中山大學 SUN YAT-SEN UNIVERSITY 计算机学院（软件学院） SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compiler Design
编 译 器 构 造 实 验

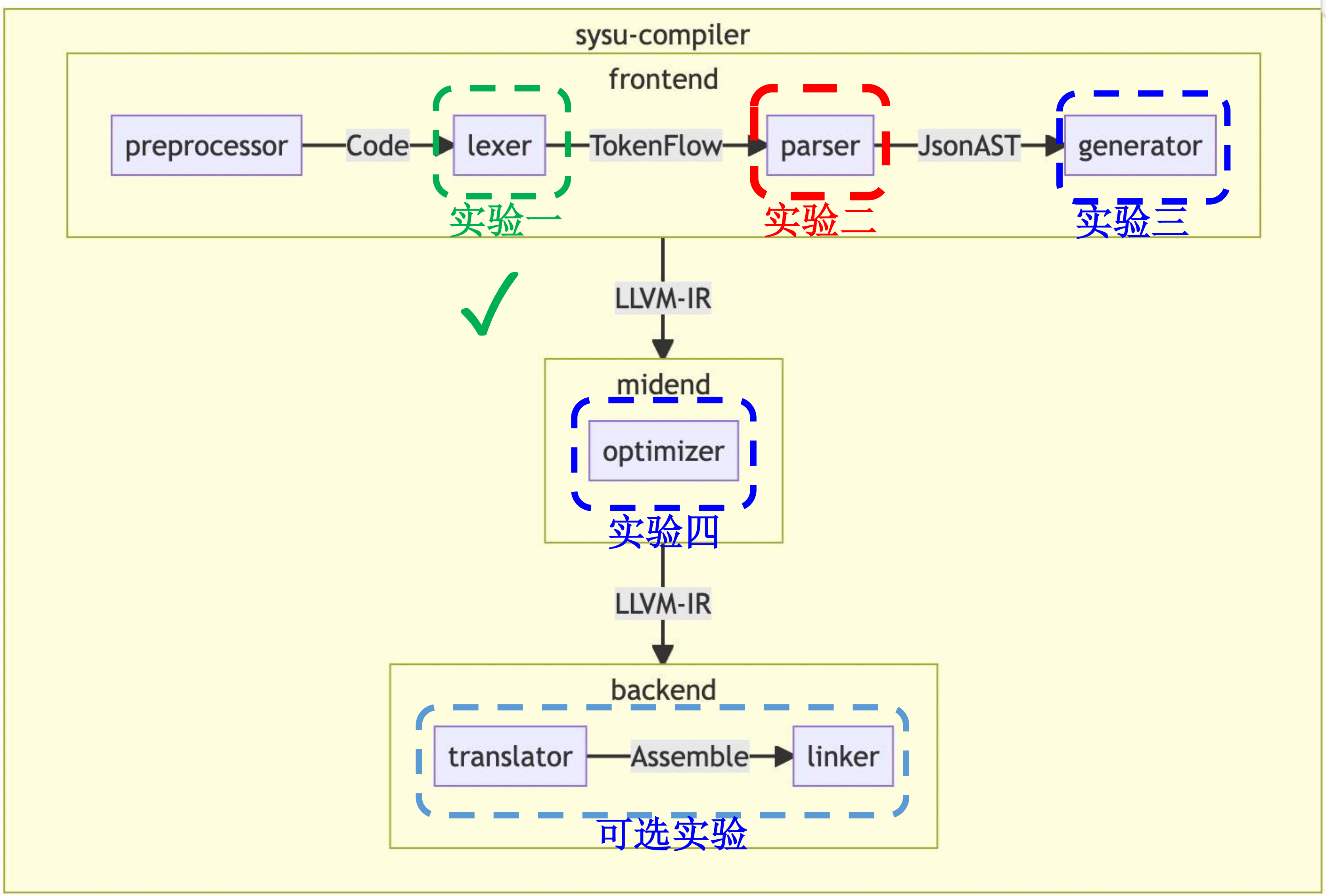
Lab 4： Project-2

张献伟

xianweiz.github.io

DCS292, 3/14/2023

Schedule[实验安排]



Project 2: What?

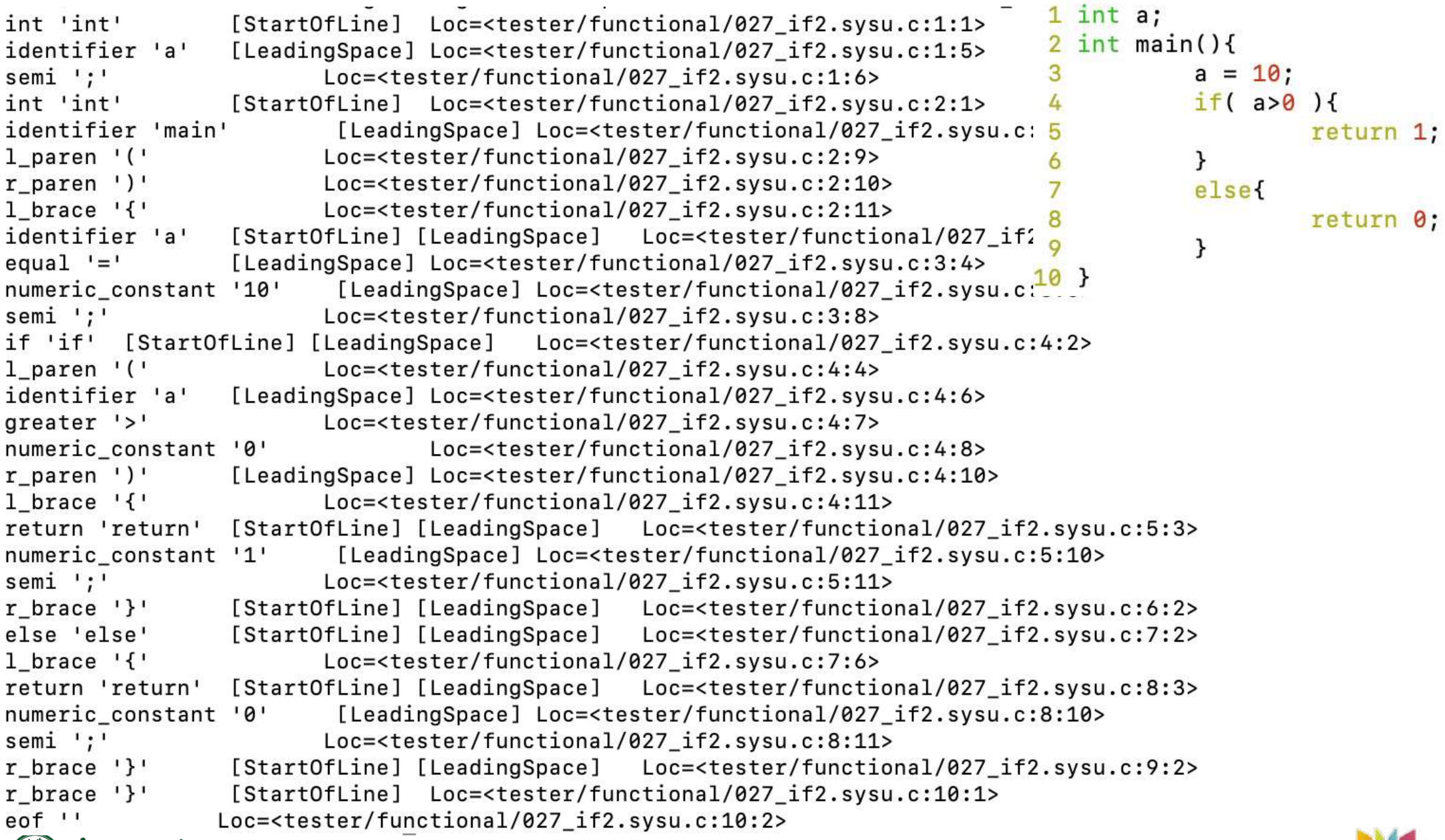
- 文档描述：
  - Readme: https://github.com/arcsysu/SYsU-lang/tree/main/parser
  - Wiki: https://github.com/arcsysu/SYsU-lang/wiki/%E5%AE%9E%E9%AA%8C%E4%BA%8C%E8%AF%AD%E6%B3%95%E5%88%86%E6%9E%90
- 基于YACC/Bison实现一个语法分析器
  - 输入：token序列（由Project 1或Clang提供）
  - 输出：抽象语法树（类似Clang AST）
- 总体流程
  - 引入Project1的lexer.l（可能需要简单修改）
  - 理解SYsU语言语法，构建上下文无关文法（CFG）规则
  - 使用YACC/Bison表示CFG文法
  - 提供语义动作，逐步构建分析树
- 截止时间
  - **4/18/2023**

Project 2: How?

- 实现
  - $vim parser/parser.y
  - $vim <*其他辅助文件*>
- 编译
  - $cmake --build ~/sysu/build -t install
    - 输出：~/sysu/build/parser
- 运行
  - $(export PATH=~/sysu/bin:$PATH \ CPATH=~/sysu/include:$CPATH \ LD_LIBRARY_PATH=~/sysu/lib:$LD_LIBRARY_PATH && sysu-preprocessor tester/functional/000_main.sysu.c | <THE_LEXER>| sysu-parser)
    - Clang提供token：<THE_LEXER> = clang -cc1 -dump-tokens 2>&1
    - Project1提供token： <THE_LEXER> = sysu-lexer

Clang Tokens

- $clang -cc1 -dump-tokens tester/functional/027_if2.sysu.c

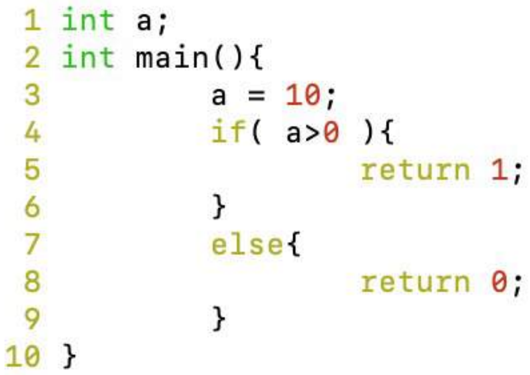


```
int 'int'        [StartOfLine]  Loc=<tester/functional/027_if2.sysu.c:1:1>
identifier 'a'   [LeadingSpace] Loc=<tester/functional/027_if2.sysu.c:1:5>
semi ';'                Loc=<tester/functional/027_if2.sysu.c:1:6>
int 'int'        [StartOfLine]  Loc=<tester/functional/027_if2.sysu.c:2:1>
identifier 'main'        [LeadingSpace] Loc=<tester/functional/027_if2.sysu.c:
l_paren '('             Loc=<tester/functional/027_if2.sysu.c:2:9>
r_paren ')'             Loc=<tester/functional/027_if2.sysu.c:2:10>
l_brace '{'             Loc=<tester/functional/027_if2.sysu.c:2:11>
identifier 'a'   [StartOfLine] [LeadingSpace]   Loc=<tester/functional/027_if2
equal '='        [LeadingSpace] Loc=<tester/functional/027_if2.sysu.c:3:4>
numeric_constant '10'    [LeadingSpace] Loc=<tester/functional/027_if2.sysu.c:
semi ';'                Loc=<tester/functional/027_if2.sysu.c:3:8>
if 'if'  [StartOfLine] [LeadingSpace]   Loc=<tester/functional/027_if2.sysu.c:4:2>
l_paren '('             Loc=<tester/functional/027_if2.sysu.c:4:4>
identifier 'a'   [LeadingSpace] Loc=<tester/functional/027_if2.sysu.c:4:6>
greater '>'             Loc=<tester/functional/027_if2.sysu.c:4:7>
numeric_constant '0'            Loc=<tester/functional/027_if2.sysu.c:4:8>
r_paren ')'      [LeadingSpace] Loc=<tester/functional/027_if2.sysu.c:4:10>
l_brace '{'             Loc=<tester/functional/027_if2.sysu.c:4:11>
return 'return'  [StartOfLine] [LeadingSpace]   Loc=<tester/functional/027_if2.sysu.c:5:3>
numeric_constant '1'     [LeadingSpace] Loc=<tester/functional/027_if2.sysu.c:5:10>
semi ';'                Loc=<tester/functional/027_if2.sysu.c:5:11>
r_brace '}'      [StartOfLine] [LeadingSpace]   Loc=<tester/functional/027_if2.sysu.c:6:2>
else 'else'      [StartOfLine] [LeadingSpace]   Loc=<tester/functional/027_if2.sysu.c:7:2>
l_brace '{'             Loc=<tester/functional/027_if2.sysu.c:7:6>
return 'return'  [StartOfLine] [LeadingSpace]   Loc=<tester/functional/027_if2.sysu.c:8:3>
numeric_constant '0'     [LeadingSpace] Loc=<tester/functional/027_if2.sysu.c:8:10>
semi ';'                Loc=<tester/functional/027_if2.sysu.c:8:11>
r_brace '}'      [StartOfLine] [LeadingSpace]   Loc=<tester/functional/027_if2.sysu.c:9:2>
r_brace '}'      [StartOfLine]  Loc=<tester/functional/027_if2.sysu.c:10:1>
eof ''          Loc=<tester/functional/027_if2.sysu.c:10:2>
```

