关于STL容器的简单总结

1、结构体中重载运算符的示例

```
//结构体小于符号的重载
struct buf {
   int a,b;
   bool operator < (const buf& c1) const {  //注意：第二个const一定不能少
   return a<c1.a;
   }
};
//或
struct foo {
    int a,b;
};
bool operator < (const foo &x, const foo &y)
{return x.a < y.a;}
```

2、队列（queue）

```
#include <queue>
queue<int>a;    //定义
a.push(x);      //压入
a.pop();      //弹出
a.size();   //取大小
a.front();    //访问队首元素
a.back();     //访问队尾元素
a.empty();    //判断队列是否为空
```

3、优先队列（priority_queue）

```
#include <queue>
priority_queue<int,vector<int>,greater<int> > c;      //定义从小到大的int类型的优先队列
priority_queue<int> c;                                //定义从大到小的int类型的优先队列
c.push();         //压入
c.pop();          //弹出队首元素
c.top();          //访问队首元素
c.empty();        //判断队列是否为空
//如是结构体必须重载'<'
```

4、双端队列（deque）

```
#include <deque>
deque <int> b;    //定义双端队列
b.push_front(x); //在队首压入元素
b.push_back(x);  //在队尾压入元素
b.pop_front();   //弹出队首元素
b.pop_back();    //弹出队尾元素
b.size();        //取大小
b.back();        //访问队尾元素
b.front();       //访问队首元素
b.empty();       //判断队列是否为空
//可用[]访问
```

5、栈（stack)

```
#include <stack>
stack<int> d;    //定义
d.push(x);       //压入元素
d.pop();         //弹出栈顶元素
d.top();         //访问栈顶元素
d.size();        //取大小
d.empty();       //判断栈是否为空
```

6、 集合（set)

```
#include <set>
set<int> e;
e.insert(i);     //插入元素
e.erase(i);      //删除值为i的元素
e.count(i);      //查看值为i的元素是否存在
e.empty();       //判断set是否为空
set<int>:: iterator rit;         //定义迭代器
rit = (e.insert(j)).first    //返回插入后元素对应的迭代器
rit=e.find(i);       //返回值为i的元素的迭代器，如果没找到返回的是e.end()
rit=e.lower_bound(i)  //返回值大于等于i的第一个元素的迭代器， 如果没有大于等于i的元素返回e.end()
rit=e.upper_bound(i)  //返回值大于i的第一个元素的迭代器， 如果没有大于i的元素返回e.end()
for(rit=e.begin();rit!=e.end();rit++)   //正序遍历，值为*rit
set<int>::reverse_iterator rit;          //反向遍历的迭代器
for(rit=e.rbegin();rit!=e.rend();rit++) //反向遍历必须这么写
注： 不能直接写e.erase(e.rbegin());
//如使用结构体，必须重载< 或写仿函数
```

7、可重集（multiset)

```
#include <set>
multiset<int> e;
e.insert(i);     //插入元素
e.erase(i);      //删除所有值为i的元素
e.erase(e.find(i)) //删除一个值为i的元素
e.count(i);      //统计值为i的元素的数量
e.empty();       //判断multiset是否为空
multiset<int>:: iterator rit;         //定义迭代器
```

```
rit = (e.insert(j)).first     //返回插入后元素对应的迭代器
rit=e.find(i);        //返回第一个值为i的元素的迭代器，如果没找到返回的是e.end()
rit=e.lower_bound(i)  //返回值大于等于i的第一个元素的迭代器， 如果没有大于等于i的元素返回e.end()
rit=e.upper_bound(i)  //返回值大于i的第一个元素的迭代器， 如果没有大于i的元素返回e.end()
for(rit=e.begin();rit!=e.end();rit++)   //正序遍历，值为*rit
set<int>::reverse_iterator rit;          //反向遍历的迭代器
for(rit=e.rbegin();rit!=e.rend();rit++) //反向遍历必须这么写
//如使用结构体，必须重载< 或写仿函数
```

注：如希望用多种不同排序方式对`set/multiset`内元素进行排序，则应该重载运算符，写成仿函数形式：

```
struct t {
    int a, b, c;
};
struct cmp1
{
    bool operator () (const t &x, const t &y)
    {return x.a < y.a;}
};
struct cmp2
{
    bool operator () (const t &x, const t &y)
    {return x.b < y.b;}
};
struct cmp3
{
    bool operator () (const t &x, const t &y)
    {return x.c < y.c;}
};
set <t, cmp1> st;
set <t, cmp2> st2;
set <t, cmp3> st3;
```

