C++ STL

作者：Zhuoran Wang Email: wangzhuoran200105@126.com

- vector
- string
- queue
- stack
- deque
- set, map, multiset, multimap
- unordered_set, unordered_map, unordered_multiset, unordered_multimap
- bitset

1. vector

变长数组， 倍增的思想

系统为某一个程序分配空间时，有一个很大的特点，其所需时间基本上与所申请的空间无关，而与申请次数有关。举例来说，申请一个长度为1000的数组和申请1000个长度为1的数组，后者要远远慢于前者，正因为有这个特点，vector采用了倍增的思想。

当程序运行语句

```
vector<int> v;
```

之后，系统为我们首先分配了一个不太大的空间，当有这些空间用完之后，vector会被倍增，之前的数原封不动的copy进新vector里。

这样，当我们要插入n个数在一个vector里的时候，空间申请的次数大概是logn的，时间复杂度均摊下来是O(1)的

```
#include <iostream>
#include <cstdio>
```

```
#include <cstring>
#include <algorithm>
#include <vector>

using namespace std;

int main()
{
    vector<int> v; //声明一个vector，不宣告长度
    vector<int> a(10); //声明一个长度为10 的vector
    vector<int> b(10, 3); // 声明一个长度为10的vector，并且把每一个数都设置为3

    cout<<b.size()<<endl; //返回元素个数，这里为10
    cout<<b.empty()<<endl; //返回容器是否为空
    //a.size()
    //a.empty()
    //这两个函数是所有的容器都支持的操作，时间复杂度为O(1)

    b.clear(); //清空vector，注意并不是所有容器都支持清空操作

    cout<<b.size()<<endl; //这里为0

    //v.front()/v.back() 返回第一个数/返回最后一个数
    //v.push_back(num) 在最后加入num
    //v.pop_back() 把最后一个数删掉
}
```

vector的长度

```
v.size()
```

vector是否为空

```
v.empty()
```

清空vector

```
v.clear();
```

vector的迭代器

```
v.begin() //第一个数的地址
v.end() //最后一个数的后面一个数
//可以看成指针
```

插入删除

```
v.push_back(num) //在最后加入num
v.pop_back(); //把最后一个数删掉
```

遍历方式

```
for(int i = 0; i<v.size(); i++) cout<<v[i]<<endl;

for(vector<int>::iterator i = v.begin(); i != v.end(); i++) cout<< *i <<endl;
//for(auto i = v.begin(); i != v.end(); i++) cout<< *i <<endl;

for(auto x : v) cout<< x <<endl;
//auto 是C++11的新特性
```

pair<type，type>

C++内置二元组

```
pair<int, string> pii;
//pii.first
//pii.second

pii = make_pair(1, "yxc");
pii = {10, "wzr"};//C++11
```

支持比较运算，按字典序来比

如果有三种属性

```
pair<int, pair<int, int>> piii;
```

2. string

字符串

substr(), c_str()

```
string str;

str.size()
str.empty()
str.clear()
```

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>

using namespace std;

int main()
{

    string str = "yxc";

    str += " def";
```

```
    str += 'c';

    cout<<str<<endl;

    cout<<str.substr(1, 2)<<endl;

    cout<<str.substr(1)<<endl; //从第1个字符到最后一个
    cout<<str.substr(1, 10)<<endl; //超过了字符串长度，全部输出

    printf("%s\n", str.c_str()); //c_str()返回字符串的起始地址

    return 0;
}
```

3. queue

队列，先入先出的数据结构， push(), front(), pop()

```
    #include <queue>
    size()
    empty()
    push()  //向队尾插入一个元素
    front()  //返回队头元素
    back()  //返回队尾元素
    pop()  //弹出队头元素
```

注意queue是没有clear()这个函数的，如果要清空一个队列

```
queue<int> q;

q = queue<int>();
```

直接新构建一个队列就可以了

4. priority_queue

优先队列，实际的实现是用**堆（heap）**， push(), top(), pop()

