class template

std::list

<list>

```
template < class T, class Alloc = allocator<T> > class list;
```

List

Lists are sequence containers that allow constant time insert and erase operations anywhere within the sequence, and iteration in both directions.

List containers are implemented as doubly-linked lists; Doubly linked lists can store each of the elements they contain in different and unrelated storage locations. The ordering is kept internally by the association to each element of a link to the element preceding it and a link to the element following it.

They are very similar to forward_list: The main difference being that forward_list objects are single-linked lists, and thus they can only be iterated forwards, in exchange for being somewhat smaller and more efficient.

Compared to other base standard sequence containers (array, vector and deque), lists perform generally better in inserting, extracting and moving elements in any position within the container for which an iterator has already been obtained, and therefore also in algorithms that make intensive use of these, like sorting algorithms.

The main drawback of `list`s and forward_lists compared to these other sequence containers is that they lack direct access to the elements by their position; For example, to access the sixth element in a `list`, one has to iterate from a known position (like the beginning or the end) to that position, which takes linear time in the distance between these. They also consume some extra memory to keep the linking information associated to each element (which may be an important factor for large lists of small-sized elements).

Container properties

Sequence
Elements in sequence containers are ordered in a strict linear sequence. Individual elements are accessed by their position in this sequence.

Doubly-linked list
Each element keeps information on how to locate the next and the previous elements, allowing constant time insert and erase operations before or after a specific element (even of entire ranges), but no direct random access.

Allocator-aware
The container uses an allocator object to dynamically handle its storage needs.

Template parameters

`T`
Type of the elements.
Aliased as member type `list::value_type`.

`Alloc`
Type of the allocator object used to define the storage allocation model. By default, the allocator class template is used, which defines the simplest memory allocation model and is value-independent.
Aliased as member type `list::allocator_type`.

Member types

C++98 C++11

member type	definition	notes
value_type	The first template parameter (T)	
allocator_type	The second template parameter (Alloc)	defaults to: allocator<value_type>
reference	allocator_type::reference	for the default allocator: value_type&
const_reference	allocator_type::const_reference	for the default allocator: const value_type&
pointer	allocator_type::pointer	for the default allocator: value_type*
const_pointer	allocator_type::const_pointer	for the default allocator: const value_type*
iterator	a bidirectional iterator to value_type	convertible to const_iterator
const_iterator	a bidirectional iterator to const value_type	
reverse_iterator	reverse_iterator<iterator>	
const_reverse_iterator	reverse_iterator<const_iterator>	
difference_type	a signed integral type, identical to: iterator_traits<iterator>::difference_type	usually the same as ptrdiff_t
size_type	an unsigned integral type that can represent any non-negative value of difference_type	usually the same as size_t

Member functions

(constructor)	Construct list (public member function)
(destructor)	List destructor (public member function)
operator=	Assign content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

empty	Test whether container is empty (public member function)
size	Return size (public member function)
max_size	Return maximum size (public member function)

Element access:

front	Access first element (public member function)
back	Access last element (public member function)

Modifiers:

assign	Assign new content to container (public member function)
emplace_front C++11	Construct and insert element at beginning (public member function)
push_front	Insert element at beginning (public member function)
pop_front	Delete first element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
emplace C++11	Construct and insert element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
resize	Change size (public member function)
clear	Clear content (public member function)

Operations:

splice	Transfer elements from list to list (public member function)
remove	Remove elements with specific value (public member function)
remove_if	Remove elements fulfilling condition (public member function template)
unique	Remove duplicate values (public member function)
merge	Merge sorted lists (public member function)
sort	Sort elements in container (public member function)
reverse	Reverse the order of elements (public member function)

Observers:

get_allocator	Get allocator (public member function)

Non-member function overloads

relational operators (list)	Relational operators for list (function)
swap (list)	Exchanges the contents of two lists (function template)