C++ 算法库 <algorithm>

C++ 标准库中的 <algorithm> 头文件提供了一组用于操作容器（如数组、向量、列表等）的算法。这些算法包括排序、搜索、复制、比较等，它们是编写高效、可重用代码的重要工具。

<algorithm> 头文件定义了一组模板函数，这些函数可以应用于任何类型的容器，只要容器支持迭代器。这些算法通常接受两个或更多的迭代器作为参数，表示操作的起始和结束位置。

语法

大多数 <algorithm> 中的函数都遵循以下基本语法：

```
algorithm_name(container.begin(), container.end(), ...);
```

这里的 container 是一个容器对象，begin() 和 end() 是容器的成员函数，返回指向容器开始和结束的迭代器。

实例

1. 排序算法

函数：sort

定义：对容器中的元素进行排序。

语法：

```
sort(container.begin(), container.end(), compare_function);
```

其中 compare_function 是一个可选的比较函数，用于自定义排序方式。

实例

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

int main() {
    std::vector<int> numbers = {5, 2, 9, 1, 5, 6};
    std::sort(numbers.begin(), numbers.end());

    for (int num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

输出结果：

```
1 2 5 5 6 9
```

std::partial_sort：对部分区间排序，前 n 个元素为有序。

```
std::partial_sort(vec.begin(), vec.begin() + 3, vec.end());
```

std::stable_sort：稳定排序，保留相等元素的相对顺序。

```
std::stable_sort(vec.begin(), vec.end());
```

2. 搜索算法

函数：find

定义：在容器中查找与给定值匹配的第一个元素。

语法：

```
auto it = find(container.begin(), container.end(), value);
```

如果找到，it 将指向匹配的元素；如果没有找到，it 将等于 container.end()。

实例

```cpp
#include <algorithm>
#include <vector>
#include <iostream>

int main() {
    std::vector<int> numbers = {1, 2, 3, 4, 5};
    auto it = std::find(numbers.begin(), numbers.end(), 3);

    if (it != numbers.end()) {
        std::cout << "Found: " << *it << std::endl;
    } else {
        std::cout << "Value not found." << std::endl;
    }

    return 0;
}
```

输出结果：

```
Found: 3
```

std::binary_search：对有序区间进行二分查找。

```
std::sort(vec.begin(), vec.end());  // 先排序
bool found = std::binary_search(vec.begin(), vec.end(), 4);
```

std::find_if：查找第一个满足特定条件的元素。

```
auto it = std::find_if(vec.begin(), vec.end(), [](int x) { return x > 3; });
```

3. 复制算法

函数：copy

定义：将一个范围内的元素复制到另一个容器或数组。

语法：

```
copy(source_begin, source_end, destination_begin);
```

实例：

实例

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

int main() {
    std::vector<int> source = {1, 2, 3, 4, 5};
    int destination[5];
    std::copy(source.begin(), source.end(), destination);

    for (int i = 0; i < 5; ++i) {
        std::cout << destination[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

输出结果：

```
1 2 3 4 5
```

4. 比较算法

函数：equal

定义：比较两个容器或两个范围内的元素是否相等。

语法：

```
bool result = equal(first1, last1, first2);
```

或

```
bool result = equal(first1, last1, first2, compare_function);
```

实例

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

int main() {
    std::vector<int> v1 = {1, 2, 3, 4, 5};
    std::vector<int> v2 = {1, 2, 3, 4, 5};

    bool are_equal = std::equal(v1.begin(), v1.end(), v2.begin());
    std::cout << (are_equal ? "Vectors are equal." : "Vectors are not equal.") << st
d::endl;

    return 0;
}
```

输出结果：

```
Vectors are equal.
```

5. 修改算法

std::reverse：反转区间内的元素顺序。

```
std::reverse(vec.begin(), vec.end());
```

std::fill：将指定区间内的所有元素赋值为某个值。

```
std::fill(vec.begin(), vec.end(), 0);  // 所有元素设为 0
```

std::replace：将区间内的某个值替换为另一个值。

```
std::replace(vec.begin(), vec.end(), 1, 99);  // 将所有 1 替换为 99
```

std::copy：将区间内的元素复制到另一个区间。

```
std::vector<int> vec2(6);
std::copy(vec.begin(), vec.end(), vec2.begin());
```

6. 排列算法

std::next_permutation：生成字典序的下一个排列，如果没有下一个排列则返回 false。

```
std::vector<int> vec = {1, 2, 3};
do {
    for (int n : vec) std::cout << n << " ";
```

```
    std::cout << std::endl;
} while (std::next_permutation(vec.begin(), vec.end()));
```

std::prev_permutation：生成字典序的上一个排列。

```
std::prev_permutation(vec.begin(), vec.end());
```

7. 归并算法

std::merge：将两个有序区间合并到一个有序区间。

```
std::vector<int> vec1 = {1, 3, 5};
std::vector<int> vec2 = {2, 4, 6};
std::vector<int> result(6);
std::merge(vec1.begin(), vec1.end(), vec2.begin(), vec2.end(), result.begin());
```

std::inplace_merge：在单个区间中合并两个有序子区间。

```
std::inplace_merge(vec.begin(), middle, vec.end());
```

8. 集合算法

std::set_union：计算两个有序集合的并集。

```
std::vector<int> result(10);
auto it = std::set_union(vec1.begin(), vec1.end(), vec2.begin(), vec2.end(),
result.begin());
result.resize(it - result.begin());
```

std::set_intersection：计算两个有序集合的交集。

```
auto it = std::set_intersection(vec1.begin(), vec1.end(), vec2.begin(), vec2.end(),
result.begin());
result.resize(it - result.begin());
```

std::set_difference：计算集合的差集。

```
auto it = std::set_difference(vec1.begin(), vec1.end(), vec2.begin(), vec2.end(),
result.begin());
result.resize(it - result.begin());
```

9. 其他有用算法

std::accumulate （需要 <numeric> 库）： 计算范围内元素的累计和。

```
#include <numeric>
int sum = std::accumulate(vec.begin(), vec.end(), 0);
```

std::for_each：对区间内的每个元素执行操作。

```
std::for_each(vec.begin(), vec.end(), [](int& x) { x += 1; });
```

std::min_element 和 **std::max_element**：查找区间内的最小值和最大值。

```
auto min_it = std::min_element(vec.begin(), vec.end());
auto max_it = std::max_element(vec.begin(), vec.end());
```

<algorithm> 是 C++ 标准库中一个非常强大的工具，它提供了大量通用的算法，可以极大地简化编程