C++20 Modules

Ramses van Zon

May 1, 2024

Outline:

- What is the point of modules?
- How were they implemented before C++20?
- Why is that not good enough for C++?
- How do C++20/23 modules solve that?
- Show-and-tell:
  - Why is this so hard for compilers and build systems?
  - What is needed to make these work CURRENTLY?

What is the point of modules?

What is a module?

What is the point of modules?

What is a module?

A module:

What is the point of modules?

What is a module?

A module:

- implements of a specific functionality,

What is the point of modules?

What is a module?

A module:

- implements of a specific functionality,
- is self-contained,

What is the point of modules?

What is a module?

A module:

- implements of a specific functionality,
- is self-contained,
- and has a well-defined interface to other code.

What is the point of modules?

What is a module?

A module:

- implements of a specific functionality,
- is self-contained,
- and has a well-defined interface to other code.

Advantages

What is the point of modules?

What is a module?

A module:

- implements of a specific functionality,
- is self-contained,
- and has a well-defined interface to other code.

Advantages

- Modular code is more manageable,

What is the point of modules?

What is a module?

A module:

- implements of a specific functionality,
- is self-contained,
- and has a well-defined interface to other code.

Advantages

- Modular code is more manageable,
- Allows separate testing and debugging,

What is the point of modules?

What is a module?

A module:

- implements of a specific functionality,
- is self-contained,
- and has a well-defined interface to other code.

Advantages

- Modular code is more manageable,
- Allows separate testing and debugging,
- Allows for reuse of compiled code

What is the point of modules?

What is a module?

A module:

- implements of a specific functionality,
- is self-contained,
- and has a well-defined interface to other code.

Advantages

- Modular code is more manageable,
- Allows separate testing and debugging,
- Allows for reuse of compiled code
- Allows for parallel compilation
- Reusable in other code,
  (possibly without recompilation, e.g. as a library)

What is the point of modules?

What is a module?

A module:

- implements of a specific functionality,
- is self-contained,
- and has a well-defined interface to other code.

Advantages

- Modular code is more manageable,
- Allows separate testing and debugging,
- Allows for reuse of compiled code
- Allows for parallel compilation
- Reusable in other code,
  (possibly without recompilation, e.g. as a library)
- Encapsulation: other code does not need to know the implementation, and cannot interfere with it.

How were they implemented before C++20?

C++ before C++20 use the good-ol' "header file + source file" approach from C.

Let's look at this briefly.

Implementation + interface

```cpp
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
double fifthroot(double x){
   double result = ...; // computes result;
   return result;
}
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

Implementation + interface

```cpp
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
double fifthroot(double x){
   double result = ...; // computes result;
   return result;
}
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

Compilation:

```
$ $CXX fifthroot.cpp -o fifthroot
$ ./fifthroot
0.4
```

Implementation + interface

```
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
double fifthroot(double x){
   double result = ...; // computes result;
   return result;
}
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

Compilation:

```
$ $CXX fifthroot.cpp -o fifthroot
$ ./fifthroot
0.4
```

(all examples will be from a Unix-like command-line for simplicity, and use $CXX for the compiler).

Implementation + interface

```
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
double fifthroot(double x){
   double result = ...; // computes result;
   return result;
}
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

Compilation:

```
$ $CXX fifthroot.cpp -o fifthroot
$ ./fifthroot
0.4
```

(all examples will be from a Unix-like command-line for simplicity, and use $CXX for the compiler).

Let's reuse the the `fifthroot` function and hide its implementation:

Implementation + interface

```
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
double fifthroot(double x){
   double result = ...; // computes result;
   return result;
}
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

Compilation:

```
$ $CXX fifthroot.cpp -o fifthroot
$ ./fifthroot
0.4
```

(all examples will be from a Unix-like command-line for simplicity, and use $CXX for the compiler).

Let's reuse the the `fifthroot` function and hide its implementation:

```
// fifthroot.h - interface
#ifndef _FIFTH_ROOT_H_
#define _FIFTH_ROOT_H_
double fifthroot(double x);
#endif
```

```
// fifthroot.cpp - implementation
double fifthroot(double x){
   double result = ...;  // computes result;
   return result;
}
```

```
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
#include "fifthroot.h"
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

Implementation + interface

```
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
double fifthroot(double x){
   double result = ...; // computes result;
   return result;
}
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

Compilation:

```
$ $CXX fifthroot.cpp -o fifthroot
$ ./fifthroot
0.4
```

(all examples will be from a Unix-like command-line for simplicity, and use $CXX for the compiler).

Let's reuse the the `fifthroot` function and hide its implementation:

```
// fifthroot.h - interface
#ifndef _FIFTH_ROOT_H_
#define _FIFTH_ROOT_H_
double fifthroot(double x);
#endif
```

```
// fifthroot.cpp - implementation
double fifthroot(double x){
   double result = ...;  // computes result;
   return result;
}
```

```
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
#include "fifthroot.h"
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

```
$ $CXX calcfifthroot.cpp -c -o calcfifthroot.o
$ $CXX fifthroot.cpp -c -o fifthroot.o
$ $CXX calcfifthroot.o fifthroot.o -o calcfifthroot
```

Why not build with CMake?

Because we need to be explicit in this talk to be able to understand what's going on.

Also CMake's C++20 module support lags behind compiler support, which itselfs varies a lot.

Why is that not good enough for C++?

Why is that not good enough for C++?

1. Requires the preprocessor

Single inclusion done portably requires preprocessor macros. Macros are BAD, and much effort has been put in the C++ language to make the preprocessor unnecessary (e.g. `const`, `constexpr`, `template`).

Why is that not good enough for C++?

1. Requires the preprocessor

Single inclusion done portably requires preprocessor macros. Macros are BAD, and much effort has been put in the C++ language to make the preprocessor unnecessary (e.g. `const`, `constexpr`, `template`).

Why is the preprocessor BAD? Because they do not check types, follow their own syntax, are another way to generate code outside of c++; all this makes them error-prone.

Why is that not good enough for C++?

1. Requires the preprocessor

Single inclusion done portably requires preprocessor macros. Macros are BAD, and much effort has been put in the C++ language to make the preprocessor unnecessary (e.g. `const`, `constexpr`, `template`).

Why is the preprocessor BAD? Because they do not check types, follow their own syntax, are another way to generate code outside of c++; all this makes them error-prone.

2. Error prone

It takes discipline and experience to properly separate the interface from the implementation.

Why is that not good enough for C++?

1. Requires the preprocessor

Single inclusion done portably requires preprocessor macros. Macros are BAD, and much effort has been put in the C++ language to make the preprocessor unnecessary (e.g. `const`, `constexpr`, `template`).

Why is the preprocessor BAD? Because they do not check types, follow their own syntax, are another way to generate code outside of c++; all this makes them error-prone.

2. Error prone

It takes discipline and experience to properly separate the interface from the implementation.

3. Templates

Templates do not define classes and functions, but define how to generate these.

Its full definition of the template class must be available in every 'translation unit' that needs that an instantation of it, i.e, in the header files.

This is also poses the issue of multiple definitions of the same classes and functions.

How do C++20/23 modules solve that?

#1. Preprocessor needed? Not any more, no need for header guards.

How do C++20/23 modules solve that?

#1. Preprocessor needed? Not any more, no need for header guards.

#2. Error prone? You explicitly state what goes in the interface with the `export` command.

How do C++20/23 modules solve that?

#1. Preprocessor needed? Not any more, no need for header guards.

#2. Error prone? You explicitly state what goes in the interface with the `export` command.

#3. Templates? You control what is visible, generated code does not get inserted into the client code but stays part of the module 'purvue'. file.

Hello world in C++23

In an ideal world, the prototypical “hello world” program would change

Hello world in C++23

In an ideal world, the prototypical “hello world” program would change

From

```cpp
// helloworld.cpp
#include <iostream>
int main() {
   std::cout << "Hello world!" << std::endl;
}
```

Hello world in C++23

In an ideal world, the prototypical “hello world” program would change

From

```cpp
// helloworld.cpp
#include <iostream>
int main() {
   std::cout << "Hello world!" << std::endl;
}
```

To

```cpp
// helloworld.cpp
import std;
int main() {
   std::println("Hello world!");
}
```

Note that there are no processor lines and all of std is in one module.

Hello world in C++23

In an ideal world, the prototypical “hello world” program would change

From

```
// helloworld.cpp
#include <iostream>
int main() {
   std::cout << "Hello world!" << std::endl;
}
```

To

```
// helloworld.cpp
import std;
int main() {
   std::println("Hello world!");
}
```

Note that there are no processor lines and all of std is in one module.

