苏 州 大 学 实 验 报 告

院、系	计算机学院	年级专业	19 计算机类	姓名	张昊	学号	1927405160
课程名称	数据结构课程实践					成绩	
指导教师	孔芳	同组实验者	无		实验日期	2020 年 11 月 25 日	

实 验 名 称 二叉树

一、问题描述及要求

产生一个菜单驱动的演示程序，用以说明二叉树的使用。元素由单个键组成，键为单个字符。用户能演示的二叉树基本操作至少包括：构造二叉树，按先序、中序、后序、层序遍历这棵二叉树，求二叉树的深度、宽度，统计度为 0，1，2 的结点数等。

二叉树采用链式存储结构。对二叉查找树做上述工作，且增加以下操作：插入、删除给定键的元素、查找目标键。

二、概要设计

1. 问题简析

本次实验内容是设计一个二叉树使用的演示程序，与用户的交互方式为菜单。实验需要完成在实现了二叉树和二叉查找树的基本操作的基础之上，设计并提供与用户交互的菜单。

2. 系统功能列表

程序要实现如下功能：

1. 用户选择创建普通的二叉树或者创建二叉查找树；
2. 用户创建一棵二叉树；
3. 按先序、中序、后序、层序遍历二叉树；
4. 求二叉树的深度、宽度；
5. 统计度为 0，1，2 的结点数；
6. 在二叉查找树中插入、删除给定键的元素、查找目标键；
7. 清空并重新建立一棵二叉树（二叉查找树）；
8. 用户还可以选择更换树的种类。

3. 程序界面设计

程序启动，首先输出菜单，等待用户输入构造二叉树的类型：二叉树、二叉查找树。

```
  ###################################
          The Type of Tree
  ###################################
     [1] Binary Tree
     [2] Binary Search Tree
    [13] EXIT: exit the program
  ###################################
Your choice:
```

根据用户输入，输出不同的提示信息来引导用户创建二叉树。当用户输入有误时，给出错误信息并重新提示用户输入。具体见下：

```
Your choice: 1
Now initialize the tree.

> Choose your creation type:
> [1] Using preorder sequence and inorder sequence
> [2] Using inorder sequence and postorder sequence

Your choice: 1
Please input the inorder sequence: 123456
Please input another sequence: 2356401
Catch an error when creating tree: the size of containers are not equal.
Try again.
Please input the inorder sequence: 123456
Please input another sequence: 235641
Catch an error when creating tree: root node not found in inorder
container. Try again.
Please input the inorder sequence: 123456
Please input another sequence: 213546
Press 'Enter' to continue...
```

二叉树创建完成后，清屏并输出菜单，等待用户输入功能序号，根据用户输入输出对应信息，或完成对应功能。菜单的设计如下所示。

```
  ########################################
          The Functions of Tree
  ########################################
     [1] Preorder traversal
     [2] Inorder traversal
     [3] Postorder traversal
     [4] Level traversal
     [5] The height of the tree
     [6] The width of the tree
     [7] The number of nodes with degree n
   --- --- --- --- --- --- --- --- --- ---
     [8] insert an element
```

```
    [9] remove an element
   [10] search an element
  --- --- --- --- --- --- --- --- --- ---
   [11] RESET: clear the tree
   [12] BACK: return to main menu
   [13] EXIT: exit the program
 ########################################
```

其中选项 8，9，10 只在用户创建时选择了二叉查找树（Binary Search Tree）后才会输出。

如果用户输入了有效的编号后，程序会调用二叉树的相应方法，（有些会进一步根据用户后续的输入）输出对应的信息，如下图所示。

```
Your choice: 3
The postorder sequence of tree is: 146532
Press 'Enter' to continue...
```

```
Your choice: 7
Please input the degree: 2
There are 2 nodes with degree 2 in the tree
Press 'Enter' to continue...
```

```
Your choice: 15
Undefined command.
Press 'Enter' to continue...
```

4. 程序结构设计思路

程序为演示二叉树和二叉查找树的使用，采用二叉树的数据结构来组织用户输入的演示用数据，其中将二叉树的结点封装为二叉树节点结构体 BinaryNode，将二叉树封装成二叉树模板类 BinaryTree，并从模板类 BinaryTree 派生出二叉查找树类模板类 BinarySearchTree。各类的模板参数均为数据元素（二叉查找树中称为**记录**）的类型。上述三个类放置于命名空间 tree 中。各个类的公有方法的介绍见下表。

