

C++ Day 2016
29 Ottobre / Firenze

C++17 per tutti i giorni

Marco Arena – Italian C++ Community



www.italiancpp.org





Bartosz Milewski

@BartoszMilewski



Following

It looks like C++17 will be a big disappointment. Jacksonville committee could not agree on any major features. Oh, well -- back to Haskell.

Visualizza traduzione

RETWEET

64

MI PIACE

70



Josh Brandoff

@jbrandoff



Segui

Hitler expresses his disappointment with C++17 [youtube.com/watch?v=ND-TuW...](https://www.youtube.com/watch?v=ND-TuW...)
Probably not fun to have at Meetups...

Visualizza traduzione



Hitler on C++17

Hitler gets to know about the outcome of the C++ Standards Meeting in Lenexa, May 2015

[youtube.com](https://www.youtube.com)



reddit

CPP

commenti



This is an archived post. You won't be able to vote or comment.



126



Why I am not happy with C++17 (C++ 17 outlook March 2016 vs April 2015) (self.cpp)

inviato 6 mesi fa * da jbandela

The response to the Jacksonville trip reports has been disappointment. Some people have suggested that this is due to too high expectations about C++17, and that we should all be happy. I decided to take a look at the reason for my expectations of C++, and came across this paper from Bjarne Stroustrup from April 27, 2015

<https://isocpp.org/files/papers/D4492.pdf>



C++17

Rispetto al C++11, il C++17 ha molte più features «tiny» e alla portata di tutti, per il nostro lavoro quotidiano.

Quali features possiamo già provare?



The screenshot shows the Wandbox web interface. On the left, there is a configuration panel for C++ with the following options:

- clang HEAD 4.0.0 (trunk) [dropdown]
- Warnings
- Optimization
- Verbose
- Boost 1.61.0 [dropdown]
- Sprout
- MessagePack
- C++1z(GNU) [dropdown]
- no pedantic [dropdown]

On the right, there is a code editor with the following C++ code:

```
1 #include <map>
2 #include <iostream>
3 #include <string_view>
4
5 using namespace std;
6
7 int main()
8 {
9     map<string, int> mm;
10    mm["clang"] = 10;
11    mm["gcc"] = 20;
12    for (auto& [name, value] : mm)
13        cout << name << " " << value << endl;
14 }
```

<http://melpom.org/wandbox>

string*

Can we do better?

```
vector<string> split(const string& str, const char* delims)
{
    vector<string> ret;
    string::size_type start = 0;
    auto pos = str.find_first_of(delims, start);
    while (pos != string::npos) {
        if (pos != start)
            ret.push_back(str.substr(start, pos - start));
        start = pos + 1;
        pos = str.find_first_of(delims, start);
    }
    if (start < str.length())
        ret.push_back(str.substr(start, str.length() - start));

    return ret;
}
```

string_view

```
vector<string_view> split(string_view str, string_view delims)
{
    vector<string_view> ret;
    string::size_type start = 0;
    auto pos = str.find_first_of(delims, start);
    while (pos != string::npos) {
        if (pos != start)
            ret.push_back(str.substr(start, pos - start));
        start = pos + 1;
        pos = str.find_first_of(delims, start);
    }
    if (start < str.length())
        ret.push_back(str.substr(start, str.length() - start));

    return ret;
}
```


string_view

```
StringContainer form_name = form.GetField("name"); // some UI control
string_view nameView = form_name.cptr(); // .cptr() returns a const char*

auto trimmed = nameView.substr(nameView.find_first_not_of(' ')); // trim

// we have map<string, Profile, less<>> nameToProfile, somewhere
auto profileIt = nameToProfile.find(trimmed); // no new string..
```

http://en.cppreference.com/w/cpp/string/basic_string_view

to_chars/from_chars

```
char arr[5] {};  
auto value1 = 10, value2 = 20;  
auto ptrStart = arr; auto ptrEnd = arr + 5;  
auto res = to_chars(ptrStart, ptrEnd, value1);  
if (!res.overflow) // fitted the buffer  
    res = to_chars(res.ptr, ptrEnd, value2);  
// ['1', '0', '2', '0', \0]  
  
string_view sv {arr, arr+2}; // 10  
int value1;  
auto out = from_chars(sv.begin(), sv.end(), value1);  
if (out.ec) {} // if parse error
```

