二叉树非递归遍历（简易版）

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MaxSize 100

typedef struct TreeNode
{
    char data;
    struct TreeNode *lchild,*rchild;
} TreeNode, *BiTree;

typedef struct Stack
{
    BiTree data[MaxSize];
    int top;
}Stack;

void InitStack(Stack *S);
bool isEmpty(Stack *S);
void CreateBiTree(BiTree *T);

void PreOrderTraverse(BiTree T,Stack *S);
void InOrderTraverse(BiTree T,Stack *S);
void PostOrderTraverse(BiTree T,Stack *S);

/*初始化栈*/
void InitStack(Stack *S)
{
    S->top = -1;
}
/*判断栈是否为空*/
bool isEmpty(Stack *S)
{
    if(S->top == -1)return true;
    else return false;
}
/*创建二叉树*/
void CreateBiTree(BiTree *T)
{
    char ch;
    scanf("%c",&ch);

    if(ch == '#')
    {
        *T = NULL;
        return;
    }
```

```
    else
    {
        *T = (BiTree)malloc(sizeof(TreeNode));
        (*T)->data = ch;
        CreateBiTree(&(*T)->lchild);
        CreateBiTree(&(*T)->rchild);
    }
}

/*前序遍历*/
void PreOrderTraverse(BiTree T,Stack *S)
{
    InitStack(S);
    BiTree p = T;

    printf("先序遍历:");
    while (p || !isEmpty(S))//遍历结束且栈为空时结束
    {
        if(p)
        {
            printf("%c ",p->data);//输出根节点
            S->data[++S->top] = p;//记录父节点
            p = p->lchild;//遍历左节点
        }
        else
        {
            p = S->data[S->top--];//寻找根节点
            p = p->rchild;//遍历右节点
        }
    }
}

/*中序遍历*/
void InOrderTraverse(BiTree T,Stack *S)
{
    InitStack(S);
    BiTree p = T;

    printf("中序遍历:");
    while (p || !isEmpty(S))
    {
        if(p)
        {
            S->data[++S->top] = p;
            p = p->lchild;
        }
        else
        {
            p = S->data[S->top--];//获取根节点
            printf("%c ",p->data);
            p = p->rchild;
        }
    }
}
```

```
/*后序遍历*/
void PostOrderTraverse(BiTree T,Stack *S)
{
    InitStack(S);
    BiTree p = T,r = NULL;

    printf("后序遍历:");
    while (p || !isEmpty(S))
    {
        if(p)
        {
            S->data[++S->top] = p;
            p = p->lchild;
        }
        else
        {
            p = S->data[S->top];
            if(p->rchild && p->rchild != r)//要输出根节点之前应该输出的一个节点是
右节点
                p = p->rchild;                //若右节点不为空 并且 右节点 不等于
r（前一个访问的节点）
                //说右有节点没有被访问过
            else
            {
                --S->top;
                printf("%c ",p->data);
                r = p;
                p = NULL;
            }
        }
    }
}
int main(int argc, char const *argv[]) {
    BiTree T;
    Stack S;
    CreateBiTree(&T);
    PreOrderTraverse(T,&S);printf("\n");
    InOrderTraverse(T,&S);printf("\n");
    PostOrderTraverse(T,&S);printf("\n");
    return 0;
}
```

输入：

```
ABDG##H###CE#I##F##
```

输出：

```
前序遍历非递归实现: A B D G H C E I F
中序遍历非递归实现: G D H B A E I C F
后序遍历非递归实现: G H D B I E F C A
```