BinaryNode 结构体	
`BinaryNode();`	默认构造函数
`BinaryNode(const Entry &data);`	构造函数
BinaryTree 类	
`BinaryTree();`	默认构造函数
`BinaryTree(const BinaryTree<Entry> &);`	拷贝构造函数
`BinaryTree<Entry> &operator=(const BinaryTree<Entry> &);`	赋值构造函数
`virtual ~BinaryTree();`	析构函数（虚函数）

`void preorder(Function visit);`	先序遍历 visit 为结点访问函数
`void inorder(Function visit);`	中序遍历 visit 为结点访问函数
`void postorder(Function visit);`	后序遍历 visit 为结点访问函数
`void levelOrder(Function visit);`	层序遍历 visit 为结点访问函数
`bool empty() const;`	判断是否为空二叉树
`int height() const;`	二叉树的高度
`int width() const;`	二叉树的宽度
`int count(int degree) const;`	统计度为 degree 的结点
`void clear();`	清空二叉树

`void createByPreorderInorder(`
`const std::vector<Entry> &v1,`
`const std::vector<Entry> &v2);`
	使用先序遍历序列 v1 和中序遍历序列 v2 创建二叉树

`void createByInorderPostorder(`
`const std::vector<Entry> &v1,`
`const std::vector<Entry> &v2);`
	使用中序遍历序列 v1 和后序遍历序列 v2 创建二叉树

BinarySearchTree 类	
公有方法继承自 BinaryTree 类	
`bool search(Record &target) const;`	查找目标记录 true：查找成功 false：查找失败
`bool insert(const Record &newData);`	新增记录 true：插入成功 false：插入失败，记录已存在
`bool remove(const Record &oldData);`	删除记录 true：删除成功 false：删除失败，记录不存在
`void create(const std::vector<Record> &);`	创建二叉查找树

此外，为实现菜单驱动的功能，程序作如下两个设计。

第一个是设计命名空间 menu 来管理菜单的常量以及相应的函数，其中包含了在

程序界面设计部分提到的程序中使用的所有菜单的定义，并使用枚举类型枚举菜单数字对应的功能。具体内容包含如下表所示。

菜单字符串常量	
`const char WELCOME[];`	首页欢迎提示语
`const char PUBLIC_FUNC_MENU[];`	公有方法菜单
`const char BST_ONLY_FUNC_MENU[];`	仅二叉查找树使用的菜单
`const char PUBLIC_CONTROL_MENU[];`	公有控制菜单
`const char TREE_TYPE_MENU[];`	树的类型菜单
`const char BINARY_TREE_CREATION_MENU[];`	创建树的菜单（普通二叉树）
类型定义	
`using SelectedNumber = int;`	接收到的用户的选项
`enum class TreeType;`	树的类型菜单选项
`enum class TreeFunc;`	树的方法菜单选项
`enum class BinaryTreeCreation;`	创建菜单选项（普通二叉树）
常用函数	
`SelectedNumber getSelectedNumber(const std::string &);`	从标准输入读取一个整数作为用户输入的选项
`void putChar(char &ch);`	向标准输入输出一个字符

第二个是将控制台程序与用户的交互抽象为“会话”（session），并将控制台与用户的交互动作封装成控制台类 Console，使用时将 Console 类实例化为会话 session，并调用相应的方法来实现输出菜单，用户输入，响应输出这一流程。由于生成的二叉树类型由用户决定，存在不确定性，故使用动态分配内存的方式创建二叉树对象，并用基类指针指向该对象；使用成员变量标识二叉树的种类。综上所述，控制台类 Console 的设计如下表所示。

属性（私有成员）	
`tree::BinaryTree<char> *tree;`	二叉树对象指针，对象动态创建
`menu::TreeType type;`	二叉树类型
公有方法	
`Console();`	构造函数
`~Console();`	析构函数
`void initialize();`	初始化一个会话
`int exec();`	执行会话进程
私有方法	
`void onStart();`	会话启动时的初始化工作

