哈尔滨工业大学（深圳）

数据结构实验报告

树形结构及其应用

学　院：　　计算机科学与技术

姓　名：　　胡博涵

学　号：　　SZ170110113

专　业：　　计算机类

日　期：　　2018.5.18

一、问题分析

任务一要解决的问题是根据给定的二叉树的先序遍历和中序遍历序列，用链式存储结构重构这棵二叉树。这题的给定的两个序列，就唯一确定了一个二叉树的结构，可以根据序列，分别确定左子树、右子树的两个遍历序列，从而不断地细化进行下去，这和递归的定义很类似，所以采用设计递归函数的方法来进行二叉树的重构。

任务二要解决的问题是将给定的带括号的中缀运算表达式以树状形式存储，并输出其后缀和前缀表达式形式，完成其计算。在计算机中，涉及到符号的优先级问题，需要我们设计一个数据结构来进行优先级的判定、比较和操作的执行。对表达式树的求值，是一个递归定义的过程，因此要设计一个递归函数来进行求解操作。

二、详细设计

2.1 设计思想

任务一：给定一棵二叉树的先序遍历序列和中序遍历序列，可以唯一确定一个二叉树的结构。其思想是：先序遍历序列第一个节点，是整个二叉树的根节点。这样，在中序遍历中找到该节点的位置，就可以把中序遍历序列切分成左子树的中序遍历序列 In_L 和右子树的中序遍历序列 In_R，根据中序遍历左右子树序列的长度，删除先序遍历的第一个节点，将先序遍历序列切分左子树 Pre_L 和右子树 Pre_R。再根据左右子树的两个序列，递归地建立左右两个子树。

任务二：将中缀表达式转换为表达式树，最重要的是对表达式树结构的分析。在表达式树中，每一个子树代表着一次运算。优先级越高的运算，在表达式树中的深度越深。在对表达式进行求值时，应从树的最深处开始求值，先求出子树的值，再返回上一层进行计算。因此，优先级最高的，应当位于表达式树的最深处，优先级最低的，应当位于表达式的根节点。

（0）运算符优先级的判断

采用二维表的形式存储，第一维（纵）对应栈顶运算符，第二维（横）对应当前运算符。对括号的处理：括号对表达式具有隔离作用，因此若当前位置为左括号，左括号优先级应当最高，以便优先让括号内的式子计算，因此应当入栈。左括号位于栈顶时，应处于最低的优先级，以便让其后的表达式顺利计算，右括号不应该入栈，应处于最低的优先级，与左括号同等的优先级，待完成运算后，才可以和左括号一同弹出。此外，对于非法的表达式，也做了一些容错处理。

优先级表：

表格采用二维数组进行存储		当前运算符						
		+	-	*	/	(	)	\0
栈顶运算符	+	>	>	<	<	<	>	>
	-	>	>	<	<	<	>	>
	*	>	>	>	>	<	>	>
	/	>	>	>	>	<	>	>
	(	<	<	<	<	<	=	不合法
	)	不合法	不合法	不合法	不合法	不合法	不合法	不合法
	\	<	<	<	<	<	不合法	=

```
const char prior[NUM_OPTR][NUM_OPTR] = { //运算符优先级次序判断[栈顶][目前]
    //当前所指运算符（横），栈顶所指为列
              /*              +    -    *    /    (    )    \0   */
             /* + */       '>', '>', '<', '<', '<', '>', '>',
             /* - */       '>', '>', '<', '<', '<', '>', '>',
             /* * */       '>', '>', '>', '>', '<', '>', '>',
             /* / */       '>', '>', '>', '>', '<', '>', '>',
             /* ( */       '<', '<', '<', '<', '<', '=', ' ',
             /* ) */       ' ', ' ', ' ', ' ', ' ', ' ', ' ', //右括号不可能在栈顶
             /* \0 */      '<', '<', '<', '<', '<', ' ', '<',
};
```

（1）表达式树的建立

设立两个栈，一个是表达式栈 Exp，另一个是操作符栈 Op。线性扫描表达式。

①碰到操作数时，封装操作数为一个树节点，推入表达式栈 Exp。

②碰到运算符时，判断与操作符栈 Op 栈顶运算符的优先级关系

i)若是优先级高于栈顶，则栈顶运算符需要推迟运算，当前运算符入栈

ii)若是优先级低于栈顶，则栈顶的运算符可以立即计算，即需要立刻组成一个子树。此时，弹出 Op 栈顶的运算符，弹出 Exp 栈顶的两个子表达式树，组成一个二叉树，将根节点推入 Exp 栈。重复以上操作，直至运算符栈 Op 为空。

（2）表达式树的求值

根据表达式树的定义，任意一个子树的值等于其左子树的值和其右子树的值，经过根节点运算符运算后的结果。因此，只需设计一个函数，计算左右子树的值，并根据根节点的操作，进行整个表达式子树的计算。

（3）表达式的前/中/后缀转换

只需输出表达式树的前序/后序遍历结果即可。

2.2 存储结构及操作

(1) 存储结构

结构体 BinNode（二叉树节点类型）	存储数据	数据域 Data	整形/字符型联合体
	连接左子	左子指针 lChild	结构体指针
	连接右子	右子指针 rChild	结构体指针
顺序栈 Stack	存储数据	数据域 Data	BinNode 指针
	指向栈顶	栈顶位置 top	整形
联合体 Data	方便统一接口， 存储不同的数据类型	运算符 char	类型只能是两者之一
		运算数 number	

