//5.2.4节 算法PreOrder

```
//递归先根遍历以结点t为根结点的子树
template <class T>
void BinTree<T>::PreOrder(BinTreeNode<T> *t) const
{
    if   ( t != NULL )                    //当子树为空时，终止遍历
        {
            cout << t->GetData() <<endl ;      // 输出结点 t 的数据值
            PreOrder ( t -> GetLeft() );       // 先根遍历 t 的左子树
            PreOrder ( t ->GetRight() );       // 先根遍历 t 的右子树
        }
};
```

5.2 节 链接存储二叉树类 BinTree 的定义

```
/*  二叉树结点类声明*/
template<class T>
class BinTreeNode
{
private:
    BinTreeNode<T> *left , *right;                          // 指向左、右子结点的指针
    T data ;                                                 // 数据域
public:
    BinTreeNode(const T & item,BinTreeNode<T> *lptr = NULL,BinTreeNode<T> *rptr = NULL):
    data(item), left(lptr), right(rptr)   { }    // 构造函数
    BinTreeNode<T> * GetLeft(void)const { return left ; }        //返回左子结点
    void SetLeft(BinTreeNode<T> *L){ left = L ; }               //设置左子结点
    BinTreeNode<T> * GetRight(void)const { return right ; }      //返回右子结点
    void SetRight(BinTreeNode<T> *R){ right = R ;}              //设置右子结点
    T& GetData() {     return     data;   }
    void SetData(const T & item){ data = item ; }
} ;
/*二叉树类BinTree的声明*/
template <class T>
class BinTree
{
private:
    BinTreeNode<T>   *root;                                  // 指向二叉树根结点的指针
    T stop;                                          //构造二叉树时的输入结束符，即：若输入stop则停止输入
public:
    BinTree(BinTreeNode<T> * t=NULL): root(t) { }          // 构造函数
    virtual ~ BinTree(){ Del(root) ; }                       // 析构函数，删除整棵二叉树
    void PreOrder(BinTreeNode<T> *t)const ;            // 先根遍历并输出以结点t为根结点的子树
    void InOrder(BinTreeNode<T> *t)const ;             // 中根遍历并输出以结点t为根结点的子树
    void PostOrder(BinTreeNode<T> *t)const ;           // 后根遍历并输出以结点t为根结点的子树
    void LevelOrder (BinTreeNode<T> *t )const;   // 层次遍历并输出以结点t为根结点的子树
    void NorecPreOrder(BinTreeNode<T> *t)const ;      // 非递归先根遍历并输出以结点t为根的子树
    void NorecInOrder(BinTreeNode<T> *t)const ;       // 非递归中根遍历并输出以结点t为根的子树
    void NorecPostOrder(BinTreeNode<T> *t)const;     // 非递归后根遍历并输出以结点t为根的子树
    void CreateBinTree(T tostop) ;                   //创建二叉树
    BinTreeNode<T> * Create() ;
    BinTreeNode<T> * CopyTree ( BinTreeNode<T> *t );              // 复制以结点t为根的二叉树
    // 在以结点t为根结点的子树中搜索结点p的父结点
    BinTreeNode<T> *Father(BinTreeNode<T> *t,BinTreeNode<T> *p);
    //在以结点t为根结点的子树中查找data域为item的结点
    BinTreeNode<T> * Find(BinTreeNode<T> *t,const T & item) const ;
    void InsertLeft(BinTreeNode<T> *t,T item);   //在结点t的左侧插入data域为item的结点
    void InsertRight(BinTreeNode<T> *t,T item);   //在结点t的右侧插入data域为item的结点
    void DelSubtree( BinTreeNode<T> *t );                // 从树中删除结点t及其左右子树
```

```
    void Del(BinTreeNode<T> *t);                    // 删除结点t及其左右子树

    //其他操作
    BinTreeNode<T> *GetRoot() { return root ; }
    void SetRoot(BinTreeNode<T> * t) { root=t ; }
    T GetStop() { return stop ; }
    void SetStop(T tostop) {    stop=tostop ; }
    int IsEmpty(){ return root == NULL ? 1 : 0 ; }          //判断二叉树是否为空
} ;
```