** **

**课程实验报告**

课程名： 数据结构实验

学 院： 数学与计算机学院 系 计算机科学与技术系

专 业： 计算机科学与技术

班 级： 计算机222班

学 号： 5418122020

姓 名： 马星

任课教师： 任 燕

授课学期： 2022 年~~~~ 2023 年 1 学期

**目 录**

[实验一 二叉树实现](#_Toc31771)

[实验二 对称二叉树](#_Toc32327)

[实验三 和为target的路径](#_Toc2598)

**实验一二叉树**

学生姓名： 马星 学 号： 5418122020 专业班级： 计算机222班

实验类型：□ 验证 □ 综合 □ 设计 □ 创新 实验日期： 实验成绩：

一、实验项目名称

二、实验目的

三、实验基本原理

四、主要仪器设备及耗材

五、实验步骤

六、实验数据及处理结果

七、思考讨论题或体会或对改进实验的建议

八、参考资料

*// Author: 绝迹的星   
// Created on 2023/11/26  
  
  
#include* <stdio.h>  
*#include* <stdlib.h>  
*#include* <stdbool.h>  
*#include* "Status.c"  
  
*#define* STACK\_INIT\_SIZE 100  
*#define* STACKINCREMENT 10  
  
*typedef char* TElemType;  
*typedef struct* BiTNode {  
 TElemType data;  
 *struct* BiTNode \*left, \*right; *// 左右孩子指针*} BiTNode, \*BiTree;  
  
*typedef struct* {  
 BiTree \*top;  
 BiTree \*base;  
 *int* stacksize;  
} SqStack;  
  
  
Status InitStack(SqStack \*s) {  
 s->base = malloc(*sizeof*(BiTree) \* STACK\_INIT\_SIZE);  
 *if* (!s->base) exit(ERROR);  
 s->top = s->base;  
 s->stacksize = STACK\_INIT\_SIZE;  
 *return* OK;  
}  
  
  
bool StackIsEmpty(SqStack s) {  
 *return* s.base == s.top;  
}  
  
*int* StackLength(SqStack s) {  
 *return* s.top - s.base;  
}  
  
Status Push(SqStack \*s, BiTree e) {  
 *if* (s->stacksize == s->top - s->base) { *//栈满,追加存储空间* s->base = realloc(s->base, *sizeof*(*int*) \* (s->stacksize + STACKINCREMENT));  
 *if* (!s->base) exit(ERROR); *// 存储分配失败* s->top = s->base + STACKINCREMENT;  
 s->stacksize = s->stacksize + STACKINCREMENT;  
 }  
 \*s->top++ = e;  
 *return* OK;  
}  
  
Status Pop(SqStack \*s, BiTree \*e) {  
 *if* (StackLength(\*s) == 0) exit(ERROR);  
 \*e = \*--s->top;  
 *return* OK;  
}  
  
BiTree GetTop(SqStack s) {  
 *if* (s.base == s.top) exit(ERROR);  
 *return* \*(s.top - 1);  
}  
  
*//前序遍历,递归实现*Status PreOderTraverse(BiTree T, Status (\*visit)(TElemType e)) {  
 *if* (T) {  
 *if* (visit(T->data))  
 *if* (PreOderTraverse(T->left, visit))  
 *if* (PreOderTraverse(T->right, visit)) *return* OK;  
 *return* ERROR;  
 }  
 *return* OK;  
}  
  
*// 非递归*Status PreOderTraverse2(BiTree T, Status (\*visit)(TElemType e)) {  
 SqStack S;  
 InitStack(&S);  
 BiTree p = T;  
 *while* (p || !StackIsEmpty(S)) {  
 *if* (p) {  
 *if* (!visit(p->data)) exit(ERROR);  
 Push(&S, p);  
 p = p->left;  
 } *else* {  
 Pop(&S, &p);  
 p = p->right;  
 }  
 }  
 *return* OK;  
}  
  
*//中序*Status InOderTraverse(BiTree T, Status (\*visit)(TElemType e)) {  
 *if* (T) {  
 *if* (InOderTraverse(T->left, visit))  
 *if* (visit(T->data))  
 *if* (InOderTraverse(T->right, visit))  
 *return* OK;  
 *return* ERROR;  
 }  
 *return* OK;  
}  
  
*// 非递归*Status InOderTraverse2(BiTree T, Status (\*visit)(TElemType e)) {  
 SqStack S;  
 InitStack(&S);  
 BiTree p = T;  
 *while* (p || !StackIsEmpty(S)) {  
 *if* (p) {  
 Push(&S, p);  
 p = p->left;  
 } *else* {  
 Pop(&S, &p);  
 *if* (!visit(p->data)) exit(ERROR);  
 p = p->right;  
 }  
 }  
 *return* OK;  
}  
  
Status PostOderTraverse(BiTree T, Status (\*visit)(TElemType e)) {  
 *if* (T) {  
 *if* (PostOderTraverse(T->left, visit))  
 *if* (PostOderTraverse(T->right, visit))  
 *if* (visit(T->data))  
 *return* OK;  
 *return* ERROR;  
 }  
 *return* OK;  
}  
  
