Magic Behind Xcode

Compilation

Mobile Warsaw, 2015

> whoami

Twitter:
@1101_debian

Github:
@AlexDenisov

Freenode:
AlexDenisov

Blog:
http://lowlevelbits.org

Outline

- Compilation process
- LLVM/Clang
- Q & A

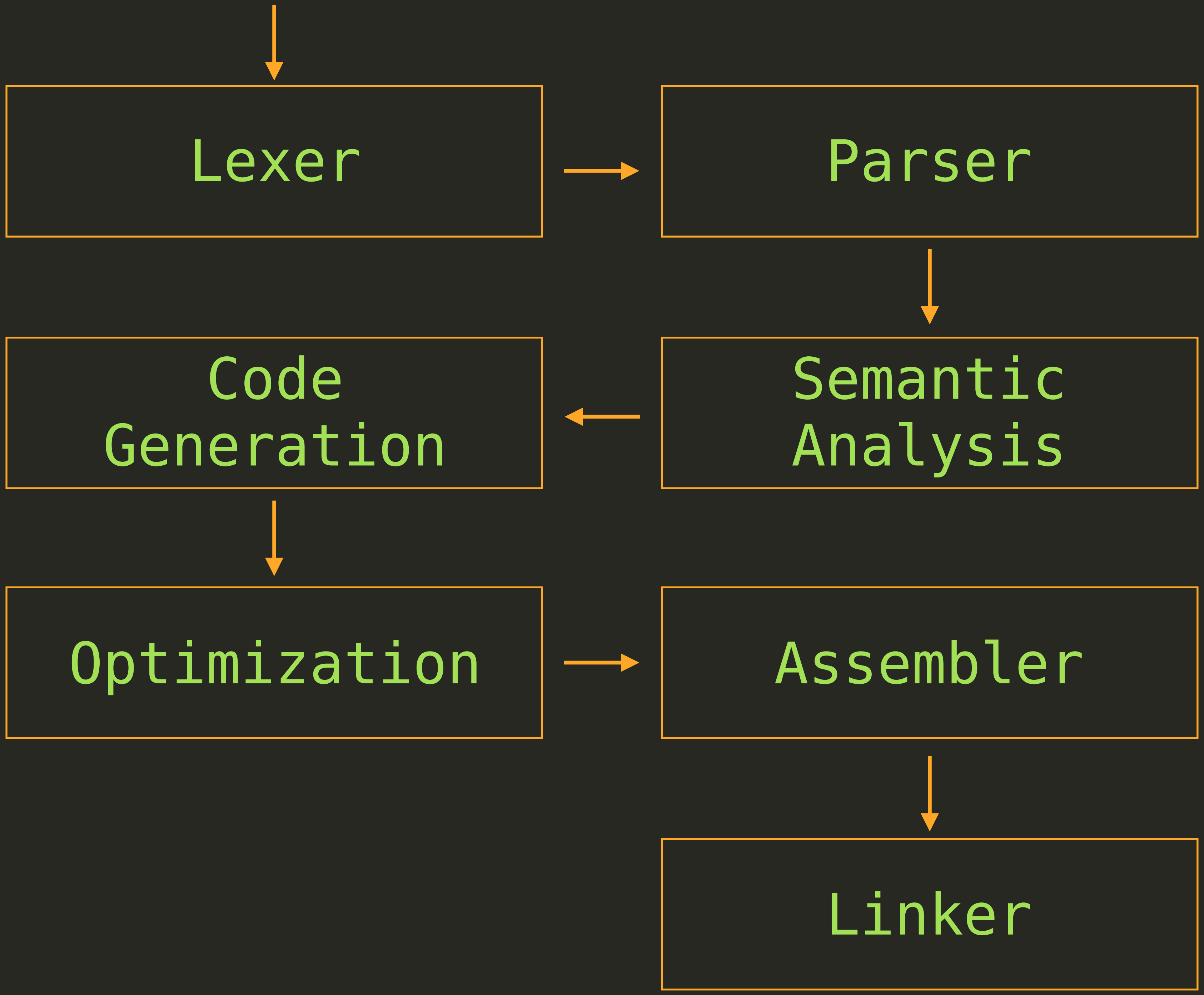
Compilation Process

```c
int main(){
  return 0;
}
```

```
0000000 cf fa ed fe 07 00 00 01 03 00 00 80 02 00 00 00
0000010 0f 00 00 00 38 03 00 00 85 00 20 00 00 00 00 00
0000020 19 00 00 00 48 00 00 00 5f 5f 50 41 47 45 5a 45
0000030 52 4f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000040 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
0000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000060 00 00 00 00 00 00 00 00 19 00 00 00 38 01 00 00
0000070 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000080 00 00 00 00 01 00 00 00 00 10 00 00 00 00 00 00
0000090 00 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00
00000a0 07 00 00 00 05 00 00 00 03 00 00 00 00 00 00 00
00000b0 5f 5f 74 65 78 74 00 00 00 00 00 00 00 00 00 00
00000c0 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
00000d0 98 0f 00 00 01 00 00 00 08 00 00 00 00 00 00 00
00000e0 98 0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000f0 00 04 00 80 00 00 00 00 00 00 00 00 00 00 00 00
0000100 5f 5f 75 6e 77 69 6e 64 5f 69 6e 66 6f 00 00 00
0000110 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000120 a0 0f 00 00 01 00 00 00 48 00 00 00 00 00 00 00
0000130 a0 0f 00 00 02 00 00 00 00 00 00 00 00 00 00 00
0000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000150 5f 5f 65 68 5f 66 72 61 6d 65 00 00 00 00 00 00
0000160 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000170 e8 0f 00 00 01 00 00 00 18 00 00 00 00 00 00 00
0000180 e8 0f 00 00 03 00 00 00 00 00 00 00 00 00 00 00
0000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001a0 19 00 00 00 48 00 00 00 5f 5f 4c 49 4e 4b 45 44
00001b0 49 54 00 00 00 00 00 00 00 10 00 00 01 00 00 00
00001c0 00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00
00001d0 d8 00 00 00 00 00 00 00 07 00 00 00 01 00 00 00
00001e0 00 00 00 00 00 00 00 00 22 00 00 80 30 00 00 00
00001f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
int main(){
  return 0;
}
```

```
0000000 cf fa ed fe 07 00 00 01 03 00 00 80 02 00 00 00
0000010 0f 00 00 00 38 03 00 00 85 00 20 00 00 00 00 00
0000020 19 00 00 00 48 00 00 00 5f 5f 50 41 47 45 5a 45
0000030 52 4f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000040 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
0000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000060 00 00 00 00 00 00 00 00 19 00 00 00 38 01 00 00
0000070 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000080 00 00 00 00 01 00 00 00 00 10 00 00 00 00 00 00
0000090 00 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00
00000a0 07 00 00 00 05 00 00 00 03 00 00 00 00 00 00 00
00000b0 5f 5f 74 65 78 74 00 00 00 00 00 00 00 00 00 00
00000c0 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
00000d0 98 0f 00 00 01 00 00 00 08 00 00 00 00 00 00 00
00000e0 98 0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000f0 00 04 00 80 00 00 00 00 00 00 00 00 00 00 00 00
0000100 5f 5f 75 6e 77 69 6e 64 5f 69 6e 66 6f 00 00 00
0000110 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000120 a0 0f 00 00 01 00 00 00 48 00 00 00 00 00 00 00
0000130 a0 0f 00 00 02 00 00 00 00 00 00 00 00 00 00 00
0000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000150 5f 5f 65 68 5f 66 72 61 6d 65 00 00 00 00 00 00
0000160 5f 5f 54 45 58 54 00 00 00 00 00 00 00 00 00 00
0000170 e8 0f 00 00 01 00 00 00 18 00 00 00 00 00 00 00
0000180 e8 0f 00 00 03 00 00 00 00 00 00 00 00 00 00 00
0000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001a0 19 00 00 00 48 00 00 00 5f 5f 4c 49 4e 4b 45 44
00001b0 49 54 00 00 00 00 00 00 00 10 00 00 01 00 00 00
00001c0 00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00
00001d0 d8 00 00 00 00 00 00 00 07 00 00 00 01 00 00 00
00001e0 00 00 00 00 00 00 00 00 22 00 00 80 30 00 00 00
00001f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```


Lexer
Parser
Semantic Analysis
Code Generation
Optimization
Assembler
Linker

```c
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

https://github.com/AlexDenisov/mbx/blob/master/calc.c

Lexer

```
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

```
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

```
(KW ‘const’)
```

```
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

