STL containers (2)
Associative Containers

2023

국민대학교 소프트웨어학부

STL

- STL: 표준 템플릿 라이브러리(Standard Template Library)
- 많은 프로그래머들이 공통적으로 사용하는 자료 구조와 알고리즘들을 template 으로 구현한 클래스
- namespace std 에 포함되어 있음

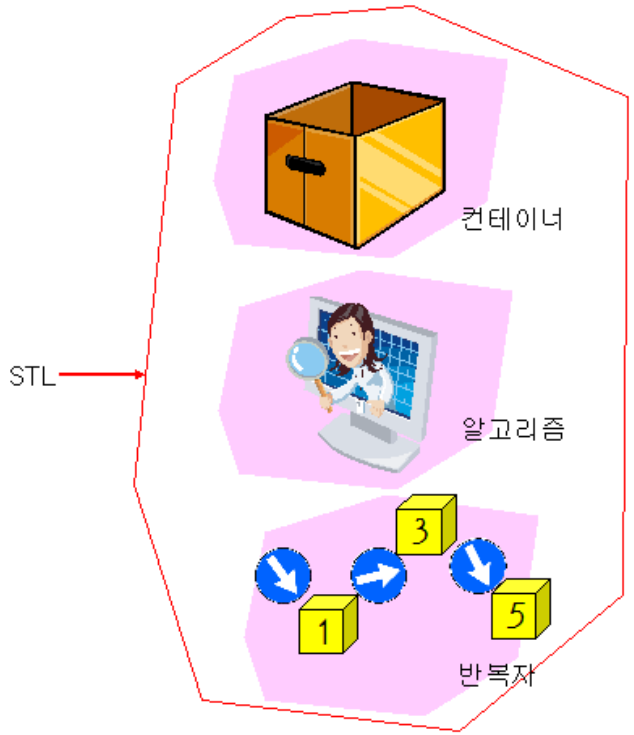
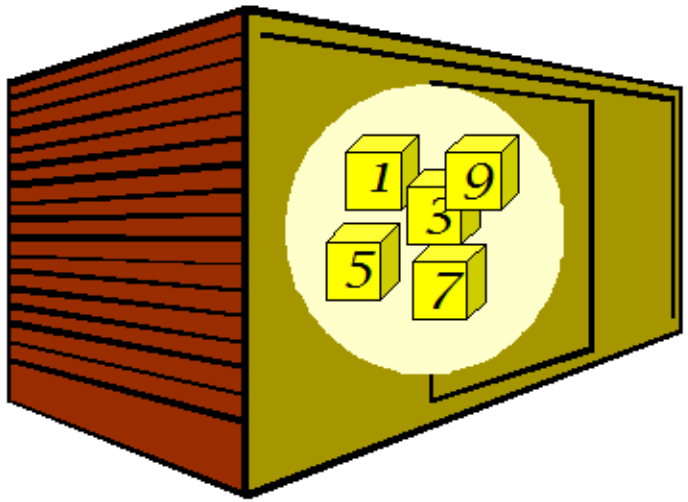


그림 18-1 STL

STL containers



컨테이너

그림 18-2 컨테이너의 개념

분류	컨테이너 클래스	설명	헤더 파일
순차 컨테이너	vector	벡터처럼 입력된 순서대로 저장	<vector>
	list	순서가 있는 리스트	<list>
	deque	양끝에서 입력과 출력이 가능	<deque>
연관 컨테이너	set	수학에서의 집합 구현	<set>
	multiset	다중 집합(중복을 허용)	<set>
	map	사전과 같은 구조	<map>
	multimap	다중 맵(중복을 허용)	<map>
컨테이너 어댑터	stack	스택(후입선출)	<stack>
	queue	큐(선입선출)	<queue>
	priority_queue	우선순위큐(우선순위가 높은 원소가 먼저 출력)	<queue>

연관 컨테이너 Associative container

- associative : 원소들을 접근할 때 순차적 접근이 아니라 key 값에 의해 접근된다. (= 순차 접근을 위한 [] 연산자가 없다.)
- 그러나 원소들은 정렬되어 저장된다. (보통 binary search tree 로 구현된다.)

- 집합(set): 중복이 없는 자료들이 정렬되어서 저장된다.
- 맵(map): 키-값(key-value)의 형식으로 저장된다. 키가 제시되면 해당되는 값을 찾을 수 있다.
- 다중-집합(multiset): 집합과 유사하지만 자료의 중복이 허용된다.
- 다중-맵(multimap): 맵과 유사하지만 키가 중복될 수 있다.

