华东师范大学计算机科学与技术学院上机实践报告

课程名称：数据结构	**年级**：2022 级	**上机实践成绩**：
指导教师：金健	**姓名**：岳锦鹏	**上机实践时间**：2 学时
上机实践名称：第五章作业	**学号**：10213903403	**上机实践日期**：2023/11/10

一、 实验目的

1. 使用第五章的知识解决树的综合问题。

二、 实验内容

1. 用顺序表或链表形式实现一棵树，树中节点存放 2 个内容：整数编号，和字符串的文本。每个节点的编号，是整棵树自上而下，自左而右依次编号的。构造这棵树的方式：一行代表 1 个节点对应的儿子节点，第 1 个数据是编号，后续字符串以空白字符分隔，依次作为该节点的儿子，并自动编号。如果新录入的编号，在已有树中不存在，则发出错误提示；如果新录入节点后，对原有的编号有影响，则更新原有编号，使满足编号自上而下、自左而右依次编号的规则。假设树根为 0 号节点，一开始就存在。（注：允许在节点上，增加层次字段，比如，0 号节点的层次为 1；1 号节点“张三”的层次为 2）
   输入案例：
   0 张三 李四 王五　　：表示，3 个人都是 0 号节点的儿子
   2 师大 统计 计算机　　：表示，3 个院系是李四儿子
   0 赵六　　：表示，0 号节点下增加 1 个儿子，此时，2 号节点的儿子编号将需要更新
2. 在第 1 题的基础上，将树转换成二叉树，遵循左孩子右兄弟的原则。

三、 实验原理

1. 程序设计原理。

四、 实验步骤

1. 问题抽象
2. 编写程序
3. 调试程序
4. 完善总结

五、 调试过程、结果和分析

1. VSCode 调试很方便，但用了 CLion 后发现 CLion 调试的手感更好，仔细思考后发现可能原因之一是有时候 CLion 启动调试和重启调试更快，虽然可能只有一秒甚至不到一秒的时间，但在注意力高度集中找 bug 时不到一秒的间断也可能造成分心；
2. 还有可能的原因是 CLion 的变量监视窗口可以步入步过时保持展开的层级；

3. 而且 CLion 的自动补全更加完善，比如输入一个带参数的函数名会自动加好括号并将光标移动到括号内，输入一个无参数的函数名会自动加括号并将光标移动到括号右侧，删除左括号会自动删除相邻的右括号等等；

六、 总结

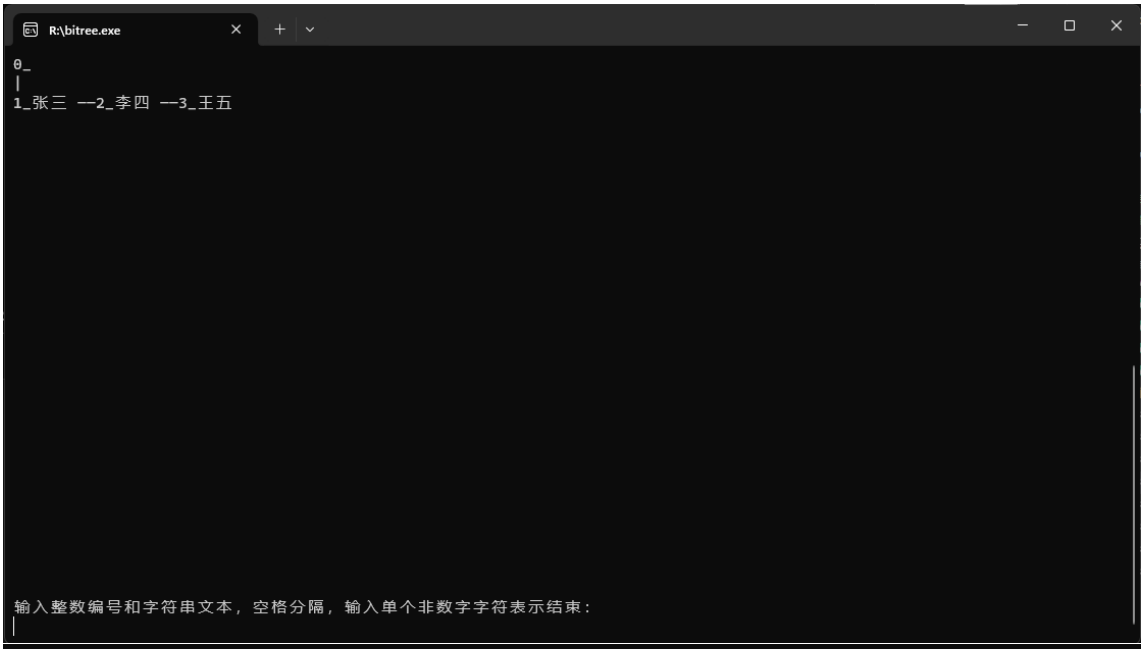
1. 虽然 C++ 的智能指针可以自动管理对象的生命周期，但也许在这种不需要长期维护，也不需要安全性的项目上使用原始指针更方便，只是要自己实现析构函数罢了；
2. 为了简化代码便于从一个入口文件直接编译，直接使用了 include cpp 源文件的方法，可以从 main.cpp 入口直接编译；
3. 将此题中的树转化成二叉树后，此题中的遍历方式不是先序中序后序层序中的任何一种，非常有创新点！👍 受到广度优先搜索的启发，代码中采用了标准库中的队列实现，遍历每个节点时将它的左节点入队，每个节点出队时将它扩展到所有的右节点依次遍历；
4. 虽然代码中也维护了线索化的标志域，不过好像没有用到线索化二叉树；
5. 在命令行输出时要计算每个节点的宽度，防止子节点相互遮挡；
6. 还能做的更好，但由于时间原因，就暂时这样了，以后可以完善一下，比如可以在命令行输出得更美观一些，使用多线程等。

