编译原理研讨课实验 PR001 实验报告

CACT 39 组
康家锐 吕恒磊 王东宇

一、任务说明

进行 Antlr 的安装。编写文法文件并应用 Antlr 生成 lexer 和 parser。用测试样例验证结果，对符合词法和语法规范的.cact 文件返回 0 值，不符合的返回非 0 值。

二、成员组成

吕恒磊、王东宇、康家锐

三、实验设计

1.设计思路

主要根据给出的 CACT 语言的文法进行 g4 文件的编写，对每一条替换文法进行改写。对于未给出的几个常量类型标识符进行自己的设计，主要是对该种常量类型数据可能的格式进行分类，设置出对应的子类型，并使常量类型可以被替换为对应的子类型，再使子类型可以替换为具体的终结符。

对于返回值的输出，首先查看主函数中关于返回信息的函数，试图找到返回的信息。在同学的帮助之下，最终找到了一个返回错误个数的变量，利用其作为输出就可以得到正确的输出了。

2.实验实现

首先安装 antlr，用$ antlr4 验证：

```
compiler39@PowerEdge-M640-Blade-8:~/compiler/grammar$ antlr4
ANTLR Parser Generator  Version 4.8
 -o ___              specify output directory where all output is generated
 -lib ___            specify location of grammars, tokens files
 -atn                generate rule augmented transition network diagrams
 -encoding ___       specify grammar file encoding; e.g., euc-jp
 -message-format ___ specify output style for messages in antlr, gnu, vs2005
 -long-messages      show exception details when available for errors and warnings
 -listener           generate parse tree listener (default)
 -no-listener        don't generate parse tree listener
 -visitor            generate parse tree visitor
 -no-visitor         don't generate parse tree visitor (default)
 -package ___        specify a package/namespace for the generated code
 -depend             generate file dependencies
 -D<option>=value    set/override a grammar-level option
 -Werror             treat warnings as errors
 -XdbgST             launch StringTemplate visualizer on generated code
 -XdbgSTWait         wait for STViz to close before continuing
 -Xforce-atn         use the ATN simulator for all predictions
 -Xlog               dump lots of logging info to antlr-timestamp.log
 -Xexact-output-dir  all output goes into -o dir regardless of paths/package
```

运行测试样例以验证：

正确样例：

```
compiler39@PowerEdge-M640-Blade-8:~/compiler/build$ ./compiler ../samples_lex_and_sy
ntax/00_true_main.cact
0 lexer error(s)
0 parser error(s)
```

错误样例：

```
compiler39@PowerEdge-M640-Blade-8:~/compiler/build$ ./compiler ../samples_lex_and_sy
ntax/01_false_hex_num.cact
line 3:13 extraneous input 'x' expecting {',', ';'}
0 lexer error(s)
1 parser error(s)
```

3.其它

在修改文法书写时我们还自己手动编写了一些样例，方便我们查看现有程序能否识别一些特定类型和查找具体出错步骤。

四、总结

1.实验结果总结

我们在编写、修改 g4 代码的过程中，加深了对编译器语法分析逻辑的理解，加强了文法设计能力；通过阅读 antlr 生成的代码，找到了错误信息和变量定义打印的机制，加深了对代码工作原理的理解，以及 C++代码的阅读能力。最终成功的完成了本次实验的目标。

在整个实验过程中我们也遇到了诸多问题：

在进行 g4 的编写、验证时发现 float 与 double 类型变量无法识别

更改后可以识别 float 类型，double 类型依然无法正常识别，最终直接用其子类型 SmallConst（小数）和 ScienceConst（科学计数法）替代该类型以实现。

另外，在面对如 int a,b 格式的声明时，如果在 b 处出错就会导致 walk 下会运行的某个函数内存越界出错。实际上我们反复试验才锁定在了这个地方，并且对这条语法的修改尝试都会导致在编译一步就出错。最终我们采取了注释掉整个 walk，利用之前 line 相关的信息进行输出回避了这一问题。

这次实验中我们还首次尝试了同时登陆账户进行工作，不过遇到了一些问题。主要是 antlr 每次使用时都会失效，需要重新按安装步骤设置路径，即使没有人对其进行更改仍然会发生这一问题。后来才意识到需要将配置命令输入.bash.rc 文件中才能每次启动新的终端都能完成配置。这部分浪费了一些时间。

在输出方面我们也遇到了一些问题，主要是一开始我们计划设置一全局变量作为是否出错的标致以及输出，并在打印报错信息的函数处直接对其赋值。但是，由于具体的报错函数不在 compiler 下，我们无法应用我们的修改，只能去想办法去一层层的找调用关系或者主函数中一层层的找最上层的判断正确与否的函数。最终，在 walk 下找到了一个之前因为为空而被我们忽视的函数在每次出错时都会被调用，在其中进行全局变量的修改就实现了对结果的输出。

2.分成员总结

康家锐：通过这次实验，我对文法分析是如何实现的有了更深入的认识，同时也初步的了解到了如何应用 antlr 这一工具。不过利用 antlr 生成的 parser 代码十分繁杂，对我们后续尝试修改造成了很大的阻碍，这一点还需要在以后逐渐适应。

王东宇：熟悉了 antlr 的安装和使用，但由于对面向对象编程不够熟悉，在阅读该工具生成的代码时遇到了一定的阻力，并且在使用生成的语法分析器时，遇到过莫名其妙的 bug。但这些困难都被我们一一克服，并在最终完成了实验。

吕恒磊：实验代码量不算大，但工具链很繁琐，遇到了许多暂时解释不了的问题，比如 g4 文件某个写法不行但换个写法就可以，比如有时遍历语法树时会莫名段错误等等，希望答疑时能解决。总的来说了解了语法与词法分析的具体实现步骤，还是很有收获的。