



STL

-CHAPTER3-

SOULSEEK

목차

1. set – 연관 컨테이너

2. multiset – 연관 컨테이너

3. map – 연관 컨테이너

4. multimap – 연관 컨테이너

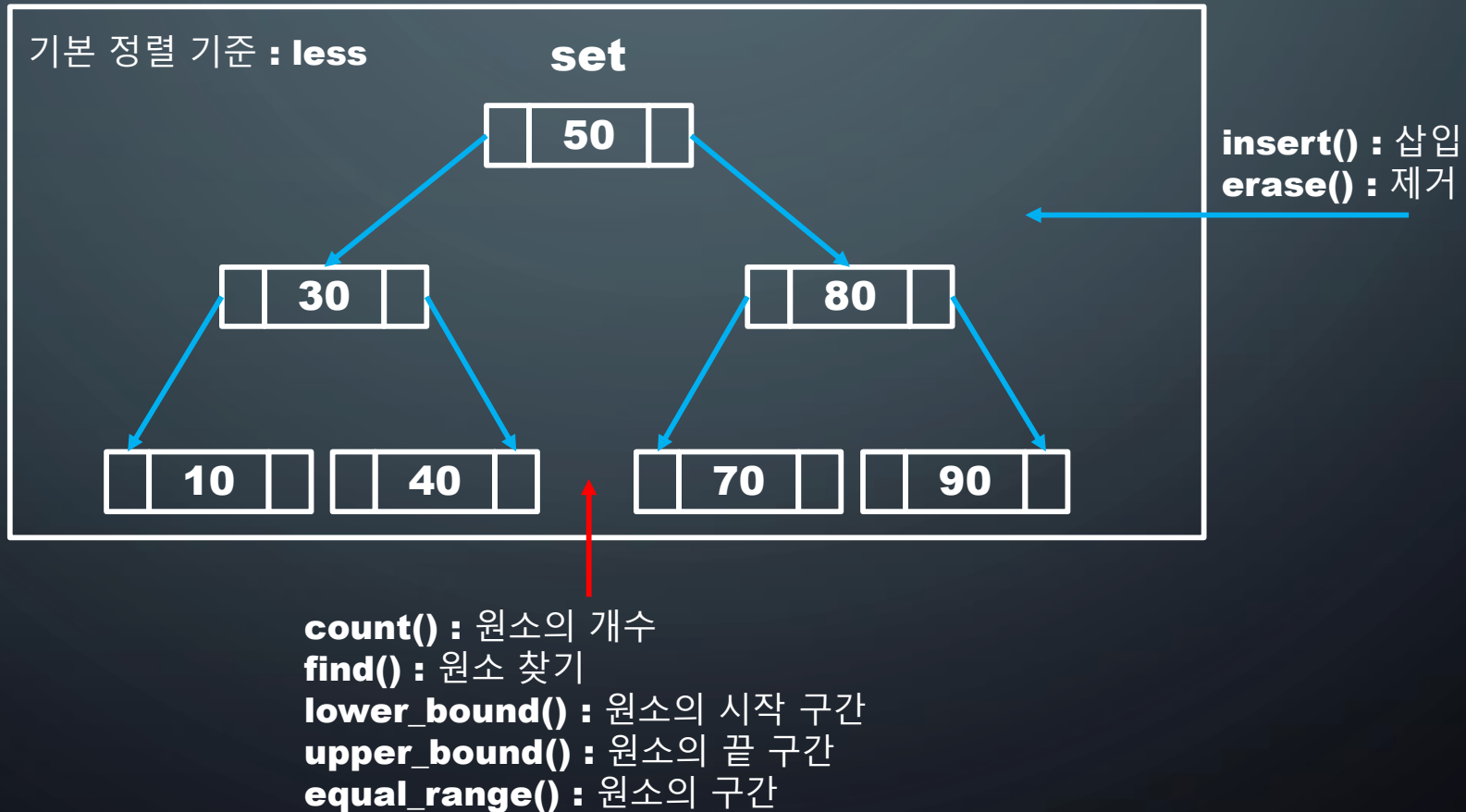
The image features a dark blue background with a subtle radial gradient. In the four corners, there are decorative white line art elements resembling electronic circuit traces or a stylized atomic structure, with small circles at the end of the lines.

SET

1.SET

특징

- 노드 기반의 컨테이너
- 특정 정렬 기준에 의해 원소가 자동 정렬되는 컨테이너
- 원소 찾기(검색)를 로그 시간 복잡도에 수행할 수 있도록 균형 이진 트리로 구현 되어있다.
- 여러 가지 찾기 관련 함수를 제공한다.
- 원소가 **key**가 되고, **key**의 정렬 기준이 핵심이다.



1.SET

템플릿 형식

template<typename Key, typename Pred = less<Key>, typename Allocator = allocator<Key>>

→ **Key**는 원소의 형식이며, **Pred**는 정렬 기준인 조건자, 기본은 **less**

class set

생성자

set s	s 는 빈 컨테이너이다.
set s(pred)	s 는 빈 컨테이너로 정렬 기준은 pred 조건자 사용
set s(s2)	s 는 s2 컨테이너의 복사본이다(복사 생성자 호출)
set s(b, e)	s 는 반복자 구간 [b, e) 로 초기화된 원소를 갖는다.
set s(b, e, pred)	s 는 반복자 구간 [b, e) 로 초기화된 원소를 갖고 pred 조건자를 갖는다.