七、 附件

说明：

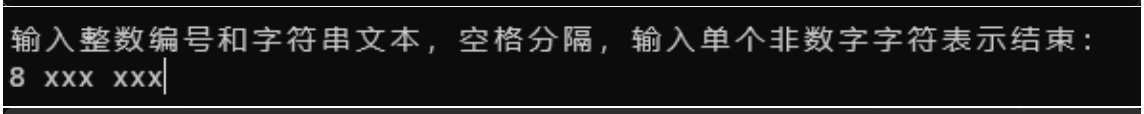
1. 由于多叉树实现的复杂度太高，此题直接使用二叉树完成。
2. 为了便于阅读，使用 CodeGeeX 生成了部分代码的注释，注释不一定正确或有用，仅供参考。

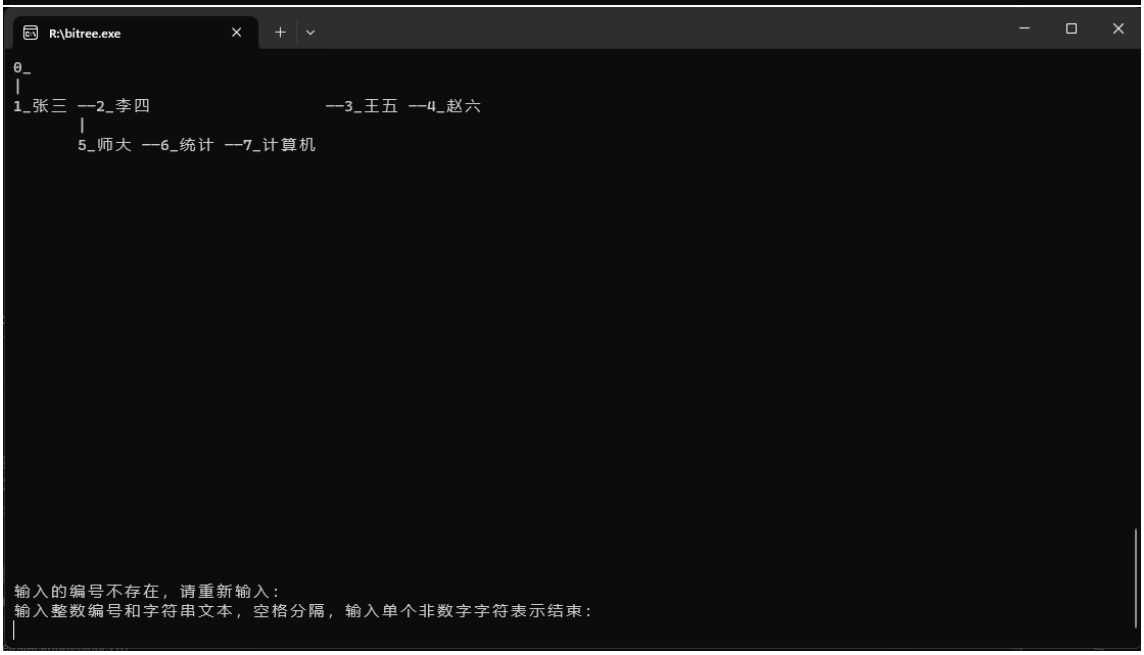
运行时截图：



输入整数编号和字符串文本，空格分隔，输入单个非数字字符表示结束：
2 师大 统计 计算机

输入整数编号和字符串文本，空格分隔，输入单个非数字字符表示结束：
0 赵六





代码：

1. bitree.cpp

```
//
// Created by 423A35C7 on 2023-11-12.
//

// #include <functional>
// #include <list>
#include <cassert>
#include <queue>
```

```cpp
#include <string>

struct Node {
    int num = 0;
    std::string text;
    // 线索化的标记
    bool if_left_chain = false;
    bool if_right_chain = false;
    // 左右子树或前驱后继
    Node* left = nullptr;
    Node* right = nullptr;

    Node(int num, const std::string& text)
        : num(num),
          text(text) {
    }

    ~Node() {
        remove_left();
        remove_right();
    }

    void remove_left() {
        if (left != nullptr && !if_left_chain) {
            // delete the node
            left->~Node();
            // delete the memory
            delete left;
            // set the pointer to nullptr
            left = nullptr;
        }
    }

    void remove_right() {
        if (right != nullptr && !if_right_chain) {
            // delete the node
            right->~Node();
            // delete the memory
            delete right;
            // set the pointer to nullptr
            right = nullptr;
        }
```

```
51      }
52
53  };
54  class BiTree {
55      Node *find(int num) {
56          Node *temp;
57          // 遍历查找 num
58          for (temp = head; temp && temp->num < num; temp = temp->right);
59          return temp;
60      }
61
62      void insert_left(Node *parent, const std::string& text) {
63          // 断言 parent 不为空
64          assert(parent != nullptr);
65          // parent->remove_left();
66          // 如果 parent->left 为空，则插入新节点
67          if (parent->left == nullptr)
68              parent->left = new Node {0, text};
69          else
70              insert_right(parent->left, text);
71      }
72
73      void insert_right(Node *parent, const std::string& text) {
74          // 断言 parent 不为空
75          assert(parent != nullptr);
76          // 记录 tail
77          Node *tail = parent;
78          // 遍历查找 tail->right
79          for (; !tail->if_right_chain && tail->right; tail = tail->right);
80          // 插入新节点
81          tail->right = new Node {0, text};
82          tail->if_right_chain = false;
83      }
84  public:
85      Node *head;
86      int length = 0;
87      BiTree() {
88          // 初始化头节点
89          this->head = new Node(0, "");
90          // 初始化节点数量
91          this->length = 1;
92      }
```

```
93      ~BiTree() {
94          // 删除头节点的左右子节点
95          this->head->remove_left();
96          this->head->remove_right();
97          // 删除头节点
