测试说明

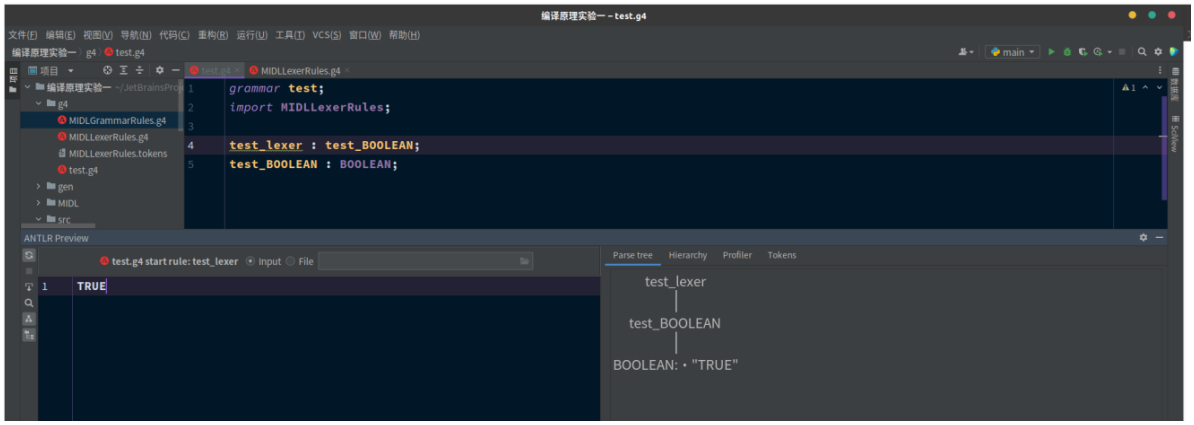
测试G4中词法，文法是否正确定义

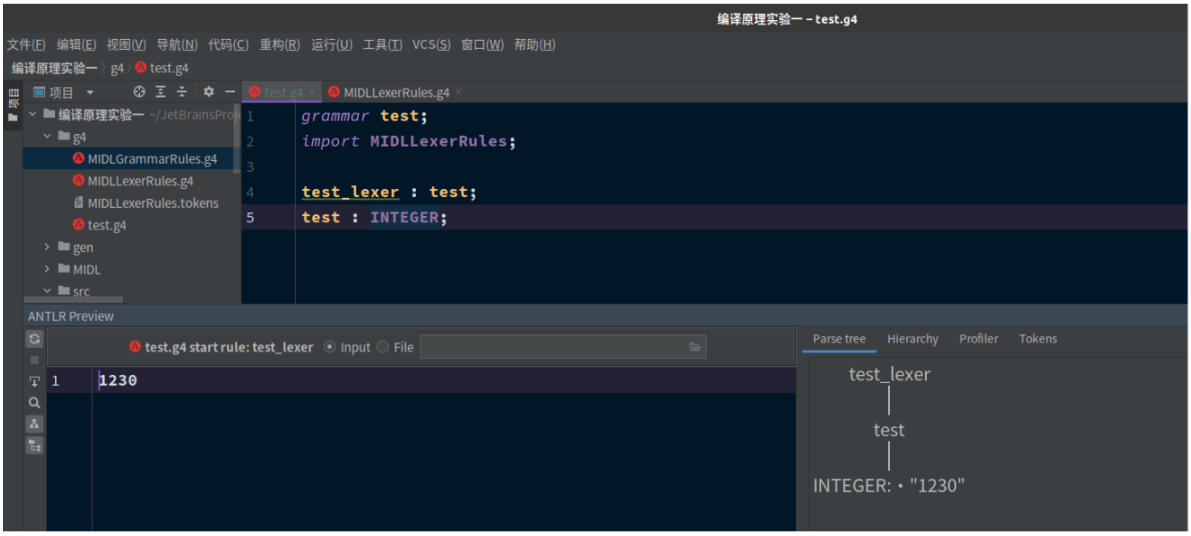
这部分主要是在构建g4文件的同时进行测试的，通过使用Pycharm的ANTLR 4插件来进行测试。

