STL Wrap-Up

- 내일배움캠프 스탠다드반 -

발표자 : 장재근

2025. 08. 04. (월)

목 차

Chapter 01 **STL 이란?**

'STL' ?

'STL' 을 사용하는 이유?

Chapter 02 **STL 컨테이너**

STL 컨테이너

- vector, list, deque
- set, map
- priority_queue

Chapter 03 **STL 반복자**

반복자 'Iterator' ?

Chapter 04 **STL 알고리즘**

다양한 알고리즘 함수들

- Algorithm
- Numeric

목 차

'STL' ?

Standard Template Library

여러 자료 구조, 함수, 알고리즘 등을 쓰기 쉽게 정형화한 라이브러리를 말합니다.
STL은 크게, 컨테이너, 반복자, 알고리즘, 함수자로 구성되어 있습니다.

- STL 컨테이너: 자료 구조, 데이터를 저장하는 객체입니다.
- STL 반복자: 포인터와 비슷한 개념, 컨테이너의 데이터 위치를 가리킵니다. 데이터에 접근할 때 용이합니다.
- STL 알고리즘: 자주 사용되는 알고리즘 일부를 라이브러리에 포함 / 정렬, 탐색, 삽입, 삭제를 용이하게 해줍니다.
- STL 함수자: 함수처럼 동작하는 객체를 말합니다. / operator() 연산자

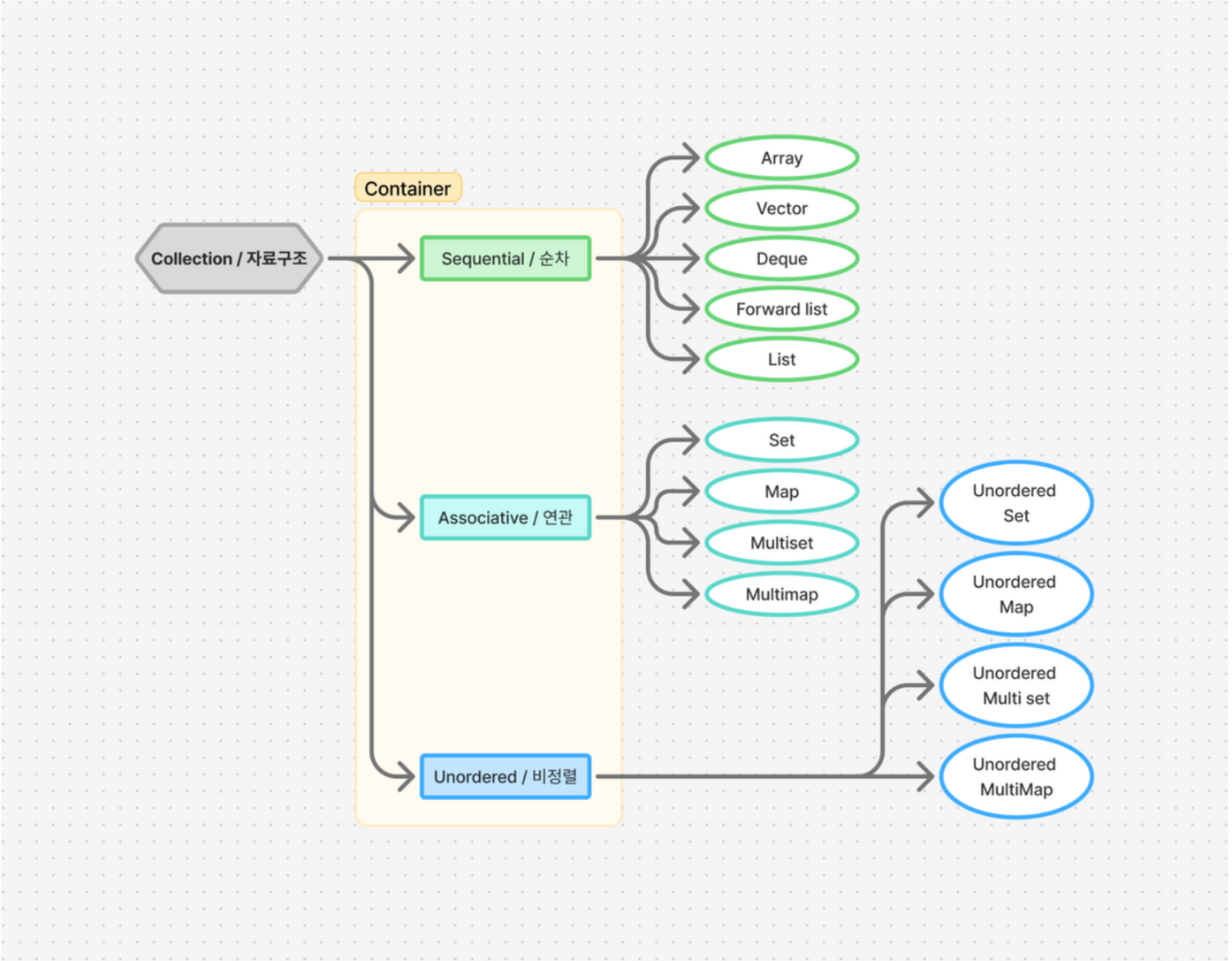
'STL' 을 사용하는 이유?

