- Iterator Category

반복자 카테고리

1. vector 는 sort() 알고리즘을 사용할 수 있다.

2. list 는 sort() 알고리즘을 사용할 수 없다.

3. 왜 사용할 수 없을까 ?

- Iterator Category

핵심 정리

1. 컨테이너의 종류에 따라 반복자에 적용할 수 있는 연산의 종류가 다르다.

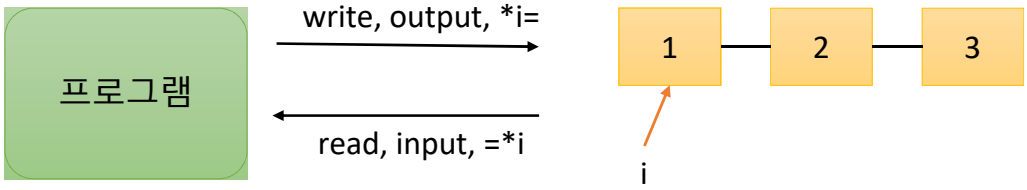
반복자	연산
list의 반복자	++, -- 모두 가능
forward_list 의 반복자	++ 만 가능.

2. 반복자는 적용할 수 있는 연산에 따라 5가지(C++17 부터는 6가지)로 분류.

반복자 카테고리(iterator category)	연산
입력 반복자 (input iterator)	입력(=*i), ++
출력 반복자 (output iterator)	출력(*i=), ++
전진형 반복자 (forward iterator)	입력, ++, multiple pass
양방향 반복자 (bidirectional iterator)	입력, ++, --, multiple pass
임의 접근 반복자 (random access iterator)	입력, ++, --, +, -, []
인접한 반복자 (contiguous iterator) – C++17	입력, ++, --, +, -, [], *(i+n) == *(addressof(*a) + n)

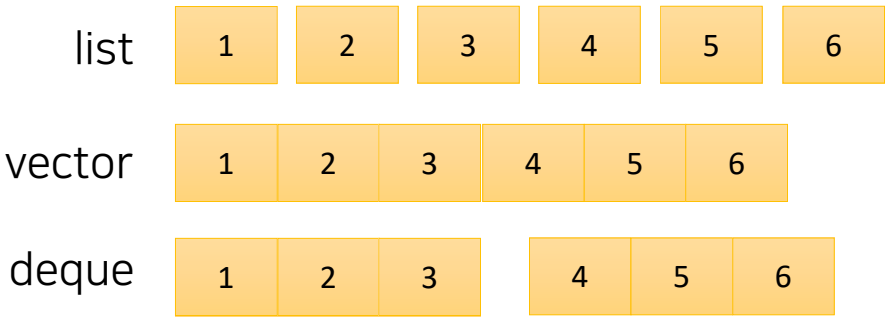
반복자 카테고리 관련 용어 정리

1. 입력 vs 출력



- 입력(input) : 반복자를 통해서 컨테이너 안의 요소를 읽어 오는 것 (= *i)
- 출력(output) : 반복자를 통해서 컨테이너 안의 요소에 값을 쓰는 것 (*i =)

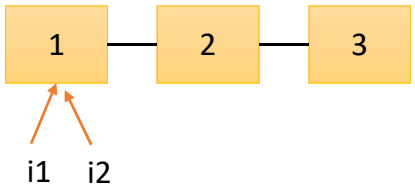
2. ++ vs +



- list 반복자 : ++ 연산은 제공되지만 + 연산은 제공되지 않는다.
- vector 반복자 : ++과 +연산이 모두 제공된다.

- + 연산이 제공되어도 반드시 연속된 메모리인 것은 아니다.

