**编译原理与设计**

**实验报告**

实验名称： Lab 6：语义分析实验

姓名/学号： 李昊阳/1120203053

1. **实验目的和内容**

**实验目的：**

（1）熟悉 C 语言的语义规则，了解编译器语义分析的主要功能；

（2）掌握语义分析模块构造的相关技术和方法，设计并实现具有一定分析功能的 C 语言语义分析模块；

（3）掌握编译器从前端到后端各个模块的工作原理，语义分析模块与其他模块之间的交互过程。

**实验内容：**

语义分析阶段的工作为基于语法分析获得的分析树构建符号表，并进行语义检查。如果存在非法的结果，请将结果报告给用户，其中语义检查的内容主要包括：

* 变量使用前是否进行了定义；
* 变量是否存在重复定义；
* break 语句是否在循环语句中使用；
* 函数调用的参数个数和类型是否匹配；
* 函数使用前是否进行了定义或者声明；
* 运算符两边的操作数的类型是否相容；
* 数组访问是否越界；
* goto 的目标是否存在；
* ……

本次语义检查的前（1）-（3）为要求完成内容，而其余为可选内容。

具体步骤如实验指导文档所示。

1. **实验环境**

设备：RedmiBook 14 锐龙版

操作系统：Windows 10 Pro, 64-bit (Build 19045.2604) 10.0.19045

Java：jdk-17

Java IDE：IntelliJ IDEA 2022.2.5 (Community Edition)

Antlr：ANTLRv4 4.11.1

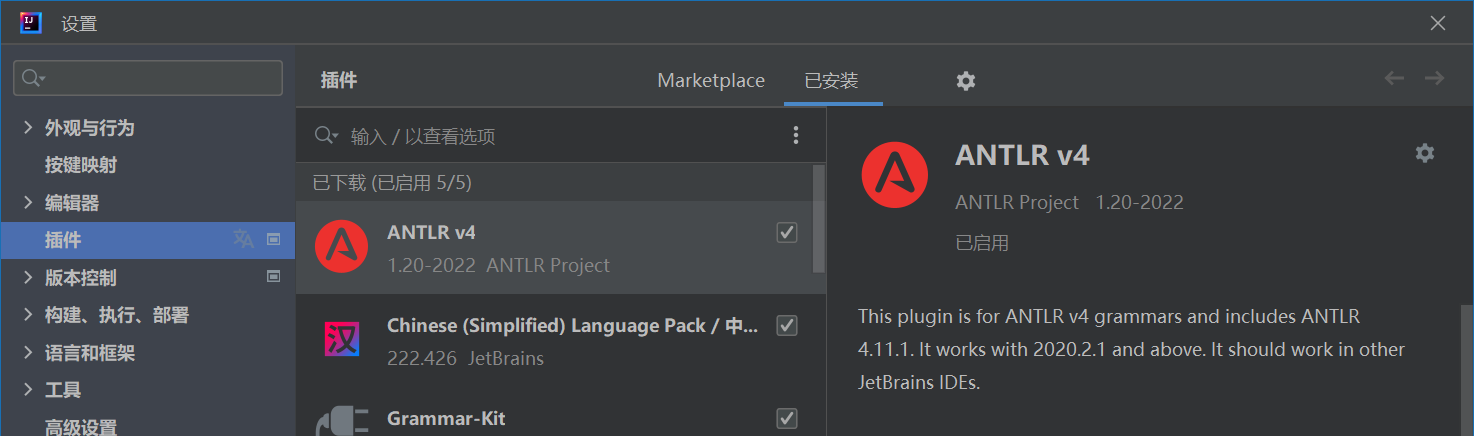
1. **实现的具体过程和步骤**

按照实验文档的指示，在语法分析实验的基础上，构建符号表，并基于符号表和抽象语法树进行语义检查。要求对于给定的程序，输出检测到的语义错误信息。要求输出信息中包含错误编号，且错误编码与下表一致，错误信息可以自行定义。具体表格见实验文档。

由于上次语法分析实验是基于BIT-MiniCC框架做的语法分析并生成语法树，具体代码繁琐而复杂、语法树定义不规范，因此不适宜用作本次语义分析实验的语法分析树。综合考虑后，使用IntelliJ + Antlr插件的方式，进行语义分析实验。

Antlr (ANother Tool for Language Recognition) 是一款强大的语法分析器生成工具，可用于读取、处理、执行和翻译结构化的文本或者二进制文件。它被广泛应用于学术领域和工业生产实践，是众多语言、工具和框架的基石。我们可以使用Antlr来开发DSL（Domain Specific Language，领域特定语言），或者一些实用工具，比如配置文件读取器、遗留代码转换器和Json解析器等等。