연산자

s1 == s2	s1 과 s2 의 모든 원소가 같은가?(bool 형식)
s1 != s2	s1 과 s2 의 모든 원소 중 하나라도 다른 원소가 있는가? (bool 형식)
s1 < s2	문자열 비교처럼 s2 가 s1 보다 큰가?(bool 형식)
s1 <= s2	문자열 비교처럼 s2 가 s1 보다 크거나 같은가?(bool 형식)
s1 > s2	문자열 비교처럼 s1 이 s2 보다 큰가?(bool 형식)
s1 >= s2	문자열 비교처럼 s1 이 s2 보다 크거나 같은가?(bool 형식)

1.SET

멤버 함수

p=s.begin()	p 는 s 의 첫 원소를 가리키는 반복자다(const , 비 const 버전 있음)
s.clear()	s 의 모든 원소를 제거한다.
n=count(k)	원소 k 의 개수를 반환한다.
s.empty()	s 가 비었는지 조사한다.
p=s.end()	p 는 s 의 끝을 표시하는 반복자다(const , 비 const 버전 있음)
pr=s.equal_range(k)	Pr 은 k 원소의 반복자 구간인 pair 객체다.(const , 비 const 버전 있음)
q=s.erase(p)	p 가 가리키는 원소를 제거한다. q 는 다음 원소를 가리킨다.
q=s.erase(b,e)	반복자 구간[b, e)의 모든 원소를 제거한다. q 는 다음 원소를 가리킨다.
n=s.erase(k)	k 원소를 모두 제거한다. n 은 제거한 개수다.
p=s.find(k)	p 는 k 원소의 위치를 가리키는 반복자다(const , 비 const 버전 있음)
pr=s.insert(k)	s 컨테이너에 k 를 삽입한다. Pr 은 삽입한 원소를 가리키는 반복자와 성공 여부의 bool 값인 pair 객체다.
q=s.insert(p,k)	p 가 가리키는 위치부터 빠르게 k 를 삽입한다. Q 는 삽입한 원소를 가리키는 반복자
s.Insert(b,e)	반복자 구간[b, e)의 원소를 삽입한다.
pred=s.key_comp()	pred 는 s 의 key 정렬 기본인 조건자다(const , 비 const 버전 있음)
p=s.lower_bound(k)	p 는 k 의 시작 구간을 가리키는 반복자다.(const , 비 const 버전 있음)
n=s_max_size()	n 는 s 가 담을 수 있는 최대 원소의 개수다(메모리 크기)

1.SET

멤버 함수

p=s.rbegin()	p 는 s 의 역 순차열의 첫 원소를 가리키는 반복자다(const , 비 const 있음)
p=s.rend()	p 는 s 의 역 순차열의 끝을 표시하는 반복자다.(const , 비 const 있음)
s.size()	s 원소의 개수다
s.swap(s2)	s 와 s2 를 swap 한다.
p=s.upper_bound(k)	p 는 k 의 끝 구간을 가리키는 반복자다(const , 비 const 버전 있음)
pred=s.value_comp()	pred 는 s 의 value 정렬 기준인 조건자다(value_compare 타입)

1.SET

멤버 형식	
allocator_type	메모리 관리자 형식
const_iterator	Const 반복자 형식
const_pointer	Const value_type* 형식
const_reference	Const value_type& 형식
const_reverse_iterator	Const 역 반복자 형식
difference_type	두 반복자 차이의 형식
iterator	반복자 형식
key_compare	키(key)조건자(비교) 형식(set 은 key 가 value 이므로 value_compare 와 같음)
key_type	키(key)의 형식(set 은 key 가 value 이므로 value_type 과 같음)
pointer	Value_type* 형식
reference	Value_type& 형식
reverse_iterator	역 반복자 형식
size_type	첨자(index)나 원소의 개수 등의 형식
value_compare	원소 조건자(비교) 형식
value_type	원소의 형식

1.SET

insert() 사용 예제..

```
void main( )  
{
```

```
    set<int> s; // 정수 원소를 저장하는 기본 정렬 기준이 less인 빈 컨테이너 생성
```

```
    s.insert(50); //랜덤으로 원소(key)를 삽입한다.
```

```
    s.insert(30);
```

```
    s.insert(80);
```

```
    s.insert(40);
```

```
    s.insert(10);
```

```
    s.insert(70);
```

```
    s.insert(90);
```

```
    set<int>::iterator iter; // 기본 정렬 기준이 less인 set의 양방향 반복자
```

```
    for( iter = s.begin() ; iter != s.end() ; ++iter)
```

```
        cout << *iter << " "; // inorder 2진 트리 탐색 순서로 출력된다.
```

```
    cout << endl;
```

```
    s.insert(50); //중복된 원소(key)를 삽입한다. 실패!!
```

