Passing function as argument

- Function Pointer: pointer to function (can be stored in arrays, copied, assigned)

```
int f(int n) {
    std::cout << n << '₩n';
    return n * n;
}
int main() {
    int (*pFunc)(int) = f;
    //std::function<int(int)> pFunc = f;
    int x = pFunc(7);
}
```

```
//std::function<bool(int, int)> compare
void Sort(int* arr, int size,
                    bool (*compare)(int, int)){
    for(int i=0; i<size-1; ++i)
        for(int j=i+1; j<size; ++j)
            if (compare(arr[i], arr[j])) swap(...);
}
bool asc(int a, int b) { return a > b; }
bool desc(int a, int b) { return a < b; }
int main(){
    int arr[3] {2, 1, 3};
    Sort(arr, 3, asc); Print(arr, 3);
    Sort(arr, 3, desc); Print(arr, 3);
}
```

https://en.cppreference.com/w/cpp/language/pointer

Functor (function call operator)

- When a user-defined class overloads the function call operator, operator(), it becomes a FunctionObject type.
- An object of such a type can be used in a function-call-like expression:

```
struct Linear {
    double a, b;
    double operator()(double x) {
        return a*x + b;
    }
};
int main() {
    Linear f{2, 1};
    double f_0 = f(0);
}
```

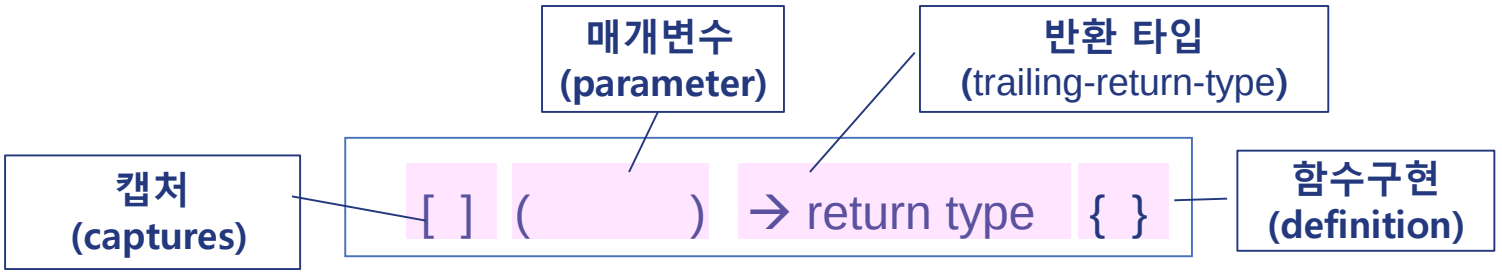
```
struct Sum {
    int sum;
    Sum() : sum(0) { }
    void operator()(int n) { sum += n; }
};

Sum s = std::for_each(v.begin(), v.end(), Sum());
//void Sort(int* arr, int size,
//                              Sum& compare) {...}
```

https://en.cppreference.com/w/cpp/language/operators

Lambda function

- unnamed function object capable of capturing variables in scope.




```
void Sort(int* arr, int size,
                    bool (*compare)(int, int)){
    for(int i=0; i<size-1; ++i)
        for(int j=i+1; j<size; ++j)
            if (compare(arr[i], arr[j])) swap(...);
}

Sort(arr, 3, [ ] (int a, int b) → bool { return a >b; } ) ;
```

https://en.cppreference.com/w/cpp/language/lambda