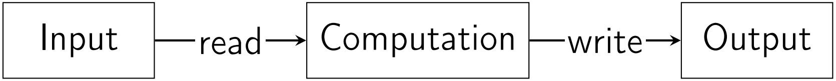
The Standard Template Library II



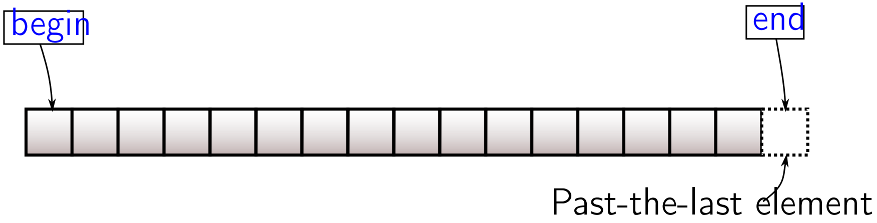
- Dealing with data as sequence of elements
- Containers, **Iterators**, **Algorithms**

Iterators

- An object that behaves as a pointer

```
std::vector<int> v = {1, 2, 3, 4, 5};
std::vector<int>::iterator it = v.begin();
std::cout << *it << std::endl; // 1
```

- All containers provide iterators



Iterators

- An object that behaves as a pointer

```
std::vector<int> v = {1, 2, 3, 4, 5};
std::vector<int>::iterator it = v.begin();
std::cout << *it << std::endl; // 1
```

- All containers provide iterators
- Secret: sometimes

```
std::is_same_v<std::vector<int>::iterator, int*> == true
```

Iterators

- An object that behaves as a pointer

```cpp
std::vector v = {1, 2, 3, 4, 5};
auto it = v.begin();
std::cout << *it << std::endl; // 1
++it;
std::cout << *it << std::endl; // 2
--it;
std::cout << *it << std::endl; // 1
```

- All containers provide iterators

Iterators

- An object that behaves as a pointer

```
std::vector v = {1, 2, 3, 4, 5};
for(auto it = v.begin(); it != v.end(); ++it){
  std::cout << *it << std::endl;
}
// Equivalent to:
for(auto &x: v){
  std::cout << x << std::endl;
}
```

- All containers provide iterators

Algorithms

- 99% of the time:

```
std::algorithm_name(start_iterator,
                    end_iterator,
                    other_parameters);
```

Algorithms

```cpp
std::vector v = {5, 4, 3, 2, 1};
int sum = std::accumulate(v.begin(),
                          v.end(),
                          0);
```

Algorithms

```
std::vector v = {5, 4, 3, 2, 1};
// Sort ascending
std::sort(v.begin(), v.end());
// Sort descending
std::sort(v.rbegin(), v.rend());
```

Algorithms

```
std::vector v = {5, 4, 3, 2, 1};
auto isEven = [](auto i){return i%2==0;}
auto neven = std::count_if(v.begin(),
                           v.end(),
                           isEven);
```

Algorithms

```cpp
#include <execution>
#include <algorithm>
#include <tbb/parallel_sort.h>
int main(){
  std::vector v = {5, 4, 3, 2, 1};
  // Parallel sort
  std::sort(std::execution::par_unseq,
   ↪  input.begin(), input.end());
  // Another version
  tbb::parallel_sort(input.rbegin(),
   ↪  input.rend());
}
```