
accu
2021
VIRTUAL EVENT
Bloomberg
Engineering
undo
mosaic
CONSULTANTS TO FINANCIAL SERVICES
A Practical Introduction to C++20's Modules
Hendrik Niemeyer

A Practical Introduction to C++20's Modules

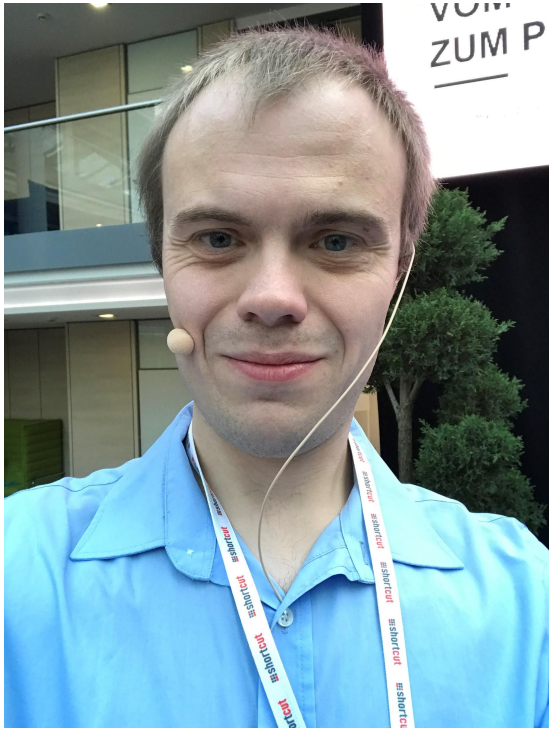
Hendrik Niemeyer (he/him)

Link to Slides:

https://github.com/hniemeyer/ModulesTalk

Feedback and Questions

- Twitter: @hniemeye
- LinkedIn: hniemeyer87
- Xing: Hendrik_Niemeyer
- GitHub: hniemeyer



Things I like: C++, Rust, Docker

Modular Programming

- Modules separate a program into independent and interchangeable units
- Modules provide public interfaces
- Modules hide the actual implementation and possibly data and functionality (information hiding)
- A module is not a package

Modular Programming in C++ So Far

- Source and header files provide some sort of modularization and reuse
- An executable is the sum of its translation units produced by the linker
- The same headers get included and compiled all over again throughout your code
- The order of include directives is important

General Compiler Support

- gcc 11(partial)
- clang 8 (partial)
- MSVC 19.28 (VS2019 16.8)

Build System Support

- MSBuild
- build2
- meson (experimental, only with Visual Studio)

Modules with gcc and build2

- build gcc trunk from source
- install build2 from master branch (latest staged version)

See: https://build2.org/blog/build2-cxx20-modules-gcc.xhtml

CMake?

- no official support for C++ modules yet
- only projects on GitHub adding experimental support
- problem: CMake needs to look into files for C++ module support
- My opinion: Do not use

Let’s write some modules

Module Names

- A number of identifiers joined by dots .
- The dot carries no meaning (no submodules)
- Just a possibility to communicate hierarchy
- The name of the module can only be referred to in the module's declaration or in an import declaration

```
export module name.with.dots;
```

Modules and Namespaces

- Unlike other languages (e.g. Rust, Python) a C++ module does not implicitly introduce a new namespace
- Having two exported entities with the same name and signature in two different modules in the global namespace leads to an ill-formed program, no diagnostic required
- Advice: Introduce a namespace with the same name as your module

Module Units

- module interface unit (contains the export keyword in the module declaration)
- module implementation unit (does not have the export keyword in the module declaration)
- module partition
  - module interface partition
  - module implementation partition

Module Partitions

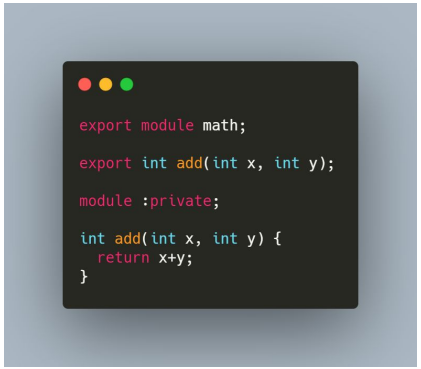
- Possibility to subdivide modules
- cannot be imported separately
- the subdivision is not visible to the user
- all module partitions must be exported in the primary module interface unit

```
//math.ixx
export module math;

export import :modulo;
```

```
//math_modulo.ixx
export module math:modulo;

export int mod(int a, int b) {return a % b;}
```