Headers		<set>		<map>	
Members		**set**	**multiset**	**map**	**multimap**
	constructor	set	multiset	map	multimap
	destructor	~set	~multiset	~map	~multimap
	assignment	operator=	operator=	operator=	operator=
iterators	`begin`	begin	begin	begin	begin
	`end`	end	end	end	end
	`rbegin`	rbegin	rbegin	rbegin	rbegin
	`rend`	rend	rend	rend	rend
const iterators	`cbegin`	cbegin	cbegin	cbegin	cbegin
	`cend`	cend	cend	cend	cend
	`crbegin`	crbegin	crbegin	crbegin	crbegin
	`crend`	crend	crend	crend	crend
capacity	`size`	size	size	size	size
	`max_size`	max_size	max_size	max_size	max_size
	`empty`	empty	empty	empty	empty
	`reserve`				
element access	`at`			at	
	`operator[]`			operator[]	
modifiers	`emplace`	emplace	emplace	emplace	emplace
	`emplace_hint`	emplace_hint	emplace_hint	emplace_hint	emplace_hint
	`insert`	insert	insert	insert	insert
	`erase`	erase	erase	erase	erase
	`clear`	clear	clear	clear	clear
	`swap`	swap	swap	swap	swap
operations	`count`	count	count	count	count
	`find`	find	find	find	find
	`equal_range`	equal_range	equal_range	equal_range	equal_range
	`lower_bound`	lower_bound	lower_bound	lower_bound	lower_bound
	`upper_bound`	upper_bound	upper_bound	upper_bound	upper_bound

set / multiset in <set>

- 집합(set)은 동일한 키를 중복해서 가질 수 없다.
- 원소들은 크기 순으로 정렬되어 저장된다.
- (예) A = { 1, 2, 3, 4, 5 }는 집합이지만 B = { 1, 1, 2, 2, 3 }은 집합이 아니다.
- iterators are bidirectional.

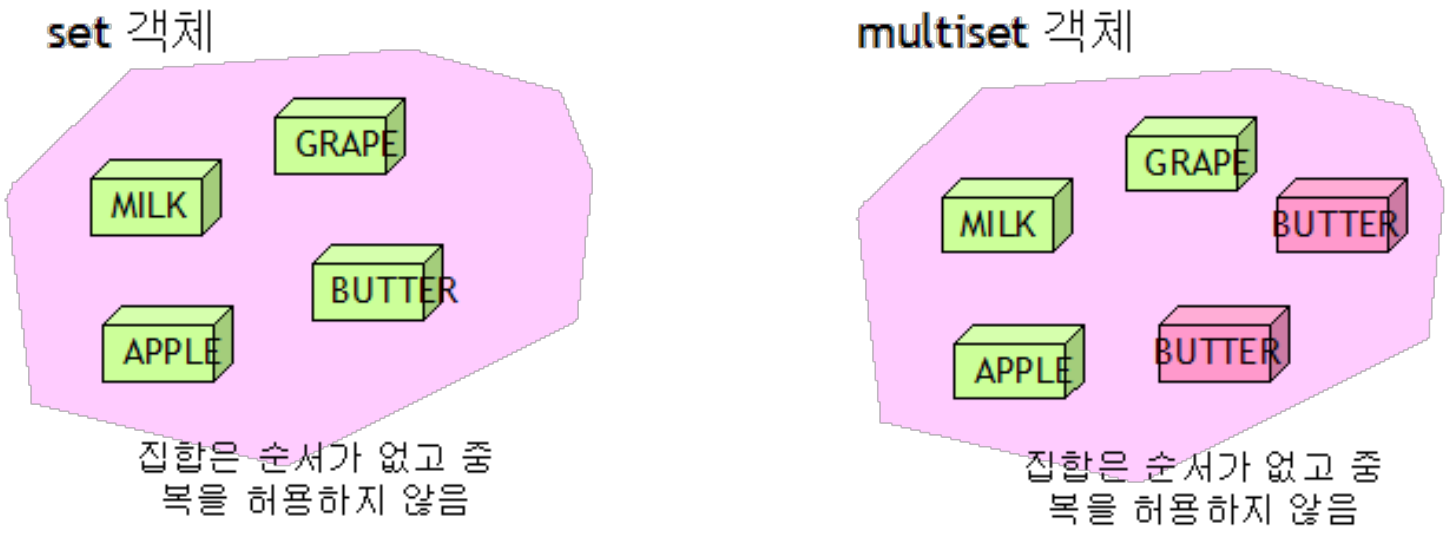


그림 18-9 집합과 다중집합

Modifiers:

insert	Insert element (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_hint C++11	Construct and insert element with hint (public member function)

