< vector 컨테이너 >

템플릿 형식	
Template< typename T , typename Allocator = allocator<T>> class vector	T는 vector 컨테이너 원소의 형식

생성자	
vector v	v는 빈컨테이너
vector v(n)	v는 기본값으로(0) 초기화된 n개의 원소를 갖는다
vector v(n,x)	v는 x 값으로 초기화된 n개의 원소를 갖는다
vector v(v2)	v는 v2 컨테이너의 복사본이다(복사 생성자 호출)
vector v(b,e)	v는 반복자 구간 [b,e)로 초기화된 원소를 갖는다
맴버 함수	
v.assign(n,x)	v에 x 값으로 n개의 원소를 할당한다
v.assign(b,e)	v를 반복자 구간 [b,e)로 할당한다
v.at(i)	v의 i번째 원소를 참조한다 (const, 비 const 버전이 있으며 범위 점검을 포함)
v.back( )	v의 마지막 원소를 참조한다 (const, 비 const 버전이 있음)
p=v.begin()	p는 v의 첫 원소를 가리키는 반복자 (const, 비 const 버전이 있음)
x=v.capacity( )	x는 v에 할당된 공간의 크기
v.clear( )	v의 모든 원소를 제거한다
v.empty( )	v가 비었는지 조사한다
p=v.end( )	p는 v의 끝을 표시하는 반복자 (const, 비 const 버전이 있음)
q=v.erase(p)	p가 가리키는 원소를 제거한다. q는 다음 원소를 가리킨다.
q=v.erase(b,e)	반복자 구간 [b,e]의 모든 원소를 제거한다. q는 다음 원소를 가리킨다.
v.front( )	v의 첫 번째 원소를 참조한다 (const, 비 const 버전이 있음)
q=v.insert(p,x)	p가 가리키는 위치에 x값을 삽입한다. q는 삽입한 원소를 가리키는 반복자다.
v.insert(p, n, x)	p가 가리키는 위치에 n개의 x값을 삽입한다.
v.insert(p, b, e)	p가 가리키는 위치에 반복자 구간 [b,e)의 원소를 삽입한다
x=v.max_size( )	x는 v가 담을 수 있는 최대 원소의 개수다(메모리의 크기)
v.pop_back( )	v의 마지막 원소를 제거한다
v.push_back(x)	v의 끝에 x를 추가한다.
p=v.rbegin( )	p는 v의 역 순차열의 첫 원소를 가리키는 반복자다 (const, 비 const 버전이 있음)

p=v.rend( )	p는 v의 역 순차열의 끝을 표식하는 반복자 (const, 비 const 버전이 있음)
v.reserve(n)	n개의 원소를 저장할 공간을 예약한다.
v.resize(n)	v의 크기를 n으로 변경하고 확장되는 공간의 값을 기본값으로 초기화한다.
v.resize(n,x)	v의 크기를 n으로 변경하고 확장되는 공간의 값을 x으로 초기화한다.
v.size( )	v는 원소의 개수다.
v.swap(v2)	v와 v2를 swap한다
연산자	
V1 == v2	v1과 v2의 모든 원소가 같은가? (bool 형식)
V1 != v2	v1과 v2의 모든 원소 중 하나라도 다른 원소가 있는가? (bool 형식)
V1 < v2	문자열 비교처럼 v2가 v1보다 큰가? (bool 형식)
V1 <= v2	문자열 비교처럼 v2가 v1보다 크거나 같은가? (bool 형식)
V1 > v2	문자열 비교처럼 v2가 v1보다 작은가? (bool 형식)
V1 >= v2	문자열 비교처럼 v2가 v1보다 작거나 같은가? (bool 형식)
v[i]	v의 I 번째 원소를 참조한다 (const, 비 const 버전이 있으며 범위 점검이 없음)
맴버 형식 /* 첨언을 하자면 ex) vector<int>::iterator iter 로 불러서 사용한다. */	
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
pointer	value_type* 형식
reference	value_type& 형식
reverse_iterator	역 반복자 형식
size_type	첨자(index)나 원소의 개수 등의 형식
value_type	원소의 형식

< deque 컨테이너 >

템플릿 형식	
Template< typename T , typename Allocator = allocator<T>> class deque	T는 deque 컨테이너 원소의 형식

생성자	
deque dq	dq는 빈컨테이너
deque dq(n)	dq는 기본값으로(0) 초기화된 n개의 원소를 갖는다
deque dq(n,x)	dq는 x 값으로 초기화된 n개의 원소를 갖는다
deque dq(dq2)	dq는 dq2 컨테이너의 복사본이다(복사 생성자 호출)
deque dq(b,e)	dq는 반복자 구간 [b,e)로 초기화된 원소를 갖는다
맴버 함수	
dq.assign(n,x)	dq에 x 값으로 n개의 원소를 할당한다
dq.assign(b,e)	dq를 반복자 구간 [b,e)로 할당한다
dq.at(i)	dq의 i번째 원소를 참조한다 (const, 비 const 버전이 있으며 범위 점검을 포함)
dq.back( )	dq의 마지막 원소를 참조한다 (const, 비 const 버전이 있음)
p=dq.begin()	p는 dq의 첫 원소를 가리키는 반복자 (const, 비 const 버전이 있음)
dq.clear( )	dq의 모든 원소를 제거한다
dq.empty( )	dq가 비었는지 조사한다
p=dq.end( )	p는 dq의 끝을 표시하는 반복자 (const, 비 const 버전이 있음)
q=dq.erase(p)	p가 가리키는 원소를 제거한다. q는 다음 원소를 가리킨다.
q=dq.erase(b,e)	반복자 구간 [b,e]의 모든 원소를 제거한다. q는 다음 원소를 가리킨다.
dq.front( )	dq의 첫 번째 원소를 참조한다 (const, 비 const 버전이 있음)
q=dq.insert(p,x)	p가 가리키는 위치에 x값을 삽입한다. q는 삽입한 원소를 가리키는 반복자다.
dq.insert(p, n, x)	p가 가리키는 위치에 n개의 x값을 삽입한다.
dq.insert(p, b, e)	p가 가리키는 위치에 반복자 구간 [b,e)의 원소를 삽입한다
x=dq.max_size( )	x는 dq가 담을 수 있는 최대 원소의 개수다(메모리의 크기)
dq.pop_back( )	dq의 마지막 원소를 제거한다
dq.pop_front( )	dq의 첫 원소를 제거한다
dq.push_back(x)	dq의 끝에 x를 추가한다.
dq.push_front(x)	dq의 앞쪽에 x를 추가한다.

