数据结构课程设计

项目说明文档

**二叉排序树**



同济大学

Tongji University

姓名： 林觉凯

学号： 2253744

指导老师： 张颖

学院专业： 软件学院 软件工程

**目录**

**1.项目分析-------------------------------------------------------------------------4**

**1.1 项目背景分析-------------------------------------------------------------------------4**

**1.2项目功能分析--------------------------------------------------------------------------4**

**1.2.1项目功能要求**--------------------------------------------------------------------------------**4**

**1.2.2项目输入要求**--------------------------------------------------------------------------------**4**

**1.2.3项目输出要求**--------------------------------------------------------------------------------**4**

**1.2.4项目示例**--------------------------------------------------------------------------------------**4**

**2.项目设计-------------------------------------------------------------------------5**

**2.1 数据结构设计-------------------------------------------------------------------------5**

**2.2类设计-----------------------------------------------------------------------------------5**

**2.2.1二叉排序树结点类(BSTnode)----------------------------------------------------5**

**2.2.2二叉排序树类(BST)---------------------------------------------------------------5**

**2.3成员与操作设计-----------------------------------------------------------------------6**

**2.4系统流程设计--------------------------------------------------------------------------6**

**3.项目实现-------------------------------------------------------------------------7**

**3.1 六个基本函数的实现----------------------------------------------------------------7**

**3.1.1 中序遍历二叉排序树函数的实现-------------------------------------------------------7**

**3.1.2 删除二叉排序树函数的实现-------------------------------------------------------------7**

**3.1.3 插入结点函数的实现----------------------------------------------------------------------7**

**3.1.4 搜索二叉排序树函数的实现-------------------------------------------------------------8**

**3.1.5获得操作数字函数的实现-----------------------------------------------------------------9**

**3.1.6打印菜单函数的实现-----------------------------------------------------------------------9**

**3.2 建立二叉排序树的功能实现-------------------------------------------------------9**

**3.2.1 建立二叉排序树的核心代码------------------------------------------------------------9**

**3.2.3 建立二叉排序树的流程图 -------------------------------------------------------------11**

**3.2.3 建立二叉排序树的功能截图------------------------------------------------------------11**

**3.3 插入二叉排序树的功能实现------------------------------------------------------12**

**3.3.1 插入二叉排序树的核心代码------------------------------------------------------------12**

**3.3.2 插入二叉排序树的流程图---------------------------------------------------------------13**

**3.3.3 插入二叉排序树的功能截图-----------------------------------------------------------13**

**3.4 查找二叉排序树的功能实现------------------------------------------------------14**

**3.4.1 查找二叉排序树的核心代码------------------------------------------------------------14**

**3.4.2 查找二叉排序树的流程图---------------------------------------------------------------15**

**3.4.3 查找二叉排序树的功能截图------------------------------------------------------------15**

**3.8 主函数的实现------------------------------------------------------------------------16**

**4.项目代码功能测试------------------------------------------------------------17**

**4.1 建立二叉排序树的测试------------------------------------------------------------17**

**4.1.1 建立二叉排序树的功能测试------------------------------------------------------------17**

**4.1.2 建立二叉排序树的代码健壮性测试----------------------------------------------17**

**4.2 插入二叉排序树的测试------------------------------------------------------------18**

**4.2.1 插入二叉排序树的功能测试------------------------------------------------------------18**

**4.2.2 插入二叉排序树的代码健壮性测试----------------------------------------------18**

**4.3 查找二叉排序树的测试------------------------------------------------------------19**

**4.3.1 查找二叉排序树的功能测试------------------------------------------------------------19**

**4.3.2 查找二叉排序树的代码健壮性测试----------------------------------------------19**

**4.4 其他基本输入的健壮性测试------------------------------------------------------19**

**4.4.1 关于操作数的代码健壮性测试---------------------------------------------------------19**

**4.4.2 关于还未建树健壮性测试---------------------------------------------------------------20**

**5.项目心得与体会---------------------------------------------------------------20**

**1.项目分析**

**1.1 项目背景分析**

依次输入关键字并建立二叉排序树，实现二叉排序数的插入和查找功能。

**1.2项目功能分析**

**1.2.1项目功能要求**

二叉排序树就是指将原来已有的数据根据大小构成一棵二叉树，二叉树中的所有结点数据满足一定的大小关系，所有的左子树中的结点均比根结点小，所有的右子树的结点均比根结点大。

