|  |  |
| --- | --- |
| [std::stoi](http://classfoo.com/ccby/article/IakWNV) **C++11** | 将字符串转化成带符号（Signed）整数 |
| [std::stol](http://classfoo.com/ccby/article/0itzNW) **C++11** | 将字符串转化成带符号整数 |
| [std::stoll](http://classfoo.com/ccby/article/qtzNNX) **C++11** | 将字符串转化成带符号整数 |
| [std::stoul](http://classfoo.com/ccby/article/ipC2NY) **C++11** | 将字符串转化成无符号（Unsigned）整数 |
| [std::stoull](http://classfoo.com/ccby/article/yEMrNZ) **C++11** | 将字符串转化成无符号整数 |
| [std::stof](http://classfoo.com/ccby/article/qrNlN0) **C++11** | 将字符串转化成浮点数 |
| [std::stod](http://classfoo.com/ccby/article/dqtsN1) **C++11** | 将字符串转化成浮点数 |
| [std::stold](http://classfoo.com/ccby/article/kHthN2) **C++11** | 将字符串转化成浮点数 |
| [std::to\_string](http://classfoo.com/ccby/article/va0VN3) **C++11** | 将一个整数或浮点数转化成字符串 |
| [std::to\_wstring](http://classfoo.com/ccby/article/z8LPN4) **C++11** | 将一个整数或浮点数转化成宽字符串 |
| [std::hash<std::string>](http://classfoo.com/ccby/article/KM4GN5) **C++11** | 字符串的啥希（Hash）支持 |
|  |

由头文件 [<cctype>](http://classfoo.com/ccby/article/uaR5c0) 定义的相关内容（实际引用的是 [<locale>](http://classfoo.com/ccby/article/Hgkxc4) 中的内容，也就是说，下面的内容即属于**[字符串库](http://classfoo.com/ccby/article/PMZdbE)**，又属于[**本地化库**](http://classfoo.com/ccby/article/mwuqbw)）：

|  |  |
| --- | --- |
| [isalnum](http://classfoo.com/ccby/article/qHubRj) | 检测一个字符是否是数字或字母类型的（Alphanumeric） |
| [isalpha](http://classfoo.com/ccby/article/AYDbRk) | 检测一个字符是否是字母类型的（Alphabetic） |
| [islower](http://classfoo.com/ccby/article/89kxRl) | 检测一个字符是否是小写字母 |
| [isupper](http://classfoo.com/ccby/article/cgwKRm) | 检测一个字符是否是大写字母 |
| [isdigit](http://classfoo.com/ccby/article/CuckRn) | 检测一个字符是否是数字类型的 |
| [isxdigit](http://classfoo.com/ccby/article/mI4mRo) | 检测一个字符是否是十六进制（Hexadecimal）字符 |
| [iscntrl](http://classfoo.com/ccby/article/WIgkRp) | 检测一个字符是否是控制（Control）字符 |
| [isgraph](http://classfoo.com/ccby/article/frgvRq) | 检测一个字符是否是图型（Graphical）字符 |
| [isspace](http://classfoo.com/ccby/article/pA9cRr) | 检测一个字符是否是空格字符 |
| [isblank](http://classfoo.com/ccby/article/diIYRs) **C++11** | 检测一个字符是否是空档字符 |
| [isprint](http://classfoo.com/ccby/article/Ngm2Rt) | 检测一个字符是否是可打印字符 |
| [ispunct](http://classfoo.com/ccby/article/ou8KRu) | 检测一个字符是否是标点符号字符 |

由头文件 [<cctype>](http://classfoo.com/ccby/article/uaR5c0) 定义的相关内容（实际引用的是 [<locale>](http://classfoo.com/ccby/article/Hgkxc4) 中的内容，也就是说，下面的内容即属于**[字符串库](http://classfoo.com/ccby/article/PMZdbE)**，又属于[**本地化库**](http://classfoo.com/ccby/article/mwuqbw)）：

|  |  |
| --- | --- |
| [tolower](http://classfoo.com/ccby/article/rUfARv) | 将一字符转化成对应的小写字母 |
| [toupper](http://classfoo.com/ccby/article/6GXRRw) | 将一字符转化成对应的大写字母 |

由头文件 [<cstdlib>](http://classfoo.com/ccby/article/XTBfcd) 定义的相关内容：

|  |  |
| --- | --- |
| [atof](http://classfoo.com/ccby/article/fwa7Rx) | 将一个单字节字符串转化成一个浮点数 |
| [atoi](http://classfoo.com/ccby/article/vgarRy) | 将一个单字节字符串转化成一个整数 |
| [atol](http://classfoo.com/ccby/article/67FfRz) | 将一个单字节字符串转化成一个整数 |
| [atoll](http://classfoo.com/ccby/article/okPARA) **C++11** | 将一个单字节字符串转化成一个整数 |
| [strtol](http://classfoo.com/ccby/article/D9waRB) | 将一个单字节字符串转化成一个整数 |
| [strtoll](http://classfoo.com/ccby/article/3oWJRC) **C++11** | 将一个单字节字符串转化成一个整数 |
| [strtoul](http://classfoo.com/ccby/article/M3fMRD) | 将一个单字节字符串转化成一个无符号整数 |
| [strtoull](http://classfoo.com/ccby/article/02eIRE) **C++11** | 将一个单字节字符串转化成一个无符号整数 |
| [strtof](http://classfoo.com/ccby/article/cFmdRF) **C++11** | 将一个单字节字符串转化成一个浮点数 |
| [strtod](http://classfoo.com/ccby/article/RKk2RG) | 将一个单字节字符串转化成一个浮点数 |
| [strtold](http://classfoo.com/ccby/article/UOrQRH) **C++11** | 将一个单字节字符串转化成一个浮点数 |

