



C++ Korea와 함께하는
제2회 마이크로소프트 멜팅팟 세미나

빠르게 변화하는 모든 C++에 몸을 맡겨라!

2016. 5. 28(토) 13:00 ~ 18:00
한국마이크로소프트 대회의실(11층)



Ranges for The C++ Standard Library

최동민 / Nexon Korea



- 다를 것
 - Ranges와 Ranges를 이루는 문법들
- 다루지 않을 것
 - Concept
 - 시간의 제약
- 코드 생략
 - const, constexpr, explicit, overload, noexcept, namespace 등
 - 가독성을 위해

당신의 코드는 안녕하십니까?

문제 인식



- 태초에 **Iterator** 가 있었습니다.

```
for (std::vector<int>::const_iterator it = numbers.begin();  
     it != numbers.end(); ++it)  
{  
    // ***  
}
```

- 매우 불편했습니다.

C++11 – 혁신의 바람

빠르게 변화하는
모던 C++에 몸을 맡겨라!



“내 너희에게 **auto**와
range-based for 를 주노라”

C++11 – 혁신의 바람

빠르게 변화하는
모던 C++에 몸을 맡겨라!

```
for (std::vector<int>::const_iterator it = numbers.begin();  
     it != numbers.end(); ++it)  
{  
    // ***  
}
```



```
for (auto e : numbers)  
{  
    // ***  
}
```

해피엔딩?

빠르게 변화하는
모던 C++에 몸을 맡겨라!



아직도 풀리지 않은 숙제

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 나는 컨테이너 전체를 소팅 하고 싶은데

왜 `begin()`, `end()` 요?

```
std::vector<int> numbers{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};  
std::sort(numbers.begin(), numbers.end());
```

- 오늘 소개할 Ranges 라이브러리 사용 시

```
std::sort(numbers.begin(), numbers.end());
```



```
ranges::sort(numbers);
```

- 모든 STL 알고리즘 함수에 대해
범위 기반 알고리즘 함수 제공

안녕하세요. 처음 뵙겠습니다.

Ranges

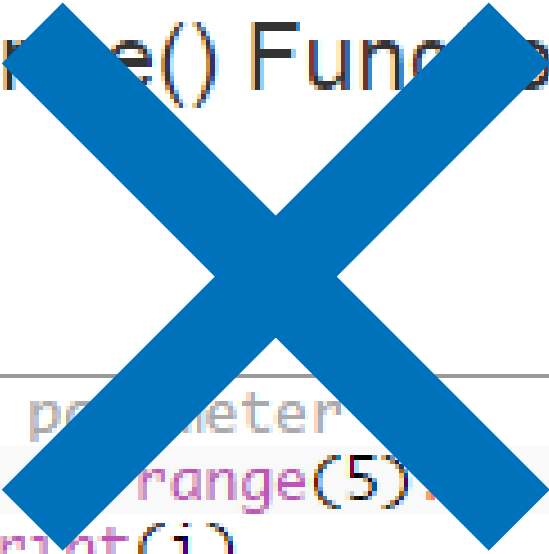


Ranges?

빠르게 변화하는
모던 C++에 몸을 맡겨라!

Python's range() Function Examples

Simple Usage



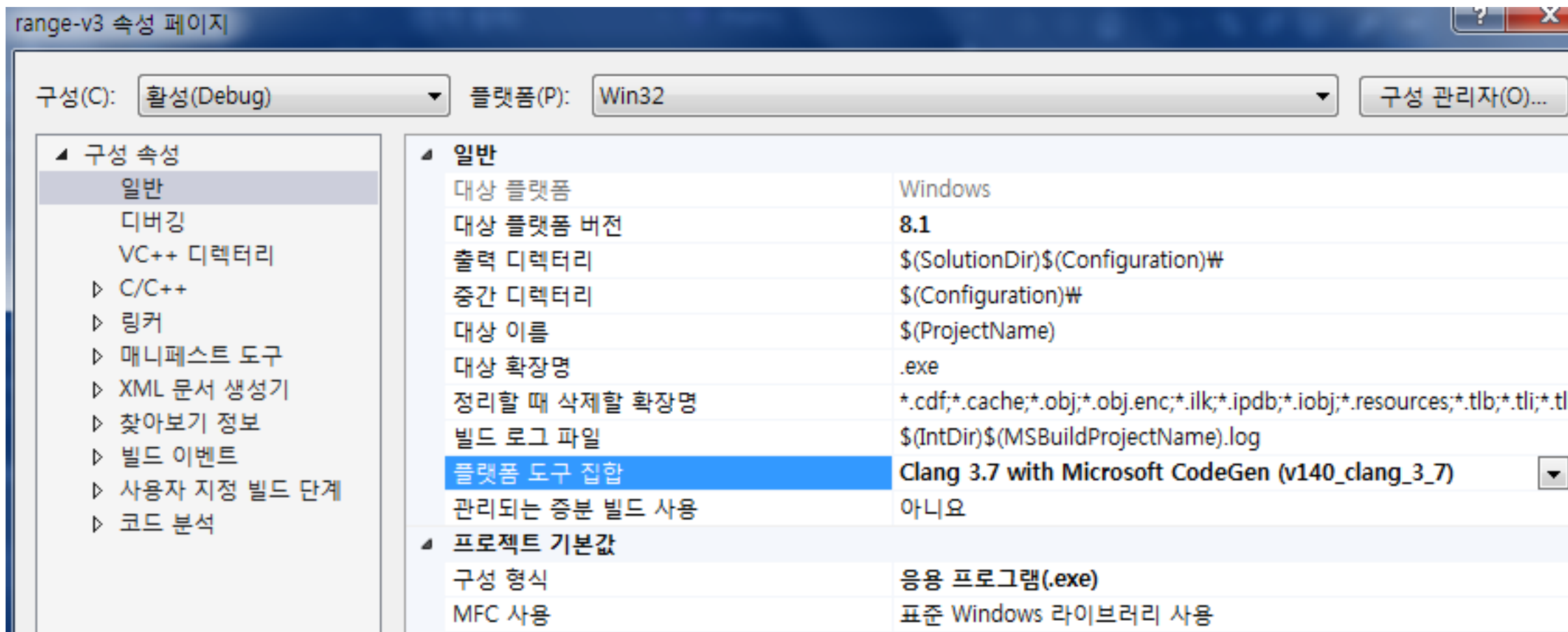
```
1 >>> # One parameter
2 >>> for i in range(5):
3 ...     print(i)
```

