**实验四**

#include <cstdio>

#include <cstdlib>

#include <stack>

using namespace std;

typedef char ElementType;

typedef struct TreeNode\* BinTree;

struct TreeNode

{

ElementType Data;

BinTree lchild;

BinTree rchild;

};

void createBinTree(BinTree &T)

{

char ch;

scanf("%c",&ch);

if(ch == '0')T = NULL;

else

{

if(!(T = (TreeNode\*)malloc(sizeof(TreeNode))))return;

T->Data = ch;

createBinTree(T->lchild);

createBinTree(T->rchild);

}

}

//非递归先序

void PreOrderTraversal(BinTree BT)

{

BinTree T;

stack<BinTree> s;

T = BT;

while(T || !s.empty())

{

while(T)

{

s.push(T);

printf("%5c",T->Data);

T = T->lchild;

}

if(!s.empty())

{

T = s.top();

s.pop();

T = T->rchild;

}

}

}

//非递归中序

void InOrderTraversal(BinTree BT)

{

BinTree T;

stack<BinTree> s;

T = BT;

while(T || !s.empty())

{

while(T)

{

s.push(T);

T = T->lchild;

}

if(!s.empty())

{

T = s.top();

s.pop();

printf("%5c",T->Data);

T = T->rchild;

}

}

}

//非递归后序

void PostOrderTraversal(BinTree BT)

{

BinTree T;

T = BT;

stack<BinTree> s;

stack<BinTree> q;

while(T || !s.empty())

{

while(T)

{

s.push(T);

q.push(T);

T = T->rchild;

}

if(!s.empty())

{

T = s.top();

s.pop();

T = T->lchild;

}

}

//逆向输出堆栈Q中的元素

while(!q.empty())

{

T = q.top();

q.pop();

printf("%5c",T->Data);

}

}

int main()

{

BinTree T;

createBinTree(T);

PreOrderTraversal(T);

printf("\n");

InOrderTraversal(T);

printf("\n");

PostOrderTraversal(T);

return 0;

}

因为树的本身就是用递归定义的，因此采用递归的方法实现三种遍历，代码简洁且容易理解，而若采用非递归方法实现三种遍历，则要用栈来模拟实现。先序遍历需要记录当前节点以及保存每次出栈的结点，中序遍历则同时需要记录每次弹出栈中结点时的值，后序遍历较为麻烦，需要一个额外的指针指向上次记录的节点，如果一个节点没有左右孩子，那么记录，如果一个节点的左右孩子都已经记录过，那么这个节点也可以记录。而额外的指针就是用来判断左右孩子有没有被记录过的。在本次实验一开始完全不知道怎么开始，在与同学之间的交流与同学的提醒后才渐渐懂了实验的基本流程。