由头文件 [<cstring>](http://classfoo.com/ccby/article/RpRRcG) 定义的相关内容：

|  |  |
| --- | --- |
| [strcpy](http://classfoo.com/ccby/article/nIhSRK) | 拷贝一个字符串到另一个字符串中 |
| [strncpy](http://classfoo.com/ccby/article/ol3bRL) | 从一个字符串中拷贝指定数目的字符到另一个字符串中 |
| [strcat](http://classfoo.com/ccby/article/BCHbRM) | 连接两个字符串 |
| [strncat](http://classfoo.com/ccby/article/3M5tRN) | 将一个字符串中指定数目的字符拷贝连接到另一个字符串后面 |
| [strxfrm](http://classfoo.com/ccby/article/ZyZ3RO) | 使用本地化转化字符串 |

由头文件 [<cstring>](http://classfoo.com/ccby/article/RpRRcG) 定义的相关内容：

|  |  |
| --- | --- |
| [strlen](http://classfoo.com/ccby/article/CcZhRP) | 返回给定字符串的长度 |
| [strcmp](http://classfoo.com/ccby/article/h8HARQ) | 比较两个字符串 |
| [strncmp](http://classfoo.com/ccby/article/YXg1RR) | 比较两个字符串中指定数目的字符 |
| [strcoll](http://classfoo.com/ccby/article/bPtSRS) | 按当前本地化属性比较两个字符串 |
| [strchr](http://classfoo.com/ccby/article/AOfpRT) | 返回指定字符在字符串中第一次出现的位置 |
| [strrchr](http://classfoo.com/ccby/article/2H4KRU) | 返回指定字符在字符串中最后一次（即返向第一次）出现的位置 |
| [strspn](http://classfoo.com/ccby/article/w5zJRV) | 返回字符串中第一个不在指定字符串中出现的字符的下标 |
| [strcspn](http://classfoo.com/ccby/article/5nK2RW) | 返回字符串中第一个在指定字符串中出现的字符的下标 |
| [strpbrk](http://classfoo.com/ccby/article/Vnt1RX) | 返回指向字符串中第一个在指定字符串中出现的字符的指针 |
| [strstr](http://classfoo.com/ccby/article/MbFBRY) | 在一个字符串中查找另一个字符串第一次出现的位置 |
| [strtok](http://classfoo.com/ccby/article/FO7IRZ) | 使用分隔符字符串分解另一个字符串 |

由头文件 [<cstring>](http://classfoo.com/ccby/article/RpRRcG) 定义的相关内容（既属于[字符串库](http://classfoo.com/ccby/article/PMZdbE)，又属于[通用工具库](http://classfoo.com/ccby/article/njPrN)）：

|  |  |
| --- | --- |
| memchr | 在一个缓存中搜索指定字符 |
| memcmp | 比较两个缓存 |
| memset | 用指定字符填充一个缓存 |
| memcpy | 拷贝一个缓存到另一个缓存 |
| memmove | 移动一个缓存到另一个缓存 |

1. // [<string>](http://classfoo.com/ccby/article/TWd5cZ)
2. // 默认（1）
3. explicit basic\_string (const allocator\_type& alloc = allocator\_type());
4. // 拷贝（2）
5. basic\_string (const basic\_string& str);
6. basic\_string (const basic\_string& str, const allocator\_type& alloc);
7. // 子串（3）
8. basic\_string (const basic\_string& str, size\_type pos, size\_type len = npos,
9. const allocator\_type& alloc = allocator\_type());
10. // 来自 C 型字符串（4）
11. basic\_string (const charT\* s, const allocator\_type& alloc = allocator\_type());
12. // from buffer (5)
13. basic\_string (const charT\* s, size\_type n,
14. const allocator\_type& alloc = allocator\_type());
15. // 填充（6）
16. basic\_string (size\_type n, charT c,
17. const allocator\_type& alloc = allocator\_type());
18. // 范围（7）
19. template <class InputIterator>
20. basic\_string (InputIterator first, InputIterator last,
21. const allocator\_type& alloc = allocator\_type());
22. // 初始化列表（8）
23. basic\_string (initializer\_list<charT> il,
24. const allocator\_type& alloc = allocator\_type());
25. // 移动（9）
26. basic\_string (basic\_string&& str) noexcept;
27. basic\_string (basic\_string&& str, const allocator\_type& alloc);
28. 迭代器

