# 北京邮电大学软件学院

# 2023-2024学年第1学期实验报告

**课程名称： 数据结构**

**实验名称：** 树及其应用

**实验完成人：**

**姓名：**\_\_\_\_\_禄禄鱼\_\_\_**学号：**\_\_\_\_\_\_\_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**（封面 5%）**

**日 期： 2023 年 11 月 6 日**

1. **实验目的**

（说明通过本实验希望达到的目的5%）

希望帮助学生熟悉各种存储结构的特征，以及如何应用树结构解决具体问题（即原理与应用的结合）。

* 1. **实验内容**

1. **实验内容**

（说明本实验的内容5%）

**必做内容**

* + 1. **二叉树的建立与遍历**

**[问题描述]**

　　建立一棵二叉树，并对其进行遍历（先序、中序、后序），打印输出遍历结果。

**[基本要求]**

　　从键盘接受输入（先序），以二叉链表作为存储结构，建立二叉树（以先序来建立），并采用递归算法对其进行遍历（先序、中序、后序），将遍历结果打印输出。

**[测试数据]**

ABCффDEфGффFффф（其中ф表示空格字符）则输出结果为

先序：ABCDEGF

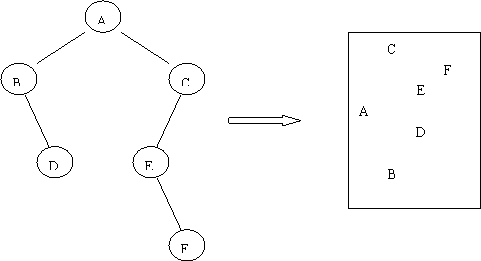
　　中序：CBEGDFA

　　后序：CGBFDBA

* + 1. **打印二叉树结构**

**[问题描述]**

　　按凹入表形式横向打印二叉树结构，即二叉树的根在屏幕的最左边，二叉树的左子树在屏幕的下边，二叉树的右子树在屏幕的上边。例如：



**[测试数据]**

　　由学生依据软件工程的测试技术自己确定。注意测试边界数据，如空二叉树。

**[实现提示]**

　　（1）利用RDL遍历方法；

（2）利用结点的深度控制横向位置。

1. **实验环境**

（说明本实验需要的环境5%）

Clion 2023

1. **实验过程和实验结果**

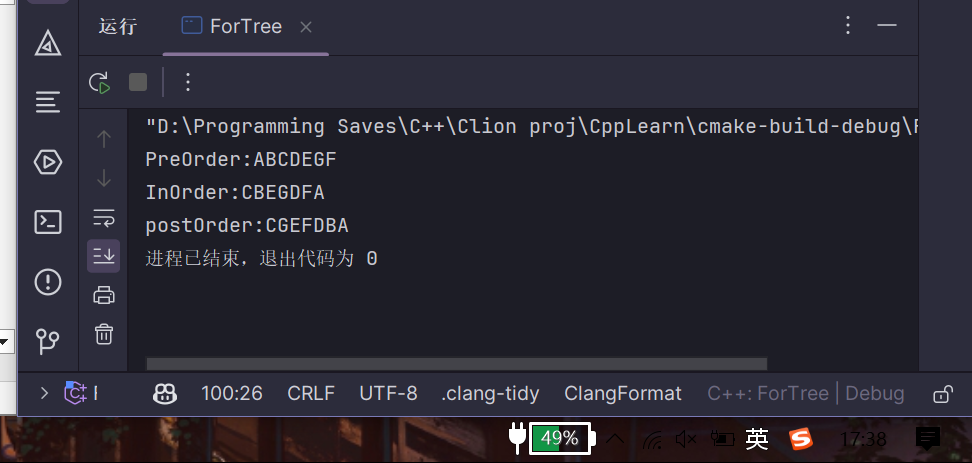
（问题分析，设计方案、算法、设计图、程序、调试过程截图、运行结果截图等70%）

实验1

代码

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
typedef struct node {  
 char data**;** struct node \*lchild**;** struct node \*rchild**;**} node**;**node\* createNode() {  
 node \*pNode = (node \*)malloc(sizeof(node))**;** pNode->lchild = NULL**;** pNode->rchild = NULL**;** return pNode**;**}  
node\* recurseBuild(char\* str**,**int\* index**,**int len)  
{  
 if(str[\*index] == ' ' || str[\*index] == '\0') {  
 return NULL**;** }  
 node\* pNode = createNode()**;** pNode->data = str[\*index]**;** (\*index)++**;** if(\*index == len) {  
 return pNode**;** }  
 pNode->lchild = recurseBuild(str**,**index**,**len)**;** (\*index)++**;** if(\*index == len) {  
 return pNode**;** }  
 pNode->rchild = recurseBuild(str**,** index**,**len)**;** return pNode**;**}  
// 从键盘持续读取字符,先序方式插入二叉树,读取到#或者null为止  
node\* buildTreeWithString(char\* str) {  
  
 if(\*str == ' ' || \*str == '\0') {  
 return NULL**;** }  
