Container classification: Algorithms and iterators

```
/*
    . Container classification and container specific iterators:
        .#1: Classification
            . Sequence containers: vector, list, deque, forward_list, array
            . Associative containers: set, multiset, map, multimap
            . Unordered associative containers: unordered_set, unordered_multiset, unordered_map, unordered_multimap
            . Container adaptors: stack, queue, priority_queue

            IMPORTANT NOTE:   Once you see a container, ask your self what kind of iterator it provides.
                              This will help you understand what kind of algorithms you can use with that container.

        .#2: Container specific iterators
            . vector: random access iterator
            . list: bidirectional iterator


        .#3: Container, algorithms and iterator examples.
*/
```

Sequence containers

Container	Iterator Type	Common Algorithms	Requirements of Algorithms
std::vector	Random access iterator	std::sort, std::reverse, std::find	- std::sort: Needs random access iterators for fast access. - std::reverse: Needs bidirectional iterators. - std::find: Works with input iterators.
std::list	Bidirectional iterator	std::remove, std::unique, std::reverse	- std::remove: Requires forward iterators. - std::unique: Needs forward iterators for removing duplicates. - std::reverse: Needs bidirectional iterators.
std::deque	Random access iterator	std::sort, std::reverse, std::rotate	- std::sort: Requires random access iterators. - std::reverse: Needs bidirectional iterators. - std::rotate: Works with forward iterators.
std::forward_list	Forward iterator	std::remove, std::unique, std::merge	- std::remove: Needs forward iterators. - std::unique: Requires forward iterators for duplicates. - std::merge: Needs sorted ranges and forward iterators.
std::array	Random access iterator	std::sort, std::copy, std::reverse	- std::sort: Needs random access iterators. - std::copy: Needs input and output iterators. - std::reverse: Requires bidirectional iterators.

Sequence containers

```
//A function template to print any container
/*
template<typename Container>
void print_container(const Container& container){
    for (const auto& elem : container){
        fmt::print("{} ", elem);
    }
    fmt::print("\n");
}
*/

// We clearly communicate that the function requires an input range: that is,
// a range that can be iterated over in a forward direction.
export template<std::ranges::input_range Container>
void print_container(const Container& container) {
    for (const auto& elem : container) {
        fmt::print("{} ", elem);
    }
    fmt::print("\n");
}
```

Sequence containers

```
//A function template to print any container
/*
template<typename Container>
void print_container(const Container& container){
    for (const auto& elem : container){
        fmt::print("{} ", elem);
    }
    fmt::print("\n");
}
*/

// We clearly communicate that the function requires an input range: that is,
// a range that can be iterated over in a forward direction.
export template<std::ranges::input_range Container>
void print_container(const Container& container) {
    for (const auto& elem : container) {
        fmt::print("{} ", elem);
    }
    fmt::print("\n");
}
```

Sequence containers: Vector and algorithms

```cpp
fmt::print("Vector operations\n");

 std::vector<int> vec = {5, 3, 8, 1, 2};
 print_container(vec);

 // Sort the vector: from smallest to largest by default
 std::sort(std::begin(vec), std::end(vec));
 print_container(vec);

 // Reverse the vector
 std::reverse(std::begin(vec), std::end(vec));
 print_container(vec);

 // Find an element
 auto it = std::find(std::begin(vec), std::end(vec), 8);

 if (it != std::end(vec)) {
     fmt::println("Found: {}", *it);
 } else {
     fmt::println("Not Found");
 }

 //Sort the vector in descending order
 std::sort(std::begin(vec), std::end(vec), std::greater<int>());
 print_container(vec);

 //Sort the vector in ascending order, through a lambda as a comparator
 std::sort(std::begin(vec), std::end(vec), [](int a, int b){return a < b;});
 print_container(vec);
```

Sequence containers: List and algorithms

```cpp
//Here we use member algorithms.
fmt::print("List operations\n");

std::list<int> lst = {1,1, 2, 3, 4, 3, 5, 2};
print_container(lst);

// Remove elements with value 3
//lst.remove(3);
print_container(lst);

// Remove consecutive duplicates
lst.unique();
print_container(lst);

// Reverse the list
lst.reverse();
print_container(lst);
```

Sequence containers: Deque and algorithms

```cpp
std::deque<int> deq = {5, 3, 8, 1, 2};
print_container(deq);

// Sort the deque: from smallest to largest by default
std::sort(std::begin(deq), std::end(deq));
print_container(deq);

// Reverse the deque
std::reverse(std::begin(deq), std::end(deq));
print_container(deq);

// Rotate the deque
std::rotate(std::begin(deq), std::begin(deq) + 1, std::end(deq));
print_container(deq);
```

Sequence containers: Forward_list and algorithms

```cpp
std::forward_list<int> flst = {5, 3, 8, 1, 2};
print_container(flst);

// Remove elements with value 3
flst.remove(3);
print_container(flst);

// Remove consecutive duplicates
flst.unique();
print_container(flst);

// Merge two sorted lists
std::forward_list<int> flst2 = {4, 6, 7};
flst.merge(flst2);
print_container(flst);
```

Associative containers

Container	Iterator Type	Common Algorithms	Requirements of Algorithms
std::set	Bidirectional iterator	std::find, std::set_union, std::accumulate	- std::find: Works with input iterators, bidirectional suffices. - std::set_union: Needs input iterators. - std::accumulate: Needs input iterators.
std::map	Bidirectional iterator	std::for_each, std::transform, std::copy_if	- std::for_each: Works with input iterators, bidirectional works. - std::transform: Works with input and output iterators. - std::copy_if: Works with input and output iterators.
std::multiset	Bidirectional iterator	std::count, std::partial_sum, std::remove	- std::count: Works with input iterators. - std::partial_sum: Needs input iterators. - std::remove: Needs forward iterators, bidirectional suffices.
std::multimap	Bidirectional iterator	std::equal_range, std::find_if, std::distance	- std::equal_range: Works with forward iterators. - std::find_if: Works with input iterators. - std::distance: Works with input iterators.

Associative containers: Set

```cpp
std::set<int> my_set = {1, 3, 5, 7, 9};

// Find an element
auto it = my_set.find(5);
if (it != my_set.end()) {
    fmt::print("Found: {}\n", *it);
}

// Perform set union with another set
std::set<int> other_set = {2, 4, 6, 8, 10};
std::set<int> result_set;
std::set_union(my_set.begin(), my_set.end(),
               other_set.begin(), other_set.end(),
               std::inserter(result_set, result_set.begin()));

fmt::print("Set union: ");
for (int n : result_set) {
    fmt::print("{} ", n);
}
fmt::print("\n");

// Calculate the sum of all elements
int sum = std::accumulate(my_set.begin(), my_set.end(), 0);
// There is no ranges version of accumulate as of the time of this writting:
// Reference: https://stackoverflow.com/questions/63933163/why-didnt-accumulate-make-it-into-ranges-for-c20
fmt::print("Sum of elements: {}\n", sum);
```