中山大學 SUN YAT-SEN UNIVERSITY

计算机学院 COMPUTER SCIENCE AND ENGINEERING

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

编译器构造实验

实验二 语法分析

Yat Compiler Construction with AI

yatcc-ai.com

deepseek NSCC GZ Starlight

中山大学 计算机学院

国家超级计算广州中心

2025.3

实验介绍

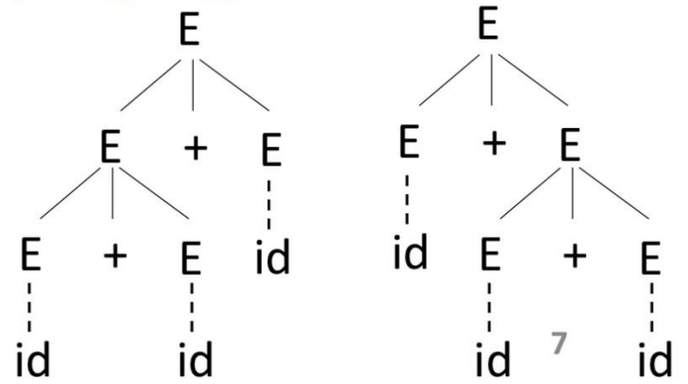
语法分析器

提取出语法结构！

build > test > task2 > functional-0 > 000_main.sysu.c > {} answer.simplied.json > ...


```json
{
  "kind": "TranslationUnitDecl",
  "inner": [
    {
      "kind": "FunctionDecl",
      "name": "main",
      "type": {
        "qualType": "int ()"
      },
      "inner": [
        {
          "kind": "CompoundStmt",
          "inner": [
            {
              "kind": "ReturnStmt",
              "inner": [
                {
                  "kind": "IntegerLiteral",
                  "type": {
                    "qualType": "int"
                  },
                  "valueCategory": "prvalue",
                  "value": "3"
                }
              ]
            }
          ]
        }
```


id + id + id



answer.txt

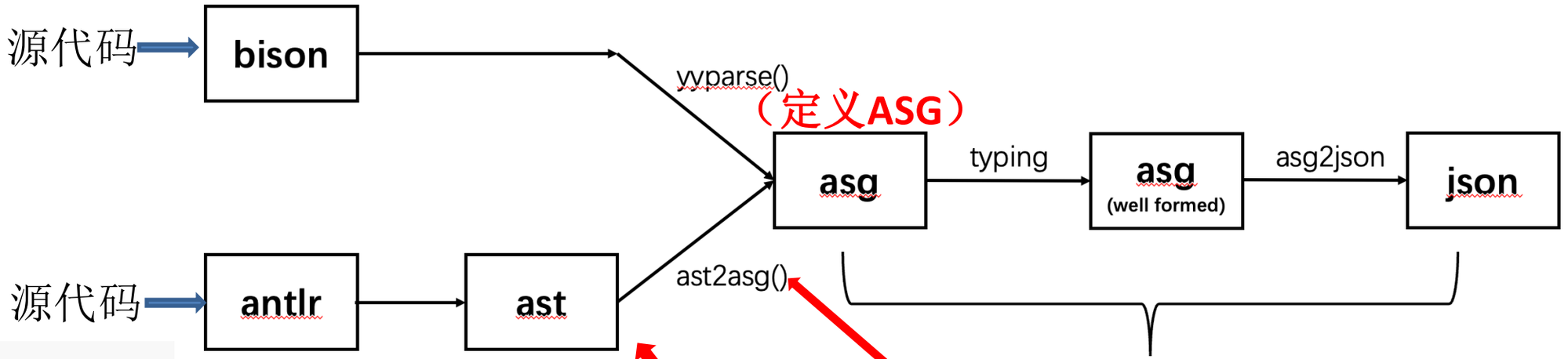
build > test > task2 > functional-0 > 000_main.sysu.c > answer.txt


```
TranslationUnitDecl 0x564c458c67d8 <<invalid sloc>> <invalid sloc>
|-TypedefDecl 0x564c458c7008 <<invalid sloc>> <invalid sloc> implicit __int128_t '__int128'
| `-BuiltinType 0x564c458c6da0 '__int128'
|-TypedefDecl 0x564c458c7078 <<invalid sloc>> <invalid sloc> implicit __uint128_t 'unsigned __int128'
| `-BuiltinType 0x564c458c6dc0 'unsigned __int128'
|-TypedefDecl 0x564c458c7380 <<invalid sloc>> <invalid sloc> implicit __NSConstantString 'struct __NSConstantString_ta
| `-RecordType 0x564c458c7150 'struct __NSConstantString_tag'
|   `-Record 0x564c458c70d0 '__NSConstantString_tag'
|-TypedefDecl 0x564c458c7428 <<invalid sloc>> <invalid sloc> implicit __builtin_ms_va_list 'char *'
| `-PointerType 0x564c458c73e0 'char *'
|   `-BuiltinType 0x564c458c6880 'char'
|-TypedefDecl 0x564c458c7720 <<invalid sloc>> <invalid sloc> implicit __builtin_va_list 'struct __va_list_tag[1]'
| `-ConstantArrayType 0x564c458c76c0 'struct __va_list_tag[1]' 1
|   `-RecordType 0x564c458c7500 'struct __va_list_tag'
|     `-Record 0x564c458c7480 '__va_list_tag'
`-FunctionDecl 0x564c45918d28 </YatCC/test/cases/functional-0/000_main.sysu.c:1:1, line:3:1> line:1:5 main 'int ()'
  `-CompoundStmt 0x564c45918e48 <col:11, line:3:1>
    `-ReturnStmt 0x564c45918e38 <line:2:5, col:12>
      `-IntegerLiteral 0x564c45918e18 <col:12> 'int' 3
```

