数据结构实验报告——实验七

学号：__20201060330__姓名：___胡诚皓___得分：_____

一、实验目的

1. **复习二叉树的逻辑结构、存储结构及基本操作；**
2. **掌握使用顺序结构实现二叉树。**

二、实验内容

阅读程序 bitree.h 和 functions.cpp,完成以下题目：

1. 写出 functions 文件中每个函数对应的功能、输入要求，输出或返回内容；
2. 将 functions 文件的每个函数中你认为的关键代码罗列出来，并注释其作用；
3. 创建主程序，给出友好界面对 functions 文件的每个函数进行测试；
4. 根据所完成的主程序修改 bitree 头文件，使其没有缺失代码或多余代码。

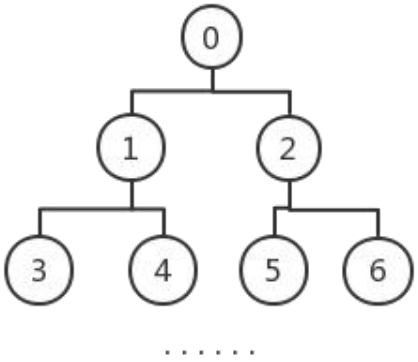
三、数据结构及算法描述

数据结构

宏定义和类型定义说明：

名称	说明
TelemType	即为 char 类型，作为二叉树中结点
MAX_TREE_SIZE	即为 100，作为二叉树的最大结点数
Status	即为 int 类型，作为函数的返回值的类型，约定值为 OK/ERROR
Boolean	即为 int 类型，作为逻辑布尔值使用，约定值为 TRUE/FALSE
TRUE、OK	即为 1，TRUE 作为 Boolean 类型的约定值，OK 作为 Status 类型的约定值
FALSE、ERROR	即为 0，FALSE 作为 Boolean 类型的约定值，ERROR 作为 Status 类型的约定值
SqBiTree	即为 char[100]，作为顺序存储二叉树的结构
QElemType	即为 int，作为队列元素
position	结构体，包含两个 int 类型的变量 level、order，level 表示二叉树中的某一层，order 表示 level 层中从左往右的序号。level 与 order 都从 1 开始计。
QueuePtr、LinkQueue	定义与使用方法与之前的实验报告中相同

其中 SqBiTree，用顺序存储方式存储二叉树，按层存储（下图中结点中数字为对应的顺序存储下标）



算法描述：

void InitBiTree(SqBiTree T)
使用 for 循环遍历顺序存储的二叉树 T，将所有元素都先置空，即赋值为空字符

void CreateBiTree(SqBiTree T)
①初始化局部变量，将 char 数组 s 全部设为空字符
②调用 InitBiTree 将 T 置空
③先去除所有换行符，以避免读入上次输出/输入的缓冲区残留字符；再用 gets 函数读入剩下的字符。最后通过 for 循环将局部变量 s 中读入的各个字符赋值给 T

Boolean BiTreeEmpty(const SqBiTree T)
通过判断根结点是否为空字符来确定该二叉树是否为空

int BiTreeDepth(SqBiTree const T)
①从后往前循环，找到顺序存储中二叉树的最后一个结点
②根据“每一层的最后一个结点下标为 2^n”来判断循环是否结束

TElemType Root(SqBiTree T)
调用 BiTreeEmpty 判断 T 是否为空，返回 ERROR，否则直接返回根结点的值（即顺序存储下标为 0 对应的元素）

TElemType Value(SqBiTree T, position e)
①先计算 level 层以上的所有结点的个数，即 level 本层第一个结点对应的下标

②本层第一个结点的下标加上 order 再减 1 就是对应结点的下标，直接返回对应值

Status Assign(SqBiTree T, position e, TElemType value)

①与 Value 函数中同理可以求得位置 e 对应的下标

②对根结点的修改进行特殊处理

③排除两种无效情况：给无父结点的叶结点赋非空值、给有子结点的父节点赋空值

④将对应元素赋值为 value

TElemType Parent(SqBiTree T, TElemType e)

①找到值为 e 的结点对应的下标

②根结点下标为 0 时，下标为 i 的结点的父结点为 $\frac{i+1}{2}$-1

TElemType LeftChild(SqBiTree T, TElemType e)

①找到值为 e 的结点对应的下标

②根结点下标为 0 时，下标为 i 的结点的左子结点为 $2i+1$

TElemType RightChild(SqBiTree T, TElemType e)

①找到值为 e 的结点对应的下标

②根结点下标为 0 时，下标为 i 的结点的右子结点为 $2i+2$

TElemType LeftSibling(SqBiTree T, TElemType e)

①找到值为 e 的右结点对应的下标

②根结点下标为 0 时，下标为 i 的结点的左兄弟结点为 $i-1$

TElemType RightSibling(SqBiTree T, TElemType e)

①找到值为 e 的左结点对应的下标

②根结点下标为 0 时，下标为 i 的结点的左兄弟结点为 $i+1$

void Move(SqBiTree Q, int j, SqBiTree T, int i)

①递归地将 Q 中 j 结点的左子树移动，作为 T 中 i 结点的左子树

②递归地将 Q 中 j 结点的右子树移动，作为 T 中 i 结点的右子树

③将 Q 的 j 结点本身移到 T 中 i 结点上，同时将 Q 的 j 结点置空

void InsertChild(SqBiTree T, TElemType p, int LR, SqBiTree C)

