Code Completion in Clang-Repl

Yuquan (Fred) Fu

Vassil Vassilev
What is Clang-Repl?
```cpp
#include <iostream>

std::string str = "Hello, World!";
std::cout << str << " has " << str.length() << " characters!\n";

Hello, World! has 13 characters!
```

```bash
clang-repl> #include <iostream>
clang-repl> std::string str = "Hello, World!";
clang-repl> std::cout << str << " has " << str.length() << "characters!\n";
Hello, World! has 13 characters!
```
```cpp
#include <iostream>

std::string str = "Hello, World!";
std::cout << str << " has " << str.length() << " characters!\n";
Hello, World! has 13 characters!
str.
```
```cpp
#include <iostream>

std::string str = "Hello, World!";
std::cout << str << " has " << str.length() << " characters!\n";
Hello, World! has 13 characters!
```

My Project!
What have we achieved?
Basic Code Completion

clang-repl> class ModulePointerAndOffsetLessThanFunctionObject{ ... };
Basic Code Completion

clang-repl> class ModulePointerAndOffsetLessThanFunctionObject{ ... }; 
clang-repl> ModulePointerAndOffsetLessThanFunctionObject
Semantic Code Completion

clang-repl> int number1 = 42, number2 = 84;
clang-repl> std::string name1 = "Fred", name2 = "Vassil";
clang-repl> template <typename T> T pickOne(T v1, T v2) {...};
clang-repl> pickOne(number1, ←
Semantic Code Completion

clang-repl> int number1 = 42, number2 = 84;
clang-repl> std::string name1 = "Fred", name2 = "Vassil";
clang-repl> template <typename T> T pickOne(T v1, T v2) {...};
clang-repl> pickOne(number1, number1,
number1
number2
Overcame challenges of reusing Sema/CodeComplete to implement code completion in REPL
Challenges
Declaration Visibility Issue

```cpp
int num1 = 84;
int num2 = 76;
int num3 = 42;
int res = 1 + num1 + num2 + num3;
```

```bash
clang-repl> int num1 = 84;
clang-repl> int num2 = 76;
clang-repl> int num3 = 42;
clang-repl> int res = 1 + num1 + num2 + num3;
```
Declaration Visibility Issue

```c
int num1 = 84;
int num2 = 76;
int num3 = 42;
int res = 1 + num1 + num2 + num3;
```

Declaration Visibility Issue

Why does the code completion system fail to see previously defined declarations in REPL?

- A file is one single translation unit enclosed by one ASTContext

```cpp
int num1 = 84;
int num2 = 76;
int num3 = 42;
int res = 1 + num1;
```

1 ASTContext & 1 TranslationUnit
Declaration Visibility Issue

Why does the code completion system fail to see previously defined declarations in REPL?

- A REPL session contains multiple partial translation units enclosed by two ASTContexts

```plaintext
clang-repl> int num1 = 84;
clang-repl> int num2 = 76;
clang-repl> int num3 = 42;
clang-repl> int res = 1 + n*
```

2 ASTContexts & 4 PartialTranslationUnits
Solution to Declaration Visibility Issue

clang-repl> ...  
clang-repl> int num1 = 84;  
clang-repl> int num2 = 76;  
clang-repl> int num3 = 42;  
clang-repl> int res = 1 + num1;

clang-repl> int num2 = 76;
clang-repl> int num3 = 42;
clang-repl> int res = 1 + num1;
Solution to Declaration Visibility Issue

```plaintext
clang-repl> ...  
clang-repl> int num1 = 84;  
clang-repl> int num2 = 76;  
clang-repl> int num3 = 42;  
clang-repl> int res = 1 + num3
```

![Diagram showing the solution to declaration visibility issue]
Solution to Declaration Visibility Issue

```cpp
clang-repl> ... 
clang-repl> int num1 = 84;
clang-repl> int num2 = 76;
clang-repl> int num3 = 42;
clang-repl> int res = 1 + num3
```

![Diagram showing the solution to declaration visibility issue](image)
Solution to Declaration Visibility Issue

```cpp
clang-repl> ...  
clang-repl> int num1 = 84;
clang-repl> int num2 = 76;
clang-repl> int num3 = 42;
clang-repl> int res = 1 + num3;
```
Code Completion for Top Level Expressions

• Top level expressions are syntactically invalid in a regular C++ file
• Top level expressions are bread and butter in REPL

```
clang-repl> int num = 42;
clang-repl> 1 + n
```
Code Completion for Top Level Expressions

• Top level expressions are syntactically invalid in a regular C++ file
• Top level expressions are bread and butter in REPL

```
clang-repl> int num = 42;
clang-repl> 1 + n
```
Code Completion for Top Level Expressions

