



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)



# C++17 Key Features Summary

옥찬호 / Nexon Korea, C++ Korea



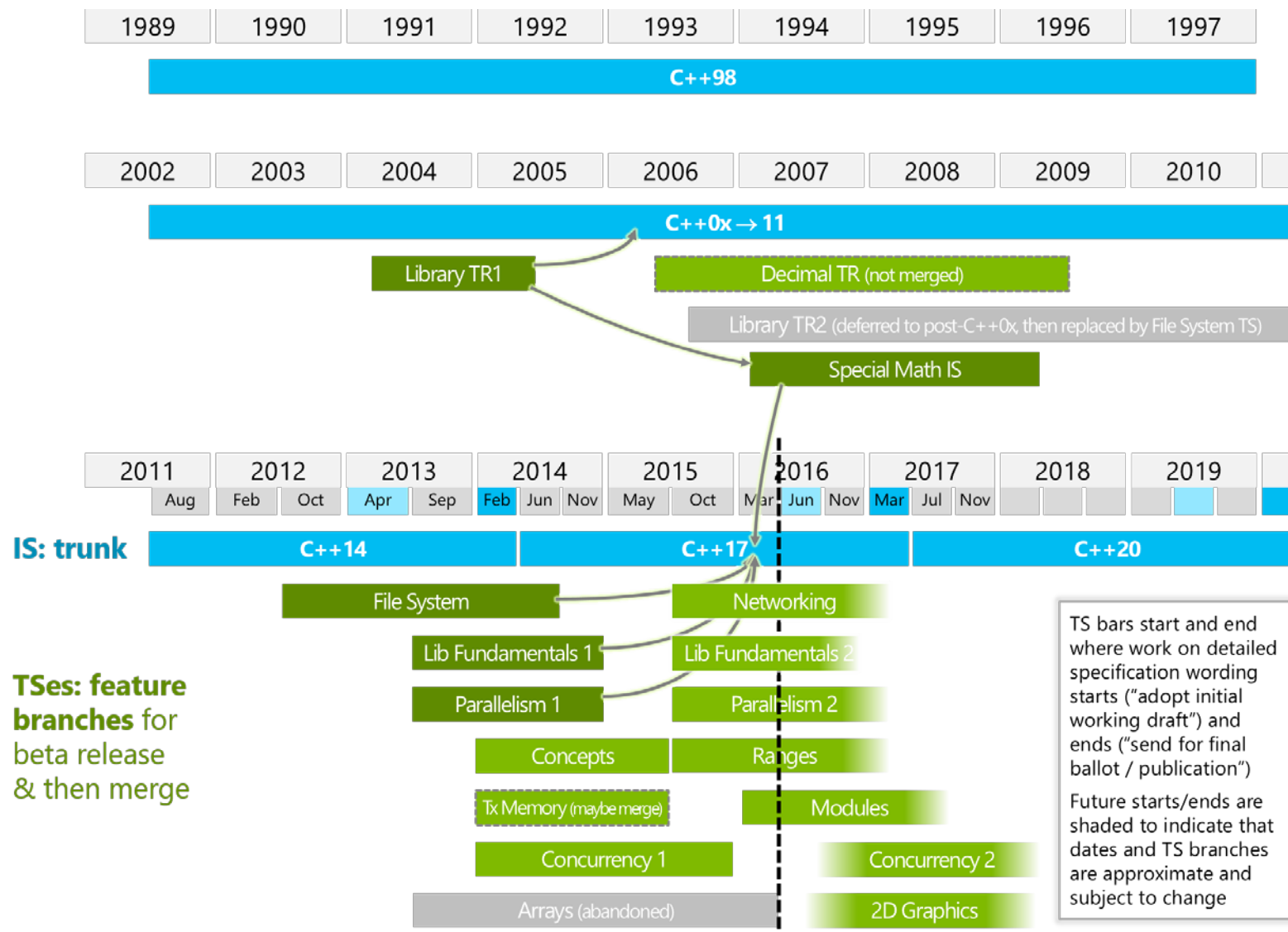
# 시작하기 전에...

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- C++17에 추가될 주요 기능들을 설명합니다.
- C++17에 추가될 주요 기능 뿐만 아니라,  
이후 TS(Technical Specification)에 추가될 기능들도 소개합니다.
- C++17에 추가될 기능 중 C++11/14 지식이 필요한 경우,  
이해를 돕기 위해 사전 지식을 먼저 설명합니다.
- 모든 예제 코드는 발표 이후 Github를 통해 제공됩니다.

# Current C++ Status

빠르게 변화하는  
모던 C++에 몸을 맡겨라!



# C++17 / TS Features

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- Module, Range, Concept
- Uniform Call Syntax
- Coroutine
- Contract
- Networking Library (Boost.asio based)
- Parallel STL
- `std::string_view`
- `std::any`, `std::optional` (Boost.any, Boost.optional based)
- ...

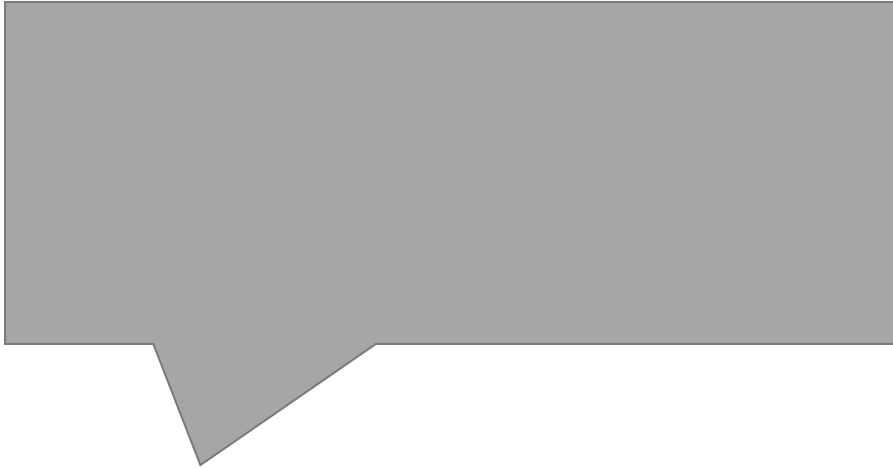
# Module



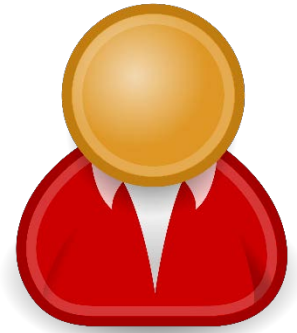
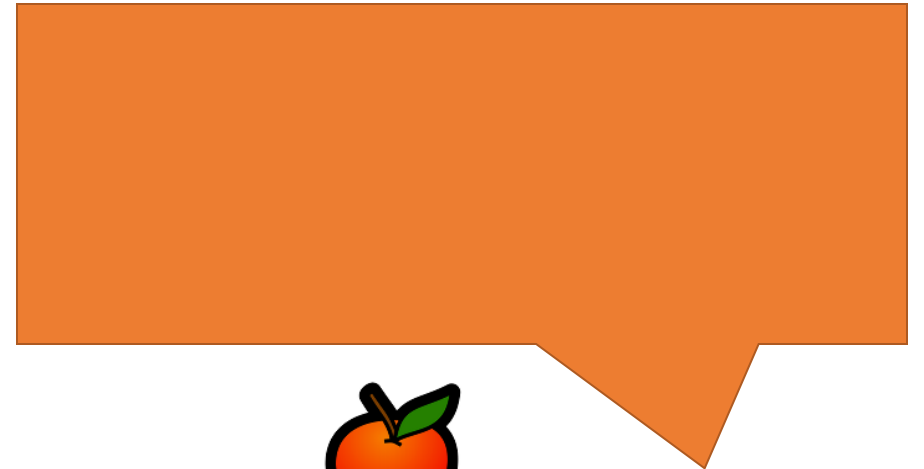
이런 상황을 고려해 봅시다.

# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!