①找到值为 p 的结点对应的下标 j

②根据 LR 计算出 j 的左/右子结点的下标

③若以 k 为根的子树不是空树，就把从 T 的 k 结点开始的子树移至 k 的右结点开始的子树
④把从 C 的 i 结点开始的子树移至 T 的 k 结点上

Status DeleteChild(SqBiTree T, position p, int LR)
①将位置 p 转换为对应下标 i
②i 的左/右结点不空则将其入队
③将 i 结点置空
④出队一个下标赋值给 i。若队列为空，则 flag 被赋值为 FALSE

void VisitFunc(TElemType e)
输出除了空格和空字符之外的字符

void PreTraverse(SqBiTree T, int e)
①访问当前结点
②若当前结点有左子树，递归访问当前结点的左子树
③若当前结点有右子树，递归访问当前结点的右子树

void PreOrderTraverse(SqBiTree T)
若二叉树 T 不为空，则调用 PreTraverse 先序遍历 T

void InTraverse(SqBiTree T, int e)
①若当前结点有左子树，递归访问当前结点的左子树
②访问当前结点
③若当前结点有右子树，递归访问当前结点的右子树

void InOrderTraverse(SqBiTree T)
若二叉树 T 不为空，则调用 InOrderTraverse 中序遍历 T

void PostTraverse(SqBiTree T, int e)
①若当前结点有左子树，递归访问当前结点的左子树
②若当前结点有右子树，递归访问当前结点的右子树
③访问当前结点

void PostOrderTraverse(SqBiTree T)
若二叉树 T 不为空，则调用 PostOrderTraverse 中序遍历 T

void LevelOrderTraverse(SqBiTree T)

①从后往前循环，找到顺序存储中二叉树的最后一个结点
②由于顺序存储本来就是按层序存储二叉树的，因此从前往后遍历数组，在输出的时候跳过空结点即可

void Print(SqBiTree T)
①外层循环层数，调用 BiTreeDepth 得到 T 的层数
②内层从左往右循环第 j 层的各个结点，第 j 层有 2^{n-1} 个结点。调用 Value 获取对应位置结点的值。同样地，在输出时跳过空结点

InitQueue、DestroyQueue、EnQueue、DeQueue 与 QueueEmpty 函数在之前的实验报告中已经说明过，此处不再赘述。

四、详细设计

```
#include "bitree.h"
#include <iostream>

using namespace std;

int main() {
    SqBiTree root, root2;
    position pos;
    char ch;
    int LR;
    InitBiTree(root);
    InitBiTree(root2);
    LinkQueue queue;
    InitQueue(queue);

    cout << "进行函数测试，有些函数在其他函数中已经有调用，不再单独测试" << endl;
    cout << "按层序构造二叉树 root" << endl;
    CreateBiTree(root);
    cout << "二叉树根结点的值为" << Root(root) << endl;

    cout << "查询某个位置的结点" << endl;
    cout << "请输入层号（根结点为第 1 层）" << endl;
    cin >> pos.level;
    cout << "请输入该层的某个位置（该层最左边的结点为第 1 个）" << endl;
    cin >> pos.order;
    cout << "第" << pos.level << "层" << "的第" << pos.order << "个结点的值为" << Value(root,
pos) << endl;
```

```
cout << "修改某个位置的结点" << endl;
cout << "请输入层号（根结点为第 1 层）" << endl;
cin >> pos.level;
cout << "请输入该层的某个位置（该层最左边的结点为第 1 个）" << endl;
cin >> pos.order;
cout << "修改为（字符型）" << endl;
cin >> ch;
if (Assign(root, pos, ch) == ERROR)
    cout << "修改失败" << endl;
else
    cout << "修改成功" << endl;

cout << "按值查询结点" << endl;
cout << "查询父结点，请输入要查询的值（字符型）" << endl;
cin >> ch;
cout << ch << "的父结点的值为" << Parent(root, ch) << endl;
cout << "查询左子结点，请输入要查询的值（字符型）" << endl;
cin >> ch;
cout << ch << "的左子结点的值为" << LeftChild(root, ch) << endl;
cout << "查询右子结点，请输入要查询的值（字符型）" << endl;
cin >> ch;
cout << ch << "的右子结点的值为" << RightChild(root, ch) << endl;
cout << "查询左兄弟结点，请输入要查询的值（字符型）" << endl;
cin >> ch;
cout << ch << "的左兄弟结点的值为" << LeftSibling(root, ch) << endl;
cout << "查询右兄弟结点，请输入要查询的值（字符型）" << endl;
cin >> ch;
cout << ch << "的右兄弟结点的值为" << RightSibling(root, ch) << endl;

cout << "构造一棵只有右结点的树 root2" << endl;
CreateBiTree(root2);
cout << "将 root2 移至 root 的一个结点上" << endl;
cout << "请输入目标结点" << endl;
cin >> ch;
cout << "移至" << ch << "的左（0）/右（1）结点，请输入 0 或 1" << endl;
cin >> LR;
InsertChild(root, ch, LR, root2);
cout << "操作完成" << endl;

cout << "删除 root 中某一结点的左/右子树" << endl;
cout << "请输入目标层号（根结点为第 1 层）" << endl;
cin >> pos.level;
cout << "请输入目标层的某个位置（该层最左边的结点为第 1 个）" << endl;
cin >> pos.order;
```

```
    cout << "删除" << Value(root, pos) << "结点的左（0）/右（1）子树，请输入 0 或 1" << endl;
    cin >> LR;
    DeleteChild(root, pos, LR);

    cout << "遍历二叉树 root" << endl;
    cout << "前序遍历：";
    PreOrderTraverse(root);
    cout << endl << "中序遍历：";
    InOrderTraverse(root);
    cout << endl << "后序遍历：";
    PostOrderTraverse(root);
    cout << "调用 Print 层序遍历：" << endl;
    Print(root);
    cout << "调用 LevelOrderTraverse 层序遍历：";
    LevelOrderTraverse(root);


    cout << "测试队列销毁 DestroyQueue 函数" << endl;
    EnQueue(queue, 10);
    if (DestroyQueue(queue))
        cout << "测试通过" << endl;
    else
        cout << "测试失败" << endl;

    return 0;
}
```