98          delete this->head;
99      }
100
101     void update() {
102         // 左节点入队，出队时扩展到所有的右节点
103         std::queue<Node*> q;
104         Node *current = head, *next = head->left;
105         int current_num = 0;
106         // while (!q.empty()) {
107             // current->if_right_chain = true;
108             // current->right = q.front();
109             // current = q.front();
110             // q.pop();
111             // Node *next = current;
112             // for (; next && !next->if_right_chain;  current = next, next
                ↪  = next->right) {
113             //     next->num = current_num++;
114             //     if (next->left) {
115             //         q.push(next->left);
116             //     }
117             // }
118        // 当前节点为空，则跳出循环
119         while (current) {
120             // 当前节点的编号
121             current->num = current_num++;
122             // 如果当前节点有左子节点，则将左子节点放入队列
123             if (current->left) {
124                 q.push(current->left);
125             }
126
127             // 发生间断，从队列中重新取值
128             if (!current->right || current->if_right_chain) {
129                 // 如果当前节点没有右子节点，或者当前节点已经处理完右子节点，
                    ↪  则跳出循环
130                 if (q.empty()) break;
131                 // 从队列中取出下一个节点
132                 next = q.front();
```

```
133                 // 从队列中删除该节点
134                 q.pop();
135                 // 标记当前节点是线索化右子节点
136                 current->if_right_chain = true;
137             } else {
138                 // 标记当前节点不是线索化右子节点
139                 current->if_right_chain = false;
140             }
141             // 将当前节点的右子节点设置为下一个节点
142             current->right = next;
143             // 将当前节点设置为下一个节点
144             current = next;
145             // 将下一个节点设置为下一个节点的右子节点
146             next = next->right;
147         }
148         // 更新树的层数
149         this->length = current_num;
150     }
151 
152     void insert(int num, std::queue<std::string> &children) {
153         // 找到要插入的父节点
154         Node *parent = this->find(num);
155         // 断言父节点不为空
156         assert(parent != nullptr);
157         // 如果子节点为空，则直接返回
158         if (children.empty()) return;
159         // 插入左子节点
160         this->insert_left(parent, children.front());
161         // 从队列中弹出第一个子节点
162         children.pop();
163         // 将父节点指向左子节点
164         parent = parent->left;
165         // 如果队列不为空，则插入右子节点
166         while (!children.empty()) {
167             this->insert_right(parent, children.front());
168             // 从队列中弹出第一个子节点
169             children.pop();
170             // 将父节点指向右子节点
171             parent = parent->right;
172         }
173         // 更新树的序号
174         this->update();
```

```
    }
};