- 범위를 다루는 Generic Library
 - <Begin, End>
- 지원 컴파일러
 - clang 3.4.0 이상
 - gcc 4.9.0 이상
- 차기 C++ 표준 포함 예정
- <https://github.com/ericniebler/range-v3>

Ranges

빠르게 변화하는
모던 C++에 몸을 맡겨라!

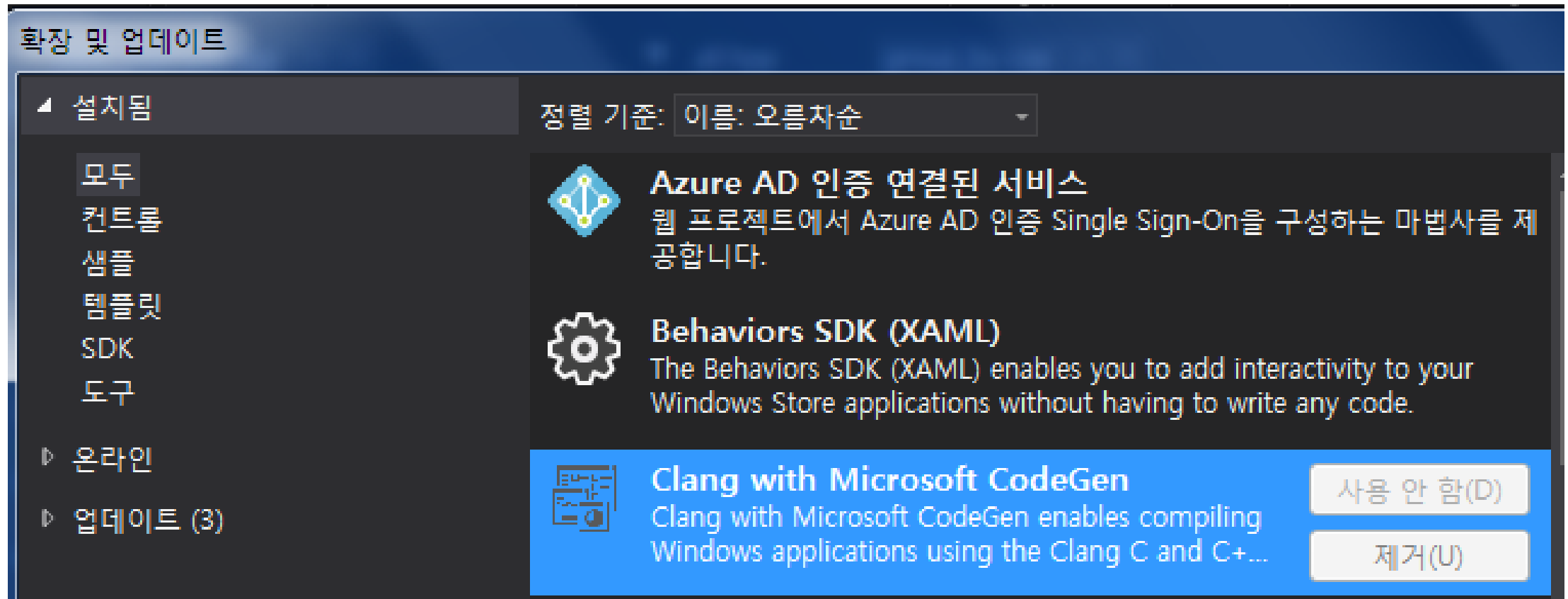
• VS로는 못하나요?



Ranges

빠르게 변화하는
모던 C++에 몸을 맡겨라!

• Clang 3.7 이 없는데요?



Ranges : 범위 기반 알고리즘 제공

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- partition

```
auto it = std::partition(v.begin(), v.end(), [](auto e){  
    return e % 2 == 0;  
})
```



```
auto it = ranges::partition(v, [](auto e){  
    return e % 2 == 0;  
})
```


이게 다입니까?

빠르게 변화하는
모던 C++에 몸을 맡겨라!