```
(KW ‘const’), (TYPE ‘float’)
```

```
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

```
(KW ‘const’), (TYPE ‘float’), (ID ‘factor'),
(EQ ‘=‘), (NUM ’42.f’), (SEMI ‘;’)
```

```
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

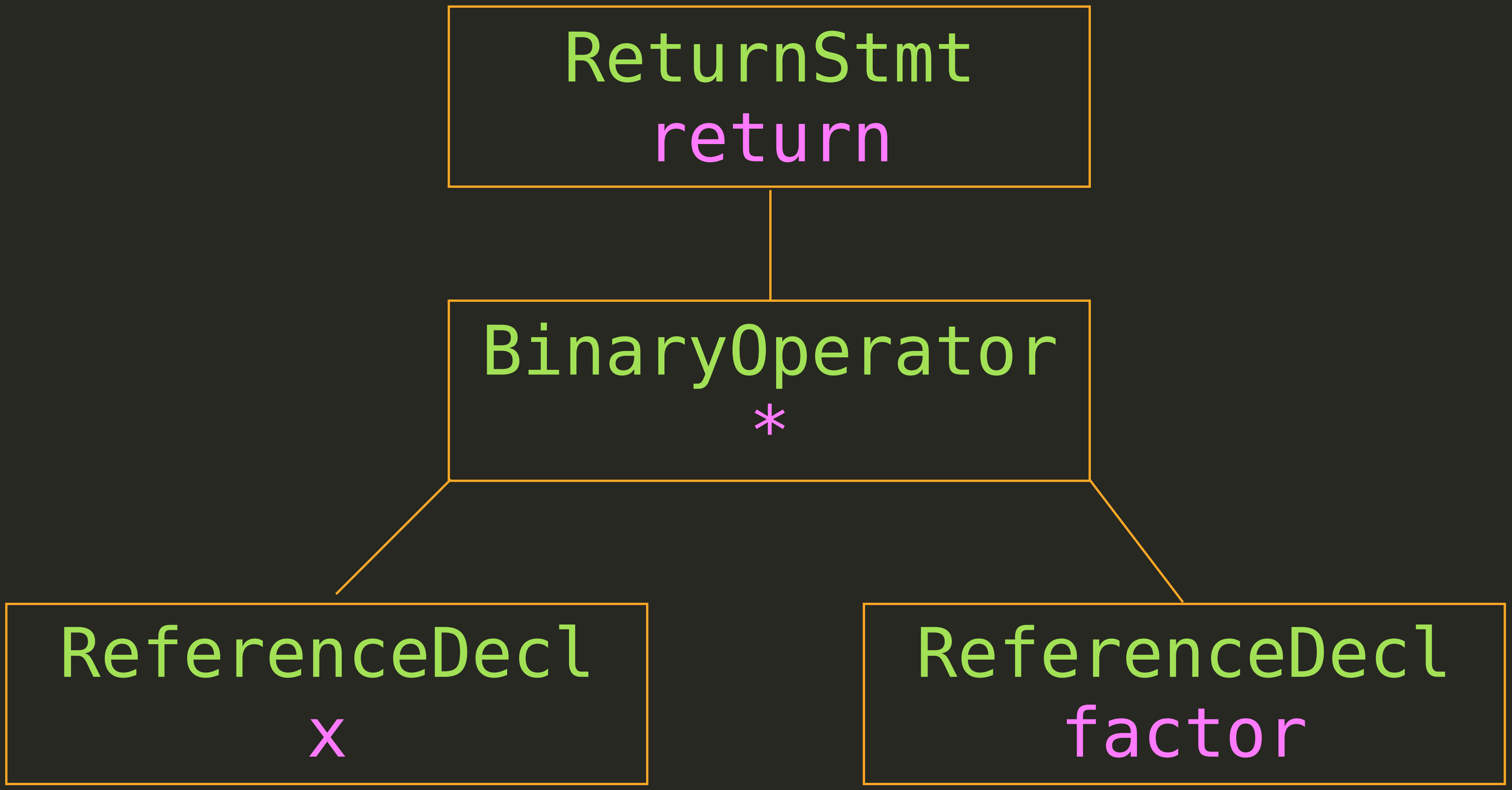
↓

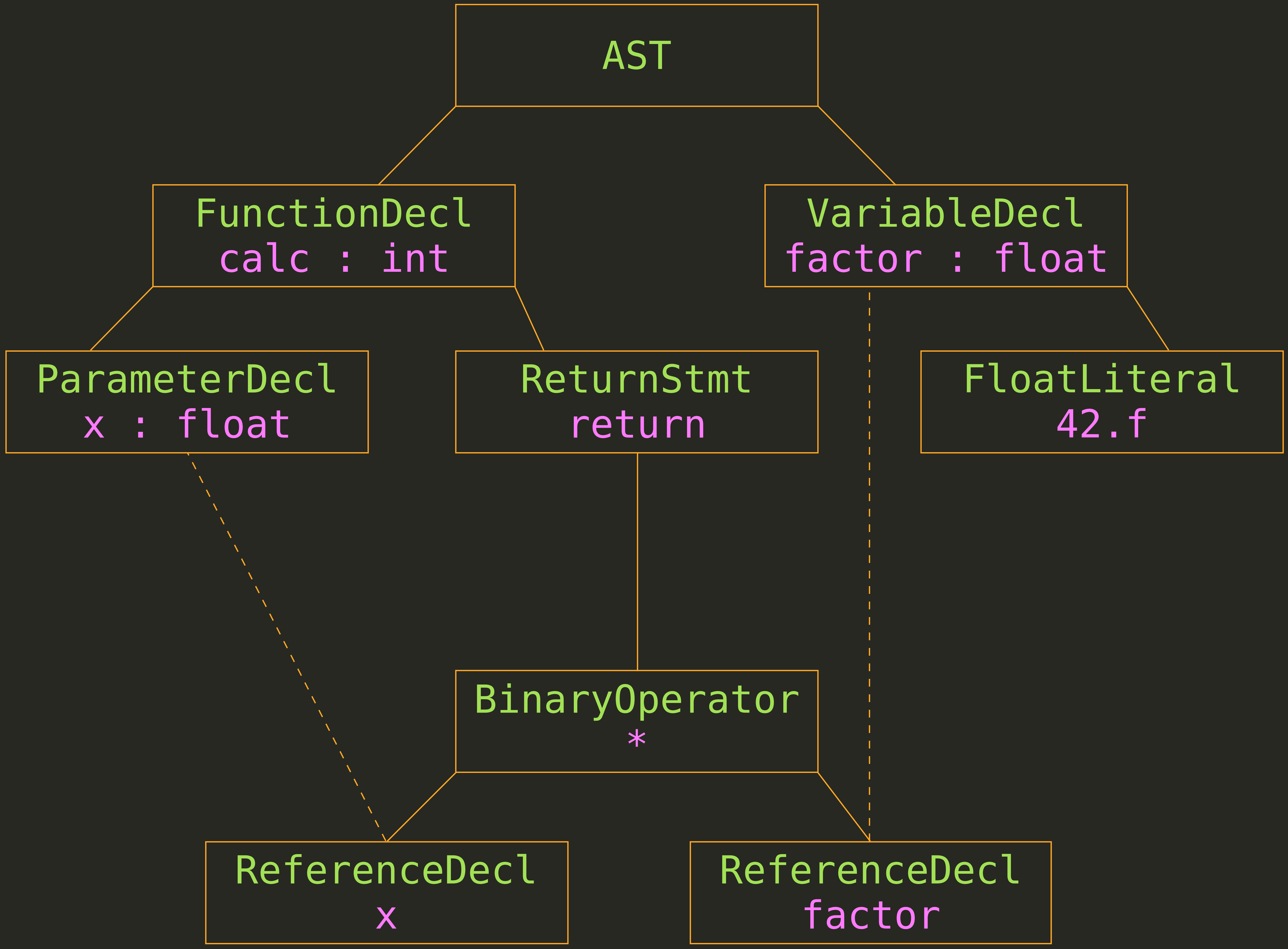
```
(KW ‘const’), (TYPE ‘float’), (ID ‘factor'),
(EQ ‘=‘), (NUM ’42.f’), (SEMI ‘;’), (TYPE ‘int’),
(ID ‘calc’), (L_PAREN ‘(‘), (TYPE ‘float’), (ID 'x')
(R_PAREN ‘)’), (L_BRACE ‘{‘), (KW ‘return’),
(ID ‘factor’), (STAR ‘*’), (ID ‘x’), (SEMI ‘;’),
(R_BRACE ‘}’), (EOF '')
```

Parser

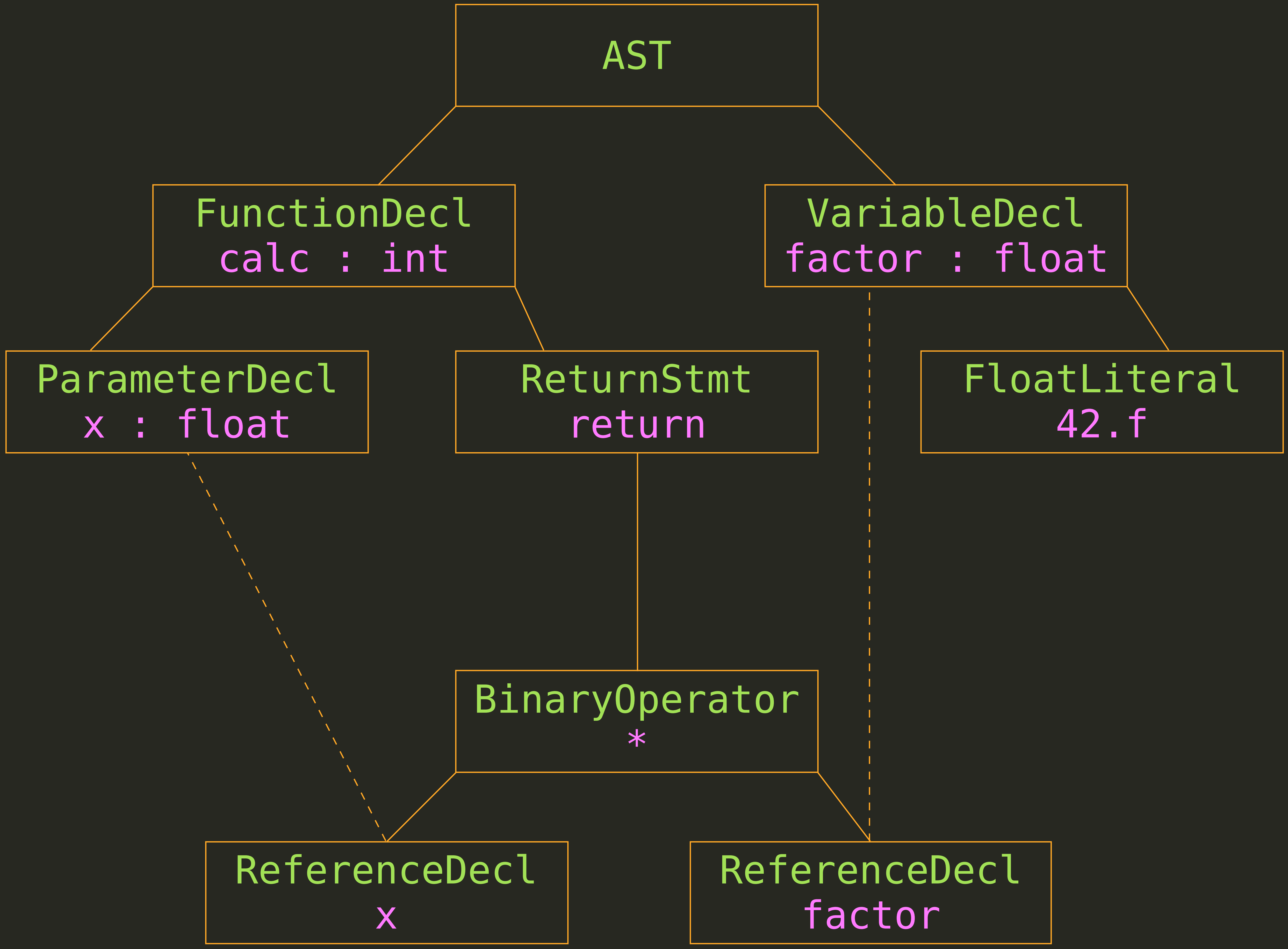
```
