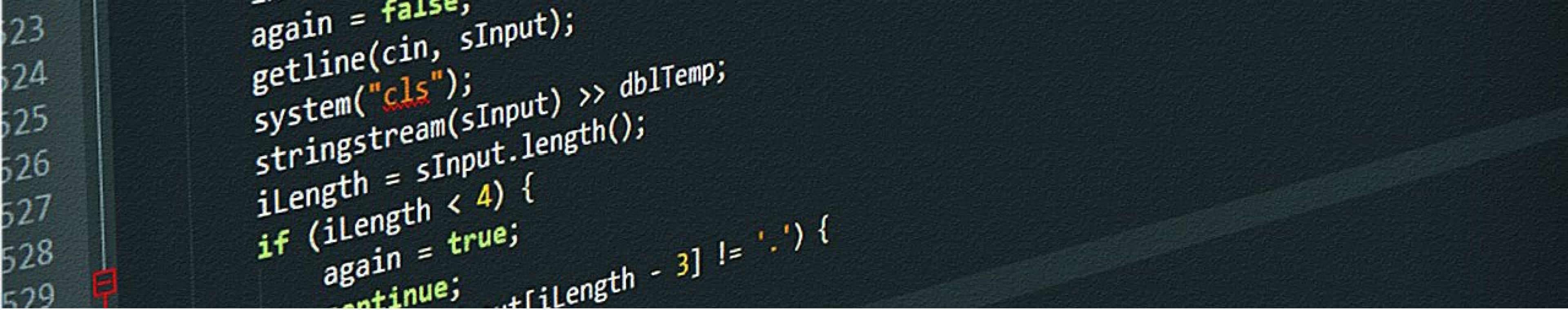


```
again = false;
getline(cin, sInput);
system("cls");
stringstream(sInput) >> dblTemp;
iLength = sInput.length();
if (iLength < 4) {
    again = true;
    ntinue;
          t[iLength - 3] != '.') {
```

Thomas

C23-09.3 Std library algorithms

C23 - Advanced Algorithms and Programming

v5

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Standard library algorithms
General

- Standard library offers more than 80 algorithms
  - Creating and setting values, copying, merging algorithms
  - Finding, sorting, counting algorithms
  - Numerical algorithms

- Algorithms usually work on containers, taking one or more sequences
  - An input sequence is defined by a pair of iterators `[first, last)`
  - An output sequence is defined by an iterator to its first element

- The algorithms report "failures" (e.g., not finding the requested object) usually by returning the end of the input sequence (`last`) [1]

[1] Note that this is the "element" behind the last real element in the container.

Standard library algorithms

find()

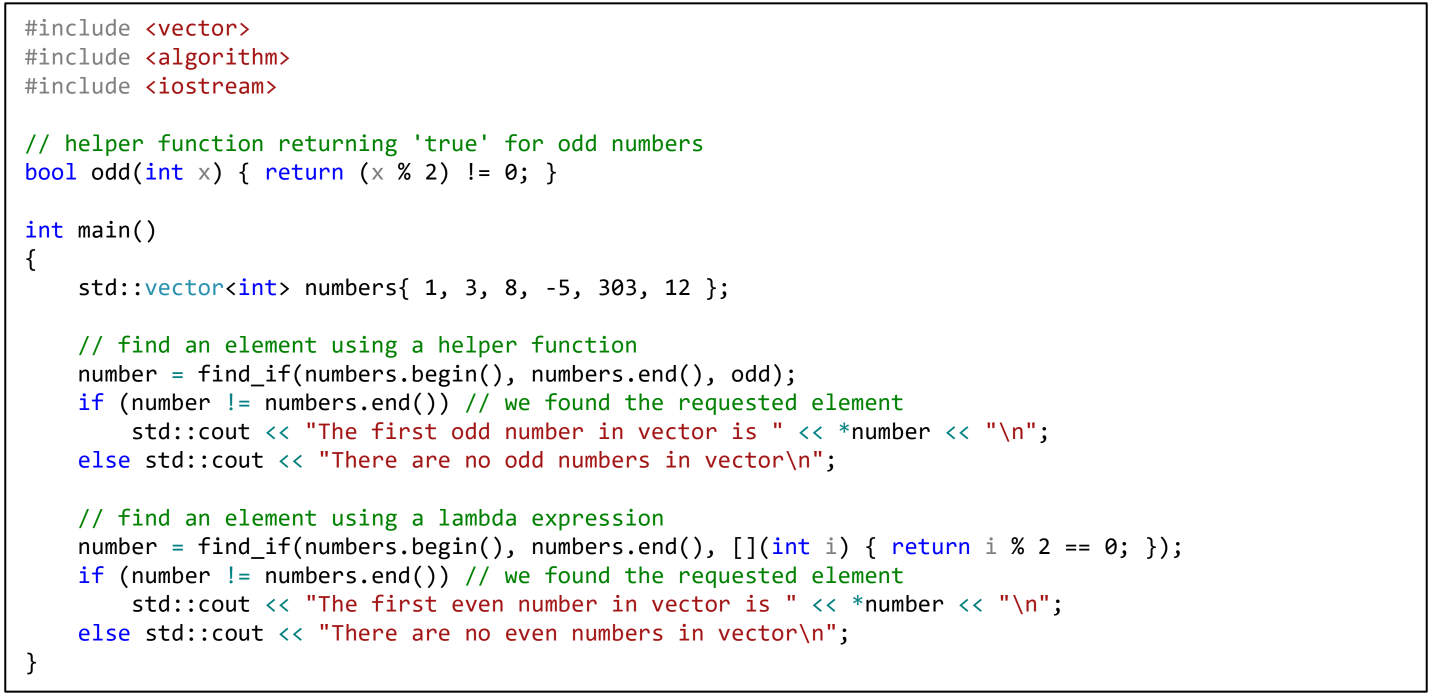
- The `find()` algorithm finds an element with a given value within a sequence:

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

int main()
{
    std::vector<int> numbers{ 1, 3, 8, -5, 303, 12 };

    // find a specific element in vector
    auto number = find(numbers.begin(), numbers.end(), 8); // number is an std::vector<int>::iterator

    if (number != numbers.end()) // we found the requested element
        std::cout << "Vector contains the element " << *number << "\n";
    else
        std::cout << "The requested element does not exist in vector\n";
}
```

Standard library algorithms

find_if()

- The find_if() algorithm finds the first element within a sequence that matches a certain aspect:



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

// helper function returning 'true' for odd numbers
bool odd(int x) { return (x % 2) != 0; }

int main()
{
    std::vector<int> numbers{ 1, 3, 8, -5, 303, 12 };

    // find an element using a helper function
    number = find_if(numbers.begin(), numbers.end(), odd);
    if (number != numbers.end()) // we found the requested element
        std::cout << "The first odd number in vector is " << *number << "\n";
    else std::cout << "There are no odd numbers in vector\n";

    // find an element using a lambda expression
    number = find_if(numbers.begin(), numbers.end(), [](int i) { return i % 2 == 0; });
    if (number != numbers.end()) // we found the requested element
        std::cout << "The first even number in vector is " << *number << "\n";
    else std::cout << "There are no even numbers in vector\n";
}
```

Standard library algorithms

copy(), replace() and fill()

```
#include <algorithm>
#include <iostream>
#include <vector>
#include "print_vector.h“

int main()
{
    constexpr int int_array[]{ 1, 3, 5, 7, 11, 13 };

    // determine the size of int_array and create a vector of the same size
    constexpr size_t array_size = sizeof(int_array) / sizeof(int_array[0]);
    std::vector<int> int_vector(array_size);

    // copy the content of int_array into the vector
    std::copy(int_array, int_array + array_size, int_vector.begin());
    print_vector(int_vector);

    // replace all "5" with "0"
    std::replace(int_vector.begin(), int_vector.end(), 5, 0);
    print_vector(int_vector);

    // now fill the whole vector with "0"
    std::fill(int_vector.begin(), int_vector.end(), 0);
    print_vector(int_vector);
}
```

Standard library algorithms

Specific use of copy()

- Here, copy() print elements into an output stream by copying them to an ostream iterator:

```
#pragma once
#include <vector>

template<typename T>
void print_vector(const std::vector<T>& v)
{
    // print the content of the vector to an ostream by copying it to an ostream_iterator
    // here, the ostream_iterator is associated to cout, and inserts a blank as delimiter after each element
    std::cout << "The vector content is: ";
    std::copy(v.cbegin(), v.cend(), std::ostream_iterator<T>(std::cout, " "));
    std::cout << std::endl;
}
```

Standard library algorithms

sort() (1)

- The sort() algorithm sorts a sequence according to a given sorting criterium:

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

// compare two values with each other and return 'true' if the first element
// shall be placed before before the second one
bool reverseSortHelper(const std::string& first, const std::string& second) { return first > second; }

int main()
{
    std::vector<std::string> companies{ "Lenovo", "Asus", "Samsung", "Apple" };

    // sort uses the standard comparison (element1 < element2)
    sort(companies.begin(), companies.end());
    std::cout << "Companies sorted in standard order: ";
    for (auto c: companies)
        std::cout << c << " ";

    // sort using a custom comparison function
    sort(companies.begin(), companies.end(), reverseSortHelper);
    std::cout << "\nCompanies sorted in reverse order: ";
    for (auto c: companies)
        std::cout << c << " ";
}
```

Standard library algorithms

sort() (2)

- The `sort()` algorithm may also use lambda expressions or standard comparison functions:

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

int main()
{
    std::vector<std::string> companies{ "Lenovo", "Asus", "Samsung", "Apple" };

    // sort using a lambda expression
    sort(companies.begin(), companies.end(), [](const std::string& first, const std::string& second) { return first < second; });
    std::cout << "\nCompanies sorted again in standard order: ";
    for (auto c : companies)
        std::cout << c << " ";

    // sort using a standard library compare function object
    sort(companies.begin(), companies.end(), std::greater<std::string>());
    std::cout << "\nCompanies sorted in reverse order: ";
    for (auto c : companies)
        std::cout << c << " ";
}
```

Standard library algorithms

sort() (3)

- The sort() algorithm may also use function objects:

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

int main()
{
    std::vector<std::string> companies{ "Lenovo", "Asus", "Samsung", "Apple" };

    // sort using a function object
    struct Is_shorter
    {
        bool operator()(const std::string& first, const std::string& second) const { return first.size() < second.size(); }
    };

    std::sort(companies.begin(), companies.end(), Is_shorter());
    std::cout << "\nCompanies sorted according to string length: ";
    for (auto c : companies)
        std::cout << c << " ";

    std::cout << std::endl;
}
```

Standard library algorithms

all_of(), none_of() and any_of()

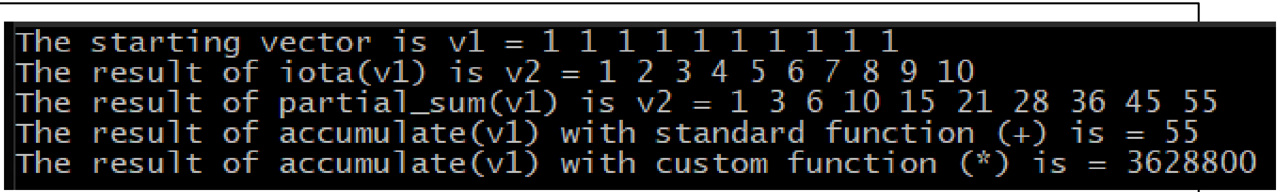
```cpp
#include <vector>
#include <algorithm>
#include <iostream>
#include <functional>

struct Divisible_by {
    Divisible_by(int x) : m_x(x){}
    bool operator()(int x) const { return x % m_x == 0; }
private:
    const int m_x;
};

int main() {
    std::vector<int> numbers{ 1, 9, -11, 45, 707, -3 };
    std::cout << "Among the given numbers...\n";

    // testing for all_of using a generated function with the second argument of modulus bound to '2'
    if (std::all_of(numbers.cbegin(), numbers.cend(), std::bind(std::modulus<int>(), std::placeholders::_1, 2)))
        std::cout << "- All numbers are odd\n";

    // testing for none_off using a lambda expression
    if (std::none_of(numbers.cbegin(), numbers.cend(), [](int i) { return i % 2 == 0; }))
        std::cout << "- None of the numbers is even\n";

    // testing for any_of using a function object
    if (std::any_of(numbers.cbegin(), numbers.cend(), Divisible_by(3)))
        std::cout << "- At least one number is divisible by 3\n";
}
```

Standard library algorithms – numerical algorithms

iota(), partial_sum() and accumulate()

- Standard library also contains many numeric algorithms

```cpp
#include <vector>
#include <numeric>
#include <algorithm>
#include "main.h"

int main()
{
    std::vector<int> v1(10, 1), v2(10);
    print("The starting vector is v1 =", v1);

    std::iota(v1.begin(), v1.end(), 1);
    print("The result of iota(v1) is v2 =", v1);

    std::partial_sum(v1.cbegin(), v1.cend(), v2.begin());
    print("The result of partial_sum(v1) is v2 =", v2);

