STL 기초

Vector/Deque/List

한컴에듀케이션

STL Container

- **Sequence**
  - array : static array
  - **vector** : dynamic array
  - **deque** : dynamic array
  - foward_list : singly linked list
  - **list** : doubly linked list

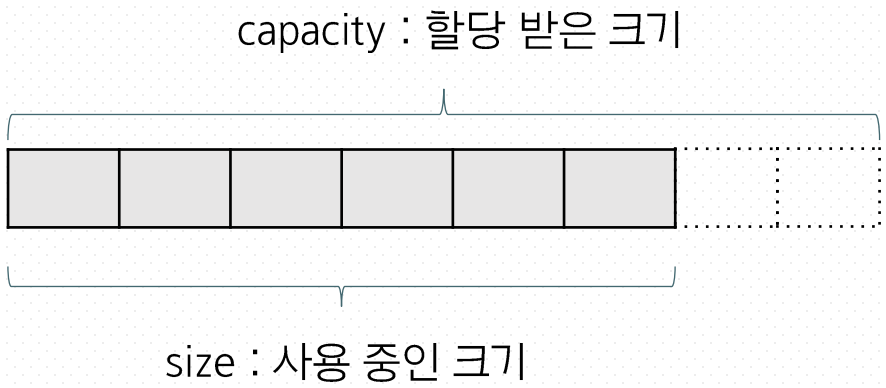
- **Adaptors**
  - stack : LIFO
  - queue : FIFO
  - priority_queue : 우선순위 큐

- **Associative** (Red-Black Tree)
  - set : (Key) 중복X
  - multiset : (Key) 중복O
  - map : (Key,Value), 중복X
  - multimap : (Key,Value), 중복O

- **Unordered associative** (Hash)
  - unordered_set : (Key) 중복X
  - unordered_multiset : (Key) 중복O
  - unordered_map : (Key,Value), 중복X
  - unordered_multimap : (Key,Value), 중복O

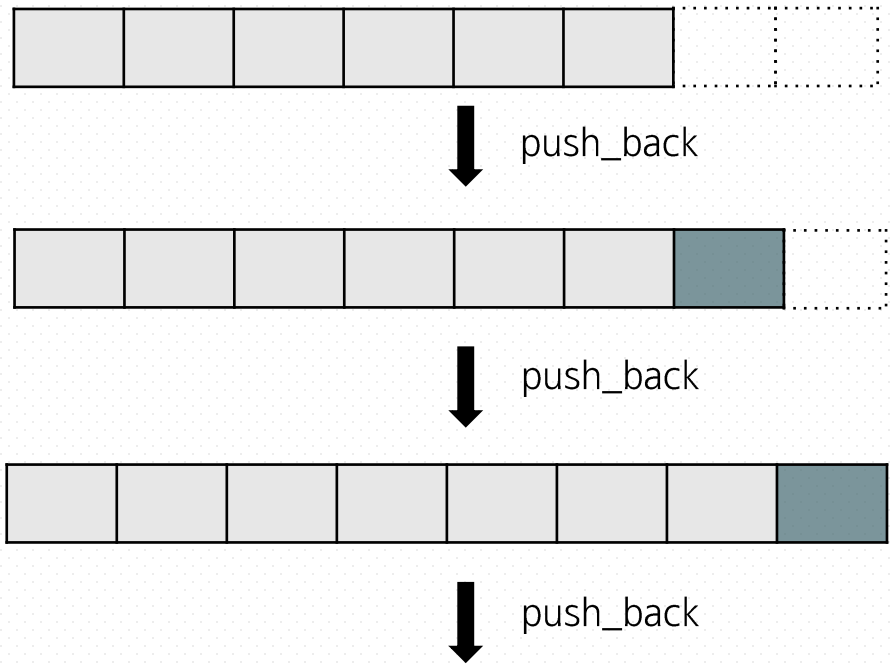
vector

- dynamic array
- random access iterator
- #include<vector>



	Complexity
Space	O(n)
Search	O(n)
Insert	O(n)
Delete	O(n)
Push_back	O(1)
Pop_back	O(1)
Migration	O(n)