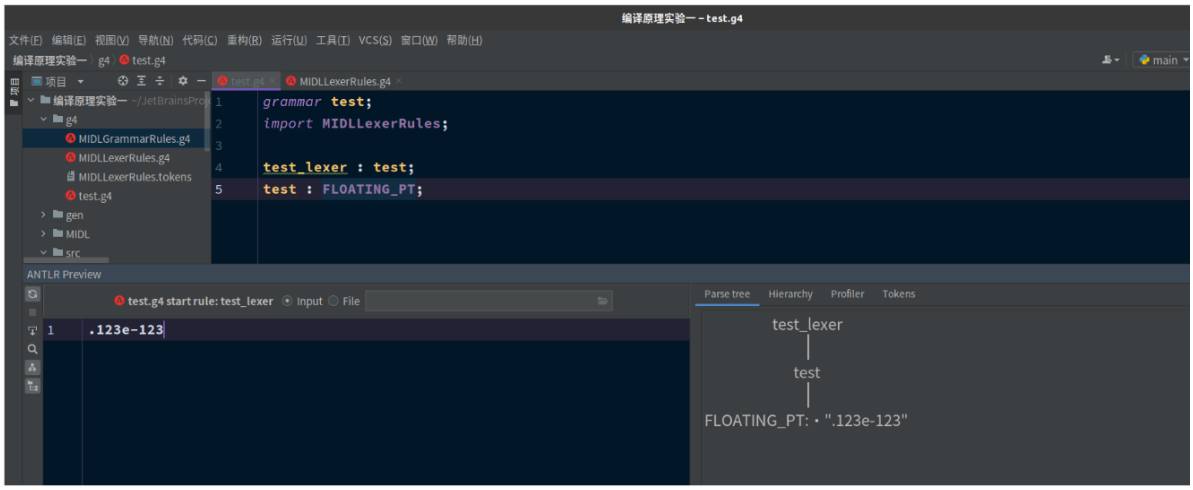
具体的测试方案为：

- 首先写好`MIDLLexerRules.g4`文件，其中只包含词法相关的内容
- 然后新建一个`test.g4`文件，其中写一些简单的语法用于**对词法进行测试**（ANTLR 4插件不支持直接测试词法）
- 之后写好`MIDLGrammarRules.g4`文件，使用插件**对文法进行测试**

