本文档借鉴了知乎stl速成文档的同时大量翻阅了官方文档en.cppreference.com,也就是相当于2.5手笔记

C++ STL

容器,迭代器,算法,仿函数,适配器,分配器

1 Definitions

1.1 容器 Containers

序列式容器

- **Vector** : 向量
- **deque** : 双端队列
- **list** : 双向链表
- **forward_list** : 单向链表
- **array** : C++11 新增的容器,比 vector 效率高,其大小固定,无法动态扩展或收缩,只允许访问和替换存储的元素,且使用要在 std 命名空间内。

关联式容器

- **set\multiset** : 内部的元素依据其值自动排序
- **map\multimap** : 键值/实值
- multi意味着可以存在相同元素
- 容器类自动申请和释放内存,无需new和delete操作

1.2 迭代器

- 重载了*, + + , = = , ! = , = 运算符
- 容器提供迭代器,算法使用迭代器.
- 常见迭代器类型: iterator,const_iterator,reverse_iterator,const_reverse_iterator

1.3 容器适配器

种类	默认顺序容器	可用顺序容器	说明
stack	deque	vector,list,deque	

种类	默认顺序容器	可用顺序容器	说明
queue	deque	list,deque	基础容器必须提供push_front()运算
priority_queue	vector	vector,deque	基础容器必须提供随机访问功能

2 容器详解

2.1 vector

头文件

```
#include <vector>
```

声明

```
vector<T> a; //这里定义了一个一维可变长度数组.T代表int等基本数据类型,结构体,类,stl容器,原生或智能指针
vector<int> v(n); //这里定义了一个长度为n的一维数组,缺省值为0
vector<int> v(n,1); //初始值为1
vector<int> v{1,2,3}; //3个元素
vector<int> v = a; //拷贝
```

方法

```
v.push_back(e) //尾部增加e
v.pop_back() //尾部删除
v.size()
v.insert(it,e) //在迭代器it处放入e,见下文详解
v.erase(first,last) //删除[迭代器first,迭代器last)
v.empty()
v.begin(),v.end() //返回迭代器
v.front(),v.back() //返回数值
v.clear() //清空
```

> **insert()方法详解**

- 声明

```
iterator insert( const_iterator pos, const T& value );                    (1)  (constexpr since C++20)
iterator insert( const_iterator pos, T&& value );                         (2)  (since C++11)
                                                                               (constexpr since C++20)
iterator insert( const_iterator pos,
                 size_type count, const T& value );                       (3)  (constexpr since C++20)
template< class InputIt >
iterator insert( const_iterator pos, InputIt first, InputIt last );       (4)  (constexpr since C++20)
iterator insert( const_iterator pos, std::initializer_list<T> ilist );    (5)  (since C++11)
                                                                               (constexpr since C++20)
```

- 举例

```
vector<int> v = {1,2,3}
auto it = v.begin()+1; //比较好写的迭代器声明方式,指向第二个位置(下标为1,值为2),auto的作用是自动推导
v.insert(it,1.5); // 声明(1)(2),v={1,1.5,2,3}
it++;
v.insert(it,2,2.5); // 声明(3),v={1,1.5,2,2.5,2.5,3}


std::vector<int> v1 = {1, 2, 3};
std::vector<int> v2 = {10, 20, 30};
auto it = v1.begin() + 1;
v.insert(it, v2.begin(), v2.end());  //声明(4),v={1,10,20,30,2,3}
// v.insert(it, {10, 20, 30});  //声明(5)
```

访问

机考请直接使用数组下标访问,这里展示一下其他访问方法供复习c++

```
vector <int> V{1,2,3,4,5};

//下标访问, 和普通数组方法相同
for (int i = 0; i < 5; i++)
    cout << V[i] << " ";
    cout << *(V.begin() + i) << " "; //这个是一样的

//迭代器访问, 类似指针操作
vector<int>::iterator it; //先声明迭代器变量 it
for (it = V.begin(); it != V.end(); it++)
    cout << *it << " ";

//智能指针auto访问, 这个真简洁好吧
for (auto val : V) //这里auto的作用是自动推导类型,所以用int也行
    cout << val << " ";
```

二维初始化

```
vector<int> v[5]; //第一维固定长度为5,第二维可变
vector<vector<int>> V; //储存多个vector<int> 变量
//这两种方式中,V[0]都是一个vector<int>,可以push_back可以pop_back int变量,第二种V可以调用push_back泵
```

2.2 deque

方法

比vector多了push_front()和pop_front()

> erase可以擦除值和区间,但参数都是迭代器,注意deque的插入和删除操作可能会使迭代器失效

2.3 list

代码上list更是和deque差别不大

2.4 map

初始化

```
map<T1,T2> mp; 键类型是T1,值类型是T2
```

map会按照键的顺序从小到大自动排序, 因此键的类型必须可以比较大小, 支持比较操作符

方法

- O(log N)

```
mp.find(key) //返回键位key的迭代器
mp.erase(it),mp.erase(first,last),mp(key) //多了一个根据键值删除
mp.insert() //更推荐直接用operator[]插入
mp.lower_bound(key) //返回首个键不小于key的迭代器
mp.upper_bound(key) //返回首个键大于key的迭代器
mp.count(key) //键为key的元素个数(注意multimap)
```

- O(1)

```
mp.size()
mp.empty()
mp.begin() //第一个元素的迭代器地址
mp.end() //最后一个元素的下一个地址
mp.rbegin() //最后一个元素的迭代器地址
mp.rend() //第一个元素的上一个地址
//前两个是正向遍历用,后两个是逆向遍历用
```