Observers:

key_comp	Return comparison object (public member function)
value_comp	Return comparison object (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific value (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

std::set::find

<set>

http://www.cplusplus.com/reference/set/set/find/

C++98 C++11

```
const_iterator find (const value_type& val) const;
iterator       find (const value_type& val);
```

Get iterator to element

```
1  // set::find
2  #include <iostream>
3  #include <set>
4
5  int main ()
6  {
7    std::set<int> myset;
8    std::set<int>::iterator it;
9
10   // set some initial values:
11   for (int i=1; i<=5; i++) myset.insert(i*10);    // set: 10 20 30 40 50
12
13   it=myset.find(20);
14   myset.erase (it);
15   myset.erase (myset.find(40));
16
17   std::cout << "myset contains:";
18   for (it=myset.begin(); it!=myset.end(); ++it)
19     std::cout << ' ' << *it;
20   std::cout << '\n';
21
22   return 0;
23 }
```

Output:

```
myset contains: 10 30 50
```

std::set::erase

http://www.cplusplus.com/reference/set/set/erase/

C++98 | C++11

```
(1)      void erase (iterator position);
(2) size_type erase (const value_type& val);
(3)      void erase (iterator first, iterator last);
```

Erase elements

```
 5  int main ()
 6  {
 7    std::set<int> myset;
 8    std::set<int>::iterator it;
 9
10    // insert some values:
11    for (int i=1; i<10; i++) myset.insert(i*10);  // 10 20 30 40 50 60 70 80 90
12
13    it = myset.begin();
14    ++it;                                         // "it" points now to 20
15
16    myset.erase (it);
17
18    myset.erase (40);
19
20    it = myset.find (60);
21    myset.erase (it, myset.end());
22
23    std::cout << "myset contains:";
24    for (it=myset.begin(); it!=myset.end(); ++it)
25      std::cout << ' ' << *it;
26    std::cout << '\n';
```

Get URL

options | compilation | execution

```
myset contains: 10 30 50
```

std::set::count

http://www.cplusplus.com/reference/set/set/count/

```
size_type count (const value_type& val) const;
```

Count elements with a specific value

Searches the container for elements equivalent to *val* and returns the number of matches.

```cpp
// set::count
#include <iostream>
#include <set>

int main ()
{
  std::set<int> myset;

  // set some initial values:
  for (int i=1; i<5; ++i) myset.insert(i*3);    // set: 3 6 9 12

  for (int i=0; i<10; ++i)
  {
    std::cout << i;
    if (myset.count(i)!=0)
      std::cout << " is an element of myset.\n";
    else
      std::cout << " is not an element of myset.\n";
  }

  return 0;
}
```

Output:

```
0 is not an element of myset.
1 is not an element of myset.
2 is not an element of myset.
3 is an element of myset.
4 is not an element of myset.
5 is not an element of myset.
6 is an element of myset.
7 is not an element of myset.
8 is not an element of myset.
9 is an element of myset.
```

std::multiset::count

'www.cplusplus.com/reference/set/multiset/count/

```
size_type count (const value_type& val) const;
```

Count elements with a specific key

Searches the container for elements equivalent to *val* and returns the number of matches.

```
1  // multiset::count
2  #include <iostream>
3  #include <set>
4
5  int main ()
6  {
7    int myints[]={10,73,12,22,73,73,12};
8    std::multiset<int> mymultiset (myints,myints+7);
9
10   std::cout << "73 appears " << mymultiset.count(73) << " times in mymultiset.\n";
11
12   return 0;
13 }
```

Output:

```
73 appears 3 times in mymultiset.
```

C++ shell

cpp.sh
online C++ compiler

about cpp.sh

```cpp
// multiset::count
#include <iostream>
#include <set>

int main ()
{
  int myints[]={10,73,12,22,73,73,12};
  std::multiset<int> mymultiset (myints,myints+7);

  std::cout << "73 appears " << mymultiset.count(73) << " times in mymultiset.\n";

  auto it = mymultiset.begin();
  for(;it != mymultiset.end(); it++) std::cout << *it << " ";
  return 0;
}
```

Get URL

Run

options | compilation | execution

```
73 appears 3 times in mymultiset.
10 12 12 22 73 73 73

Exit code: 0 (normal program termination)
```

std::set::lower_bound

C++98 | C++11

```
      iterator lower_bound (const value_type& val);
const_iterator lower_bound (const value_type& val) const;
```

