



# On-Demand Parsing in Clang

Vipul Cariappa

2026-06-24

# Problem Statement

- A typical C++ translation unit often includes a substantial number of header files, while only a limited subset of the declarations and definitions contained within those headers is actually utilized.
- Modern compilers eagerly parse the entirety of the included code, resulting in increased compilation time and higher memory consumption.
- Example:

```cpp
#include <memory> // includes unique_ptr, shared_ptr, weak_ptr, ...

int main() {
    auto x = std::make_unique<int>(1); // only unique_ptr used
    return *x;
}
```

# Proposed Solution

- To design and implement a mechanism within the Clang compiler that defers the parsing of function and class definitions until the point at which they are first required.
- This form of on-demand parsing can avoid processing unused code paths, thereby improving efficiency.

# Technical Approach

- Tokenize input character stream to lexical tokens as usual.
- When a function or class definition is encountered store the token stream until the closing braces, `}`, within a container, and update the lookup table as usual.
- When the function or the class is looked up, its first usage, we parse the token stream into Abstract Syntax Tree (AST).

# Challenges and Considerations

# Silent Acceptance of Invalid Unused Code

- Deferred parsing introduces the possibility that syntactically incorrect code may go undetected if it is never referenced.

```
1  int foo() { return 2 + ; } // invalid code
2  int main() { return 0; }   // foo never used
```

- While this behavior is consistent with the deferred parsing model, it deviates from standard C++ requirements.

# Impact on Name Lookup Semantics

```
1  int foo(int) { return 1; }
2  int bar() { return foo(0.0); }
3  int foo(double) { return 2; }
4  int main() { return bar(); }
```

- According to the C++ standard, `main` should return `1`. However, if `bar` is parsed only after both overloads of `foo` are available, overload resolution may yield a different result.
- To address this we will need to build a source-location-aware name lookup, such that lookups are restricted to declarations that precede the definition point of the entity being parsed.

# Missing Symbols in Object Files

In large-scale projects, especially those involving static/dynamic libraries, many translation units do not contain a main function and may not directly reference all defined symbols. As a result, deferred parsing could prevent certain definitions from ever being compiled.

# JIT-Guided Parsing for Incremental Compilation

- LLVM's Orc JIT infrastructure provides the `llvm::orc::MaterializationUnit` abstraction, which enables lazy generation of `llvm::Module` instances for individual symbols.
- In this model, each symbol is registered with a callback responsible for generating its corresponding IR only when the symbol is required.

# Thank You