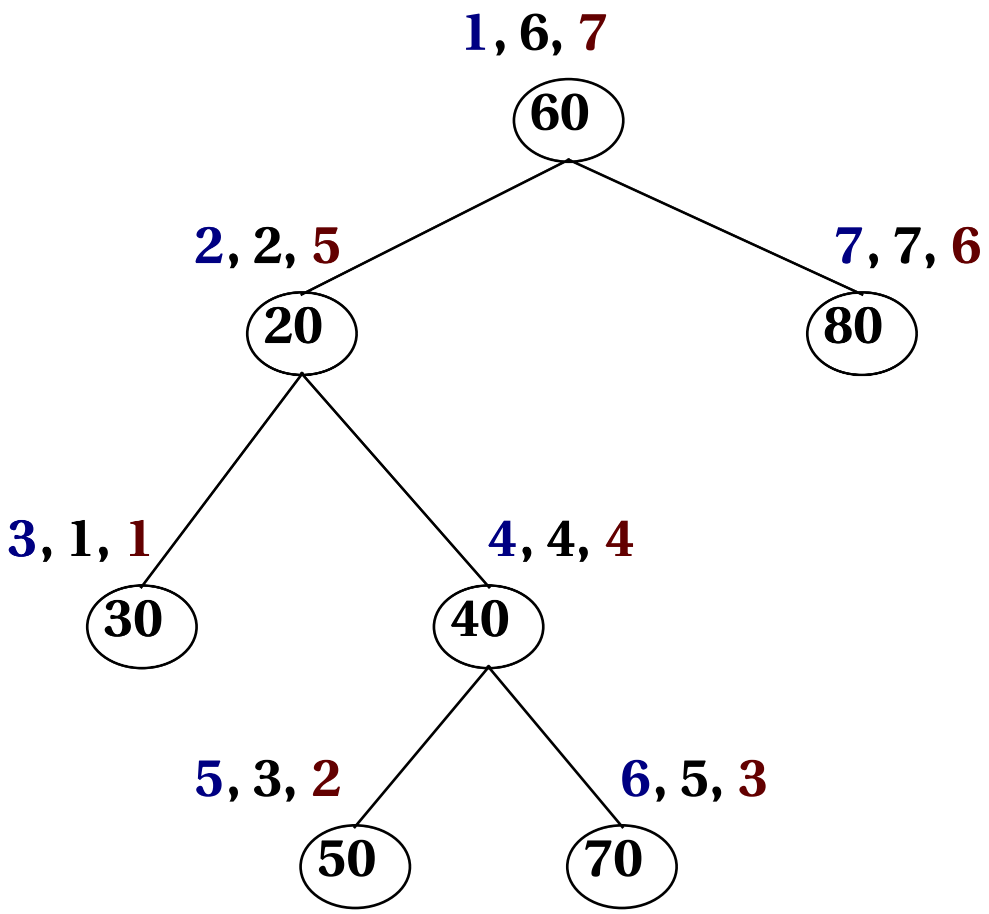
```
char expr[255] = "ab+c*";
BTree<char>  *bt;
Stack< BTree<char>*  > st;

for (int i = 0; expr[i]; i++) {
    if (isalpha (expr[i])) {
       bt = new BTree<char>  (expr[i]);
       st.Push (bt);
    } else {
       BTree<char>  *l, *r;
       st.Pop (r); st.Pop (l);
       bt = new BTree<char>  (expr[i], l, r);
       st.Push (bt);
       delete l; delete r;
     }
  }
```