Compilation & run:

```
$ $CXX helloworld.cpp -o helloworld
$ ./helloworld
Hello world!
```

Hello world in C++23

In an ideal world, the prototypical “hello world” program would change

From

```
// helloworld.cpp
#include <iostream>
int main() {
   std::cout << "Hello world!" << std::endl;
}
```

To

```
// helloworld.cpp
import std;
int main() {
   std::println("Hello world!");
}
```

Compilation & run:

```
$ $CXX helloworld.cpp -o helloworld
$ ./helloworld
Hello world!
```

Note that there are no processor lines and all of std is in one module.

Compilation & run:

```
$ $CXX -std=c++23 helloworld.cpp -o helloworld
$ ./helloworld
Hello world!
```

Modularization in C++20

Before C++20

After C++20

Modularization in C++20

Before C++20

```
// fifthroot.h - interface
#ifndef _FIFTH_ROOT_H_
#define _FIFTH_ROOT_H_
double fifthroot(double x);
#endif
```

```
// fifthroot.cpp - implementation
#include <cmath>
double fifthroot(double x){
   double result = ...;  // computes result;
   return result;
}
```

```
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
#include "fifthroot.h"
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

```
$ $CXX calcfifthroot.cpp -c -o calcfifthroot.o
$ $CXX fifthroot.cpp -c -o fifthroot.o
$ $CXX calcfifthroot.o fifthroot.o -o calcfifthroot
```

After C++20

Modularization in C++20

Before C++20

```
// fifthroot.h - interface
#ifndef _FIFTH_ROOT_H_
#define _FIFTH_ROOT_H_
double fifthroot(double x);
#endif
```

```
// fifthroot.cpp - implementation
#include <cmath>
double fifthroot(double x){
   double result = ...;  // computes result;
   return result;
}
```

```
// calcfifthroot.cpp - computes a fifth root
#include <iostream>
#include "fifthroot.h"
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

```
$ $CXX calcfifthroot.cpp -c -o calcfifthroot.o
$ $CXX fifthroot.cpp -c -o fifthroot.o
$ $CXX calcfifthroot.o fifthroot.o -o calcfifthroot
```

After C++20

```
// fifthroot.cpp
export module fifthroot;
import <cmath>;  // this is a 'header unit'
export double fifthroot(double x){
   double result = ...;  // computes result;
   return result;
}
```

```
// calcfifthroot.cpp
import <iostream>;  // this is a 'header unit'
import fifthroot;
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

```
$ $CXX fifthroot.cpp -flagforamodule
$ $CXX calcfifthroot.cpp -c -o calcfifthroot.o
$ $CXX calcfifthroot.o -o calcfifthroot
```

But much of this is currently unsupported!

Why is this so hard for compilers and build systems?

- It needs a BMI: Binary Module Interface.
- A BMI introduces dependencies.
- It also requires a place to be stored.
- It requires the compiler to destinguish module units and regular files.
- BMIs are compiler, compiler-version, and compiler-flags dependent.
- What to do with the STL?
- How to interface legacy code?

Where do we stand CURRENTLY?

Many tutorials and videos on C++ modules are very optimistic and glance over how to compile these.

We want to be more practical.

We will take our two examples:

helloworld

This involves using the standard library as modules.

calcfifthroot

This one involves creating a module of our own.

Let's see what is involved to make them work with g++12 and clang 18. (I don't have access to MSVC).

Hello world compilation with g++ 12

```
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

Hello world compilation with g++ 12

```
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

```
$ g++ --version
g++ (Gentoo 12.3.1_p20230526 p2) 12.3.1 20230526
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPO
```

Hello world compilation with g++ 12

```
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

```
$ g++ -std=c++23 helloworld.cpp -o helloworld
```

Hello world compilation with g++ 12

```cpp
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

```
$ g++ -std=c++23 helloworld.cpp -o helloworld
helloworld.cpp:3:1: error: ‘import’ does not name a type
    3 | import std;
      | ^~~~~~
helloworld.cpp:3:1: note: C++20 ‘import’ only available with ‘-fmodules-
helloworld.cpp: In function ‘int main()’:
helloworld.cpp:5:10: error: ‘println’ is not a member of ‘std’
    5 |     std::println("Hello world!");
      |          ^~~~~~~
```

Hello world compilation with g++ 12

```
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

```
$ g++ -std=c++23 -fmodules-ts helloworld.cpp -o helloworld
```

Hello world compilation with g++ 12

```cpp
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

