Tree & Binary Trees (2)

College of Computer Science, CQU

Outline

- Binary Tree Node ADT
- Binary Tree Traversals
  - Level Order
  - Preorder
  - Inorder
  - Postorder
- Traversal algorithm
  - recursive algorithm
  - Non recursive algorithm

Binary Tree Node ADT

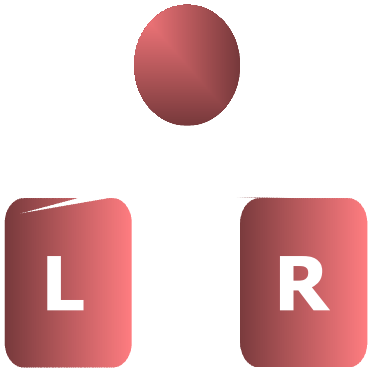
```
// Binary tree node abstract class
template <typename E> class BinNode {
public:
    virtual ~BinNode() {} // Base destructor
    // Return the node's value
    virtual E& element() = 0;
    // Set the node's value
    virtual void setElement(const E&) = 0;
    // Return the node's left child
    virtual BinNode* left() const = 0;
```

Binary Tree ADT

```
  // Set the node's left child
  virtual void setLeft(BinNode*) = 0;
  // Return the node's right child
  virtual BinNode* right() const = 0;
  // Set the node's right child
  virtual void setRight(BinNode*) = 0;
  // Return true if the node is a leaf, false otherwise
  virtual bool isLeaf() = 0;
};
```

Binary Tree Traversals

- Traversal: Each node is visited once and can only be visited once.
- Traversal is easy to Linear structure. But to nonlinear structure, it is needed to linearize nonlinear structure according to certain rules
- The binary tree consists of three basic units: root, left subtree and right subtree.



Binary Tree Traversals

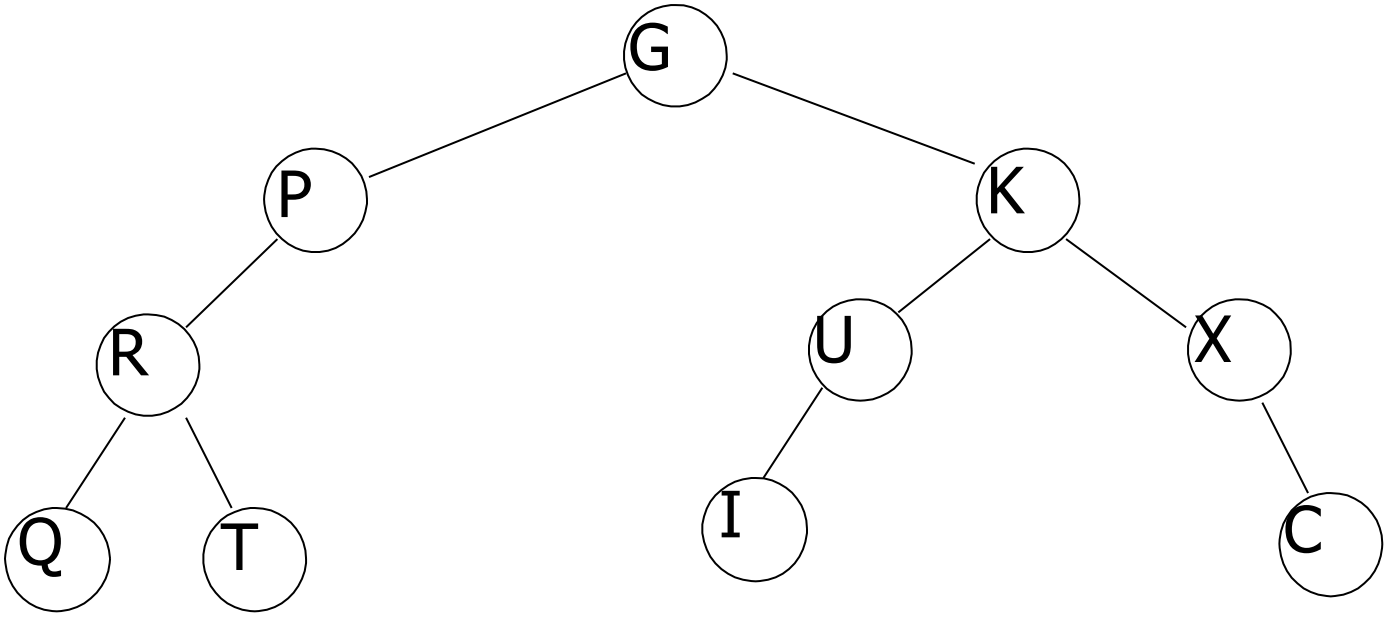
- Let L, D, and R stand for moving left, visiting the root node, and moving right.
- There are six possible combinations of traversal
  - DLR, LDR, LRD, DRL, RDL, RLD
- Adopt convention that we traverse left before right, only 3 traversals remain
  - DLR, LDR, LRD
  - preorder, inorder , postorder