p=dq.rbegin( )	p는 dq의 역 순차열의 첫 원소를 가리키는 반복자다 (const, 비 const 버전이 있음)
p=dq.rend( )	p는 dq의 역 순차열의 끝을 표식하는 반복자 (const, 비 const 버전이 있음)
dq.resize(n)	dq의 크기를 n으로 변경하고 확장되는 공간의 값을 기본값으로 초기화한다.
dq.resize(n,x)	dq의 크기를 n으로 변경하고 확장되는 공간의 값을 x으로 초기화한다.
dq.size( )	dq는 원소의 개수다.
dq.swap(dq2)	dq와 dq2를 swap한다
연산자	
dq1 == dq2	dq1과 dq2의 모든 원소가 같은가? (bool 형식)
dq1 != dq2	dq1과 dq2의 모든 원소 중 하나라도 다른 원소가 있는가? (bool 형식)
dq1 < dq2	문자열 비교처럼 dq2가 dq1보다 큰가? (bool 형식)
dq1 <= dq2	문자열 비교처럼 dq2가 dq1보다 크거나 같은가? (bool 형식)
dq1 > dq2	문자열 비교처럼 dq2가 dq1보다 작은가? (bool 형식)
dq1 >= dq2	문자열 비교처럼 dq2가 dq1보다 작거나 같은가? (bool 형식)
dq[i]	dq의 i번째 원소를 참조한다 (const, 비 const 버전이 있으며 범위 점검이 없음)
맴버 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
pointer	value_type* 형식
reference	value_type& 형식
reverse_iterator	역 반복자 형식
size_type	첨자(index)나 원소의 개수 등의 형식
value_type	원소의 형식

< list 컨테이너 >

템플릿 형식	
Template< typename T , typename Allocator = allocator<T>> class list	T는 list 컨테이너 원소의 형식

생성자	
list lt	lt는 빈컨테이너
list lt(n)	lt는 기본값으로(0) 초기화된 n개의 원소를 갖는다
list lt(n,x)	lt는 x 값으로 초기화된 n개의 원소를 갖는다
list lt(lt2)	lt는 lt2 컨테이너의 복사본이다(복사 생성자 호출)
list lt(b,e)	lt는 반복자 구간 [b,e)로 초기화된 원소를 갖는다
맴버 함수	
lt.assign(n,x)	lt에 x 값으로 n개의 원소를 할당한다
lt.assign(b,e)	lt를 반복자 구간 [b,e)로 할당한다
lt.back( )	lt의 마지막 원소를 참조한다 (const, 비 const 버전이 있음)
p=lt.begin()	p는 lt의 첫 원소를 가리키는 반복자 (const, 비 const 버전이 있음)
lt.clear( )	lt의 모든 원소를 제거한다
lt.empty( )	lt가 비었는지 조사한다
p=lt.end( )	p는 lt의 끝을 표시하는 반복자 (const, 비 const 버전이 있음)
q=lt.erase(p)	p가 가리키는 원소를 제거한다. q는 다음 원소를 가리킨다.
q=lt.erase(b,e)	반복자 구간 [b,e]의 모든 원소를 제거한다. q는 다음 원소를 가리킨다.
lt.front( )	dq의 첫 번째 원소를 참조한다 (const, 비 const 버전이 있음)
q=lt.insert(p,x)	p가 가리키는 위치에 x값을 삽입한다. q는 삽입한 원소를 가리키는 반복자다.
lt.insert(p, n, x)	p가 가리키는 위치에 n개의 x값을 삽입한다.
lt.insert(p, b, e)	p가 가리키는 위치에 반복자 구간 [b,e)의 원소를 삽입한다
x=lt.max_size( )	x는 lt가 담을 수 있는 최대 원소의 개수다(메모리의 크기)
lt.merge(lt2)	lt2를 lt로 합병 정렬한다(오름차 순:less) 정렬된 두 list를 하나의 정렬된 list로 합병하므로 합병할 두 list는 정렬되어 있어야 합니다
lt.merge(lt2, pred)	lt2를 lt로 합병 정렬한다. pred(조건자(함수))를 기준으로 합병한다. (pred는 이항 조건자)

lt.pop_back( )	lt의 마지막 원소를 제거한다
lt.pop_front( )	lt의 첫 원소를 제거한다
lt.push_back(x)	lt의 끝에 x를 추가한다.
lt.push_front(x)	lt의 앞쪽에 x를 추가한다.
p=lt.rbegin( )	p는 lt의 역 순차열의 첫 원소를 가리키는 반복자다 (const, 비 const 버전이 있음)
lt.remove(x)	X 원소를 모두 제거한다
lt.remove_if(pred)	pred(단항 조건자)가 '참'인 모든 원소를 제거한다
p=lt.rend( )	p는 lt의 역 순차열의 끝을 표식하는 반복자 (const, 비 const 버전이 있음)
lt.resize(n)	lt의 크기를 n으로 변경하고 확장되는 공간의 값을 기본값으로 초기화한다.
lt.resize(n,x)	lt의 크기를 n으로 변경하고 확장되는 공간의 값을 x으로 초기화한다.
lt.reverse( )	lt의 순차열을 뒤집는다
lt.size( )	lt는 원소의 개수다.
lt.sort( )	lt의 모든 원소를 오름차 순(less)으로 정렬한다.
lt.sort(pred)	lt의 모든 원소를 pred(조건자)를 기준으로 정렬한다. (pred는 이항 조건자)
lt.splice(p, lt2)	p가 가리키는 위치에 lt2의 모든 원소를 잘라 붙인다.
lt.splice(p, lt2, q)	p가 가리키는 위치에 lt2의 q가 가리키는 모든 원소를 잘라 붙인다.
lt.splice(p, lt2,b,e)	p가 가리키는 위치에 lt2의 순차열 [b,e)을 잘라 붙인다.
lt.swap(lt2)	lt와 lt2를 swap한다
lt.unique( )	인접한 원소의 값이 같다면 유일한 원소의 순차열로 만든다
lt.unique(pred)	인접한 원소가 pred(이항 조건자)의 기준에 맞다면 유일한 원소의 순차열로 만든다
연산자	
li1 == il2	li1과 li2의 모든 원소가 같은가? (bool 형식)
li1 != li2	li1과 li2의 모든 원소 중 하나라도 다른 원소가 있는가? (bool 형식)
li1 < li2	문자열 비교처럼 li2가 li1보다 큰가? (bool 형식)
li1 <= li2	문자열 비교처럼 li2가 li1보다 크거나 같은가? (bool 형식)
li1 > li2	문자열 비교처럼 li2가 li1보다 작은가? (bool 형식)
li1 >= li2	문자열 비교처럼 li2가 li1보다 작거나 같은가? (bool 형식)
li[i]	dq의 i번째 원소를 참조한다 (const, 비 const 버전이 있으며 범위 점검이 없음)
맴버 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
pointer	value_type* 형식
reference	value_type& 형식
reverse_iterator	역 반복자 형식
size_type	첨자(index)나 원소의 개수 등의 형식
value_type	원소의 형식

< set 컨테이너 >

템플릿 형식	
Template< typename Key , typename pred=less<Key>, typename Allocator = allocator<T> > class set	Key는 set 컨테이너 원소의 형식이며, Pred는 set의 정렬 기준인 조건자이다. 기본 조건자는 less이다.