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0067r4.html>

Reading strings from C-arrays...

```
void ExternalAPI(char* out, int size);
```

```
char tmp[1024] {}; // all \0
```

```
ExternalAPI(tmp, 1024);
```

```
string s(tmp); // uff
```

non-const `string::data()`

```
void ExternalAPI(char* out, int size);
```

```
string s(1024, '\0'); // ok  
ExternalAPI(s.data(), s.size());
```

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0272r1.html>

Filesystem

Filesystem (aka: Boost.Filesystem)

```
path p = current_path();
directory_iterator it{p};
while (it != directory_iterator{})
    std::cout << *it++ << '\n';
```

```
path p{"cppday16.txt"};
std::cout << p.stem() << '\n';
std::cout << p.extension() << '\n';
```

```
path p{"C:\\"};
p /= "Windows\\System";
std::cout << p.string() << '\n';
```

<http://en.cppreference.com/w/cpp/filesystem>

Associative Containers Additions

Efficient update or insert?

```
string& update_or_insert(int key, string&& s)
{
    return mm[key] = move(s);
}
```

(a parte possibili svantaggi di performance)
E se `key/value` non sono default constructible?

More efficient update or insert

```
string& update_or_insert(int key, string&& s)
{
    auto p = cache.equal_range(key);
    if (p.first != p.second)
        return it.first->second = move(s);
    return cache.emplace_hint(p.first, key, move(s))->second;
}
```

Funziona, ma non è il massimo su `unordered_map...`

insert_or_assign

```
string& update_or_insert(int key, string&& s)
{
    return mm.insert_or_assign(key, move(s)).first->second;
}
```

Massima efficienza anche su unordered_map

http://en.cppreference.com/w/cpp/container/map/insert_or_assign

emplace

```
std::map<std::string, std::unique_ptr<Foo>> m;
```

```
m["foo"] = nullptr;
```

```
auto ptr = std::make_unique_ptr<Foo>;
```

```
auto res = m.emplace("foo", std::move(ptr));
```

```
assert(ptr); // ?
```

try_emplace

```
std::map<std::string, std::unique_ptr<Foo>> m;
```

```
m["foo"] = nullptr;
```

```
auto ptr = std::make_unique<Foo>();
```

```
auto res = m.try_emplace("foo", std::move(ptr));
```

```
assert(ptr); // never fires
```

try_emplace

```
template< class... Args >
std::pair<iterator, bool> emplace(Args&&... args);
template <class... Args>
pair<iterator, bool> try_emplace(const key_type& k, Args&&... args);

map<string, string> mm;

mm.emplace(piecewise_construct, forward_as_tuple("pippo"), forward_as_tuple(10, 'c'));

mm.try_emplace("pippo", 10, 'c');
```

http://en.cppreference.com/w/cpp/container/map/try_emplace
<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4279.html>

map update key de' noantri

```
map<string, string> languageToExample;
```

```
...
```

```
auto toRemove = languageToExample.find("objective-c");
```

```
languageToExample.emplace("swift", move(toRemove->second));
```

```
languageToExample.erase(toRemove);
```

extract

```
map<string, string> languageToExample;
```

```
...
```

```
auto toUpdate = languageToExample.extract("objective-c");
```

```
toUpdate.key() = "swift";
```

```
languageToExample.insert(move(toUpdate)); // move back
```

extract

```
set<unique_ptr<MyType>> elems;
```

```
...
```

```
auto handle = elems.extract(elems.begin());
```

```
auto ptr = move(handle.value()); // moved out
```

<http://en.cppreference.com/w/cpp/container/map/extract>

merge

```
map<string, string> contactsPhoneMarcoOld;  
map<string, string> contactsPhoneMarcoNew;
```

```
// conflicts are not extracted from contactsPhoneMarcoOld  
contactsPhoneMarcoNew.merge(move(contactsPhoneMarcoOld));
```

```
// possible as well (IMHO: less explicit)  
contactsPhoneMarcoNew.merge(contactsPhoneMarcoOld);
```

<http://en.cppreference.com/w/cpp/container/map/merge>

Zucchero sintattico (da diabete)

Copy elision garantita *

```
vector<int> CreateVector()  
{  
    vector<int> v;  
    // complex logic  
    return v; // won't be copied/moved  
}
```

(*) Sotto certe condizioni: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0135r1.html>

Structured bindings

```
tuple<double, double, int> CreateParams() { ... }
```

```
// x, y, dim are independent variables
```

```
auto [x, y, dim] = CreateParams();
```

```
auto [x, y, std::ignore] = CreateParams(); // not proposed
```

