数据结构实验五

姓名：刘俊傲　　学号：**U201617047**　　班级：软工**1603**

1. 根据中序遍历结果和前序（后序）遍历结果重构二叉树

1. 问题描述：

   > 给出一棵二叉树的先序（或后序）遍历结果，以及中序遍历结果，如何构造这棵树

2. 问题分析与算法设计:

   > 1. 根据前序（后序也适用）遍历结果，找到树根，设为ROOT，并生成一个以ROOT为根节点，左右子树都为NULL的树
   > 2. 根据ROOT在中序遍历中的索引，将中序遍历结果分为：左子树，右子树；同时我们也要明确左右子树对应的前序（后序）遍历结果
   > 3. 递归的执行步骤1和2即可构造出唯一的二叉树。

3. 算法实现：

```
// 由前序序列和中序序列构造二叉树
#include <stdio.h>
#include <stdlib.h>

typedef struct BiNode* BiTree;
BiTree createBiTree(BiTree tree, char pre[], char in[]);

/************************************
    *二叉链表表示二叉树
    *data : 节点数据
    *lchild : 左孩子
    *rchild : 右孩子
************************************/
typedef struct BiNode
{
    char data;
    struct BiNode * lchild;
    struct BiNode * rchild;
};

/************************************
    *由前序序列和中序序列建立二叉树的过程
    *t为要建立的二叉树
    *pre为前序序列
    *in为中序序列
************************************/
BiTree createBiTree(BiTree tree, char pre[], char in[])
{
    if (pre == NULL || in == NULL)
    {
        return NULL;
    }
    int index, l_in = 0, r_in = 0, l_pre = 0, r_pre = 0;
    //根
    char rootNode = pre[0];
    //根在中序序列中的位置
    for (int i = 0; i < sizeof(in); i++)
    {
        if (rootNode == in[i])
        {
            index = i;
        }
    }
    char lchild_in[sizeof(in)];
    char rchild_in[sizeof(in)];
    //左孩子的中序序列
    for (int i = 0; i < index; i++)
    {
        lchild_in[l_in] = in[i];
        l_in++;
    }
    //右孩子的中序序列
    for (int i = index + 1; i < sizeof(in); i++)
```

```
    {
        rchild_in[r_in] = in[i];
        r_in++;
    }
    char lchild_pre[sizeof(pre)];
    char rchild_pre[sizeof(pre)];
    //左孩子的前序序列
    for (int i = 1; i < l_in + 1; i++)
    {
        lchild_pre[l_pre] = pre[i];
        l_pre++;
    }
    //右孩子的前序序列
    for (int i = l_in + 1; i < sizeof(pre); i++)
    {
        rchild_pre[r_pre] = pre[i];
        r_pre++;
    }

    tree = (BiTree)malloc(sizeof(BiNode));
    if (tree != NULL)
    {
        tree->data = rootNode;
        //递归创建左孩子
        createBiTree(tree->lchild, lchild_pre, lchild_in);
        //递归创建右孩子
        createBiTree(tree->rchild, rchild_pre, rchild_in);
    }
    return tree;
}
```

2. 二叉树的层序遍历

1. 问题描述：

   > 二叉树的层序遍历。使用队列作为辅助存储，按树的结点的深度，从根开始依次访问所有结点

2. 问题分析与算法设计:

   > 在进行层次遍历时，可设置一个队列结构，遍历从二叉树的根结点开始，首先将根结点指针入队列，然后从对列头取出一个元素，每取一个元素，同时执行下面两个操作：
   >
   > 1. 访问该元素所指结点，也就是打印出来
   > 2. 若该元素所指结点的左、右孩子结点非空，则将该元素所指结点的左孩子指针和右孩子指针顺序入队
   >
   > 此过程不断进行，当队列为空时，二叉树的层次遍历结束

3. 算法实现：

```
//二叉树的层序遍历

#include<stdio.h>
#include<stdlib.h>

#define YES 1
#define NO 0

typedef struct BiNode* biTree;
typedef struct QueueNode* queuePtr;
typedef struct QueueLink* queueLink;

void createbitree(biTree tree, int &sum);
void initQueue(queueLink queue);
void enQueue(queueLink quene, biTree tree);
void deQueue(queueLink quene, biTree tree);
int queueempty(queueLink quene);

/***************************
    *定义树结点结构
    *data :  存储的数据
    *lchild : 左儿子
    *rchild :  右儿子
*****************************/
typedef struct BiNode
{
    char data;
    struct BiNode *lchild;
    struct BiNode *rchild;
};

/***************************
    *定义队列结点结构
    *bitree : 树节点
    *next : 下一位置指针
*****************************/
typedef struct QueueNode
{
    biTree bitree;
    struct queueNode *next;
};

/***************************
    *定义队列指针
    *front : 队列前节点位置指针
    *rear : 队列后节点位置指针
***************************/
typedef struct QueueLink
{
    queueptr front;
    queueptr rear;
};
```

```
/**************************
    *创建一个二叉树
    *tree : 树的基本结构
    *sum : 树的深度
**************************/
void createbitree(biTree tree, int &sum)
{
    char ch;
    scanf("%c", &ch);
    if (ch == ' ')
    {
        tree = NULL;
    }
    else
    {
        tree = (biTree)malloc(sizeof(BiNode));
        if (tree == NULL)
        {
            return;
        }
        tree->data = ch;
        sum++;
        createbitree(tree->lchild, sum);
        createbitree(tree->rchild, sum);
    }
}

/**************************
    *初始化一个带头结点的队列
    *queue : 队列的基本结构
**************************/
void initQueue(queueLink queue)
{
    queue->front = queue->rear = (queuePtr)malloc(sizeof(QueueNode));
    queue->front->next = NULL;
}

/**************************
    *入队列
    *quene : 队列链表
    *tree : 树节点
**************************/
void enQueue(queueLink quene, biTree tree)
{
    queuePtr queueptr;
    queueptr = (queuePtr)malloc(sizeof(QueueNode));
    queueptr->bitree = tree;
    queueptr->next = NULL;
    quene->rear->next = queueptr->next;
    quene->rear = queueptr;
}


/**************************
```

```
    *出队列
    *quene : 队列链表
    *tree : 树节点
**************************/
void deQueue(queueLink quene, biTree tree)
{
    char data;
    queuePtr queueptr;
    queueptr = quene->front->next;
    tree = queueptr->bitree;
    data = tree->data;
    quene->front->next = queueptr->next;
    if (quene->rear == queueptr)
        quene->rear = quene->front;
    free(queueptr);
    printf("%c ", data);
}

/***************************
    *判断队列是否为空
    *quene : 队列链表
****************************/
int queueempty(queueLink quene)
{
    if (quene->front->next == NULL)
        return YES;
    return NO;
}

/***************************
    *按层次遍历树中结点
    *tree : 树节点
****************************/
void traverse(biTree tree)
{
    queueLink queuelink;
    biTree bitree;
    //初始化一个带头结点的队列
    initQueue(queuelink);
    bitree = tree;
    //入队列
    enQueue(queuelink, bitree);
    while (queueempty(queuelink) != 1)
    {
        //出队列
        deQueue(queuelink, bitree);
        if (bitree->lchild != NULL)
            enQueue(queuelink, bitree->lchild);
        if (bitree->rchild != NULL)
            enQueue(queuelink, bitree->rchild);
    }
    printf("\n");

}
```

```
//主函数
void main()
{
    int n = 0;
    biTree tree;
    createbitree(tree, n);
    printf("该二叉树共有%d个结点.\n", n);
    printf("按层次遍历树中结点其输出顺序为: \n");
    traverse(tree);
}
```