五、程序代码

原来文件中的代码已经全部经过优化（包括细节上的数组越界风险、整体格式等），并在相应语句上都加了详细注释。

Question 1:

1. void InitBiTree(SqBiTree T)

功能：初始化顺序存储的二叉树，将 T 中的各元素置零

输入要求：SqBiTree 类型的 T

输出或返回内容：无输出，无返回值

2. void CreateBiTree(SqBiTree T)

功能：根据输入按层序创建一棵二叉树

输入要求：需保证 T 存在，按层序输入结点的值(字符)，空格表示空结点，结点数≤MAX_TREE_SIZE

输出或返回内容：无输出，无返回值

3. Status BiTreeEmpty(SqBiTree T)

功能：判断二叉树 T 是否为空

输入要求：需保证二叉树 T 存在

输出或返回内容：若 T 为空二叉树，返回 TRUE，否则返回 FALSE

4. int BiTreeDepth(SqBiTree T)

功能：得到二叉树 T 的深度

输入要求：需保证二叉树 T 存在

输出或返回内容：返回二叉树 T 的深度

5. TElemType Root(SqBiTree T)

功能：得到二叉树 T 根的值

输入要求：需保证二叉树 T 存在

输出或返回内容：若 T 为空，返回 ERROR；若 T 不为空，返回二叉树 T 根结点的值

6. TElemType Value(SqBiTree T, position e)

功能：查询 T 中位置 e 对应的结点的值

输入要求：需保证二叉树 T 存在

输出或返回内容：返回 T 中 e 位置对应的结点值

7. Status Assign(SqBiTree T, position e, TElemType value)

功能：修改 T 中位置 e 对应结点的值

输入要求：需保证二叉树存在

输出或返回内容：修改成功返回 OK，失败返回 ERROR

8. TElemType Parent(SqBiTree T, TElemType e)

功能：查找值为 e 的结点并返回其父结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为 e 的结点（根结点除外）的父结点的值；若树为空或找不到，则返回“空”

9. TElemType LeftChild(SqBiTree T, TElemType e)

功能：查找值为 e 的结点并返回其左子结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为 e 的结点（根结点除外）的左子结点的值；若树为空或找不到，则返回“空”

10. TElemType RightChild(SqBiTree T, TElemType e)

功能：查找值为 e 的结点并返回其右子结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为 e 的结点（根结点除外）的右子结点的值；若树为空或找不到，则返回“空”

11. TElemType LeftSibling(SqBiTree T, TElemType e)

功能：查找值为 e 的右结点并返回其左兄弟结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为 e 的右结点（根结点除外）的左兄弟结点的值；若树为空或找不到，则返回“空”

12. TElemType RightSibling(SqBiTree T, TElemType e)

功能：查找值为 e 的左结点并返回其右兄弟结点的值

输入要求：需保证二叉树存在

输出或返回内容：返回按层序找到的第一个值为 e 的左结点（根结点除外）的右兄弟结点的值；若树为空或找不到，则返回“空”

13. void Move(SqBiTree Q, int j, SqBiTree T, int i)

功能：把二叉树 Q 以 j 为根结点的子树移至 T 的 i 结点上

输入要求：需保证二叉树 Q 与 T 都存在

输出或返回内容：无输出，无返回值

14. void InsertChild(SqBiTree T, TElemType p, int LR, SqBiTree C)

功能：将二叉树 C 根据 LR 插入为 T 的值为 p 的结点的左或右子树

输入要求：二叉树 T 存在，p 是 T 中某个结点的值，LR 为 0 或 1，非空二叉树 C 与 T 不相交，C 的右子树为空

输出或返回内容：无输出，无返回值

15. Status DeleteChild(SqBiTree T, position p, int LR)

功能：根据 LR 为 0 或 1,删除 T 中位置 p 结点的左或右子树

输入要求：二叉树 T 存在，p 指向 T 中某个结点，LR 为 1 或 0

输出或返回内容：删除成功返回 OK，若 T 为空树，无法进行删除操作，返回 ERROR

16. void VisitFunc(TElemType e)

功能：输出除了空格和空字符之外的字符

输入要求：TElemType（此处为 char）类型的 e

输出或返回内容：若 e 不为空格和空字符，则输出它

17. void PreTraverse(SqBiTree T, int e)

功能：作为先序遍历的递归调用函数

输入要求：e 必须小于 MAX_TREE_SIZE

输出或返回内容：每次都调用 VisitFunc 访问当前结点

18. void PreOrderTraverse(SqBiTree T)

功能：先序遍历二叉树 T

输入要求：需要保证 T 存在

输出或返回内容：调用 PreTraverse 时会有对应输出

19. void InTraverse(SqBiTree T, int e)

功能：作为中序遍历的递归调用函数

输入要求：e 必须小于 MAX_TREE_SIZE

输出或返回内容：每次都调用 VisitFunc 访问当前结点

20. void InOrderTraverse(SqBiTree T)

功能：中序遍历二叉树 T

输入要求：需要保证 T 存在

输出或返回内容：调用 InTraverse 时会有对应输出

21. void PostTraverse(SqBiTree T, int e)

功能：作为后序遍历的递归调用函数

输入要求：e 必须小于 MAX_TREE_SIZE

输出或返回内容：每次都调用 VisitFunc 访问当前结点

22. void PostOrderTraverse(SqBiTree T)

