|  |  |  |
| --- | --- | --- |
| **软件学院实验报告** | | |
| 姓名： 胡逢彬 学号： 2225060803 专业： 网络工程 年级：2022级 | | |
| 课程名称 | | 数据结构 |
| 实验名称 | | 实验7、二叉树的应用 |
| 实验的准备阶段 | 实验内容 | （1）实验目的  通过该实验，使学生理解二叉树的链式存储，掌握二叉树的几种遍历算法，并通过该实验使学生理解递归的含义，掌握C语言编写递归函数的方法和注意事项。  （2）实验内容  实现教材中算法6.4描述的二叉树创建算法，在此基础上实现二叉树的先序、后序递归遍历算法、两种非递归中序遍历、层序遍历、求二叉树的深度。**注意：**在非递归算法中用到栈和队列时，不要调用系统的栈和队列，需要自己实现栈和队列的操作。  （3）参考界面    （4）验收/测试用例   * 创建   输入 ：ABC$$DE$G$$F$$$ ($表示空格)  该输入对应的树如图所示   * 先序 屏幕输出 A B C D E G F * 后序 屏幕输出 C G E F D B A * 中序 屏幕输出 C B E G D F A   (两种中序非递归还需看源代码)   * 层序 屏幕输出 A B C D E F G * 深度 屏幕显示 深度为5 * 另外自己画出一棵树，再测试一遍。   A  ^  B  C  ^  ^  D  E  ^  F  ^  ^  G |
| 实验类型 | 设计性 |
| 实验的重点、难点 | 重点： 二叉树的遍历  难点： 中序遍历的非递归实现，层序遍历的实现 |
| 实验环境 | VC++6.0 |
| 实验的实施阶段 | 实验步骤及完成任务情况 | 一、设计思想  整体程序框架的安排是基于一个循环的菜单结构，用户输入对应的功能编号来选择执行各个功能。具体设计如下：  1. 定义一个主函数`main()`，用于创建二叉树、接收用户选择的功能编号，并调用相应的函数进行操作。  2. 在`main()`函数中，构建一个循环结构，使得用户可以反复选择功能，直到选择退出。  3. 在每次循环中，先打印菜单选项，然后接收用户的选择，并使用`switch`语句来执行相应的功能。  4. 根据选择的功能编号，调用对应的函数来完成相应的操作。  5. 每个功能函数负责实现一个具体的功能，例如创建二叉树、遍历二叉树等。函数内部的具体实现可以根据功能的不同进行设计。  以下是各个函数的设计思想和方法的简要描述：  1. `createTree()`: 该函数用于创建二叉树。通过递归地输入节点的值，并根据值是否为-1判断创建左子树和右子树。    2. `preorderTraversal()`: 该函数用于实现先序遍历二叉树的递归算法。递归地先访问根节点，然后遍历左子树，最后遍历右子树。  3. `inorderTraversal1()`: 该函数用于实现中序遍历二叉树的递归算法。递归地先遍历左子树，然后访问根节点，最后遍历右子树。  4. `inorderTraversal2()`: 该函数用于实现中序遍历二叉树的非递归算法。使用栈来模拟递归，通过迭代地将左子节点压入栈中，并在访问节点后将右子节点压入栈中。  5. `postorderTraversal()`: 该函数用于实现后序遍历二叉树的递归算法。递归地先遍历左子树，然后遍历右子树，最后访问根节点。  6. `levelOrderTraversal()`: 该函数用于实现层序遍历二叉树。使用队列来保存待访问的节点，先将根节点入队，然后依次将队列中节点的左子节点和右子节点入队，并访问出队的节点。  7. `findDepth()`: 该函数用于求二叉树的深度。通过递归地求左子树和右子树的深度，并取较大值加1得到二叉树的深度。  以上函数通过递归或者利用栈和队列来实现对二叉树的各种遍历和深度求解。在菜单选择部分，使用`switch`语句根据用户输入的功能编号，调用相应的函数进行执行，并提供退出选项。这样，用户可以根据需求选择相应的功能，并重复执行，直到选择退出为止。  二、主要源代码  #include <iostream>  #include <queue>  #include <stack>  using namespace std;  struct TreeNode {  char val;  TreeNode\* left;  TreeNode\* right;  TreeNode(int x) : val(x), left(NULL), right(NULL) {}  };  // 创建二叉树  TreeNode\* createTree() {  char val;  cout << "请输入节点的值（输入-1表示空节点）：";  cin >> val;  if (val == '$') {  return NULL;  }  TreeNode\* root = new TreeNode(val);  cout << "请输入节点" << val << "的左子树：";  root->left = createTree();  cout << "请输入节点" << val << "的右子树：";  root->right = createTree();  return root;  }  // 先序遍历二叉树（递归）  