STL 기초

STL Algorithm

한컴에듀케이션

Algorithm Library

- 주로 컨테이너 반복자(배열 주소 값)로 다양한 작업을 수행하도록 도와준다.
- 반복자 없이 값으로만 수행되는 함수도 있다.
- function을 인자로 설정해주기도 한다.
- range는 항상 [fisrt, last) 이다. last 미포함

함수의 형태는 대부분 아래와 같다.

- func(iterator first, iterator last, T value) : find, count, lower_bound, upper_bound
- func(iterator first, iterator last) : sort, max_element, min_element
- func(iterator first, iterator last, function f) : for-each, find_if, count_if, sort
- func(T a, T b) : swap, max, min
- func({ initializer list }) : max, min

function f를 설정하는 방법

1. naïve function
2. function object
3. lambda

ex) vector를 절대값 기준으로 오름차순 정렬하는 경우

1. naïve function

```
bool comp(int a, int b) {
    return abs(a) < abs(b);
}
```

2. function object

```
struct Compare {
    bool operator()(const int a, const int b) const{
        return abs(a) < abs(b);
    }
}comp;
```

3. lambda

```
auto comp = [](int a, int b) { return abs(a) < abs(b); }
```

```
sort(arr, arr+n, comp);
```

Algorithm library 주요 함수

- sort, partial_sort
- nth_element
- find, find_if
- swap
- max, min, minmax
- lower_bound, upper_bound
- max_element, min_element, minmax_element
- for_each
- count, count_if
- remove, remove_if etc

sort()

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last );

template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp );
```

[first, last) 구간을 comp 기준에 맞게 정렬

comp를 명시하지 않으면 operator< 기준에 맞게 정렬

bool comp(const T&lhs, const T&rhs)

Compare requirement 만족 필수

O(n log n)

```
vector<int> arr{ 5,3,-2,1,-4 };
sort(arr.begin(), arr.end());                    // -4, -2, 1, 3, 5
sort(arr.begin(), arr.end(), greater<int>{});    // 5, 3, 1, -2 -4
```

sort()

1. function

```
bool absSort(int l, int r) { return abs(l) < abs(r); }

sort(v.begin(), v.end(), absSort);
```

2. function object

```
struct AbsSort {
    bool operator()(int l, int r) { return abs(l) < abs®; }
}absSort;

sort(arr.begin(), arr.end(), AbsSort{});
sort(arr.begin(), arr.end(), absSort);
```

3. lambda

```
sort(
    arr.begin(),
    arr.end(),
    [](auto&l, auto&r) { return abs(l) < abs(r); }
);
```

partial_sort()

```
template< class RandomIt >
void partial_sort( RandomIt first, RandomIt middle, RandomIt last );
```

```
template< class RandomIt, class Compare >
void partial_sort( RandomIt first, RandomIt middle, RandomIt last,
                   Compare comp );
```

: [first, last) 구간에서 우선순위 기준으로 [first, middle) 만 정렬한다.

: heap sort 기반

: default 값과 compare 설정은 sort와 동일

: O(*(last - fisrt) log (middle - fisrt)*)

```
vector<int> arr{ 5,3,-2,1,-4 };
partial_sort(arr.begin(), arr.begin() + 2, arr.end());  // -4 -2 5 3 1
```

nth_element()

```
template< class RandomIt >
void nth_element( RandomIt first, RandomIt nth, RandomIt last );

template< class RandomIt, class Compare >
void nth_element( RandomIt first, RandomIt nth, RandomIt last,
                  Compare comp );
```

- [first, last) 구간에서 comp 기준(default : <)으로 nth 위치의 값을 선택하여 왼쪽, 오른쪽에 comp 기준에 맞게 값들을 배치한다.
- < 기준으로 보면 nth위치 값의 왼쪽에는 *nth보다 작거나 같은 값, 오른쪽에는 *nth보다 크거나 같은 값이 들어간다. 그 값들은 정렬되어 있지는 않다.
- comp 기준은 sort와 동일하다.
- IntroSelect 기반, average O(n)

```
vector<int> v;
nth_element(v.begin(), v.begin()+3, v.end());  // x x x o y y y y
```

find(), find_if()

find()

```
template< class InputIt, class T >
InputIt find( InputIt first, InputIt last, const T& value );
```

- [first, last) 구간에서 값이 value인 첫번째 element의 iterator 반환
- auto it = find(arr.begin(), arr.end(), 1);

---

find_if()

