Experiment topic: Basic operation of binary tree

Experiment purpose :

1. Master the definition of binary tree;
2. Master the basic operations of the binary tree, such as the establishment of the binary tree, traversal, statistics of the number of nodes, calculation of the depth of the tree, etc.

Experiment content:

Implement the following algorithm recursively :

1. Represent a binary tree with a binary linked list, and build a binary tree (algorithm 5.3);
2. Output the in-order traversal result of the binary tree (algorithm 5.1);
3. Output the result of the preorder traversal of the binary tree (see the lecture note);
4. Output the post-order traversal results of the binary tree (see lecture notes);
5. Calculate the depth of the binary tree (algorithm 5.5);
6. Count the number of nodes in the binary tree (algorithm 5.6);
7. Count the number of leaf nodes of the binary tree;
8. Count the number of nodes with a degree of 1 in the binary tree;
9. Output the path from each leaf node to the root node in the binary tree .

CODE:

BiTNode.cpp

```
#include "StdAfx.h"

#include "BiTNode.h"


void CreateBiTree(BiTree &T)

{ //Enter the value of the node in the binary tree (one character) according to the order of the order, and create the binary tree T represented by the binary linked list

    char ch;

    cin>>ch;

    if(ch=='#') //End of recursion, build an empty tree

        T=NULL; //Create binary tree recursively

    else

    {

        T=new BiTNode; //Generate root node

        T->data=ch; //Set the root node data field to ch

        CreateBiTree(T->lchild); //Create the left subtree recursively

        CreateBiTree(T->rchild); //Create the right subtree recursively
```

```
    } //else
}

void InOrderTraverse(BiTree T)
{ //Recursive algorithm for inorder traversal of binary tree T
    if(T) //if the binary tree is not empty
    {
        InOrderTraverse(T->lchild); //traverse the left subtree in order
        cout<<T->data; //Access the root node
        InOrderTraverse(T->rchild); //Traverse right subtree in order
    }
}

void PreOrderTraverse(BiTree T)
{ //Recursive algorithm for preorder traversal of binary tree T
    if(T) //if the binary tree is not empty
    {
        cout<<T->data; //Access the root node
        PreOrderTraverse(T->lchild); //Traverse the left subtree in preorder
        PreOrderTraverse(T->rchild); //Traverse right subtree in preorder
    }
}

void PostOrderTraverse(BiTree T)
{ //Recursive algorithm for post-order traversal of binary tree T
    if(T) //if the binary tree is not empty
    {
        PostOrderTraverse(T->lchild); //Postorder traversal of the left subtree
        PostOrderTraverse(T->rchild); //Postorder traversal of the right subtree
        cout<<T->data; //Access the root node
    }
}

int Depth(BiTree T)
{ // Calculate the depth of the binary tree
```

```
    if(T==NULL)
        return 0; //If it is an empty tree, the depth is 0, and the recursion ends
    else
    {
        int m=Depth(T->lchild); //recursively calculate the depth of the left subtree as m
        int n=Depth(T->rchild); //Recursively calculate the depth of the right subtree as m
        if(m>n) //Add 1 to the greater of the depth m and n of the binary tree
            return(m+1);
        else
            return(n+1);
    }
}

int NodeCount(BiTree T)
{ //Statistics of the number of nodes in the binary tree T
    if(T==NULL) //If it is an empty tree, the number of nodes is 0, and the recursion ends
        return 0;
    else
        return NodeCount(T->lchild)+NodeCount(T->rchild)+1; //Otherwise, the number of nodes is the number of nodes in the left subtree + the number of nodes in the right subtree+1
}

int LeafNodeCount(BiTree T)
{ //Statistics of the number of leaf nodes in the binary tree T
    if(T==NULL) //If it is an empty tree, the number of leaves is 0
        return 0;
    if(T->lchild==NULL&&T->rchild==NULL) //If it is the root node, the number of leaf nodes is 1
        return 1;
    else
        return LeafNodeCount(T->lchild)+LeafNodeCount(T->rchild); //Otherwise, the number of leaf nodes is the number of leaf nodes in the left subtree + the number of leaf nodes in the right subtree
}

int OneNodeCount(BiTree T)
{ //Statistics of the number of nodes with a degree of 1 in the binary tree T
    if(T==NULL) //If it is an empty tree, the number of nodes with degree 1 is 0
```

```
        return 0;
    if(T->lchild==NULL&&T->rchild!=NULL) // if the left subtree is empty
        return OneNodeCount(T->rchild)+1; //The number of nodes with degree 1 is the number of nodes with degree 1 in the left subtree+1
    if(T->lchild!=NULL&&T->rchild==NULL) //if the right subtree is empty
        return OneNodeCount(T->lchild)+1; //The number of nodes with degree 1 is the number of nodes with degree 1 in the right subtree+1

    else //both left and right subtrees are not empty
        return OneNodeCount(T->lchild)+OneNodeCount(T->rchild); //The number of nodes with degree 1 is the number of nodes with degree 1 in the left subtree + the number of nodes with degree 1 in the right subtree
}

void AllPath(BiTree T, TElemType path[], int pathlen)
{ //Output the path from each leaf node to the root node in the binary tree, pathlen is initially 0
    if(T!=NULL)
    {
        if(T->lchild==NULL&&T->rchild==NULL) //leaf node
        {
            cout<<" "<<T->data<<" to the root node path: "<<T->data;
            for(int i=pathlen-1;i>=0;i--)
                cout<<path[i];
            cout<<endl;
        }
        else
        {
            path[pathlen++]=T->data; //put the current node into the path, add 1 to the path length
            AllPath(T->lchild,path,pathlen); //Recursively traverse the left subtree
            AllPath(T->rchild,path,pathlen); //Recursively traverse the right subtree
            pathlen--; //restore environment
        }
    }
}

void ChangeLR(BiTree &T)
{ //Exchange the left child and right child of each node of the binary tree
```

```
        BiTree temp;

        if(T->lchild==NULL||T->rchild==NULL) //One of the left and right subtrees is empty, return

                return;

        else //Exchange the left and right child nodes

        {

                temp=T->lchild;

                T->rchild = T->lchild;

                T->rchild=temp;

        }

        ChangeLR(T->lchild); //recursively exchange the left subtree

        ChangeLR(T->rchild); //recursively exchange the right subtree

}
```

