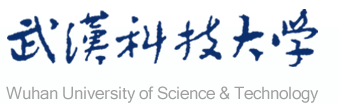
****

《数据结构》实验报告

**专 业：** 计算机科学与技术（卓越计划）

**班 级：** 计算机（卓越）1901

**学 号：** 201913137031

**姓 名：** 杨尔行

**实验3 二叉树的遍历\***

**一、【实验目的】**

1. 了解二叉树的前序、中序、后序和层次序列排列；
2. 将C语言同二叉树的数据结构联系起来；
3. 掌握生成的二叉树的链表结构；
4. 掌握如何按层次输出二叉树的所有结点；
5. 掌握如何将动态二叉树转换为静态二叉链表。

**二、【实验环境】**

**硬件**：Intel Core(TM) i5-8300H CPU 2.30GHZ

**软件**：Windows10 家庭中文版 64位操作系统；Visual Studio 2019 集成开发环境

**三、【实验内容】**

创建一个二叉树，对这棵动态二叉树进行分析，将其用静态二叉链表表示。二叉树的动态二叉链表结构中的每个结点有三个字段：data，lchild，rchild。静态二叉链表是用数组作为存储空间，每个数组元素存储二叉树的一个结点，也有三个字段：data，lchild，rchild。lchild和rdhild分别用于存储左右孩子的下标。

**四、【实验程序】**

1)程序源代码

**文件名：main.c**

#include "bintree.h"

#include <stdio.h>

void visit(BinTreeNode\* node)

{

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

}

int main()

{

// 1 4 2 \* \* 7 \* \* 3 \* 5 \* \*

printf("input a tree in preorder:");

BinTreeNode\* t = get\_bintree\_preorder(); // 先序输入二叉树

printf("\ntraverse in level-order:\n");

lvl\_trav(t, visit); // 层序遍历（输出）二叉树

SBinTree st = convert\_to\_static\_tree(t); // 转为静态二叉树

print\_static\_tree(&st);// 输出

free\_static\_tree(st);// 回收资源

free\_bintree(t);

}

**文件名：bintree.h**

#ifndef \_BINTREE\_H\_

#define \_BINTREE\_H\_

#include <stdlib.h>

typedef int data\_t;

#define data\_fmt "%d"

/// 二叉树节点

typedef struct BinTreeNode

{

/// 左孩子

struct BinTreeNode\* lchild;

/// 右孩子

struct BinTreeNode\* rchild;

/// 数据

data\_t data;

} BinTreeNode;

/// 静态二叉树节点

typedef struct SBinTreeNode

{

/// 左孩子

int lchild;

/// 右孩子

int rchild;

/// 数据

data\_t data;

}SBinTreeNode;

/// 静态二叉树

typedef struct SBinTree

{

/// 节点数组

SBinTreeNode\* base;

/// 大小

size\_t size;

}SBinTree;

/// 输入先序创建的二叉树

BinTreeNode\* get\_bintree\_preorder();

/// 回收二叉树

void free\_bintree(BinTreeNode\* tree);

/// 层序遍历

void lvl\_trav(BinTreeNode\* root, void (\*visit)(BinTreeNode\* node));

/// 转为静态二叉树

SBinTree convert\_to\_static\_tree(BinTreeNode\* root);

/// 回收静态二叉树

void free\_static\_tree(SBinTree st);

/// 输出静态二叉树

void print\_static\_tree(SBinTree\* tree);

#endif // \_BINTREE\_H\_

**文件名：bintree.cpp**

#include "bintree.h"

#include <queue>

#include <stdio.h>

#include <stdlib.h>

#include <assert.h>

#define NO\_DATA 1

#define HAS\_DATA 0

/// 从stdin获取数据，填入指针。返回是否有数据。

static int get\_data(data\_t\* data)

{

data\_t in\_data;

char in\_char;

int scan\_ret = scanf(data\_fmt, &in\_data);

if (scan\_ret == 1) { // 有效数据

\*data = in\_data;

return HAS\_DATA;

}

if (scanf("%c", &in\_char) == 1 && in\_char == '\*') {

return NO\_DATA;

}

printf("unexpected data\n");

abort();

}

/// 递归先序输入二叉树

void fwd\_get\_bintree\_rcrs(BinTreeNode\*\* root)

{

data\_t data;

if (get\_data(&data) == NO\_DATA) {

\*root = NULL;

}

else {

(\*root) = (BinTreeNode\*)malloc(sizeof(BinTreeNode));

(\*root)->data = data;

fwd\_get\_bintree\_rcrs(&((\*root)->lchild));

fwd\_get\_bintree\_rcrs(&((\*root)->rchild));

}

}

/// 先序输入二叉树

BinTreeNode\* get\_bintree\_preorder()

{

BinTreeNode\* tree = (BinTreeNode\*)malloc(sizeof(BinTreeNode));

fwd\_get\_bintree\_rcrs(&tree);

return tree;

}

/// 回收二叉树的内存

void free\_bintree(BinTreeNode\* tree)

{

if (!tree) return;

BinTreeNode\* l = tree->lchild, \* r = tree->rchild;

free(tree);

free\_bintree(l);

free\_bintree(r);

}

/// 层序遍历

void lvl\_trav(BinTreeNode\* root, void (\*visit)(BinTreeNode\*))

{

using std::queue;

if (!root) return;

queue<BinTreeNode\*> q;

q.push(root);

while (!q.empty()) { // 队列为空，遍历结束

BinTreeNode\* front\_node = q.front();

q.pop();

if (front\_node) {

visit(front\_node);

q.push(front\_node->lchild);

q.push(front\_node->rchild);

