Modularized C++

A modern approach to C++ project setup

Alexander Christensen

Copenhagen C/C++ Meetup, August 2022

Contents

Motivation

Evolution of C++

Status Quo

Header files are problematic

Blank slate is impossible

The modules proposal

What is a module?

The promise of modules

- That we may rid our projects of header files
- No more include directories
- No more inline functions and methods everywhere
- No more "header-only" libraries
- No more magic macro customizations for building our code
- The preprocessor has no understanding of types, but the compiler has - let's use it!
- Everything will be easier, simpler, and better
- My estimate: $\sim$50% language complexity reduction
- Vastly improved build times
- No loss of expressiveness

The building blocks

We get 4 new "classifications" of files:

- Header unit (temporary solution of creating a BMI from a header file)
  - `g++ -std=c++20 -fmodules-ts -xc++-system-header iostream`
  - Creates BMI in `./gcm.cache/usr/include/c++/11/iostream.gcm`
- Module interface unit / primary module interface unit
  - This is a translation unit which exports a module
- Module partition / submodule
  - Another translation unit which belongs to a module interface
- Module implementation unit
  - A translation unit which may provide implementations to declarations in module interface

Module interface file structure

```
module;

#define NDEBUG
#include <assert.h>

export module foo;

import <string>;

export {
    int magic_value()  return 42;
}
```

← global module fragment (*May be used for preprocessor directives. Not required.*)

← module declaration (*The rest of the file is considered part of this module.*)

← import declaration (*Other modules or header units may be imported.*)

← export declaration (*Everything inside is visible for consumers of the module.*)