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

DCS292：编译器构造实验(Compiler Construction)

Task 2：语法分析Bison

Yat Compiler Construction with AI

yatcc-ai.com

张献伟、YatCC团队

中山大学　计算机学院

国家超级计算广州中心

2025.3.23

实验概述：从Token流到ASG

整体流程

源代码 /test/cases/**/*.sysu.c → Flex task1:词法分析/clang → **Bison** task2:语法分析 → AST/ASG 抽象语法树/图 → JSON /build/test/task2/**/output.json

补充lex.cpp

/build/test/task1/**/answer.txt

参考task1的标准输出**answer.txt**，补充**kTokenId**数据结构，保证所有可能出现的token类型都被正确定义。

task > 2 > bison > lex.cpp

```
int
come_line(const char* yytext, int yyleng, int yylineno)
{
  char name[64];
  char value[64];
  sscanf(yytext, "%s '%[^']'", name, value);

  static const std::unordered_map<std::string, int> kTokenId = {
    { "identifier", IDENTIFIER },
    { "numeric_constant", CONSTANT },
    { "int", INT },
    { "void", VOID },
    { "return", RETURN },
```

完成par.y

撰写**文法规则**，补充**语义动作**，创建并填充ASG节点。

par.y

task > 2 > bison > par.y

```
%type <TranslationUnit> translation_unit

%token <RawStr> IDENTIFIER CONSTANT
%token INT VOID

%token RETURN

%start start

%%

// 起始符号
start
  : {
```

代码结构：Bison框架文件组织

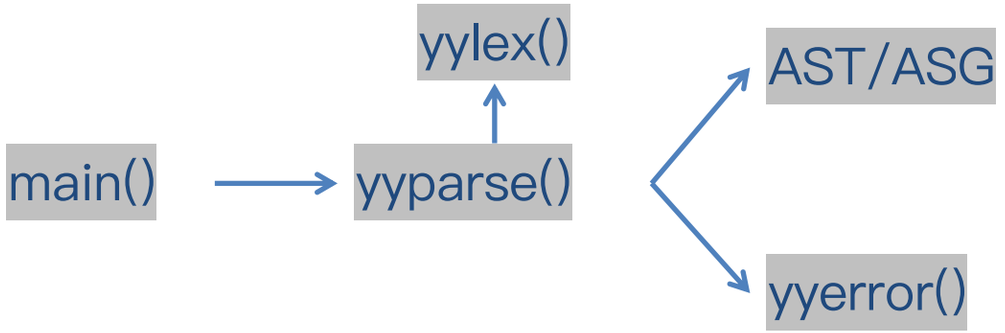
bison/ 目录

- **lex.cpp / lex.hpp**
  词法分析辅助函数
- **lex.l**
  flex词法规则
- **main.cpp**
  程序入口
- **par.cpp / par.hpp**
  语法分析辅助函数
- **par.y**
  Bison语法规则

common/ 目录

asg.hpp	Typing
Asg2Json	Obj

执行流程



补充lex.cpp的kTokenId

任务说明

参考task1的标准输出**answer.txt**，补充**lex.cpp**文件中的**kTokenId**数据结构。

目标：保证answer.txt中可能出现的所有token类型，都在kTokenId中被定义。

Token流格式

```
// answer.txt 示例
int 'int' [StartOfLine]
identifier 'main'
l_paren '('
r_paren ')'
l_brace '{'
return 'return'
numeric_constant '3'
semi ';'
eof ''
```

lex.l中的处理

```
// lex.l 核心规则
^#[^\n]* /* 屏蔽#开头的行 */
<> { return YYEOF; }
.*\n { COME_LINE(); }
. { COME(YYUNDEF); }
```

COME_LINE()宏会调用lex.cpp中的**come_line()**函数，提取每行的第一个单词（tokenId）和第二个单词中的引号内容（tokenValue）。

实现提示

- 使用task1的flex实现的同学对此会比较熟悉
- 建立tokenId字符串到Bison token的映射
- 匹配失败会 Assertion failed

Bison简介：语法分析器生成工具

核心概念

Bison是**语法分析器生成工具**，不是语法分析器本身。它根据你提供的语法规则，生成符合规则的语法分析器。

类比理解：flex是"词典"，Bison是"语法老师"。flex识别单词，Bison理解句子结构。

.y文件结构

```
// 前言：声明、头文件
%union { ... }
%token ...
%type ...

%%
// 主体：文法规则
start: translation_unit;
expr: expr '+' term;

%%
// 后记：辅助函数
```

配合流程

- flex对文件进行正则匹配，生成token流
- 每个token传给Bison进行语法分析
- Bison执行移进归约，执行语义动作

语义值机制

```
17  %union {
18      std::string* RawStr;
19      par::Decls* Decls;
20      par::Exprs* Exprs;
21
22      asg::TranslationUnit* T
23      asg::Type* Type;
24      asg::Expr* Expr;
```