 int \*index = (int \*)malloc(sizeof(int))**;** (\*index)=0**;** int len = strlen(str)**;** node\* root = createNode()**;** root->data = str[\*index]**;** (\*index)++**;** root->lchild = recurseBuild(str**,**index**,**len)**;** (\*index)++**;** root->rchild = recurseBuild(str**,**index**,**len)**;** return root**;**}  
//前序遍历二叉树  
void preOrder(node\* root) {  
 if(root == NULL) {  
 return**;** }  
 printf("%c"**,**root->data)**;** preOrder(root->lchild)**;** preOrder(root->rchild)**;**}  
//中序遍历二叉树  
void inOrder(node\* root) {  
 if(root == NULL) {  
 return**;** }  
 inOrder(root->lchild)**;** printf("%c"**,**root->data)**;** inOrder(root->rchild)**;**}  
//后序遍历二叉树  
void postOrder(node\* root) {  
 if(root == NULL) {  
 return**;** }  
 postOrder(root->lchild)**;** postOrder(root->rchild)**;** printf("%c"**,**root->data)**;**}  
  
int main(){  
// char str[100];  
// scanf("%s",str);  
 char\* str\_test = "ABC DE G F "**;** node\* root = buildTreeWithString(str\_test)**;** //前序中序后序遍历  
 printf("PreOrder:")**;** preOrder(root)**;** printf("\n")**;** printf("InOrder:")**;** inOrder(root)**;** printf("\n")**;** printf("postOrder:")**;** postOrder(root)**;** return 0**;**}

运行结果



实验2

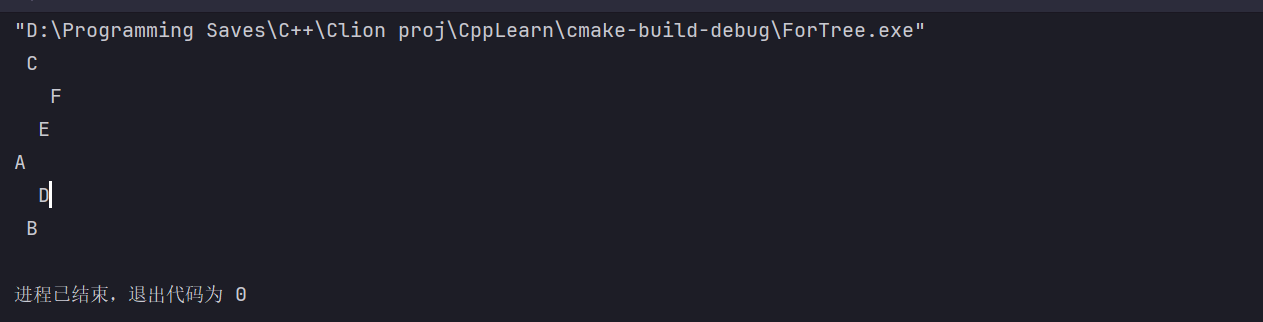
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
typedef struct node {  
 char data**;** struct node \*lchild**;** struct node \*rchild**;**} node**;**node\* createNode() {  
 node \*pNode = (node \*)malloc(sizeof(node))**;** pNode->lchild = NULL**;** pNode->rchild = NULL**;** return pNode**;**}  
node\* recurseBuild(char\* str**,**int\* index**,**int len)  
{  
 if(str[\*index] == ' ' || str[\*index] == '\0') {  
 return NULL**;** }  
 node\* pNode = createNode()**;** pNode->data = str[\*index]**;** (\*index)++**;** if(\*index == len) {  
 return pNode**;** }  
 pNode->lchild = recurseBuild(str**,**index**,**len)**;** (\*index)++**;** if(\*index == len) {  
 return pNode**;** }  
 pNode->rchild = recurseBuild(str**,** index**,**len)**;** return pNode**;**}  
// 从键盘持续读取字符,先序方式插入二叉树,读取到#或者null为止  
node\* buildTreeWithString(char\* str) {  
  