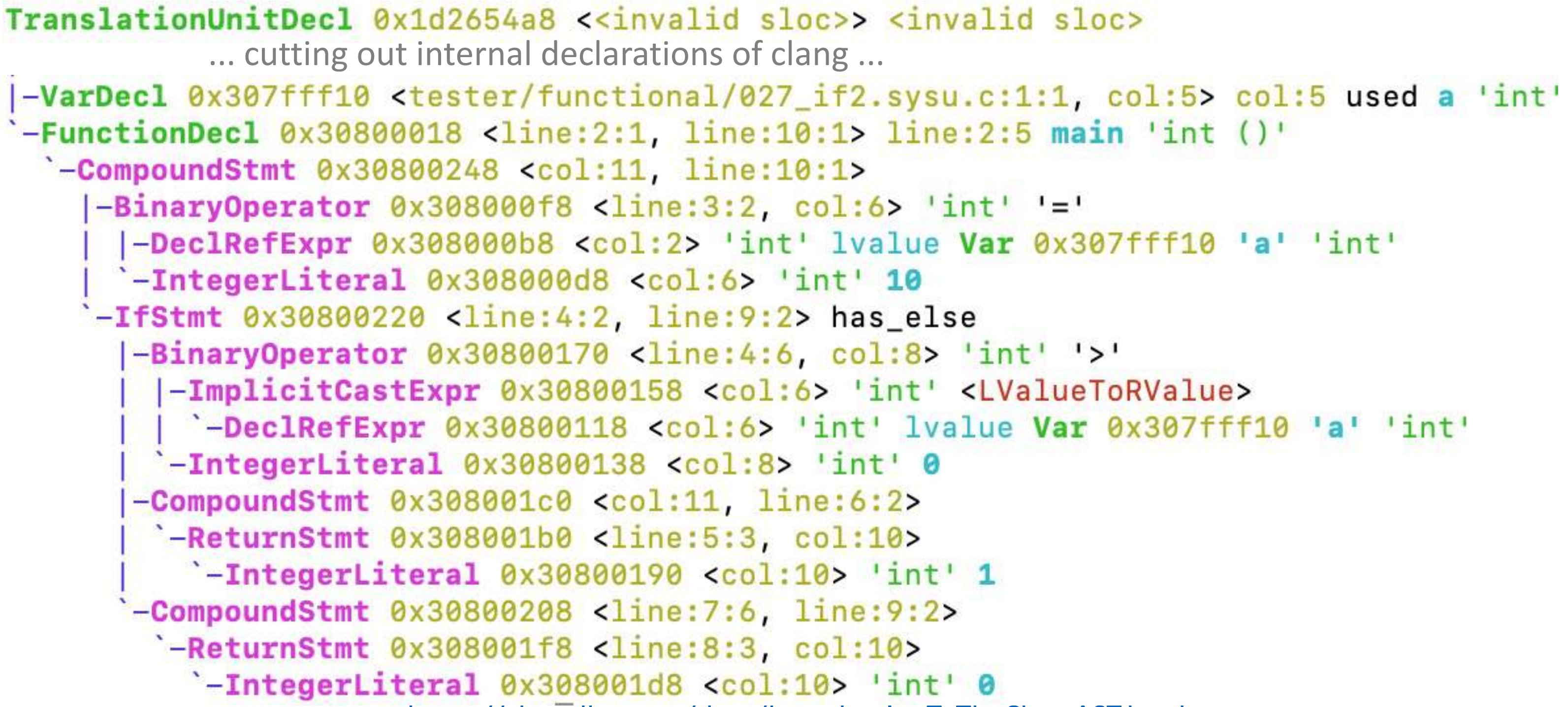
```
int a;
int main(){
	a = 10;
	if( a>0 ){
		return 1;
	}
	else{
		return 0;
	}
}
```

Clang AST

- $clang -Xclang -ast-dump -fsyntax-only tester/functional/027_if2.sysu.c



```
int a;
int main(){
        a = 10;
        if( a>0 ){
                return 1;
        }
        else{
                return 0;
        }
}
```

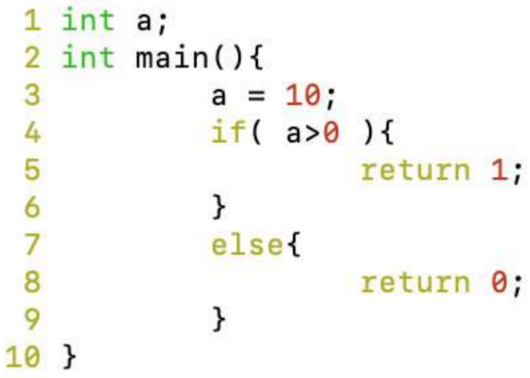


```
TranslationUnitDecl 0x1d2654a8 <<invalid sloc>> <invalid sloc>
          ... cutting out internal declarations of clang ...
|-VarDecl 0x307fff10 <tester/functional/027_if2.sysu.c:1:1, col:5> col:5 used a 'int'
`-FunctionDecl 0x30800018 <line:2:1, line:10:1> line:2:5 main 'int ()'
  `-CompoundStmt 0x30800248 <col:11, line:10:1>
    |-BinaryOperator 0x308000f8 <line:3:2, col:6> 'int' '='
    | |-DeclRefExpr 0x308000b8 <col:2> 'int' lvalue Var 0x307fff10 'a' 'int'
    | `-IntegerLiteral 0x308000d8 <col:6> 'int' 10
    `-IfStmt 0x30800220 <line:4:2, line:9:2> has_else
      |-BinaryOperator 0x30800170 <line:4:6, col:8> 'int' '>'
      | |-ImplicitCastExpr 0x30800158 <col:6> 'int' <LValueToRValue>
      | | `-DeclRefExpr 0x30800118 <col:6> 'int' lvalue Var 0x307fff10 'a' 'int'
      | `-IntegerLiteral 0x30800138 <col:8> 'int' 0
      |-CompoundStmt 0x308001c0 <col:11, line:6:2>
      | `-ReturnStmt 0x308001b0 <line:5:3, col:10>
      |   `-IntegerLiteral 0x30800190 <col:10> 'int' 1
      `-CompoundStmt 0x30800208 <line:7:6, line:9:2>
        `-ReturnStmt 0x308001f8 <line:8:3, col:10>
          `-IntegerLiteral 0x308001d8 <col:10> 'int' 0
```

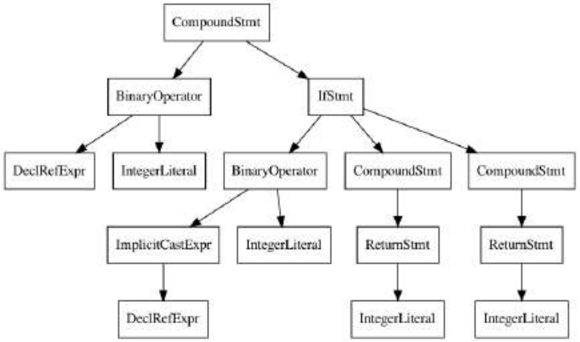
https://clang.llvm.org/docs/IntroductionToTheClangAST.html

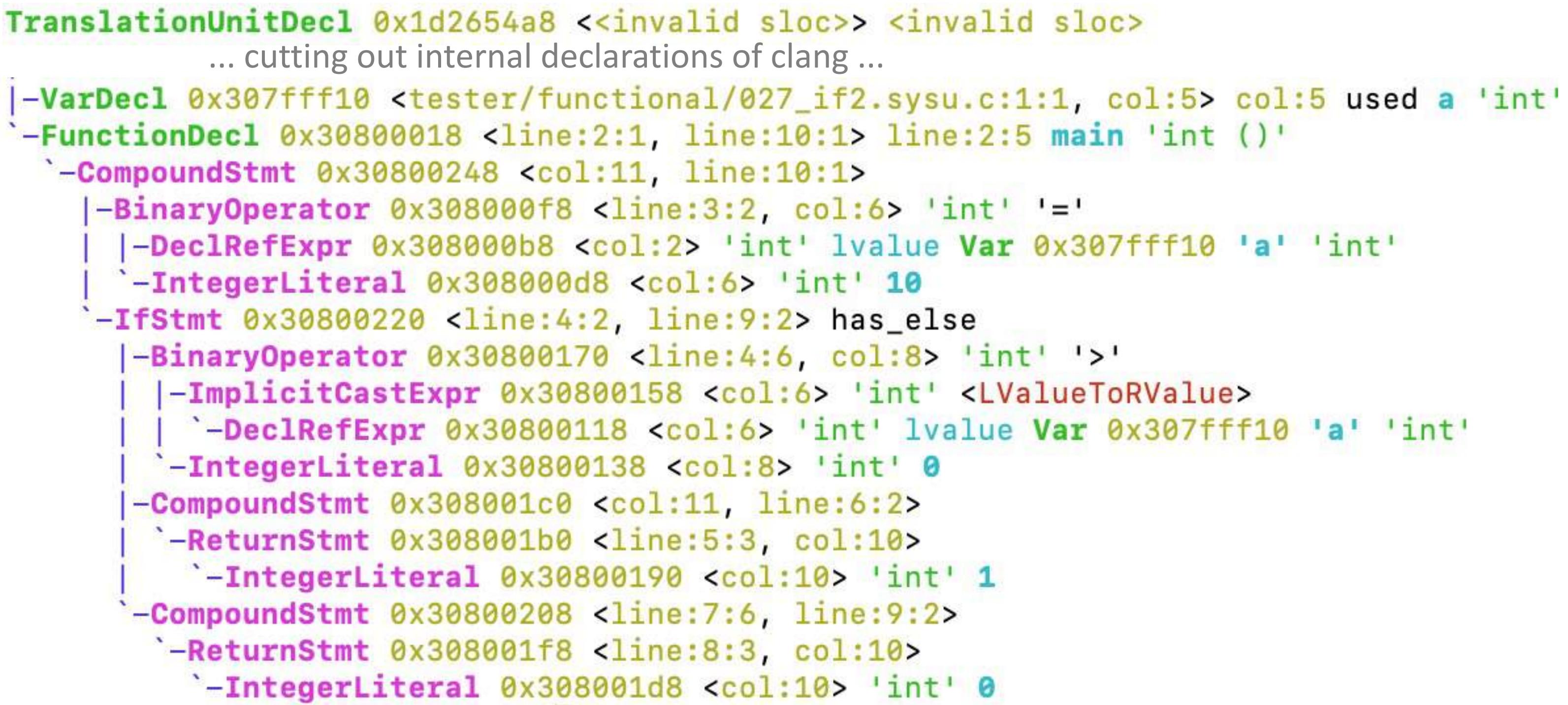
Clang AST

- $clang -Xclang -ast-dump -fsyntax-only tester/functional/027_if2.sysu.c



```
 1 int a;
 2 int main(){
 3         a = 10;
 4         if( a>0 ){
 5                 return 1;
 6         }
 7         else{
 8                 return 0;
 9         }
10 }
```





```
TranslationUnitDecl 0x1d2654a8 <<invalid sloc>> <invalid sloc>
          ... cutting out internal declarations of clang ...
|-VarDecl 0x307fff10 <tester/functional/027_if2.sysu.c:1:1, col:5> col:5 used a 'int'
`-FunctionDecl 0x30800018 <line:2:1, line:10:1> line:2:5 main 'int ()'
  `-CompoundStmt 0x30800248 <col:11, line:10:1>
    |-BinaryOperator 0x308000f8 <line:3:2, col:6> 'int' '='
    | |-DeclRefExpr 0x308000b8 <col:2> 'int' lvalue Var 0x307fff10 'a' 'int'
    | `-IntegerLiteral 0x308000d8 <col:6> 'int' 10
    `-IfStmt 0x30800220 <line:4:2, line:9:2> has_else
      |-BinaryOperator 0x30800170 <line:4:6, col:8> 'int' '>'
      | |-ImplicitCastExpr 0x30800158 <col:6> 'int' <LValueToRValue>
      | | `-DeclRefExpr 0x30800118 <col:6> 'int' lvalue Var 0x307fff10 'a' 'int'
      | `-IntegerLiteral 0x30800138 <col:8> 'int' 0
      |-CompoundStmt 0x308001c0 <col:11, line:6:2>
      | `-ReturnStmt 0x308001b0 <line:5:3, col:10>
      |   `-IntegerLiteral 0x30800190 <col:10> 'int' 1
      `-CompoundStmt 0x30800208 <line:7:6, line:9:2>
        `-ReturnStmt 0x308001f8 <line:8:3, col:10>
          `-IntegerLiteral 0x308001d8 <col:10> 'int' 0
```

https://clang.llvm.org/docs/IntroductionToTheClangAST.html

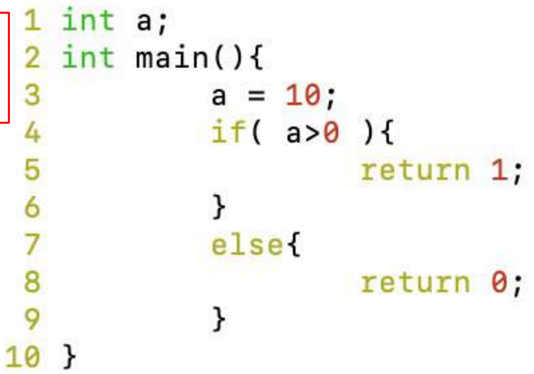
Clang AST

- $clang -Xclang -ast-dump -fsyntax-only tester/functional/027_if2.sysu.c

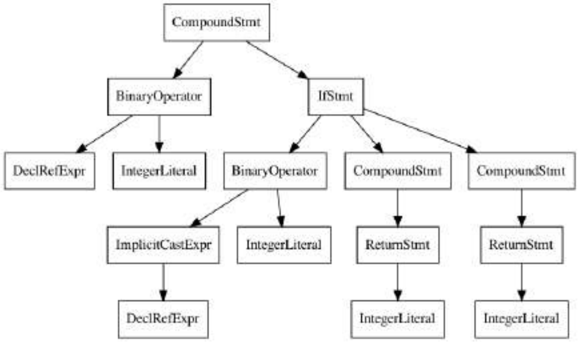
The toplevel declaration in a translation unit is always the translation unit declaration