1. 安装Antlr插件并构建项目

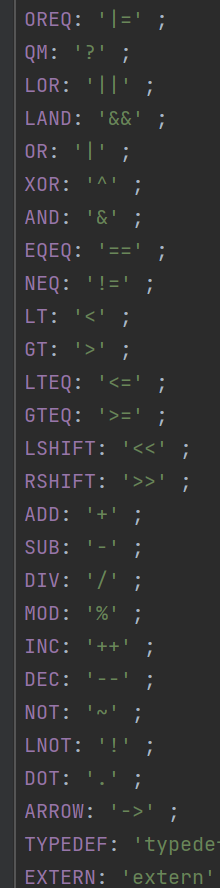
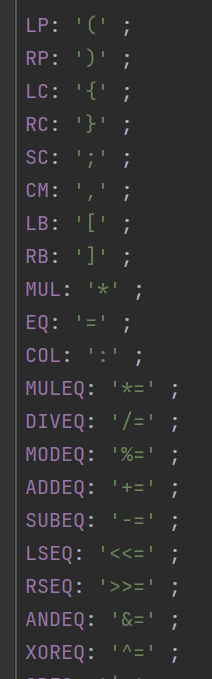
在idea中安装Antlr插件：

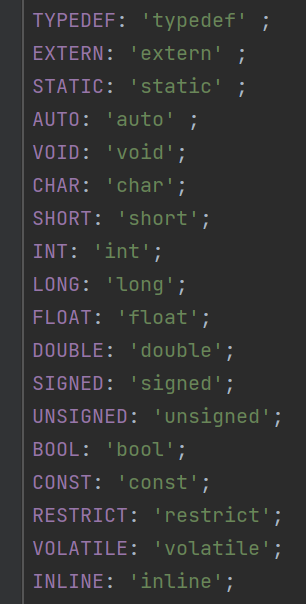
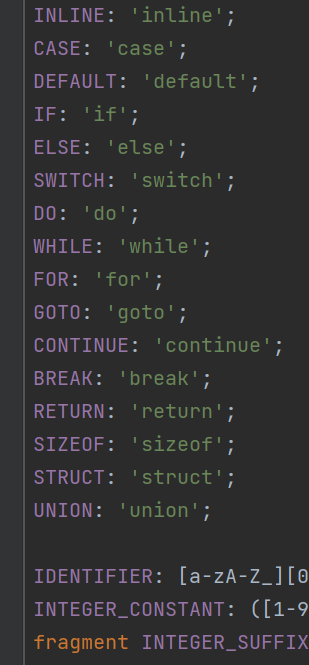
随后新建Maven项目，在pom.xml中添加有关Antlr的依赖：

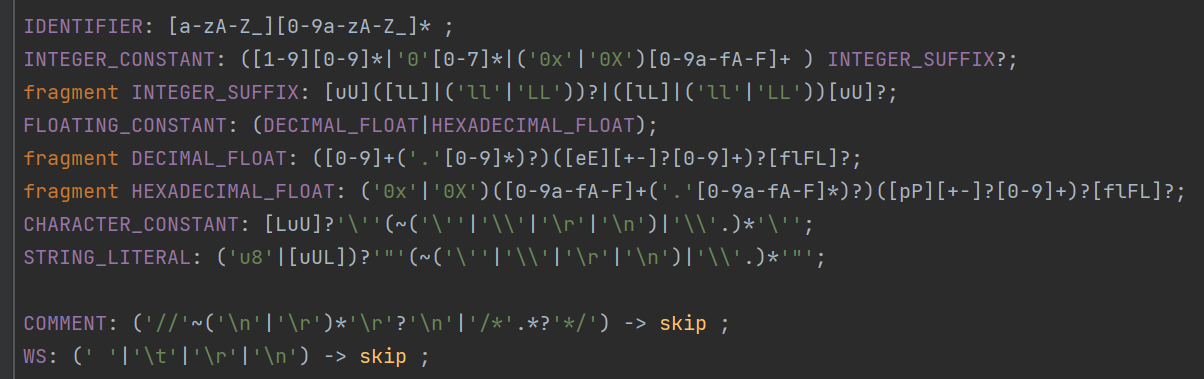
 新建lab6.g4语法与词法的规则文件，新建主控程序Main类。

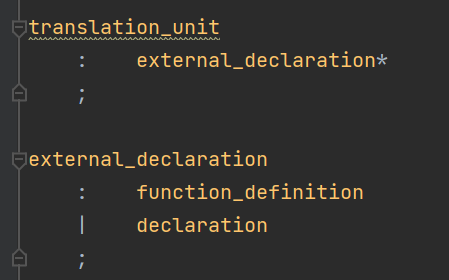
1. 编写lab6.g4文件

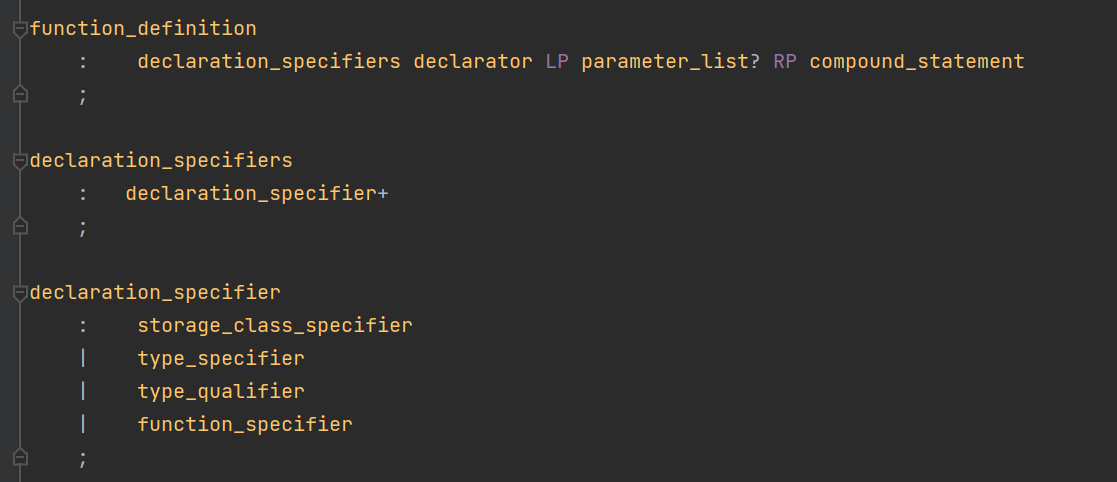
此前实验都是通过jflex、BIT-MiniCC、或AntlrWork进行编写的，对C语言的描述不详尽或不规范，因此需重新编写词法与语法规则。

