

# 불어오는 변화의 바람

## C++98 to C++11/14

김명신 부장 / Microsoft  
옥찬호 대표 / C++ Korea



# WHY C++

# WHY NOT

대중성

범용성

고성능

아는 게 이것 뿐이라

낮은 개발 생산성

복잡성

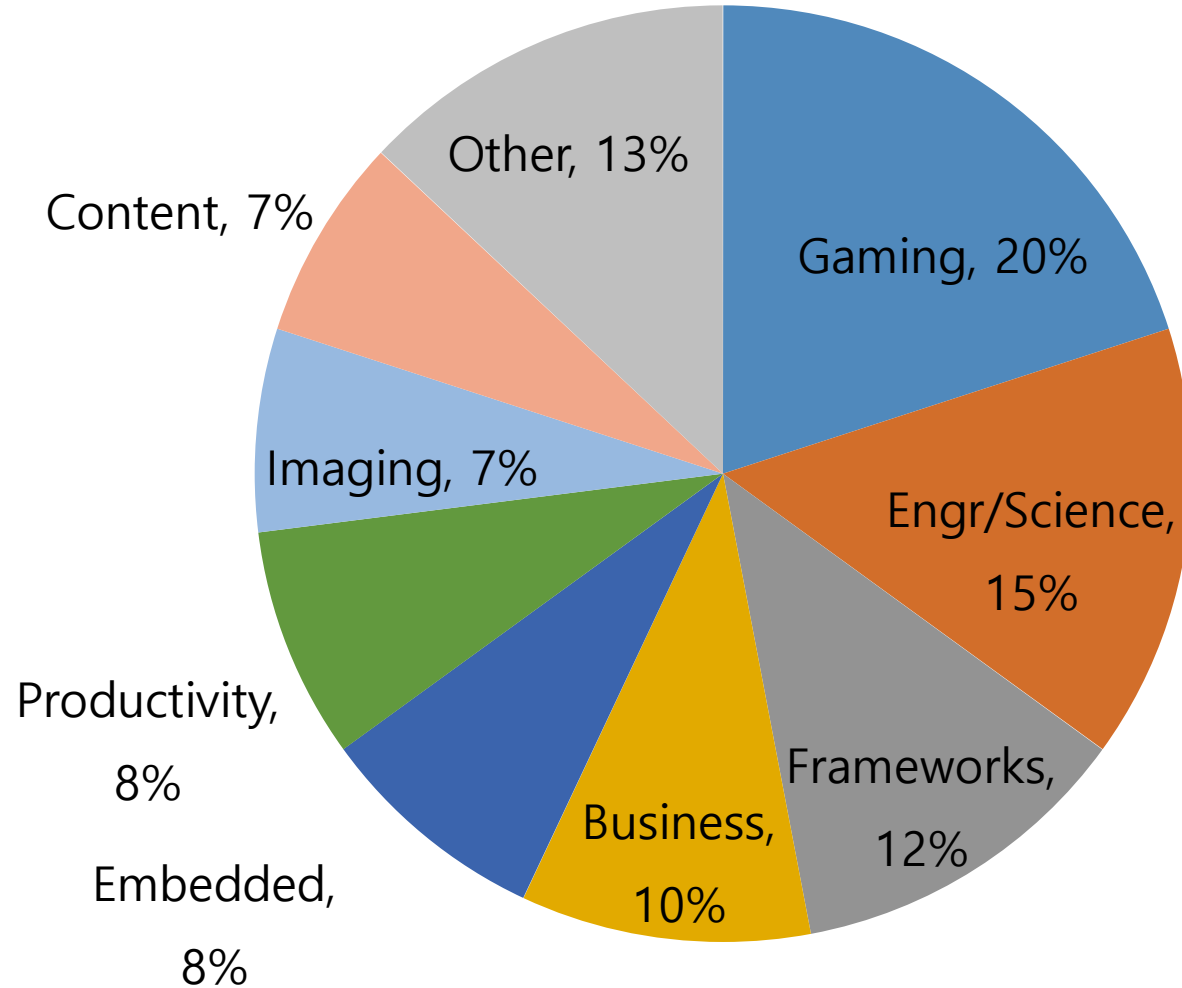
낮은 편의성

발전, 개선 중단

# 대중성

Programming Language	2014	2009	2004	1999	1994	1989
C	1	2	2	1	1	1
Java	2	1	1	3	-	-
Objective-C	3	26	36	-	-	-
C++	4	3	3	2	2	2
C#	5	5	8	17	-	-
PHP	6	4	5	32	-	-
Python	7	6	6	22	22	-
JavaScript	8	8	9	9	-	-
Visual Basic. NET	9	-	-	-	-	-
Perl	10	7	4	4	10	21
Pascal	15	13	80	7	3	23