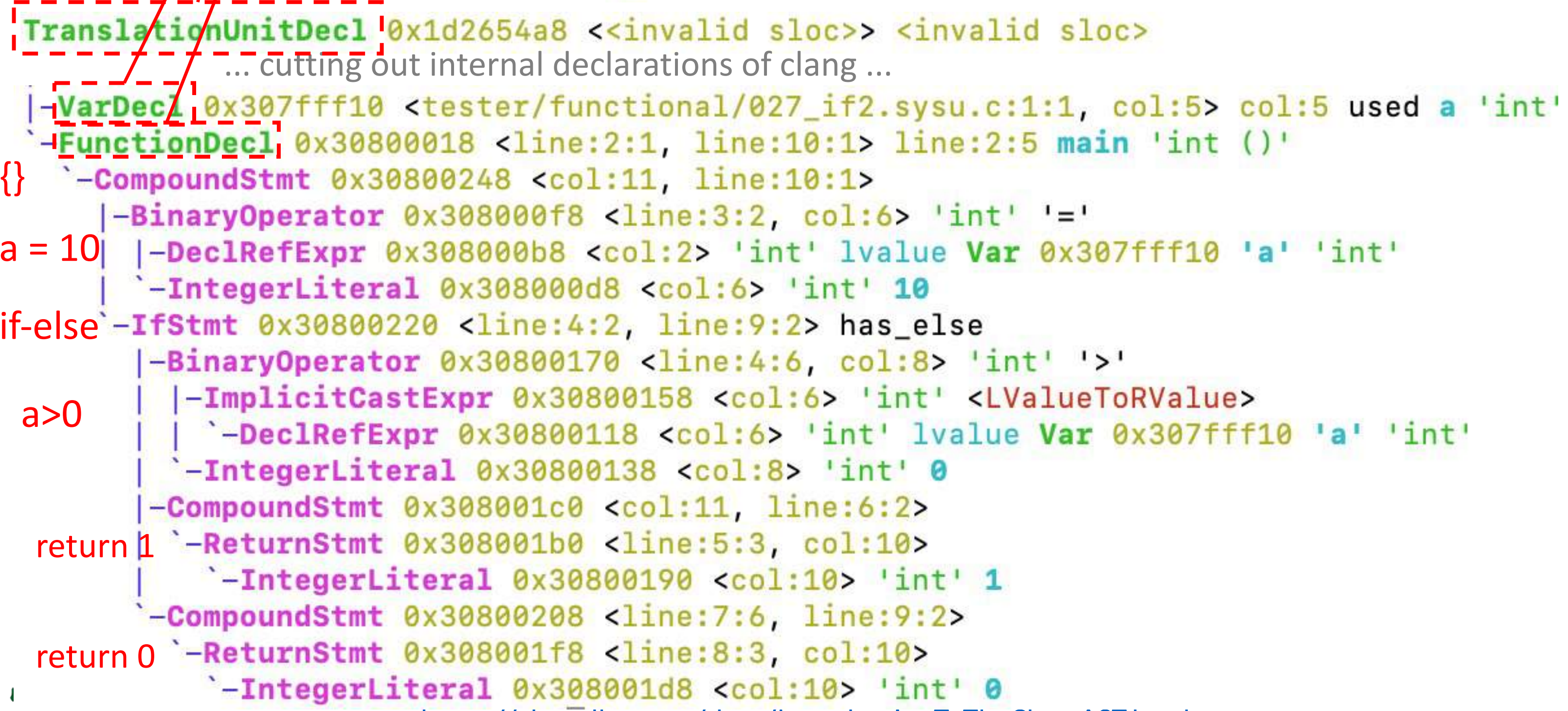
a variable declaration or definition

a function declaration or definition



```
int a;
int main(){
        a = 10;
        if( a>0 ){
                return 1;
        }
        else{
                return 0;
        }
}
```





```
TranslationUnitDecl 0x1d2654a8 <<invalid sloc>> <invalid sloc>
```

... cutting out internal declarations of clang ...

```
|-VarDecl 0x307fff10 <tester/functional/027_if2.sysu.c:1:1, col:5> col:5 used a 'int'
`-FunctionDecl 0x30800018 <line:2:1, line:10:1> line:2:5 main 'int ()'
  `-CompoundStmt 0x30800248 <col:11, line:10:1>
    |-BinaryOperator 0x308000f8 <line:3:2, col:6> 'int' '='
    | |-DeclRefExpr 0x308000b8 <col:2> 'int' lvalue Var 0x307fff10 'a' 'int'
    | `-IntegerLiteral 0x308000d8 <col:6> 'int' 10
    `-IfStmt 0x30800220 <line:4:2, line:9:2> has_else
      |-BinaryOperator 0x30800170 <line:4:6, col:8> 'int' '>'
      | |-ImplicitCastExpr 0x30800158 <col:6> 'int' <LValueToRValue>
      | | `-DeclRefExpr 0x30800118 <col:6> 'int' lvalue Var 0x307fff10 'a' 'int'
      | `-IntegerLiteral 0x30800138 <col:8> 'int' 0
      |-CompoundStmt 0x308001c0 <col:11, line:6:2>
      | `-ReturnStmt 0x308001b0 <line:5:3, col:10>
      |   `-IntegerLiteral 0x30800190 <col:10> 'int' 1
      `-CompoundStmt 0x30800208 <line:7:6, line:9:2>
        `-ReturnStmt 0x308001f8 <line:8:3, col:10>
          `-IntegerLiteral 0x308001d8 <col:10> 'int' 0
```

{}

a = 10

if-else

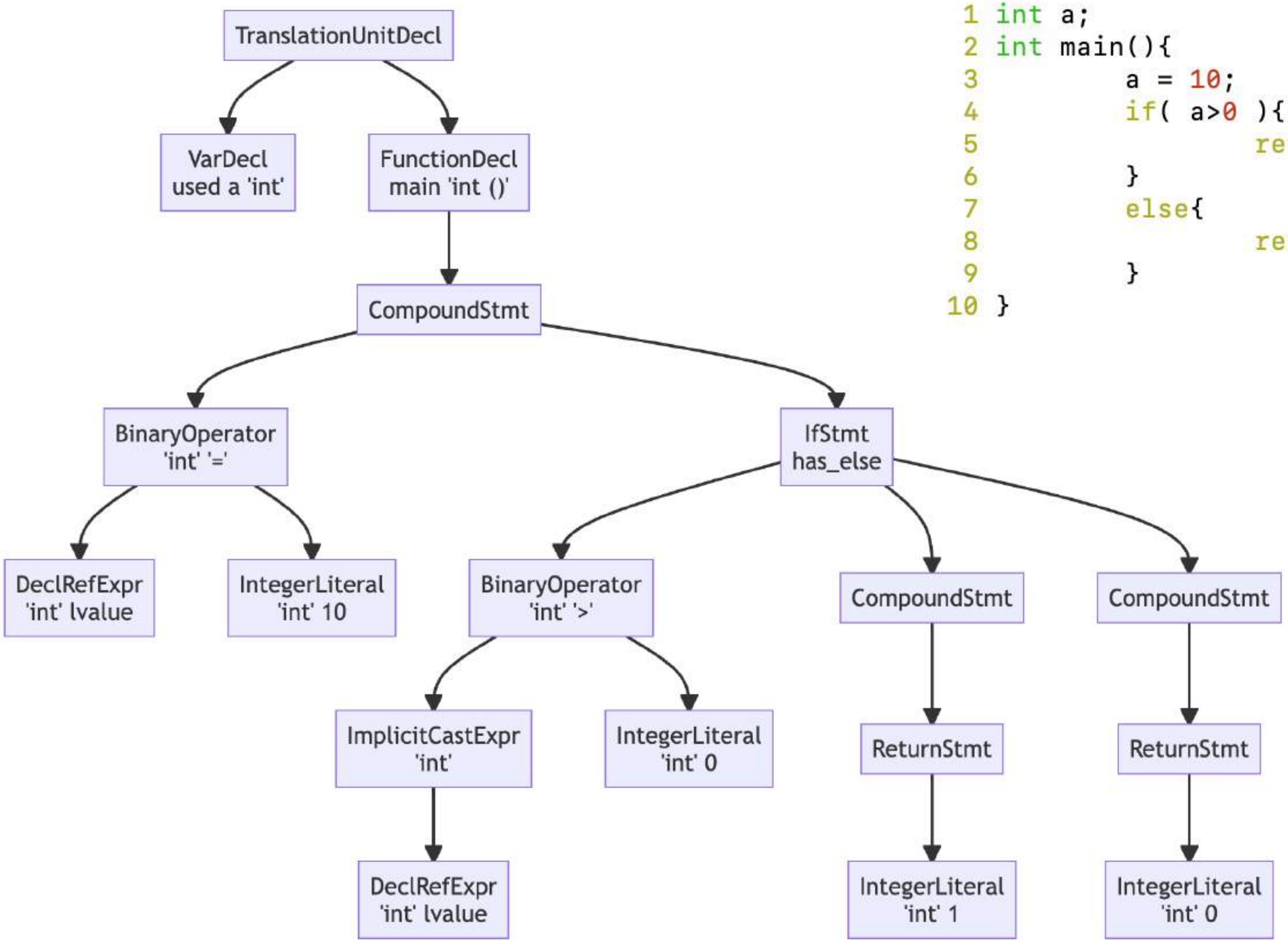
a>0

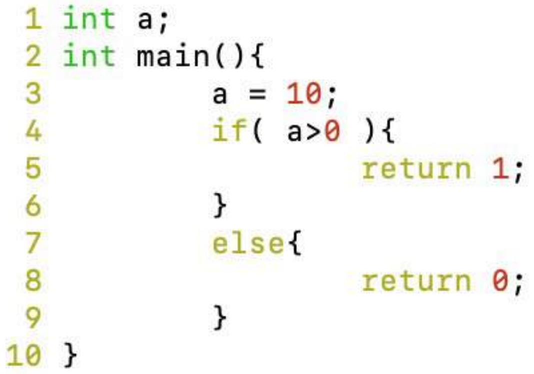
return 1

return 0

https://clang.llvm.org/docs/IntroductionToTheClangAST.html

Clang AST (cont.)

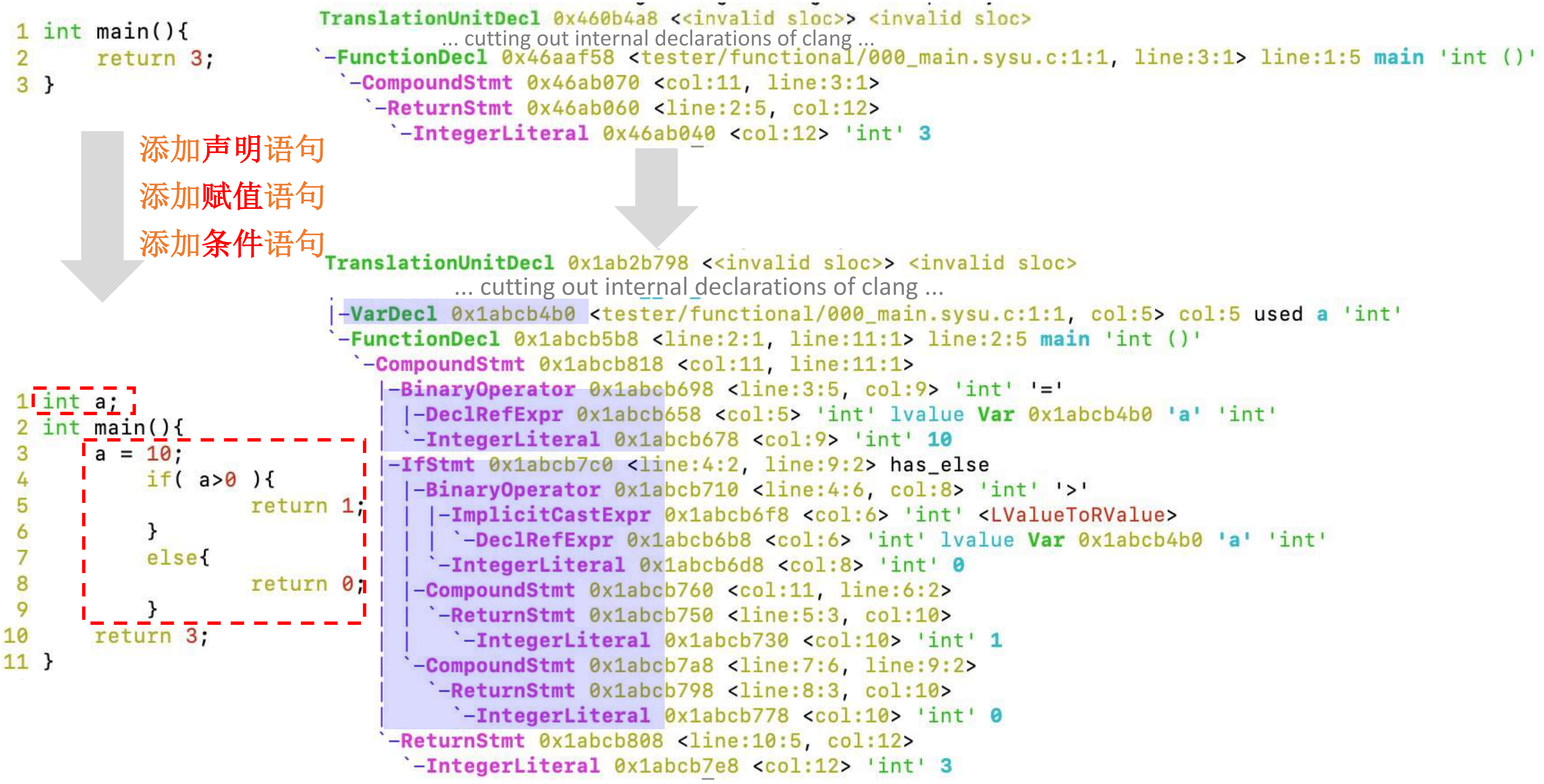




```
int a;
int main(){
        a = 10;
        if( a>0 ){
                return 1;
        }
        else{
                return 0;
        }
}
```

Example

- $clang -Xclang -ast-dump -fsyntax-only tester/functional/000_main.sysu.c



```
1 int main(){
2     return 3;
3 }
```

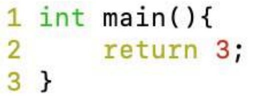
```
TranslationUnitDecl 0x460b4a8 <<invalid sloc>> <invalid sloc>
        ... cutting out internal declarations of clang ...
`-FunctionDecl 0x46aaf58 <tester/functional/000_main.sysu.c:1:1, line:3:1> line:1:5 main 'int ()'
  `-CompoundStmt 0x46ab070 <col:11, line:3:1>
    `-ReturnStmt 0x46ab060 <line:2:5, col:12>
      `-IntegerLiteral 0x46ab040 <col:12> 'int' 3
```

添加声明语句
添加赋值语句
添加条件语句

```
1 int a;
2 int main(){
3     a = 10;
4         if( a>0 ){
5                 return 1;
6         }
7         else{
8                 return 0;
9         }
10    return 3;
11 }
```

