实验07二叉树及应用

实验周次：第14周 学时：2学时 地点：学院机房

学号：3190707121 姓名：武新纪 班级：人工智能191 序号：21

提示：请务必填写 以上个人信息。

实验题1计50分，实验题2计50分。

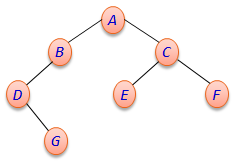
**【实验目的】**

1.领会二叉链存储结构和掌握二叉树中的各种基本运算算法设计。

2.领会二叉链的各种遍历过程以及遍历算法的递归和非递归设计。

**【实验内容】**

**实验题1**. 编写一个程序btree.cpp，实现二叉树的基本运算，并在此基础上设计一个程序exp-7-1.cpp完成以下功能。也可以将代码统一放在exp-7-1.cpp文件中。给出如下二叉树：



1）给上图二叉树创建二叉链存储结构b,该二叉树的括号表示法对应字符串为“*A*(*B*(*D*(，*G*))，*C*(*E*，*F*))。

2）输出二叉树b.

3）输出结点值为字符‘C’的结点的左右孩子结点的值。

4）输出二叉树b的高度。

5）释放二叉树b.

**程序：**

**//btree.cpp**

//二叉树的基本运算算法

#include <stdio.h>

#include <malloc.h>

#define MaxSize 100

typedef char ElemType;

typedef struct node

{

    ElemType data;          //数据元素

    struct node \*lchild;    //指向左孩子结点

    struct node \*rchild;    //指向右孩子结点

} BTNode;

void CreateBTree(BTNode \* &b,const char \*str) //创建二叉树

{

    BTNode \*St[MaxSize],\*p=NULL;

    int top=-1,k,j=0;

    char ch;

    b=NULL;             //建立的二叉树初始时为空

    ch=str[j];

    while (ch!='\0')    //str未扫描完时循环

    {

        switch(ch)

        {

        case '(':top++;St[top]=p;k=1; break;        //为左孩子结点

        case ')':top--;break;

        case ',':k=2; break;                            //为孩子结点右结点

        default:p=(BTNode \*)malloc(sizeof(BTNode));

                p->data=ch;p->lchild=p->rchild=NULL;

                if (b==NULL)                            //\*p为二叉树的根结点

                    b=p;

                else                                //已建立二叉树根结点

                {

                    switch(k)

                    {

                    case 1:St[top]->lchild=p;break;

                    case 2:St[top]->rchild=p;break;

                    }

                }

        }

        j++;

        ch=str[j];

    }

}

void DestroyBTree(BTNode \*&b)   //销毁二叉树

{   if (b!=NULL)

    {   DestroyBTree(b->lchild);

        DestroyBTree(b->rchild);

        free(b);

    }

}

BTNode \*FindNode(BTNode \*b,ElemType x) //查找值为x的结点

{

    BTNode \*p;

    if (b==NULL)

        return NULL;

    else if (b->data==x)

        return b;

    else

    {

        p=FindNode(b->lchild,x);

        if (p!=NULL)

            return p;

        else

            return FindNode(b->rchild,x);

    }

}

BTNode \*LchildNode(BTNode \*p)

{

    return p->lchild;

}

BTNode \*RchildNode(BTNode \*p)

{

    return p->rchild;

}

int BTHeight(BTNode \*b)     //求二叉树b的高度

{

    int lchildh,rchildh;

    if (b==NULL) return(0);                 //空树的高度为0

    else

    {

        lchildh=BTHeight(b->lchild);    //求左子树的高度为lchildh

        rchildh=BTHeight(b->rchild);    //求右子树的高度为rchildh

        return (lchildh>rchildh)? (lchildh+1):(rchildh+1);

    }

}

void DispBTree(BTNode \*b)  //以括号表示法输出二叉树

{

    if (b!=NULL)

    {   printf("%c",b->data);

        if (b->lchild!=NULL || b->rchild!=NULL)

        {   printf("(");                        //有孩子结点时才输出(

            DispBTree(b->lchild);               //递归处理左子树

            if (b->rchild!=NULL) printf(",");   //有右孩子结点时才输出,

            DispBTree(b->rchild);               //递归处理右子树

            printf(")");                        //有孩子结点时才输出)

        }

    }

}

**//exp7-1.cpp**

#include"btree.cpp"

int main()

{

    BTNode \*b, \*p, \*lp, \*rp;

    printf("二叉树的基本运算如下:\n");

    printf("  (1)创建二叉树\n");

    CreateBTree(b, "A(B(D(,G)),C(E,F))");

    printf("  (2)输出二叉树:");

    DispBTree(b);

    printf("\n");

    printf("  (3)C结点:");

    p = FindNode(b, 'C');

    if (p != NULL)

    {

        lp = LchildNode(p);

        if (lp != NULL)

            printf("左孩子为%c ", lp->data);

        else

            printf("无左孩子 ");

        rp = RchildNode(p);

        if (rp != NULL)

            printf("右孩子为%c", rp->data);

        else

            printf("无右孩子 ");

    }

    printf("\n");

    printf("  (4)二叉树b的高度:%d\n", BTHeight(b));

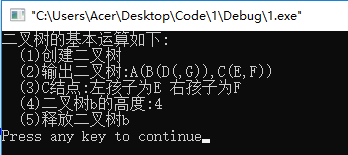
    printf("  (5)释放二叉树b\n");

    DestroyBTree(b);

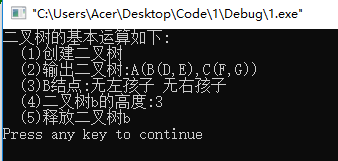
    return 0;

