CS Studies - Containers

Initialization

initializer_list: initialize container with initializer_list<T>

There is no need to use pointers in initializing containers since **std::allocator** automatically manages their resources and stores them in **heap**.

Containers

A container is an object used to store other objects responsible for taking care of the management of the memory used by the objects it contains.

Sequence Containers

std::Sequence Containers

array<T, size>

- fixed size array
- #include <array>

<array.h> template<class T, std::size_ N> struct array.

- can be initialized with *aggregate-initialization*[1][2]

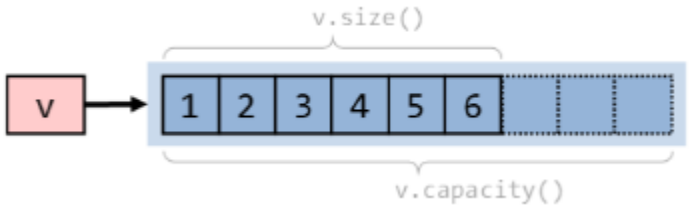
This container is aggregate type which means its composition in initialization applies recursively. Aggregate data type is a type of data that can be referenced as a single entity, and yet consists of more than one piece of data. It combines the performance and accessibility of a C-style array with benefits of a standard container such as size identification, assignment support, and random access iterators.

```
std::array<int, 3> a = {1, 2, 3};
```

[1] aggregate-initialization: Initializes an aggregate from an initializer list. It is a form of list-initialization.
[2] list-initialization: Initializes an object from *braced-init-list*. e.i) T object = {arg1, arg2, …};

- Default constructed array is not empty and the complexity of swapping is linear.
- Partially satisfies the requirements of Sequence Container. If its length is zero, array.begin() equals array.end(). front() or back () on it will be undefined.
- Can be used as a tuple[3] of n elements of same type
- contiguous memory
- randomaccess
- fast linear traversal

vector<T>

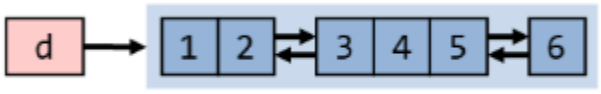


- dynamic array
- C++'s default container
- #include <vector>
- contiguous memory
- random access
- fast linear traversal
- fast insertion/deletion at the ends

Vectors usually occupy more space than static arrays, because more memory is allocated to handle future growth. This way a vector does not need to reallocate each time an element is inserted, but only when the additional memory is exhausted. https://en.cppreference.com/w/cpp/container/vector

- total amount allocated can be accessed through capacity()
- extra memory can be returned to the system via shrink_to_fit()

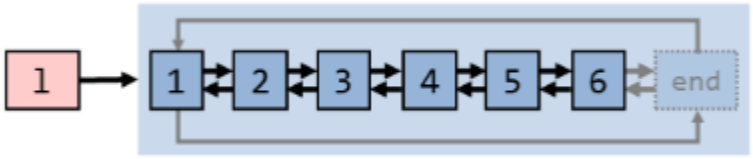
deque<T>



- double-ended queue
- fast insertion/deletion at both ends
- never validates pointers or references to the rest of its elements
- not stored contiguously

[3] ordered set of values. In this context, it means there can be arrays of arrays.

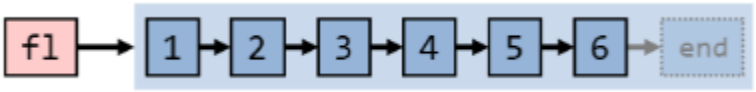
list<T>



- double-linked list
- fast splicing
- operations without copy/move of elements

List allows constant time insert and erase operations anywhere within the sequence and iteration in both directions. They are implemented as doubly-linked lists, meaning they can store each of the elements they contain in different and unrelated storage locations. The ordering is kept internally by the association to each element of a link to the element preceding it and a link to the lament following it.

forward_list<T>



forward_list is similar to list except it is singly linked, meaning they only store link to the next element, where list stores both last and next element (two links)

- lower memory than list
- only forward movements