**实验四、二叉树的基本操作**

**一、实验目的**

1、要求利用二叉链表方法建立二叉树。   
2、掌握二叉链表的基本操作的算法实现。

3、通过二叉链表的基本操作的分析，进一步熟悉C语言函数的基本结构，掌握程序中的头文件、实现文件和主文件之间的相互关系及各自的作用。

4、进一步掌握C程序的结构和自顶向下、逐步求精的结构化程序设计方法，熟悉C程序的输入、编辑、调试和运行的全过程。

**二、实验准备工作**

在开始实验前，请认真回顾课程相关的内容，请查阅并复习有关C语言结构体变量、数组和指针的定义和操作的相关内容。要求每位同学的计算机安装MicrosoftVisualC++6.0开发环境。

**三、实验内容与步骤**

1．利用二叉链表建立二叉树，实现二叉树的先、中、后序三种遍历算法等操作。   
编写算法实现如下功能：   
（1）根据先序遍历建立二叉树，要求利用后序遍历输出对应的二叉树元素序列，观察输入的内容与输出的内容是否符合对应算法的要求。

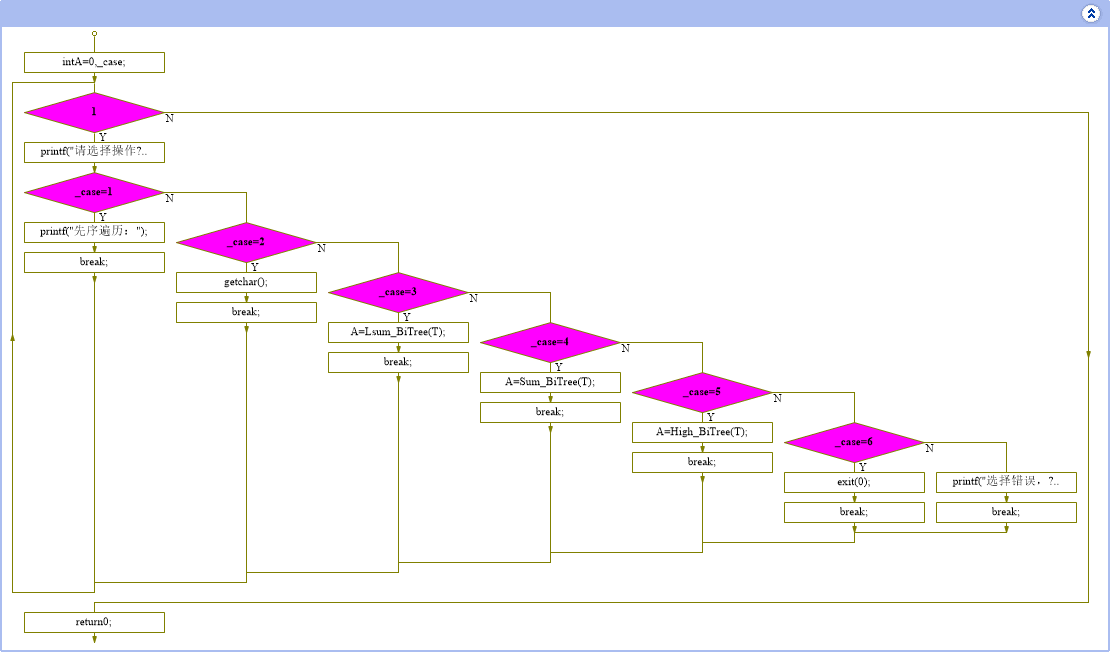
（2）根据已创建的二叉树，要求在二叉树中查找指定元素x，并此基础上进行插入和删除基本操作。  
（3）计算对应二叉树的结点个数、叶子结点个数、高度。

