实现二叉树的各种遍历算法

姓名：凌强 班级：21计算机二班 学号：2021413579

实验目的：

1. 掌握二叉树的逻辑结构和存储结构，学会对二叉树进行基本操作。
2. 掌握二叉树的各种遍历算法。
3. 掌握对二叉树遍历算法的实际应用，培养思维。

实验内容：

1. 初始化二叉树，接收输入的数据，生成二叉树，以链表方式存储。
2. 通过输入的数字序号选择对应的功能以完成二叉树的先序遍历、中序遍历、后序遍历以及层次遍历等操作。
3. 通过先序和中序或者中序和后序的方式构建二叉树进行实验。

# **需求分析**

1. 完成对二叉树的创建、求长度、先序遍历、中序遍历、后序遍历以及层次遍历等基本操作。
2. 演示程序中，规定二叉树元素为 char型。
3. 演示程序以用户和计算机的对话方式执行，在运行框中出现提示信息之后，由用户在键盘上输入完成相关命令的执行；相应结果显示在其后。
4. 程序执行的命令包括：
   1. 输出用户界面。
   2. 读取用户输入的数据，通过先序和中序或者中序和后序分别创建两个二叉树。
   3. 输入数字选择对应的功能。
   4. 求先序遍历
   5. 求中序遍历
   6. 求后序遍历
   7. 求层次遍历
   8. 销毁二叉树。
   9. 退出程序。
5. 测试数据：见（[六、测试结果](#_测试结果)）。

# **概要设计**

1. 二叉树的抽象数据定义：

typedef struct tree

{

    char data;

    struct tree \*lchild,\*rchild;

} DBTree;

1. 基本操作：

*//根据中序遍历和后序遍历构建二叉树*

DBTree\* CreateDBTreeByAft(char \**instr*, char \**afterstr*, int *length*)

*//根据前序遍历和中序遍历构建二叉树*

DBTree\* CreateDBTreeByPre(char \**instr*, char \**prestr*, int *length*)

*//节点个数*

int NodeCount(DBTree \**bt*)

*//先序遍历*

void PreOrder(DBTree \**bt*)

*//中序遍历*

void InOrder(DBTree \**bt*)

*//后序遍历*

void PostOrder(DBTree \* *bt*)

*//层次遍历*

void LevelOrder(DBTree \**DT*)

*//打印主界面*

void welcome()

1. 程序包含两个模块：
2. 主函数模块：

用于与测试用户交换数据。

1. 操作函数模块：

用于指引测试用户下一步操作，操作链表。

# **详细设计**

## 二叉树的定义

*#include* <iostream>

*#include* <cstring>

using namespace std;

*#define* Max 100

typedef struct tree

{

    char data;

    struct tree \*lchild,\*rchild;

} DBTree;

## 操作函数

*//根据中序遍历和后序遍历构建二叉树*

DBTree\* CreateDBTreeByAft(char \**instr*, char \**afterstr*, int *length*)

{

*if* (*length* == 0)*//都构建完毕*

    {

*return* NULL;

    }

    char c = *afterstr*[*length* - 1];*//树的根为后序遍历最后一个字符*

    int i = 0;

*while* ((*instr*[i] != c) && i < *length*)*//找到中序遍历中该根的位置*

    {

        i = i + 1;

    }

    int leftlength = i;*//确定左子树中结点的数目*

    int rightlength = *length* - i - 1;*//确定右子树中结点的数目*

    DBTree \*T;

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

    T->data = c;*//建立新结点，每次都使其等于每棵小子树的根结点*

*// printf("%c",c);//调试*

    T->lchild = NULL;

    T->rchild = NULL;

    T->lchild = CreateDBTreeByAft(&*instr*[0], &*afterstr*[0], leftlength);*//创建左子树*

    T->rchild = CreateDBTreeByAft(&*instr*[i + 1], &*afterstr*[i], rightlength);*//创建右子树*

*return* T;

}

*//根据前序遍历和中序遍历构建二叉树*

DBTree\* CreateDBTreeByPre(char \**instr*, char \**prestr*, int *length*)

{

*if* (*length* == 0)*//都构建完毕*

    {

*return* NULL;

    }

    char c = *prestr*[0];*//树的根为前序遍历第一一个字符*

    int i = 0;

*while* ((*instr*[i] != c) && i < *length*)*//找到中序遍历中该根的位置*

    {

        i = i + 1;