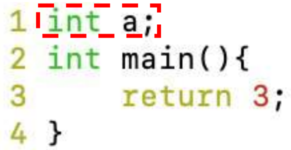
```
TranslationUnitDecl 0x1ab2b798 <<invalid sloc>> <invalid sloc>
        ... cutting out internal declarations of clang ...
|-VarDecl 0x1abcb4b0 <tester/functional/000_main.sysu.c:1:1, col:5> col:5 used a 'int'
`-FunctionDecl 0x1abcb5b8 <line:2:1, line:11:1> line:2:5 main 'int ()'
  `-CompoundStmt 0x1abcb818 <col:11, line:11:1>
    |-BinaryOperator 0x1abcb698 <line:3:5, col:9> 'int' '='
    | |-DeclRefExpr 0x1abcb658 <col:5> 'int' lvalue Var 0x1abcb4b0 'a' 'int'
    | `-IntegerLiteral 0x1abcb678 <col:9> 'int' 10
    |-IfStmt 0x1abcb7c0 <line:4:2, line:9:2> has_else
    | |-BinaryOperator 0x1abcb710 <line:4:6, col:8> 'int' '>'
    | | |-ImplicitCastExpr 0x1abcb6f8 <col:6> 'int' <LValueToRValue>
    | | | `-DeclRefExpr 0x1abcb6b8 <col:6> 'int' lvalue Var 0x1abcb4b0 'a' 'int'
    | | `-IntegerLiteral 0x1abcb6d8 <col:8> 'int' 0
    | |-CompoundStmt 0x1abcb760 <col:11, line:6:2>
    | | `-ReturnStmt 0x1abcb750 <line:5:3, col:10>
    | |   `-IntegerLiteral 0x1abcb730 <col:10> 'int' 1
    | `-CompoundStmt 0x1abcb7a8 <line:7:6, line:9:2>
    |   `-ReturnStmt 0x1abcb798 <line:8:3, col:10>
    |     `-IntegerLiteral 0x1abcb778 <col:10> 'int' 0
    `-ReturnStmt 0x1abcb808 <line:10:5, col:12>
      `-IntegerLiteral 0x1abcb7e8 <col:12> 'int' 3
```

Example: int a;



```
int main(){
    return 3;
}
```

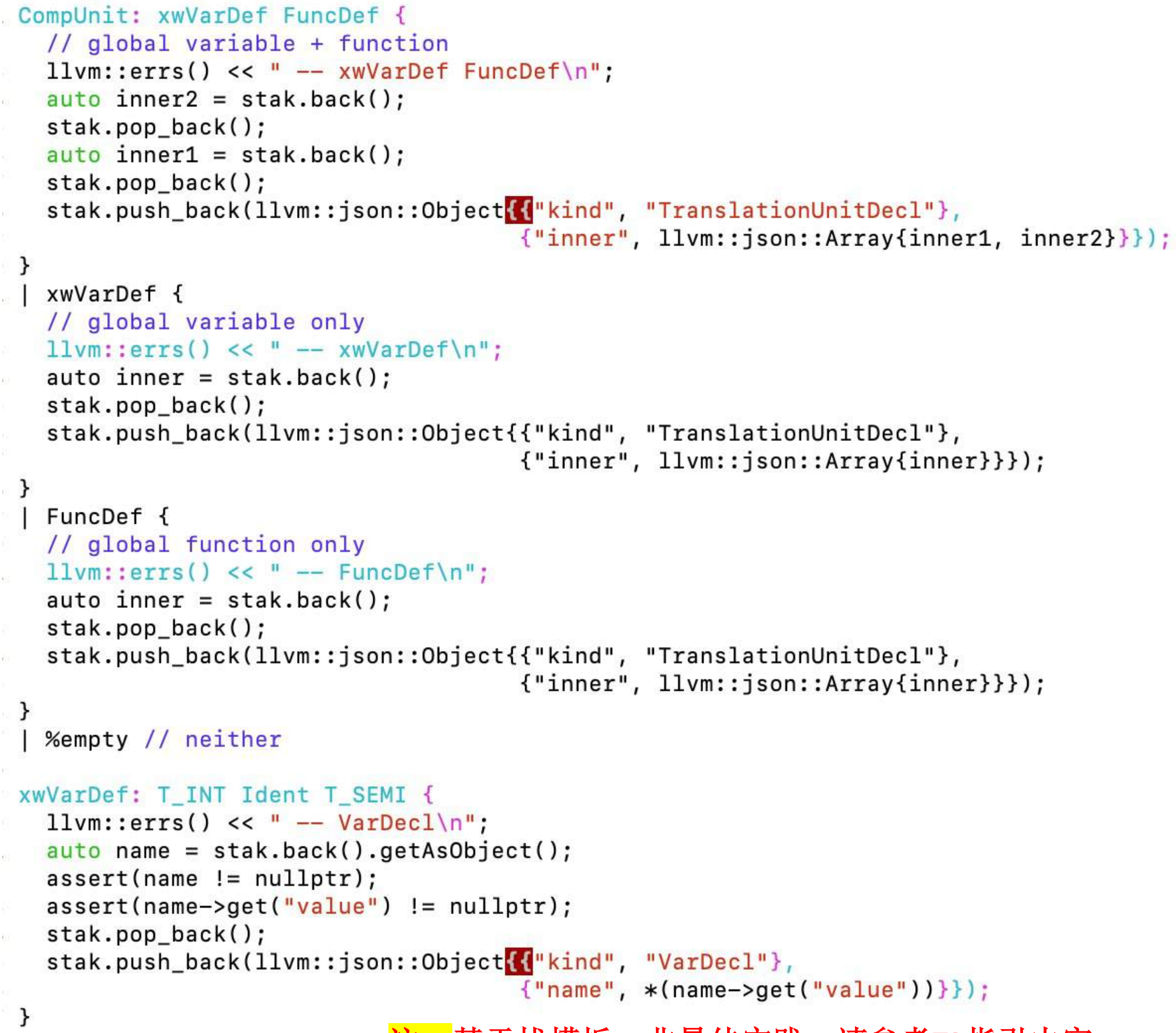
添加声明语句



```
int a;
int main(){
    return 3;
}
```

VarDecl → int id;

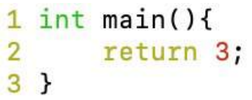
VarDecl → Type Vars;
Type → int | float | double | …;
Vars → Vars VarDef | VarDef
VarDef → id ‘=‘ Initval | id
Initval → val



```
CompUnit: xwVarDef FuncDef {
  // global variable + function
  llvm::errs() << " -- xwVarDef FuncDef\n";
  auto inner2 = stak.back();
  stak.pop_back();
  auto inner1 = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "TranslationUnitDecl"},
                                    {"inner", llvm::json::Array{inner1, inner2}}});
}
| xwVarDef {
  // global variable only
  llvm::errs() << " -- xwVarDef\n";
  auto inner = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "TranslationUnitDecl"},
                                    {"inner", llvm::json::Array{inner}}});
}
| FuncDef {
  // global function only
  llvm::errs() << " -- FuncDef\n";
  auto inner = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "TranslationUnitDecl"},
                                    {"inner", llvm::json::Array{inner}}});
}
| %empty // neither

xwVarDef: T_INT Ident T_SEMI {
  llvm::errs() << " -- VarDecl\n";
  auto name = stak.back().getAsObject();
  assert(name != nullptr);
  assert(name->get("value") != nullptr);
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "VarDecl"},
                                    {"name", *(name->get("value"))}});
}
```

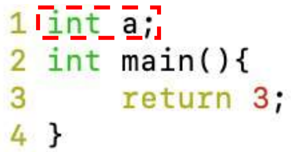
注：基于栈模板，非最佳实践；请参考TA指引内容。

Example: int a;



```
int main(){
    return 3;
}
```

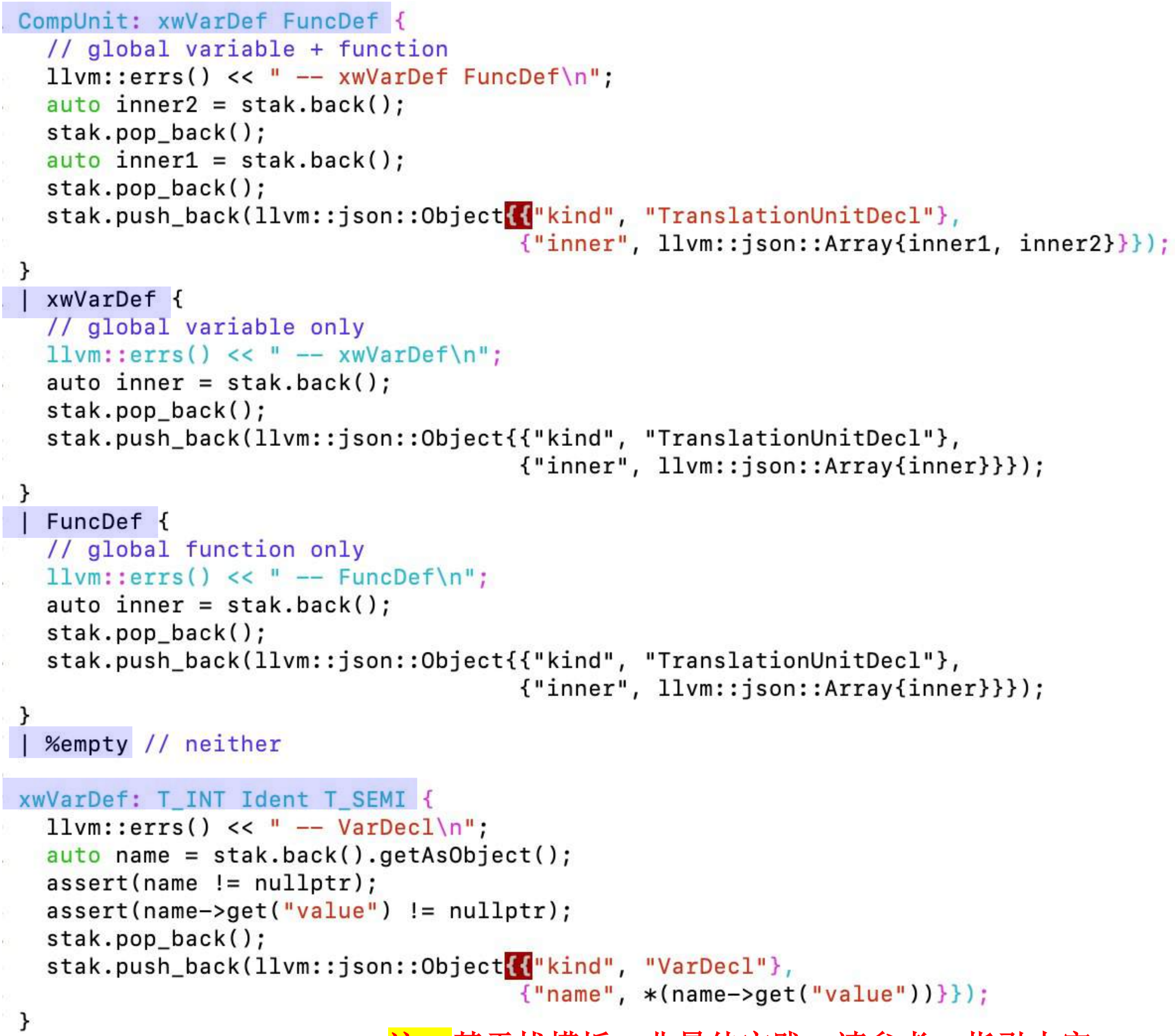
添加声明语句



```
int a;
int main(){
    return 3;
}
```

VarDecl → int id;

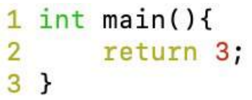
VarDecl → Type Vars;
Type → int | float | double | …;
Vars → Vars VarDef | VarDef
VarDef → id ‘=‘ Initval | id
Initval → val



```
CompUnit: xwVarDef FuncDef {
    // global variable + function
    llvm::errs() << " -- xwVarDef FuncDef\n";
    auto inner2 = stak.back();
    stak.pop_back();
    auto inner1 = stak.back();
    stak.pop_back();
    stak.push_back(llvm::json::Object{{"kind", "TranslationUnitDecl"},
                                       {"inner", llvm::json::Array{inner1, inner2}}});
}
| xwVarDef {
    // global variable only
    llvm::errs() << " -- xwVarDef\n";
    auto inner = stak.back();
    stak.pop_back();
    stak.push_back(llvm::json::Object{{"kind", "TranslationUnitDecl"},
                                       {"inner", llvm::json::Array{inner}}});
}
| FuncDef {
    // global function only
    llvm::errs() << " -- FuncDef\n";
    auto inner = stak.back();
    stak.pop_back();
    stak.push_back(llvm::json::Object{{"kind", "TranslationUnitDecl"},
                                       {"inner", llvm::json::Array{inner}}});
}
| %empty // neither

xwVarDef: T_INT Ident T_SEMI {
    llvm::errs() << " -- VarDecl\n";
    auto name = stak.back().getAsObject();
    assert(name != nullptr);
    assert(name->get("value") != nullptr);
    stak.pop_back();
    stak.push_back(llvm::json::Object{{"kind", "VarDecl"},
                                       {"name", *(name->get("value"))}});
}
```

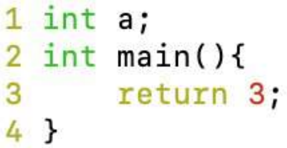
注：基于栈模板，非最佳实践；请参考TA指引内容。

Example: a = 10;