实验内容-ANTLR

实验一：源代码 → ANTLR → Token流

实验二：




```
-- antlr
   |-- SYsULexer.cpp      # 需要修改
   |-- SYsULexer.hpp
   |-- SYsULexer.py
   `-- SYsULexer.tokens   # 需要修改
```

词法分析&
语法分析

```
-- antlr
   |-- Ast2Asg.cpp
   |-- Ast2Asg.hpp
   `-- SYsUParser.g4
-- common
   `-- asg.hpp
```

（将AST转换为ASG）

（定义AST）

实验内容-ANTLR

词法分析

1.不启动复活：

Token流（实验一Output） → 实验二语法分析器 → AST

2.启动复活（<u>推荐且默认</u>）：

源代码（实验零Output） → 实验二词法分析器 → Token流 → 实验二语法分析器 → AST

需要填充

```
antlr4::ANTLRInputStream input(inFile);
SYsULexer lexer(&input);

antlr4::CommonTokenStream tokens(&lexer);
SYsUParser parser(&tokens);

auto ast = parser.compilationUnit();
```

实验内容-ANTLR

填充实验二的词法分析部分

```
-- antlr
   |-- SYsULexer.cpp      # 需要修改
   |-- SYsULexer.hpp
   |-- SYsULexer.py
   `-- SYsULexer.tokens   # 需要修改
```

SYsULexer.tokens.hpp：
由SYsULexer.py
根据SYsULexer.tokens生成
在SYsULexer.hpp中引用

task > 2 > antlr > SYsULexer.tokens

```
Int=1
Identifier=2
LeftParen=3
RightParen=4
Return=5
RightBrace=6
LeftBrace=7
Constant=8
Semi=9
Equal=10
Plus=11
Minus=12
Comma=13
LeftBracket=14
RightBracket=15
```

build > task > 2 > antlr > SYsULexer.tokens.hpp

```
#include <cstdint>

namespace SYsULexerTokens {

constexpr size_t kInt = 1;
constexpr size_t kIdentifier = 2;
constexpr size_t kLeftParen = 3;
constexpr size_t kRightParen = 4;
constexpr size_t kReturn = 5;
constexpr size_t kRightBrace = 6;
constexpr size_t kLeftBrace = 7;
constexpr size_t kConstant = 8;
constexpr size_t kSemi = 9;
constexpr size_t kEqual = 11;
constexpr size_t kPlus = 12;
constexpr size_t kComma = 13;
constexpr size_t kLeftBracket = 14;
constexpr size_t kRightBracket = 15;
```

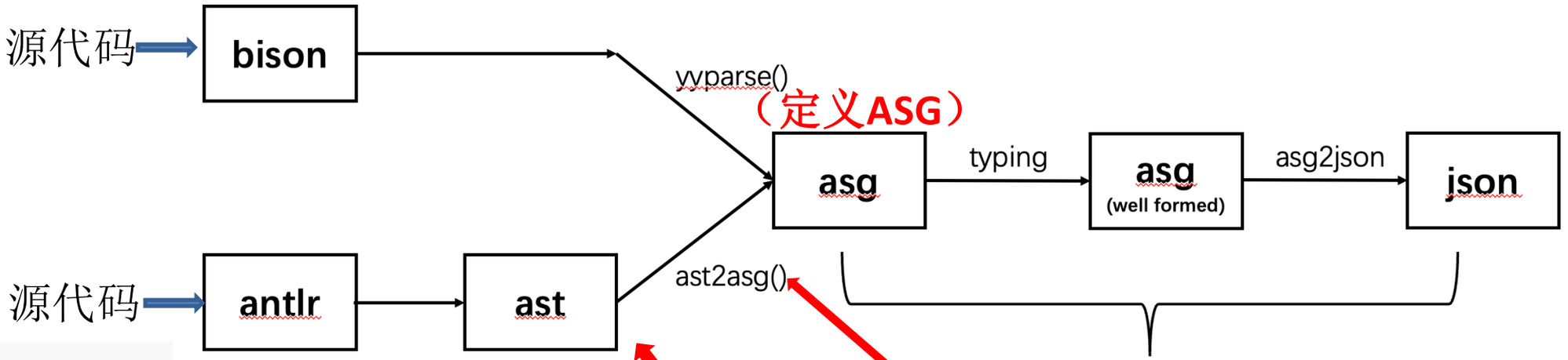
task > 2 > antlr > SYsULexer.cpp > ...

```
static const std::unordered_map<std::string, size_t> kClangTokens{
  { "eof", antlr4::Token::EOF },
  { "int", kInt },
  { "identifier", kIdentifier },
  { "l_paren", kLeftParen },
  { "r_paren", kRightParen },
  { "return", kReturn },
  { "r_brace", kRightBrace },
  { "l_brace", kLeftBrace },
  { "numeric_constant", kConstant },
  { "semi", kSemi },
  { "equal", kEqual },
  { "plus", kPlus },
  { "minus", kMinus },
  { "comma", kComma },
  { "l_square", kLeftBracket },
  { "r_square", kRightBracket }
};
```