!(이게 다예요)
C'est tout

마르그리트 뒤라스 지음 | 고종석 옮김

Ranges 를 왜 써야 하나요?

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 편의성(Convenience)
 - 코드가 짧아져요.(!)
- 느긋한 계산(Lazy Evaluation)
 - 무한 범위 표현 가능
 - 불필요한 계산 회피
- 조합성(Composability)
 - 함수형 프로그래밍에서 내세우는 그것
 - 직관적인 표현 가능
 - 코드 생산성 향상

여러분의 코드에 미칠 영향들

기능 소개



Ranges : 컨테이너로의 전환

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- $[0, 10)$ 으로 채우기

- 2009년의 나

```
int arr[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

- 2010년의 나

```
vector<int> v(10);  
for (int i = 0; i < 10; ++i)  
    v[i] = i;
```

- 2016년의 나

```
vector<int> v = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

Ranges : 컨테이너로의 전환

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- $[0, 10)$ 으로 채우기

- `std::iota()`

```
//#include<numeric>
vector<int> v(10);
std::iota(v.begin(), v.end(), 0);
```

- `ranges::iota()`

```
vector<int> v(10);
ranges::iota(v, 0);
```

Ranges : 컨테이너로의 전환

빠르게 변화하는
모던 C++에 몸을 맡겨라!

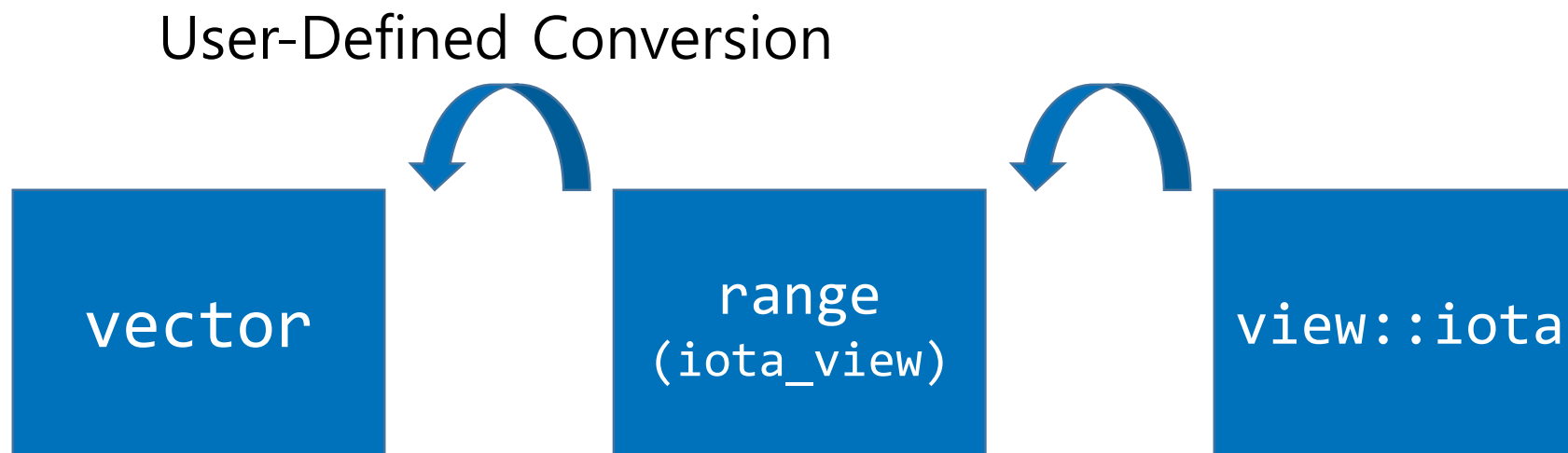
- $[0, 10)$ 으로 채우기
 - `ranges::view::iota()`

```
using namespace ranges;  
vector<int> v = view::iota(0, 10);
```

Ranges : 컨테이너로의 전환

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- `ranges::view::iota()`



User-Defined Conversion

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 분수 클래스 with Getter

```
struct Fraction{
    int numerator;
    int denominator;

    double get(){
        return static_cast<double>(numerator) / denominator;
    }
};

int main(){
    Fraction f{ 10, 20 };
    cout << f.get() << endl; // 0.5
}
```


User-Defined Conversion

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 마치 처음부터 double 이었던 것 처럼

```
struct Fraction{
    int numerator;
    int denominator;

    operator double(){
        return static_cast<double>(numerator) / denominator;
    }
};

int main(){
    Fraction f{ 10, 20 };
    cout << f << endl; // 0.5
}
```

Ranges : 기본 구조

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- `view::iota()` 를 구현해 봅시다.

```
template <class T>
struct simple_iota_view{
    T from_;
    T to_;

    operator vector<T>(){
        vector<T> v;
        v.reserve(to_ - from_);
        for (; from_ != to_; ++from_)
            v.emplace_back(from_);
        return v;
    }
};
```

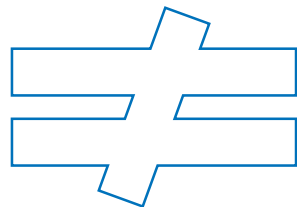
Ranges : 기본 구조

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 비교

현재 결과

```
vector<int> v = simple_iota_view<int>{ 0, 10 };
```



우리가 원하는 결과

```
vector<int> v = view::iota(0, 10);
```

Ranges : 기본 구조

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- Factory Functor

```
struct simple_iota_fn
{
    template<class T>
    simple_iota_view<T> operator()(T beg, T end)
    {
        return { beg, end };
    }
} simple_iota;
```