|  |  |
| --- | --- |
| [begin](http://classfoo.com/ccby/article/eT63Nb) | 返回指向字符串起始位置的迭代器（[iterator](http://classfoo.com/ccby/article/zIeqik)） |
| [end](http://classfoo.com/ccby/article/xtceNc) | 返回指向字符串末尾位置的迭代器 |
| [rbegin](http://classfoo.com/ccby/article/OSvvNd) | 返回指向字符串逆序起始位置的逆序迭代器（[reverse\_iterator](http://classfoo.com/ccby/article/RDvFd0)） |
| [rend](http://classfoo.com/ccby/article/Jk0nNe) | 返回指向字符串逆序末尾位置的逆序迭代器 |
| [cbegin](http://classfoo.com/ccby/article/s1HTNf) **C++11** | 返回指向字符串起始位置的常迭代器（[const\_iterator](http://classfoo.com/ccby/article/x5I2il)） |
| [cend](http://classfoo.com/ccby/article/zK9lNg) **C++11** | 返回指向字符串末尾位置的常迭代器 |
| [crbegin](http://classfoo.com/ccby/article/CWrxNh) **C++11** | 返回指向字符串逆序起始位置的常逆序迭代器（[const\_reverse\_iterator](http://classfoo.com/ccby/article/g2bxim)） |
| [crend](http://classfoo.com/ccby/article/74RCNi) **C++11** | 返回指向字符串逆序末尾位置的常逆序迭代器 |

1. 元素数量

|  |  |
| --- | --- |
| [size](http://classfoo.com/ccby/article/EIgfNj) | 返回有效字符个数 |
| [length](http://classfoo.com/ccby/article/oR26Nk) | 返回有效字符个数，跟 size 返回相同的值 |
| [max\_size](http://classfoo.com/ccby/article/KZe4Nl) | 返回支持的最大字符个数 |
| [resize](http://classfoo.com/ccby/article/8T5HNm) | 改变有效字符个数 |
| [capacity](http://classfoo.com/ccby/article/bjXRNn) | 返回当前可使用的最大字符内存块数（即存储容器） |
| [reserve](http://classfoo.com/ccby/article/bvLkNo) | 请求改变存储容量 |
| [clear](http://classfoo.com/ccby/article/pO0aNp) | 清空字符串 |
| [empty](http://classfoo.com/ccby/article/amKKNq) | 检测字符串是否是空的 |
| [shrink\_to\_fit](http://classfoo.com/ccby/article/BWt1Nr) **C++11** | 请求移除未使用的存储空间 |

1. 元素访问

|  |  |
| --- | --- |
| [operator[]](http://classfoo.com/ccby/article/IHDiNs) | 访问字符 |
| [at](http://classfoo.com/ccby/article/GRxcNt) | 访问字符 |
| [back](http://classfoo.com/ccby/article/5EZtNu) **C++11** | 访问最后一个字符 |
| [front](http://classfoo.com/ccby/article/5rnrNv) **C++11** | 访问第一个字符 |

1. 修改器

|  |  |
| --- | --- |
| [operator+=](http://classfoo.com/ccby/article/iCn1Nw) | 附加（Append）到字符串 |
| [append](http://classfoo.com/ccby/article/8PMcNx) | 附加到字符串 |
| [push\_back](http://classfoo.com/ccby/article/b0zVNy) | 附加字符到字符串 |
| [assign](http://classfoo.com/ccby/article/vMYONz) | 赋值内容到字符串 |
| [insert](http://classfoo.com/ccby/article/usVeNA) | 插入到字符串 |
| [erase](http://classfoo.com/ccby/article/lC2uNB) | 从字符串中清除字符 |
| [replace](http://classfoo.com/ccby/article/YVWyNC) | 替换字符串的部分内容 |
| [swap](http://classfoo.com/ccby/article/NjlBND) | 交换字符串对象 |
| [pop\_back](http://classfoo.com/ccby/article/uAzSNE) **C++11** | 删除最后一个字符 |

1. 字符串操作

|  |  |
| --- | --- |
| [c\_str](http://classfoo.com/ccby/article/raRWNF) | 返回 C 型字符串 |
| [data](http://classfoo.com/ccby/article/qG7NNG) | 返回字符串数据 |
| [get\_allocator](http://classfoo.com/ccby/article/H2dINH) | 获得内存分配器 |
| [copy](http://classfoo.com/ccby/article/OSMyNI) | 从字符串中拷贝字符序列 |
| [find](http://classfoo.com/ccby/article/JRLMNJ) | 从字符串查找字符或字符串，返回第一次找到的位置 |
| [rfind](http://classfoo.com/ccby/article/w4aKNK) | 从字符串查找字符或字符串，返回最后一次找到的位置 |
| [find\_first\_of](http://classfoo.com/ccby/article/ymwFNL) | 从字符串查找（由参数确定的）任意匹配的字符，返回第一次找到的位置 |
| [find\_last\_of](http://classfoo.com/ccby/article/gDXHNM) | 从字符串查找（由参数确定的）任意匹配的字符，返回最后一次找到的位置 |
| [find\_first\_not\_of](http://classfoo.com/ccby/article/HyBsNN) | 从字符串查找（由参数确定的）任意不匹配的字符，返回第一次找到的位置 |
| [find\_last\_not\_of](http://classfoo.com/ccby/article/6B57NO) | 从字符串查找（由参数确定的）任意不匹配的字符，返回最后一次找到的位置 |
| [substr](http://classfoo.com/ccby/article/MIfbNP) | 产生子串basic\_string substr (size\_type pos = 0, size\_type len = npos) const; |
| [compare](http://classfoo.com/ccby/article/DPXzNQ) | 比较字符串 |