지나가던 시민  
(개발자)

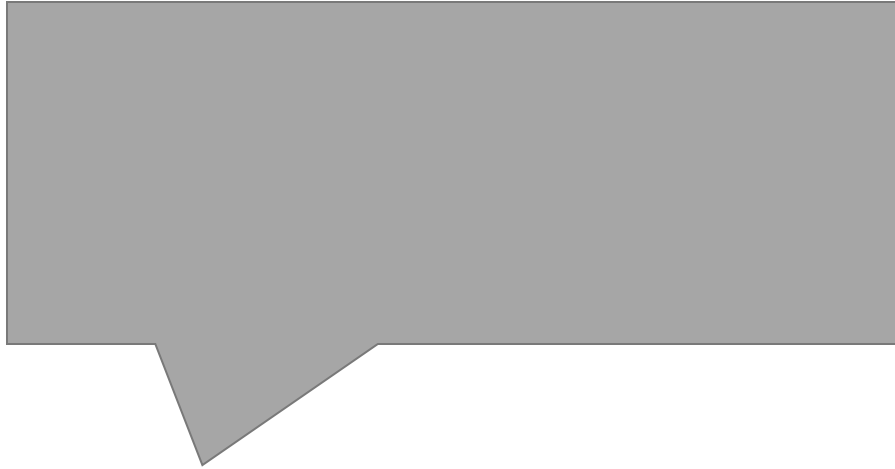


과일 가게  
아저씨



# Module

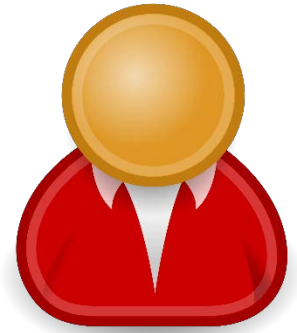
빠르게 변화하는  
모던 C++에 몸을 맡겨라!



지나가던 시민  
(개발자)



싱싱한 과일 팝니다~



과일 가게  
아저씨

# Module

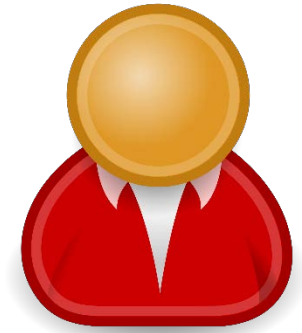
빠르게 변화하는  
모던 C++에 몸을 맡겨라!

사과가 맛있어 보이네요.  
사과 하나 주세요.



지나가던 시민  
(개발자)

싱싱한 과일 팝니다~

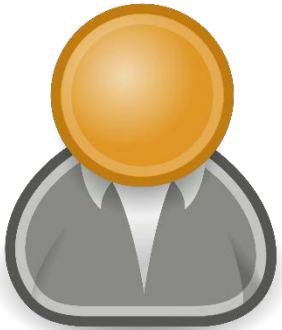


과일 가게  
아저씨

# Module

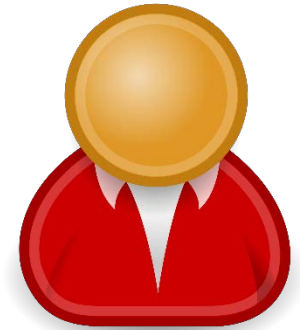
빠르게 변화하는  
모던 C++에 몸을 맡겨라!

사과가 맛있어 보이네요.  
사과 하나 주세요.



지나가던 시민  
(개발자)

저희 가게가 내일 휴일이라,  
맛보시라고 포도랑 체리도  
드릴테니 잡썰보세요~



과일 가게  
아저씨

# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

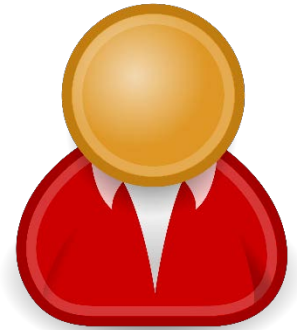
와, 감사합니다.  
잘 먹을까요!



지나가던 시민  
(개발자)



저희 가게가 내일 휴일이라,  
맛보시라고 포도랑 체리도  
드릴테니 잡썰보세요~



과일 가게  
아저씨

# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

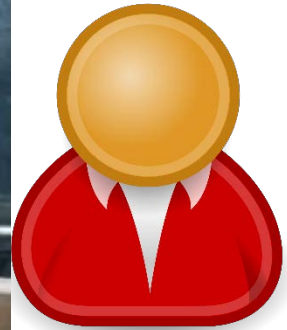
와,  
즐거워!

다음 주

정말 가게가 내의 휴일이라,  
랑 체리도  
보세요~



지나가던 시  
(개발자)

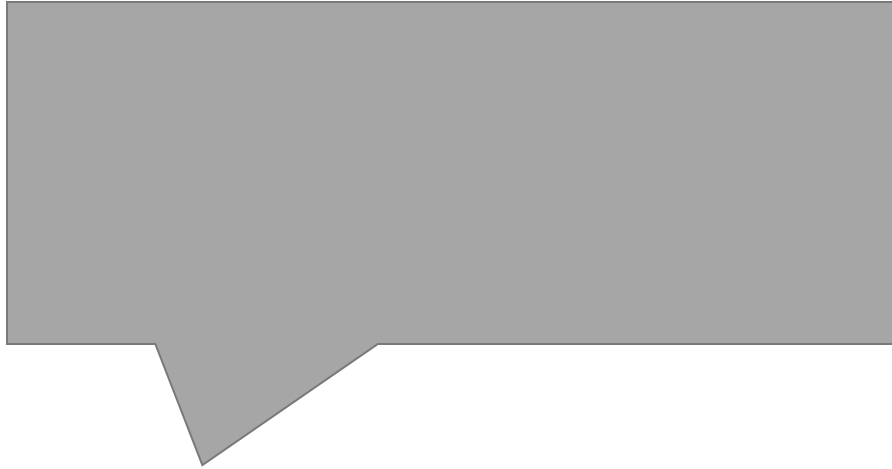


과일 가게  
아저씨

이제 대상을 좀 바꿔봅시다.

# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

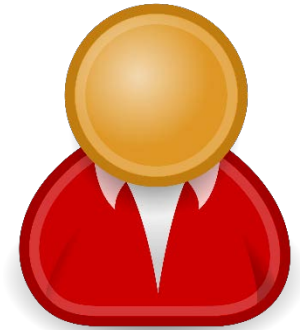


지나가던 개발자

싱싱한 함수 팝니다~

`<cpp_math.h>`

```
int Max(int a, int b);  
int Min(int a, int b);  
int Sum(int a, int b);  
float Avg(int a, int b);
```



C++

# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

오, sum 함수 필요했는데...  
sum 함수 하나 주세요!

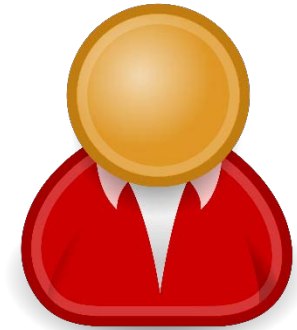


지나가던 개발자

싱싱한 함수 팝니다~

<cpp\_math.h>

```
int Max(int a, int b);  
int Min(int a, int b);  
int Sum(int a, int b);  
float Avg(int a, int b);
```



C++



# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

오, sum 함수 필요했는데...  
sum 함수 하나 주세요!