    }

    int leftlength = i;*//确定左子树中结点的数目*

    int rightlength = *length* - i - 1;*//确定右子树中结点的数目*

    DBTree \*T;

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

    T->data = c;*//建立新结点，每次都使其等于每棵小子树的根结点*

*// printf("%c",c);//调试*

    T->lchild = NULL;

    T->rchild = NULL;

    T->lchild = CreateDBTreeByPre(&*instr*[0], &*prestr*[1], leftlength);*//创建左子树*

    T->rchild = CreateDBTreeByPre(&*instr*[i + 1], &*prestr*[i+1], rightlength);*//创建右子树*

*return* T;

}

*//节点个数*

int NodeCount(DBTree \**bt*)

{

    int num1,num2;

*if*(*bt*==NULL)

    {

*return* 0;

    }

*else*

    {

        num1 = NodeCount(*bt*->lchild);

        num2 = NodeCount(*bt*->rchild);

*return* num1 + num2 + 1 ;

    }

}

*//先序遍历*

void PreOrder(DBTree \**bt*)

{

*if*(*bt*)

    {

        cout<<*bt*->data;

        PreOrder(*bt*->lchild);

        PreOrder(*bt*->rchild);

    }

}

*//中序遍历*

void InOrder(DBTree \**bt*)

{

*if*(*bt*)

    {

        InOrder(*bt*->lchild);

        cout<<*bt*->data;

        InOrder(*bt*->rchild);

    }

}

*//后序遍历*

void PostOrder(DBTree \* *bt*)

{

*if*(*bt*)

    {

        PostOrder(*bt*->lchild);

        PostOrder(*bt*->rchild);

        cout<<*bt*->data;

    }

}

*//层次遍历*

void LevelOrder(DBTree \**DT*)

{

    DBTree \*p;

    DBTree \* qu[Max];

    int front,rear;

    front = rear = 0;

    rear++;qu[rear]=*DT*;

*while*(front!=rear)

    {

        front = (front+1)%Max;

        p = qu[front];

        cout<<p->data;

*if*(p->lchild)

        {

            rear = (rear+1)%Max;

            qu[rear]=p->lchild;

        }

*if*(p->rchild)

        {

            rear = (rear+1)%Max;

            qu[rear]=p->rchild;

        }

    }

}

*//打印主界面*

void welcome()

{

    system("cls");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓  ☆      二 叉 树 遍 历 测 试      ☆  〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓★★★★★         ★★★★★★★         ★★★★★〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓★★★★★         ★★★★★★★         ★★★★★〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          1.先序遍历        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          2.中序遍历        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          3.后序遍历        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          4.层次遍历        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          0.退出系统        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆                            ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

}

## 主函数完善

int main(){

    int choice,len;

    DBTree \* DT;

    char prestr[Max],instr[Max],aftstr[Max];

    welcome();

    cout<<"1.根据先序遍历和中序遍历构建二叉树。"<<endl;

    cout<<"2.根据后序遍历和中序遍历构建二叉树。"<<endl;

    cout<<"请输入想要构建二叉树的方式："<<endl;

    cin>>choice;

*if*(choice==1){

        cout<<"请分别输入先序遍历和中序遍历(空格分隔):"<<endl;

        cin>>prestr>>instr;

        len = strlen(prestr);

        DT = CreateDBTreeByPre(instr,prestr,len);

    }*else*{

        cout<<"请分别输入后序遍历和中序遍历(空格分隔):"<<endl;

        cin>>aftstr>>instr;

        len = strlen(aftstr);

        DT = CreateDBTreeByAft(instr,aftstr,len);

    }

    cout<<"构建成功！"<<endl;

*while*(1){

        cout<<"请输入功能序号："<<endl;

        cin>>choice;

*switch* (choice)

        {

*case* 1:

            cout<<"先序遍历为:";

            PreOrder(DT);

            cout<<endl;

*continue*;

*case* 2:

            cout<<"中序遍历为:";

            InOrder(DT);

            cout<<endl;

*continue*;

*case* 3:

            cout<<"后序遍历为:";

            PostOrder(DT);

            cout<<endl;

*continue*;

*case* 4:

            cout<<"层次遍历为:";

            LevelOrder(DT);

            cout<<endl;

*continue*;

*case* 0:

*break*;

*default*:

            cout<<"先序遍历为:";

            PreOrder(DT);

            cout<<endl;

*continue*;

        }

    }

    system("pause");

*return* 0;

