

# C++ Club Meeting Notes

---

Gleb Dolgich

2020-01-16

# C++ std::string\_view for better performance: An example use case

Article

Reddit

Arthur O'Dwyer: std::string\_view is a borrow type

*Borrow types are essentially “borrowed” references to existing objects. They lack ownership; they are short-lived; they generally can do without an assignment operator. They generally appear only in function parameter lists; because they lack ownership semantics, they generally cannot be stored in data structures or returned safely from functions.*

cppreference: std::basic\_string\_view (C++17)

## A hidden gem: inner\_product

- Article

# A hidden gem: inner\_product

4



Conor Hoekstra @code\_report

↳ @cjdb\_ns & @TartanLlama

This makes me so incredibly happy! I literally just yesterday googled, C++17 / C++20 zip to see if they had anything, because I wrote some code in both C++ and #Python and Python was so much more beautiful.

```
int solve(int h, vector<int> w, vector<int> l) {
    int p = 0;
    for (int i = 0; i < w.size(); ++i)
        p = max(p, w[i] - l[i] / 4);
    return max(0, p - h);
}

def solve(h, w, l):
    p = max(a - b//4 for a, b in zip(w, l))
    return max(0, p - h)
```

29w • 03/12/2018 • 17:47



Conor Hoekstra @code\_report

↳ @cjdb\_ns & @TartanLlama

Also, I just discovered std::inner\_product - a beautiful temporary solution to a lack of zip.

#cpp #inner\_product

```
int solve(int h, vector<int> w, vector<int> l) {
    return max(0, inner_product(begin(w), end(w), begin(l), 0,
        [](){auto a, auto b { return max(a, b); },
        [](){auto a, auto b { return a - b / 4; }}) - h);
}
```

27w • 16/12/2018 • 09:30



## Lingo 0.1.0: Text encoding for modern C++

- GitHub: C++11, MIT
- Reddit

# Argumentum

*Argumentum is a C++17 library for writing command-line program interfaces, inspired by Python argparse*

- GitHub: C++17, MPL
- Reddit

## Modules are Coming – Bryce Adelstein Lelbach

- YouTube
- Reddit

*Modules will have a greater impact than any other feature added post-c++98.*

# Modules are Coming – Bryce Adelstein Lelbach

## Textual Inclusion

```
math.hpp  
#pragma once  
  
int square(int a);
```

```
math.cpp  
#include "math.hpp"  
  
int square(int a) { return a * a; }
```

```
main.cpp  
#include "math.hpp"  
  
int main() { return square(42); }
```

## Modular Import

```
math.ixx  
export module math;  
  
export int square(int a);
```

```
math.mxx  
module math;  
  
int square(int a) { return a * a; }
```

```
main.cpp  
import math;  
  
int main() { return square(42); }
```

Copyright (C) 2019 Bryce Adelstein Lelbach

6



Core C++  
2019



## Modules are Coming – Bryce Adelstein Lelbach

*“III-formed, no diagnostics required” (IFNDR)*

# Modules are Coming – Bryce Adelstein Lelbach

```
tree_node.hpp
#pragma once

template <typename T>
struct tree_node {
    T value;
    std::vector<tree_node*> children;
#ifdef DEBUG
    tree_node* parent;
#endif
};
```

a.cpp

```
#define DEBUG
#include "tree_node.hpp"

// ...
```

b.cpp

```
#include "tree_node.hpp"

// ...
```

Copyright (C) 2019 Bryce Adelstein Lelbach

37



# Core C++ 2019



# Modules are Coming – Bryce Adelstein Lelbach

## Textual inclusion

```
1 #include <foo.hpp>
2 #include "foo.hpp"
```

## Modular import

```
1 import foo;
2 import <foo.hpp>;
3 import "foo.hpp";
```

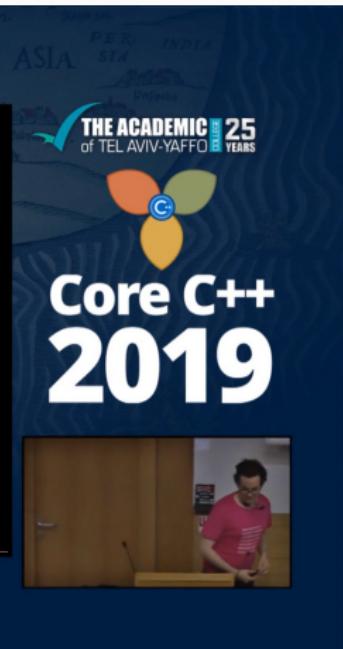
# Modules are Coming – Bryce Adelstein Lelbach