Traversals

Preorder
60,20,30,40,50,70,80

Inorder
30,20,50,40,70,60,80

Postorder
30,50,70,40,20,80,60

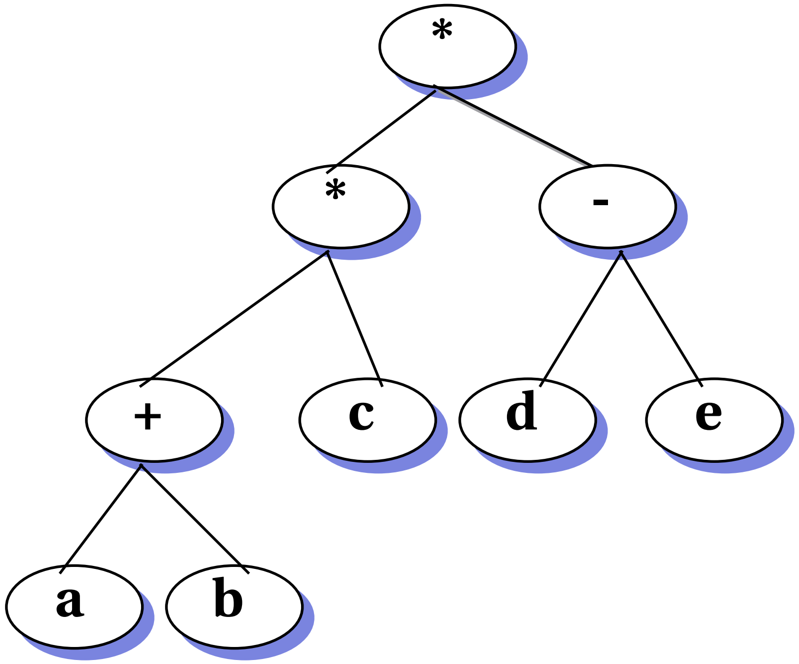


Expression Trees

Infix:
(a+b)*c*(d-e)

