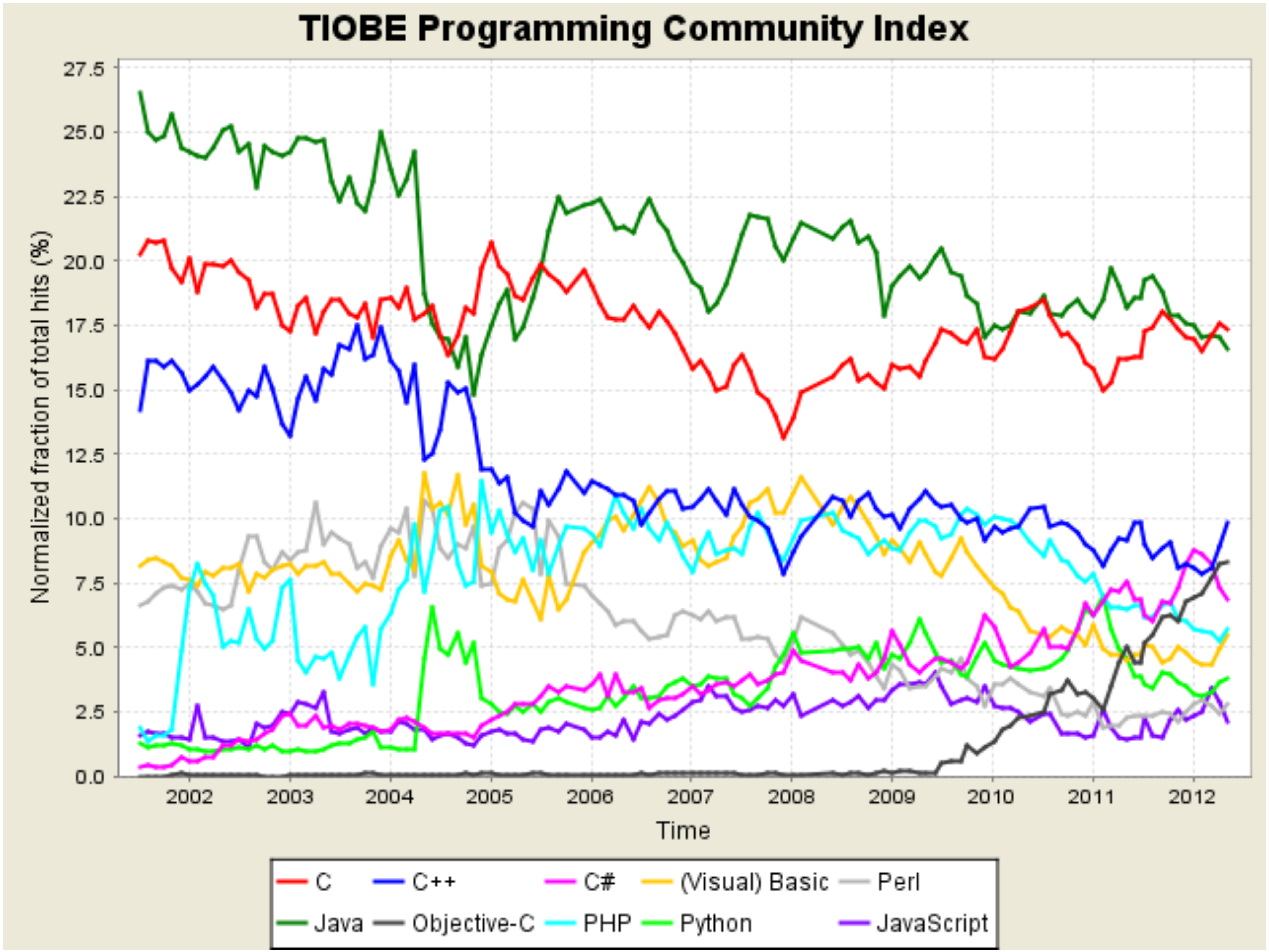
C++11 - the new C++ standard

Olve Maudal & Lars Gullik Bjønnes

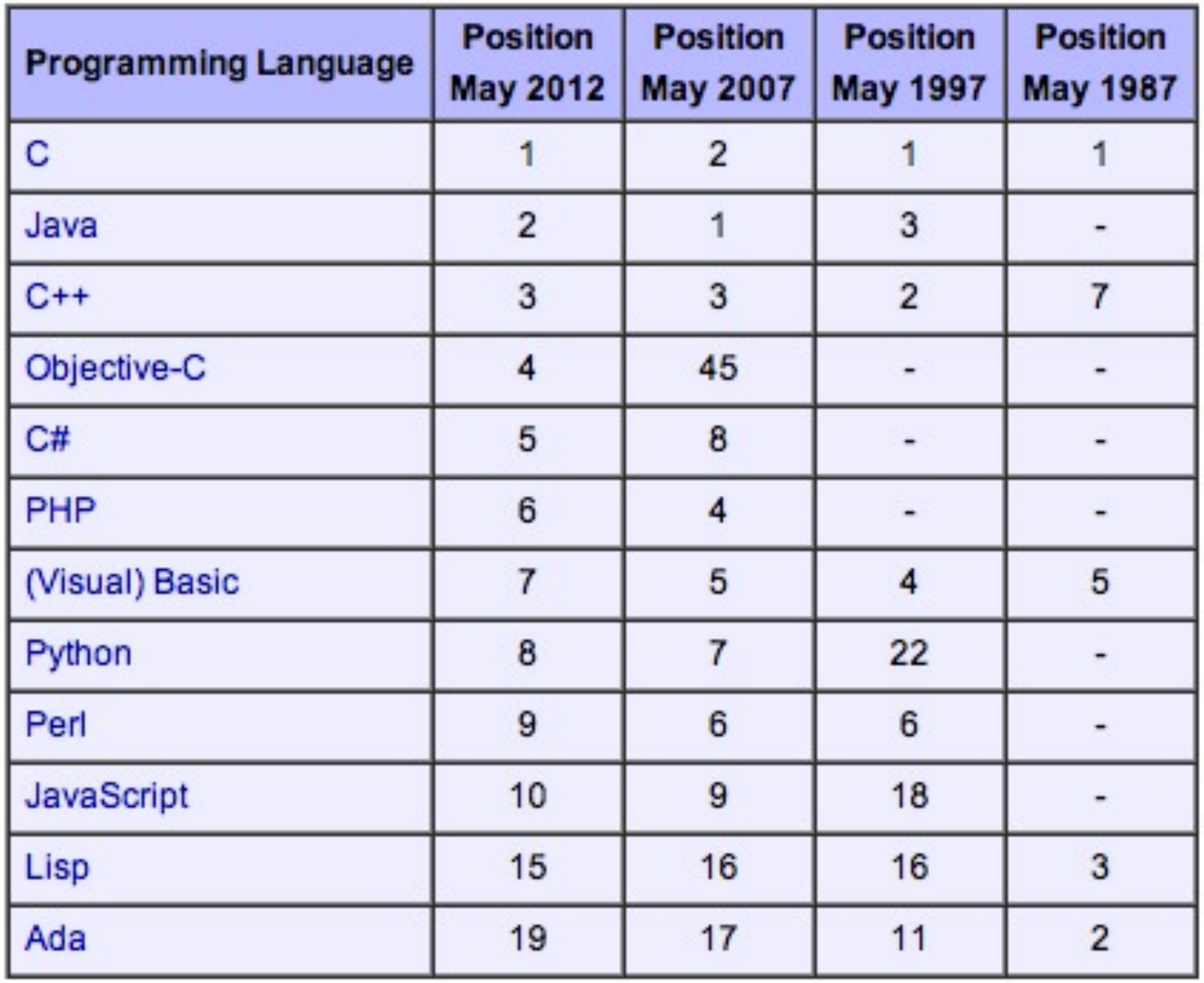
C++11 is the new standard for C++. It adds features like lambda expressions, move semantics, type inference, new memory model supporting threading and a much improved standard library, just to mention a few...

90 minute presentation for baksia.org
May 30, 2012



www.tiobe.com

May 2012



Programming Language	Position May 2012	Position May 2007	Position May 1997	Position May 1987
C	1	2	1	1
Java	2	1	3	-
C++	3	3	2	7
Objective-C	4	45	-	-
C#	5	8	-	-
PHP	6	4	-	-
(Visual) Basic	7	5	4	5
Python	8	7	22	-
Perl	9	6	6	-
JavaScript	10	9	18	-
Lisp	15	16	16	3
Ada	19	17	11	2

Why C++11?

Why C++11?

Why C++?

Why C++11?

Why C++?

Why C?

```
#include <iostream>

struct X {
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
}
```

```cpp
#include <iostream>

struct X {
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
}
```

```
$ g++ foo.cpp && ./a.out
```

```
#include <iostream>

struct X {
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
}
```

```
$ g++ foo.cpp && ./a.out
12
```

```cpp
#include <iostream>

class X {
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
}
```

```
#include <iostream>

class X {
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
}
```

```
$ g++ foo.cpp && ./a.out
```

```cpp
#include <iostream>

class X {
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
}
```

```
$ g++ foo.cpp && ./a.out
12
```

```
#include <iostream>

class X {
public:
    X() : a(6), b('z'), c(7) {}
    int get_answer() { return a * c; }
private:
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
    std::cout << x.get_answer() << std::endl;
}
```

```
#include <iostream>

class X {
public:
    X() : a(6), b('z'), c(7) {}
    int get_answer() { return a * c; }
private:
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
    std::cout << x.get_answer() << std::endl;
}
```

```
$ g++ foo.cpp && ./a.out
```

```
#include <iostream>

class X {
public:
    X() : a(6), b('z'), c(7) {}
    int get_answer() { return a * c; }
private:
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
    std::cout << x.get_answer() << std::endl;
}
```

```
$ g++ foo.cpp && ./a.out
12
```

```
#include <iostream>

class X {
public:
    X() : a(6), b('z'), c(7) {}
    int get_answer() { return a * c; }
private:
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
    std::cout << x.get_answer() << std::endl;
}
```

```
$ g++ foo.cpp && ./a.out
12
42
```