생성자	
set s	s는 빈컨테이너
set s(pred)	s는 빈 컨테이너로 정렬 기준은 pred 조건자 사용한다.
set s(s2)	s는 s2 컨테이너의 복사본이다(복사 생성자 호출)
set s(b,e)	s는 반복자 구간 [b,e)로 초기화된 원소를 갖는다
set s(b,e,pred)	s는 반복자 구간 [b,e)로 초기화된 원소를 갖는다. 정렬 기준은 pred 조건자를 사용한다
맴버 함수	
p=s.begin()	p는 s의 첫 원소를 가리키는 반복자 (const, 비 const 버전이 있음)
s.clear( )	s의 모든 원소를 제거한다
n=s.count(k)	원소 k의 개수를 반환한다
s.empty( )	s가 비었는지 조사한다
p=s.end( )	p는 s의 끝을 표시하는 반복자 (const, 비 const 버전이 있음)
pr=s.equal_range(k)	pr은 k 원소의 반복자 구간인 pair 객체다. (const, 비 const 버전이 있음)
q=lt.erase(p)	p가 가리키는 원소를 제거한다. q는 다음 원소를 가리킨다.
q=lt.erase(b,e)	반복자 구간 [b,e]의 모든 원소를 제거한다. q는 다음 원소를 가리킨다.
n=s.erase(k)	K 원소를 모두 제거한다. n은 제거한 개수다
p=s.find(k)	p는 k 원소의 위치를 가리키는 반복자다. (const, 비 const 버전이 있음) 만약 원소가 없으면 끝 표시 (past-the-end) 반복자를 반환합니다.
pr=s.insert(k)	S 컨테이너에 k를 삽입한다. pr은 삽입한 원소를 가리키는 반복자와 성공 여부의 bool 값은 pair 객체다
s.insert(b, e)	반복자 구간 [b,e)의 원소를 삽입한다
Pred = s.key_comp( )	pred는 s의 key 정렬 기준인 조건자다(key_compare 타입)
p=s.lower_bound(k)	p는 k의 시작 구간를 가리키는 반복자다. (const, 비 const 버전이 있음)

n=s.max_size( )	n는 s가 담을 수 있는 최대 원소의 개수다(메모리의 크기)
p=s.rbegin( )	p는 s의 역 순차열의 첫 원소를 가리키는 반복자다 (const, 비 const 버전이 있음)
p=s.rend( )	p는 s의 역 순차열의 끝을 표식하는 반복자다 (const, 비 const 버전이 있음)
s.size( )	S 원소의 개수다
s.swap(s2)	s와 s2를 swap한다
p=s.upper_bound(k)	p는 k의 끝 구간을 가리키는 반복자다 (const, 비 const 버전이 있음)
Pred = s.value_comp( )	pred는 s의 value 정렬 기준인 조건자다. (value_compare 타입)
연산자	
s1 == s2	s1과 s2의 모든 원소가 같은가? (bool 형식)
s1 != s2	s1과 s2의 모든 원소 중 하나라도 다른 원소가 있는가? (bool 형식)
s1 < s2	문자열 비교처럼 s2가 s1보다 큰가? (bool 형식)
s1 <= s2	문자열 비교처럼 s2가 s1보다 크거나 같은가? (bool 형식)
s1 > s2	문자열 비교처럼 s2가 s1보다 작은가? (bool 형식)
s1 >= s2	문자열 비교처럼 s2가 s1보다 작거나 같은가? (bool 형식)
맴버 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(set은 key가 value이므로 value_compare와 같음)
key_type	키(key)의 형식(set은 key가 value이므로 value_compare와 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

< multiset 컨테이너 >

생성자	
multset s	s는 빈컨테이너
multset s(pred)	s는 빈 컨테이너로 정렬 기준은 pred 조건자 사용한다.
multset s(s2)	s는 s2 컨테이너의 복사본이다(복사 생성자 호출)
multset s(b,e)	s는 반복자 구간 [b,e)로 초기화된 원소를 갖는다
multset s(b,e,pred)	s는 반복자 구간 [b,e)로 초기화된 원소를 갖는다. 정렬 기준은 pred 조건자를 사용한다

시퀀스 컨테이너와 달리 모든 연관 컨테이너 (set, multset, map, multimap)은 같은 인터페이스(생성자, 멤버함수, 연산자)를 제공하므로 set에 정리해둔 표를 참고해라

< map 컨테이너 >

템플릿 형식	
Template< typename Key , typename pred=less<Key>, typename Allocator = allocator<pair<const Key,Value>> > class map	Key는 value는 map 컨테이너 원소의 key와 value의 형식이다, Pred는 map의 정렬 기준인 조건자이다. 기본 조건자는 less이다.

나머진 set와 같다	
연산자	
m[k] = v	m컨테이너에 원소(k,v)를 추가하거나 key에 해당하는 원소의 value를 v로 갱신한다.