STL은 다양한 자료구조(컨테이너)와 알고리즘, 반복자 등을 템플릿 기반으로 제공해줍니다.
이로 인해 개발자가 직접 복잡한 자료구조나 알고리즘을 구현하는 번거로움을 줄여줍니다.

- 템플릿 기반으로 작성되어 있어, 재사용성과 유지보수, 호환성이 좋습니다.
- 최적화된 알고리즘과 자료구조를 제공하기 때문에, 효율성이 좋습니다.
- 오랜 시간 사용되어 왔고, 다양한 환경에서 검증되었기 때문에 신뢰성이 높습니다.

목 차

STL 컨테이너



목 차

STL 컨테이너

1. vector

- 동적 배열로 구현된 순차 컨테이너
- 랜덤 접근이 상대적으로 빠릅니다.
- 마지막 원소를 삽입, 삭제할 때 효율적입니다.
- 중간 삽입, 삭제 시 비효율적입니다. // 데이터의 이동 발생

※vector의 주요 메서드

```
front()                 // 맨 앞의 원소 반환
back()                  // 맨 뒤의 원소 반환
push_back(value)        // 맨 뒤에 원소 추가
pop_back()              // 맨 뒤의 원소 제거
insert(iterator, value) // 지정된 위치에 원소 삽입
erase(iterator)         // 지정된 위치의 원소 삭제
clear()                 // 모든 원소 삭제
size()                  // 원소 개수 반환
capacity()              // 할당된 메모리 크기 반환
at(index) / [index]     // 지정된 인덱스의 원소 반환
data()                  // 내부 배열 포인터 반환
```

목 차

STL 컨테이너

2. list

- 이중 링크드 리스트로 구현된 순차 컨테이너
- 메모리가 연속적이지 않아 랜덤 접근은 상대적으로 느립니다.
- 원소를 중간에 삽입, 삭제할 때 효율적입니다.

※list의 주요 메서드

```
push_front(value)          // 맨 앞에 원소 추가
pop_front()                // 맨 앞의 원소 제거
push_back(value)           // 맨 뒤에 원소 추가
pop_back()                 // 맨 뒤의 원소 제거
insert(iterator, value)    // 지정된 위치에 원소 삽입
erase(iterator)            // 지정된 위치의 원소 삭제
```

목 차

STL 컨테이너

3. deque

- 양방향 동적 배열로 구현된 순차 컨테이너
- 랜덤 접근이 상대적으로 빠릅니다.
- 맨 앞과 맨 뒤의 원소를 삽입, 삭제할 때 효율적입니다.
- 중간 삽입, 삭제가 비효율적입니다.