}

}

}

/// 递归转为静态树，返回节点在静态树中的位置

int convert\_to\_static\_tree\_impl(BinTreeNode\* root, SBinTree\* st)

{

if (!root) return -1;

SBinTreeNode\* end = st->base + st->size;

st->size += 1;

end->data = root->data;

end->lchild = convert\_to\_static\_tree\_impl(root->lchild, st);

end->rchild = convert\_to\_static\_tree\_impl(root->rchild, st);

return end - st->base;

}

/// 非递归外壳函数

SBinTree convert\_to\_static\_tree(BinTreeNode\* root)

{

SBinTree st;

st.base = (SBinTreeNode\*)malloc(sizeof(SBinTreeNode) \* 100);

st.size = 0;

convert\_to\_static\_tree\_impl(root, &st);

return st;

}

/// 输出静态树

void print\_static\_tree(SBinTree\* tree)

{

printf("\nstatic tree: ");

for (int i = 0; i < tree->size; i++) {

printf("\nindex = %2d lchild = %2d rchild = %2d data = %2d", i, tree->base[i].lchild, tree->base[i].rchild, tree->base[i].data);

}

}

/// 回收静态树

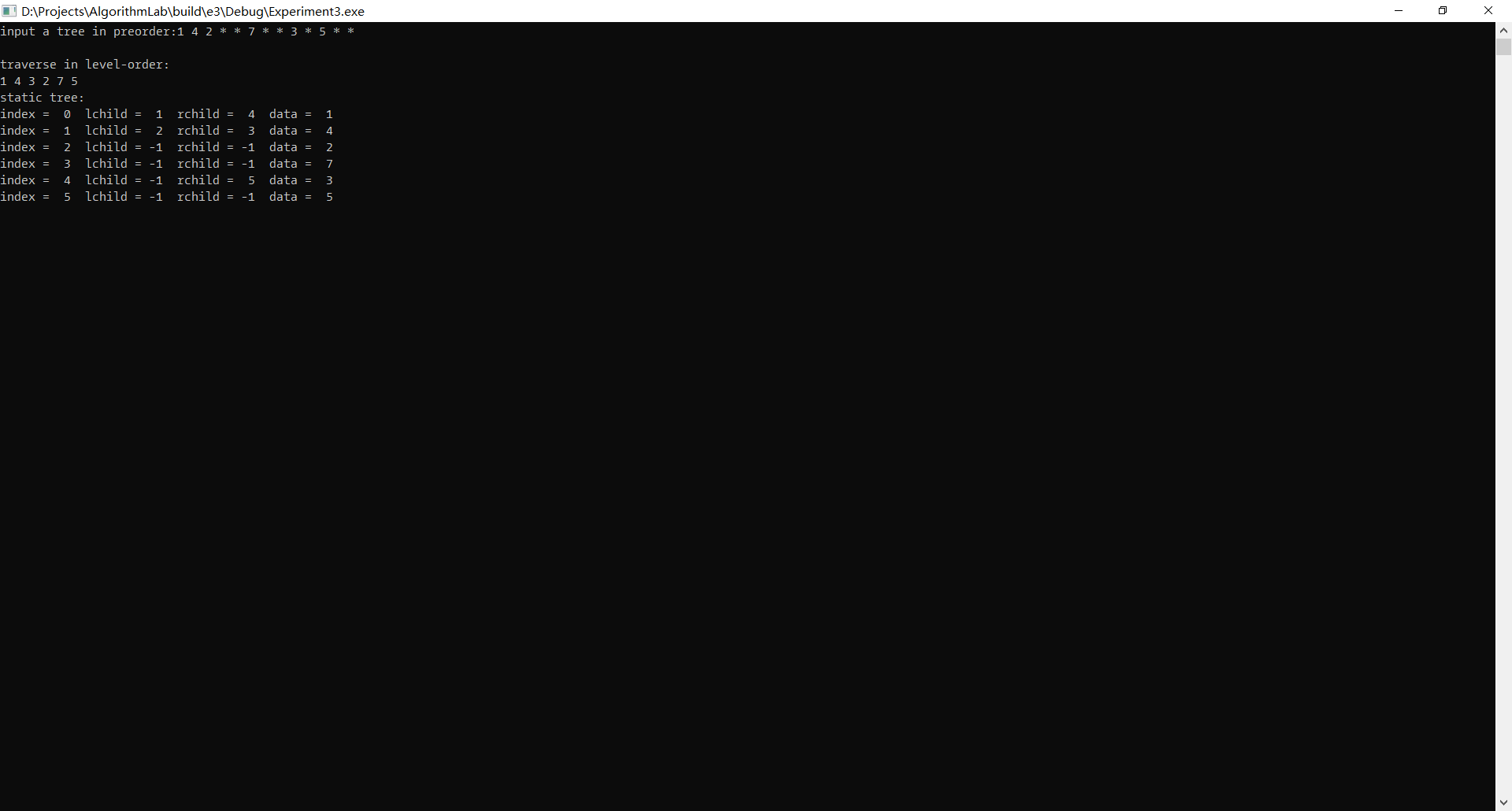
void free\_static\_tree(SBinTree st)

{

free(st.base);

}

2)程序运行结果截图



**六、【实验小结】**

本次实验使用了第6章线性表的知识，实现了二叉树的递归创建和层序遍历，并将二叉树存放到了静态链表中。层序遍历利用了队列先进先出的性质，将队首节点出队，同时又将其子节点入队，巧妙的节省了空间。创建和转换静态二叉树都采用了递归的算法，可见递归是二叉树相关算法的常用手段。