词法部分，首先定义终结符，对所有标点符号命名：

 定义关键字：

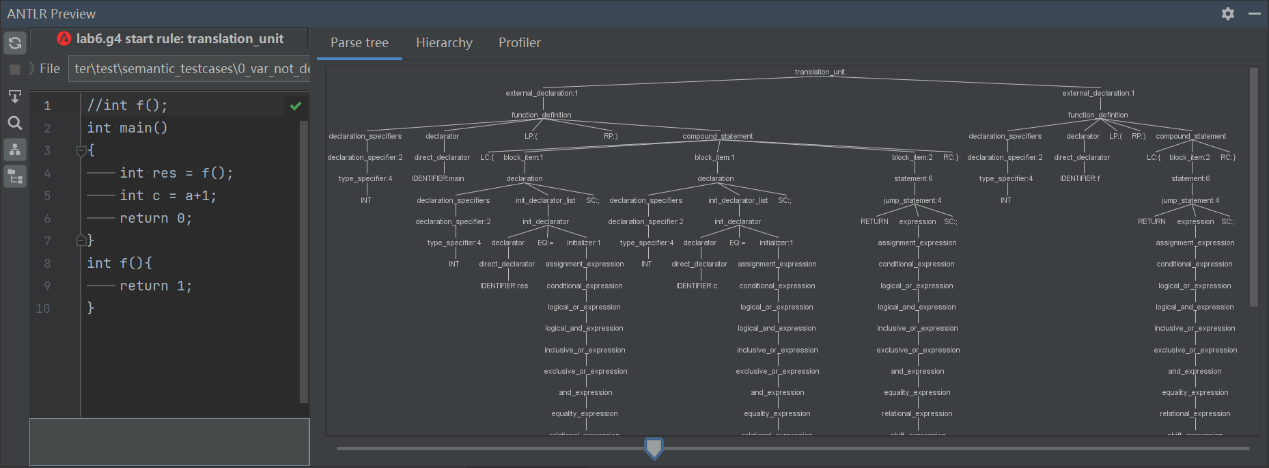
 然后定义标识符、常数、字符串、注释以及空白字符等等：

 语法部分，参考C11-n1570文档，实现了绝大部分C语言语法，并将其改写为了Antlr能够运行的非左递归或间接左递归的文法。Antlr支持正则语言，因此将C语言文法进行了简化，如使用‘+’、‘\*’等正则重复语句对递归进行简化，使用‘?’视为可选项opt等。以下为部分语法：

 translation\_unit为语法分析入口，即语法分析树根结点。

其余语法见lab6.g4。

1. 生成Antlr语法树

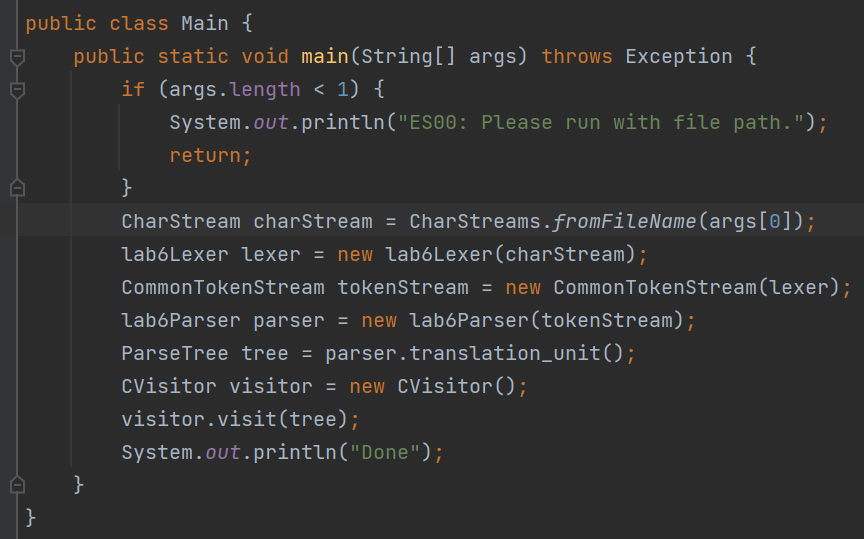
在translation\_unit处右键Test Rule translation\_unit，若输入的文本或文件的语法无误，则会正确生成语法分析树，以测试文件0\_var\_not\_defined.c为例，生成的语法分析树如下所示：