시퀀스 컨테이너와 달리 모든 연관 컨테이너 (set, multset, map, multimap)은 같은 인터페이스(생성자, 멤버함수, 연산자)를 제공하므로 set에 정리해둔 표를 참고해라

< multimap 컨테이너 >

< 원소를 수정하지 않는 알고리즘 >

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
p = adjacent_find(b,e)	p는 구간 [b,e)의 원소 중 최초로 연속되서 나타나는 같은 원소를 찾는다.
p = adjacent_find(b,e,f)	p는 구간 [b,e)의 원소 중 f(*p,*(p+1))이 참인 첫 원소를 찾는다.
n = count(b,e,x)	n은 구간 [b,e)의 원소 중 x 원소의 개수
n = count_if(b,e,f)	n은 구간 [b,e)의 원소 중 f(*p)가 참인 원소의 개수
equal(b,e,b2)	[b,e)와 [b2, b2+(e-b))의 모든 원소가 같은가?
equal(b,e,b2,f)	[b,e)와 [b2, b2+(e-b))의 모든 원소가 f(*p, *q)가 참인가?
p=find(b,e,x)	p는 구간 [b,e)에서 x와 같은 첫 원소의 반복자
p=find_end(b,e,b2,e2)	p는 구간 [b,e)의 순차열 중 구간 [b2,e2)의 순차열과 일치하는 순차열 첫 원소의 반복자. 단 [b2,e2)와 일치하는 순차열이 여러 개라면 마지막 순차열 첫 원소의 반복자
p=find_end(b,e,b2,e2,f)	p는 구간 [b,e)의 순차열 중 구간 [b2,e2)의 순차열과 일치하는 순차열 첫 원소의 반복자. 단 [b2,e2)와 일치하는 순차열이 여러 개라면 마지막 순차열 첫 원소의 반복자. 이때 비교는 f를 사용
p=find_first_of(b,e,b2,e2)	p는 구간 [b,e)에서 구간 [b2,e2)의 원소 중 같은 원소가 발견된 첫 원소의 반복자
p=find_first_of(b,e,b2,e2,f)	p는 구간 [b,e)에서 구간 [b2,e2)의 원소 중 같은 원소가 발견된 첫 원소의 반복자. 비교는 f를 사용
p=find_if(b,e,f)	p는 구간 [b,e)에 f(*p)가 참인 첫 원소를 가리키는 반복자
f=for_each(b,e,f)	구간 [b,e)의 모든 원소에 f(*p) 동작을 적용한다. f를 다시 되돌려준다.
lexicographical_compare (b,e,b2,e2)	구간 [b,e)의 순차열이 구간 [b2,e2)의 순차열보다 작다면 true,아니면 false를 반환한다. 이때 작음은 사전순이다
lexicographical_compare (b,e,b2,e2)	구간 [b,e)의 순차열이 구간 [b2,e2)의 순차열보다 작다면 true,아니면 false를 반환한다. 이때 작음은 [b,e)의 반복자 p와 [b2,e2)의 반복자 q에 대해 f(*p,*q)가 참이다.
k=max(a,b)	k는 a와 b중 더 큰것
k=max(a,b,f)	k는 a와 b중 더 큰것 이때 큰 것은 f(a,b)를 사용
p=max_element(b,e)	p는 구간 [b,e)에서 가장 큰 원소의 반복자
p=max_element(b,e)	p는 구간 [b,e)에서 가장 큰 원소의 반복자 이때 비교는 f를 사용
k=min(a,b)	k는 a와 b중 더 작은것
k=min(a,b,f)	k는 a와 b중 더 작은것 이때 작은 것은 f(a,b)를 사용
p=min_element(b,e)	p는 구간 [b,e)에서 가장 작은 원소의 반복자
p=min_element(b,e)	p는 구간 [b,e)에서 가장 작은 원소의 반복자 이때 비교는 f를 사용

pair(p,q) = mismatch(b,e,b2)	(p,q)는 구간 [b,e)와 [b2,b2+(e-b))에서 !(*p==*q)인 첫 원소를 가리키는 반복자의 쌍
pair(p,q) = mismatch(b,e,b2,f)	(p,q)는 구간 [b,e)와 [b2,b2+(e-b))에서 !f(*p,*q)인 첫 원소를 가리키는 반복자의 쌍
p=search(b,e,b2,e2)	p는 구간 [b,e)의 순차열 중 구간 [b2,e2)의 순차열과 일치하는 순차열 첫 원소의 반복자(find_end()와 비슷하나 find_end()는 일치하는 순차열의 마지막 순차열의 반복자)
p=search(b,e,b2,e2,f)	p는 구간 [b,e)의 순차열 중 구간 [b2,e2)의 순차열과 일치하는 순차열 첫 원소의 반복자(find_end()와 비슷하나 find_end()는 일치하는 순차열의 마지막 순차열의 반복자) 이때 비교는 f를 사용
p=search_n(b,e,n,x)	p는 구간 [b,e)의 원소 중 x 값이 n개 연속한 첫 원소의 반복자
p=search_n(b,e,n,x,f)	p는 구간 [b,e)의 원소 중 f(*p,x)가 참인 값이 n개 연속한 첫 원소의 반복자

< 원소를 수정하는 알고리즘 >

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
p=copy(b,e,t)	구간 [b,e)의 원소를 [t,p)로 모두 복사한다
p=copy_backward(b,e,t)	구간 [b,e)의 원소를 마지막 원소부터 [p,t)로 모두 복사한다
fill(b,e,x)	구간 [b,e)의 모든 원소를 x로 채운다
fill_n(b,n,x)	구간 [b,b+n)의 모든 원소를 x로 채운다
f=for_each(b,e,f)	구간 [b,e)의 모든 원소에 f(*p)동작을 적용한다. f를 다시 되돌려준다.
generate(b,e,f)	구간 [b,e)의 모든 원소를 f( )로 채운다.
generate_n(b,n,f)	구간 [b,b+n)의 모든 원소를 f( )로 채운다.
iter_swap(p,q)	반복자 p,q,가 가리키는 *p와 *q의 원소를 교환한다
p=merge(b,e,b2,e2,t)	정렬된 순차열 [b,e)과 [b2,e2)를 [t,p)로 합병 정렬한다.
p=merge(b,e,b2,e2,t,f)	정렬된 순차열 [b,e)과 [b2,e2)를 [t,p)로 합병 정렬한다. 이때 비교는 f를 사용한다.
replace(b,e,x,x2)	구간 [b,e)의 x인 원소를 x2로 수정한다.
replace_if(b,e,f,x2)	구간 [b,e)의 f(*p)가 참인 원소를 x2로 수정한다.
P = replace_copy(b,e,t,x,x2)	구간 [b,e)의 x인 원소를 x2로 수정하여 [t,p)로 복사한다
P = replace_copy_if (b,e,t,f,x2)	구간 [b,e)의 f(*p)가 참인 원소를 x2로 수정하여 [t,p)로 복사한다
swap(a,b)	a와 b를 교환한다.
swap_ranges(b,e,b2)	구간 [b,e)의 원소와 구간 [b2,b2+(e-b))의 원소를 교환한다
p=transform(b,e,t,f)	구간 [b,e)의 모든 원소를 f(*p)하여 [t,t+(e-b))에 저장한다. p는 저장된 마지막 원소의 반복자(t+(e-b))다.
p=transform(b,e,b2,t,f)	구간 [b,e)과 [b2,b2+(e-b))의 두 순차열의 반복자 p,q일때 모든 원소를 f(*p,*q)하여 [t,t+(e-b))에 저장한다. p는 저장된 마지막 원소의 반복자(t+(e-b))다

