```
#include <stdio.h>
#include <string.h>
#define TElemType int
//构造结点的结构体
typedef struct BiTNode{
    TElemType data;//数据域
    struct BiTNode *lchild,*rchild;//左右孩子指针
}BiTNode,*BiTree;
//初始化树的函数
void CreateBiTree(BiTree *T){
    *T=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->data=1;
    (*T)->lchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->rchild=(BiTNode*)malloc(sizeof(BiTNode));

    (*T)->lchild->data=2;
    (*T)->lchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->lchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->lchild->rchild->data=5;
    (*T)->lchild->rchild->lchild=NULL;
    (*T)->lchild->rchild->rchild=NULL;
    (*T)->rchild->data=3;
    (*T)->rchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->rchild->lchild->data=6;
    (*T)->rchild->lchild->lchild=NULL;
    (*T)->rchild->lchild->rchild=NULL;
    (*T)->rchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->rchild->rchild->data=7;
    (*T)->rchild->rchild->lchild=NULL;
    (*T)->rchild->rchild->rchild=NULL;
    (*T)->lchild->lchild->data=4;
    (*T)->lchild->lchild->lchild=NULL;
    (*T)->lchild->lchild->rchild=NULL;
}

//模拟操作结点元素的函数，输出结点本身的数值
void displayElem(BiTNode* elem){
    printf("%d ",elem->data);
}
//后序遍历
void PostOrderTraverse(BiTree T){
    if (T) {
        PostOrderTraverse(T->lchild);//遍历左孩子
        PostOrderTraverse(T->rchild);//遍历右孩子
        displayElem(T);//调用操作结点数据的函数方法
```

```
    }
    //如果结点为空，返回上一层
    return;
}
int main() {
    BiTree Tree;
    CreateBiTree(&Tree);
    printf("后序遍历: \n");
    PostOrderTraverse(Tree);
}
```

```
#include <stdio.h>
#include <string.h>
#define TElemType int
int top=-1;//top变量时刻表示栈顶元素所在位置
//构造结点的结构体
typedef struct BiTNode{
    TElemType data;//数据域
    struct BiTNode *lchild,*rchild;//左右孩子指针
}BiTNode,*BiTree;
//初始化树的函数
void CreateBiTree(BiTree *T){
    *T=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->data=1;
    (*T)->lchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->rchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->lchild->data=2;
    (*T)->lchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->lchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->lchild->rchild->data=5;
    (*T)->lchild->rchild->lchild=NULL;
    (*T)->lchild->rchild->rchild=NULL;
    (*T)->rchild->data=3;
    (*T)->rchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->rchild->lchild->data=6;
    (*T)->rchild->lchild->lchild=NULL;
    (*T)->rchild->lchild->rchild=NULL;
    (*T)->rchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->rchild->rchild->data=7;
    (*T)->rchild->rchild->lchild=NULL;
    (*T)->rchild->rchild->rchild=NULL;
    (*T)->lchild->lchild->data=4;
    (*T)->lchild->lchild->lchild=NULL;
    (*T)->lchild->lchild->rchild=NULL;
```

```
}
//弹栈函数
void pop( ){
    if (top==-1) {
        return ;
    }
    top--;
}
//模拟操作结点元素的函数，输出结点本身的数值
void displayElem(BiTNode* elem){
    printf("%d ",elem->data);
}

//后序遍历非递归算法
typedef struct SNode{
    BiTree p;
    int tag;
}SNode;
//后序遍历使用的进栈函数
void postpush(SNode *a,SNode sdata){
    a[++top]=sdata;
}
//后序遍历函数
void PostOrderTraverse(BiTree Tree){
    SNode a[20];//定义一个顺序栈
    BiTNode * p;//临时指针
    int tag;
    SNode sdata;
    p=Tree;
    while (p||top!=-1) {
        while (p) {
            //为该结点入栈做准备
            sdata.p=p;
            sdata.tag=0;//由于遍历是左孩子，设置标志位为0
            postpush(a, sdata);//压栈
            p=p->lchild;//以该结点为根结点，遍历左孩子
        }
        sdata=a[top];//取栈顶元素
        pop();//栈顶元素弹栈
        p=sdata.p;
        tag=sdata.tag;
        //如果tag==0，说明该结点还没有遍历它的右孩子
        if (tag==0) {
            sdata.p=p;
            sdata.tag=1;
            postpush(a, sdata);//更改该结点的标志位，重新压栈
            p=p->rchild;//以该结点的右孩子为根结点，重复循环
```

```
        }
        //如果取出来的栈顶元素的tag==1，说明此结点左右子树都遍历完了，可以调用操作
函数了
        else{
            displayElem(p);
            p=NULL;
        }
    }
}
int main(){
    BiTree Tree;
    CreateBiTree(&Tree);
    printf("后序遍历: \n");
    PostOrderTraverse(Tree);
}
```