STL: Sequential Containers

In this lesson, we will discuss some sequential containers and focus on std::array. It has unique characteristics among all sequential containers of the Standard Template Library.

WE'LL COVER THE FOLLOWING

The sequential containers have much in common, but each container has a special domain. Before we dive into the details of `std::array`, we provide an overview of all five sequential containers of the `std namespace`.

Criteria	std::array	std::vector	std::deque	std::list	std::forward_list
Size	static	dynamic	dynamic	dynamic	dynamic
Implementation	static array	dynamic array	sequence of arrays	double linked list	single linked list
Access	random access	random access	random access	forward and backward	forward
Optimised for		end O(1)	begin and end O(1)	• Begin and end O(1) • Arbitrary O(1)	• Begin O(1) • Arbitrary O(1)
Memoy allocation		yes	no	no	No
Release of memory		`shrink_to_fit()`	`shrink_to_fit()`	always	Always
Strength	• No memory allocation • Minimal memory requirements	95% solution	insertion and deletion at the beginning and at the end	insertion and deletion at an arbitrary position	• Fast insertion and deletion • Minimal memory requirements
Weakness	no dynamic cemory allocation	insertion and deletion at arbitrary positions O(n)	insertion and deletion at arbitrary positions O(n)	no random access	no random access

STL: std::array

`std::array` combines the best of two worlds. On one hand, `std::array` knows its size and has the efficiency of a C array. On the other hand, `std::array` has

its size and has the efficiency of a C array. On the other hand, `std::array` has the interface of a `std::vector`. This is what an `std::array` looks like:


```
std::array<int, 10> myArr{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- `std::array`:
  - Needs the header `<array>`
  - Is a homogeneous container of fixed length

Initialization of the elements

You must keep the rule of aggregate initialization in mind when you work with these elements:

```
std::array<int, 10> //arr elements are not initialized
std::array<int, 10> arr{} //elements are default initialized
std::array<int, 10> arr{1, 2, 3, 4, 5} // Remaining elements are default i
nitialized
```

Index Access

`std::array` supports three types of index access.

```
- arr[n];
- arr.at(n);
- std::get<n>(arr);
```

The most commonly used types of index are the angle brackets, but it does not check the boundaries of the `arr`. This acts in opposition to `arr.at(n)`. You will eventually get an `std::range-error` exception. The last type shows the relationship of the `std::array` with the `std::tuple` since both are containers of fixed length.

Let's take a look at an example of sequential containers in the next lesson.