}

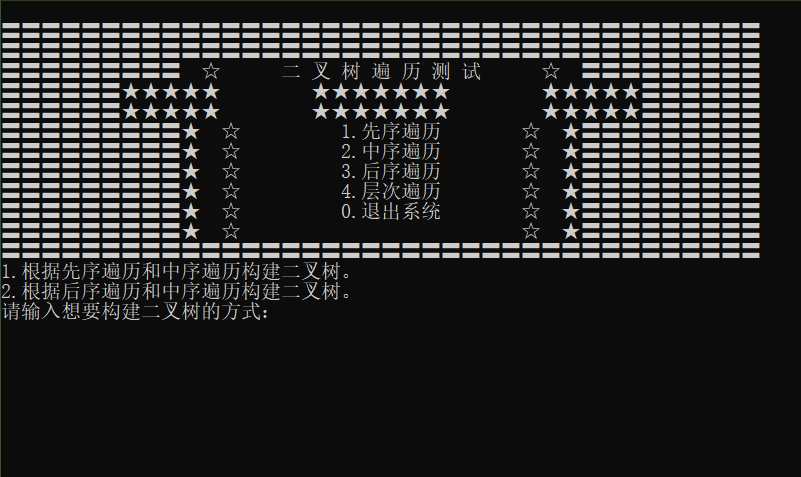
# **调试分析**

1. 模块划分较合理。
2. 本次测试根据用户选择可通过先序和中序或者中序和后序并采用递归的方式创建二叉树。
3. 二叉树的三种简单遍历方式相比于层次遍历实现要简单得多，用递归就能巧妙的解绝，而层次遍历需要队列存储节点。
4. 时间复杂度相当。
5. 本次实验报告完成了对二叉树的各种遍历运算，是一次良好的程序设计训练。

# **用户使用说明**

程序运行环境为vscode,执行文件为DBTree.cpp

用户界面：



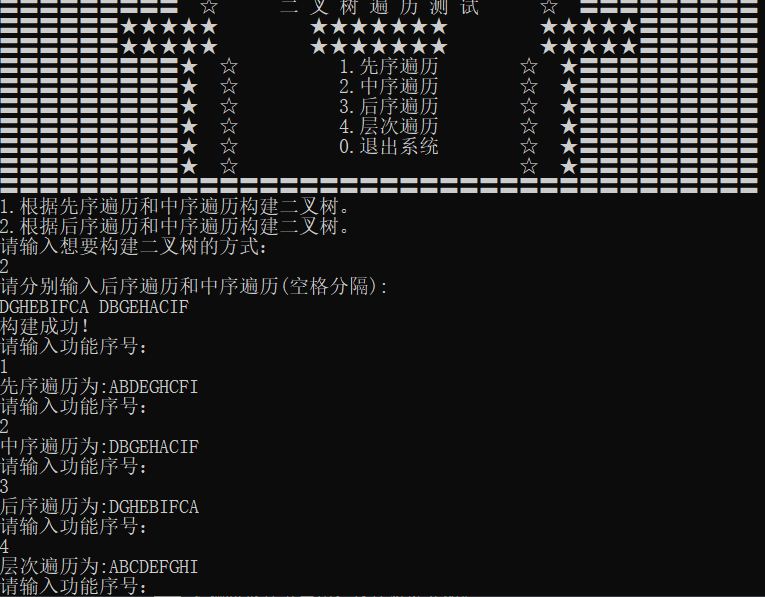
先输入想要构建二叉树的方式，然后依次对应序列，最后创建成功，选择对应的功能。

# **测试结果**

## 先序和中序：



## 中序和后序：



# **实验体会**

通过本次测试，学习了二叉树的逻辑和存储结构，以及四种遍历运算，并且了解到了各自的优缺点，大致总结一下吧：

1. 先序中序后序：
   * 1. 采用递归的话执行时间普遍是要小于层次遍历的
     2. 三种遍历方式比较的话，二叉树较小的情况下，三种方式遍历时间大致相同，但对于比较大的二叉树，中序遍历执行时间最短，其次是先序遍历，后序遍历最慢。
     3. 非递归的遍历方式执行时间要更长。
2. 层次：
3. 符合人阅读方式的遍历方式，但是算法实现要比前三种复杂一些，执行时间最慢。
4. 不能用递归实现，只能用队列等方式临时存储节点以便层次遍历。

# **附录**

源程序：

DBTree.cpp