**重庆交通大学**

**《算法与数据结构》课程**

**实验报告**

**班 级： 2018级菁英班 曾凡深**

**实验项目名称： 数据结构实验五: 二叉树实验**

**实验项目性质： 技术实践**

**实验所属课程： 数据结构A**

**实验室(中心)： B01 407**

**指 导 教 师 ： 鲁云平**

**实验完成时间： 2019 年 10 月 11 日**

教师评阅意见：

签名： 年 月 日

实验成绩：

**一、实验目的**

1、实现二叉树的存储结构（二叉链表或三叉链表等存储结构任选）  
2、熟悉二叉树基本术语的含义  
3、掌握二叉树相关操作的具体实现方法

**二、实验内容及要求**

1. 建立二叉树

2. 计算结点所在的层次

3. 统计结点数量和叶结点数量

4. 计算二叉树的高度

5. 计算结点的度

6. 找结点的双亲和子女

7. 二叉树前序、中序、后序遍历的递归实现和非递归实现及层次遍历

8. 二叉树的复制

9. 二叉树的输出等

**三、系统分析**

（1）数据方面：结点结构体有T data 数据域，三个结点指针BTNode<T>

\*parent、\*leftChild、\*rightChild；二叉树类有一个根节点BTNode<T> \*root，一个数据输入停止标志T flag。

（2）功能方面：前序遍历创建二叉树（递归）、广义表建立二叉树（非递归）、前序遍历（递归）、中序遍历（递归）、后序遍历（递归）、前序遍历（非递归）、中序遍历（非递归）、

后序遍历（非递归）、层次序遍历（非递归）、销毁二叉树（递归）、计算结点所在的层次（递归）、结点数（递归）、叶结点数（递归）、树的高度（递归)、寻找结点（前序遍历（非递归）改）、复制二叉树（递归）、结点的度、获取双亲、获取左右子女、获取Flag、设置Flag等。

**四、系统设计**

（1）设计的主要思路

三叉链表存储表示，改进于二叉链表，增加指向父节点的指针，能更好地实现结点间的访问。

（2）数据结构的设计

1. 使用三叉链来实现二叉树。

2. 使用结构体封装结点。

3. 设计一个二叉树类，包含根节点和数据输入停止标志。

（3）基本操作的设计

BinaryTree(BinaryTree<T> &BT);//拷贝构造函数  
BinaryTree() : root(nullptr) {}//构造函数  
BinaryTree(T flag) : flag(flag), root(nullptr) {}//指定结束标志的构造函数  
~BinaryTree() { Destroy(root); } //析构函数  
  
void PreorderCreateBinaryTree(const T \*&str) { PreorderCreateBinaryTree(root, str); }  
  
void GeneralizedTableCreateBinaryTree(const T \*&str) { GeneralizedTableCreateBinaryTree(root, str); }  
  
void PreOrder() { PreOrder(root); }  
  
void InOrder() { InOrder(root); }  
  
void PostOrder() { PostOrder(root); }  
  
void NonRecursivePreOrder() { NonRecursivePreOrder(root); }  
  
void NonRecursiveInOrder() { NonRecursiveInOrder(root); }  
  
void NonRecursivePostOrder() { NonRecursivePostOrder(root); }  
  
void LevelOrder() { LevelOrder(root); }  
  
void Destroy() { Destroy(root); }  
  
unsigned int NodeLevel(const T &x) { return NodeLevel(root, x, 0); }  
  
unsigned int Size() { return Size(root); }  
  
unsigned int LeafSize() { return LeafSize(root); }  
  
unsigned int Height() { return Height(root); }  
  
unsigned int DegreeOfNode(const T &x); //结点的度  
  
T Parent(const T &x);  
  
T LeftChild(const T &x);  
  