功能：后序遍历二叉树 T

输入要求：需要保证 T 存在

输出或返回内容：调用 PostTraverse 时会有对应输出

23. void LevelOrderTraverse(SqBiTree T)

功能：层序遍历二叉树 T

输入要求：需要保证 T 存在

输出或返回内容：对每个非空结点调用 VisitFunc 进行访问

24. void Print(SqBiTree T)

功能：层序递归遍历 T，并逐层从左到右输出

输入要求：需要保证 T 存在

输出或返回内容：输出每一层的层号，每一层从左往右输出非空结点

25. Status InitQueue(LinkQueue &Q)

功能：队列初始化

输入要求：Q 为 LinkQueue 类型

输出或返回内容：初始化成功返回 OK，否则返回 ERROR

26. Status DestroyQueue(LinkQueue &Q)

功能：销毁队列，释放各个元素所占空间

输入要求：需要保证 Q 存在

输出或返回内容：全部释放成功返回 OK

27. Status EnQueue(LinkQueue &Q, QElemType e)

功能：元素 e 入列 Q

输入要求：需要保证 Q 存在

输出或返回内容：操作成功返回 OK，失败则返回 ERROR

28. Status QueueEmpty(LinkQueue Q)

功能：判断队列 Q 是否为空

输入要求：需要保证 Q 存在

输出或返回内容：队列为空返回 TRUE，队列非空返回 FALSE

29. Status DeQueue(LinkQueue &Q, QElemType &e)

功能：从 Q 中出列一个元素存于 e 中

输入要求：需要保证 Q、e 存在

输出或返回内容：队列为空返回 TRUE，队列非空返回 FALSE

Question 2:

1. /* 初始化顺序存储的二叉树 */

```
void InitBiTree(SqBiTree T) {
    for (int i = 0; i < MAX_TREE_SIZE; i++)
        T[i] = '\0';//初值设为空字符，方便后期调试观察
}
```

2. /* 按层序次序输入二叉树中结点的值(字符型)，构造顺序存储的二叉树 T，原数据将被置零 */

```
void CreateBiTree(SqBiTree T) {
    int l;
    char s[MAX_TREE_SIZE];
    char ch;
    memset(s, '\0', sizeof(s));
    InitBiTree(T);
    printf("请按层序输入结点的值(字符)，空格表示空结点，结点数≤%d:\n",
MAX_TREE_SIZE);
    while ((ch = (char) getchar()) == '\r' || ch == '\n');//吃掉换行符
    gets(s);//读入字符串
    //存入第一个字符
    T[0] = ch;
    l = (int) strlen(s);//求字符串的长度
    //从输入的第二个字符开始循环
    for (int i = 1; i <= l; i++)//将字符串赋值给 T
        T[i] = s[i - 1];
}
```

3. /* 假设二叉树 T 存在。若 T 为空二叉树，则返回 TRUE，否则 FALSE */

```
Boolean BiTreeEmpty(const SqBiTree T) {
    if (T[0] == '\0')//根结点为空字符，则树为空
        return TRUE;
    else
        return FALSE;
}
```

4. /* 假设二叉树 T 存在。返回 T 的深度 */

```
int BiTreeDepth(SqBiTree const T) {
    int i, j = -1;
    //找到最后一个结点
    for (i = MAX_TREE_SIZE - 1; i >= 0; i--)
        if (T[i] != NULL)
            break;
    i++;//便于计算
    do
        j++;
    while (i >= pow(2, j));//使用 do-while 循环配合 j=-1 的初始化计算层数
    return j;
}
```

5. /* 假设二叉树 T 存在。当 T 不空，返回 T 的根，否则返回 ERROR */

```
TElemType Root(SqBiTree T) {
    if (BiTreeEmpty(T))//虽然保证 T 存在，但 T 仍有可能为空
        return ERROR;
    else
        return T[0];
}
```

6. /* 假设二叉树 T 存在。e 是 T 中某个结点的位置，返回对应结点的值 */

```
TElemType Value(SqBiTree T, position e) {
    //计算之前所有层的结点数
    int pre_node = (int) pow(2, e.level - 1) - 1;
    return T[pre_node + e.order - 1];
}
```

7. /* 假设二叉树 T 存在。将 T 中 e 位置的结点赋新值 value */

```
Status Assign(SqBiTree T, position e, TElemType value) {
    //根据 e 计算 T 中对应下标
    int i = (int) pow(2, e.level - 1) + e.order - 2;
    //特判修改根结点值的操作
    if (i == 0) {
        if (T[i] == '\0')
            return ERROR;
        else {
            T[i] = value;
            return OK;
```

```
    }
    }
    if (value != '\0' && T[(i + 1) / 2 - 1] == '\0')//不允许给无父结点的叶结点赋非空值
        return ERROR;
    if (value == '\0' && (T[i * 2 + 1] != '\0' || T[i * 2 + 2] != '\0'))//不允许给有子结点的父节点赋空值
        return ERROR;
    T[i] = value;
    return OK;
}
```

8. /* 假设二叉树 T 存在。按层序找到第一个值为 e 的结点（根结点除外）并返回其父结点的值；否则返回“空” */

```
TElemType Parent(SqBiTree T, TElemType e) {
    int i;
    if (T[0] == '\0')//空树直接返回“空”
        return '\0';
    for (i = 1; i <= MAX_TREE_SIZE - 1; i++)//直接从存在父结点的下标 1 开始
        if (T[i] == e)//找到 e，直接返回其父结点的值
            return T[(i + 1) / 2 - 1];
    return '\0';//未找到 e，返回“空”
}
```