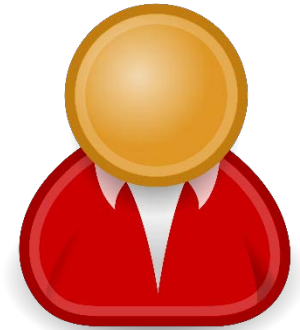


지나가던 개발자

저희는 1+3 패키지로만  
판매하고 있습니다.  
전부 가져가세요~

<cpp\_math.h>

```
int Max(int a, int b);  
int Min(int a, int b);  
int Sum(int a, int b);  
float Avg(int a, int b);
```



C++

# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

Max랑 Min 함수는  
이미 구현해봤는데...  
어쩌지 ㅠㅠ

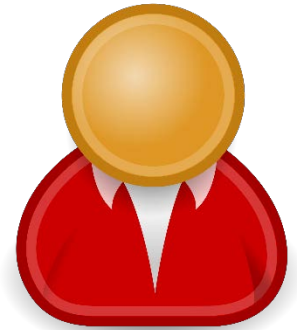


지나가던 개발자

<cpp\_math.h>

```
int Max(int a, int b);  
int Min(int a, int b);  
int Sum(int a, int b);  
float Avg(int a, int b);
```

저희는 1+3 패키지로만  
판매하고 있습니다.  
전부 가져가세요~



C++

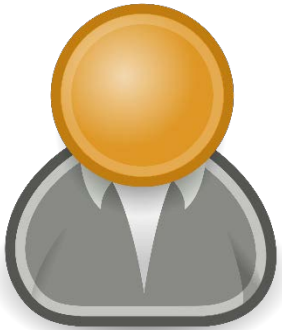
# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

Max랑 Min 함수는  
이미 구현해봤는데...  
어쩌지 ㅠㅠ

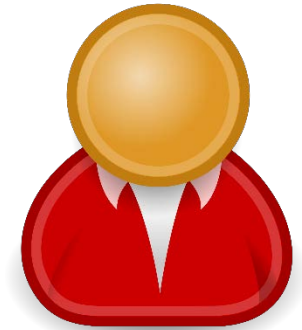
<cpp\_math.h>

```
int Max(int a, int b);  
int Min(int a, int b);  
int Sum(int a, int b);  
float Avg(int a, int b);
```



지나가던 개발자

그건 우리 알 바 아님 ㅋ

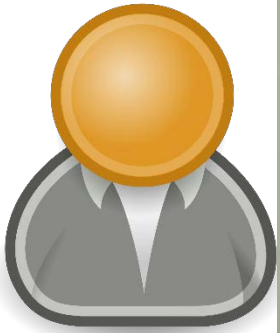


C++

# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

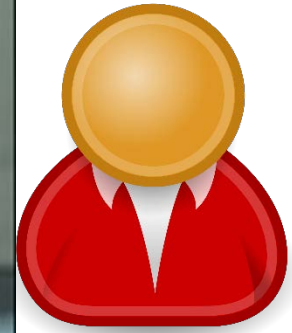
Max<sup>라</sup> Min 하스느  
이미  
0



지나가던 개털



아님 ㅋ



C++

- C++에서는 `#include`를 통해 헤더 파일을 포함
- 문제는 헤더 파일 안에 있는 모든 기능을 포함한다는 것!  
(포함하고 싶지 않은 기능도 가져오게 됨)
- 다른 언어에서는 필요한 기능만 가져올 수 있는 기능을 제공
  - C# : `using System.Linq;`
  - Java : `import java.util.Scanner;`
  - Python : `from path import pi`
- C++에서도 필요한 기능만 가져올 수 있다면 좋지 않을까?  
→ Module의 등장!

- 모듈에서 기억해야 할 3가지 키워드!
  - `module` : 모듈로 정의할 이름을 지정  
Ex) **module** M1 : 정의할 모듈의 이름은 M1이다.
  - `import` : 가져올 모듈을 이름을 지정  
Ex) **import** M1 : 가져올 모듈의 이름은 M1이다.
  - `export` : 내보낼 함수의 인터페이스를 지정  
Ex) **export** int f(int, int)  
내보낼 함수의 이름은 f이고, 리턴 타입은 int, 매개 변수는 int, int다.

# Module

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

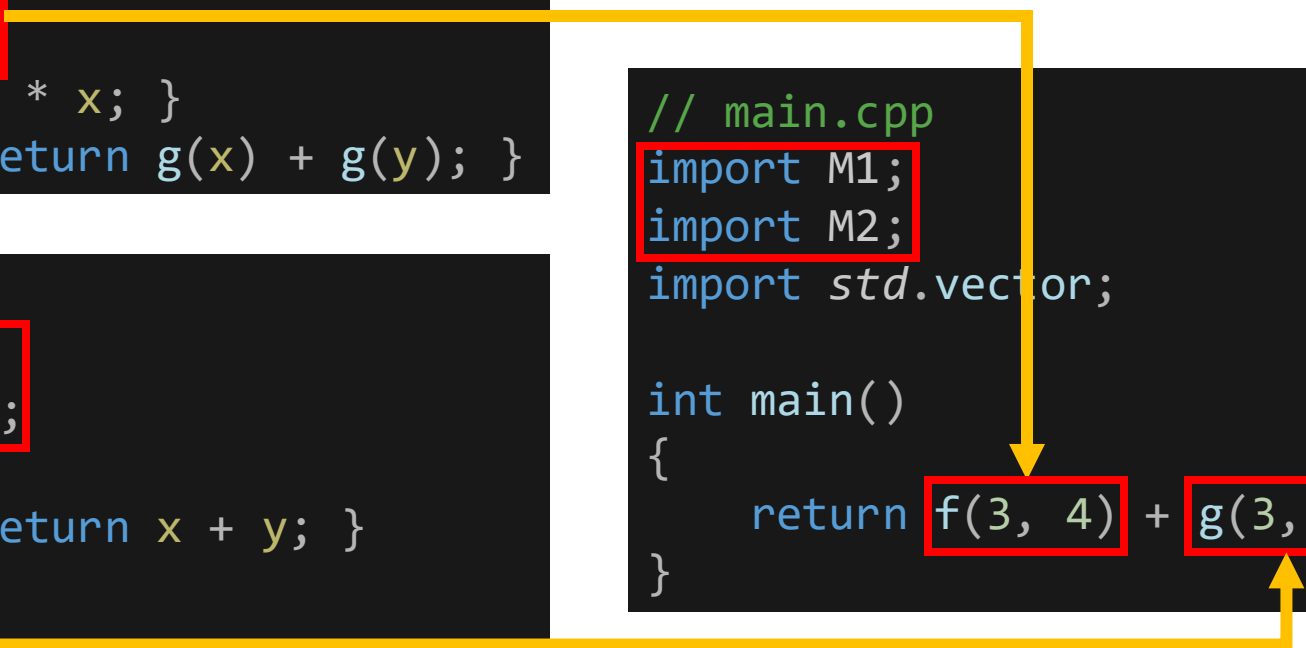
- Example

```
// m1.cpp
module M1;
export int f(int, int);
int g(int x) { return x * x; }
int f(int x, int y) { return g(x) + g(y); }
```

```
// m2.cpp
module M2;
export bool g(int, int);
import std.math;
int f(int x, int y) { return x + y; }
int g(int x, int y)
{
    return f(abs(x), abs(y));
}
```

```
// main.cpp
import M1;
import M2;
import std.vector;

int main()
{
    return f(3, 4) + g(3, 4);
}
```



# Concept





# Concept

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- 다음 코드의 출력 결과는?

```
#include <algorithm>
#include <list>

int main()
{
    std::list<int> l = { 2, 1, 3 };
    std::sort(l.begin(), l.end());
}
```

1, 2, 3?

아니면 3, 2, 1?



# Concept

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

Error List

Entire Solution 9 Errors 0 Warnings 0 Messages Build + IntelliSense

	Code	Description
✖	C2784	'unknown-type std::operator -(std::move_iterator<_Ranlt> &,const std::move_iterator<_Ranlt2> &)': could not deduce template argument for 'std::move_iterator<_Ranlt> &' from 'std::_List_iterator<std::_List_val<std::_List_simple_types<int>>>'
✖	C2784	'unknown-type std::operator -(const std::reverse_iterator<_Ranlt> &,const std::reverse_iterator<_Ranlt2> &)': could not deduce template argument for 'const std::reverse_iterator<_Ranlt> &' from 'std::_List_iterator<std::_List_val<std::_List_simple_types<int>>>'
✖	C2676	binary '-': 'std::_List_iterator<std::_List_val<std::_List_simple_types<int>>>' does not define this operator or a conversion to a type acceptable to the predefined operator
✖	C2672	'_Debug_pointer_if': no matching overloaded function found
✖	C2784	'unknown-type std::operator -(std::move_iterator<_Ranlt> &,const std::move_iterator<_Ranlt2> &)': could not deduce template argument for 'std::move_iterator<_Ranlt> &' from 'std::_List_iterator<std::_List_val<std::_List_simple_types<int>>>'
✖	C2784	'unknown-type std::operator -(const std::reverse_iterator<_Ranlt> &,const std::reverse_iterator<_Ranlt2> &)': could not deduce template argument for 'const std::reverse_iterator<_Ranlt> &' from 'std::_List_iterator<std::_List_val<std::_List_simple_types<int>>>'
✖	C2676	binary '-': 'std::_List_iterator<std::_List_val<std::_List_simple_types<int>>>' does not define this operator or a conversion to a type acceptable to the predefined operator
✖	C2672	'_Sort': no matching overloaded function found
✖	C2780	'void std::_Sort(_Ranlt,_Ranlt,_Diff,_Pr)': expects 4 arguments - 3 provided