 if(\*str == ' ' || \*str == '\0') {  
 return NULL**;** }  
 int \*index = (int \*)malloc(sizeof(int))**;** (\*index)=0**;** int len = strlen(str)**;** node\* root = createNode()**;** root->data = str[\*index]**;** (\*index)++**;** root->lchild = recurseBuild(str**,**index**,**len)**;** (\*index)++**;** root->rchild = recurseBuild(str**,**index**,**len)**;** return root**;**}  
//前序遍历二叉树  
void preOrder(node\* root) {  
 if(root == NULL) {  
 return**;** }  
 printf("%c"**,**root->data)**;** preOrder(root->lchild)**;** preOrder(root->rchild)**;**}  
//中序遍历二叉树  
void inOrder(node\* root) {  
 if(root == NULL) {  
 return**;** }  
 inOrder(root->lchild)**;** printf("%c"**,**root->data)**;** inOrder(root->rchild)**;**}  
//后序遍历二叉树  
void postOrder(node\* root) {  
 if(root == NULL) {  
 return**;** }  
 postOrder(root->lchild)**;** postOrder(root->rchild)**;** printf("%c"**,**root->data)**;**}  
  
void printTreeWithDepth(node\* root**,**int\* depth)  
{  
 if(root == NULL) {  
// printf("\n");  
 return**;** }  
 (\*depth)++**;** printTreeWithDepth(root->rchild**,**depth)**;** for (int i = 0**;** i < (\*depth)**;** ++i) {  
 printf(" ")**;** }  
 printf("%c\n"**,**root->data)**;** printTreeWithDepth(root->lchild**,**depth)**;** (\*depth)--**;**}  
//按凹入表打印二叉树  
void printTree(node\* root)  
{  
 int\* depth = (int\*)malloc(sizeof(int))**;** (\*depth) = 0**;** if(root == NULL) {  
 printf("Tree is Empty")**;** return**;** }  
 printTreeWithDepth(root->rchild**,**depth)**;** printf("%c\n"**,**root->data)**;** printTreeWithDepth(root->lchild**,**depth)**;**}  
int main(){  
// char str[100];  
// scanf("%s",str);  
 char\* str\_test = "AB D CE F "**;** node\* root = buildTreeWithString(str\_test)**;**// //前序中序后序遍历  
// printf("PreOrder:");  
// preOrder(root);  
// printf("\n");  
// printf("InOrder:");  
// inOrder(root);  
// printf("\n");  
// printf("postOrder:");  
// postOrder(root);  
 //凹入表打印二叉树  
 printTree(root)**;** return 0**;**}

在实验1的基础上利用字符串创建二叉树,然后通过递归来输出(公用变量depth)

输入



输出



1. **实验心得**

（10%）

在进行数据结构实验，特别是关于二叉树的创建和遍历时，我深刻体会到树结构的优越性和灵活性。树的层次性使得数据的组织和检索变得更加高效，而不同遍历算法则展示了不同的应用场景。这个实验让我更好地理解了数据结构的基本概念，同时也培养了解决问题和思维抽象的能力。通过实际编程和调试，我感受到了编程技能的提升，同时也认识到了树结构在计算机科学中的广泛应用，为今后的学习和工作提供了坚实的基础。