※deque의 주요 메서드

```
front()                 // 맨 앞의 원소 반환
back()                  // 맨 뒤의 원소 반환
push_front(value)       // 맨 앞에 원소 추가
pop_front()             // 맨 앞의 원소 제거
push_back(value)        // 맨 뒤에 원소 추가
pop_back()              // 맨 뒤의 원소 제거
insert(iterator, value) // 지정된 위치에 원소 삽입
erase(iterator)         // 지정된 위치의 원소 삭제
clear()                 // 모든 원소 삭제
at(index) / [index]     // 지정된 인덱스의 원소 반환
```

목 차

STL 컨테이너

4. set

- 레드/블랙 트리로 구현한 연관 컨테이너
- key(value)만 저장, 중복 key를 가지지 못하고 자동으로 정렬 상태를 유지합니다.
- 검색/삽입/삭제가 모두 평균 O(logN)의 시간 복잡도를 가집니다.

※set의 주요 메서드

```
insert(value)          // 원소 삽입
erase(value)           // 원소 삭제
find(value)            // 원소 검색
size()                 // 원소 개수 반환
lower_bound(value)     // value 이상인 첫 번째 원소 반환
upper_bound(value)     // value 초과인 첫 번째 원소 반환
```

목 차

STL 컨테이너

5. map

- 레드/블랙 트리로 구현한 연관 컨테이너
- key와 value를 쌍으로 저장, 중복 key를 가지지 못하고 자동으로 정렬 상태를 유지합니다.
- 검색/삽입/삭제가 모두 평균 O(logN)의 시간 복잡도를 가집니다.

※map의 주요 메서드

```
insert(pair)        // key-value 삽입
erase(key)          // key로 원소 제거
find(key)           // key로 원소 검색 (iterator 반환)
[key]               // key로 value 접근 or 대입
size()              // 원소 개수 반환
```

목 차

STL 컨테이너

6. multiset

- 동일한 값(key) 중복을 허용하는 set
- 정렬된 상태의 중복 원소를 가지는 배열이 필요할 경우 사용합니다.

7. multimap

- 동일한 key에 여러 개의 value를 허용하는 map
- 하나의 key로 여러 개의 value를 관리하고 싶은 경우 사용

※multimap의 주요 메서드

```
insert(pair)          // key-value 삽입
erase(key)            // key로 원소 제거
find(key)             // key로 원소 검색 (iterator 반환)
equal_range(key)      // 동일한 key의 value들의 범위를 반환
```

목 차

STL 컨테이너

8. priority_queue

- 힙(Heap) 자료구조로 구현한 queue
- 삽입/삭제의 시간복잡도가 항상 O(log N)입니다.
- 가장 우선순위가 높은 원소는 O(1)의 시간복잡도를 가집니다.

※priority_queue의 주요 메서드

```
push(value)          // 원소 삽입
pop(value)           // 원소 제거
top()                // 가장 우선순위가 높은 원소 조회
```

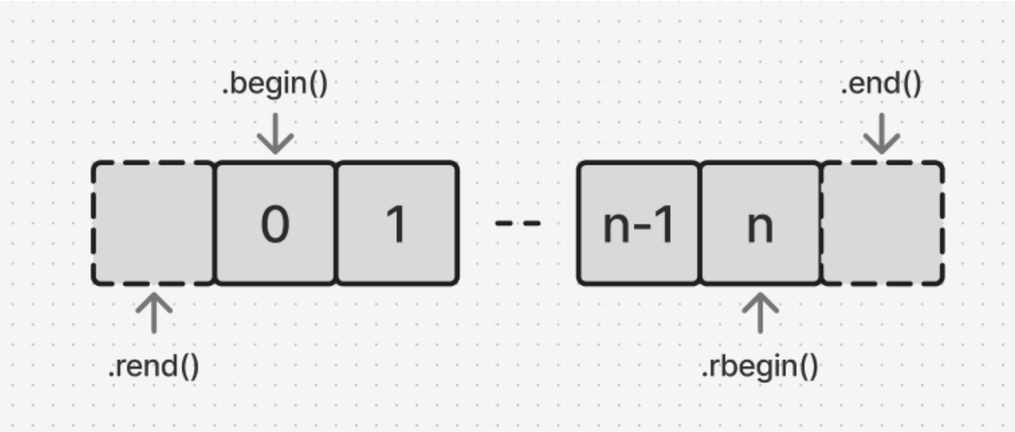
※선언 시, greater<T>를 인자로 줄 경우, 최소 힙으로 전환

목 차

STL 반복자

Iterator

- 컨테이너 데이터를 순회하기 위한 일종의 포인터
- 알고리즘, 반복문에 자주 사용됩니다.
- 정방향 반복자 / begin(), end()
- 역방향 반복자 / rbegin(), rend()
- 상수 반복자 / cbegin(), cend()
- it++, it--로 반복자의 위치 이동이 가능합니다.