(2) 涉及的操作

函数名	传入参数	函数功能	返回值
操作对象：顺序栈			
initStack	无	创建一个栈	栈指针 S
push	栈指针 S、操作元素 A	将 A 压入栈 S	无
pop	栈指针 S	将 S 顶部元素弹出	弹出的元素
top	栈指针 S	无	S 顶部的元素
isEmpty	栈指针 S	返回栈是否为空	真值
操作对象：二叉树节点			
initbinNode	数据联合体 Data	创建一个二叉树节点	节点指针 p
initbinNode_num	双精度浮点数 number	调用 initbinNode 创建存储数字的二叉树节点	节点指针 p
initbinNode_op	char 类型	调用 initbinNode 创建存储运算符的二叉树节点	节点指针 p
deletebinTree	根节点指针 root	递归释放整个二叉树	无
Traverse_Pre_re	根节点指针 root	递归先序遍历二叉树	无
Traverse_In_re	根节点指针 root	递归中序遍历二叉树	无
Traverse_Pos_re	根节点指针 root	递归后序遍历二叉树	无
calculate	表达式树根节点 root	递归计算左右子树的值，并根据根节点的运算符进行运算	计算结果
printTree	根节点 root	输出树的结构	无

其余函数：

任务一：

cut

功能：截取字符串的部分（便于截取遍历序列的子树部分）

传入参数：待截取的字符串、起始位置（闭合）、结束位置（开）

返回值：截取后的子序列

buildBinTree

功能：根据传入的序列（子序列），建立相应的二叉树

传入参数：先序遍历序列、中序遍历序列、先序遍历区间起点、先序遍历序列区间终点、中序 遍历序列区间起点、中序遍历序列区间终点

返回值：根据该序列重构的二叉树

isSym

功能：判断二叉树是否镜像对称

传入参数：二叉树左子和右子

返回值：布尔值

任务二：

readDigit

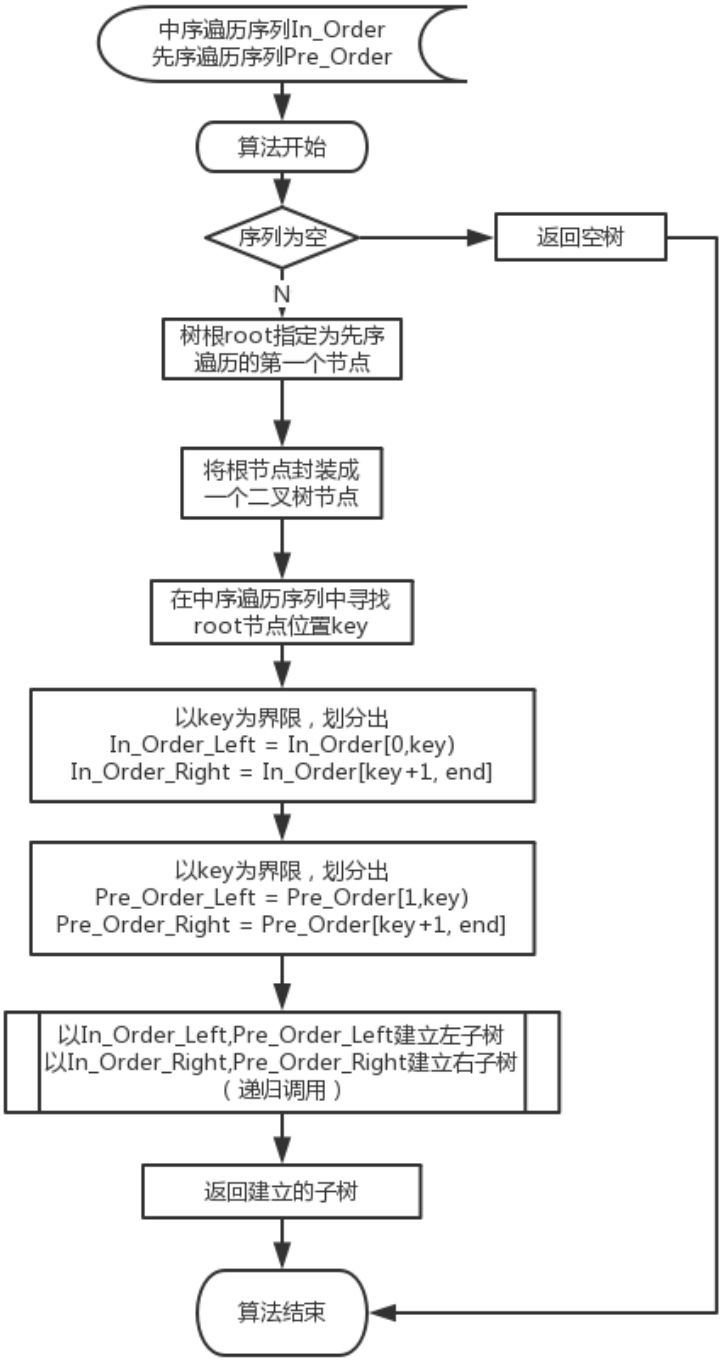
功能：在表达式字符串序列中，从当前传入的指针位置开始，读入一个数字，并把指针移动到下一个运算符的位置

