

# C++ Programming

17<sup>th</sup> Study: Standard Template Library #3  
- algorithm



C++ Korea 윤석준 (icysword77@gmail.com)

The background is a solid blue color. Scattered across the background are several squares of various sizes, some of which are formed by a dotted pattern, creating a geometric, pixelated effect.

# algorithm

## 알고리즘

# algorithm

- STL 컨테이너의 원소를 조사, 변경, 관리, 처리할 목적
- STL 컨테이너 뿐만 아니라 일반 배열 구조에도 적용이 가능
- 100 여가지 알고리즘을 STL에서 제공

# algorithm의 분류

- 변경 불가 알고리즘 : 원소 값을 변경하지 않음
- 변경 가능 알고리즘 : 원소 값을 변경함
- 정렬 알고리즘 : 원소의 순서를 변경함
- 범용 수치 알고리즘 : 집계 수치 연산

# Non-modifying sequence algorithm

변경 불가 알고리즘

# Non-modifying sequence algorithm

## Non-modifying sequence operations:

<b>all_of</b> <small>C++11</small>	Test condition on all elements in range (function template )
<b>any_of</b> <small>C++11</small>	Test if any element in range fulfills condition (function template )
<b>none_of</b> <small>C++11</small>	Test if no elements fulfill condition (function template )
<b>for_each</b>	Apply function to range (function template )
<b>find</b>	Find value in range (function template )
<b>find_if</b>	Find element in range (function template )
<b>find_if_not</b> <small>C++11</small>	Find element in range (negative condition) (function template )
<b>find_end</b>	Find last subsequence in range (function template )
<b>find_first_of</b>	Find element from set in range (function template )
<b>adjacent_find</b>	Find equal adjacent elements in range (function template )
<b>count</b>	Count appearances of value in range (function template )
<b>count_if</b>	Return number of elements in range satisfying condition (function template )
<b>mismatch</b>	Return first position where two ranges differ (function template )
<b>equal</b>	Test whether the elements in two ranges are equal (function template )
<b>is_permutation</b> <small>C++11</small>	Test whether range is permutation of another (function template )
<b>search</b>	Search range for subsequence (function template )
<b>search_n</b>	Search range for elements (function template )

<http://www.cplusplus.com/reference/algorithm>

# std::find

- 컨테이너에 있는 데이터 중 원하는 것을 찾기 위한 알고리즘
- 리턴 타입은 iterator이며, (찾으면 해당 위치, 못 찾으면 end())
- 인자로써 iterator 2개 (검색할 범위), 찾을 값을 받음

```
template<class InputIterator, class Type>  
InputIterator find(InputIterator _First, InputIterator _Last, const Type& _Val);
```

# std::find 예제

```
#include <iostream>
#include <vector>
#include <algorithm>

void main()
{
    std::vector<int> V{ 10, 20, 30, 40, 50 };

    auto it30 = std::find(V.begin(), V.end(), 30);
    auto it25 = std::find(V.begin(), V.end(), 25);

    if (it30 != V.end())
        std::cout << "Find 30 !" << std::endl;
    if (it25 != V.end())
        std::cout << "Find 25 !" << std::endl;
}
```



# std::find\_if

- 컨테이너에 있는 데이터 중 조건에 맞는 것을 찾기 위한 알고리즘
- 리턴 타입은 iterator이며, (찾으면 첫 번째 인자의 위치, 못 찾으면 end())
- 인자로써 iterator 2개 (검색할 범위), 찾을 조건을 받음

```
template<class InputIterator, class Predicate>  
InputIterator find_if(InputIterator _First, InputIterator _Last, Predicate _Pred);
```

# std::find\_if 예제

```
#include <iostream>
#include <vector>
#include <algorithm>

void main()
{
    std::vector<int> V{ 10, 20, 30, 40, 50 };

    auto GT25 = [](int n) { return n > 25; };

    auto itFI = std::find_if(V.begin(), V.end(), GT25);

    if (itFI != V.end())
        std::cout << (*itFI) << std::endl;
}
```