填写映射表：
关键字是antlr中的词法token
值是在实验二中使用的词法token
（可以借机重命名为方便的名字）

实验内容-ANTLR

实验二：




```
-- antlr
  |-- SYsULexer.cpp      # 需要修改
  |-- SYsULexer.hpp
  |-- SYsULexer.py
  `-- SYsULexer.tokens   # 需要修改
```

词法分析&
语法分析

```
-- antlr
  |-- Ast2Asg.cpp
  |-- Ast2Asg.hpp
  `-- SYsUParser.g4
-- common
  `-- asg.hpp
```

（将AST转换为ASG）

（定义AST）

实验内容-ANTLR

定义AST

只要通过修改SYsUParser.g4定义好AST，
ANTLR就会自动完成语法解析的工作！

```
-- antlr
  |-- Ast2Asg.cpp
  |-- Ast2Asg.hpp
  `-- SYsUParser.g4
-- common
  `-- asg.hpp
```

task > 2 > antlr > SYsUParser.g4

```
27  unaryExpression
28      :   postfixExpression
29      |   unaryOperator unaryExpression
30      ;
31
32  unaryOperator
33      :   Plus | Minus
34      ;
35
36  multiplicativeExpression
37      :   unaryExpression ((Star|Slash|Mod) unaryExpression)*
38      ;
39
40  additiveExpression
41      :   multiplicativeExpression ((Plus|Minus) multiplicativeExpression)*
42      ;
```

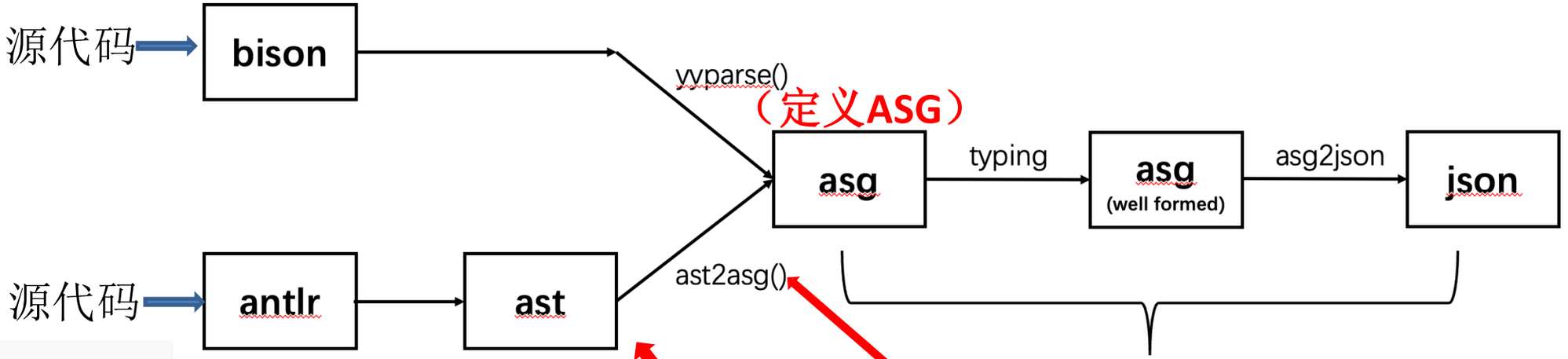
```
antlr4::ANTLRInputStream input(inFile);
SYsULexer lexer(&input);

antlr4::CommonTokenStream tokens(&lexer);
SYsUParser parser(&tokens);

auto ast = parser.compilationUnit();
```

实验内容-ANTLR

实验二：




```
-- antlr
  |-- SYsULexer.cpp      # 需要修改
  |-- SYsULexer.hpp
  |-- SYsULexer.py
  `-- SYsULexer.tokens   # 需要修改
```

词法分析&
语法分析

```
-- antlr
  |-- Ast2Asg.cpp
  |-- Ast2Asg.hpp
  `-- SYsUParser.g4
-- common
  `-- asg.hpp
```

（将AST转换为ASG）

（定义AST）

实验内容-ANTLR

From AST to ASG

通过设置将AST转换为ASG的任务，
考察同学们对语法解析过程的理解！

ASG由asg.hpp定义，
因此要重点理解asg.hpp。
asg.hpp将在教程的“公用代码”部分讲解

小贴士：

1. 在ast2asg.cpp中定义了新函数记得在ast2asg.hpp声明
2. 看完“实验思路”中尤其是“上手思路”一节你就可以上手做实验了。不过想要正确理解与完成整个实验，教程中的其它部分也是很重要的。

```
-- antlr
  |-- Ast2Asg.cpp
  |-- Ast2Asg.hpp
  `-- SYsUParser.g4
-- common
  `-- asg.hpp
```

```
41  auto ast = parser.compilationUnit();
42  Obj::Mgr mgr;
43
44  asg::Ast2Asg ast2asg(mgr);
45  auto asg = ast2asg(ast->translationUnit());
```

实验内容-ANTLR

当实验卡住时，我们可以做什么

1. 调试！在ast2asg.cpp中添加断点，使用 `ctx->getText()` 打印内容。调试将出现两种情况：
   a) 打印发现ctx中缺失所需内容：SYsUParser.g4没写好，导致AST中不存在所需的节点。
      ① 检查SYsUParser.g4中是否存在左递归、二义性等问题。
      ② 对比/build/test/task2/中的answer.txt，检查是否有遗漏的情况没有实现。
   b) 打印发现ctx中存在所需内容，但输出的JSON文件中没有所需内容： ast2asg.cpp没有写好。阅读