%union

定义所有可能的语义值类型

$n / $$

访问第n个符号/左部符号的语义值

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

par.y文法撰写

前言部分

1. 定义终结符与非终结符
2. 使用 union 和类型定义符号的语义值
3. 定义其他逻辑与规则

声明符号类型

```
// %token: 终结符
%token INT VOID CONST
%token <RawStr> IDENTIFIER CONSTANT
// %type: 非终结符
%type <ExprStmt> expression_statement
%type <Stmt> block_item statement jump_statement
%type <Decls> external_declaration
```

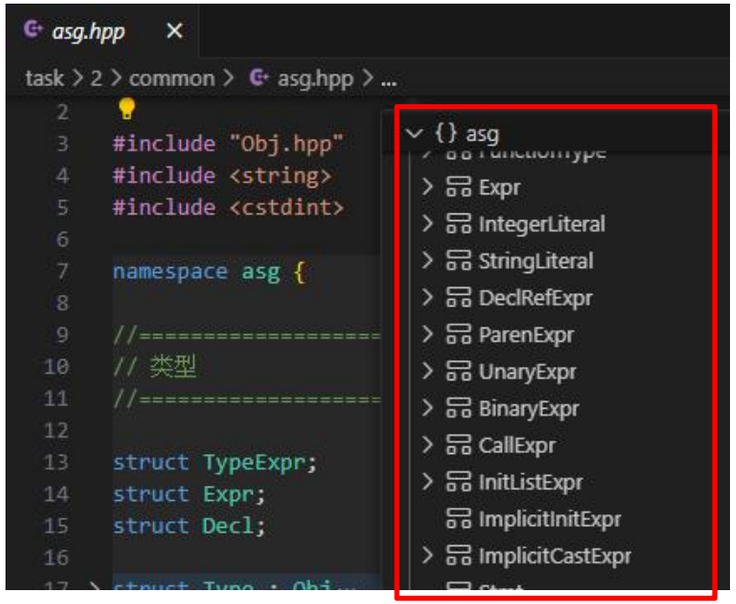
%union 定义

```
17  %union {
18      std::string* RawStr;
19      par::Decls* Decls;
20      par::Exprs* Exprs;
21
22      asg::TranslationUnit* TranslationUnit;
23      asg::Type* Type;
24      asg::Expr* Expr;
25      asg::Decl* Decl;
```

主体：文法规则

语义动作与ASG节点

ASG节点类型



语义动作示例

```
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;
    }
  ;
```

重要提示

- 仔细阅读asg.hpp，理解每个结构体的成员含义
- 不清楚结构体用法时，查看asg2json.cpp中的打印方式
- 根据asg.hpp推断需要实现哪些产生式