C

```c
#include <stdio.h>

struct X {
    int a;
    char b;
    int c;
};

int get_answer(struct X * this) {
    return this->a * this->c;
}

int main()
{
    struct X x = {6,'z',7};
    printf("%lu\n", sizeof x);
    printf("%d\n", get_answer(&x));
}
```

C

```c
#include <stdio.h>

struct X {
    int a;
    char b;
    int c;
};

int get_answer(struct X * this) {
    return this->a * this->c;
}

int main()
{
    struct X x = {6,'z',7};
    printf("%lu\n", sizeof x);
    printf("%d\n", get_answer(&x));
}
```

```
$ gcc foo.cpp && ./a.out
```

C

```
#include <stdio.h>

struct X {
    int a;
    char b;
    int c;
};

int get_answer(struct X * this) {
    return this->a * this->c;
}

int main()
{
    struct X x = {6,'z',7};
    printf("%lu\n", sizeof x);
    printf("%d\n", get_answer(&x));
}
```

```
$ gcc foo.cpp && ./a.out
12
```

C

```
#include <stdio.h>

struct X {
    int a;
    char b;
    int c;
};

int get_answer(struct X * this) {
    return this->a * this->c;
}

int main()
{
    struct X x = {6,'z',7};
    printf("%lu\n", sizeof x);
    printf("%d\n", get_answer(&x));
}
```

```
$ gcc foo.cpp && ./a.out
12
42
```

C++

```cpp
#include <iostream>

class X {
public:
    X() : a(6), b('z'), c(7) {}
    int get_answer() { return a * c; }
private:
    int a;
    char b;
    int c;
};

int main()
{
    X x;
    std::cout << sizeof x << std::endl;
    std::cout << x.get_answer() << std::endl;
}
```

C

```c
#include <stdio.h>

struct X {
    int a;
    char b;
    int c;
};

int get_answer(struct X * this) {
    return this->a * this->c;
}

int main()
{
    struct X x = {6,'z',7};
    printf("%lu\n", sizeof x);
    printf("%d\n", get_answer(&x));
}
```

```
%include “mylib.asm”

section .text
get_answer:
        mov eax, dword [esp+4]
        imul eax, dword [esp+12]
        ret

global start
start:
        ; allocate and initialize struct
        sub esp, 12
        mov [esp], dword 6
        mov [esp+4], dword 'z'
        mov [esp+8], dword 7

        ; print 12
        push dword 12
        call print_int_and_nl
        add esp, 4

        ; calculate answer and print result
        call get_answer
        push eax
        call print_int_and_nl
        add esp, 4

        ; exit with return value 0
        push 0
        mov eax, 1
        push dword 0
        int 80h
```

```
%include “mylib.asm”

section .text
get_answer:
        mov eax, dword [esp+4]
        imul eax, dword [esp+12]
        ret

global start
start:
        ; allocate and initialize struct
        sub esp, 12
        mov [esp], dword 6
        mov [esp+4], dword 'z'
        mov [esp+8], dword 7

        ; print 12
        push dword 12
        call print_int_and_nl
        add esp, 4

        ; calculate answer and print result
        call get_answer
        push eax
        call print_int_and_nl
        add esp, 4

        ; exit with return value 0
        push 0
        mov eax, 1
        push dword 0
        int 80h
```

```
$ nasm -f macho ex_sizeof.asm
```

```
%include “mylib.asm”

section .text
get_answer:
        mov eax, dword [esp+4]
        imul eax, dword [esp+12]
        ret

global start
start:
        ; allocate and initialize struct
        sub esp, 12
        mov [esp], dword 6
        mov [esp+4], dword 'z'
        mov [esp+8], dword 7

        ; print 12
        push dword 12
        call print_int_and_nl
        add esp, 4

        ; calculate answer and print result
        call get_answer
        push eax
        call print_int_and_nl
        add esp, 4

        ; exit with return value 0
        push 0
        mov eax, 1
        push dword 0
        int 80h
```

```
$ nasm -f macho ex_sizeof.asm
$ ld ex_sizeof.o
```

```
%include “mylib.asm”

section .text
get_answer:
        mov eax, dword [esp+4]
        imul eax, dword [esp+12]
        ret

global start
start:
        ; allocate and initialize struct
        sub esp, 12
        mov [esp], dword 6
        mov [esp+4], dword 'z'
        mov [esp+8], dword 7

        ; print 12
        push dword 12
        call print_int_and_nl
        add esp, 4

        ; calculate answer and print result
        call get_answer
        push eax
        call print_int_and_nl
        add esp, 4

        ; exit with return value 0
        push 0
        mov eax, 1
        push dword 0
        int 80h
```

```
$ nasm -f macho ex_sizeof.asm
$ ld ex_sizeof.o
$ ./a.out
```

```
%include “mylib.asm”

section .text
get_answer:
        mov eax, dword [esp+4]
        imul eax, dword [esp+12]
        ret

global start
start:
        ; allocate and initialize struct
        sub esp, 12
        mov [esp], dword 6
        mov [esp+4], dword 'z'
        mov [esp+8], dword 7

        ; print 12
        push dword 12
        call print_int_and_nl
        add esp, 4

        ; calculate answer and print result
        call get_answer
        push eax
        call print_int_and_nl
        add esp, 4

        ; exit with return value 0
        push 0
        mov eax, 1
        push dword 0
        int 80h
```

```
$ nasm -f macho ex_sizeof.asm
$ ld ex_sizeof.o
$ ./a.out
12
```

```
%include “mylib.asm”

section .text
get_answer:
        mov eax, dword [esp+4]
        imul eax, dword [esp+12]
        ret

global start
start:
        ; allocate and initialize struct
        sub esp, 12
        mov [esp], dword 6
        mov [esp+4], dword 'z'
        mov [esp+8], dword 7

        ; print 12
        push dword 12
        call print_int_and_nl
        add esp, 4

        ; calculate answer and print result
        call get_answer
        push eax
        call print_int_and_nl
        add esp, 4

        ; exit with return value 0
        push 0
        mov eax, 1
        push dword 0
        int 80h
```

```
$ nasm -f macho ex_sizeof.asm
$ ld ex_sizeof.o
$ ./a.out
12
42
```

