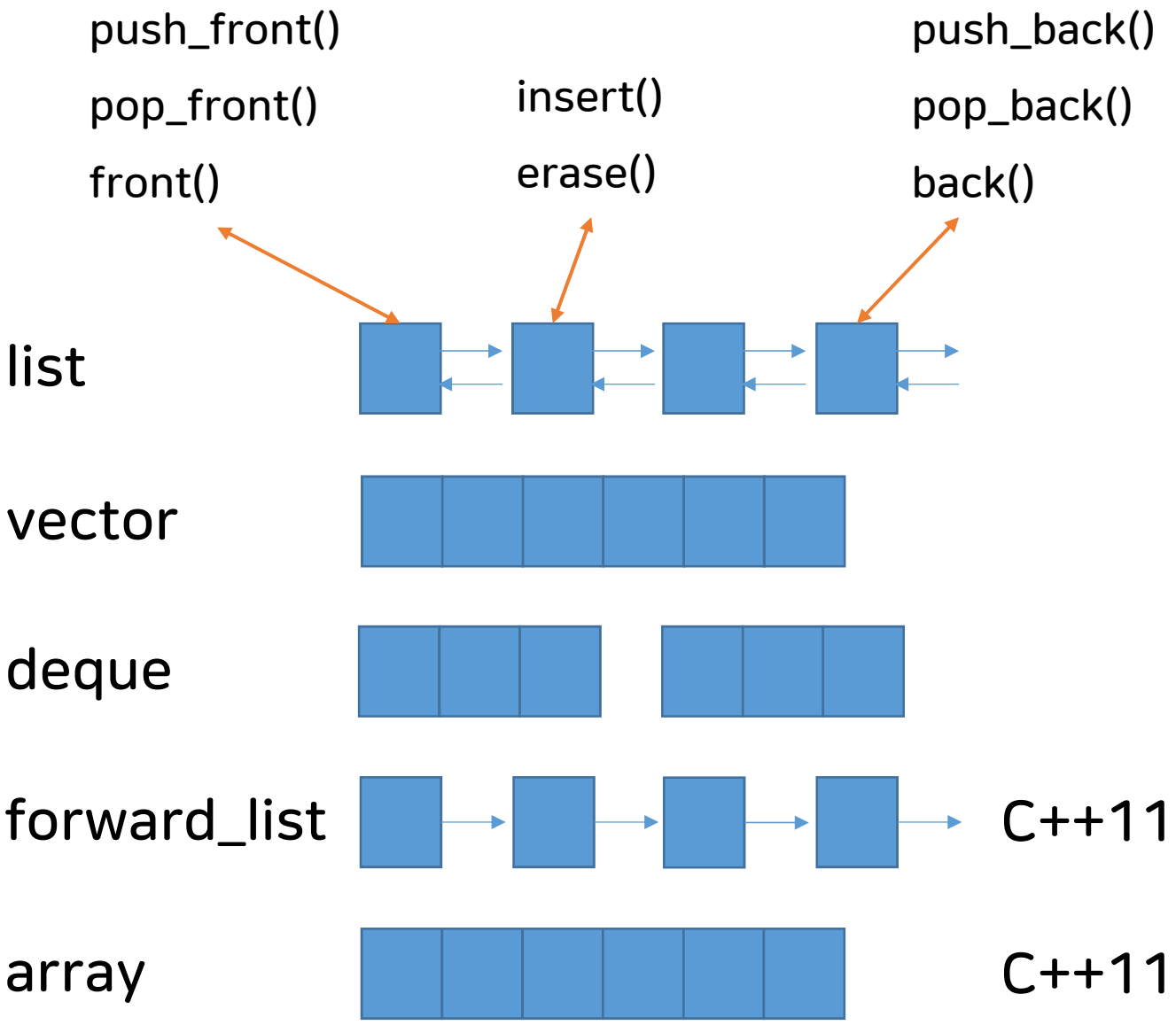
각 요소가 삽입된 순서대로 선형적으로 놓인 컨테이너



컨테이너	반복자 종류		전방 삽입	임의 삽입	후방 삽입
list	양방향	++, --	O	O	O
vector	임의 접근	++, --, [], +, -	X	O	O
deque	임의 접근	++, --, [], +, -	O	O	O
forward_list	전진형	++	O	O	X
array	임의 접근	++, --, [], +, -	X	X	X

