**数据结构上机实习报告**

唯一确定一颗二叉树

班级：191174班

学号：20161001764

姓名：牟鑫一

日期：2018.11.23

1. 课程设计题目与要求
   1. 题目：唯一确定一棵二叉树
   2. 系统功能要求

已知一棵二叉树的前序序列和中序序列的遍历序列。

* + - 构造一棵二叉树；
    - 证明构造正确（即分别以前序和中序遍历该树，将得到的结果与给出的序列进行比较）；
    - 对该二叉树进行后序遍历，输出后序遍历序列；
    - 用凹入法输出该二叉树。

1. 需求分析
   1. 问题描述

如果给出了遍历二叉树的前序序列和中序序列，则可以构造出唯一的一棵二叉树，试编写实现上述功能的程序。

扩展：如果给出了遍历二叉树的中序序列和后序序列，也可以构造出唯一的一棵二叉树。

* 1. 系统环境

Microsoft Visual C++ 6.0

* 1. 运行要求

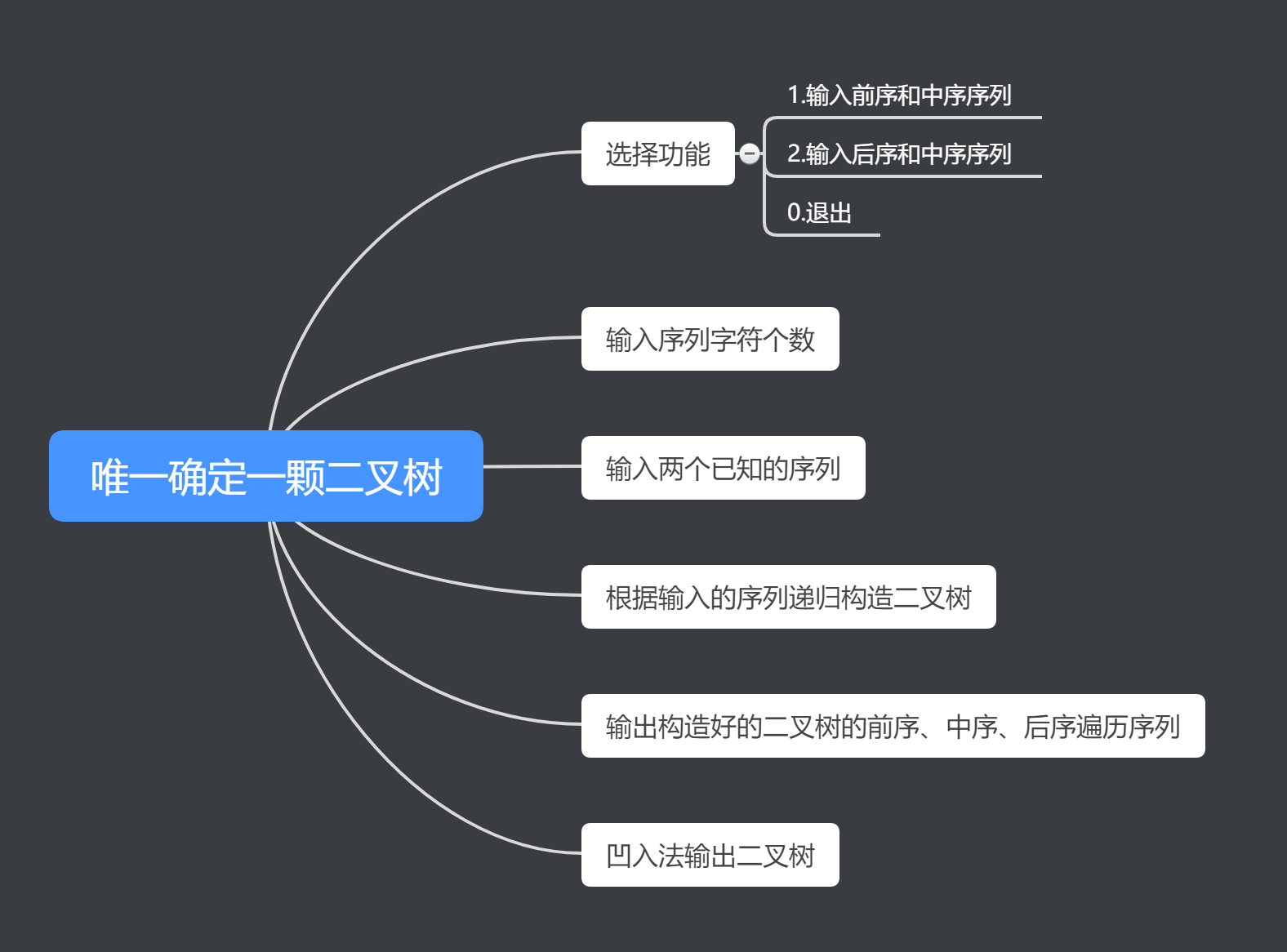
Windows环境下运行

三、概要设计

* 1. 数据结构和储存结构的设计

二叉树结构

* 1. 逻辑结构设计



* 1. 算法设计

用两个字符串\*VLR（前序序列）和\*LVR（中序序列）来递归构造二叉树，二叉树的每个结点含一个数据域（保存当前结点值）和两个指针域（分别指向当前节点的左孩子和右孩子）。

