小项目2：已知一个按先序序列输入的字符序列，如abc,,de,g,,f,,,(其中逗号表示空节点)。请建立二叉树并按中序和后序方式遍历二叉树，最后求出叶子节点个数和二叉树深度。

输入

 输入一个长度小于50个字符的字符串。

输出

输出共有4行：  
第1行输出中序遍历序列；  
第2行输出后序遍历序列；  
第3行输出叶子节点个数；  
第4行输出二叉树深度。

示例输入

abc,,de,g,,f,,,

示例输出

cbegdfa cgefdba 3 5

完成程序代码和项目报告。

**项目2实验报告**

1. **实验内容简介**

本实验共需要完成五项任务：

* + - 1. 根据输入先序建立二叉树；
      2. 按中序方式遍历二叉树并输出序列；
      3. 按后序方式遍历二叉树并输出序列；
      4. 求出二叉树的叶子结点个数；
      5. 求出二叉树的深度。

1. **程序设计思路**
   * + 1. **定义二叉树结点数据结构**

建立一个Node结构体，其中包含一个char类型的data用于存储字母，一个Node类型的\*Lchild指针指向左子树和一个Node类型的\*Rchild指针指向右子树。再建立一个指向Node这种结构的\*p指针。

* + - 1. **如何先序建立二叉树？**

建立一个build(p &T)函数用于先序建立二叉树，T为返回node的指针。函数中定义一个char类型的c用于读入输入，若c为“，”，则为空结点，T=NULL；若T为换行符“\n”，则输入结束并返回；否则c就是字母结点，则创建一个新的指针T，并让T->data=c，接着递归创建其左子树和右子树。

* + - 1. **如何按中序、后序方式遍历二叉树并输出序列？**

要按中序方式遍历二叉树，先访问左子树，再访问根节点，最后访问右子树；要按后序方式遍历二叉树，先访问左子树，再访问右子树，最后访问根节点；在访问根节点时打印该结点的data即可完成输出。

* + - 1. **如何求出二叉树的叶子结点个数？**

叶子结点就是左右子树都是空子树的结点，据此完成判断。如果二叉树为空，则叶子结点数为0；如果二叉树只有一个结点，则叶子结点数为1；否则，二叉树的叶子结点数为左右子树叶子结点数之和，据此完成递归。

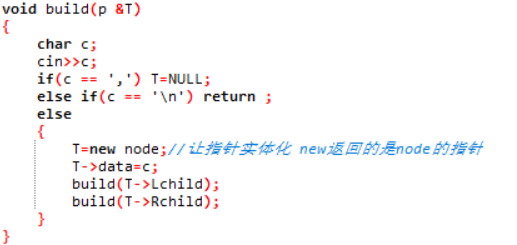
* + - 1. **如何求出二叉树的深度？**

如果二叉树为空，则深度为0；如果二叉树只有一个结点，则深度为1；否则，二叉树的深度为左右子树深度的最大值+1。

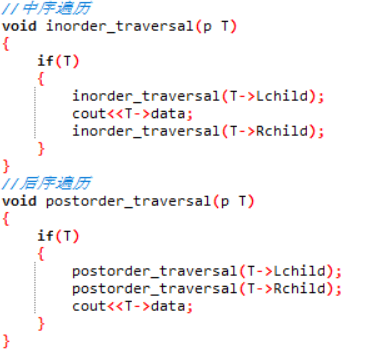
1. **算法分析与设计**

每项任务都被设计成一个封装好的函数，这样使整个程序井然有序。

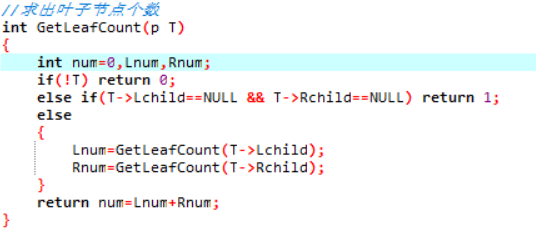
* + - 1. **先序建立二叉树**



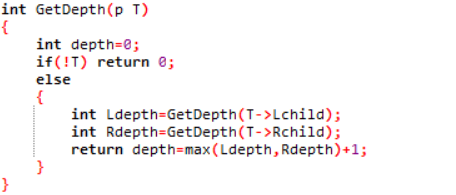
* + - 1. **按中序、后序方式遍历二叉树并输出序列**



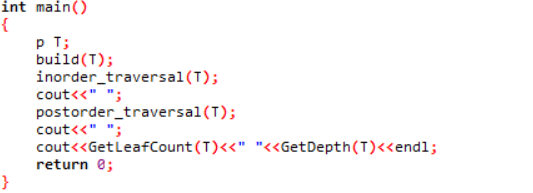
* + - 1. **输出二叉树的叶子结点数**



* + - 1. **输出二叉树的深度**



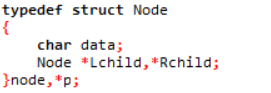
* + - 1. **主函数**



1. **数据结构设计**

二叉树结点数据结构采用结构体的形式实现，其中包含一个char类型的data用于存储字母，一个Node类型的\*Lchild指针指向左子树和一个Node类型的\*Rchild指针指向右子树。再建立一个指向Node这种结构的\*p指针。

具体代码如下：



1. **系统实现**

整个程序的流程图如下所示：

输入先序序列

建立二叉树

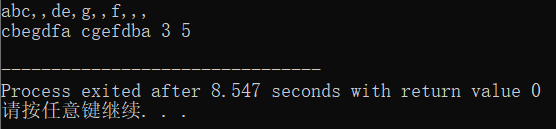
输出中序、后序序列

输出叶子结点数

输出二叉树深度

结束

1. **测试结果**



**附件：**

**示例二叉树**

**源代码**

#include<bits/stdc++.h>

using namespace std;

//定义结点结构体

typedef struct Node

{

char data;

Node \*Lchild,\*Rchild;

}node,\*p;

//先序建立二叉树

void build(p &T)

{

char c;

cin>>c;

if(c == ',') T=NULL;

else if(c == '\n') return ;

else

{

T=new node;//让指针实体化 new返回的是node的指针

T->data=c;

build(T->Lchild);

build(T->Rchild);

}

}

//中序遍历

void inorder\_traversal(p T)

{

if(T)

{

inorder\_traversal(T->Lchild);

cout<<T->data;

inorder\_traversal(T->Rchild);

}

}

//后序遍历

void postorder\_traversal(p T)

{

if(T)

{

postorder\_traversal(T->Lchild);

postorder\_traversal(T->Rchild);

cout<<T->data;

}

}

//求出叶子节点个数

int GetLeafCount(p T)

{

int num=0,Lnum,Rnum;

if(!T) return 0;

else if(T->Lchild==NULL && T->Rchild==NULL) return 1;

else

{

Lnum=GetLeafCount(T->Lchild);

Rnum=GetLeafCount(T->Rchild);

}

return num=Lnum+Rnum;

}

//求出二叉树的深度

int GetDepth(p T)

{

int depth=0;

if(!T) return 0;

else

{

int Ldepth=GetDepth(T->Lchild);

int Rdepth=GetDepth(T->Rchild);

return depth=max(Ldepth,Rdepth)+1;

}

}

int main()

{

p T;

build(T);

inorder\_traversal(T);

cout<<" ";

postorder\_traversal(T);

cout<<" ";

cout<<GetLeafCount(T)<<" "<<GetDepth(T)<<endl;

return 0;

}