# 범용성



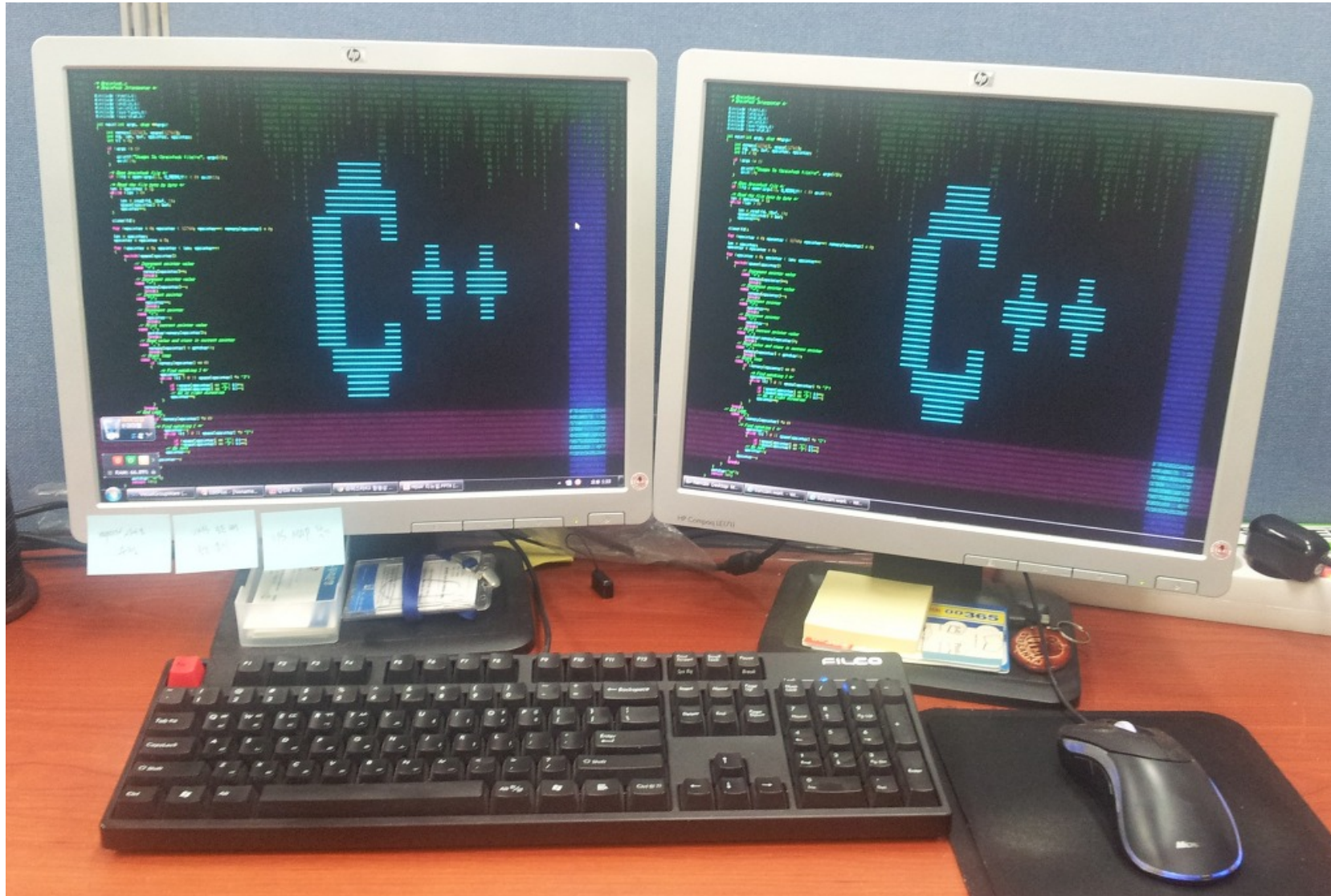
# 고성능

## C++ 설계의 2가지 원칙

Leave no room for  
lower-level language  
below c++

What you don't use you  
don't pay for  
(zero-overhead principle)

# 아이폰





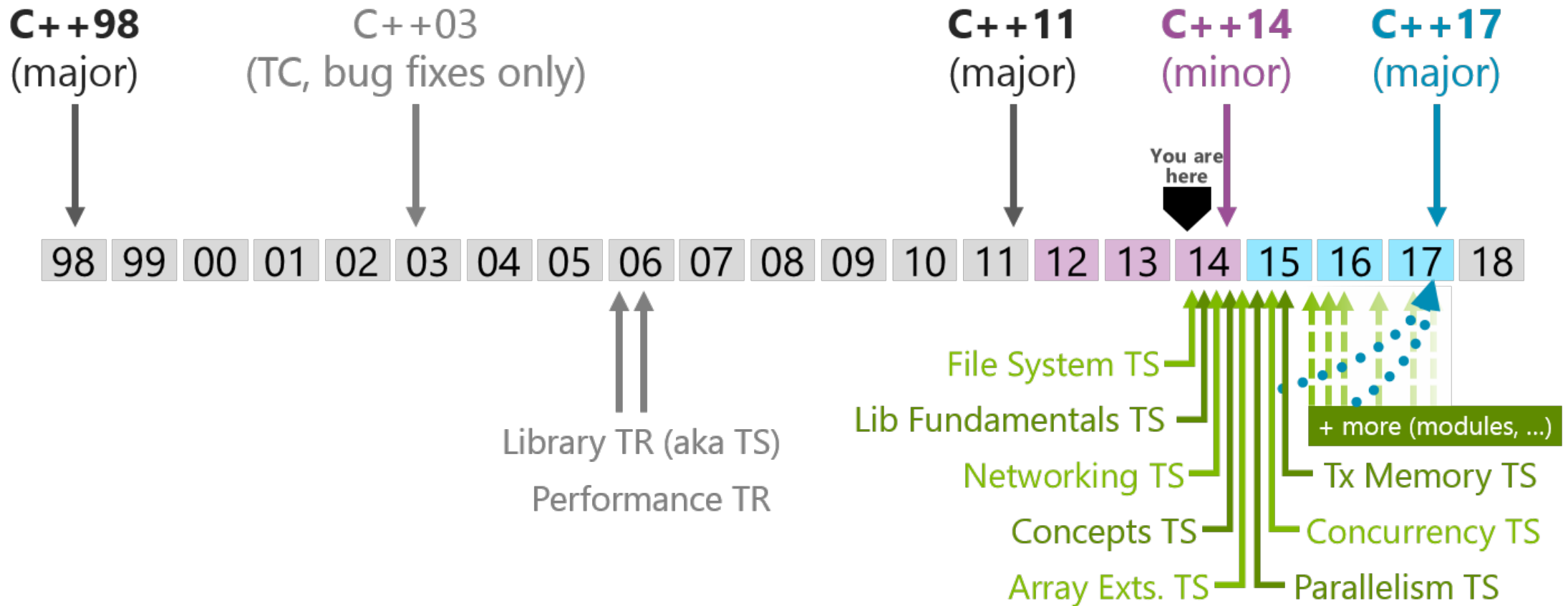
# 낮은 개발 생산성,복잡성,낮은 편의성

- 일부 동의!
- Modern C++의 탄생 배경, 새로운 변화
- 개발비 훨씬 < 운영비



# 발전, 개선 중단

- 살아 움직이는 언어





Performance

# 성능

## King of the Performance / \$

Power	Size	Experiences
<b>Performance / watt</b> Driver at all scales - on-die, mobile, desktop, datacenter	<b>Performance / transistor</b> Limits on processor resources -desktop, mobile	<b>Performance / cycle</b> Bigger experiences on smaller hardware; pushing envelope means every cycle matters