右键lab6.g4，选择Generate ANTLR Recognizer，即可生成后续分析所需的lexer、parser与visitor类

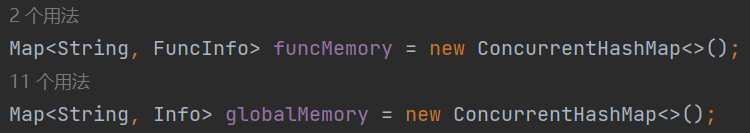
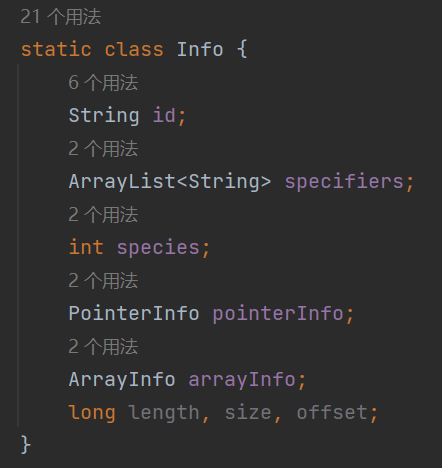
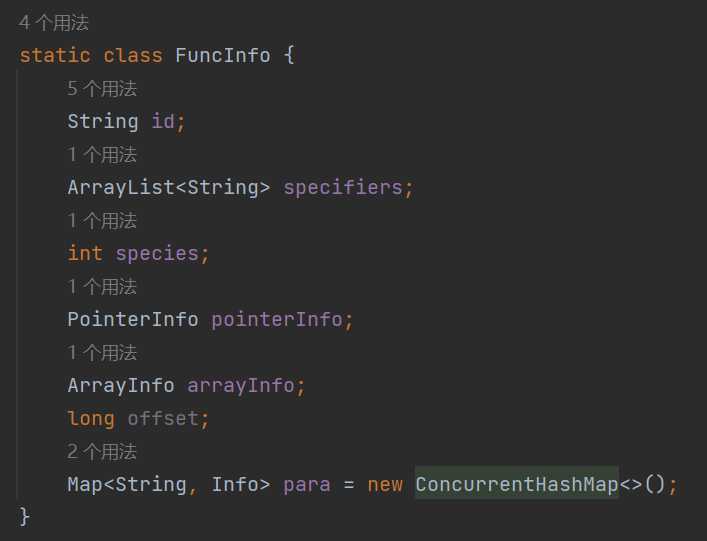
1. 编写主函数，通过Antlr提供的visitor进行语义分析

Antlr提供了两种方法来访问ParseTree：一种是通过Parse-Tree Listener的方法；另一种是通过Parse-Tree Visitor的方法。两种方法最大的区别是Listener会自动访问 AST 的所有节点，而Visitor如果要访问当前节点的子树，则需要手工实现。

为了实现特定的语义分析需求，选择重写visitor类：定义CVisitor类，继承自lab6.g4生成的lab6BaseVisitor<T>泛型类，因为返回值较为复杂，令泛型为Object，便于后续属性值的传递。

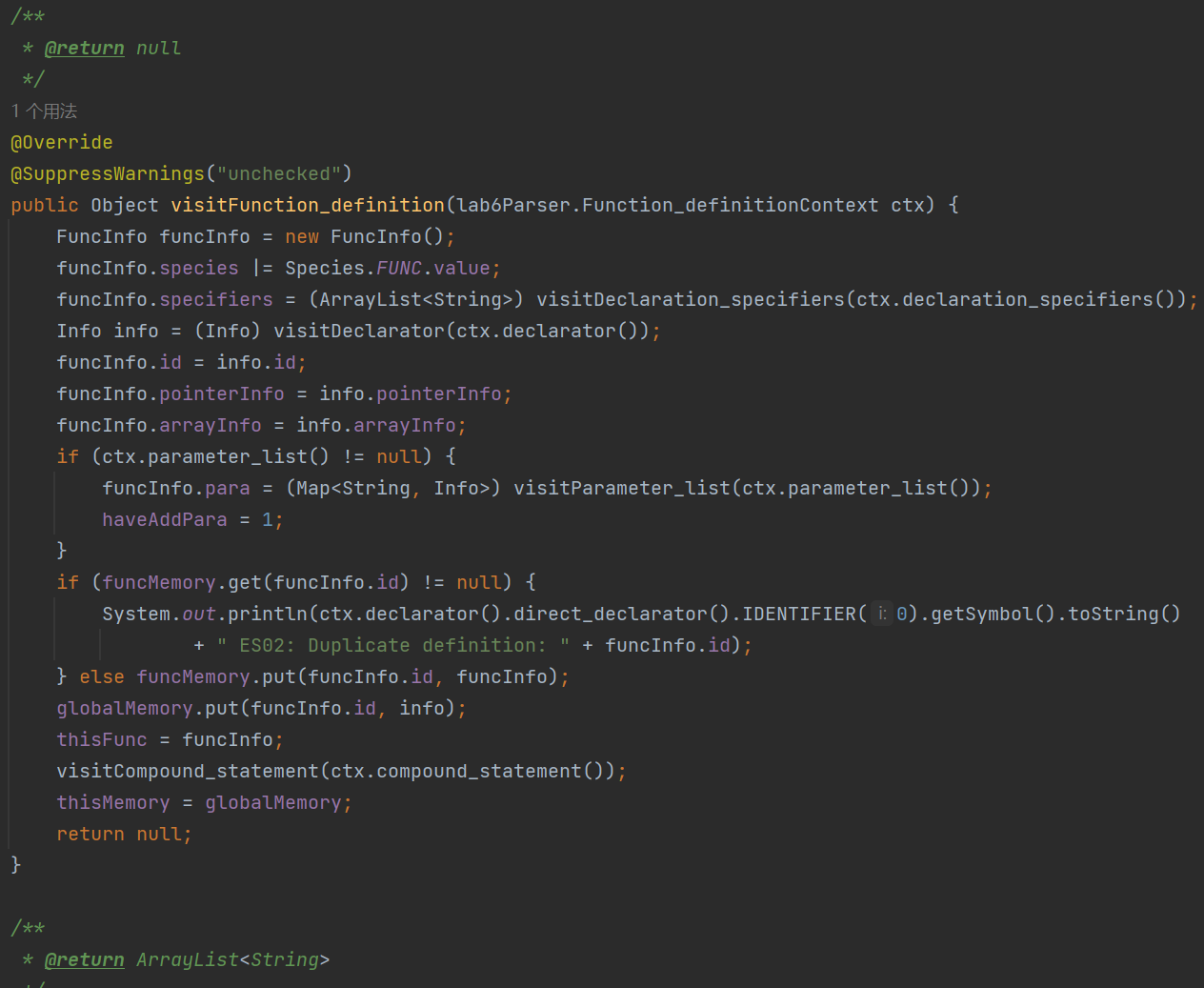
在Main类中，以文件路径作为参数输入，分别进行lab6Lexer的词法分析、lab6Parser的语法分析、ParseTree生成语法树，然后通过CVisitor遍历该语法树。