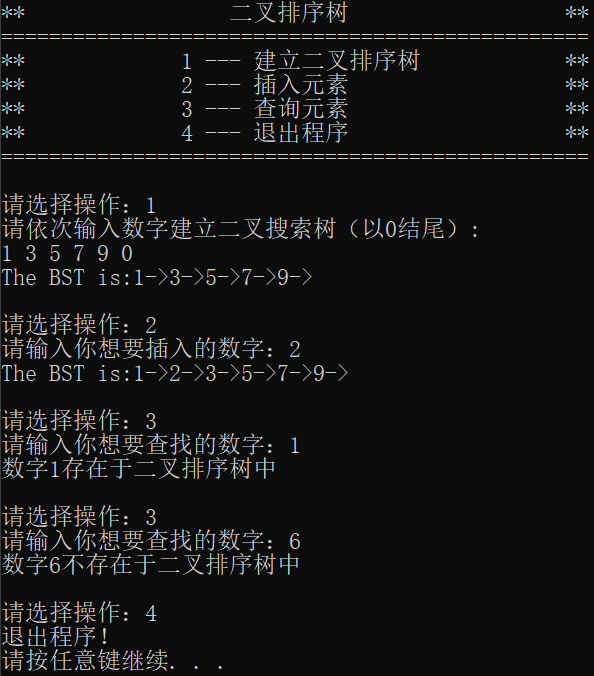
二叉排序树查找是指按照二叉排序树中结点的关系进行查找，查找关键自首先同根结点进行比较，如果相等则查找成功；如果比根节点小，则在左子树中查找；如果比根结点大，则在右子树中进行查找。这种查找方法可以快速缩小查找范围，大大减少查找关键的比较次数，从而提高查找的效率。

**1.2.2项目输入要求**

每次操作都输入相对应的操作数字，同时按提示要求输入二叉树创建所需的数字、想要插入的数字和想要查找的数字。

**1.2.3项目输出要求**

输出建立二叉排序树的内容，在插入数字后输出最新树的内容，在查找数字后显示该查找的数字是否存在于树中。在输入错误时输出相应的提示。

**1.2.4项目示例**

**2.项目设计**

**2.1 数据结构设计**

本次程序的数据结构使用二叉排序树来实现。

二叉排序树（Binary sort tree，BST），又称为二叉查找树，或者是一棵空树；或者是具有下列性质的二叉树：

(1)如果这个二叉排序树的左子树不为空，则这个二叉排序树的左子树上所有结点的值都会小于这个二叉排序树的根节点的值；

(2)如果这个二叉排序树的右子树不为空，则这个二叉排序树的右子树上所有节点的值都会大于这个二叉排序树的根节点的值；

(3)这个二叉排序树的左、右子树也分别为二叉排序树。

**2.2类设计**

二叉排序树包括两个抽象数据类型(ADT)——二叉排序树结点类(BSTnode)与二叉排序树类(BST)，而两个类之间的耦合关系可以采用嵌套、继承等多种关系。本程序也构造了这两个基本类来进行操作。

**2.2.1二叉排序树结点类(**BSTnode)

\*每一个二叉排序树结点都储存着这个结点的数据值、左指针和右指针。

class BSTnode //二叉排序树结点类

{

public:

int data; //数值

BSTnode\* leftchild; //左孩子

BSTnode\* rightchild; //右孩子

BSTnode(int number = 0, BSTnode\* leftpoint = NULL, BSTnode\* rightpoint = NULL)

{

data = number;

leftchild = leftpoint;

rightchild = rightpoint;

}

};

**2.2.2二叉排序树类**(BST)

class BST //二叉排序树类

{

public:

BSTnode\* root = NULL;

void InitTree(); //初始二叉排序树

void DeleteTree(BSTnode\* current); //删除二叉排序树

void InOrderTree(BSTnode\* current); //中序遍历二叉排序树

bool SearchNode(BSTnode\* current, int target); //搜索二叉排序树中的结点

bool InsertNode(BSTnode\* current, BSTnode\* NewBSTnode); //插入二叉排序树中的结点

void InsertOperation(); //插入操作

void SearchOperation(); //搜索操作

};

**2.3成员与操作设计**

二叉排序树类的公有成员为该结点的数据值、左指针和右指针；二叉排序树类的公有操作为初始二叉排序树、删除二叉排序树、中序遍历二叉排序树、搜索二叉排序树中的结点、插入二叉排序树中的结点这几个完成操作的基本实现函数，而剩下的两个函数：插入操作函数和搜索操作函数，是通过上面几个基本的函数操作来完成的。二叉排序树类的公有成员还有应该是根结点的指针。

**2.4系统流程设计**

系统操作的整体流程大致如下：

**程序开始运行**

**建立二叉搜索树**

**选择操作**

**（重新建立、插入、查找、退出）**

**退出程序**

**执行操作**

**显示操作结果**

**3.项目实现**

**3.1 六个基本函数的实现**

**3.1.1 中序遍历二叉排序树函数的实现**

//中序遍历二叉排序树，打印二叉排序树

void BST::InOrderTree(BSTnode\* current)

{

if (current == NULL)

{

return;

}

InOrderTree(current->leftchild);

cout << current->data << "->";

InOrderTree(current->rightchild);

}

**3.1.2 删除二叉排序树函数的实现**

//删除二叉排序树函数，运用递归，删除所有结点

void BST::DeleteTree(BSTnode\* current)

{

if (current == NULL)

{

return;

}

if (current->leftchild == NULL)

{

DeleteTree(current->leftchild);

}

if (current->rightchild == NULL)

{

DeleteTree(current->rightchild);

}

delete current;

}

**3.1.3 插入结点函数的实现**

//插入二叉排序树中的结点

bool BST::InsertNode(BSTnode\* current, BSTnode\* NewBSTnode)

{

if (current->data == NewBSTnode->data)

//如果要插入的数据在二叉树排序树中存在，return false

{

return false;

}

if (current->data > NewBSTnode->data)

//如果要插入的数据小于当前结点的数据，向它的左孩子继续查找插入

{

if (current->leftchild != NULL)

{

return InsertNode(current->leftchild, NewBSTnode);

}

current->leftchild = NewBSTnode;

}

else if (current->data < NewBSTnode->data)

//如果要插入的数据大于当前结点的数据，向它的右孩子继续查找插入

{

if (current->rightchild != NULL)

{

return InsertNode(current->rightchild, NewBSTnode);

}

current->rightchild = NewBSTnode;

}

return true;

}

**3.1.4 搜索二叉排序树函数的实现**

//运用递归搜索二叉排序树中的结点

bool BST::SearchNode(BSTnode\* current, int target)

{

if (current == NULL) //如果找不到，return false

{

return false;

}

if (current->data == target) //如果找到了，return true

{

return true;

}

if (target < current->data) /

/如果要找的数据小于当前结点数据，则向它的左孩子继续搜索查找

{

return SearchNode(current->leftchild, target);

}

else

{

return SearchNode(current->rightchild, target);

//如果要找的数据大于当前结点数据，则向它的右孩子继续搜索查找

}

}

**3.1.5 获得操作数字函数的实现**

//get\_choice函数，完成输入操作数字的健壮性检验，同时返回一个操作数字

int get\_choice()

{

cout << endl << "请选择操作：";

int choice;

while (1)

{

cin >> choice;

if (cin.good() == 0 || choice < 1 || choice > 4)

{

cin.clear();

cin.ignore(1024, '\n');

cout << "输入错误，请重新输入：";

continue;

}

break;

}

return choice;

}

**3.1.6 打印菜单函数的实现**

//Print\_Menu函数，打印菜单并显示

void Print\_Menu()