前序序列的第一个字符是根，设置参数k存前序序列的第一个字符在中序序列中的位置，在中序字符串中搜索直到LVR[k]与VLR[0]相同，即找到中序序列第k个字符为根，新建一个二叉树结点保存当前找到的根。继续递归建立左子树，VLR+1，LVR不变，k表示左子树结点数传入参数；递归建立右子树，VLR指针后移k+1，LVR也后移k+1，n-(k+1)作为函数参数。递归遍历直到找完所有字符即确定了一个唯一的二叉树。

利用后序序列和中序序列构造二叉树类似，但需要先倒置LRV和LVR，用类似方法先递归构造二叉树的右子树，再递归构造二叉树的左子树。

递归遍历输出二叉树。

四、详细设计

* 1. 类的函数成员和数据成员设计
     + **结构Treenode：**
       - T data; //数据
       - Treenode<T> \* leftchild, \*rightchild;      //左指针、右指针
     + **类Tree**
       - Treenode<T> \* root; //指向树根
       - T revalue; //数据
       - Tree() { root = NULL; }
       - ~Tree() { delete root; }
       - Tree(T & x) { revalue = x; root = NULL; }
       - Treenode<T> \* createTree(T \*VLR, T \*LVR, int n);

 //VLR(value-leftchild-rightchild)、LVR(leftchild-value-rightchild)

* + - * Treenode<T> \* CreateTree(T \*LRV, T \*LVR, int n);

//VLR前序、LVR中序、LRV后序

* + - * Treenode<T> \* getroot() { return root; } //返回根结点
      * void PreOrder(Treenode<T> \* p);    //前序遍历
      * void InOrder(Treenode<T> \* p);    //中序遍历
      * void PostOrder(Treenode<T> \* p);    //后序遍历
      * void InOrder(Treenode<T> \* p, int n);    //凹入法输出二叉树

