Lecture 11.16

Standard Template Library - Algorithm

SE271 Object-Oriented Programming (2020)

Yeseong Kim

Original slides from Prof. Shin at DGIST

Short Notice

- 수요일 (18일) 수시 면접 일정으로 인해 휴강합니다

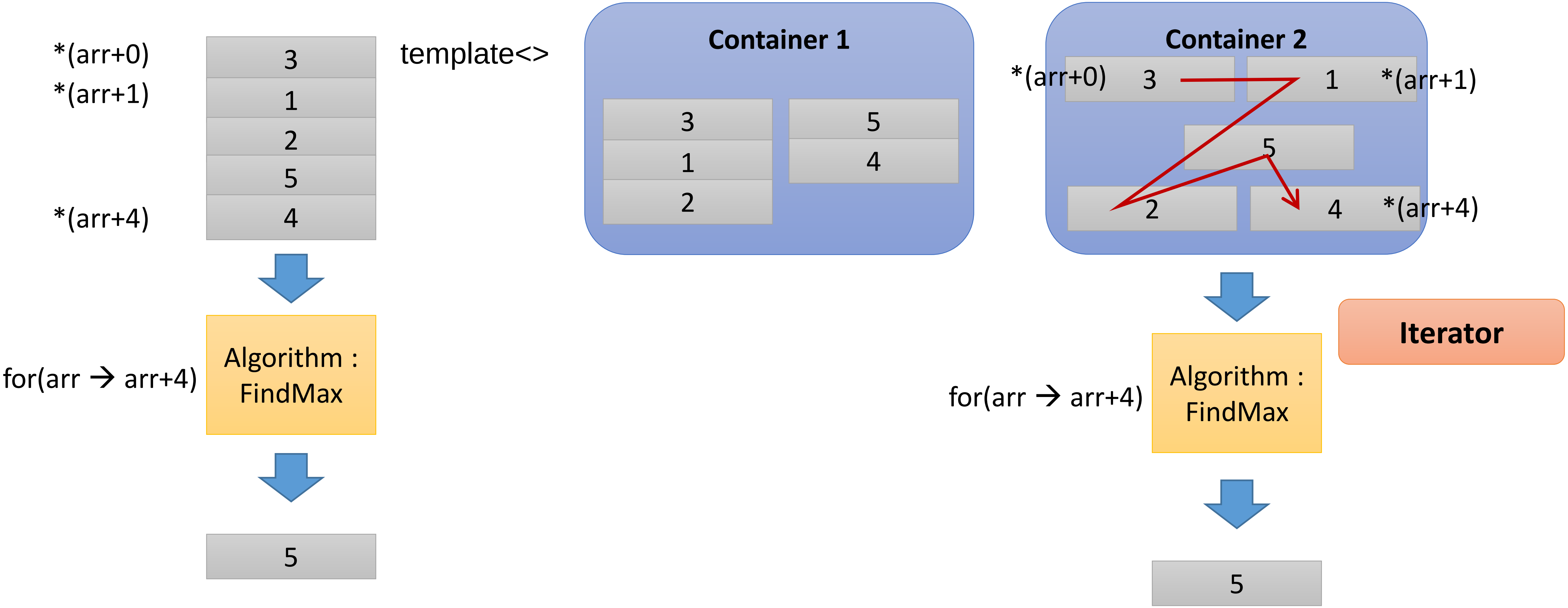
- Team Project
  - Will have a presentation with a recorded video (4 minutes for each team)
  - Will write a report (perhaps 3~5 pages)

Today’s Topic

- Algorithm
- Functor
- Lambda

STL: Standard Template Library

int arr[5]={3,1,2,5,4};



*(arr+0)

*(arr+1)

*(arr+4)

3

1

2

5

4

for(arr → arr+4)

Algorithm : FindMax

5

template<>

Container 1

3

1

2

5

4

Container 2

*(arr+0)

3

1

*(arr+1)

5

2

4

*(arr+4)

Iterator

for(arr → arr+4)

Algorithm : FindMax

5

Algorithms

- STL provides most common algorithms (e.g., sort, search) on elements stored in a container
  - An algorithm is a **function template** operating on sequences of elements
- Iterators are used to identify input and/or output
  - Two iterators are often used to specify range of input
  - Algorithm may return an iterator, a value, or modify elements in an output iterator (e.g., copy())
- Some algorithms (e.g., replace(), sort()) modify elements in a container, but no algorithm add or remove elements of a container
- STL library provides generic programming, i.e., a style of computer programming in which algorithms are written in terms of types to-be-specified-later that are then instantiated when needed for specific types provided as parameters