## 不修改内容的序列操作

由头文件 [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) 定义。

|  |  |
| --- | --- |
| [adjacent\_find](http://classfoo.com/ccby/article/EQS6eT) | 查找两个相邻（Adjacent）的等价（Identical）元素 |
| [all\_of](http://classfoo.com/ccby/article/ZT4neL) **C++11** | 检测在给定范围中是否所有元素都满足给定的条件 |
| [any\_of](http://classfoo.com/ccby/article/GYvOeM) **C++11** | 检测在给定范围中是否存在元素满足给定条件 |
| [count](http://classfoo.com/ccby/article/tZaqeV) | 返回值等价于给定值的元素的个数 |
| [count\_if](http://classfoo.com/ccby/article/vlCuhH) | 返回值满足给定条件的元素的个数 |
| [equal](http://classfoo.com/ccby/article/g6yteX) | 返回两个范围是否相等 |
| [find](http://classfoo.com/ccby/article/uApYeP) | 返回第一个值等价于给定值的元素 |
| [find\_end](http://classfoo.com/ccby/article/6LCXeS) | 查找范围 A 中与范围 B 等价的子范围最后出现的位置 |
| [find\_first\_of](http://classfoo.com/ccby/article/CFnEeU) | 查找范围 A 中第一个与范围 B 中任一元素等价的元素的位置 |
| [find\_if](http://classfoo.com/ccby/article/eM1IeQ) | 返回第一个值满足给定条件的元素 |
| [find\_if\_not](http://classfoo.com/ccby/article/tyhWeR) **C++11** | 返回第一个值不满足给定条件的元素 |
| [for\_each](http://classfoo.com/ccby/article/2CGheO) | 对范围中的每个元素调用指定函数 |
| [mismatch](http://classfoo.com/ccby/article/BsZdeW) | 返回两个范围中第一个元素不等价的位置 |
| [none\_of](http://classfoo.com/ccby/article/g4CWeN) **C++11** | 检测在给定范围中是否不存在元素满足给定的条件 |
| [search](http://classfoo.com/ccby/article/60aseZ) | 在范围 A 中查找第一个与范围 B 等价的子范围的位置 |
| [search\_n](http://classfoo.com/ccby/article/0UGXe0) | 在给定范围中查找第一个连续 n 个元素都等价于给定值的子范围的位置 |

* #include [<iostream>](http://classfoo.com/ccby/article/Yxj7ge)
* #include [<algorithm>](http://classfoo.com/ccby/article/7ZqseK)
* #include [<functional>](http://classfoo.com/ccby/article/Qz6HcH)
* #include [<vector>](http://classfoo.com/ccby/article/KCu6dM)
* // 用在此处是为了方便简洁, 在实际编程中慎用
* using namespace std;
* void main()
* {
* int iarray[]={0,1,2,3,4,5,6,6,6,7,8};
* vector<int> foo1(iarray,iarray+sizeof(iarray)/sizeof(int));
* int iarray1[]={6,6};
* vector<int> foo2(iarray1,iarray1+sizeof(iarray1)/sizeof(int));
* int iarray2[]={5,6};
* vector<int> foo3(iarray2,iarray2+sizeof(iarray2)/sizeof(int));
* int iarray3[]={0,1,2,3,4,5,7,7,7,9,7};
* vector<int> foo4(iarray3,iarray3+sizeof(iarray3)/sizeof(int));
* //找出foo1之中相邻元素值相等的第一个元素
* cout<<\*adjacent\_find(foo1.begin(),foo1.end())<<endl;
* //找出foo1之中元素值为6的元素个数
* cout<<count(foo1.begin(),foo1.end(),6)<<endl;
* //找出foo1之中小于7的元素个数
* cout<<count\_if(foo1.begin(),foo1.end(),bind2nd(less<int>(),7))<<endl;
* //找出foo1之中元素值为4的第一个元素所在位置的元素
* cout<<\*find(foo1.begin(),foo1.end(),4)<<endl;
* //找出foo1之中大于2的第一个元素所在位置的元素
* cout<<\*find\_if(foo1.begin(),foo1.end(),bind2nd(greater<int>(),2))
* <<endl;
* //找出foo1之中子序列foo2所出现的最后一个位置，再往后3个位置的元素
* cout<<\*(find\_end(foo1.begin(),foo1.end(),foo2.begin(),
* foo2.end())+3)<<endl;
* //找出foo1之中子序列foo2所出现的第一个位置，再往后3个位置的元素
* cout<<\*(find\_first\_of(foo1.begin(),foo1.end(),foo2.begin(),
* foo2.end())+3)<<endl;
* //子序列foo3在foo1中出现的起点位置元素
* cout<<\*search(foo1.begin(),foo1.end(),foo3.begin(),foo3.end())
* <<endl;
* //查找连续出现3个6的起点位置元素
* cout<<\*search\_n(foo1.begin(),foo1.end(),3,6,equal\_to<int>())<<endl;
* //判断两个区间foo1和foo4相等否(0为假，1为真）
* cout << equal(foo1.begin(), foo1.end(), foo4.begin()) << endl;
* //查找区间foo4在foo1中不匹配点的位置
* pair<std::vector<int>::iterator,std::vector<int>::iterator>result=
* mismatch(foo1.begin(),foo1.end(),foo4.begin());
* cout<< result.first - foo1.begin() << endl;
* }