词法测试举例：

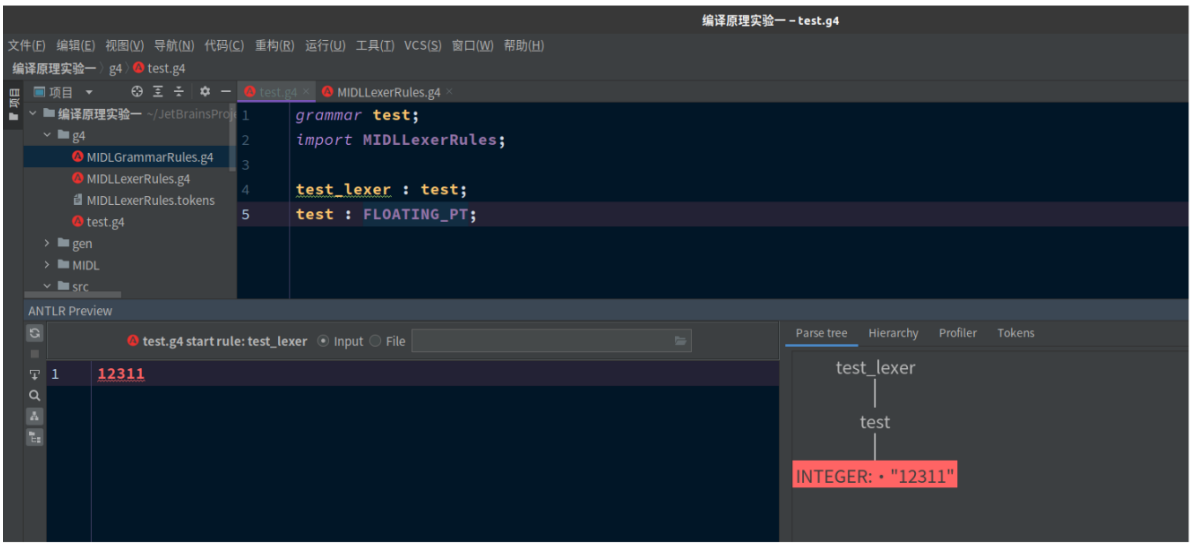




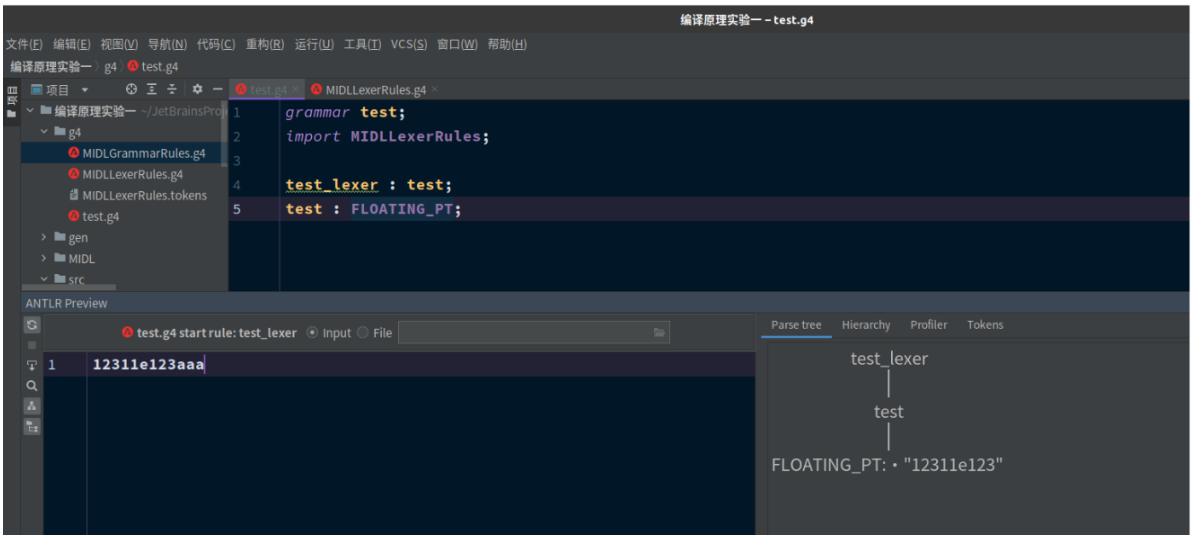


如果词法定义有问题或者输入的有问题的话：

他可能会识别成其他的词法然后报错，比如

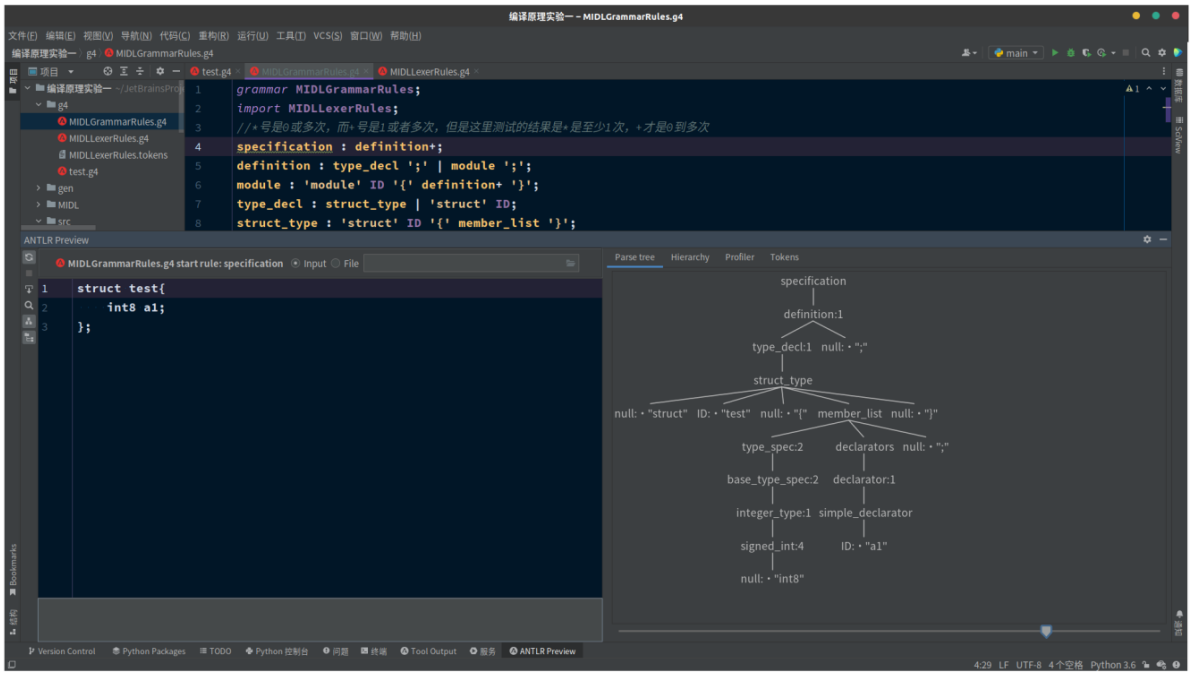


或者对字符串后面的部分不作处理，比如

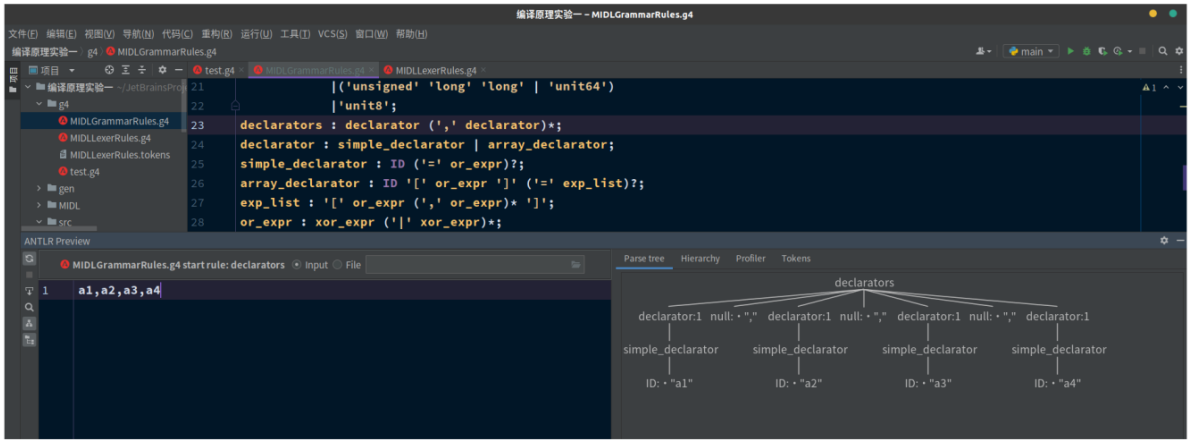


文法测试举例：

测试 `specification : definition+;`



测试 declarators : declarator (',' declarator)*;



测试设计的抽象语法树是否正确构建

测试用例说明

用例编号	用例说明	备注
test_0	测试ID、base_type_spec和struct_type	无
test_1	测试有多个definition和多个declarators的情况	definition只解析为type_decl
test_2	测试有多个definition和多个declarators的情况	type_decl和module都有出现
test_3	测试scoped_name	无
test_4	测试simple_declarator	simple_declarator比较简单的情况
test_5	测试simple_declarator	simple_declarator比较复杂的情况
test_6	测试array_declarator	array_declarator的情况
test_7	测试simple_declarator和array_declarator都存在的情况	无
test_8	测试错误的ID格式	错误类型包括：定义关键词为ID，使用数字为ID，数字开头作为ID，使用字符串为ID，错误的斜杠
test_9	测试错误的simple_declarator与错误的array_declarator	错误类型包括：不加"["或者"]"，不符合or_expr的定义，不符合and_expr的定义等