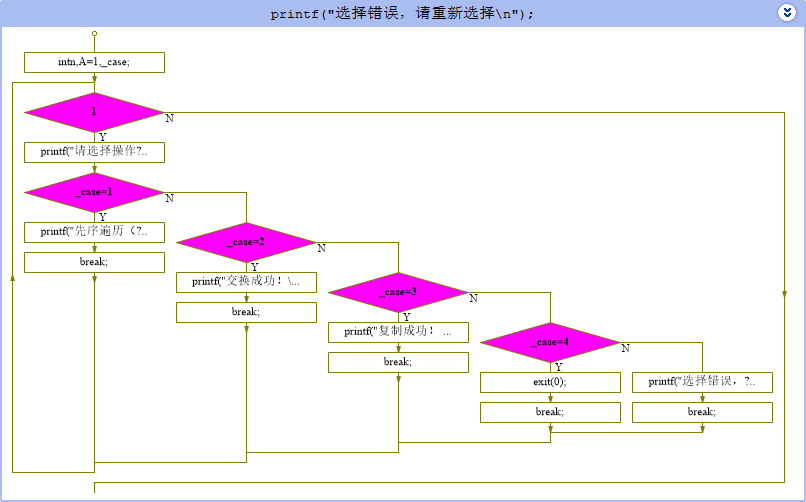
2．假设有n个结点的完全二叉树使用顺序存储存放在一维数组A[1……n]中，以此序列为建立该二叉树的二叉链表表示；   
（1）要求编写出非递归先序遍历的算法，输出树对应的结点。

（2）编写一非递归算法，交换二叉树的左右子树；   
（3）要求编写复制一棵二叉树算法。

（4）将创建的二叉树进行线索化。

1. **算法步骤**





**五、实验内容代码及运行结果截图**

#ifndef \_\_BITREE\_H

#define \_\_BITREE\_H

#include <stdio.h>

#include <stdlib.h>

typedef struct BiNode

{

char data;

struct BiNode \*lchild, \*rchild;

}BiNode, \*BiTree;

typedef struct

{

BiTree data[1024];

int top;

}SeqStack,\*Stack;

BiTree Creat\_BiTree(); //创建二叉树

void In\_order(BiTree T); //先序遍历

void Pre\_order(BiTree T); //中序遍历

void Post\_order(BiTree T); //后序遍历

int Sreach(BiTree T, char X,Stack S); //寻找X

void Action (BiTree T, char X, Stack S); //指定操作

void InsertL(BiTree T, char x,Stack S); //插入元素

void DeleteL(BiTree T); //删除X的子树

void Delete(BiTree T);

int Lsum\_BiTree(BiTree T); //二叉树叶子数

int Sum\_BiTree(BiTree T); //二叉树结点数

int High\_BiTree(BiTree T); //二叉树高度

#endif

#include "BiTree.h"

//创建二叉树

BiTree Creat\_BiTree()

{

BiTree T;

char ch;

scanf\_s("%c", &ch,1);

if (ch == '0')

T = NULL;

else

{

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

if (T)

{

T->data = ch;

T->lchild=Creat\_BiTree();

T->rchild=Creat\_BiTree();

}

}

return T;

}

//先序遍历

void Pre\_order(BiTree T)

{

if (T)

{

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

Pre\_order(T->lchild);

Pre\_order(T->rchild);

}

}

//中序遍历

void In\_order(BiTree T)

{

if (T)

{

In\_order(T->lchild);

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

In\_order(T->rchild);

}

}

//后序遍历

void Post\_order(BiTree T)

{

if (T)

{

Post\_order(T->lchild);

Post\_order(T->rchild);

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

}

}

//查找元素X

int Sreach(BiTree T, char X, Stack S)

{

if (T)

{

if (T->data == X)

{

S->top++;

S->data[S->top] = T;

}

Sreach(T->lchild, X, S);

Sreach(T->rchild, X, S);

}

}

//指定操作

void Action(BiTree T, char B, Stack S)

{

int \_case;

if (S->top != -1)

{

if (S->top > 0)

{

printf("共%d个%c元素，选择一个元素（填序号）：", S->top + 1, B);

scanf\_s("%d", &S->top, 1);

S->top = S->top - 1;

}

printf("选择一个操作：1、插入 2、删除 （其他数字不操作）\n");

scanf\_s("%d", &\_case);

getchar();

switch (\_case)

{

case 1:

printf("输入插入的元素：");

scanf\_s("%c", &B, 1);