传入参数：字符串指针的二级指针

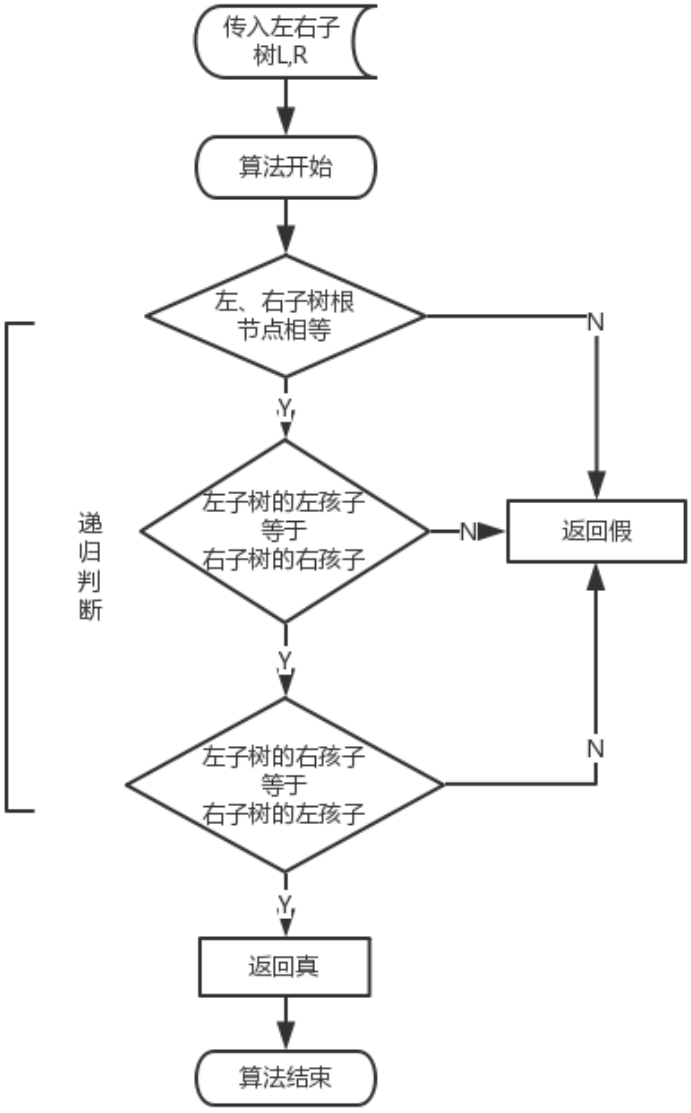
返回值：读取到的数字

2.3 程序整体流程

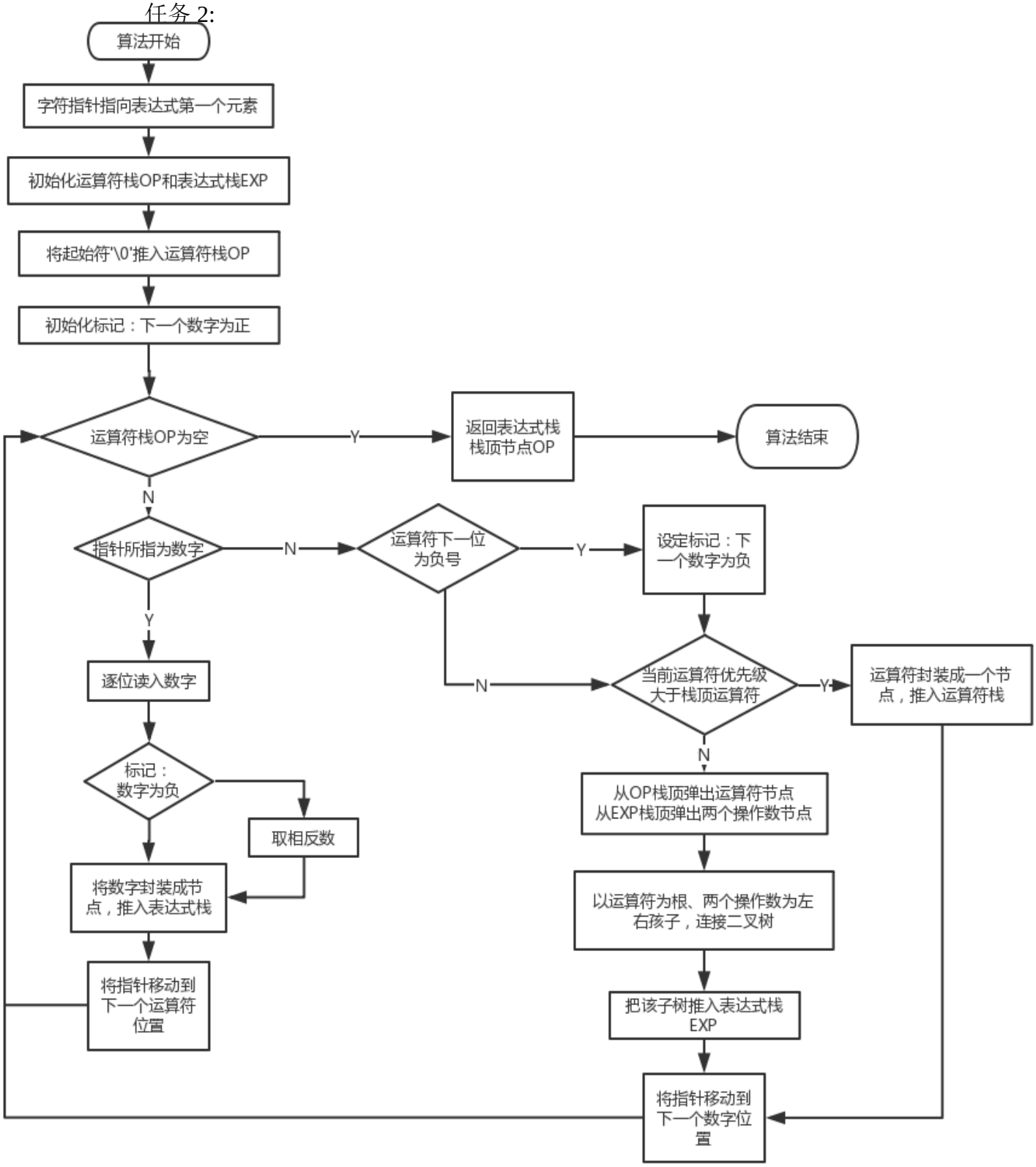
任务 1: 重构二叉树



判断对称性：


传入左右子树L,R
算法开始
左、右子树根节点相等
N
Y
递归判断
左子树的左孩子等于右子树的右孩子
N
返回假
Y
左子树的右孩子等于右子树的左孩子
N
Y
返回真
算法结束

任务 2:


算法开始
字符指针指向表达式第一个元素
初始化运算符栈OP和表达式栈EXP
将起始符'\0'推入运算符栈OP
初始化标记：下一个数字为正
运算符栈OP为空
Y
返回表达式栈栈顶节点OP
算法结束
N
指针所指为数字
N
运算符下一位为负号
Y
设定标记：下一个数字为负
Y
