```
#include<iostream>
using namespace std;
#include<stdio.h>
#include <stdlib.h>       //使用了动态内存分配函数


#define OK 1                    //表示操作正确的常量
#define OVERFLOW -2             //内存溢出错误常量
#define ERROR 0                 //表示操作错误的常量
#define TRUE 1                  //表示逻辑真的常量
#define FALSE 0                 //表示逻辑假的常量


//>>>>>>>>>>>>>>>>>>>>>>>>>自定义数据类型<<<<<<<<<<<<<<<<<<<<<<<<<

typedef int  Status;        //状态码为int类型，用于保存操作结果（1成功0失败）
typedef char TElemType;     //二叉树节点数据域的元素类型
typedef enum {Link, Thread}Pointering;


//---------------二叉树的二叉链表存储表示-------------------
typedef struct BiThrNode{
    TElemType  data;
    struct BiThrNode  *lchild,*rchild;    //孩子结点指针
    Pointering LTag, RTag;
}BiThrNode,*BiThrTree;

#include "S_Q.h"            //栈和队列

//按先序序列构造线索二叉树
Status CreateBiThrTree(BiThrTree &T){

    //ch存储从键盘接收的字符
    TElemType ch;

    //从键盘接收字符
    ch = getchar();

    //判断输入的字符是否是空格
    if(ch == ' ') { //输入空格表示结点为空
        T = NULL;
    }//if
    else{ //不是空格，按正常结点对待

        //申请结点空间
        //if(!(T = (BiNode *)malloc(sizeof(BiNode))))
        //等效于以下两行代码
        //T = (BiNode *)malloc(sizeof(BiNode));
        //if(!(T = (BiNode *)malloc(sizeof(BiNode))))
        if(!(T = (BiThrNode *)malloc(sizeof(BiThrNode)))){
            printf("内存分配失败！\n");
            exit(OVERFLOW);
        }//if

        //生成根结点，并值Tag为Link
        T->data = ch;

        //递归的构建左子树
        CreateBiThrTree(T->lchild);
        if(T->lchild)
            T->LTag = Link;

        //递归的构建右子树
        CreateBiThrTree(T->rchild);
        if(T->rchild)
            T->RTag = Link;
    }//else

    //操作成功
    return OK;
}//CreateBiThrTree

//全局变量，始终指向刚刚访问过的结点
BiThrTree pre;

//线索化-------------------------------------------------
void InThreading(BiThrTree &p){
    if(p){
        InThreading(p->lchild);      //左子树线索化
        if(!p->lchild){              //前趋线索化
            p->LTag = Thread;
            p->lchild = pre;
        }
        if(!pre->rchild){            //后继线索化
```

```
            pre->RTag = Thread;
            pre->rchild = p;
        }
        pre = p;        //保持pre指向p的前趋
        InThreading(p->rchild);
    }
}//InThreading
//中序遍历二叉树T，并将其中序线索化，Thrt指向头结点
Status InOrderThreading(BiThrTree &Thrt, BiThrTree T){
    Thrt = new BiThrNode;
    if(!Thrt)
        exit(OVERFLOW);
    //建立头结点
    Thrt->LTag = Link;  Thrt->RTag = Link;
    Thrt->rchild = Thrt;    //右指针回指
    if(!T)                  //如果二叉树空，则左指针回指
        Thrt->lchild = Thrt;
    else{
        Thrt->lchild = T;  pre = Thrt;
        InThreading(T);     //中序遍历进行中序线索化
        //最后一个结点的线索化
        pre->rchild = Thrt; pre->RTag = Thread;
        Thrt->rchild = pre;
    }
    return OK;
}//InorderThreading



//定义函数指针类型
//Status: 函数的返回值类型
//VisitFunc: 指针名
//VertexType: 函数的参数列表
typedef Status(*VisitFunc)(TElemType);

//visit函数
Status PrintElement(TElemType e){
    cout << e ;
}

//线索遍历中序线索二叉树
Status InOrderTraverse_Thr(BiThrTree T, VisitFunc visit){
    //T指向头结点，头结点的lchild指向根结点
    BiThrNode *p = T->lchild;           //p指向根结点
    while(p!=T){
        while(p->LTag==Link)        //循环查找中序遍历的第一个结点
            p = p->lchild;
        if(!visit(p->data))         //访问最左端的左子树为空的结点
            return ERROR;
        while(p->RTag==Thread && p->rchild!=T){     //沿线索访问结点
            p = p->rchild;
            visit(p->data);
        }
        p = p->rchild;      //线索断裂时，需要向右寻找直接后继
    }
    return OK;
}//InOrderTraverse_Thr



int main(){
    BiThrTree T;
    CreateBiThrTree(T);
    BiThrNode *Thrt = new BiThrNode;
    InOrderThreading(Thrt,T);
    InOrderTraverse_Thr(Thrt, PrintElement);

    return 0;
}
```