Ranges : 기본 구조

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 최종 결과

```
vector<int> v = simple_iota(0, 10);
```

Ranges : 기본 구조

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 왜 이렇게 복잡하죠?

```
template <class T>
struct simple_iota_view 구현체 클래스
{
    // ***
};

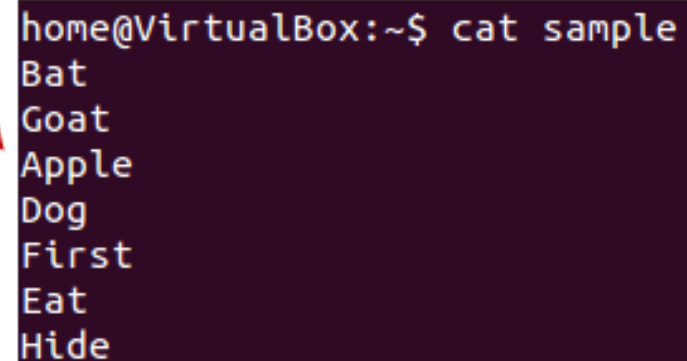
struct simple_iota_fn Functor 클래스
{
    // ***
} simple_iota; 호출할 Functor Instance
```

Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- Linux/Unix 파이프라인
 - Vertical Bar (|)
 - Bar 왼쪽의 연산 결과를
오른쪽 연산의 입력으로 사용

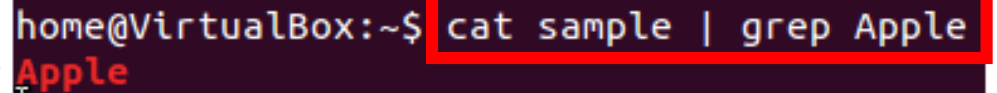
The contents of the 'sample' file



```
home@VirtualBox:~$ cat sample
Bat
Goat
Apple
Dog
First
Eat
Hide
```

A terminal window showing the command 'cat sample' and its output. The output lists several words: Bat, Goat, Apple, Dog, First, Eat, and Hide. A red arrow points from the title 'The contents of the 'sample' file' to the terminal output.

Using 'grep' for searching Apple



```
home@VirtualBox:~$ cat sample | grep Apple
Apple
```

A terminal window showing the command 'cat sample | grep Apple'. The command is highlighted with a red box. The output is 'Apple', which is also highlighted with a red box. A red arrow points from the title 'Using 'grep' for searching Apple' to the terminal output.

현재 C++ 에서는?

```
grep(cat("sample"), "Apple")
```

Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

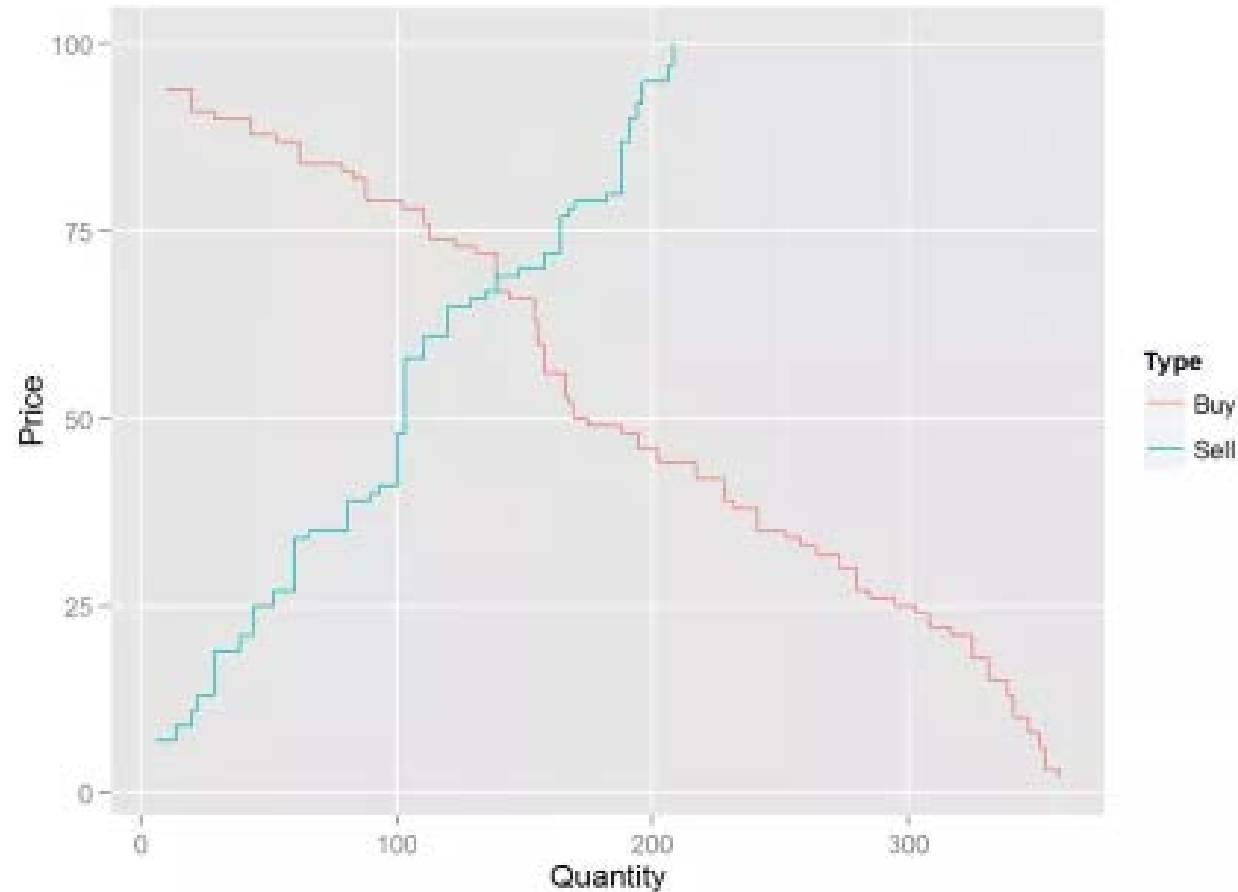
- R 파이프라인
 - %>% 를 통해 전달

```
# Group the data, aggregate the bids, and plot the supply and demand curves.  
auction.data %>%  
  group_by(Type) %>%  
  do(aggregate_bids(.)) %>%  
  qplot(Quantity, Price, col = Type, geom = "step", data = .) %>%  
  print
```


Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- R 파이프라인



Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

```
vector<int> v = view::iota(0, 5) |  
               view::cycle |  
               view::take(15);
```

Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- Vertical Bar (|) 를 통해 전달
- 예시
 - 0 ~ 4 를 반복하는 임의의 길이의 수열을 구해라
 - [0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0]

```
vector<int> v =  
    view::iota(0, 5) |  
    view::cycle |  
    view::take(15);
```

[0, 1, 2, 3, 4] 수열을 만들고
그걸 **무한히** 반복하게 한 뒤
그 중 앞에서 15개(원하는 만큼)만 추린다.

Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 비교 (0 ~ 4 를 반복하는 임의의 길이의 수열을 구해라)

for 루프

```
vector<int> v(15);  
for (int i = 0; i < v.size(); ++i)  
    v[i] = i % 5;
```

std::generate()

```
int n = 0;  
vector<int> v(15);  
std::generate(v.begin(), v.end(), [&n] {  
    return n++ % 5;  
});
```

ranges

```
vector<int> v =  
    view::iota(0, 5) |  
    view::cycle |  
    view::take(15);
```

Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- [0~15) 에서 5개씩 그룹 지어 각 그룹의 합을 구해라
 - $0 + 1 + 2 + 3 + 4 = 10$
 - $5 + 6 + 7 + 8 + 9 = 35 \dots$

```
vector<int> v =  
    view::iota(0, 15) |  
    view::group_by(quotient) |  
    view::transform(sum);
```

```
for (auto e : v)  
    cout << e << " ";  
cout << endl;  
// 10 35 60
```

```
auto quotient = [](int a, int b){  
    return a / 5 == b / 5;  
};
```

```
auto sum = [](auto rng) {  
    return ranges::accumulate(rng, 0);  
};
```

Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- [0~15) 에서 5개씩 그룹 지어 각 그룹의 합을 구해라
 - $0 + 1 + 2 + 3 + 4 = 10$
 - $5 + 6 + 7 + 8 + 9 = 35 \dots$

```
vector<int> v =  
    view::iota(0, 15) |  
    view::group_by(quotient) |  
    view::transform(sum);  
  
for (auto e : v)  
    cout << e << " ";  
cout << endl;  
// 10 35 60
```

```
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]  
[[0,1,2,3,4],[5,6,7,8,9],[10,11,12,13,14]]  
[10,35,60]
```

Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- ~~[0~15) 에서 5개씩 그룹 지어 각 그룹의 합을 구해라~~
- **0부터** 5개씩 그룹 지어 각 그룹의 합을 **6개** 구해라?

```
vector<int> v =  
    view::iota(0, 15) |  
    view::group_by(quotient) |  
    view::transform(sum);  
  
for (auto e : v)  
    cout << e << " ";  
cout << endl;  
// 10 35 60
```

?

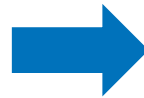
Ranges : 파이프라인

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 조합성(Composability)

- ~~[0~15) 에서 5개씩 그룹 지어 각 그룹의 합을 구해라~~
- **0부터** 5개씩 그룹 지어 각 그룹의 합을 **6개** 구해라?

```
vector<int> v =  
    view::iota(0, 15) |  
    view::group_by(quotient) |  
    view::transform(sum);  
  
for (auto e : v)  
    cout << e << " ";  
cout << endl;  
// 10 35 60
```



```
vector<int> v =  
    view::iota(0) |  
    view::group_by(quotient) |  
    view::transform(sum) |  
    view::take(6);  
  
for (auto e : v)  
    cout << e << " ";  
cout << endl;  
// 10 35 60 85 110 135
```


Ranges : 출력

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 컨테이너 안에 무엇이 들었나

```
vector<int> v = { ? };
```

Ranges : 출력

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 옆동네 파이썬에서는..
 - 컨테이너 원소 쉽게 확인 가능

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Ranges : 출력

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 우리는..?

```
auto sq = [](int n) { return n * n; };

std::vector<int> v =
    view::iota(0, 10) |
    view::transform(sq);

for (auto e : v)
    cout << e << " ";
cout << endl;
// 0 1 4 9 16 25 36 49 64 81
```

Ranges : 출력

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 이상과 현실
 - 추가적인 코드가 불가피

```
cout << v << endl;  
// 0 1 4 9 16 25 36 49 64 81
```



```
cout << v << endl;
```

이러한 피연산자와 일치하는 "<<" 연산자가 없습니다.