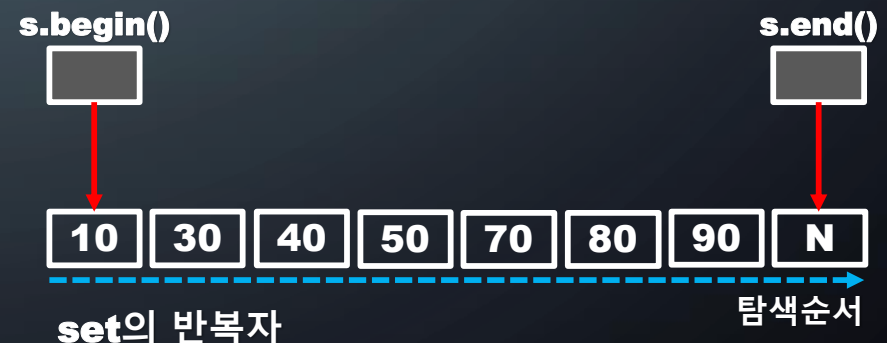
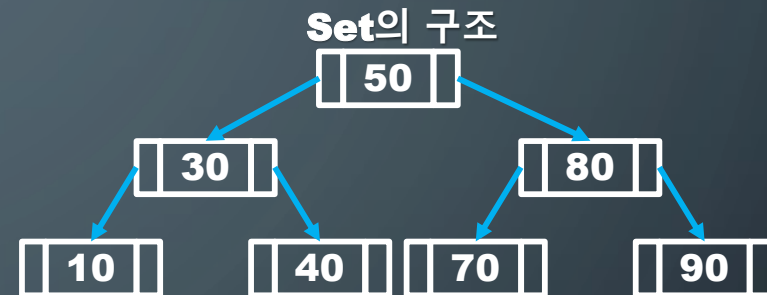
```
    s.insert(50);
```

```
    for( iter = s.begin() ; iter != s.end() ; ++iter)
```

```
        cout << *iter << " "; // 결과는 같다.
```

```
    cout << endl;
```

```
}
```



1.SET

insert()의 반환값(pair) 예제..

```
void main( )
{
    set<int> s;

    pair<set<int>::iterator, bool> pr;

    pr = s.insert(50); // 50 원소의 첫 번째 삽입
    s.insert(40);
    s.insert(80);

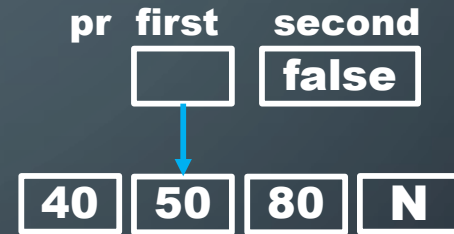
    if( true == pr.second )
        cout << *pr.first << " 삽입 성공!" << endl;
    else
        cout << *pr.first << "가 이미 있습니다. 삽입 실패!" << endl;

    set<int>::iterator iter;
    for( iter = s.begin() ; iter != s.end() ; ++iter)
        cout << *iter << " ";
    cout << endl;

    pr = s.insert(50); // 50 원소의 두 번째 삽입. 실패!!

    if( true == pr.second )
        cout << *pr.first << " 삽입 성공!" << endl;
    else
        cout << *pr.first << "가 이미 있습니다. 삽입 실패!" << endl;

    for( iter = s.begin() ; iter != s.end() ; ++iter)
        cout << *iter << " ";
    cout << endl;
}
```



중복된 원소를 삽입하게 될 때, Error가 발생한다. 삽입 시 반환 값을 받게 하면 반환 받은 값에 first가 원소가 있는 지점을 가리키고 second가 삽입 성공 여부를 가지게 되는 것을 이용해보자.

1.SET

insert()를 사용해 지정한 곳에 삽입하는 예제..

```
void main( )
{
    set<int> s;
    pair<set<int>::iterator, bool> pr;

    s.insert(50);
    s.insert(30);
    s.insert(80);
    s.insert(40);
    s.insert(10);
    s.insert(70);
    pr=s.insert(90); //pr.first는 90원소의 반복자

    set<int>::iterator iter;
    for( iter = s.begin() ; iter != s.end() ; ++iter)
        cout << *iter << " ";
    cout << endl;

    s.insert(pr.first, 85); //90원소의 반복자에서 검색 시작 후 삽입한다.

    for( iter = s.begin() ; iter != s.end() ; ++iter)
        cout << *iter << " ";
    cout << endl;
}
```

1.SET

정렬 기준을 사용한 생성 예제..

```
void main( )
{
    // 정렬 기준으로 greater<int> 조건자를 사용.
    set<int, greater<int>> s;
    s.insert(50);
    s.insert(30);
    s.insert(80);
    s.insert(40);
    s.insert(10);
    s.insert(70);
    s.insert(90);

    // greater<int> 조건자를 사용하는 반복자 생성
    set<int, greater<int>>::iterator iter;

    for( iter = s.begin() ; iter != s.end() ; ++iter)
        cout << *iter << " ";

    cout << endl;
}
```

1.SET

key_comp(), value_comp() 사용 예제..

```
void main( )
```

```
{
```

```
    set<int, less<int> > s_less; // set<int> s와 같습니다.
```

```
    set<int, greater<int> > s_greater; // 정렬 기준으로 greater<int> 조건자를 사용.
```

```
    s_less.insert(50);
```

```
    s_less.insert(80);
```

```
    s_less.insert(40);
```

```
    s_greater.insert(50);
```

```
    s_greater.insert(80);
```

```
    s_greater.insert(40);
```

```
    set<int, less<int> >::key_compare l_cmp = s_less.key_comp();
```

```
    cout << l_cmp(10, 20) << endl; // 10 < 20 연산
```

```
    set<int, greater<int> >::key_compare g_cmp = s_greater.key_comp();
```

```
    cout << g_cmp(10, 20) << endl; // 10 > 20 연산
```

```
    //key_comp(), value_comp()의 타입을 문자열로 출력한다.
```

```
    cout <<"key_compare type: " << typeid( s_less.key_comp() ).name() << endl;
```

```
    cout <<"key_compare type: " << typeid( s_greater.key_comp() ).name() << endl;
```

```
    cout <<"value_compare type: " << typeid( s_less.value_comp() ).name() << endl;
```

```
    cout <<"value_compare type: " << typeid( s_greater.value_comp() ).name() << endl;
```

```
}
```

각 조건자를 생성한다.

set<int, less<int>>::key_compare 오름차순 정렬로 생성한 **set**의 조건자를 생성한다.**set<int, greater<int>>::key_compare** 내림차순 정렬로 생성한 **set**의 조건자를 생성한다.