## Importable headers:

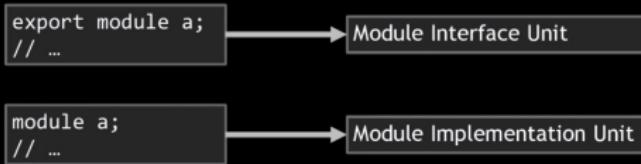
- Most C++ standard library headers\*.
- Some system headers.
- Headers you proclaim importable†.

\* C standard library headers (<cfoo>, <foo.h>) are not required to be importable.

† The mechanism for indicating which headers are importable is implementation defined.

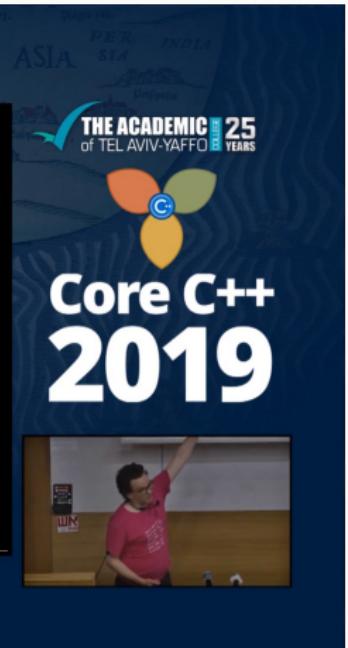


# Modules are Coming – Bryce Adelstein Lelbach



Copyright (C) 2019 Bryce Adelstein Lelbach

74



# Modules are Coming – Bryce Adelstein Lelbach

Only one module declaration per translation unit:

## Interface

```
1 export module a;  
2 //...  
3 export module b; // Error
```

## Implementation

```
1 module a;  
2 //...  
3 module b; // Error
```

# Modules are Coming – Bryce Adelstein Lelbach

## Module unit structure

```
1 export module ...;  
2 import ...;  
3 ...
```

## Modules are Coming – Bryce Adelstein Lelbach

```
1 export declaration ...
2 export {
3     declaration ...
4 }
```

## Modules are Coming – Bryce Adelstein Lelbach

```
1 export {  
2     void f();  
3     struct A;  
4     int i{0};  
5 }
```

```
1 export void f();  
2 export struct A;  
3 export int i{0};
```

## Modules are Coming – Bryce Adelstein Lelbach

```
1 export template <typename T>
2 T square(T t) { return t*t; }
3
4 export template <typename T>
5 struct is_const : false_type {};
6
7 export template <typename T>
8 struct is_const<T const> : true_type {};
```

## Modules are Coming – Bryce Adelstein Lelbach

```
1 export namespace foo { struct A; } // foo::A exported
2
3 namespace foo { struct B; } // foo::B not exported
```

## Modules are Coming – Bryce Adelstein Lelbach

```
1 export typedef int int32_t;  
2  
3 export using unsigned uint32_t;
```

## Modules are Coming – Bryce Adelstein Lelbach

```
1 | export import a;
```

# Modules are Coming – Bryce Adelstein Lelbach

```
a.ixx
export module a;

struct S { int m; };
export S foo();
```

```
main.cpp
import a;

int main() {
    auto s0 = foo();
    s0.m = 42;
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

90



**Core C++  
2019**



# Modules are Coming – Bryce Adelstein Lelbach

```
a.ixx
export module a;

struct S { int m; };
export S foo();
```

```
main.cpp
import a;

int main() {
    auto s0 = foo();
    s0.m = 42;

    S s1{};
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

91



Core C++  
**2019**



# Modules are Coming – Bryce Adelstein Lelbach



a.ixx

```
export module a;
struct S { int m; };
export S foo();
```

main.cpp

```
import a;
int main() {
    auto s0 = foo();
    s0.m = 42;
    decltype(foo()) s1{};
}
```