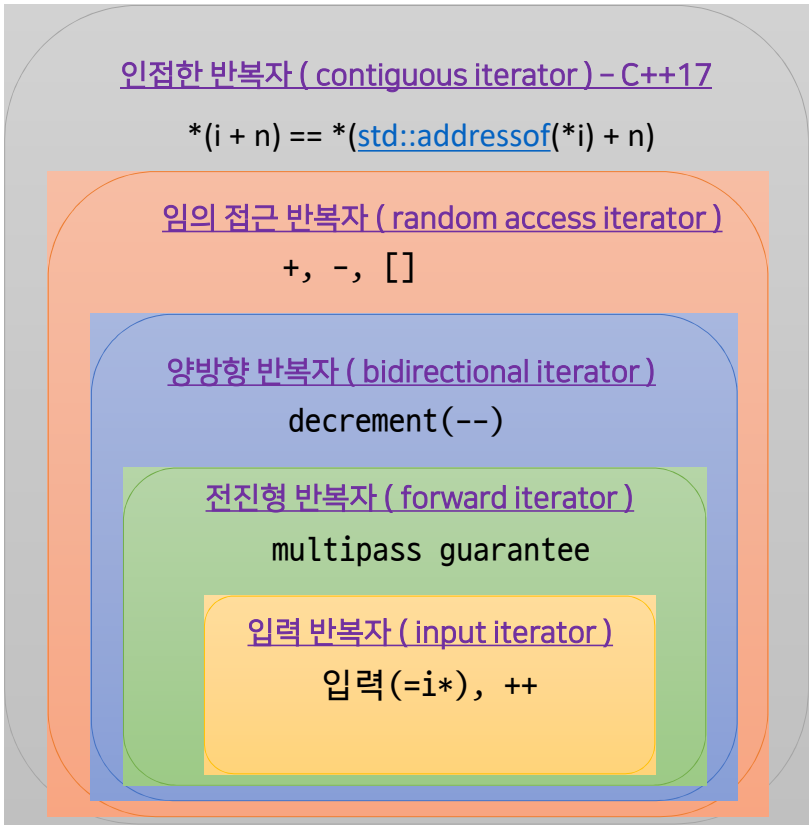
mutipass guarantee 개념



1. multipass guarantee
   - 2개의 반복자 i1, i2 에 대해서 다음을 만족.
   - i1 == i2 라면 *i1 == *i2
   - i1 == i2 라면 ++i1 == ++i2
   - 2개의 이상의 반복자가 컨테이너의 요소에 동일하게 접근함을 보장
   - list의 반복자는 multipass 를 보장한다.
   - stream iterator 와 insert iterator 는 multipass 를 지원하지 못한다.

