阅读：5,571　　作者：解学武

二叉树前序遍历、中序遍历和后序遍历及C语言非递归实现

递归算法底层的实现使用的是栈存储结构，所以可以直接使用栈写出相应的非递归算法。

先序遍历的非递归算法

从树的根结点出发，遍历左孩子的同时，先将每个结点的右孩子压栈。当遇到结点没有左孩子的时候，取栈顶的右孩子。重复以上过程。

实现代码函数：

```
01.  //先序遍历非递归算法
02.  void PreOrderTraverse(BiTree Tree){
03.      BiTNode* a[20];//定义一个顺序栈
04.      BiTNode * p;//临时指针
05.      push(a, Tree);//根结点进栈
06.      while (top!=-1) {
07.          p=getTop(a);//取栈顶元素
08.          pop();//弹栈
09.          while (p) {
10.              displayElem(p);//调用结点的操作函数
11.              //如果该结点有右孩子，右孩子进栈
12.              if (p->rchild) {
13.                  push(a,p->rchild);
14.              }
15.              p=p->lchild;//一直指向根结点最后一个左孩子
16.          }
17.      }
18.  }
```

中序遍历的非递归算法

从根结点开始，遍历左孩子同时压栈，当遍历结束，说明当前遍历的结点没有左孩子，从栈中取出来调用操作函数，然后访问该结点的右孩子，继续以上重复性的操作。

实现代码函数：

```
01.  //中序遍历非递归算法
02.  void InOrderTraverse1(BiTree Tree){
03.      BiTNode* a[20];//定义一个顺序栈
```

```
    BiTNode * p;//临时指针
    push(a, Tree);//根结点进栈
    while (top!=-1) {//top!=-1说明栈内不为空，程序继续运行
        while ((p=getTop(a)) &&p){//取栈顶元素，且不能为NULL
            push(a, p->lchild);//将该结点的左孩子进栈，如果没有左孩子，NULL进栈
        }
        pop();//跳出循环，栈顶元素肯定为NULL，将NULL弹栈
        if (top!=-1) {
            p=getTop(a);//取栈顶元素
            pop();//栈顶元素弹栈
            displayElem(p);
            push(a, p->rchild);//将p指向的结点的右孩子进栈
        }
    }
}
```

补：中序遍历非递归算法的另一种实现

中序遍历过程中，只需将每个结点的左子树压栈即可，右子树不需要压栈。当结点的左子树遍历完成后，只需要以栈顶结点的右孩子为根结点，继续循环遍历即可。

实现代码：

```
void InOrderTraverse2(BiTree Tree){
    BiTNode* a[20];//定义一个顺序栈
    BiTNode * p;//临时指针
    p=Tree;
    //当p为NULL或者栈为空时，表明树遍历完成
    while (p || top!=-1) {
        //如果p不为NULL，将其压栈并遍历其左子树
        if (p) {
            push(a, p);
            p=p->lchild;
        }
        //如果p==NULL，表明左子树遍历完成，需要遍历上一层结点的右子树
        else{
            p=getTop(a);
            pop();
            displayElem(p);
            p=p->rchild;
        }
    }
}
```

后序遍历的非递归算法

后序遍历是在遍历完当前结点的左右孩子之后，才调用操作函数，所以需要在操作结点进栈时，为每个结点配备一个标志位。当遍历该结点的左孩子时，设置当前结点的标志位为 0，进栈；当要遍历该结点的右孩子时，设置当前结点的标志位为 1，进栈。

这样，当遍历完成，该结点弹栈时，查看该结点的标志位的值：如果是 0，表示该结点的右孩子还没有遍历；反之如果是 1，说明该结点的左右孩子都遍历完成，可以调用操作函数。

实现代码函数：

```
01.  //后序遍历函数
02.  void PostOrderTraverse(BiTree Tree){
03.      SNode a[20];//定义一个顺序栈
04.      BiTNode * p;//临时指针
05.      int tag;
06.      SNode sdata;
07.      p=Tree;
08.      while (p||top!=-1) {
09.          while (p) {
10.              //为该结点入栈做准备
11.              sdata.p=p;
12.              sdata.tag=0;//由于遍历是左孩子，设置标志位为0
13.              postpush(a, sdata);//压栈
14.              p=p->lchild;//以该结点为根结点，遍历左孩子
15.          }
16.          sdata=a[top];//取栈顶元素
17.          pop();//栈顶元素弹栈
18.          p=sdata.p;
19.          tag=sdata.tag;
20.          //如果tag==0，说明该结点还没有遍历它的右孩子
21.          if (tag==0) {
22.              sdata.p=p;
23.              sdata.tag=1;
24.              postpush(a, sdata);//更改该结点的标志位，重新压栈
25.              p=p->rchild;//以该结点的右孩子为根结点，重复循环
26.          }
27.          //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
28.          else{
29.              displayElem(p);
30.              p=NULL;
31.          }
32.      }
33.  }
```