```
$ g++ -std=c++23 -fmodules-ts helloworld.cpp -o helloworld
In module imported at helloworld.cpp:3:1:
std: error: failed to read compiled module: No such file or directory
std: note: compiled module file is ‘gcm.cache/std.gcm’
std: note: imports must be built before being imported
std: fatal error: returning to the gate for a mechanical issue
compilation terminated.
```

Hello world compilation with g++ 12

```
// helloworld.cpp
import <print>;
int main() {
  std::println("Hello world!");
}
```

```
$ g++ -std=c++23 -fmodules-ts helloworld.cpp -o helloworld
```

Hello world compilation with g++ 12

```cpp
// helloworld.cpp
import <print>;
int main() {
  std::println("Hello world!");
}
```

```
$ g++ -std=c++23 -fmodules-ts helloworld.cpp  -o helloworld
helloworld.cpp:2:8: fatal error: print: No such file or directory
    2 | import <print>;
      |        ^~~~~~~
compilation terminated.
```

Hello world compilation with g++ 12

```
// helloworld.cpp
import <iostream>;
int main() {
  std::cout << "Hello world!\n";
}
```

```
$ g++ -std=c++23 -fmodules-ts helloworld.cpp -o helloworld
```

Hello world compilation with g++ 12

```
// helloworld.cpp
import <iostream>;
int main() {
  std::cout << "Hello world!\n";
}
```

```
$ g++ -std=c++23 -fmodules-ts helloworld.cpp -o helloworld
In module imported at helloworld.cpp:2:1:
/cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc
/cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc
/cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc
/cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc
compilation terminated.
```

Hello world compilation with g++ 12

```
// helloworld.cpp
import <iostream>;
int main() {
  std::cout << "Hello world!\n";
}
```

```
$ g++ -std=c++23 -fmodules-ts helloworld.cpp -o helloworld
In module imported at helloworld.cpp:2:1:
/X/iostream: error: failed to read compiled module: No such file or dire
/X/iostream: note: compiled module file is ‘gcm.cache/./X/iostream.gcm’
/X/iostream: note: imports must be built before being imported
/X/iostream: fatal error: returning to the gate for a mechanical issue
compilation terminated.
```

```
X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
```

Hello world compilation with g++ 12

```
// helloworld.cpp
import <iostream>;
int main() {
  std::cout << "Hello world!\n";
}
```

```
$ g++ -std=c++23 -fmodules-ts helloworld.cpp -o helloworld
In module imported at helloworld.cpp:2:1:
/X/iostream: error: failed to read compiled module: No such file or dire
/X/iostream: note: compiled module file is ‘gcm.cache/./X/iostream.gcm’
/X/iostream: note: imports must be built before being imported
/X/iostream: fatal error: returning to the gate for a mechanical issue
compilation terminated.
```

```
X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
```

List of Issues:

- Needs special flag `-fmodules-ts`.
- No ‘std’ module.
- No `print` header (despite `-std=c++23` ! - needs g++ 14).
- iostream header still needs to be compiled to a header unit

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ g++ -std=c++23 -fmodules-ts /$X/iostream
```

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ g++ -std=c++23 -fmodules-ts /$X/iostream
ld:/$X/iostream: file format not recognized; treating as linker script
ld:/$X/iostream:1: syntax error
collect2: error: ld returned 1 exit status
```

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ g++ -std=c++23 -fmodules-ts -xc++-system-header /$X/iostream
```

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ g++ -std=c++23 -fmodules-ts -xc++-system-header /$X/iostream
cc1plus: fatal error: /$X/iostream: No such file or directory
compilation terminated.
```

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ g++ -std=c++23 -fmodules-ts -xc++-system-header /$X/iostream
cc1plus: fatal error: /$X/iostream: No such file or directory
compilation terminated.
$ ls -l /$X/iostream
-rw-r--r--. 1 cvmfs cvmfs-reserved 2697 Jan 10 13:43 /$X/iostream
```

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ g++ -std=c++23 -fmodules-ts -xc++-system-header iostream
```

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ g++ -std=c++23 -fmodules-ts -xc++-system-header iostream
$
```

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ g++ -std=c++23 -fmodules-ts -xc++-system-header iostream
$ find -name '*iostream*'
```

Header unit compilation with g++ 12

- A header unit is a precompiled entity created from a (set of) C++ header file(s).
- They are essentially pre-compiled header files acting as modules.

Let's try this for iostream:

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ g++ -std=c++23 -fmodules-ts -xc++-system-header iostream
$ find -name '*iostream*'
./gcm.cache/$X/iostream.gcm
$
```

So now can we compile *helloworld*?