Return iterator to lower bound

Returns an iterator pointing to the first element in the container which is not considered to go before *val* (i.e., either it is equivalent or goes after).

```
// set::lower_bound/upper_bound
#include <iostream>
#include <set>

int main ()
{
  std::set<int> myset;
  std::set<int>::iterator itlow,itup;

  for (int i=1; i<10; i++) myset.insert(i*10); // 10 20 30 40 50 60 70 80 90

  itlow=myset.lower_bound (30);                //       ^
  itup=myset.upper_bound (60);                 //                   ^

  myset.erase(itlow,itup);                     // 10 20 70 80 90

  std::cout << "myset contains:";
  for (std::set<int>::iterator it=myset.begin(); it!=myset.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Edit & Run

Notice that `lower_bound(30)` returns an iterator to `30`, whereas `upper_bound(60)` returns an iterator to `70`.

```
myset contains: 10 20 70 80 90
```

std::set::equal_range

http://www.cplusplus.com/reference/set/set/equal_range/

C++98 | C++11

```
pair<const_iterator,const_iterator> equal_range (const value_type& val) const;
pair<iterator,iterator>             equal_range (const value_type& val);
```

Get range of equal elements

Returns the bounds of a range that includes all the elements in the container that are equivalent to *val*.

```
1  // set::equal_elements
2  #include <iostream>
3  #include <set>
4
5  int main ()
6  {
7    std::set<int> myset;
8
9    for (int i=1; i<=5; i++) myset.insert(i*10);    // myset: 10 20 30 40 50
10
11   std::pair<std::set<int>::const_iterator,std::set<int>::const_iterator> ret;
12   ret = myset.equal_range(30);
13
14   std::cout << "the lower bound points to: " << *ret.first << '\n';
15   std::cout << "the upper bound points to: " << *ret.second << '\n';
16
17   return 0;
18 }
```

```
the lower bound points to: 30
the upper bound points to: 40
```

class template

std::pair

<utility>

```
template <class T1, class T2> struct pair;
```

Pair of values

This class couples together a pair of values, which may be of different types (`T1` and `T2`). The individual values can be accessed through its public members `first` and `second`.

Pairs are a particular case of `tuple`.

Template parameters

`T1`

Type of member `first`, aliased as `first_type`.

`T2`

Type of member `second`, aliased as `second_type`.

Member types

member type	definition	notes
`first_type`	The first template parameter (`T1`)	Type of member `first`.
`second_type`	The second template parameter (`T2`)	Type of member `second`.

Member variables

member variable	definition
`first`	The first value in the `pair`
`second`	The second value in the `pair`

std::multiset::equal_range

http://www.cplusplus.com/reference/set/multiset/equal_range/

C++98 | C++11

```
pair<const_iterator,const_iterator> equal_range (const value_type& val) const;
pair<iterator,iterator>             equal_range (const value_type& val);
```

Get range of equal elements

```
1  // multiset::equal_elements
2  #include <iostream>
3  #include <set>
4
5  typedef std::multiset<int>::iterator It;  // aliasing the iterator type used
6
7  int main ()
8  {
9    int myints[]= {77,30,16,2,30,30};
10   std::multiset<int> mymultiset (myints, myints+6);  // 2 16 30 30 30 77
11
12   std::pair<It,It> ret = mymultiset.equal_range(30); //       ^        ^
13
14   mymultiset.erase(ret.first,ret.second);
15
16   std::cout << "mymultiset contains:";
17   for (It it=mymultiset.begin(); it!=mymultiset.end(); ++it)
18     std::cout << ' ' << *it;
19   std::cout << '\n';
20
21   return 0;
22 }
```