Binary Tree Traversals

- Suppose that we need to visit all of the nodes in a binary tree. In what order can this be done? The most common:
  - Preorder
  - Inorder
  - Postorder
  - Level Order
- Level order is breadth-first traversal, the other three are depth-first traversal.

Preorder Traversal

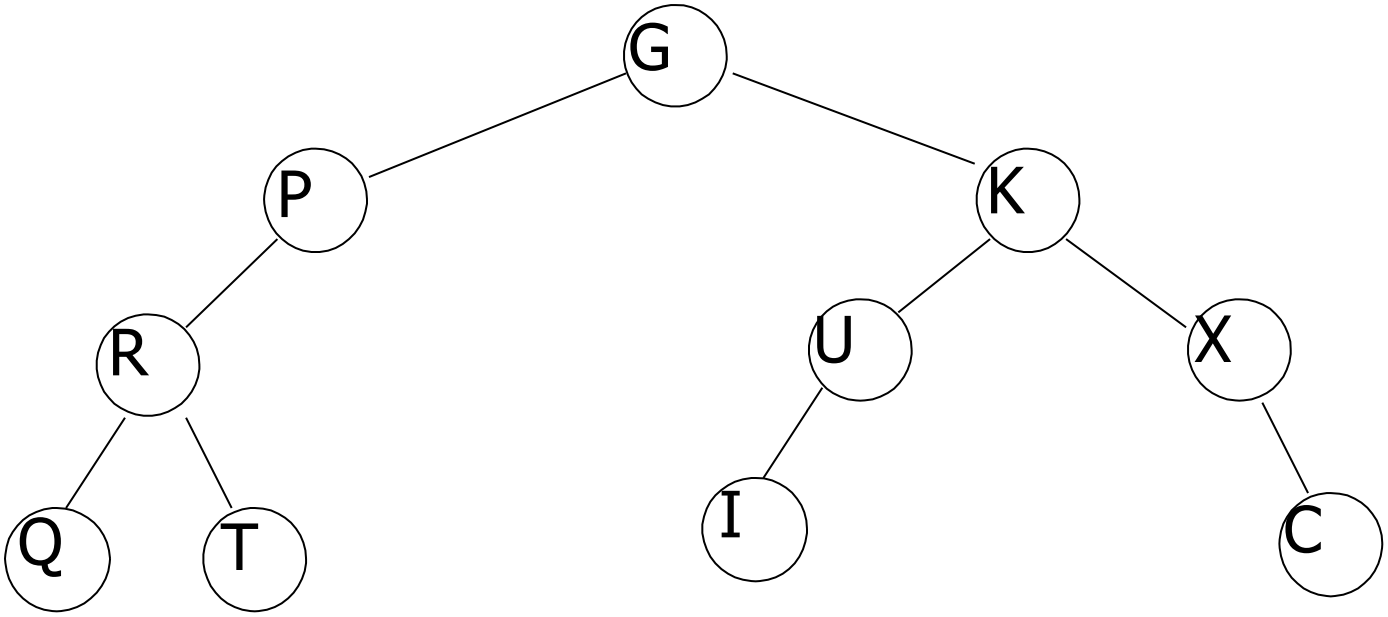
- Visit the root node.
- Traverse the left subtree.
- Traverse the right subtree.



