Vectors

- sequence containers representing arrays that can change in size
- like arrays, vectors use contiguous (consecutive memory blocks) storage locations for their elements
- this means elements can also be accessed using offsets on regular pointers to its elements and just as efficiently as arrays
- unlike arrays, their size can change dynamically with their storage being handled automotically by the container
- vectors use a dynamically allocated array to store their elements
- array may need to be reallocated in order to groze in size when new elements are inserted, which implies allocating a new array and moving all elements to it
- relatively expensive task in terms of processing time
- vectors do not reallocate each time an element is added
- vectors may allocate some extra storage to accommodate for possible growth
- may have an actual capacity greater than the storage strickly needed to contain its elements
- reallocations should only happen at logarithmically growing intervals of size such that the insertion of individual elements at the end of the vertor can be provided with amortized constant time (average time taken per operation, if you do many operations)
- compared to arrays, vectors consume more memory in exchange for the ability to manage storage and grow dynamically in an efficient way
- compared to other dynamic sequence containers (i.e. deques, lists and forward_lists), vectors are very efficient at accessing its elements (like arrays) and relatively efficient adding/removing elements from its end
- for operations which involve inserting/removing elements at positions other than the end, they perform worse than the others (due to contiguous memory usage)
- less consistent iterators and references than lists or forward_lists

Properties

Sequence

- elements in sequence containers are ordered in a strict linear sequence
- individual elements are accessed by their position in this sequence

Dynamic array

- allows direct access to any element in the sequence, even through pointer arithmetics, and provides relatively fast addition/removal of elements at the end of the sequence

Allocator-aware

- container uses an allocator object to dynamically handle its storage needs

Parameters

`n` - Initial container size (i.e. number of elements). The member type `size_type` is an unsigned integral type.

`val` - Value to fill container with; each of the `n` elements in the container will be initialized to a copy of this value. The member type `value_type` is the type of the elements in the container, defined in `vector` as an alias of its first parameter.

`first, last` - Input iterators to the initial and final positions in a range. The range used is `(first, last)`, which includes all the elements between `first` and `last`, including the element pointed by first but not the element pointed by `last`.

`x` - Another vector object of the same type, whose contents are either copied or acquired.

`il` - An `initializer_list` object. These objects are automatically constructed from `initializer list` declarators. The member type `value_type` is the type of the elements in the container, defined in `vector` as an alias of its first parameter.

Examples

1) Empty container constructor (default constructor)
   - Constructs an empty container, i.e. no elements.

     ```
     std::vector<int> first;
     ```

2) Fill constructor
   - Constructs a container with `n` elements. Each element is a copy of `val` (if provided).

     ```
     std::vector<int> second(4, 100);
     ```

3) Initializer list constructor
   - Constructs a container with a copy of each of the elements in the list (same order)

     ```
     std::vector<int> third = {10, 20, 30};
     ```

4) Range constructor
   - Constructs a container with as many elements as the range `(first, last)`, with each element *emplace-constructed* from its corresponding element in that range, in the same order.

```
std::vector<int> fourth (second.begin(), second.end());
```

5) Copy constructor

- Constructs a container with a copy of each of the elements in `x`, in the same order

```
std::vector<int> fifth (third);
```

6) Move constructor

- Constructs a container that aquires the elements of `x`

```
int myints[] = {1, 2, 3, 4, 5};
// specify number of elements by checking array size
std::vector<int> sixth (myints, myints + sizeof(myints) / sizeof(int))
```

Emplace

- Extends a container by inserting a new element at `position`
- New element is constructed in place using `args` as the arguments for its construction
- Parameters:
  - position
    - Position in the container where the new element is inserted
    - Member type `const_iterator` is a random access iterator type which points to a const element
  - args
    - Arguments forwarded to construct the new element
- Return value
  - An iterator which points to the newly emplaced element
  - Member type `iterator` is a random access iterator type that points to an element

Example

```
std::vector<int> v = {10, 20, 30};
auto it v.emplace(v.begin() + 1, 100);
v.emplace(it, 200);
v.emplace(v.end(), 300);
```

```
v contains: 10, 200, 100, 20, 30, 300
```