Why C++11?

Why C++?

Why C?

Why C++11?

Why C++?

Why C?

Managed vs Unmanaged
Productivity vs Performance
Effectiveness vs Efficiency

Why C++11?

Why C++?

Why C?

Managed vs **Unmanaged**
Productivity vs **Performance**
Effectiveness vs **Efficiency**

C++ combats global warming

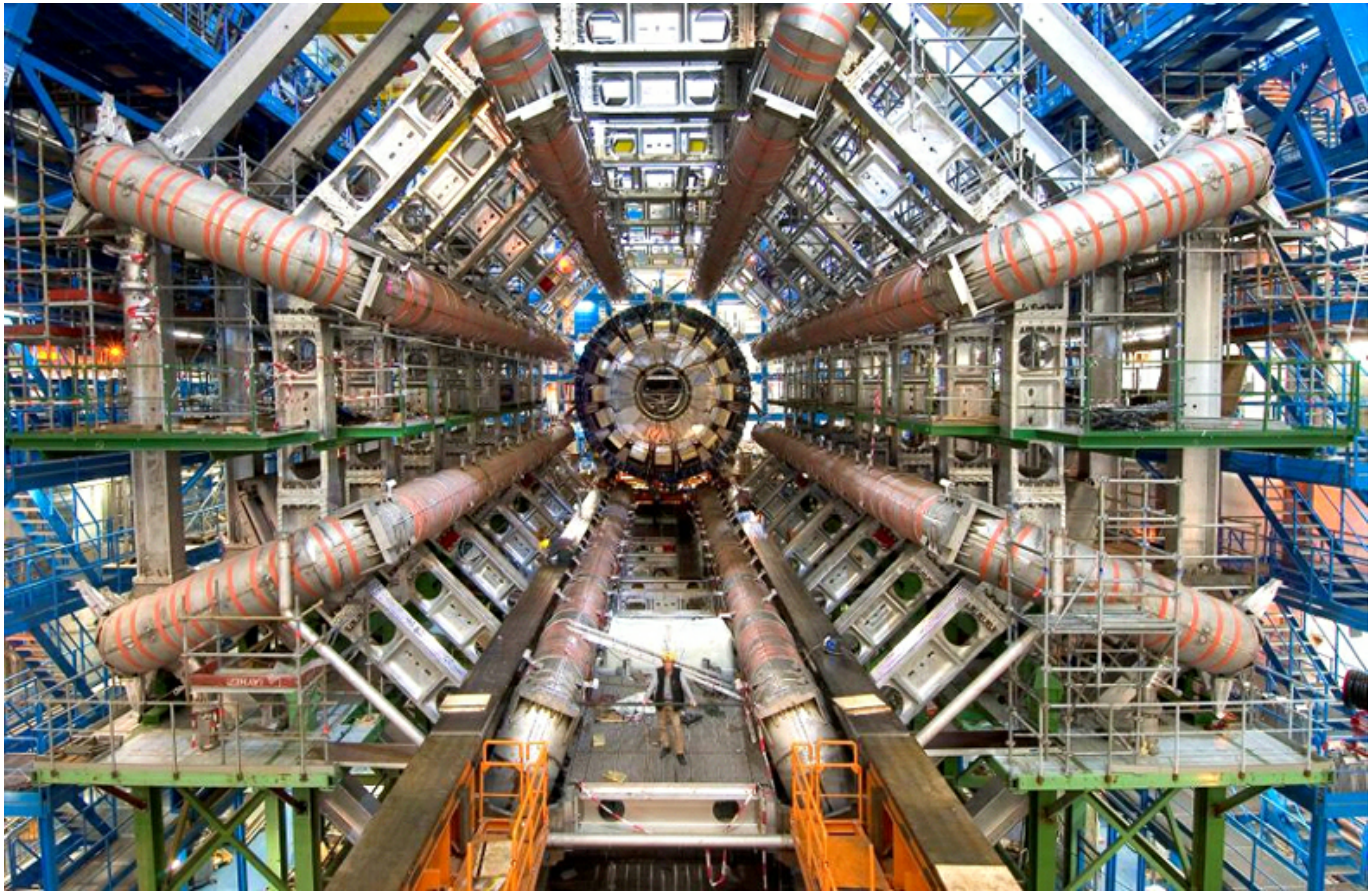
http://www.youtube.com/watch?v=FBWeO2HYEc0

