编译原理研讨课实验PR001实验报告

任务说明

本实验任务主要包含以下内容：

- 掌握如何安装Antlr；
- 根据给出的CACT文法规范，编写.g4文件，并通过Antlr生成lexer和parser，对输入的CACT语言源代码(.cact文件)进行词法和语法分析；
- 修改Antlr中默认的文法错误处理机制，使得在遇到词法和语法错误时，能进行相应的处理。

成员组成

姓名	学号
杜政坤	2019K8009909005
王畅路	2019K8009907018
王意晨	2019K8009929016

实验设计

设计思路

1. Parser 部分直接将提供的CACT文法规范转化为 .g4 文件的要求即可，Lexer 部分根据CACT规范自行设计并添加到 .g4 文件中，具体设计在实验实现中；
2. 在main函数中对于词法分析和语法分析出现错误的情况返回非零值，实现错误处理。

实验实现

1.参考本实验提供的CACT语言规范来编写.g4文件

Parser部分主要按照CACT语言规范来给出产生式，部分产生式按照具体情况进行了调整

例如开始符号的产生式：

```
CompUnit → [ CompUnit ] ( Decl | FuncDef )
```

等价于重复一次或多次Decl或FuncDef。考虑到结束符EOF的匹配，在.g4文件中对该产生式的描述如下：

```
compUnit
    : (decl | funcDef)+ EOF
    ;
```

对于Lexer部分需要补充：

- 标识符 Ident 标识符可以由大小写字母、数字以及下划线组成，但必须以字母或者下划线开头。`IdentNondigit` 代表了字母下划线开头。

  ```
  Ident
      : IdentNondigit [a-zA-Z_0-9]*
      ;

  fragment
  IdentNondigit
      : [a-zA-Z_]
      ;
  ```

- 整型常量 IntConst 整形常量包括了十进制 八进制 十六进制数

  ```
  IntConst
      : DecimalConst
      | OctalConst
      | HexadecimalConst
      ;
  ```

  十进制数可以为 0，或者以非零数开头的数字串

  ```
  fragment
  DecimalConst
      : '0'
      | NonzeroDigit Digit*
      ;
  ```

  八进制数需要以数字 0 开头，并且只能跟 [0 - 7]的数

  ```
  fragment
  OctalConst
      : '0' OctalDigit+
      ;
  ```

  十六进制数需要以0x或者0X开头，之后跟 [0 - 9 a - f A - F] 的数

  ```
  fragment
  HexadecimalConst
      : HexadecimalPrefix HexadecimalDigit+
      ;
  fragment
  HexadecimalPrefix
  ```

```
    : '0x'
    | '0X'
    ;
fragment
HexadecimalDigit
    : [0-9a-fA-F]
    ;
```

- 布尔型常量 BoolConst 包含`true`和`false`

```
BoolConst : 'true' | 'false';
```

- 单精度浮点常量 FloatConst 和 双精度浮点常量 DoubleConst

  单精度浮点常量就是在双精度浮点常量后面加上F或f，因此只需要在double的规则后加上FloatSuffix

  浮点数的构成有三种情况，分别是浮点数串，浮点数串连接科学计数串，以及十进制数连接科学计数串

```
DoubleConst
    : FloatSequence
    | FloatSequence ScientificNotation
    | Digit* ScientificNotation
    ;

FloatConst
    : FloatSequence FloatSuffix
    | FloatSequence ScientificNotation FloatSuffix
    | Digit* ScientificNotation FloatSuffix
    ;
fragment
FloatSuffix
    : 'F'
    | 'f'
    ;
```

  浮点数串允许小数点前没有任何串，但是此时小数点后必须有数字串。或者小数点前有数字串，小数点后可以没有任何数字串。

```
fragment
FloatSequence
    : Digit* '.' Digit+
    | Digit+ '.' Digit*
    ;
```

  科学计数串由e/E打头，可选的符号位，以及不能为空的数字串组成

```
fragment
ScientificNotation
    : ('e' | 'E') ('+' | '-')? Digit+
    ;
```

2.错误处理

查看`antlr`的API可以知道，`CACTLexer`的父类`antlr4::Lexer`和`CACTParser`的父类`antlr4::Parser`均提供了`getNumberOfSyntaxErrors`方法，通过他们可以获取词法和语法分析过程中出现的错误数量，再根据错误数量进行正确的处理。那么很容易写出处理错误的代码：

```
   if(lexer.getNumberOfSyntaxErrors() >0 || parser.getNumberOfSyntaxErrors()
> 0)
        return 1;
    else
        return 0;
```

在主程序结束前执行即可。

总结

实验结果总结

通过本次实验我们熟悉了antlr的使用；编写文法文件后利用antlr生成了符合CACT语法标准的词法和语法分析器，并实现了词法和语法错误处理功能，通过了samples中的测试样例。熟悉了CACT的文法规则，以及如何其构造paser 和 lexer。

分成员总结

1. 王畅路

   通过本次实验，我了解了antlr是怎么工作的，同时我也有很多不太明白的东西，比如listen模式中的动作是如何完成的，函数是如何自动生成的，通过查阅antlr的API和复习一些C++多态的知识，让我对antlr工具有了更深的认识。

2. 王意晨

   本次实验比较简单，我们只需要根据实验要求仿照给定的例子将CACT语言的文法复现即可。有一点需要说明的是我们并没有考虑浮点数以及整数的符号问题，不过对实验结果的影响不是很大。通过本实验，我对antlr有了初步的认识，比如了解了怎么在.g4文件中编写语法，并对git的使用方法有了更深一步的了解。

3. 杜政坤

   在本次实验的工作中，我们需要按照给定的规范和示例将CACT的语言规范实现到.g4文件中。通过本实验，我对antlr有了初步的认识，主要了解了在.g4文件中编写语法分析和词法分析的方法，在这一过程中，我也查阅了C语言的g4文法，进而对CACT有了更深的了解。同时，我也阅读了一些实验的框架代码，为之后的实验奠定一定的基础。除了实验之外，我们对git协同工作也有了更深刻的体验，多人同时开发没有想象中的容易协调。