## 修改内容的序列操作

由头文件 [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) 定义。

|  |  |
| --- | --- |
| [copy](http://classfoo.com/ccby/article/ViRLe1) | 将一个范围中的元素拷贝到新的位置处 |
| [copy\_backward](http://classfoo.com/ccby/article/9Zb4e4) | 将一个范围中的元素按逆序拷贝到新的位置处 |
| [copy\_if](http://classfoo.com/ccby/article/ImeAe3) **C++11** | 将一个范围中满足给定条件的元素拷贝到新的位置处 |
| [copy\_n](http://classfoo.com/ccby/article/alRKe2) **C++11** | 拷贝 n 个元素到新的位置处 |
| [fill](http://classfoo.com/ccby/article/W2Q6fd) | 将一个范围的元素赋值为给定值 |
| [fill\_n](http://classfoo.com/ccby/article/8D8vfe) | 将某个位置开始的 n 个元素赋值为给定值 |
| [generate](http://classfoo.com/ccby/article/rJ9eff) | 将一个函数的执行结果保存到指定范围的元素中，用于批量赋值范围中的元素 |
| [generate\_n](http://classfoo.com/ccby/article/QURdfg) | 将一个函数的执行结果保存到指定位置开始的 n 个元素中 |
| [iter\_swap](http://classfoo.com/ccby/article/0zOGe7) | 交换两个迭代器（Iterator）指向的元素 |
| [move](http://classfoo.com/ccby/article/y6vrV7) **C++11** | 将一个范围中的元素移动到新的位置处 |
| [move\_backward](http://classfoo.com/ccby/article/jCvEe5) **C++11** | 将一个范围中的元素按逆序移动到新的位置处 |
| [random\_shuffle](http://classfoo.com/ccby/article/gCDFfp) | 随机打乱指定范围中的元素的位置 |
| [remove](http://classfoo.com/ccby/article/1MQYfh) | 将一个范围中值等价于给定值的元素删除 |
| [remove\_if](http://classfoo.com/ccby/article/vFepfi) | 将一个范围中值满足给定条件的元素删除 |
| [remove\_copy](http://classfoo.com/ccby/article/DN5qI7) | 拷贝一个范围的元素，将其中值等价于给定值的元素删除 |
| [remove\_copy\_if](http://classfoo.com/ccby/article/KxXoI8) | 拷贝一个范围的元素，将其中值满足给定条件的元素删除 |
| [replace](http://classfoo.com/ccby/article/jwIre9) | 将一个范围中值等价于给定值的元素赋值为新的值 |
| [replace\_copy](http://classfoo.com/ccby/article/vcHcfb) | 拷贝一个范围的元素，将其中值等价于给定值的元素赋值为新的值 |
| [replace\_copy\_if](http://classfoo.com/ccby/article/0Lrqfc) | 拷贝一个范围的元素，将其中值满足给定条件的元素赋值为新的值 |
| [replace\_if](http://classfoo.com/ccby/article/NmDLfa) | 将一个范围中值满足给定条件的元素赋值为新的值 |
| [reverse](http://classfoo.com/ccby/article/cNTmfl) | 反转排序指定范围中的元素 |
| [reverse\_copy](http://classfoo.com/ccby/article/bg2ffm) | 拷贝指定范围的反转排序结果 |
| [rotate](http://classfoo.com/ccby/article/Hv0wfn) | 循环移动指定范围中的元素 |
| [rotate\_copy](http://classfoo.com/ccby/article/ohMRfo) | 拷贝指定范围的循环移动结果 |
| [shuffle](http://classfoo.com/ccby/article/vUmBfq) **C++11** | 用指定的随机数引擎随机打乱指定范围中的元素的位置 |
| [swap](http://classfoo.com/ccby/article/t8JYcO) | 交换两个对象的值 |
| [swap\_ranges](http://classfoo.com/ccby/article/S4j7e6) | 交换两个范围的元素 |
| [transform](http://classfoo.com/ccby/article/qZU2e8) | 对指定范围中的每个元素调用某个函数以改变元素的值 |
| [unique](http://classfoo.com/ccby/article/YQJ1fj) | 删除指定范围中的所有连续重复元素，仅仅留下每组等值元素中的第一个元素。 |
| [unique\_copy](http://classfoo.com/ccby/article/kizwfk) | 拷贝指定范围的唯一化（参考上述的 unique）结果 |

## [划分操作](http://classfoo.com/ccby/article/tZTzs#sec_4QjYtz)

由头文件 [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) 定义。

|  |  |
| --- | --- |
| [is\_partitioned](http://classfoo.com/ccby/article/i2r1fr) **C++11** | 检测某个范围是否按指定谓词（Predicate）划分过 |
| [partition](http://classfoo.com/ccby/article/ggYOfs) | 将某个范围划分为两组 |
| [partition\_copy](http://classfoo.com/ccby/article/IjuNfu) **C++11** | 拷贝指定范围的划分结果 |
| [partition\_point](http://classfoo.com/ccby/article/eZnXfv) **C++11** | 返回被划分范围的划分点 |
| [stable\_partition](http://classfoo.com/ccby/article/LOb0ft) | 稳定划分，两组元素各维持相对顺序 |

