# COOL 语言词法分析器开发报告

Compiler Principle Assignment

姓名: 谭润
学号: 20238131027
班级: 大数据二班
项目代码：Github 链接

2025 年 10 月 29 日

## 摘要

本实验基于 Flex 工具实现了 COOL 语言词法分析器的设计与开发过程。首先深入阐释了 Flex 的核心原理，包括通过正则表达式构建 NFA、经子集构造法转化为 DFA 并最小化的自动机生成流程，以及最长匹配、规则优先级的模式匹配机制，还有状态机在字符串、嵌套注释等复杂场景的应用；其次结合代码实现，详解关键字大小写不敏感匹配、标识符分类识别、字符串转义与长度限制、嵌套注释解析及错误检测等功能的实现逻辑；最后通过多维度测试（基础功能、字符串注释、错误处理及集成测试），验证了词法分析器对 COOL 源代码的正确 Token 化能力，以及与编译器后续阶段的协同效果，完整呈现从词法规则定义到可执行分析器的全流程。

## 1 项目概述与环境

### 1.1 项目目标

本实验基于 Flex 词法分析器，为 COOL 语言设计并实现了一款功能完备的词法分析器。该分析器能精准识别 COOL 语言的词法单元（含大小写不敏感的关键字、分类区分的标识符、各类常量及多字符操作符），支持单行与嵌套多行注释的正确解析，同时具备完善的错误处理能力（可检测未闭合字符串、字符串过长、非法字符等问题并标注行号），最终生成符合规范的 Token 序列，确保能与编译器后续的语法分析、语义分析阶段顺畅协作，通过多项测试验证了设计的词法分析器的可用性与正确性。

### 1.2 开发环境

#### 1.2.1 硬件配置（云服务器信息）

- **CPU**: x86_64 架构，2 核
- **内存**: 总容量 2.0Gi
- **硬盘**: 根分区容量总计约 11G（文件系统为 ext4）

#### 1.2.2 软件环境

- **操作系统**: Ubuntu 22.04.1 LTS
- **内核版本**: 5.15.0-60-generic
- **Flex 版本**: 2.6.4-8build2
- **G++ 版本**: g++ (Ubuntu 11.3.0-1ubuntu1 22.04) 11.3.0
- **Make 版本**: GNU Make 4.3
- **SPIM 版本**: SPIM Version 8.0 of January 8, 2010
- **coolc 编译器版本**: 0.1
- **vscode 编辑器版本**: vscode 1.105.1(user setup)

#### 1.2.3 项目目录结构

```
~/code/School/school_homework_code/compiler/shiyan2
|-- a.out
|-- cgen -> /usr/class/bin/cgen
|-- coolc/
|-- cool.flex
```

```
|-- cool-lex.cc
|-- hello.cl
|-- lexer
|-- makefile
|-- parser -> /usr/class/bin/parser
|-- report/
|-- semant -> /usr/class/bin/semant
|-- test_basic.cl
|-- test_error.cl
|-- test_string.cl
```

### 1.2.4 环境配置过程

安装各种依赖包:

```
sudo dpkg --add-architecture i386
sudo apt update
sudo apt install libc6:i386
sudo apt install lib32z1
sudo apt install zlib1g:i386
sudo apt install libncurses5:i386
```

安装各种工具:

```
sudo apt install flex
sudo apt install spim
```

安装 coolc 编译器:

```
git clone https://github.com/aweinert/coolc.gi
sudo apt install openjdk-11-jdk
sudo apt install maven
```