G, P, R, Q, T, K, U, I, X, C

Inorder Traversal

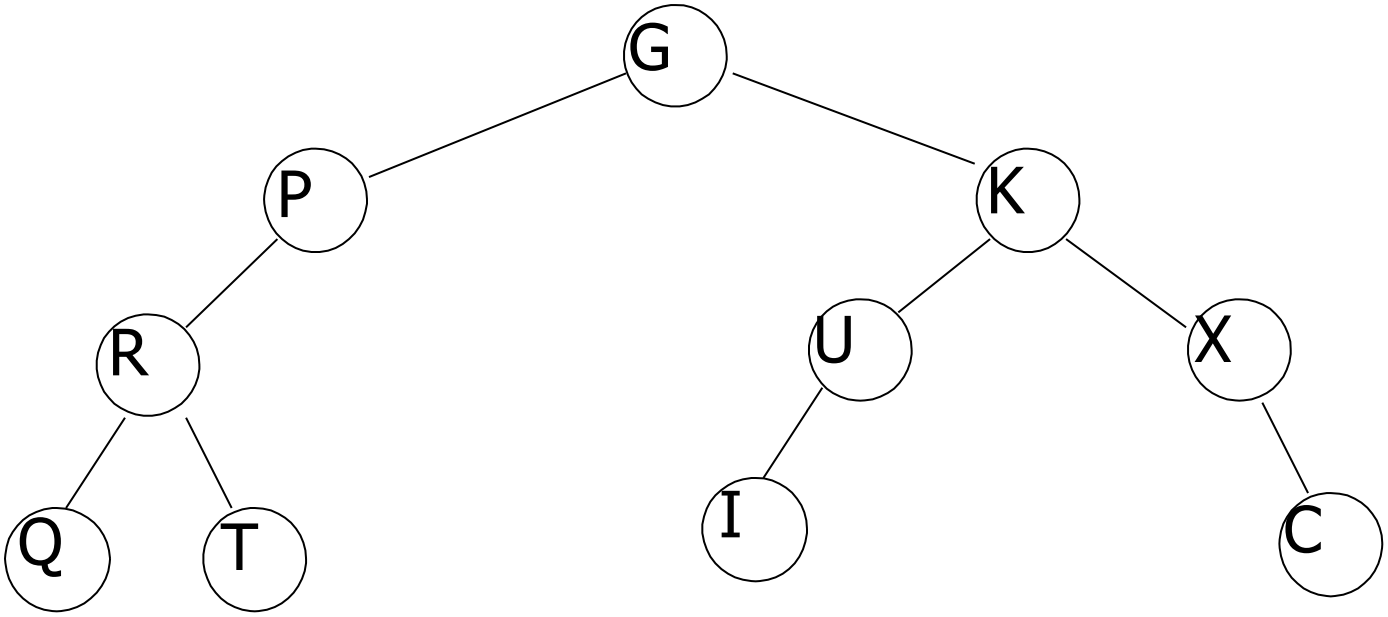
- Traverse the left subtree.
- Visit the root node.
- Traverse the right subtree.