C++ combats global warming

http://www.youtube.com/watch?v=FBWeO2HYEc0

Where is C and C++ relevant?

Motion
Patient Info #245498
Monday, February 28, 3:35 PM
CLINICAL PATHOLOGY LABORATORY REQUEST FORM



ENERGY micro
EFM32
GIANT GECKO
ENERGY micro
EFM32
GECKO
ENERGY
EFM32
TINY GECKO
ENERGY
EFM32
ZERO GECKO

legacy

legacy
embedded

legacy
embedded
realtime

legacy
embedded
realtime
datacenters

legacy
embedded
realtime
datacenters
mobile computing

legacy
embedded
realtime
datacenters
mobile computing
supercomputing

legacy
embedded
realtime
datacenters
mobile computing
supercomputing
green computing

legacy
embedded
realtime
datacenters
mobile computing
supercomputing
green computing

and

legacy
embedded
realtime
datacenters
mobile computing
supercomputing
green computing

and
competition programming!

FizzBuzz Kata

Imagine the scene. You are eleven years old, and in the five minutes before the end of the lesson, your Maths teacher decides he should make his class more "fun" by introducing a "game". He explains that he is going to point at each pupil in turn and ask them to say the next number in sequence, starting from one. The "fun" part is that if the number is divisible by three, you instead say "Fizz" and if it is divisible by five you say "Buzz". So now your maths teacher is pointing at all of your classmates in turn, and they happily shout "one!", "two!", "Fizz!", "four!", "Buzz!"... until he very deliberately points at you, fixing you with a steely gaze... time stands still, your mouth dries up, your palms become sweatier and sweatier until you finally manage to croak "Fizz!". Doom is avoided, and the pointing finger moves on.

So of course in order to avoid embarassment infront of your whole class, you have to get the full list printed out so you know what to say. Your class has about 33 pupils and he might go round three times before the bell rings for breaktime. Next maths lesson is on Thursday. Get coding!

Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz?".

Sample output:
```
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16 17 Fizz 19 Buzz ... etc up to 100
```

http://codingdojo.org/cgi-bin/wiki.pl?KataFizzBuzz

plain implementation of fizzbuzz

```
#include <iostream>

bool is_multiple_of_3(int i) { return i % 3 == 0; }

bool is_multiple_of_5(int i) { return i % 5 == 0; }

int main()
{
    for (int i = 1; i <= 100; ++i) {
        if (is_multiple_of_3(i) && is_multiple_of_5(i) )
            std::cout << "FizzBuzz" << std::endl;
        else if (is_multiple_of_3(i))
            std::cout << "Fizz" << std::endl;
        else if (is_multiple_of_5(i))
            std::cout << "Buzz" << std::endl;
        else
            std::cout << i << std::endl;
    }
}
```

now try to do this kata with, range based for loop, lambda, threads, containers, algorithms, variadic templates...

We are working on a new project...
Which language to use?

can a script language do the job?

is there support for a vm based language?

is C powerful enough for your task?

if no to all previous questions, then you may want to consider C++!