Hello world compilation with g++ 12

```
// helloworld.cpp
import <iostream>;
int main() {
  std::cout << "Hello world!\n";
}
```

```
$ g++ -std=c++23 -fmodules-ts -xc++-system-header iostream
$ g++ -std=c++23 -fmodules-ts helloworld.cpp -o helloworld
```

Hello world compilation with g++ 12

```cpp
// helloworld.cpp
import <iostream>;
int main() {
  std::cout << "Hello world!\n";
}
```

```
$ g++ -std=c++23 -fmodules-ts -xc++-system-header iostream
$ g++ -std=c++23 -fmodules-ts helloworld.cpp -o helloworld
$ ./helloword
Hello world!
```

Success!

Hello world compilation with clang++ 17

```
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

Hello world compilation with clang++ 17

```
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

```
$ clang++ --version
clang version 17.0.6
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /cvmfs/soft.computecanada.ca/easybuild/software/2023/x86-6
```

Hello world compilation with clang++ 17

```
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

```
$ clang++ -std=c++23 helloworld.cpp -o helloworld
```

Hello world compilation with clang++ 17

```cpp
// helloworld.cpp
import std;
int main() {
  std::println("Hello world!");
}
```

```
$ clang++ -std=c++23 helloworld.cpp -o helloworld
helloworld.cpp:2:8: fatal error: module 'std' not found
    2 | import std;
      |        ^~~
1 error generated.
```

Hello world compilation with clang++ 17

```
// helloworld.cpp
import <print>;
int main() {
  std::println("Hello world!");
}
```

```
$ clang++ -std=c++23 helloworld.cpp -o helloworld
```

Hello world compilation with clang++ 17

```
// helloworld.cpp
import <print>;
int main() {
  std::println("Hello world!");
}
```

```
$ clang++ -std=c++23 helloworld.cpp -o helloworld
helloworld.cpp:2:8: fatal error: 'print' file not found
    2 | import <print>;
      |        ^~~~~~~
1 error generated.
```

Hello world compilation with clang++ 17

```
// helloworld.cpp
import <iostream>;
int main() {
  std::cout << "Hello world!\n";
}
```

```
$ clang++ -std=c++23 helloworld.cpp -o helloworld
```

Hello world compilation with clang++ 17

```cpp
// helloworld.cpp
import <iostream>;
int main() {
  std::cout << "Hello world!\n";
}
```

```
$ clang++ -std=c++23 helloworld.cpp  -o helloworld
helloworld.cpp:2:8: error: header file <iostream> (aka /cvmfs/soft.com
putecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12
/include/g++-v12/iostream) cannot be imported because it is not known
to be a header unit
    2 | import <iostream>;
      |        ^
helloworld.cpp:4:5: error: use of undeclared identifier 'std'
    4 |     std::cout << "Hello world!\n";
      |     ^
2 errors generated.
```

Hello world compilation with clang++ 17

```
// helloworld.cpp
import <iostream>;
int main() {
  std::cout << "Hello world!\n";
}
```

```
$ clang++ -std=c++23 helloworld.cpp  -o helloworld
helloworld.cpp:2:8: error: header file <iostream> (aka /cvmfs/soft.com
putecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12
/include/g++-v12/iostream) cannot be imported because it is not known
to be a header unit
    2 | import <iostream>;
      |        ^
helloworld.cpp:4:5: error: use of undeclared identifier 'std'
    4 |     std::cout << "Hello world!\n";
      |     ^
2 errors generated.
```

List of Issues:

- Different from g++: no special flag.
- No 'std' module.
- No `print` header
- iostream header still needs to be compiled to a header unit
- Note that clang uses the g++ headers!

Header unit compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 /$X/iostream
```

Header unit compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/clang++-v
$ clang++ -std=c++23 /$X/iostream
ld:/$X/iostream: file format not recognized; treating as linker script
ld:/$X/iostream:1: syntax error
clang++: error: linker command failed with exit code 1 (use -v to see invocation)
```

Header unit compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header /$X/iostream
```

Header unit compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header /$X/iostream
/$X/iostream:36:13: warning: #pragma system_header ignored in main file [-Wpragma-system-header-outside-he
   36 | #pragma GCC system_header
      |             ^
1 warning generated.
```

Header unit compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header -Wno-pragma-system-header-outside-header /$X/iostream
$
```

Header unit compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header -Wno-pragma-system-header-outside-header /$X/iostream
$ find -name '*iostream*'
```

Header unit compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header -Wno-pragma-system-header-outside-header /$X/iostream
$ find -name '*iostream*'
./iostream.pcm
```

So now can we compile *helloworld*?