Ranges : 출력

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- view 연산의 결과는 (대부분) 그대로 출력 가능

```
cout << view::iota(0, 10) << endl;
```



```
[0,1,2,3,4,5,6,7,8,9]
```

Ranges : 출력

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 파이프라인을 auto로 받아서 출력

```
auto rng = view::iota(0, 10) |  
           view::transform(sq);  
  
cout << rng << endl;
```

- 컨테이너를 "범위"로 만들어 출력

```
cout << view::all(v) << endl;
```

Ranges : 출력

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 그룹 지어진 범위 출력

```
auto quotient = [](int a, int b) {  
    return a / 5 == b / 5;  
};  
  
auto rng = view::iota(0, 15) |  
           view::group_by(quotient);  
  
cout << rng << endl;  
// [[0,1,2,3,4],[5,6,7,8,9],[10,11,12,13,14]]
```

- Insertion Operator 오버로딩 *it 는 하위 range 일 수 있음

```
//struct view_interface
friend Stream &operator<<(Stream &sout, Derived &rng){
    auto it = ranges::begin(rng);
    auto const e = ranges::end(rng);
    if(it == e)
        return sout << "[]";
    sout << '[' << *it;
    while(++it != e)
        sout << ',' << *it;
    sout << ']';
    return sout;
}
```


이게 어떻게 가능하지?

자료형



```
auto quotient = [](int a, int b) {  
    return a / 5 == b / 5;  
};  
  
auto rng = view::iota(0) |  
           view::group_by(quotient);  
  
vector<int> v = rng | view::take(10);  
  
cout << view::all(v) << endl;
```

대체 타입이 뭐지?

빠르게 변화하는
모던 C++에 몸을 맡겨라!



- 개념적 타입

- `Range<T>` 로 표현
- 중첩 가능

```
// Range<Range<int>>
auto rng = view::iota(0, 15) |
           view::group_by(quotient);
// [[0,1,2,3,4],[5,6,7,8,9],[10,11,12,13,14]]
```

- **실제** 타입 * 가독성을 위해 간략화함

- **take_exactly_view<iota_view<int,void>,0>**

```
auto rng = view::iota(0) |  
            view::take(10);  
  
cout << typeid(rng).name() << endl;  
//take_exactly_view<iota_view<int,void>,0>
```

Ranges : 기본 구조

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 생각해 봅시다

```
simple_iota(0);
```

```
template <class T>  
struct simple_iota_view;
```

```
simple_take(10)
```

```
template <class T>  
struct simple_take_view;
```

```
simple_take_view<simple_iota_view>
```

- 표현식 템플릿 (expression template)

- 수식 전체가 하나의 타입

```
auto sum = v1 + v2 + v3;  
//우항 : VecSum<VecSum<Vec, Vec>, Vec>
```

- 값이 실제로 필요할 때 계산

```
cout << sum[0] << endl;  
  
struct VecSum {  
    double operator[](size_t i) {  
        return _u[i] + _v[i];  
    }  
}
```

Ranges : 타입

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 표현식 별 타입 * 가독성을 위해 간략화함

- `iota` | `cycle` | `take`

```
auto rng = view::iota(0, 5) |  
           view::cycle |  
           view::take(15);  
  
cout << typeid(rng).name() << endl;  
// take_exactly_view_<cycled_view<iota_view<int>>>
```


Ranges : 타입

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 표현식 별 타입 * 가독성을 위해 간략화함
- `iota` | `group_by` | `transform`

```
auto rng = view::iota(0, 15) |  
           view::group_by(quotient) |  
           view::transform(sum);  
  
cout << typeid(rng).name() << endl;
```

?

Ranges : 타입

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- `?AU?$transform_view@U?$group_by_view@U?$take_exactly_view_@U?...` 생략



Ranges : 타입

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 대체 왜...?