Postfix:
ab+c*de-*

Prefix:
**+abc-de



Traversals

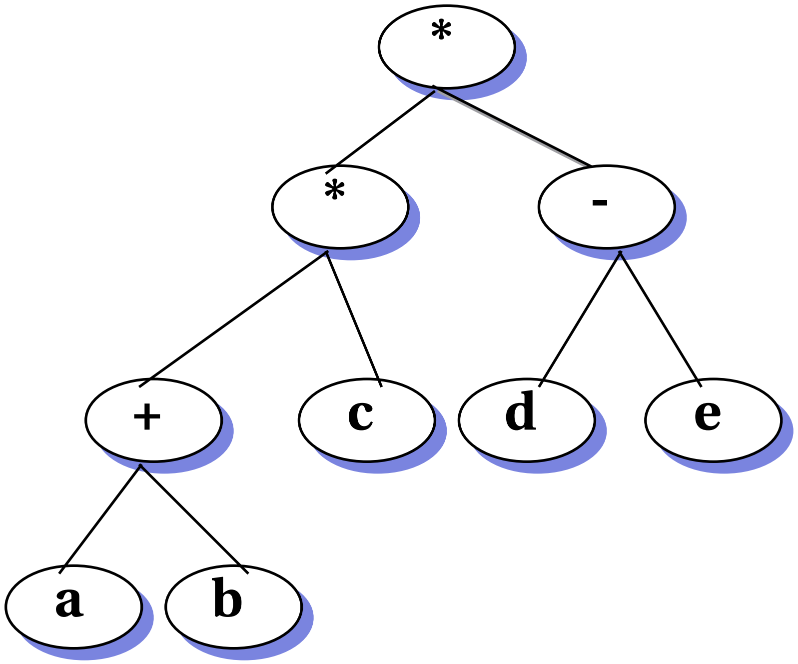
- Inorder
- Postorder
- Preorder

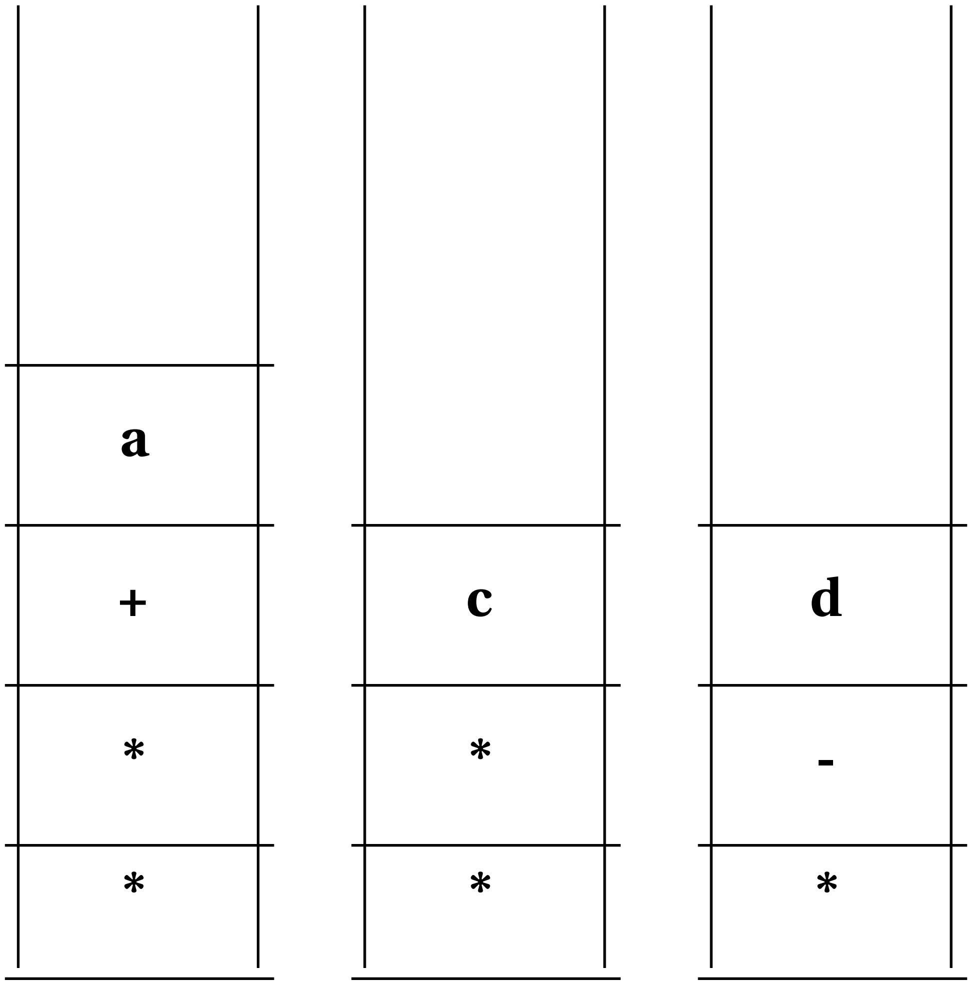
Inorder

```
void inorder (TreeNode* ptr, void (*Visit) (T, int), int d)
 {
  if (ptr != NULL)
    {
     inorder (ptr->left, Visit, d+1);
     Visit (ptr->key, d);
     inorder (ptr->right, Visit, d+1);
    }
 }
```

Inorder - iterative

Idea - simulate the stack





Doubleclick to edi

a+

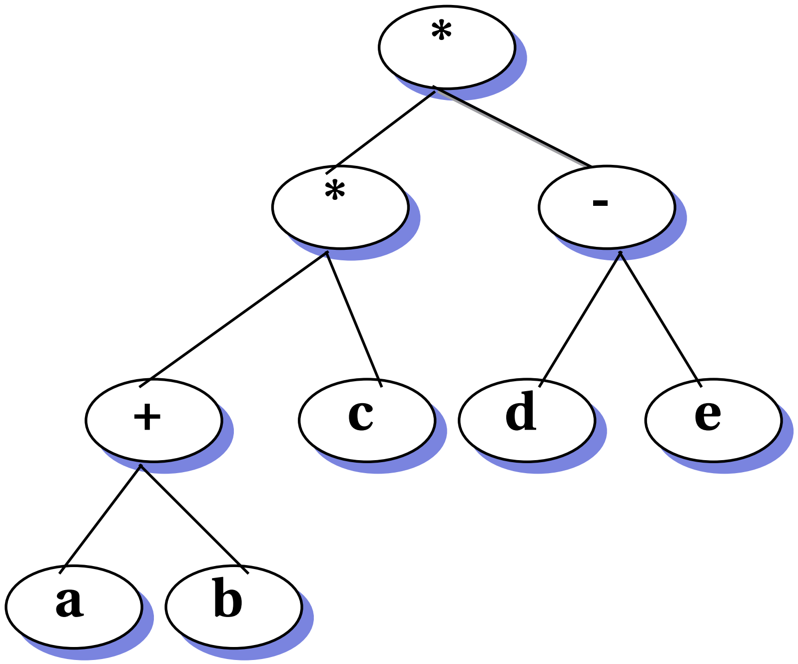
a+b*c*

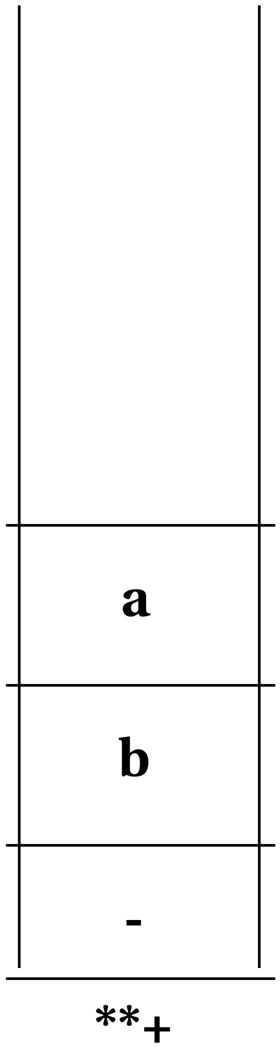
Preorder

```
void preorder (TreeNode* ptr, void (*Visit) (T, int), int d)
 {
  if (ptr != NULL)
   {
    Visit (ptr->key, d);
    preorder (ptr->left, Visit, d+1);
    preorder (ptr->right, Visit, d+1);
   }
 }
```