## [排序操作](http://classfoo.com/ccby/article/tZTzs#sec_z3Y8tA)

由头文件 [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) 定义。

|  |  |
| --- | --- |
| [is\_sorted](http://classfoo.com/ccby/article/xwYEfA) **C++11** | 检测指定范围是否已排序 |
| [is\_sorted\_until](http://classfoo.com/ccby/article/gZtgfB) **C++11** | 返回最大已排序子范围 |
| [nth\_element](http://classfoo.com/ccby/article/fXklfC) | 部份排序指定范围中的元素，使得范围按给定位置处的元素划分 |
| [partial\_sort](http://classfoo.com/ccby/article/hegUfy) | 部份排序 |
| [partial\_sort\_copy](http://classfoo.com/ccby/article/YFtUfz) | 拷贝部分排序的结果 |
| [sort](http://classfoo.com/ccby/article/iA1Yfw) | 排序 |
| [stable\_sort](http://classfoo.com/ccby/article/f7jefx) | 稳定排序 |

## [二分法查找操作](http://classfoo.com/ccby/article/tZTzs#sec_gpILtB)

由头文件 [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) 定义。

|  |  |
| --- | --- |
| [binary\_search](http://classfoo.com/ccby/article/WftIfG) | 判断范围中是否存在值等价于给定值的元素 |
| [equal\_range](http://classfoo.com/ccby/article/YZjAfF) | 返回范围中值等于给定值的元素组成的子范围 |
| [lower\_bound](http://classfoo.com/ccby/article/b6OsfD) | 返回指向范围中第一个值大于或等于给定值的元素的迭代器 |
| [upper\_bound](http://classfoo.com/ccby/article/FN7IfE) | 返回指向范围中第一个值大于给定值的元素的迭代器 |

* #include [<vector>](http://classfoo.com/ccby/article/KCu6dM)
* #include [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) // for lower\_bound()、greater<int>()
* #include [<functional>](http://classfoo.com/ccby/article/Qz6HcH)
* #include [<iostream>](http://classfoo.com/ccby/article/Yxj7ge)
* // 用在此处是为了方便简洁, 在实际编程中慎用
* using namespace std;
* // 返回elem1的绝对值是否比elem2的绝对值小
* bool mod\_lesser(int elem1, int elem2)
* {
* if(elem1 < 0)
* elem1 = - elem1;
* if(elem2 < 0)
* elem2 = - elem2;
* return (elem1 < elem2);
* }
* int main(void)
* {
* vector <int> foo1;
* vector <int>::iterator Iter1, Result1;
* int i, j;
* // 插入数据
* for(i = -3; i <= 6; i++)
* foo1.push\_back(i);
* for(j =-5; j <= 2; j++)
* foo1.push\_back(j);
* cout<<"操作: sort(foo1.begin(), foo1.end())"<< endl;
* sort(foo1.begin(), foo1.end());
* cout<<"使用默认的二元谓词“是否小于”排序后的结果：";
* for(Iter1 = foo1.begin(); Iter1 != foo1.end(); Iter1++)
* cout<<\*Iter1<<" ";
* cout<<endl;
* vector <int> foo2(foo1);
* vector <int>::iterator Iter2, Result2;
* cout<<"操作: sort(foo2.begin(), foo2.end(), greater<int>())"<<endl;
* sort(foo2.begin(), foo2.end(), greater<int>());
* cout<<"使用二元谓词“是否大于”排序后的结果：";
* for(Iter2 = foo2.begin(); Iter2 != foo2.end(); Iter2++)
* cout<<\*Iter2<<" ";
* cout<<endl;
* vector <int> foo3(foo1);
* vector <int>::iterator Iter3, Result3;
* cout<<"操作: sort(foo3.begin(), foo3.end(), mod\_lesser)"<<endl;
* sort(foo3.begin(), foo3.end(), mod\_lesser);
* cout<<"使用二元谓词 mod\_lesser 排序后的结果：";
* for(Iter3 = foo3.begin(); Iter3 != foo3.end(); Iter3++)
* cout<<\*Iter3<<" ";
* cout<<endl;
* cout<<"操作: lower\_bound(foo1.begin(), foo1.end(), 5)"<<endl;
* // 使用默认的二元谓词“是否小于”二分查找值为5的元素的位置
* Result1 = lower\_bound(foo1.begin(), foo1.end(), 5);
* cout<<"在foo1中二分查找值为5的元素的位置对应的值（即5）："<<\*Result1<<endl;
* Result2 = lower\_bound(foo2.begin(), foo2.end(), 5, greater<int>());
* // 使用二元谓词函数 greater<int>() 二分查找值为5的元素的位置
* cout<<"在foo1中二分查找值为5的元素的位置对应的值（即5）："<<\*Result2<<endl;
* cout<<"操作: lower\_bound(foo3.begin(), foo3.end(), 5, mod\_lesser)"<<endl;
* // 使用二元谓词函数 mod\_lesser 二分查找值为5的元素的位置
* Result3 = lower\_bound(foo3.begin(), foo3.end(), 5, mod\_lesser);
* cout<<"在foo1中二分查找值为5的元素的位置对应的值（即5或-5）："<<\*Result3<<endl;
* return 0;
* }
* #include [<algorithm>](http://classfoo.com/ccby/article/7ZqseK)
* #include [<iostream>](http://classfoo.com/ccby/article/Yxj7ge)
* #include [<iterator>](http://classfoo.com/ccby/article/7ZPydZ)
* #include [<vector>](http://classfoo.com/ccby/article/KCu6dM)
* // 用在此处是为了方便简洁, 在实际编程中慎用
* using namespace std;
* int main()
* {
* vector<int> v;
* vector<int>::iterator iter;
* pair<vector<int>::iterator, vector<int>::iterator> vecpair;
* for(int i = 1; i<= 20; i++) {
* v.push\_back(i%6);
* }
* sort(v.begin(), v.end());
* //将各个元素拷贝到输出流迭代器中
* copy(v.begin(), v.end(), ostream\_iterator<int>(cout, " "));
* cout << endl;
* /\* lower\_bound \*/
* cout << "lower\_bound function, value = 3: " << endl;
* iter = lower\_bound(v.begin(), v.end(), 3);
* cout << " [first, iter] = ";
* copy(v.begin(), iter, ostream\_iterator<int>(cout, " "));
* cout << endl;
* cout << " [iter, last] = ";
* copy(iter, v.end(), ostream\_iterator<int>(cout, " "));
* cout << endl;
* /\* upper\_bound \*/
* cout << "upper\_bound function, value = 3: " << endl;
* iter = upper\_bound(v.begin(), v.end(), 3);
* cout << " [first, iter] = ";
* copy(v.begin(), iter, ostream\_iterator<int>(cout, " "));
* cout << endl;
* cout << " [iter, last] = ";
* copy(iter, v.end(), ostream\_iterator<int>(cout, " "));
* cout << endl;
* /\* equal\_range \*/
* cout << "euqual\_range function value = 3: " << endl;
* vecpair = equal\_range(v.begin(), v.end(), 3);
* cout << " [vecpair->first, vecpair->second] = ";
* copy(vecpair.first, vecpair.second, ostream\_iterator<int>(cout, " "));
* cout << endl;
* /\* binary\_search \*/
* cout << "binary\_search function value = 3: " << endl;
* cout << "3 is " << (binary\_search(v.begin(), v.end(), 3) ? "": "not ") << " in array" << endl;
* /\* binary\_search \*/
* cout << "binary\_search function value = 6: " << endl;
* cout << "6 is " << (binary\_search(v.begin(), v.end(), 6) ? "": "not ") << " in array" << endl;
* }