# Concept

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

Error List			
Entire Solution			
	Code	Desc	
✖	C2784	'unkn	'std::_
✖	C2784	'unkn	std::re
✖	C2676	binary	
✖	C2672	'_Deb	
✖	C2784	'unkn	'std::_
✖	C2784	'unkn	std::re
✖	C2676	binary	
✖	C2672	'_Sort	
✖	C2780	'void	

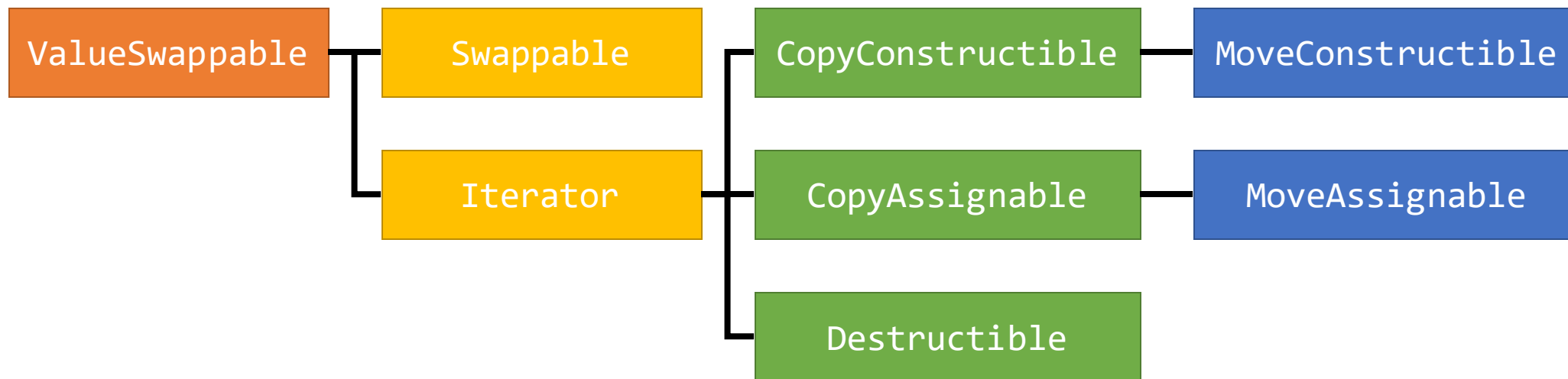


		ator<_Ranlt> &' from	
		ed operator	
		ator<_Ranlt> &' from	
		ed operator	

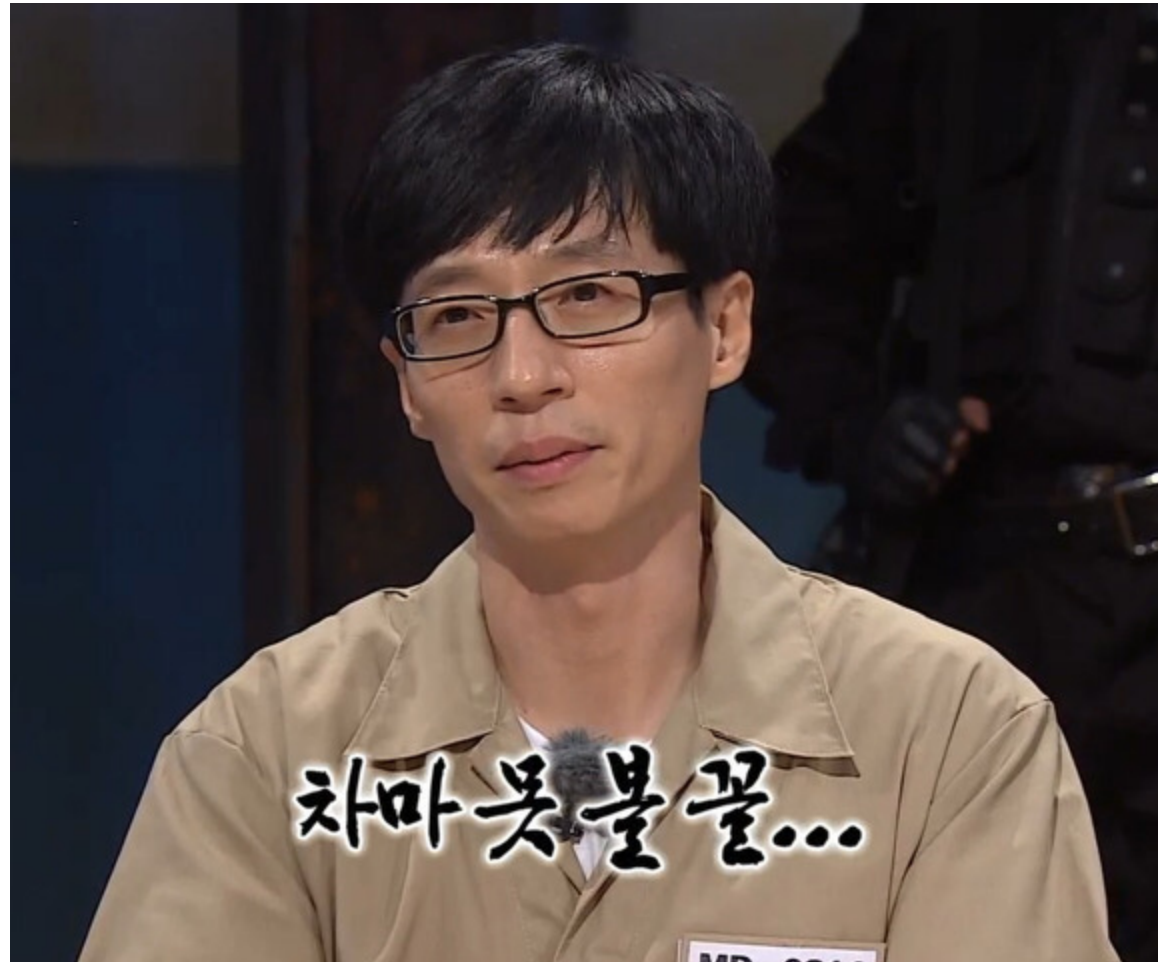
- 오류의 발생 원인
  - `template <class RandomIt>`  
`void sort(RandomIt first, RandomIt last);`
  - 여기서 `RandomIt`는 `ValueSwappable`과 `RandomAccessIterator`의  
요구사항을 만족해야 함
  - 하지만 `int`에는 반복자(iterator)가 존재하지 않는다!

```
std::list<int> l = { 2, 1, 3 };  
std::sort(l.begin(), l.end());
```

- ValueSwappable의 요구사항



- 그러면 RandomAccessIterator는요?



- C#에서는 제약 사항과 관련해 다양한 클래스/인터페이스 제공
  - ICloneable
  - IComparable
  - IConvertible
  - IFormattable
  - Nullable
  - ...
- C++에서는? 어... 음...  
→ Concept의 등장!