五、测试

* 1. 测试一

前序序列为ABDEGCFHIJ

中序序列为DBGEAHFIJC



* 1. 讨论和分析

前序序列为-×+abc/de

中序序列为a+b×c-d/e



六、源程序代码

1. #include"pch.h"
2. #include<iostream>
3. **using** **namespace** std;
5. **template**<**class** T>
6. **struct** Treenode {
7. T data;
8. Treenode<T> \* leftchild, \*rightchild;      //左指针、右指针
9. Treenode() { leftchild = NULL; rightchild = NULL; }
10. Treenode(T & x, Treenode<T> \* l = NULL, Treenode<T> \* r = NULL) { data = x; leftchild = l; rightchild = r; }
11. };
13. **template** <**class** T>
14. **class** Tree {
15. **protected**:
16. Treenode<T> \* root;     //根结点指针
17. T revalue;
18. **public**:
19. Tree() { root = NULL; }
20. ~Tree() { **delete** root; }
21. Tree(T & x) { revalue = x; root = NULL; }
22. Treenode<T> \* createTree(T \*VLR, T \*LVR, **int** n);   //VLR(value-leftchild-rightchild)、LVR(leftchild-value-rightchild)
23. Treenode<T> \* CreateTree(T \*LRV, T \*LVR, **int** n);   //VLR前序、LVR中序、LRV后序
24. Treenode<T> \* getroot() { **return** root; }      //返回根结点
25. **void** PreOrder(Treenode<T> \* p);    //前序遍历
26. **void** InOrder(Treenode<T> \* p);    //中序遍历
27. **void** PostOrder(Treenode<T> \* p);    //后序遍历
28. **void** InOrder(Treenode<T> \* p, **int** n);    //凹入法输出二叉树
29. };
31. **template** <**class** T>
32. Treenode<T> \* Tree<T>::createTree(T \*VLR, T \*LVR, **int** n) {  //根据前序和中序序列构造二叉树
33. **if** (n == 0) **return** NULL;
34. **int** k = 0;
35. **while** (LVR[k] != VLR[0] && k < n - 1) { k++; }    //在中序序列中找到与前序序列根结点值相同的结点
36. Treenode<T> \* temp = **new** Treenode<T>(VLR[0]);
37. temp->leftchild = createTree(VLR + 1, LVR, k);    //递归
38. temp->rightchild = createTree(VLR + k + 1, LVR + k + 1, n - k - 1);
39. **return** temp;     //返回构造好的二叉树的根结点
40. }
42. **template**<**class** T> Treenode<T> \* Tree<T>::CreateTree(T \*LRV, T \*LVR, **int** n) {  //根据后序和中序序列构造二叉树
43. **if** (n == 0) **return** NULL;
44. **int** k = 0;
45. **while** (LVR[k] != LRV[0]) { k++; }     //在中序序列中找到与后序序列根结点值相同的结点
46. Treenode<T> \* temp = **new** Treenode<T>(LRV[0]);
47. temp->rightchild = CreateTree(LRV + 1, LVR, k);    //递归
48. temp->leftchild = CreateTree(LRV + k + 1, LVR + k + 1, n - k - 1);
49. **return** temp;     //返回构造好的二叉树的根结点
50. }
52. **template**<**class** T> **void** Tree<T>::PreOrder(Treenode<T> \* p) {    //前序遍历
53. **if** (p != NULL) {
54. cout << p->data;
55. PreOrder(p->leftchild);
56. PreOrder(p->rightchild);
57. }
58. }
60. **template**<**class** T> **void** Tree<T>::InOrder(Treenode<T> \* p) {    //中序遍历
61. **if** (p != NULL) {
62. InOrder(p->leftchild);
63. cout << p->data;
64. InOrder(p->rightchild);
65. }
66. }
68. **template**<**class** T> **void** Tree<T>::PostOrder(Treenode<T> \* p) {    //后序遍历
69. **if** (p != NULL) {
70. PostOrder(p->leftchild);
71. PostOrder(p->rightchild);
72. cout << p->data;
73. }
74. }
76. **template**<**class** T> **void** Tree<T>::InOrder(Treenode<T> \* p, **int** n) {    //凹入法输出二叉树
77. **if** (p != NULL) {
78. InOrder(p->rightchild, n + 1);
79. **for** (**int** i = 0; i < n; i++) {
80. cout << "     ";
81. }
82. **if** (p != root) cout << "----";
83. cout << p->data << endl;
84. InOrder(p->leftchild, n + 1);
85. }
86. }
88. **int** main() {
89. **int** n;  //要输入的字符的数目
90. **int** x = 1;  //功能选择
91. **while** (x != 0) {
92. cout << "\n========= Menu =========\n" << endl;
93. cout << "1.输入前序和中序序列" << endl;
94. cout << "2.输入后序和中序序列" << endl;
95. cout << "0.退出" << endl;
96. cout << "\n========================\n" << endl;
97. cout << "请选择：";
98. cin >> x;
99. **char** \*VLR = **new** **char**[20];
100. **char** \*LVR = **new** **char**[20];
101. **char** \*LRV = **new** **char**[20];
102. **switch** (x) {
103. **case** 1: {
104. Tree<**char**> tree;
105. cout << "\n========================\n" << endl;
106. cout << "请输入序列字符个数: ";
107. cin >> n;
108. cout << "\n========================\n" << endl;
109. cout << "请输入前序序列：" << endl;
110. cin >> VLR;
111. cout << "\n========================\n" << endl;
112. cout << "请输入中序序列：" << endl;
113. cin >> LVR;
114. Treenode<**char**> \* p = tree.createTree(VLR, LVR, n);
115. cout << "\n========================\n" << endl;
116. cout << "前序遍历：";
117. tree.PreOrder(p);
118. cout << endl;
119. cout << "中序遍历：";
120. tree.InOrder(p);
121. cout << endl;
122. cout << "后序遍历：";
123. tree.PostOrder(p);
124. cout << endl;
125. cout << "\n========================\n" << endl;
126. tree.InOrder(p, 0);
127. }**break**;
128. **case** 2: {
129. Tree<**char**> tree;
130. cout << "\n========================\n" << endl;
131. cout << "请输入序列字符个数: ";
132. cin >> n;
133. cout << "\n========================\n" << endl;
134. cout << "请输入后序序列：" << endl;
135. cin >> LRV;
136. cout << "\n========================\n" << endl;
137. cout << "请输入中序序列：" << endl;
138. cin >> LVR;
139. **for** (**int** i = 0; i < (n / 2); i++) {      //倒置后序序列、中序序列
140. **char** temp;
141. temp = LRV[i];
142. LRV[i] = LRV[n - 1 - i];
143. LRV[n - 1 - i] = temp;
144. temp = LVR[i];
145. LVR[i] = LVR[n - 1 - i];
146. LVR[n - 1 - i] = temp;
147. }
148. Treenode<**char**> \* p = tree.CreateTree(LRV, LVR, n);
149. cout << "\n========================\n" << endl;
150. cout << "前序遍历：";
151. tree.PreOrder(p);
152. cout << endl;
153. cout << "中序遍历：";
154. tree.InOrder(p);
155. cout << endl;
156. cout << "后序遍历：";
157. tree.PostOrder(p);
158. cout << endl;
159. cout << "\n========================\n" << endl;
160. tree.InOrder(p, 0);
161. }**break**;
162. **default**:**break**;
163. }
164. }
165. **return** 0;
166. }