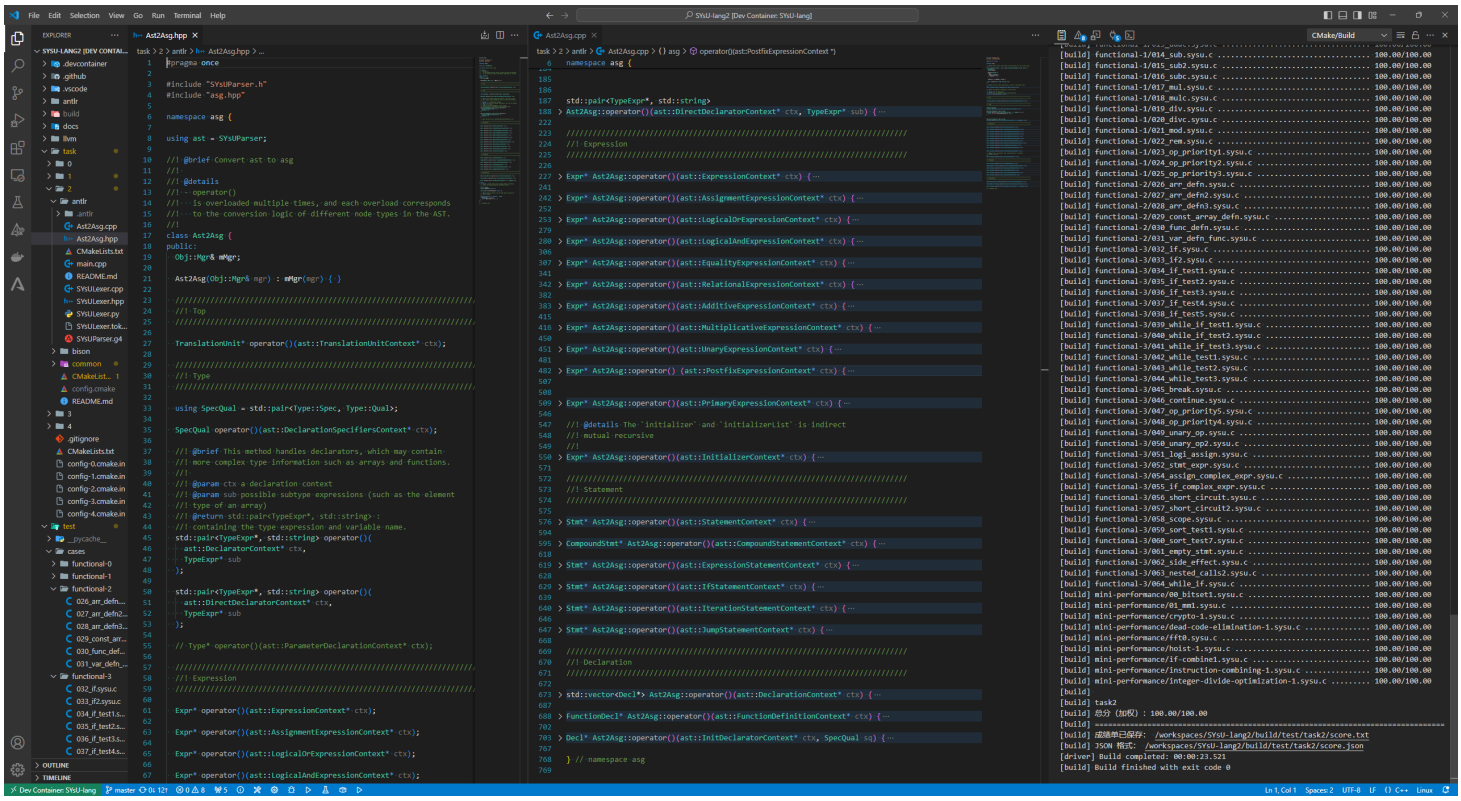
Compilers project 1

姓名	学号	日期
丁艺桦	21307301	2024-04-22

1. 运行结果截图



2. 实验感想以及建议

整个实验做完觉得很累。

做的过程中主要是完成实验需要的信息比较分散觉得累。比如 `breakStmt` 和 `continueStmt` 中的 loop 指针，想了很久要怎么设置，知道同学告诉我在群里说了那个不需要处理（但是早已被其他消息刷掉了）

还有印象最深的是最后八十多分的时候就只差函数多参数那些例子是损坏的。一直以为是函数多参数定义搞错了，但是翻来覆去查了一个多小时也没找到问题在哪。最后是去问了同学才知道，其实是我 `postfixExpr` 的处理搞错了， `callExpr` 中的参数表，应该放入的是 `assignmentExpr` 的解析，但是我放的是其上层 `expression` 的解析。区别是：前者的二元运算符是 `kAssign` ，后者的二元运算符是 `kComma` 。就是因为这细微的差别，让输出文件都损坏了（尽管这两个处理函数返回的指针都是 `Expr*` ）。

另外，在实验开始时，现有的代码里面有一些风格不统一的问题，比如有些处理函数明明不需要使用 children 以及转类型但是用了（在别处这种简单的返回则没有使用 children + cast）：

```
327  Expr* Ast2Asg::operator()(ast::PostfixExpressionContext* ctx) {
328      auto p = ctx->primaryExpression();
329      return self(p);
330      // auto children = ctx->children;
331      // auto sub = self(dynamic_cast<ast::PrimaryExpressionContext*>(children[0]));
332      // return sub;
333  }
```

（注释是原本提供的代码，新增的 2 行是我自己修改的）。虽然是小毛病，但是给人的体验就不是那么的舒适。

最后再吐槽一下实验文档，虽然这次新增了如何上手的部分，确实比较友好。但是前面的项目说明中，出现**疑似 GPT 生成的内容**：

下面对这个文件进行一些说明：

首先是对引号内的关键词进行解释,但是实际上我们需要关心的只有 `kind` , `name` , `value` , `type` 等几个。

- id: 唯一标识符，用于区分AST中的每一个节点。
- kind: 节点类型，表示该节点代表的源代码结构的种类，如TypedefDecl（类型定义声明）、BuiltinType（内置类型）、FunctionDecl（函数声明）等。
- loc: 位置信息，通常包含文件名、行号和列号，用于指示源代码中该元素的位置。在此例中，位置信息似乎被省略了。
- range: 范围信息，指出了源代码中该节点覆盖的起始和结束位置。它有begin和end两个属性，每个属性可能包含偏移量、行号、列号等信息，用于准确定位代码片段。
- inner: 内部节点，这个列表包含了当前节点下的子节点。例如，一个FunctionDecl节点会包含它的参数和函数体等子节点。
- isImplicit: 表示该声明是否是隐式的，即没有在源代码中直接写出来，而是由编译器自动生成的。
- name: 节点名称，比如类型名称、函数名称等。
- type: 节点类型，包含了类型信息，如__int128、unsigned __int128等。对于类型节点，qualType属性描述了类型的完整限定名。
- decl: 声明信息，某些节点（如RecordType）可能包含对声明本身的引用。
- size: 大小，主要用于数组类型，表示数组的元素数量。
- valueCategory: 值类别，如prvalue，表示纯右值。
- value: 节点值，对于字面量如整数字面量，这个字段包含了具体的值。

（上图中蓝色选中部分）

根据我长期的使用经验，这个说话方式基本上八九不离十是 GPT 生成的，我就觉得没什么必要。与其复制粘贴 GPT 的回答，比如在文档里面教我们怎么用合适的 prompt 和一连串追问问题去把一个项目结构和代码问明白。