# Modfying sequence algorithm

변경 가능 알고리즘

# Modifying sequence algorithm

## Modifying sequence operations:

<code>copy</code>	Copy range of elements (function template )
<code>copy_n</code> <small>(C++11)</small>	Copy elements (function template )
<code>copy_if</code> <small>(C++11)</small>	Copy certain elements of range (function template )
<code>copy_backward</code>	Copy range of elements backward (function template )
<code>move</code> <small>(C++11)</small>	Move range of elements (function template )
<code>move_backward</code> <small>(C++11)</small>	Move range of elements backward (function template )
<code>swap</code>	Exchange values of two objects (function template )
<code>swap_ranges</code>	Exchange values of two ranges (function template )
<code>iter_swap</code>	Exchange values of objects pointed by two iterators (function template )
<code>transform</code>	Transform range (function template )
<code>replace</code>	Replace value in range (function template )
<code>replace_if</code>	Replace values in range (function template )
<code>replace_copy</code>	Copy range replacing value (function template )
<code>replace_copy_if</code>	Copy range replacing value (function template )
<code>fill</code>	Fill range with value (function template )
<code>fill_n</code>	Fill sequence with value (function template )
<code>generate</code>	Generate values for range with function (function template )
<code>generate_n</code>	Generate values for sequence with function (function template )
<code>remove</code>	Remove value from range (function template )
<code>remove_if</code>	Remove elements from range (function template )
<code>remove_copy</code>	Copy range removing value (function template )
<code>remove_copy_if</code>	Copy range removing values (function template )
<code>unique</code>	Remove consecutive duplicates in range (function template )
<code>unique_copy</code>	Copy range removing duplicates (function template )
<code>reverse</code>	Reverse range (function template )
<code>reverse_copy</code>	Copy range reversed (function template )

<http://www.cplusplus.com/reference/algorithm>

# std::remove

- 컨테이너에 있는 데이터 중 특정 값을 삭제하기 위한 알고리즘
- 삭제 후에도 컨테이너의 크기가 변하지 않음

삭제 대상을 뒤쪽으로 옮겨 놓음

완전히 지우려면 erase()를 이용하여 리턴된 위치부터 end()까지 지워야 함

```
template<class InputIterator, class Type>  
InputIterator remove(InputIterator _First, InputIterator _Last, const Type& _Val);
```

# std::remove 예제

```
#include <iostream>
#include <vector>
#include <algorithm>

void main()
{
    std::vector<int> V{ 10, 20, 30, 40, 50 };

    std::cout << "V.size() = " << V.size() << std::endl;

    auto itR = std::remove(V.begin(), V.end(), 40);

    if (itR != V.end())
    {
        std::cout << "After remove() : " << V.size() << std::endl;
        V.erase(itR, V.end());
        std::cout << "After erase() : " << V.size() << std::endl;
    }
}
```

# std::remove\_if

- 컨테이너에 있는 데이터 중 조건에 맞는 값을 삭제하기 위한 알고리즘
- 삭제 후에도 컨테이너의 크기가 변하지 않음

삭제 대상을 뒤쪽으로 옮겨 놓음

완전히 지우려면 erase()를 이용하여 리턴된 위치부터 end()까지 지워야 함

```
template<class InputIterator, class Predicate>  
InputIterator remove_if(InputIterator _First, InputIterator _Last, Predicate _Pred);
```

# std::remove\_if 예제

```
#include <iostream>
#include <vector>
#include <algorithm>

void main()
{
    std::vector<int> V{ 10, 15, 30, 45, 50 };

    auto isOdd = [](int n) { return n % 2 == 1; };

    auto itRI = std::remove_if(V.begin(), V.end(), isOdd);

    if (itRI != V.end())
    {
        std::cout << "After remove_if() : " << V.size() << std::endl;
        V.erase(itRI, V.end());
        std::cout << "After erase() : " << V.size() << std::endl;
    }
}
```



# Sorting/Merge algorithm

정렬 알고리즘

# Sorting/Merge algorithm

## Sorting:

<b>sort</b>	Sort elements in range (function template )
<b>stable_sort</b>	Sort elements preserving order of equivalents (function template )
<b>partial_sort</b>	Partially sort elements in range (function template )
<b>partial_sort_copy</b>	Copy and partially sort range (function template )
<b>is_sorted</b> <small>(C++11)</small>	Check whether range is sorted (function template )
<b>is_sorted_until</b> <small>(C++11)</small>	Find first unsorted element in range (function template )
<b>nth_element</b>	Sort element in range (function template )

## Merge (operating on sorted ranges):

<b>merge</b>	Merge sorted ranges (function template )
<b>inplace_merge</b>	Merge consecutive sorted ranges (function template )
<b>includes</b>	Test whether sorted range includes another sorted range (function template )
<b>set_union</b>	Union of two sorted ranges (function template )
<b>set_intersection</b>	Intersection of two sorted ranges (function template )
<b>set_difference</b>	Difference of two sorted ranges (function template )
<b>set_symmetric_difference</b>	Symmetric difference of two sorted ranges (function template )

