WG14 Proposal: N3458

Cleaner C/C++ Headers Boilerplate with `extern` Blocks

Author: Yair Lenga

Email: Yair.lenga@gmail.com

Date: 2025-06-08

Document Number: N3458

Project: Programming Language C

Proposal Category: Language extension

Target Audience: WG14

Abstract

This proposal recommends that the C language adopt the `extern "C"` block syntax, already available in C++, as a no-op in C. This change allows C/C++ shared header files to use `extern "C"` blocks without needing preprocessor conditionals such as `#ifdef __cplusplus`. This improves readability, simplifies boilerplate, and encourages consistent interoperability between C and C++.

Motivation

In practice, many header files are written for both C and C++ usage. To preserve C linkage in C++ while remaining valid in C, developers commonly wrap declarations like this:

```
#ifdef __cplusplus
extern "C" {
#endif

void foo(int);

#ifdef __cplusplus
}
#endif
```

WG14 Proposal: N3458

This pattern adds clutter, is tedious to maintain, and can introduce subtle bugs. By allowing `extern "C"` blocks to appear natively in C?and simply be ignored?we eliminate the need for preprocessor guards.

Proposal

We propose allowing the C++-style `extern "C" { ... }` construct in C.

In C:

- The block is parsed but has no effect on linkage.
- All declarations inside are treated as normal.

In C++:

- Retains existing `extern "C"` semantics.

This enables truly shared headers without conditional guards.

Rationale

- No new keywords or grammar needed.
- Fully compatible with existing C++ headers.
- Simplifies headers and tooling.
- Aligns C and C++ header syntax.

Specification

Allow `extern "C" { ... }` in C when the string literal is exactly "C".

The block's contents are parsed as standard declarations, with no change to linkage or behavior.

Example:

```
extern "C" {
    int add(int, int);
}
```

Backward Compatibility

Fully backward-compatible. In C, this syntax is currently rejected; under this proposal, it is accepted and ignored.

Legacy headers using `#ifdef __cplusplus` still compile. Libraries can transition incrementally.

Internal projects with controlled build environments are likely to adopt this sooner. As C2y adoption grows, the simplified pattern will become increasingly common in both proprietary and open-source codebases.

This proposal takes a meaningful step toward long-term, frictionless interoperability between C and C++.

Implementation Considerations

- Update parser to accept `extern "C" { ... }` as a no-op block.
- No effect on linkage, symbol mangling, or ABI.
- Tooling (e.g., formatters, documentation generators) can treat it as grouping or ignore it.

Alternatives Considered

- A new keyword like `external "c"` ? rejected due to added grammar and portability risk.
- Macro guards ? verbose and error-prone.
- Pragmas or attributes ? toolchain-specific and less readable.

Prior Art

- C++ has supported `extern "C"` since C++98.
- Existing practice shows nearly all shared headers rely on conditional macros today.

Summary

This proposal allows `extern "C"` blocks to be used directly in C source and header files. It simplifies header maintenance, improves clarity, and eliminates unnecessary preprocessor complexity. It is a small change with a large ergonomic benefit, and is consistent with C2y?s goals of improving language expressiveness and compatibility.