- `iota` | `cycle` | `take`

```
auto rng = view::iota(0, 5) |  
           view::cycle |  
           view::take(15);  
// take_exactly_view<cycled_view<iota_view<int>>>
```

- `iota` | `group_by` | `transform`

```
auto rng = view::iota(0, 15) |  
           view::group_by(quotient) |  
           view::transform(sum);  
// ?AU?$transform_view@U?$group_by_view@U?$take_exa  
ctly_view_@U?...생략
```

- Type Display
 - 컴파일러 오류 메시지로 타입 확인

```
template <class T>  
struct TD;  
  
/***/  
  
TD<decltype(rng)>();
```

- Type Displayer 로 확인한 타입
- `iota` | `group_by`(람다) | `transform`(람다) * 가독성을 위해 간략화함

```
error : implicit instantiation of undefined template  
'TD<transform_view<group_by_view<iota_view<int>,  
(lambda at type_of_range.cpp:32:18)>,  
(lambda at type_of_range.cpp:36:13)>>
```

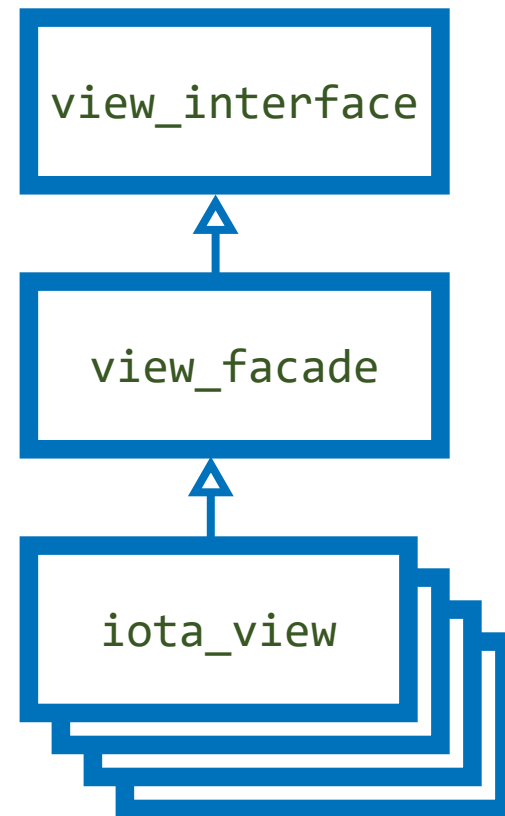
- 실제 타입은 매번 바뀌는데

어떻게 모두 **Range<T>** 처럼 사용할 수 있죠?

Ranges : 타입

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 개념적 타입 **Range<T>** 가 가능한 이유
 - 공통 인터페이스 상속
 - Range<T>는 일종의 인터페이스
 - view_facade 를 상속받아
User-Define View **쉽게** 정의 가능



표현식



- 자연수

- [명사] <수학> 1부터 시작하여 하나씩 더하여 얻는 수를 통틀어 이르는 말. 1, 2, 3 따위이다. (후략) 네이버 국어사전
- $x \in \mathbb{N}$
- 기존에 자연수를 표현하던 방법

```
// v는 자연수로 이루어진 벡터
void foo(vector<int> v) {
    for(auto i : v) {
        assert(0 < i);
        /**/
    }
}
```

Ranges : 자연수 표현

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- Ranges 자연수 표현

```
auto naturals = view::ints(1);
```

- 무한 수열 표현

- 반드시 `view::take()` 와 같은 한정 함수와 함께 사용

```
auto s = naturals | view::take(10);  
cout << s << endl;  
//[1,2,3,4,5,6,7,8,9,10]
```

Range Comprehension

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 수학 시간에 배운 것처럼 Range 만들기
 - 지금까지 소개한 방식

```
// 집합  $S = \{ 2 * x \mid x \text{는 자연수} \}$   
auto twice = [](int i) { return 2 * i; };  
auto s = view::ints(1) | view::transform(twice);
```

Range Comprehension

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 수학 시간에 배운 것처럼 Range 만들기
 - `view::for_each` 를 이용한 방식

```
// 집합  $S = \{ 2 * x \mid x \text{는 자연수} \}$   
auto s = view::ints(1) | view::for_each([](int i){  
    return ranges::yield(i * 2);  
});
```

Range Comprehension

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- `view::for_each` 연산자 오버로딩

- `view::for_each`

```
auto s = view::ints(1) | view::for_each([](int i){  
    return ranges::yield(i * 2); //  
});
```

- operator extraction equal (`>>=`) 오버로딩

```
auto s = view::ints(1) >>= [](int i) {  
    return ranges::yield(i * 2);  
};
```

Range Comprehension

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 피타고라스 수
 - $a^2 + b^2 = c^2$ 을 만족하는 세 자연수 쌍 (a, b, c) 100개를 구하시오.

```
auto triples = view::ints(1) >>= [] (int z) { return  
    view::ints(1, z+1) >>= [=](int x) { return  
        view::ints(x, z+1) >>= [=](int y) { return  
            yield_if(x*x + y*y == z*z,  
                std::make_tuple(x, y, z));  
        }; }; };
```

Ranges의 양대 산맥

View Action



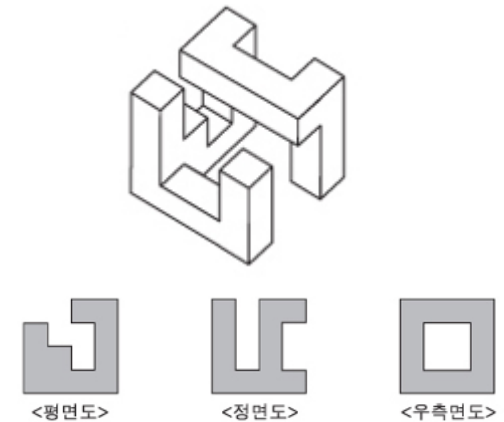
Ranges : View / Action namespace

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- **ranges::view**

- 데이터를 특정 시점으로 보기 위한 연산들
- 원본 데이터의 복사나 변환 X
- Lazy Evaluation 가능
 - 표현식 템플릿

```
auto sum = v1 + v2 + v3;  
//우향 : VecSum<VecSum<Vec, Vec>, Vec>
```



