中国科学院大学
《编译原理（研讨课）》实验报告

姓名 吴修齐 学号 2022K8009922026

姓名 吴🄵 学号 2022K8009929033

姓名 卢柯圳 学号 2022K8009929023

实验项目编号 prj1

一、 实验任务说明

本实验需要完成的任务包括：

• 熟悉 ANTLR 的安装和使用，了解 ANTLR 工具生成词法-语法源码的能力；

• 掌握 ANTLR 生成 lexer 和 parser 的流程，完成词法和语法分析；

• 按照 CACT 的文法规范编写 ANTLR 文法文件，通过 ANTLR 工具生成 CACT 源码的词法-语法分析器和访问接口使用 visitor 模式对.cact 文件进行词法和文法分析；

• 修改 ANTLR 中默认的文法错误处理机制，对于符合词法和文法规范的，cact 文件返回值为 0，否则返回非零值

二、 实验过程

• 设计思路

1. **ANTLR 文法文件**

对于 ANTLR 文法文件，需要参照和补全实验讲义给出的 CACT 语言规范，并结合编写 ANTLR 代码的语法规则来定义词法表达式和文法表达式。

2. **main 文件**

main 函数设计思想

输入处理：通过命令行参数读取源文件，将其打包为 ANTLRInputStream 输入流，作为词法和语法分析的输入源。

词法/语法分析：生成 CACTLexer 和 CACTParser 实例，将输入流转换为词法符号流（CommonTokenStream），最终构建语法分析器。

错误处理：移除默认错误监听器，替换为自定义的 BailErrorListener，确保语法错误时立即终止程序并输出错误信息。

语法树构建：调用 parser.wholeProgram() 生成完整的语法树（根节点为 WholeProgramContext），验证语法正确性后输出提示。

遍历模式：通过自定义的 Analysis 类（继承自 CACTBaseVisitor），显式调用 visit 方法遍历语法树，进行进一步分析。

Visitor 遍历模式设计思想

显式调用：与 Listener 模式不同，Visitor 模式需手动触发节点访问。代码中通过 visitor.visit(tree) 显式启动遍历，需在子类中重写 visitXXX 方法实现具体逻辑。

生成依赖：使用 Visitor 模式时，需在 ANTLR 生成语法工具链时指定-visitor 参数，以生成 CACTBaseVisitor 基类。代码中已包含该头文件，表明生成流程已正确配置。

递归控制：Visitor 模式允许自定义遍历逻辑（如跳过某些节点），而 Listener 模式由 ParseTreeWalker 自动完成全部节点的遍历。当前代码未覆盖默认行为，需在 Analysis 类中实现具体访问逻辑。

与 Listener 模式的对比

触发方式：

Listener 模式通过 ParseTreeWalker::walk 自动触发回调（如 enterCompUnit/exitCompUnit）。

Visitor 模式需显式调用 visit 方法，且需在子类中实现节点访问逻辑。

灵活性：

Listener 适合全局遍历，逻辑分散在各个回调函数中。

Visitor 更适合集中控制遍历过程（如条件跳转），需手动管理子节点访问顺序。

代码体现：本次代码选择 Visitor 模式，通过 visitor.visit(tree) 触发遍历，符合显式调用的特征。

三、 实验思考

• 设计编译器的目录结构？

编译器项目的目录结构设计通常遵循两种主要组织范式：功能划分和层次划分。功能划分模式将编译过程的各个阶段（如词法分析、语法分析、语义处理等）作为独立模块进行目录组织，这种设计显著提升了系统的模块化程度；而层次划分模式则通过隔离前端与后端架构，有效降低了跨层级间的耦合度。

在本实验项目中，我们采用了一种以功能导向为主的混合式目录结构设计方案，具体组织如下：

核心编译器逻辑实现位于/src 目录

ANTLR 文法定义文件及相关生成器代码置于/grammar 目录

第三方依赖项统一存放在/deps 目录

测试用例集合则组织在/samples 目录中

项目构建采用 CMake 工具实现自动化构建管理，并通过配套的测试脚本体系来提高代码验证的效率。这种结构设计既保持了功能模块的独立性，又兼顾了项目整体的可维护性。

• 如何把表达式优先级体现在文法设计中？

1. 层级嵌套结构（从高到低）：

括号/一元运算 → 乘除模 → 加减 → 关系运算 → 相等性 → 逻辑与 → 逻辑或

规则层级：primaryExpression → unaryExpression → multiplicationExpression → additionExpression → relationalExpression → equalityExpression → logicalAndExpression → logicalOrExpression

