编译原理研讨课第一次实验报告

小组成员：武治行 周熙博 李浩宇

一、任务说明

1.熟悉 ANTLR 的安装和使用，了解 ANTLR 工具的作用和工作原理，搭建 ANTLR 环境并运行 demo

2.根据 CACT 文法规范编写 ANTLR 文法文件，完成词法和语法分析功能，完善词法文法错误检查，实现一个简单的编译器前端，并通过测试样例

二、实验设计

设计思路:

1.设计编译器的目录结构：

> /cact/src/main.cpp: 编译器的主框架

2.表达式优先级在文法设计中的体现

不同表达式的优先级可以通过文法的设计来实现，比方说

A -> B (+ B)*

B -> C (* C)*

就实现了乘法运算的优先级高于加法运算。

利用类似的方式，我们把各种表达式的优先级做出区分，从高到低依次为：

一元运算(包括+、-、！)表达式

乘法、除法、取模表达式

加法、减法表达式

大于、小于、大于等于、小于等于表达式

等于、不等于表达式

逻辑与表达式

逻辑或表达式

3.设计数值常量的词法规则

数值常量分为两个类型：整型和单精度浮点型

其中，整型常量分为三种表示方式：

十进制：一个非零的数字，连接 0 个或者更多[0-9]个数字

八进制：以数字 0 开始，连接 0 个或者更多个[0-7]的数字

十六进制：以 0x 或者 0X 开始，连接 1 个或者更多个[0-9]的数字或者[a-f]或者[A-F]

浮点常量分为两个表示方式：

小数 (e 或者 E (正负符号)? 数字序列)? (f 或者 F)?

数字序列 (e 或者 E (正负符号)? 数字序列) (f 或者 F)?

其中数字序列由 1 个或者更多个[0-9]的数字组成

小数包括三种形式：

'.' 数字序列

数字序列 '.' 数字序列

数字序列 '.'

4.替换 ANTLR 的默认异常处理方法

修改 src 文件夹下的 main.cpp 修改其检查异常的方式

三、实验实现

1.搭建 ANTLR 环境，运行 demo

使用ANTLR工具生成 visitor 的 C++代码

```bash
java -jar ../deps/antlr-4.13.1-complete.jar -Dlanguage=Cpp \ -no-listener - visitor ./Hello.g4
```

编译 cact 项目文件夹,在 build目录下测试

```bash
mkdir -p build
cd build
cmake ..
make
cp ./build/compiler ./
./compiler
```

2.编写语法描述文件

> /cact/grammar/CACT.g4 解析：

（1）各类表达式的文法设计如下，这样的文法设计决定了各类运算的优先级顺序

在 ANTLR 文法中，运算符优先级通过规则的嵌套层次隐式定义，外层规则对应低优先级运算符，内层规则对应高优先级运算符。

a.规则分层结构:

表达式解析规则按优先级从高到低分层定义，外层规则引用内层规则，形成优先级链：

```
/*  优先级由低到高：or → and → eq → rel → add → mul → unary */
mulExpr
    : unary ( (MUL | DIV | MOD) unary )*
    ;

addExpr
    : mulExpr ( (ADD | SUB) mulExpr )*
    ;

relExpr
    : addExpr ( (LT | LE | GT | GE) addExpr )*
    ;

eqExpr
    : relExpr ( (EQ | NEQ) relExpr )*
    ;

andExpr
    : eqExpr (AND eqExpr)*
```

```
    ;

orExpr
    : andExpr (OR andExpr)*
    ;
```

b.关键代码段解析:

•乘除模优先级高于加减，addExpr 由 mulExpr 构成，意味着所有乘除模运算在加减前完成

```
addExpr: mulExpr (ADD | SUB mulExpr)* ;
mulExpr: unary (MUL | DIV | MOD unary)* ;
```

•一元运算符优先级最高，unary 规则直接操作原子项（primary），确保一元运算先于其他运算组合。

```
unary   : ('+'|'-'|'!') unary | primary ;
```

•括号强制优先级，括号内的 expr 作为 primary 处理，直接进入最高优先级层级。

```
primary: '(' expr ')' ;   // 括号内表达式作为独立子树
```