9. /* 假设二叉树 T 存在。按层序找到第一个值为 e 的结点（根结点除外）并返回其左子结点的值；否则返回“空” */

```
TElemType LeftChild(SqBiTree T, TElemType e) {
    int i;
    if (T[0] == '\0')//空树直接返回“空”
        return '\0';
    for (i = 0; i <= MAX_TREE_SIZE - 1; i++)
        if (T[i] == e)//找到 e，直接返回其左子结点的值
            return T[i * 2 + 1];
    return '\0';//未找到 e，返回“空”
}
```

10. /* 假设二叉树 T 存在。按层序找到第一个值为 e 的结点（根结点除外）并返回其右子结点的值；否则返回“空” */

```
TElemType RightChild(SqBiTree T, TElemType e) {
    int i;
```

```
    if (T[0] == '\0')//空树直接返回“空”
        return '\0';
    for (i = 0; i <= MAX_TREE_SIZE - 1; i++)
        if (T[i] == e)//找到 e，直接返回其右子结点的值
            return T[i * 2 + 2];
    return '\0';//未找到 e，返回“空”
}
```

11. /* 假设二叉树 T 存在。按层序找到第一个值为 e 的右结点（根结点除外）并返回其左兄弟结点的值；否则返回“空” */

```
TElemType LeftSibling(SqBiTree T, TElemType e) {
    int i;
    if (T[0] == '\0')//空树直接返回“空”
        return '\0';
    for (i = 1; i <= MAX_TREE_SIZE - 1; i++)
        if (T[i] == e && i % 2 == 0)//找到 e，其序号为偶数，即右孩子
            return T[i - 1];
    return '\0';//未找到 e，返回“空”
}
```

12. /* 假设二叉树 T 存在。按层序找到第一个值为 e 的左结点（根结点除外）并返回其右兄弟结点的值；否则返回“空” */

```
TElemType RightSibling(SqBiTree T, TElemType e) {
    int i;
    if (T[0] == '\0')//空树直接返回“空”
        return '\0';
    for (i = 1; i <= MAX_TREE_SIZE - 1; i++)
        if (T[i] == e && i % 2)//找到 e，其序号为奇数，即左孩子
            return T[i + 1];
    return '\0';//未找到 e，返回“空”
}
```

13. /* 把从 q 的 j 结点开始（包括 j 结点本身）的子树移为从 T 的 i 结点开始的子树 */

```
void Move(SqBiTree Q, int j, SqBiTree T, int i) {
    //若 q 的左子树不空，把 q 的 j 结点的左子树移为 T 的 i 结点的左子树
    if (2 * j + 1 < MAX_TREE_SIZE && Q[2 * j + 1] != '\0')
        Move(Q, (2 * j + 1), T, (2 * i + 1));
    //若 q 的右子树不空，把 q 的 j 结点的右子树移为 T 的 i 结点的右子树
    if (2 * j + 2 < MAX_TREE_SIZE && Q[2 * j + 2] != '\0')
        Move(Q, (2 * j + 2), T, (2 * i + 2));
```

```
    //把 q 的 j 结点移为 T 的 i 结点
    T[i] = Q[j];
    //把 q 的 j 结点置空
    Q[j] = '\0';
}
```

14. /* 假设二叉树 T 存在，p 是 T 中某个结点的值，LR 为 0 或 1，非空二叉树 C 与 T 不相交，C 的右子树为空。根据 LR 为 0 或 1,将 C 作为 T 中 p 结点的左或右子树；若 p 结点的原来有左或右子树，则先将该子树作为 C 的右子树 */

```
void InsertChild(SqBiTree T, TElemType p, int LR, SqBiTree C) {
    int j, k, i = 0;
    //查找 p 的下标
    for (j = 0; j < (int) pow(2, BiTreeDepth(T)) - 1; j++)
        if (T[j] == p)//找到了 p 的下标 j
            break;
    k = 2 * j + 1 + LR;//T 中将要被 C 替换掉的地方
    if (T[k] != '\0') //T 中以 k 为根的子树不是空树
        Move(T, k, T, 2 * k + 2);//把从 T 的 k 结点开始的子树移至 k 的右结点开始的子树
    Move(C, i, T, k); //把从 C 的 i 结点开始的子树移至 T 的 k 结点上
}
```

15. /* 假设二叉树 T 存在，p 指向 T 中某个结点，LR 为 1 或 0。根据 LR 为 0 或 1，删除 T 中 p 所指结点的左或右子树 */

```
Status DeleteChild(SqBiTree T, position p, int LR) {
    int i;
    Boolean flag = TRUE;//队列不空的标志
    LinkQueue q;
    InitQueue(q);//初始化队列，用于存放待删除的结点
    i = (int) pow(2, p.level - 1) + p.order - 2;//将位置 p 转换为下标
    if (T[i] == '\0')//T 为空树
        return ERROR;
    i = i * 2 + 1 + LR;//待删除子树的根结点的下标
    while (flag) {
        if (2 * i + 1 < MAX_TREE_SIZE && T[2 * i + 1] != '\0')//左结点不空
            EnQueue(q, 2 * i + 1);//入队左结点的下标
        if (2 * i + 2 < MAX_TREE_SIZE && T[2 * i + 2] != '\0')//右结点不空
            EnQueue(q, 2 * i + 2);//入队右结点的下标
        T[i] = '\0';//删除当前结点
```

```
        flag = DeQueue(q, i);//获取队列中下一个结点下标并获取队列是否为空
    }
    return OK;
}
```

16. /* 输出除了空格和空字符之外的字符 */

```
void VisitFunc(TElemType e) {
    if (e != ' ' && e != '\0')
        printf("%c", e);
}
```