```
template< class InputIt, class UnaryPredicate >
InputIt find_if( InputIt first, InputIt last,
                 UnaryPredicate p );
```

- [first, last) 구간에서 f의 결과가 true인 첫번째 element의 iterator 반환
- UnaryPredicate : bool p(T a)
- auto it = find_if(arr.begin(), arr.end(), [](auto x) { return x < 10; });

swap(), max(), min()

void swap(T a, T b)	: a, b의 값을 바꾼다
T max(T a, T b)	: operator< 기준 큰 값
T max(T a, T b, Compare comp)	: comp 기준(Compare requirement 만족)
T min(T a, T b)	: operator< 기준 작은 값
T min(T a, T b, Compare comp)	: comp 기준(Compare requirement 만족)
pair<T,T> minmax(T a, T b)	: operator< 기준 (first:작은 값 , second:큰 값)
pair<T,T> minmax(T a, T b, Compare comp)	: comp 기준(Compare requirement 만족)

lower_bound(), upper_bound()

```
template< class ForwardIt, class T >
ForwardIt lower_bound( ForwardIt first, ForwardIt last, const T& value );

template< class ForwardIt, class T, class Compare >
ForwardIt lower bound( ForwardIt first, ForwardIt last, const T& value, Compare comp );
```

lower_bound

- [first, last) 구간에서 comp 기준으로 value보다 크거나 같은 첫번째 element iterator 반환
- auto it = lower_bound(arr.begin(), arr.end(), 5);

upper_bound

- [first, last) 구간에서 comp 기준으로 value보다 큰 첫번째 element iterator 반환
- auto it = upper_bound(arr.begin(), arr.end(), 5);

- comp 명시되지 않았을 때는 operator< 기준
- 정렬된 구간에서만 사용
- binary search 기반, RandomIt : O(log n), Otherwise : O(n)
- comp 기준은 Compare requirement 필요없이 BinaryPredicate만 만족

min_element(), max_element(), minmax_element()

```
template< class ForwardIt >
ForwardIt max_element( ForwardIt first, ForwardIt last );

template< class ForwardIt, class Compare >
ForwardIt max element( ForwardIt first, ForwardIt last, Compare comp );
```

max_element

- [first, last) 구간에서 comp 기준으로 가장 큰 element iterator 반환
- comp가 없다면 operator< 기준

min_element

- max_element와 형식, 사용방법 동일

minmax_element

- pair<ForwardIt, ForwardIt> 반환
  first: min iterator , second: max iterator

for_each()

```
template< class InputIt, class UnaryFunction >
UnaryFunction for_each( InputIt first, InputIt last, UnaryFunction f );
```

[first, last) 구간의 모든 element에 대해 f를 수행한다.

UnaryFunction

```
vector<int> arr{ 1,2,3,4,5 };

for_each(arr.begin(), arr.end(), [](int x) { x++; });  // 1, 2, 3, 4, 5
for_each(arr.begin(), arr.end(), [](int& x) { x++; }); // 2, 3, 4, 5, 6
```

count(), count_if()

count()

```
template< class InputIt, class T >
typename iterator_traits<InputIt>::difference_type
    count( InputIt first, InputIt last, const T &value );
```

- [first, last) 구간에서 값이 value인 개수를 반환
- int cnt = count(arr.begin(), arr.end(), 1);

---

count_if()

```
template< class InputIt, class UnaryPredicate >
typename iterator_traits<InputIt>::difference_type
    count_if( InputIt first, InputIt last, UnaryPredicate p );
```

- [first, last) 구간에서 p의 결과가 true인 element의 개수를 반환
- UnaryPredicate : bool p(T a)
- int cnt = count_if(arr.begin(), arr.end(), [](auto x) { return x < 10; });

remove(), remove_if()

remove()

```
template< class ForwardIt, class T >
ForwardIt remove( ForwardIt first, ForwardIt last, const T& value );
```

: [first, last) 구간에서 값이 value인 element를 모두 지우고 새로운 past-the-end iterator를 반환

: 실제 해당 객체의 end() iterator가 변경되는건 아니므로 필요하면 erase를 통해 지워야 한다.

```
auto it = remove(arr.begin(), arr.end(), 1);
erase(it, arr.end());
```

※ past-the-end iterator : arr.end()

---

remove_if()

```
template< class ForwardIt, class UnaryPredicate >
ForwardIt remove_if( ForwardIt first, ForwardIt last, UnaryPredicate p );
```

: [first, last) 구간에서 p를 만족하는 element를 모두 지우고 새로운 past-the-end iterator를 반환

: UnaryPredicate : bool f(T a)

```
auto it = remove_if(arr.begin(), arr.end(), [](auto x) { return x < 10; });
erase(it, arr.end());
```


감사합니다