《数据结构》实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 马昆 | 班级 | 软件工程专升本1班 | 学号 | 2206831544 |
| 实验名称 | **实验五 树和二叉树** | | | | |
| 实验时间 | 2023年4月27日 | | 成绩 | |  |
| 1. **实验目的：**   1.掌握二叉树的逻辑结构；  2.掌握二叉树的建立方法；  3.掌握二叉树遍历算法的应用。  **二、实验工具：**  软件平台：Windows7或以上版本，Visual Studio 2019  **三、实验原理：**  1、算法描述：  （1）建立二叉树的函数：使用根左右的方式迭代创建二叉树，无形参，返回创建的树结点的指针。创建一个char类型的变量用于接收用户输入，如果输入的是#号，就返回空指针，作为迭代退出条件。如果用户输入不为#号，就创建一个树结点，并将它的数据赋值为用户输入的数据，然后再将它的左孩子赋值为调用本函数的返回值，右孩子也赋值为本函数的返回值。然后返回这个创建的树结点。  （2）先序遍历函数：采用迭代的思想，形参为树结点指针，无返回值。如果根节点为空直接结束函数，如果不为空就先打印根节点数据，然后再调用本身，传入根结点的左孩子指针。再次调用本身，传入根节点的右孩子指针。  （3）中序遍历函数：与前序遍历函数的实现一样，只是交换了自身调用的顺序。采用迭代的思想，形参为树结点指针，无返回值。如果根节点为空直接结束函数，如果不为空就先调用本身，传入根结点的左孩子指针。然后打印根节点数据。再次调用本身，传入根节点的右孩子指针。  （4）后序遍历函数：与前序、中序遍历函数的实现一样，只是交换了自身调用的顺序。采用迭代的思想，形参为树结点指针，无返回值。如果根节点为空直接结束函数，如果不为空就先调用本身，传入根结点的左孩子指针。再次调用本身，传入根节点的右孩子指针。然后打印根节点数据。  **四、实验步骤和内容：**  1. 建立一棵二叉树，树的形态自定，完成以下操作：  （1）输出该二叉树的前序遍历序列；  （2）输出该二叉树的中序遍历序列；  （3）输出该二叉树的后序遍历序列。  表格 程序及运行结果   |  | | --- | | **BiTree.h**  #pragma once  #include <iostream>  using namespace std;  // 二叉树的结点  template <typename DataType>  struct BiNode  {  DataType data; // 结点数据  BiNode<DataType>\* left\_child, \* right\_child; // 结点的左孩子，右孩子  };  template <typename DataType>  class BiTree  {  public:  BiTree() { root = Create(); } // 构造函数，建立一颗二叉树  ~BiTree() { Release(root); } // 析构函数，释放各节点的存储空间  void PreOrder() { PreOrder(root); } // 前序遍历二叉树  void InOrder() { InOrder(root); }; // 中序遍历二叉树  void PostOrder() { PostOrder(root); } // 后序遍历二叉树  void LevelOrder(); // 层序遍历二叉树  private:  BiNode<DataType>\* Create(); // 构造函数调用  void Release(BiNode<DataType> \* bt); // 析构函数调用  void PreOrder(BiNode<DataType> \* bt); // 前序遍历函数调用  void InOrder(BiNode<DataType> \* bt); // 中序遍历函数调用  void PostOrder(BiNode<DataType> \* bt); // 后序遍历函数调用  BiNode<DataType>\* root; // 指向根节点的头指针  };  // 前序遍历函数调用  template <typename DataType>  void BiTree<DataType>::PreOrder(BiNode<DataType>\* bt)  {  if (bt == nullptr)  {  return;  }  else  {  cout << bt->data << "\t";  PreOrder(bt->left\_child);  PreOrder(bt->right\_child);  }  }  // 中序遍历函数调用  template <typename DataType>  void BiTree<DataType>::InOrder(BiNode<DataType>\* bt)  {  if (bt == nullptr)  {  return;  }  else  {  InOrder(bt->left\_child);  cout << bt->data << "\t";  InOrder(bt->right\_child);  }  }  // 后序遍历函数调用  template <typename DataType>  void BiTree<DataType>::PostOrder(BiNode<DataType>\* bt)  {  if (bt == nullptr)  {  return;  }  else  {  PostOrder(bt->left\_child);  PostOrder(bt->right\_child);  cout << bt->data << "\t";  }  }  // 层序遍历二叉树  template <typename DataType>  void BiTree<DataType>::LevelOrder()  {  BiNode<DataType>\* Q[100], \* q = nullptr;  int front = -1, rear = -1;  if (root == nullptr) return;  Q[++rear] = root;  while (front != rear)  {  q = Q[++front];  cout << q->data << "\t";  if (q->left\_child != nullptr) Q[++rear] = q->left\_child;  if (q->right\_child != nullptr) Q[++rear] = q->right\_child;  }  }  // 建立二叉树，构造函数调用  template <typename DataType>  BiNode<DataType>\* BiTree<DataType>::Create()  {  char ch;  cin >> ch;  if (ch == '#')  {  return nullptr;  }  else  {  BiNode<DataType>\* bt;  bt = new BiNode<DataType>;  bt->data = ch;  bt->left\_child = Create();  bt->right\_child = Create();  return bt;  }  }  // 释放各节点的存储空间，析构函数调用  template <typename DataType>  void BiTree<DataType>::Release(BiNode<DataType>\* bt)  {  if (bt == nullptr) return;  else  {  Release(bt->left\_child);  Release(bt->right\_child);  delete bt;  }  } | | **Main.cpp**  #include "BiTree.h"  int main()  {  BiTree<char>T{};  cout << "\n该二叉树的前序遍历序列是：" << endl;  T.PreOrder();  cout << "\n该二叉树的中序遍历序列是：" << endl;  T.InOrder();  cout << "\n该二叉树的后序遍历序列是：" << endl;  T.PostOrder();  cout << "\n该二叉树的层序遍历序列是：" << endl;  T.LevelOrder();  return 0;  } | | **运行结果 案例1**  假设树的结构如下：      **运行结果 案例2** |   **五、实验总结：**  通过本次实验，学习了二叉树的链式存储的实现，并完成了相关代码的编写，理解了二叉树前序、中序、后序以及层序的遍历思想以及实现。但是代码实现都是按照书上案例编写的，自己实现起来还是稍微有一些问题。而且大脑模拟对二叉树的几种遍历时，总是有些混乱，没有找到合适的方法。  **六、教师评语：** | | | | | |