Cpp20 Modules: no pain, no gain

commit szczecin::cpp#2

Author: Marcin Ławicki <marcin.lawicki@meelogic.com>

Date: Thu Jun 13 19:40:00 2019 +0200

Table of contents

What is this all about?

Modules are one of the main new features included in the C++20 standard and one of the most anticipated features in history of the language.

Modules are an attempt to introduce modularization into the land of C++.

How is it working currently?

- Each source file and its includes form a single translation unit.
- Each translation unit is compiled in separation.
- Compiled translation units, an object files, are linked together to form the program/executable.

And in more details

1. preprocessing of the source code - expansion of macros
2. compilation of the source code into assembler code
3. assembly of the assembler code into object code
4. linking of the objects in order to create an executable program file

Disadvantages of current approach

- we have to rely on C++ preprocessor
- symbols space pollution
- huge translation units
- compilation is a very redundant process

Here come modules - the promise of improvement

- improved compilation times
- better isolation of interface and implementation
- no symbol space pollution
- no cross-impact between modules
- it will be even possible to get rid of the preprocessor

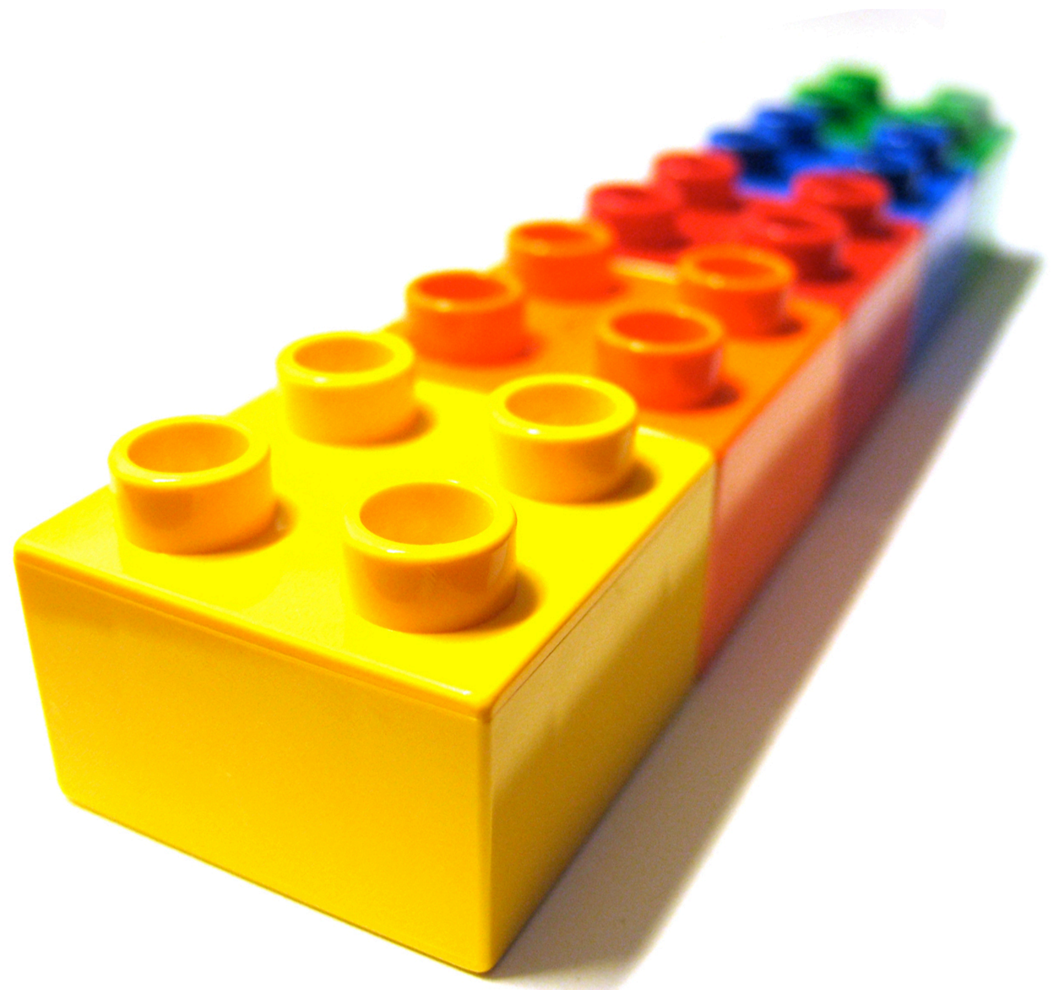
What is a module?

„A module unit is a translation unit that contains a module declaration. A named module is the collection of module units with the same module name."

- C++20 Draft

Anatomy of a module

- module interface unit
- module implementation unit
- module partition



Module interface and implementation unit

- module interface unit contains `export` keyword
- there should be exactly one module interface unit with no module partition - a primary module interface unit

```
export module module_name;
```

- any other module unit is a module implementation unit

```
module module_name;
```

Anatomy of a module - partitions

- a module unit whose module declaration contains module partition
- a named module shall not contain multiple module partitions with the same module partition declaration
- all module partitions of a module that are module interface units shall be directly or indirectly exported by the primary module interface unit
- module partitions can be imported only by other module units in the same module
- example of module interface partition unit declaration:

```
export module_name:module_part_name;
```

Module - simple example

„Talk is cheap. Show me the code.”

- Linus Torvalds

Interoperability with legacy code

- it is possible to **`import`** header files as header units
- all declarations from header unit are implicitly exported and attached to the global module
- declarations imported with header unit are not exported

How can I use it?

- Remember: current modules support does not represent the final quality of the product!
- gcc option: -fmodules-ts
- clang option: -fmodules-ts
- cl: /std:c++latest /experimental:module

Look at the modules without rose-colored glasses

non-trivial to implement in current build systems

some kind of pre-processing is still required

no modularization of standard library in C++20

risk of slowing down the compilation process in complex projects

poor adoption / dead feature / creation of dialects

FIN