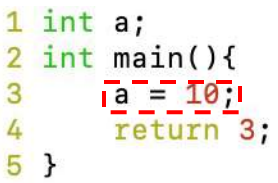
```
int main(){
    return 3;
}
```

添加声明语句

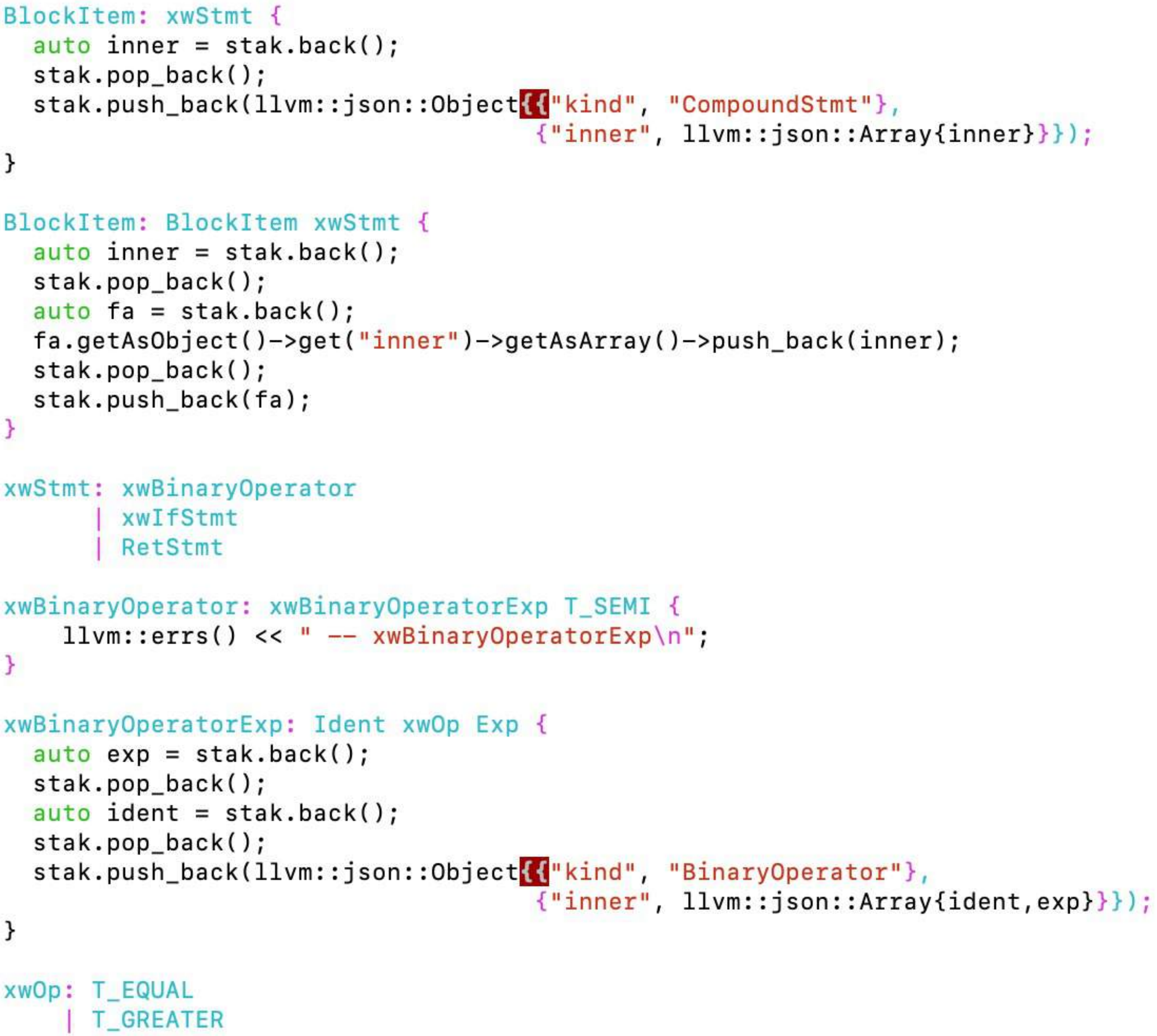


```
int a;
int main(){
    return 3;
}
```

添加赋值语句



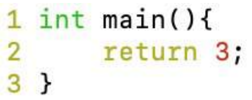
```
int a;
int main(){
    a = 10;
    return 3;
}
```



```
BlockItem: xwStmt {
  auto inner = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "CompoundStmt"},
                                    {"inner", llvm::json::Array{inner}}});
}

BlockItem: BlockItem xwStmt {
  auto inner = stak.back();
  stak.pop_back();
  auto fa = stak.back();
  fa.getAsObject()->get("inner")->getAsArray()->push_back(inner);
  stak.pop_back();
  stak.push_back(fa);
}

xwStmt: xwBinaryOperator
      | xwIfStmt
      | RetStmt

xwBinaryOperator: xwBinaryOperatorExp T_SEMI {
    llvm::errs() << " -- xwBinaryOperatorExp\n";
}

xwBinaryOperatorExp: Ident xwOp Exp {
  auto exp = stak.back();
  stak.pop_back();
  auto ident = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "BinaryOperator"},
                                    {"inner", llvm::json::Array{ident,exp}}});
}

xwOp: T_EQUAL
    | T_GREATER
```

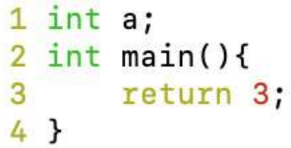
注：基于栈模板，非最佳实践；请参考TA指引内容。

Example: a = 10;



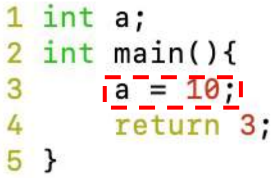
```
int main(){
    return 3;
}
```

添加声明语句

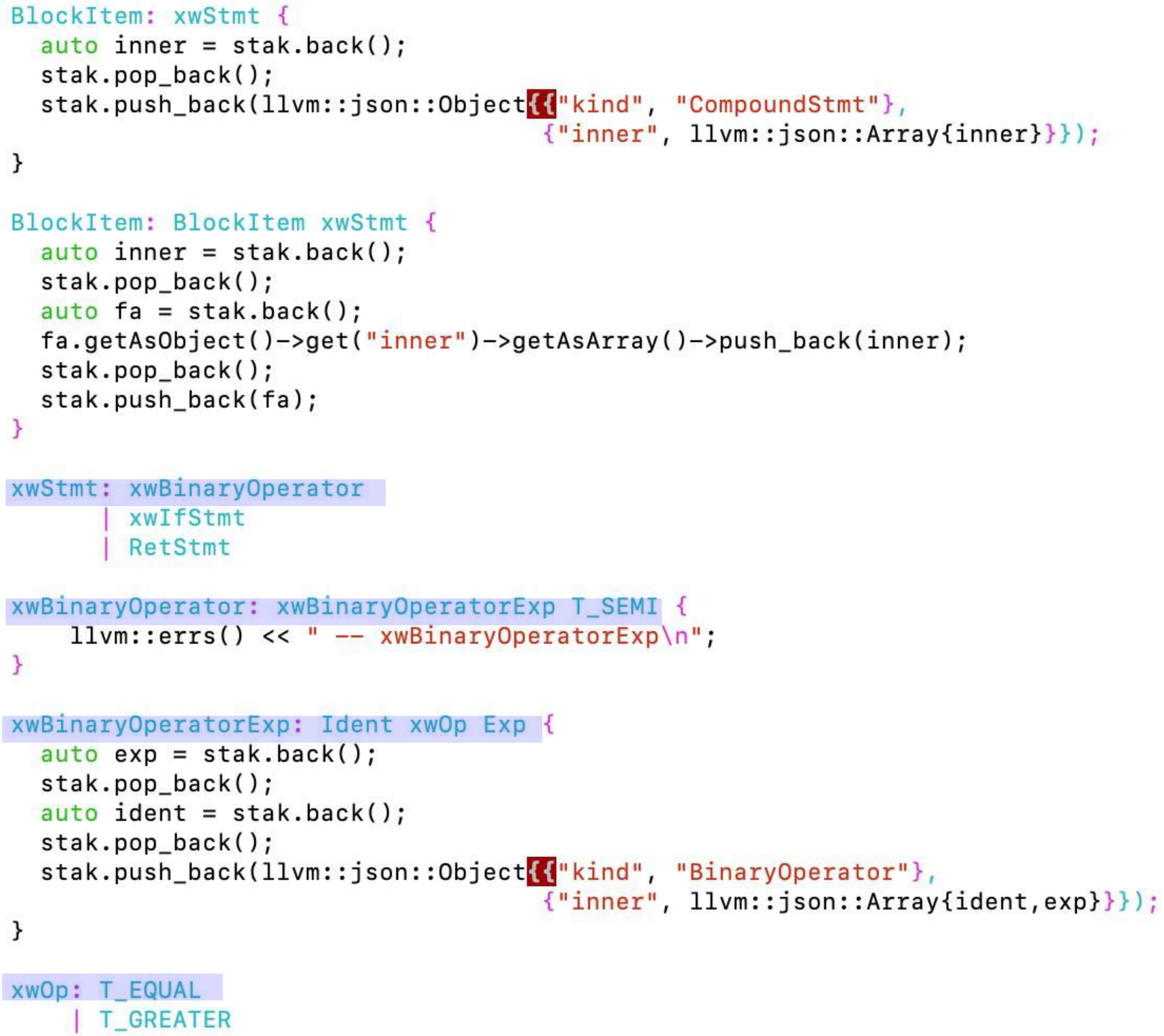


```
int a;
int main(){
    return 3;
}
```

添加赋值语句



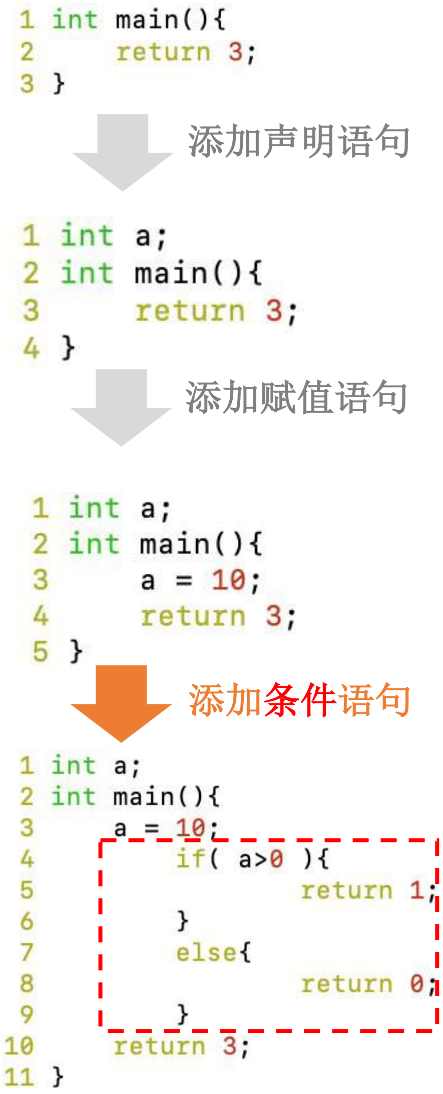
```
int a;
int main(){
    a = 10;
    return 3;
}
```



```
BlockItem: xwStmt {
  auto inner = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "CompoundStmt"},
                                     {"inner", llvm::json::Array{inner}}});
}

BlockItem: BlockItem xwStmt {
  auto inner = stak.back();
  stak.pop_back();
  auto fa = stak.back();
  fa.getAsObject()->get("inner")->getAsArray()->push_back(inner);
  stak.pop_back();
  stak.push_back(fa);
}

xwStmt: xwBinaryOperator
      | xwIfStmt
      | RetStmt

xwBinaryOperator: xwBinaryOperatorExp T_SEMI {
    llvm::errs() << " -- xwBinaryOperatorExp\n";
}

xwBinaryOperatorExp: Ident xwOp Exp {
  auto exp = stak.back();
  stak.pop_back();
  auto ident = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "BinaryOperator"},
                                     {"inner", llvm::json::Array{ident,exp}}});
}

xwOp: T_EQUAL
    | T_GREATER
```

注：基于栈模板，非最佳实践；请参考TA指引内容。

Example: if-else;



```
int main(){
    return 3;
}
```

添加声明语句

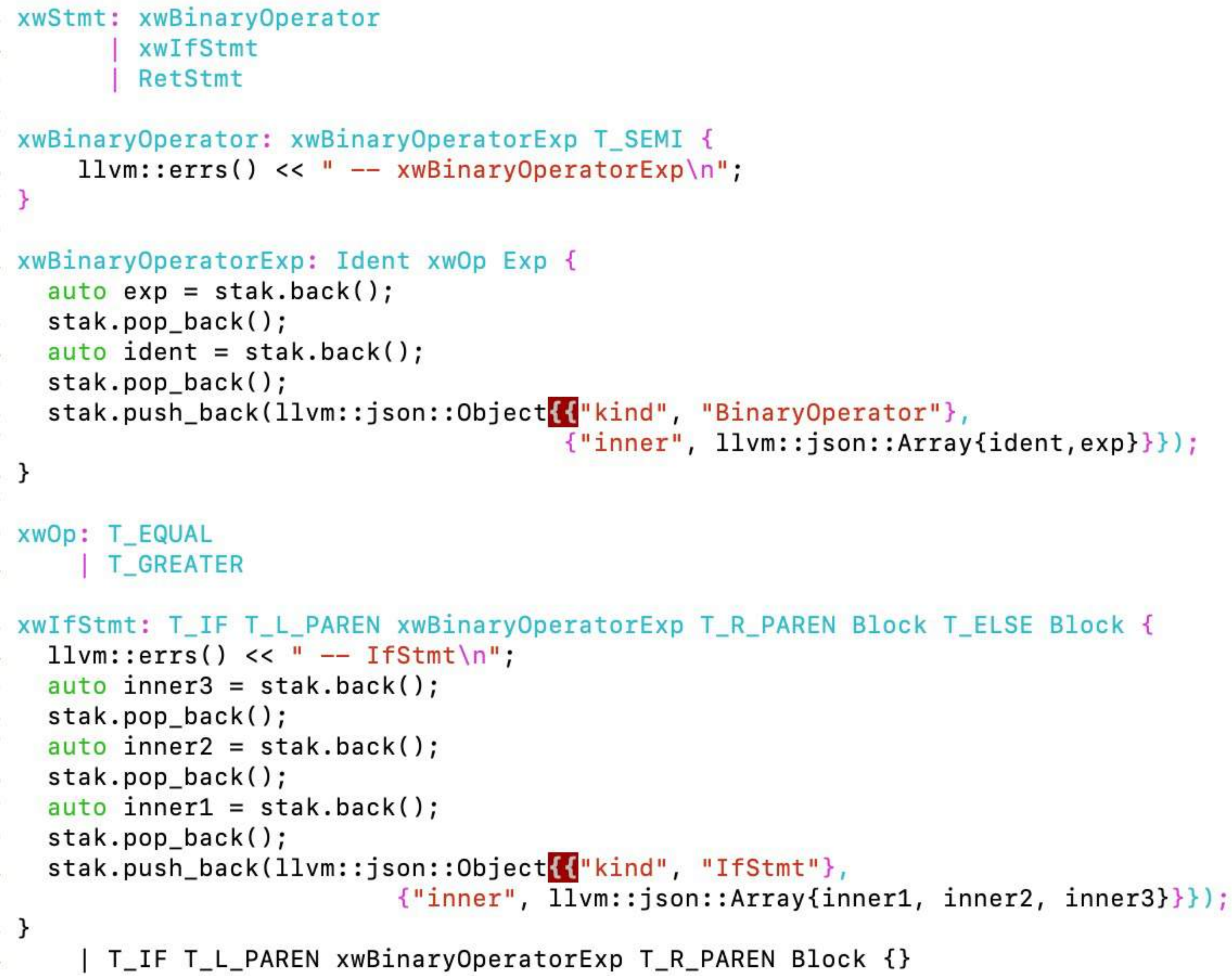
```
int a;
int main(){
    return 3;
}
```