    // standard accumulate (+)
    std::cout << "The result of accumulate(v1) with standard function (+) is = "
        << std::accumulate(v1.cbegin(), v1.cend(), 0) << "\n";
    // accumulate mit custom lambda expression (*)
    std::cout << "The result of accumulate(v1) with custom function (*) is = "
        << std::accumulate(v1.cbegin(), v1.cend(), 1, [](int a, int b) { return a * b; }) << "\n";
}
```

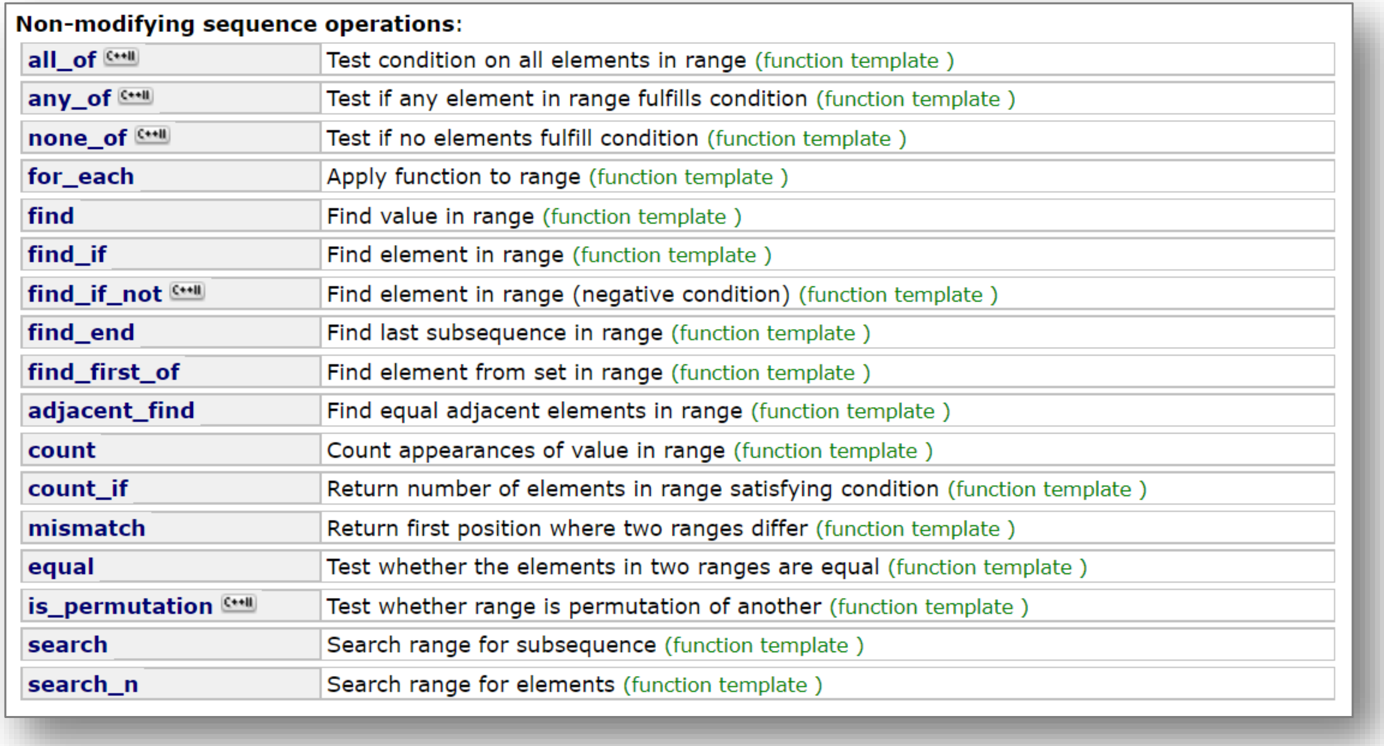