*// 非递归*Status PostOderTraverse2(BiTree T, Status (\*visit)(TElemType e)) {  
 SqStack S;  
 InitStack(&S);  
 BiTree p = T, record = 0;  
 *while* (p || !StackIsEmpty(S)) {  
 *if* (p) {  
 Push(&S, p);  
 p = p->left;  
 } *else* {  
 Pop(&S, &p);  
 *if* (p->right && p->right != record) {  
 Push(&S, p);  
 p = p->right;  
 } *else* {  
 *if* (!visit(p->data)) exit(ERROR);  
 record = p;  
 p = NULL;  
 }  
 }  
 }  
 *return* OK;  
}  
  
*//创建二叉树*Status CreateBiTree(BiTree \*T) {  
 *// 按先序次序输入二叉树中结点的值,(一个字符),空格字符表示空树  
 // 构造二叉树表表示的二叉树T  
 char* ch;  
 scanf("%c", &ch);  
 *if* (ch == ' ') {  
 \*T = NULL;  
 *return* OK;  
 }  
 *if* (!(\*T = malloc(*sizeof*(BiTNode)))) {  
 exit(ERROR);  
 }  
 (\*T)->data = ch; *//生成根节点* CreateBiTree(&(\*T)->left); *// 构造左子树* CreateBiTree(&(\*T)->right); *// 构造右子树  
 return* OK;  
}  
  
  
*/\*\*  
 \* 统计节点数量  
 \*/  
int* CountBiTNodes(BiTree T) {  
 *return* T ? CountBiTNodes(T->left) + CountBiTNodes(T->right) + 1 : 0;  
}  
  
*int* CountLeafNodes(BiTree T) {  
 *if* (!T) {  
 *return* 0;  
 }  
 *if* (!T->right && !T->left) {  
 *return* 1;  
 }  
 *return* CountLeafNodes(T->right) + CountLeafNodes(T->left);  
}  
  
*int* BiTreeDepth(BiTree T) {  
 *if* (T) {  
 *int* leftDepth = BiTreeDepth(T->left);  
 *int* rightDepth = BiTreeDepth(T->right);  
 *return* leftDepth > rightDepth ? leftDepth + 1 : rightDepth + 1;  
 }  
 *return* 0;  
}

**实验二 对称二叉树**

学生姓名： 马星 学 号： 5418122020 专业班级： 计算机222班

实验类型：□ 验证 □ 综合 □ 设计 □ 创新 实验日期： 实验成绩：

一、实验项目名称

二、实验目的

三、实验基本原理

四、主要仪器设备及耗材

五、实验步骤（完整内容见光盘）

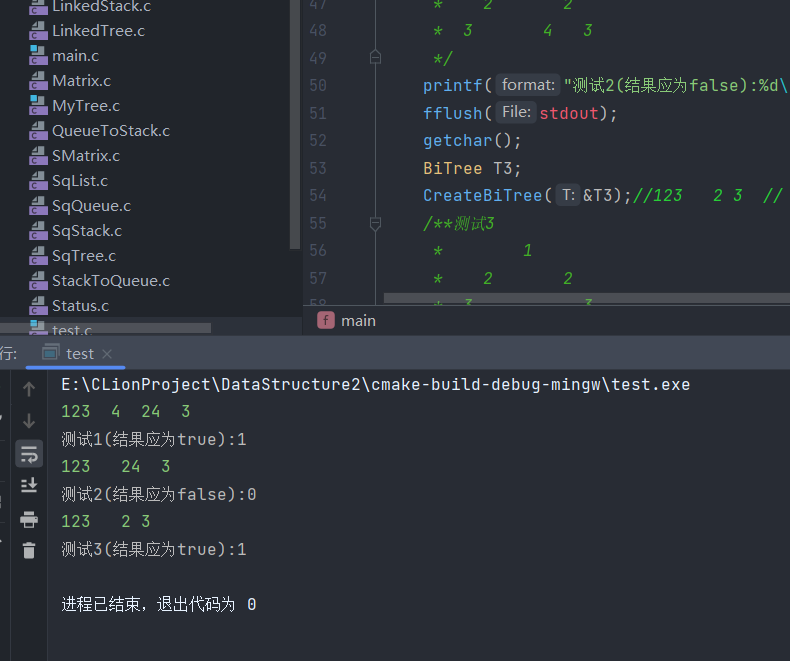
六、实验数据及处理结果

七、思考讨论题或体会或对改进实验的建议

八、参考资料

*#include* "MyTree.c"  
*#include* <stdio.h>  
*#include* <stdbool.h>  
*/\*\*  
 \* 判断两颗子树是否镜像相同  
 \* @param leftNode 左子树根节点  
 \* @param rightNode 右子树根节点  
 \*/*bool isSame(BiTNode \*leftNode, BiTNode \*rightNode) {  
 *if* (leftNode == NULL && rightNode == NULL) {  
 *return* true;  
 }  
 *if* (leftNode == NULL || rightNode == NULL) {  
 *return* false;  
 }  
 *if* (leftNode->data != rightNode->data) {  
 *return* false;  
 }  
 *// 左子树的左子树应该与右子树的右子树镜像相同  
 // 左子树的右子树应该与右子树的左子树镜像相同  
 return* isSame(leftNode->left, rightNode->right) && isSame(leftNode->right, rightNode->left);  
  