添加赋值语句

```
int a;
int main(){
    a = 10;
    return 3;
}
```

添加条件语句

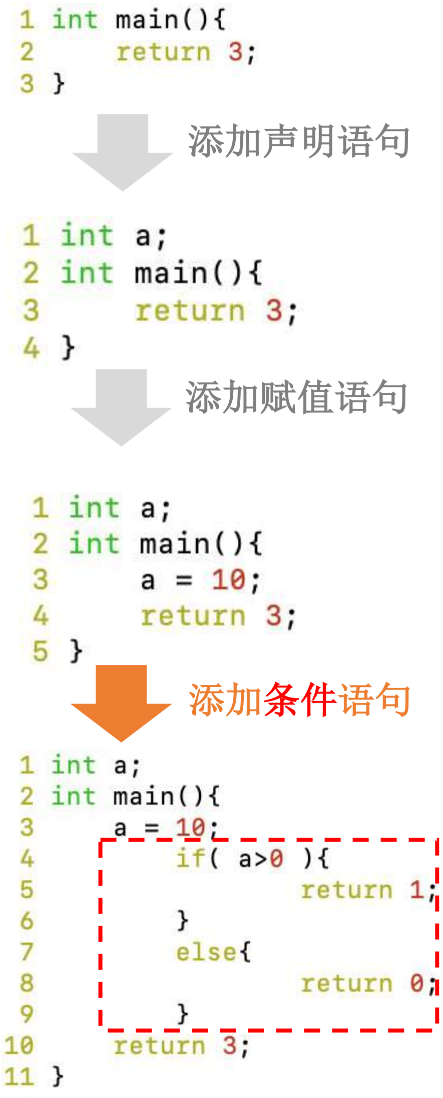
```
int a;
int main(){
    a = 10;
        if( a>0 ){
                return 1;
        }
        else{
                return 0;
        }
    return 3;
}
```



```
xwStmt: xwBinaryOperator
      | xwIfStmt
      | RetStmt

xwBinaryOperator: xwBinaryOperatorExp T_SEMI {
    llvm::errs() << " -- xwBinaryOperatorExp\n";
}

xwBinaryOperatorExp: Ident xwOp Exp {
  auto exp = stak.back();
  stak.pop_back();
  auto ident = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "BinaryOperator"},
                                  {"inner", llvm::json::Array{ident,exp}}});
}

xwOp: T_EQUAL
    | T_GREATER

xwIfStmt: T_IF T_L_PAREN xwBinaryOperatorExp T_R_PAREN Block T_ELSE Block {
  llvm::errs() << " -- IfStmt\n";
  auto inner3 = stak.back();
  stak.pop_back();
  auto inner2 = stak.back();
  stak.pop_back();
  auto inner1 = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "IfStmt"},
                             {"inner", llvm::json::Array{inner1, inner2, inner3}}});
}
    | T_IF T_L_PAREN xwBinaryOperatorExp T_R_PAREN Block {}
```

注：基于栈模板，非最佳实践；请参考TA指引内容。

Example: if-else;



```
int main(){
    return 3;
}
```

添加声明语句

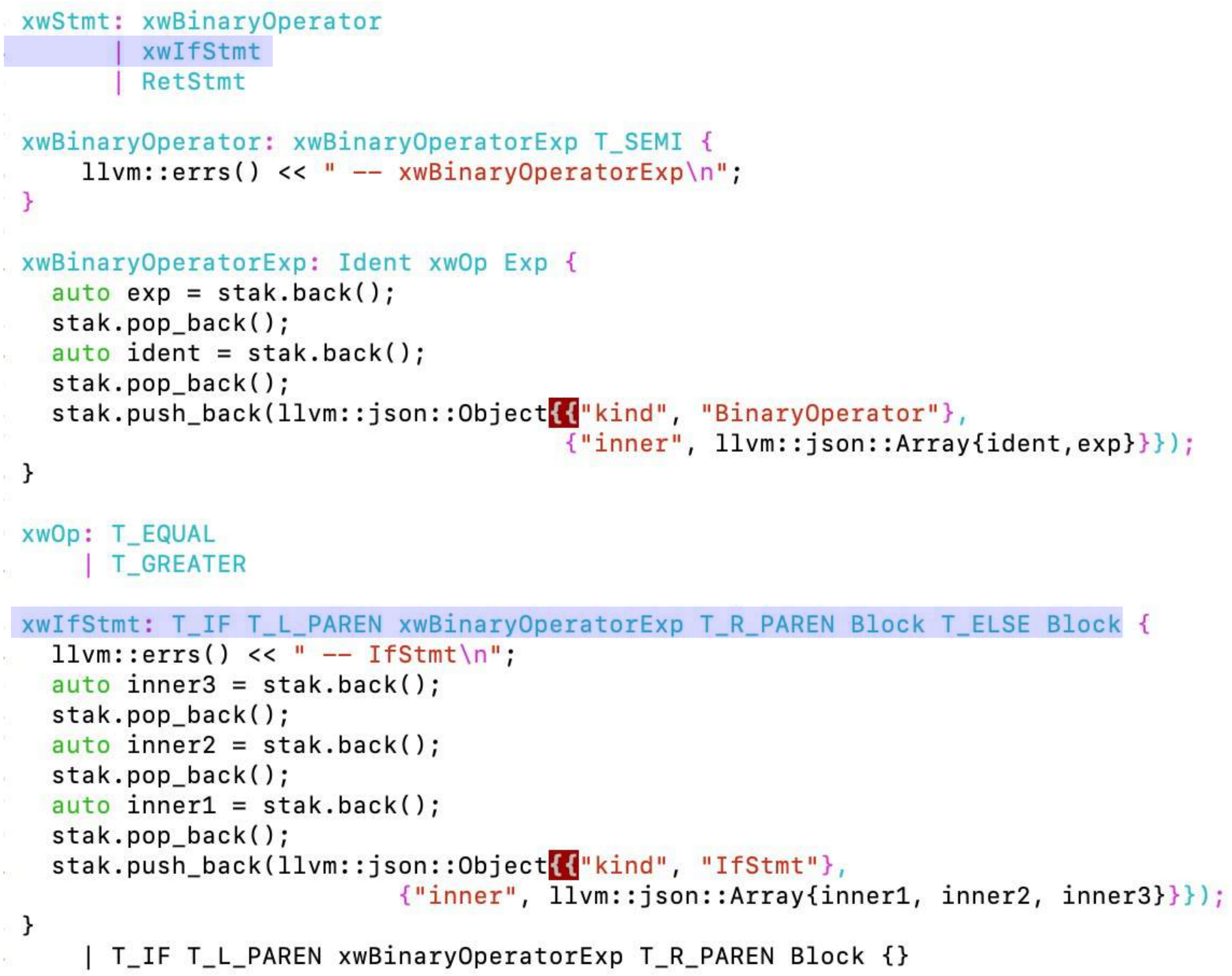
```
int a;
int main(){
    return 3;
}
```

添加赋值语句

```
int a;
int main(){
    a = 10;
    return 3;
}
```

添加条件语句

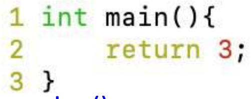
```
int a;
int main(){
    a = 10;
    if( a>0 ){
        return 1;
    }
    else{
        return 0;
    }
    return 3;
}
```



```
xwStmt: xwBinaryOperator
      | xwIfStmt
      | RetStmt

xwBinaryOperator: xwBinaryOperatorExp T_SEMI {
    llvm::errs() << " -- xwBinaryOperatorExp\n";
}

xwBinaryOperatorExp: Ident xwOp Exp {
  auto exp = stak.back();
  stak.pop_back();
  auto ident = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "BinaryOperator"},
                                    {"inner", llvm::json::Array{ident,exp}}});
}

xwOp: T_EQUAL
    | T_GREATER

xwIfStmt: T_IF T_L_PAREN xwBinaryOperatorExp T_R_PAREN Block T_ELSE Block {
  llvm::errs() << " -- IfStmt\n";
  auto inner3 = stak.back();
  stak.pop_back();
  auto inner2 = stak.back();
  stak.pop_back();
  auto inner1 = stak.back();
  stak.pop_back();
  stak.push_back(llvm::json::Object{{"kind", "IfStmt"},
                           {"inner", llvm::json::Array{inner1, inner2, inner3}}});
}
    | T_IF T_L_PAREN xwBinaryOperatorExp T_R_PAREN Block {}
```

注：基于栈模板，非最佳实践；请参考TA指引内容。

Example: Parse Tree



```
int main(){
    return 3;
}
```

yylex()

```
{
  "value": "main"
}
{
  "kind": "IntegerLiteral",
  "value": "3"
}
```

①

RetStmt: T_RETURN Exp T_SEMI {

```
{
  "value": "main"
}
{
  "inner": [
    {
      "kind": "IntegerLiteral",
      "value": "3"
    }
  ],
  "kind": "ReturnStmt"
}
```

②

BlockItem: xwStmt {

```
{
  "value": "main"
}
{
  "inner": [
    {
      "inner": [
        {
          "kind": "IntegerLiteral",
          "value": "3"
        }
      ],
      "kind": "ReturnStmt"
    }
  ],
  "kind": "CompoundStmt"
}
```

③

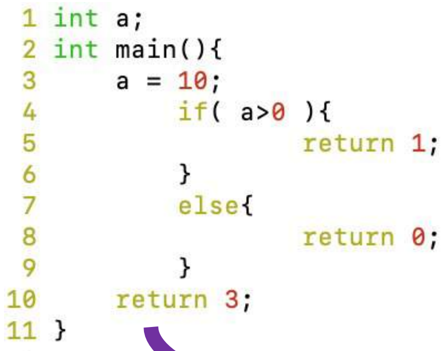
FuncDef: T_INT Ident T_L_PAREN T_R_PAREN Block {

```
{
  "inner": [
    {
      "inner": [
        {
          "inner": [
            {
              "kind": "IntegerLiteral",
              "value": "3"
            }
          ],
          "kind": "ReturnStmt"
        }
      ],
      "kind": "CompoundStmt"
    }
  ],
  "kind": "FunctionDecl",
  "name": "main"
}
```

④

注：基于栈模板，非最佳实践；请参考TA指引内容。

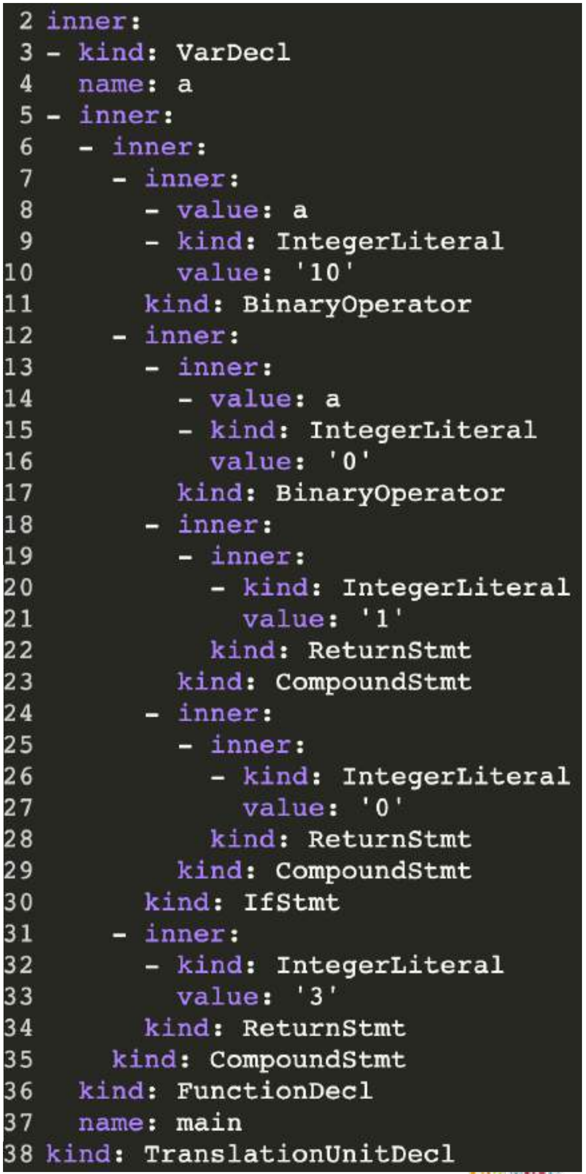
Example: Parse Tree (cont.)



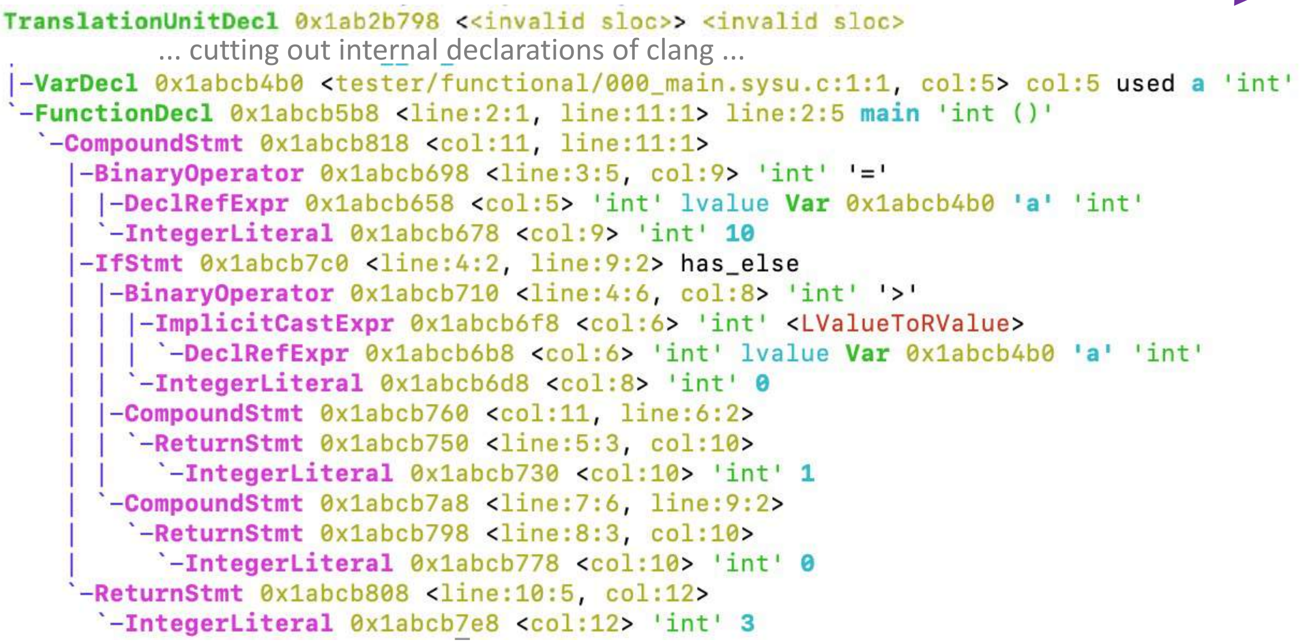
```
int a;
int main(){
    a = 10;
        if( a>0 ){
                return 1;
        }
        else{
                return 0;
        }
    return 3;
}
```