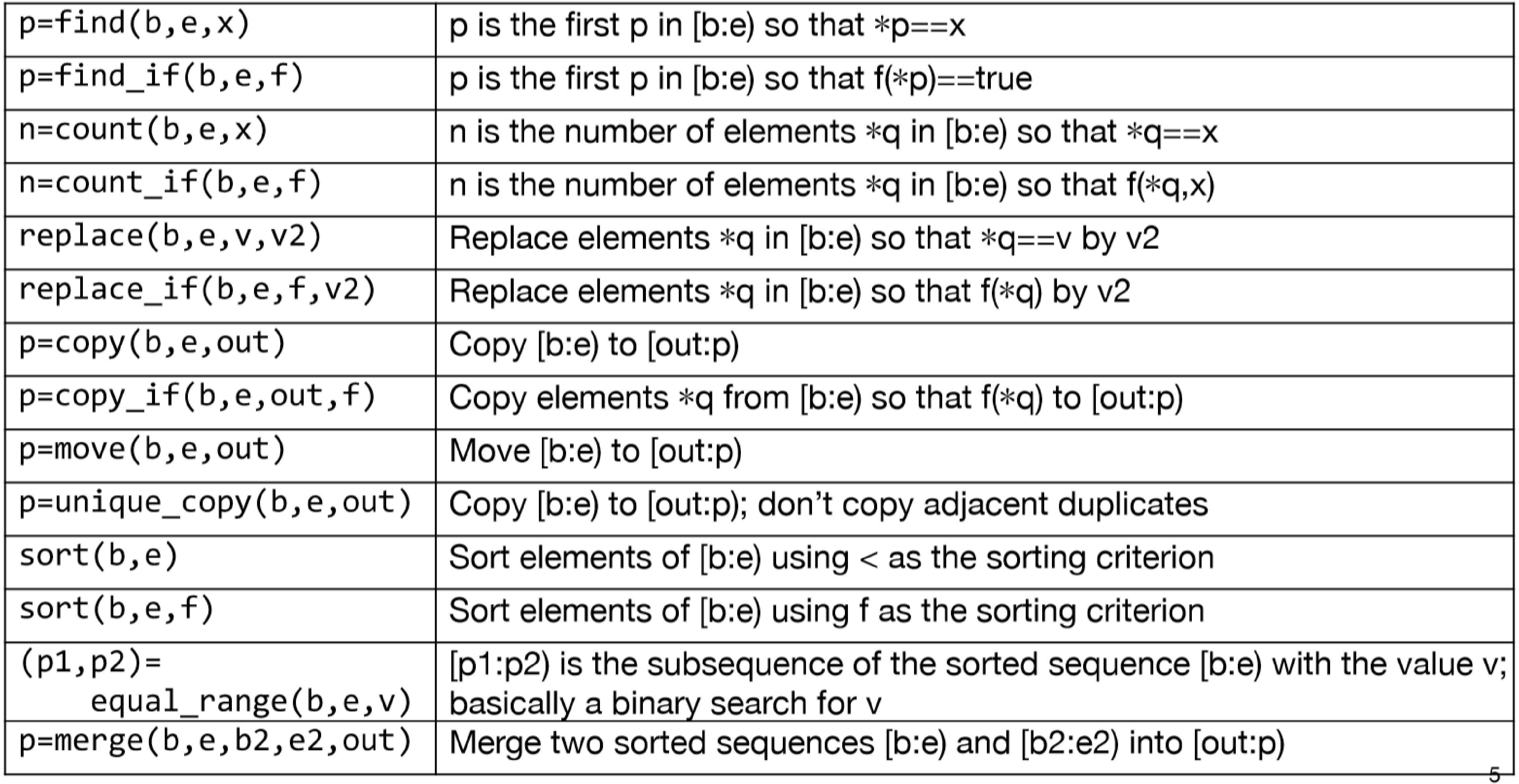
Types of Algorithms

- Non-modifying sequence operations
- Modifying sequence operations
- Partitioning operations
- Sorting operations
- Binary search operations
- Set operations
- Heap operations
- Minimum/maximum operations
- Numeric operations

```
#include <algorithm>
#include <numeric>
```