목 차

STL 알고리즘

Algorithm

1. 정렬 sort

```
sort(begin_it, end_it) // 오름차순
sort(begin_it, end_it, compare)
```

2. 순열 permutaion

- 정렬된 상태인 배열이어야합니다.

```
next_permutation(begin_it, end_it) // 다음 순열
prev_permutation(begin_it, end_it) // 이전 순열
```

3. K번째 값 찾기 nth_element

- pivot을 기반으로 n번째로 작은 원소를 배열의 n번째 위치로 보낸 뒤, 좌측은 보다 작은 원소들로, 우측은 보다 큰 원소들로 반정렬 상태로 만들어줍니다.
- 평균 시간 복잡도가 O(n)입니다.

```
nth_element(begin_it, begin_it + n, end_it)
```

목 차

STL 알고리즘

Algorithm

4. 중복 제거 unique
- 정렬된 배열에서 중복된 원소들을 뒤로 이동시키고, 중복된 원소들의 시작점(iterator)를 반환합니다.

```
erase(unique(begin_it, end_it), end_it)
```

5. 이진 탐색 lower_bound, upper_bound
- n개의 원소를 가진 배열에 대한 시간 복잡도 O(log n)을 가집니다.
- lower_bound // k 이상의 값이 처음 나오는 위치의 반복자를 반환합니다.
- upper_bound // k 초과의 값이 처음 나오는 위치의 반복자를 반환합니다.

```
lower_bound(begin_it, end_it, k)
under_bound(begin_it, end_it, k)
```

6. 원소 변환 transform
- 구간 내의 원소를 변환시켜 주는 알고리즘입니다.
- 단항 연산

```
transform(v1.begin(), v1.end(), v2.begin(), [](function)) // v1의 원소들을 변환해 v2에 삽입
```

- 이항 연산

```
// v1과 v2의 원소끼리 더해서 v3에 저장하는 예시
transform(v1.begin(), v1.end(), v2.begin(), v3.begin(), plus<int>())
```

목 차

STL 알고리즘

Algorithm

7. 최소값, 최대값 min_element, max_element, minmax_element
 - 컨테이너에서 최소값과 최대값의 iterator를 반환합니다.

```
min_element(begin_it, end_it)
max_element(begin_it, end_it)
auto [min_it, max_it] = minmax_element(begin_it, end_it)
```

8. 특정 원소의 개수 구하기

```
count(begin_it, end_it, k) // k와 같은 원소의 개수 반환
count_if(begin_it, end_it, [](int x){ return x % 2 == 0; }) // 조건의 만족하는 원소의 개수를 반환하는 예시
```

9. 두 원소의 거리 구하기

```
distance(first_it, second_it)
```

10. n칸 뒤의 원소 구하기

```
advance(begin_it, n)
```

목 차

STL 알고리즘

Numeric

1. 연속 구간 합 구하기 partial_sum

```
partial_sum(v.begin(), v.end(), psum.begin()) // v의 0 ~ idx까지의 원소의 합을 psum의 idx에 대입
psum[k-1] - psum[n-1] // n~k까지의 원소만 구하고 싶은 경우
```

2. 총합 구하기 accumulate

```
accumulate(begin_it, end_it, 초기값) // 초기값 + 모든 원소의 합
accumulate(begin_it, end_it, 초기값, [](int a, int b){ return a + b + 2; }) // 초기값 + (모든 원소 + 2)의 합
accumulate(begin_it, end_it, 초기값, multiplies<int>()) // 초기값 * 모든 원소의 곱
```

감사합니다

THANK YOU