Q, R, T, P, G, I, U, K, X, C

Postorder Traversal

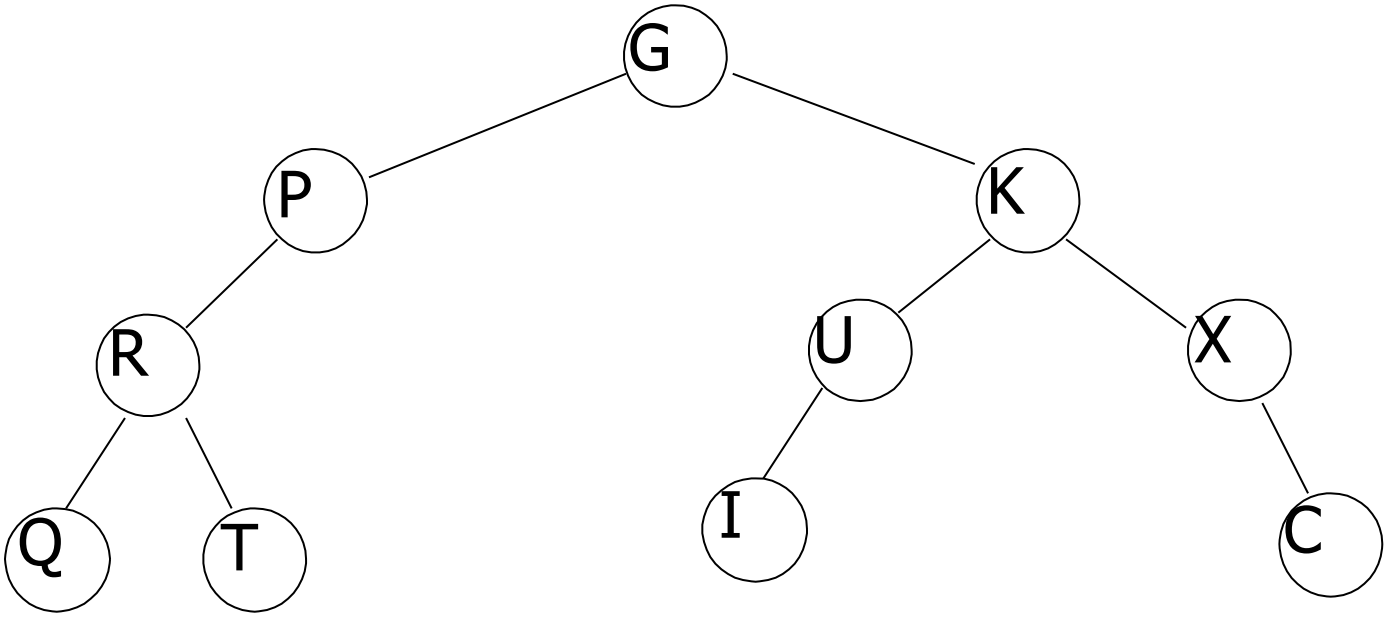
- Traverse the left subtree.
- Traverse the right subtree.
- Visit the root node.



Q, T, R, P, I, U, C, X, K, G

Level Order Traversal

- Visit the nodes from level to level, beginning with the root node.

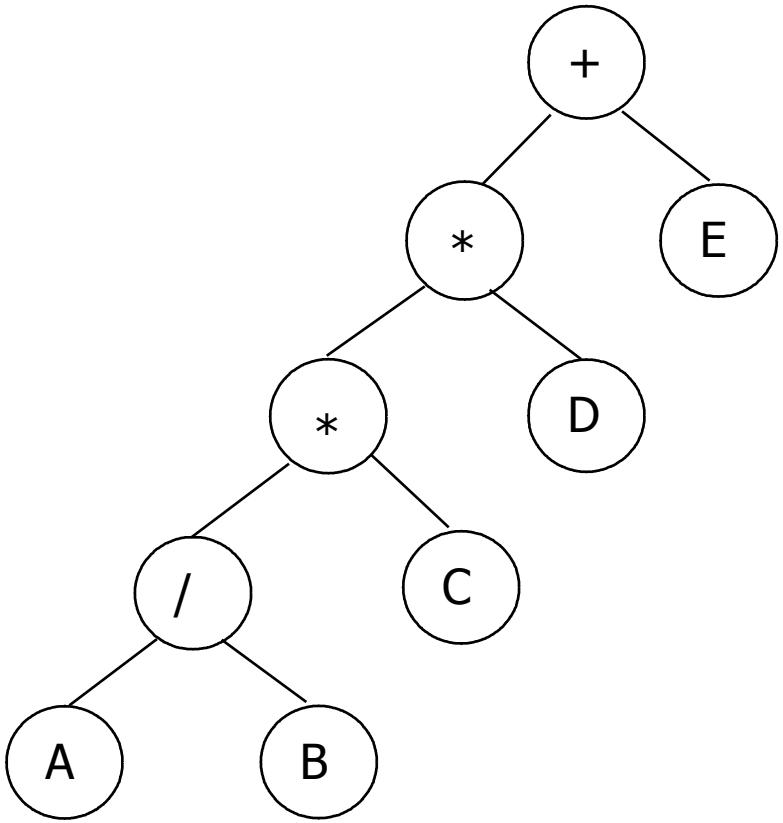


G, P, K, R, U, X, Q, T, I, C

Expression Tree

- A Binary Tree built with operands and operators.
- Also known as a parse tree.
- Used in compilers.
- Notation
  - Preorder
    - Prefix Notation
  - Inorder
    - Infix Notation
  - Postorder
    - Postfix Notation