https://en.cppreference.com/w/cpp/algorithm

Examples



p=find(b,e,x)	p is the first p in [b:e) so that *p==x
p=find_if(b,e,f)	p is the first p in [b:e) so that f(*p)==true
n=count(b,e,x)	n is the number of elements *q in [b:e) so that *q==x
n=count_if(b,e,f)	n is the number of elements *q in [b:e) so that f(*q,x)
replace(b,e,v,v2)	Replace elements *q in [b:e) so that *q==v by v2
replace_if(b,e,f,v2)	Replace elements *q in [b:e) so that f(*q) by v2
p=copy(b,e,out)	Copy [b:e) to [out:p)
p=copy_if(b,e,out,f)	Copy elements *q from [b:e) so that f(*q) to [out:p)
p=move(b,e,out)	Move [b:e) to [out:p)
p=unique_copy(b,e,out)	Copy [b:e) to [out:p); don’t copy adjacent duplicates
sort(b,e)	Sort elements of [b:e) using < as the sorting criterion
sort(b,e,f)	Sort elements of [b:e) using f as the sorting criterion
(p1,p2)= equal_range(b,e,v)	[p1:p2) is the subsequence of the sorted sequence [b:e) with the value v; basically a binary search for v
p=merge(b,e,b2,e2,out)	Merge two sorted sequences [b:e) and [b2:e2) into [out:p)

Algorithm: find (p = find(b,e,x))

```
template<typename InputIt, typename T>

constexpr InputIt find(InputIt first, InputIt last,
const T& value) {

    for (; first != last; ++first) {

        if (*first == value) {

            return first;

        }

    }

    return last;

}
```

```
int main() {
    int n1 = 3, n2 = 5;
    std::vector<int> v{ 0, 1, 2, 3, 4 };
    auto result1 = std::find(std::begin(v), std::end(v), n1);
//      auto result2 = std::find(v.begin(), v.end(), n2);

    if (result1 != std::end(v))
        std::cout << "v contains: " << n1 << '\n';
    else
        std::cout << "v does not contain: " << n1 << '\n';
}
```

Algorithm: sort (sort(b,e))

```
#include <iostream>

#include <algorithm>

#include <vector>


int main(){

    std::vector<int> v{ 3, -4, 5, -6, 10 };

    std::sort(v.begin()+1, v.end()-1);

    for (auto i : v) {

        std::cout << i << " ";

    }

}
```

Algorithm: merge (p = merge(b,e,b2,e2,out))

```
template<class InputIt1, class InputIt2, class OutputIt>
OutputIt merge(InputIt1 first1, InputIt1 last1, InputIt2 first2, InputIt2 last2, OutputIt d_first){…}
```

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

int main(){
    std::vector<int> v1{ 1,2,3,4 }, v2{4,5,6,7};
    // merge
    std::vector<int> dst(v1.size()+v2.size());
    std::merge(v1.begin(), v1.end(), v2.begin(), v2.end(), dst.begin());
    std::for_each(dst.begin(), dst.end(), Print);
}
```

[Recap] Function Pointer

- **Syntax**

```
// declaration
return_type (* function_pointer) (parameters);
// assignment
function_pointer = function_name;
```

- **Example**

```
int iMenu{ 0 };
int iNum1{ 1 }, iNum2{ 2 };
int (*func_ptr) (int, int);
cin >> iMenu;
func_ptr = (iMenu == 1) ? Add : Sub;
cout <<"Result : " << func_ptr(iNum1, iNum2);


func_ptr = (iMenu == 1) ? f(Add) : f(Sub);
```

```
void f (int (*func_ptr) (int, int) ) {
    cout << func_ptr(iNum1, iNum2);
}
```

Function Object (functor)

- Function object
  - Function object is a function-like object
  - Has the same function with status (cf. function)
  - Are able to use it as a template parameter

```cpp
template<typename T>
T Plus(T n1, T n2) {
    return n1+n2;
}

cout << Plus<int>(10, 20);
```

```cpp
template<typename T>
class Plus {
public:
    T operator()(T n1, T n2){
        return n1+n2;
    }
};
Plus<int> p;
cout << p.operator()(10, 20);
cout << p(10, 20);
cout << Plus<int>()(10, 20);
```

Function Object

- STL functors (#include <functional>)

Function Object	Operator	Function Object	Operator
plus	+	greater	>
minus	-	greater_equal	>=
multiplies	*	less	<
divides	/	less_equal	<=
modulus	%	logical_and	&&
negate	-	logical_or	||
equal_to	==	logical_not	!
not_equal_to	!=		

```
#include <functional>
plus<int> iP;
greater<int> iG;

cout << iP(1, 2);
cout << iG(1, 2);

cout << minus<int>()(2, 1);
```

Algorithm: for_each (f = for_each(b,e,f))