(KW ‘return’) (ID ‘factor’) (STAR ‘*’) (ID ‘x’)
```

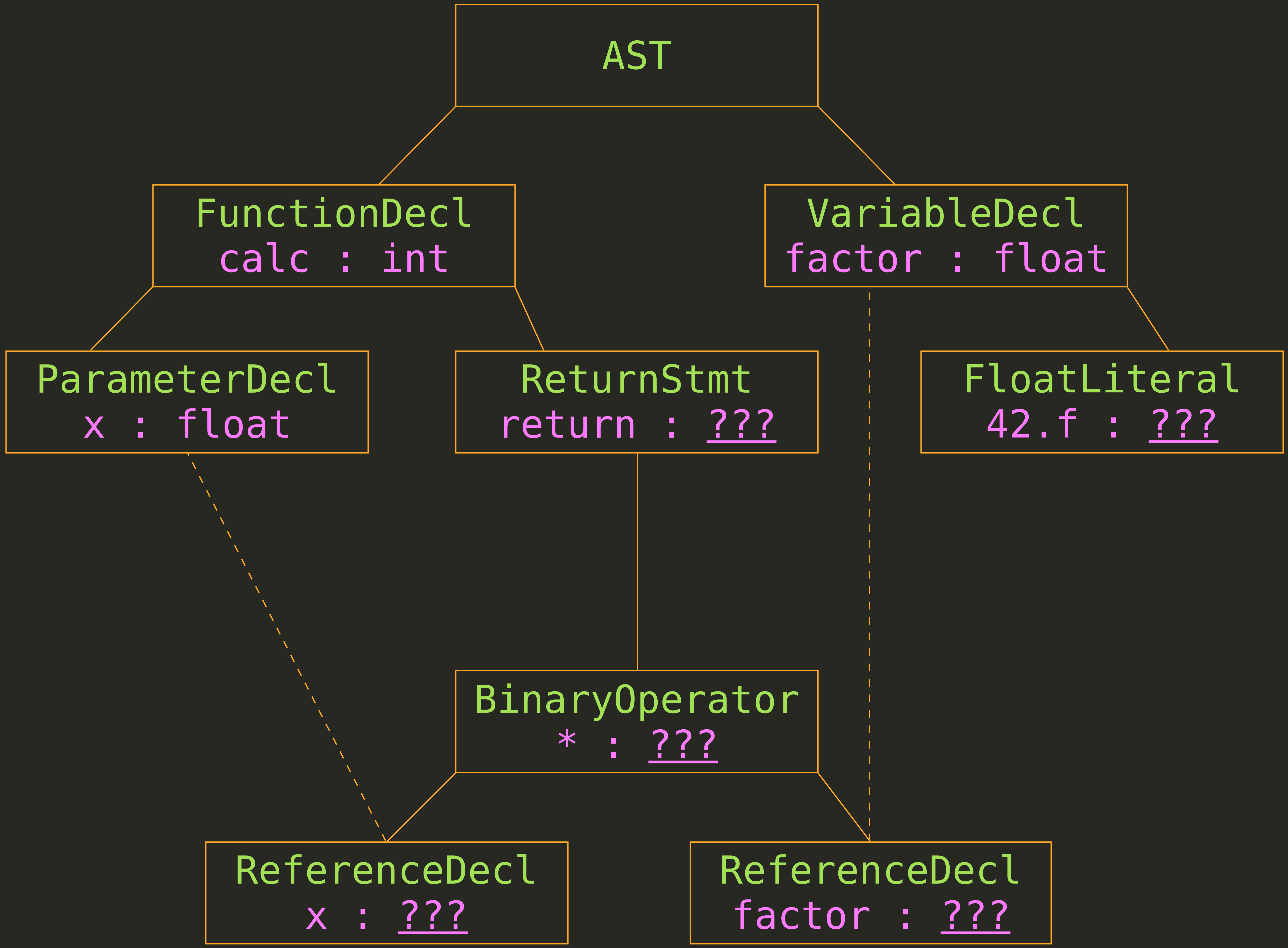
```
(KW ‘return’) (ID ‘factor’) (STAR ‘*’) (ID ‘x’)
```

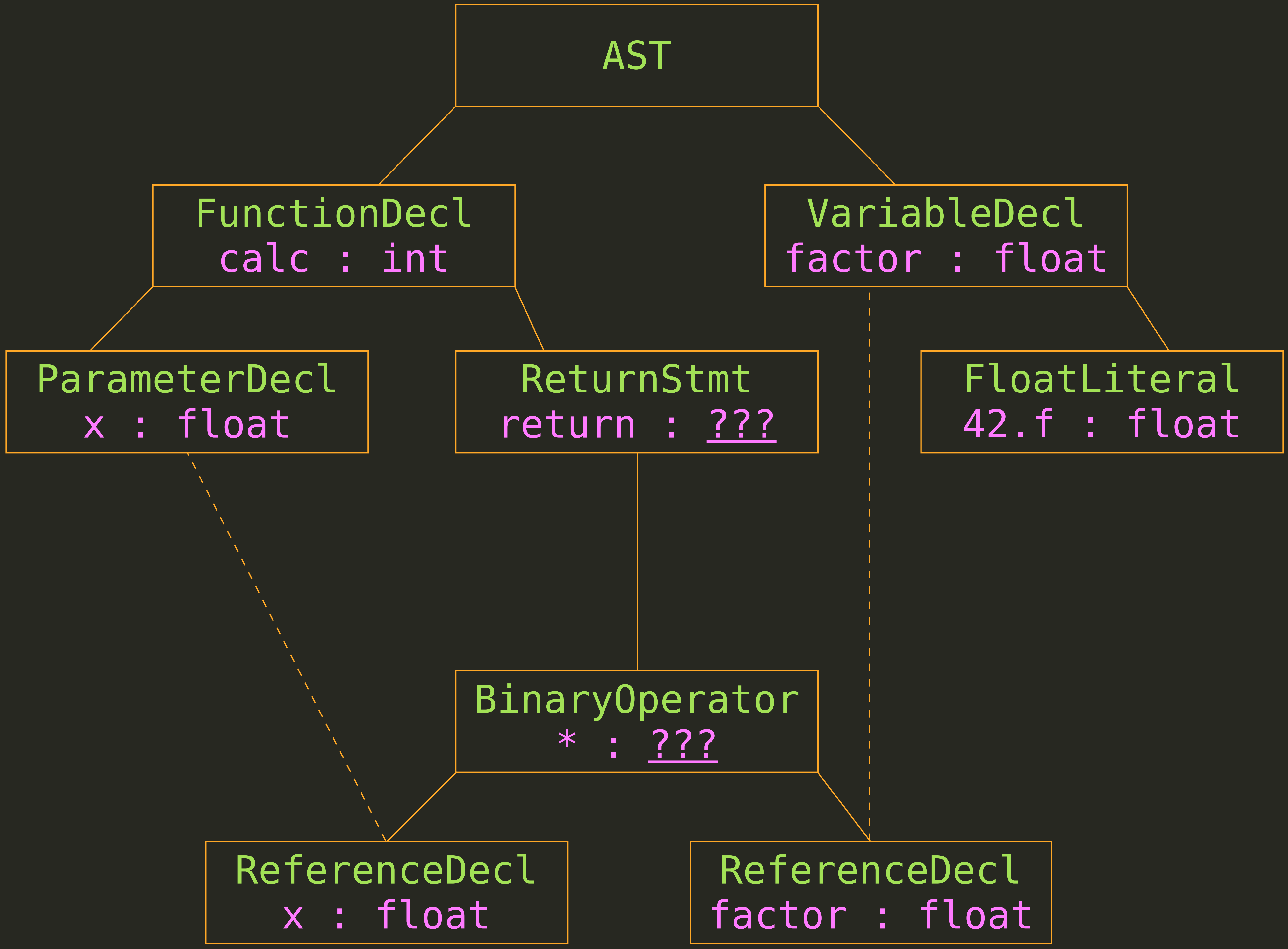


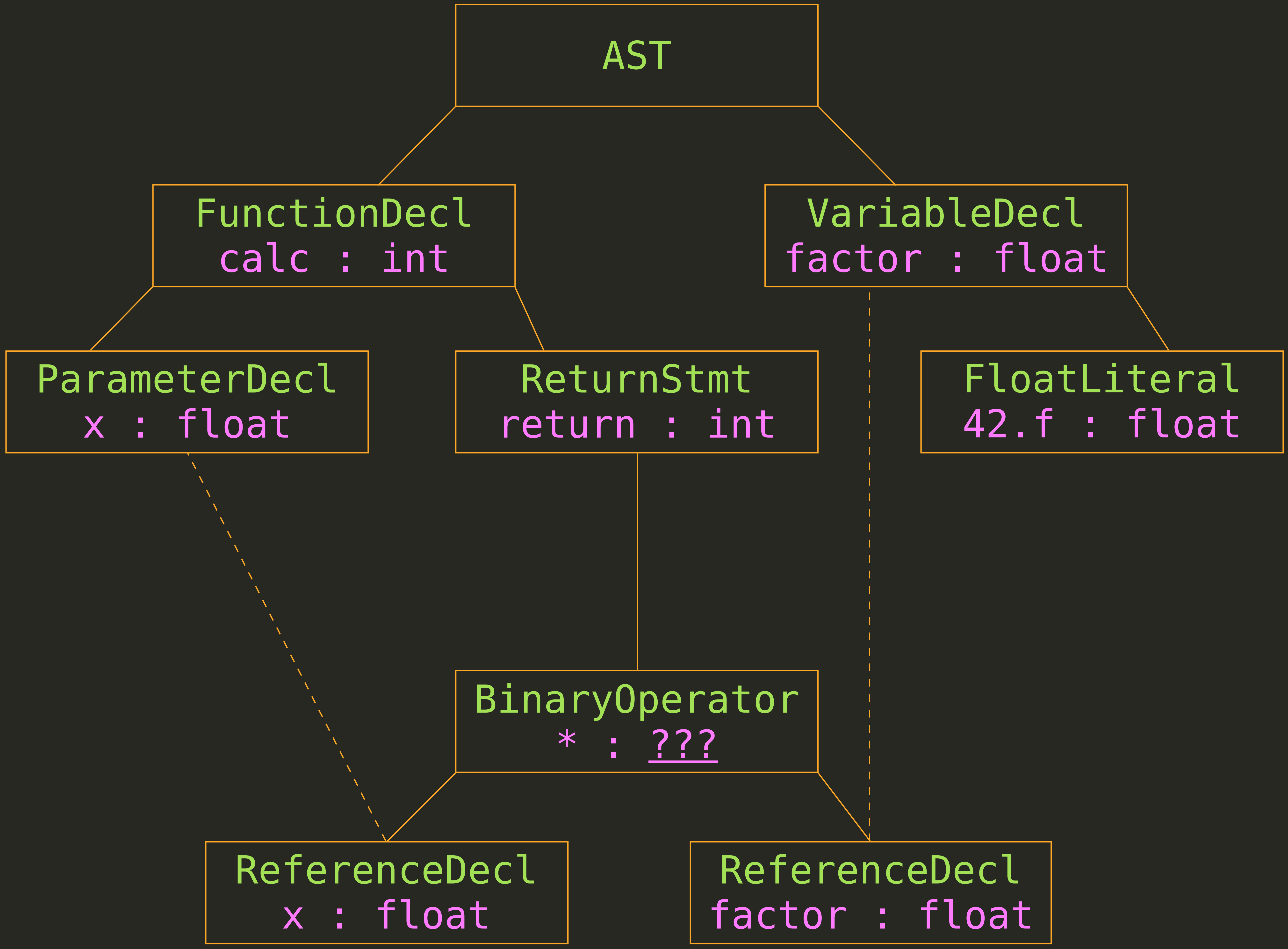

AST
FunctionDecl
calc : int
VariableDecl
factor : float
ParameterDecl
x : float
ReturnStmt
return
FloatLiteral
42.f
BinaryOperator
*
ReferenceDecl
x
ReferenceDecl
factor

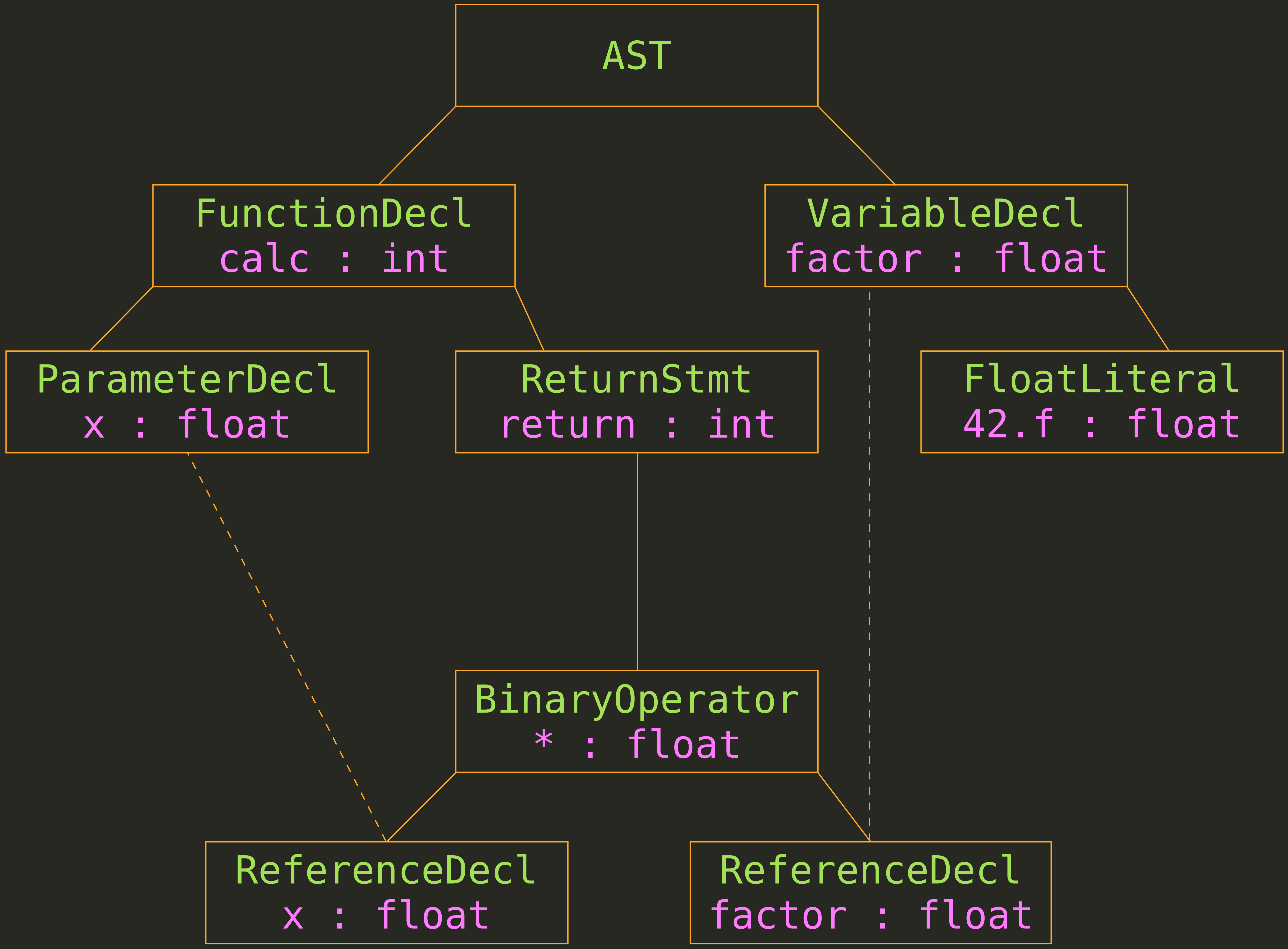
Semantic Analysis

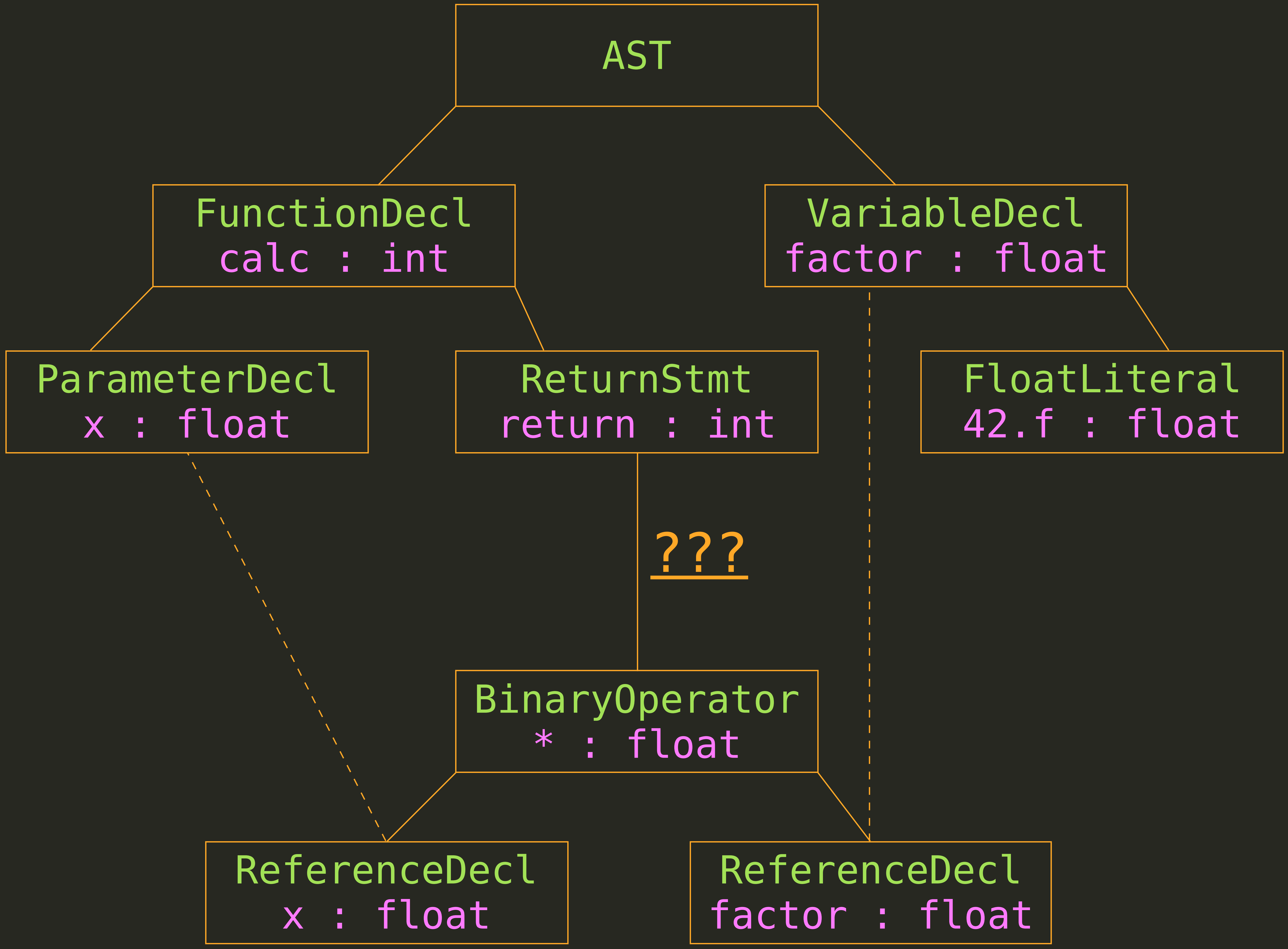

AST
FunctionDecl
calc : int
VariableDecl
factor : float
ParameterDecl
x : float
ReturnStmt
return
FloatLiteral
42.f
BinaryOperator
*
ReferenceDecl
x
ReferenceDecl
factor