<http://www.cplusplus.com/reference/algorithm>

# std::sort

- 컨테이너에 있는 데이터를 기준에 맞게 정렬하기 위한 알고리즘
- 정렬 기준은 STL에서 기본적으로 제공해주는 함수 개체를 사용해도 되며, 사용자가 직접 작성한 함수나 Lambda 도 가능  
(기본형에 대해서는 생략이 가능하며, 생략하면 오름차순)

```
template<class RandomAccessIterator, class Pr>  
void sort(RandomAccessIterator First, RandomAccessIterator _Last,  
          BinaryPredicate _Comp);
```

# std::sort 예제

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

void main()
{
    std::vector<int> V{ 10, 30, 15, 50, 45 };

    for (auto it = V.begin(); it != V.end(); it++)
        std::cout << (*it) << '\t';
    std::cout << std::endl;

    std::sort(V.begin(), V.end(), std::less<int>());

    for (auto it = V.begin(); it != V.end(); it++)
        std::cout << (*it) << '\t';
    std::cout << std::endl;
}
```

# std::merge

- 정렬된 컨테이너 2개를 합치면서 정렬하기 위한 알고리즘
- 2개의 컨테이너의 시작 끝 위치(4개)와 합쳐진 값이 들어갈 위치, 정렬하기 위한 기준을 인자로 받음

```
template<class InputIterator1, class InputIterator2,  
         class OutputIterator, class BinaryPredicate>  
OutputIterator merge(InputIterator1 _First1, InputIterator1 _Last1,  
                    InputIterator2 First2, InputIterator2 Last2,  
                    OutputIterator _Result, BinaryPredicate _Comp);
```

# std::merge 예제

```
#include <iostream>
#include <vector>
#include <deque>
#include <algorithm>
#include <functional>

void main()
{
    std::vector<int> V{ 10, 30, 15, 50, 45 };
    std::sort(V.begin(), V.end(), std::less<int>());

    std::deque<int> V2{ 13, 78, 57, 24, 69 };
    std::sort(V2.begin(), V2.end(), std::less<int>());

    std::vector<int> VR;
    VR.resize(V.size() + V2.size());

    auto isLess = [](int a, int b) { return a < b; };

    std::merge(V.begin(), V.end(), V2.begin(), V2.end(), VR.begin(), isLess);

    for (auto it = VR.begin(); it != VR.end(); it++)
        std::cout << (*it) << '\t';
    std::cout << std::endl;
}
```


The background is a solid blue color. It features several white squares of various sizes, some of which are outlined with a dotted pattern. These squares are scattered across the slide, with some appearing as simple white shapes and others as dotted outlines. The text is centered on the left side of the slide.

# Numeric algorithm

## 수치 알고리즘

# Numeric algorithm

## *fx* Functions

<b>accumulate</b>	Accumulate values in range (function template )
<b>adjacent_difference</b>	Compute adjacent difference of range (function template )
<b>inner_product</b>	Compute cumulative inner product of range (function template )
<b>partial_sum</b>	Compute partial sums of range (function template )
<b>iota</b> 	Store increasing sequence (function template )

<http://www.cplusplus.com/reference/numeric>



# std::accumulate

- 컨테이너에 요소들의 합을 구하기 위한 알고리즘
- 기본 연산은 더하기지만, 다른 연산을 직접 지정도 가능
- 인자로 연산의 범위(2개), 집계 시작값, 연산 함수

```
template<class InputIterator, class Type, class BinaryOperation>  
Type accumulate(InputIterator First, InputIterator Last,  
                Type _Val,          BinaryOperation _Binary_op);
```

# std::accumulate 예제

```
#include <iostream>
#include <vector>
#include <numeric>

void main()
{
    std::vector<int> V{ 1, 2, 3, 4};

    int nSum = std::accumulate(V.begin(), V.end(), 0);           // vector의 합
    int nSum10 = std::accumulate(V.begin(), V.end(), 10);        // 10 + vector의 합

    auto multi = [](int a, int b) { return a * b; };
    int nMulti = std::accumulate(V.begin(), V.end(), 1, multi);  // 인자의 곱

    auto nSquire = [](int a, int b) { return a + b * b; };
    int nSumSquire = std::accumulate(V.begin(), V.end(), 0, nSquire); // 제곱의 합
}
```