（2）数值常量的词法规则如下，实现了设计思路中所述的整型和单精度浮点型两种数值常量

a.整型常量 (IntConst)

```
IntConst
      :
'0'                                  // 纯零（十进制）
      | [
1-9] [0-9]*                        // 十进制（非零开头）
      |
'0' [0-7]+                          // 八进制（0 开头，后接 0-7）
      | (
'0x'|'0X') [0-9a-fA-F]+       // 十六进制（0x 或 0X 开头）
      ;
```

b.单精度浮点型常量 (FloatConst)

```
FloatConst
      : ([
0-9]+)? '.' [0-9]* ([eE] [+\-]? [0-9]+)? [fF]   // 强制后缀 f/F
      ;
```

c.双精度浮点型常量 (DoubleConst)

```
DoubleConst
      : ([0-9]+)? '.' [0-9]* ([eE] [+\-]? [0-9]+)?   // 无后缀或指数形式
      ;
```

d.关键设计实现

•词法规则优先级：

在 ANTLR 中，词法规则按声明顺序优先匹配。由于 FloatConst 和 DoubleConst 的规则在 IntConst 之后定义，避免了数字部分被误识别为整数。

•类型区分：

单精度浮点:必须显式添加 f/F 后缀。

双精度浮点:允许无后缀或指数形式。

•语法引用：

在语法规则的 primary 中，通过独立的词法符号区分常量类型：

```
primary
    : '(' expr ')'
    | lVal
    | IntConst
    | DoubleConst
    | FloatConst
    | TRUE
    | FALSE
    ;
```

3.使用 ANTLR 工具，根据语法描述文件，生成词法分析器、语法分析器等模块

```
cd cact/grammar
java -jar ../deps/antlr-4.13.1-complete.jar -Dlanguage=Cpp Hello.g4 -visitor -no-listener
```

4.替换 ANTLR 的默认异常处理方法

关键代码解析：> /cact/src/SemanticAnalyzer.cpp

（1）重写 visitErrorNode

在自定义 Visitor 中覆盖错误节点访问方法。拦截语法错误节点，替换 ANTLR 默认的终端错误输出，实现自定义错误处理逻辑。

```
std::any visitErrorNode(tree::ErrorNode *node) override {
      std::cout << "visit error node!" << std::endl;   // 自定义错误处理
      return nullptr;
}
```

（2）输入流加载方式

从 ANTLRInputStream input(stream) 改为 input.load(stream)。适配 ANTLR 4.13+版本的接口变化，确保文件流正确加载。

```
ANTLRInputStream input;
input.load(stream);
```

（3）解析入口调用

从 visit(parser.r()) 改为 visit(parser.compUnit())。

通过访问整个编译单元（语法树根节点），保证所有潜在错误节点都能被遍历到，触发自定义错误处理。

5.运行 cmake project，编译源代码，生成可执行的 compiler 文件

```
mkdir -p build
cd build
cmake ..
make -j
```

6.在测试用例上运行 compiler，并根据测试结果修改 bug

```
./compiler ../test/samples_lex_and_syntax/00_true_main.cact
```

四、成员分工总结：

李浩宇：本次实验中，我负责实验报告的撰写。在撰写实验报告的过程中，我对于 ANTLR 工具的功能和使用方式有了更全面的了解，对实验设计思路进行了总结，学习了前端词法分析和语法分析的整个流程，理解了各个项目文件在整个系统中发挥的作用，同时学习了 EBNF 规范，掌握了语法描述文件的编写方式。

周熙博：本次实验中，我参与了 g4 和源代码的编写。在编写代码的过程中对于 ANTLR 工具有了更深的了解，特别是通过单步跟踪阅读 ANTLR 生成的 visitor，使我对于抽象语法树有了更深的认识。

武治行：本次实验中，我参与了 g4 代码的编写与检查，并且参与了由 g4 生成的代码的 debug。在 debug 跟踪调试的过程中，我更加清晰的认识了 antlr4 的运行环境，自动生成语法树的访问以及根据文法进行语法分析最基本的步骤。通过对 g4 文件的学习，我掌握了最基本的文法编写规范，也对正则的表达式规范有了一个初步的了解