作业5.1

AVL 树的设计与实现

本次作业要求设计 AVL 树存储结构，并实现 AVL 树建立（插入）、删除和查找算法，并反映插入和删除操作算法的各种旋转变化。

存储结构

```
class AVL
{
private:
    //AVL树节点的存储结构
    struct Node
    {
        int Data, Size, Height;
        Node *LCh, *RCh;
        Node()
        {
            LCh = RCh = NULL;
            Size = Height = 0;
        }
    };
public:
    typedef Node * NodeP;
    NodeP Root;
};
```

函数说明

1. 设计 AVL 的左右链存储结构；

```
struct Node
    {
        int Data, Size, Height;
        Node *LCh, *RCh;
        Node()
        {
            LCh = RCh = NULL;
            Size = Height = 0;
        }
    };
```

2. 实现 AVL 左右链存储结构上的插入（建立）、删除、查找和排序算法。

插入：`void Insert(NodeP &x, int Data)` 插入值为 `Data` 的节点；

删除：`int Delete(NodeP &x, int Data)` 删除值为 `Data` 的节点；

查找：`int Rank(NodeP x, int Data)` 查找 `Data` 在树中的次序；

排序：`string GetSorted()` 输出排好序的序列。

3. 测试数据以文件形式保存，能反映插入和删除操作的四种旋转，并输出相应结果。

`freopen("Homework5_In.txt", "r", stdin)` `freopen("Homework5_Out.txt", "w", srdout)`

当AVL树旋转时，会输出相应的旋转类型。

自测

测试说明

由于本程序未加入`system("pause")`，建议在CMD/Terminal中测试。

本程序采用文件输入输出。

另提供样例测试数据输入 `Homework5_In.txt` ，和期望输出 `Homework5_Out.txt` 。

操作说明

> 0. Quit 退出
> 1. Insert 插入节点
> 2. Delete 删除节点
> 3. Get Rank 查找在树中的次序
> 4. Get Sorted Sequence 输出排好序的序列

数据说明

Homework5_In.txt

```
1       输入1
1
1       输入2
2
1       输入3
3
1       输入4
4
1       输入5
5
1       输入6
6
1       输入7
7
1       输入8
8
1       输入9
9
4       输出排好序的序列
2       删除1
1
2       删除3
3
```

```
3       查找6在树中的次序
6
3       查找8在树中的次序
8
4       输出排好序的序列
0       结束
```

Homework5_Out.txt

```
************************
 0. Quit
 1. Insert
 2. Delete
 3. Get Rank
 4. Get Sorted Sequence
************************
RR              输入3后RR旋转（可通过下面的不使用文件输入输出查看旋转是在什么时候进行的）
RR              输入5后RR旋转
RR              输入6后RR旋转
RR              输入7后RR旋转
RR              输入9后RR旋转
1 2 3 4 5 6 7 8 9   输出排好序的序列
RR              删除1和3后RR旋转
4               6在树中第4小
6               8在树中第6小
2 4 5 6 7 8 9       输出排好序的序列
```

不使用文件输入输出

```
************************
 0. Quit
 1. Insert
 2. Delete
 3. Get Rank
 4. Get Sorted Sequence
************************
1
1
1
2
1
3
RR
1
4
1
5
RR
1
6
RR
1
7
RR
1
8
```

```
1
9
RR
4
1 2 3 4 5 6 7 8 9
2
1
2
3
RR
3
6
4
3
8
6
4
2 4 5 6 7 8 9
0
```