Arithmetic Expression Using BT



inorder traversal

A / B * C * D + E

infix expression

preorder traversal

+ * * / A B C D E

prefix expression

postorder traversal

A B / C * D * E +

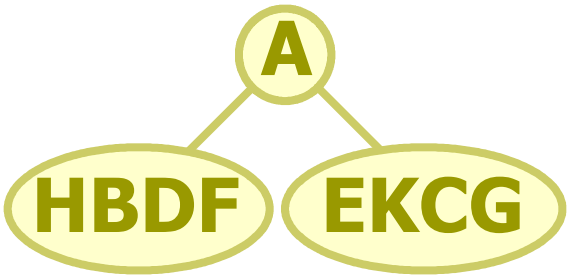
postfix expression

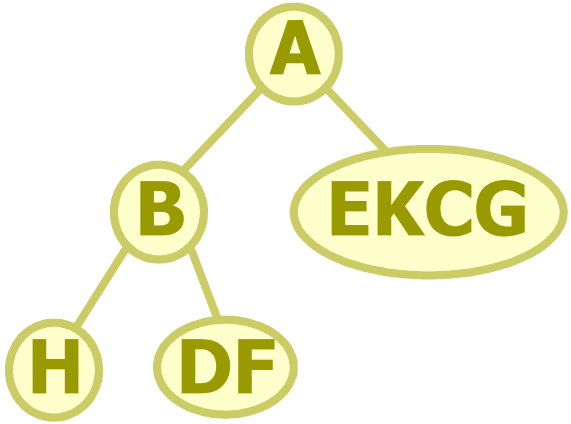
level order traversal

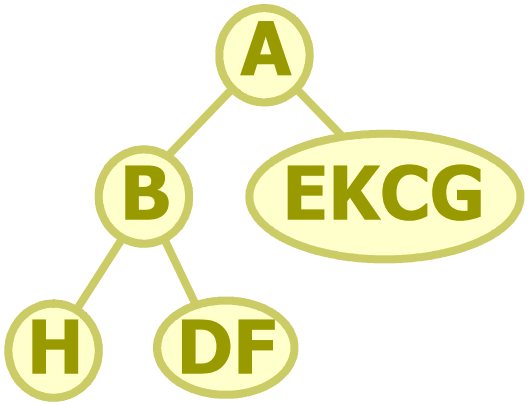
+ * E * D / C A B

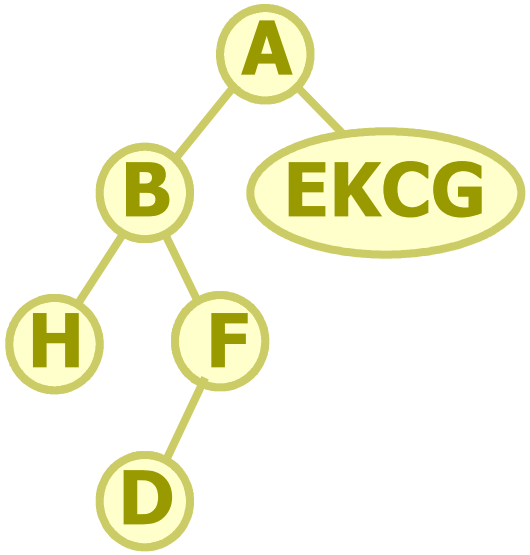
Binary Tree Traversals

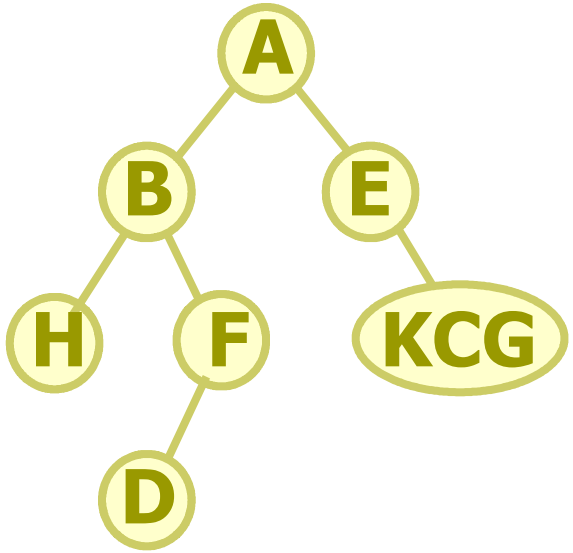
