Common Algorithms in STL of C++（C++ STL中的常见算法函数）

今天刚刚上了课，疯了，居然快睡着了，不过，那么多的算法，还是蛮有用的，下面分享一下常用算法函数的用法和特点。写得不好的，欢迎指出。
首先，今天分享的函数都使用前都要加上下面的语句

```
#include <algorithm>
#include <numeric>
#include <iterator>  //  一般来讲，STL都应该
                     //  要用迭代器的头文件
```

algorithm头文件中

fill系和generate系

```
@Jannie Kung
//  从开始的迭代器到结束的迭代器指向的内容都赋上value
fill(iterator start, iterator finish, T value);
//  从开始的迭代器到start + number个迭代器指向的
//  内容都赋上value
fill(iterator start, int number, T value);
//  generate系和fill系函数的不同在于它的第三个参数
//  可以是函数，更加灵活
generate(iterator start, iterator finish, function);
//  从开始的迭代器到start + number个迭代器指向的内容通过
//  调用函数来实现
generate_n(iterator start, int number, function);
```

比较函数

```
@Jannie Kung
//  字典序比较返回类型bool，但不能完成排序等等的操作
lexicographical_compare(iterator first, iterator first +
sizeOfFirstArray, iterator second, iterator second +
sizeOfSecondArray);
//  比较两个container是否相等，返回类型bool
equal(iterator first_to_begin, iterator first_to_finish, iterator
second);
//  若两个container不匹配，将返回第一个container不匹配的迭代器。
mismatch(iterator first_to_begin, iterator first_to_finish,
iterator second);
```

remove系

```
@Jannie Kung
//  从开始的迭代器到结束迭代器把和value相等的值去掉，
//  返回新的iterator end，表明结束的迭代器不再
//  是原来的finish迭代器
remove(iterator start, iterator finish, T value);
//  和上面一个函数不同的是它是把结果重新
//  赋给一个新的container，返回类型为void
remove_copy(iterator start, iterator finish, iterator
new_one_of_another_container, T value);
//  和上面两个函数不同的是它是通过函数进行筛选，
//  而不仅仅是value
remove_if(iterator start, iterator finish, function);
//  和上面函数不同的是它是通过函数进行筛选后把结果
//  放在新的container中
remove_copy_if(iterator start, iterator finish, iterator
new_one_of_another_container, function);
```

replace系

```
@Jannie Kung
//  把start到finish迭代器中的elder_value中换成new_value
replace(iterator start, iterator finish, T elder_value, T
new_value);
// 和上面函数不同的是它是在原有功能基础上把结果放到
//  一个新的container里面
replace_copy(iterator start, iterator finish, iterator new_start,
T elder_value, T new_value);
//  和replace系列第一个函数不同的是它把筛选同样地
//  通过调用函数完成
replace_if(iterator start, iterator finish, function, T
new_value);
//  和上面函数不同的是它是把replace后的结果放到
//  新的container里面
replace_copy_if(iterator start, iterator finish, iterator
new_start, function, T new_value);
```

algorithm和numeric头文件

mathematical algorithms

```
@Jannie Kung
//  任意打乱container中数据的顺序（目前还不清楚有什么作用）
random_shuffle（iterator start, iterator finish）;
//  记下从start到finish的迭代器中value出现的次数，
//  返回类型int
count(iterator start, iterator finish, T value);
//  通过function而不是value筛选条件
count_if(iterator start, iterator finish, function);
// 返回从start到finish迭代器中的最大值
min_element(iterator start, iterator finish);
//  返回从start到finish迭代器中的最小值
max_element(iterator start, iterator finish);
//  值得注意的是，要保证使用max_element和min_element
//  函数的时候container不能为空，否则将会返回end()，
//  出现compile error.

//  通过对原container调用function，之后把结果
//  赋给新的container
transform(iterator start, iterator finish, iterator new_start,
function);
//  类似for循环遍历从start到finish的迭代器，每一次
//  都调用function函数
for_each(iterator start, iterator finish, function);
//  累加从start到finish迭代器的数据，
//  sum代表初始化的和，可以人为设定为任意值，
//  而且，该函数定义在numeric而非algorithm头文件中
accumulate(iterator start, iterator finish, sum);
```

Basic Searching & Sorting Algorithms

```
@Jannie Kung
//  在start到finish的迭代器中寻找是否有与value相等的，
//  如果有，返回该值所在的迭代器，否则返回end()
find(iterator start, iterator finish, T value);
//  通过调用函数调用筛选的条件，如一个int类型的vector中，
//  函数可以实现greater than nine功能
find_if(iterator start, iterator finish, function);
//  sort函数不解释，而且可以加入第三个函数参数以函数为标准进行排序，
//  而且，sort这个傲娇的孩子只接受random_access iterator，
//  如vector一类，类似deque，list这一类的迭代器是不可行的
sort(iterator start, iterator finish);
//  二分查找value,查看是否在start到finish迭代器中
//  是否有等于该值的位置，若有，返回true；反之，返回false。
binary_search(iterator start, iterator finish, T value);
```

Swap

```
@Jannie Kung
//  简单swap不解释
swap(T & value, T & value1);
//  进行的是iterator的交换
iter_swap(T & value, T & value1);
//  start到middle的迭代器和middle到finish的迭代器进行交换
swap_ranges(iterator start, iterator middle, iterator finish);
```

其他各项函数

```
@Jannie Kung
//  把start到finish的迭代器复制到新的container里面，
//  而且是从后面开始插入
copy_backward(iterator start, iterator finish, iterator
new_start);
//  把两个container融合到一个新的container里面
merge(iterator first_start, iterator first_finish, iterator
second_start, iterator second_finish, iterator new_start);
//  把从start到finish的迭代器里面重复的内容删去，
//  返回新的end iterator
unique(iterator start, iterator finish);
//  把container倒序排放
reverse(iterator start, iterator finish);
//  first到middle的迭代器和middle到finish分别排序再组合，
//  再排序，可以用于排序，而且iterator不同于sort那般局限，
//  使用范围更广
inplace_merge(iterator first, iterator middle, iterator finish);
```


作者：Jannie Kung
邮箱：kcnnow(at)gmail.com