1.SET

find() 사용 예제..

```
void main()
{
```

```
    set<int> s;
```

```
    s.insert(50);
```

```
    s.insert(30);
```

```
    s.insert(80);
```

```
    s.insert(40);
```

```
    s.insert(10);
```

```
    s.insert(70);
```

```
    s.insert(90);
```

```
    set<int>::iterator iter;
```

```
    for (iter = s.begin(); iter != s.end(); ++iter)
```

```
        cout << *iter << " ";
```

```
    cout << endl;
```

```
    iter = s.find(30); // 30의 반복자를 반환
```

```
    if (iter != s.end())
```

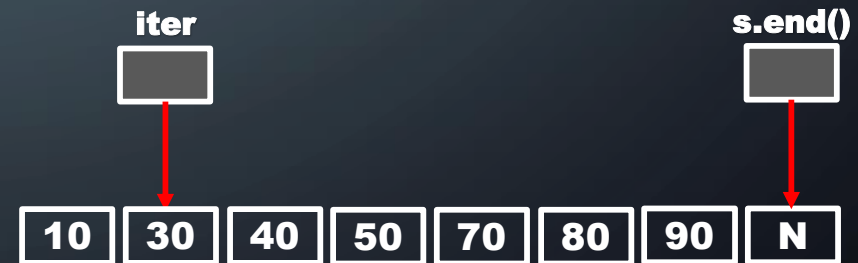
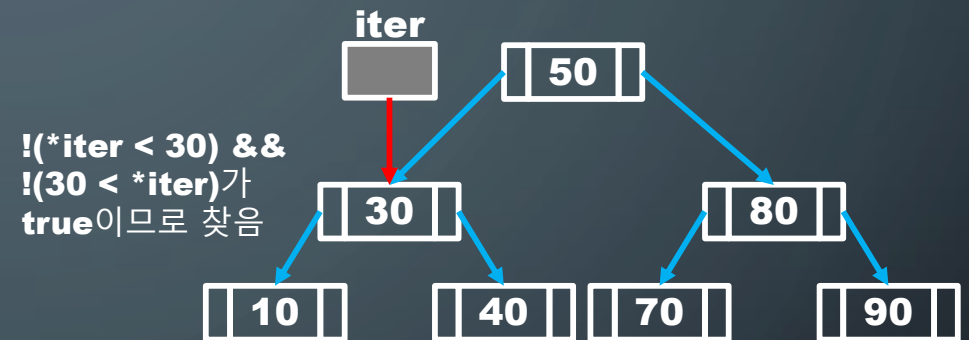
```
        cout << *iter << "가 s에 있다!" << endl;
```

```
    else
```

```
        cout << "30이 s에 없다!" << endl;
```

```
}
```

연관 컨테이너에서는 **key**를 찾을 때 **==**연산을 하지 않고 **<**연산을 하기 때문에 두 원소 **a, b**가 **!(a < b) && !(b < a)**인 경우가 **a == b**인 경우가 된다.



1.SET

정렬 기준의 비교로 **find**를 이해해보자

```
void main()
{
    set<int, less<int> > s; // 정렬 기준 less

    // 30과 50의 비교
    cout << (!s.key_comp()(30, 50)
        && !s.key_comp()(50, 30)) << endl; //다르다

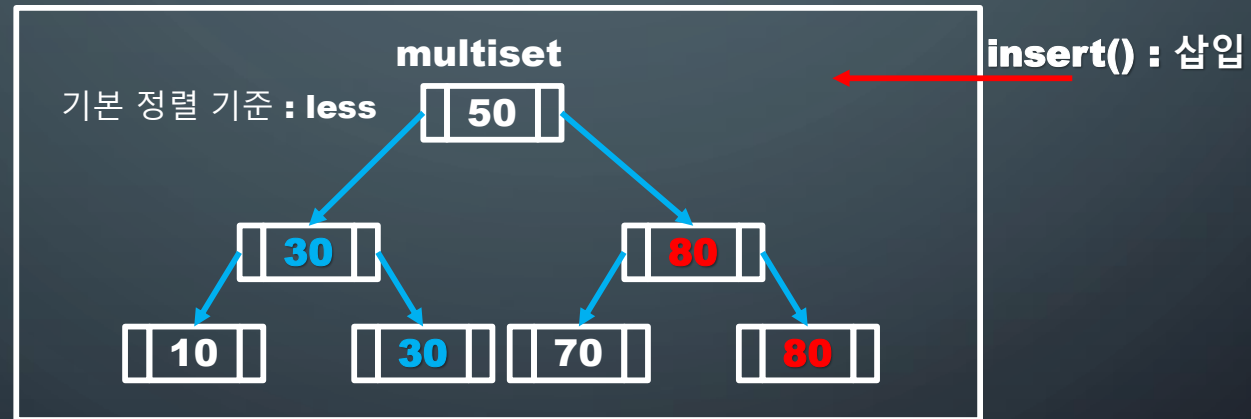
    // 30과 30의 비교
    cout << (!s.key_comp()(30, 30)
        && !s.key_comp()(30, 30)) << endl; //같다(equivalence)
}
```


The image features a dark blue gradient background with faint, light blue concentric circles centered behind the text. In the four corners, there are decorative white line art elements resembling circuit traces or neural network connections, with small circles at various points.

