```cpp
//日期: 2018/ 时间:
#include <stdio.h>
#include <stdlib.h>
#include <queue>
using namespace std;

struct node{
    int data;
    node* lchild;
    node* rchild;
    int layer;      //层次遍历时用到
};

//新建节点
node* newNode(int v){
    node* Node = new node;  //申请一个node型变量的地址空间
    Node->data = v;
    Node->lchild = Node->rchild = NULL;
    return Node;
}

//二叉树的查找、修改
void search(node* root,int x,int newdata){
    if(root == NULL){
        return;     //空树，走进死胡同（递归边界）
    }
    if(root->data == x){
        root->data = newdata;
    }
    search(root->lchild,x,newdata);
    search(root->rchild,x,newdata);
}
//二叉树的插入
//注意根节点要使用引用，否则插入不会成功
void insert(node* &root,int x){
    if(root == NULL){           //空树，说明查找失败，也即插入位置（递归边界）
        root = newNode(x);
        return;
    }
    /*
    if(由二叉树的某种性质，x应该插入在左子树){
        insert(root->lchild,x); //往左子树搜索
    } else {
        insert(root->rchild,x); //往右子树搜索
    } */
}

//二叉树的创建
node* Create(int data[],int n){
    node* root = NULL;          //新建空根节点root
    for(int i=0;i<n;i++){
        insert(root,data[i]);
    }
    return root;        //返回根节点
}

```

```
57 //*****************************二叉树的遍历*********************************
58 //先序遍历,先根遍历
59 void preorder(node* root){
60     if(root == NULL){
61         return;          //到达空树，递归边界
62     }
63     printf("%d\n",root->data);  //访问根节点root将其数据输出
64     preorder(root->lchild);
65     preorder(root->rchild);
66 }
67
68 //后序遍历，后根遍历
69 void inorder(node* root){
70     if(root == NULL){
71         return;          //到达空树，递归边界
72     }
73     inorder(root->lchild);
74     printf("%d\n",root->data);  //访问根节点root将其数据输出
75     inorder(root->rchild);
76 }
77
78 //后序遍历，后根遍历
79 void postorder(node* root){
80     if(root == NULL){
81         return;          //到达空树，递归边界
82     }
83     postorder(root->lchild);
84     postorder(root->rchild);
85     printf("%d\n",root->data);  //访问根节点root将其数据输出
86 }
87
88 //层次遍历
89 void LayerOrder(node* root){
90     queue<node*> q;          //注意队列里存放地址
91     q.push(root);            //将根节点地址入队
92     while(!q.empty()){
93         node* now = q.front();      //取出队首元素
94         q.pop();
95         printf("%d\n",now->data);       //访问队首元素
96         if(now->lchild != NULL) q.push(now->lchild);    //左子树非空
97         if(now->rchild != NULL) q.push(now->rchild);
98     }
99 }
100
101 int main(){
102
103
104     return 0;
105 }
106
107
```