# 二叉树实验报告

## 需求分析

**1.输入要求**

用户需要提供一棵二叉树的扩展前序序列。

输入的扩展前序序列应该是一个字符串，其中每个节点的值用字符表示，节点之间使用空格或其他分隔符隔开。

用户输入的序列需要经过处理，以构建相应的二叉树。

**2.二叉树构建：**

根据输入的扩展前序序列构建二叉树。扩展前序序列的处理规则可以是先处理根节点，然后递归处理左子树和右子树。

确保处理过程中能够正确地建立二叉树的结构。

1. **遍历序列输出：**

**定义：**层次遍历，也称为广度优先遍历（Breadth-First Search，简称 BFS），是一种按层级顺序遍历树或图的算法。对于二叉树而言，层次遍历从树的根节点开始，逐层访问节点，先访问当前层的所有节点，然后按照从左到右的顺序逐层遍历。

输出构建好的二叉树的中序和后序遍历序列。

中序遍历是指按照左子树-根节点-右子树的顺序遍历二叉树。

后序遍历是指按照左子树-右子树-根节点的顺序遍历二叉树。

**4.层次遍历：**

对构建好的二叉树进行层次遍历。

层次遍历是按照树的层级顺序，从上到下、从左到右逐层遍历。

**5.错误处理：**

对于输入的扩展前序序列，进行必要的格式验证，确保输入的有效性。

在遇到无法构建二叉树的情况时，向用户报告错误并提供相关信息。

## 概要设计

1. 定义了一个结构体TreeNode，用于声明结构体组成。
2. 定义了函数BuildTree，用于处理输入的二叉树扩展前序序列。
3. 定义中序遍历函数midbianli。
4. 定义后序遍历函数lastbianli。
5. 定义层次遍历函数levelorderbianli。
6. 在main中先引用BuildTree创建二叉树，再以此引用midbianli，lastbianli，levelorderbianli输出不同的二叉树。
7. 定义了一个数字读取函数readNumber。

## 详细设计

**1.**struct TreeNode{

char data;

struct TreeNode\* left;

struct TreeNode\* right;

};

TreeNode结构体声明了整形变量data，一个指向结构体的指针left，一个指向另一个结构体的指针right。

**2.**struct TreeNode\* BuildTree(char\* tree, int\* index) {

if (tree[\*index] == '\0' || tree[\*index] == '#') {

(\*index)++;

return NULL;

}

// 读取数字

int num = Number(tree, index);

// 创建节点

struct TreeNode\* root = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

root->data = num;

// 跳过可能的空格

while (tree[\*index] == ' ') {

(\*index)++;

}

// 构建左子树

root->left = BuildTree(tree, index);

// 跳过可能的空格

while (tree[\*index] == ' ') {

(\*index)++;

}

// 构建右子树

root->right = BuildTree(tree, index);

// 跳过可能的空格

while (tree[\*index] == ' ') {

(\*index)++;

}

return root;

}

BuildTree用于处理输入的二叉树的扩展的前序序列

由于二叉树扩展前序序列的定义为：如果节点为空（null），则用一个特定的符号（如"#"）表示。如果节点不为空，先输出节点的值，然后按照先左子树后右子树的顺序递归地输出左子树和右子树的扩展前序序列。

对于二叉树

1

1. 3

4 5

有：对应的扩展前序序列可以表示为："1 2 4 # # 5 # # 3 # #"。其中，根节点的值是1，它有左子树"2 4 # #"和右子树"3 # #”。

故我们得到其扩展前序序列后，对于数组tree[index]先判断它是否为空（#）或者结束符（‘\0’）。如果是则该结点为NULL。

在这之间注意，每一轮运算都需要先跳过空格来进行运算，并且需要调用Number函数来计算得出data的值。

如果都不是，那么先运用malloc为root分配存储空间，然后为该结构体（tree->data）赋值为tree[index].然后将index++后运用递归函数反复调用 root->left=BuildTree(tree,index);

和 root->right=BuildTree(tree,index);

最终返回到root，即根节点。

**3.**void midbianli(struct TreeNode\* root){

if (root!=NULL){

midbianli(root->left);

printf("%d ",root->data);

midbianli(root->right);

}

}

中序遍历函数：先判断根节点是否为空，如果不为空，则通过中序遍历并输出该二叉树。

**4.**void lastbianli(struct TreeNode\* root){

if(root!=NULL){

lastbianli(root->left);

lastbianli(root->right);

printf("%d",root->data);

