

C++11 STL Cheat Sheet

Collected by merelydust for AC^ω(ω)^ω

目录 Table of Contents

C++11 STL Cheat Sheet

目录 Table of Contents

1. 容器 Container

1.0 所有容器都有的成员函数

1.1 顺序容器 Sequence Containers

1.1.0 所有顺序容器都有的成员函数

1.1.1 动态数组 vector

1.1.2 双向链表 list

1.1.3 双向队列 deque

1.1.4 字符串 string

1.2 关联容器 Associative Containers

1.2.0 所有关联容器都有的成员函数

1.2.1 集合 set

1.2.2 字典 map

1.3 容器适配器 Container adaptors

1.3.0 所有容器适配器都有的成员函数

1.3.1 stack

1.3.2 queue

1.3.3 priority_queue

2. 算法 Algorithm

2.0 头文件

2.1 不改变原序列的算法

2.2 改变元素值的算法

2.3 删除元素的算法

2.4 改变顺序的算法

2.4.1 变序算法

2.4.2 排序算法

2.5 操作有序区间的算法

3. STL实现树和图

3.1 树的实现

3.1.1 树的建立

3.1.2 树的遍历

3.1.3 增删查改

3.1.4 应用例题

3.2 图的实现

- 3.2.1 图的建立
 - 3.2.2 图的遍历
 - 3.2.3 增删查改
 - 3.2.4 应用例题
3. 参考资料 References

1. 容器 Container

容器是类模版 充分发挥想象力~

1.0 所有容器都有的成员函数

- .size() return int
- .empty() return bool
- 两对迭代器 容器类名::iterator it & 容器类名::reverse_iterator rit

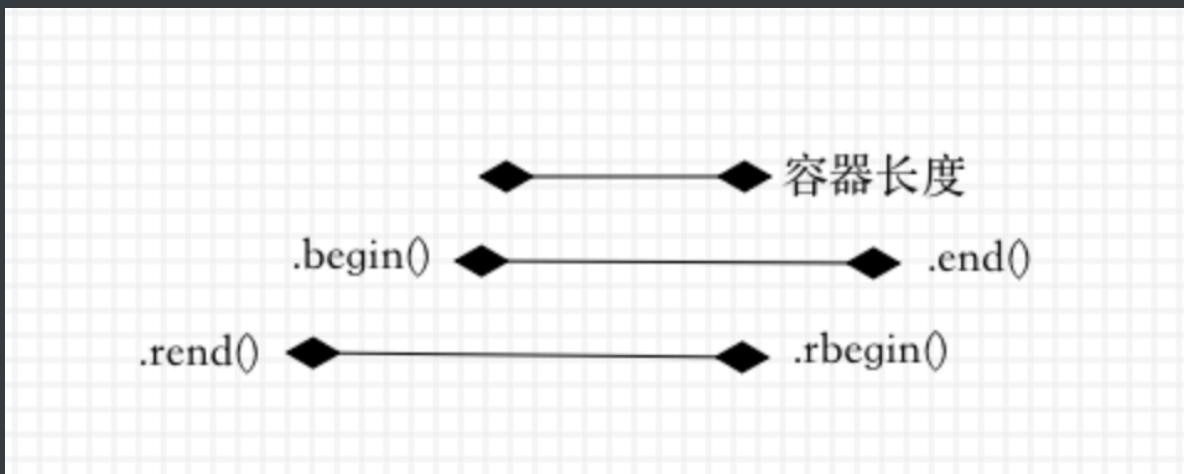


表 10.1 不同容器的迭代器的功能

容 器	迭代器功能
vector	随机访问
deque	随机访问
list	双向
set/multiset	双向
map/multimap	双向
stack	不支持迭代器
queue	不支持迭代器
priority_queue	不支持迭代器

list 容器的迭代器是双向迭代器。假设 v 和 i 的定义如下：

```
list < int > v;
list < int > ::const_iterator i;
```

则以下代码是合法的：

```
for (i = v.begin(); i != v.end(); ++i)
    cout << *i;
```

以下代码则不合法：

```
for (i = v.begin(); i < v.end(); ++i)
    cout << *i;
```

因为双向迭代器不支持用“<”进行比较。以下代码也不合法：

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

因为 list 不支持随机访问迭代器的容器，也不支持用下标随机访问其元素。

STL的sort函数需要随机访问迭代器的支持 所以不适用list 只能使用list的成员函数.sort()

- .erase(pos, howManytoErase)

1.1 顺序容器 Sequence Containers

1.1.0 所有顺序容器都有的成员函数

- .front() 返回第一个元素的引用 .back() 返回最后一个元素的引用
- .push_back(element) 增 .pop_back(element) 删

1.1.1 动态数组 vector

```
#include
```

- vector v;
- v.insert(iterator, val); v.erase(iterator start, iterator end)
remove函数返回值是空, erase返回指向下一个元素的迭代器
//如果只有一个参数 只删除该迭代器指向的参数 区别于string
- v.swap(vector &v2) v内容与另一个同类型的v2互换