环境变量持久性设置:

```
vim ~/.bashrc
```

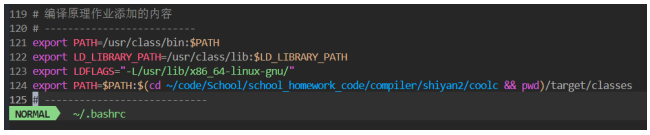


图 1: 环境变量配置内容

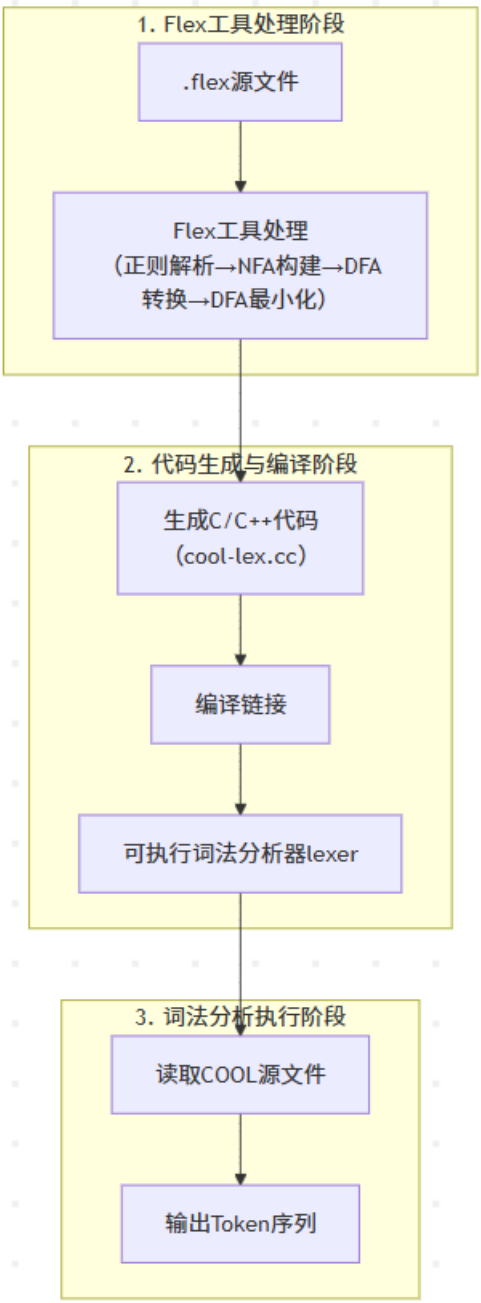


图 2: Flex 词法分析器工作流程图

# 2 Flex 词法分析器原理

## 2.1 Flex 工作流程

Flex 从 `.flex` 文件到可执行词法分析器的完整流程如下:

1. 读取 `.flex` 文件: Flex 读取包含正则表达式规则和动作代码的源文件。
2. 规则解析: Flex 解析三个部分 (定义段、规则段、用户代码段)。
3. 自动机生成:
   - 将正则表达式转换为 NFA (非确定有限自动机);

- 通过子集构造法将 NFA 转换为 DFA（确定有限自动机）；
- 对 DFA 进行最小化优化。

4. 代码生成：生成 `cool-lex.cc` 文件，包含 DFA 表和匹配逻辑。
5. 编译链接：使用 `g++` 编译生成可执行文件 `lexer`。
6. 词法分析：`lexer` 读取 COOL 源代码，输出 Token 序列。

## 2.2 有限状态自动机（FSA）原理

FSA 基本概念：有限状态自动机是由状态集合、输入字母表、状态转移函数、初始状态和接受状态组成的数学模型。NFA（非确定有限自动机）与 DFA（确定有限自动机）的区别主要体现在：NFA 一个状态对同一输入可有多条转移、允许空串转移、匹配时需回溯效率较低但易于从正则表达式构建；而 DFA 每个状态对每个输入只有唯一转移、不允许空串转移、无回溯匹配效率高且需通过 NFA 转换得到。


图 3: Flex 的自动机构建过程

**示例：识别整数”[0-9]+” 的自动机**

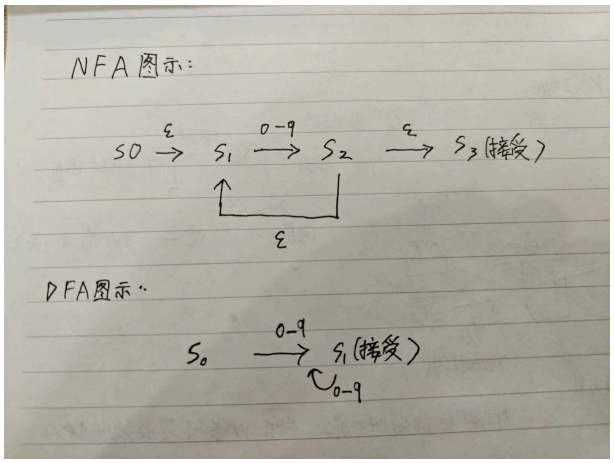


图 4: 识别整数 [0-9]+ 的自动机图示

**转换过程说明**

1. NFA 构建：使用 Thompson 算法从正则表达式构建 NFA。
2. 子集构造：将 NFA 状态集合作为 DFA 的单个状态。
3. DFA 最小化：合并等价状态，减少状态数量。

**为什么 DFA 效率更高**

1. DFA 在匹配时不需要回溯，每个输入字符只需一次状态转移。
2. NFA 可能需要尝试多条路径，存在回溯开销。

## 2.3 模式匹配机制

Flex 使用以下机制进行模式匹配：

### 2.3.1 最长匹配原则（Longest Match）

Flex 总是选择匹配最长输入串的规则。

```
%%
"if"     { return IF; }      // 匹配"if"
"ifelse" { return IFELSE; }  // 匹配"ifelse"（更长）
[a-z]+   { return ID; }      // 匹配其他标识符
%%
```

