课程名称 ：数据结构与算法

课程类型 ：必修

实验项目名称：树形结构及其应用

实验题目：树型结构的建立与遍历

班级：1503102

学号：1150310226

姓名：尹治博

—.实验目的

掌握二叉树的存储结构，各种遍历函数，掌握树形结构的基本运算，加深对二叉树的理解。

二.实验要求及其实验环境

1．至少采用两种方法，编写建立二叉树的二叉链表存储结构的程序，并以适当的形式显示并保存二叉树；   
2．采用二叉树的二叉链表存储结构，编写程序实现二叉树的先序、中序和后序遍历的递归和非递归算法以及层序遍历算法，以适当的形式显示并保存二叉树和相应的遍历序列；   
3．在二叉树的二叉链表存储结构基础上，编写程序实现二叉树的中序线索链表存储结构建立的算法，以适当的形式显示并保存二叉树的相应的线索链表；   
4．在二叉树的线索链表存储结构上，编写程序分别实现求二叉树一个结点的先序、中序和后序遍历的后继结点算法；  
5．以上条要求为基础，编写程序实现对中序线索二叉树进行先序、中序和后序遍历的非递归算法，以适当的形式显示并保存二叉树和相应的遍历序列。

1. 设计思想

通过先序，中序建立二叉树存储结构。

通过中序线索存储结构寻找后序遍历的后继节点的思想，对给定的节点，寻找右孩子，直到它没有右孩子，然后它的后继为中序线索二叉树的中序节点，如果这个节点的左孩子是给定的节点，则该节点为别人的左儿子

，他的后序后继就是目前节点的最左的儿子，若没有右儿子，继续找左儿子。否则，给定的节点就是别人的右儿子，循环右儿子直到节点为给定的节点，则上一个节点就是后继。

先序存储结构函数-》先序遍历函数-》中序遍历函数-》后序遍历函数-》先序遍历函数非递归-》中序遍历函数非递归-》后序遍历函数非递归-》中序线索二叉树存储-》先序的后继节点-》中序的后继节点-》后序的后继节点-》通过后继节点先序遍历-》通过后继节点中序遍历-》通过后继节点后序遍历。

四．测试结果

实验数据：a b d h # # i # # e # j # # c f k # # g # #

结果：先序a b d h i e j c g k g

中序h d i b e j a k f c g

后序h i d j e b k f g c a

g的后序遍历的后继节点为c

1. 系统不足与经验体会

代码不够简洁，思路不够清晰，算法较为复杂。

六．

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

char a;

node \*left;

node \*right;

int lflag;

int rflag;

}node,\*Node;

typedef struct {

Node pp;

int pl;

int pri;

}ss;

typedef struct{

Node pr;

int flag;

}Stack;

void join\_in(Node &T) //先序存储结构

{

char data;

char ii;

scanf("%c",&data);

scanf("%c",&ii);

if(data=='#')

{

T=NULL;

}

else

{

T=(node \*)malloc(sizeof(node));

T->a=data;

join\_in(T->left);

join\_in(T->right);

}

}

void zcunchu(Node &T)

{

char data;

char ii;

scanf("%c",&data);

scanf("%c",&ii);

if(data=='#')

{

T=NULL;

}

else

{

T=(node \*)malloc(sizeof(node));

zcunchu(T->left);

T->a=data;

zcunchu(T->right);

}

}

void CXB(Node T) //层序遍历

{

Stack q[100];

int top=-1;

int i=0,ti=0;

printf("%c ",T->a);

while(T!=NULL || ti!=-1)

{

if(T->left!=NULL)

{

printf("%c ",T->left->a);

top++;

ti++;

q[top].pr=T->left;

}

else

printf("# ");

if(T->right!=NULL)

{

printf("%c ",T->right->a);

top++;

ti++;

q[top].pr=T->right;

}

else

printf("# ");

T=q[i].pr;

ti--;

i++;

}

}

void DLR(Node T) // 先序递归

{

printf("%c ",T->a);

if(T->left!=NULL)

{

DLR(T->left);

}

if(T->right!=NULL)

{

DLR(T->right);

}

}

void DLR\_\_(Node T) //先序遍历 非递归

{

Stack o[100];

int top=-1;

while(T!=NULL || top!=-1)

{

while(T!=NULL)

{

printf("%c ",T->a);

top++;

o[top].pr=T;

T=T->left;

}

if(top!=-1)

{

T=o[top].pr;

top--;

T=T->right;

}

}

}

int LDR\_\_(Node T,ss p[]) // 中序遍历 非递归

{

Stack i[100];

int top1=-1;

int j=-1;

while(T!=NULL || top1!=-1)

{

while(T!=NULL)

{

top1++;

i[top1].pr=T;

T=T->left;

}

if(top1!=-1)

{

T=i[top1].pr;

top1--;

//printf("%c ",T->a); 是否打印

j++;

p[j].pp=T;

if(T->left==NULL)

{

p[j].pl=0;

T->lflag=0;

}

else

{

p[j].pl=1;

T->lflag=1;

}

if(T->right==NULL)

{

T->rflag=0;

p[j].pri=0;

}

else

{

p[j].pri=1;

T->rflag=1;

}

T=T->right;

}

}

return j;

}

void LRD\_\_(Node T) //后序遍历 非递归

{

Stack p[100];

int top2=-1;

while(T!=NULL || top2!=-1)

{

while(T!=NULL)

{

top2++;

p[top2].pr=T;

p[top2].flag=1;