`void onDestroy()`	会话结束时的清理工作
`static void clear()`	控制台清屏
`static void pause()`	暂停控制台输出
`static char getchar(const std::string &);`	从标准输入读取非空的一行并返回首个字符
`void createBinaryTree();`	创建二叉树的方法
`void createBSTree();`	创建二叉查找树的方法
`void funcProcess(menu::TreeFunc choice);`	处理二叉树的选项

三、详细设计

（一）二叉树关键算法的实现

注：所有的递归算法都设计了 protected 的辅助函数，公有方法调用相应的函数。

1 二叉树的创建

程序选择的二叉树的创建方式为利用两个遍历序列来创建二叉树，具体为从中序序列确定根节点，从先序/后序序列中确定根节点的位置和左右子树的序列长度，通过递归嵌套的方法来完成二叉树的创建。具体实现的代码详见附件。

2 二叉树的递归遍历算法

二叉树的三种递归式遍历的具体过程如下。

先序遍历：若二叉树不为空树，则①访问根结点；②先序遍历左子树；③先序遍历右子树。

中序遍历：若二叉树不为空树，则①中序遍历左子树；②访问根结点；③中序遍历右子树。

后序遍历：若二叉树不为空树，则①后序遍历左子树；②后序遍历右子树； ③访问根结点。

可见，按照先左子树后右子树的方式扫描二叉树，区别仅在于访问结点的时机。得益于递归定义的简洁性，只需数行即可实现这三种遍历算法。具体代码详见附件。

3 二叉树的层次遍历算法

层次遍历中，确定节点访问次序的原则可概括为“先上后下、先左后右”。为了保持结点访问的先后次序关系，使用队列结构来实现层次遍历。具体算法见如下伪代码：

levelOrder 算法

Input: 结点访问函数 visit

```
1.  initQueue(Q);
2.  if ( root != NULL ): Q.push(root);
3.  while ( !Q.empty() ):
      3.1     q := Q.pop();
      3.2     visit(q->data);   //访问结点 q 的数据域
```

3.3 **if** (q->left != NULL): Q.push(q->left);

3.4 **if** (q->right != NULL): Q.push(q->right);

4 求二叉树的深度

二叉树任一节点的深度，都等于其孩子节点的最大深度加一。在每一节点 v 处，只需读出其左、右孩子的深度并取二者之间的大者，再计入当前节点本身，就得到了 v 的深度。得益于二叉树的递归定义，其算法可以简洁地表示如下：

max{sub_root->left 深度, sub_root->right 深度} + 1

算法可以用 C++描述为：（注意空树的情况）

```
template<class Entry>
int tree::BinaryTree<Entry>::recursiveHeight(BinaryNode<Entry> *subRoot) {
    if (subRoot == nullptr) { return 0; }   // Empty tree
    return std::max(recursiveHeight(subRoot->left), recursiveHeight(subRoot->right)) + 1;
}
```

5 求二叉树的宽度

与深度的计算不同，二叉树的深度是同一层次上的结点数目的最大值。与二叉树的层次遍历算法类似，仅仅是在层次遍历的基础上统计各层的结点数量，并维护好其中的最大值。同样地，需要一个队列来保存各个结点的孩子结点。

新的问题是，普通的层次遍历算法内部并不清楚当前遍历的是第几层的结点，需要在其基础上进行修改。原层次遍历算法是在循环中每个结点访问后就将其不为空的左右孩子加至队列，而现在不需要访问每个结点，所以考虑在循环中统计并更新一个层次的结点数量，具体来说是在一次循环中将当前层次的所有节点的所有非空孩子节点入队，这样在下一次循环中就可以统计当前层次的结点数量了。具体的算法可由如下伪代码说明。

width 算法

1. initQueue(Q);
2. max := 0; //最大结点数
2. **if** (root != NULL): Q.push(root);
3. **while** (!Q.empty()):
   - 3.1 level := Q.size(); //当前层次结点数
   - 3.2 max = max > level ? max : level;
   - 3.3 **for** (i := 0; i < level; ++i):
     - 3.3.1 q := Q.pop();
     - 3.3.2 **if** (q->left != NULL): Q.push(q->left);
     - 3.3.3 **if** (q->right != NULL): Q.push(q->right);
4. **return** max;

6 统计度为 0，1，2 的结点数

二叉树中度为d $(0 \le d \le 2)$的结点数目等于左右子树中度为d的结点数目之和，如果根节点的左右子树均存在则加 1。借助二叉树的递归定义可以很块地得到统计相应节点的度数算法，可由如下 C++代码描述。

```
template<class Entry>
int tree::BinaryTree<Entry>::recursiveDegreeCount(
                                    BinaryNode<Entry> *subRoot, int degree) {
    if (degree < 0 || degree > 2) {
        throw std::invalid_argument{"degree out of its range"};
    }
    if (subRoot == nullptr) { return 0; } // Empty tree
    int child = 0;
    if (subRoot->left != nullptr) { child++; }
    if (subRoot->right != nullptr) { child++; }
    if (child == degree) {  // Find the same degree node
        return 1 + recursiveDegreeCount(subRoot->left, degree) + \
                   recursiveDegreeCount(subRoot->right, degree);
    }
    return recursiveDegreeCount(subRoot->left, degree) + \
           recursiveDegreeCount(subRoot->right, degree);
```

（二）二叉树关键算法的实现

1 在二叉查找树中的查找和插入

向排序树中插人一个结点首先需要查找该结点的位置，然后再执插入动作。具体地，由二叉查找树的定义，若根节点为空则查找失败，返回失败信息或插入结点；否则执行以下操作：若 target 等于 root->data，则查找成功，返回查找成功信息或插入失败信息；若 target 小于 root- >data，则在根节点的左子树上查找；若 target 大于 root->data，则在根节点的右子树上查找。可以看出，新结点作为叶子插入到二叉查找树中都是作为叶子节点插入的，且方法和查找相同，是递归的。

2 在二叉查找树中的删除

与上面查找方法相同，要删除二叉查找树中的一个节点需要首先找到这个节点的位置。但是删除结点则不同，被删除的可能是叶子结点，也可能是分支结点，当删除分支结点时就破坏了二叉查找树中原有结点之间的链接关系，需要重新修改指针，使得删除结点后仍为一棵二叉查找树。

具体地，讨论如下的两种情况。

（1）当被删除节点的度小于等于 1 时，则将其替换为非空的子树，再删掉该节点即可。

（2）当被删除节点的度等于 2 时，需要再子树中找到一个可以用来替换这一记录的另外的记录，而替换的结点的度小于等于 1，从而变为删除这个度小于等于 1 的结点，也就是转化到第（1）种情况。这个值应该是小于该记录的最大值（或为大于该记录的最小值）。以小于该记录最大值为例，按照二叉查找树的特点，这个结点应位于左子树的最右下的结点。故考虑将待删结点先向左分支移动，再沿着右分支不断下移，直至找到最右下的结点，然后交换最右下的结点和待删结点的数据域，删除待删结点即可，同时需注意边界情况。

3 二叉查找树的创建

构造一棵二叉查找树的过程是从空树开始，依次插入每一个结点。创建过程通过不断调用二叉查找树的插入算法进行。

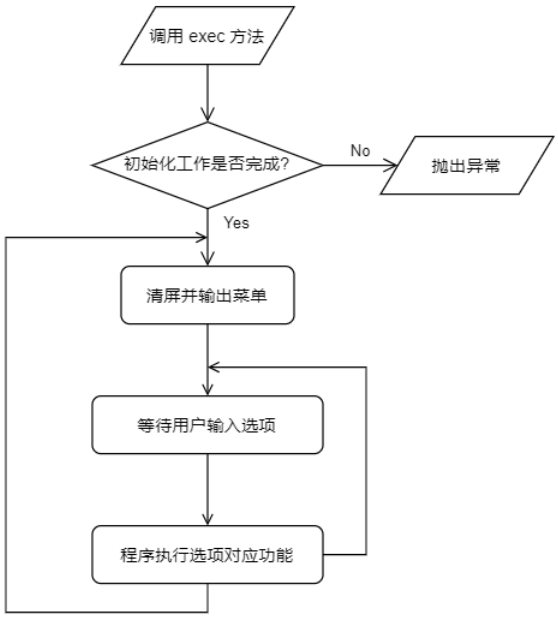
（三）菜单与控制台程序的实现

1 会话 session 的构造与初始化

本着菜单驱动的思想，构造和初始化的方法中“显示菜单->用户输入选项->程序执行选项对应功能”这一过程只会被执行一次，来完成树的类型选择和二叉树的创建。

2 执行会话的 exec 方法设计

菜单驱动程序的主要逻辑为“显示菜单->用户输入选项->程序执行选项对应功能”这一流程。在控制台类用于执行会话的 Console::exec 方法中最主要的是无限重复这一过程，不断响应用户输入并执行。具体的流程可由如下流程图描述。

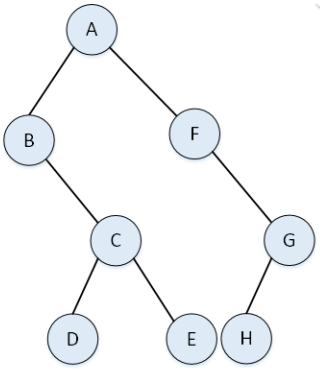


其中，等待用户输入选项由 menu 命名空间内的函数 getSelectedNumber 实现，具体实现为从标准输入读取一行并将其转换成整数；执行选项对应的功能由类的私有方法 funcProcess 实现，具体为使用 switch-case 结构，借助在 menu 内定义的枚举类型来定位调用二叉树的方法。

四、实验结果测试与分析

（一）二叉树的使用测试

运行程序。树的种类选择 1：[1] Binary Tree；构造方法选择使用先序序列和中序序列：[1] Using preorder sequence and inorder sequence；输入中序序列 BDCEAFHG，先序序列 ABCDEFGH，来构造一棵如下所示的二叉树。



程序运行截图：

```
  ####################################
            The Type of Tree
  ####################################
     [1] Binary Tree
     [2] Binary Search Tree
    [13] EXIT: exit the program
  ####################################

Your choice: 1
Now initialize the tree.

> Choose your creation type:
> [1] Using preorder sequence and inorder sequence
> [2] Using inorder sequence and postorder sequence

Your choice: 1
Please input the inorder sequence: BDCEAFHG
Please input another sequence: ABCDEFGH
Press 'Enter' to continue...
```

下面测试各选项。

输入 3，测试后序遍历序列的输出

期望输出：DECBHGFA

实际输出：

```
Your choice: 3
The postorder sequence of tree is: DECBHGFA
```

输出符合预期。

输入 4，测试层次遍历序列的输出

期望输出：ABFCGDEH

实际输出：

```
Your choice: 4
The level order sequence of tree is: ABFCGDEH
```

输出符合预期。

输入 5，测试程序输出树的深度（高度）

期望输出：4

实际输出：

```
Your choice: 5
The height of tree is: 4
```

输出符合预期。

输入 6，测试程序输出树的宽度

期望输出：3

实际输出：

```
Your choice: 6
The width of tree is: 3
```

输出符合预期。

输入 7，测试程序统计度为 0，1，2 的结点数

输入 0 输出 3，输入 1 输出 3，输入 2 输出 2，输入 3 输出错误信息

实际输出：

```
Your choice: 7
Please input the degree: 0
There are 3 nodes with degree 0 in the tree
```

```
Your choice: 7
Please input the degree: 1
There are 3 nodes with degree 1 in the tree
```

```
Your choice: 7
Please input the degree: 2
There are 2 nodes with degree 2 in the tree
```

```
Your choice: 7
Please input the degree: 3
Catch an error: degree out of its range. Try again.
```

输出符合预期。

输入 8，测试程序异常输入的容错性

期望输出错误信息

实际输出：

```
Your choice: 8
Undefined command.
```

输出符合预期。

（二）二叉查找树使用测试

在上面运行的程序的基础上，输入 12，程序回到启动时的菜单。树的种类选择 2：[2] Binary Search Tree；构造序列输入 Data_Structure，构造一棵二叉查找树。

程序运行截图：

```
  ####################################
            The Type of Tree
  ####################################
     [1] Binary Tree
     [2] Binary Search Tree
    [13] EXIT: exit the program
  ####################################

Your choice: 2
Now initialize the tree.
Please input the sequence of elements: Data_Structure
Press 'Enter' to continue...
```

下面测试各选项。

输入 2，测试中遍历序列是否为有序序列

期望输出：DS_acertu

实际输出：

```
Your choice: 2
The inorder sequence of tree is:
```

原因分析：二叉查找树的构造是通过循环调用二叉查找树的插入算法来实现构建的。设计二叉查找树的插入算法时，在公用方法中调用了一个递归函数，其原型为

```
static bool recursiveSearchInsert(BinaryNode <Record> *subRoot,
                                  const Record &newData);
```

在函数执行过程中对形参 subRoot 作了修改，原意是想以此修改父节点的链接关系，但是这种修改并没有真正的带回到调用方，故导致了表面上插入但实际上未插入的选项，并且由于动态申请的空间不能被回收，从而导致内存泄漏的发生。通过排查，在删除算法中的两个递归函数也受到了这一问题的影响，受影响的函数原型如下所示。

```
static bool recursiveSearchRemove(BinaryNode <Record> *subRoot,
                                  const Record &target);
static void removeRoot(BinaryNode <Record> *subRoot);
```

解决方案：对于这一问题的解决办法有两种。一种是添加一个参数，将父节点也传递给这个函数。这种方案回修改函数原型，从而影响到调用处的逻辑，对原代码的重构比较大。另一种是采用二重指针或者指针的引用来将原指针的修改带回，从而达到效果。实际解决问题中使用了第二种策略中的指针的引用，修改后的各个函数原型如下（修改处已高亮）：

```
static bool recursiveSearchInsert(BinaryNode <Record> *&subRoot,
                                  const Record &newData);
```

```
static bool recursiveSearchRemove(BinaryNode <Record> *&subRoot,
                                  const Record &target);
static void removeRoot(BinaryNode <Record> *&subRoot);
```

这样就只需修改这三处即可，不需改动代码逻辑。

修正后的输出：

```
Your choice: 2
The inorder sequence of tree is: DS_acertu
```

输出符合预期。

在程序修正后，继续进行以下测试。

输入 10，测试查找给定键的元素

期望输入字符'_'，输出查找成功，输入字符'd'，输出查找失败

实际输出：

```
Your choice: 10
Please enter the element to be searched: _
The element '_' is found in the tree.
```

```
Your choice: 10
Please enter the element to be searched: d
The element 'd' is NOT found in the tree.
```

输出符合预期。

输入 8，测试插入给定键的元素

期望输入 8，输入字符'.'，输出插入成功，输入 2，输出.DS_acertu；输入字符'D'，输出插入失败，输入 2，输出.DS_acertu

实际输出：

```
Your choice: 8
Please enter the element to be inserted: .
The element '.' is inserted successfully.
```

```
Your choice: 2
The inorder sequence of tree is: .DS_acertu
```

```
Your choice: 8
Please enter the element to be inserted: D
The element 'D' is already in the tree.
```

```
Your choice: 2
The inorder sequence of tree is: .DS_acertu
```

输出符合预期。

输入 9，测试删除给定键的元素

期望输入 9，输入字符'.'，输出删除成功，输入 2，输出 DS_acertu；输入字符'0'，

输出删除失败，输入 2，输出 DS_acertu

实际输出：

```
Your choice: 9
Please enter the element to be removed: .
The element '.' is removed successfully.
```

```
Your choice: 2
The inorder sequence of tree is: DS_acertu
```

```
Your choice: 9
Please enter the element to be removed: 0
The element '0' is NOT in the tree.
```

```
Your choice: 2
The inorder sequence of tree is: DS_acertu
```

输出符合预期。

五、小结

通过本次实验，加深了我对二叉树和二叉查找树的理解，并且回顾了相关算法的流程和实现。在解决其中的异常结果时提高了我的 debug 能力。此外，前期对控制台类的抽象的过程进一步提高了我的面向对象思想。

六、附录

1. **源代码路径**：C++源代码位于附件中 src 目录下。
2. **文件编码**：UTF-8；**行分隔符**：LF（\n）。
3. **实验环境**：Linux 操作系统，g++编译器，基于 cmake 构建；在 CLion 集成开发环境中调试运行通过。手动编译运行方法为（在 Linux/Unix shell 中）：

```
$ mkdir build  # pwd: src/
$ cd build
$ cmake ..
$ make
$ ./BinaryTreeDemo
```