}

}

后序遍历函数，先判断根节点是否为空，如果不为空则通过后序遍历并输出该二叉树。

**5.**void levelorderbianli(struct TreeNode\* root){

if (root==NULL) return 1;

struct TreeNode\* queue[1000];

int a=0,b=0;

queue[b]=root;

b++;

while(a<b){

struct TreeNode\* temp=queue[a];

a++;

printf("%d ",temp->data);

if(temp->left!=NULL){

queue[b]=temp->left;

b++;

}

if(temp->right!=NULL){

queue[b]=temp->right;

b++;

}

}

}

层次遍历函数，即广度优先搜索。对于二叉树而言，层次遍历从树的根节点开始，逐层访问节点，先访问当前层的所有节点，然后按照从左到右的顺序逐层遍历。当根节点为空时，返回报错。否则定义一个结构体数组queue[1000]，定义一个整数型变量a=0和b=0。b即为结构体。

先令queue[b]=root，然后再b++，即到了下一层。

现在先定义一个临时的结构体temp=queue[a],即为queue的第[a]位的值。

先通过两个if函数将queue[a]的左右子节点（如果有且不为空）存入queue中，然后再以此输出。

6.

int Number(char \*tree, int\* index) {

int number = 0;

// 当遇到数字字符时，将其转换为整数，并累加到number中

while (tree[\*index] >= '0' && tree[\*index] <= '9') {

number = number \* 10 + (tree[\*index] - '0');

(\*index)++;

}

// 当遇到非数字字符时，结束循环，返回number

return number;

}

读取数字的函数，用于将以空格隔开的字符型数组读取为一个值。例如如果该字符为# # 19 # #，那么最后程序会输出19.先读入1然后乘10再加9.

**7.**int main(){

char tree[1000];

scanf("%[^\n]", tree);

int index=0;

//处理前序序列生成二叉树

struct TreeNode\* root=BuildTree(tree,&index);

//中序遍历

printf("中序遍历为：");

midbianli(root);

printf("\n");

//后序遍历

printf("后序遍历为：");

lastbianli(root);

printf("\n");

printf("层次遍历为：");

levelorderbianli(root);

printf("\n");

return 0;

}

先定义一个字符型数组tree[1000]，然后读取输入的字符串并赋值到tree[1000]中，调用 struct TreeNode\* root=BuildTree(tree,index);处理前序序列并生成二叉树，然后以此调用

midbianli(root);

lastbianli(root);

levelorderbianli(root);

来进行中序遍历、后序遍历和层次遍历。

## 调试分析：

在我的代码运算的时候曾出过一些问题，首先，当时我忘记在int main（）里面初始化index=0，导致程序在进行处理前序序列生成二叉树的时候发生了错误。

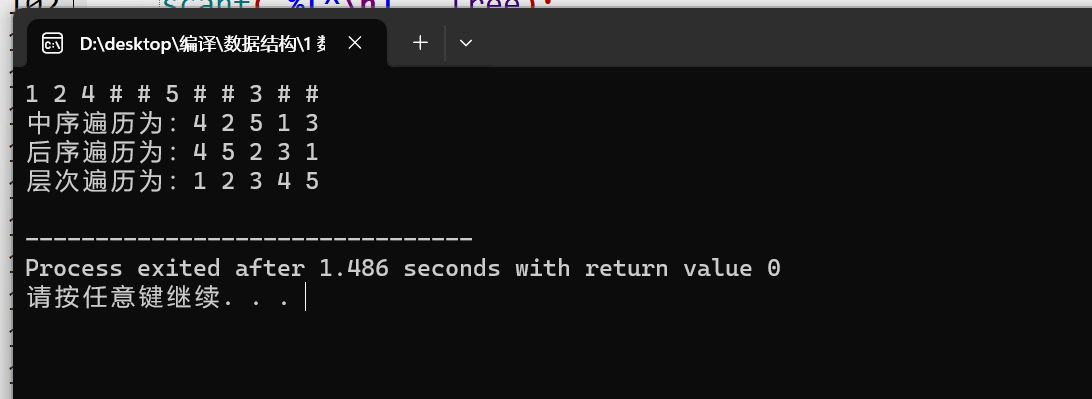
其次，在所有处理函数时，我最初都忘记了处理根节点为空的情况，造成情况考虑不够仔细和全面。