1.1.2 双向链表 list

```
#include
```

表 10.3 list 的成员函数

成员函数或成员函数模板	作用
void push_front(const T & val)	将 val 插入链表最前面
void pop_front()	删除链表最前面的元素
void sort()	将链表从小到大排序
void remove(const T & val)	删除和 val 相等的元素
remove_if	删除符合某种条件的元素
void unique()	删除所有和前一个元素相等的元素
void merge(list <T> & x)	将链表 x 合并进来并清空 x。要求链表自身和 x 都是有序的
void splice(iterator i, list <T> & x, iterator first, iterator last)	在位置 i 前面插入链表 x 中的区间 [first, last)，并在链表中删除该区间。链表自身和链表 x 可以是同一个链表，只要 i 不在 [first, last) 中即可

1.1.3 双向队列 deque

```
#include
```

deque也是可变长数组，适用于vector的操作都适用于它。比起vector，它在头尾增删元素的性能更好，有两个vector没有的成员函数：

```
.push_front(val) & .pop_front(val)
```

1.1.4 字符串 string

- 赋值可以接收char数组
- .substr(startPos, len) 如果省略len参数或者len>原字符串长度 则子串取到原串末尾
- s1.swap(s2) 交换两个字符串内容
- s1.find_first_of(s2) 寻找s1中第一次出现s2中任一字符的位置 s1.find_first_not_of(s2) 寻找第一个s2没有的字符
.find_last_of .find_last_not_of 从后往前找
- s1.replace(被替换的子串首, 被替换的子串尾, 去替换的字符串, 去替换的字符串首, 去替换的字符串尾)
- .erase(pos, len) .insert(pos, len)

1.2 关联容器 Associative Containers

不能修改元素值或map的关键字。

1.2.0 所有关联容器都有的成员函数

- .find(val)

- `.lower_bound(val)` // 返回最大位置it [begin, it)中所有元素都比val小
// 找到的是另一个区间的下限 所以[begin, it)这个区间的元素都小于val
- `.upper_bound(val)` // 返回最小位置it [it, end)中所有元素都比val大
- `.count(val)` // how many elements.value == val
- `.insert(val)`

1.2.1 集合 set

```
#include
```

`.insert(val)` 返回bool true插入成功 false val已经在集合中

1.2.2 字典 map

```
#include
```

`it->first` 访问关键字 `it->second` 访问值

1.3 容器适配器 Container adaptors

1.3.0 所有容器适配器都有的成员函数

- `.push(val)`
- `.top()` // 返回stack顶部 或 queue队头元素的引用
- `.pop()`

1.3.1 stack

```
#include
```

1.3.2 queue

访问和删除只能在队头 增加只能在队尾

```
#include
```

1.3.3 priority_queue

```
#include
```

队头元素总是最大的 内部并非完全有序

可以自定义cmp函数 cmp返回false的时候交换位置(wanted condition return false)

2. 算法 Algorithm

2.0 头文件

2.1 不改变原序列的算法

表 10.8 不变序列算法

算法名称	功 能
min	求两个对象中较小的 (可自定义比较器)
max	求两个对象中较大的 (可自定义比较器)
min_element	求区间中的最小值 (可自定义比较器)
max_element	求区间中的最大值 (可自定义比较器)
for_each	对区间中的每个元素都做某种操作
count	计算区间中等于某值的元素个数
count_if	计算区间中符合某种条件的元素个数
find	在区间中查找等于某值的元素
find_if	在区间中查找符合某条件的元素
find_end	在区间中查找另一个区间最后一次出现的位置 (可自定义比较器)
find_first_of	在区间中查找第一个出现在另一个区间中的元素 (可自定义比较器)
adjacent_find	在区间中寻找第一次出现连续两个相等元素的位置 (可自定义比较器)
search	在区间中查找另一个区间第一次出现的位置 (可自定义比较器)
search_n	在区间中查找第一次出现等于某值的连续 n 个元素 (可自定义比较器)
equal	判断两区间是否相等 (可自定义比较器)
mismatch	逐个比较两个区间的元素, 返回第一次发生不相等的两个元素的位置 (可自定义比较器)
lexicographical_compare	按字典序比较两个区间的大小 (可自定义比较器)

2.2 改变元素值的算法

算法名称	功能
for_each	对区间中的每个元素都做某种操作
copy	复制一个区间到别处
copy_backward	复制一个区间到别处，但目标区间是从后往前被修改的
transform	将一个区间的元素变形后复制到另一个区间
swap_ranges	交换两个区间的内容
fill	用某个值填充区间
fill_n	用某个值替换区间中的 n 个元素
generate	用某个操作的结果填充区间
generate_n	用某个操作的结果替换区间中的 n 个元素
replace	将区间中的某个值替换为另一个值
replace_if	将区间中符合某种条件的值替换为另一个值
replace_copy	将一个区间复制到另一个区间，复制时某个值要换成新值
replace_copy_if	将一个区间复制到另一个区间，复制时符合某条件的值要换成新值

