Introduction

The first category of containers we'll study are the sequential containers. Listed below are the major features for various types of sequential containers.

> The sequential container have a lot in common, but each container has its special domain. Before I dive into the details, I provide an overview of all five sequential containers of the std namespace.

Criteria	Array	Vector	Deque	List	Forward List
Size	static	dynamic	dynamic	dynamic	dynamic
Implementation	static array	dynamic array	sequence of arrays	doubly linked list	singly linked list
Access	random	random	random	forward and backward	forward
Optimiz	—	end: O(1)	begin	begin	begin(1);

ed for insert and delete at			and end: O(1)	and end: O(1); arbitrary : O(1)	arbitrary : O(1)
Memory reservation	—	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 begin and end	insertion and deletion at an arbitrary position	fast insertion and deletion; minimal memory requirements
Weakness	no dynamic memory; memory allocation	Insertion and deletion; at an arbitrary position:	Insertion and deletion; at an arbitrary position:	no random access	no random access

		O(n)	O(n)		

The sequential containers

A few additional remarks to the table.

O(i) stands for the complexity (runtime) of an operation. So O(1) means that the runtime of an operation on a container is constant and is independent of the size of the container. Conversely, O(n) means that the runtime depends on the number of the elements of the container. Now, what does that mean for an `std::vector`? The access time on an element is independent of the size of the `std::vector`, but the insertion or deletion of an arbitrary element with k-times more elements is k-times slower.

Although the random access on the elements of a `std::vector` has the same complexity O(1) as the random access on the element of a `std::deque`, that doesn't mean, that both operations are equally fast.

The complexity guarantee O(1) for the insertion or deletion into a doubly (`std::list`) or singly linked list (`std::forward_list`) is only guaranteed if the iterator points to the right element.

> **i std::string is like std::vector**
> Of course `std::string` is no container of the standard template library. But from a behavioural point of view, it is like a sequential container, especially like a `std::vector<char>`. Therefore I will treat `std::string` as a `std::vector<char>`.