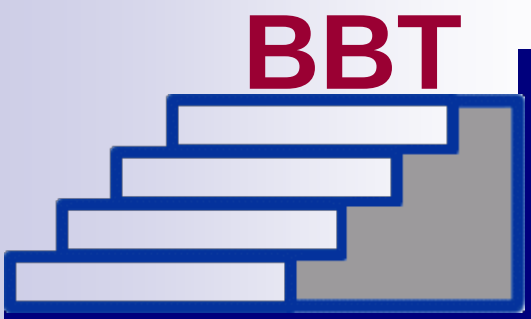

CPP

STL

Iterator

- Trivial Iterator
- Input Iterator
- Output Iterator
- Forward Iterator
- Bidirectional Iterator
- Random Access Iterator

Iterator instances –cont.

- input_iterator_tag
- output_iterator_tag
- forward_iterator_tag
- bidirectional_iterator_tag
- random_access_iterator_tag
- input_iterator
- output_iterator
- forward_iterator
- bidirectional_iterator
- random_access_iterator

Iterator instances

- istream_iterator
- ostream_iterator
- reverse_iterator
- reverse_bidirectional_iterator
- insert_iterator
- front_insert_iterator
- back_insert_iterator
- iterator_traits

Iterator_traits

```
template <class Iterator>
 struct iterator_traits {
 typedef typename Iterator::iterator_category iterator_category;
 typedef typename Iterator::value_type value_type;
 typedef typename Iterator::difference_type difference_type;
 typedef typename Iterator::pointer pointer;
 typedef typename Iterator::reference reference;
};
 template <class T>
 struct iterator_traits<T*> {
 typedef random_access_iterator_tag iterator_category;
 typedef T value_type; typedef ptrdiff_t difference_type;
 typedef T* pointer;
 typedef T& reference;
};
```

Example:

```
template <class InputIterator>
    iterator_traits<InputIterator>::value_type
    last_value(InputIterator first, InputIterator last)
 {
    iterator_traits<InputIterator>::value_type result = *first;
    for (++first; first != last; ++first) result = *first;
    return result;
}
```

Container

- General concepts
- Container
- Forward Container
- Reversible Container
- Random Access Container

Sequence

- Sequence
- Front Insertion Sequence
- Back Insertion Sequence

Associative Containers

- Associative Container
- Simple Associative Container
- Pair Associative Container
- Sorted Associative Container
- Unique Associative Container
- Multiple Associative Container
- Unique Sorted Associative Container
- Multiple Sorted Associative Container

Hash Container

- Hashed Associative Container
- HashFunction
- Unique Hashed Associative Container
- Multiple Hashed Associative Container

Associated types of Container

Value type	X::value_type	The type of the object stored in a container. The Value type must be Assignable, but need not be Default Constructible.
Iterator Type	X::iterator	The type of iterator used to iterate through a container's elements. The iterator's value type is expected to be the container's value type. A conversion from the iterator type to the const iterator type must exist. The iterator type must be an input iterator.
Const iterator type	X:: const_iterator	A type of iterator that may be used to examine, but not to modify, a container's elements.
Reference type	X::reference	A type that behaves as a reference to the container's value type.

Associated types –cont.

Const reference type	X:: const_reference	A type that behaves as a const reference to the container's value type.
Pointer type	X::pointer	A type that behaves as a pointer to the container's value type.
Distance type	X:: difference_type	A signed integral type used to represent the distance between two of the container's iterators. This type must be the same as the iterator's distance type.
Size type	X::size_type	An unsigned integral type that can represent Any nonnegative value of the container's distance type.

Container methods

Beginning of range	a.begin()	iterator if a is mutable, const_iterator otherwise
End of range	a.end()	iterator if a is mutable, const_iterator otherwise
Size	a.size()	size_type
Maximum size	a.max_size()	size_type
Empty container	a.empty()	Convertible to bool
Swap	a.swap(b)	void

Forward Container

Name	Expression	Return type
Equality	a == b	Convertible to bool
Inequality	a != b	Convertible to bool
Less	a < b	Convertible to bool
Greater	a > b	Convertible to bool
Less or equal	a <= b	Convertible to bool
Greater or equal	a >= b	Convertible to bool

Reversible Container

Reverse Iterator type	X:: reverse_iterator	A Reverse Iterator adaptor whose base iterator type is the container's iterator type. Incrementing an object of type reverse_iterator moves backwards through the container: the Reverse Iterator adaptor maps operator++ to operator--.
Const reverse iterator type	X::const_reverse _iterator	A reverse Iterator adaptor whose base iterator type is the container's const iterator type.