MULTISET

2.MULTISET

- 템플릿 형식, 인터페이스, 멤버 형식이 모두 **set**과 같다.
- **set**과 다르게 중복**key**가 가능하다.
- 중복이 허용되기 때문에 삽입여부를 반환하는 **pair**대신 위치만을 가리키는 반복자를 반환한다.



2.MULTISET

insert() 사용 예제..

```
void main()
{
    multiset<int> ms;
    multiset<int>::iterator iter;

    ms.insert(50);
    ms.insert(30);
    ms.insert(80);
    ms.insert(80); // 80 중복
    ms.insert(30); // 30 중복
    ms.insert(70);
    iter = ms.insert(10);

    cout << "iter의 원소: " << *iter << endl;

    for (iter = ms.begin(); iter != ms.end(); ++iter)
        cout << *iter << " ";

    cout << endl;
}
```

2.MULTISET

count(), find(), lower_bound(), upper_bound() 사용 예제..

```
void main()
{
    multiset<int> ms;

    ms.insert(50);
    ms.insert(30);
    ms.insert(80);
    ms.insert(80); // 80 중복
    ms.insert(30); // 30 중복
    ms.insert(70);
    ms.insert(10);

    multiset<int>::iterator iter;
    for (iter = ms.begin(); iter != ms.end(); ++iter)
        cout << *iter << " ";

    cout << endl;

    cout << "30 원소의 개수: " << ms.count(30) << endl; // 30 원소의 개수

    iter = ms.find(30); // 30 첫 번째 원소의 반복자
    cout << "iter: " << *iter << endl;

    multiset<int>::iterator lower_iter;
    multiset<int>::iterator upper_iter;

    lower_iter = ms.lower_bound(30); // 30 순차열의 시작 반복자
    upper_iter = ms.upper_bound(30); // 30 순차열의 끝 표시 반복자

    cout << "lower_iter: " << *lower_iter << ", "
           << "upper_iter: " << *upper_iter << endl;

    cout << "구간 [lower_iter, upper_iter)의 순차열: ";
    for (iter = lower_iter; iter != upper_iter; ++iter)
        cout << *iter << " ";

    cout << endl;
}
```

2.MULTISET

equal_range() 사용 예제..

```
void main()
{
    multiset<int> ms;

    ms.insert(50);
    ms.insert(30);
    ms.insert(80);
    ms.insert(80); // 80 중복
    ms.insert(30); // 30 중복
    ms.insert(70);
    ms.insert(10);

    multiset<int>::iterator iter;
    for (iter = ms.begin(); iter != ms.end(); ++iter)
        cout << *iter << " ";
    cout << endl;

    //multiset의 반복자 쌍(pair) 객체 생성
    pair<multiset<int>::iterator, multiset<int>::iterator> iter_pair;
    iter_pair = ms.equal_range(30);

    for (iter = iter_pair.first; iter != iter_pair.second; ++iter)
        cout << *iter << " "; //[iter_pair.first, iter_pair.second) 구간의 순차열
    cout << endl;
}
```

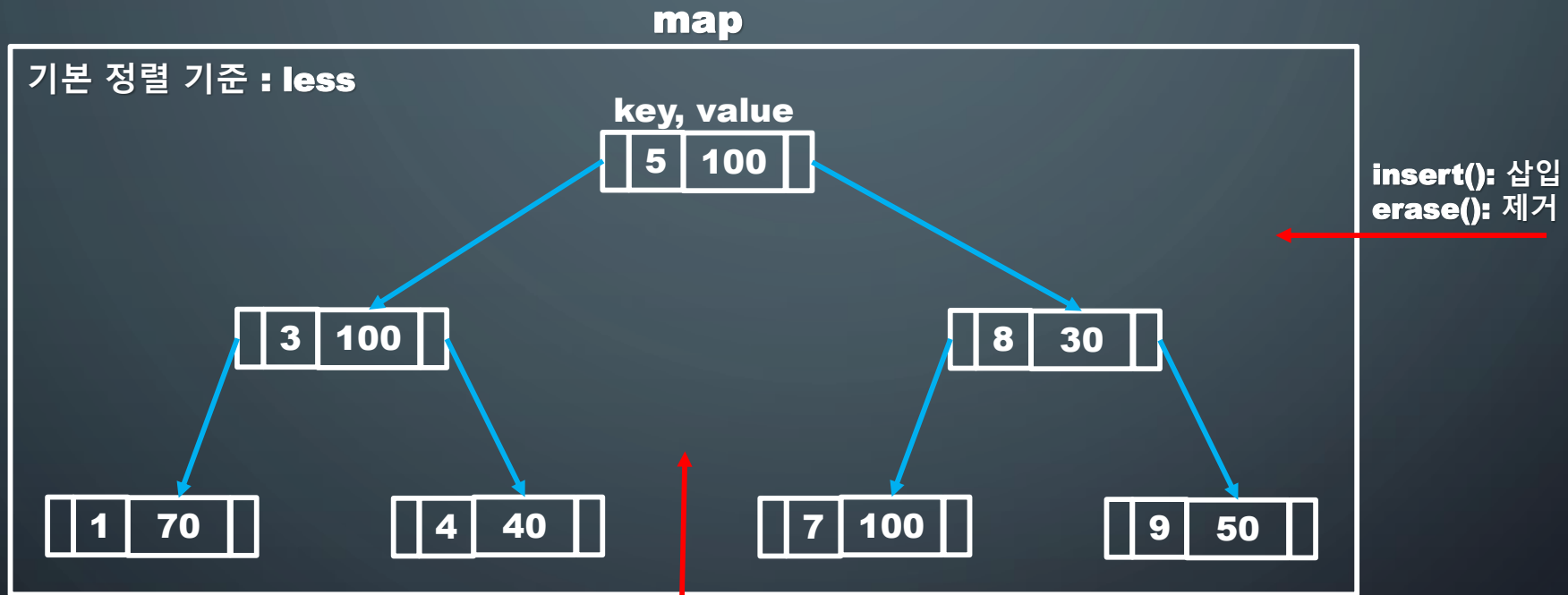
지정한 원소의 반복 시작 반복자와 끝 반복자를 반환한다.