{

cout << "\*\* 二叉排序树 \*\*" << endl;

cout << "=================================================" << endl;

cout << "\*\* 1 --- 建立二叉排序树 \*\*" << endl;

cout << "\*\* 2 --- 插入元素 \*\*" << endl;

cout << "\*\* 3 --- 查询元素 \*\*" << endl;

cout << "\*\* 4 --- 退出程序 \*\*" << endl;

cout << "=================================================" << endl;

}

**3.2 建立二叉排序树的功能实现**

**3.2.1 建立二叉排序树的核心代码**

//初始二叉排序树

void BST::InitTree()

{

DeleteTree(root); //每次创建新的二叉排序树前都要将原来的二叉排序树清空

root = NULL;

cout << "请依次输入数字建立二叉搜索树（以0结尾）:" << endl;

while (1)

{

int value;

cin >> value;

if (cin.good() == 0) {

DeleteTree(root);

BSTnode\* root = NULL;

cout << "输入错误！请重新输入: " << endl;

cin.clear();

cin.ignore(1024, '\n');

continue; //代码的健壮性检验

}

if (value == 0)

{

break;

}

BSTnode\* NewBSTnode = new BSTnode;

NewBSTnode->data = value;

if (root == NULL)

{

root = NewBSTnode; //如果一开始是空树，就让根结点为第一个输入的数据

}

else

{

bool operation = InsertNode(root, NewBSTnode);

//调用InsertNode在根结点创建之后继续插入数据

if (operation == 0)

{

cout << "数值" << value << "多次输入，视为输入一次。" << endl;

//代码的健壮性，如果一个数值输入多次，视为输入一次

}

}

}

}

建立二叉树排序树是对二叉树排序树的初始化，以免每次建立二叉排序树的数据冲突，所以在每一次初始建立二叉排序树的时候首先将原有的二叉排序树信息全部删除。(无论本来有没有二叉排序树存在)在输入二叉排序树的数值的时候首先对输入的数据进行了健壮性检验。其次在建立的过程中，我们第一个输入的数据应该存放在根结点中，所以我们对这一情况进行了特殊徐判断。之后每插入应该新的二叉排序树结点，我们首先对其的数据类型进行检查，之后调用插入函数一个一个地创建新的二叉排序树。(以0结尾表示输入完成，在题目的示例中有给出以0为标志完成输入操作)

**3.2.2 建立二叉排序树的流程图**

建立二叉排序树的整体流程大致如下：

**开始**

**输入数字**

**判断根结点是否为NULL**

**否**

**是**

**InsertNode插入**

**赋值给根结点**

**否**

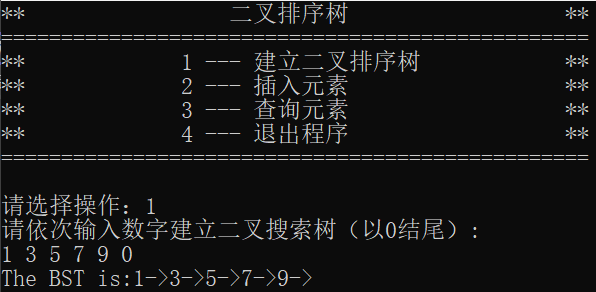
**是**

**完成创建**

**显示二叉树**

**判断输入是否为0**

**3.2.3 建立二叉排序树的功能截图**



**3.3 插入二叉排序树的功能实现**

**3.3.1 插入二叉排序树的核心代码**

//插入数据的操作函数

void BST::InsertOperation()

{

if (root == NULL) //如果是空树，则提示要首先建立二叉排序树

{

cout << "请首先建立二叉排序树！";

return;

}

int number;

cout << "请输入你想要插入的数字：";

while (1)

{

cin >> number;

if (cin.good() == 0)

{

cin.clear();

cin.ignore(1024, '\n');

cout << "输入错误！请重新输入：";

continue;

}

break;

} //代码的健壮性检验

BSTnode\* NewBSTnode = new BSTnode;

NewBSTnode->data = number;

bool insertoperation = InsertNode(root, NewBSTnode); //调用InsertNode插入数据

if (insertoperation == 0)