```
//priority_queue, 优先队列，默认是大根堆
    #include <queue>
    size()
    empty()
    push()  //插入一个元素
    top()  //返回堆顶元素
    pop()  //弹出堆顶元素
    //定义成小根堆的方式: priority_queue<int, vector<int>, greater<int>> q;
```

注意priority_queue也是没有clear()函数的

> 查阅C++泛型编程

5. stack

栈，先入后出的数据结构，push(), top(), pop()

```
//stack, 栈
    #include <stack>
    size()
    empty()
    push()  //向栈顶插入一个元素
    top()  //返回栈顶元素
    pop()  //弹出栈顶元素
```

注意stack也是没有clear()函数的

6. deque

双端队列，队头队尾都可以插入删除（可以认为是加强版的vector）

```
//deque, 双端队列
    #include <deque>
    size()
    empty()
    clear()
    front()/back()
    push_back()/pop_back()
    push_front()/pop_front()
    begin()/end()
    []  //支持随机寻址
```

（缺点就是比较慢）

7. set, map, multiset, multimap

基于**平衡二叉树**实现的（**红黑树**），本质上是动态的维护一个有序的序列

这四种容器都支持的操作

```
    size()
    empty()
    clear()
    begin()/end()
    ++, -- 返回前驱和后继，时间复杂度 O(logn)
```

set和数学里的集合很像，是不允许有重复元素的，如果加入了一个重复元素，会被忽略掉。multiset是可以有重复元素的

```
//set/multiset
                #include <set>
        insert()  //插入一个数
        find()  //查找一个数，如果不存在的话返回的是end()迭代器
        count()  //返回某一个数的个数
        erase()
            //(1) 输入是一个数x，删除所有x   O(k + logn)
            //(2) 输入一个迭代器，删除这个迭代器
        lower_bound()/upper_bound()
            lower_bound(x)  //返回大于等于x的最小的数的迭代器，最小上界
            upper_bound(x)  //返回大于x的最小的数的迭代器，最大下界
```

map/multimap

```
        insert()  //插入的数是一个pair
        erase()  //输入的参数是pair或者迭代器
        find()
        []  //随机寻址，注意multimap不支持此操作。时间复杂度是 O(logn)
        lower_bound()/upper_bound()
```

map随机寻址Demo

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <map>
using namespace std;

int main()
{

    map<string, int> m;

    m["wzr"] = 1;

    cout<< m["wzr"] <<endl;


    return 0;
}
```

8. unordered_set, unordered_map, unordered_multiset, unordered_multimap

基于**哈希表**实现

和上面的set, map, multiset, multimap，不过这里的几个增删改查的时间复杂度是 O(1)。缺点是不支持 lower_bound()/upper_bound()， 迭代器的++，--

头文件：

```
#include <unordered_map>

#include <unordered_set>
```

9. bitset

压位

比如一个整数，int，32个bits，很多题目需要压位来算，就是说，我们只需要知道每一位上的数是0是1.

举例来说，我们想开一个长度为1024的bool数组，（C++里每一个bool变量的长度是一个byte）也就是1024bytes = 1 kb，但是如果我们把这个数组压入每一位里面去的话，每个字节存8位，这样就只需要128bytes的空间

```
    bitset<10000> s; //<>里是表示大小，与前面的容器都有所区别
    //支持几乎所有的位运算，bitwise oprerator
    ~, //取反
    &, //AND
    |, //OR
    ^, //XOR
    >>, << //位移
    ==, != //比较
    [] //随机寻址
```

```
    count()  //返回有多少个1

    any()  //判断是否至少有一个1
    none()  //判断是否全为0

    set()  //把所有位置成1
    set(k, v)  //将第k位变成v
    reset()  //把所有位变成0
    flip()  //等价于~
    flip(k) //把第k位取反
```