Hello world compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header -Wno-pragma-system-header-outside-header /$X/iostream
$ find -name '*iostream*'
./iostream.pcm
$ clang++ -std=c++23 helloworld.cpp -o helloworld
```

Hello world compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header -Wno-pragma-system-header-outside-header /$X/iostream
$ find -name '*iostream*'
./iostream.pcm
$ clang++ -std=c++23 helloworld.cpp -o helloworld
helloworld.cpp:2:8: error: header file <iostream> (aka '/$X/iostream') cannot be imported because it is no
    2 | import <iostream>;
      |        ^
helloworld.cpp:4:5: error: use of undeclared identifier 'std'
    4 |     std::cout << "Hello world!\n";
      |     ^
2 errors generated.
```

Hello world compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header -Wno-pragma-system-header-outside-header /$X/iostream
$ find -name '*iostream*'
./iostream.pcm
$ clang++ -std=c++23 helloworld.cpp -fmodule-file=iostream.pcm -o helloworld
```

Hello world compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header -Wno-pragma-system-header-outside-header /$X/iostream
$ find -name '*iostream*'
./iostream.pcm
$ clang++ -std=c++23 helloworld.cpp -fmodule-file=iostream.pcm -o helloworld
helloworld.cpp:2:1: warning: the implementation of header units is in an experimental phase [-Wexperimenta
    2 | import <iostream>;
      | ^
1 warning generated.
```

Hello world compilation with clang++ 17

```
$ X=cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/lib/gcc/x86_64-pc-linux-gnu/12/include/g++-v12
$ clang++ -std=c++23 -xc++-system-header -Wno-pragma-system-header-outside-header /$X/iostream
$ find -name '*iostream*'
./iostream.pcm
$ clang++ -std=c++23 helloworld.cpp -fmodule-file=iostream.pcm -Wno-experimental-header-units -o hellowor
$ ./helloworld
Hello world!
```

Success!

Pfff. . .

calcfifthroot with modules with g++ 12

```
// fifthroot.cpp
export module fifthroot;
import <cmath>;  // this is a 'header unit'
export double fifthroot(double x){
   double result = ...;  // computes result;
   return result;
}
```

```
// calcfifthroot.cpp
import <iostream>;  // this is a 'header unit'
import fifthroot;
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

calcfifthroot with modules with g++ 12

```
// fifthroot.cpp
export module fifthroot;
import <cmath>;  // this is a 'header unit'
export double fifthroot(double x){
   double result = ...;  // computes result;
   return result;
}
```

```
// calcfifthroot.cpp
import <iostream>;  // this is a 'header unit'
import fifthroot;
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

```
$ g++ -std=c++23 -fmodules-ts -xc++-system-header iostream
$ g++ -std=c++23 -fmodules-ts -xc++-system-header cmath
$ g++ -std=c++23 -fmodules-ts -c fifthroot.cpp
$ g++ -std=c++23 -fmodules-ts -c calcfifthroot.cpp -o calcfifthroot.o
$ g++ calcfifthroot.o fifthroot.o -o calcfifthroot
$ ./calcfifthroot
0.4
```

calcfifthroot with modules with g++ 12

```
// fifthroot.cpp
export module fifthroot;
import <cmath>;  // this is a 'header unit'
export double fifthroot(double x){
   double result = ...;  // computes result;
   return result;
}
```

```
// calcfifthroot.cpp
import <iostream>;  // this is a 'header unit'
import fifthroot;
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

Success!

