

前序遍历二叉树  
void PreOrder(tree root)

{  
Tree stack[MAXSIZE], node;  
int k=-1;  
   
if (root==NULL){  
printf("tree is empty\n");  
return;  
}  
else

{  
k++;  
stack[k] = root;   
while (k > -1)

{  
node = stack[k--];  
printf("%c", node->data);  
if (node->right != NULL)

{  
stack[++k] = node->right;  
}  
if (node->left != NULL){  
stack[++k] = node->left;  
}   
}   
}  
}

中序遍历二叉树  
void InOrder(tree root)

{  
Tree stack[MAXSIZE], node;  
int top = 0;  
if (root == NULL){  
printf("tree is empty\n");  
return;  
}  
node = root;   
while (node != NULL || top > 0)

{  
while (node != NULL){  
stack[++top] = node;  
node = node->left;  
}  
node = stack[top--];  
printf("%c", node->data);  
node = node->right;  
}  
   
}

后序遍历二叉树

void BackOrder(tree root){  
Tree stack[MAXSIZE], node;  
int top = 0;  
int count = 0;  
char arr[MAXSIZE];    
if (root == NULL){  
printf("tree is empty\n");  
return;  
}  
else{  
top++;  
stack[top] = root; while (top > 0){  
node = stack[top--];  
arr[count++] = node->data;    
// 先把右子树放进去，栈是先进去后出，所以下面的左子树先出  
if (node->left != NULL){  
stack[++top] = node->left;  }  
if (node->right != NULL){  
stack[++top] = node->right;  
}   
}  
}  
for (int i=count-1; i>=0; i--){  
printf("%c", arr[i]);  
}   
}

总结：

综上来看通过比较递归操作的确简单方便，非递归的实现主要还是通过栈的先进后出的特点。先序遍历是把左子树压栈，再把右子树放入栈中。中序遍历先编写函数从根节点开始遍历左孩子同时压栈，然后调用访问右孩子，重复操作。后序遍历先把根节点压栈，然后放入左节点，最后放右节点。

最初遇到的困难是根本无从下手，多亏老师提醒了栈的用法，然后写出大体的流程图进而转化为代码实现。