C++ Alliance Fellowship
Compiler Research
July 29, 2026

IMPLEMENT MISSING C++26 FEATURES IN CLANG

Author:
- Muhammad Bassiouni

Mentors:
- Vassil Vassilev
- Aaron Jomy

[N3356] Where we left off

01

Adds a single declaration rule

Adds a single declaration to the grammar of if/switch condition.

02

Similar to C++, but not the same

First clause of condition is a declaration and can't be empty.

Second clause of condition is only an expression, not a simple declaration.

03

Adds more declaration specifiers

Because of the nature of the first clause being a declaration.

```
if (bool x = true; x) {}
if (bool x = false) return x;
if ([[maybe_unused]] bool x = true) {}
if (bool x [[maybe_unused]] = true) {}
if ([[maybe_unused]] int x = 3; x > 0) {}
if (struct A { int x;} a = {.x = 1}; a.x) {}
if (int arr[] = {1,2,3}; arr[1]) {}
if (auto x = 1; x) {}
if (static_assert(true); true) {}
if ([[clang::assume(1 > 0)]]; true) {}
if ([[]]; true) {}
if (__attribute__((assume(1 > 0))); true) {}
if (__attribute__(()); true) {}
if (__attribute__((deprecated)) auto x = 3) {}
if (auto x __attribute__((deprecated)) = 3) {}
if (__extension__ [[maybe_unused]] int x = 3; x > 0) {}
```

```
switch (int x [[maybe_unused]] = 1) {}
switch ([[maybe_unused]] int x = 1) {}
switch (__attribute__((assume(1 > 0))); 1) {default:}
switch (__attribute__(()); 1) {default:}
switch (__attribute__((deprecated)) auto x = 3) {default:}
switch (auto x __attribute__((deprecated)) = 3) {default:}
switch (__attribute__((deprecated)) auto x = 3) {default: x += 1;}
```

[N3356] What's new

01

Related PRs got merged!

The implementation is complete per paper wording and now available as part of Clang24.0.0git.

- Commit 226b0b5
- Commit 1387002 (This work inspired a new way to handle compatibility diagnostics)

02

Backported to C89

Available in every C mode supported by Clang with compatibility warnings and respective diagnostics.

03

Added `__extension__` for C and C++

Declarations may be prefixed by one or more `__extension__` markers which silence extension diagnostics for the rest of the condition, including diagnostic for init-statement extension.

Wasn't available in C++ before.

```
1  if (bool x = true; x) {}
2  if (bool x = false) return x;
3  if ([[maybe_unused]] bool x = true) {}
4  if (bool x [[maybe_unused]] = true) {}
5  if ([[maybe_unused]] int x = 3; x > 0) {}
6  if (struct A { int x;} a = {.x = 1}; a.x) {}
7  if (int arr[] = {1,2,3}; arr[1]) {}
8  if (auto x = 1; x) {}
9  if (static_assert(true); true) {}
10 if ([[clang::assume(1 > 0)]]; true) {}
11 if ([[]]; true) {}
12 if (__attribute__((assume(1 > 0))); true) {}
13 if (__attribute__(()); true) {}
14 if (__attribute__((deprecated)) auto x = 3) {}
15 if (auto x __attribute__((deprecated)) = 3) {}
16 if (__extension__ [[maybe_unused]] int x = 3; x > 0) {}
```

```
1 switch (int x [[maybe_unused]] = 1) {}
2 switch ([[maybe_unused]] int x = 1) {}
3 switch (__attribute__((assume(1 > 0))); 1) {default:}
4 switch (__attribute__(()); 1) {default:}
5 switch (__attribute__((deprecated)) auto x = 3) {default:}
6 switch (auto x __attribute__((deprecated)) = 3) {default:}
7 switch (__attribute__((deprecated)) auto x = 3) {default: x += 1;}
```

[N3356] What's new

04

GNU attribute declaration didn't make it!

Because of the current status of GCC implementation we can't decide on whether to support those or not. GCC doesn't support `(__attribute__(...); true)` yet.

But C23-style attribute declaration did make it!

05

No reported issues so far

Since the code got merged there was no user related issues regarding the new code and no affected old codebases.

Note: the new changes touches the parser heavily.

06

Regression-free implementation

No reported regressions from Ezzeldin. The new code is as good as the old one!

```
if (bool x = true; x) {}
if (bool x = false) return x;
if ([[maybe_unused]] bool x = true) {}
if (bool x [[maybe_unused]] = true) {}
if ([[maybe_unused]] int x = 3; x > 0) {}
if (struct A { int x;} a = {.x = 1}; a.x) {}
if (int arr[] = {1,2,3}; arr[1]) {}
if (auto x = 1; x) {}
if (static_assert(true); true) {}
if ([[clang::assume(1 > 0)]]; true) {}
if ([[]]; true) {}
if (__attribute__((assume(1 > 0))); true) {}
if (__attribute__(()); true) {}
if (__attribute__((deprecated)) auto x = 3) {}
if (auto x __attribute__((deprecated)) = 3) {}
if (__extension__ [[maybe_unused]] int x = 3; x > 0) {}
```

```
switch (int x [[maybe_unused]] = 1) {}
switch ([[maybe_unused]] int x = 1) {}
switch (__attribute__((assume(1 > 0))); 1) {default:}
switch (__attribute__(()); 1) {default:}
switch (__attribute__((deprecated)) auto x = 3) {default:}
switch (auto x __attribute__((deprecated)) = 3) {default:}
switch (__attribute__((deprecated)) auto x = 3) {default: x += 1;}
```

What's next

Paper in motion:
[P1814R0] CTAD for alias templates

- Wait for GCC's final implementation for N3356 paper to decide what to do with GNU-style attribute declaration (GCC doesn't follow paper wording).
- [[maybe]] fix any occurring issues related to the implementation of N3356.
- Start working on [P1814R0] CTAD for alias templates paper.

THANK YOU

Feel free to ask questions or give feedback!

FIND ME AT
muhammad.m.bassiouni@gmail.com
github.com/bassiounix
linkedin.com/in/bassiounix