

Introduction

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

The [sequential containers](#) have a lot in [common](#), but each container has its special domain. Before we dive into the details, we'll look at 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	doubled linked list	single linked list
Access	random	random	random	forward and backward	forward
Optimiz	—	end: O(1)	begin	begin	begin:

	ed for insert and delete at			and end: O(1)	and end: O(1); arbitrary : O(1)	O(1); arbitrary : O(1)
	Memory reservat ion	—	yes	no	no	no
	Release of memory	—	shrink_t o_fit	shrink_t o_fit	always	always
	Strength	no memory allocatio n; minimal memory require ments	95% solution	insertion and deletion at the begin and end	insertion and deletion at an arbitrary position	fast insertion and deletion; minimal memory require ments
	Weakne ss	no dynamic memory; memory allocatio n	Insertion and deletion; at an arbitrary position: O(n)	Insertion and deletion; at an arbitrary position: O(n)	no random access	no random access

		O(1)	O(1)		
--	--	------	------	--	--

The sequential containers

A few additional remarks to bring to the table.

O(1) 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. Opposite to that, O(n) means that the runtime depends linearly on the number of the elements of the container. 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 an `std::vector` has the same complexity, O(1), as the random access on the element of an `std::deque`, that doesn't mean, that both operations are equally fast.

The complexity guarantee O(1) for the insertion or deletion of a double (`std::list`) or single 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 not a container of the standard template library. But from a behavioral point of view, it is like a sequential container, specifically an `std::vector<char>`. Therefore, we will treat `std::string` like an `std::vector<char>`.

In the next lesson, we'll study `std::array` in detail.