< 제거 알고리즘 >

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
p=remove(b,e,x)	구간 [b,e)의 순차열을 x원소가 남지 않도록 덮어쓰기로 이동한다. 알고리즘 수행 후 순차열은 [b,p)가 된다. 단, 논리적으로 제거한다
p=remove_if(b,e,f)	구간 [b,e)의 순차열을 f(*p)가 참인 원소가 남지 않도록 덮어쓰기로 이동한다. 알고리즘 수행 후 순차열은 [b,p)가 된다. 단, 논리적으로 제거한다
p=remove_copy(b,e,t,x)	구간 [b,e)의 순차열에서 *p==x가 아닌 원소만 순차열 [t,p)에 복사한다.
p=remove_copy_if(b,e,t,f)	구간 [b,e)의 순차열에서 f(*p)가 참이 아닌 원소만 순차열 [t,p)에 복사한다.
p=unique(b,e)	구간 [b,e)의 순차열을 인접한 중복 원소(값이 같은 원소:==)가 남지 않게 덮어쓰기로 이동한다. 알고리즘 수행 후 순차열을 [b,p)가 된다. 단, 논리적으로 제거한다
p=unique(b,e,f)	구간 [b,e)의 순차열을 f(*p)가 참인 인접한 중복 원소(값이 같은 원소:==)가 남지 않게 덮어쓰기로 이동한다. 알고리즘 수행 후 순차열을 [b,p)가 된다. 단, 논리적으로 제거한다
p=unique_copy(b,e,t)	구간 [b,e)의 순차열에서 인접한 중복 원소(값이 같은 원소:==)가 아닌 원소를 순차열 [t,p)에 복사한다.
p=unique_copy(b,e,t,f)	구간 [b,e)의 순차열에서 f(*p)가 참인 인접한 중복 원소가 아닌 원소를 순차열 [t,p)에 복사한다.

< 변경 알고리즘 >

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
bool=next_permutation(b,e)	구간 [b,e)의 순차열을 사전순 다음 순열이 되게 한다. 더 이상 다음 순열이 없는 마지막 순열이라면 bool은 false이다.
bool=next_permutation(b,e,f)	구간 [b,e)의 순차열을 사전순 다음 순열이 되게 한다. 비교에 f를 사용한다. 더 이상 다음 순열이 없는 마지막 순열이라면 bool은 false이다.
bool=prev_permutation(b,e)	구간 [b,e)의 순차열을 사전순 이전 순열이 되게 한다. 더 이상 이전 순열이 없는 첫 순열이라면 bool은 false이다.
bool=prev_permutation(b,e,f)	구간 [b,e)의 순차열을 사전순 이전 순열이 되게 한다. 비교에 f를 사용한다. 더 이상 이전 순열이 없는 첫 순열이라면 bool은 false이다.
p=partition(b,e,f)	구간 [b,e)의 순차열 중 f(*p)가 참인 원소는 [b,p)의 순차열에 거짓인 원소는 [p,e)의 순차열로 분류한다.
random_shuffle(b,e)	구간 [b,e)의 순차열을 랜덤(기본 랜덤기)으로 뒤섞는다
random_shuffle(b,e,f)	구간 [b,e)의 순차열을 f를 랜덤기로 사용해서 뒤섞는다
reverse(b,e)	구간 [b,e)의 순차열을 뒤집는다
P = reverse_copy(b,e,t)	구간 [b,e)의 순차열을 뒤집어 목적지 순차열 [t,p)에 복사한다
rotate(b,m,e)	구간 [b,e)의 순차열을 왼쪽으로 회전시킨다. 첫 원소와 마지막 원소가 연결된 것처럼 모든 원소가 왼쪽으로 (m-b)만큼씩 이동한다
p=rotate_copy(b,m,e,t)	구간 [b,e)의 순차열을 회전하여 목적지 순차열 [t,p)에 복사한다
stable_partition(b,e,f)	partition( )알고리즘과 같고 원소의 상대적인 순서를 유지한다

< 정렬 알고리즘 >

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
P = partition(b,e,f)	구간 [b,e)의 순차열 중 f(*p)가 참인 원소는 [b,q)의 순차열에 거짓인 원소는 [p,e)의 순차열로 분류한다 (변경 알고리즘에도 속하므로 변경 알고리즘 참고)
stable_partition(b,e,f)	Partition() 알고리즘과 같고 원소의 상대적인 순서를 유지한다(변경 알고리즘에도 속하므로 변경 알고리즘 참고)
make_heap(b,e)	힙을 생성한다. 구간 [b,e)의 순차열을 힙 구조로 변경한다
make_heap(b,e,f)	힙을 생성한다. 구간 [b,e)의 순차열을 힙 구조로 변경한다. f는 조건자로 비교에 사용한다.
push_heap(b,e)	힙에 원소를 추가한다. 보통 push_back() 맴버 함수와 같이 사용되며, 구간 [b,e)의 순차열을 다시 힙 구조가 되게 한다.
push_heap(b,e,f)	힙에 원소를 추가한다. 보통 push_back() 맴버 함수와 같이 사용되며, 구간 [b,e)의 순차열을 다시 힙 구조가 되게 한다. f는 조건자로 비교에 사용한다.
pop_heap(b,e)	힙에서 원소를 제거한다. 구간 [b,e)의 순차열의 가장 큰 원소(첫 원소)를 제거한다
pop_heap(b,e,f)	힙에서 원소를 제거한다. 구간 [b,e)의 순차열의 가장 큰 원소(첫 원소)를 제거한다 f는 조건자로 비교에 사용한다.
sort_heap(b,e)	힙을 정렬한다. 구간[b,e)의 순차열을 힙 구조를 이용해 정렬한다
sort_heap(b,e,f)	힙을 정렬한다. 구간[b,e)의 순차열을 힙 구조를 이용해 정렬한다 f는 조건자로 비교에 사용한다
nth_element(b,m,e)	구간 [b,e)의 원소 중 m-b개 만큼 선별된 원소를 [b,m) 순차열에 놓이게 한다.
nth_element(b,m,e,f)	구간 [b,e)의 원소 중 m-b개 만큼 선별된 원소를 [b,m) 순차열에 놓이게 한다. f는 조건자로 비교에 사용한다
sort(b,e)	퀵 정렬을 기반으로 정렬한다. 구간 [b,e)를 정렬한다
sort(b,e,f)	퀵 정렬을 기반으로 정렬한다. 구간 [b,e)를 정렬한다 f는 조건자로 비교에 사용한다
stable_sort(b,e)	머지 정렬을 기반으로 정렬한다. 구간 [b,e)를 정렬하되 값이 같은 원소의 상대적인 순서를 유지한다.
stable_sort(b,e,f)	머지 정렬을 기반으로 정렬한다. 구간 [b,e)를 정렬하되 값이 같은 원소의 상대적인 순서를 유지한다. f는 조건자로 비교에 사용한다
partial_sort(b,m,e)	힙 정렬을 기반으로 정렬한다. 구간 [b,e)의 원소 중 m-b개 만큼의 상위 원소를 정렬하여 [b,m) 순차열에 놓는다.