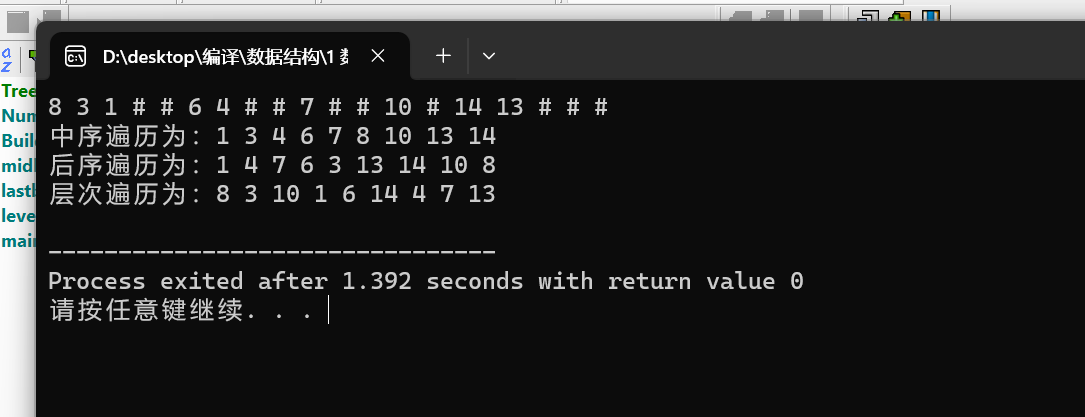
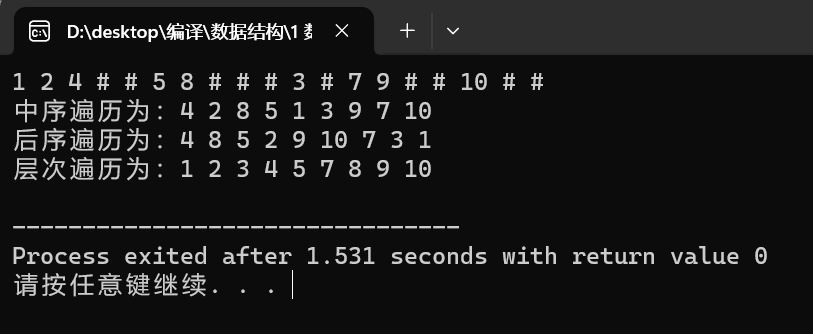
然后，在排版的时候我发现可以通过输入介绍来将最终的排版变得更加清晰易读。

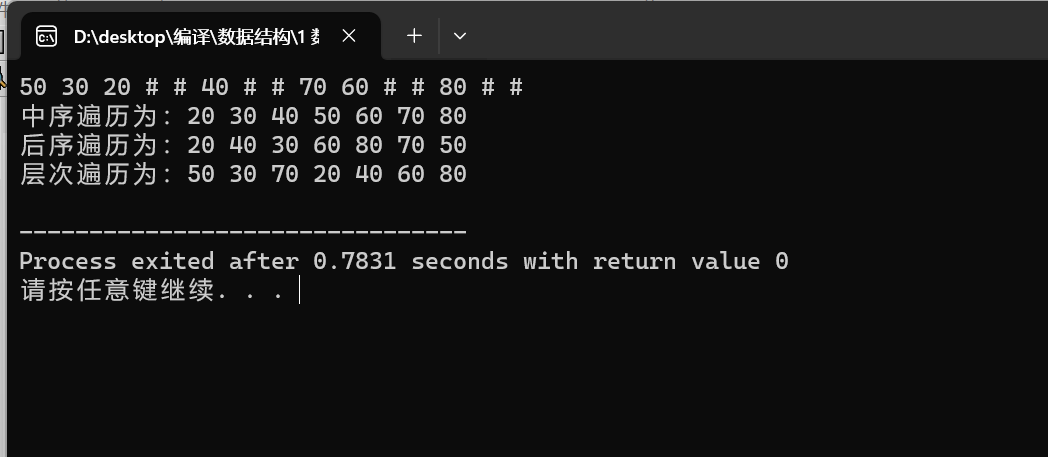
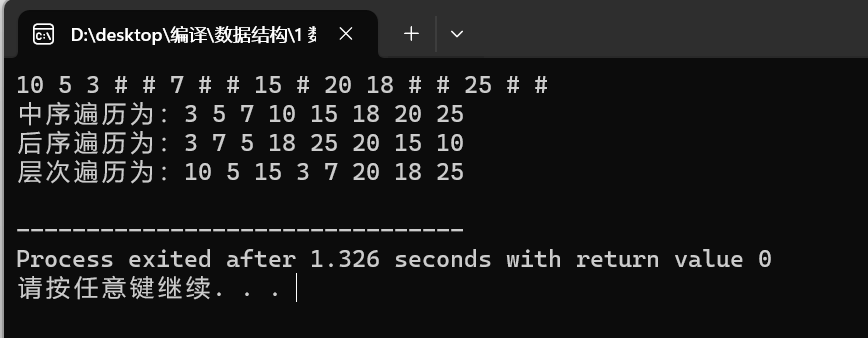
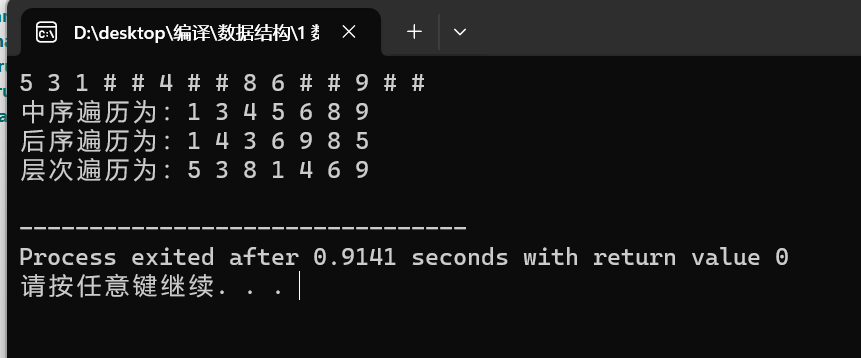
最后，在我生成的时候，我惊讶的发现由于我选择的是整形数据，如果我们读入的数据为10 5 3 # # 7 # # 15 # 20 18 # # 25 # #这种存在两位数的情况，那么系统可能会将它们看成两个不同的值，因此我又进行了处理。设置了number函数，并且优化算法，读入带有空格的值，然后筛选空格和‘#’，并且将多位数的处理同时兼具。