vector 공간 할당

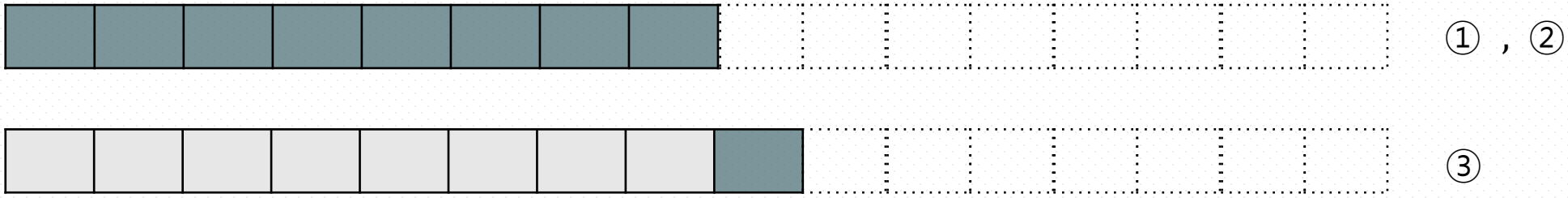


할당 받은 capacity가 꽉 찼는데 insert되는 경우

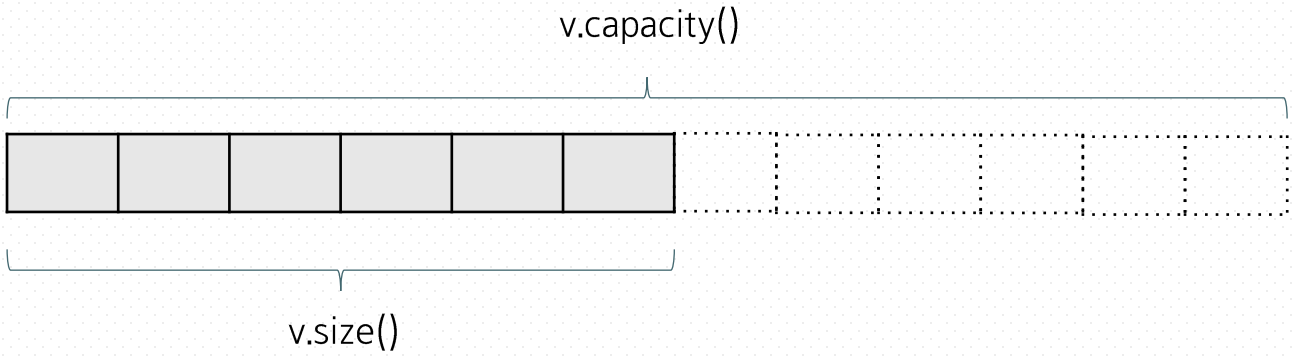
① capacity*2 만큼 새롭게 할당

② 아예 새로운 공간이므로 모든 element가migration

③ insert



vector **reserve** vs **resize**



v.clear() : size = 0, capacity는 변화 없음

v.reserve(x) : capacity를 x로 변경 , 기존 크기보다 작으면 무시

v.resize(x) : size를 x로 변경
capacity가 x보다 작으면 할당 후 변경

vector 주요 문법

```
template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
```

vector<T> v

iterator v.begin(), v.end()

reverse_iterator v.rbegin(), v.rend()

T v[int idx]

bool v.empty()

void v.clear()

size_t v.size()

size_t v.capacity()

void v.resize(int n)

void v.reserve(int n)

size_t : unsigned long long

T v.front() , v.back()

void v.push_back(T x)

void v.pop_back()

iterator v.insert(iterator pos, T x)

iterator v.insert(iterator pos, int count, T value)

iterator v.insert(iterator pos, iterator first, iterator last)

: 처음 입력된 element의 iterator

iterator v.erase(iterator pos)

iterator v.erase(iterator first, iterator last)

: 마지막 지워진 element의 다음 iterator

vector example

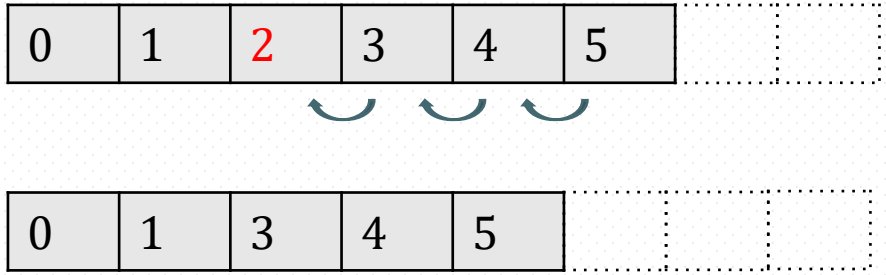
```
vector<int> v;
for (int i = 1; i <= 5; i++) v.push_back(i);

// 1 2 3 4 5
for (int i = 0; i < v.size(); i++) cout << v[i] << ' ';
for (auto x : v) cout << x << ' ';
// vector<int>::iterator it
for (auto it = v.begin(); it != v.end(); ++it) cout << *it << ' ';

// 5 4 3 2 1
// vector<int>::reverse_iterator it
for (auto it = v.rbegin(); it != v.rend(); ++it) cout << *it << ' ';

v.insert(v.begin(), 6);          // 6 1 2 3 4 5
v.insert(v.end(), 7);            // 6 1 2 3 4 5 7
v.insert(v.begin() + 3, 8);      // 6 1 2 8 3 4 5 7

v.erase(v.begin() + 1);          // 6 1 2 8 3 4 5 7
v.erase(v.end() - 1);            // 6 2 8 3 4 5 7
v.pop_back();                    // 6 2 8 3 4 5

v[2] = 0;                        // 6 2 0 3 4
v.empty();                       // 0
v.size();                        // 5

sort(v.begin(), v.end());        // 0 2 3 4 6
```

vector 원소 삭제

1. erase()

