

# Introduction

In this part of the course, we will learn to implement iterators and use them to traverse our data.

On one hand, [iterators](#) are generalizations of pointers that represent positions in a container. On the other hand, they provide powerful iteration and random access in a container.

Iterators are the glue between the generic containers and the generic algorithms of the Standard Template Library.

Iterators support the following operations:

**\***: Returns the element at the current position.

**==, !=**: Compares two positions.

**=**: Assigns a new value to an iterator.

The range-based for-loop uses the iterators implicitly.

Because iterators are not checked, they have the same issues as pointers.

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

int main(){
    std::vector<int> vec{1, 23, 3, 3, 3, 4, 5};
    std::deque<int> deq;

    // Start iterator bigger than end iterator
    std::copy(vec.begin()+2, vec.begin(), deq.begin());

    // Target container too small
    std::copy(vec.begin(), vec.end(), deq.end());
    return 0;
}
```



In the next lesson, we'll talk about three primary categories of iterators.