- A binary tree can be Uniquely constructed by preorder enumeration and inorder enumeration.
- For example, the preorder enumeration is { ABHFDECKG } and the inorder enumeration is { HBDFAEKCG }, the constructing process of binary tree is as follows:

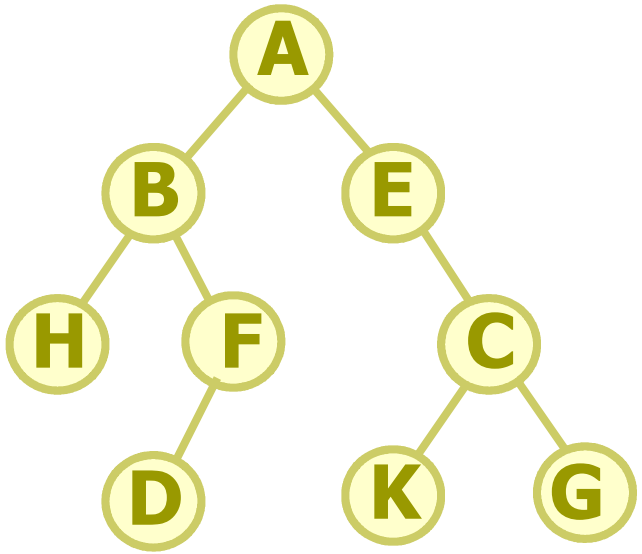




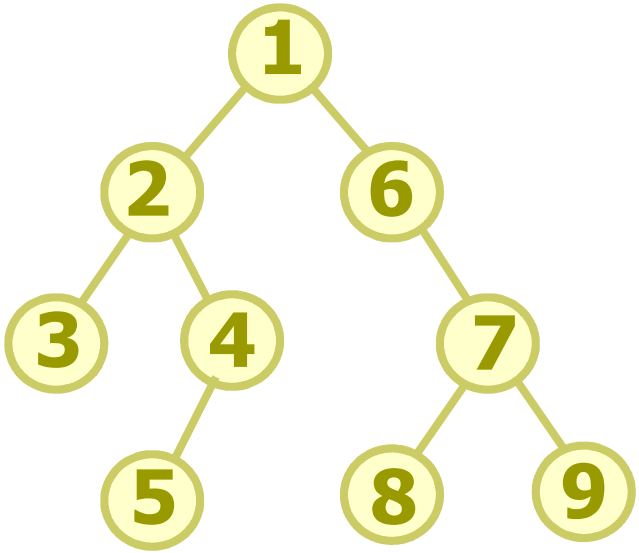

A
B
EKCG
H
DF

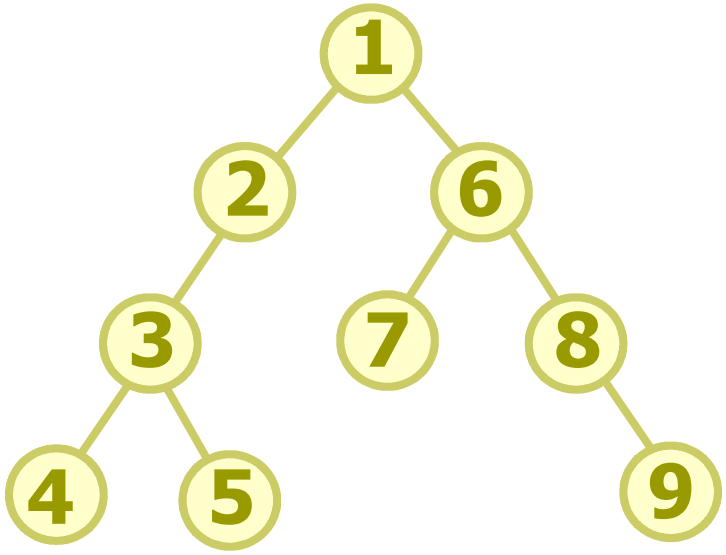

A
B
EKCG
H
F
D


A
B
E
H
F
KCG
D


A
B
E
H
F
C
D
K
G

- If there is only a preorder enumeration {1, 2, 3, 4, 5, 6, 7, 8, 9} ,we can get different binary tree.