2. 特殊优先级处理

括号强制优先级：(expression) 作为基础表达式

一元运算符（正负/逻辑非）优先级最高：-3*5 解析为 (-3)*5

关系运算（> <）优先级高于相等性判断（== !=）

3. 所有二元运算均为左结合（通过 (op term)* 递归实现）

4. 常量表达式单独定义 constantExpression, 数组维度强制要求整型常量 arrayDimensions : (LEFT_BRACKET integerConstant RIGHT_BRACKET)+

这种设计使得语法分析器在解析时，自然遵循 C 语言的标准运算符优先级规则，无需显式指定优先级表。开发者在编写代码时，编译器能自动按照：！> * / % > +- > > >= < <= > == != > > || 的顺序正确解析表达式。

• 如何设计数值常量的词法规则？

1. 调整整数规则顺序和定义

十六进制优先于八进制，确保 0x 被正确识别。十进制包含单独的 0，八进制需至少一个数字。修正后的规则：

HEXADECIMAL_NUMBER : ('0x' | '0X') [0-9a-fA-F]+;

DECIMAL_NUMBER : '0' | [1-9][0-9]*; // 包含单独的 0

OCTAL_NUMBER : '0' [0-7]+; // 至少一个八进制数字

2. 合并浮点数和指数形式

统一处理浮点数和科学计数法，避免规则冲突。修正后的规则：

FLOATING_POINT_CONSTANT

: ([0-9]+ '.' [0-9]* // 如 123.45

| '.' [0-9]+ // 如.45

| [0-9]+ '.' // 如 123.

)

([eE] [+-]? [0-9]+)? // 可选指数部分

[fF]? // 可选后缀

| [0-9]+ [eE] [+-]? [0-9]+ [fF]? // 整数后接指数，如 123e5

;

3. 调整词法规则顺序确保更具体的规则（如十六进制）优先于通用规则：

// 整数部分

HEXADECIMAL_NUMBER : ('0x' | '0X') [0-9a-fA-F]+;

DECIMAL_NUMBER : '0' | [1-9][0-9]*;

OCTAL_NUMBER : '0' [0-7]+;

// 浮点部分

FLOATING_POINT_CONSTANT : ...; // 如上

4. 处理字符常量现有规则已覆盖常规字符和转义序列，无需修改：

CHARACTER_CONSTANT : '' (ESCAPE_CHARACTER | [']) '';

最终优化点：

词法顺序：十六进制 > 浮点数 > 十进制 > 八进制，避免冲突。单独 0 归为十进制，八进制需至少一个数字（如 0123）。指数形式整合到浮点数规则，确保 123e45 或 123.45e6 被正确识别。

- **如何替换 ANTLR 的默认异常处理方法？**

自定义错误监听器：

创建了继承自 BaseErrorListener 的 BailErrorListener 类，重写 syntaxError 方法。该方法会：

输出错误位置和描述；

直接调用 exit(EXIT_FAILURE) 终止程序。

移除默认监听器：

lexer.removeErrorListeners(); // 移除词法分析器的默认监听器

parser.removeErrorListeners(); // 移除语法分析器的默认监听器

绑定自定义监听器：

lexer.addErrorListener(new BailErrorListener()); // 词法分析绑定

parser.addErrorListener(new BailErrorListener()); // 语法分析绑定

效果：

遇到词法/语法错误时，不再尝试恢复解析；

立即输出错误信息并终止程序；

最终控制台会输出 Syntax Right! 仅当输入完全符合语法规则时。

与默认行为对比：

默认行为：打印错误后继续解析（错误恢复）；

本实现行为：严格模式，首个错误直接终止，适合编译器开发的刚性要求。

四、 成员总结

吴修齐：

在本次实验中，我基于 ANTLR4 定义了 CACT 语言的语法规则，完成了前端语法分析部分的构建。在文法设计过程中，我严格对照 CACT 语言规范，重点完成了变量与常量声明、表达式、函数定义与语句块等模块的设计。对 CACT 语言规范和 ANTLR 工具有了更深入的认识和了解。

吴🄵：

通过本次实验，我主要了解了 CACT 语言的文法规则，学习了巴科斯范式规范。在研读 demo 源码框架的过程中，我学习了 antlr4 工具的使用以及 visitor、listener 两种遍历模式的原理、区别和构建方法，同时也从对 sample 的测试中进一步体会了编译的整体流程。

卢柯圳：在本次实验中，我对前段语法分析部分的代码进行了检查，确保 CACT 语言的语法规则成功实现，在这一过程中，我初步认识了 CACT 的语言规范，同时一定程度上熟悉了 ANTLR4 工具的使用。