Syntax Macros: a Case-Study in Extending Clang

Dr. Norman A. Rink
Technische Universität Dresden, Germany
norman.rink@tu-dresden.de

LLVM Cauldron
8 September 2016
Hebden Bridge, England
Who we are

Chair for Compiler Construction
(since 2014)

Prof. Jeronimo Castrillon
- code generation for multicore systems-on-chip
- dataflow programming models
- heterogeneous platforms

Dr. Sven Karol
- domain-specific languages (DSLs) and tools
- languages for numerical applications
- software composition

For more details and a full list of staff visit:
https://cfaed.tu-dresden.de/ccc-about
Macros are the world’s second oldest programming language.*

- macros are a meta-programming tool
  - can be used to abstract programming tasks
  - reduce repetition of code patterns, esp. boilerplate code
  - “old” example: macro assembler

- preprocessor (PP) macros
  - very widely used
  - textual replacement → no type safety, poor diagnostics (but improving)

- syntax macros
  - expand to sub-trees of the AST (abstract syntax tree)
  - compose programs in the sense that ASTs are composed
  - compiler can check that the composed AST is valid

*) D. Weise, R. Crew
SCALLED\_SUBSCRIPT(a, i, c)

\[ a[c*i] \]

PP macro \hspace{1cm} syntax macro

\begin{itemize}
  \item typing of AST nodes enables
  \begin{itemize}
    \item correctness checks
    \item better diagnostics
    \item reduced prone-ness to unintended behaviour
  \end{itemize}
\end{itemize}
Defining syntax macros

signature

```
$$[Expr]\ ADD\ (Expr[int]\ var\ $\ IntegerLiteral[int]\ num)$$
```

body

```
$$\ var\ +\ $$\ num$$
```
Using syntax macros

```c
int simple() {
    int x = 1;
    x = $ADD(x, 41);
    return x;
}
```

-FunctionDecl simple 'int ()'
  `-CompoundStmt
    | `-DeclStmt
    |   `-VarDecl x 'int'
    |     `-IntegerLiteral 'int' 1
    |   `-BinaryOperator 'int' '='
    |     | `-DeclRefExpr 'int' lvalue Var 'x' 'int'
    |     `-BinaryOperator 'int'+'
    |       | `-ImplicitCastExpr 'int' <LValueToRValue>
    |       |   `-DeclRefExpr 'int' lvalue Var 'x' 'int'
    |     | `-IntegerLiteral 'int' 41
    |   `-ReturnStmt
    |     `-ImplicitCastExpr 'int' <LValueToRValue>
    |     | `-DeclRefExpr 'int' lvalue Var 'x' 'int'
```
Summary of syntax macros

- Goal: use syntax macros instead of PP macros everywhere.
  - For safety and better diagnostics.
  - Are there any theoretical limitations to replacing PP macros?

- Use cases:
  - Find (potential) errors in code that relies on PP macros.
  - Aid language designers in prototyping syntactic sugar.
  - Here: toy model used to study the extensibility of Clang.
  - Further suggestions welcome!

- Reference: “Programmable Syntax Macros” (PLDI 1993)
  - by D. Weise, R. Crew
  - Describes a more comprehensive system than the prototype discussed here.
How to parse macro definitions

$$[\text{Expr}] \ \text{ADD} \ (\text{Expr}[\text{int}] \ \text{var} \ \$ \ \text{IntegerLiteral}[\text{int}] \ \text{num}) \ \$$\text{var} + \ $$\text{num}

- Replace Parser by MacroParser in ParseAST.
- Macro signature:
  - Look out for $$ at the beginning of a statement.
  - If $$ is present, parse the macro signature.
  - Otherwise, defer to statement parsing in base class Parser.
- Macro body:
  - Look out for $$$ to indicate macro parameter expression.
  - Otherwise, defer to statement/expression parsing in Parser.

Very natural to use polymorphism to adjust the parser’s behaviour.
How to instantiate macros

$\text{ADD}(x \; 41)$

- If $\$\text{ at the beginning of an expression},
  - parse the macro parameters.
  - instantiate the macro body’s AST with the parameters pasted in.
- Otherwise defer to expression parsing in the base class Parser.

Again, very natural to use polymorphism to adjust the parser’s behaviour.
$ADD(x, 41)$

- No virtual methods needed since MacroParser knows that it calls into MacroSema for constructing the AST.

- Subtlety: Placeholder node in the AST.
  - Required to represent (formal) macro parameters in the body AST.
  - Must type-check that parameters are in scope in the macro body.

Introduction of new AST nodes is tedious.
Problems with semantics and scope

- Problem: return statements are only valid inside function scope.
  - If the macro is defined at global scope, Sema will silently produce an empty AST for the macro body.

- Problem: \( x \) may not be bound correctly.
  - If \( x \) is in scope at the macro definition, it will be bound. \( \rightarrow \) Binding may be incorrect at macro instantiation.
  - If \( x \) is not in scope, it is a free variable. \( \rightarrow \) Sema will raise an error.

This is the “open scope problem”:
What is a suitable scope for macro definitions?
## Summary of extensibility issues

<table>
<thead>
<tr>
<th>problem/need</th>
<th>solution</th>
<th>benefit</th>
<th>difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>polymorphism of Parser</td>
<td>make Parser virtual</td>
<td>enables language extensions, DSLs</td>
<td>easy, but may impact performance</td>
</tr>
<tr>
<td>polymorphism of CodeGen</td>
<td>make CodeGen virtual</td>
<td>eases implementation of new compiler flags</td>
<td>easy, but may impact performance</td>
</tr>
<tr>
<td>new AST node types</td>
<td>add generic sub-classes of Stmt, Expr etc.</td>
<td>makes the AST readily extensible, reduces boilerplate code required for prototyping</td>
<td>moderate, must integrate with existing infrastructure</td>
</tr>
<tr>
<td>adjust the behaviour of Sema to the parser’s context</td>
<td></td>
<td>enable extensions/DSLs with fully independent semantics</td>
<td>easy if doable by Scope class, moderate to hard otherwise</td>
</tr>
<tr>
<td>“open context problem”</td>
<td>separate Parser from Sema?</td>
<td>full extensibility of C/C++, including semantics</td>
<td>hard</td>
</tr>
</tbody>
</table>

- Deliberate blank: How to support embedded semantics without fully separating Parser and Sema?
- Medium-term goal: Have a clean interface for adding language extensions to Clang.
Source code for syntax macros

Sources can be found on GitHub:

Norman Rink  https://github.com/normanrink

- extended Clang:
  https://github.com/normanrink/clang-syntax-macros
- compatible (vanilla) version of LLVM:
  https://github.com/normanrink/llvm-syntax-macros

Please contribute:
questions, bugs, patches, improvements all welcome!
Syntax Macros: a Case-Study in Extending Clang

Dr. Norman A. Rink
Technische Universität Dresden, Germany
norman.rink@tu-dresden.de

Work supported by the German Research Foundation (DFG) within the Cluster of Excellence ‘Center for Advancing Electronics Dresden’ (cfaed).

Thank you.