```
The starting vector is v1 = 1 1 1 1 1 1 1 1 1 1
The result of iota(v1) is v2 = 1 2 3 4 5 6 7 8 9 10
The result of partial_sum(v1) is v2 = 1 3 6 10 15 21 28 36 45 55
The result of accumulate(v1) with standard function (+) is = 55
The result of accumulate(v1) with custom function (*) is = 3628800
```

- Most of these algorithms have a second form that lends itself to parallel execution

Standard library algorithms

Non-modifying sequence operations

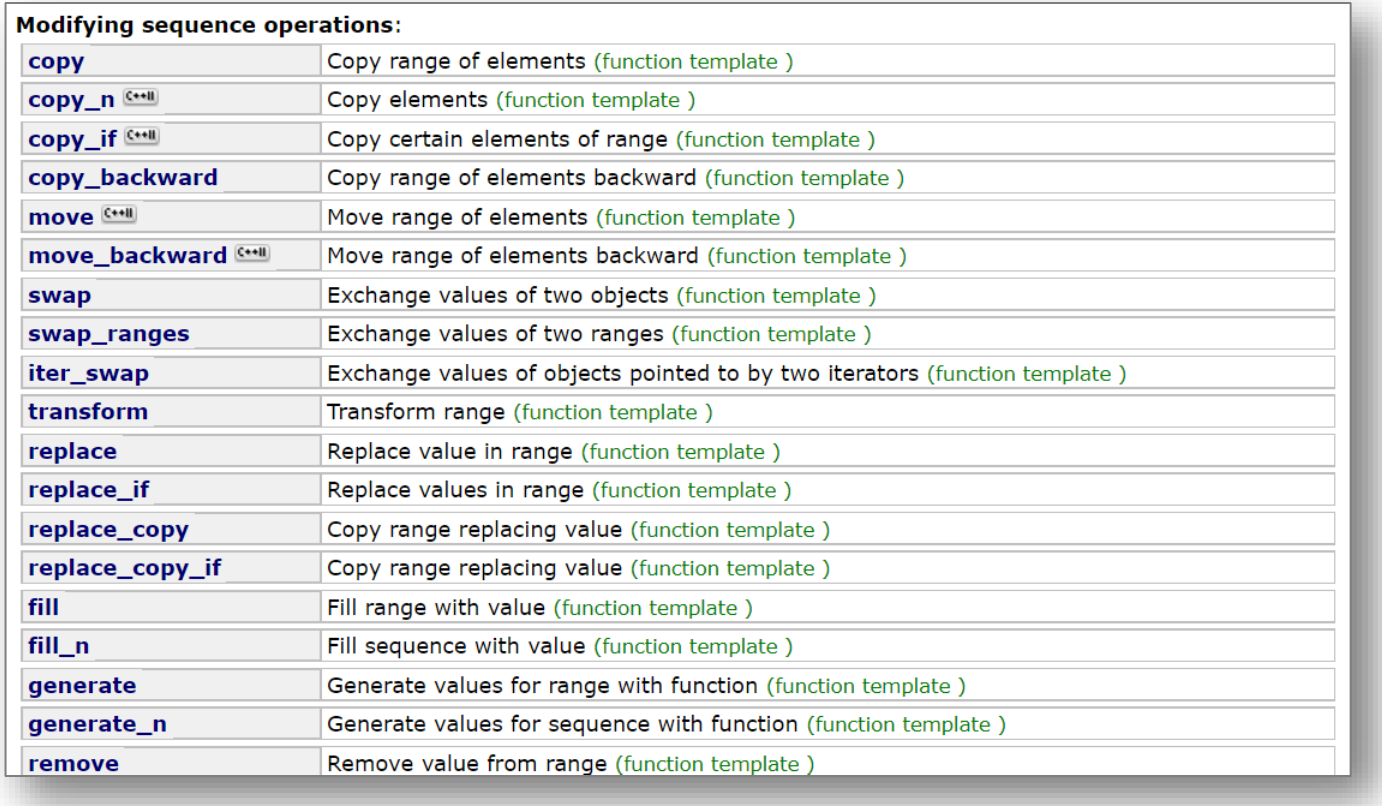


Non-modifying sequence operations:

all_of C++11	Test condition on all elements in range (function template)
any_of C++11	Test if any element in range fulfills condition (function template)
none_of C++11	Test if no elements fulfill condition (function template)
for_each	Apply function to range (function template)
find	Find value in range (function template)
find_if	Find element in range (function template)
find_if_not C++11	Find element in range (negative condition) (function template)
find_end	Find last subsequence in range (function template)
find_first_of	Find element from set in range (function template)
adjacent_find	Find equal adjacent elements in range (function template)
count	Count appearances of value in range (function template)
count_if	Return number of elements in range satisfying condition (function template)
mismatch	Return first position where two ranges differ (function template)
equal	Test whether the elements in two ranges are equal (function template)
is_permutation C++11	Test whether range is permutation of another (function template)
search	Search range for subsequence (function template)
search_n	Search range for elements (function template)

Standard library algorithms

Modifying sequence operations

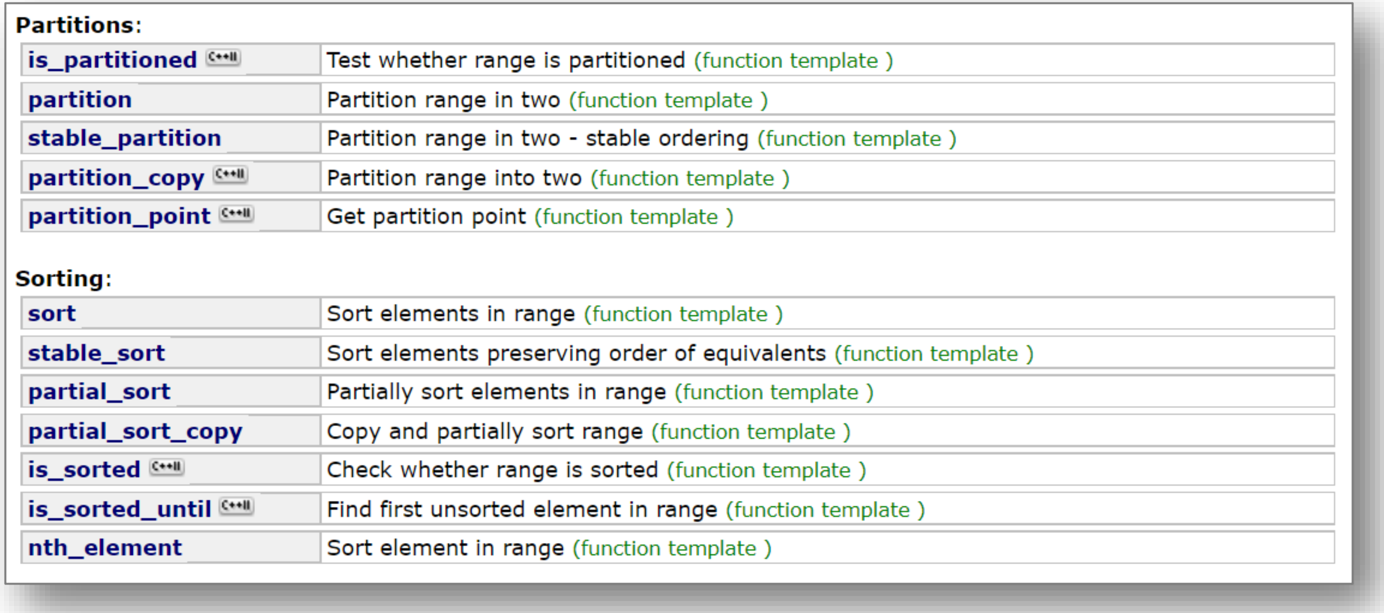


Modifying sequence operations:

copy	Copy range of elements (function template)
copy_n C++11	Copy elements (function template)
copy_if C++11	Copy certain elements of range (function template)