```
multiset contains: 2 16 77
```

map / multimap in <map>

- Map은 사전과 같이 <key,value> pairs 가 순서대로 저장되어 있다.
- map<>::operator[] 키(key)가 제시되면 값(value) 의 reference 를 반환
- map 에서는 key 가 unique 하고 multimap 에서는 그렇지 않다.
- multimap<> 에는 operator[] 가 없다. (key 가 unique 하지 않으므로)
- key 는 member variable first, value 는 member variable second
- iterators are bidirectional.
- pair 들의 집합

Map

그림 18-10 Map의 개념

Headers		<set>		<map>	
Members		**set**	**multiset**	**map**	**multimap**
	constructor	set	multiset	map	multimap
	destructor	~set	~multiset	~map	~multimap
	assignment	operator=	operator=	operator=	operator=
iterators	`begin`	begin	begin	begin	begin
	`end`	end	end	end	end
	`rbegin`	rbegin	rbegin	rbegin	rbegin
	`rend`	rend	rend	rend	rend
const iterators	`cbegin`	cbegin	cbegin	cbegin	cbegin
	`cend`	cend	cend	cend	cend
	`crbegin`	crbegin	crbegin	crbegin	crbegin
	`crend`	crend	crend	crend	crend
capacity	`size`	size	size	size	size
	`max_size`	max_size	max_size	max_size	max_size
	`empty`	empty	empty	empty	empty
	`reserve`				
element access	`at`			at	
	`operator[]`			operator[]	
modifiers	`emplace`	emplace	emplace	emplace	emplace
	`emplace_hint`	emplace_hint	emplace_hint	emplace_hint	emplace_hint
	`insert`	insert	insert	insert	insert
	`erase`	erase	erase	erase	erase
	`clear`	clear	clear	clear	clear
	`swap`	swap	swap	swap	swap
operations	`count`	count	count	count	count
	`find`	find	find	find	find
	`equal_range`	equal_range	equal_range	equal_range	equal_range
	`lower_bound`	lower_bound	lower_bound	lower_bound	lower_bound
	`upper_bound`	upper_bound	upper_bound	upper_bound	upper_bound

std::map::operator[]

http://www.cplusplus.com/reference/map/map/operator[]/

C++98 | C++11

```
mapped_type& operator[] (const key_type& k);
mapped_type& operator[] (key_type&& k);
```

Access element

If *k* matches the key of an element in the container, the function returns a reference to its mapped value.

```
1  // accessing mapped values
2  #include <iostream>
3  #include <map>
4  #include <string>
5
6  int main ()
7  {
8    std::map<char,std::string> mymap;
9
10   mymap['a']="an element";
11   mymap['b']="another element";
12   mymap['c']=mymap['b'];
13
14   std::cout << "mymap['a'] is " << mymap['a'] << '\n';
15   std::cout << "mymap['b'] is " << mymap['b'] << '\n';
16   std::cout << "mymap['c'] is " << mymap['c'] << '\n';
17   std::cout << "mymap['d'] is " << mymap['d'] << '\n';
18
19   std::cout << "mymap now contains " << mymap.size() << " elements.\n";
20
21   return 0;
22 }
```

mymap.insert(pair<char,string>('a',"an element");

mymap['d'] is created with empty value

Output:

```
mymap['a'] is an element
mymap['b'] is another element
mymap['c'] is another element
mymap['d'] is
mymap now contains 4 elements.
```

- keys are unique.

```cpp
int main ()
{
  std::map<char,std::string> mymap;

  mymap['a']="an element";
  mymap['b']="another element";
  mymap['c']=mymap['b'];
  mymap['a']="new element";

  std::cout << "mymap['a'] is " << mymap['a'] << '\n';
  std::cout << "mymap['b'] is " << mymap['b'] << '\n';
  std::cout << "mymap['c'] is " << mymap['c'] << '\n';
  std::cout << "mymap['d'] is " << mymap['d'] << '\n';

  std::cout << "mymap now contains " << mymap.size() << " elements.\n";

  return 0;
}
```

Get URL

options | compilation | execution

```
mymap['a'] is new element
mymap['b'] is another element
mymap['c'] is another element
mymap['d'] is
mymap now contains 4 elements.
```

```cpp
#include <iostream>
#include <map>

int main ()
{
  std::map<char,int> mymap;

  mymap['b'] = 100;
  mymap['a'] = 200;
  mymap['c'] = 300;

  // show content:
  for (std::map<char,int>::iterator it=mymap.begin(); it!=mymap.end(); ++it)
    std::cout << it->first << " => " << it->second << '\n';

  return 0;
}
```

Get URL

options | compilation | execution

```
a => 200
b => 100
c => 300
```

sorted with keys

multimap

operator[] 가 없음

multimap<>::insert()

multimap<>::find()

multimap<>::begin()

multimap<>::end()

```cpp
int main ()
{
  std::multimap<char,int> mymultimap;
  std::multimap<char,int>::iterator it;

  mymultimap.insert ( std::pair<char,int>('a',100) );
  mymultimap.insert ( std::pair<char,int>('z',150) );
  mymultimap.insert ( std::pair<char,int>('b',75) );
  mymultimap.insert ( std::pair<char,int>('c',300));
  mymultimap.insert ( std::pair<char,int>('a',30));