每一个用例的输出情况：

```
# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:44:41]
$ python3 main.py test/test_0

# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:44:47]
$ python3 main.py test/test_1

# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:44:57]
$ python3 main.py test/test_2

# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:44:58]
$ python3 main.py test/test_3

# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:44:59]
$ python3 main.py test/test_4

# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:44:59]
$ python3 main.py test/test_5

# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:45:00]
$ python3 main.py test/test_6

# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:45:01]
$ python3 main.py test/test_7

# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:45:02]
$ python3 main.py test/test_8
line 2:11 mismatched input 'int8' expecting ID
line 3:10 mismatched input '123' expecting ID
line 4:13 mismatched input '"test"' expecting ID
line 5:10 mismatched input '12' expecting ID
line 6:13 extraneous input '_' expecting ';'

# immortalqx @ Legion in ~/JetBrainsProjects/PycharmProjects/编译原理实验一 [15:45:03]
$ python3 main.py test/test_9
line 2:17 missing '[' at '0.1e+10'
line 2:24 mismatched input ';' expecting {',', ']'}
line 3:19 extraneous input '|' expecting {'+', '-', '~', BOOLEAN, INTEGER, FLOATING_PT, CHAR, STRING}
line 6:35 mismatched input ';' expecting {'+', '-', '~', BOOLEAN, INTEGER, FLOATING_PT, CHAR, STRING}
line 7:22 extraneous input '&' expecting {'+', '-', '~', BOOLEAN, INTEGER, FLOATING_PT, CHAR, STRING}
```

判定方法：

根据测试用例时终端输出的情况，MIDL源文件内容以及输出的格式化的抽象语法树的内容可以判断抽象语法树是否正确构建。

举例：

test用例的源代码如下

```
struct test{
    float a1=1+2-3;
    double a2=1.0-2.0*3.1;
};
```

输出的抽象语法树结构如下所示，这里包含了程序解析语法树的整个过程，其中为了明显的标志终结符，我给每一个终结符都加上了“Terminator xxx”的标签，比如对于终结符ID，标签为：“Terminator ID”。

```
====================抽象语法树的结构====================
Specification
  Definition
    Type_decl
      Struct_type
        Terminator struct
          Terminator ID
          Member_list
            Type_spec
              Base_type_spec
                Floating_pt_type
                  Terminator float
```

```
                    Declarators
                      Declarator
                        Simple_declarator
                          Terminator ID
                            Terminator =
                              Or_expr
                                Xor_expr
                                  And_expr
                                    Shift_expr
                                      Add_expr
                                        Mult_expr
                                          Unary_expr
                                            Literal
                                              Terminator Literal
                                          Terminator +
                                            Mult_expr
                                              Unary_expr
                                                Literal
                                                  Terminator Literal
                                          Terminator -
                                            Mult_expr
                                              Unary_expr
                                                Literal
                                                  Terminator Literal
          Type_spec
            Base_type_spec
              Floating_pt_type
                Terminator double
                  Declarators
                    Declarator
                      Simple_declarator
                        Terminator ID
                          Terminator =
                            Or_expr
                              Xor_expr
                                And_expr
                                  Shift_expr
                                    Add_expr
                                      Mult_expr
                                        Unary_expr
                                          Literal
                                            Terminator Literal
                                        Terminator -
                                          Mult_expr
                                            Unary_expr
                                              Literal
                                                Terminator Literal
                                              Terminator *
                                                Unary_expr
                                                  Literal
                                                    Terminator Literal
```

使用源代码建立的抽象语法树如下所示：(**下面的展示效果不好，可以直接打开.ast文件查看**)