copy_backward	Copy range of elements backward (function template)
move C++11	Move range of elements (function template)
move_backward C++11	Move range of elements backward (function template)
swap	Exchange values of two objects (function template)
swap_ranges	Exchange values of two ranges (function template)
iter_swap	Exchange values of objects pointed to by two iterators (function template)
transform	Transform range (function template)
replace	Replace value in range (function template)
replace_if	Replace values in range (function template)
replace_copy	Copy range replacing value (function template)
replace_copy_if	Copy range replacing value (function template)
fill	Fill range with value (function template)
fill_n	Fill sequence with value (function template)
generate	Generate values for range with function (function template)
generate_n	Generate values for sequence with function (function template)
remove	Remove value from range (function template)

Standard library algorithms

Partition and sorting operations



Partitions:

is_partitioned C++11	Test whether range is partitioned (function template)
partition	Partition range in two (function template)
stable_partition	Partition range in two - stable ordering (function template)
partition_copy C++11	Partition range into two (function template)
partition_point C++11	Get partition point (function template)

Sorting:

sort	Sort elements in range (function template)
stable_sort	Sort elements preserving order of equivalents (function template)
partial_sort	Partially sort elements in range (function template)
partial_sort_copy	Copy and partially sort range (function template)
is_sorted C++11	Check whether range is sorted (function template)
is_sorted_until C++11	Find first unsorted element in range (function template)
nth_element	Sort element in range (function template)

Standard library algorithms

...

... and many more

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

www.htw-berlin.de