逐位读入数字
N
当前运算符优先级大于栈顶运算符
Y
运算符封装成一个节点，推入运算符栈
标记：数字为负
取相反数
N
从OP栈顶弹出运算符节点
从EXP栈顶弹出两个操作数节点
将数字封装成节点，推入表达式栈
以运算符为根、两个操作数为左右孩子，连接二叉树
将指针移动到下一个运算符位置
把该子树推入表达式栈EXP
将指针移动到下一个数字位置

三、用户手册

任务 1:

输入数据的方式：输入两个序列，依次为该二叉树的先序遍历序列和后序遍历序列。

输出数据的方式：将会输出这个树的结构，并输出该树是否为镜像对称。

任务 2:

输入数据的方式：输入一个合法的中缀表达式。

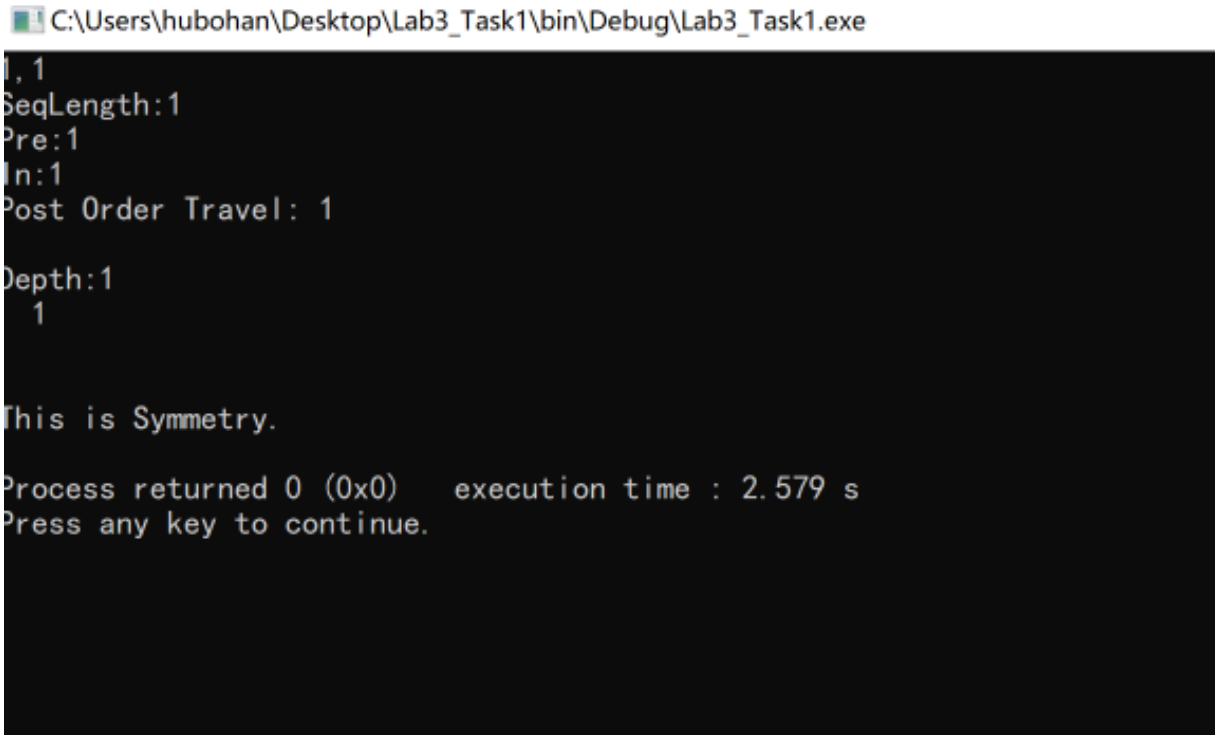
输出数据的方式：输出转换后的后缀表达式和前缀表达式，输出表达式树的结构，并且输出这个表达式的计算结果。