  //  range insertion
  std::multimap<char,int> anothermultimap;
  anothermultimap.insert(mymultimap.begin(),mymultimap.find('c'));

  // showing contents:
  std::cout << "mymultimap contains:\n";
  for (it=mymultimap.begin(); it!=mymultimap.end(); ++it)
    std::cout << (*it).first << " => " << (*it).second << '\n';

  std::cout << "anothermultimap contains:\n";
  for (it=anothermultimap.begin(); it!=anothermultimap.end(); ++it)
    std::cout << (*it).first << " => " << (*it).second << '\n';
```

Get URL

options | compilation | execution

```
mymultimap contains:
a => 100
a => 30
b => 75
c => 300
z => 150
anothermultimap contains:
a => 100
a => 30
b => 75
```

std::multimap::equal_range

http://www.cplusplus.com/reference/map/multimap/equal_range/

```
pair<const_iterator,const_iterator> equal_range (const key_type& k) const;
pair<iterator,iterator>             equal_range (const key_type& k);
```

Get range of equal elements

Returns the bounds of a range that includes all the elements in the container which have a *key* equivalent to *k*.

```
 7  std::multimap<char,int> mymm;
 8
 9  mymm.insert(std::pair<char,int>('a',10));
10  mymm.insert(std::pair<char,int>('b',20));
11  mymm.insert(std::pair<char,int>('b',30));
12  mymm.insert(std::pair<char,int>('b',40));
13  mymm.insert(std::pair<char,int>('c',50));
14  mymm.insert(std::pair<char,int>('c',60));
15  mymm.insert(std::pair<char,int>('d',60));
16
17  std::cout << "mymm contains:\n";
18  for (char ch='a'; ch<='d'; ch++)
19  {
20    std::pair <std::multimap<char,int>::iterator, std::multimap<char,int>::iterator> ret;
21    ret = mymm.equal_range(ch);
22    std::cout << ch << " =>";
23    for (std::multimap<char,int>::iterator it=ret.first; it!=ret.second; ++it)
24      std::cout << ' ' << it->second;
25    std::cout << '\n';
```

```
mymm contains:
a => 10
b => 20 30 40
c => 50 60
d => 60
```

Headers		<set>		<map>		<unordered_set>		<unordered_map>	
Members		**set**	**multiset**	**map**	**multimap**	**unordered_set**	**unordered_multiset**	**unordered_map**	**unordered_multimap**
	constructor	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap
	destructor	~set	~multiset	~map	~multimap	~unordered_set	~unordered_multiset	~unordered_map	~unordered_multimap
	assignment	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin	begin	begin	begin	begin
	end	end	end	end	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin	rbegin				
	rend	rend	rend	rend	rend				
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin
	cend	cend	cend	cend	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin	crbegin	crbegin				
	crend	crend	crend	crend	crend				
capacity	size	size	size	size	size	size	size	size	size
	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	empty	empty	empty	empty	empty
	reserve					reserve	reserve	reserve	reserve
element access	at			at				at	
	operator[]			operator[]				operator[]	
modifiers	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace
	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint
	insert	insert	insert	insert	insert	insert	insert	insert	insert
	erase	erase	erase	erase	erase	erase	erase	erase	erase
	clear	clear	clear	clear	clear	clear	clear	clear	clear
	swap	swap	swap	swap	swap	swap	swap	swap	swap
operations	count	count	count	count	count	count	count	count	count
	find	find	find	find	find	find	find	find	find
	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range
	lower_bound	lower_bound	lower_bound	lower_bound	lower_bound				
	upper_bound	upper_bound	upper_bound	upper_bound	upper_bound				
observers	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator
	key_comp	key_comp	key_comp	key_comp	key_comp				
	value_comp	value_comp	value_comp	value_comp	value_comp				
	key_eq					key_eq	key_eq	key_eq	key_eq
	hash_function					hash_function	hash_function	hash_function	hash_function
buckets	bucket					bucket	bucket	bucket	bucket
	bucket_count					bucket_count	bucket_count	bucket_count	bucket_count
	bucket_size					bucket_size	bucket_size	bucket_size	bucket_size
	max_bucket_count					max_bucket_count	max_bucket_count	max_bucket_count	max_bucket_count
hash policy	rehash					rehash	rehash	rehash	rehash
	load_factor					load_factor	load_factor	load_factor	load_factor
	max_load_factor					max_load_factor	max_load_factor	max_load_factor	max_load_factor