- A binary tree also can be uniquely constructed by postorder enumeration and inorder enumeration.

Preorder Traversal *(recursive)*

- A traversal routine is naturally written as a recursive function.

```
template <typename E>
void preorder(BinNode<E>* root) {
   if (root == NULL) return;   // Empty subtree, do
                               // nothing
   visit(root);                // Perform desired action
   preorder(root->left());
   preorder(root->right());
}
```

Preorder Traversal--preorder2

- An important decision in the implementation of any recursive function on trees is when to check for an empty subtree.
- An alternate design as follows:

```
template <typename E>
void preorder2(BinNode<E>* root) {
   visit(root); // Perform whatever action is desired
   if (root->left() != NULL) preorder2(root->left());
   if (root->right() != NULL) preorder2(root->right());
}
```

Preorder & preorder2

- **The design of preorder2 is inferior to that of preorder for following two reasons:**

(1) It can become awkward to place the check for the NULL pointer in the calling code.

(2) The more important concern with preorder2 is that it tends to be error prone.

- the original tree is empty
- Solution:

  ① an additional test for a NULL pointer at the beginning

  ② the caller of preorder2 has a hidden obligation to pass in a non-empty tree

Preorder Traversal

- Another issue to consider when designing a traversal is how to define the visitor function that is to be executed on every node.
  - Approach1: to write a new version of the traversal for each such visitor function.
  - Approach2: for the tree class to supply a generic traversal function which takes the visitor either as a template parameter or as a function parameter.

Count the Number of Nodes

```
template <typename E>
int count(BinNode<E>* root) {
   if (root == NULL) return 0;     // Nothing to count
   return 1 + count(root->left())
            + count(root->right());
}
```

Inorder Traversal *(recursive)*

```
template <typename E>
void inorder(BinNode<E>* root) {
   if (root == NULL) return;   // Empty subtree, do
                               // nothing
      inorder(root->left());
      visit(root);                    // Perform desired action
      inorder(root->right());
}
```

Postorder Traversal *(recursive)*

```
template <typename E>
void inorder(BinNode<E>* root) {
   if (root == NULL) return;   // Empty subtree, do
                               // nothing
      postorder(root->left());
      postorder(root->right());
      visit(root);             // Perform desired action
}
```

Level Order Traversal *(using queue)*

```
void PrintLevelOrder( T){
    Queue Q;
    BinNode* B;
    Enqueue(Q,T);       // Insert root into Q
    while ( !Q.IsEmpty() ) {
        B = DeQueue(Q);
        printf( B->Element );
        if (B->Left != NULL)
            Enqueue(Q,B->Left); // Insert left child into Q
        if(B->Right!=NULL)
            Enqueue(Q,B->Right);        // Insert right child into Q
    }
}
```

Non recursive algorithm

- Basic idea of inorder traversal using a stack:
  - Push a node into the stack when meet it, and traversal its left subtree, after pop this node and visit it, and then traversal its right subtree.

Inorder Traversal *(Non recursive)*

```
void InOrderUnrec(BinNode *root){
    stack<BinNode*> s;
    BinNode *p=root;
    while (p!=NULL || s.length()!=0) {
       while (p!=NULL) {//遍历左子树
          s.push(p);
          p=p->left();
       }
       if (s.length()!=0){
          p=s.pop();
          visite(p-> Element ); //访问根结点
          p=p->right(); //通过下一次循环实现右子树遍历
       }//endif
    }//endwhile
    }
```

Reference

- Chapter 5
  - 5.1.2 & 5.2: P155—P160
- 《数据结构（ C 语言版）》，严蔚敏，吴伟民编著，清华大学出版社，1997年第1版，P128-132

-End-