```
template<typename InputIt, typename
UnaryFunction>

constexpr UnaryFunction
for_each(InputIt first, InputIt last,
UnaryFunction f){

    for (; first != last; ++first) {

        f(*first);

    }

    return f;//implicit move since C++11

}


                    void fun(const Type &a);
```

```
void Abs(int& n) {
    if (n < 0) n *= -1;
}
void Print(int& n) {
    std::cout << n << " ";
}
int main(){
    int arr[] = { 3, -4, 5, -6, 10 };
    std::for_each(arr, arr+5, Abs);
    for (auto i : arr) {
        std::cout << i << " ";
    }
    std::for_each(arr, arr + 5, Print);
}
```

https://en.cppreference.com/w/cpp/algorithm/for_each

Algorithm: sort (sort(b,e) or sort(b,e,f))

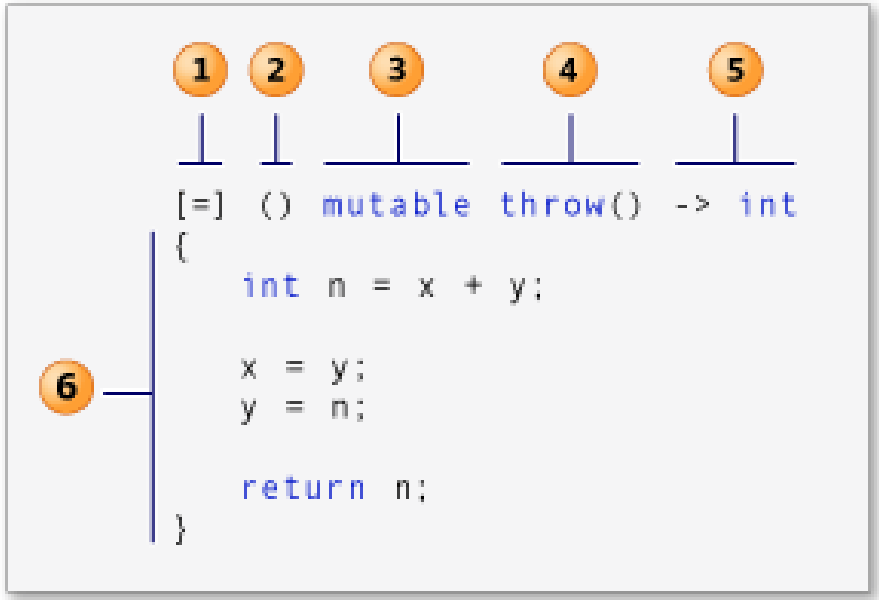
```
#include <iostream>
#include <algorithm>
#include <vector>

int main(){
    std::vector<int> v{ 3, -4, 5, -6, 10 };
    std::sort(v.begin(), v.end());
    for (auto i : v) {
        std::cout << i << " ";
    }
}
```

```
#include <functional>
int main(){
    std::vector<int> v{ 3, -4, 5, -6, 10 };
    std::greater<int> iG;
    std::sort(v.begin(), v.end(), iG);
    std::sort(v.begin(), v.end(), std::greater<int>());
    for (auto i : v) {
        std::cout << i << " ";
    }
}
```

Lambda Expression

- Supported in C++11 and later
  - A convenient way of defining an anonymous function object
  - Encapsulate a few lines of code that are passed to algorithms or asynchronous methods



```
[=] () mutable throw() -> int
{
    int n = x + y;

    x = y;
    y = n;

    return n;
}
```

1. *capture clause* (Also known as the *lambda-introducer* in the C++ specification.)
2. *parameter list* Optional. (Also known as the *lambda declarator*)
3. *mutable specification* Optional.
4. *exception-specification* Optional.
5. *trailing-return-type* Optional.
6. *lambda body*.

https://docs.microsoft.com/ko-kr/cpp/cpp/lambda-expressions-in-cpp?view=msvc-160

Simple Lambda Example With STL

```
[CAPTURE](PARAMETERS){ BODY }
```

```
int main(){
    std::vector<int> v1{ 1,2,3,4 }, v2{4,5,6,7};
    // merge
    std::vector<int> dst(v1.size()+v2.size());
    std::merge(v1.begin(), v1.end(), v2.begin(), v2.end(), dst.begin());
    std::for_each(dst.begin(), dst.end(),
        [](auto& v) {
            std::cout << v << std::endl;
        }
    );
}
```

References

- Learn C++ (https://www.learncpp.com/)
  - STL: Ch. 16

- STL
  - https://en.cppreference.com/w/cpp/algorithm

Q

ANY QUESTIONS?