partial_sort(b,m,e,f)	힙 정렬을 기반으로 정렬한다. 구간 [b,e)의 원소 중 m-b개 만큼의 상위 원소를 정렬하여 [b,m) 순차열에 놓는다. f는 조건자로 비교에 사용한다
partial_sort_copy(b,e,b2,e2)	힙 정렬을 기반으로 정렬한다. 구간 [b,e)의 원소 중 상위 e2-b2개의 원소 정도만 정렬하여 [b2,e2)로 복사한다
partial_sort_copy(b,e,b2,e2,f)	힙 정렬을 기반으로 정렬한다. 구간 [b,e)의 원소 중 상위 e2-b2개의 원소 정도만 정렬하여 [b2,e2)로 복사한다 f는 조건자로 비교에 사용한다

< 정렬된 범위 알고리즘 >

주의 : 정렬된 범위 알고리즘은 입력 순차열이 반드시 정렬돼 있어야 합니다.

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자) 출력이 정해지지 않았다는 가정 하에, 여기에 있는 모든 결과는 있다면 true 없다면 false 반환한다.
binary_search(b,e,x)	구간 [b,e)의 순차열에 x와 같은 원소가 있는가?
binary_search(b,e,x,f)	구간 [b,e)의 순차열에 x와 같은 원소가 있는가? f는 비교에 사용한다
includes(b,e,b2,e2)	구간 [b2,e2)의 모든 원소가 구간 [b,e)에도 있는가?
includes(b,e,b2,e2,f)	구간 [b2,e2)의 모든 원소가 구간 [b,e)에도 있는가? f는 비교에 사용한다
p=lower_bound(b,e,x)	p는 구간 [b,e)의 순차열에서 x와 같은 첫 원소의 반복자다
p=lower_bound(b,e,x,f)	p는 구간 [b,e)의 순차열에서 x와 같은 첫 원소의 반복자다 f는 비교에 사용한다
p=upper_bound(b,e,x)	p는 구간 [b,e)의 순차열에서 x보다 큰 첫 원소의 반복자다
p=upper_bound(b,e,x,f)	p는 구간 [b,e)의 순차열에서 x보다 큰 첫 원소의 반복자다 f는 비교에 사용한다
pair(p1,p2)=equal_range(b,e,x)	구간 [p1,p2)의 순차열은 구간 [b,e)의 순차열에서 x와 같은 원소의 구간(순차열)이다. [lower_bound( ), upper_bound( ))의 순차열과 같다
pair(p1,p2)=equal_range(b,e,x,f)	구간 [p1,p2)의 순차열은 구간 [b,e)의 순차열에서 x와 같은 원소의 구간(순차열)이다. [lower_bound( ), upper_bound( ))의 순차열과 같다 f는 비교에 사용한다
p=merge(b,e,b2,e2,t)	구간 [b,e)의 순차열과 구간 [b2,e2)의 순차열을 합병해 [t,p)에 저장한다.
p=merge(b,e,b2,e2,t,f)	구간 [b,e)의 순차열과 구간 [b2,e2)의 순차열을 합병해 [t,p)에 저장한다. f는 비교에 사용한다
inplace_merge(b,m,e)	정렬된 [b,m) 순차열과 [m,e)순차열을 [b,e) 순차열로 합병'정렬'한다
inplace_merge(b,m,e,f)	정렬된 [b,m) 순차열과 [m,e)순차열을 [b,e) 순차열로 합병'정렬'한다 f는 비교에 사용한다
p=set_union(b,e,b2,e2,t)	구간 [b,e)의 순차열과 [b2,e2)의 순차열을 정렬된 합집합으로 [t,p)에 저장한다.
p=set_union(b,e,b2,e2,t,f)	구간 [b,e)의 순차열과 [b2,e2)의 순차열을 정렬된 합집합으로 [t,p)에 저장한다. f는 비교에 사용한다
p=set_difference(b,e,b2,e2,t)	구간 [b,e)의 순차열과 [b2,e2)의 순차열을 정렬된 차집합으로 [t,p)에 저장한다.

p=set_difference(b,e,b2,e2,t,f)	구간 [b,e)의 순차열과 [b2,e2)의 순차열을 정렬된 차집합으로 [t,p)에 저장한다. f는 비교에 사용한다
p=set_symmetric_difference(b,e,b2,e2,t)	구간 [b,e)의 순차열과 [b2,e2)의 순차열을 정렬된 대칭 차집합으로 [t,p)에 저장한다.
p=set_symmetric_difference(b,e,b2,e2,t,f)	구간 [b,e)의 순차열과 [b2,e2)의 순차열을 정렬된 대칭 차집합으로 [t,p)에 저장한다. f는 비교에 사용한다

< 수치 알고리즘 >

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자) X를 초기값으로 시작한다는 이야기는 x + '[b,e) 의 모든 원소의 계산' 을 의미하는 것이다
x2=accumulate(b,e,x)	x2는 x를 초깃값으로 시작한 구간 [b,e) 순차열 원소의 합이다.
x2=accumulate(b,e,x,f)	x2는 x를 초깃값으로 시작한 구간 [b,e) 순차열 원소의 합이다. f를 누적에 사용한다.
x2=inner_product(b,e,b2,x)	x2는 x를 초깃값으로 시작한 구간 [b,e)와 구간 [b2,b2+e-b)의 내적(두 순차열의 곱의 합)이다
x2=inner_product(b,e,b2,x,f1,f2)	x2는 x를 초깃값으로 시작한 구간 [b,e)와 구간 [b2,b2+e-b)의 모든 원소끼리 f2연산 후 f1연산으로 총 연산한 결과다
p=adjacent_difference(b,e,t)	구간 [b,e)의 인접 원소와의 차를 순차열 [t,p)에 저장한다.
p=adjacent_difference(b,e,t,f)	구간 [b,e)의 인접 원소와의 차를 순차열 [t,p)에 저장한다. f를 연산에 사용한다.
p=partial_sum(b,e,t)	구간 [b,e)의 인접 원소와의 합을 순차열 [t,p)에 저장한다.
p=partial_sum(b,e,t,f)	구간 [b,e)의 인접 원소와의 합을 순차열 [t,p)에 저장한다. f를 연산에 사용한다.