- C++ 템플릿의 확장 기능 → 템플릿 매개 변수에 제약을 건다!

```
template <class T>
concept EqualityComparable()
{
    return requires(T a, T b) {
        {a == b}->Boolean; // Boolean is the concept defining
        {a != b}->Boolean; // a type usable in boolean context
    };
}
```

```
void f(const EqualityComparable&);
```

```
f("abc"s); // OK, std::string satisfies EqualityComparable
// Error: not EqualityComparable
f(std::use_facet<std::ctype<char>>(std::locale{}));
```



- 이제 컴파일러의 오류 메시지가 명확해진다!

```
In instantiation of 'void std::__sort(_RandomAccessIterator,
_RandomAccessIterator, _Compare) [with _RandomAccessIterator =
std::_List_iterator<int>; _Compare = __gnu_cxx::__ops::_Iter_less_iter]':
error: no match for 'operator-' (operand types are
'std::_List_iterator<int>' and 'std::_List_iterator<int>')
std::__lg(__last - __first) * 2,
```



```
error: cannot call function 'void std::sort(_RAIter, _RAIter) [with _RAIter
= std::_List_iterator<int>]'
```

note: concept 'RandomAccessIterator()' was not satisfied

- auto 대신 Concept을 사용해 타입 추론을 할 수도 있다.

```
auto x1 = f(y); // the type of x1 is deduced to whatever f returns
Sortable x2 = f(y); // the type of x2 is deduced, but only compiles
                  // if it satisfies Sortable
```

- Concept 리스트
  - Basic
    - DefaultConstructible, MoveConstructible, CopyConstructible, MoveAssignable, CopyAssignable, Destructible
  - Library-wide
    - EqualityComparable, LessThanComparable, Swappable, ValueSwappable, NullablePointer, Hash, Allocator, FunctionObject, Callable, Predicate, BinaryPredicate, Compare
  - Iterator
    - Iterator, InputIterator, OutputIterator, ForwardIterator, BidirectionalIterator, RandomAccessIterator, ContiguousIterator
  - ...

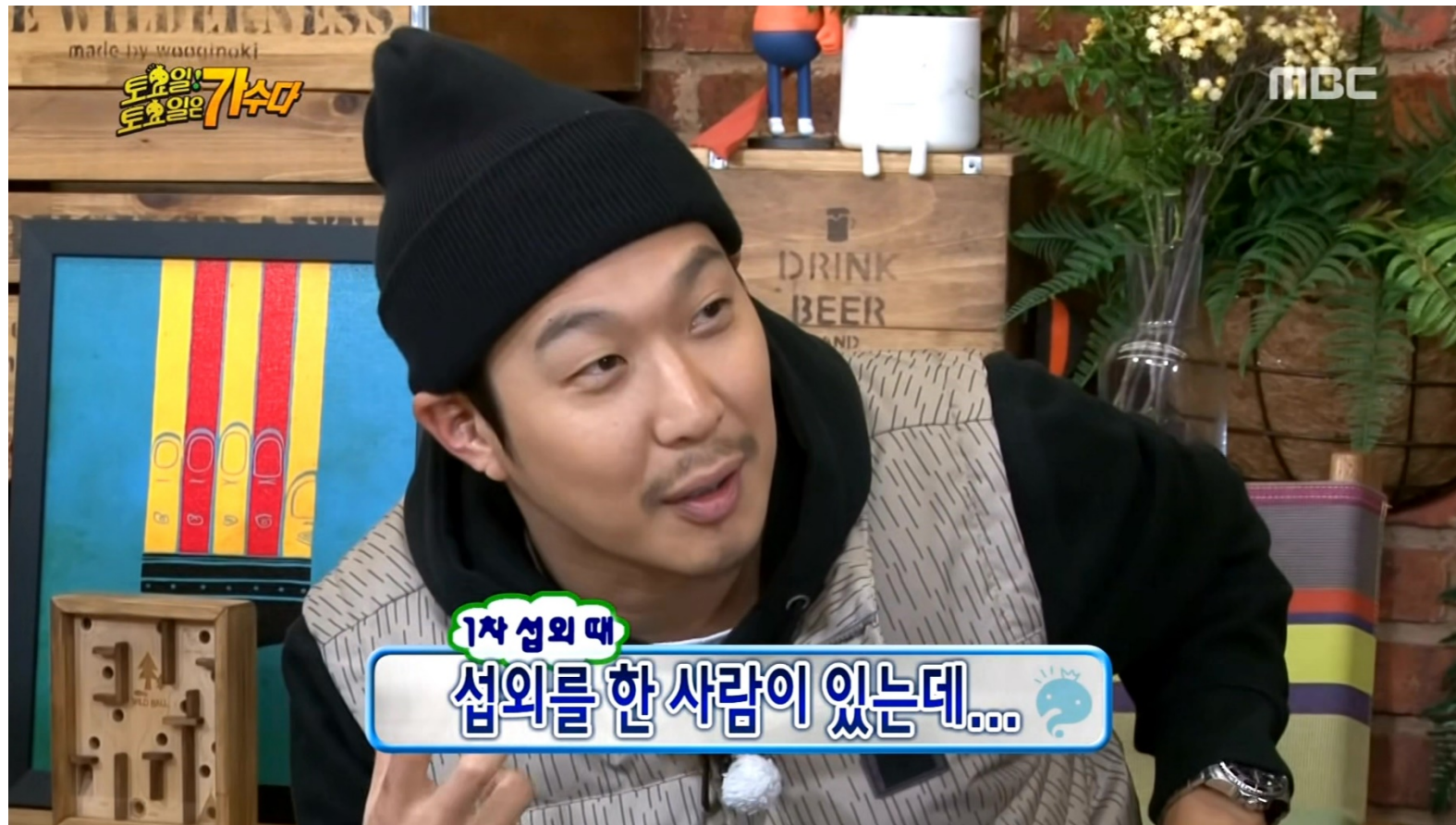
# Ranges



# Ranges

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

다음 세션에서 자세하게 알려드립니다.



# Core Features



# Fold expression

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- C++11의 가변 템플릿 (Variadic Template)

```
template <typename T>
void PrintList(T value)
{
    std::cout << value << std::endl;
}
```

```
template<typename First, typename ...Rest>
void PrintList(First first, Rest ...rest)
{
    std::cout << first << ", ";
    PrintList(rest...);
}
```

# Fold expression

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- C++11의 가변 템플릿 (Variadic Template)

```
PrintList(42, "Hello", 2.3, 'a');
```



```
42, Hello, 2.3, a
```

```
PrintList(first = 42, ...rest = "hello", 2.3, 'a');  
42  
  PrintList(first = 42, ...rest = 2.3, 'a');  
    hello  
      PrintList(first = 42, ...rest = 'a');  
        2.3  
          PrintList(first = 'a');
```

재귀 함수를 끝내기 위한  
별도의 함수 필요!

```
template <typename T>  
void PrintList(T value)  
{  
    std::cout << value << std::endl;  
}
```

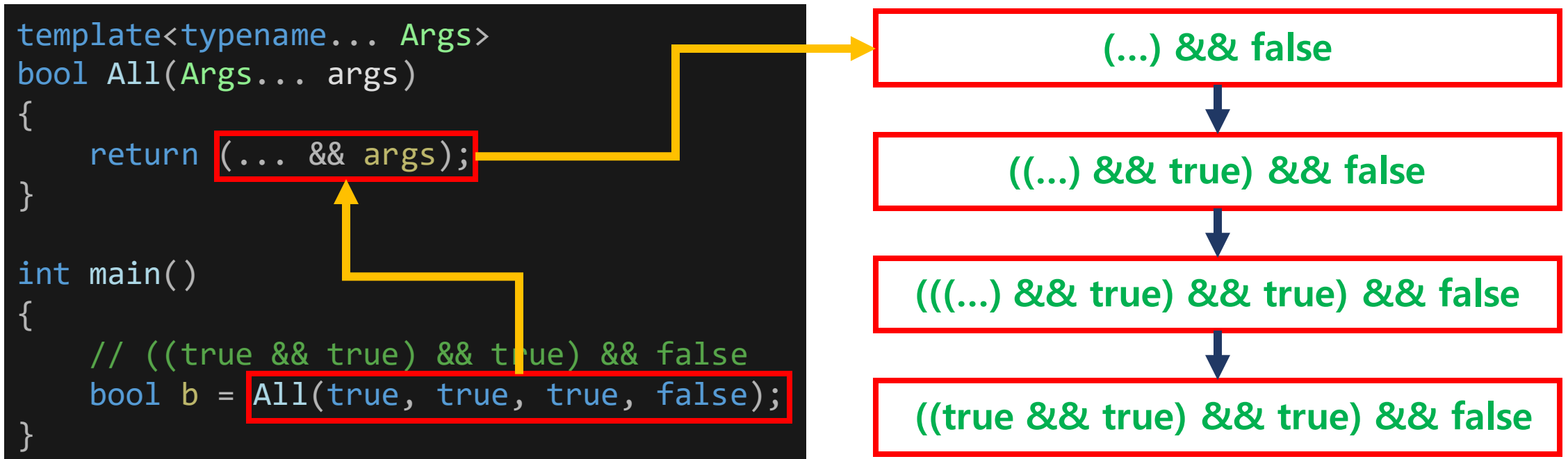


# Fold expression

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- 가변 템플릿을 사용해 표현식을 간단하게 구현할 수 있는 기능