# 시대적 배경 (mobile)



**mobile requirements**

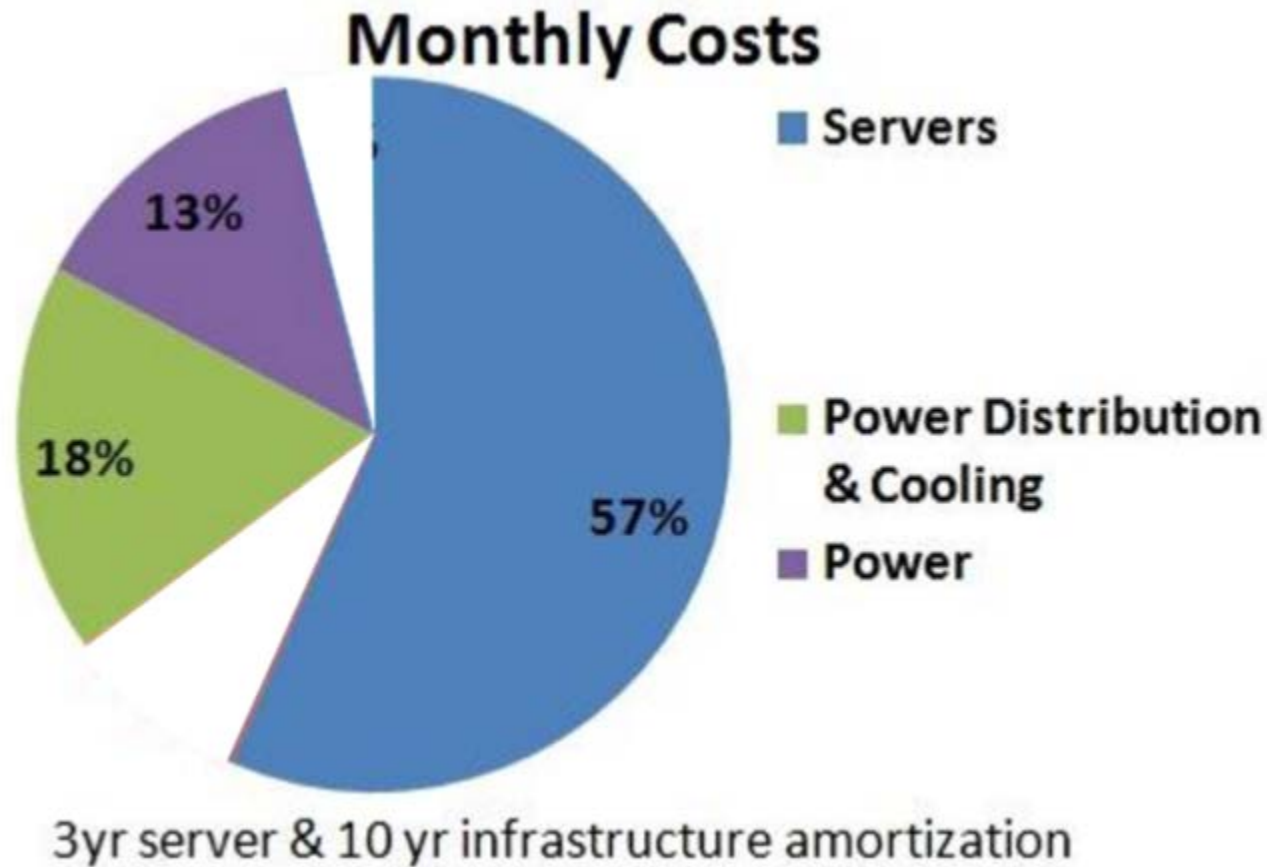


**battery:** lower power, longer life

**size:** amount of hardware  
you can have

**bigger experiences on smaller  
hardware:** pushing the envelope  
means every cycle matters

# 시대적 배경 (data center)



# 내가 지구 온난화에 기여한 바는



“

My contribution to the fight against global warming is C++'s efficiency: Just think if Google had to have twice as many server farms! Each uses as much energy as a small town. And it's not just a factor of two...

Efficiency is not just running fast or running bigger programs, it's also running using less resources.

”