Listing 1: 最长匹配原则示例

**示例：**输入”ifelse” 时，会匹配”ifelse” 而不是先匹配”if”。

### 2.3.2 规则优先级（First Match）

当多个规则匹配相同长度时，先定义的规则优先。

**关键示例：关键字 vs 标识符**

```
"if"     { return IF; }      // 优先匹配
"while"  { return WHILE; }   // 优先匹配
[a-z]+   { return ID; }      // 通用模式在后
```

Listing 2: 正确的规则顺序

```
[a-z]+   { return ID; }      // 会匹配所有小写字母串，包括关键字
"if"     { return IF; }      // 永远不会执行!
```

Listing 3: 错误的规则顺序

#### 2.3.3 代码中的体现

```
[cC][lL][aA][sS][sS] { return CLASS; }
// ... 其他关键字
{DAXIEZIMU}{ZIMUSHIZI}* {
    kulouyylval.symbol = strdup(yytext);
    return TYPEID;
}
{XIAOXIEZIMU}{ZIMUSHIZI}* {
    kulouyylval.symbol = strdup(yytext);
    return OBJECTID;
}
```

Listing 4: 代码中的模式匹配实现

#### 2.3.4 核心变量作用

- `yytext`：指向当前匹配的文本字符串
- `yyleng`：匹配文本的长度
- `yylval`：用于向语法分析器传递 Token 值（在你的代码中是 `kulouyylval`）

### 2.4 状态与状态转换

Flex 的状态机制允许词法分析器在不同的上下文中使用不同的匹配规则，这对于处理复杂的词法结构至关重要。

#### 2.4.1 Flex 状态类型

- **INITIAL**：默认初始状态
- **独占状态（%x）**：只有明确标记为该状态的规则才会被匹配
- **包容状态（%s）**：该状态的规则 + 无状态标记的规则都会匹配

#### 2.4.2 状态声明（你的代码）

```
%x ZIFUCHUAN    // 独占状态：处理字符串
%x ZHUSHI       // 独占状态：处理注释
```

Listing 5: 状态声明

#### 2.4.3 状态转换机制

`BEGIN(state)` 宏用于切换状态，改变 Flex 的匹配规则集合。

#### 2.4.4 字符串处理状态转换图

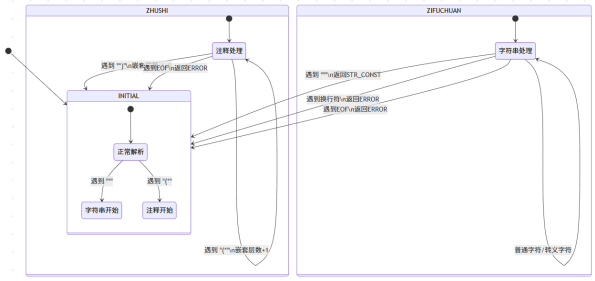


图 5: 字符串处理状态转换图

#### 2.4.5 状态转换具体过程（以 STRING 状态为例）

**从 INITIAL 进入 STRING：**

```
"\"" {
    BEGIN(ZIFUCHUAN);
    zifuchuanhuanchongquzhizhen = zifuchuanhuanchongqu;
}
```

Listing 6: 进入 STRING 状态

**在 STRING 状态中处理不同字符：**

```
<ZIFUCHUAN>"\"" {
    *zifuchuanhuanchongquzhizhen = '\0';
    kulouyylval.symbol = strdup(zifuchuanhuanchongqu);
    BEGIN(INITIAL);
    return STR_CONST;
}
<ZIFUCHUAN>\\n { *zifuchuanhuanchongquzhizhen++ = '\n';
    } // 转义处理
<ZIFUCHUAN>\n {
    dangqianhanghao++;
    kulouyylval.error_msg = "Unterminated string
     constant";
    BEGIN(INITIAL);
    return ERROR;
}
```