Tuple Protocol: tuple_size, tuple_element, get

Structured bindings

```
array<int, 3> arr{{1,2,3}};  
auto [x, y, z] = arr;  
auto [x, y] = arr;
```

```
Foo f;  
auto& [a, b] = f;
```

```
struct Foo {  
    int i=10;  
    string hello = "hello";  
};
```

Structured bindings - Beautiful iteration

```
map<string, int> mm;
```

```
for (auto& p : mm)  
    cout << p.first << ", " << p.second << "\n";
```

```
for (auto& [name, value]: mm)  
    cout << name << ", " << value << "\n";
```

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0217r2.html>

if/switch statements with initializer

```
map<string, int> mm;
```

```
...
```

```
if (auto [iter, succeeded] = mm.insert({"ciao", 30}); succeeded)  
    cout << iter->second;
```

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0305r0.html>

Template deduction: functions vs classes

```
template<typename Func>
class LambdaVisitor : SomeVisitor {
    Func visitFn; lock_guard<mutex> sg;
public:
    LambdaVisitor(Func f) : visitFn{f} {}
    void Visit(const Node& node) { visitFn(node); }
    // ...
};

template<typename Func> LambdaVisitor<Func> MakeLambdaVisitor(Func f) {
    return {f};
}

auto visitor = MakeLambdaVisitor([](const Node& n) {...});
```


Template parameters deduction for ctors

```
template<typename Func>
class LambdaVisitor : SomeVisitor {
    Func visitFn; lock_guard<mutex> sg;
public:
    LambdaVisitor(Func f) : visitFn{f} {}
    void Visit(const Node& node) { visitFn(node); }
    // ...
};
```

```
LambdaVisitor visitor { [](const Node& n) {...} };
```

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0091r2.html>

Compila in C++14

```
using Config = tuple<string, int, int>;
```

```
vector<Config> configs =
```

```
{ {"marco", 29, 250 }, {"matteo", 28, 200 }, {"luca", 33, 100} };
```

```
vector<Config> configs = {  
    make_tuple("marco", 29, 250),  
    make_tuple("matteo", 28, 200),  
    make_tuple("luca", 33, 100)  
};
```

pair & tuple constructor explicitness

Solo se uno qualsiasi dei loro tipi ha costruttore esplicito.

```
using Config = tuple<string, int, int>;
```

```
vector<Config> configs = {  
    {"marco", 29, 250 }, {"matteo", 28, 200 },  
    {"luca", 33, 100}  
};
```

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4387>

Parallel STL

Parallel STL

```
vector<int> nums = ...
```

```
auto evens = count(begin(nums), end(nums), 23);
```

```
auto evens = count(std::par, begin(nums), end(nums), 23);
```

Parallel STL

<code>std::seq</code>	Esecuzione sequenziale, su un solo thread
<code>std::par</code>	Esecuzione su 1 o più thread (sequenziale su stesso thread)
<code>std::par_unseq</code>	Esecuzione potenzialmente su più thread, vettorizzabile

Parallel STL

```
auto norm =  
    sqrt(  
        transform_reduce(par_unseq,  
                          begin(x), end(x),    // range  
                          multiplies<>{},      // transformation (map)  
                          0.0,                 // init value  
                          plus<>(),           // reduction  
        )  
    );
```

Non hanno overload parallelo

make_heap push_heap pop_heap sort_heap

is_permutation next_permutation prev_permutation

lower_bound upper_bound equal_range binary_search

accumulate partial_sum iota

copy_backward

<http://en.cppreference.com/w/cpp/algorithm>

<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4507.pdf>

Library Fundamentals

optional<T>

```
class Message {
    optional<int> size;
    optional<Data> payload;
    // ...
};

Message mex = Receive(...);
if (mex.size.has_value() && mex.payload.has_value())
{
    // use *mex.size & *mex.payload
}
```

<http://en.cppreference.com/w/cpp/utility/optional>

any / safe void*

```
vector<any> fruits;  
fruits.emplace_back(in_place<Apple>, "melinda");  
fruits.emplace_back(in_place<Banana>, "Del Monte", 5);  
  
auto& apple = any_cast<Apple&>(fruits[0]); // good or throws std::bad_any_cast  
auto banana = any_cast<Banana*>(fruits[1]); // good or nullptr
```

