Syntax Forge

Code Completion in Clang Repl

Developers : Yuquan (Fred) Fu (Computer Science, Indiana University)

Mentor : Vassil Vassilev (Princeton University/CERN)

GSoC Project Proposal

Slides of the First Talk @ CaaS Meeting

Slides of the Second Talk @ CaaS Meeting

Github : capfredf

I will give a **talk** on this topic at LLVM Developers' meeting 2023.

Overview of the Project

Clang-Repl, featuring a REPL(Read-Eval-Print-Loop) environment, allows developers to program in C++ interactively. It is a C++ interpreter built upon the Clang and LLVM incremental compilation pipeline. One of the missing upstream features in Clang-Repl is the ability to propose options for automatically completing user input or code completion. Sometimes, C++ can be quite wordy, requiring users to type every character of an expression or statement. Consequently, this causes typos or syntactic errors. For example,

```
clang-repl> class HelloMyFirstClassThatHasAReallyLongName{}
clang-repl> new H<cursor>
```

Currenctly, users need to type all the rest of thirty-eight letters. However, armed with a code completion system, users will be able to either complete the input if there is only one completion result or see a list of valid completion candidates. Furthermore, the code completion should be context-aware, and it should provide semantically relevant results with respect to the current position and the input on the current line, as opposed to showing all the symbols in the current namespace. The problem is demonstrated by the example below

```
clang-repl> struct Vehicle{};
clang-repl> struct Car : Vehicle{};
clang-repl> struct Sedan : Car{};
clang-repl> void moveCar(Car &c){};
clang-repl> Vehicle v;
clang-repl> Car c1, c2;
clang-repl> Sedan s;
clang-repl> c.move(<tab>
```

If users hit the `<tab` > key at the indicated position, listing all symbols would be distracting. It is easy to find out that among all declarations, only `c1` , `c2` and `s` are well-typed candidates. So an ideal code completion system should be able to filter out results using type information.

The project leverages existing components of Clang/LLVM and aims to provides context-aware semantic completion suggestions.

My Approach

The project mainly consists of two patches. The first patch involves building syntactic code completion based on Clang/LLVM infrastruture. The second patch goes one step further by implementing type directed code completion.

Pull Request : D154382

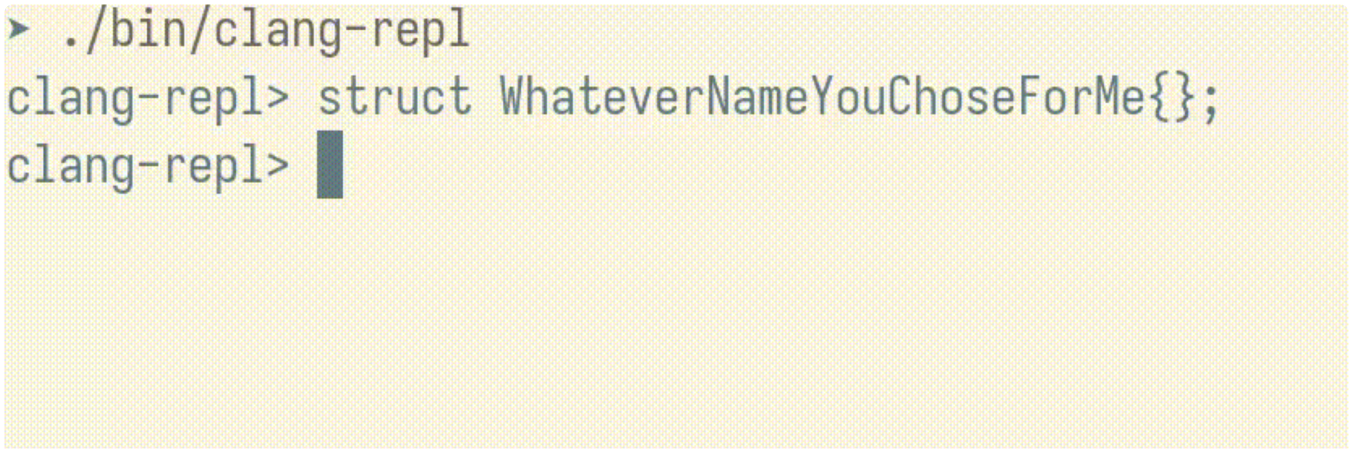
Highlights

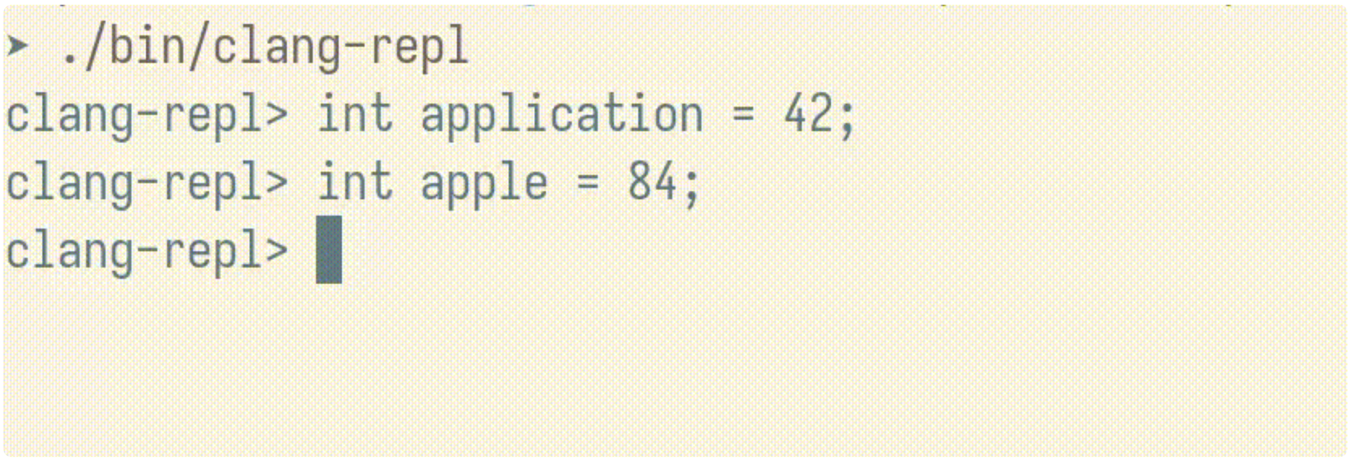
1. In the submitted patch, we have multiple iterations to integrate the new