The image features a dark blue gradient background with faint, concentric circular patterns. In the corners, there are decorative white line art elements resembling circuit boards or neural networks, with lines and small circles connecting them.

MAP

3.MAP

- 노드 기반 연관 컨테이너
- 특정 정렬을 기준으로 자동 정렬되며 **set**과 똑같은 인터페이스와 멤버 함수를 제공한다.
- **set**과 달리 **value**는 **key**에 해당하는 것을 따로 가지고 있고 **key**를 기준으로 검색한다.
- **key**를 이용한 [] 연산이 가능하다.



[] 연산자 오버로딩 지원
m[key] = value: 원소 추가 or 갱신 + 찾기 관련 멤버 함수

3.MAP

템플릿 형식

template<typename Key, typename Value, typename Pred=less<Key>, typename Allocator=allocator<pair<const key, Value>>

연산자

m[k] = v	M 컨테이너에 원소(k, v)를 추가하거나 key 에 해당하는 원소의 value 를 v 로 갱신한다.
-----------------	---

멤버 형식

allocator_type	메모리 관리자 형식
const_iterator	Const 반복자 형식
const_pointer	Const value_type* 형식
const_reference	Const value type& 형식
const_reverse_iterator	Const 역 반복자 형식
diffference_type	두 반복자 차이의 형식
iterator	반복자 형식
key_compare	키(key) 조건자(비교)형식
key_type	키(key)의 형식
mapped_type	값(value)의 형식
pointer	value_type* 형식

3.MAP

멤버 형식

reference

value_type& 형식

reverse_iterator

역 반복자 형식

size_type

첨자(**index**)나 원소의 개수 등의 형식

value_type

원소의 형식

3.MAP

insert() 사용 예제..

```
void main( )
```

```
{
```

```
//key, value 모두 정수형인 컨테이너 생성
```

```
//기본 정렬 기준 less
```

```
map<int,int> m;
```

```
m.insert( pair<int,int>(5,100) ); // 임시 pair 객체 생성 후 저장
```

```
m.insert( pair<int,int>(3,100) );
```

```
m.insert( pair<int,int>(8,30) );
```

```
m.insert( pair<int,int>(4,40) );
```

```
m.insert( pair<int,int>(1,70) );
```

```
m.insert( pair<int,int>(7,100) );
```

```
pair<int, int> pr(9,50);
```

```
m.insert( pr ); // pr 객체 생성 후 저장
```

```
map<int,int>::iterator iter;
```

```
for( iter = m.begin() ; iter != m.end() ; ++iter)
```

```
    cout <<"(" << (*iter).first <<',' << (*iter).second <<")" <<" ";
```

```
cout << endl;
```

```
// 반복자는 -> 연산자가 연산자 오버로딩되어 있으므로
```

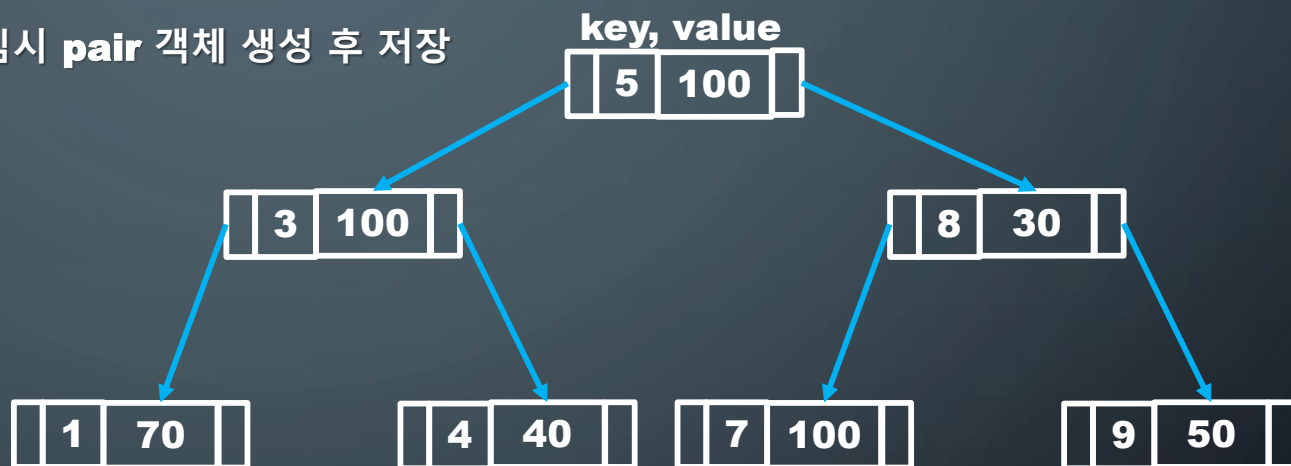
```
//포인터처럼 멤버를 -> 연산자로 접근할 수 있습니다.
```

```
for( iter = m.begin() ; iter != m.end() ; ++iter)
```

```
    cout <<"(" << iter->first <<',' << iter->second <<")" <<" ";
```

```
cout << endl;
```

```
}
```