1. 编写CVisitor，实现语义检查(1)-(3)

首先编写Info与FuncInfo内部类，用来保存变量定义与函数定义，并在全局设置funcMemory与globalMemory，即全局的函数与变量符号表，以Map<key,value>的形式保存，具体以ConcurrentHashMap<>实现。

接着重写大部分visit函数，以实现遍历语法树，同时进行语义检查的功能。

下面以几个主要引发语义检查错误的visit函数为例：

* 1. visitFunction\_definition

调用该函数时，会访问上下文（context）指向的函数结点。

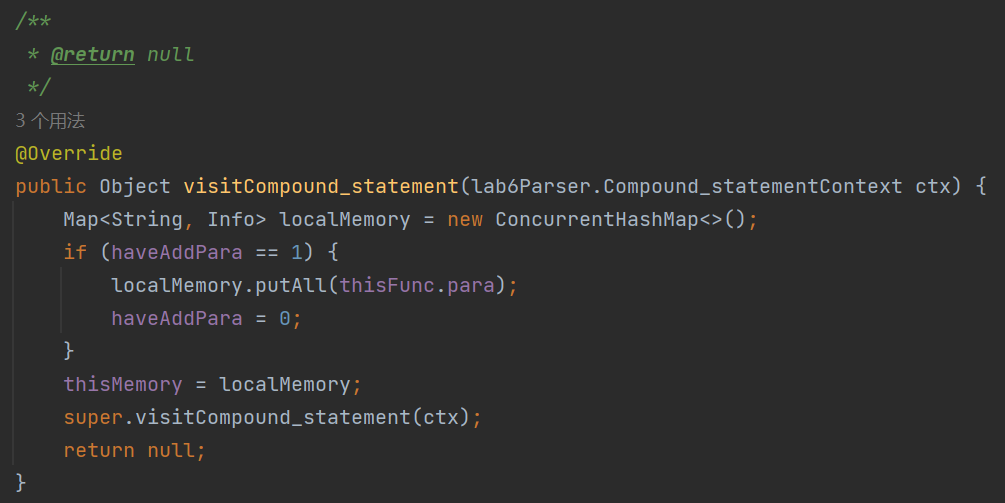
先实例化一个FuncInfo类；调用visitDeclaration\_specifiers确定函数的修饰符，将修饰符给出的属性写入funcInfo中；然后调用visitDeclarator确定函数ID以及指针、数组信息；若函数拥有参数列表，则访问参数列表visitParameter\_list，将参数保存在funcInfo的para符号表中，便于查询。