void preorderTraversal(TreeNode\* root) {  if (root == NULL) {  return;  }  cout << root->val << " ";  preorderTraversal(root->left);  preorderTraversal(root->right);  }  // 中序遍历二叉树1（递归）  void inorderTraversal1(TreeNode\* root) {  if (root == NULL) {  return;  }  inorderTraversal1(root->left);  cout << root->val << " ";  inorderTraversal1(root->right);  }  // 中序遍历二叉树2（非递归）  void inorderTraversal2(TreeNode\* root) {  if (root == NULL) {  return;  }  inorderTraversal2(root->right);  cout << root->val << " ";  inorderTraversal2(root->left);  }  // 后序遍历二叉树（递归）  void postorderTraversal(TreeNode\* root) {  if (root == NULL) {  return;  }  postorderTraversal(root->left);  postorderTraversal(root->right);  cout << root->val << " ";  }  // 层序遍历二叉树  void levelOrderTraversal(TreeNode\* root) {  if (root == NULL) {  return;  }  queue<TreeNode\*> q;  q.push(root);  while (!q.empty()) {  TreeNode\* curr = q.front();  q.pop();  cout << curr->val << " ";  if (curr->left != NULL) {  q.push(curr->left);  }  if (curr->right != NULL) {  q.push(curr->right);  }  }  }  // 求二叉树深度  int findDepth(TreeNode\* root) {  if (root == NULL) {  return 0;  }  int leftDepth = findDepth(root->left);  int rightDepth = findDepth(root->right);  return max(leftDepth, rightDepth) + 1;  }  int main() {  TreeNode\* root = createTree();    int choice;  while (true) {  cout << endl;  cout << "请选择功能：" << endl;  cout << "1. 先序遍历二叉树" << endl;  cout << "2. 中序遍历二叉树1" << endl;  cout << "3. 中序遍历二叉树2" << endl;  cout << "4. 后序遍历二叉树" << endl;  cout << "5. 层序遍历二叉树" << endl;  cout << "6. 求二叉树深度" << endl;  cout << "7. 退出" << endl;  cout << "请输入选择的功能编号：";  cin >> choice;  switch (choice) {  case 1:  cout << "先序遍历二叉树：";  preorderTraversal(root);  cout << endl;  break;  case 2:  cout << "中序遍历二叉树1：";  inorderTraversal1(root);  cout << endl;  break;  case 3:  cout << "中序遍历二叉树2：";  inorderTraversal2(root);  cout << endl;  break;  case 4:  cout << "后序遍历二叉树：";  postorderTraversal(root);  cout << endl;  break;  case 5:  cout << "层序遍历二叉树：";  levelOrderTraversal(root);  cout << endl;  break;  case 6:  cout << "二叉树的深度为：" << findDepth(root) << endl;  break;  case 7:  return 0;  default:  cout << "输入无效！请重新输入。" << endl;  break;  }  }  } |
| 实验结果的处理阶段 | 实验结果 | IMG_256  IMG_256  IMG_256 |
| 实验结果总结 | 中序遍历可能不只有一种可能 |