| 산술 연산 함수 객체

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
plus<T>	이항 연산 함수자로 +연산
minus<T>	이항 연산 함수자로 - 연산
muliplies<T>	이항 연산 함수자로 *연산
divides<T>	이항 연산 함수자로 /연산
modulus<T>	이항 연산 함수자로 %연산
negate<T>	단항 연산 함수자로 -연산

| 비교 연산 조건자

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
equal_to<T>	이항 조건자로 == 연산
not_equal_to<T>	이항 조건자로 != 연산
less<T>	이항 조건자로 < 연산
less_equal<T>	이항 조건자로 <= 연산
greater<T>	이항 조건자로 > 연산
greater_equal<T>	이항 조건자로 >= 연산

| 논리 연산 조건자

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
logical_and<T>	이항 조건자로 && 연산
logical_or<T>	이항 조건자로 || 연산
logical_not<T>	이항 조건자로 ! 연산

논리 조건자는 일반적으로 T에 bool형식을 지정합니다. 일반적으로 피연산자의 논리 조건을 비교하기 때문입니다.

ex)logical_and<bool>( ) (greater<int>() (n,10), less<int>() (n,50))

| 바인더

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
bind1st	이항 함수자의 첫 번째 인자를 고정하여 단항 함수자로 변환
bind2nd	이항 함수자의 두 번째 인자를 고정하여 단항 함수자로 변환

바인더는 이항 함수자를 단항 함수자로 변환한다.

| 부정자

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
not1	단항 조건자를 반대의 조건자로 변환
not2	이항 조건자를 반대의 조건자로 변환

| 함수 포인터 어댑터

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
ptr_fun(pred)	함수 포인터 어댑터는 일반 함수를 어댑터 적용이 가능한 함수 객체로 변환합니다.

| 멤버 함수 포인터 어댑터

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
mem_sun_ref( )	객체로 맴버 함수를 호출합니다
mem_fun( )	객체의 주소로 맴버 함수를 호출합니다

< 반복자 >

반복자 종류	사용 방식	읽기	접근	쓰기	증감	비교
입력 반복자 (input iterator)	istream_iterator	=*p	->		++	== !=
출력 반복자 (output iterator)	ostream_iterator inserter front_inserter back_inserter			*p=	++	
순방향 반복자 (forward iterator)		=*p	->	*p=	++	== !=
양방향 반복자 (bidirectional iterator)	list set 과 multiset map 과 multimap	=*p	->	*p=	++ --	== !=
임의접근 반복자 (random access iterator)	일반 포인터 vector deque	=*p	-> []	*p=	++ -- + - += -=	== != < > <= >=

알고리즘	설명(설명에 사용되는 p는 구간 [b,e)의 반복자)
X::iterator	정방향 반복자의 내장 형식, 반복자가 가리키는 원소 읽기 쓰기 가능
X::const_iterator	정방향 반복자의 내장 형식, 반복자가 가리키는 원소 읽기만 가능 반복자가 가리키는 원소의 위치 변경 불가
X::reverse_iterator	역방향 반복자의 내장 형식, 반복자가 가리키는 원소 읽기, 쓰기 가능
X::const_reverse_iterator	역방향 반복자의 내장 형식, 반복자가 가리키는 원소 읽기만 가능
inserter( )	insert_iterator객체를 생성합니다. insert_iterator객체는 컨테이너의 insert( ) 맴버 함수를 호출해 삽입 모드로 동작하게 합니다 template < class Container > std:: insert_iterator < Container > inserter (Container & c, typename Container :: iterator I) ;
back_inserter( )	back_insert_iterator객체를 생성합니다. back_insert_iterator객체는 컨테이너의 push_back( ) 맴버 함수를 호출해 뒤쪽에 추가(삽입) 하도록 합니다
front_inserter( )	front_insert_iterator객체를 생성합니다. front_insert_iterator객체는 컨테이너의 push_front( ) 맴버 함수를 호출해 앞쪽에 추가(삽입) 하도록 합니다
istream_iterator<T>	입력 스트림과 연결된 반복자로 T 형식의 값을 스트림에서 읽을 수 있습니다.
ostream_iterator<T>	출력 스트림과 연결된 반복자로 T 형식의 값을 스트림에서 읽을 수 있습니다.
advance(p, n)	p반복자를 p+=n 위치로 이동시킵니다
n=distance(p1,p2)	n은 p2-p1입니다

< 컨테이너 어댑터 >

컨테이너 어댑터는 다른 컨테이너의 인터페이스를 변경한 컨테이너입니다. STL에서는 stack, queue, priority_queue 세 가지 컨테이너 어댑터가 있습니다.

1. stack 컨테이너

템플릿 형식	
Template< typename T , typename Container = deque<T>> class stack	T는 원소의 형식이며, container는 stack에서 사용될 컨테이너 형식이며 기본 컨테이너는 deque<T>이다.

생성자	
explicit stack(const Container& = Container( ))	컨테이너의 기본 생성자를 호출해 stack을 생성하거나 인자로 받아 stack을 생성한다
맴버 함수	
bool empty( ) const	원소가 없는가?
size_type size( ) const	원소의 개수다.
void push(const value_type& x)	원소를 추가한다
void pop( )	원소를 제거한다
value_type& top()	Top 원소를 참조한다
const value_type& top ( ) const	Const 객체 Top원소를 참조한다
연산자	
s1 == s2	s1과 s2의 모든 원소가 같은가? (bool 형식)
s1 != s2	s1과 s2의 모든 원소 중 하나라도 다른 원소가 있는가? (bool 형식)
s1 < s2	문자열 비교처럼 s2가 s1보다 큰가? (bool 형식)
s1 <= s2	문자열 비교처럼 s2가 s1보다 크거나 같은가? (bool 형식)
s1 > s2	문자열 비교처럼 s2가 s1보다 작은가? (bool 형식)
s1 >= s2	문자열 비교처럼 s2가 s1보다 작거나 같은가? (bool 형식)
맴버 형식	
value_type	Container::value_type으로 T형식이다
size_type	Container::size_type으로 첨자(index)나 원소 개수 등의 형식이다.
container_type	Container 형식이며, 기본 형식은 deque<T>이다.