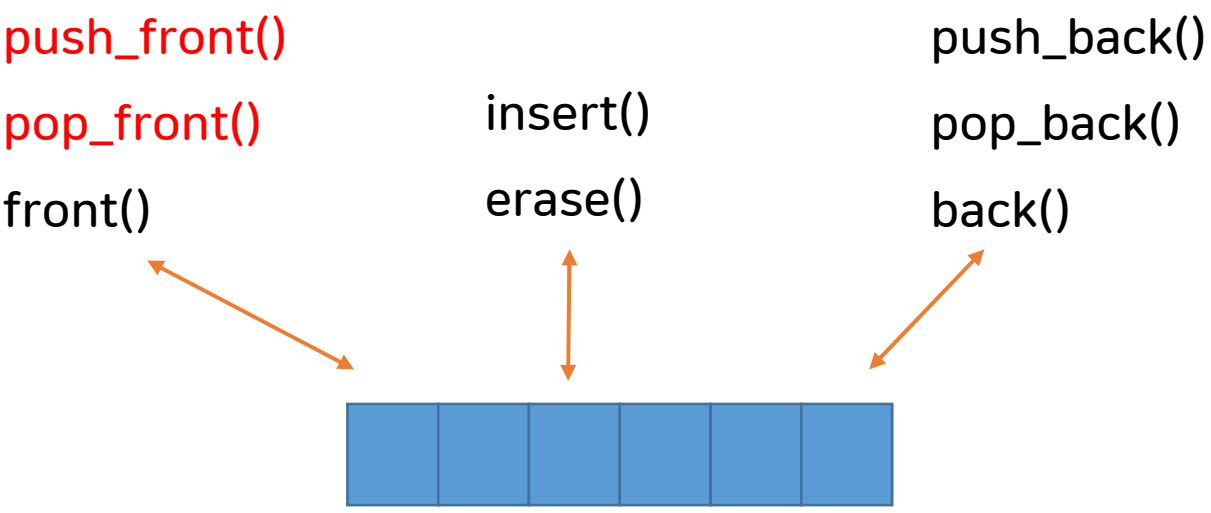
vector template

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