}

**调试结果1：**

****

**调试结果2：**

****

实验2. 编写一个程序exp-7-2.cpp,实现二叉树的先序遍历、中序遍历和后续遍历的递归和非递归算法，以及层次遍历的算法，并以实验题1创建的二叉树b为实参，在main函数中调用以上实现函数，分别输出二叉链b的所有结点的值，中间用空格或者逗号隔开。比如先序遍历序列为：A B D G C E F

**程序：**

**//exp7-2.cpp**

#include"btree.cpp"

void PreOrder(BTNode \*b) //先序遍历的递归算法

{

    if (b != NULL)

    {

        printf("%c ", b->data); //访问根结点

        PreOrder(b->lchild);    //递归访问左子树

        PreOrder(b->rchild);    //递归访问右子树

    }

}

void PreOrder1(BTNode \*b) //先序非递归遍历算法

{

    BTNode \*St[MaxSize], \*p;

    int top = -1;

    if (b != NULL)

    {

        top++; //根结点进栈

        St[top] = b;

        while (top > -1) //栈不为空时循环

        {

            p = St[top]; //退栈并访问该结点

            top--;

            printf("%c ", p->data);

            if (p->rchild != NULL) //有右孩子,将其进栈

            {

                top++;

                St[top] = p->rchild;

            }

            if (p->lchild != NULL) //有左孩子,将其进栈

            {

                top++;

                St[top] = p->lchild;

            }

        }

        printf("\n");

    }

}

void InOrder(BTNode \*b) //中序遍历的递归算法

{

    if (b != NULL)

    {

        InOrder(b->lchild);     //递归访问左子树

        printf("%c ", b->data); //访问根结点

        InOrder(b->rchild);     //递归访问右子树

    }

}

void InOrder1(BTNode \*b) //中序非递归遍历算法

{

    BTNode \*St[MaxSize], \*p;

    int top = -1;

    if (b != NULL)

    {

        p = b;

        while (top > -1 || p != NULL)

        {

            while (p != NULL) //扫描结点p的所有左下结点并进栈

            {

                top++;

                St[top] = p;

                p = p->lchild;

            }

            if (top > -1)

            {

                p = St[top]; //出栈结点p并访问

                top--;

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

                p = p->rchild;

            }

        }

        printf("\n");

    }

}

void PostOrder(BTNode \*b) //后序遍历的递归算法

{

    if (b != NULL)

    {

        PostOrder(b->lchild);   //递归访问左子树

        PostOrder(b->rchild);   //递归访问右子树

        printf("%c ", b->data); //访问根结点

    }

}

void PostOrder1(BTNode \*b) //后序非递归遍历算法

{

    BTNode \*St[MaxSize];

    BTNode \*p;

    int top = -1; //栈指针置初值

    bool flag;

    if (b != NULL)

    {

        do

        {

            while (b != NULL) //将b结点的所有左下结点进栈

            {

                top++;

                St[top] = b;

                b = b->lchild;

            }

            p = NULL;    //p指向当前结点的前一个已访问的结点

            flag = true; //flag为真表示正在处理栈顶结点

            while (top != -1 && flag)

            {

                b = St[top];        //取出当前的栈顶元素

                if (b->rchild == p) //右子树不存在或已被访问,访问之

                {

                    printf("%c ", b->data); //访问b结点

                    top--;

                    p = b; //p指向则被访问的结点

                }

                else

                {

                    b = b->rchild; //b指向右子树

                    flag = false;  //表示当前不是处理栈顶结点

                }

            }

        } while (top != -1);

        printf("\n");

    }

}

void TravLevel(BTNode \*b) //层次遍历

{

    BTNode \*Qu[MaxSize]; //定义环形队列

    int front, rear;     //定义队首和队尾指针

    front = rear = 0;    //置队列为空队

    if (b != NULL)

        printf("%c ", b->data);

    rear++; //根结点进队

    Qu[rear] = b;

    while (rear != front) //队列不为空

    {

        front = (front + 1) % MaxSize;

        b = Qu[front];         //出队结点b

        if (b->lchild != NULL) //输出左孩子,并进队

        {

            printf("%c ", b->lchild->data);

            rear = (rear + 1) % MaxSize;

            Qu[rear] = b->lchild;

        }

        if (b->rchild != NULL) //输出右孩子,并进队

        {

            printf("%c ", b->rchild->data);

            rear = (rear + 1) % MaxSize;

            Qu[rear] = b->rchild;

        }

    }

    printf("\n");

}

int main()

{

    BTNode \*b;

    CreateBTree(b, "A(B(D(,G)),C(E,F))");

    printf("二叉树b:");

    DispBTree(b);

    printf("\n");

    printf("层次遍历序列:");

    TravLevel(b);

    printf("先序遍历序列:\n");

    printf("    递归算法:");

    PreOrder(b);

    printf("\n");

    printf("  非递归算法:");

    PreOrder1(b);

    printf("中序遍历序列:\n");

    printf("    递归算法:");

    InOrder(b);

    printf("\n");

    printf("  非递归算法:");

    InOrder1(b);

    printf("后序遍历序列:\n");

    printf("    递归算法:");

    PostOrder(b);

    printf("\n");

    printf("  非递归算法:");

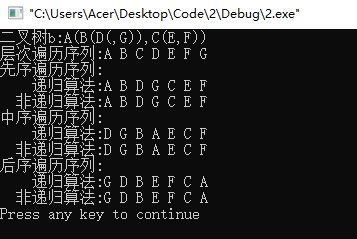
    PostOrder1(b);

    DestroyBTree(b);

    return 0;

}

**运行结果：**

****