17. /* 先序遍历的递归调用函数 */

```
void PreTraverse(SqBiTree T, int e) {
    //访问当前根结点
    VisitFunc(T[e]);
    //左子树不空，则遍历之
    if (2 * e + 1 < MAX_TREE_SIZE && T[2 * e + 1] != '\0')
        PreTraverse(T, 2 * e + 1);
    //右子树不空，则遍历之
    if (2 * e + 2 < MAX_TREE_SIZE && T[2 * e + 2] != '\0')
        PreTraverse(T, 2 * e + 2);
}
```

18. /* 先序遍历二叉树 T */

```
void PreOrderTraverse(SqBiTree T) {
    if (!BiTreeEmpty(T))//判断树不为空
        PreTraverse(T, 0);
    printf("\n");
}
```

19. /* 中序遍历的递归调用函数 */

```
void InTraverse(SqBiTree T, int e) {
    if (2 * e + 1 < MAX_TREE_SIZE && T[2 * e + 1] != '\0')//左子树不空，则遍历
之
        InTraverse(T, 2 * e + 1);
    //访问当前根结点
    VisitFunc(T[e]);
    if (2 * e + 2 < MAX_TREE_SIZE && T[2 * e + 2] != '\0')//右子树不空，则遍历
之
        InTraverse(T, 2 * e + 2);
}
```

20. /* 中序遍历二叉树 T */

```
void InOrderTraverse(SqBiTree T) {
    if (!BiTreeEmpty(T))//判断树不为空
        InTraverse(T, 0);
    printf("\n");
}
```

21. /* 后序遍历的递归调用函数 */

```
void PostTraverse(SqBiTree T, int e) {
    if (2 * e + 1 < MAX_TREE_SIZE && T[2 * e + 1] != '\0')//左子树不空，则遍历
之
        PostTraverse(T, 2 * e + 1);
    if (2 * e + 2 < MAX_TREE_SIZE && T[2 * e + 2] != '\0')//右子树不空，则遍历
之
        PostTraverse(T, 2 * e + 2);
    //访问当前根结点
    VisitFunc(T[e]);
}
```

22. /* 后序遍历二叉树 T */

```
void PostOrderTraverse(SqBiTree T) {
    if (!BiTreeEmpty(T))//判断树不为空
        PostTraverse(T, 0);
    printf("\n");
}
```

23. /* 层序遍历二叉树 */

```
void LevelOrderTraverse(SqBiTree T) {
    int i = MAX_TREE_SIZE - 1, j;
    //找到最后一个非空结点的序号
    while (T[i] == '\0')
        i--;
    //从根结点起，按层序遍历二叉树
    for (j = 0; j <= i; j++)
        //跳过空结点
        if (T[j] != '\0')
            VisitFunc(T[j]);
    printf("\n");
}
```

24. /* 逐层，每层从左到右输出二叉树 */

```
void Print(SqBiTree T) {
    int j, k;
    position p;
    TElemType e;
    //外层循环每一层
    for (j = 1; j <= BiTreeDepth(T); j++) {
        printf("第%d 层: ", j);
        //内层从左向右循环第 j 层
        for (k = 1; k <= pow(2, j - 1); k++) {
            p.level = j;
            p.order = k;
            //调用 Value 得到 T 中处于 p 位置结点的值
            e = Value(T, p);
            if (e != '\0')
                printf("%c", e);
        }
        printf("\n");
    }
}
```

25. /* 队列初始化，即构造一个空队列*/

```
Status InitQueue(LinkQueue &Q) {
    //存储空间分配失败
    if (!(Q.front = Q.rear = (QueuePtr) malloc(sizeof(QNode))))
        return ERROR;
    Q.front->next = NULL;
    return OK;
}
```

26. /* 销毁队列，同时释放各个元素所占空间 */

```
Status DestroyQueue(LinkQueue &Q) {
    while (Q.front) {
        //将队尾指针始终指向要删除的元素的下一个
        Q.rear = Q.front->next;
        free(Q.front);
        Q.front = Q.rear;
    }
    return OK;
}
```

27. /* 插入元素 e 为 Q 的新的队尾元素 */

```
Status EnQueue(LinkQueue &Q, QElemType e) {
    QueuePtr p;
    //存储空间分配失败
    if (!(p = (QueuePtr) malloc(sizeof(QNode))))
        return ERROR;
    p->data = e;
    //新元素的 next 应为 NULL
    p->next = NULL;
    Q.rear->next = p;
    //队尾变为新入队的元素
    Q.rear = p;
    return OK;
}
```

28. /* 判断队列 Q 是否为空，空返回 TRUE，非空返回 FALSE */

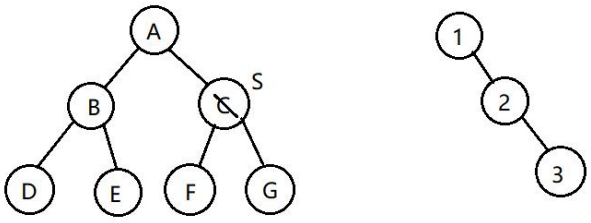
```
Status QueueEmpty(LinkQueue Q) {
    //带有头指针的队列判空方法
    if (Q.front == Q.rear)
        return TRUE;
    else
        return FALSE;
}
```

29. /* 从 Q 中将出列一个元素，用 e 保存出列元素的值。出列失败，返回 ERROR；出列成功，返回 OK */

```
Status DeQueue(LinkQueue &Q, QElemType &e) {
    QueuePtr p;
    //先判断队列 Q 是否为空
    if (QueueEmpty(Q) == TRUE)
        return ERROR;
    p = Q.front->next;
    e = p->data;
    Q.front->next = p->next;
    //带头结点的队列变为空时的特殊处理
    if (Q.rear == p)
        Q.rear = Q.front;
    free(p);
    return OK;
```

