P0713

EWG
2017-06-18
Daveed Vandevoorde (daveed@edg.com)

Identifying Module Source Code

P0629R0 (“Distinguishing Module Interface From Module Implementation” by Gabriel Dos Reis, Jason Merrill, and Nathan Sidwell) makes a fine case for being able to tell from source code what kind of source code we’re dealing with. I whole-heartedly support that proposal.

I would like to further request that a human be able to tell by just inspecting the initial content of a source file[1] whether it defines a module unit or just a pre-module C++ translation unit.

I would like to further request that a human be able to tell whether a reasonably-written[1] translation unit is a module unit or consists of pre-module C++ code by just inspecting the initial content of the corresponding primary source file.

For example, with N4737 as it standard (with or without the changes suggested by P0629R0, a module implementation file is somewhat likely to be structured as follows:

```
// global module declarations:
<decl>
<decl>
...
<decl>
module M;
// definition of M starts here
<decl>
...
<decl>
```

Only when “module M;” is encountered can be tell that this translation unit is actually a module unit.

I would very much like to see an “indication at the top” that the translation unit is a module unit (for all the reasons presented in P0629R0, but mostly because it is much friendlier to the programmers who have to read this source code).

My straw-man proposal, is to simple require

```
module ;
```

at the top of any module unit whose first declaration is not a module-declaration. The example above, would thus be written as follows:

```
module;
<decl>
<decl>
...
<decl>
module M;
// definition of M starts here
<decl>
...
<decl>
```

Alternatives include:

```
module global ;
```

and

```
module default ;
```

but the added identifier does not seem to be particularly helpful. Another possibility would be:

```
using module ;
```

but it feels to cute and not clearer.

A different approach would be to name the module up-front and “escape” the declarations that not not belong to the module, but that is considerably more disruptive to the design as currently (N4637) proposed.

---

1. Assuming no odd (preprocessor-based) obfuscation. ↩