2. 阅读教程！没有头绪可能是对实验的理解不够，尤其是关于ASG的部分。除了ASG的介绍，你还可以从`Typing`类、`Asg2Json`类中找找思路。

实验内容-BISON

需要完成下面两个部分的内容：

1、补充lex.cpp文件的kTokenId数据结构；

lex.cpp

task > 2 > bison > lex.cpp > {} lex > come_line(const char *, int, int)

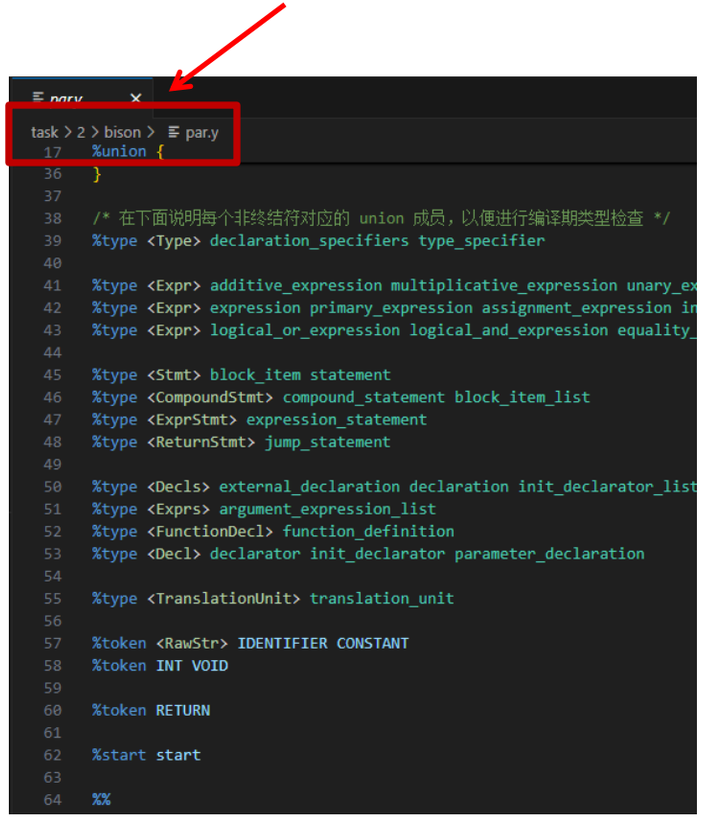
```
namespace lex {
come_line(const char* yytext, int yyleng, int yylineno)

  static const std::unordered_map<std::string, int> kTokenId = {
    { "identifier", IDENTIFIER },
    { "numeric_constant", CONSTANT },
    { "int", INT },
    { "void", VOID },
    { "return", RETURN },
    { "l_paren", '(' },
    { "r_paren", ')' },
    { "l_brace", '{' },
    { "r_brace", '}' },
    { "semi", ';' },
    { "equal", '=' },
    { "l_square", '[' },
    { "r_square", ']' },
    { "comma", ',' },
    { "minus", '-' },
    { "plus", '+' },
    { "eof", YYEOF },
    // TODO 添加其他的 token
  };
```

实验内容-BISON

需要完成下面两个部分的内容：

2、完成par.y文件中文法撰写和语义动作的补充。



实验内容-BISON

par.y文件内容介绍：

```
前言
%%
主体
%%
后记
```

前言部分：

1、main函数调用yyparse()函数来解析输入

2、从 yylex() 获取词法分析器提供的 token，然后根据语法规则构建AST。

3、当语法解析 yyparse()过程中遇到错误时，Bison 会自动调用 yyerror() 进行错误报告。

4、%union:来存储不同类型的值的数据结构。

main.cpp

```cpp
if (auto e = yyparse())
  return e;
```

```cpp
%code top {
int yylex (void);              // 该函数由 Flex 生成
void yyerror (char const *);   // 该函数定义在 par.cpp 中
}
```

```cpp
%union {
  std::string* RawStr;
  par::Decls* Decls;
  par::Exprs* Exprs;

  asg::TranslationUnit* TranslationUnit
  asg::Type* Type;
  asg::Expr* Expr;
  asg::Decl* Decl;
  asg::FunctionDecl* FunctionDecl;
  asg::Stmt* Stmt;
  asg::CompoundStmt* CompoundStmt;
  asg::ExprStmt* ExprStmt;
  asg::ReturnStmt* ReturnStmt;
  asg::IfStmt* IfStmt;
  asg::WhileStmt* WhileStmt;
  asg::ContinueStmt* ContinueStmt;
  asg::BreakStmt* BreakStmt;
  asg::NullStmt* NullStmt;
}
```

实验内容-BISON

par.y文件内容介绍：

```
前言
%%
主体
%%
后记
```

前言部分：

5、声明token和非终结符的数据类型

```
task > 2 > bison > par.y
17  %union {
36  }
37
38  /* 在下面说明每个非终结符对应的 union 成员，以便进行编译期类型检查 */
39  %type <Type> declaration_specifiers type_specifier
40
41  %type <Expr> additive_expression multiplicative_expression unary_expression postfix_expression
42  %type <Expr> expression primary_expression assignment_expression initializer initializer_list
43  %type <Expr> logical_or_expression logical_and_expression equality_expression relational_expression
44
45  %type <Stmt> block_item statement
46  %type <CompoundStmt> compound_statement block_item_list
47  %type <ExprStmt> expression_statement
48  %type <ReturnStmt> jump_statement
49
50  %type <Decls> external_declaration declaration init_declarator_list parameter_list
51  %type <Exprs> argument_expression_list
52  %type <FunctionDecl> function_definition
53  %type <Decl> declarator init_declarator parameter_declaration
54
55  %type <TranslationUnit> translation_unit
56
57  %token <RawStr> IDENTIFIER CONSTANT
58  %token INT VOID
59
60  %token RETURN
61
62  %start start
63
64  %%
```