C++

History of C++

- PhD, Simula, BCPL (Cambridge)
- C with Classes (Cpre, 1979)
- First external paper (1981)
- C++ named (1983)
- CFront 1.0 (1985)
- TC++PL, Ed1 (1985)
- ANSI X3J16 meeting (1989)
- The Annotated C++ Reference Manual (1990)
- First WG21 meeting (1991)
- The Design and Evolution of C++ (1994)
- ISO/IEC 14882:1998 (C++98)
- ISO/IEC 14882:2003 (C++03)
- ISO/IEC TR 19768:2007 (C++TR1)
- ISO/IEC 14882:2011 (C++11)

Why building on top of C?

Why building on top of C?

flexible, efficient, available, portable

The spirit of C

trust the programmer

- let them do what needs to be done
- the programmer is in charge not the compiler

keep the language small and simple

- small amount of code → small amount of assembler
- provide only one way to do an operation
- new inventions are not entertained

make it fast, even if its not portable

- target efficient code generation
- int preference, int promotion rules
- sequence points, maximum leeway to compiler

rich expression support

- lots of operators
- expressions combine into larger expressions

Design principles for C++

- C++ is designed to be a statically typed, general-purpose language that is as efficient and portable as C
- C++ is designed to directly and comprehensively support multiple programming styles (procedural programming, data abstraction, object-oriented programming, and generic programming)
- C++ is designed to give the programmer choice, even if this makes it possible for the programmer to choose incorrectly
- C++ is designed to be as compatible with C as possible, therefore providing a smooth transition from C
- C++ avoids features that are platform specific or not general purpose
- C++ does not incur overhead for features that are not used (the "zero-overhead principle")
- C++ is designed to function without a sophisticated programming environment

Mixed feelings about C++

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"
- "C++ programmers are idiots"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"
- "C++ programmers are idiots"
- "C++ is just C, but worse"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"
- "C++ programmers are idiots"
- "C++ is just C, but worse"
- "C++ is useless/unreliable/dangerous because it lacks feature/property X"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"
- "C++ programmers are idiots"
- "C++ is just C, but worse"
- "C++ is useless/unreliable/dangerous because it lacks feature/property X"
- "C++ is not Object-oriented"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"
- "C++ programmers are idiots"
- "C++ is just C, but worse"
- "C++ is useless/unreliable/dangerous because it lacks feature/property X"
- "C++ is not Object-oriented"
- "C++ compilers/tools are too buggy"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"
- "C++ programmers are idiots"
- "C++ is just C, but worse"
- "C++ is useless/unreliable/dangerous because it lacks feature/property X"
- "C++ is not Object-oriented"
- "C++ compilers/tools are too buggy"
- "The C++ standards committee is out of control"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"
- "C++ programmers are idiots"
- "C++ is just C, but worse"
- "C++ is useless/unreliable/dangerous because it lacks feature/property X"
- "C++ is not Object-oriented"
- "C++ compilers/tools are too buggy"
- "The C++ standards committee is out of control"
- "I have heard of lots of C++ disasters"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"
- "C++ programmers are idiots"
- "C++ is just C, but worse"
- "C++ is useless/unreliable/dangerous because it lacks feature/property X"
- "C++ is not Object-oriented"
- "C++ compilers/tools are too buggy"
- "The C++ standards committee is out of control"
- "I have heard of lots of C++ disasters"
- "I just don't like C++"

Mixed feelings about C++

"C++ is a horrible language. It's made more horrible by the fact that a lot of substandard programmers use it", Linus Torvalds

- "C++ sucks"
- "C++ is too complex"
- "C++ programmers are idiots"
- "C++ is just C, but worse"
- "C++ is useless/unreliable/dangerous because it lacks feature/property X"
- "C++ is not Object-oriented"
- "C++ compilers/tools are too buggy"
- "The C++ standards committee is out of control"
- "I have heard of lots of C++ disasters"
- "I just don't like C++"

"The major cause of complaints is C++ undoubted success. As someone remarked: There are only two kinds of programming languages: those people always bitch about and those nobody uses.", Bjarne Stroustrup

C++11 - 10 interesting new features

- move semantics and rvalue references (`&&`)
- automatic type deduction (`auto`, `decltype`)
- uniform initialization and initializer lists
- range based for loops
- lambdas
- atomics / future / promise / thread
- user-defined literals
- tuple
- new smart pointers (`shared_ptr`, `unique_ptr`)
- chrono

rvalue reference

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

void foo(const std::vector<int> & v) {
    std::cout << "1" << std::endl;
}

void foo(std::vector<int> & v) {
    std::cout << "2" << std::endl;
}

void foo(std::vector<int> && v) {
    std::cout << "3" << std::endl;
}

int main()
{
    std::vector<int> v = {1,2,3};
    const std::vector<int> cv = {4,5,6};

    foo(v);
    foo(cv);
    foo({7,8,9});
    foo(std::move(v));
}
```

rvalue reference

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

void foo(const std::vector<int> & v) {
    std::cout << "1" << std::endl;
}

void foo(std::vector<int> & v) {
    std::cout << "2" << std::endl;
}

void foo(std::vector<int> && v) {
    std::cout << "3" << std::endl;
}

int main()
{
    std::vector<int> v = {1,2,3};
    const std::vector<int> cv = {4,5,6};

    foo(v);
    foo(cv);
    foo({7,8,9});
    foo(std::move(v));
}
```

```
2
1
3
3
```

