C++ Alliance Fellowship
Compiler Research
June 10, 2026

IMPLEMENT MISSING C++26 FEATURES IN CLANG

Author:
- Muhammad Bassiouni

Mentors:
- Vassil Vassilev
- Aaron Jomy

COMPILER
C | R
RESEARCH

AGENDA

01 About Me

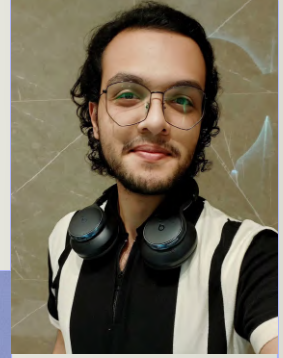
02 Papers in Motion

03 Brief About N3356

04 Brief About P1814R0

05 Current Status

About Me



- Bachelor of Artificial Intelligence (Class of 2025)

- Maintainer of LLVM C Library

- GSoC2026/LLVM Mentor (LLVM libc + compiler-rt)

Papers in Motion

C2y

N3356

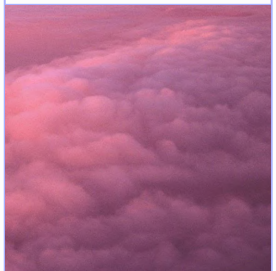
`if` declarations (if/switch).

C++20

P1814R0

Class Template Argument Deduction for Alias Templates.

Brief About Each Paper



Brief About N3356 (if/switch declaration)

01

Adds a single declaration

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.

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;}
```

Brief About P1814R0 (CTAD for Alias Templates)

01

Adds template deduction for aliases

Extends CTAD support for aliases in C++20.

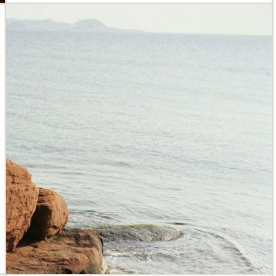
```cpp
// before

template<typename T> struct A { A(T) {} };

template<typename T> using Proxy = T;
template<typename T> using C = Proxy<A<T>>;

C<int> test {42};
```

```cpp
// after

template<typename T> struct A { A(T) {} };

template<typename T> using Proxy = T;
template<typename T> using C = Proxy<A<T>>;

C test {42};
```

Current Status

N3356

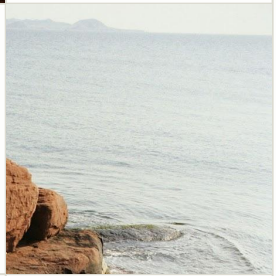
- Mostly done (see PR#198244)
- Enabled in C89 (compatible extension)
- Declaration directives are handled properly (such as `__extension__`, attribute sequence declaration and `static_assert-decl`)
- Bug free, feature-complete implementation (ahead of GCC).



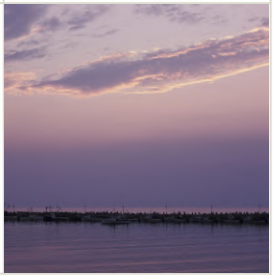
Current Status

P1814R0

- Basic implementation exist, but not stable
- Clang rejects valid code or crash in some cases
- Not enabled by default in the compiler
- Many reported issues which needs triage



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