在将funcInfo放入全局函数符号表前，需确定是否有重名函数，若有，则输出ES02错误信息，表示函数重定义，并指出重定义标识符的token信息。

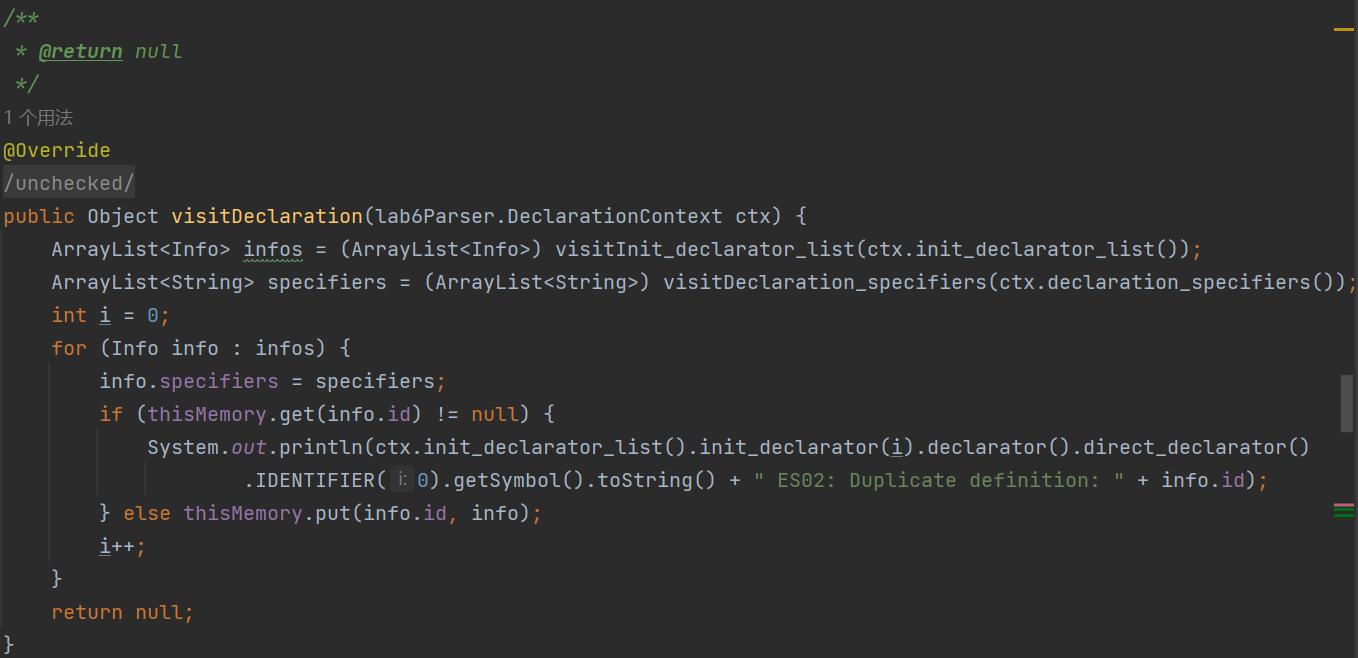
最后访问其函数主体visitCompound\_statement。

* 1. visitCompound\_statement

调用该函数时，会访问上下文指向的复合语句块。

 因为C语言具有Scope属性，对于每一个复合语句块，其内层 Scope 里面的符号对外层是不可见的。因此需单独创建一个局部符号表localMemory。若该复合语句为函数主体，则将para添加至局部符号表中，便于后续查询。

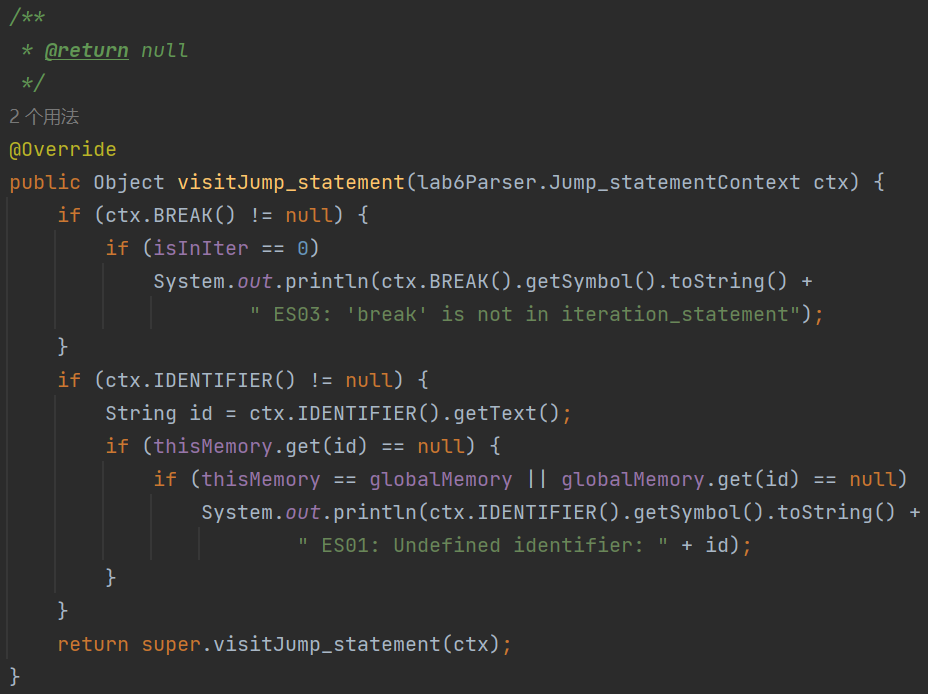
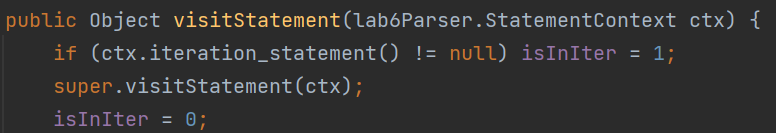
* 1. visitDeclaration

调用该函数时，会访问上下文指向的声明。

该函数会对于声明中的每个变量都实例化一个Info，对于每个Info，若其所在的Scope符号表中有相应变量，则输出ES02错误信息，表示变量重定义，并给出对应变量标识符的token信息；否则将其放入符号表中。