Fold = 접는다, Expression = 표현식 → **“표현식을 접는다!”**



# Fold expression

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- Fold expression 문법 (E : 매개 변수 묶음, op : 연산자, I : 초기값)
  - 1) 단항 오른쪽 접기 (Unary Right Fold) : E op ...  
→ E1 op (... op(En-1 op En))
  - 2) 단항 왼쪽 접기 (Unary Left Fold) : ... op E  
→ ((E1 op E2) op ...) op En
  - 3) 이항 오른쪽 접기 (Binary Right Fold) : E op ... op I  
→ E1 op(... op(En-1 op(En op I)))
  - 4) 이항 왼쪽 접기 (Binary Left Fold) : I op ... op E  
→ (((I op E1) op E2) op ...) op En

# Fold expression

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- Example 1

- 3) 이항 오른쪽 접기 (Binary Right Fold) :  $E \text{ op } \dots \text{ op } I$   
 $\rightarrow E1 \text{ op } (\dots \text{ op } (E_{n-1} \text{ op } (E_n \text{ op } I)))$

```
template<typename ...Args>
int Sum(Args&&... args)
{
    return (args + ... + (1 * 2));
}

int main()
{
    Sum(1, 2, 3, 4);
}
```

# Fold expression

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- Example 2

- 4) 이항 왼쪽 접기 (Binary Left Fold) :  $I \text{ op } \dots \text{ op } E$   
 $\rightarrow ((I \text{ op } E1) \text{ op } E2) \text{ op } \dots \text{ op } E_n$

```
template<typename ...Args>
void Printer(Args&&... args)
{
    (std::cout << ... << args) << '\n';
}

int main()
{
    Printer(1, 2, 3, "abc");
}
```

- 타입, 객체, 코드 등에 구현 정의 특성을 부여
- C++14에 추가된 특성
  - `[[noreturn]]`: 이 함수는 리턴하지 않음
  - `[[carries_dependency]]`: 종속성 체인이 함수 안팎으로 전파됨
  - `[[deprecated]]`: 사용할 수 있지만, 권장하지 않음 (더 이상 사용 X)
- C++17에 추가된 특성
  - `[[fallthrough]]`: switch-case 문에서 사용, 완료되지 않음을 나타냄
  - `[[nodiscard]]`: 함수의 리턴 값이 있는데 버리는 경우 경고문을 표시
  - `[[maybe_unused]]`: 사용하지 않는 변수에 대해 경고문을 표시하지 않음

- Example 1

```
[[noreturn]] void f()
{
    throw "error"; // OK
}

[[noreturn]] void q(int i)
{
    // behavior is undefined
    // if called with an argument <= 0
    if (i > 0) { throw "positive"; }
}
```

- Example 2

```
void f(int n)
{
    void g(), h(), i();
    switch (n)
    {
        case 1:
        case 2:
            g();
            [[fallthrough]];
        case 3: // no warning on fallthrough
            h();
        case 4: // compiler may warn on fallthrough
            i();
            [[fallthrough]]; // illformed, not before a case label
    }
}
```

- Example 3

```
struct [[nodiscard]] error_info { };
error_info enable_missile_safety_mode();
void launch_missiles();
void test_missiles()
{
    enable_missile_safety_mode(); // compiler may warn
                                // on discarding a nodiscard value
    launch_missiles();
}
error_info& foo();
void f1()
{
    foo(); // nodiscard type is not returned by value, no warning
}
```



- Example 4

```
[[maybe_unused]] void f([[maybe_unused]] bool thing1,  
                          [[maybe_unused]] bool thing2)  
{  
    [[maybe_unused]] bool b = thing1 && thing2;  
    assert(b); // in release mode, assert is compiled out, and b is unused  
               // no warning because it is declared [[maybe_unused]]  
} // parameters thing1 and thing2 are not used, no warning
```

# constexpr Lambda

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- C++11의 상수 표현식 (constexpr, Constant Expression)
  - 쉽게 구현할 수 있는 메타 프로그래밍
  - 변수의 값이나 함수의 내용을 컴파일 타임에 처리
  - 함수 내에서는 하나의 표현식만 사용할 수 있고  
반드시 리터럴 타입을 반환해야 했으나, C++14에서 완화됨

```
constexpr int n = 0;           // OK
constexpr int m = time(NULL); // error C2127
```

```
constexpr int square(int x)
{
    return x * x;
}
```

```
int n;
std::cin >> n;
square(n); // Processed in run-time
square(10); // Processed in compile-time
```

# constexpr Lambda

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- C++11의 람다식 (Lambda Expression)

	인자 (Arguments)		함수의 몸통 (Statement)
<code>[my_mod]</code>	<code>(int v)</code>	<code>-&gt; int</code>	<code>{ return v % my_mod; }</code>
개시자 (Introducer Capture)		반환 타입 (Return Type)	

# constexpr Lambda

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- C++11의 람다식 (Lambda Expression)

```
int x = 10, y = 20;

[] {}; // Do not capture
[x] (int arg) { return x; }; // value(Copy) capture x
[=] { return x; }; // value(Copy) capture all
[&] { return y; }; // reference capture all
[&, x] { return y; }; // reference capture all except x
[=, &y] { return x; }; // value(Copy) capture all except y
[this] { return this->something; }; // this capture
[=, x] {}; // error
[&, &x] {}; // error
[=, this] {}; // error
[x, x] {}; // error
```

# constexpr Lambda

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- “람다식에 constexpr를 허하노라!”

```
int main()
{
    constexpr auto add = [](int a, int b) { return a + b; };
    Test<add(1, 2)> t1;

    auto add2 = [](int a, int b) constexpr { return a + b; };
    Test<add2(1, 2)> t2;

    auto add3 = [](int a, int b) { return a + b; };
    constexpr int c = add3(1, 2);
    constexpr int(*f)(int, int) = add3;
    Test<add3(1, 2)> t3;
}
```

# constexpr Lambda

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- Example : 팩토리얼 (Factorial)

```
int main()
{
    constexpr auto Factorial = getFactorializer();
    static_assert(Factorial(5) == 120, "");
    static_assert(Factorial(10) == 3628800, "");
}
```

# constexpr Lambda

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

```
constexpr auto getFactorializer = [] {  
    unsigned int optimization[6] = {};  
    auto for_each = [](auto* b, auto* e, auto pred) {  
        auto* it = b;  
        while (it != e) pred(it++ - b);  
    };  
    for_each(optimization, optimization + 6, [&](int n) {  
        if (!n) optimization[n] = 1;  
        else optimization[n] = n * optimization[n - 1];  
    });  
    auto FacImpl = [=](auto fac, unsigned n) {  
        if (n <= 5) return optimization[n];  
        return n * fac(fac, n - 1);  
    };  
    auto Fact = [=](int n) {  
        return FacImpl(FacImpl, n);  
    };  
    return Fact;  
};
```