key type value type

map<int, int> m;

key value

m.insert(pair<int, int>(5, 100));

key type value type

3.MAP

Insert() 결과를 반환하는 예제..

```
void main()
{
    map<int, int> m;
    pair<map<int, int>::iterator, bool> pr; // insert() 결과 pair 객체

    m.insert(pair<int, int>(5, 100));
    m.insert(pair<int, int>(3, 100));
    m.insert(pair<int, int>(8, 30));
    m.insert(pair<int, int>(4, 40));
    m.insert(pair<int, int>(1, 70));
    m.insert(pair<int, int>(7, 100));

    pr = m.insert(pair<int, int>(9, 50)); // 성공!
    if (true == pr.second)
        cout << "key: " << pr.first->first << ", value: " << pr.first->second << " 저장 완료!" << endl;
    else
        cout << "key 9가 이미 m에 있습니다." << endl;

    pr = m.insert(pair<int, int>(9, 50)); // 실패!
    if (true == pr.second)
        cout << "key: " << pr.first->first << ", value: " << pr.first->second << "저장 완료!" << endl;
    else
        cout << "key: 9가 이미 m에 있습니다." << endl;
}
```

Pair로 **key, value**로 쌍을 지어서 삽입하고 그 결과를 **pair**로 **set**과 같은 형태로 반환한다.

3.MAP

연산자 사용 예제..

```
void main()
```

```
{
```

```
    map<int, int> m;
```

```
    m[5] = 100; //key 5, value 100의 원소를 m에 삽입한다.
```

```
    m[3] = 100;
```

```
    m[8] = 30;
```

```
    m[4] = 40;
```

```
    m[1] = 70;
```

```
    m[7] = 100;
```

```
    m[9] = 50;
```

```
    map<int, int>::iterator iter;
```

```
    for (iter = m.begin(); iter != m.end(); ++iter)
```

```
        cout << "(" << iter->first << ',' << iter->second << ")" << " ";
```

```
    cout << endl;
```

```
    m[5] = 200; //key 5의 value를 200으로 갱신한다.
```

```
    for (iter = m.begin(); iter != m.end(); ++iter)
```

```
        cout << "(" << iter->first << ',' << iter->second << ")" << " ";
```

```
    cout << endl;
```

```
}
```

3.MAP

정렬 기준 조건자 **greater** 사용 예제..

```
void main( )
```

```
{
```

```
    // greater 정렬 기준의 key:int, value:string 타입의 컨테이너 m 생성.
```

```
    map<int, string, greater<int>> m;
```

```
        m[5] = "five"; // 원소 추가
```

```
        m[3] = "three";
```

```
        m[8] = "eight";
```

```
        m[4] = "four";
```

```
        m[1] = "one";
```

```
        m[7] = "seven";
```

```
        m[9] = "nine";
```

```
    map<int, string, greater<int>>::iterator iter;
```

```
    for( iter = m.begin() ; iter != m.end() ; ++iter)
```

```
        cout << "(" << iter->first << ',' << iter->second << ")" << " ";
```

```
    cout << endl;
```

```
    cout << m[9] << " "; //key와 매핑된 value를 출력합니다.
```

```
    cout << m[8] << " ";
```

```
    cout << m[7] << " ";
```

```
    cout << m[5] << " ";
```

```
    cout << m[4] << " ";
```

```
    cout << m[3] << " ";
```

```
    cout << m[1] << endl;
```

```
}
```