8、map

```
#include <map>
map<string,int> f; //定义一个map，其中string是键值（就像一个人的名字一样）的类型，int是值的类型，可以随便换。 键值需要重载 <。
f[s]=d;  //把一个键值为s，值为d的元素 插入到此map中或覆盖原有映射（因为map重载了[]所以可以直接这样写）
f.count(s); //统计键值为s的元素的个数，因为在map中键值是排好序的集合，所以count()的返回值不是1就是0
f.erase(s);             //删掉键值是s的元素
f.size();               //取大小
f.empty();              //判断map是否为空
map<string,int>:: iterator rit;    //定义map的迭代器 ，遍历的时候可能会用到
rit=f.find(s);                     //返回键值为s的元素的迭代器
rit->second;             //迭代器为s映射的值，如把second改成first则是s
//查询可以直接用[]
//map就像一个完美的哈希表，但内部由红黑树实现， 因此操作复杂度均为O(log(n))，有了map妈妈再也不用担心查找数据了！
```

9、vector

```
#include <vector>
vector <int> vec;              //定义一个vector， 内部元素类型为int。
vector <int>::iterator it;     //定义vector<int> 的迭代器
vec.push_back(i)               //向vec后面加入元素i
vec.push_front(i)              //向vec前面加入元素i
vec.begin()                    //返回vec的第一个元素对应的迭代器， 如果为空返回vec.end()
vec.size()                     //返回vector内的元素个数
vec.erase(it)                  //删除it对应元素， 同时后面的元素整体前移一位。 注：复杂度为O(N)
vec.clear()                    //清空vec， 但不释放内存
vector <int>().swap(vec)       //清空vec并释放内存（若卡内存，多组数据的题推荐这样清零）
```

10、sort()

```
#include <algorithm>
vector <int> vec;
sort(vec.begin(), vec.end());//vector的sort方式
int a[105];
sort(a, a + 105);//将整个a数组从小到大排序
double b[1005];
bool cmp (double c, double d)//自定义排序方法
{return c > d;}
sort(b + 1, b + 1 + 1000, cmp)//将b[1] ~ b[1000]的元素从大到小排序
```

11、二分查找

注：`lower_bound``upper_bound`

```
#include <algorithm>
int a[1005], pos;
int *b;
b = lower_bound(a + 1, a + 1001, i)   //返回a[1] ~ a[1000]第一个大于等于i的元素的指针， 若没有则返回a[1001]的指针
b = upper_bound(a + 1, a + 1001, i)   //返回a[1] ~ a[1000]第一个大于i的元素的指针， 若没有则返回a[1001]的指针
pos = b - a;                          //得到其下标
vector <int> vec;
vector <int>::iterator it;
it = lower_bound(vec.begin(), vec.end(), i)  //返回vec中第一个大于等于i的元素的迭代器， 若没有则返回vec.end()
it = upper_bound(vec.begin(), vec.end(), i)  //返回vec中第一个大于i的元素的迭代器， 若没有则返回vec.end()
set <int> st;
set <int>::iterator it;
it = st.lower_bound(st.begin(), st.end(), i)  //返回st中第一个大于等于i的元素的迭代器， 若没有则返回st.end()
it = st.lower_bound(st.begin(), st.end(), i)  //返回st中第一个大于i的元素的迭代器， 若没有则返回st.end()
```

12、reverse()

```
#include <algorithm>
int a[105];
reverse(a + 1, a + 1 + 100);           //将a[1] ~ a[100]及其之间的元素前后翻转
vector <int> vec;
reverse(vec.begin(), vec.end())        //前后翻转整个vec里面的元素
```

13、bitset

```
//bitset可以当bool数组用， 但其内部为unsigned int压位而成， 支持左右移， 赋值， 清零等操作。 01背包用bitset优化可以做到O(N^2/32)
#include <bitset>
bitset <1005> bt;                          //声明一个大小为1005的bitset
bt[0] = 1;                                 //将bt[0]设为1
int a;
a = bt.count();                            //返回bt中1的个数
bt.reset();                                //将bt所有位清零
bt.set();                                  //将bt所有位设为1
bt.flip();                                 //将bt所有位异或1
bt.flip(i);                                //将bt第i位翻转
bt = bt | (bt << 1)                        //将bt的第i位与第i + 1位取或， 复杂度为O(N/32)
bt = bt ^ (bt >> 2)                        //将bt的第i位和第i - 2位异或， 复杂度为O(N/32)
```

14、*_builtin系列*

```
//以下函数复杂度均为O(1)
int a = __builtin_popcount(233)                //返回233二进制位上1的个数
int b = __builtin_ffs(666)                     //返回666二进制位从低向高第一个1的位置
int c = __builtin_ctz(19260817)                //返回19260817二进制位后缀0的个数
//注： 以上函数所传变量默认均为unsigned int， 若要传long long 请在后面加上"ll"。
例如： int d = __builtin_popcountll(100000000000011);
```