# Call syntax

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

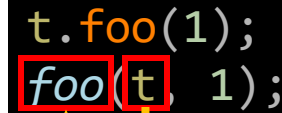
- 함수 호출 문법의 개선, .으로부터 불인 해제!

```
class Test {  
public:  
    void foo(int a) { }  
};  
  
int main() {  
    vector<int> v1;  
    vector<int> v2;  
    v1.swap(v2); // OK  
    swap(v1, v2); // OK in C++ 17  
  
    Test t;  
    t.foo(1); // OK  
    foo(t, 1); // OK in C++ 17  
}
```



```
v1.swap(v2);  
swap(v1, v2);
```

v1의 swap



```
t.foo(1);  
foo(t, 1);
```

t의 foo



# Operator. (Dot)

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- .도 연산자 오버로딩 할 수 있는 시대! → "Smart Reference"

```
template<class X>
class Ref
{
public:
    Ref(int a) : p{ new X{ a } } {}
    X& operator.() { /* maybe some code here */ return *p; }
    ~Ref() { delete p; }
    void rebind(X* pp) { delete p; p = pp; } // ...
private:
    X* p;
};

Ref<X> x{ 99 };
x.f(); // means (x.operator.()).f() means (*x.p).f()
x = X{ 9 }; // means x.operator.() = X{9} means (*x.p) = X{9}
```

# Nested namespace

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- 네임스페이스를 중첩해서 사용할 수 있는 기능

```
// until C++14
namespace A
{
    namespace B
    {
        namespace C
        {
            // ...
        }
    }
}
```

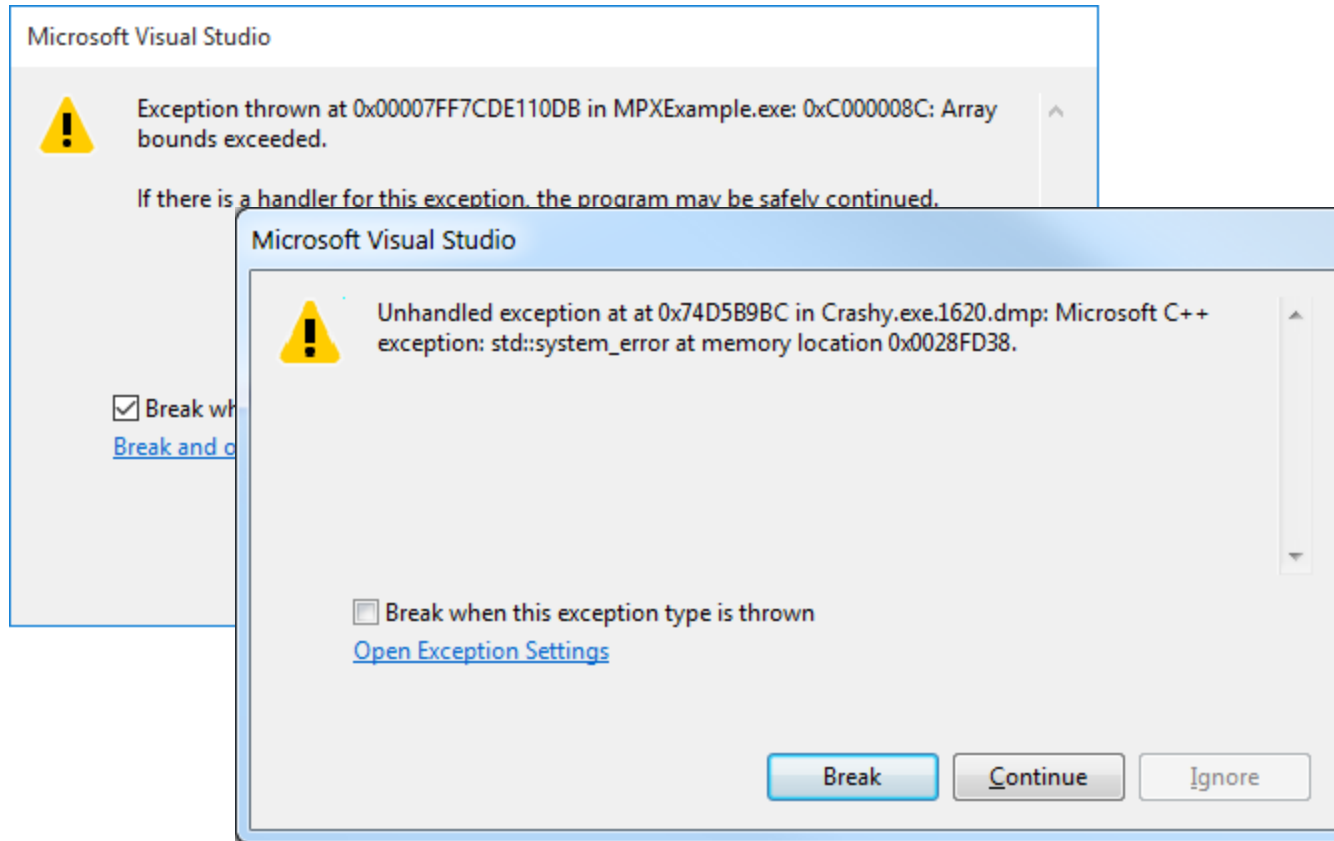


```
// since C++17
namespace A::B::C
{
    // ...
}
```

# Contract

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- 우리는 늘 버그와 싸운다!



# Contract

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

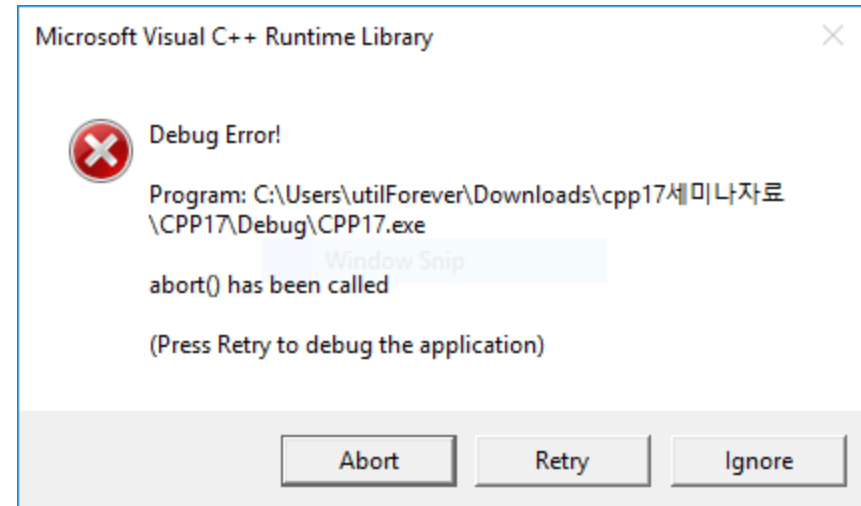
- 그래서 우리가 사용하는 방법, `assert()`!

```
void print_number(int* myInt)
{
    assert(myInt != nullptr);
    std::cout << *myInt << "\n";
}

int main()
{
    int a = 10;
    int *b = nullptr, *c = nullptr;

    b = &a;

    print_number(b);
    print_number(c);
}
```



- 계약(Contract)이란, 함수 앞에 사전 조건과 사후 조건을 추가해 assert()처럼 특정 조건을 만족하는지 검사할 수 있는 기능 (assert()보다 가독성이 좋아 더 직관적이다)
  - 사전 조건(Pre-condition) : **expects**
  - 사후 조건(Post-condition) : **ensures**

```
template<typename T>
class ArrayView
{
    T& operator[](size_t i)[[expects:i < size()]];

    ArrayView(const vector<T>& v)[[ensures:data() == v.data()]];
};
```

- Examples

```
struct A
{
    bool f() const; bool g() const;
    virtual std::string bhar() [[expects:f() && g()]];
    virtual int hash() [[ensures:g()]];
    virtual void gash() [[expects:g()]];
    virtual double fash(int i) const [[expects:i > 0]];
};

struct B : A
{
    std::string bhar() override [[expects:f()]]; // ERROR: weakening.
    int hash() override [[ensures:f() && g()]]; // ERROR: strengthening.
    void gash() override [[expects:g()]]; // OK: repeat from base.
    double fash(int) override; // OK: inherited from base.
};
```