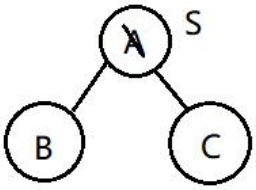
}

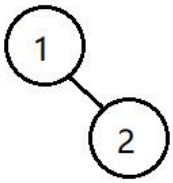
六、测试和结果

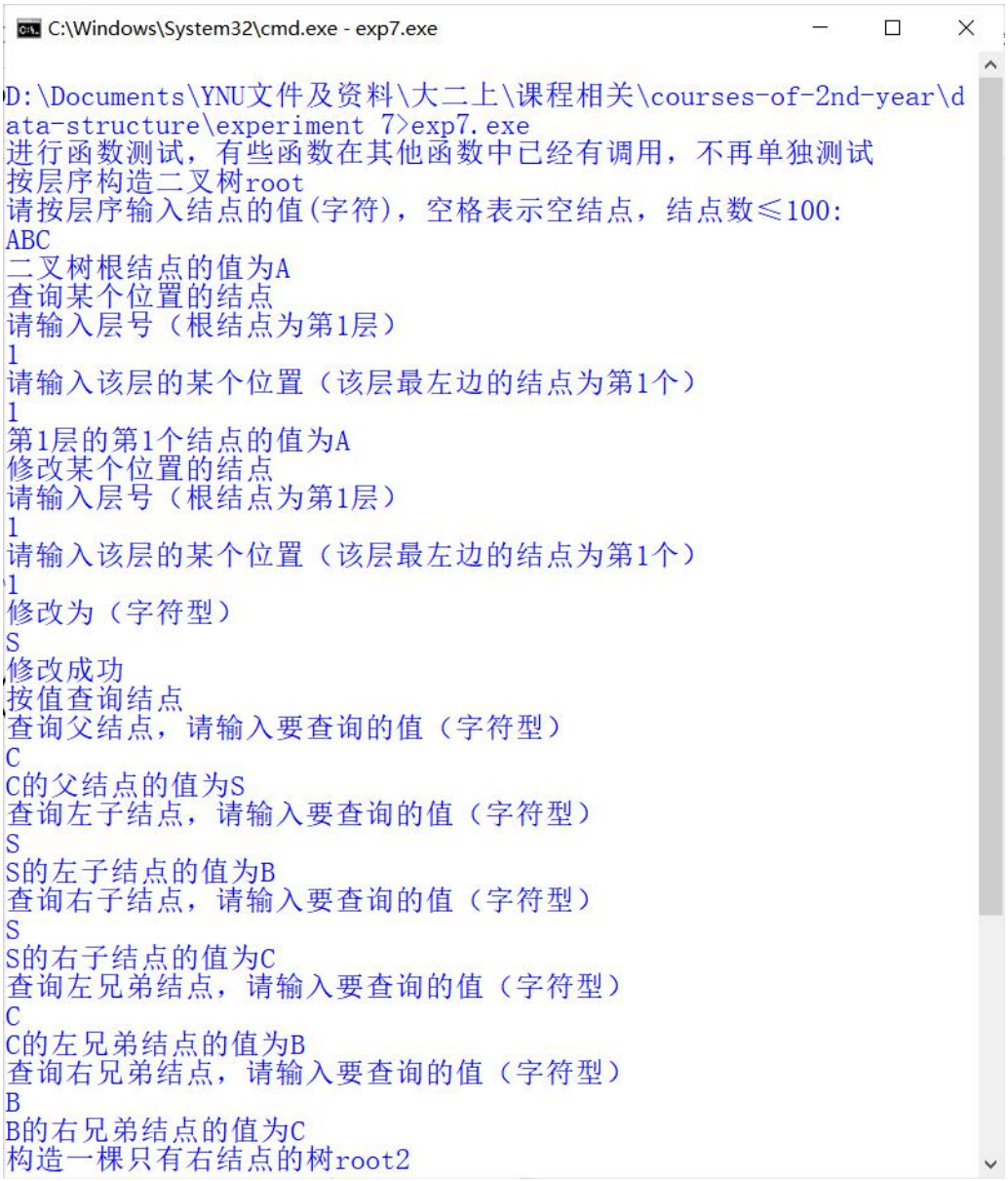


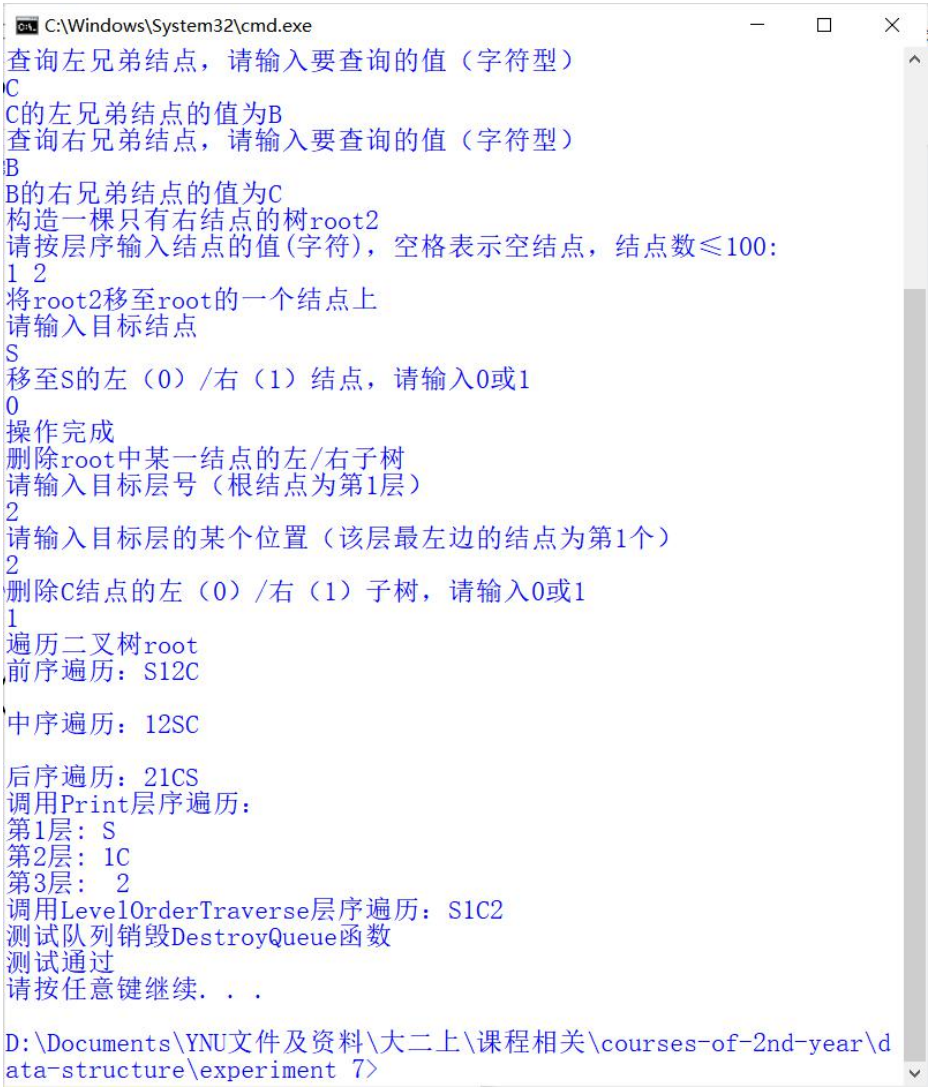

```
C:\Windows\System32\cmd.exe - exp7.exe

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\d
ata-structure\experiment 7>exp7.exe
进行函数测试，有些函数在其他函数中已经有调用，不再单独测试
按层序构造二叉树root
请按层序输入结点的值(字符)，空格表示空结点，结点数≤100:
ABCDEFG
二叉树根结点的值为A
查询某个位置的结点
请输入层号（根结点为第1层）
2
请输入该层的某个位置（该层最左边的结点为第1个）
2
第2层的第2个结点的值为C
修改某个位置的结点
请输入层号（根结点为第1层）
2
请输入该层的某个位置（该层最左边的结点为第1个）
2
修改为（字符型）
S
修改成功
按值查询结点
查询父结点，请输入要查询的值（字符型）
B
B的父结点的值为A
查询左子结点，请输入要查询的值（字符型）
D
D的左子结点的值为
查询右子结点，请输入要查询的值（字符型）
S
S的右子结点的值为G
查询左兄弟结点，请输入要查询的值（字符型）
B
B的左兄弟结点的值为
```

```
C:\Windows\System32\cmd.exe
B的左兄弟结点的值为
查询右兄弟结点，请输入要查询的值（字符型）
B
B的右兄弟结点的值为S
构造一棵只有右结点的树root2
请按层序输入结点的值(字符)，空格表示空结点，结点数≤100:
1 2   3
将root2移至root的一个结点上
请输入目标结点
B
移至B的左（0）/右（1）结点，请输入0或1
1
操作完成
删除root中某一结点的左/右子树
