# 如何调用堆栈实现后序遍历的非递归程序？

## 回答区的一种答案：

必须注意，这两种老师给的算法，并没有完全模拟递归算法中栈的运行情况。

在中序非递归算法中，pop出一个节点后访问之，然后将T指向其右子节点。但是必须说明的是，在递归算法中实际存在的那个栈中，访问了当前节点（printf生效）之后并没有弹出参数为这个节点的函数！因为需要再次调用自身（参数为T->right），而且当前函数并未结束。

那为什么这种非完全模拟的算法是正确的呢？在中序递归算法中，考虑某时刻某个节点Ti（也可以看成一棵子树），Ti没有左子树，那么按照中序遍历规则这时候应该访问Ti了。访问完成后，将Ti->right入栈。当子树Ti->right遍历完成并返回Ti这一层函数的时候，Ti这一层的函数也结尾了，那么需要pop出并将控制权交给上一层。在中序非递归算法中，Ti被访问后直接出栈，当其右子树Ti->right访问完成后，Ti本身就不在栈里面了，在这时省去了一个pop的流程。

类似的，前序非递归算法中，在访问Ti后，将其入栈。如果Ti->left为空，则弹出Ti，并将Ti->right入栈。对比递归前序算法，递归前序算法并没有在这时候弹出Ti，而是在Ti->right遍历完成后返回Ti这一层时，才弹出Ti，将控制权交给上一层。也是在这时省去了一个弹出Ti的流程。

如果想挪动一下printf就得到非递归后序遍历算法的话，那是不可能的。因为在Ti的左，右子树遍历完成后，要访问Ti的时候，这种算法下Ti已经不在栈里面了！

如果想利用堆栈来实现（后序）非递归遍历，可以模拟递归算法中的栈，Ti得留在栈中。递归算法中的栈的，存储了函数的状态，包括参数，返回地址（就是执行到哪一步了）。利用栈实现（后序）非递归遍历，栈元素需要包括当前节点（函数参数），执行状态（相当于返回地址）。

↓现在可以随意改printf的位置来实现前，中，后序遍历了↓

1. enum State {start,return\_from\_left,return\_from\_right};
2. //以两次递归调用作为划分，将函数状态划为3类
3. //还未递归调用自己，将左子树作为参数递归调用并返回后，将右子树作为参数递归调用并返回后
4. typedef struct {
5. enum State state;
6. BinTree T;
7. } StackElem;
8. //省略栈的实现
10. //后序遍历非递归
11. void PostOrderTraversal (BinTree BT)
12. {
13. //item为当前执行的“函数”信息，栈中都是没有执行完的
14. StackElem item={start,BT};
15. Stack S=CreatStack(MaxSize);
16. while (true){
17. //退出条件：需要返回上层时栈为空
18. if (item.T){
19. if (item.state==start){
20. //当前入栈并递归左子树
21. Push(S,item);
22. item.T=item.T->left;
23. item.state=start;
24. }
25. else if (item.state==return\_from\_left){
26. //当前入栈并递归右子树
27. Push(S,item);
28. item.T=item.T->right;
29. item.state=start;
30. }
31. else{
32. //item.state==return\_from\_right
33. printf("%d",item.T->Data);
35. //返回上一层
36. if (!IsEmpty(S)){
37. item=Pop(S);
38. item.state++;
39. }
40. else{
41. //退出
42. break;
43. }
44. }
45. }
46. else{
47. if (!IsEmpty(S)){
48. item=Pop(S);
49. item.state++;
50. else{
51. //退出
52. break;
53. }
54. }
56. }
57. }