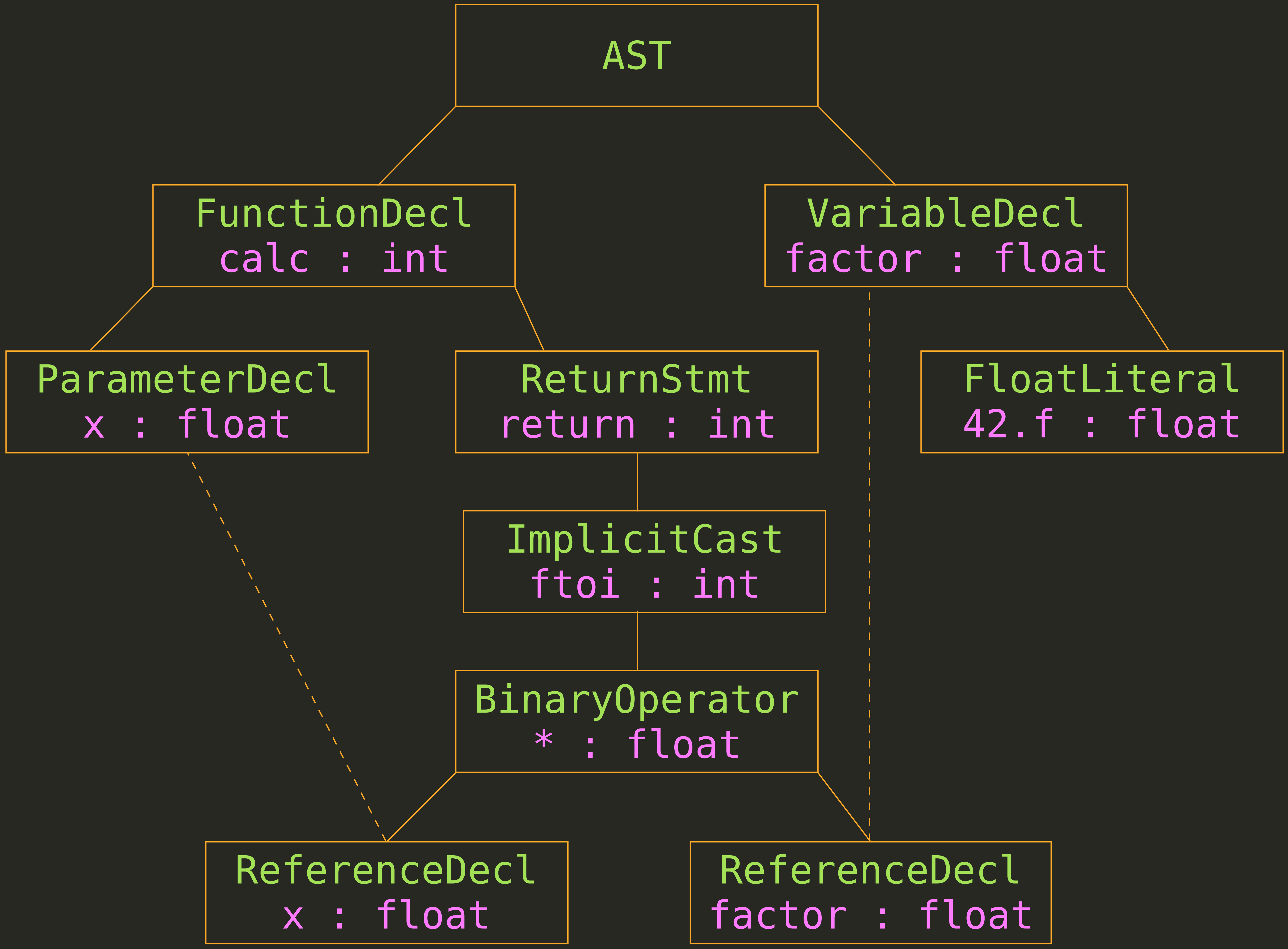

AST
FunctionDecl
calc : int
VariableDecl
factor : float
ParameterDecl
x : float
ReturnStmt
return : ???
FloatLiteral
42.f : ???
BinaryOperator
* : ???
ReferenceDecl
x : ???
ReferenceDecl
factor : ???


AST
FunctionDecl
calc : int
VariableDecl
factor : float
ParameterDecl
x : float
ReturnStmt
return : ???
FloatLiteral
42.f : float
BinaryOperator
* : ???
ReferenceDecl
x : float
ReferenceDecl
factor : float


AST
FunctionDecl
calc : int
VariableDecl
factor : float
ParameterDecl
x : float
ReturnStmt
return : int
FloatLiteral
42.f : float
BinaryOperator
* : ???
ReferenceDecl
x : float
ReferenceDecl
factor : float


AST
FunctionDecl
calc : int
VariableDecl
factor : float
ParameterDecl
x : float
ReturnStmt
return : int
FloatLiteral
42.f : float
BinaryOperator
* : float
ReferenceDecl
x : float
ReferenceDecl
factor : float


AST
FunctionDecl
calc : int
VariableDecl
factor : float
ParameterDecl
x : float
ReturnStmt
return : int
FloatLiteral
42.f : float
???
BinaryOperator
* : float
ReferenceDecl
x : float
ReferenceDecl
factor : float


AST
FunctionDecl
calc : int
VariableDecl
factor : float
ParameterDecl
x : float
ReturnStmt
return : int
FloatLiteral
42.f : float
ImplicitCast
ftoi : int
BinaryOperator
* : float
ReferenceDecl
x : float
ReferenceDecl
factor : float

Code Generation