Bjarne Stroustrup, June 2011

# Cross Platform Development



# Cross Platform Development



Version 1

-

Java

.NET

Version 2

**Objective-C,  
C & C++**

Incl. C++ wrappers  
over Objective-c

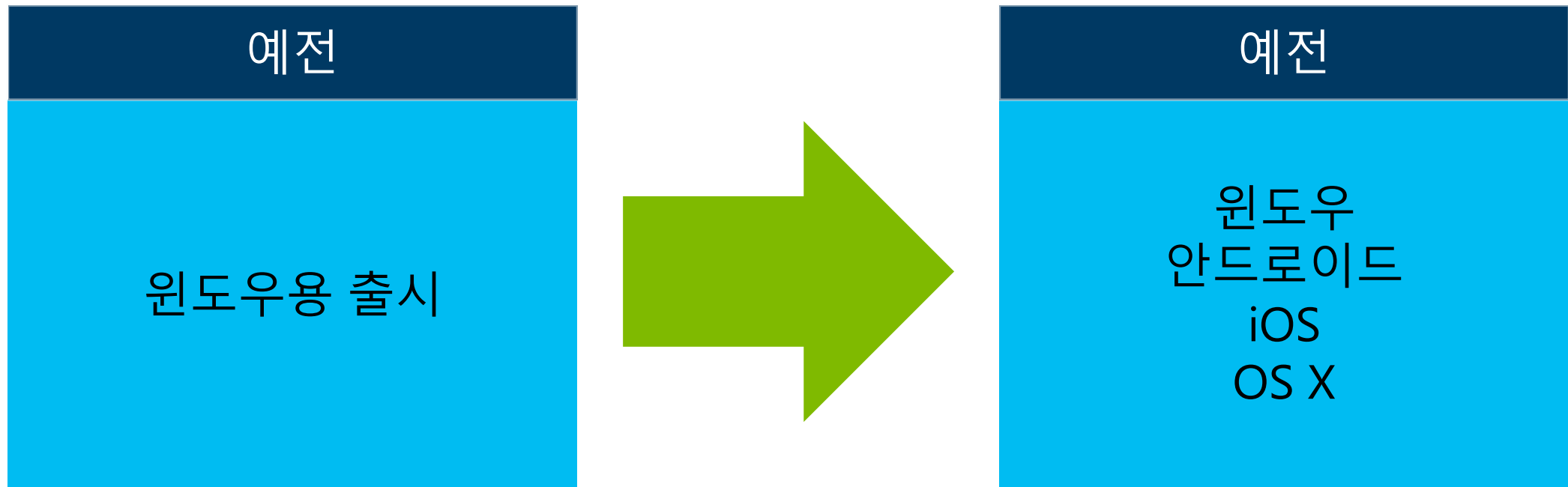
Java,  
**C & C++ NDK**

Incl. java-free C++  
Apps

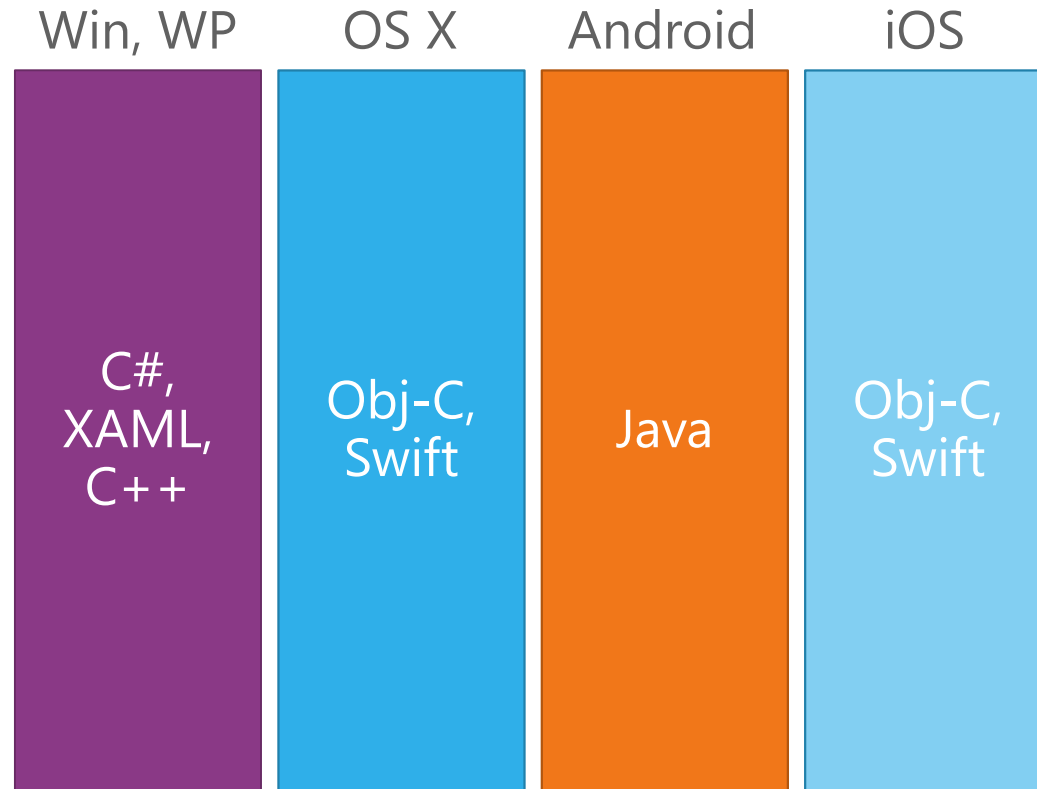
.NET.  
**C & C++**  
WINRTP

# 95%+ 사용자를 위한 앱을 개발하려면

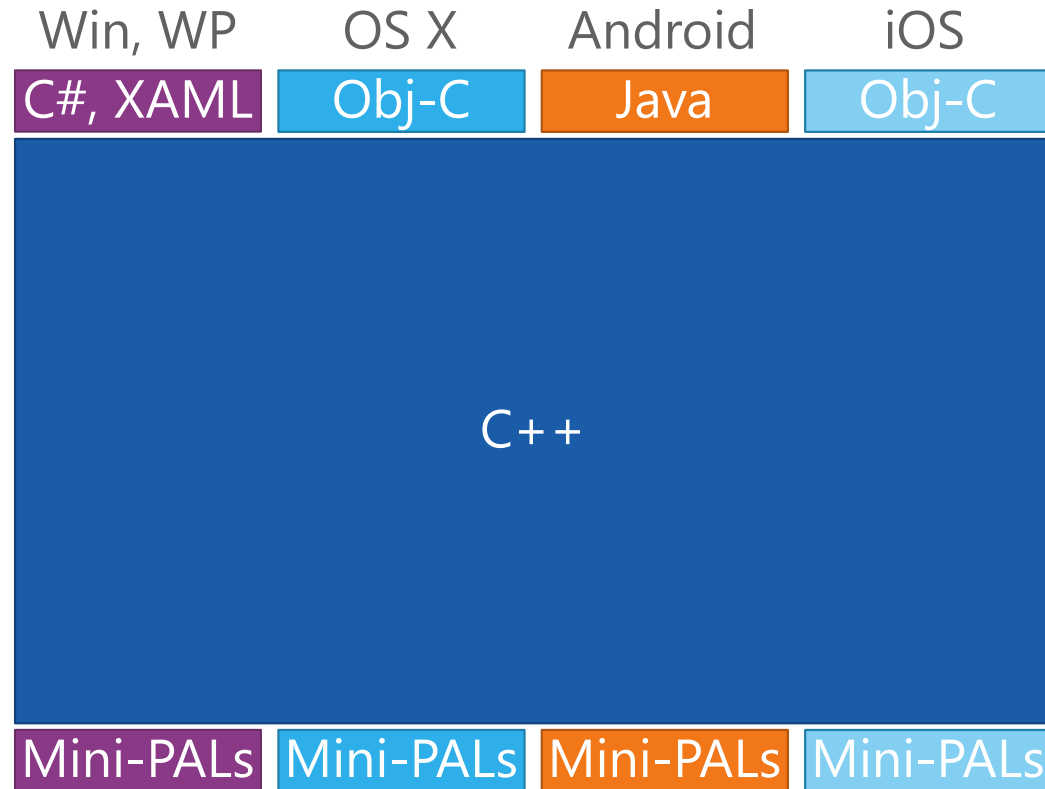
## This is new World!



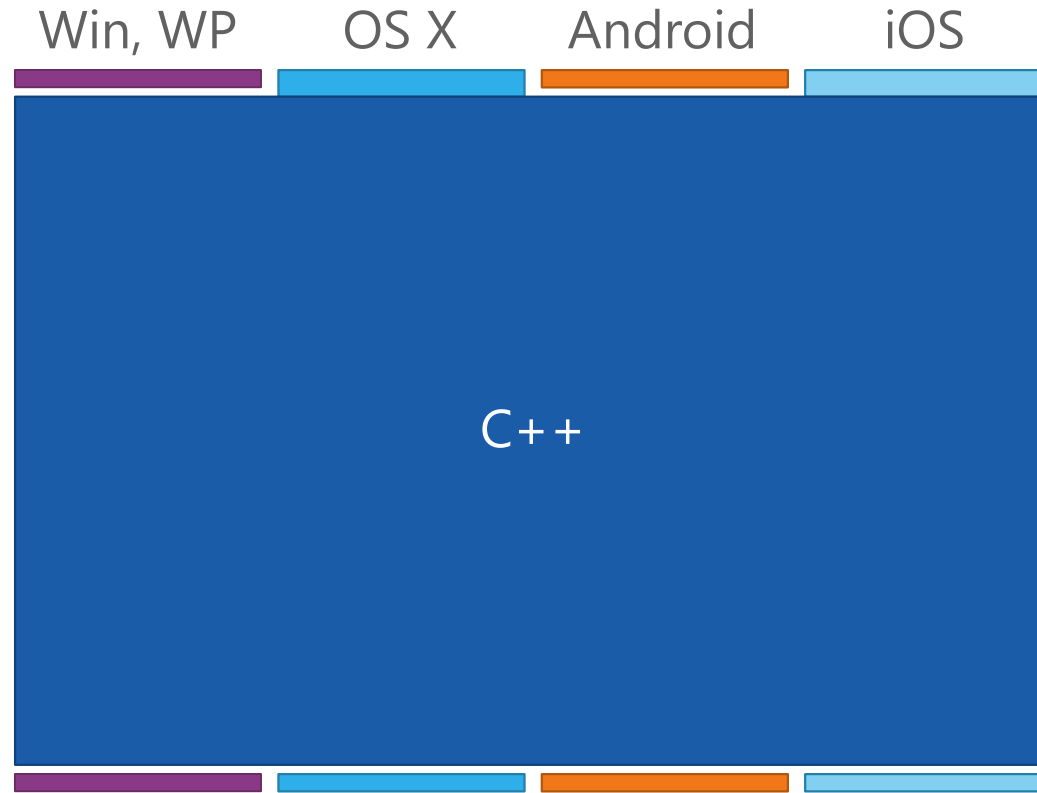
# 이것 만이 유일한 해답일까?