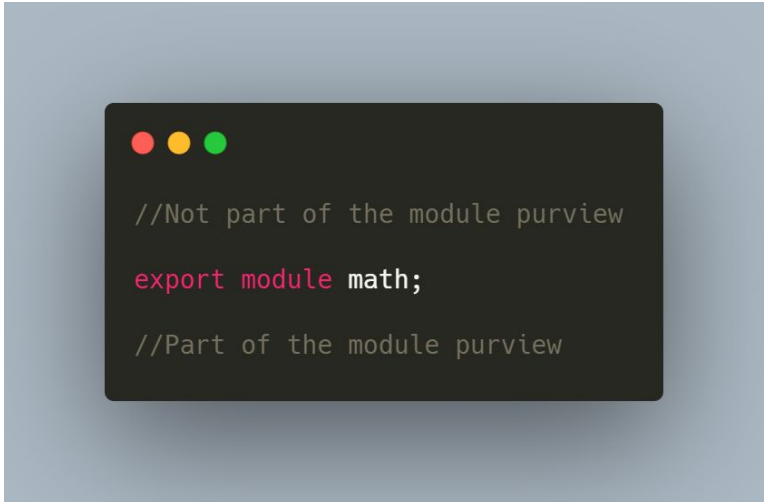
Private Module Fragment

- Keep interface and implementation separate without having multiple files
- must be a single file module
- modification of the private module fragment cannot trigger recompilation of importers of the module
- Possibly faster incremental builds



Module Purview

- Module unit purview: Everything from module declaration to the end of the translation unit
- Module purview: Set of purviews from its module units



Linkage

- internal linkage: inaccessible outside of current translation unit (things inside anonymous namespaces, static things, ...)
- external linkage: Identical between translation units
- module linkage: not internal, not exported and attached to a named module

Global Module

- Everything must be attached to a module
- global module: implicit, unnamed module containing all code not declared in a module
- the global module is the only unnamed module

What Can Be Exported?

- variables, classes, structs, functions, namespaces, template functions/classes, concepts
- but NOT with internal linkage like
  - static variables and functions
  - anything defined within an anonymous namespace
- export declarations must occur on namespace level (e.g. cannot export member variables)
- exporting a namespace implicitly exports everything in in it
- an export block can be declared

Things which cannot be exported

```
namespace {
  export int stuff() {return 0;} //not ok
}

export static double my_pi = 3.14; //not ok

struct Point {
  export int x; // not ok
  int y;
};
```

Thing which can be exported

```
export template <typename T> int add(T x, T y) {return x+y;)

export template<typename T>
concept Addable = requires (T a, T b) { a+b; };

export double pi_is_exactly_three = 3.0;

export struct Point {
  int x;
  int y;
};
```

Implicit Exports

```
export namespace fun {
  int cool_number() {return 1987;} //implicit export
}

export {
  int awesome_number() {return 1988;} //implicit export
}
```

Import

- It is not forbidden to use import (and also export) as a name in your code (please dont)
- In a module unit imports must happen before the first declaration
- In a non-module unit imports may occur after declarations
- imports are only allowed at global scope
- a module cannot import itself
- Cyclic imports are not allowed

Import

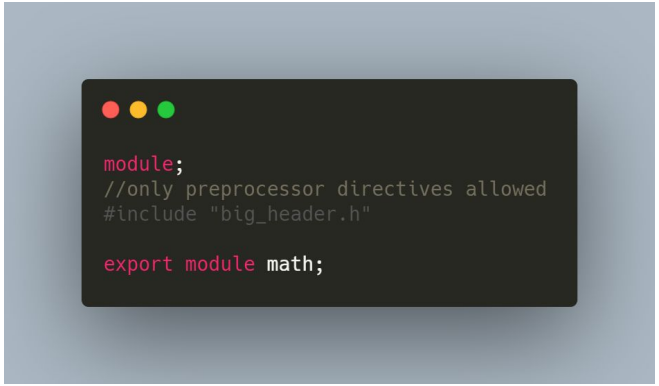
```
export module math;

import algorithms;

import math; //not ok, import itself

namespace fun {
  import fun_stuff; //not ok, import outside of global scope
}

int add(int x, int y) {return x+y;}

import more_algorithms; //not ok, import after declaration
```

include and modules

- including a header in the module purview is not a good idea
- everything in the header will be in the module with module linkage

Global Module Fragment

- Only preprocessor directives allowed
- Things are attached to the global module and not to the named module
- Declarations not used in the named module are discarded and not attached to the named module
- Use for headers which rely preprocessor state from the includer



Header Units

- You can “import” headers
- This does not convert them magically into modules
- Code is treated as if it a was module with everything exported
- macros from the header will be available for the importer
- #define statements in the importer have no effect on the imported header
- Will not work on all headers (rely on preprocessor state is a no go)
- Will work on headers from the standard library

Header Units

```
export module math;
import "big_header.h";
```

Advice for Headers

- Your own header: Try to convert it into a module
- third-party library: Try header units first and if this does not work including in the global module fragment
- Standard Library: Will work as header units (except the C headers)

Advantages

- encapsulation and information hiding
- faster compile times

Disadvantages

- compiler support
- no modularized standard library yet
- third-party libraries not modularized yet

Advice

- Visual Studio users: Use and learn modules with small (hobby-, side-) projects
- all others: Wait for compiler and build system support

More Information

- Modules the beginner's guide - Daniela Engert - Meeting C++ 2019
- Modules are coming - Bryce Adelstein Lelbach - Meeting Cpp 2019
- Understanding C++ Modules: Part 1: Hello Modules, and Module Units
- A Tour of C++ Modules in Visual Studio
- Standard C++20 Modules support with MSVC in Visual Studio 2019 version 16.8

Feedback and Questions

- Twitter: @hniemeye
- LinkedIn: hniemeyer87
- Xing: Hendrik_Niemeyer
- GitHub: hniemeyer