Copyright (C) 2019 Bryce Adelstein Lelbach

92

## Modules are Coming – Bryce Adelstein Lelbach

- *Visible*: in scope, can be named
- *Reachable*: in scope, not necessarily namable

# Modules are Coming – Bryce Adelstein Lelbach

## Kinds of Linkage

	Example	Visible From	Notes
External Linkage	<pre>extern void foo(); export void bar(); extern int i{}; export bool b{};</pre>	Other translation units.	
Module Linkage	<pre>struct S; int foo(); int i{};</pre>	This module.	In non-modular units, entities with module linkage have external linkage.
Internal Linkage	<pre>static void foo(); static int i{}; bool const b{}; namespace { /* ... */ }</pre>	This translation unit.	
No Linkage	<pre>int main() {     int i{}; }</pre>	This scope.	

Copyright (C) 2019 Bryce Adelstein Lelbach

109



# Modules are Coming – Bryce Adelstein Lelbach

```
a.ixx
export module a;
#if defined(DEBUG)
// ...
#else
// ...
#endif
```

```
main.cpp
#define DEBUG
import a;
```

The definition of DEBUG isn't seen by the imported module.



# Modules are Coming – Bryce Adelstein Lelbach

## Module unit structure

```
1 module;  
2 #pp-directive ...;  
3 export module ...;  
4 import ...;  
5 ...  
6 module : private;  
7 ...
```

# Modules are Coming – Bryce Adelstein Lelbach

## Textual Inclusion

```
a.hpp  
#pragma once  
  
struct pimpl;  
  
a.cpp  
#include "a.hpp"  
  
struct pimpl { /* ... */ };
```

## Modular Import

```
a.ixx  
export module a;  
  
export struct pimpl;  
  
module : private;  
  
struct pimpl { /* ... */ };
```

Copyright (C) 2019 Bryce Adelstein Lelbach

152



**Core C++  
2019**



# Modules are Coming – Bryce Adelstein Lelbach

```
square.ixx
export module square;

export template <typename T>
T square(T a) { return a * a; }
```

```
add.ixx
export module add;

export template <typename T>
T add(T a, T b) { return a + b; }
```

```
math.ixx
export module math;

export import square;
export import add;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

155



# Modules are Coming – Bryce Adelstein Lelbach

```
square.ixx
export module math:square;

export template <typename T>
T square(T a) { return a * a; }
```

```
add.ixx
export module math:add;

export template <typename T>
T add(T a, T b) { return a + b; }
```

```
math.ixx
export module math;

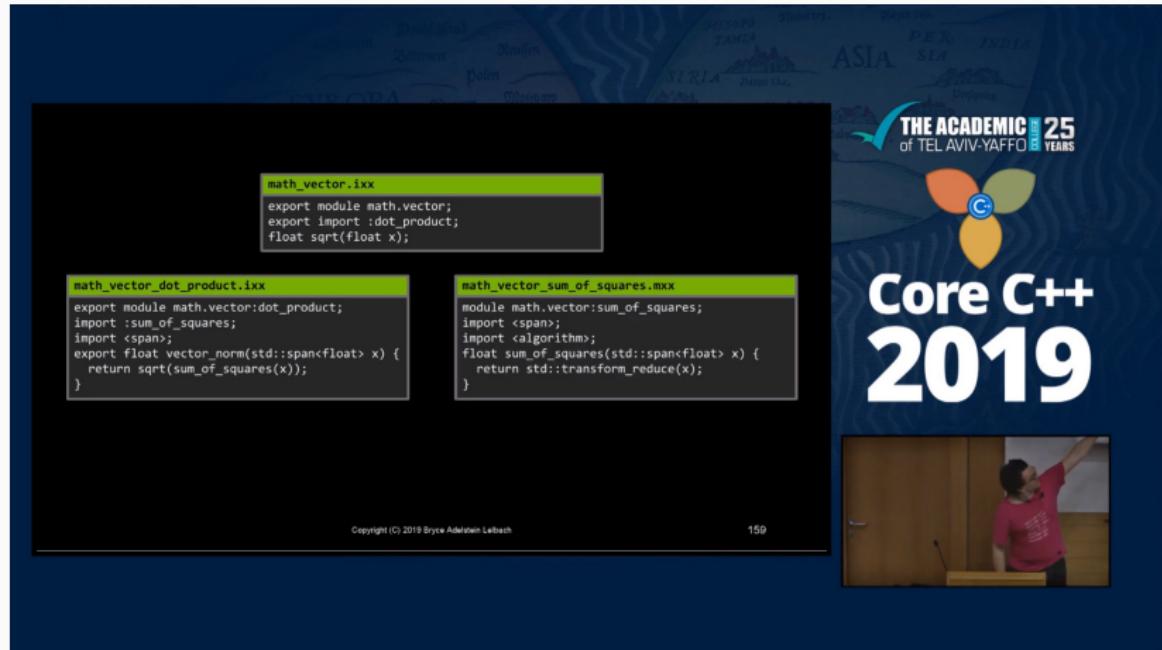
export import :add;
export import :square;
```

Copyright (C) 2019 Bryce Adelstein Lelbach

156



# Modules are Coming – Bryce Adelstein Lelbach



The background of the slide features a detailed map of the Asian continent and parts of the Middle East, with labels for countries like India, China, Japan, and others.

**THE ACADEMIC 25  
of TEL AVIV-YAFFO YEARS**

**Core C++  
2019**

A small video frame in the bottom right corner shows a man in a red t-shirt standing at a podium, gesturing with his right arm raised.