inner":[{"kind":"VarDecl","name":"a"},{"inner":[{"inner":[{"
ner":[{"value":"a"},{"kind":"IntegerLiteral","value":"10"}],"
nd":"BinaryOperator"},{"inner":[{"inner":[{"value":"a"},{"ki
d":"IntegerLiteral","value":"0"}],"kind":"BinaryOperator"},{"
ner":[{"inner":[{"kind":"IntegerLiteral","value":"1"}],"kind":
ReturnStmt"}],"kind":"CompoundStmt"},{"inner":[{"inner":[
kind":"IntegerLiteral","value":"0"}],"kind":"ReturnStmt"}],"
nd":"CompoundStmt"}],"kind":"IfStmt"},{"inner":[{"kind":"I
tegerLiteral","value":"3"}],"kind":"ReturnStmt"}],"kind":"Co
mpoundStmt"}],"kind":"FunctionDecl","name":"main"}],"kin
d":"TranslationUnitDecl"}

json2yaml.com



```
inner:
- kind: VarDecl
  name: a
- inner:
  - inner:
    - inner:
      - value: a
      - kind: IntegerLiteral
        value: '10'
      kind: BinaryOperator
    - inner:
      - inner:
        - value: a
        - kind: IntegerLiteral
          value: '0'
        kind: BinaryOperator
      - inner:
        - inner:
          - kind: IntegerLiteral
            value: '1'
          kind: ReturnStmt
        kind: CompoundStmt
      - inner:
        - inner:
          - kind: IntegerLiteral
            value: '0'
          kind: ReturnStmt
        kind: CompoundStmt
      kind: IfStmt
    - inner:
      - kind: IntegerLiteral
        value: '3'
      kind: ReturnStmt
    kind: CompoundStmt
  kind: FunctionDecl
  name: main
kind: TranslationUnitDecl
```



```
TranslationUnitDecl 0x1ab2b798 <<invalid sloc>> <invalid sloc>
        ... cutting out internal declarations of clang ...
|-VarDecl 0x1abcb4b0 <tester/functional/000_main.sysu.c:1:1, col:5> col:5 used a 'int'
`-FunctionDecl 0x1abcb5b8 <line:2:1, line:11:1> line:2:5 main 'int ()'
  `-CompoundStmt 0x1abcb818 <col:11, line:11:1>
    |-BinaryOperator 0x1abcb698 <line:3:5, col:9> 'int' '='
    | |-DeclRefExpr 0x1abcb658 <col:5> 'int' lvalue Var 0x1abcb4b0 'a' 'int'
    | `-IntegerLiteral 0x1abcb678 <col:9> 'int' 10
    |-IfStmt 0x1abcb7c0 <line:4:2, line:9:2> has_else
    | |-BinaryOperator 0x1abcb710 <line:4:6, col:8> 'int' '>'
    | | |-ImplicitCastExpr 0x1abcb6f8 <col:6> 'int' <LValueToRValue>
    | | | `-DeclRefExpr 0x1abcb6b8 <col:6> 'int' lvalue Var 0x1abcb4b0 'a' 'int'
    | | `-IntegerLiteral 0x1abcb6d8 <col:8> 'int' 0
    | |-CompoundStmt 0x1abcb760 <col:11, line:6:2>
    | | `-ReturnStmt 0x1abcb750 <line:5:3, col:10>
    | |   `-IntegerLiteral 0x1abcb730 <col:10> 'int' 1
    | `-CompoundStmt 0x1abcb7a8 <line:7:6, line:9:2>
    |   `-ReturnStmt 0x1abcb798 <line:8:3, col:10>
    |     `-IntegerLiteral 0x1abcb778 <col:10> 'int' 0
    `-ReturnStmt 0x1abcb808 <line:10:5, col:12>
      `-IntegerLiteral 0x1abcb7e8 <col:12> 'int' 3
```

中山大學 SUN YAT-SEN UNIVERSITY

注：基于栈模板，非最佳实践；请参考TA指引内容。

TA实践指引

- 王永康
  - Wiki, https://github.com/arcsysu/SYsU-lang/wiki/%E5%AE%9E%E9%AA%8C%E4%BA%8C%E8%AF%AD%E6%B3%95%E5%88%86%E6%9E%90

- 张天祎
  - 模板， https://github.com/wufeng15226/SYsU-lang/tree/zty_dev/parser

- 顾宇浩
  - SYsU-lang实验攻略，https://blog.csdn.net/u014132143/article/details/129489861

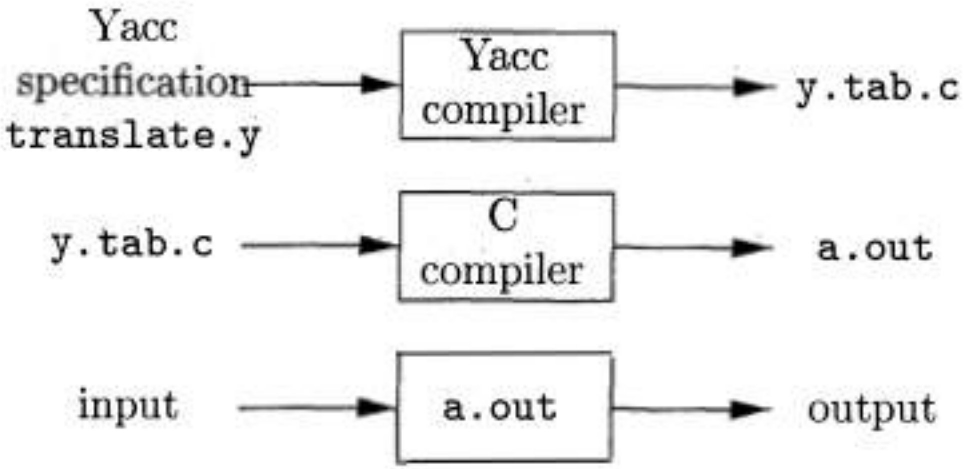
References

- Parser细节(文法、状态等)
  - $bison -v parser.y
    - □ 输出：./parser.output
- 文法规则参考
  - https://buaa-se-compiling.github.io/miniSysY-tutorial/
  - https://github.com/Komorebi660/SysYF-Compiler/blob/master/grammar/SysYFParser.yy
- Jason to XML
  - https://json2yaml.com/
- Clang/LLVM Tutorial
  - Introduction to Clang AST, https://clang.llvm.org/docs/IntroductionToTheClangAST.html
  - https://www.cs.rochester.edu/u/criswell/asplos19/ASPLOS19-LLVM-Tutorial.pdf
- Bison
  - Introduction to Bison, https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/120%20Introducing%20bison.pdf
  - Compiler construction using Flex and Bison, http://www.admb-project.org/tools/flex/compiler.pdf
  - Bison, https://www.gnu.org/software/bison/manual/bison.pdf

Backup …

Yacc Overview

- Yacc is an LALR(1) parser generator
  - YACC: Yet Another Compiler-Compiler
  - Parse a language described by a context-free grammar (**CFG**)
  - Yacc constructs an **LALR(1)** table
- Available as a command on the UNIX system
  - Bison: free GNU project alternative to Yacc



Yacc Specification

- **Definitions** section[定义]:
  - C declarations within %{ %}
  - Token declarations
- **Rules** section[规则]:
  - Each rule consists of a grammar production and the associated semantic action
- **Subroutines** section[辅助函数]:
  - User-defined auxiliary functions

```
%{
 #include …
%}
%token NUM VAR
- - - - - - - - - - - - - - - - - - - -
%%
production { semantic action }
…
- - - - - - - - - - - - - - - - - - - -
%%
…
```

Write a Grammar in Yacc

- A set of productions <head> → <body>$_1$ | … | <body>$_n$ would be written in YACC as:

  <head> : <body>$_1$ { <semantic action>$_1$ }

  …

  : <body>$_n$ { <semantic action>$_n$ }

  ;

- Usages
  - Tokens that are single characters can be used directly within productions, e.g. ‘+’
  - Named tokens must be declared first in the declaration part using %token *TokenName*

Write a Grammar in Yacc (cont.)

- Semantic actions may refer to values of the synthesized attributes of terminals and non-terminals in a production:

  $X : Y_1\ Y_2\ Y_3 \ldots Y_n$ { action }

  - $$ refers to the value of the attribute of X (non-terminal)
  - $i refers to the value of the attribute of Y_i (terminal or non-terminal)
  - Normally the semantic action computes a value for $$ using $i's

- Example: E → E + T | T

```
expr : expr '+' term { $$ = $1 + $2 }
       | term
       ;
```

Write a Grammar in Yacc (cont.)

- Semantic actions may refer to values of the synthesized attributes of terminals and non-terminals in a production:

  $X : Y_1\ Y_2\ Y_3 \ldots Y_n$ { action }

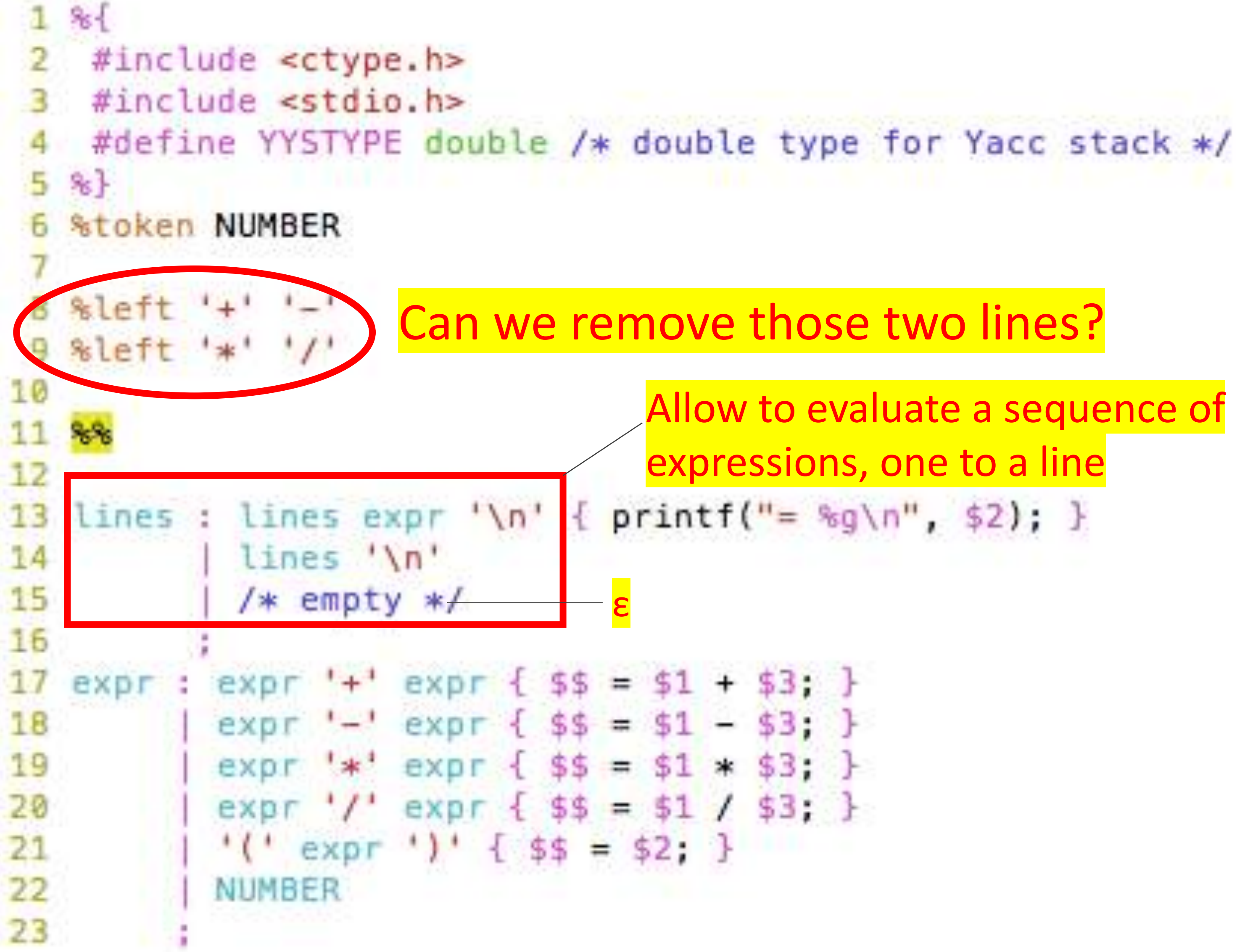
  - $$ refers to the value of the attribute of X (non-terminal)
  - $i refers to the value of the attribute of Y_i (terminal or non-terminal)
  - Normally the semantic action computes a value for $$ using $i's