Preorder - iterative

Idea - save one a stack nodes to visit

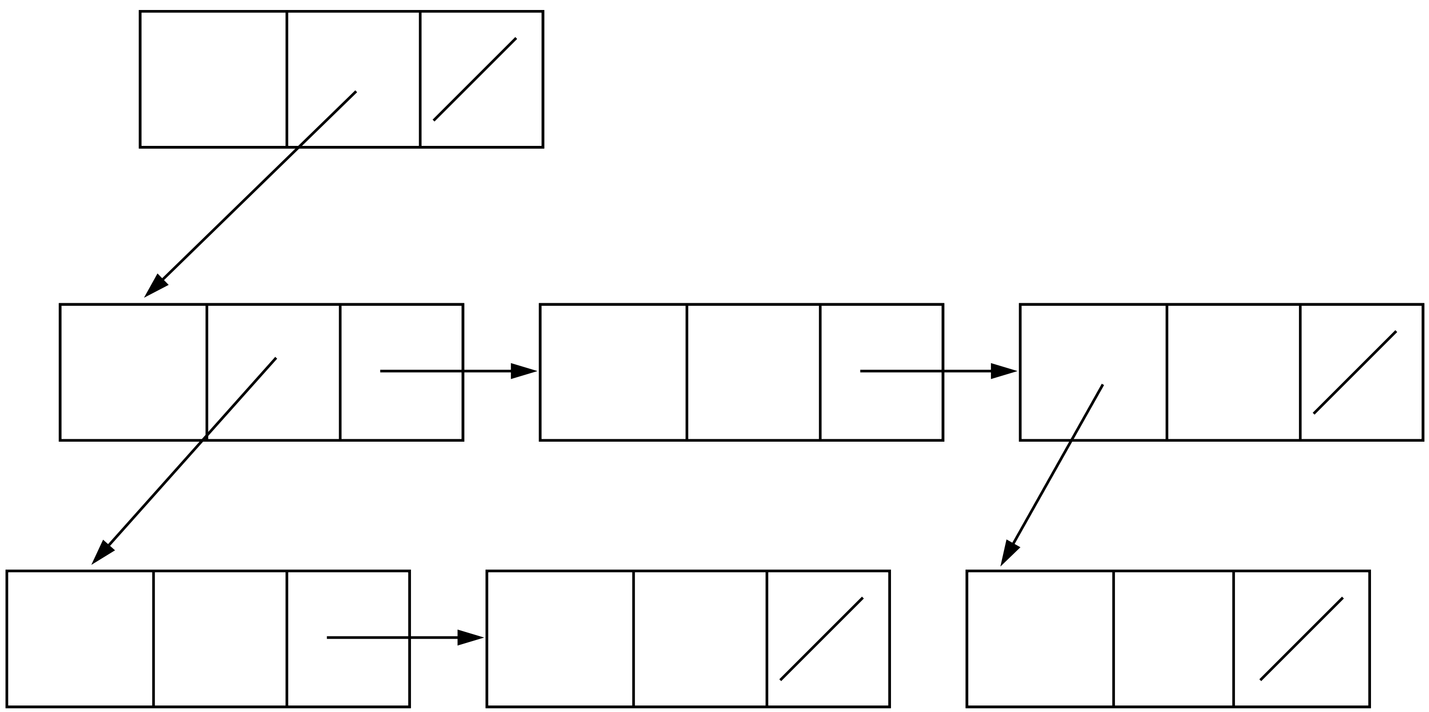




Postorder

```
void postorder (TreeNode* ptr, void (*Visit) (T, int), int d)
 {
  if (ptr != NULL)
   {
    postorder (ptr->left, Visit, d+1);
    postorder (ptr->right, Visit, d+1);
    Visit (ptr->key, d);
   }
 }
```

General Trees



Example

```
template <class T>
class Node {
public:
  T key;
  Node *children, *next;
};
```

N-ary Trees

```
template  <class T>
class Node {
public:
  T key;
   Node **children;
};
```