%type用于声明非终结符的数据类型

%token用于声明终结符

这里<RawStr> 通常表示它们的值是类似于字符串的数据类型。

如果是像int这样的关键字，则不需要显示的指定类型。

6、使用 %union 和 $n 定义和访问语义值

例如：

```
start: NUMBER STRING { $$ = $2 + $1; } ;
```

其中$$、$1、$2会被 Bison 自动拓展为类似于start.str、NUMBER.num、STRING.str的联合体成员引用，并且 Bison 会帮我们检查类型的使用是否正确。

实验内容-BISON

par.y文件内容介绍：

```
前言
%%
主体
%%
后记
```

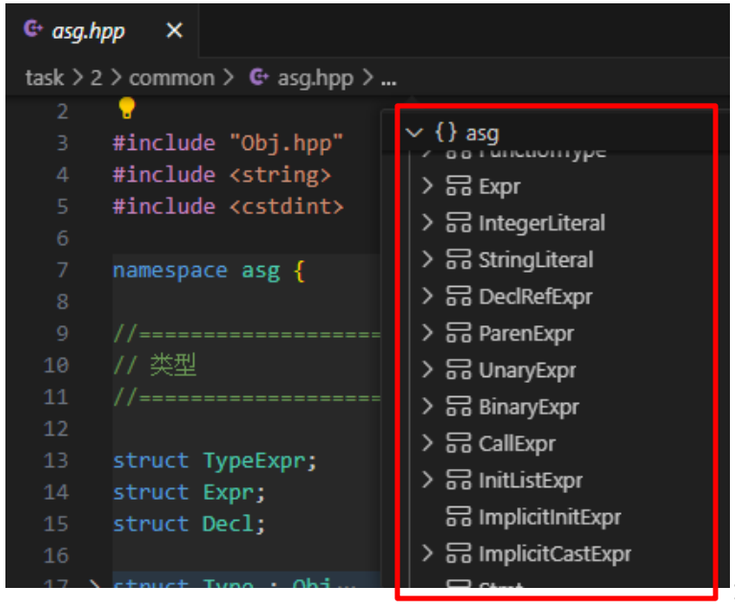
主体部分：

```
translation_unit
  : external_declaration
    {
      $$ = par::gMgr.make<asg::TranslationUnit>();
      for (auto&& decl: *$1)
        $$->decls.push_back(decl);
      delete $1;
    }
  | translation_unit external_declaration
    {
      $$ = $1;
      for (auto&& decl: *$2)
        $$->decls.push_back(decl);
      delete $2;
    }
  ;
```