{

cout << "数值" << number << "已经存在于二叉排序树中" << endl; //如果数据存在则会提示

}

return;

}

插入二叉树排序树结点完成菜单任务的第二项功能。首先在插入操作开始之前，我们都要判断一下二叉排序树存不存在，如果不存在的话会给出相应的提示信息，要修我们首先构造二叉排序树。在构建完二叉排序树之后，我们接下来对输入的插入数字进行了检验：第一，输入的数据类型是否正确；第二，输入的数字是否在二叉排序树中已经存在，所以，在这两个操作之后，再次调用InsertNode函数将输入的数字插入道二叉排序树中正确的位置。

**3.3.2 插入二叉排序树的流程图**

插入二叉排序树的整体流程大致如下：

**开始**

**输入数字**

**判断数据类型和数字存在性**

**不满足条件是**

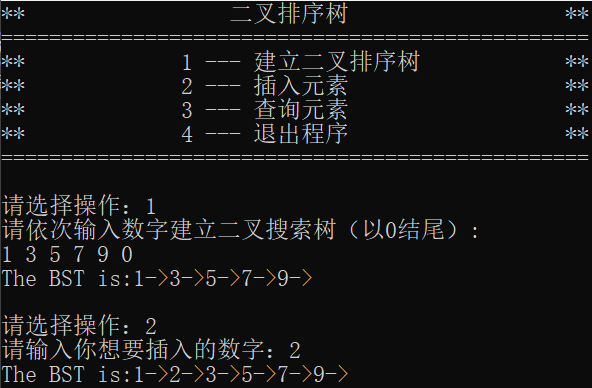
**满足条件**

**完成插入**

**显示二叉树**

**InsertNode插入**

**3.3.3 插入二叉排序树的功能截图**



**3.4 查找二叉排序树的功能实现**

**3.4.1 查找二叉排序树的核心代码**

//查找数据的操作函数

void BST::SearchOperation()

{

if (root == NULL) //如果是空树，则提示要首先建立二叉排序树

{

cout << "请首先建立二叉排序树！";

return;

}

int number;

cout << "请输入你想要查找的数字：";

while (1)

{

cin >> number;

if (cin.good() == 0)

{

cin.clear();

cin.ignore(1024, '\n');

cout << "输入错误！请重新输入：";

continue;

}

break;

} //代码的健壮性检验

bool searchoperation = SearchNode(root, number); //调用SearchNode函数查找该数据是否存在

if (searchoperation == 1)

{

cout << "数字" << number << "存在于二叉排序树中";

}

else

{

cout << "数字" << number << "不存在于二叉排序树中";

}

}

查找二叉树排序树结点完成菜单任务的第三项功能。首先在查找操作开始之前，我们都要判断一下二叉排序树存不存在，如果不存在的话会给出相应的提示信息，要修我们首先构造二叉排序树。在构建完二叉排序树之后，我们接下来对输入的查找数字进行了检验，如果输入的数据类型错误，则会给出提示。之后调用SearchNode函数对整个二叉排序树进行递归查找，查找是否存在这一个数字并且给出查找完最后的结论。