move semantics

```
#include <iostream>
#include <vector>
#include <string>

template<typename T> void P(const T & x) { std::cout << x << " "; }

class msg
{
public:
    msg(std::string to, std::vector<int8_t> data) : to_(to), data_(data) { P(1); }
    msg(const msg & other) : to_(other.to_), data_(other.data_) { P(2); }
    msg(msg && other) : to_(other.to_), data_(other.data_) { P(3); }
    ~msg() { P(9); }
private:
    std::string to_;
    std::vector<int8_t> data_;
};

msg get_default_msg()
{
    return msg("all", {42});
}

int main()
{
    msg m1("olve", {3,1,4});
    msg m2(m1);
    msg m3(std::move(get_default_msg()));
}
```

move semantics

```
#include <iostream>
#include <vector>
#include <string>

template<typename T> void P(const T & x) { std::cout << x << " "; }

class msg
{
public:
    msg(std::string to, std::vector<int8_t> data) : to_(to), data_(data) { P(1); }
    msg(const msg & other) : to_(other.to_), data_(other.data_) { P(2); }
    msg(msg && other) : to_(other.to_), data_(other.data_) { P(3); }
    ~msg() { P(9); }
private:
    std::string to_;
    std::vector<int8_t> data_;
};

msg get_default_msg()
{
    return msg("all", {42});
}

int main()
{
    msg m1("olve", {3,1,4});
    msg m2(m1);
    msg m3(std::move(get_default_msg()));
}
```

```
1 2 1 3 9 9 9 9
```

tuple / range-based for loop

```
#include <iostream>
#include <tuple>
#include <vector>

int main()
{
    std::vector<std::tuple<std::string, int, double>> times;
    times.push_back(std::make_tuple("Olve", 1971, 12.43));
    times.push_back(std::make_tuple("Lars Gullik", 1972, 11.22));
    for (auto t : times)
        std::cout << std::get<0>(t) << std::endl;

    std::string name;
    int year;
    double time;
    std::tie(name, year, time) = times[1];
    std::cout << name << " (" << year << ") : " << time << std::endl;
}
```

tuple / range-based for loop

```cpp
#include <iostream>
#include <tuple>
#include <vector>

int main()
{
    std::vector<std::tuple<std::string, int, double>> times;
    times.push_back(std::make_tuple("Olve", 1971, 12.43));
    times.push_back(std::make_tuple("Lars Gullik", 1972, 11.22));
    for (auto t : times)
        std::cout << std::get<0>(t) << std::endl;

    std::string name;
    int year;
    double time;
    std::tie(name, year, time) = times[1];
    std::cout << name << " (" << year << ") : " << time << std::endl;
}
```

```
Olve
Lars Gullik
Lars Gullik (1972) : 11.22
```

variadic templates

```
#include <iostream>

void my_printf(const char * s)
{
    while (*s)
        std::cout << *s++;
}


template<typename T, typename... Args>
void my_printf(const char * s, const T & value, const Args & ... args)
{
    while (*s) {
        if (*s == '*') {
            std::cout << value;
            my_printf(++s, args...);
            return;
        }
        std::cout << *s++;
    }
}

int main()
{
    std::string s("Hello");
    my_printf("*: * scalar *!\n", s, 42, 3.14);
}
```

variadic templates

```
#include <iostream>

void my_printf(const char * s)
{
    while (*s)
        std::cout << *s++;
}


template<typename T, typename... Args>
void my_printf(const char * s, const T & value, const Args & ... args)
{
    while (*s) {
        if (*s == '*') {
            std::cout << value;
            my_printf(++s, args...);
            return;
        }
        std::cout << *s++;
    }
}

int main()
{
    std::string s("Hello");
    my_printf("*: * scalar *!\n", s, 42, 3.14);
}
```

```
Hello: 42 scalar 3.14!
```

lambda

```
#include <iostream>
#include <functional>

void call_func(const std::function<void(int)> & g)
{
    g(42);
}

std::function<std::string(void)> get_func()
{
    return []() { return "Hello"; };
}

int main()
{
    call_func( [](int i){ if (i == 42) std::cout << "OK" << std::endl; } );
    auto p = [](int i) {return i % 3 == 0 && i % 5 == 0; };
    if ( p(15) )
        std::cout << "FizzBuzz" << std::endl;
    auto f = get_func();
    std::cout << f() << std::endl;
}
```