- Example: E → E + T | T

```
expr : expr '+' term { $$ = $1 + $2 }
       | term
       ;
```

default action: {$$ = $1 }

Example: E → E+E|E-E|E*E|E/E|(E)|num



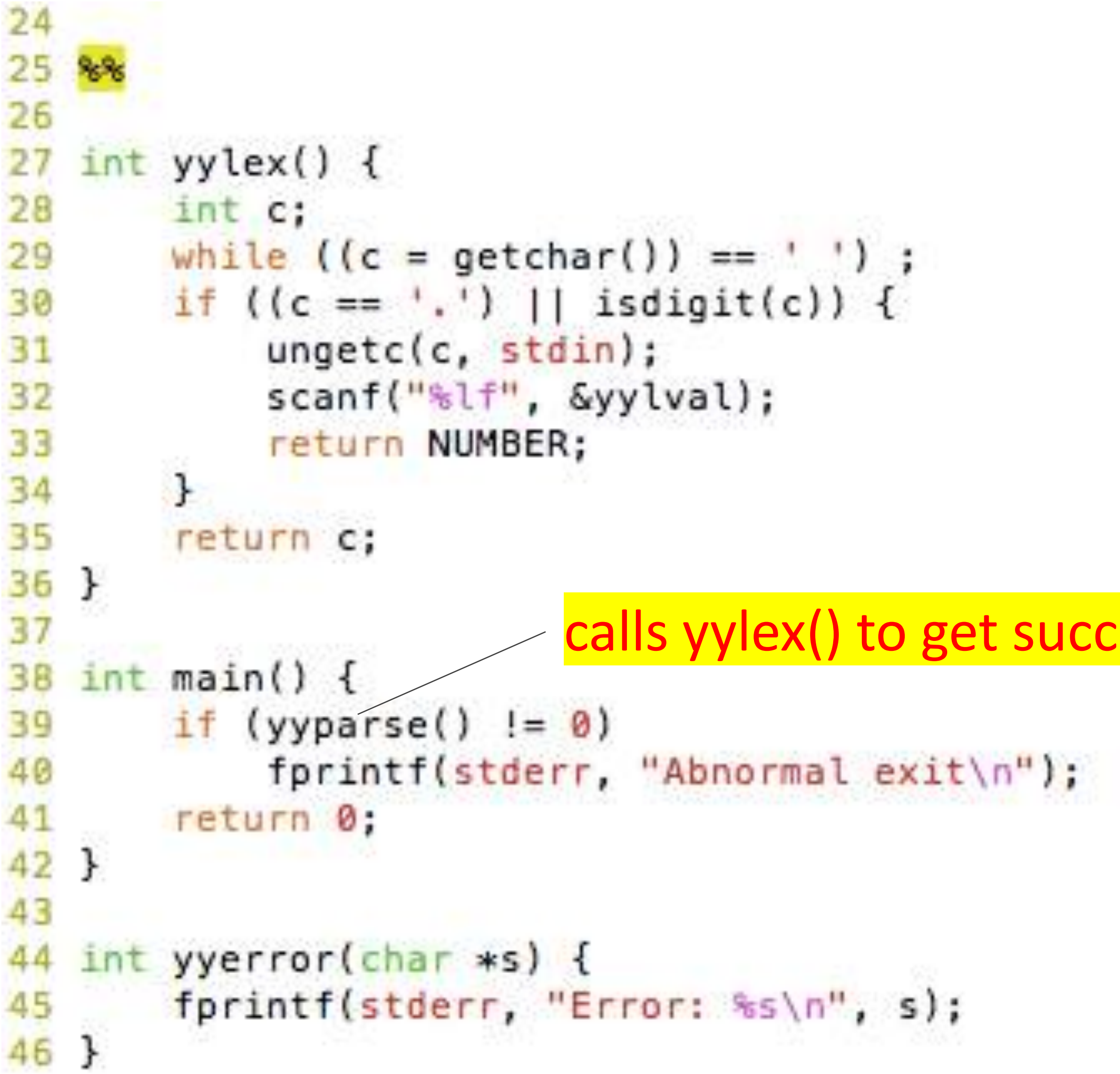
```
%{
 #include <ctype.h>
 #include <stdio.h>
 #define YYSTYPE double /* double type for Yacc stack */
%}
%token NUMBER

%left '+' '-'
%left '*' '/'

%%

lines : lines expr '\n' { printf("= %g\n", $2); }
      | lines '\n'
      | /* empty */
      ;
expr : expr '+' expr { $$ = $1 + $3; }
     | expr '-' expr { $$ = $1 - $3; }
     | expr '*' expr { $$ = $1 * $3; }
     | expr '/' expr { $$ = $1 / $3; }
     | '(' expr ')' { $$ = $2; }
     | NUMBER
     ;
```

Can we remove those two lines?

Allow to evaluate a sequence of expressions, one to a line

ε

Example (cont.)



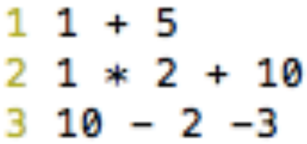
```

%%

int yylex() {
    int c;
    while ((c = getchar()) == ' ') ;
    if ((c == '.') || isdigit(c)) {
        ungetc(c, stdin);
        scanf("%lf", &yylval);
        return NUMBER;
    }
    return c;
}

int main() {
    if (yyparse() != 0)
        fprintf(stderr, "Abnormal exit\n");
    return 0;
}

int yyerror(char *s) {
    fprintf(stderr, "Error: %s\n", s);
}
```

calls yylex() to get successive tokens

Compile and Run …

- Compile
  - $yacc -d *parser*.y
  - $clang -o *test* y.tab.c

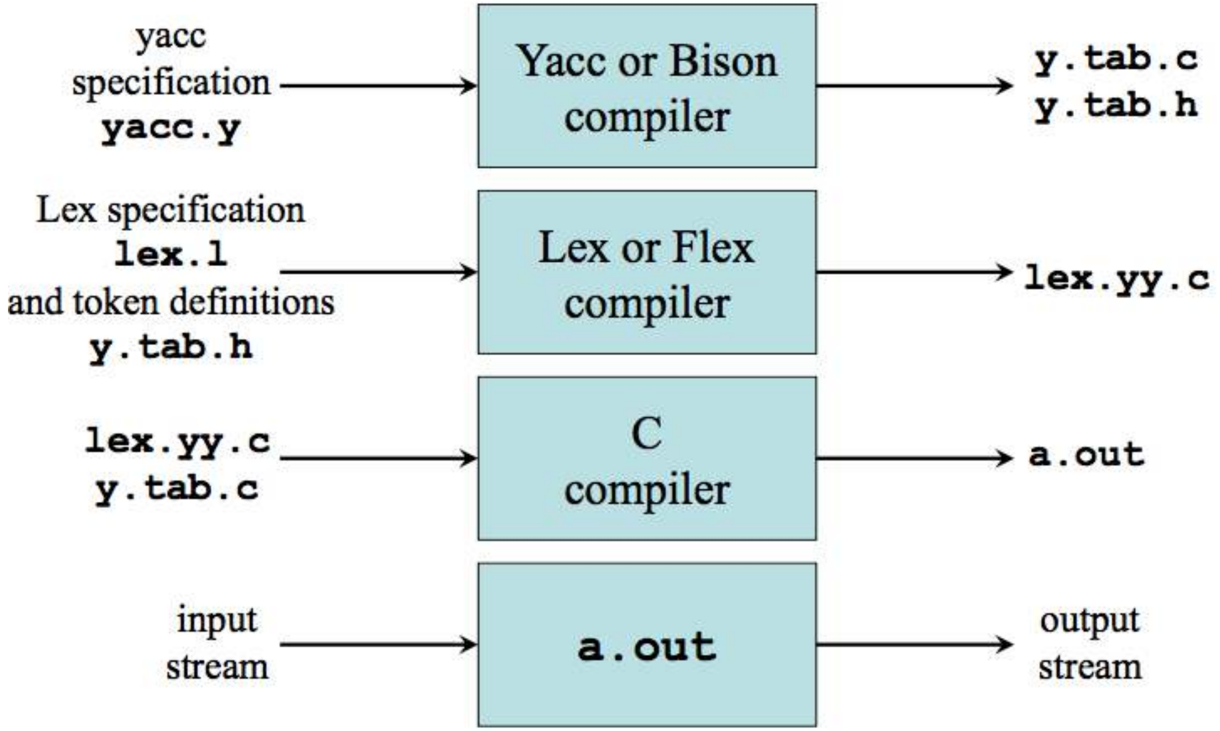
- Run
  - $./*test* < exprs.txt



```
1 + 5
1 * 2 + 10
10 - 2 -3
```

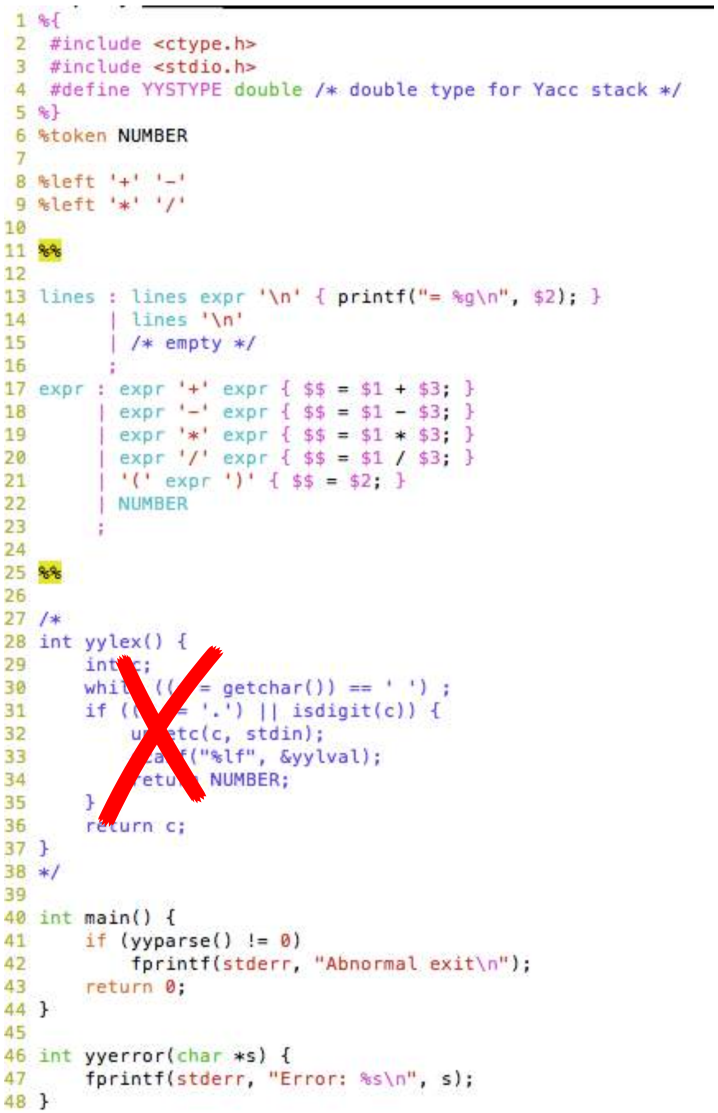
Yacc + Lex

- Lex was designed to produce lexical analyzers that could be used with Yacc
- Yacc generates a parser in y.tab.c and a header y.tab.h
- Lex includes the header and utilizes token definitions
- Yacc calls yylex() to obtain tokens



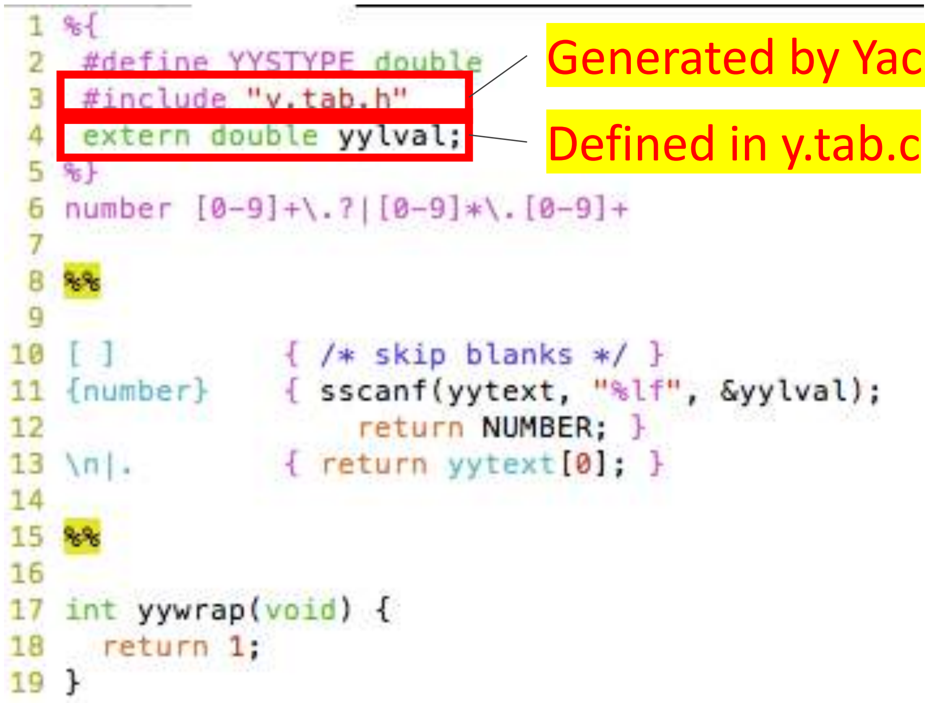
Example: Yacc + Lex

parser.y



```
%{
 #include <ctype.h>
 #include <stdio.h>
 #define YYSTYPE double /* double type for Yacc stack */
%}
%token NUMBER

%left '+' '-'
%left '*' '/'

%%

lines : lines expr '\n' { printf("= %g\n", $2); }
        | lines '\n'
        | /* empty */
        ;
expr : expr '+' expr { $$ = $1 + $3; }
       | expr '-' expr { $$ = $1 - $3; }
       | expr '*' expr { $$ = $1 * $3; }
       | expr '/' expr { $$ = $1 / $3; }
       | '(' expr ')' { $$ = $2; }
       | NUMBER
       ;

%%

/*
int yylex() {
    int c;
    while ((c = getchar()) == ' ') ;
    if ((c == '.') || isdigit(c)) {
        ungetc(c, stdin);
        scanf("%lf", &yylval);
        return NUMBER;
    }
    return c;
}
*/

int main() {
    if (yyparse() != 0)
        fprintf(stderr, "Abnormal exit\n");
    return 0;
}

int yyerror(char *s) {
    fprintf(stderr, "Error: %s\n", s);
}
```

lexer.l



```
%{
 #define YYSTYPE double
 #include "y.tab.h"
 extern double yylval;
%}
number [0-9]+\.?|[0-9]*\.[0-9]+

%%

[ ]         { /* skip blanks */ }
{number}    { sscanf(yytext, "%lf", &yylval);
                 return NUMBER; }
\n|.        { return yytext[0]; }

%%

int yywrap(void) {
   return 1;
}
```

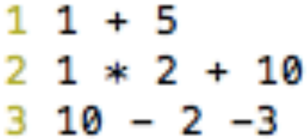
Generated by Yacc

Defined in y.tab.c

Compile and Run …

- Compile
  - $yacc -d *parser*.y
  - $lex *lexer*.l
  - $clang -o *test* y.tab.c lex.yy.c

- Run
  - $./*test* < exprs.txt



```
1 + 5
1 * 2 + 10
10 - 2 -3
```

References

- 编译原理（第2版），章节4.9
- Yacc/Bison - Parser Generators, https://tldp.org/LDP/LG/issue87/ramankutty.html
- Lex and Yacc – A Quick Tour, https://courses.cs.washington.edu/courses/cse322/07au/slides/lec25.pdf
- ANTLR, Yacc, and Bison, https://www.cs.csustan.edu/~xliang/Courses/CS4300-20F/Notes/Ch4c.pdf
- Yacc Practice, https://epaperpress.com/lexandyacc/pry1.html
- The Lex & Yacc Page, http://dinosaur.compilertools.net/