# 이런 대안도



# 해봤더니 (PowerPoint, 95% of code is shared)



# 불어오는 변화의 바람

## C++98 to C++11/14

옥찬호 대표 / C++ Korea





# 목차

1. C++ Korea 소개
2. C++11/14 New Features
3. C++17 Features Preview
4. Related C++ Libraries

# C++ Korea 소개

# C++ Korea

- <https://www.facebook.com/groups/cppkorea/>
- 2013년 11월 개설
- 2014년 마이크로소프트 커뮤니티 멜팅팟 프로그램 2기 선정
- 2014년 10월 1기 운영진 선발
- 2014년 12월 멜팅팟 세미나 개최
- Effective Modern C++ 스터디 예정
  - 자세한 사항은 그룹 공지사항 참조



# 페이스북 이벤트

- <https://www.facebook.com/groups/cppkorea/>
- C++ Korea 그룹에 가입하시고, 현장 사진을 올려주세요!
- 세션 종료 후 추첨을 통해 2분을 선정하여 C++ 관련 서적을 드립니다.
- 추첨 방법 : `srand(NULL);`



# ISO C++ Committee

- <http://isocpp.org/std/the-committee>
- ISO C++ 표준을 승인하는 기구
- 정기적으로 모여 새로운 C++ 표준에 추가하거나 변경, 삭제될 기능을 논의
- 여러 그룹으로 구성되어 있음



SG = Study Group

# VC++ Conformance Update

VC++ Nov 2013 CTP		Visual Studio 2015	
		Available in CTP3+	
		Med. probability RTM	
__func__ Extended sizeof	Thread-safe function local static init	Inline namespaces	UCNs in literals char16_t, char32_t
Implicit move generation	alignof alignas	thread_local Unrestricted unions	Attributes Expression SFINAE
Ref-qualifiers: & and && for *this	noexcept (unconditional)	noexcept (full)	C++11 preprocessor (incl. C++98 & C11)
C++14 decltype(auto)	Inheriting constructors	constexpr (except ctors, literal types)	C++98 two-phase lookup
C++14 auto function return type deduction	User-defined literals	constexpr (full)	C++14 extended constexpr
C++14 generic lambdas (partial)	C++14 generalized lambda capture	C++14 generic lambdas (full)	C++14 variable templates
C++TS? await (partial)	C++14 libs: std:: user- defined literals	C++TS? await	C++TS concepts



# C++11/14 New Features

# Overview

`vector<vector<int>>`    **=default, =delete**    `atomic<T>`    `auto f() -> int`

user-defined literals    `thread_local`    **C++11**    `array<T, N>`    `decltype`

`vector<LocalType>`    **noexcept**

**initializer lists**    `regex`    **extern template**

`constexpr`    `raw string literals`    **async**    `unordered_map<int, string>`

template aliases    `nullptr`    `auto i = v.begin();`    **delegating constructors**

**lambdas**    `override, final`    **variadic templates**    **rvalue references**  
`[] { foo(); }`    `template <typename T...>`    (move semantics)

`unique_ptr<T>`    `thread, mutex`    `function<>`    `future<T>`    `static_assert(x)`  
`shared_ptr<T>`    **for (x : coll)**    **strongly-typed enums**  
`weak_ptr<T>`    `enum class E {...};`    `tuple<int, float, string>`

# Overview

`vector<vector<int>>`    **=default, =delete**    `atomic<T>`    `auto f() -> int`

user-defined literals    `thread_local`    **C++11**    `array<T, N>`    `decltype`

`vector<LocalType>`    **noexcept**

**initializer lists**    `regex`    **extern template**

`constexpr`    `raw string literals`    **async**    `unordered_map<int, string>`

template aliases    `nullptr`    `auto i = v.begin();`    **delegating constructors**

**lambdas**    `override, final`    **variadic templates**    **rvalue references**  
`[] { foo(); }`    `template <typename T...>`    **(move semantics)**

`unique_ptr<T>`    `thread, mutex`    `function<>`    **future<T>**    `static_assert(x)`  
`shared_ptr<T>`    **for (x : coll)**    **strongly-typed enums**  
`weak_ptr<T>`    `enum class E {...};`    `tuple<int, float, string>`

# Overview

vector<vector<int>>    **=default, =delete**    atomic<T>    auto f() -> int

user-defined literals    **thread\_local**    array<T, N>

vector<LocalType>    **C++11**    **decltype**

**initializer lists**    regex    **noexcept**    **extern template**

constexpr    raw string literals    **async**    unordered\_map<int, string>

template aliases    **nullptr**    **auto i = v.begin();**    **delegating constructors**

**lambdas**    **override, final**    **variadic templates**    **rvalue references**  
[[]{ foo(); }          template <typename T...>    (move semantics)

unique\_ptr<T>    thread, mutex    function<>    **future<T>**    static\_assert(x)

shared\_ptr<T>    **for (x : coll)**    **strongly-typed enums**    tuple<int, float, string>  
weak\_ptr<T>    enum class E {...};



# auto

```
map<string, string>::const_iterator it = m.cbegin();  
double const param = config["param"];  
singleton& s = singleton::instance();
```



```
auto it = m.begin();  
auto const param = config["param"];  
auto& s = singleton::instance();
```