```
$ g++ -std=c++23 -fmodules-ts -xc++-system-header iostream
$ g++ -std=c++23 -fmodules-ts -xc++-system-header cmath
$ g++ -std=c++23 -fmodules-ts -c fifthroot.cpp
$ g++ -std=c++23 -fmodules-ts -c calcfifthroot.cpp -o calcfifthroot.o
$ g++ calcfifthroot.o fifthroot.o -o calcfifthroot
$ ./calcfifthroot
0.4
```

calcfifthroot with modules with g++ 12

```
// fifthroot.cpp
export module fifthroot;
import <cmath>;  // this is a 'header unit'
export double fifthroot(double x){
   double result = ...;  // computes result;
   return result;
}
```

```
// calcfifthroot.cpp
import <iostream>;  // this is a 'header unit'
import fifthroot;
int main(){
    std::cout << fifthroot(0.01024) << "\n";
}
```

Success!

```
$ find -name '*fifthroot*' -o -name '*gcm'
./calcfifthroot.cpp
./fifthroot.cpp
./calcfifthroot.o
./fifthroot.o
./gcm.cache/fifthroot.gcm
./gcm.cache/$X/cmath.gcm
./gcm.cache/$X/iostream.gcm
./calcfifthroot
```

```
$ g++ -std=c++23 -fmodules-ts -xc++-system-header iostream
$ g++ -std=c++23 -fmodules-ts -xc++-system-header cmath
$ g++ -std=c++23 -fmodules-ts -c fifthroot.cpp
$ g++ -std=c++23 -fmodules-ts -c calcfifthroot.cpp -o calcfifthroot.o
$ g++ calcfifthroot.o fifthroot.o -o calcfifthroot
$ ./calcfifthroot
0.4
```

Remarks

- The gcm and pcm files are BMIs

Remarks

- The gcm and pcm files are BMIs
- Compiling a module produces two files: the object file and the BMI

Remarks

- The gcm and pcm files are BMIs
- Compiling a module produces two files: the object file and the BMI
- Except for header units

Remarks

- The gcm and pcm files are BMIs
- Compiling a module produces two files: the object file and the BMI
- Except for header units
- BMIs need to be compiled with the user code.

Remarks

- The gcm and pcm files are BMIs
- Compiling a module produces two files: the object file and the BMI
- Except for header units
- BMIs need to be compiled with the user code.
- Modules cannot define macros and cannot change the user code preprocessor state.

Remarks

- The gcm and pcm files are BMIs
- Compiling a module produces two files: the object file and the BMI
- Except for header units
- BMIs need to be compiled with the user code.
- Modules cannot define macros and cannot change the user code preprocessor state.
- Except for header units

Remarks

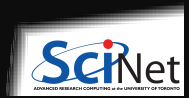
- The gcm and pcm files are BMIs
- Compiling a module produces two files: the object file and the BMI
- Except for header units
- BMIs need to be compiled with the user code.
- Modules cannot define macros and cannot change the user code preprocessor state.
- Except for header units
- If header units need other headers, the order of compilation can matter

Remarks

- The gcm and pcm files are BMIs
- Compiling a module produces two files: the object file and the BMI
- Except for header units
- BMIs need to be compiled with the user code.
- Modules cannot define macros and cannot change the user code preprocessor state.
- Except for header units
- If header units need other headers, the order of compilation can matter
- Support of compilers and build systems is fragile, but work is done to improve support.

Conclusions

- C++ modules are coming, but support of compilers and build systems is still in progress.



Conclusions

- C++ modules are coming, but support of compilers and build systems is still in progress.
- The point here was to get something to actually work.

Conclusions

- C++ modules are coming, but support of compilers and build systems is still in progress.
- The point here was to get something to actually work.
- I've not show all the difficulties, particularly with dependencies and header units.

Conclusions

- C++ modules are coming, but support of compilers and build systems is still in progress.
- The point here was to get something to actually work.
- I've not show all the difficulties, particularly with dependencies and header units.
- As hard as this is on getting support into a build system for modules, it should be clear that it is also essential; who wants to do all this work to get modules to work?

Conclusions

- C++ modules are coming, but support of compilers and build systems is still in progress.
- The point here was to get something to actually work.
- I've not show all the difficulties, particularly with dependencies and header units.
- As hard as this is on getting support into a build system for modules, it should be clear that it is also essential; who wants to do all this work to get modules to work?
- This was a far-from-comprehensive introduction to C++ modules. The full framework is quite interesting and expansive. For that, see eg.

  Andreas Weis's talk: https://www.youtube.com/watch?v=_x9K9_q2ZXE

  Daniela Engert's talk: https://www.youtube.com/watch?v=nP8QcvPpGeM

  Daniel Ruoso's talk: https://www.youtube.com/watch?v=_LGR0U5Opdg

Questions?