Reversible Container –cont.

Name	Expression	Return type
Beginning of range	a.rbegin()	reverse_iterator if a is mutable, const_reverse_iterator otherwise
End of range	a.rend()	reverse_iterator if a is mutable, const_reverse_iterator otherwise

Sequence

- A Sequence is a variable-sized **Container** whose elements are arranged in a strict linear order. It supports insertion and removal of elements.

Sequence methods

Name	Expression	Type requirements	Return type
Fill constructor	X(n, t)		X
Fill constructor	X a(n, t);		
Default fill Constructor	X(n)	T is DefaultConstructible.	X
Default fill Constructor	X a(n);	T is DefaultConstructible.	
Range Constructor	X(i, j)	i and j are Input Iterators whose value type is convertible to T	X
Range Constructor	X a(i, j);	i and j are Input Iterators whose value type is convertible to T	
Front	a.front()		reference if a is mutable, const_reference otherwise.

Sequence methods –cont.

Insert	a.insert(p, t)		X:: iterator
Fill insert	a.insert(p, n, t)	a is mutable	void
Range insert	a.insert(p, i, j)	i and j are Input Iterators whose value type is convertible to T . a is mutable	void
Erase	a.erase(p)	a is mutable	iterator
Range erase	a.erase(p,q)	a is mutable	iterator
Clear	a.clear()	a is mutable	void
Resize	a.resize(n, t)	a is mutable	void
Resize	a.resize(n)	a is mutable	void

Back Insertion Sequence

Name	Expression	Type requirements	Return type
Back	a.back()		reference if a is mutable, otherwise const_reference.
Push back	a.push_back(t)	a is mutable.	void
Pop back	a.pop_back()	a is mutable.	void

Associative Container

An Associative Container is a variable-sized Container that supports efficient retrieval of elements (values) based on keys. It supports insertion and removal of elements, but differs from a Sequence in that it does not provide a mechanism for inserting an element at a specific position

New type

Key type	X::key_type	The type of the key associated with X::value_type. Note that the key type and value type might be the same.

Associative Container methods

Name	Expression	Return type
Default Constructor	X() X a	
Erase key	a.erase(k)	size_type
Erase element	a.erase(p)	void
Erase range	a.erase(p, q)	void
Clear	a.clear()	void
Find	a.find(k)	iterator if a is mutable, otherwise const_iterator
Count	a.count(k)	size_type
Equal range	a.equal_range(k)	pair<iterator, iterator> if a is mutable, otherwise pair<const_iterator, const_iterator>.

Pair Associative Container

- A Pair Associative Container is an Associative Container that associates a key with some other object. The value type of a Pair Associative Container is pair<const key_type, data_type>.
- Refine: associative Container, reversible Container

Sorted Associative Container

Key type	X::key_type	The type of the key associated with X::value_type.
Data type	X::data_type	The type of the data associated with X::value_type. A Pair Associative Container can be thought of as a mapping from key_type to data_type.
Value type	X::value_type	The type of object stored in the container. The value type is required to be pair<const key_type, data_type>.

Types –cont.

X::key_compare	The type of a Strict Weak Ordering used to compare keys. Its argument type must be X::key_type.
X::value_compare	The type of a Strict Weak Ordering used to compare values. Its argument type must be X::value_type, and it compares two objects of value_type by passing the keys associated with those objects to a function object of type key_compare.

Sorted container methods

Name	Expression	Return type
Default Constructor	X() X a;	
Constructor with compare	X(c) X a(c)	
Key comparison	a.key_comp()	X::key_compare
Value comparison	a::value_compare()	X::value_compare
Lower bound	a.lower_bound(k)	iterator if a is mutable, otherwise const_iterator.
Upper bound	a.upper_bound(k)	iterator if a is mutable, otherwise const_iterator.
Equal range	a.equal_range(k)	pair<iterator, iterator> if a is mutable, otherwise pair<const_iterator, const_iterator>.

