Add support for preprocessing directives elifdef and elifndef

Committee: ISO/IEC JTC1 SC22 WG21

Document Number: P2334R1
Date: 2021-04-30

Authors: Melanie Blower

Reply to: Melanie.Blower@intel.com

Motivation

This paper is being submitted as a liaison activity from WG14 C Language Working Group. The proposal [1] was discussed in the March 2021 meeting and approved (15 in favor, 1 opposed, 4 abstentions) for inclusion into C23. This paper is being proposed to WG21 to avoid preprocessor incompatibilities with C and because the utility is valuable to C++ users of the preprocessor.

Summary of discussion at WG14 meeting:

Arguments against:

- We have a guidelines is to keep the preprocessor stable, and have only one way of doing things.
- Prior art argument is weak

Argument for:

- Improve teachability
- One WG14 member was able to add the functionality to his compiler in 5 minutes
- Orthogonality, aesthetics
- I like saving keystrokes
- One WG14 member found an instance where a patch was reverted in libc because it used #elifdef (demonstrating that even experienced programmers can find this unexpected)
- JeanHeyd received emails from folks that were very excited we could fix this
- Melanie received an unsolicited email from a senior embedded engineer expressing same

Introduction and Rationale

The `#ifdef` and `#ifndef` preprocessing directives exist today as shorthand for `#if defined(identifier)` and `#if !defined(identifier)`, respectively. However, no analogous shorthand preprocessing directives exist for `#elif defined(identifier)` and `#elif !defined(identifier)`. Some users have expressed surprise that these directives are not available: there are comments on the stack overflow (see https://stackoverflow.com/questions/20729032/can-we-use-elif-in-c, https://stackoverflow.com/questions/9461927/invalid-preprocessing-directive-for-elseifdef-in-xcode, https://stackoverflow.com/questions/65138617/is-there-a-c-preprocessor-which-can-replace-contiguous-else-and-ifdef-directives) and twitter (see

https://twitter.com/samykamkar/status/956784258164563968) forums from newbie users discussing the absence of these preprocessing directives.

Add two new preprocessing directives `#elifdef` and `#elifndef` that exactly parallel the functionality supplied in the existing `#ifdef` and `#ifndef` directives. This improves the expressivity and predictability of the language, especially for new users.

Note that these directives do not add any new, problematic combinations of conditional inclusion directives. e.g., this code is fine:

```
#if FOO
#elifdef BAR
#else
#endif
```

by the same reasoning that this code is already fine today:

```
#ifdef BAR
#elif FOO
#else
#endif
```

While this is a new preprocessor feature and cannot be treated as a defect report, implementations that support older versions of the standard are encouraged to implement this feature in the older language modes as well as C++23.

Prior Art

The major C and C++ compilers do not support these directives but since it is a straightforward extension of the existing capability it is certain to be simple to implement. There is some prior art that's adjacent to C and C++: software.hixie.ch provides a C-like preprocessor with #elifdef and #elifndef support, pikt.org provides tools for Linux administration and includes a file preprocessor that supports #elifdef and #elifndef. There are other tools such as lypp, a Lex Yacc preprocessor that supports %elifdef and %elifndef, and gpp, a generic preprocessor.

https://software.hixie.ch/utilities/unix/preprocessor/
http://pikt.org/pikt/ref/ref.3.ifdef_endifdef_define_setdef.html
https://github.com/trixirt/lypp
https://docs.rs/gpp/0.6.0/gpp/

Note that #ifdef has been present since C89.

Proposed Wording

The wording proposed is a diff from WG21 N4878 Green text is new text, while ~~red~~ text is deleted text.

In [cpp.pre]p. 1, modify *elif-group:*

elif-group : **# elif** *constant-expression new-line group*$_{opt}$

 # elifdef *identifier new-line group*$_{opt}$

 # elifndef *identifier new-line group*$_{opt}$

In [cpp.cond]p. 7, modify the first sentence as follows:

The #ifdef, ~~and~~ #ifndef, #elifdef, and #elifndef directives, and the defined conditional inclusion operator, shall treat __has_- include and __has_cpp_attribute as if they were the names of defined macros.

In [cpp.cond]p. 13, modify the first sentence as follows:

Preprocessing directives of the forms

 # ifdef *identifier new-line group*$_{opt}$

 # ifndef *identifier new-line group*$_{opt}$

 # elifdef *identifier new-line group*$_{opt}$

 # elifndef *identifier new-line group*$_{opt}$

check whether the identifier is or is not currently defined as a macro name.

In the same paragraph, modify the second sentence as follows:

Their conditions are equivalent to #if defined identifier, ~~and~~ #if !defined identifier, #elif defined identifier, and #elif !defined identifier respectively.

References

[1] http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2645.pdf Add support for preprocessing directives elifdef and elifndef