

Project 1 - Lexer

2025 年 9 月 15 日

1 项目要求

SPL 语言是一种类 C 语言。它定义了一组词法符号（Lexical Tokens）。你的任务是在 `Splc.g4` 文件中声明这些词法符号，并为每个词法符号编写相应的正则表达式规则。

1.1 Keywords

INT	：匹配关键字 <code>int</code> ；
CHAR	：匹配关键字 <code>char</code> ；
STRUCT	：匹配关键字 <code>struct</code> ；
RETURN	：匹配关键字 <code>return</code> ；
IF	：匹配关键字 <code>if</code> ；
ELSE	：匹配关键字 <code>else</code> ；
WHILE	：匹配关键字 <code>while</code> ；

1.2 Operators

ASSIGN : 匹配赋值运算符 “=” ;
PLUS : 匹配加号 “+” ;
MINUS : 匹配减号 “-” ;
STAR : 匹配乘号 “*” ;
DIV : 匹配除号 “/” ;
MOD : 匹配取模符号 “%” ;
LT : 匹配小于号 “<” ;
LE : 匹配小于等于号 “<=” ;
GT : 匹配大于号 “>” ;
GE : 匹配大于等于号 “>=” ;
EQ : 匹配等于号 “==” ;
NEQ : 匹配不等号 “!=” ;
AND : 匹配逻辑与 “&&” ;
OR : 匹配逻辑或 “||” ;
NOT : 匹配逻辑非 “!” ;
INC : 匹配自增符号 “++” ;
DEC : 匹配自减符号 “--” ;
DOT : 匹配成员访问符 “.” ;
ARROW : 匹配指针访问符 “->” ;
AMP : 匹配取地址符 “&” ;

1.3 Separators

SEMI : 匹配分号 “;” ;
COMMA : 匹配逗号 “,” ;
LPAREN : 匹配左括号 “(” ;
RPAREN : 匹配右括号 “)” ;
LBRACE : 匹配左花括号 “{” ;
RBRACE : 匹配右花括号 “}” ;
LBRACK : 匹配左方括号 “[” ;
RBRACK : 匹配右方括号 “]” ;

1.4 Identifier and Literals

Identifier : 匹配以字母或下划线开头, 后续可以包含字母、数字或下划线的字符串 (不能是空串);
Number : 匹配由数字组成的整数 (十进制数字) 且开头数字不为 0;
Char : 匹配单引号括起来的单个字符, 例如 'a'; 支持转义字符, 形式为“反斜杠加特定字符”, 如 '\n' (换行)、'\t' (制表)、'\'' (单引号)、'\\' (反斜杠自身) 等;

1.5 Whitespaces and Comments

WS : 匹配空格、制表符、换行等空白符, 并跳过;
LINE_COMMENT : 匹配以 “//” 开头直到行尾的注释, 并跳过;
BLOCK_COMMENT : 匹配 “/*” 和最近的 “*/” 之间的多行注释, 并跳过;

2 初始代码

初始代码位于 <https://github.com/sqlab-sustech/CS323-Compilers-2025F-Projects> 的 `project1-base` 分支。关于如何从初始代码开始, 请参照 Project Zero 和 Project Tutorial。

2.1 框架说明

在初始代码中, 我们提供了 `framework.project1.Grader` 类, 其核心代码如下:

```
CharStream input = CharStreams.fromStream(sourceStream);
Splc lexer = new Splc(input);
CommonTokenStream tokens = new CommonTokenStream(lexer);
tokens.fill();
Vocabulary vocabulary = lexer.getVocabulary();
for (Token token : tokens.getTokens()) {
    this.writer.append(String.format("Token: %s, Raw: %s\n",
        vocabulary.getSymbolicName(token.getType()),
        token.getText())
);
}
```

这一段代码会调用 ANTLR 根据你所撰写的 `Splc.g4` 语法文件所生成的 `Splc`, 并将字符流 `sourceStream` 转换为 `Token Stream`; 随后, 它会遍历所有 `Token` 并打印它们。你需要保证你的程序输出和我们的参考输出完全一致。

在 `Main` 类中, 我们提供了两种测试方法: 传入一个源文件流, 或者直接传入一个字符串。

```
public static void main(String[] args) throws FileNotFoundException {
    FileInputStream fis = new FileInputStream("testcases/project1/case1.splc");
    Grader grader1 = new Grader(fis);
    grader1.run();

    System.out.println("----");
```

```
    Grader grader2 = new Grader("1 + 2");
    grader2.run();
}
```

你可以在 Main 类中随意测试你的 Sp1c。

2.2 评分方式

我们将在服务器上根据你提交的 Sp1c.g4 文件用 ANTLR 生成 Sp1c 类，并运行 framework 中我们提供的 Grader。我们会将你的程序输出与我们的标准输出进行对比。

2.3 评分细则

系统共提供 20 个测试样例，每个样例 5 分，总分 100 分。你需要确保以下两点：

- 每一个 Token 的类型必须正确（其类型名称须与上文中所规定的名称一致）；
- 每一个 Token 对应的 Lexeme 内容必须与源程序中实际内容完全对应。

2.4 示例

2.4.1 示例输入

```
int main(){
    int a = 1 + 2;
    return 0;
}
```

2.4.2 示例输出

```
Token: INT, Raw: int
Token: Identifier, Raw: main
Token: LPAREN, Raw: (
Token: RPAREN, Raw: )
Token: LBRACE, Raw: {
Token: INT, Raw: int
Token: Identifier, Raw: a
Token: ASSIGN, Raw: =
Token: Number, Raw: 1
Token: PLUS, Raw: +
Token: Number, Raw: 2
Token: SEMI, Raw: ;
Token: RETURN, Raw: return
Token: Number, Raw: 0
Token: SEMI, Raw: ;
Token: RBRACE, Raw: }
Token: EOF, Raw: <EOF>
```