컴파일 타임 추론

# decltype

```
template<class T, class U>  
??? add(T x, U y)  
{  
    return x + y;  
}
```

템플릿 함수의 반환형은 컴파일 타임 때 알고 있어야 함

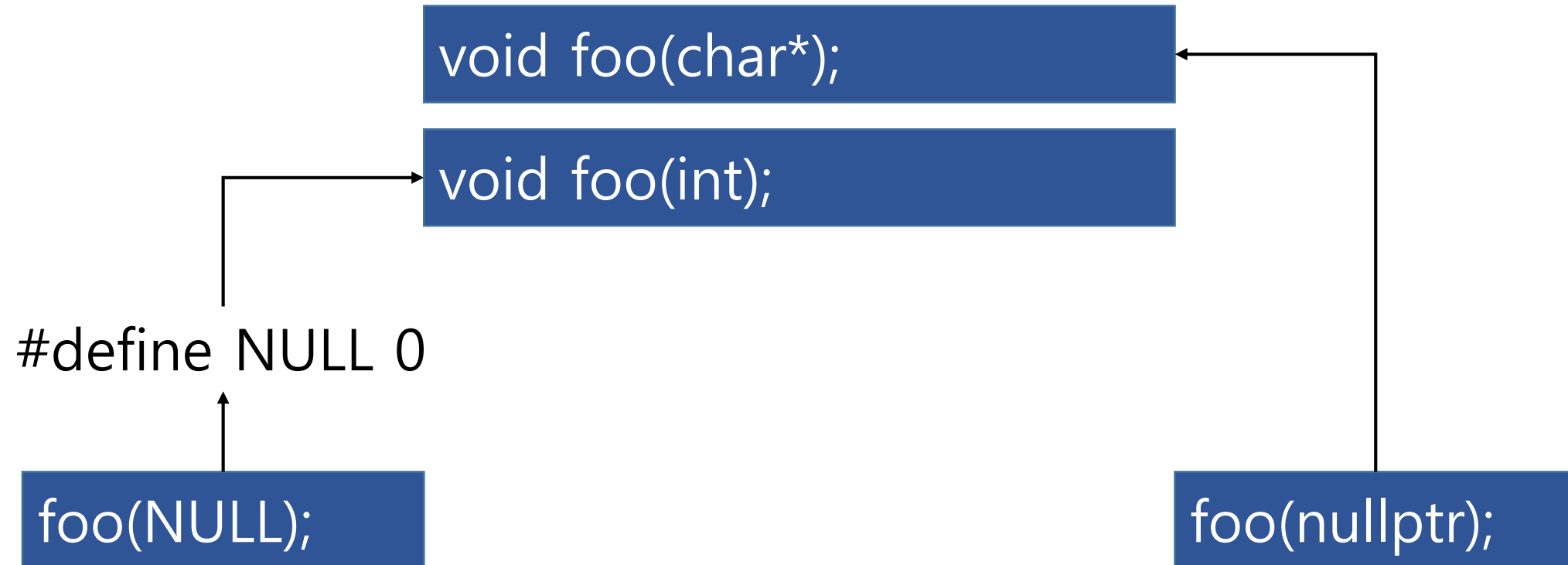


# decltype

```
template<class T, class U>  
decltype(x+y) add(T x, U y)  
{  
    return x + y;  
}
```

decltype를 사용하면 컴파일 타임 때 반환형을 추론

# nullptr



# Strongly-typed Enums

```
enum Alert { green, yellow, red };
```

```
int lightColor = red;
```

enum → int  
Conversion



```
enum class Alert { green, yellow, red };  
int lightColor = red; // 오류  
Alert lightColor = Alert::red;  
int convertColor = static_cast<int>(Alert::red);
```

# Uniform Initialization

```
vector<int> v;  
v.push_back(10);  
v.push_back(20);
```

```
map<int, string> labels;  
labels.insert(make_pair(1, "Open"));  
labels.insert(make_pair(2, "Close"));
```

데이터를 컨테이너에 **각각** 추가  
추가하는 방법도 **다름**

# Uniform Initialization

```
vector<int> v = {10, 20};
```

Diagram illustrating Uniform Initialization for a `vector`. The type `vector<int>` is highlighted with a red box. The initializer list `{10, 20}` is highlighted with a green box. An arrow points from the initializer list to the text `initializer_list<int>`, indicating that the list is of type `initializer_list<int>`.

```
map<int, string> labels {  
    {1, "Open"}, {2, "Close"}  
};
```

Diagram illustrating Uniform Initialization for a `map`. The type `map<int, string>` is highlighted with a red box. The initializer list `{1, "Open"}, {2, "Close"}` is highlighted with a green box. An arrow points from the initializer list to the text `initializer_list<pair<int, string>>`, indicating that the list is of type `initializer_list<pair<int, string>>`.

데이터를 컨테이너에 **일괄** 추가  
추가하는 방법도 **같음**

# std::initializer\_list

```
template<class T>
struct S {
    vector<T> v;
    S(initializer_list<T> l) : v(l) { }
    void append(initializer_list<T> l) {
        v.insert(v.end(), l.begin(), l.end());
    }
};
```

```
S<int> s = {1, 2, 3, 4, 5};
s.append({6, 7, 8});
```

```
S<char> t = {'a', 'b'};
s.append({'c', 'd', 'e', 'f'});
```

# Variadic Template

```
template<class T>
void print_list(T value)
{
    cout << value << endl;
}
```

```
template<class First, class ...Rest>
void print_list(First first, Rest ...rest) {
    cout << first << ", ";
    print_list(rest...);
}
```

# Variadic Template

```
print_list(42, "hello", 2.3, 'a');
```



```
42, hello, 2.3, a
```

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

```
42
```

```
print_list(first = "hello", ...rest = 2.3, 'a')
```

```
hello
```

```
print_list(first = 2.3, ...rest = 'a')
```

```
2.3
```

```
print_list(value = 'a')
```

```
a
```



재귀 함수를 끝내기 위한  
별도의 함수 필요

```
template<class T>  
void print_list(T value) { ... }
```



# Delegating Constructors

```
class A {  
    int a;  
    void validate(int x) {  
        if (x > 0 && x <= 42) a = x;  
    }  
public:  
    A(int x)        { validate(x); }  
    A()             { validate(42); }  
    A(string s)     { int x = stoi(s); validate(x); }  
};
```

생성자를 각각 구현하고, 별도의 함수 호출

# Delegating Constructors

```
class A {  
    int a;  
public:  
    A(int x) {  
        if (x > 0 && x <= 42) a = x;  
    }  
    A()          : A(42) {}  
    A(string s)  : A(stoi(s)) {}  
};
```

1개의 생성자를 구현한 뒤, 위임 호출

# C++17 Features Preview

# C++17 Features Preview

- Extended constants evaluation
- Folding expressions
- `uncaught_exceptions`
- Forwarding references
- `u8` character literals
- Nested namespace definitions
- `auto x{y};`
- `auto_ptr`, `bind1st`/`bind2nd`, `ptr_fun`/`mem_fun`/`mem_fun_ref`, `random_shuffle` all removed

# Folding Expressions

```
template <typename... Args>
bool all(Args... args) {
    return (args && ...);
}

bool b      = all(true, true, true, false);
bool b2     = all();
```

Diagram illustrating the folding of the expression `(args && ...)` in the `all` function. The expression is highlighted with a red box. An arrow points from the red box to the expanded expression `((true && true) && true) && false`, which is shown in green text. Another arrow points from the red box to the `all(true, true, true, false);` line in the code below.