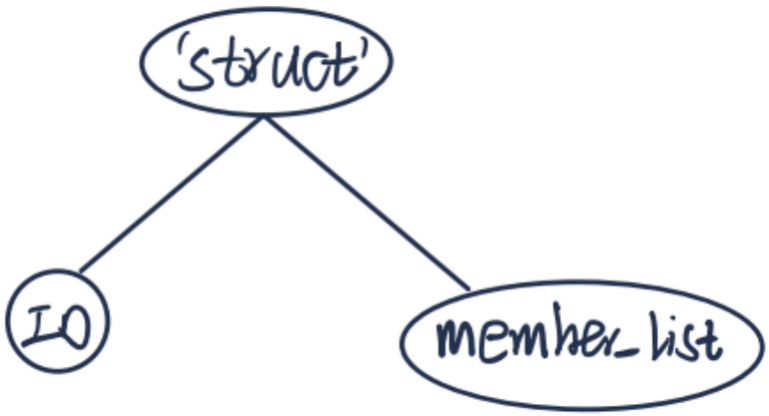
```
====================抽象语法树的内容====================
  struct
   test
```

```
    float
     a1
      =
       1
       +
        2
       -
        3
    double
     a2
      =
       1.0
       -
        2.0
        *
         3.1
```

分析&判断：

对于源代码，编译器首先应当一直解析到`struct_type`，即`struct test{member_list}`这个部分，然后开始有分叉。

这部分的抽象语法树结构如下所示：



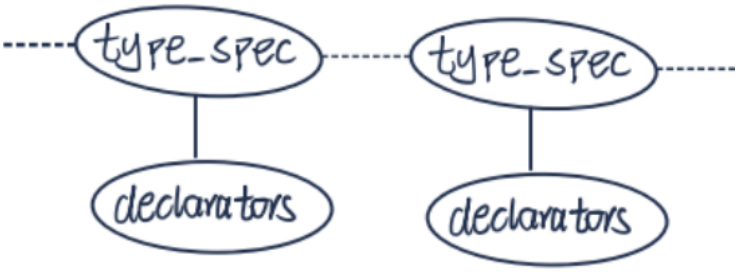
从图中可以看出这与下面这部分的代码是对应的。

```
Specification
  Definition
    Type_decl
      Struct_type
        Terminator struct
          Terminator ID
          Member_list
```

此时ID作为终结符，已经被解析了，所以之后主要是处理Member_list。这部分对应的代码为：

```
    float a1=1+2-3;
    double a2=1.0-2.0*3.1;
```

由于我们只定义了两个变量，所以这部分对应的抽象语法树结构为：



这与上面输出的抽象语法树结构也是对应的：

```
Member_list
  Type_spec
    ..........
  Type_spec
```

之后就是分别解析 `type_spec` 与 `declarators`

以 `float a1=1+2-3` 为例，首先是解析 `type_spec` 到终结符 `float`：

```
            Type_spec
              Base_type_spec
                Floating_pt_type
                  Terminator float
```

而在抽象语法树的定义中，`declarators` 是 `type_spec` 的孩子节点，即源代码中的声明 `a1=1+2-3` 作为类型float的孩子节点。那么，它应该作为上面 `Terminator float` 的孩子节点进行解析

这部分的过程如下所示：

```
                  Terminator float
                    Declarators
                      Declarator
                        Simple_declarator
                          Terminator ID
                            Terminator =
                              Or_expr
                                Xor_expr
                                  And_expr
                                    Shift_expr
                                      Add_expr
                                        Mult_expr
                                          Unary_expr
                                            Literal
                                              Terminator Literal
                                          Terminator +
                                            Mult_expr
                                              Unary_expr
                                                Literal
                                                  Terminator Literal
                                          Terminator -
                                            Mult_expr
                                              Unary_expr
                                                Literal
                                                  Terminator Literal
```

这部分对应到源代码中为：

```
float
 a1
  =
   1
   +
    2
   -
    3
```