## [集合操作](http://classfoo.com/ccby/article/tZTzs#sec_exlwtC)

由头文件 [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) 定义。

|  |  |
| --- | --- |
| [includes](http://classfoo.com/ccby/article/6lkafJ) | 判断一个集合是否是另一个集合的子集 |
| [inplace\_merge](http://classfoo.com/ccby/article/dHjOfI) | 就绪合并 |
| [merge](http://classfoo.com/ccby/article/30GYfH) | 合并 |
| [set\_difference](http://classfoo.com/ccby/article/pBoqfM) | 获得两个集合的差集 |
| [set\_intersection](http://classfoo.com/ccby/article/cE4pfL) | 获得两个集合的交集 |
| [set\_symmetric\_difference](http://classfoo.com/ccby/article/NP6ffN) | 获得两个集合的对称差 |
| [set\_union](http://classfoo.com/ccby/article/Y749fK) | 获得两个集合的并集 |

## [堆操作](http://classfoo.com/ccby/article/tZTzs#sec_GdontD)

由头文件 [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) 定义。

|  |  |
| --- | --- |
| [is\_heap](http://classfoo.com/ccby/article/ZaX3fS) | 检测给定范围是否满足堆结构 |
| [is\_heap\_until](http://classfoo.com/ccby/article/ietNI9) **C++11** | 检测给定范围中满足堆结构的最大子范围 |
| [make\_heap](http://classfoo.com/ccby/article/Jh0dfQ) | 用给定范围构造出一个堆 |
| [pop\_heap](http://classfoo.com/ccby/article/6uJ1fP) | 从一个堆中删除最大的元素 |
| [push\_heap](http://classfoo.com/ccby/article/B7CtfO) | 向堆中增加一个元素 |
| [sort\_heap](http://classfoo.com/ccby/article/lvILfR) | 将满足堆结构的范围排序 |

* #include[<iostream>](http://classfoo.com/ccby/article/Yxj7ge)
* #include[<algorithm>](http://classfoo.com/ccby/article/7ZqseK)
* #include[<vector>](http://classfoo.com/ccby/article/KCu6dM)
* #include[<functional>](http://classfoo.com/ccby/article/Qz6HcH)
* namespace ClassFoo{
* using namespace std;
* void PrintIntVector(vector<int>& foo) {
* vector <int>::iterator Iter1;
* for ( Iter1 = foo.begin( ) ; Iter1 != foo.end( ) ; Iter1++ )
* cout << \*Iter1 << " ";
* cout << endl;
* }
* void HeadOperation1(){
* int a[] = {1, 12, 15, 20, 30};
* vector<int> foo(a, a + sizeof(a) / sizeof(a[0]));
* PrintIntVector(foo);
* make\_heap(foo.begin(), foo.end(), greater<int>());
* PrintIntVector(foo);
* pop\_heap(foo.begin(), foo.end(), greater<int>());
* foo.pop\_back();
* PrintIntVector(foo);
* foo.push\_back(99);
* push\_heap(foo.begin(), foo.end(), greater<int>());
* PrintIntVector(foo);
* sort\_heap(foo.begin(), foo.end(), greater<int>());
* PrintIntVector(foo);
* }
* }
* int main(int argc, char\* argv[])
* {
* ClassFoo::HeadOperation1();
* return 0;
* }