非递归算法的完整实现

```c
#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;
}
//前序和中序遍历使用的进栈函数
void push(BiTNode** a,BiTNode* elem){
    a[++top]=elem;
}
//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
```

```
47. void displayElem(BiTNode* elem){
48.     printf("%d ",elem->data);
49. }
50. //拿到栈顶元素
51. BiTNode* getTop(BiTNode**a){
52.     return a[top];
53. }
54. //先序遍历非递归算法
55. void PreOrderTraverse(BiTree Tree){
56.     BiTNode* a[20];//定义一个顺序栈
57.     BiTNode * p;//临时指针
58.     push(a, Tree);//根结点进栈
59.     while (top!=-1) {
60.         p=getTop(a);//取栈顶元素
61.         pop();//弹栈
62.         while (p) {
63.             displayElem(p);//调用结点的操作函数
64.             //如果该结点有右孩子，右孩子进栈
65.             if (p->rchild) {
66.                 push(a,p->rchild);
67.             }
68.             p=p->lchild;//一直指向根结点最后一个左孩子
69.         }
70.     }
71. }
72. //中序遍历非递归算法
73. void InOrderTraverse1(BiTree Tree){
74.     BiTNode* a[20];//定义一个顺序栈
75.     BiTNode * p;//临时指针
76.     push(a, Tree);//根结点进栈
77.     while (top!=-1) {//top!=-1说明栈内不为空，程序继续运行
78.         while ((p=getTop(a)) &&p){//取栈顶元素，且不能为NULL
79.             push(a, p->lchild);//将该结点的左孩子进栈，如果没有左孩子，NULL进栈
80.         }
81.         pop();//跳出循环，栈顶元素肯定为NULL，将NULL弹栈
82.         if (top!=-1) {
83.             p=getTop(a);//取栈顶元素
84.             pop();//栈顶元素弹栈
85.             displayElem(p);
86.             push(a, p->rchild);//将p指向的结点的右孩子进栈
87.         }
88.     }
89. }
90. //中序遍历实现的另一种方法
91. void InOrderTraverse2(BiTree Tree){
92.     BiTNode* a[20];//定义一个顺序栈
93.     BiTNode * p;//临时指针
```

```
94.      p=Tree;
95.      //当p为NULL或者栈为空时，表明树遍历完成
96.      while (p || top!=-1) {
97.          //如果p不为NULL，将其压栈并遍历其左子树
98.          if (p) {
99.              push(a, p);
100.             p=p->lchild;
101.         }
102.         //如果p==NULL，表明左子树遍历完成，需要遍历上一层结点的右子树
103.         else{
104.             p=getTop(a);
105.             pop();
106.             displayElem(p);
107.             p=p->rchild;
108.         }
109.     }
110. }
111. //后序遍历非递归算法
112. typedef struct SNode{
113.     BiTree p;
114.     int tag;
115. }SNode;
116. //后序遍历使用的进栈函数
117. void postpush(SNode *a,SNode sdata){
118.     a[++top]=sdata;
119. }
120. //后序遍历函数
121. void PostOrderTraverse(BiTree Tree){
122.     SNode a[20];//定义一个顺序栈
123.     BiTNode * p;//临时指针
124.     int tag;
125.     SNode sdata;
126.     p=Tree;
127.     while (p||top!=-1) {
128.         while (p) {
129.             //为该结点入栈做准备
130.             sdata.p=p;
131.             sdata.tag=0;//由于遍历是左孩子，设置标志位为0
132.             postpush(a, sdata);//压栈
133.             p=p->lchild;//以该结点为根结点，遍历左孩子
134.         }
135.         sdata=a[top];//取栈顶元素
136.         pop();//栈顶元素弹栈
137.         p=sdata.p;
138.         tag=sdata.tag;
139.         //如果tag==0，说明该结点还没有遍历它的右孩子
140.         if (tag==0) {
```

```
141.                 sdata.p=p;
142.                 sdata.tag=1;
143.                 postpush(a, sdata);//更改该结点的标志位，重新压栈
144.                 p=p->rchild;//以该结点的右孩子为根结点，重复循环
145.             }
146.             //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
147.             else{
148.                 displayElem(p);
149.                 p=NULL;
150.             }
151.         }
152.     }
153.     int main(){
154.         BiTree Tree;
155.         CreateBiTree(&Tree);
156.         printf("前序遍历: \n");
157.         PreOrderTraverse(Tree);
158.         printf("\n中序遍历算法1: \n");
159.         InOrderTraverse1(Tree);
160.         printf("\n中序遍历算法2: \n");
161.         InOrderTraverse2(Tree);
162.         printf("\n后序遍历: \n");
163.         PostOrderTraverse(Tree);
164.     }
```

运行结果

```
前序遍历:
1 2 4 5 3 6 7
中序遍历算法1:
4 2 5 1 6 3 7
中序遍历算法2:
4 2 5 1 6 3 7
后序遍历:
4 5 2 6 7 3 1
```