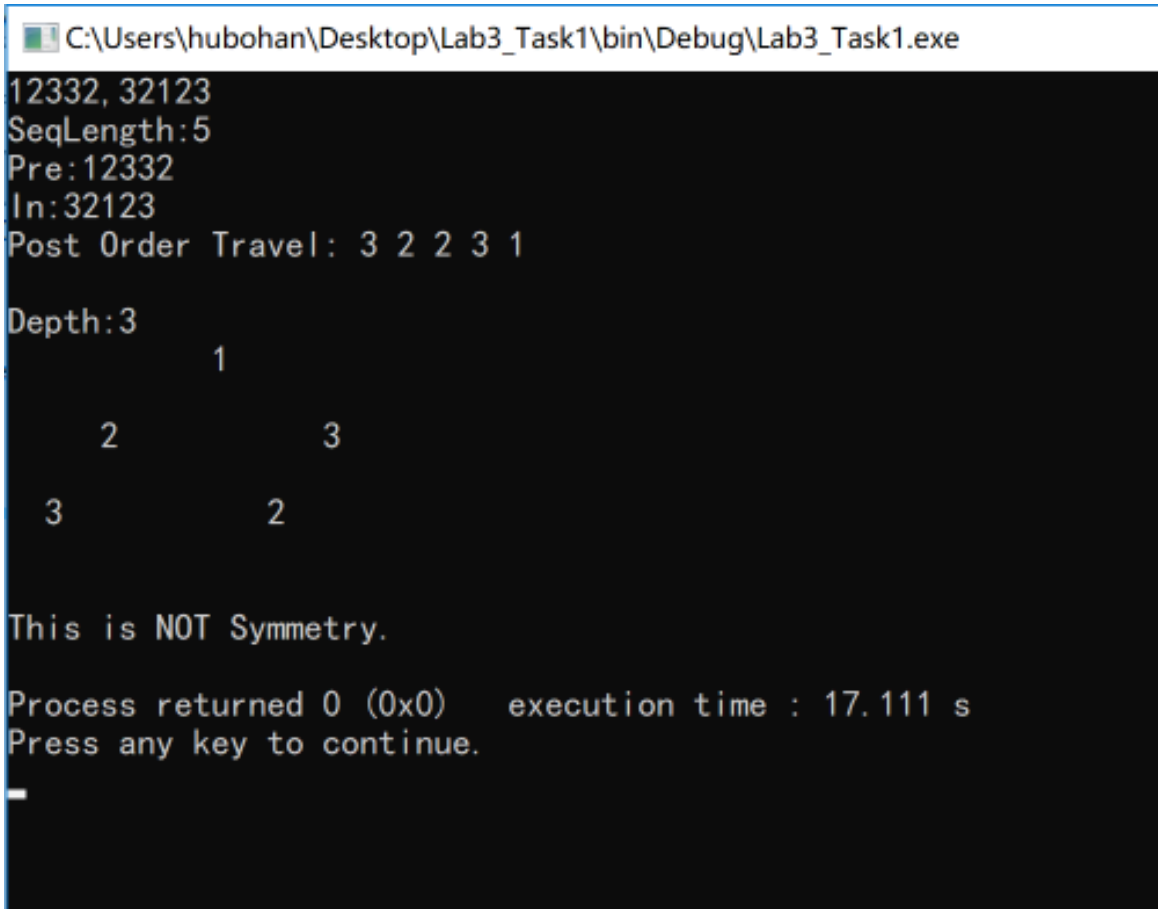
四、总结

该实验涉及到的数据结构有栈、二叉树。其中栈已经在之前的实验中使用过很多次，在这次实验中，主要是起到了延迟缓冲的作用——即在不能马上断定某个操作是否能执行的时候，需要先缓存该指令，等待执行的时机。二叉树主要起到了存储数据的作用。在这次实验中，涉及的算法有二叉树的前序、中序、后序遍历，给定二叉树的序列，重构二叉树。主要的思想是递归算法，因为二叉树就是递归定义的概念。大到整个二叉树，小到每一个叶子节点都可以看作一个二叉树，每一个二叉树都是由其根节点、左子树、右子树构成，这为我们使用递归的思想编写算法提供了帮助。在重构二叉树时，已知先序遍历或后序遍历，加之中序遍历，可以查找树根，重构二叉树，也是一种典型的递归算法。在实验中，由于采用链式存储结构，容易造成在递归的函数中未处理好传入的参数，造成调用时访问空指针程序崩溃的情况，需要对这种情况在函数入口处进行处理。