2. queue 컨테이너

템플릿 형식	
Template< typename T , typename Container = deque<T>> class queue	T는 원소의 형식이며, container는 queue에서 사용될 컨테이너 형식이며 기본 컨테이너는 deque<T>이다.

생성자	
explicit queue(const Container& = Container( ))	컨테이너의 기본 생성자를 호출해 queue을 생성하거나 인자로 받아 queue을 생성한다
맴버 함수	
bool empty( ) const	원소가 없는가?
size_type size( ) const	원소의 개수다.
void push(const value_type& x)	원소를 추가한다
void pop( )	원소를 제거한다
value_type& front( )	첫 원소를 참조한다
const value_type& front( ) const	Const 객체 첫원소를 참조한다
value_type& top( )	마지막 원소를 참조한다
const value_type& top ( ) const	Const 객체 마지막원소를 참조한다
연산자	
s1 == s2	s1과 s2의 모든 원소가 같은가? (bool 형식)
s1 != s2	s1과 s2의 모든 원소 중 하나라도 다른 원소가 있는가? (bool 형식)
s1 < s2	문자열 비교처럼 s2가 s1보다 큰가? (bool 형식)
s1 <= s2	문자열 비교처럼 s2가 s1보다 크거나 같은가? (bool 형식)
s1 > s2	문자열 비교처럼 s2가 s1보다 작은가? (bool 형식)
s1 >= s2	문자열 비교처럼 s2가 s1보다 작거나 같은가? (bool 형식)
맴버 형식	
value_type	Container::value_type으로 T형식이다
size_type	Container::size_type으로 첨자(index)나 원소 개수 등의 형식이다.
container_type	Container 형식이며, 기본 형식은 deque<T>이다.

3. priority_queue 컨테이너

템플릿 형식	
Template< typename T , typename Container = vector<T>, typename comp=less<typename container::value_type>> class queue	T는 원소의 형식이며, container는 priority_queue에서 사용될 컨테이너 형식이며 기본 컨테이너는 vector<T>이다. comp는 우선순위를 결정할 정렬 기준이며 기본 정렬 기준은 less<T>이다

생성자	
explicit priority_queue(const comp& = comp( ), const Container& = Container( ))	컨테이너의 기본 생성자를 호출해 priority_queue을 생성하거나 인자로 받아 priority_queue을 생성한다
맴버 함수	
bool empty( ) const	원소가 없는가?
size_type size( ) const	원소의 개수다.
void push(const value_type& x)	원소를 추가한다
void pop( )	원소를 제거한다
value_type& top( )	Top 원소를 참조한다
const value_type& top ( ) const	Const 객체 Top원소를 참조한다
맴버 형식	
value_type	Container::value_type으로 T형식이다
size_type	Container::size_type으로 첨자(index)나 원소 개수 등의 형식이다.
container_type	Container 형식이며, 기본 형식은 deque<T>이다.

< string 컨테이너 >

string처럼 C++ 표준 컨테이너의 요구사항을 모두 만족하지 않는 컨테이너를 유사 컨테이너 라고 합니다

생성자	
string s	기본 생성자로 s를 생성
string s(sz)	sz 문자열로 s를 생성
string s(sz,n)	sz문자열에서 n개의 문자로 s를 생성
string s(n,c)	n개의 c문자로 s를 생성
string s(iter1, iter2)	반복자 구간 [iter1,iter2)의 문자로 s를 생성
string s(p1, p2)	포인터 구간 [p1,p2)의 문자로 s를 생성
맴버 함수	
s.append(sz)	s에 sz를 붙임
s.assign(sz)	s에 sz문자열을 할당
s.at(i)	s의 i번째 원소를 참조한다 (const, 비 const 버전이 있으며 범위 점검을 포함)
p=s.begin()	p는 s의 첫 문자를 가리키는 반복자 (const, 비 const 버전이 있음)
s.c_str( )	C스타일(널문자를 포함한)의 문자열의 주소를 반환
n=s.capacity( )	n는 s에 할당된 메모리 공간의 크기
s.clear( )	s의 모든 원소를 제거한다
s.compare(s2)	s와 s2를 비교
s.empty( )	s가 비었는지 조사한다
p=s.end( )	p는 s의 끝을 표시하는 반복자 (const, 비 const 버전이 있음)
q=s.erase(p)	p가 가리키는 원소를 제거한다. q는 다음 원소를 가리킨다.
q=s.erase(b,e)	반복자 구간 [b,e]의 모든 원소를 제거한다. q는 다음 원소를 가리킨다.
s.find(c)	c문자를 검색
s.insert(n,sz)	n의 위치에 sz를 삽입
s.length	문자의 개수
n=s.max_size( )	n는 s가 담을 수 있는 최대 문자의 개수(메모리의 크기)
s.puch_back(c)	s의 끝에 문자 c를 추가
p=s.rbegin( )	p는 s의 역 순차열의 첫 원소를 가리키는 반복자다 (const, 비 const 버전이 있음)
p=s.rend( )	p는 s의 역 순차열의 끝을 표식하는 반복자 (const, 비 const 버전이 있음)
s.replace(pos,n,sz)	Pos 위치의 n개의 문자를 sz으로 바꿈
s.reserve(n)	n개의 원소를 저장할 공간을 예약한다.
s.resize(n)	s의 크기를 n으로 변경하고 확장되는 공간의 값을 기본값으로 초기화한다.

s.resize(n,c)	s의 크기를 n으로 변경하고 확장되는 공간의 값을 c으로 초기화한다.
s.rfind(c)	C 문자를 끝부터 찾음
s.size( )	s는 원소의 개수다.
s2=s.subwtr(pos)	s2는 pos부터의 s문자열
s.swap(s2)	s와 s2를 swap한다
연산자	
s[i]	i번째 위치의 문자
s+=s2	s와 s2의 합을 s에 할당
s+s2	s와 s2를 합한 새로운 string 객체
s=s2	s2를 s에 할당
out<<s	s를 스트림에 씀
in>>s	스트림에서 s로 읽음
getline(in,s)	스트림에서 s로 한 줄을 읽음
그외 비교 연산	==, !=, <, >, <=, >=
맴버 형식 /* 첨언을 하자면 ex) vector<int>::iterator iter 로 불러서 사용한다. */	
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
npos	찾기 관련 '실패' 정의 값이며 일반적으로 npos는 -1이다
pointer	value_type* 형식
reference	value_type& 형식
reverse_iterator	역 반복자 형식
size_type	첨자(index)나 원소의 개수 등의 형식
value_type	원소의 형식