```
@factor = constant float 42.0

define calc(float %x) {
entry:
  movf %x, %r1
  movf @factor, %r2
  %r3 = fmul %r1, %r2
  movf %r3, %r0
  ret
}
```

Optimization

```
@factor = constant float 42.0

define calc(float %x) {
entry:
  movf %x, %r1
  movf @factor, %r2
  %r3 = fmul %r1, %r2
  movf %r3, %r0
  ret
}
```

```
@factor = constant float 42.0

define calc(float %x) {
entry:
  %r0 = fmul @factor, %x
  ret
}
```

Assembler

```
_calc:
    push {r7, lr}
    mov  r7, sp
    mov  r1, #36175872
    orr  r1, r1, #1073741824
    bl ___mulsf3
    bl ___fixsfsi
    pop  {r7, lr}
    mov  pc, lr

    .section  __TEXT,__const
    .globl _factor @ @factor
    .align 2
_factor:
    .long  1109917696 @ float 42
```

Linker

```c
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

```
> clang -c calc.c -o calc.o
```

```
extern int calc(float);

int main() {
  printf("%d\n", calc(2.f));
  return 0;
}
```

```
  > clang -c main.c -o main.o
```

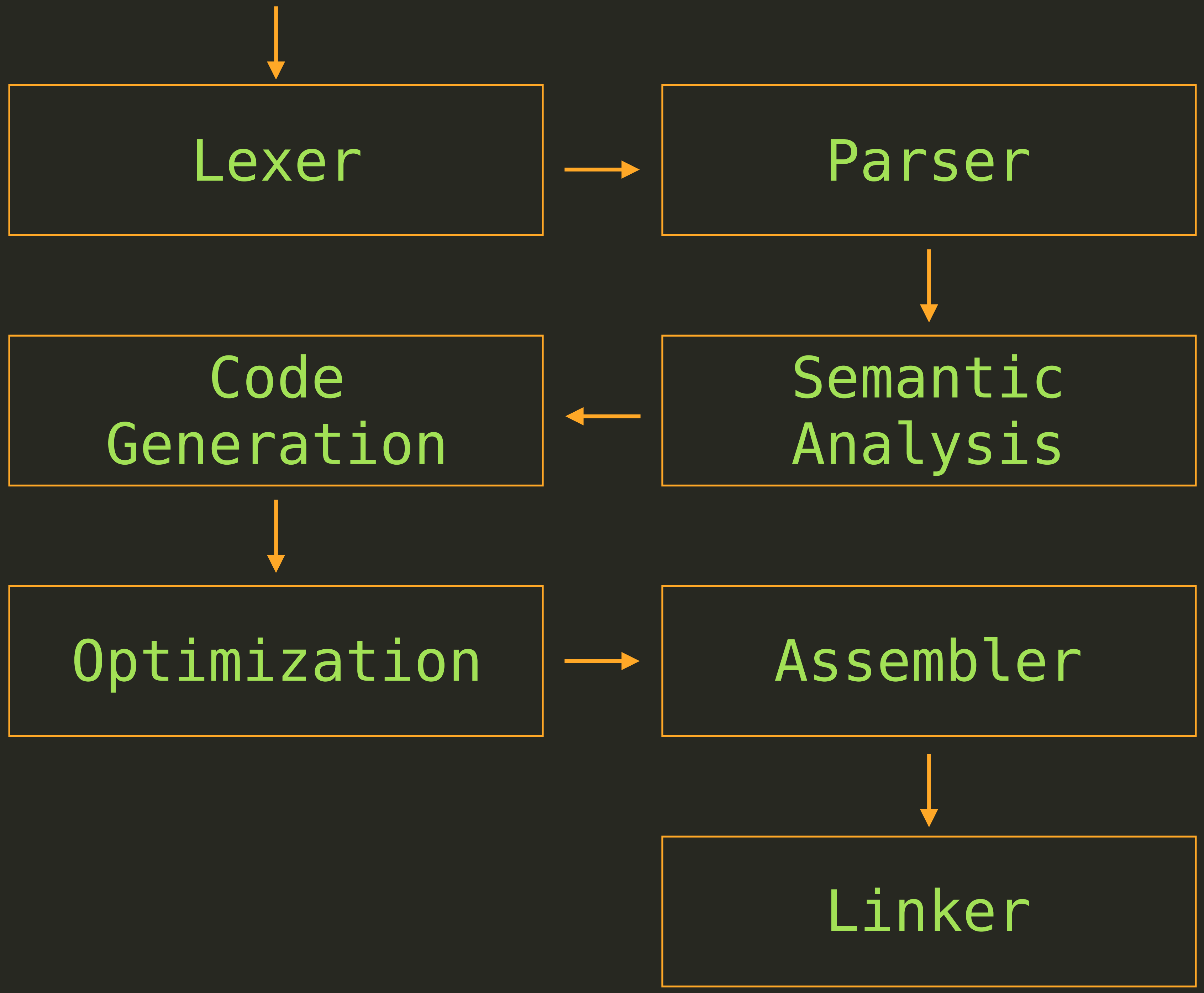
https://github.com/AlexDenisov/mbx/blob/master/main.c

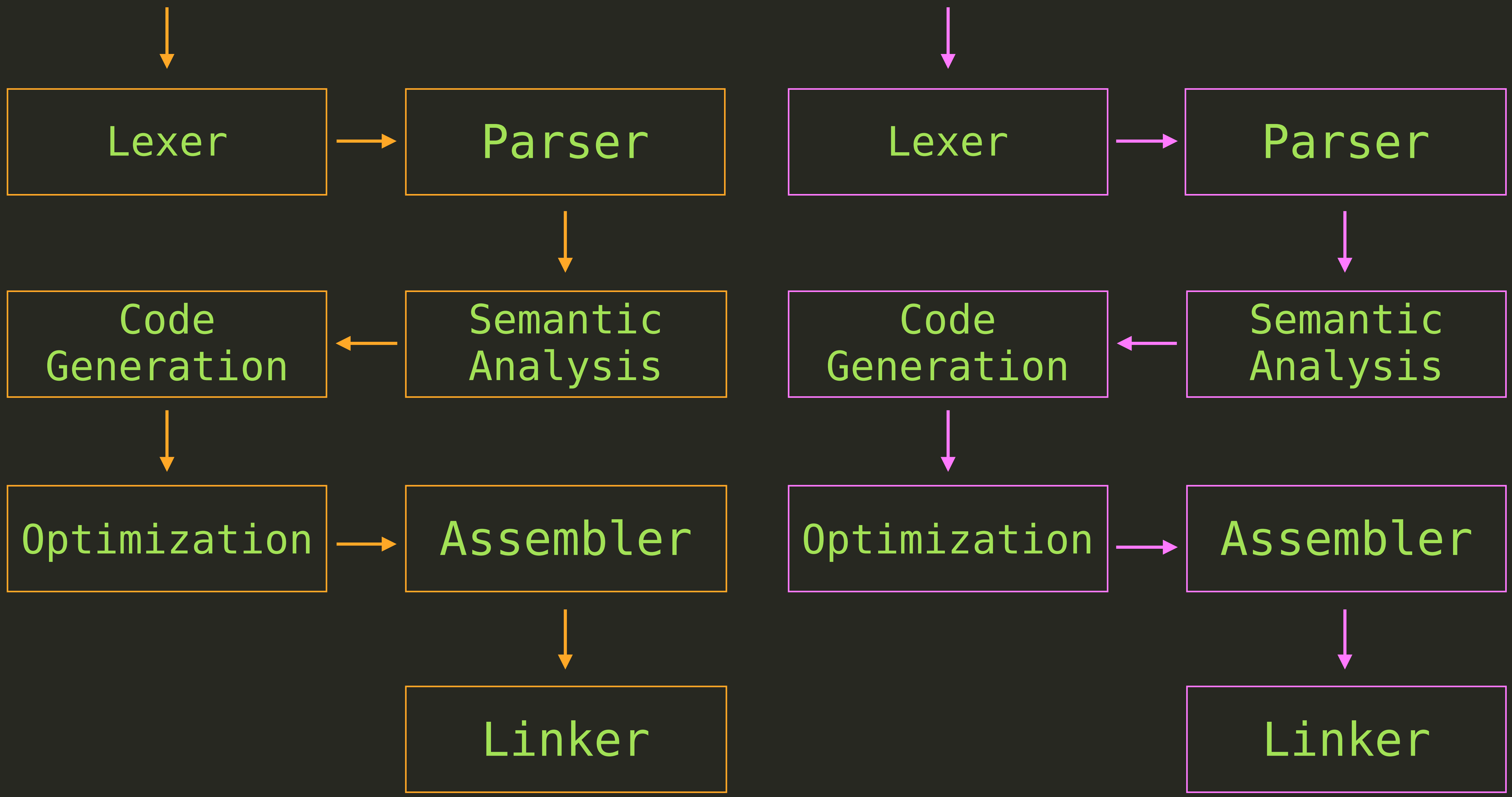
```
> nm main.o
                 U _calc
0000000000000000 T _main
                 U _printf
```