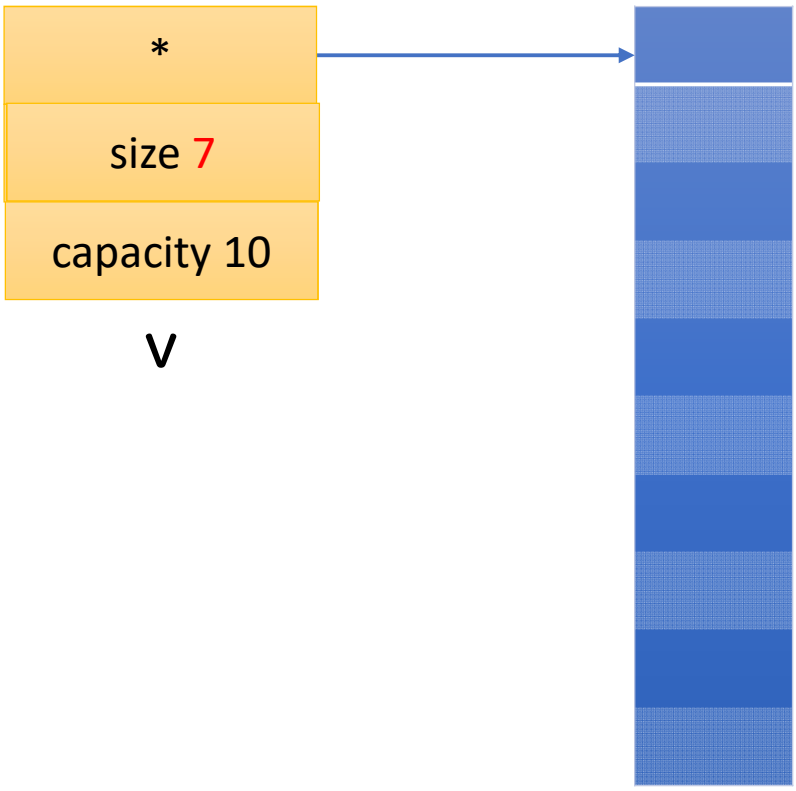
vector의 생성

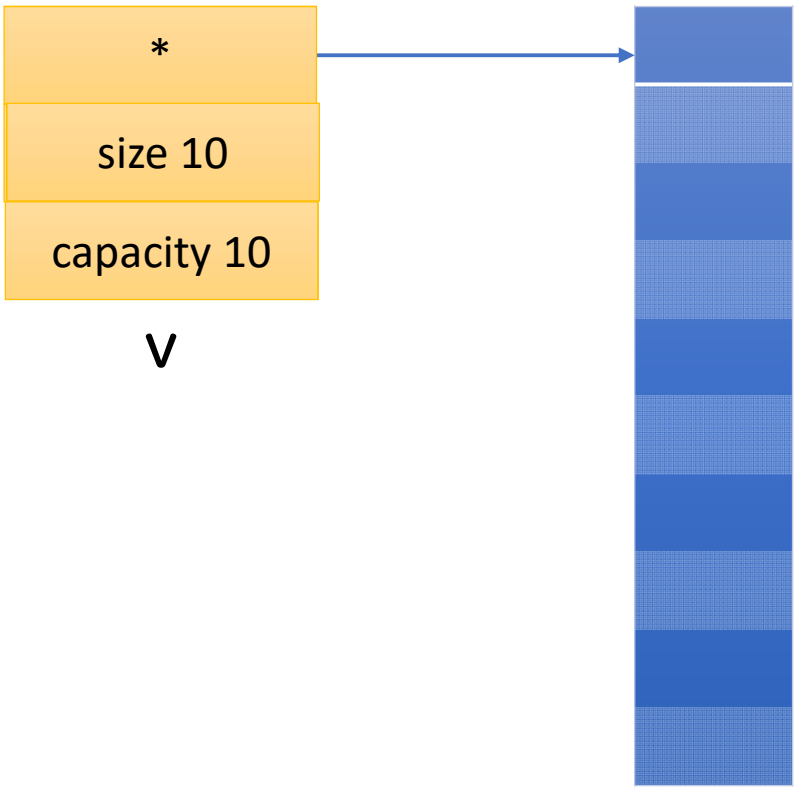
- vector<int> v(10, 0);
- vector<int> v{10, 0};