```

2. view.cpp

```
//
// Created by 423A35C7 on 2023-11-12.
//
// 11:47
// 14:55

#include <algorithm>
#include <iostream>
#include <string>
#include "bitree.cpp"

#define move_and_output(x, y, str) printf("\033[s\033[%d;%dH%s\033u", x, y, str)
// 移动并输出到指定位置
#define moveto(x, y) printf("\033[s\033[%d;%dH", x, y)
// 移动到指定位置
#define movedown(x) printf("\033[%dB", (x))
// 向下移动指定行
#define save_cursor() printf("\033[s")
// 保存光标位置
#define restore_cursor() printf("\033[u")
// 恢复光标位置
#define clear_char(num)           \
for (int i = 0; i < num; i++) \
printf(" ")
// 清空指定位置的字符
#define clear_screen() printf("\033[2J")

// class ViewBox {
//     Node *left_node;
//     int x, y;
//     int width;
//     int height;
//     struct Size {
//         int width;
```

```
35 //         int height;
36 //     };
37 //     Size calculate_size(Node *left_node) {
38 //         Size size {left_node->text.length(), 1};
39 //         if (left_node->left) {
40 //             Size increment;
41 //             increment = calculate_size(left_node->left);
42 //
43 //         }
44 //     }
45 // };
46
47 enum LINE {NONE, HORIZONTAL, VERTICAL};
48
49 int get_length(Node *node) {
50     // 2 是 _ 和空格的长度
51     return std::to_string(node->num).length() + 2 + node->text.length();
52 }
53
54 void print(Node *node, int x, int y, LINE line_type) {
55     // 移动到指定位置
56     moveto(2 * x, y);
57     // 保存当前光标位置
58     save_cursor();
59     // 如果 line_type 为 VERTICAL, 则输出 “|”
60     if (line_type == VERTICAL) {
61         std::cout << "|";
62     }
63     // 恢复光标位置
64     restore_cursor();
65     // 向下移动 1 个单位
66     movedown(1);
67     // 如果 line_type 为 HORIZONTAL, 则输出 “———”
68     if (line_type == HORIZONTAL) {
69         std::cout << "———";
70     }
71     // 输出 node 的 num 和 text
72     std::cout << node->num << "_" << node->text << " ";
73 }
74
75 int output(Node *left_node, int x, int y) {
76     int width = 0;
```

```
77      if (left_node == nullptr) return width;
78      // 打印左节点
79      print(left_node, x, y, VERTICAL);
80      // 计算宽度
81      width += std::max(get_length(left_node), output(left_node->left, x +
        ↪  1, y + width));
82      // 遍历右节点
83      for (Node *temp = left_node; temp->right && !temp->if_right_chain;
        ↪  temp = temp->right) {
84          // 打印右节点
85          print(temp->right, x, y + width, HORIZONTAL);
86          // 计算宽度
87          // 2 是——的长度
88          width += std::max(2 + get_length(temp->right),
            ↪  output(temp->right->left, x + 1, y + width));
89          // width += temp->right->text.length();
90      }
91      return width;
92      // if (left_node->left == nullptr) {
93      //     return width;
94      // }
95      // depth += 1;
96      // return std::max(width, output(left_node->left,  depth));
97  }
```