## [最大/最小操作](http://classfoo.com/ccby/article/tZTzs#sec_93QVtE)

由头文件 [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) 定义。

|  |  |
| --- | --- |
| [is\_permutation](http://classfoo.com/ccby/article/Q1FweY) **C++11** | 判断一个序列是否是另一个序列的一种排序 |
| [lexicographical\_compare](http://classfoo.com/ccby/article/FgvffZ) | 比较两个序列的字典序 |
| [max](http://classfoo.com/ccby/article/4BQyfU) | 返回两个元素中值最大的元素 |
| [max\_element](http://classfoo.com/ccby/article/PkmpfX) | 返回给定范围中值最大的元素 |
| [min](http://classfoo.com/ccby/article/mZXRfT) | 返回两个元素中值最小的元素 |
| [min\_element](http://classfoo.com/ccby/article/LvaCfW) | 返回给定范围中值最小的元素 |
| [minmax](http://classfoo.com/ccby/article/tQvwfV)**C++11** | 返回两个元素中值最大及最小的元素 |
| [minmax\_element](http://classfoo.com/ccby/article/brr6fY) **C++11** | 返回给定范围中值最大及最小的元素 |
| [next\_permutation](http://classfoo.com/ccby/article/GH8Wf0) | 返回给定范围中的元素组成的下一个按字典序的排列 |
| [prev\_permutation](http://classfoo.com/ccby/article/AIyAf1) | 返回给定范围中的元素组成的上一个按字典序的排列 |

* #include [<iostream>](http://classfoo.com/ccby/article/Yxj7ge)
* #include [<vector>](http://classfoo.com/ccby/article/KCu6dM)
* #include [<algorithm>](http://classfoo.com/ccby/article/7ZqseK)
* #define CLASSFOO\_VECTOR(type, name, ...) \
* static const type name##\_a[] = \_\_VA\_ARGS\_\_; \
* std::vector<type> name(name##\_a, name##\_a + sizeof(name##\_a) / sizeof(\*name##\_a))
* namespace ClassFoo{
* void PrintIntVector(std::vector<int>& foo) {
* std::vector <int>::iterator Iter1;
* for ( Iter1 = foo.begin( ) ; Iter1 != foo.end( ) ; Iter1++ )
* std::cout << \*Iter1 << " ";
* std::cout << std::endl;
* }
* void NextPermutation() {
* CLASSFOO\_VECTOR(int,foo,{2,3,7});
* std::cout << "排列开始前：";
* PrintIntVector(foo);
* while ( std::next\_permutation(foo.begin(),foo.end())) {
* PrintIntVector(foo);
* }
* std::cout << "排列结束后：";
* PrintIntVector(foo);
* }
* }
* int main() {
* ClassFoo::NextPermutation();
* }

## [数值操作](http://classfoo.com/ccby/article/tZTzs#sec_6wcStF)

由头文件 [<numeric>](http://classfoo.com/ccby/article/Md8qf8) 定义。

|  |  |
| --- | --- |
| [accumulate](http://classfoo.com/ccby/article/w82HKL) | 累加范围中的元素 |
| [adjacent\_difference](http://classfoo.com/ccby/article/UT6FKN) | 计算算输出相邻元素的差值 |
| [inner\_product](http://classfoo.com/ccby/article/C9K5KM) | 计算两个范围的内积 |
| [iota](http://classfoo.com/ccby/article/Tv1lKK) **C++11** | 用顺序递增的值赋值指定范围内的元素 |
| [partial\_sum](http://classfoo.com/ccby/article/l6CZKO) | 部份求和 |

1. #include [<iostream>](http://classfoo.com/ccby/article/Yxj7ge) // std::cout
2. #include [<algorithm>](http://classfoo.com/ccby/article/7ZqseK) // std::generate
3. #include [<vector>](http://classfoo.com/ccby/article/KCu6dM) // std::vector
4. #include [<ctime>](http://classfoo.com/ccby/article/u3IrcL) // std::time
5. #include [<cstdlib>](http://classfoo.com/ccby/article/XTBfcd) // std::rand, std::srand
7. namespace ClassFoo{
8. // 生成器函数:
9. int RandomNumber () { return (std::rand()%100); }
11. // 生成器对象:
12. struct c\_unique {
13. int current;
14. c\_unique() {current=0;}
15. int operator()() {return ++current;}
16. } UniqueNumber;
18. void PrintIntVector(std::vector<int>& foo) {
19. std::vector <int>::iterator Iter1;
20. for ( Iter1 = foo.begin( ) ; Iter1 != foo.end( ) ; Iter1++ )
21. std::cout << \*Iter1 << " ";
22. std::cout << std::endl;
23. }
25. void GenerateVector() {
26. std::srand ( unsigned ( std::time(0) ) );
28. std::vector<int> foo (8);
30. // 用随机数赋值序列
31. std::generate (foo.begin(), foo.end(), RandomNumber);
32. PrintIntVector(foo);
33. // 用唯一值赋值序列
34. std::generate (foo.begin(), foo.end(), UniqueNumber);
35. PrintIntVector(foo);
36. }
37. }
38. int main () {
39. ClassFoo::GenerateVector();
40. return 0;
41. }