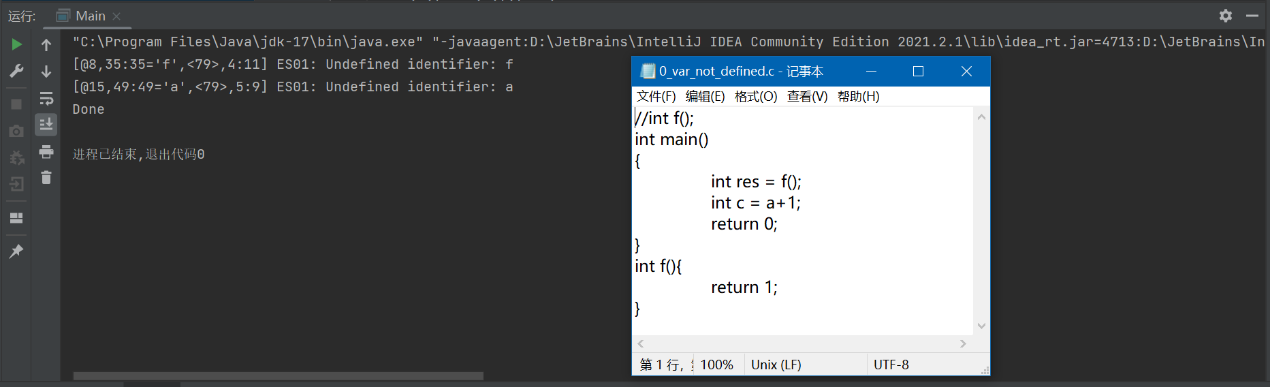
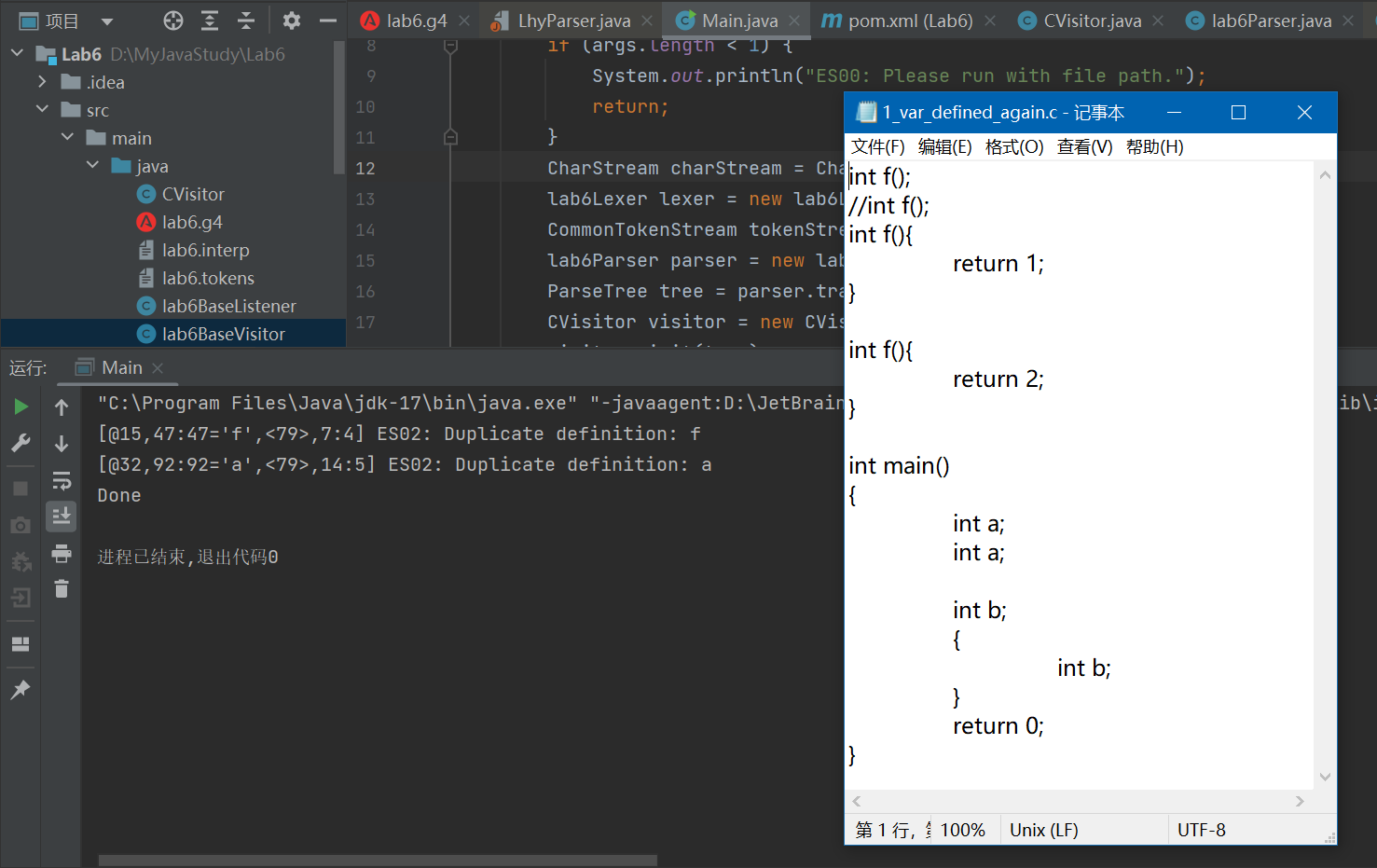
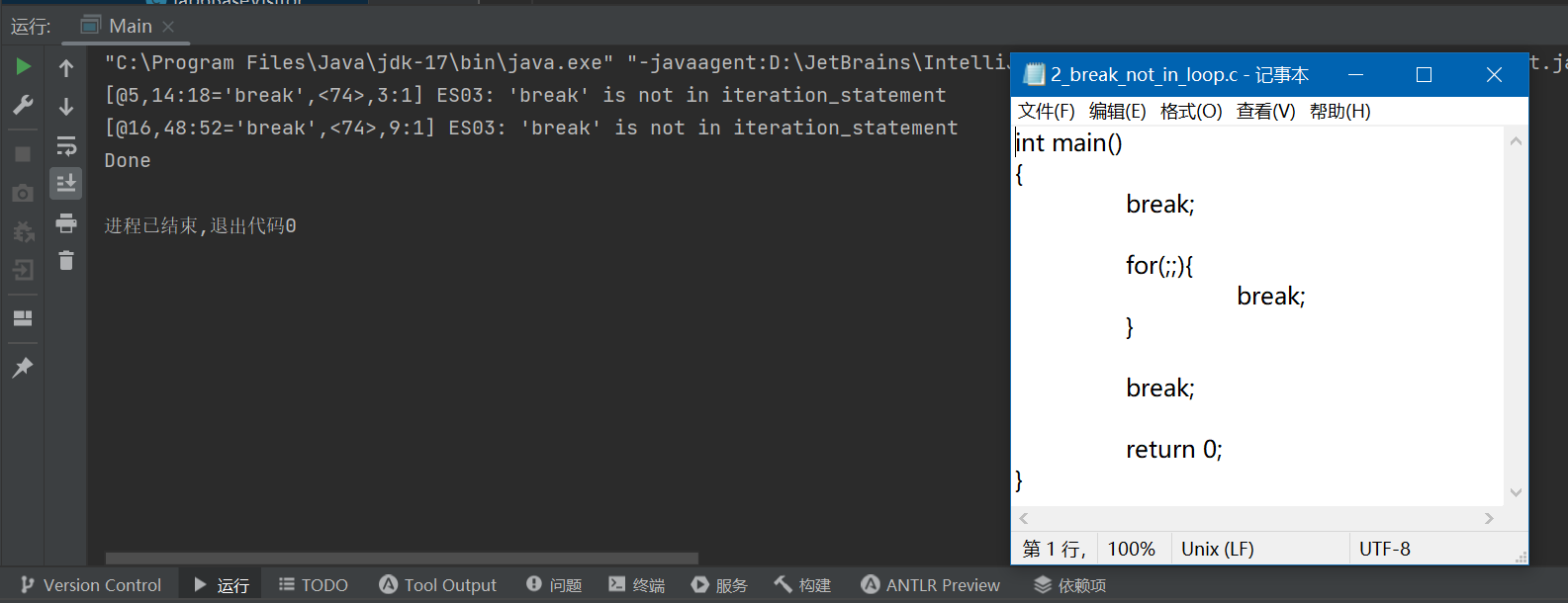
* 1. visitJump\_statement