}  
*/\*\*  
 \* 判断树是否为对称二叉树  
 \* @param node 根节点  
 \*/*bool isSymmetry(BiTNode node) {  
 *return* isSame(node.left, node.right);  
}  
  
*int* main() {  
 BiTree T1;  
 CreateBiTree(&T1);*//123 4 24 3 //  
 /\*\*测试1  
 \* 1  
 \* 2 2  
 \* 3 4 4 3  
 \*/* printf("测试1(结果应为true):%d\n", isSymmetry(\*T1));  
 fflush(stdout);  
 getchar();  
 BiTree T2;  
 CreateBiTree(&T2);*//123 24 3 //  
 /\*\*测试2  
 \* 1  
 \* 2 2  
 \* 3 4 3  
 \*/* printf("测试2(结果应为false):%d\n", isSymmetry(\*T2));  
 fflush(stdout);  
 getchar();  
 BiTree T3;  
 CreateBiTree(&T3);*//123 2 3 //  
 /\*\*测试3  
 \* 1  
 \* 2 2  
 \* 3 3  
 \*/* printf("测试3(结果应为true):%d\n", isSymmetry(\*T3));  
 fflush(stdout);  
}

测试结果:



**实验三 打印和为target的路径**

学生姓名： 马星 学 号： 5418122020 专业班级： 计算机222班

实验类型：□ 验证 □ 综合 □ 设计 □ 创新 实验日期： 实验成绩：

一、实验项目名称

二、实验目的

三、实验基本原理

四、主要仪器设备及耗材

五、实验步骤（完整内容见光盘）

六、实验数据及处理结果

七、思考讨论题或体会或对改进实验的建议

八、参考资料

*// Author: 绝迹的星   
// Created on 2023/12/1  
  
#include* "MyTree.c"  
*#include* <stdio.h>  
  
*/\*\*  
 \* 打印栈中存储的路径  
 \* @param stack 栈  
 \*/  
void* printStack(SqStack stack) {  
 printf("path: ");  
 *int* len = StackLength(stack);  
 *for* (*int* i = 0; i < len; i++) {  
 printf("%c", stack.base[i]->data);  
 *if* (i != len - 1) {  
 printf("->");  
 }  
 }  
 printf("\n");  
  
}  
  
*/\*\*  
 \* 向左右搜索  
 \* @param node 当前节点  
 \* @param target 目标值  
 \* @param stack 用栈存储路径  
 \*/  
void* search(BiTNode \*node, *int* target, SqStack \*stack) {  
 *//当node为叶子节点,且target为0时说明找到路径  
 if* (node->left == NULL && node->right == NULL && target == 0) {  
 printStack(\*stack);  
 *return*;  
 }  
 *//如果node为叶子节点,而target不为0  
 //或者target小于等于0,而node不是叶子节点  
 //出栈回溯  
 if* (node->left == NULL && node->right == NULL || target <= 0) {  
 *return*;  
 }  
 BiTree \_;*//pop函数的占位参数  
 //搜索左边  
 if* (node->left != NULL) {  
 Push(stack, node->left);  
 search(node->left, target - node->left->data + '0', stack);  
 Pop(stack, &\_);*//回溯* }  
 *//搜索右边  
 if* (node->right != NULL) {  
 Push(stack, node->right);  
 search(node->right, target - node->right->data + '0', stack);  
 Pop(stack, &\_);*//回溯* }  
  
}  
  
*/\*\*  
 \* 打印和为target的路径  
 \* @param T 树  
 \* @param target 目标值   
 \*/  
void* printPath(BiTNode T, *int* target) {  
 SqStack stack;*//用栈存储路径* InitStack(&stack);  
 Push(&stack, &T);*//根节点必须入栈* search(&T, target - T.data + '0', &stack);*//搜索左右,+'0'是因为存储的数据为字符类型,要转为数字*}  
  
*int* main() {  
 BiTree T1;  
 CreateBiTree(&T1);*//123 4 32 3 //  
 /\*\*测试1  
 \* 1  
 \* 2 3  
 \* 3 4 2 3  
 \* target=7  
 \* ans:1->2->4 1->3->3  
 \*/* printPath(\*T1, 7);  
 fflush(stdout);  
 getchar();  
  
 BiTree T2;  
 CreateBiTree(&T2);*//12 3 1 3 //  
 /\*\*测试2  
 \* 1  
 \* 2 1  
 \* 3 3  
 \* target=6  
 \* ans:1->2->3  
 \*/* printPath(\*T2, 6);  
 fflush(stdout);  
 getchar();  
  
 BiTree T3;  
 CreateBiTree(&T3);*//12 1 3 //  
 /\*\*测试3  
 \* 1  
 \* 2 1  
 \* 3  
 \* target=3  
 \* ans:1->2  
 \*/* printPath(\*T3, 3);  
 fflush(stdout);  
 getchar();  
}

测试结果:

