**二叉查找树**

1. **实验目的和要求**

用非递归实现二叉查找树类。

**二、实验内容**

1. 实验准备
   1. 理论知识介绍
      1. 树的定义:
         1. 树形结构的特点是一个数据就够可以有很多个直接后继，但只有一个直接前驱，而线性结构中的每个数据元素至多只有一个直接前驱和一个直接后继。
      2. 二叉树
         1. 二叉树是结点的有限集合，它或者为空，或者由一个根结点及两颗互不相交的左、右子树构成，而其左、右子树又都是二叉树。
         2. 如果一颗二叉树中任意一层的结点个数都达到了最大值，那么这颗二叉树称为满二叉树或丰满树。
         3. 完全二叉树是在满二叉树的最底层自右至左依次（不能跳过任何一个结点）去掉若干个结点。
         4. 二叉树的链接实现
            1. 二叉树的主要储存方式是链接储存，用链接储存结构储存二叉树是很自然的，二叉树的链接储存就是用指针指出父子关系。大多数树的应用只需要找结点的儿子，因此每个结点只需要指出它的子结点储存在哪里。但也有一些应用需要找结点的父亲，因此还必须指出它的父结点储存在哪里。这样就星辰了两种链接储存结构：标准储存结构和广义的标准储存结构。
            2. 标准储存结构也称为二叉链表。在二叉链表中，每个储存结点由三个字段组成，储存结构数据元素值的数据字段以及指向左、右儿子的指针字段。data储存二叉树节点中的数据元素，left和right存放左、右儿子的地址。left和right又是也被称为左、右指针。
      3. 查找树
         1. 二叉查找树
            1. 二叉查找树的定义

二叉查找树也成为二叉排序树。它或者是一颗空树，或者是一颗同时满足下列条件的二叉树：

1. 若左子树不空，则左子树中的所有元素的键值都比根结点的键值小。
2. 若右子树不空，则右子树中的所有元素的键值都比根结点的键值大。
3. 它的左右子树也都是二叉查找树。

根据二叉查找树的基本特性，可以引申出二叉查找树的另一个重要性质：中序遍历一颗二叉查找树所得到的访问序列是按键值的递增次序排列的。因此，二叉查找树也能用于排序一组数据。

1. 实验项目
   1. 分析
      1. 要利用非递归的方法实现二叉查找树，我们知道在一般的情况下递归函数都是可以和循环函数相互替换的，所以我们就可以这个性质来设计循环的函数来对现有的递归函数进行替换从而实现我们想要的结果。
      2. 由于二叉查找树的层数是不确定的，我们刚开始只知道根结点的位置，所以我们利用的循环应该是while循环而不是for循环。
      3. 那么我们就可以模仿书上的递归函数的实现来对二叉查找树进行循环的实现。
   2. 方案
      1. 先利用和书上相同的构造设计好整个二叉查找树的声明
      2. 由于我们不再利用递归函数，所以我们就可以不需要私有的find,insert,remove函数了
      3. 利用while循环来对find,insert,remove函数进行重新定义。
   3. 测试数据

int main()

{

int a[] = { 10,8,6,21,87,56,4,0,11,3,22,7,5,34,1,2,9 };

BinarySearchTree<int> tree;

for (int i = 0; i < 17; ++i)

{

tree.insert(a[i]);

}

cout << "find 2 is " << (tree.find(2) ? "true" : "false") << endl;

tree.remove(2);

cout << "after delete 2, find 2 is " << (tree.find(2) ? "true" : "false") << endl;

cout << "find 3 is " << (tree.find(3) ? "true" : "false") << endl;

tree.remove(3);

cout << "after delete 3, find 3 is " << (tree.find(3) ? "true" : "false") << endl;

cout << "find 1 is " << (tree.find(1) ? "true" : "false") << endl;

cout << "find 22 is " << (tree.find(22) ? "true" : "false") << endl;

cout << "find 56 is " << (tree.find(56) ? "true" : "false") << endl;

return 0;

}

* 1. 测试结果

find 2 is true

after delete 2, find 2 is false

find 3 is true

after delete 3, find 3 is false

find 1 is true

find 22 is true

find 56 is true

* 1. 出错情况

第一个错误：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Severity | Code | Description | Project | File | Line | Suppression State |
| Error | C2065 | 'NULL': undeclared identifier | BinarySearchTree | c:\users\93137\source\repos\binarysearchtree\binarysearchtree\binary\_search\_tree.h | 22 |  |

经过查询，我需要将”stddef.h”头文件包括在项目里面才能避免这个错误的发生，虽然最终问题得到解决，但是我不太清楚为什么在以前的项目中我使用NULL都没有任何问题。

第二个错误：

find 2 is false

after delete 2, find 2 is false

find 3 is false

after delete 3, find 3 is false

find 1 is false

find 22 is false

find 56 is false

经过监视，发现tree的结构如下

- tree {root=0x00000000 <NULL> } BinarySearchTree<int>

- root 0x00000000 <NULL> BinarySearchTree<int>::BinaryNode \*

data <Unable to read memory>

left <Unable to read memory>

right <Unable to read memory>

这应该是insert函数没有达到应有的效果，最终发现是因为用t替换root指针之后，没有对root指针进行赋值。

第三个错误：

- t 0x00008123 {???} int \*

<Unable to read memory> int

经过测试发现，当delete掉一个指针之后，这个指针指向的内容并不是NULL，所以导致了判断错误；

**三、实验小结**

1. 重点

二叉查找树的非递归实现和二叉查找树的应用。

1. 难点

由于C++引用在初始化过程中就绑定好了其对应的变量，绑定好之后引用就不能再解绑了，所以在递归函数中我们可以使用引用，但是如果到了循环体中我们就不能再使用引用了，否则就会出现错误。

1. 收获、体会

利用非递归的方法实现二叉查找树需要我们将递归函数中的引用修改为合适的指针从而在找到我们想要的元素值后我们才可以对父结点的内容进行修改，否则我们就要重新遍历一边二叉查找树，这样会大大降低程序的运行效率。