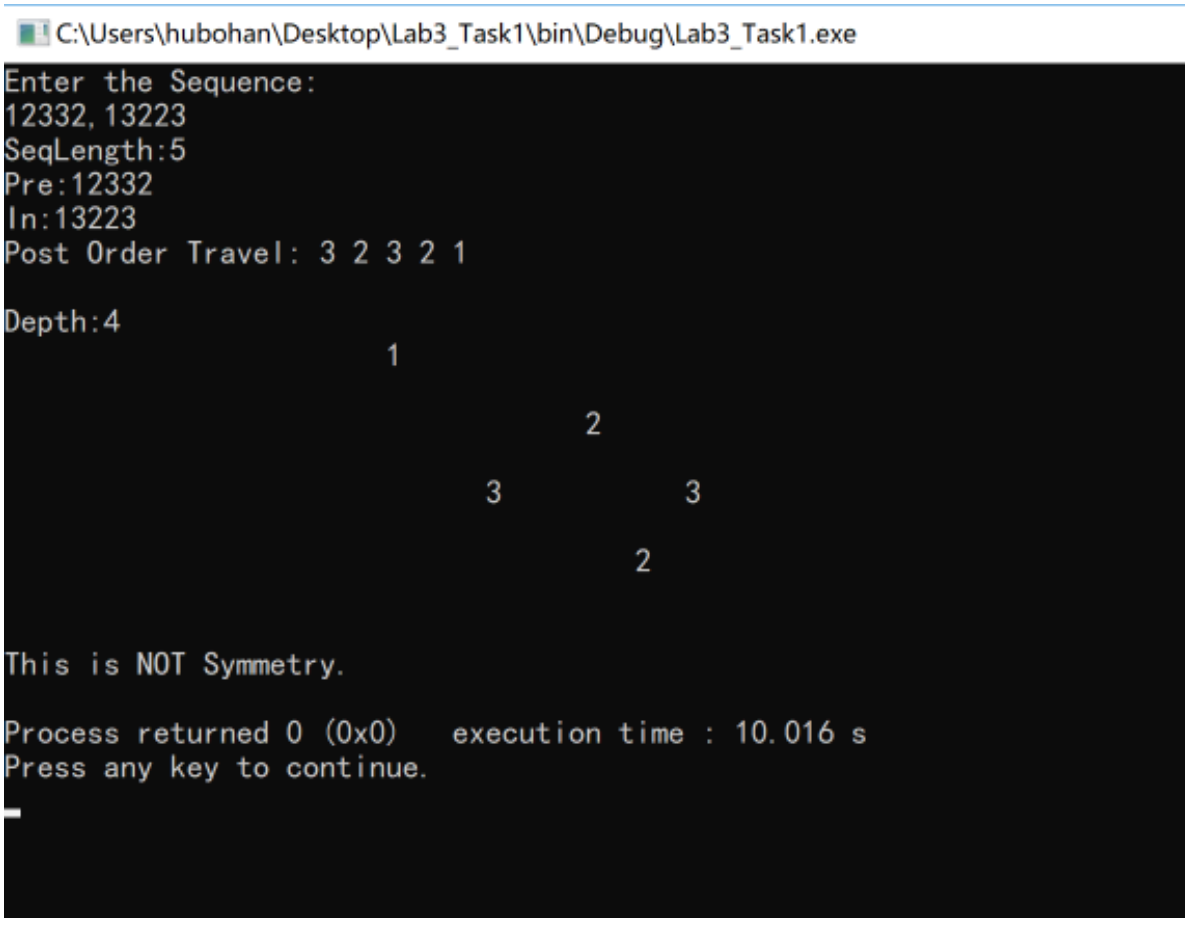
五、结果

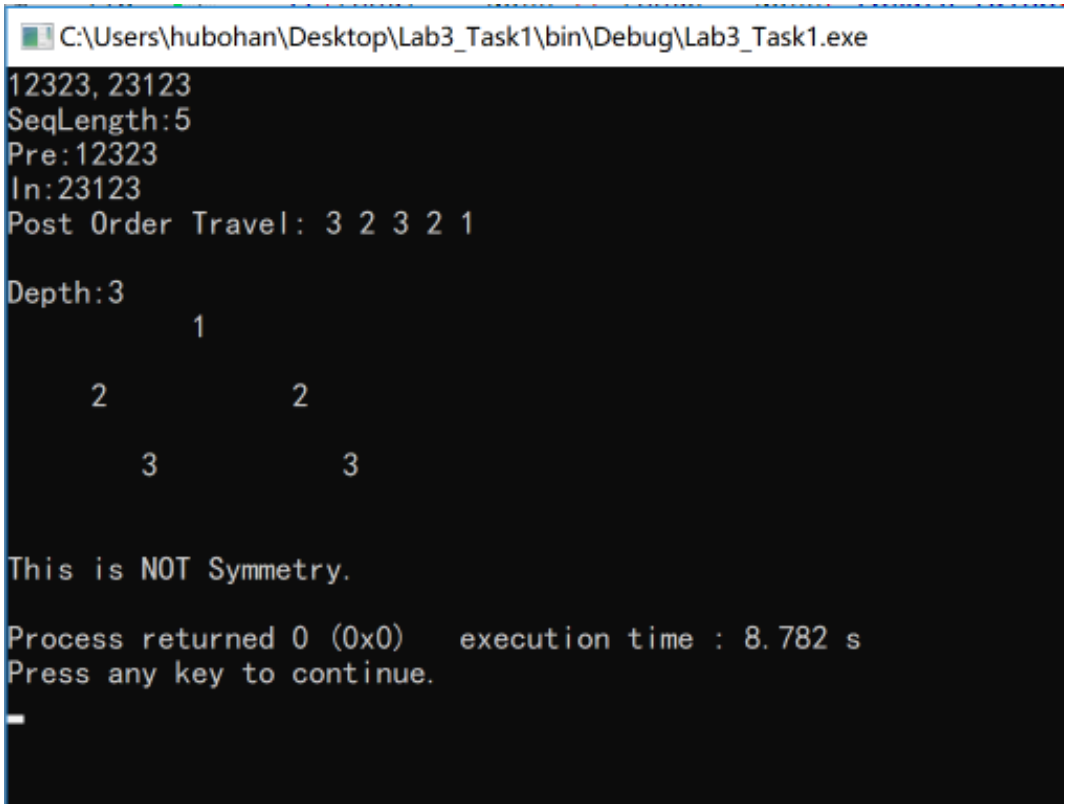
任务 1:


C:\Users\hubohan\Desktop\Lab3_Task1\bin\Debug\Lab3_Task1.exe
1,1
SeqLength:1
Pre:1
In:1
Post Order Travel: 1
Depth:1
1
This is Symmetry.
Process returned 0 (0x0) execution time : 2.579 s
Press any key to continue.

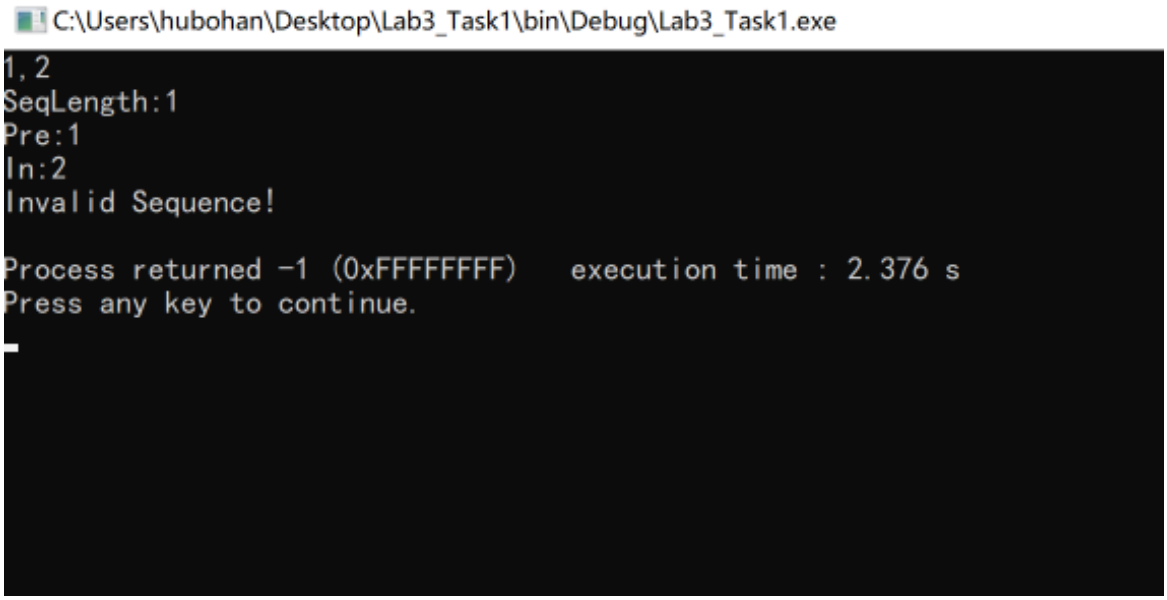

C:\Users\hubohan\Desktop\Lab3_Task1\bin\Debug\Lab3_Task1.exe
12332,32123
SeqLength:5
Pre:12332
In:32123
Post Order Travel: 3 2 2 3 1
Depth:3
1
2 3
3 2
This is NOT Symmetry.
Process returned 0 (0x0) execution time : 17.111 s
Press any key to continue.


C:\Users\hubohan\Desktop\Lab3_Task1\bin\Debug\Lab3_Task1.exe
1234243,3241423
SeqLength:7
Pre:1234243
In:3241423
Post Order Travel: 3 4 2 4 3 2 1
Depth:3
1
2 2
3 4 4 3
This is Symmetry.
Process returned 0 (0x0) execution time : 8.719 s
Press any key to continue.


C:\Users\hubohan\Desktop\Lab3_Task1\bin\Debug\Lab3_Task1.exe
Enter the Sequence:
12332,13223
SeqLength:5
Pre:12332
In:13223
Post Order Travel: 3 2 3 2 1
Depth:4
1
2
3 3
2
This is NOT Symmetry.
Process returned 0 (0x0) execution time : 10.016 s
Press any key to continue.