<http://en.cppreference.com/w/cpp/utility/any>

variant<Args...> / safe union

```
using ConfigEntry = variant<string, int, double, vector<double>>;
```

```
map<string, ConfigEntry> programConfig;
```

```
programConfig["Company"] = "Sole";
```

```
programConfig["Version"] = 15;
```

```
programConfig["Coeffs"] = {1.0, 2.0};
```

```
cout << paramConfig["Version"].index(); // 1 (in boost è which())
```

```
cout << paramConfig["Company"].get<string>();
```

<http://en.cppreference.com/w/cpp/utility/variant>

Generic Programming

Nested namespace declarations

```
namespace std::experimental
{
    // ...
}
```

apply & invoke

```
void print(const string& arg, int i) { ... }
```

```
auto args = tuple{"Welcome to C++ Day"s, 2016};
```

```
apply(print, args);
```

```
array<int, 4> coefficients { {1, 2, 3, 4} }; // satisfies "tuple protocol"
```

```
cout << apply(func, coefficients);
```

```
cout << invoke(func, 1, 2, 3, 4);
```

```
// do not work on overloads
```

make_from_tuple

```
static const tuple<int> tinyConfig = ...;
static const tuple<int, double> mediumConfig = ...;
static const tuple<int, string, double, bool> completeConfig = ...;
```

```
Game CreateGame(int config) {
    switch(config)
    case 1:
        return make_from_tuple<Game>(tinyConfig);
    case 2:
        return make_from_tuple<Game>(mediumConfig);
    case 3:
        return make_from_tuple<Game>(completeConfig);
}
```


if

```
template<size_t Dim>
struct Vector {
    float v[Dim];
    float operator[](size_t idx) { return v[idx]; }

    auto CrossProduct(const Vector& other) {
        if (Dim==2) {
            return v[0]*other[1] - v[1]*other[0];
        }
        if (Dim==3) {
            return Vector<3>{v[1]*other[2]-v[2]*other[1], ...}
        }
    }
};
```

if constexpr

```
template<size_t Dim>
struct Vector {
    float v[Dim];
    float operator[](size_t idx) { return v[idx]; }

    auto CrossProduct(const Vector& other) {
        if constexpr(Dim==2) {
            return v[0]*other[1] - v[1]*other[0];
        }
        if constexpr(Dim==3) {
            return Vector<3>{v[1]*other[2]-v[2]*other[1], ...}
        }
    }
};
```

Il mio C++17 per tutti i giorni

Il mio C++17 per tutti i giorni

- `string_view` & co.
- Filesystem
- Associative containers additions
- Syntactic sugar da diabete & rettifiche
- Parallel STL
- Library fundamentals (`optional`, `any`, `variant`)
- Generic Programming (`apply`, `make_from_tuple`, `constexpr if`)

Che altro c'è?

Elenco dettagliato delle novità del C++17:

<http://stackoverflow.com/questions/38060436/what-are-the-new-features-in-c17>

C++17 in VS "15" Preview:

<https://blogs.msdn.microsoft.com/vcblog/2016/10/11/c1417-features-and-stl-fixes-in-vs-15-preview-5/>

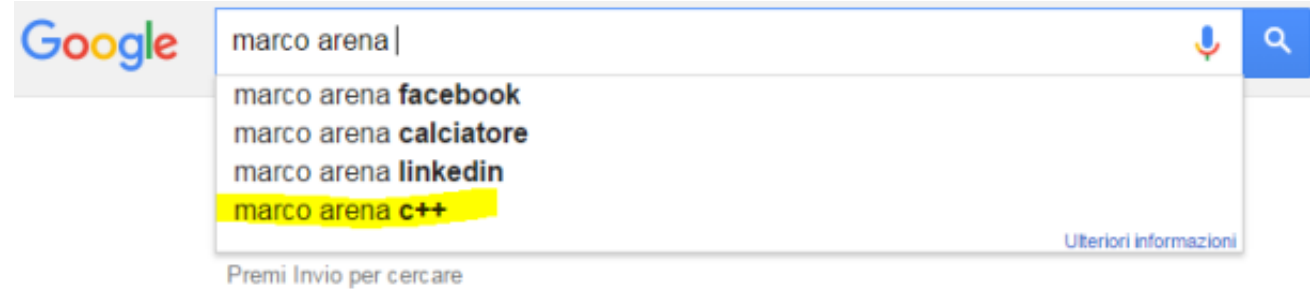
Status di Clang:

http://clang.llvm.org/cxx_status.html

Status di GCC:

<https://gcc.gnu.org/projects/cxx-status.html#cxx11>

Chi sono



marco@italiancpp.org

marcoarena.wordpress.com

Grazie!