【C++】Day30

Class	C++
Date	@January 2, 2022
Material	
Series Number	
Summary	

【Ch9】Sequential Containers

A container holds a collection of objects of a specified type.

The sequential containers let the programmer control the order in which the elements are stored and accessed. That order does not depend on the values of the elements. The order corresponds to the position at which elements are put into the container.

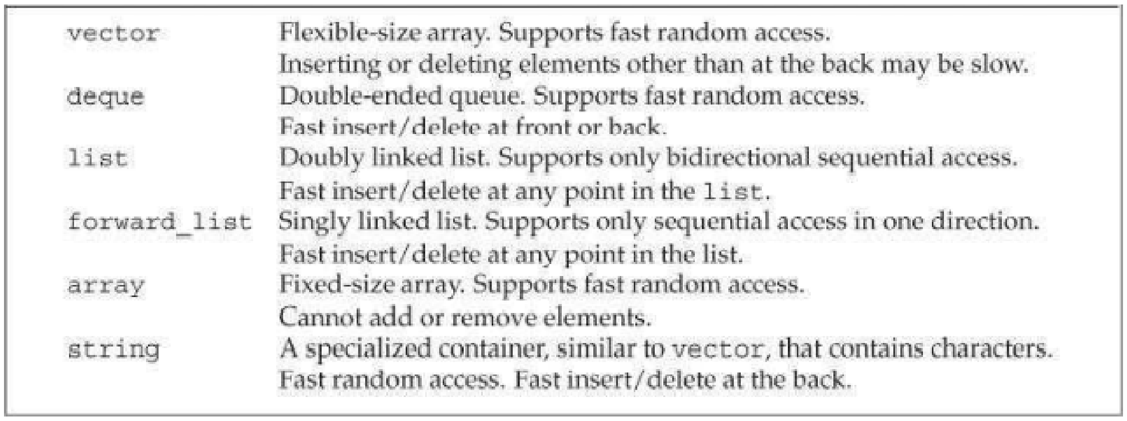
9.1 Overview of the Sequential Containers

The sequential containers, listed below, all provide fast sequential access to their elements.

However, these containers offer different performance trade-offs relative to:

- The costs to add or delete elements to the container
- The costs to perform nonsequential access to elements of the container.

Table 9.1. Sequential Container Types



`vector`	Flexible-size array. Supports fast random access. Inserting or deleting elements other than at the back may be slow.
`deque`	Double-ended queue. Supports fast random access. Fast insert/delete at front or back.
`list`	Doubly linked list. Supports only bidirectional sequential access. Fast insert/delete at any point in the `list`.
`forward_list`	Singly linked list. Supports only sequential access in one direction. Fast insert/delete at any point in the list.
`array`	Fixed-size array. Supports fast random access. Cannot add or remove elements.
`string`	A specialized container, similar to `vector`, that contains characters. Fast random access. Fast insert/delete at the back.

With the exception of array, which is a fixed-size container, the containers provide efficient, flexible memory management.

`string` and `vector` supports fast random access, but adding or deleting an element in the middle might be painful and time consuming.

The `list` and `forward_list` containers are designed to make it fast to add or remove an element anywhere in the container. However, these containers do not support fast random access.

A `deque` supports fast random access. As with string and vector, adding or removing elements in the middle is expensive operation. However, adding or removing elements at either end of the deque is a fast operation.

Note: Modern C++ programs should use the library containers rather than more primitive structures like arrays.

Deciding which Sequential Container to Use

Tip: Ordinarily, it is best to use vector unless theire is a good reason to prefer another container.

There are a few rules of thumb that apply to selecting which container to use:

- Use a `vector` unless you have a reason to use another container
- If the program requires random access to elements, use a `vector` or a `deque`.
- If the program needs to insert or delete elements in the middle of the container, use a `list` or `forward_list`

- If the program needs to insert or delete elements at the front and the back, but not in the middle, use a `deque`
- If the program needs to insert elements in the middle of the container only while reading input, and subsequently needs random access to the elements:
    - First, decide whether you actually need to add elements in the middle of a container. It is often easier to append to a `vector` and then call the library sort function to reorder the container when you're done with input
    - If you must insert into the middle, considering using a `list` for the input phase. Once the input is complete, copy the list into a vector.

Best Practices: If you're not sure which container to use, write your code so that it uses only operations common to both vectors and lists: Use iterators, not subscripts, and avoid random access to elements. That way it will be easy to use either a vector or a list as necessary.