C:\Users\hubohan\Desktop\Lab3_Task1\bin\Debug\Lab3_Task1.exe
12323,23123
SeqLength:5
Pre:12323
In:23123
Post Order Travel: 3 2 3 2 1
Depth:3
1
2
2
3
3
This is NOT Symmetry.
Process returned 0 (0x0) execution time : 8.782 s
Press any key to continue.

异常处理：


C:\Users\hubohan\Desktop\Lab3_Task1\bin\Debug\Lab3_Task1.exe
1,2
SeqLength:1
Pre:1
In:2
Invalid Sequence!
Process returned -1 (0xFFFFFFFF) execution time : 2.376 s
Press any key to continue.

任务 2:

C:\Users\hubohan\Desktop\Lab3_Task2\bin\Debug\Lab3_Task2.exe

```
Input the expression:34+5-(8+4)*6
Prefix Expression: - + 34.0 5.0 * + 8.0 4.0 6.0
Postfix Expression: 34.0 5.0 + 8.0 4.0 + 6.0 * -
34+5-(8+4)*6 = -33.0
Expression Tree:

Depth:4
                        -

          +                           *

    34            5             +             6

                            8       4


Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

C:\Users\hubohan\Desktop\Lab3_ALL\Lab3_Task2\bin\Debug\Lab3_Task2.exe

```
Input the expression:1+-2*-3/-4+--5*(9*-10)
Prefix Expression: - - + 1 * -2 -3 -4 * 5 * 9 -10
Postfix Expression: 1 -2 -3 * + -4 - 5 9 -10 * * -
1+-2*-3/-4+--5*(9*-10) = 461
Expression Tree:

Depth:5
                                        -

                    -                                       *

         +                    -4                    5                    *

    1          *                                                  9         -10

           -2    -3


Process returned 0 (0x0)   execution time : 20.562 s
Press any key to continue.
```

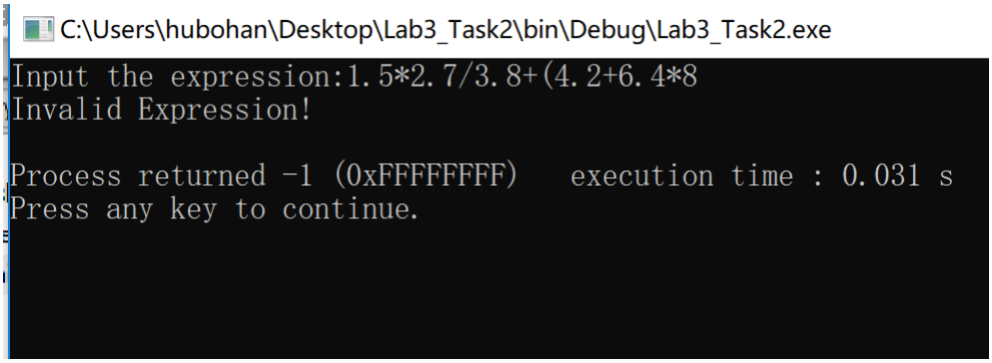

C:\Users\hubohan\Desktop\Lab3_Task2\bin\Debug\Lab3_Task2.exe
Input the expression:34+-----5-(-8+--4)*6
Prefix Expression: - + 34.0 -5.0 * + -8.0 4.0 6.0
Postfix Expression: 34.0 -5.0 + -8.0 4.0 + 6.0 * -
34+-----5-(-8+--4)*6 = 53.0
Expression Tree:
Depth:4
-
+ *
34 -5 + 6
-8 4
Process returned 0 (0x0) execution time : 0.047 s
Press any key to continue.


C:\Users\hubohan\Desktop\Lab3_Task2\bin\Debug\Lab3_Task2.exe
Input the expression:1+2*3/(4+6*8)
Prefix Expression: + 1.0 / * 2.0 3.0 + 4.0 * 6.0 8.0
Postfix Expression: 1.0 2.0 3.0 * 4.0 6.0 8.0 * + / +
1+2*3/(4+6*8) = 1.1
Expression Tree:
Depth:5
+
1 /
* +
2 3 4 *
6 8
Process returned 0 (0x0) execution time : 0.062 s
Press any key to continue.

小数点处理：


C:\Users\hubohan\Desktop\Lab3_Task2\bin\Debug\Lab3_Task2.exe
Input the expression:1.5*2.7/3.8+(4.2+6.4*8)
Prefix Expression: + / * 1.5 2.7 3.8 + 4.2 * 6.4 8.0
Postfix Expression: 1.5 2.7 * 3.8 / 4.2 6.4 8.0 * + +
1.5*2.7/3.8+(4.2+6.4*8) = 56.5
Expression Tree:
Depth:4
+
/ +
* 3.8 4.2 *
1.5 2.7 6.4 8.0
Process returned 0 (0x0) execution time : 0.047 s
Press any key to continue.

异常表达式处理：


C:\Users\hubohan\Desktop\Lab3_Task2\bin\Debug\Lab3_Task2.exe
Input the expression:1.5*2.7/3.8+(4.2+6.4*8
Invalid Expression!
Process returned -1 (0xFFFFFFFF) execution time : 0.031 s
Press any key to continue.