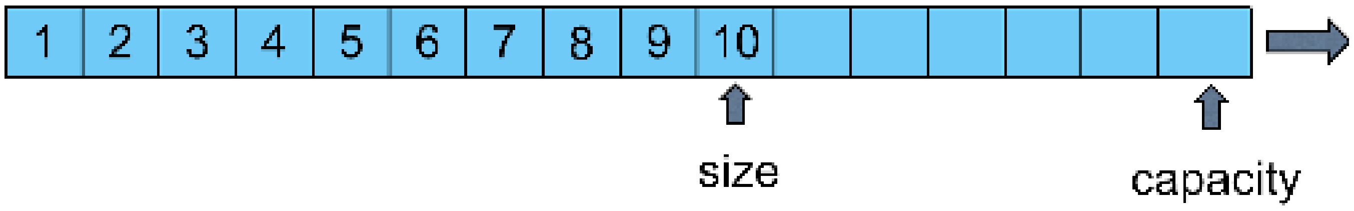
Vectors

Vectors are more refined version of arrays. They simplify the insertion and deletion of values.



std::vector is a homogeneous container, for which it's length can be adjusted at runtime. `std::vector` needs the header `<vector>`. As it stores its elements contiguously in memory, `std::vector` supports pointer arithmetic.

```
for (int i= 0; i < vec.size(); ++i){
  std::cout << vec[i] == *(vec + i) << std::endl; // true
}
```

Make sure to distinguish the round and curly braces in the creation of an `std::vector`

If we construct a `std::vector`, we mustkeep a few things in mind. The constructor with round braces in the following example creates an `std::vector` with a capacity of 10 elements, while the constructor with curly braces creates an `std::vector` with the element 10.

```
std::vector<int> vec(10);
std::vector<int> vec{10};
```

The same rules hold true for the expressions `std::vector<int>(10, 2011)` or `std::vector<int>{10, 2011}`. In the first case, we get an `std::vector` with 10 elements, initialised to 2011. In the second case, we get an `std::vector` with the elements 10 and 2011. The reason for this behaviour is that curly braces are interpreted as initialiser lists so the sequence constructor is used.

Let’s look into an example to better understand the concept:

```cpp
#include <utility>
#include <vector>

int main(){

  std::vector<int> first;
  std::vector<int> second(4, 2011);
  std::vector<int> third(second.begin(), second.end());
  std::vector<int> forth(second);
  std::vector<int> fifth(std::move(second));
  std::vector<int> sixth{1, 2, 3, 4, 5};

}
```

Size vs. Capacity

The number of elements an `std::vector` has usually take up less space than what is already reserved. There is a simple reason for this. With extra memory already allocated, the size of the `std::vector` can increase without an expensive allocation of new memory.

There are a few methods for smartly handling memory:

Method	Description
`vec.size()`	Returns the number of elements of `vec`.

`vec.capacity()`	Returns the number of elements, which `vec` can have without reallocation.
`vec.resize(n)`	`vec` will be increased to `n` elements.
`vec.reserve(n)`	Reserve memory for at least `n` elements.
`vec.shrink_to_fit()`	Reduces `capacity` of `vec` to the `size`.

Memory management of std::vector

The call `vec.shrink_to_fit()` is not binding. That means the runtime can ignore it. But on popular platforms, I always observed the desired behavior.

So let's see the methods in the application.

```
// vector.cpp
#include <iostream>
#include <vector>

int main(){
  std::vector<int> intVec1(5, 2011);
  intVec1.reserve(10);
  std::cout << intVec1.size() << std::endl;     // 5
  std::cout << intVec1.capacity() << std::endl; // 10

  intVec1.shrink_to_fit();
  std::cout << intVec1.capacity() << std::endl; // 5

  std::vector<int> intVec2(10);
  std::cout << intVec2.size() << std::endl;     // 10

  std::vector<int> intVec3{10};
  std::cout << intVec3.size() << std::endl;     // 1

  std::vector<int> intVec4{5, 2011};
  std::cout << intVec4.size() << std::endl;     // 2
  return 0;
}
```

`std::vector vec` has a few methods to access its elements. `vec.front()`, yields the first element, and `vec.back()` yields the last element of `vec`. To read or write the (n+1)-th element of `vec`, we can use the index operator `vec[n]` or the method `vec.at(n)`. The second one checks the boundaries of `vec`, so that we eventually get an `std::range_error` exception.

Besides the index operator, `std::vector` offers additional methods to assign, insert, create or remove elements. See the following overview.

Method	Description
`vec.assign( ... )`	Assigns one or more elements, a range or an initializer list.
`vec.clear()`	Removes all elements from `vec`.
`vec.emplace(pos, args ... )`	Creates a new element before `pos` with the `args` in `vec` and returns the new position of the element.
`vec.emplace_back(args ... )`	Creates a new element in `vec` with `args ...`.
`vec.erase( ... )`	Removes one element or a range and returns the next position.
`vec.insert(pos, ... )`	Inserts one or more elements, a range or an initializer list and returns the new position of the element.
`vec.pop_back()`	Removes the last element.
`vec.push_back(elem)`	Adds a copy of `elem` at the end of

`vec.push_back(elem)`	`vec`.

Modify the elements of a std::vector

Further information

- std::vector

To build upon our understanding of this concept, let’s solve an exercise in the next lesson.