- **ranges::action**

- 데이터 원본을 직접 수정하는 연산들 (STL처럼)

Ranges : View namespace

빠르게 변화하는
모던 C++에 몸을 맡겨라!

lazy_yield_if	adjacent_filter	adjacent_remove_if	all	bounded
const	counted	cycle	delimit	drop
indirect	intersperse	ints	iota	iter_take_while
move	partial_sum	remove_if	repeat	repeat_n
split	stride	tail	take_exactly	take_while
c_str	chunk	closed_ints	closed_iota	concat
drop_while	for_each	generate	generate_n	group_by
iter_transform	iter_zip_with	join	keys	make_view
replace	replace_if	reverse	single	slice
tokenize	transform	unbounded	unique	values

Ranges : Action namespace

빠르게 변화하는
모던 C++에 몸을 맡겨라!

drop	drop_while	join	split	stable_sort
make_action	remove_if	shuffle	slice	sort
stride	take	take_while	transform	unique

Ranges : Action

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 컨테이너 직접 수정

```
std::default_random_engine gen;  
std::vector<int> v = view::iota(0, 10);  
  
action::shuffle(v, gen); v |= action::shuffle(gen);  
  
cout << view::all(v) << endl;  
//[2,7,8,4,0,6,1,9,3,5]
```

Ranges : Action

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- 복사

```
auto v2 = v | ranges::copy | action::sort;  
cout << view::all(v2) << endl;  
//[0,1,2,3,4,5,6,7,8,9]
```

- 이동

```
auto v2 = v | ranges::move | action::sort;  
cout << view::all(v2) << endl;  
//[0,1,2,3,4,5,6,7,8,9]
```

Ranges : Action

빠르게 변화하는
모던 C++에 몸을 맡겨라!

- View 와 함께 파이프라이닝

```
// 짝수만 셔플  
v = view::ints(0, 10);  
v | view::stride(2) | action::shuffle(gen);  
cout << view::all(v) << endl;  
//[6,1,8,3,2,5,0,7,4,9]
```

- 특정한 순회 방식에 알고리즘 적용
- 추가 컨테이너 필요 X

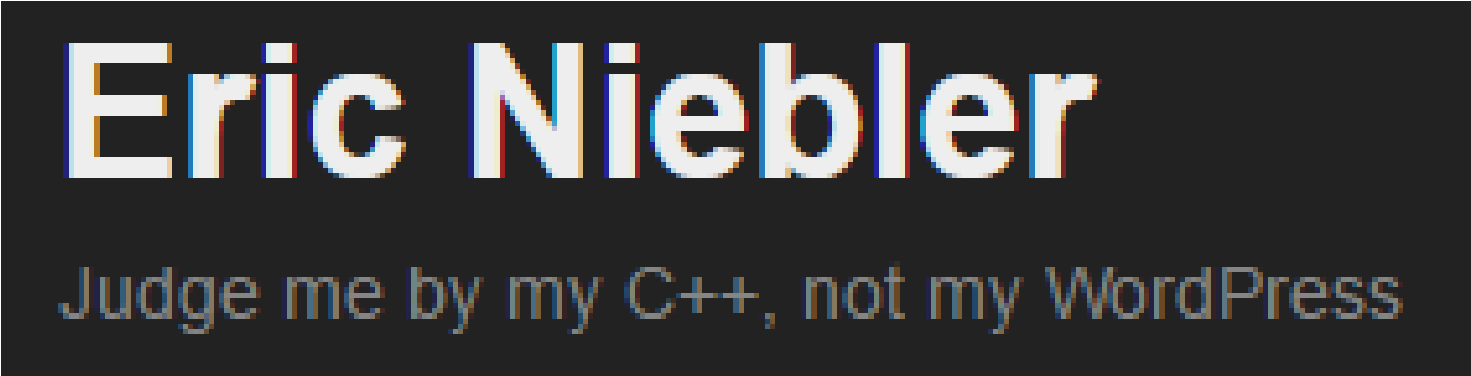
Summary



- 장점
 - **간결하고 강력한 표현**
 - 범위기반 알고리즘
 - 파이프라인
 - Lazy Evaluation
 - Range Comprehension
 - 표준 컨테이너를 매개로 기존 코드에 쉽게 이식 가능
- 단점
 - 컴파일 시간 (개선 중)

Thanks!

빠르게 변화하는
모던 C++에 몸을 맡겨라!

The logo for Eric Niebler features his name in a large, white, sans-serif font with a subtle red and blue shadow effect, set against a dark grey rectangular background.

Eric Niebler

Judge me by my C++, not my WordPress

QnA



- 라이브러리 : <https://github.com/ericniebler/range-v3>
- 매뉴얼 : <https://ericniebler.github.io/range-v3/>
- 기타 참조
 - <http://ericniebler.com/>
 - [Effective Modern C++](#)
 - <http://stackoverflow.com/search?q=range-v3>
 - <https://en.wikipedia.org/>
 - <http://www.r-statistics.com/>
 - <http://en.cppreference.com/>