山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号： | 姓名： | | 班级： |
| 实验题目：二叉树操作 | | | |
| 实验学时：4小时 | | 实验日期：2018.11.12 | |
| 实验目的：  掌握二叉树的基本概念，二叉树的存储结构使用链表。 | | | |
| 硬件环境： 笔记本电脑 | | | |
| 软件环境：Win10+Vistual Studio 2017 | | | |
| 实验步骤与内容：  实验内容：  1、输入一个完全二叉树的层次遍历字符串，创建这个二叉树，输出这个二叉树的前序遍历字符串、中序遍历字符串、后序遍历字符串、结点数目、二叉树高度(上述每一个结果独立一行显示)。  2、输入二叉树前序序列和中序序列(各元素各不相同)，创建这个二叉树，输出该二叉树的后序序列、层次遍历。  对逗号的判别另一种方法：  void PreOrder(BinaryDTreeNode<T> \*t) //前序遍历 { if (t) { visit(t); if (t->LChild != NULL) cout << ","; PreOrder(t->LChild); if (t->RChild != NULL) cout << ","; PreOrder(t->RChild); } } void InOrder(BinaryDTreeNode<T> \*t) //中序遍历 { if (t != NULL) { InOrder(t->LChild); if (t->LChild != NULL) cout << ","; visit(t); if (t->RChild != NULL) cout << ","; InOrder(t->RChild); } }  实验步骤：  binaryTree.h  #pragma once  #include <iostream>  #include <string>  #include "binaryTreeNode.h"  using namespace std;  static string outputString = "";  class binaryTree  {  friend class binaryTree;  public:  binaryTree() { root = NULL; }  ~binaryTree() {};  bool empty() const { return treeSize == 0; }  int size(); //返回树中的节点数  int getSize(binaryTreeNode \*p);  //bool rootElement(int &x);  int\* rootElement() const;  void makeTree(const int &element, binaryTree &leftChild, binaryTree &rightChild);  void makeTree(binaryTreeNode \*r) { root = r; }  void preOrder(void(\*visit)(binaryTreeNode \*p)) { preOrder(visit, root); }  void inOrder(void(\*visit)(binaryTreeNode \*p)) { inOrder(visit, root); }  void postOrder(void(\*visit)(binaryTreeNode \*p)) { postOrder(visit, root); }  void levelOrder(void(\*visit)(binaryTreeNode \*p)) { levelOrder(visit, root); };  //二叉树类的扩充  void addNode(const int &u); //删除二叉树并释放其节点  int height(); //返回树的高度  int treeHeight(binaryTreeNode \*root);  //binaryTreeNode \*getRoot(); //获得根节点  void preOrderOutput() { outputString = ""; preOrder(output); outputString.erase(outputString.end() - 1); cout << outputString << endl;}  void inOrderOutput() { outputString = ""; inOrder(output); outputString.erase(outputString.end() - 1); cout << outputString << endl;}  void postOrderOutput() { outputString = ""; postOrder(output); outputString.erase(outputString.end() - 1); cout << outputString << endl;}  void levelOrderOutput() { outputString = ""; levelOrder(output); outputString.erase(outputString.end() - 1); cout << outputString << endl;}  static void output(binaryTreeNode \*t)  {  outputString += t->element;  outputString += ",";  }  private:  int treeSize;  binaryTreeNode \*root;  void preOrder(void(\*visit)(binaryTreeNode \*p), binaryTreeNode \*t);  void inOrder(void(\*visit)(binaryTreeNode \*p), binaryTreeNode \*t);  void postOrder(void(\*visit)(binaryTreeNode \*p), binaryTreeNode \*t);  void levelOrder(void(\*visit)(binaryTreeNode \*p), binaryTreeNode \*t);  };  binaryTreeNode.h  #pragma once  #include <iostream>  struct binaryTreeNode  {  public:  int element;  binaryTreeNode \*leftChild; //左子树  binaryTreeNode \*rightChild; //右子树  binaryTreeNode() { leftChild = rightChild = NULL; }  binaryTreeNode(const int& theElement)  {  element = theElement;  leftChild = rightChild = 0;  }  binaryTreeNode(const int& theElement, binaryTreeNode \*theleftChild, binaryTreeNode \*therightChild)  {  element = theElement;  leftChild = theleftChild;  rightChild = therightChild;  }  };  binaryTree.cpp  #include <iostream>  #include <string>  #include <queue>  #include <stack>  #include "binaryTree.h"  using namespace std;  constexpr auto MAX = 100;//前中序输入的最大数目,本来是#define MAX定义一个宏，后来VS提示改成这个就这样了  int\* binaryTree::rootElement() const  {// Return NULL if no root. Otherwise, return pointer to root element.  //返回NULL，如果是空树。