InsertL(T, B, S);

printf("插入成功!");

break;

case 2:

DeleteL(S->data[S->top]);

printf("删除成功!");

break;

}

}

else printf("该元素不存在\n");

}

//插入元素

void InsertL(BiTree T,char x, Stack S)

{

BiTree p;

if (p = (BiTree)malloc(sizeof(BiNode)))

{

p->data = x;

p->lchild = S->data[S->top]->lchild;

p->rchild = S->data[S->top]->rchild;

S->data[S->top]->lchild = p;

S->data[S->top]->rchild = NULL;

}

}

//删除X元素的子树

void DeleteL(BiTree T)

{

Delete(T->lchild);

Delete(T->rchild);

T->lchild = NULL;

T->rchild = NULL;

}

void Delete(BiTree T)

{

if (T)

{

Delete(T->lchild);

Delete(T->rchild);

}

free(T);

}

//二叉树叶子数

int Lsum\_BiTree(BiTree T)

{

if (T->lchild == NULL && T->rchild == NULL)

return 1;

return Lsum\_BiTree(T->lchild) + Lsum\_BiTree(T->rchild);

}

//二叉树结点数

int Sum\_BiTree(BiTree T)

{

if (T)

return Sum\_BiTree(T->lchild) + Sum\_BiTree(T->rchild)+1;

else

return 0;

}

//二叉树高度

int High\_BiTree(BiTree T)

{

int treeHeight = 0;

if (T != NULL)

{

int leftHeight = High\_BiTree(T->lchild);

int rightHeight = High\_BiTree(T->rchild);

treeHeight = leftHeight >= rightHeight ? leftHeight + 1 : rightHeight + 1;

}

return treeHeight;

}

#include "BiTree.h"

int main()

{

int A=0, \_case;

char B;

printf("创建链式二叉树：");

BiTree T = Creat\_BiTree();

while (1)

{

printf("请选择操作：\n1.先序、中序、后序遍历 2.查找结点X 3.叶子总数 4.结点总数 5.二叉树高度 6.退出\n");

scanf\_s("%d", &\_case);

switch (\_case)

{

case 1:

printf("先序遍历：");

Pre\_order(T);

printf("\n");

printf("中序遍历：");

In\_order(T);

printf("\n");

printf("后序遍历：");

Post\_order(T);

printf("\n");

break;

case 2:

getchar();

printf("输入X元素：");

scanf\_s("%c",&B,1);

Stack S= (Stack)malloc(sizeof(SeqStack));

S->top = -1;

Sreach(T, B, S);

Action(T, B, S);

break;

case 3:

A=Lsum\_BiTree(T);

printf("叶子总数：%d\n",A);

break;

case 4:

A = Sum\_BiTree(T);

printf("结点总数：%d\n",A);

break;

case 5:

A = High\_BiTree(T);

printf("二叉树高度:%d\n",A);

break;

case 6:

exit(0);

break;

default:

printf("选择错误，请重新选择\n");

break;

}

}

return 0;

}

#ifndef \_\_BITREE\_H

#define \_\_BITREE\_H

#include <stdio.h>

#include <stdlib.h>

typedef struct BiNode

{

char data;

struct BiNode \*lchild, \*rchild;

}BiNode, \*BiTree;

typedef struct BiThrNode

{

char data;

struct BiThrNode \*lchild, \*rchild;

unsigned ltag;

unsigned rtag;

}BiThrNodeType,\*BiThrTree;

BiTree Creat\_BiTree(char str[], int n,int A); //创建二叉树

void NRPreOrder(BiTree T); //先序遍历（非递归）

void Change\_child(BiTree T); //交换左右子树

BiThrTree Copy\_BiTree(BiTree T); //复制二叉树至T2

BiThrTree InOrderThr(BiThrTree T); //线索化二叉树T2

void InThreading(BiThrTree p);

void Pre\_orders(BiThrTree T2); //T2先序遍历（检查）

#endif

#include "BiTree.h"

BiThrNodeType \*p;

//创建二叉树

BiTree Creat\_BiTree(char str[], int n,int A)