요소 삽입/삭제/접근



- emplace_xxx() 함수 ➔ "사용자 정의 타입과 컨테이너" 동영상 참고


*
size 7
capacity 10
v


*
size 10
capacity 10
v

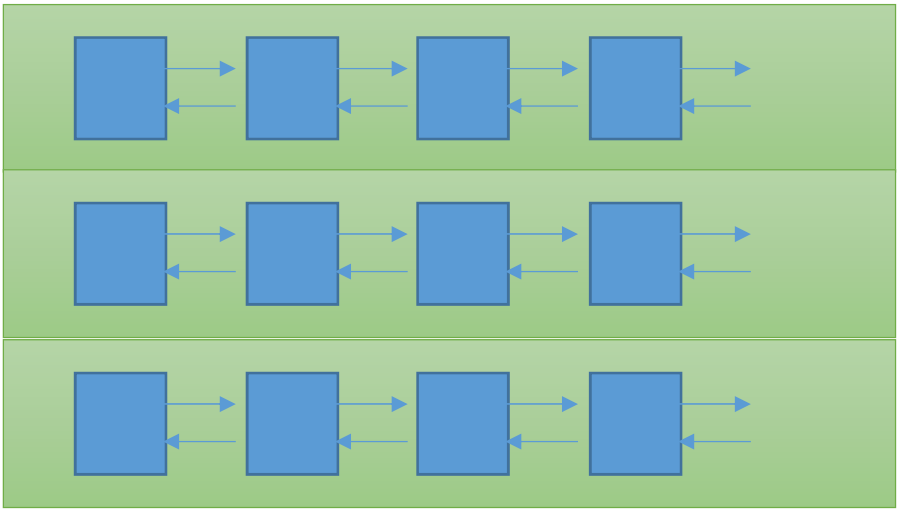
#include <iostream>

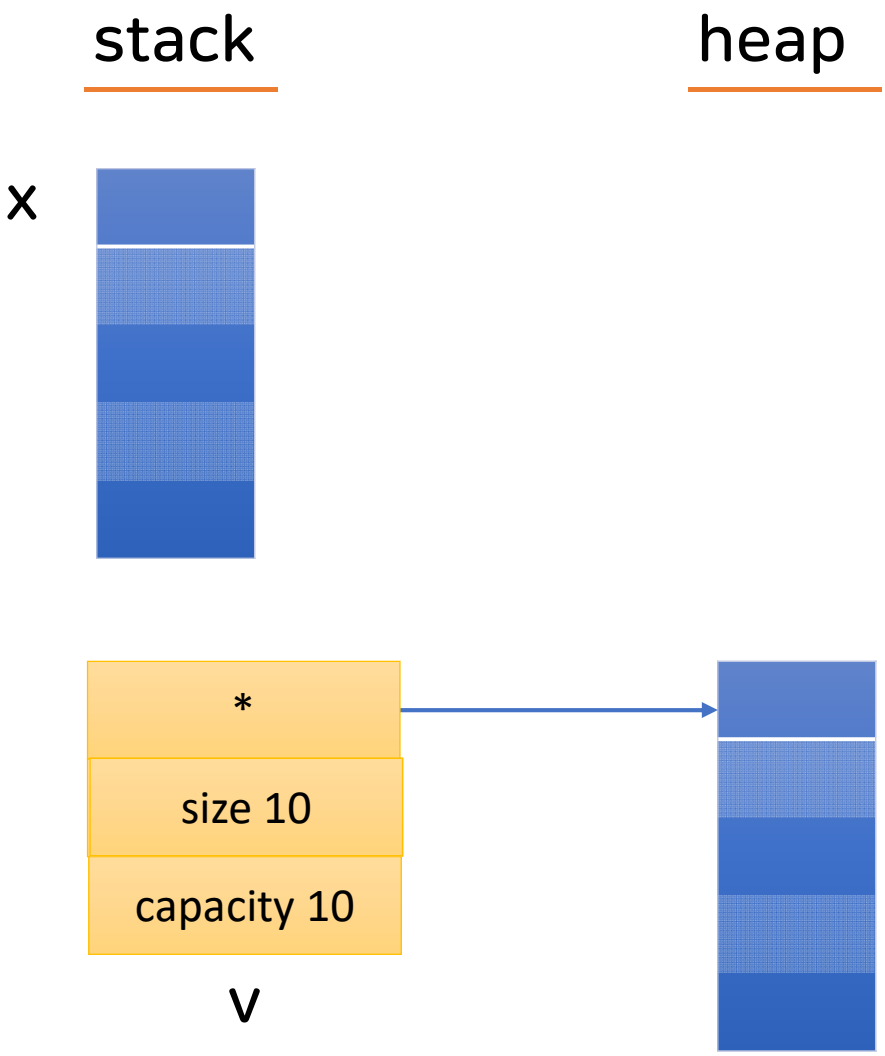
#include <string>

.....