lambda

```cpp
#include <iostream>
#include <functional>

void call_func(const std::function<void(int)> & g)
{
    g(42);
}

std::function<std::string(void)> get_func()
{
    return []() { return "Hello"; };
}

int main()
{
    call_func( [](int i){ if (i == 42) std::cout << "OK" << std::endl; } );
    auto p = [](int i) {return i % 3 == 0 && i % 5 == 0; };
    if ( p(15) )
        std::cout << "FizzBuzz" << std::endl;
    auto f = get_func();
    std::cout << f() << std::endl;
}
```

```
OK
FizzBuzz
Hello
```

thread and mutex

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <mutex>

static std::mutex m;

void print(const std::string & s)
{
    //std::lock_guard<std::mutex> lock(m);
    for (char c : s)
        std::cout.put(c);
    std::cout << std::endl;
}

int main()
{
    auto f1 = std::thread (print, "Hello from a first thread");
    auto f2 = std::thread (print, "Hello from a second thread");
    print("Hello from the main thread");
    f2.join();
    f1.join();
}
```

thread and mutex

```cpp
#include <iostream>
#include <string>
#include <thread>
#include <mutex>

static std::mutex m;

void print(const std::string & s)
{
    //std::lock_guard<std::mutex> lock(m);
    for (char c : s)
        std::cout.put(c);
    std::cout << std::endl;
}

int main()
{
    auto f1 = std::thread (print, "Hello from a first thread");
    auto f2 = std::thread (print, "Hello from a second thread");
    print("Hello from the main thread");
    f2.join();
    f1.join();
}
```

```
HHeHelellllolo o f frfroromom m t aha e f simeracsiotnn  dtt
hhtrrheeraaedda

d
```

user defined literals

```
#include <iostream>

class FontSize
{
public:
    explicit FontSize(double s) : size_(s) {}
    explicit operator double () const { return size_; }
    double size() const { return size_; }
private:
    double size_;
};

auto operator "" _pt (long double fs) -> decltype(FontSize(fs))
{
    return FontSize(fs);
}

int main()
{
    //FontSize fs1 = 5.0;                  // error
    FontSize fs2 = FontSize(5.2);          // ok
    FontSize fs3 = 5.5_pt;                 // ok
    //std::cout << fs2 << std::endl;       // error
    std::cout << double(fs3) << std::endl; // ok
    std::cout << fs3.size() << std::endl;  // ok
}
```

user defined literals

```cpp
#include <iostream>

class FontSize
{
public:
    explicit FontSize(double s) : size_(s) {}
    explicit operator double () const { return size_; }
    double size() const { return size_; }
private:
    double size_;
};

auto operator "" _pt (long double fs) -> decltype(FontSize(fs))
{
    return FontSize(fs);
}

int main()
{
    //FontSize fs1 = 5.0;                   // error
    FontSize fs2 = FontSize(5.2);           // ok
    FontSize fs3 = 5.5_pt;                  // ok
    //std::cout << fs2 << std::endl;        // error
    std::cout << double(fs3) << std::endl; // ok
    std::cout << fs3.size() << std::endl;  // ok
}
```

```
5.5
5.5
```

chrono / decltype

```
#include <iostream>
#include <chrono>
#include <ctime>
#include <iomanip>

std::ostream & operator<<(std::ostream & out,
                          std::chrono::system_clock::time_point & tp)
{
    std::time_t t = std::chrono::system_clock::to_time_t(tp);
    out << std::ctime(&t);
}

auto get_now() -> std::chrono::system_clock::time_point
{
    return std::chrono::system_clock::now();
}

void foo(const decltype(get_now()) & tp)
{
    std::time_t t = std::chrono::system_clock::to_time_t(tp);
    std::cout << std::ctime(&t);
}

int main()
{
    auto now = std::chrono::system_clock::now();
    std::cout << now;
    auto then = now + std::chrono::hours(4*24*365);
    std::cout << then;
    foo(get_now());
}
```

chrono / decltype

```
#include <iostream>
#include <chrono>
#include <ctime>
#include <iomanip>

std::ostream & operator<<(std::ostream & out,
                          std::chrono::system_clock::time_point & tp)
{
    std::time_t t = std::chrono::system_clock::to_time_t(tp);
    out << std::ctime(&t);
}

auto get_now() -> std::chrono::system_clock::time_point
{
    return std::chrono::system_clock::now();
}

void foo(const decltype(get_now()) & tp)
{
    std::time_t t = std::chrono::system_clock::to_time_t(tp);
    std::cout << std::ctime(&t);
}

int main()
{
    auto now = std::chrono::system_clock::now();
    std::cout << now;
    auto then = now + std::chrono::hours(4*24*365);
    std::cout << then;
    foo(get_now());
}
```

```
Thu May 31 09:55:07 2012
Mon May 30 09:55:07 2016
Thu May 31 09:55:07 2012
```

http://en.wikipedia.org/wiki/C++11
http://www.open-std.org/jtc1/sc22/wg21/
http://en.cppreference.com/w/