Dmd as a Library – Myth & Reality

Razvan Nitu – Dconf 2023
Overview

• Compiler Architecture
• Goals of Compiler Library
• Current status
• Limitations
• Future work
• Conclusions
Compiler Architecture
DMD Architecture

void fun(string str) {
    import std.io;
    writeln(str);
}

class Lexer {
    nextToken()
    peekPastParen()
    allocateToken
}

class Parser : Lexer {
    parseModule()
    parseExpression()
    parseDeclaration()
}

Abstract Syntax Tree

- FuncDeclaration
  - fun
  - void
  - string str
  - TypeBasic
  - VarDeclaration
    - importDeclaration
    - import std.io
    - writeln
    - CallExp
      - IdentifierExp
      - str
Goals for the Compiler Library
Goals for the Compiler Library

- Have a separation between compilation phases.
- Have the ability to modify AST nodes.
- Have the ability to modify the compilation phases.
- Have the ability to ask the compiler for any kind of information.
- Have the ability to use multiple instances of the compiler in the same program.
- Library and release compiler share the same code base.
- No performance penalty for the release compiler.
We are not (yet) concerned about

- Incremental compilation
- Compiler library performance
- Potential compilation slowdown when compiling the compiler
Current Status
Progress

- Template parser with AST Family
- Extract semantic analysis from AST nodes
- Use dmdlib for existing projects
- Customize error handling
Template Parser with AST Family

```java
import dmd.lexer;
import dmd.expression;
import dmd.statement;

// ...
// import entire compiler codebase

class Parser : Lexer
{
    Statement parseStatement()
    {
        // ...
        Statement s = new Statement(/* whatever params */);
        // ...
    }

    Expression parseAssignExpression()
    {
        // ...
        AssignExp exp = new AssignExp(/* whatever params */);
        // ...
    }
}
```

```java
import dmd.lexer;

class Parser(AS): Lexer
{
    AST.Statement parseStatement()
    {
        // ...
        AST.Statement s = new AST.Statement(/* whatever params */);
        // ...
    }

    AST.Expression parseAssignExpression()
    {
        // ...
        AST.AssignExp exp = new AST.AssignExp(/* whatever params */);
        // ...
    }
}
```
Template parser with AST Family

```java
struct ASTCodeGen {
    // import all files containing ast nodes
    public import dmd.expression;
    public import dmd.statement;
    // ...
}

struct ASTBase {
    class Expression {
        // contains only fields and methods required at parse time
    }
    class Statements {
        // contains only fields and methods required at parse time
    }
    // ...
}
```
The problem of ASTBase

- Extract methods and fields that rely on semantic information so that ASTBase == ASTCodegen.
- Extract the common code for ASTBase and ASTCodegen into a mixin template. (PR: https://github.com/dlang/dmd/pull/6966)
class BinExp : Expression
{
    // ...
    override Expression semantic()
    {
        // do semantic
    }
    // ...
    override void accept(Visitor v)
    {
        v.visit(this);
    }
}
class AssignExp : BinExp
{
    // ...
    override Expression semantic()
    {
        // do semantic
    }
    // ...
    override void accept(Visitor v)
    {
        v.visit(this);
    }
}
class ExpressionSemanticVisitor : Visitor
{
    Expression result;
    override void visit(BinExp be)
    {
        // do semantic and set result somewhere
    }
    override void visit(AssignExp)
    {
        // do semantic and set result somewhere
    }
    // ...
    Expression semantic(Expression exp)
    {
        auto sv = new ExpressionSemanticVisitor();
        exp.accept(sv);
        return sv.result;
    }
}
Use dmdlib in Existing Projects

- DCD
- D-Scanner
DCD

- D Completion Daemon
- Uses libdparse, dsymbol
Scope

Abstract Syntax Tree

FuncDeclaration
  fun
  void
  string str
  ImportDeclaration
    import stdstdio
    IdentifierExp
      str
      then
      else
      CompoundStatement
      CompoundStatement

CallExp
  writeln

VarDeclaration
  CompoundStatement

IfStatement
  s1
  s2
  s3
  s4
  s5
Need to publicize scopes
D-Scanner

- Replace libdparse
- Lexer modifications:
  - Range interface
  - Lex white spaces and newlines
  - Template parser with lexer
- Add location to AST nodes
Limitations
struct MyAST
{
    import dmd.expression;
    import dmd.statement;
    //...

class MyBinExp : dmd.expression.BinExp
{
    // add extra functionality
}

class BinExp : Expression { /* ... */}
class AssignExp : BinExp { /*...*/}
Other limitations

- Cannot obtain scope information
- Information gets lost during semantic passes
- Most classes/methods are `extern(C++)`
- Dmd uses null terminated strings
Moving Forward

- Finish D-Scanner integration of dmdlib
- SAoC – integrate dmdlib in dformat
- Unused import tool
- Extract more semantic information from AST nodes
- Get to: new Compiler(MyParser!(MyAST, MyLexer), MySemanticAnalysis).
Conclusions
Conclusions

- Have a separation between compilation phases.
- Have the ability to modify AST nodes.
- Have the ability to modify the compilation phases.
- Have the ability to ask the compiler for any kind of information.
- Have the ability to use multiple instances of the compiler in the same program.
- Library and release compiler share the same code base.
- No performance penalty for the release compiler.
Useful links

- Dmd as a library talk 2017
- Integrating dmd as a library in D-scanner 2022 talk
- DCD work
- Unused import tool
Questions