否则，返回根节点的element域  if (treeSize == 0)  return NULL; // no root  else  return &root->element;  }  //将leftChild，rightChild，element合并成一颗新树  //leftChild、rightChild和this必须是不同的树  void binaryTree::makeTree(const int &element, binaryTree &leftChild, binaryTree &rightChild)  {  root = new binaryTreeNode(element, leftChild.root, rightChild.root);  leftChild.root = rightChild.root = 0; //禁止通过其他途径访问leftChild和rightChild  }  void binaryTree::preOrder(void(\*visit)(binaryTreeNode \*p), binaryTreeNode \*t)  {  if (t)  {  visit(t);  preOrder(visit, t->leftChild);  preOrder(visit, t->rightChild);  }  }  void binaryTree::inOrder(void(\*visit)(binaryTreeNode \*p), binaryTreeNode \*t)  {  if (t)  {  inOrder(visit, t->leftChild);  visit(t);  inOrder(visit, t->rightChild);  }  }  void binaryTree::postOrder(void(\*visit)(binaryTreeNode \*p), binaryTreeNode \*t)  {  if (t)  {  postOrder(visit, t->leftChild);  postOrder(visit, t->rightChild);  visit(t);  }  }  void binaryTree::levelOrder(void(\*visit)(binaryTreeNode \*p), binaryTreeNode \*t)  {  queue<binaryTreeNode\*>myQueue;  while (t)  {  visit(t);  if (t->leftChild)  myQueue.push(t->leftChild);  if (t->rightChild)  myQueue.push(t->rightChild);  if (!myQueue.empty())  {  t = myQueue.front();  myQueue.pop();  }  else  {  break;  }  }  }  void binaryTree::addNode(const int &u)  {  if (!root)  {  root = new binaryTreeNode;  root->element = u;  root->leftChild = 0;  root->rightChild = 0;  return;  }  binaryTreeNode \*newNode = new binaryTreeNode;  newNode->element = u;  queue<binaryTreeNode\*>myQueue;  binaryTreeNode \*t;  t = root;  while (t)  {  if (t->leftChild)  {  myQueue.push(t->leftChild);  }  if (t->rightChild)  {  myQueue.push(t->rightChild);  }  if (!t->leftChild)  {  t->leftChild = newNode;  return;  }  else  {}  if (!t->rightChild)  {  t->rightChild = newNode;  return;  }  else  {}  if (!myQueue.empty())  {  t = myQueue.front();  myQueue.pop(); //队列中删除一个节点并且将其赋值给t  }  }  }  int binaryTree::height()  {  return treeHeight(root);  }  int binaryTree::treeHeight(binaryTreeNode \*root)  {  if (root == NULL)  return 0;  else  return 1 + max(treeHeight(root->leftChild), treeHeight(root->rightChild));  }  int binaryTree::size()  {  return getSize(root);  }  int binaryTree::getSize(binaryTreeNode \*p)  {  if (p == NULL)  return 0;  else  return(getSize(p->leftChild) + 1 + getSize(p->rightChild));  }  源.cpp  #include <iostream>  #include <string>  #include <queue>  #include <stack>  #include "binaryTree.h"  using namespace std;  binaryTreeNode \*create(char \*pre, char \*in, int len)  {  int k;  if (len <= 0)  return NULL;  binaryTreeNode \*head = new binaryTreeNode;  head->element = \*pre;  char \*p;  for (p = in; \*p != NULL; p++) //应该是\*p!= NULL, 而不是p!=NULL  if (\*p == \*pre)  break;  //判断失败  if (\*p == NULL)  {  cout << "没有答案" << endl;  return NULL;  }  k = p - in;  head->leftChild = create(pre + 1, in, k);  head->rightChild = create(pre + k + 1, p + 1, len - k - 1);  return head;  }  int main()  {  binaryTree y;  cout << "Input1" << endl;  string myInput = " ";  cin >> myInput;  int i = 0;  while (myInput[i])  {  char temp = myInput[i];  y.addNode(temp);  i++;  }  cout << "Output1" << endl;  y.preOrderOutput();  y.inOrderOutput();  y.postOrderOutput();  cout << y.size() << endl;  cout << y.height() << endl;  cout << "Input2" << endl;  string input1;  cin >> input1;  string input2;  cin >> input2;  char\*preOrder = (char\*)input1.c\_str();  char\*inOrder = (char\*)input2.c\_str();  binaryTreeNode \*myRoot = create(preOrder, inOrder, input2.length());  binaryTree newTree;  newTree.makeTree(myRoot);  cout << "Output2" << endl;  newTree.PostOutput();  newTree.LevelOutput();  cout << "End" << endl;  system("pause");  return 0;  } | | | |
| 结论分析与体会：  运行结果： | | | |