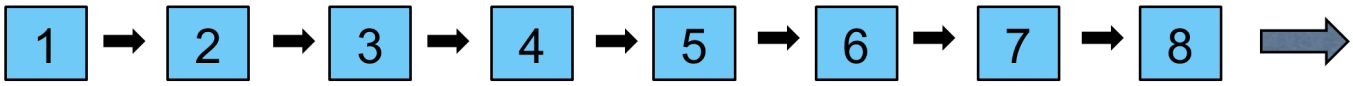
Forward Lists

A forward list is the primitive form of the list structure we studied in the previous lesson. Nevertheless, forward lists are still useful.



std::forward_list is a singly linked list, which needs the header `<forward_list>`. `std::forward_list` has a drastically reduced interface and is optimized for minimal memory requirements.

`std::forward_list` has a lot in common with `std::list`:

- It doesn't support the random access.
- The access of an arbitrary element is slow because in the worst case, we have to iterate forward through the whole list.
- To add or remove an element is fast, if the iterator points to the right place.
- If we add or remove an element, the iterator remains valid.
- Operations always refer to the beginning of the `std::forward_list` or the position past the current element.

Being able to iterate through an `std::forward_list` forward has a great impact. The iterators cannot be decremented and therefore, operations like `--` (decrement) on iterators are not supported. For the same reason, `std::forward_list` has no backward iterator. `std::forward_list` is the only sequential container which doesn't know its size.

> 🔑 **std::forward_list has a very special domain**
> `std::forward_list` is the replacement for single linked lists. It's optimized for minimal memory management and performance if the insertion, extraction, or movement of elements only affects adjacent elements. This is typical for sorting algorithms.

elements. This is typical for sorting algorithms.

The following are the special methods of `std::forward_list`:

Method	Description
`forw.before_begin()`	Returns an iterator before the first element.
`forw.emplace_after(pos, args...)`	Creates an element after `pos` with the arguments `args...`.
`forw.emplace_front(args...)`	Creates an element at the beginning of `forw` with the arguments `args...`.
`forw.erase_after(pos, ...)`	Removes from `forw` the element `pos` or a range of elements, starting with `pos`.
`forw.insert_after(pos, ...)`	Inserts new elements after `pos`. These elements can be single elements, ranges or initialiser lists.
`forw.merge(c)`	Merges the sorted forward list c into the sorted forward list `forw`, so that `forw` keeps sorted.
`forw.merge(c, op)`	Merges the forward sorted list c into the forward sorted list `forw`, so that `forw` keeps sorted. Uses `op` as sorting criteria.
`forw.splice_after(pos, ...)`	Splits the elements in `forw` before `pos`. The elements can be single elements, ranges or lists.

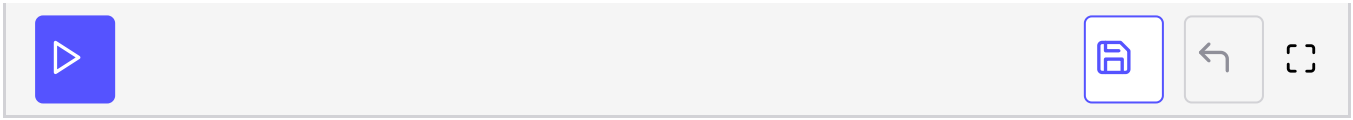
`forw.unique()`	Removes adjacent element with the same value.
`forw.unique(pre)`	Removes adjacent elements, fulfilling the predicate `pre`.

Special methods of std::forward_list

Let's have a look at how the unique methods of `std::forward_list` work.

```
// forwardList.cpp
#include <iostream>
#include <algorithm>
#include <forward_list>

using std::cout;

int main(){
  std::forward_list<int> forw;
  std::cout << forw.empty() << std::endl; // 1 (1 denoted true)

  forw.push_front(7);
  forw.push_front(6);
  forw.push_front(5);
  forw.push_front(4);
  forw.push_front(3);
  forw.push_front(2);
  forw.push_front(1);
  for (auto i: forw) cout << i << " "; // 1 2 3 4 5 6 7
  cout<<"\n";

  forw.erase_after(forw.before_begin());
  cout<< forw.front(); // 2
  cout<<"\n";

  std::forward_list<int> forw2;
  forw2.insert_after(forw2.before_begin(), 1);
  forw2.insert_after(++forw2.before_begin(), 2);
  forw2.insert_after(++(++(forw2.before_begin())), 3);
  forw2.push_front(1000);
  for (auto i= forw2.cbegin();i != forw2.cend(); ++i) cout << *i << " ";  // 1000 1 2 3
  cout<<"\n";

  auto IteratorTo5= std::find(forw.begin(), forw.end(), 5);
  forw.splice_after(IteratorTo5, std::move(forw2));
  for (auto i= forw.cbegin(); i != forw.cend(); ++i) cout << *i << " ";  // 2 3 4 5 1000 1 2
  cout<<"\n";

  forw.sort();
  for (auto i= forw.cbegin(); i != forw.cend(); ++i) cout << *i << " ";
    // 1 2 2 3 3 4 5 6 7 1000
  cout<<"\n";
```

```
  forw.reverse();
  for (auto i= forw.cbegin(); i != forw.cend(); ++i) cout << *i << " ";
    // 1000 7 6 5 4 3 3 2 2 1
  cout<<"\n";

  forw.unique();
  for (auto i= forw.cbegin(); i != forw.cend(); ++i) cout << *i << " ";
    // 1000 7 6 5 4 3 2 1
  cout<<"\n";

  return 0;
}
```



std::forward_list

To build upon our understanding of this topic, let’s answer a few questions in the next lesson.