语义值符号

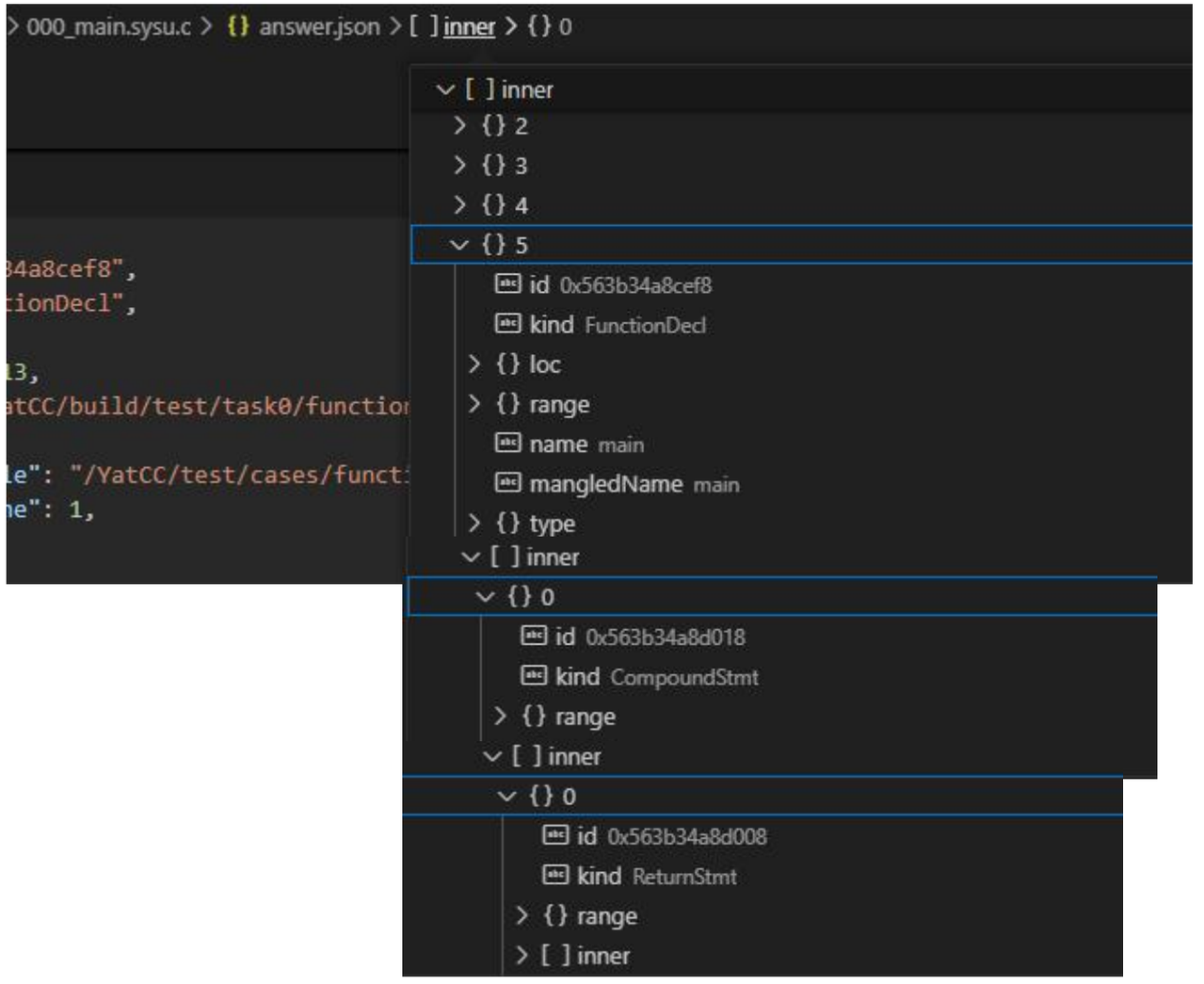
$$	$1 $2
左部符号语义值	右部第n个符号

查看实验结果

/YatCC/build/test/task2/function-0/000_main.sysu.c/answer.json

```
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：

1. kind、name、value（60分）
2. type值（40分）

P.S. 今年新增了隐藏测例（不超范围、）

最终Task2成绩=本地评分*0.95+隐藏测例评分*0.05

评分标准

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工具使用

理论与实验步骤

YatCC 文档助手

代码上手与调试

Roo Code

AI工具使用

```
## 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`
```

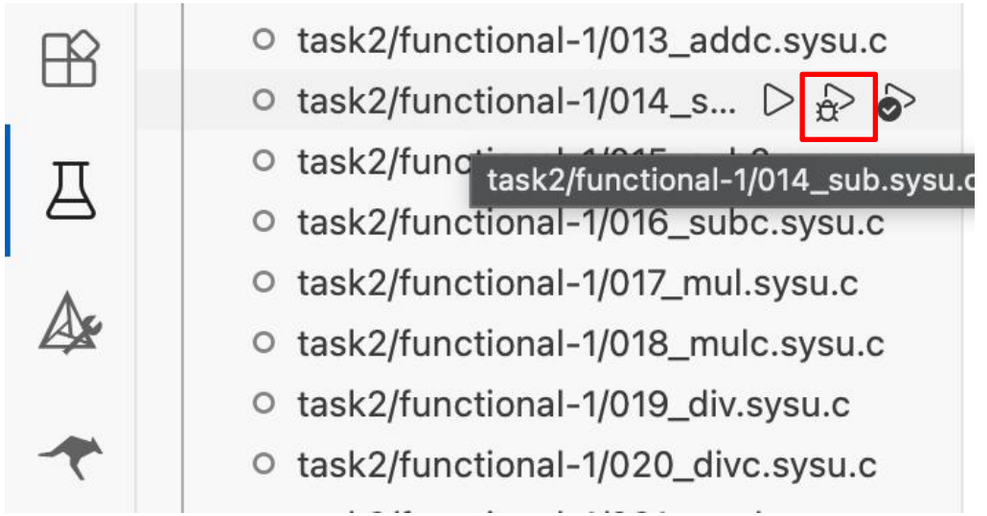
可以使用 meta prompt 生成prompt

代码调试

调试方法：

1、在语义动作里添加cout

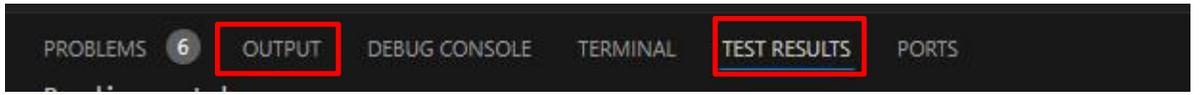
2、步步调试



查看归约过程：

1、编译成功

2、编译不成功



谢　谢！