```c
/*************************************************************************
  > File Name: binary_tree.c
  > Author: Sakura7301
  > Email: sakuraduck@foxmail.com
  > Github: https://github.com/Sakura7301
  > Created Time: 2021年07月28日 星期三 16时39分56秒
 ************************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>

typedef struct treeNode{
    int val;
    struct treeNode *left;
    struct treeNode *right;
}Tree,*LPTree;

LPTree createNode(int data)
{
    LPTree newNode=malloc(sizeof(Tree));
    newNode->val=data;
    newNode->left=NULL;
    newNode->right=NULL;
    return newNode;
}

void insertNode(LPTree parentNode,LPTree LchildNode,LPTree RchildNode)
{
    parentNode->left=LchildNode;
    parentNode->right=RchildNode;
}


//打印函数
void treePrintf(LPTree node)
{
    printf("%d\t",node->val);
}



//先序遍历:递归
void preOrder(LPTree root)
{
    if(NULL!=root)
    {
        treePrintf(root);
        preOrder(root->left);
        preOrder(root->right);
    }
}
//先序遍历：非递归
void preOrderStack(LPTree root)
{
    if(NULL!=root)
    {
        //准备栈，使用数组模拟;
```

```
        struct treeNode *stack[10];
        int top=-1;
        LPTree temp=root;
        while(temp||top!=-1)
        {
            while(temp)
            {
                treePrintf(temp);
                stack[++top]=temp;
                temp=temp->left;
            }
            if(top!=-1)//top!=-1代表栈中存有数据
            {
                temp=stack[top];
                top--;
                temp=temp->right;
            }
        }
    }
}



//中序遍历：递归
void midOrder(LPTree root)
{
    if(NULL!=root)
    {
        midOrder(root->left);
        treePrintf(root);
        midOrder(root->right);
    }
}
//中序遍历：非递归
void midOrderStack(LPTree root)
{
    if(NULL!=root)
    {
        //准备栈，使用数组模拟;
        struct treeNode *stack[10];
        int top=-1;
        LPTree temp=root;
        while(temp||top!=-1)
        {
            while(temp)
            {
                stack[++top]=temp;
                temp=temp->left;//一路访问左子树
            }
            if(top!=-1)//top!=-1代表栈中存有数据
            {
                temp=stack[top--];
                treePrintf(temp);
                temp=temp->right;
            }
        }
    }
}


//后序遍历：递归
```

```
void lastOrder(LPTree root)
{
    if(NULL!=root)
    {
        lastOrder(root->left);
        lastOrder(root->right);
        treePrintf(root);
    }
}
//后序遍历：非递归
void lastOrderStack(LPTree root)
{
    if(NULL!=root)
    {
        //准备栈，使用数组模拟
        struct treeNode *stack[10];
        int top=-1;
        LPTree temp=root;
        struct treeNode *flag=NULL;
        while(temp)
        {
            stack[++top]=temp;//入栈
            temp=temp->left;//一路访问left，直到left==NULL
        }
        while(top!=-1)//top!=-1代表栈中存有数据
        {
            temp=stack[top--];//出栈
            if(temp->right==NULL||temp->right==flag)
            {
                treePrintf(temp);//打印数据
                flag=temp;//标记已打印的数据
            }
            else
            {
                stack[++top]=temp;//没有访问过，那么先入栈
                temp=temp->right;//访问它的right
                while(temp)
                {
                    stack[++top]=temp;
                    temp=temp->left;
                }
            }
        }
    }
}

void test01()
{

    //创建结点
    LPTree A=createNode(1);
    LPTree B=createNode(2);
    LPTree C=createNode(7);
    LPTree D=createNode(3);
    LPTree E=createNode(5);
    LPTree F=createNode(8);
    LPTree G=createNode(9);
    LPTree H=createNode(4);
    LPTree I=createNode(6);
    //建立联系
    insertNode(A,B,C);
```

```
    insertNode(B,D,E);
    insertNode(C,F,G);
    insertNode(D,H,NULL);
    insertNode(E,NULL,I);

    //遍历
    printf("根节点为A的二叉树遍历：\n递归：\n");
    printf("先序遍历:");
    preOrder(A);
    printf("\n中序遍历:");
    midOrder(A);
    printf("\n后序遍历:");
    lastOrder(A);
    printf("\n");
    printf("非递归：\n");
    printf("先序遍历:");
    preOrderStack(A);
    printf("\n");
    printf("中序遍历:");
    midOrderStack(A);
    printf("\n");
    printf("后序遍历:");
    lastOrderStack(A);
    printf("\n");
}
int main()
{
    test01();
    return 0;

}
```