调用此函数时，会访问上下文指向的跳转语句。

 需注意，在访问Statement结点时，若子结点是循环语句，则会将isInIter的标识置1，以表示目前处于循环语句中，访问结束后，标识重置为0：

因此，在跳转语句中，若有BREAK且isInIter为0，则输出ES03错误信息，表示该BREAK语句不在循环语句中，并指出该BREAK对应终结符的token信息。

除此之外，在跳转语句中，若有标识符，则查询所属Scope的符号表，若无该变量，则查询更外层Scope的符号表，直至全局符号表。若均无信息，则输出ES01错误信息，表示该变量使用前未定义，并指出该变量对应标识符的token信息。其他含有标识符的visit函数均同上进行语义检查。

1. **运行效果截图**
2. 0\_var\_not\_defined.c
3. 1\_var\_defined\_again.c
4. 2\_break\_not\_in\_loop.c
5. **实验心得体会**

本次实验是编译原理与设计的第六次实验。本次实验我选择使用了IntelliJ+Antlr工具，更加规范化了词法、语法与语义方面的内容。本次实验让我第一次认识到了语义分析过程，再一次深入地阅读了C语言的规范文件。从访问者与使用者的角度上，深入了解了语法树的构成与Visitor模式。

本次实验相当于词法、语法、语义三次实验的大统一，我再一次从头完成了以前多个实验。由于以前实验的生涩与困难，我走了许多弯路，但总结与收获了许多经验，便在本次实验上，我以极快的速度完成了之前的工作，更加轻车熟路。我细细研究了Antlr的Visitor模式，仔细琢磨该如何重写方法，了解其底层是如何实现访问的，考究函数的返回与泛型的使用，最终完成了本次实验。收获颇丰，是一次非常有意义的实验。

但是由于精力与时间问题，本次语义检查内容只完成了前三项，但许多方法的重写已为后续的语义检查打下了基础，Info中也预留了多项属性未曾使用。我相信其他检查功能的添加也仅仅是时间问题。