- O(N)

```
mp.clear()
```

遍历

- 对于 `map` 元素和其迭代器 `it`，`map` 的键和值可以分别通过 `it->first` 和 `it->second` 来访问

```
map<int,string> mp = {{0,"zero"}};
mp.insert({
    {1, "one"},
    {2, "two"}
});
mp[3] = "three";

for (auto i : mp) //智能指针,正向遍历
cout << i.first << " " << i.second << endl;

map<int, int>::iterator it = mp.begin();//正向遍历
while (it != mp.end()) {
    cout << it->first << " " << it->second << endl;
    it ++;
}

auto it = mp.rbegin(); //逆向遍历
while (it != mp.rend()) {
    cout << it->first << " " << it->second << endl;
    it ++;
}
```

与无序映射unordered_map的比较

- **map**：内部用**红黑树**实现具有**自动排序**(按键从小到大)功能,查找增删等操作的平均时间复杂度为 $O(logN)$, 但**空间占用较大**.
- **unordered_map**：内部用**哈希表**实现, 查找增删等操作的平均时间复杂度是 $O(1)$, 内部元素无序杂乱, 元素的插入顺序和哈希值有关, 查找速度非常快, 内存占用相对较低, 但**建立哈希表耗时较大**.

2.5 set

方法

和 map 一样

遍历

```
for (auto i : s){
    cout << i << endl;
}
```

> 访问最后一个元素 `cont<<*s.rbegin()<<endl`
> 添加元素直接 `insert(ele)` 即可

排序方式

```
set<int> s1; //默认从小到大的排序
set<int,greater<int>> s2; //更改为从大到小排序
set <int, function <bool (int, int)>> s([&](int i, int j){
    return i > j; //初始化时使用匿名函数定义比较规则
});
```

与unordered_set比较

类似map

2.6 string

初始化

```
string str1;                 //生成空字符串
string str2("123456");       //生成"123456"的复制串
string str3("123456", 0, 3); //结果为"123"，从 0 位置开始，长度为3
string str4("123456", 5);    //结果为"12345"，默认从 0 位置开始，长度为5
string str5(4, '3');         //结果为"3333"，构造 4 个字符'3'连接而成的字符串
string str6(str2, 2);        //结果为"3456"，截取从第三个元素（2对应第三位置）到最后的字符串
```

基本操作

- 支持各种比较操作符和=
- 读入数据
  - 读入一行字符串, 遇到空格,回车即结束

    ```
    string s;cin>>s;
    ```

  - 读入一行字符串(包括空格), 遇到回车即结束

    ```
    getline (cin, s);
    ```

方法

```
s.size() / s.length()                    //返回string对象的字符个数，两者执行效果相同
s.max_size()                             //返回string对象中最多包含的字符数，超出会抛出length_error
s.capacity()                             //重新分配内存之前，string对象能包含的最大字符数

s.push_back(ele)                         //在末尾插入数据
s.pop_back()                             //在末尾删除数据
s.insert(pos, ele)                       //在pos位置前插入ele
//这里的pos无需是迭代器,可以直接是整数索引,而vector不能这样
s.append(str)                            //在s字符串结尾添加str字符串

s.erase(iterator pos)                    //删除字符串中位置pos的迭代器所指向的字符
s.erase(iterator first, iterator last)   //删除字符串中迭代器区间[first, last)上的所有字符
s.erase (pos, len)                       //删除字符串中从索引位置pos开始的len个字符
s.clear()                                //删除字符串中所有字符

s.replace(pos, n, str)                   //把当前字符串从索引pos开始的n个字符替换为str
s.replace(pos, n, n1, c)                 //把当前字符串从索引pos开始的n个字符替换为n1个字符c
s.replace(it1, it2, str)                 //把当前字符串[it1, it2)区间的字符替换为str

s.tolower(s[i])                          //把s[i]字符转换为小写
s.toupper(s[i])                          //把s[i]字符转换为大写
// 这两个转换字符大小写的功能需要通过<cctype>库实现

s.find(str, pos)                         //在当前字符串的pos索引位置查找子串str
s.find(c, pos)                           //在当前字符串的pos索引位置查找字符c
```

- find 与 rfind

```
string s("dog bird chicken bird cat");
cout << s.find("chicken") << endl;       //结果为9，返回的是首字母在字符串中的下标
cout << s.find('i', 6) << endl;          //结果为11，返回的是在下标6开始的找到的第一个i的下标
cout << s.rfind("chicken") << endl;      //结果为9，返回的是从末尾开始查找反向找到的该单词第一个字母
cout << s.rfind('i') << endl;            //结果为18，返回的是从末尾开始查找反向找到的第一个i的下标
```

3 容器适配器

操作	stack	queue	priority_queue
规则	后进先出（LIFO）	先进先出（FIFO）	按优先级排序（默认最大堆，优先级高的元素先出）
头文件	`#include <stack>`	`#include <queue>`	`#include <queue>`
声明	`stack<T> s;`	`queue<T> q;`	`priority_queue<T> pq;`
push	`s.push(x);`	`q.push(x);`	`pq.push(x);`
pop	`s.pop();`	`q.pop();`	`pq.pop();`
top/front	`s.top();`	`q.front();`	`pq.top();`
back	N/A	`q.back();`	N/A
empty	`s.empty();`	`q.empty();`	`pq.empty();`
size	`s.size();`	`q.size();`	`pq.size();`