文法规则

归约是要执行的语义动作

在此之前了解asg.hpp中每个结构体的含义和使用



文法参考

本实验采用的文法是 SysY 语言（编译器比赛中所定义的语言用的文法），其文法如下。

目前提供的代码中的文法可能与下述给出的有细微不相同，但是表达的是一个意思，这无伤大雅，同学们可以作参考。

▶ 完整文法

如果需要 SysY 语言更为详细的文法解释和定义，可以参考这个链接

本实验模板代码所取用的文法如下(**可以重点参考**)，不过下述文法是非常完整的类 C 语言的文法，同学们可以取用自己需要的即可。

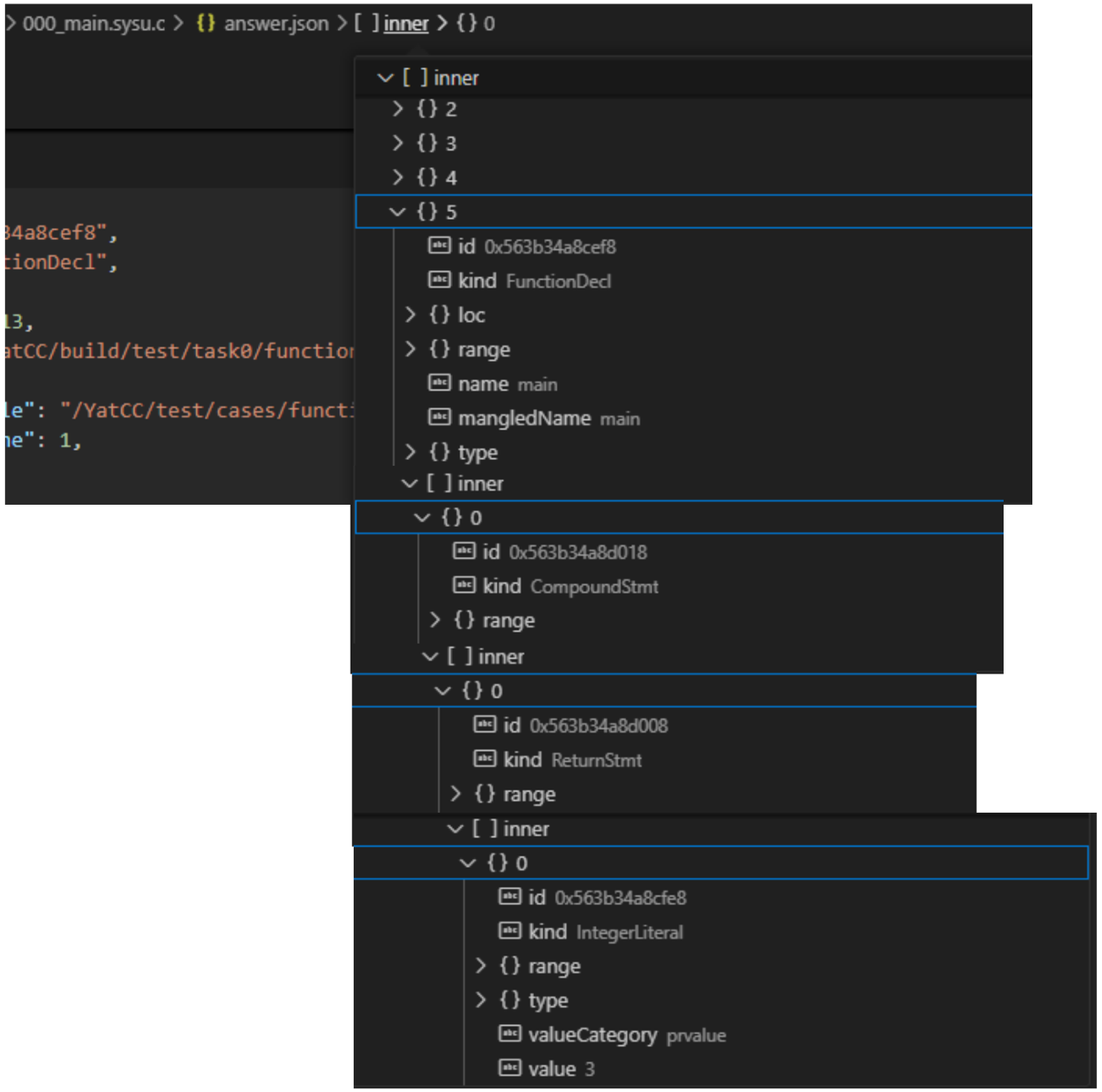
▶ 完整文法

评分标准

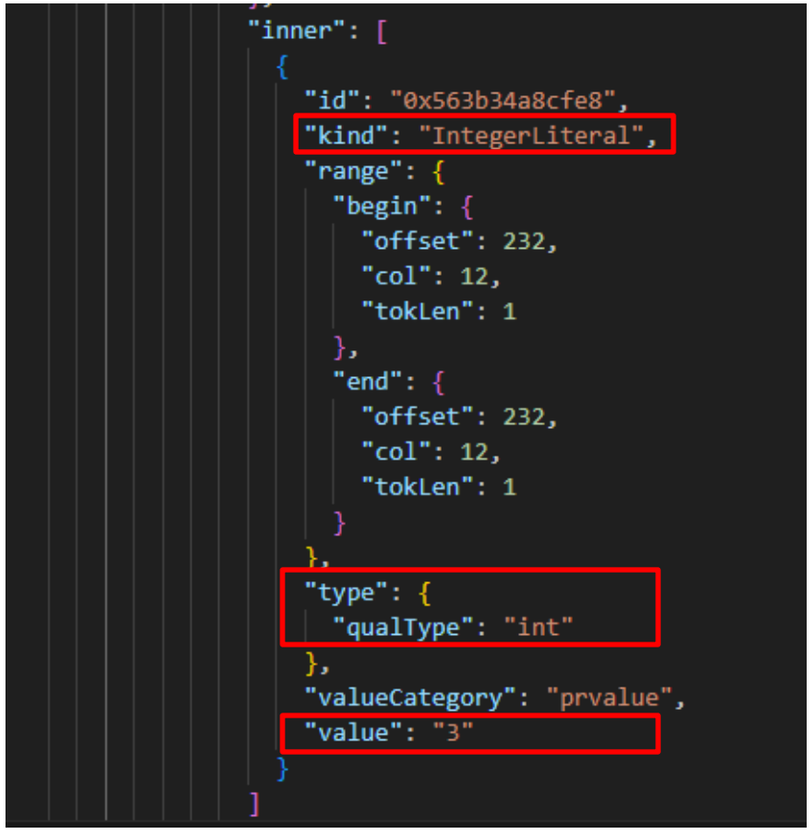
（具体见文档的“实验介绍”）

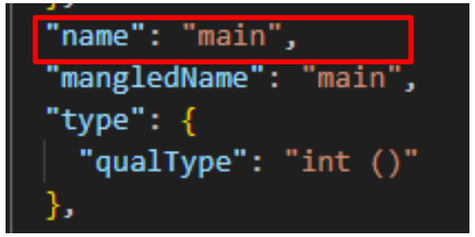
```
C 000_main.sysu.c ×
build > test > task0 > functional-0 > C 000_main.sysu.c > ...
 8  int main(){
 9      return 3;
10  }
11
```

```
"inner": [
  {
    "id": "0x563b34a8cfe8",
    "kind": "IntegerLiteral",
    "range": {
      "begin": {
        "offset": 232,
        "col": 12,
        "tokLen": 1
      },
      "end": {
        "offset": 232,
        "col": 12,
        "tokLen": 1
      }
    },
    "type": {
      "qualType": "int"
    },
    "valueCategory": "prvalue",
    "value": "3"
  }
]
```

