```cpp
#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

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

#include "S_Q.h"            //栈和队列

//按先序序列构造二叉树
Status CreateBiTree(BiTree &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 = (BiNode *)malloc(sizeof(BiNode)))){
            printf("内存分配失败！\n");
            exit(OVERFLOW);
        }//if

        //生成根结点
        T->data = ch;

        //递归的构建左子树
        CreateBiTree(T->lchild);

        //递归的构建右子树
        CreateBiTree(T->rchild);
    }//else

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

//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 ;
}

//二叉树的三种遍历的递归实现-----------------------------------------------
//先序遍历___递归算法
Status PreOrderTraverse(BiTree T, VisitFunc visit){
    if(T){
        if( visit(T->data) )
            if( PreOrderTraverse(T->lchild, visit) )
                if( PreOrderTraverse(T->rchild, visit) )
                    return OK;
        return ERROR;
    }else
```

```
        return OK;
}//PreOrderTraverse

//中序遍历___递归算法
Status InOrderTraverse(BiTree T, VisitFunc visit){
    if(T){
        if( InOrderTraverse(T->lchild, visit) )
            if( visit(T->data) )
                if( InOrderTraverse(T->rchild, visit) )
                    return OK;
        return ERROR;
    }else
        return OK;
}//InOrderTraverse

//后序遍历___递归算法
Status PostOrderTraverse(BiTree T, VisitFunc visit){
    if(T){
        if( PostOrderTraverse(T->lchild, visit) )
            if( PostOrderTraverse(T->rchild, visit) )
                if( visit(T->data) )
                    return OK;
        return ERROR;
    }else
        return OK;
}//PostOrderTraverse


//二叉树的三种遍历的非递归实现----------------------------------------------
//先序遍历___非递归算法
Status PreOrderTraverse_nonRecursion(BiTree T, VisitFunc visit){
    SqStack S;
    InitStack_Sq(S);
    BiNode *p = T;
    while(!StackEmpty_Sq(S) || p){
       if(p){//访问根节点，根指针进栈，遍历左子树
            if(!visit(p->data))
                return ERROR;
            Push_Sq(S,p);    //根结点进栈
            p = p->lchild;   //遍历左子树
       }else{//根指针退栈，遍历右子树
            Pop_Sq(S,p);
            p = p->rchild;
       }//else
    }//while
    DestroyStack_Sq(S);
    return OK;
}//PreOrderTraverse_nonRecursion

//中序遍历___非递归算法，实现1
Status InOrderTraverse_nonRecursion1(BiTree T, VisitFunc visit){
    SqStack S;
    InitStack_Sq(S);
    Push_Sq(S, T);
    BiNode *p;
    while(!StackEmpty_Sq(S)){
        while(GetTop_Sq(S,p) && p) //向左走到尽头
            Push_Sq(S,p->lchild);
        Pop_Sq(S,p);               //空指针退栈
        if(!StackEmpty_Sq(S)){
            //访问结点向右一步
            Pop_Sq(S,p);
            if(!visit(p->data))
                return ERROR;
            Push_Sq(S,p->rchild);
        }//if
    }//while
    DestroyStack_Sq(S);
    return OK;
}//InOrderTraverse_nonRecursion1

//中序遍历___非递归算法，实现2
Status InOrderTraverse_nonRecursion2(BiTree T, VisitFunc visit){
    SqStack S;
    InitStack_Sq(S);
    BiNode *p = T;
    while(!StackEmpty_Sq(S) || p){
       if(p){//根指针进栈，遍历左子树
            Push_Sq(S,p);
            p = p->lchild;
       }else{//根指针退栈，访问根结点遍历右子树
            Pop_Sq(S,p);
            if(!visit(p->data))
                return ERROR;
```

```
169             p = p->rchild;
170         }//else
171     }//while
172     DestroyStack_Sq(S);
173     return OK;
174 }//InOrderTraverse_nonRecursion2
175 
176 //后序遍历___非递归算法
177 Status PostOrderTraverse_nonRecursion(BiTree T, VisitFunc visit){
178     SqStack S;
179     InitStack_Sq(S);
180     BiNode *p = T;
181     BiNode *q;  //用q标记已经被访问了的rchild
182 
183     while(!StackEmpty_Sq(S) || p){
184         while(p){    //向左走到尽头
185             Push_Sq(S,p);
186             p=p->lchild;
187         }
188 
189         //重置指针q的值为NULL
190         q = NULL;
191 
192         //栈不为空
193         while(!StackEmpty_Sq(S)){
194             GetTop_Sq(S, p);    //p指向栈顶元素
195             //这个条件表示p指向了叶子结点或者p的左右子树均被遍历过
196             if(p->rchild == NULL || p->rchild == q){
197                 if(!visit(p->data))//访问根结点
198                     return ERROR;
199                 if(p == T)
200                     return ERROR;
201                 q = p;                  //q指向的是p的上一次遍历过的结点
202                 Pop_Sq(S, p);             //根指针出栈
203             }//if
204             else{
205                 p = p->rchild;           //访问右子树
206                 break;                   //退出内层循环
207             }//else
208         }//while
209     }//while
210     DestroyStack_Sq(S);
211     return OK;
212 }//PostOrderTraverse_nonRecursion
213 
214 //-----------------------------------------------二叉树的层次遍历
215 Status LevelOrderTraverse(BiTree T, VisitFunc visit){
216     if(T==NULL)
217         return ERROR;
218     SqQueue Q;
219     InitQueue_Sq(Q);
220     BiNode *p=T;
221     EnQueue_Sq(Q, p);
222     while(!QueueEmpty_Sq(Q)){
223         DeQueue_Sq(Q, p);
224         if(!visit(p->data))
225             return ERROR;
226         if(p->lchild)
227             EnQueue_Sq(Q, p->lchild);
228         if(p->rchild)
229             EnQueue_Sq(Q, p->rchild);
230     }
231     DestroyQueue_Sq(Q);
232     return OK;
233 }
234 
235 int main(){
236     BiTree T;
237     cout << "请输入所要建立二叉树的先序序列(空子树用空格代替):\n";
238     CreateBiTree(T);
239     PreOrderTraverse(T, PrintElement);
240     cout << endl;
241     InOrderTraverse(T, PrintElement);
242     cout << endl;
243     PostOrderTraverse(T, PrintElement);
244     cout << endl;
245     cout << "---------------------------------------" << endl;
246     PreOrderTraverse_nonRecursion(T,PrintElement);
247     cout << endl;
248     InOrderTraverse_nonRecursion1(T,PrintElement);
249     cout << endl;
250     InOrderTraverse_nonRecursion2(T,PrintElement);
251     cout << endl;
252     PostOrderTraverse_nonRecursion(T,PrintElement);
```

```
253         cout << endl;
254         cout << "---------------------------------------" << endl;
255         LevelOrderTraverse(T,PrintElement);
256 
257         return 0;
258     }
259 
```