T RightChild(const T &x);  
  
void CopyBinaryTree(BinaryTree<T> &BT);//复制二叉树  
  
T getFlag() const;  
  
void setFlag(T flag);  
  
friend ostream &operator<<(ostream &os, const BinaryTree &tree) {  
 os << tree.flag << " ";  
 string str;  
 if (tree.root != nullptr) {  
 stack<BTNode<T> \*> s;  
 s.push(tree.root);  
 BTNode<T> \*p;  
 while (!s.empty()) {  
 p = s.top();  
 s.pop();  
 str += p->data;  
 if (p->rightChild != nullptr)  
 s.push(p->rightChild);  
 else  
 str += tree.flag;  
 if (p->leftChild != nullptr)  
 s.push(p->leftChild);  
 else  
 str += tree.flag;  
 }  
 }  
 os << str << endl;  
 return os;  
}

**五、编程环境与实验步骤**

（1）编程环境

Windows规格：

版本 Windows10专业版

版本号 1903

操作系统版本 18362.295

编程工具软件：

CLion 2019.2

（2）实验步骤

打开CLion 2019.2，创建新项目（自动创建main.cpp），文件名: BinaryTree，新建.h文件，名为：BinaryTree.h，编写代码，最后到main.cpp编写测试代码。

（3）编译参数

"C:\Program Files\JetBrains\CLion 2019.1\bin\cmake\win\bin\cmake.exe" -DCMAKE\_BUILD\_TYPE=Debug -G "CodeBlocks - MinGW Makefiles" C:\Users\10944\Desktop\BinaryTree

-- The C compiler identification is GNU 7.3.0

-- The CXX compiler identification is GNU 7.3.0

-- Check for working C compiler: D:/Zfans/Zfans\_Software/Qt/Tools/mingw730\_64/bin/gcc.exe

-- Check for working C compiler: D:/Zfans/Zfans\_Software/Qt/Tools/mingw730\_64/bin/gcc.exe -- works

-- Detecting C compiler ABI info

-- Detecting C compiler ABI info - done

-- Detecting C compile features

-- Detecting C compile features - done

-- Check for working CXX compiler: D:/Zfans/Zfans\_Software/Qt/Tools/mingw730\_64/bin/g++.exe

-- Check for working CXX compiler: D:/Zfans/Zfans\_Software/Qt/Tools/mingw730\_64/bin/g++.exe -- works

-- Detecting CXX compiler ABI info

-- Detecting CXX compiler ABI info - done

-- Detecting CXX compile features

-- Detecting CXX compile features - done

-- Configuring done

-- Generating done

-- Build files have been written to: C:/Users/10944/Desktop/BinaryTree/cmake-build-debug

**六、实现代码**

**实现代码见附件。**

**七、测试结果与说明**

#include <iostream>  
#include "BinaryTree.h"  
  
using namespace std;  
  