핵심 개념

- Iterator Category



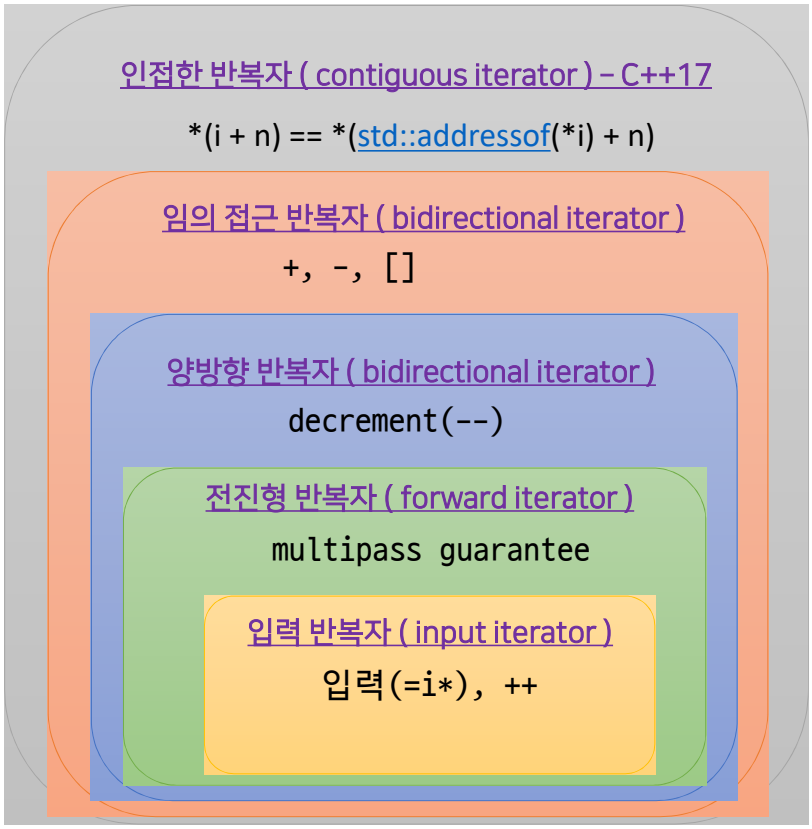
출력 반복자 (output iterator)
출력(i*=), ++

출력 가능한 반복자 : “mutable iterator”

iterator category	연산	
input iterator	입력(=*i), ++	istream_iterator
output iterator	출력(*i=), ++	ostream_iterator
forward iterator	입력, ++	forward_list
bidirectional iterator	입력, ++, --	list
random access iterator	입력, ++, --, +, -, []	deque
contiguous iterator) – C++17	입력, ++, --, +, -, [], *(i+n) == *(addressof(*a) + n)	array, vector

핵심 개념

- Iterator Category



출력 반복자 (output iterator)

출력(i*=), ++

출력 가능한 반복자 : “mutable iterator”

1. 왜 반복자 카테고리(iterator category)가 중요 한가 ?
2. 다양한 알고리즘의 인자로 요구하는 반복자 카테고리가 무엇인지를 알아야 한다.
   - find : input iterator
   - reverse : bidirectional iterator
   - sort : random access iterator
3. list 는 sort() 알고리즘을 사용할 수 없다. 하지만 멤버 함수 sort()를 사용할 수 있다.

- Tag Dispatching

핵심 개념

1. advance 함수
   - <iterator> 헤더
   - 반복자를 N 만큼 전진(후진) 하는 함수.

2. advance 함수 구현하기
   - 반복자를 이동 할 때 어떤 방법을 사용 해야 할까 ?
   - + : 임의 접근 반복자만 가능하다.
   - ++ : 모든 반복자가 가능하지만, 임의 접근 반복자의 경우 +로 이동하는 것이 빠르다.
   - 임의 접근 반복자인 경우와 그렇지 않은 경우를 다르게 구현 해야 한다. - tag dispatching 기술을 사용.

- Tag Dispatching

핵심 개념

1. iterator category tag
   - <iterator> 헤더
   - 반복자의 5가지 category 를 나타내는 타입. empty class 로 제공.
   - C++17 에서 추가된 contiguous iterator 는 별도의 tag 가 제공되지 않음.

2. “iterator_category” member type
   - 모든 반복자는 내부적으로 “iterator_category” 라는 이름의 “member type”을 제공해야 한다.
   - 반복자 카테고리에 따라서 다른 알고리즘이 필요한 경우 “iterator_category” member type 을 활용한 “tag dispatching” 기법을 사용

- Tag dispatching

핵심 개념

1. tag dispatching
   - 함수 인자를 사용한 함수 오버로딩
   - type_traits 와 enable_if 사용 – C++ “Template Programming 참고”
   - if constexpr 사용 – C++17

2. Raw Pointer 를 전진시키기 위해 eadvance() 함수를 사용할 수 있을까 ?
   - “iterator traits” 동영상 참고

- 반복자와 member type

핵심 개념

1. 반복자(iterator)를 만들 때 반드시 member type을 제공해야 한다.

Member type	Definition
value_type	T
difference_type	Signed integer type (usually std::ptrdiff_t)
reference	value_type&
pointer	value_type*
iterator_category	컨테이너 마다 다른 정의

2. iterator class
   - <iteratror> 헤더
   - 모든 반복자의 기반 클래스 – C++17 이전 까지, C++17 부터는 사용되지 않음.

- 알고리즘과 category

핵심 개념

1. 알고리즘 함수 만들 때
    - template 인자의 이름을 iterator category 를 나타 내도록 표기하는 것이 좋다.