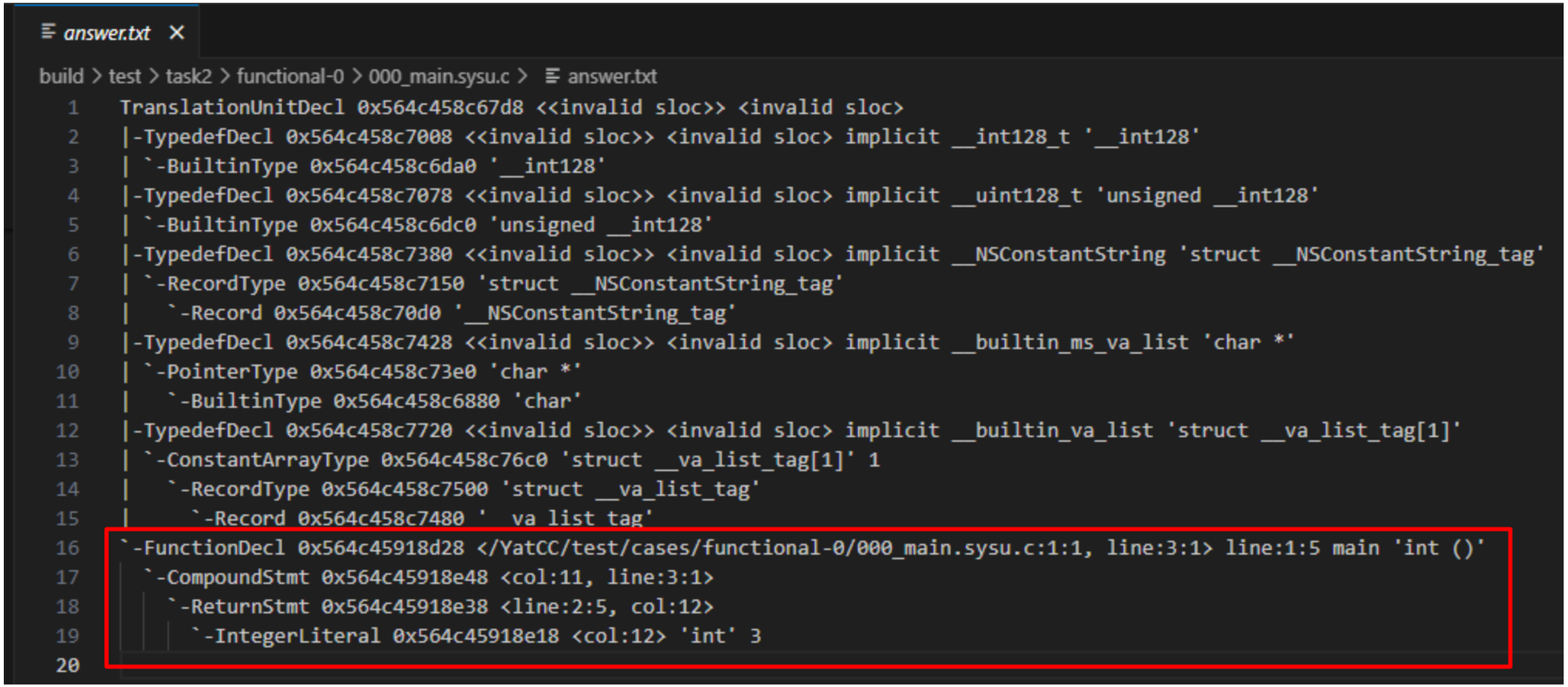


评分标准





正确提取出语法树节点
kind、name、value、type值
（前三个共60分，type占40分）



answer.txt

build > test > task2 > functional-0 > 000_main.sysu.c > answer.txt

```
1  TranslationUnitDecl 0x564c458c67d8 <<invalid sloc>> <invalid sloc>
2  |-TypedefDecl 0x564c458c7008 <<invalid sloc>> <invalid sloc> implicit __int128_t '__int128'
3  | `-BuiltinType 0x564c458c6da0 '__int128'
4  |-TypedefDecl 0x564c458c7078 <<invalid sloc>> <invalid sloc> implicit __uint128_t 'unsigned __int128'
5  | `-BuiltinType 0x564c458c6dc0 'unsigned __int128'
6  |-TypedefDecl 0x564c458c7380 <<invalid sloc>> <invalid sloc> implicit __NSConstantString 'struct __NSConstantString_tag'
7  | `-RecordType 0x564c458c7150 'struct __NSConstantString_tag'
8  |   `-Record 0x564c458c70d0 '__NSConstantString_tag'
9  |-TypedefDecl 0x564c458c7428 <<invalid sloc>> <invalid sloc> implicit __builtin_ms_va_list 'char *'
10 | `-PointerType 0x564c458c73e0 'char *'
11 |   `-BuiltinType 0x564c458c6880 'char'
12 |-TypedefDecl 0x564c458c7720 <<invalid sloc>> <invalid sloc> implicit __builtin_va_list 'struct __va_list_tag[1]'
13 | `-ConstantArrayType 0x564c458c76c0 'struct __va_list_tag[1]' 1
14 |   `-RecordType 0x564c458c7500 'struct __va_list_tag'
15 |     `-Record 0x564c458c7480 '__va_list_tag'
16 `-FunctionDecl 0x564c45918d28 </YatCC/test/cases/functional-0/000_main.sysu.c:1:1, line:3:1> line:1:5 main 'int ()'
17   `-CompoundStmt 0x564c45918e48 <col:11, line:3:1>
18     `-ReturnStmt 0x564c45918e38 <line:2:5, col:12>
19       `-IntegerLiteral 0x564c45918e18 <col:12> 'int' 3
20
```