# Folding Expressions

```
template <typename... Args>
bool all(Args... args) {
    return (args && ...);
}

bool b    = all(true, true, true, false);
bool b2   = all();
```

Diagram illustrating expression folding:

- The expression `(args && ...)` in the `return` statement is highlighted with a red box.
- A dashed arrow points from this box to the word `true` in green, indicating the result of the fold.
- A dashed arrow points from the `all()` call in the variable declaration `bool b2 = all();` (where `all()` is also highlighted with a red box) up to the `return` statement, showing the call being folded into the function body.

# Related C++ Libraries

# Boost

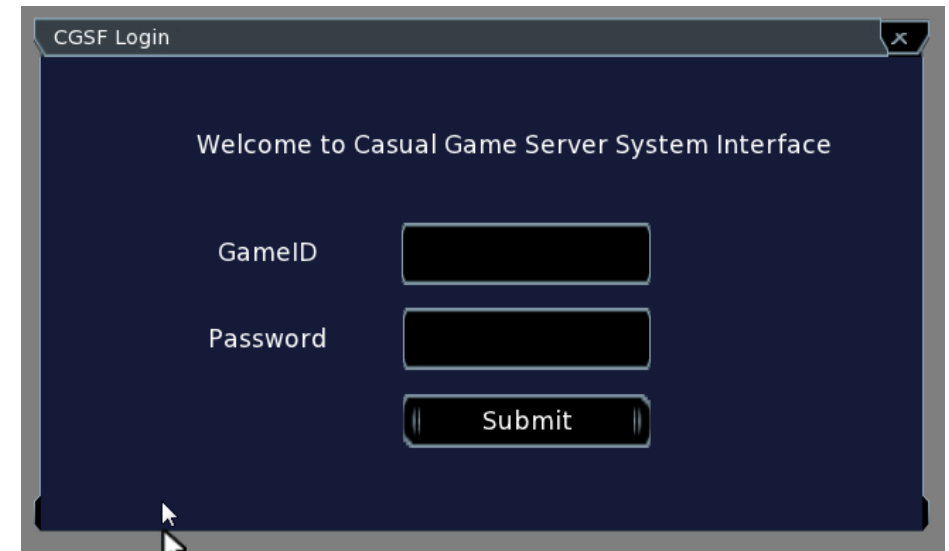
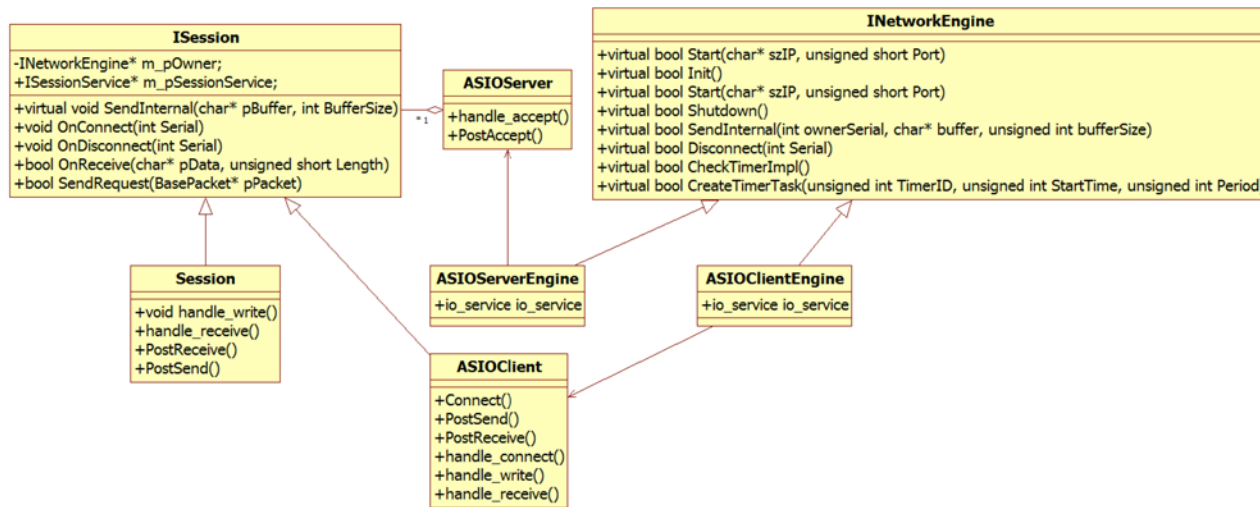
- <http://www.boost.org>
- C++ 위원회 멤버들로부터 시작된 오픈 소스 라이브러리
- C++ 표준 라이브러리가 업데이트 될 때 Boost 라이브러리에 있는 많은 기능들이 채택됨
- Boost.Asio, Boost.Log, Boost.ScopeExit 등
- References
  - boost.org 문서
  - Boost.Asio를 이용한 네트워크 프로그래밍 (한빛미디어, 2013)





# Case Studies : CGSF

- <https://github.com/pdpdds/CGSF>
- 캐주얼 게임을 위해 제작된 서버 라이브러리
- Boost.Asio를 커스터마이징해 네트워크 엔진으로 사용



# Qt

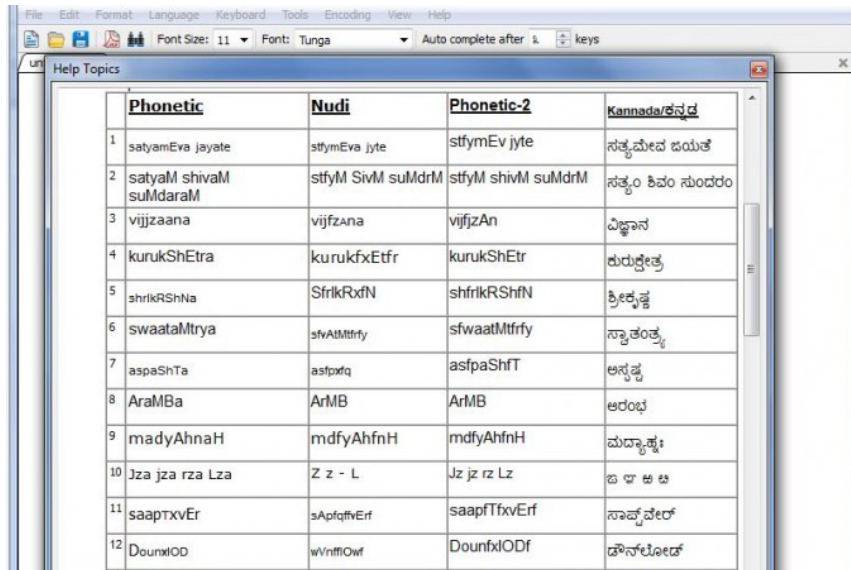
- <http://qt-project.org>
- GUI 프로그램 개발에 널리 쓰이는 크로스 플랫폼 프레임워크
- C++을 주로 사용하지만, Python, Ruby, C, Perl, Pascal과도 연동
- SQL DB 접근, XML 처리, Thread 관리, 파일 관리 API 제공
- References
  - qt-project.org 문서
  - QT5 프로그래밍 가이드(성안당, 2014)
  - QT4를 이용한 C++ GUI 프로그래밍 (아이티씨, 2009)