最终调试后，得出最终程序。

## 五、测试结果：







## 六、附录

#include <stdio.h>

#include <stdlib.h>

struct TreeNode{

int data;

struct TreeNode\* left;

struct TreeNode\* right;

};

int Number(char \*tree, int\* index) {

int number = 0;

// 当遇到数字字符时，将其转换为整数，并累加到number中

while (tree[\*index] >= '0' && tree[\*index] <= '9') {

number = number \* 10 + (tree[\*index] - '0');

(\*index)++;

}

// 当遇到非数字字符时，结束循环，返回number

return number;

}

struct TreeNode\* BuildTree(char\* tree, int\* index) {

if (tree[\*index] == '\0' || tree[\*index] == '#') {

(\*index)++;

return NULL;

}

// 读取数字

int num = Number(tree, index);

// 创建节点

struct TreeNode\* root = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

root->data = num;

// 跳过可能的空格

while (tree[\*index] == ' ') {

(\*index)++;

}

// 构建左子树

root->left = BuildTree(tree, index);

// 跳过可能的空格

while (tree[\*index] == ' ') {

(\*index)++;

}

// 构建右子树

root->right = BuildTree(tree, index);

// 跳过可能的空格

while (tree[\*index] == ' ') {

(\*index)++;

}

return root;

}

//中序遍历

void midbianli(struct TreeNode\* root){

if (root!=NULL){

midbianli(root->left);

printf("%d ",root->data);

midbianli(root->right);

}

}

//后序遍历

void lastbianli(struct TreeNode\* root){

if(root!=NULL){

lastbianli(root->left);

lastbianli(root->right);

printf("%d ",root->data);

}

}

//层次遍历

void levelorderbianli(struct TreeNode\* root){

if (root==NULL) return;

struct TreeNode\* queue[1000];

int a=0,b=0;

queue[b]=root;

b++;

while(a<b){

struct TreeNode\* temp=queue[a];

a++;

printf("%d ",temp->data);

if(temp->left!=NULL){

queue[b]=temp->left;

b++;

}

if(temp->right!=NULL){

queue[b]=temp->right;

b++;

}

}

}

int main(){

char tree[1000];

scanf("%[^\n]", tree);

int index=0;

//处理前序序列生成二叉树

struct TreeNode\* root=BuildTree(tree,&index);

//中序遍历

printf("中序遍历为：");

midbianli(root);

printf("\n");

//后序遍历

printf("后序遍历为：");

lastbianli(root);

printf("\n");

printf("层次遍历为：");

levelorderbianli(root);

printf("\n");

return 0;

}