- O(n)
- 순서 유지

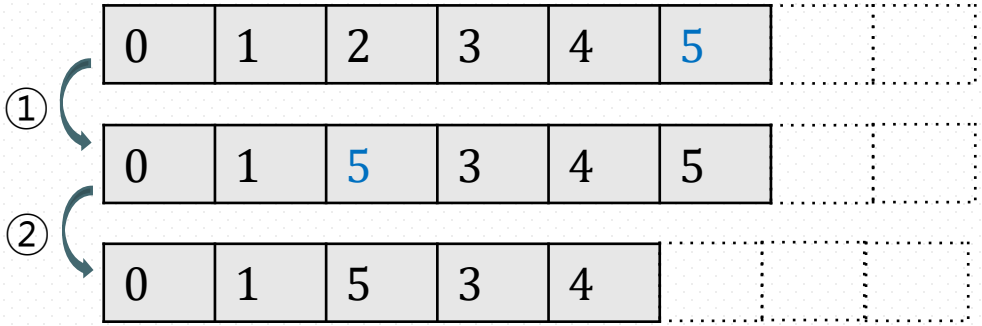
v.erase(v.begin() + 2)



2. 마지막 element로 덮어쓰기

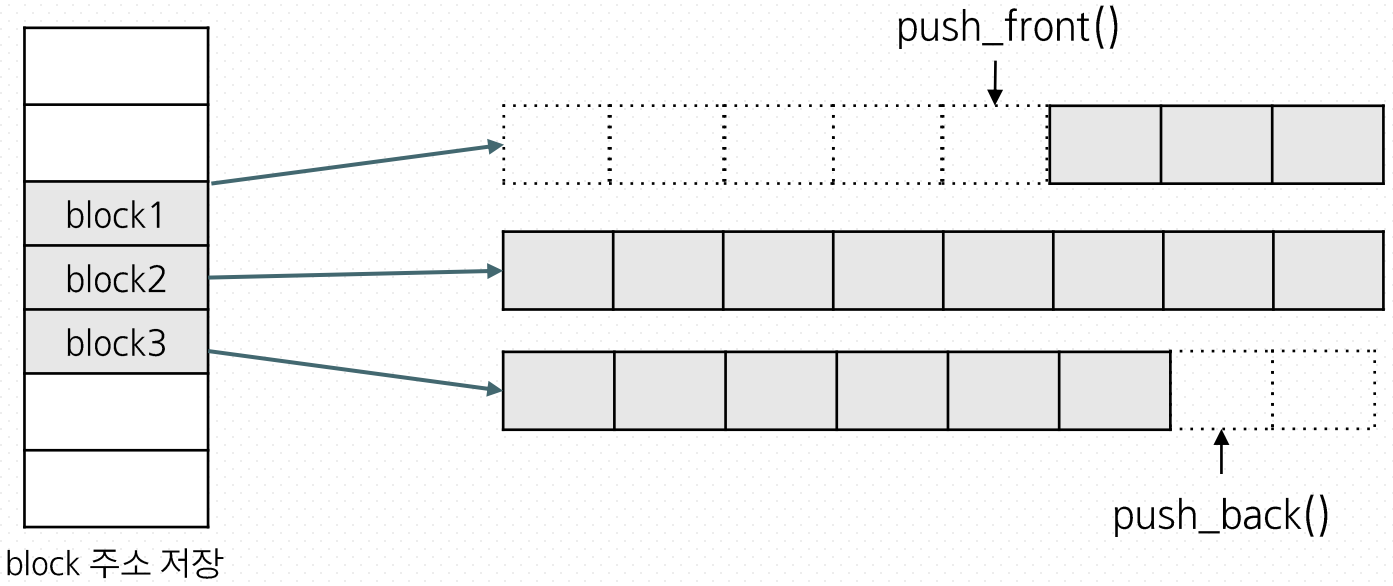
- O(1)
- 순서 유지 X

① v[2] = v.back();
② v.pop_back();



deque

- dynamic array
- random access iterator
- #include<deque>
- vector와의 차이점은 front에 insert/erase가 O(1)
- **사용법은 vector와 (거의)동일하며, push_front(), pop_front() 제공**
- 일반적인 연산은 vector보다 느리므로 front에 insert/erase가 필요한 경우에만 사용
- block 주소를 저장하는 공간이 꽉 찬 경우에 migration 발생