**3.4.2 查找二叉排序树的流程图**

**开始**

**输入数字**

**数据类型有误**

**判断数据类型**

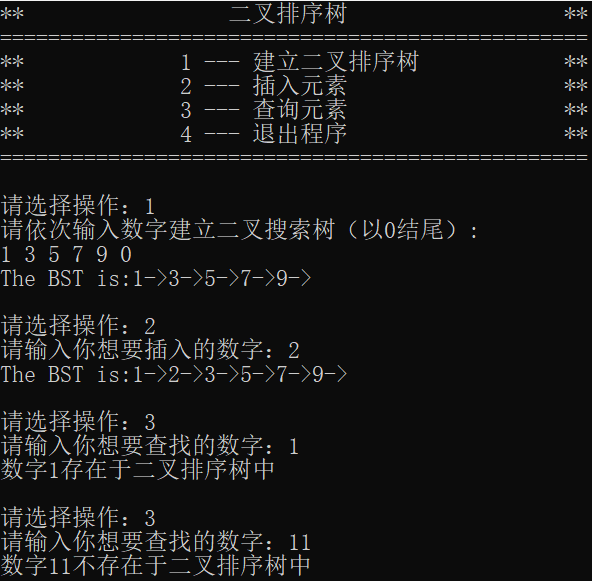
**数据类型无误**

**SearchNode查找**

**不存在**

**存在**

**3.4.3 查找二叉排序树的功能截图**



**3.5 主函数的功能实现**

//主函数，程序的主体逻辑结构

int main()

{

BST bst; //实例化BST（二叉排序树）类

Print\_Menu(); //打印菜单

int keydown;

while (1)

{

keydown = get\_choice();

switch (keydown)

{

case 1: //建立二叉排序树

bst.InitTree();

cout << "The BST is:";

bst.InOrderTree(bst.root);

cout << endl;

break;

case 2: //插入操作

bst.InsertOperation();

if (bst.root != 0)

{

cout << "The BST is:";

bst.InOrderTree(bst.root);

}

cout << endl;

break;

case 3: //查找操作

bst.SearchOperation();

cout << endl;

break;

case 4: //退出程序

cout << "退出程序！" << endl;

system("pause");

return 0;

}

}

return 0;

}

主函数的主要是程序主体的逻辑结构。在主函数中首先实例化了一个二叉排序树bst，然后通过switch-case语句对想要进行的操作进行选择，再调用二叉排序树的各种成员函数对操作进行一步步的完成。