```
math_vector.ixx
export module math.vector;
export import :dot_product;
float sqrt(float x);
```

```
math_vector_dot_product.mxx
export module math.vector:dot_product;
import :sum_of_squares;
import <span>;
export float vector_norm(std::span<float> x) {
    return sqrt(sum_of_squares(x));
}
```

```
math_vector_sum_of_squares.mxx
module math.vector:sum_of_squares;
import <span>;
import <algorithm>;
float sum_of_squares(std::span<float> x) {
    return std::transform_reduce(x,
```

Copyright (C) 2019 Bryce Adelstein Lelbach

159

# Modules are Coming – Bryce Adelstein Lelbach

## Kinds of Translation Units

	Example	Extension	Artifact	Notes
Non-Modular Unit	#include "..." ...	.cpp	.o	
Header Unit	// Created by: import <...>;	.hpp	.cmi .o (optional)	
Module Interface Unit	export module ...; ...	.ixx	.cmi .o (optional)	Exactly one per module.
Module Implementation Unit	module ...; ...	.mxs	.o	At most one per module.
Module Partition Interface Unit	export module ...:...; ...	.ixx	.cmi .o (optional)	
Module Partition Implementation Unit	module ...:...; ...	.mxs	.o	

Copyright (C) 2019 Bryce Adelstein Lelbach

161



# Modules are Coming – Bryce Adelstein Lelbach

## How are modules found?

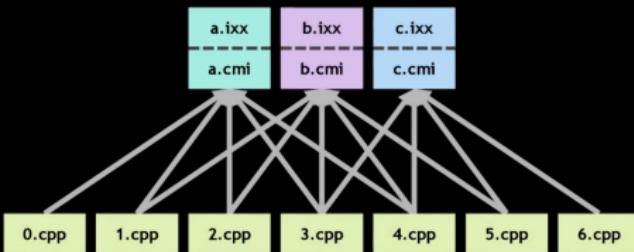
- Not specified by the standard.
- Unlike headers, modules are programmatically named.
- A file name <-> module name mapping is not straightforward.
  - Modules have to be precompiled.
  - Partitions span multiple files.

```
bar.ixx
export module foo;
// ...
```



# Modules are Coming – Bryce Adelstein Lelbach

Implicit Precompilation is Problematic

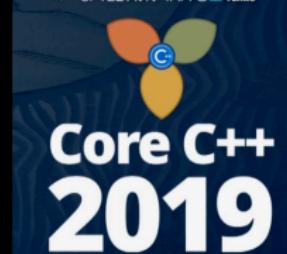


If precompilation is implicit and builds are parallel,  
how do we decide who builds the CMIs?

Copyright (C) 2019 Bryce Adelstein Lelbach

173

THE ACADEMIC 25  
of TEL AVIV-YAFFO YEARS



## Modules are Coming – Bryce Adelstein Lelbach

- Tools can no longer rely on simple lookup mechanism (include directories and header file names) to understand C++ projects.
- Dependency scanning now requires a C++ parser, not just a C preprocessor.