# MLIR 编译框架的使用与探索

## 第一部分：词法分析

上海交通大学计算机系

## 1　内容简介

在第一部分，我们要构建一个词法分析器来识别 Pony 语言中的各种词法单元 (Token)，包括关键字（如 var、def 和 return）、特殊符号、数字以及变量名/函数名等。我们要通过相关函数来获取 Token，判断其合法性，并针对非法格式输出错误信息。

## 2　功能实现

注意事项：

- 在 Lexer.h 搜索”TODO”，可以看到需要实现的相关函数以及具体要求；
- 在处理非法情况时，要求编译器在终端输出尽可能详细的错误信息；
- 在实现具体功能之前，须阅读 Lexer.h 中已有代码。

### 2.1　词法分析功能实现

文件地址：/pony_compiler/pony/include/pony/Lexer.h
要求实现以下功能：
1). 实现成员函数 getNextChar()

- 注意几种 corner case 的特殊处理，比如读到某行结尾，读到文档结尾等情况；
- 注意行列等位置信息的更新。

2). 补充成员函数 getTok()

- 能够识别 “return”、“def” 和 “var” 三个关键字；
- 能够识别标识符：
  - 标识符以字母开头；
  - 标识符由字母、数字或下划线组成；
  - 按照使用习惯，标识符中不能出现连续的下划线；
  - 按照使用习惯，要求标识符中有数字时，数字须位于标识符末尾。
    例如：有效的标识符可以是 abc3, b_, placeholder 等。
- 改进识别 number 的方法，使编译器可以识别并在终端报告非法 number。
  非法表示包括：9.9.9，9..9，.999，..9，9.. 等。

### 2.2 词法分析验证程序实现

文件地址：/pony_compiler/pony/ponyc.cpp + /pony_compiler/pony/include/pony/Lexer.h
要求实现以下功能：
1). 补充 ponyc.cpp 文件中“词法分析器正确性”验证程序 int dumpToken()
2). 扩展 Lexer.h 文件中 getTok() 函数，在识别每种 Token 的同时，将其存放在某种数据结构中，以便最终在终端输出:

- 输入文件为 Pony 语言定义的函数等;
- 如果词法分析器没有识别出错误，则按顺序输出识别到的 Token;
- 如果词法分析器识别出错误，则在终端输出详细的错误信息;
- 输出信息的具体形式可参考接下来第 3 节中的测试示例。

## 3 实验验证

在对词法分析器构建完毕后，可以通过运行测试用例 test_1 至 test_7 来检查词法分析器的正确性。以 test_1 为例：build pony 并执行下面测试用例，以验证词法分析器能够识别出各种合法的词法单元：

```
$ cmake --build . --target pony
$ ../build/bin/pony ../test/test_1.pony -emit=token
```

如果执行结果如下图所示，表示词法分析器分析正确。

```
root@4dae1f2a64aa:/home/workspace/pony_compiler/build# ./bin/pony ../test/test_1.pony -emit=token
def main ( ) { var a [ 2 ] [ 3 ] = [ 1 , 2 , 3 , 4 , 5 , 6 ] ; } EOF
```

**图 1.** 正确测试用例的执行结果

以 test_2 为例：build pony 并执行下面测试用例，以验证词法分析器能够识别出各种非法的词法单元并输出错误信息：

```
root@4dae1f2a64aa:/home/workspace/pony_compiler/build# ./bin/pony ../test/test_2.pony -emit=token
Identifier <a__bcde> contain continuous _
def multiply_transpose ( a , b ) { return transpose ( a ) * transpose ( b ) ; } def main ( ) { var ERROR_IDENTIFIER = [ [ 1 , 2 , 3 ] ,
 [ 4 , 5 , 6 ] ] ; var b < 2 , 3 > = [ 1 , 2 , 3 , 4 , 5 , 6 ] ; var c = multiply_transpose ( a , b ) ; print ( c ) ; } EOF
```

**图 2.** 错误测试用例的执行结果