{

BiTree T;

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

if (T)

{

T->data = str[A-1];

A = A + A;

if (A <= n)

T->lchild = Creat\_BiTree(str, n, A);

else

T->lchild = NULL;

if (A + 1 <= n)

T->rchild = Creat\_BiTree(str, n, A + 1);

else

T->rchild = NULL;

}

return T;

}

//先序遍历（非递归）

void NRpOrder(BiTree T)

{

BiTree stack[1024],p;

int top = -1;

if (T == NULL) return;

p = T;

while (!(p == NULL && top == -1))

{

while (p)

{

printf("%c",p->data);

top++;

stack[top] = p;

p = p->lchild;

}

p = stack[top];

top--;

p = p->rchild;

}

**}**

//交换左右子树

void Change\_child(BiTree T)

{

BiTree stack[1024], p,temp;

int top = -1;

if (T == NULL) return;

p = T;

while (!(p == NULL && top == -1))

{

while (p)

{

top++;

stack[top] = p;

p = p->lchild;

}

p = stack[top];

top--;

p = p->rchild;

temp = stack[top+1]->lchild;

stack[top+1]->lchild = stack[top+1]->rchild;

stack[top+1]->rchild = temp;

}

}

//复制二叉树

BiThrTree Copy\_BiTree(BiTree T)

{

BiThrTree T2=NULL;

if (T)

{

T2 = (BiThrTree)malloc(sizeof(BiThrNodeType));

if (T2)

{

T2->data = T->data;

T2->lchild= Copy\_BiTree(T->lchild);

T2->rchild= Copy\_BiTree(T->rchild);

}

}

return T2;

}

//线索化二叉树

BiThrTree InOrderThr(BiThrTree p)

{

BiThrTree head;

head = (BiThrNodeType\*)malloc(sizeof(BiThrNodeType));

if (head)

{

head->ltag = 0;

head->rtag = 1;

head->rchild = head;

if (!p)

head->lchild = head;

else

{

head->lchild = p;

p = head;

InThreading(p);

p->rchild = head;

p->rtag = 1;

head->rchild = p;

}

return head;

}

}

void InThreading(BiThrTree p)

{

if (p)

{

InThreading(p->lchild);

if (p->lchild == NULL)

{

p->ltag = 1;

p->lchild = p;

}

if (p->rchild == NULL)

{

p->rtag = 1;

p->rchild = p;

}

p = p;

InThreading(p->rchild);

}

}

//T2先序遍历（检查）

void p\_orders(BiThrTree T2)

{

if (T2)

{

printf("%c", T2->data);

p\_orders(T2->lchild);

p\_orders(T2->rchild);

}

}

void P(BiTree T)

{

BiTree temp;

if (T == NULL)

return;

else

{

Change\_child(T->lchild);

Change\_child(T->rchild);

temp = T->lchild;

T->lchild = T->rchild;

T->rchild = temp;

}

}

#include "BiTree.h"

int main(){

int n, A = 1, \_case;

printf("输入节点数n：");

scanf\_s("%d", &n);

char str[1024];

printf("输入数组：");

scanf\_s("%s", str,1024);

BiTree T = Creat\_BiTree(str, n, A);

while (1){

printf("请选择操作：\n1.先序遍历（非递归） 2.交换左右子树 3.复制并线索化T为T2 4.退出\n");

scanf\_s("%d", &\_case);

switch (\_case){

case 1:

printf("先序遍历（非递归）：");

NRPreOrder(T);

printf("\n");

break;

case 2:

printf("交换成功！\n");

Change\_child(T);

break;

case 3:

printf("复制成功！ ");

BiThrTree T2=Copy\_BiTree(T);

printf("先序遍历（检查）：");

Pre\_orders(T2);

printf(" 线索化成功！");

T2 = InOrderThr(T2);

printf("\n");

break;

case 4:

exit(0);

break;

default:

printf("选择错误，请重新选择\n");

break;

}

}

}

**六、实验结果分析**

调试过程中遇到的问题时间花费太多，debug 熬夜了；

心得与体会：心得没有什么，多多写代码，就可以了。