2.3 删除元素的算法

算 法	功 能
remove	删除区间中等于某个值的元素
remove_if	删除区间中满足某种条件的元素
remove_copy	复制区间到另一个区间，等于某个值的元素不复制
remove_copy_if	复制区间到另一个区间，符合某种条件的元素不复制
unique	删除区间中连续相等的元素，只留下一个（可自定义比较器）
unique_copy	复制区间到另一个区间。连续相等的元素，只复制第一个到目标区间 (可自定义比较器)

上述算法的原型如下：

remove_if

原型： iterator remove_if(iterator first, iterator last, Pred op);

功能：删除[first, last) 中所有使得 $op(x)$ 为真的 x 。如果有 n 个元素被删除，则返回值就是 $last - n$ 。

remove_copy

原型： iterator remove_copy(iterator first, iterator last, iterator dest, const T & val);

功能：将[first, last) 复制到 dest 开始的地方，等于 val 的元素不复制。程序员要确保从 dest 开始有足够的空间存放复制过来的元素。假设有 n 个元素被复制，返回值就是迭代器 $dest + n$ 。

remove_copy_if

原型： iterator remove_copy(iterator first, iterator last, iterator dest, Pred op);

功能：将[first, last) 复制到 dest 开始的地方，使得 $op(x)$ 为真的元素 x 不复制。程序员要确保从 dest 开始有足够的空间存放复制过来的元素。假设有 n 个元素被复制，返回值就是迭代器 $dest + n$ 。

unique

原型： iterator unique(iterator first, iterator last);

功能：对[first, last) 中连续相等的元素，只留下第一个，删除其他元素。如果删除了 n 个元素，返回值就是迭代器 $last - n$ 。

unique_copy

2.4 改变顺序的算法

2.4.1 变序算法

算 法	功 能
reverse	颠倒区间的前后次序
reverse_copy	把一个区间颠倒后的结果复制到另一个区间，源区间不变
rotate	将区间进行循环左移
rotate_copy	将区间以首尾相接的形式进行旋转后的结果复制到另一个区间，源区间不变
next_permutation	将区间改为下一个排列（可自定义比较器）
prev_permutation	将区间改为上一个排列（可自定义比较器）
random_shuffle	随机打乱区间内元素的顺序
partition	把区间内满足某个条件的元素移到前面，不满足该条件的移到后面
stable_partition	把区间内满足某个条件的元素移到前面，不满足该条件的移到后面。而且对于这两部分元素，分别保持它们原来的先后次序不变

2.4.2 排序算法

算法名称	功能
sort	将区间从小到大排序（可自定义比较器）
stable_sort	将区间从小到大排序，并保持相等元素间的相对次序（可自定义比较器）
partial_sort	对区间部分排序，直到最小的 n 个元素就位（可自定义比较器）
partial_sort_copy	将区间前 n 个元素的排序结果复制到别处，源区间不变（可自定义比较器）
nth_element	对区间部分排序，使得第 n 小的元素（ n 从 0 开始）就位，而且比它小的元素都在它前面，比它大的元素都在它后面（可自定义比较器）
make_heap	使区间成为一个“堆”（可自定义比较器）
push_heap	将元素加入一个“堆”区间（可自定义比较器）
pop_heap	从“堆”区间删除堆顶元素（可自定义比较器）
sort_heap	对一个“堆”区间进行排序，排序结束后，该区间就是普通的有序区间，不再是“堆”了（可自定义比较器）

2.5 操作有序区间的算法

表 10.13 有序区间算法

算法名称	功能
binary_search	判断区间中是否包含某个元素
includes	判断是否一个区间中的每个元素都在另一个区间中
lower_bound	查找最后一个不小于某值的元素的位置
upper_bound	查找第一个大于某值的元素的位置
equal_range	同时获取 lower_bound 和 upper_bound
merge	合并两个有序区间到第三个区间
set_union	将两个有序区间的并复制到第三个区间
set_intersection	将两个有序区间的交复制到第三个区间
set_difference	将两个有序区间的差复制到第三个区间
set_symmetric_difference	将两个有序区间的对称差复制到第三个区间
inplace_merge	将两个连续的有序区间原地合并为一个有序区间

3. STL实现树和图

3.1 树的实现

3.1.1 树的建立

3.1.2 树的遍历

3.1.3 增删查改

3.1.4 应用例题

3.2 图的实现

3.2.1 图的建立

3.2.2 图的遍历

- BFS 广度优先
- DFS 深度优先

3.2.3 增删查改

3.2.4 应用例题

3. 参考资料 References

- 1. 郭炜《新标准C++程序设计》