- Top level expressions are syntactically invalid in a regular C++ file
- Top level expressions are bread and butter in REPL

```
clang-repl> int num = 42;
clang-repl> 1 + n
```

CompletionContext::Kind = CCC_TopLevel
Code Completion for Top Level Expressions

• Top level expressions are syntactically invalid in a regular C++ file
• Top level expressions are bread and butter in REPL

```
clang-repl> int num = 42;
clang-repl> 1 + n
```
CompletionContext::Kind = CCC_TopLevelOrExpression
Code Completion for Top Level Expressions

- Top level expressions are syntactically invalid in a regular C++ file
- Top level expressions are bread and butter in REPL

```bash
clang-repl> int num = 42;
clang-repl> 1 + n
```

CompletionContext::Kind = CCC_TopLevelOrExpression
Code Completion for Top Level Expressions

• Top level expressions are syntactically invalid in a regular C++ file
• Top level expressions are bread and butter in REPL

```
clang-repl> int num = 42;
clang-repl> 1 + num
CompletionContext::Kind = CCC_TopLevelOrExpression
```
Semantic Code Completion
What Semantic Code Completion Needs

• What context is the cursor in?

```
clang-repl> f
```
```
clang-repl> car.
```

• How to get the type w.r.t the cursor position?

```
clang-repl> pickOne(name1, )
```
Key Structure for Semantic Code Completion

**CodeCompletionContext**

- ::getKind() shows the context kind
  
  ```
  clang-repl> f
  CCC_TopLevelOrExpression
  clang-repl> car.
  CCC_DotMemberAccess
  ```

- ::getPreferedType() reveals the type w.r.t the current cursor position
  
  ```
  clang-repl> pickOne(name1, std::string
  ```
Key Structure for Semantic Code Completion

- CodeCompletionContext::getBaseType() returns the type of the expressions before the dot

```cpp
class Car {
  public:
    int getPrice();
    void sell(Person& p);
};
```

```cpp
Car car1
.car1.~
getPrice
sell
```
Implementation
First Attempt

```cpp
clang-repl> int num = 42;
clang-repl> int res = 1 + n;
```

New IncrementalCompilerInstance
First Attempt

```
clang-repl> int num = 42;
clang-repl> int res = 1 + n
```
clang-repl> int num = 42;
clang-repl> int res = 1 + n;
First Attempt

```cpp
clang-repl> int num = 42;
clang-repl> int res = 1 + num
```
First Attempt

```
clang-repl> int num = 42;
c clang-repl> int res = 1 + n
```
Using ASTUnit

```cpp
clang-repl> int num = 42;
clang-repl> int res = 1 + n
```

```c
ASTUnit* au = LoadFromCompilerInvocationAction(New IncrementalCI)
```
Using ASTUnit

```cpp
clang-repl> int num = 42;
clang-repl> int res = 1 + n
```

```
ASTUnit* au = LoadFromCompilerInvocationAction(New IncrementalCI)
au->codeComplete("<<<input>>>", 1, 15, New IncrementalStxOnlyAct, New ReplCodeCompletion)
```
Using ASTUnit

```
clang-repl> int num = 42;
clang-repl> int res = 1 + n
```

![Diagram of ASTUnit usage]
Using ASTUnit

```plaintext
clang-repl> int num = 42;
clang-repl> int res = 1 + n
```

![Diagram of ASTUnit process]

1. `ASTUnit* au = LoadFromCompilerInvocationAction(New IncrementalCI)`
2. `au->codeComplete("<<<input>>>", 1, 15), New IncrementalStxOnlyAct, New ReplCodeCompletion)`
3. `BeginSourceFile`
4. `Execute`
5. `ReplCodeCompletionConsumer::ProcessCompletionResults`
6. `Trigger Code Completion`
7. `EnableCodeCompletion(FrontendOpts.CompletionAt)`
Using ASTUnit

```javascript
clang-repl> int num = 42;
clang-repl> int res = 1 + n
```

```javascript
ASTUnit* au = LoadFromCompilerInvocationAction(New IncrementalCI)
au->codeComplete("<<input>>", 1, 15), New IncrementalStxOnlyAct, New ReplCodeCompletion)
BeginSourceFile
Execute
ReplCodeCompletionConsumer::ProcessCompletionResults
Trigger Code Completion
EnableCodeCompletion(FrontendOpts.CompletionAt)
```
Conclusions

• Solved the visibility issue with ASTImporter and ExternalSource
• Enabled code completion in top level expressions with a new CompletionContext
• Leveraged Sema modules to achieve semantic code completions
• Concise implementation with minimal invasive changes