3.MAP

찾기 관련 멤버 함수 사용 예시..

```
void main( )
{
    map<int,int> m;

    m[5] = 100;
    m[3] = 100;
    m[8] = 30;
    m[4] = 40;
    m[1] = 70;
    m[7] = 100;
    m[9] = 50;

    map<int,int>::iterator iter;
    for( iter = m.begin() ; iter != m.end() ; ++iter)
        cout << "(" << iter->first << ',' << iter->second << ")" << " ";
    cout << endl;

    iter = m.find( 5 );
    if( iter != m.end() )
        cout << "key 5에 매핑된 value: " << iter->second << endl;

    map<int,int>::iterator lower_iter;
    map<int,int>::iterator upper_iter;
    lower_iter = m.lower_bound(5);
    upper_iter = m.upper_bound(5);

    cout << "구간 [lower_iter, upper_iter)의 순차열: ";
    for( iter = lower_iter ; iter != upper_iter ; ++iter)
        cout << "(" << iter->first << ',' << iter->second << ")" << " ";
    cout << endl;

    pair<map<int,int>::iterator, map<int,int>::iterator> iter_pair;
    iter_pair = m.equal_range(5);

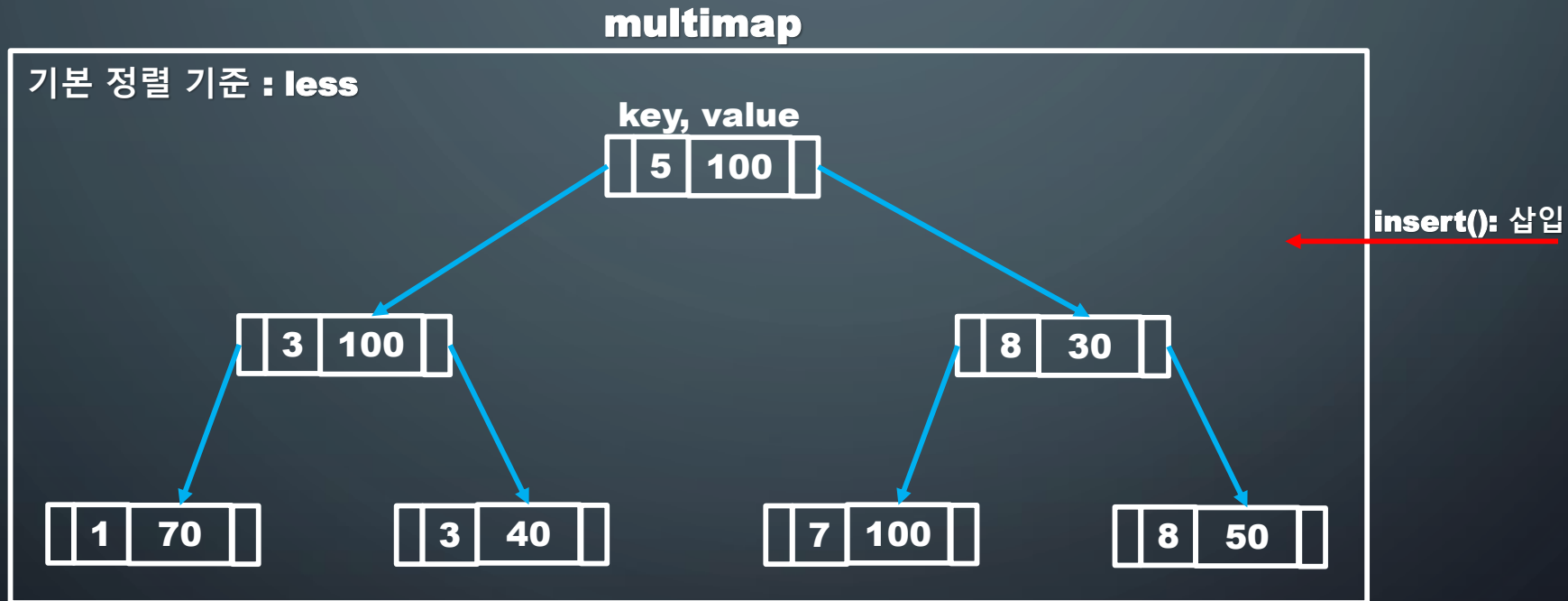
    cout << "구간 [iter_pair.first, iter_pair.second)의 순차열: ";
    for( iter = iter_pair.first ; iter != iter_pair.second ; ++iter)
        cout << "(" << iter->first << ',' << iter->second << ")" << " ";
    cout << endl;
}
```

The image features a dark blue background with a subtle radial gradient. In the four corners, there are decorative white line art elements resembling circuit traces or a stylized network. These elements consist of thin lines connecting small circles, some of which are larger than others, creating a sense of connectivity and technology.

MULTIMAP

4. MULTIMAP

- 템플릿 형식, 인터페이스, 멤버 형식이 모두 **map**과 같다.
- **map**과 다르게 중복**key**가 가능하다.
- 중복이 허용되기 때문에 [] 연산자를 사용 할 수 없다.



4.MULTIMAP

count()와 **find()** 사용 예제..

```
void main()
```

```
{
```

```
    multimap<int, int> mm;
```

```
    mm.insert(pair<int, int>(5, 100));
```

```
    mm.insert(pair<int, int>(3, 100));
```

```
    mm.insert(pair<int, int>(8, 30));
```

```
    mm.insert(pair<int, int>(3, 40));
```

```
    mm.insert(pair<int, int>(1, 70));
```

```
    mm.insert(pair<int, int>(7, 100));
```

```
    mm.insert(pair<int, int>(8, 50));
```

```
    multimap<int, int>::iterator iter;
```

```
    for (iter = mm.begin(); iter != mm.end(); ++iter)
```

```
        cout << "(" << iter->first << ',' << iter->second << ")" << " ";
```

```
    cout << endl;
```

```
    cout << "key 3의 원소의 개수는 " << mm.count(3) << endl;
```

```
    iter = mm.find(3);
```

```
    if (iter != mm.end())
```

```
        cout << "첫 번째 key 3에 매핑된 value: " << iter->second << endl;
```

```
}
```


4.MULTIMAP

lower_bound(), upper_bound(), equal_range() 사용 예제..

```
void main()
{
    multimap<int, int> mm;

    mm.insert(pair<int, int>(5, 100));
    mm.insert(pair<int, int>(3, 100));
    mm.insert(pair<int, int>(8, 30));
    mm.insert(pair<int, int>(3, 40));
    mm.insert(pair<int, int>(1, 70));
    mm.insert(pair<int, int>(7, 100));
    mm.insert(pair<int, int>(8, 50));

    multimap<int, int>::iterator lower_iter;
    multimap<int, int>::iterator upper_iter;
    lower_iter = mm.lower_bound(3);
    upper_iter = mm.upper_bound(3);

    cout << "구간 [lower_iter, upper_iter)의 순차열: ";
    multimap<int, int>::iterator iter;
    for (iter = lower_iter; iter != upper_iter; ++iter)
        cout << "(" << iter->first << ',' << iter->second << ")" ";
    cout << endl;

    pair<multimap<int, int>::iterator, multimap<int, int>::iterator> iter_pair;
    iter_pair = mm.equal_range(3);

    cout << "구간 [iter_pair.first, iter_pair.second)의 순차열: ";
    for (iter = iter_pair.first; iter != iter_pair.second; ++iter)
        cout << "(" << iter->first << ',' << iter->second << ")" ";
    cout << endl;
}
```