components with the existing infrastructure while not reinventing the wheel. For each code completion, we create a special AST unit called `ASTUnit` with the current input and invoke its method `ASTUnit::codeComplete` with a completion point to do the heavy-lifting job.

2. `Sema/CodeComplete*` are a collection of modules in Clang that play an central role in code completion. We added new completion contexts so the `Sema/CodeComplete*` can provide correct completion results for the new declaration kind that Clang-Repl uses model statements on the global scope. The underlying reason is that in a regular C++ file, expression statements are not allowed to appear at the top level. Therefore, `Sema/CodeComplete*` would exclude invalid completion candidates for expression statements, which are nonetheless common inputs at the REPL.
3. `Sema/CodeComplete*` assume the input is an intact source file or AST context by default. Because a new compiler instance is created whenever code completion is triggered, `Sema/CodeComplete*` would not be able to see all declarations defined by previous inputs in the same REPL session. The solution is to construct an `ExternalASTSource` with `ASTContext` s from both the code completion and main compiler instances, and use that `ExternalASTSource` as the external source of the code completion's `ASTContext` . Code completion invokes `ExternalASTSource::completeVisibleDeclsMap` , where we import decls from the main `ASTContext` to the code completion `ASTContext` .

Demo


```
➤ ./bin/clang-repl
clang-repl> struct WhateverNameYouChoseForMe{};
clang-repl> █
```

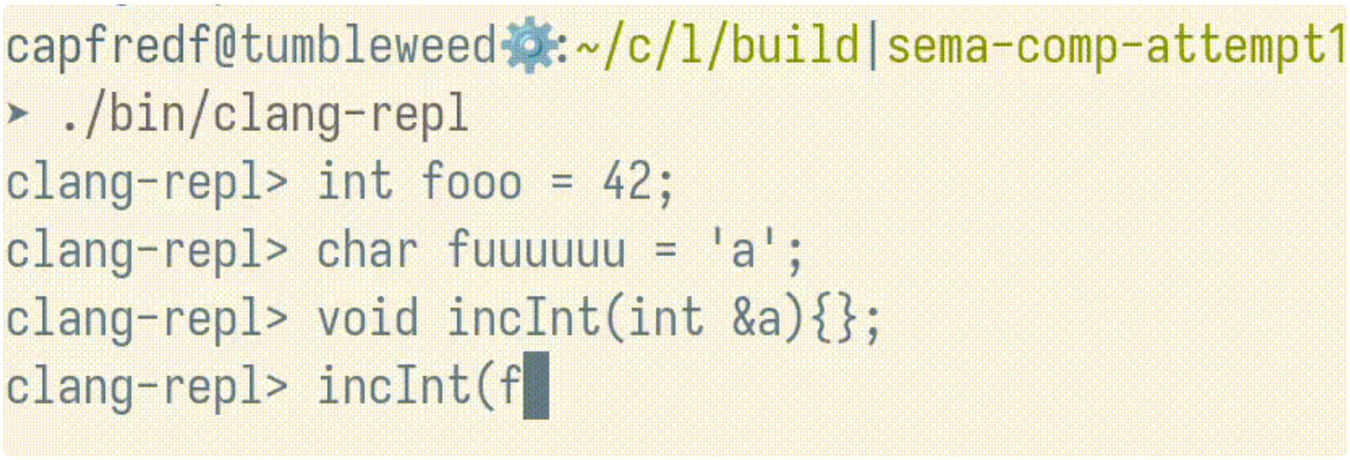


```
➤ ./bin/clang-repl
clang-repl> int application = 42;
clang-repl> int apple = 84;
clang-repl> 
```

Future Work

Pull Request : D159128

The type-directed code completion is still a work in progress. It was developed based on an early version of the patch submitted. With this feature, code completion results are further narrowed down to well-typed candidates with respect to completion points. Here is a screecast:



```
capfredf@tumbleweed⚙:~/c/l/build|sema-comp-attempt1
➤ ./bin/clang-repl
clang-repl> int fooo = 42;
clang-repl> char fuuuuuu = 'a';
clang-repl> void incInt(int &a){};
clang-repl> incInt(f
```

Conclusion & Acknowledgments

The journey has been incredibly thrilling. I have honed my C++ skills and delved into Clang/LLVM with a focus on interactions of components responsible for parsing. Thanks to everything I learned from the project, I feel confident in becoming a better Clang/LLVM contributor and compiler hacker.

Last but not the least, I would like to express gratitude to my mentor Vassil for his many valuable discussions and feedback regarding the patch. His guidance ensured the project procceeded smoothly. Without him, I would have not been able to complete the project in a timely manner.