Unique Associative Container

- A Unique Associative Container is an Associative Container with the property that each key in the container is unique: no two elements in a Unique Associative Container have the same key.

Unique container Methods

Name	Expression	Type requirements	Return type
Range constructor	X(i, j) X a(i, j);	i and j are Input Iterators whose value type is convertible to T	
Insert element	a.insert(t)		pair<X::iterator, bool>
Insert range	a.insert(i, j)	i and j are Input Iterators whose value type is convertible to X::value_type.	void
Count	a.count(k)		size_type

Multiple Associative Container

- A Multiple Associative Container is an Associative Container in which there may be more than one element with the same key. That is, it is an Associative Container that does not have the restrictions of a Unique Associative Container.

methods

Name	Expression	Type requirements	Return type
Range constructor	X(i, j) X a(i, j);	i and j are Input Iterators whose value type is convertible to T	
Insert element	a.insert(t)		X::iterator
Insert range	a.insert(i, j)	i and j are Input Iterators whose value type is convertible to X::value_type.	void

Unique Sorted Associative Container

Name	Expression	Type requirements	Return type
Range Constructor	X(i, j) X a(i, j);	i and j are Input Iterators whose value type is convertible to T .	
Range Constructor with compare	X(i, j, c) X a(i,j,c)	i and j are Input Iterators whose value type is convertible to T . C is an object of type key_compare.	
Insert with hint	a.insert(p, t)		iterator
Insert range	a.insert(i, j)	i and j are Input Iterators whose value type is convertible to X::value_type.	void

Container instances

model	fc	revc	ranc	sc	fsc	bsc
1. vector	y	y	y	y		y
2. deque	y	y	y	y	y	y
3. list	y	y		y	y	y
4. slist	y			y		y

List functions

void splice(iterator position, list<T, Alloc>& x);
void splice(iterator position,list<T, Alloc>& x, iterator i);
void splice(iterator position,list<T, Alloc>& x,iterator f, iterator l);
void remove(const T& val);
template<class Predicate> void remove_if(Predicate p);
void unique();
template<class BinaryPredicate> void unique(BinaryPredicate p);
void merge(list<T, Alloc>& x);
template<class BinaryPredicate> void merge(list<T, Alloc>& x,BinaryPredicate Comp);
void reverse();
void sort();
template<class BinaryPredicate>void sort(BinaryPredicate comp);

slist functions

iterator previous(iterator pos)
const_iterator previous(const_iterator pos)
iterator insert_after(iterator pos)
iterator insert_after(iterator pos, const value_type& x);
template<class InputIterator>void insert_after(iterator pos, InputIterator f, InputIterator l)
void insert_after(iterator pos, size_type n, const value_type& x)
iterator erase_after(iterator pos)
iterator erase_after(iterator before_first, iterator last)
void splice(iterator position, slist<T, Alloc>& x);
void splice(iterator position, slist<T, Alloc>& x, iterator i);
void splice(iterator position, slist<T, Alloc>& x, iterator f, iterator l);
void remove(const T& val);

slist functions –cont.

void splice_after(iterator pos, iterator prev)
void splice_after(iterator pos,iterator before_first, iterator before_last)
template<class Predicate> void remove_if(Predicate p);
void unique();
template<class BinaryPredicate> void unique(BinaryPredicate p);
void merge(slist<T, Alloc>& x);
template<class BinaryPredicate> void merge(slist<T, Alloc>& x, BinaryPredicate Comp);
void reverse();
void sort();
template<class BinaryPredicate>void sort(BinaryPredicate comp);

Container instances –cont.

model	fc	ac	sac	pac	sortac	uac	mac	usac	msac	h	hu	hm
1. set	y	y	y		y	y		y				
2. map	y	y		y	y	y		y				
3. multiset	y	y	y		y		y		y			
4. multimap	y	y		y	y		y		y			
5. hash_set	y	y	y			y				y	y	
6. hash_map	y	y		y		y				y	y	
7. hash_multiset	y	y	y				y			y		y
8. hash_multimap	y	y		y			y			y		y
9. hash	y	y								y		