# static\_assert

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- 컴파일 타임에 assert 검사를 수행

```
template <class T>
void swap(T& a, T& b)
{
    static_assert(std::is_copy_constructible<T>::value, "Swap requires copying");
    static_assert(std::is_nothrow_move_constructible<T>::value &&
        std::is_nothrow_move_assignable<T>::value, "Swap may throw");
    auto c = b; b = a; a = c;
}
```

```
int main()
{
    int a, b; swap(a, b);
    no_copy nc_a, nc_b; swap(nc_a, nc_b);
}
```

error C2338: Swap requires copying

- 컴파일 타임에 assert 검사를 수행

```
template <class T>
struct data_structure
{
    static_assert(std::is_default_constructible<T>::value,
        "Data Structure requires default-constructible elements");
};

struct no_copy { no_copy(const no_copy&) = delete; no_copy() = default; };
struct no_default { no_default() = delete; };
```

```
int main()
{
    data_structure<int> ds_ok;
    data_structure<no_default> ds_error;
}
```

error C2338: Data Structure requires default-constructible elements



- C++11의 문제 : auto와 {}로 인해 발생하는 모호함

```
int main()
{
    int v1{ 123 };           // 1  int
    auto v2{ 123 };          // 2  std::initializer_list<int>
    auto v3 = { 123 };       // 3  std::initializer_list<int>
    auto v4{ 1, 2, 3 };      // 4  std::initializer_list<int>
    auto v5 = { 1, 2, 3 };   // 5  std::initializer_list<int>
}
```

```
int main()
{
    int v1{ 123 };           // int
    auto v2{ 123 };          // std::initializer_list<int>
    auto v5 = { 1, 2, 3 };   // std::initializer_list<int>
}
```

- auto와 {}를 통한 Initializer List의 모호함 제거

```
int main()
{
    auto x1{ 1 };           // int
    auto x2 = { 2 };        // initializer_list<int>
    auto x3 = { 1, 2 };     // initializer_list<int>
    auto x4{ 1, 2 };        // error
}
```

# Removed keywords

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- C++17부터 사라지는 키워드들

```
// 1. register
register int a;

// 2. bool increment
bool b = 0;
++b; // bool increment

// 3. auto_ptr
auto_ptr<int> ap;

// 4. random_shuffle
int x[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
random_shuffle(x, x + 10);
```

# Library Features



- 순차적인 STL 처리 → 병렬적인 STL 처리

```
vector<int> v = { 1,2,3 };

// standard sequential sort
sort(v.begin(), v.end());

// explicitly sequential sort
sort(seq, v.begin(), v.end());

// permitting parallel execution
sort(par, v.begin(), v.end());

// permitting vectorization as well
sort(par_vec, v.begin(), v.end());
```

```
// sort with dynamically-selected execution
size_t threshold = 30;
execution_policy exec = seq;

if (v.size() > threshold)
{
    exec = par;
}

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

- Stackless Resumable Functions
  - Lightweight, customizable coroutines
  - Targeted to another technical specifications
  - Experimental implementation in Visual Studio 2015 Update 1
- $2 \times 2 \times 2$ 
  - Two new keywords : await, yield
  - Two new concepts : Awaitable, Coroutine Promise
  - Two new types : resumable\_handle, resumable\_traits
- 자세한 내용은 [CppCon 2014: Gor Nishanov, "await 2.0: Stackless Resumable Function"] 참조  
(<https://www.youtube.com/watch?v=KUHSjfSbINE>)

# Coroutine

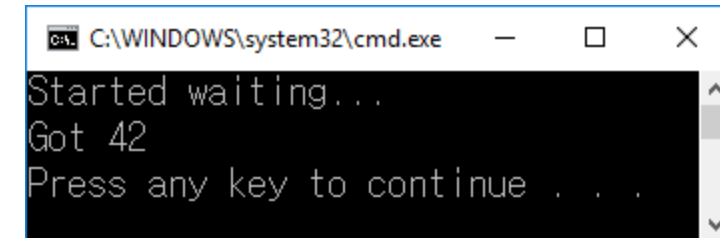
빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- Example 1 : 비동기 작업 (Asynchronous Operations)

```
// this could be some long running computation or I/O
future<int> CalculateTheAnswer()
{
    return std::async([] {
        this_thread::sleep_for(1s);
        return 42;
    });
}

// Here is a resumable function
future<void> Coroutine()
{
    std::cout << "Started waiting... \n";
    auto result = __await CalculateTheAnswer();
    std::cout << "Got " << result << "\n";
}
```

```
int main()
{
    Coroutine().get();
}
```



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window shows the output of the program: "Started waiting..." followed by a new line, "Got 42" followed by a new line, and "Press any key to continue . . .".

# Coroutine

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- Example 2 : 제너레이터 패턴 (Generator Pattern)

```
generator<int> Fibonacci()  
{  
    int a = 0;  
    int b = 1;  
    for (;;)   
    {  
        __yield_value a;  
        auto next = a + b;  
        a = b;  
        b = next;  
    }  
}
```

```
int main()  
{  
    for (auto v : Fibonacci())  
    {  
        if (v > 50)  
            break;  
        cout << v << endl;  
    }  
}
```



```
C:\WINDOWS\system32\cmd.exe  
0  
1  
1  
2  
3  
5  
8  
13  
21  
34  
Press any key to continue
```



- Boost.any 기반 라이브러리
- 모든 종류의 정보를 저장 (단, 복사 생성자를 지원해야 함)
- any\_cast()를 통해 원하는 타입으로 변환된 결과를 얻을 수 있음

```
void foo(std::any a)
{
    double d = std::any_cast<double>(a);
    std::cout << d << std::endl;
}

int main()
{
    foo(3.4);
}
```

# optional

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- Boost.optional 기반 라이브러리
- 함수에서 "유효하지 않은 값"을 리턴하기 위해 만들어짐

```
std::optional<int> Sqrt(int x)
{
    if (x < 0)
        return std::optional<int>();

    int i = 0;
    for (i = 0; i * i <= x; ++i) { }
    return std::optional<int>(i - 1);
}
```

```
int main()
{
    for (int i = -5; i <= 5; ++i) {
        std::optional<int> x = Sqrt(i);

        if (x)
            std::cout << *x << std::endl;
        else
            std::cout << "Invalid\n";
    }
}
```

- Boost.filesystem 기반 라이브러리
- 파일, 경로, 디렉터리와 관련된 작업들을 수행할 수 있음
- Visual Studio 2015에서는 헤더 파일  
`<experimental/filesystem>`을 추가해 사용할 수 있음

```
namespace fs = std::experimental::filesystem;

int main() {
    fs::create_directories("sandbox/a/b");
    std::ofstream("sandbox/file1.txt");
    std::ofstream("sandbox/file2.txt");
    for (auto& p : fs::directory_iterator("sandbox"))
        std::cout << p << '\n';
    fs::remove_all("sandbox");
}
```

- Boost.asio 기반 라이브러리
- 네트워크 및 비동기 프로그래밍 관련 라이브러리

```
try {
    io_service io_service;
    tcp::acceptor acceptor(io_service, tcp::endpoint(tcp::v4(), 13));
    for (;;) {
        tcp::socket socket(io_service);
        acceptor.accept(socket);
        string message = make_daytime_string();
        boost::system::error_code ignored_error;
        boost::asio::write(socket, boost::asio::buffer(message), ignored_error);
    }
} catch (std::exception& e) {
    std::cerr << e.what() << std::endl;
}
```

- C++17 표준화 작업은 현재 진행중입니다.
- 소개한 기능 외에도 다양한 추가/변경 사항들이 있습니다.
- 새 기능이 추가되었다고 무조건 다 알아야 할 필요는 없습니다.
- 모던 C++의 기능을 무조건 사용하기 보다는,  
어떤 기능인지 파악하고 필요한 곳에 적절히 사용합시다.

# References

빠르게 변화하는  
모던 C++에 몸을 맡겨라!

- <http://www.cplusplus.com/reference/>
- <http://en.cppreference.com/w/>
- <https://www.isocpp.org/>
- <http://www.open-std.org/>
- <https://blogs.msdn.microsoft.com/vcblog/>
- <https://github.com/chriskohlhoff/asio-tr2>
- <https://github.com/ericniebler/range-v3>
- <https://github.com/CppCon>

# 감사합니다!

Question?

[utilForever@gmail.com](mailto:utilForever@gmail.com)

<http://fb.com/utilForever>

<http://www.github.com/utilForever>

