# Binary Tree

1. 如果t为NULL，则t->leftchild会出错
2. 顺序遍历用非递归，前、中、后序遍历用递归
3. 链式存储转顺序存储要用数组+递归

# Huffman Tree

1.typedef重命名

例：

typedef struct LNode //字符链表结点

{

char ch;

unsigned int weight;

char \*code;

struct LNode \*next;

}LNode, \*huffmancharslist;

后面可以 huffmancharslist a=new LNode[10]

也可以p=(huffmancharslist)malloc(sizeof(LNode));

2.char \*a=new char[100]

分配一个sizeof（char）\*100的空间，首地址赋给a；

接下来用strcpy（a，“hello”）给这个堆区

3. map

【头文件】

#include<map>

using namespace std;

1、map的定义

map<typename1,typename2>mp;

与其他STL容器在定义上不一样，因为map需要确定映射前类型（键key）和映射后类型（值value）

所以需要在<>内填写两个类型

其中一个是键的类型

第二个是值的类型

如果是int型映射到int型，就相当于是普通的int型数组

***但是如果是字符串到整型的映射，必须是string而不是char数组***

map<string,int>mp

这时因为char数组作为数组是不能被作为键值的。所以字符串作映射，只能用string

而map的键值也可以是STL容器

map<set<int>,string>mp

2、map容器内元素的访问

①通过下标访问

【注意】map的键是唯一的

mp['c']=20;

②通过迭代器访问

定义方式与其他STL容器迭代器相同

map迭代器的使用方式和其他STL容器的迭代器不同：

因为map的每一对映射都有两个typename

这决定了必须通过一个it来同时访问键和值

事实上，map可以使用it->first来访问键

使用it->second来访问值

1. #include<iostream>
2. #include<map>
3. using namespace std;
4. int main()
5. {
6. map<char,int>mp;
7. mp['m']=20;
8. mp['r']=30;
9. mp['a']=40;
10. for(map<char,int>::iterator it=mp.begin();it!=mp.end();it++)
11. {
12. printf("%c %d\n",it->first,it->second);
13. }
14. return 0;
15. }

map会以键的大小从小到大的顺序自动排序

即按照a<m<r的顺序排列这三对映射

这是因为map内部是使用红黑树（set也是）

在建立映射的过程中会自动实现从小到大的排序功能

3、map常用函数

①find（）

find(key)返回键是key的映射的迭代器

②erase（）

(1)删除单个元素

a.mp.erase(it),it为需要删除的元素的迭代器

b.mp.erase(key),key为欲删除的映射的键

e.g.mp.erase('c');

(2)删除一个区间内的所有元素

还是左闭右开

(3)size()

用来获得map中映射的次数

(4)clear()

清空

4. 常引用就是在引用之前加上const对引用进行限制，使得该引用在函数中不能进行改变（但可以改变原变量的值），例如

int a = 6; *// 定义整型变量a，初值为6*

const int &b = a; *// 声明常引用，不允许改变b的值*

b = 8; *// 改变常引用b的值，错误*

a = 8; *// 改变a的值，正确*