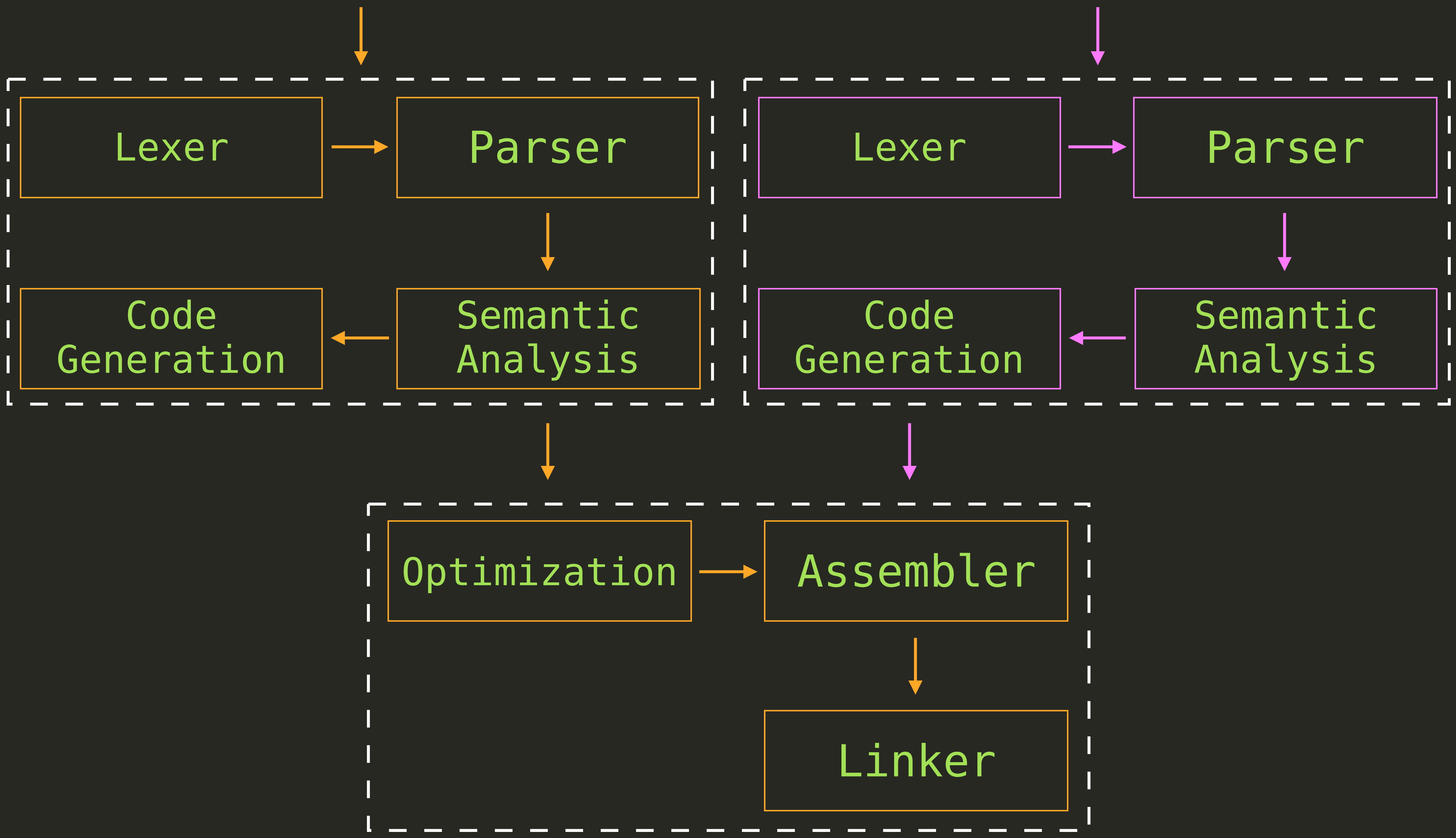
```
> nm main.o
                 U _calc
0000000000000000 T _main
                 U _printf

> ld -lc calc.o main.o -o main

> nm main
0000000000001f30 T _calc
0000000000001fc8 S _factor
0000000000001f60 T _main
                 U _printf
```

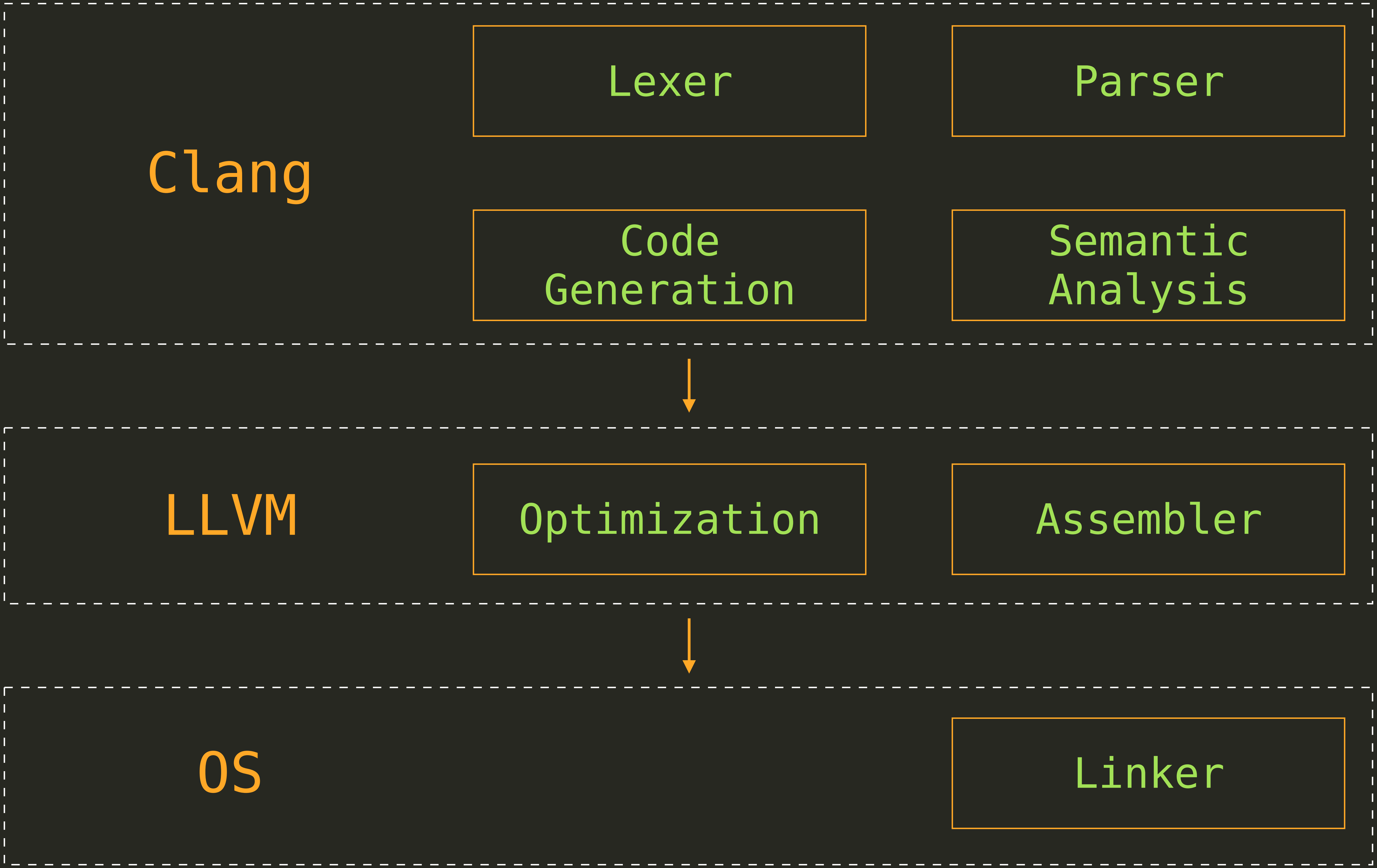
LLVM & Clang

LLVM & Clang & Swift


Lexer
Parser
Semantic Analysis
Code Generation
Optimization
Assembler
Linker


Lexer
Parser
Semantic Analysis
Code Generation
Optimization
Assembler
Linker
Lexer
Parser
Semantic Analysis
Code Generation
Optimization
Assembler
Linker


Lexer
Parser
Code Generation
Semantic Analysis
Lexer
Parser
Code Generation
Semantic Analysis
Optimization
Assembler
Linker


Clang
Lexer
Parser
Code Generation
Semantic Analysis
LLVM
Optimization
Assembler
OS
Linker

Tokens

```c
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

```
> clang -cc1 -dump-tokens calc.c
```

```
const 'const'    [StartOfLine]  Loc=<calc.c:1:1>
float 'float'    [LeadingSpace] Loc=<calc.c:1:7>
identifier 'factor' [LeadingSpace] Loc=<calc.c:1:13>
equal '='  [LeadingSpace] Loc=<calc.c:1:20>
numeric_constant '42.f'    [LeadingSpace] Loc=<calc.c:1:22>
semi ';'     Loc=<calc.c:1:26>
int 'int'  [StartOfLine]  Loc=<calc.c:3:1>
identifier 'calc'   [LeadingSpace] Loc=<calc.c:3:5>
l_paren '('     Loc=<calc.c:3:9>
float 'float'       Loc=<calc.c:3:10>
identifier 'x'   [LeadingSpace] Loc=<calc.c:3:16>
r_paren ')'     Loc=<calc.c:3:17>
l_brace '{'   [LeadingSpace] Loc=<calc.c:3:19>
return 'return'  [StartOfLine] [LeadingSpace]   Loc=<calc.c:4:3>
identifier 'factor' [LeadingSpace] Loc=<calc.c:4:10>
star '*'   [LeadingSpace] Loc=<calc.c:4:17>
identifier 'x'   [LeadingSpace] Loc=<calc.c:4:19>
semi ';'     Loc=<calc.c:4:20>
r_brace '}'   [StartOfLine]  Loc=<calc.c:5:1>
eof ''    Loc=<calc.c:6:1>
```

AST

AST: Clang