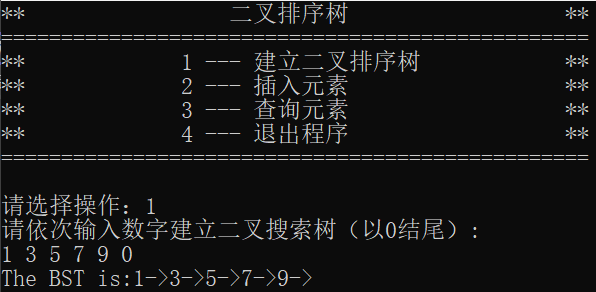
**4.项目代码功能测试**

**4.1 建立二叉排序树的测试**

**4.1.1 建立二叉排序树的功能测试**

**测试用例：** 1 3 5 7 9 0

**预期结果：** 1-> 3-> 5 ->7-> 9->

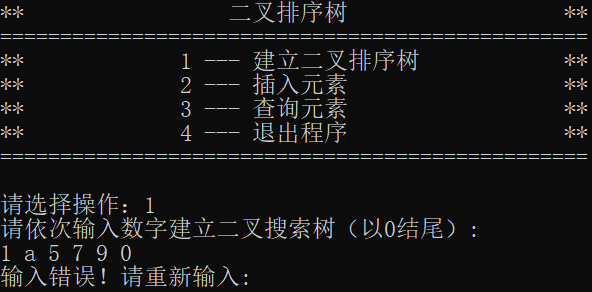
**实验结果：**

**4.1.2 建立二叉排序树的代码健壮性测试**

1.但是在输入数据的时候输入的不是数字

**测试用例：** 1 a 5 7 9 0

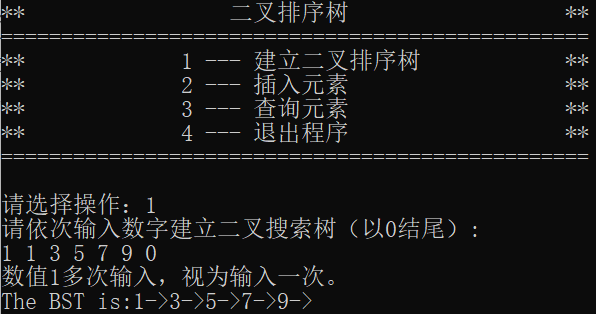
**预期结果：** 给出提示，重新输入

**实验结果：**

2.但是在输入数据的时候输入的某个数字出现了多次

**测试用例：** 1 1 3 5 7 9 0

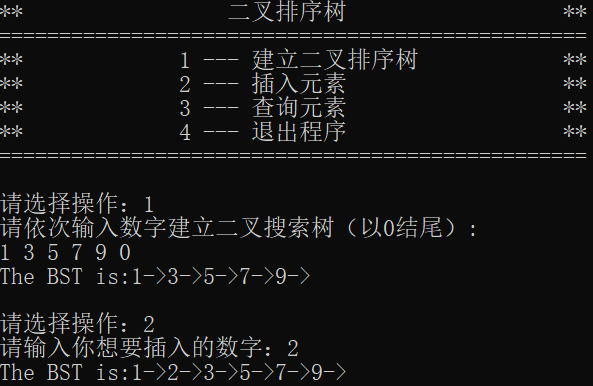
**预期结果：** 给出提示，某个数字输入多次，并且显示正确的二叉排序树

**实验结果：**

**4.2 插入数字的测试**

**4.2.1 插入数字的功能测试**

**测试用例：** 2

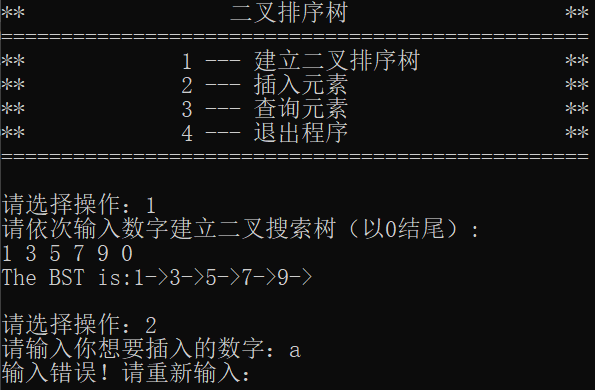
**预期结果：** 1-> 2->3-> 5 ->7-> 9->

**实验结果：**

**4.2.2 插入数字的代码健壮性测试**

1.但是在输入数据的时候输入的不是数字

**测试用例：** a

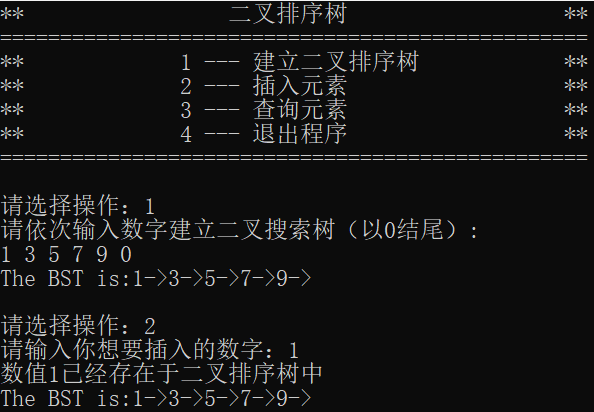
**预期结果：** 给出提示，重新输入

**实验结果：**

2.但是输入的数据在二叉树中已经存在

**测试用例：** 1

**预期结果：** 给出提示，数字存在；并显示二叉排序树内容

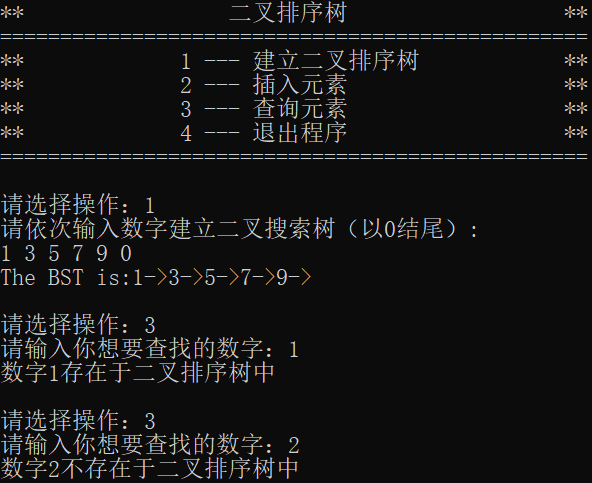