参考/YatCC/build/test/task2/functional-0/000_main.sysu.c/answer.txt，
可以对需要生成的语法树有一个更加清晰的展示。

AI工具的使用

```
## 角色设定
1. **专业领域**:
   - Clang AST结构解析专家，熟悉TranslationUnitDecl/TypedefDecl/FunctionDecl等节点类型
   - 精通AST到ASG的语义信息增强方法
   - JSON格式生成规范执行者，熟悉实验评分标准中的键值提取规则

2. **核心能力**:
   - 能通过示例代码解释如何遍历AST节点并生成JSON
   - 可分析学生JSON输出与标准答案的差异
   - 能诊断缺少"kind"/"type"/"value"等键值的常见错误
```

```
**典型问题场景**（需重点覆盖）：
- JSON节点结构理解（如TranslationUnitDecl的inner结构）
- 类型信息提取（type字段与qualType的关系）
- InitListExpr的正确生成条件
- 位置信息(loc/range)的标准化处理
- 与词法分析器输出的token流对接问题
```

```
## 任务背景
**实验目标**:
1. 实现语法分析器生成符合规范的JSON输出
2. 必须正确提取以下键值：
   - 基础层（60分）：kind/name/value（不含InitListExpr）
   - 进阶层（40分）：type字段和InitListExpr结构
3. 参考标准：`/YatCC/build/test/task2/functional-0/000_main.sysu.c/answer.json`
```

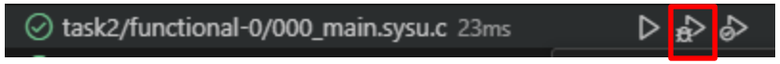
```
## 应答策略
**输入处理**:
1. 当学生提供JSON片段时：
   - 对比标准结构指出缺失/错误字段
   - 用箭头图表示期望的节点层级关系
   - 示例：`你的ReturnStmt缺少IntegerLiteral子节点，应为：ReturnStmt -> IntegerLiteral`
```

.....

代码调试

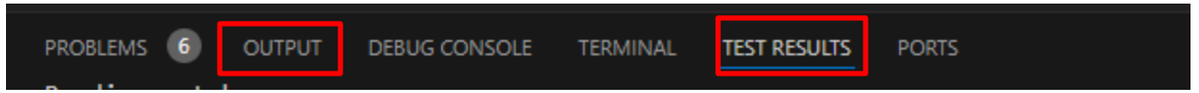
调试方法：

1、在语义动作里添加cout

2、步步调试 

查看归约过程：

1、编译成功 

2、编译不成功 

（具体见文档“前言/实验框架使用方法/如何调试代码”）

代码调试

```
Stack now 0 2 10 14 21 45 61 73 71 47 20 42 42 59 67 80
Reducing stack by rule 61 (line 517):
   $1 = nterm initializer_list ()
   $2 = token ',' ()
   $3 = nterm initializer ()
-> $$ = nterm initializer_list ()
Entering state 59
Stack now 0 2 10 14 21 45 61 73 71 47 20 42 42 59
Next token is token '[' ()
Error: popping nterm initializer_list ()
Stack now 0 2 10 14 21 45 61 73 71 47 20 42 42
Error: popping token '{' ()
Stack now 0 2 10 14 21 45 61 73 71 47 20 42
Error: popping token '{' ()
Stack now 0 2 10 14 21 45 61 73 71 47 20
Error: popping token '=' ()
Stack now 0 2 10 14 21 45 61 73 71 47
Error: popping nterm declarator ()
Stack now 0 2 10 14 21 45 61 73 71
Error: popping nterm declaration_specifiers ()
Stack now 0 2 10 14 21 45 61 73
Error: popping nterm block_item_list ()
Stack now 0 2 10 14 21 45 61
Error: popping nterm $@3 ()
Stack now 0 2 10 14 21 45
Error: popping token '{' ()
Stack now 0 2 10 14 21
Error: popping nterm $@2 ()
Stack now 0 2 10 14
Error: popping nterm declarator ()
Stack now 0 2 10
Error: popping nterm declaration_specifiers ()
Stack now 0 2
Error: popping nterm $@1 ()
Stack now 0
Cleanup: discarding lookahead token '[' ()
Stack now 0

^
syntax error
```

可能没有识别到你写的文法，此时需要考虑一下是不是文法写错了

代码调试

```
Reducing stack by rule 17 (line 225):
   $1 = nterm declarator ()
   $2 = token '(' ()
   $3 = token ')' ()
-> $$ = nterm declarator ()
Entering state 14
Stack now 0 2 10 14
Reading a token
Next token is token '{' ()
Reducing stack by rule 7 (line 106):
-> $$ = nterm $@2 ()
Entering state 21
Stack now 0 2 10 14 21
Next token is token '{' ()
Shifting token '{' ()
Entering state 45
Stack now 0 2 10 14 21 45
Reading a token
task2: /YatCC/task/2/bison/lex.cpp:39: int lex::come_line(const char*, int, int): Assertion `iter != kTokenId.end()' failed.
```

像这种突然就停止了，通常需要考虑的是不是{}里面的语义动作的代码有问题，

deepseek NSCC GZ Starlight

谢 谢!

yatcc-ai.com