```
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

```
> clang -cc1 -ast-dump calc.c
```

```
TranslationUnitDecl <<invalid sloc>> <invalid sloc>
|-VarDecl <calc.c:1:1, col:22> col:13 used factor 'const float' cinit
| `-FloatingLiteral <col:22> 'float' 4.200000e+01
`-FunctionDecl <line:3:1, line:5:1> line:3:5 calc 'int (float)'
  |-ParmVarDecl <col:10, col:16> col:16 used x 'float'
  `-CompoundStmt <col:19, line:5:1>
    `-ReturnStmt <line:4:3, col:19>
      `-ImplicitCastExpr <col:10, col:19> 'int' <FloatingToIntegral>
        `-BinaryOperator <col:10, col:19> 'float' '*'
          |-ImplicitCastExpr <col:10> 'float' <LValueToRValue>
          | `-DeclRefExpr <col:10> 'const float' lvalue Var 'factor' 'const float'
          `-ImplicitCastExpr <col:19> 'float' <LValueToRValue>
            `-DeclRefExpr <col:19> 'float' lvalue ParmVar 'x' 'float'
```

```objc
@class NSString;

__attribute__((objc_root_class))
@interface User

@property (copy) NSString *firstname;
@property (copy) NSString *lastname;

@end
```

```
auto index = clang_createIndex(0, 1);
auto tu = clang_parseTranslationUnit(index, “ast_clang.m");
auto block = ^ CXChildVisitResult (CXCursor cursor) {
 CXCursorKind cursor_kind = clang_getCursorKind(cursor);
 switch(cursor_kind) {
   case CXCursor_ObjCInterfaceDecl:{
     CXString cxname = clang_getCursorSpelling(cursor);
     printf("class: %s\n", clang_getCString(cxname));
     clang_disposeString(cxname);
    } break;
    case CXCursor_ObjCPropertyDecl:{
      CXString cxname = clang_getCursorSpelling(cursor);
      printf(" -> %s\n", clang_getCString(cxname));
      clang_disposeString(cxname);
    } break;
    default:break;
 }
 return CXChildVisit_Recurse;
};
auto cursor = clang_getTranslationUnitCursor(tu);
clang_visitChildrenWithBlock(cursor, block);
clang_disposeTranslationUnit(tu);
clang_disposeIndex(index);
```

https://github.com/AlexDenisov/mbx/blob/master/dump_classes_clang.c

```
> clang dump_classes.c -o dump_classes -lclang
> ./dump_classes

class: User
 -> firstname
 -> lastname
```

AST: Swift

```swift
let factor = 42.0

func calc(x: Double) -> Double {
  return x * factor
}
```

```
> swiftc -dump-ast calc.swift
```

```
(source_file
  (var_decl "factor" type='Double' access=internal let storage_kind=stored)
  (top_level_code_decl
    (brace_stmt
      (pattern_binding_decl
        (pattern_named type='Double' 'factor')
        (call_expr implicit type='Double' location=calc.swift:1:14 range=[calc.swift:1:14 - line:1:14]
          (constructor_ref_call_expr implicit type='(_builtinFloatLiteral: FPIEEE80) -> Double' location=calc.swift:1:14 range=[calc.swift:1:14 - line:1:14]
            (declref_expr implicit type='Double.Type -> (_builtinFloatLiteral: FPIEEE80) -> Double' location=calc.swift:1:14 range=[calc.swift:1:14 - line:1:14]
decl=Swift.(file).Double.init(_builtinFloatLiteral:) specialized=no)
            (type_expr implicit type='Double.Type' location=calc.swift:1:14 range=[calc.swift:1:14 - line:1:14] typerepr='<<IMPLICIT>>'))
          (tuple_expr implicit type='(_builtinFloatLiteral: FPIEEE80)' location=calc.swift:1:14 range=[calc.swift:1:14 - line:1:14] names=_builtinFloatLiteral
            (float_literal_expr type='FPIEEE80' location=calc.swift:1:14 range=[calc.swift:1:14 - line:1:14] value=42.0))))
)
  (func_decl "calc(_:)" type='(Double) -> Double' access=internal captures=(factor)
    (body_params
      (pattern_tuple type='(Double)'
        (pattern_typed type='Double'
          (pattern_named type='Double' 'x')
          (type_ident
            (component id='Double' bind=type)))))
    (result
      (type_ident
        (component id='Double' bind=type)))
    (brace_stmt
      (return_stmt
        (binary_expr type='Double' location=calc.swift:4:12 range=[calc.swift:4:10 - line:4:14]
          (declref_expr type='(Double, Double) -> Double' location=calc.swift:4:12 range=[calc.swift:4:12 - line:4:12] decl=Swift.(file).* specialized=no)
          (tuple_expr type='(Double, Double)' location=calc.swift:4:10 range=[calc.swift:4:10 - line:4:14]
            (declref_expr type='Double' location=calc.swift:4:10 range=[calc.swift:4:10 - line:4:10] decl=calc.(file).func decl.x@calc.swift:3:11 specialized=no)
            (declref_expr type='Double' location=calc.swift:4:14 range=[calc.swift:4:14 - line:4:14] decl=calc.(file).factor@calc.swift:1:5 direct_to_storage
specialized=no)))))))
```

```
(var_decl "factor" type='Double'
          access=internal let storage_kind=stored)
```

S-Expression

```
(var_decl "factor" type='Double'
          access=internal let storage_kind=stored)
```

http://rosettacode.org/wiki/S-Expressions

S-Expression
(almost)

```
(var_decl "factor" type='Double'
          access=internal let storage_kind=stored)
```

http://rosettacode.org/wiki/S-Expressions

```
class String {}

class User {
  var firstname : String?
  var lastname : String?
}
```

```ruby
ast = SXP.read File.read “swift.ast"

def properties class_def
  class_def.map do |node|
    node[1] if node[0] == :var_decl
  end.compact
end

ast.each do |node|
  next unless node[0] == :class_decl
  puts "class: #{node[1]}"
  properties(node).each do |p|
    puts " -> #{p}"
  end
end
```

https://github.com/AlexDenisov/mbx/blob/master/dump_classes_swift.rb

```
> swiftc -dump-ast user.swift > swift.ast
> gem install sxp
> ruby dump_classes.rb

class: String
class: User
 -> firstname
 -> lastname
```

LLVM IR

```c
const float factor = 42.f;

int calc(float x) {
    return factor * x;
}
```

```
> clang -S -emit-llvm calc.c
```

```
@factor = constant float 4.200000e+01, align 4

define i32 @calc(float %x) #0 {
entry:
  %x.addr = alloca float, align 4
  store float %x, float* %x.addr, align 4
  %0 = load float* %x.addr, align 4
  %mul = fmul float 4.200000e+01, %0
  %conv = fptosi float %mul to i32
  ret i32 %conv
}
```

https://github.com/AlexDenisov/mbx/blob/master/calc_clang.ll

```swift
let factor = 42.0

func calc(x: Double) -> Double {
  return x * factor
}
```

```
 > swiftc -emit-ir calc.swift
```

```
@_Tv4calc6factorSd = global double zeroinitializer, align 8

define i32 @main(i32, i8**) {
entry:
  store double 4.200000e+01, double* getelementptr inbounds
        (double* @_Tv4calc6factorSd, i32 0, i32 0), align 8
  ret i32 0
}

define hidden double @_TF4calc4calcFSdSd(double) {
entry:
  %1 = load double* getelementptr inbounds
        (double* @_Tv4calc6factorSd, i32 0, i32 0), align 8
  %2 = fmul double %0, %1
  ret double %2
}
```

https://github.com/AlexDenisov/mbx/blob/master/calc_swift.ll

Summary

Summary

- Learn your tools

Summary

- Learn your tools
- Provide feedback, don’t make complaints

Summary

- Learn your tools
- Provide feedback, don’t make complaints
- Give back to community

What's next?

Clang:
http://www.objc.io/issues/6-build-tools/compiler

LLVM:
http://aosabook.org/en/llvm.html

libclang:
https://www.mikeash.com/pyblog/friday-qa-2014-01-24-introduction-to-libclang.html

New ObjectiveC feature:
http://lowlevelbits.org/nsvalue-and-boxed-expressions/

What's next?

Slides:

https://speakerdeck.com/alexdenisov/magic-behind-xcode

Supplementary material:

https://github.com/AlexDenisov/mbx

'Hidden' gems:

```
> swiftc -help-hidden
> clang -help-hidden
```

Questions?

Twitter:
@1101_debian

Blog:
http://lowlevelbits.org