Proposal for C2y
Document Number: N4019

Title:	Tightening the scope of `#error`
Date:	2026-11-18
Author:	Joshua Crofts
Proposal category:	Change to existing feature

Abstract: Under certain compilers, if an `#error` directive is encountered, it will cause translation to fail; however, it does not stop translation unit processing immediately, leading to long lists of cascading error messages. This paper proposes to unify this behaviour across all implementations so that translation ceases upon encounter.

Prior art: N3960

Tightening the scope of `#error`

Document Number: N4019
Date: 2026-11-18
Reply-to: Joshua Crofts (crofts@kernel.org)

1 Motivation

Under certain compilers, if an `#error` directive is encountered during the preprocessing stage, an error message is emitted, yet preprocessing still continues. This can lead to build failures, causing long lists of potential cascading error messages, which are not very helpful to the user.

Consider the following two files:

```
// header.h
#ifdef TARGET_ARCH_SUPPORTED
typedef struct { double data[16]; int flags; } Matrix;
#else
#error "Unsupported target architecture!"
#endif

void matrix_init(Matrix *m);

// main.c
#include "header.h"

Matrix g_m;

void setup(Matrix *m) {
        m->flags = 1;
        m->data[0] = 1.0;
}

int main(void) {
        setup(&g_m);
        matrix_init(&g_m);
        return 0;
}
```

Compiling `main.c` will cause the error "Unsupported target architecture!" to be emitted, yet translation will still continue, outputting more errors; in this case unknown type name and implicit declaration errors.

2 Proposal

This paper proposes to tighten the scope of `#error`, terminating translation immediately upon encounter instead of continuing.

3 Quality of Implementation and Existing Workarounds

Currently, implementations disagree on how to handle `#error`: MSVC treats an `#error` directive as a fatal compilation error and halts translation immediately, whereas GCC and Clang emit the diagnostic and continue preprocessing and compiling the translation unit.

Existing compiler flags (such as `-Wfatal-errors` in GCC or `-ferror-limit=1` in Clang) do not adequately resolve this issue. These options are indiscriminate: they cause the compiler to abort on the *first* ordinary syntax error or typo.

In contrast, an `#error` directive is an explicit assertion written by the programmer stating that the environment is fundamentally broken, and any subsequent code in the translation unit cannot be meaningfully processed. Continuing translation will only lead to cascading errors and unhelpful noise. Tightening the scope of `#error` guarantees the intentional stop portably, without the need for global compiler flags.

4 Proposed wording

The proposed wording is a diff against the ISO/IEC 9899:202y (en) — N3998 working draft [1]. Green text is newly added text, red text is removed text.

Modify 6.10.7p1:

A preprocessing directive of either form
`# error` pp-tokens$_{opt}$ new-line
A preprocessing directive of the form
`# warning` pp-tokens$_{opt}$ new-line
causes the implementation to produce a diagnostic message that includes the specified sequence of preprocessing tokens.

A preprocessing directive of the form
`# error` pp-tokens$_{opt}$ new-line
causes the implementation to produce a diagnostic message that includes the specified sequence of preprocessing tokens; translation of the translation unit ceases.

5 Acknowledgements

I would like to thank Alex Celeste, Alejandro Colomar and Joseph Myers for their remarks on the prior art, N3960, which helped greatly. I would also like to thank Robert Seacord for his guidance and tips.

6 References

[1] ISO/IEC 9899:202y (en) — N3998 working draft,
`https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3998.pdf`