	Complexity
Space	O(n)
Search	O(n)
Insert	O(n)
Delete	O(n)
Push_back **Push_front**	O(1)
Pop_back **Pop_front**	O(1)
Migration	O(block 수)

deque 주요 문법

```
template<
    class T,
    class Allocator = std::allocator<T>
> class deque;
```

deque<T> dq

iterator dq.begin(), dq.end()

reverse_iterator dq.rbegin(), dq.rend()

T dq[int idx]

bool dq.empty()

void dq.clear()

size_t dq.size()

size_t dq.capacity()

void dq.resize(int n)

size_t : unsigned long long

T dq.front() , dq.back()

void dq.push_front(T x), dq.push_back(T x)

void dq.pop_front(), dq.pop_back()

iterator dq.insert(iterator pos, T x)

iterator dq.insert(iterator pos, int count, T value)

iterator dq.insert(iterator pos, iterator first, iterator last)

: 처음 입력된 element의 iterator

iterator dq.erase(iterator pos)

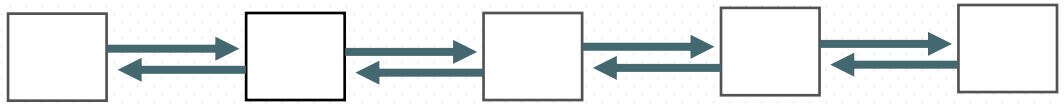
iterator dq.erase(iterator first, iterator last)

: 마지막 지워진 element의 다음 iterator

list

Doubly linked list

- Bi-directional iterator
- #include<list>



front back

	Complexity
Space	O(n)
Search	O(n)
Insert	O(1)
Delete	O(1)

list 주요 문법

```
template<
    class T,
    class Allocator = std::allocator<T>
> class list;
```

list<T> li

T li.front()
T li.back()

iterator li.begin(), li.end()
reverse_iterator li.rbegin(),
li.rend()

void li.clear()
bool li.empty()
size_t li.size()

void li.push_back(T x)
void li.pop_back()
void li.push_front(T x)
void li.pop_front()

iterator li.insert(iterator pos, T x)
iterator li.insert(iterator pos, int count, T x)
iterator li.insert(iterator pos, iterator first, iterator last)
: 처음 입력된 element의 iterator

iterator erase(iterator pos)
iterator erase(iterator first, iterator last)
: 마지막 지워진 element의 다음 iterator

void li.splice(iterator pos, list<T> li2) : li의 pos에 li2 전체 이동
void li.splice(iterator pos, list<T> li2, iterator it) : li의 pos에 li2 의 it 이동
void li.splice(iterator pos, list<T> li2, iterator first, iterator last) : li의 pos에 li2 의 [first, last) 구간 이동

void li.sort() : operator< 기준으로 정렬
void li.sort(function compare) : compare 기준으로 정렬
: O(n log n)

list example1

```
list<int> li, li2;

for (int i = 1; i <= 5; i++)
        li.push_back(i);                          : 1 2 3 4 5

li.insert(li.end(), 6);                           : 1 2 3 4 5 6
li.insert(next(li.begin(), 3), 7);                : 1 2 3 7 4 5 6

li.pop_front();                                   : 1 2 3 7 4 5 6
li.erase(li.begin());                             : 2 3 7 4 5 6
li.erase(--li.end());                             : 3 7 4 5 6

li.sort();                                        : 3 4 5 7
li.sort(greater<int>{});                          : 7 5 4 3

li.splice(li.end(), li, li.begin());              : 5 4 3 7
li2.splice(li2.begin(), li);                      : li = empty, li2 = 5 4 3 7

li.splice(li.begin(),                             : li  = 5 4
          li2,                                    : li2 = 3 7
          li2.begin(), next(li2.begin(), 2));
```

list example2

```
struct Data { int id, value; };

list<Data> li;
list<Data>::iterator it[6];

for (int i = 1; i <= 5; i++)
    it[i] = li.insert(li.end(), { i, 0 });        // insert 하며 iterator 기록

/* 정방향 순회 */
for (auto d : li) d.id , d.value;
for (auto it = li.begin(); it != li.end(); ++it) it->id, it->value;

/* 역방향 순회 */
for (auto it = li.rbegin(); it != li.rend(); ++it) it->id, it->value;

it[5]->value = 3;                              // (1,0) (2,0) (3,0) (4,0) (5,3)
it[3]->value = 4;                              // (1,0) (2,0) (3,4) (4,0) (5,3)

li.erase(it[1]);                               // (1,0) (2,0) (3,0) (4,0) (5,3)
li.splice(li.begin(), li, it[5]);              // (5,3) (2,0) (3,0) (4,0) (5,3)
```


감사합니다