请输入目标层号（根结点为第1层）
2
请输入目标层的某个位置（该层最左边的结点为第1个）
1
删除B结点的左（0）/右（1）子树，请输入0或1
0
遍历二叉树root
前序遍历：AB123SFG

中序遍历：B123AFSG

后序遍历：321BFGSA
调用Print层序遍历：
第1层：A
第2层：BS
第3层：1FG
第4层：  2
第5层：    3
调用LevelOrderTraverse层序遍历：ABS1FG23
测试队列销毁DestroyQueue函数
测试通过
请按任意键继续. . .

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 7>
```






C:\Windows\System32\cmd.exe - exp7.exe
D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 7>exp7.exe
进行函数测试，有些函数在其他函数中已经有调用，不再单独测试
按层序构造二叉树root
请按层序输入结点的值(字符)，空格表示空结点，结点数≤100:
ABC
二叉树根结点的值为A
查询某个位置的结点
请输入层号（根结点为第1层）
1
请输入该层的某个位置（该层最左边的结点为第1个）
1
第1层的第1个结点的值为A
修改某个位置的结点
请输入层号（根结点为第1层）
1
请输入该层的某个位置（该层最左边的结点为第1个）
1
修改为（字符型）
S
修改成功
按值查询结点
查询父结点，请输入要查询的值（字符型）
C
C的父结点的值为S
查询左子结点，请输入要查询的值（字符型）
S
S的左子结点的值为B
查询右子结点，请输入要查询的值（字符型）
S
S的右子结点的值为C
查询左兄弟结点，请输入要查询的值（字符型）
C
C的左兄弟结点的值为B
查询右兄弟结点，请输入要查询的值（字符型）
B
B的右兄弟结点的值为C
构造一棵只有右结点的树root2



七、用户手册

输入的要查找的结点值、结点位置，都不能是空结点。**输出中的空（即对应位置没有内容）表示目标结点不存在。**