T=T->left;

}

while(top2!=-1 && p[top2].flag==2)

{

T=p[top2].pr;

top2--;

printf("%c ",T->a);

}

if(top2!=-1)

{

p[top2].flag=2;

T=p[top2].pr->right;

}

if(top2==-1)

{

break;

}

}

}

void LDR(Node T) //中序递归

{

if(T->left!=NULL)

{

LDR(T->left);

}

printf("%c ",T->a);

if(T->right!=NULL)

{

LDR(T->right);

}

}

void LRD(Node T) //后序遍历 递归

{

if(T->left!=NULL)

{

LRD(T->left);

}

if(T->right!=NULL)

{

LRD(T->right);

}

printf("%c ",T->a);

}

void nimei(node \*head,ss na[],int j)

{

int i=0;

node \*p;

while(i<=j)

{

if(na[i].pl==0)

{

if(i!=0)

na[i].pp->left=na[i-1].pp;

else

na[i].pp->left=head;

}

if(na[i].pri==0)

{

if(i!=j)

na[i].pp->right=na[i+1].pp;

else

na[i].pp->right=head;

}

i++;

}

}

node \*yyy(node \*p) //中序线索 中序 后一个节点

{

if(p->rflag==0)

{

return p->right;

}

else

{

p=p->right;

while(p->lflag==1)

{

p=p->left;

}

return p;

}

}

int aj=0;

node \*pop;

int zhongxu(node \*head1,node \*T) //中序线索 中序下一个节点

{

while(aj==0 && T->lflag==1)

{

T=T->left;

}

aj++;

printf("%c ",T->a);

pop=yyy(T);

if(pop!=head1)

zhongxu(head1,pop);

else

return 0;

}

node \*xxx(node \*head1,node \*p) //中序线索 先序下一个节点

{

if(p->lflag!=0)

{

return p->left;

}

else if(p->rflag!=0)

{

return p->right;

}

else

{

if(p->right==head1)

{

return p->right;

}

while(p->right->rflag==0)

{

p=p->right;

}

return p->right->right;

}

}

node \*pip;

int xianxu(node \*head1,node \*T) //中序线索 先序遍历非递归

{

pip=xxx(head1,T);

if(pip!=head1)

printf("%c ",pip->a);

else

return 0;

xianxu(head1,pip);

}

node \*zzz(node \*head1,node \*p)

{

int q=0;

Stack iii[100];

int top2=-1;

node \*T;

T=head1->left;

while(T!=NULL || top2!=-1)

{

while(T!=NULL)

{

top2++;

iii[top2].pr=T;

iii[top2].flag=1;

if(T->lflag==1)

T=T->left;

else

T=NULL;

}

while(top2!=-1 && iii[top2].flag==2)

{

T=iii[top2].pr;

top2--;

if(q==1)

return T;

if(T->a==p->a)

q=1;

}

if(top2!=-1)

{

iii[top2].flag=2;

T=iii[top2].pr;

if(T->rflag==0)

T=NULL;

else

T=T->right;

}

if(top2==-1)

{

return head1;

}

}

}

int pup=0;

int houxu(node \*head1,node \*T)

{

if(pup==0)

{

while(T->lflag==1)

{

T=T->left;

}

if(T->rflag==1)

T=T->right;

}

pup++;

printf("%c ",T->a);

pip=zzz(head1,T);

if(pip!=head1)

houxu(head1,pip);

else

return 0;

}

node \*nima(node \*head1,node \*T)

{

node \*ptr;

ptr=T;

int a=1;

if(T==head1->left)

return head1;

while(T->rflag==1)

{

T=T->right;

}

T=T->right;

if(T->left==ptr)

{

if(T->rflag==0)

{

printf("%c ",T->a);

return 0;

}

while(a==1)

{

T=T->right;

while(T->lflag==1)

{

T=T->left;

}

if(T->rflag==0)

return T;

}

}

else

{

T=T->left;

while(T->right!=ptr)

{

T=T->right;

}

return T;

}

}

int main()

{

Node head;

node\* head1;

int tio;

ss na[100];

node \*pr;

node \*po;

join\_in(head);

LDR(head);

LRD(head);

DLR(head);

head1=(node \*)malloc(sizeof(node));

head1->a='#';

head1->left=head;

head1->right=head; //中序线索存储结构

head1->rflag=1;

head1->lflag=1;

tio=LDR\_\_(head,na);

nimei(head1,na,tio); //中序线索存储结构

pr=nima(head1,head1->left->right->right); //中序线索 后序后一个节点

printf("\n%c\n",pr->a);

//xxx(head1,head1->left); //中序线索 先序后一个节点

//yyy(head1,head1->left); //中序线索 中序后一个节点

//zzz(head1,head1->left); //中序线索 后序后一个节点

//houxu(head1,head1); //中序线索 后序遍历非递归

//xianxu(head1,head1); //中序线索 先序遍历非递归

//zhongxu(head1,head1); //中序线索 中序遍历非递归

//join\_in(head);

//LRD\_\_(head); //后序遍历非递归算法

//LRD(head); //后序遍历递归算法

// printf("\n");

// LDR(head); //中序遍历递归算法

// LDR\_\_(head); //中序遍历非递归算法

// printf("\n");

// DLR\_\_(head); //先序遍历非递归算法

// DLR(head); //先序遍历递归算法；

// printf("\n");

// CXB(head); //层序遍历

return 0;

}