int main() {  
 BinaryTree<char> bt01('#');  
 const char \*str01 = "ABC##DE##F##G#H##";  
 cout << "使用前序遍历创建二叉树（递归）str01 = \"ABC##DE##F##G#H##\"：";  
 bt01.PreorderCreateBinaryTree(str01);  
 cout << "\n前序遍历（递归）：";  
 bt01.PreOrder();  
 cout << "\n前序遍历（非递归）：";  
 bt01.NonRecursivePreOrder();  
 cout << "\n中序遍历（递归）：";  
 bt01.InOrder();  
 cout << "\n中序遍历（非递归）：";  
 bt01.NonRecursiveInOrder();  
 cout << "\n后序遍历（递归）：";  
 bt01.PostOrder();  
 cout << "\n后序遍历（非递归）：";  
 bt01.NonRecursivePostOrder();  
 cout << "\n层次序遍历（非递归）：";  
 bt01.LevelOrder();  
 const char \*str02 = "A(B(D,E(G,)),C(,F))#";  
 cout << "\n使用广义表建立二叉树（非递归）str02 = \"A(B(D,E(G,)),C(,F))#\"：";  
 bt01.GeneralizedTableCreateBinaryTree(str02);  
 cout << "\n A ";  
 cout << "\n B C ";  
 cout << "\n D E F ";  
 cout << "\n G ";  
 cout << "\n前序遍历（递归）：";  
 bt01.PreOrder();  
 cout << "\n前序遍历（非递归）：";  
 bt01.NonRecursivePreOrder();  
 cout << "\n中序遍历（递归）：";  
 bt01.InOrder();  
 cout << "\n中序遍历（非递归）：";  
 bt01.NonRecursiveInOrder();  
 cout << "\n后序遍历（递归）：";  
 bt01.PostOrder();  
 cout << "\n后序遍历（非递归）：";  
 bt01.NonRecursivePostOrder();  
 cout << "\n层次序遍历（非递归）：";  
 bt01.LevelOrder();  
 cout << "\n计算结点E所在层次：";  
 cout << bt01.NodeLevel('E');  
 cout << "\n节点数量：";  
 cout << bt01.Size();  
 cout << "\n叶结点数量：";  
 cout << bt01.LeafSize();  
 cout << "\n二叉树的高度：";  
 cout << bt01.Height();  
 cout << "\n计算结点B的度：";  
 cout << bt01.DegreeOfNode('B');  
 cout << "\n计算结点C的度：";  
 cout << bt01.DegreeOfNode('C');  
 cout << "\n结点D的双亲：";  
 cout << bt01.Parent('D');  
 cout << "\n结点E的子女：";  
 if (bt01.LeftChild('E'))  
 cout << bt01.LeftChild('E');  
 if (bt01.RightChild('E'))  
 cout << bt01.RightChild('E');  
 cout << "\n拷贝构造（bt02拷贝bt01）：";  
 BinaryTree<char> bt02(bt01);  
 cout << "\n层次序遍历bt02（非递归）：";  
 bt02.LevelOrder();  
 BinaryTree<char> bt03;  
 cout << "\nbt03复制bt02：";  
 bt03.CopyBinaryTree(bt02);  
 cout << "\n层次序遍历bt03（非递归）：";  
 bt03.LevelOrder();  
 cout << "\n二叉树输出（bt01）：";  
 cout << bt01;  
 return 0;  
}

**由于个人精力有限，测试代码仅70行左右，仔细测试了BinaryTree中的所有方法，均未发现问题。**

**八、实验分析**

（1）算法的性能分析

算法的时间复杂度为O（n）；

1. 数据结构的分析

优点：查找，插入，删除都快（如果数保持平衡）。

缺点：删除算法复杂。

适用场景：数据是随机分布的、数据量较大、频繁的查找和插入操作（可以提供O(log n)级的查找、插入和删除操作）。

**九、实验总结**

主要针对本实验的分析、设计、实现、测试等环节进行总结，包含收获与不足，此部分的阐述应较为详细。

实验目的是实现二叉树的存储结构，通过实验深入理解二叉树的操作特点，索性用三叉链实现了一个二叉树BinaryTree，定义了结点结构体有T data 数据域，三个结点指针BTNode<T> \*parent、\*leftChild、\*rightChild；二叉树类有一个根节点BTNode<T> \*root，一个数据输入停止标志T flag。

二叉树的基本操作有前序遍历创建二叉树（递归）、广义表建立二叉树（非递归）、前序遍历（递归）、中序遍历（递归）、后序遍历（递归）、前序遍历（非递归）、中序遍历（非递归）、后序遍历（非递归）、层次序遍历（非递归）、销毁二叉树（递归）、计算结点所在的层次（递归）、结点数（递归）、叶结点数（递归）、树的高度（递归)、寻找结点（前序遍历（非递归）改）、复制二叉树（递归）、结点的度、获取双亲、获取左右子女、获取Flag、设置Flag等，广泛的使用到了递归的方法，实现起来比较绕脑，想得深了就会迷失在其中，改用非递归方式就会用到队列和栈，实现思路也比较绕脑，需要静下心来仔细推敲（实现代码见附件）。

由于个人精力有限，测试代码仅70行左右，仔细测试了BinaryTree中的所有方法，均未发现问题。

本次实验的收获是掌握了用三叉链实现二叉树的存储结构的技巧，熟悉了二叉树基本术语的含义，掌握了二叉树相关操作的具体实现方法。

本次实验的不足是没有将代码优化到最简。