3. main.cpp

```
1   //
2   // Created by 423A35C7 on 2023-11-12.
3   //
4   // 14:55
5   // 16:57
6
7   #include "view.cpp"
8
9   int main() {
10      // 创建二叉树
11      BiTree bi_tree;
12      // 父节点编号
13      int parent = 0;
14      // 子节点队列
```

```
15      std::queue<std::string> children;
16      // 文本
17      std::string text;
18      // 文本长度
19      text.reserve(100);
20      // 临时变量
21      int temp;
22      // 错误信息
23      std::string error_message;
24      // 重新输入
25      next_input:
26      // 清屏
27      clear_screen();
28      // 输出二叉树
29      output(bi_tree.head, 1, 1);
30      // 移动光标到指定位置
31      moveto(100, 1);
32      // 输出提示信息
33      std::cout << error_message << std::endl << " 输入整数编号和字符串文本,
        ↪  空格分隔, 输入单个非数字字符表示结束: " << std::endl;;
34      // 循环输入
35      while (std::cin >> parent) {
36          // 如果编号不存在
37          if (parent >= bi_tree.length) {
38              // 设置错误信息
39              error_message = " 输入的编号不存在, 请重新输入: ";
40              // 忽略输入
41              std::cin.ignore(1024, '\n');
42              // 重新输入
43              goto next_input;
44          }
45          // 循环输入
46          while ((temp = std::cin.get()) > 0) {
47              // 根据输入的值进行判断
48              switch (temp) {
49                  // 换行符
50                  case '\n':
51                      // 如果文本不为空
52                      if (!text.empty()) {
53                          // 将文本添加到子节点队列中
54                          children.push(text);
55                          // 清空文本
```

```
                        text.clear();
                    }
                    // 将父节点编号和子节点队列插入二叉树
                    bi_tree.insert(parent, children);
                    // 重新输入
                    goto next_input;
                // 空格
                case ' ':
                    // 如果文本不为空
                    if (!text.empty()) {
                        // 将文本添加到子节点队列中
                        children.push(text);
                        // 清空文本
                        text.clear();
                    }
                break;
                // 其他字符
                default:
                    // 将字符添加到文本中
                    text.push_back(temp);
                break;
            }
        }
    }
    // 添加节点
    // children.emplace(" 张三");
    // children.emplace(" 李四");
    // children.emplace(" 王五");
    // bi_tree.insert(parent, children);
    // parent = 2;
    // children.emplace(" 师大");
    // children.emplace(" 统计");
    // children.emplace(" 计算机");
    // bi_tree.insert(parent, children);
    // parent = 0;
    // children.emplace(" 赵六");
    // bi_tree.insert(parent, children);
    return 0;
}
```

4. CMakeLists.txt

```
cmake_minimum_required(VERSION 3.9)
project(chapter5)

# set(CMAKE_CXX_STANDARD 11)

# include_directories(
#         include
# )

add_executable(bitree
        main.cpp)

SET(EXECUTABLE_OUTPUT_PATH R:/)

if (CMAKE_BUILD_TYPE STREQUAL Release)
#    # 也可以 set(CMAKE_CXX_FLAGS_RELEASE ...)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -fexec-charset=GBK")
endif()

target_link_libraries(bitree)
```