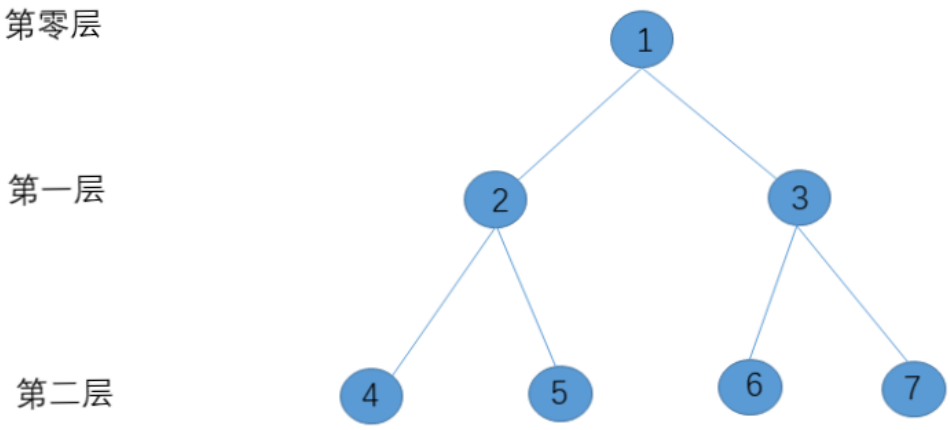
题目描述

一棵树由若干个树节点组成，其中每个树节点都存储了一些数据（`int`，`double` 或者 `Complex` 类型，其中`Complex` 类型需要自己实现）。完美二叉树是指每个节点有两个子节点，并且在树的第i层共有2^i个节点的树。下图是一棵层高为3的完美二叉树。



树的遍历顺序分为先序，中序，后序三种。

1. 先序遍历是指当前节点的数据会在其两个子节点之前被访问。例如， 上图中树的先序遍历顺序为1->2->4->5->3->6->7。
2. 中序遍历是指当前节点的数据会在左子节点之后访问，而在右子节点之前访问。 例如，上图中树的中序遍历顺序为 4->2->5->1->6->3->7。
3. 后序遍历是指当前节点的数据会在其两个子节点之后被访问。例如，上图中树的后序遍历顺序为4->5->2->6->7->3->1。

实现类模板 `Tree` 使之完成二叉树的以下功能:

```
#include <functional>
using namespace std;
template <class T>
class Tree{
    public:
        Tree(int K, const T& value);
        void preorder_show();
        void postorder_show();
        void midorder_show();
        void for_each(std::function<T(T t)> f);
        T accumulate(T init_value);
        T accumulate(std::function<T(T t,T val)> f,const T& init_value);
        int count(T init_value);

};
```

接口说明如下:

1. `Tree(int K, const T &value)`,树的初始化。构造一棵层高为K的完美二叉树，其所有节点的数据都为T。其中完美二叉树是指树的第i层有`2^i`个节点的二叉树。注意，我们把树的根节点视作第0层。
2. `void preorder_show()`，先序打印出所有树节点的数据。数据之间以一个空格分隔。
3. `void midorder_show()`，中序打印出所有树节点的数据。数据之间以一个空格分隔。
4. `void postorder_show()`，后序打印出所有树节点的数据。数据之间以一个空格分隔。
5. `void for_each(std::function<T(T a)> f)`将树上每个节点的数据由`t`改变为`f(t)`，其中`f`为形参中指定的参数类型为T并且返回值类型也为T的函数。
6. `T& accumulate(T init_value)`与`T& accumulate(std::function<T(T t,T init_value)> f)`,计算树上所有节点的数据和一个初始值`init_value`的"和"，并将其返回。默认的和就是"+"。函数`f`用于指定"和"的计算方式，其第一个参数为当前位置的数据`t`，第二个参数为上一次调用该函数的返回值`val`（初始为`init_value`），返回值为本次调用之后的结果`f(t,val)`。注意，节点被访问之后其数据会由`t`被改变为`f(t,val)`。
7. `int count(T init_value)`，返回树的节点中包含与`init_value`值相同的数据的节点个数。

注意，`for_each`以及`accumulate`函数对树的遍历方式为先序遍历。

另外，你还需要定义一个复数类`Complex`,使其能够完成示例调用中的相关功能:

```
class Complex{
    public:
    Complex();
    Complex(double real, double image);
};
```

其中:

1. `Complex()`为默认构造函数，构造一个值为`0`的复数对象;
2. `Complex(double real, double image`，构造一个实部为`real`，虚部为`image`的复数对象

注意`Complex`类的打印方式如下:

1. 当`a,b`皆为`0`时，打印`0`
2. 当`a = 0`且`b`不为`0`时，则打印`bi`
3. 若`b<0`且`a`不为`0`，则打印`a - |b|i`
4. 若`b=0`，且`a`不为0，则打印`a`
5. 若`b>0`，且`a`不为0,则打印`a+bi`

示例调用

```
#include "Tree.h"
#include "Complex.h"
int main()
{
    Complex c(2,-1);
    cout << c << endl;
    //构造一棵层高为2的完美二叉树，共有三个节点，根节点与其左右两个子节点的值皆为 2-1i
    Tree<Complex> tree1(1,c);
    tree1.preorder_show();

    //对树上节点求和，初始val为 1
```

```
    // 首先访问第零层的根节点，计算 1 + (2-1i) ,其结果为 3-1i, 将根节点的数据与val都更新为
3-1i
    // 再访问第一层最左边节点，计算 (3-1i) + (2-1i),其结果为 5-2i,将左子节点的数据与 val
都更新为 5-2i
    // 再访问第一层第二个节点，也即根节点的右子节点，计算 (5-2i) + (2-1i), 其结果为 7-3i。
    // 经过该函数运算后，先序访问时树上的节点中的数据依次为 3-1i, 5-2i, 7-3i,
    // 函数返回运算结束后val的值 7-3i
    tree1.accumulate(Complex(1,0));
    tree1.preorder_show();
    cout << tree1.count(Complex(3,-1)) << endl;
    tree1.accumulate([](Complex x,Complex y){return x * y - x - y;},Complex());

    Tree<int> tree2(2,1);
    tree2.preorder_show();
    cout << tree2.accumulate([](int x,int y){return x+y;},0) << endl;
    tree2.preorder_show();
}
```

其对应的输出应为:

```
2-1i
```

```
2-1i 2-1i 2-1i
```

```
3-1i 5-2i 7-3i
```

```
1
```

```
1 1 1 1 1 1 1
```

```
7
```

```
1 2 3 4 5 6 7
```

提交要求

1. 本次测试需要提四个文件: `Tree.h`,`Tree.cpp, Complex.h, Complex.cpp`, 请将这四个文件压缩为 `Tree.zip` 提交
2. 浮点数输出时，应保留小数点后两位数字。
3. 我们规定，如果两个浮点数之间的差小于`1e-8`，则认为这两个浮点数相等。