.....

v





std::array

- C++11 에서 추가됨
- 배열과 동일하게 stack 에 버퍼를 만드는 컨테이너
- 실제 배열을 사용하므로 전방삽입, 후방삽입, 임의삽입이 모두 불가능 하다.

반복자를 만들 때 raw pointer 보다는 객체형으로 만드는 것이 좋다.

- raw pointer 를 값으로 리턴 하면 ++ 연산을 사용할 수 없다.
- 객체형 반복자는 ++ 연산을 사용할 수 있다.
- raw pointer도 next() 알고리즘을 사용하면 다음으로 이동 할수 있다.

사용자 정의 타입을 컨테이너에 넣 을 때

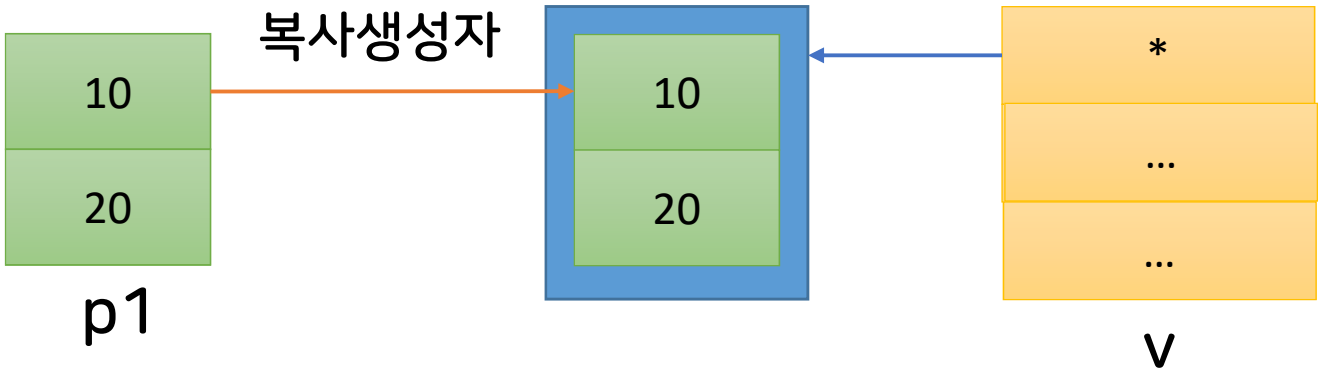
- 디폴트 생성자가 없을 경우, 복사 생성을 위한 객체를 전달

사용자 정의 타입과 sort 알고리즘

- sort의 조건자 버전을 사용한다.
- 사용자 정의 타입에 < 와 == 연산자를 제공한다.

std::rel_ops namespace

- >, !=, <=, >= 연산자의 일반화 버전을 제공



- vector 에 사용자 정의 타입을 넣을 때는 emplace_ 계열의 함수를 사용하는 것이 좋다.

allocator 개념

- 메모리 할당 을 추상화한 타입

std::allocator

- C++ 표준 allocator
- 주요 멤버 함수

allocate	메모리 할당
deallocate	메모리 해지
construct	할당된 메모리에 객체 생성(생성자 호출)
destroy	객체 파괴(소멸자 호출)

STL Container 와 allocator

- template<class T, class Allocator = std::allocator<T>>
  class vector;
- get_allocator() 멤버함수.

사용자 정의 allocator 만들려면

- allocate/deallocate/construct/destroy
- Allocator Concept 를 만족해야 한다.
- C++11 부터는 allocator_traits 가 제공되기 때문에 많은 요소가 optional 로 되어 있다.
- 최소 요구 조건은 아래 멤버만 제공하면 된다.

Member type	value_type
Member function	디폴트 생성자
	템플릿 생성자
	allocate(size_t)
	deallocate(T*, size_t)
equality	==
	!=