BiTree.cpp

```
// BiTree.cpp : Defines the entry point for the console application.

//


#include "stdafx.h"

#include "BiTNode.h"


int main()

{

        BiTree T;

//Test example AB#CD##E##F#GH###

cout<<"Preorder traversal input:";

CreateBiTree(T);


cout<<"Inorder traversal output:";

InOrderTraverse(T);


cout<<endl<<"Preorder traversal output:";

PreOrderTraverse(T);


cout<<endl<<"post-order traversal output:";

PostOrderTraverse(T);
```

```
cout<<endl<<"The depth of the tree:"<<Depth(T);

cout<<endl<<"Number of nodes:"<<NodeCount(T);

cout<<endl<<"The number of leaf nodes:"<<LeafNodeCount(T);

cout<<endl<<"Number of nodes with degree 1:"<<OneNodeCount(T);

cout<<endl<<"All paths from each leaf node to the root node in the binary tree:"<<endl;

char path[256];
int pathlen=0;
AllPath(T,path,pathlen);

BiTree tem=T;
ChangeLR(tem);
cout<<"The result of pre-order traversal output exchange:";
PreOrderTraverse(tem);
        cout<<endl;
return 0;
}
```

StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//      BiTree.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information

#include "StdAfx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

BiTNode.h

```
// ---------- Binary linked list storage representation of binary tree ---------
```

```
typedef struct BiTNode
{
	TElemType data; //node data field
	struct BiTNode *lchild,*rchild; // left and right child pointers
}BiTNode,*BiTree;



void CreateBiTree(BiTree &T); //Input the value (one character) of the nodes in the binary tree in the order of the first order, and create the binary tree T represented by the binary linked list
void InOrderTraverse(BiTree T); //Recursive algorithm for inorder traversal of binary tree T
void PreOrderTraverse(BiTree T); //Recursive algorithm for preorder traversal of binary tree T
void PostOrderTraverse(BiTree T); //Recursive algorithm for post-order traversal of binary tree T
int Depth(BiTree T); //Calculate the depth of the binary tree
int NodeCount(BiTree T); //Statistics of the number of nodes in the binary tree T
int LeafNodeCount(BiTree T); //count the number of leaf nodes in the binary tree T
int OneNodeCount(BiTree T); //Statistics of the number of nodes with degree 1 in binary tree T
void AllPath(BiTree T,TElemType path[],int pathlen); //Output the path from each leaf node to the root node in the binary tree, pathlen is initially 0
void ChangeLR(BiTree &T); //Exchange the left child and right child of each node of the binary tree
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#if !defined(AFX_STDAFX_H__CF4282C6_A4F2_41C4_BB61_FD204A3EFB3A__INCLUDED_)
#define AFX_STDAFX_H__CF4282C6_A4F2_41C4_BB61_FD204A3EFB3A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define WIN32_LEAN_AND_MEAN		// Exclude rarely-used stuff from Windows headers
```

```
#include <stdio.h>

#include <iostream>

using namespace std;


typedef char TElemType;


// TODO: reference additional headers your program requires here

//{{AFX_INSERT_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__CF4282C6_A4F2_41C4_BB61_FD204A3EFB3A__INCLUDED_)
```

Test Results: