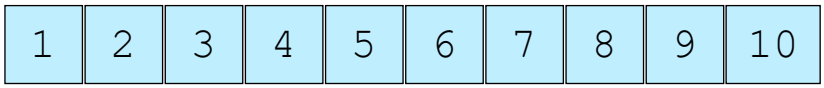
Arrays

The array type is perhaps the most popular sequential container. This lesson will cover its properties in detail.

This is what an array looks like:



std::array is a homogeneous container of fixed length. It requires the header `<array>`. An instance of `std::array` combines the memory and runtime characteristic of a C array with the interface of an `std::vector`.In particular, an `std::array` knows its size. We can use STL algorithms on instances of `std::array`.

Keep a few special rules in mind for initializing an `std::array`.

- `std::array<int, 10> arr`: The 10 elements are not initialized.
- `std::array<int, 10> arr{}`: The 10 elements initialized to 0 by default.
- `std::array<int, 10> arr{1, 2, 3, 4, 5}`: The unspecified elements are initialized to 0 by default.

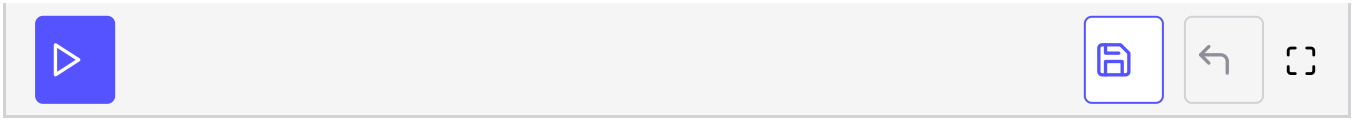
`std::array` supports three types of index access.

```
- arr[n];
- arr.at(n);
- std::get<n>(arr);
```

The most commonly used first type of index access using angle brackets does not check the boundaries of the `arr`. This is in contrast to `arr.at(n)`. We will eventually get an `std::range-error` exception. The last form in the above snippet shows the relationship of `std::array` with the std::tuple, because both are containers of fixed length.

Here is a little bit of arithmetic using `std::array`:

```
// array.cpp
#include <iostream>
#include <array>
#include <numeric>

using namespace std;

int main(){
  std::array<int, 10> arr{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  for (auto a: arr) std::cout << a << " " ;    // 1 2 3 4 5 6 7 8 9 10
  cout << "\n";

  double sum= accumulate(arr.begin(), arr.end(), 0);
  std::cout << sum << std::endl;               // 55

  double mean= sum / arr.size();
  std::cout << mean << std::endl;              // 5.5
  std::cout << (arr[0] == std::get<0>(arr));   // 1 (1 represents true)

  return 0;
}
```



std::array

To get a stronger grip on this topic, let's solve an example in the next lesson.