Components

In this lesson, we'll examine the three basic elements which make C++ functional.

WE'LL COVER THE FOLLOWING

The three components of the STL are:

The Standard Template Library (STL) consists of three components from a bird's-eye view. Those are containers, algorithms that run on the containers, and iterators that connect both of them. This abstraction of generic programming enables you to combine algorithms and containers uniquely. The containers have only minimal requirements for their elements.

Containers

The C++ Standard Library has a rich collection of containers. From a bird's eye, we have sequential and associative containers. Associative containers can be classified as ordered or unordered associate containers.

Sequential Containers

Each of the sequential containers has a unique domain, but in 95% of the use cases `std::vector` is the right choice. `std::vector` can dynamically adjust its size, automatically manages its memory and provides you with outstanding performance. In contrast, `std::array` is the only sequential container that cannot adjust its size at runtime. It is optimized for minimal memory and performance overhead. While `std::vector` is good at putting new elements at its end, you should use `std::deque` to put an element also at the beginning. With `std::list` being a doubly-linked list and `std::forward_list` as a singly linked list, we have two additional containers that are optimized for operations at arbitrary positions in the container, with high performance.

Associative Containers

Associative containers are containers of key-value pairs. They provide their values by their respective key. A typical use case for an associative container is a phone book, where you use the key *family name* to retrieve the value *phone number*. C++ has eight different associative containers. On one side there are the associative containers with ordered keys: `std::set`, `std::map`, `std::multiset` and `std::multimap`. On the other side there are the unordered associative containers: `std::unordered_set`, `std::unordered_map`, `std::unordered_multiset` and `std::unordered_multimap`.

Ordered and Unordered Associative Containers

Let's look first at the ordered associative containers. The difference between `std::set` and `std::map` is that the former has no associated value. The difference between `std::map` and `std::multimap` is, that the latter can have more than one identical key. This naming conventions also holds for the unordered associative containers, which have a lot in common with the ordered ones. The key difference is performance. While the ordered associative containers have an access time depending logarithmically on the number of elements, the unordered associative containers allow constant access time. Therefore the access time of the unordered associative containers is independent of their size. The same rule holds true for `std::map` as it does for `std::vector`. In 95 % of all use cases `std::map` should be your first choice for an associative container because the keys are sorted.

Container adapters

Container adapters provide a simplified interface to the sequential containers. C++ has `std::stack`, `std::queue` and `std::priority_queue`.

Iterators

Iterators act as glue between the containers and the algorithms. The container creates them. As generalized pointers, you can use them to iterate forward and backward or to an arbitrary position in the container. The type of iterator you get depends on the container. If you use an iterator adapter, you can directly access a stream.

Algorithms

The STL gives you more than 100 algorithms. By specifying the execution policy, you can run most of the algorithms sequential, parallel, or parallel and vectorized. Algorithms operate on elements or a range of elements. Two iterators define a range. The first one defines the beginning, the second one, called end iterator, defines the end of the range. It's important to know that the end iterator points to *one element past the end of the range*.

The algorithms can be used in a wide range of applications. You can find elements or count them, find ranges, compare or transform them. There are algorithms to generate, replace or remove elements from a container. Of course, you can sort, permute or partition a container or determine the minimum or maximum element of it. A lot of algorithms can be further customized by *callables* like functions, function objects or lambda-functions. The *callables* provide special criteria for search or elements transformation. They highly increase the power of the algorithm.

In the next lesson, I will give an overview of the C++ Standard Library.