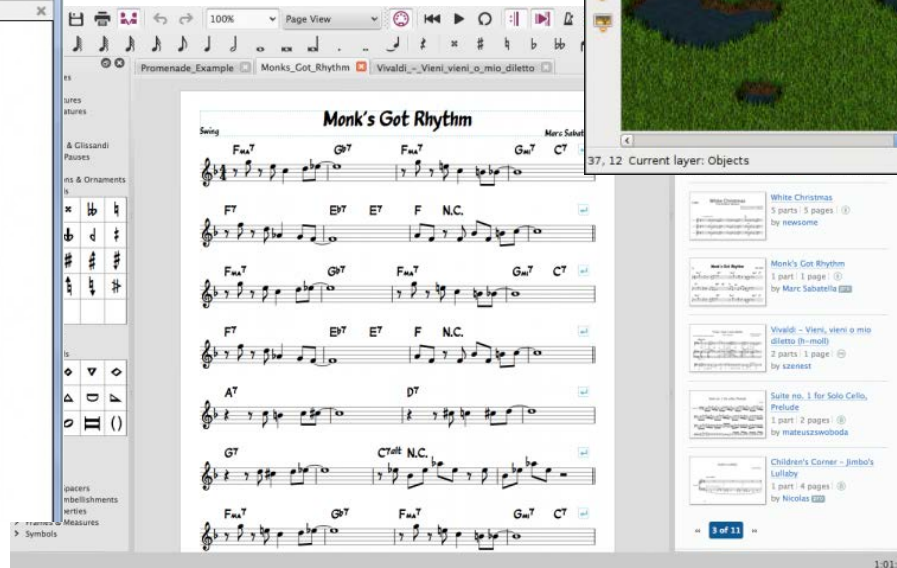
Code less.  
Create more.  
Deploy everywhere.

# Case Studies

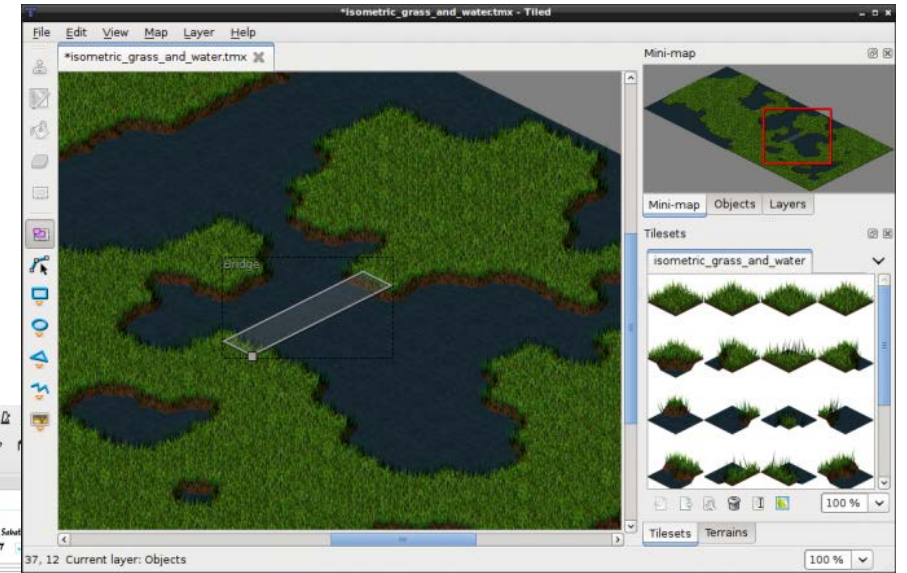
- MuseScore – 작곡 프로그램
- Pada Software – 워드프로세서
- Tiled – 2D 타일 맵 에디터



	Phonetic	Nudi	Phonetic-2	Kannada/ನುಡಿ
1	satyamEva jayate	stfymEva jyte	stfymEv jyte	ಸತ್ಯಮೇವ ಜಯತೇ
2	satyaM shivaM suMdarāM	stfym SivM suMdrM	stfym shivM suMdrM	ಸತ್ಯಂ ಶಿವಂ ಸುಂದರಂ
3	vijjaana	vijfzana	vijfzAn	ವಿಜ್ಞಾನ
4	kurukShEtra	kurukfxEtr	kurukShEtr	ಕುರುಕ್ಷೇತ್ರ
5	shrikRShNa	StrikRxfN	shfrikRShfN	ಶ್ರೀಕೃಷ್ಣ
6	swaataMtrya	sfvaatMtrfy	sfwaatMtrfy	ಸ್ವಾತಂತ್ರ್ಯ
7	aspaShTa	asfpaxq	asfpaShfT	ಅಸ್ಪೃಶ್ಯ
8	AraMBa	ArMB	ArMB	ಅರಂಭ
9	madyAhnaH	mdfyAhfnH	mdfyAhfnH	ಮಧ್ಯಾಹ್ನ
10	Jza jza rza Lza	Z z - L	Jz jz rz Lz	ಜ ಝ ಜಞ
11	saaptxvEr	sAptfxfvEr	saaptTfxvEr	ಸಾಪ್ತವೇರ
12	DounxOD	wVnfxOwf	DounfxODf	ದೌನ್‌ವೇಡ್



The screenshot shows the MuseScore software interface. The main window displays musical notation for a piece titled "Monk's Got Rhythm" by Marc Sabatella. The notation is in 4/4 time and includes various chords and melodic lines. The interface includes a toolbar at the top, a sidebar on the left with various tools, and a bottom status bar showing the current measure and time.



# C++ AMP(Accelerated Massive Parallelism)

- <http://msdn.microsoft.com/ko-kr/library/hh265136.aspx>
- VS C++에서 GPGPU 프로그래밍 환경을 제공
- 또 다른 컴파일러나 다른 구문을 배울 필요가 없음
- C++ AMP는 DirectX의 DirectCompute 사용,  
Microsoft Vista 이상에서만 사용 가능 (DirectX10에서 지원)
- References
  - MSDN 문서
  - C++ AMP : Visual C++와 GPGPU를 이용한  
대규모 병렬 프로그래밍 (한빛미디어, 2013)



# C++ AMP Demo



Microsoft