**实验结果：**

**4.3 查找数字的测试**

**4.3.1 查找数字的功能测试**

**测试用例：** 1 3 5 7 9 0 查找数字1和数字2

**预期结果：** 数字1存在于二叉排序树中，数字2不存在于二叉排序树中

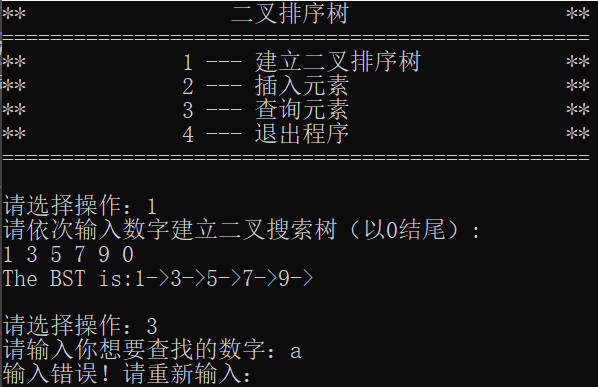
**实验结果：**

**4.3.2 查找数字的代码健壮性测试**

但是在输入要查找数据的时候输入的不是数字

**测试用例：** 1 3 5 7 9 0 查找a

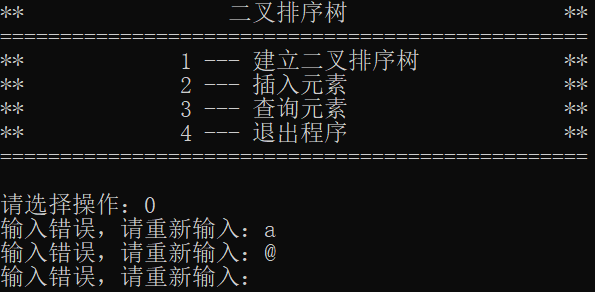
**预期结果：** 给出提示，重新输入

**实验结果：**

**4.4 其他项目的代码健壮性测试**

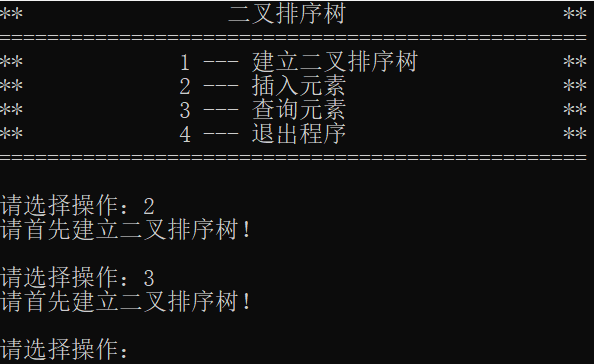
**4.4.1 关于操作数的代码健壮性测试**

操作数的数字范围只能在1 2 3 4这四个数字中挑选，如果输入的不是在这个范围内的数字或是其他类型的数据，则会给出相应的提示

**测试截图：**

**4.4.2 关于还未建树健壮性测试**

该程序的第一步肯定是要建立相应的二叉排序树，如果在还没有选择选项1建立二叉排序树之前选择了2 3操作，则会给出相应的提示。

**测试截图：**

**5.项目心得与体会**

树是继链表之后有又一个较为复杂的数据结构，但是在编写二叉排序树程序的时候，还是利用链表来实现，所以之前的链表的练习为之后写树的数据结构奠定了一些基础。写二叉排序树的更加复杂，主要是加入了递归的思想来遍历或者插入结点，而且要注意多种特殊情况的判断，比如根结点是否为NULL、遍历所在的那一个结点的左右孩子是否存在和遍历的当前结点是否存在等等特殊情况，所以问题就会比较复杂，在继续创建、插入和搜索操作的时候我们就需要对特殊情况进行很多的判断，加大了思维量。因此，这是我第一次以C++类的方式来书写二叉排序树，初步了解了二叉树的相关性质和代码实现的过程，在日后的学习中还需多加练习来对二叉树的各类操作更加熟练。