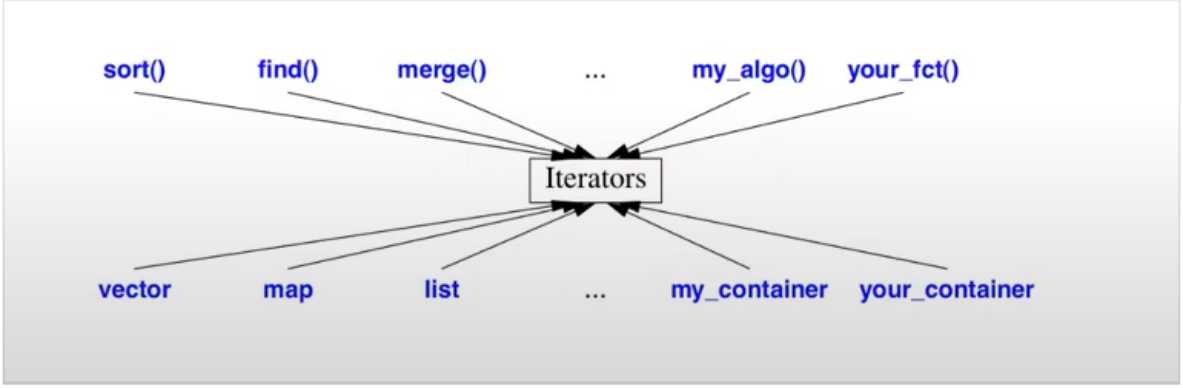
Algorithms

https://en.cppreference.com/w/cpp/algorithm



- About 80 standard algorithms
- Defined in #include <algorithm>
- They operate on sequences defined by a pair of iterators (for inputs) or a single iterator (for outputs).

std::sort

```
int main() {
  array<int, 10> s = {5, 7, 4, 2, 8, 6, 1, 9, 0, 3};

  cout << "Before sorting: ";
  Print(s);

  std::sort(s.begin(), s.end());
  cout << "After  sorting: ";
  Print(s);

  return 0;
}
```

std::find

```cpp
int main() {
  const int n1 = 3;
  std::vector<int> v{0, 1, 2, 3, 4};

  auto result1 = std::find(v.begin(), v.end(), n1);

  if (result1 != std::end(v)) {
    cout << "v contains: " << n1 << endl;
  } else {
    cout << "v does not contain: " << n1 << endl;
  }
}
```

std::fill

```cpp
int main() {
  std::vector<int> v{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

  std::fill(v.begin(), v.end(), -1);

  Print(v);
}
```

std::count

```cpp
int main() {
  std::vector<int> v{1, 2, 3, 4, 4, 3, 7, 8, 9, 10};

  const int n1 = 3;
  const int n2 = 5;
  int num_items1 = std::count(v.begin(), v.end(), n1);
  int num_items2 = std::count(v.begin(), v.end(), n2);
  cout << n1 << " count: " << num_items1 << endl;
  cout << n2 << " count: " << num_items2 << endl;

  return 0;
}
```

std::count_if

```cpp
inline bool div_by_3(int i) { return i % 3 == 0; }

int main() {
  std::vector<int> v{1, 2, 3, 3, 4, 3, 7, 8, 9, 10};

  int n3 = std::count_if(v.begin(), v.end(), div_by_3);
  cout << "# divisible by 3: " << n3 << endl;
}
```

std::for_each

```cpp
int main() {
  std::vector<int> nums{3, 4, 2, 8, 15, 267};

  // lambda expression, lecture_9
  auto print = [](const int& n) { cout << " " << n; };

  cout << "Numbers:";
  std::for_each(nums.cbegin(), nums.cend(), print);
  cout << endl;

  return 0;
}
```

std::all_of

```cpp
inline bool even(int i) { return i % 2 == 0; };
int main() {
  std::vector<int> v(10, 2);
  std::partial_sum(v.cbegin(), v.cend(), v.begin());
  Print(v);

  bool all_even = all_of(v.cbegin(), v.cend(), even);
  if (all_even) {
    cout << "All numbers are even" << endl;
  }
}
```

Output:

```
Among the numbers: 2 4 6 8 10 12 14 16 18 20
All numbers are even
```

std::rotate

```
int main() {
  std::vector<int> v{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  cout << "before rotate: ";
  Print(v);

  std::rotate(v.begin(), v.begin() + 2, v.end());
  cout << "after  rotate: ";
  Print(v);
}
```

Output:

```
before rotate: 1 2 3 4 5 6 7 8 9 10
after  rotate: 3 4 5 6 7 8 9 10 1 2
```

std::transform

```
auto UpperCase(char c) { return std::toupper(c); }
int main() {
  const std::string s("hello");
  std::string S{s};
  std::transform(s.begin(),
                 s.end(),
                 S.begin(),
                 UpperCase);

  cout << s << endl;
  cout << S << endl;
}
```

Output:

```
hello
HELLO
```

std::accumulate

```cpp
int main() {
  std::vector<int> v{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

  int sum = std::accumulate(v.begin(), v.end(), 0);

  int product = std::accumulate(v.begin(),
                                v.end(),
                                1,
                                std::multiplies());

  cout << "Sum    : " << sum << endl;
  cout << "Product: " << product << endl;
}
```

Output:

```
Sum    : 55
Product: 3628800
```

std::max

```cpp
int main() {
  using std::max;
  cout << "max(1, 9999) : " << max(1, 9999) << endl;
  cout << "max('a', 'b'): " << max('a', 'b') << endl;
}
```

Output:

```
max(1, 9999) : 9999
max('a', 'b'): b
```

std:: min_element

```cpp
int main() {
  std::vector<int> v{3, 1, 4, 1, 0, 5, 9};

  auto result = std::min_element(v.begin(), v.end());
  auto min_location = std::distance(v.begin(), result);
  cout << "min at: " << min_location << endl;
}
```

Output:

```
min at: 4
```

std::minmax_element

```cpp
int main() {
  using std::minmax_element;

  auto v = {3, 9, 1, 4, 2, 5, 9};
  auto [min, max] = minmax_element(begin(v), end(v));

  cout << "min = " << *min << endl;
  cout << "max = " << *max << endl;
}
```

Output:

```
min = 1
max = 9
```

std::clamp

```cpp
int main() {
  // value should be between [kMin,kMax]
  const double kMax = 1.0F;
  const double kMin = 0.0F;

  cout << std::clamp(0.5, kMin, kMax) << endl;
  cout << std::clamp(1.1, kMin, kMax) << endl;
  cout << std::clamp(0.1, kMin, kMax) << endl;
  cout << std::clamp(-2.1, kMin, kMax) << endl;
}
```

Output:

```
0.5
1
0.1
0
```