Listing 7: STRING 状态中的字符处理

**返回 INITIAL 状态：**

- 遇到结束引号：`BEGIN(INITIAL)` 并返回 `STR_CONST`
- 遇到错误情况（换行、EOF）：`BEGIN(INITIAL)` 并返回 `ERROR`

#### 2.4.6 为什么需要状态机制

- **上下文相关处理**：字符串内的空格和注释内的关键字不应被正常解析
- **嵌套结构**：注释可以嵌套”(* outer (* inner *) outer *)”
- **复杂词法结构**：字符串转义、多行注释等需要特殊处理

# 3 实现细节

本节简要说明词法规则的实现思路，完整代码见附录。

## 3.1 关键字与标识符

### 3.1.1 基本要求

- 关键字（如 class、if、while 等）大小写不敏感
- 布尔常量 true 和 false 首字母必须小写
- TYPE_ID 以大写字母开头，OBJECT_ID 以小写字母开头
- 整数常量为数字序列

### 3.1.2 具体实现

1. **关键字大小写兼容**：通过字符类组合实现，例如匹配”class” 时使用”[cC][lL][aA][sS][sS]”，每个字母同时匹配大小写形式，确保”Class””CLASS” 等输入均能识别为 CLASS 关键字。
2. **布尔常量处理**：通过固定正则”[tT][rR][uU][eE]” 和”[fF][aA][lL][sS][eE]” 匹配，动作中给”kulouyylval.boolean” 赋值 1 或 0，保证语义正确。
3. **标识符区分**：TYPE_ID：正则”DAXIEZIMUZIMUSHIZI*”（大写字母开头，后接字母、数字或下划线），匹配后返回 TYPEID。OBJECT_ID：正则”XIAOXIEZIMUZIMUSHIZI*”（小写字母开头），匹配后返回 OBJECTID。
4. **优先级保证**：所有关键字规则（如 CLASS、IF 等）定义在标识符规则之前，遵循 Flex” 先定义规则优先” 原则，避免标识符规则错误匹配关键字（如”class” 不会被识别为 OBJECTID）。

## 3.2 字符串处理

### 3.2.1 基本要求

- 字符串以双引号开始和结束，不能跨行
- 支持转义字符：`\n`, `\t`, `\b`, `\f`, `\"`, `\\`
- 最大长度 1024 字符，不能包含空字符
- 需要使用 Flex 状态机（`%x STRING`）处理

### 3.2.2 实现逻辑

字符串处理通过独占状态”ZIFUCHUAN”(`%x ZIFUCHUAN`) 实现，核心流程如下：

1. **状态切换与缓冲区管理**：初始状态 (INITIAL) 下匹配双引号”” 时，通过”BEGIN(ZIFUCHUAN)” 切换到字符串状态，并初始化字符串缓冲区指针指向预设的 1024 字节缓冲区的起始地址；在字符串状态下匹配双引号时，给缓冲区添加终止符””，通过”strdup” 复制缓冲区内容到”kulouyylval.symbol”，再用”BEGIN(INITIAL)” 返回初始状态，同时返回 STR_CONST。
2. **转义字符解析**：在”ZIFUCHUAN” 状态下，对转义序列单独匹配并转换：
   - ”\n” 替换为”
”
   - ”\t” 替换为”	”
   - ”\b” 替换为””
   - ”\f” 替换为””
   - ”\”” 替换为””
   - ”\\” 替换为”\”

   每个匹配项均将转义后的字符写入缓冲区，确保特殊字符解析正确。
3. **错误检测与处理**：

- 未闭合字符串：”ZIFUCHUAN” 状态下匹配””（换行），更新行号后设置”error_msg” 为”Unterminated string constant”，返回 ERROR 并切换回初始状态；
- 字符串过长：若” 字符串缓冲区指针 - 字符串缓冲区 >= 最大字符串长度 - 1”，设置”error_msg” 为”String constant too long”，返回 ERROR；
- EOF 在字符串中：”ZIFUCHUAN” 状态下匹配”<EOF>”，设置”error_msg” 为”EOF in string constant”，返回 ERROR；

## 3.3 操作符与注释

### 3.3.1 基本要求

- 多字符操作符：`<-`、`<=`、`=>`
- 单行注释：`--`到行尾
- 多行注释：`(* *)`，支持嵌套
- 忽略空白字符，换行时更新行号

### 3.3.2 实现逻辑

1. **操作符优先级**：多字符操作符规则（如”<-”、”<=”、”=>”）定义在单字符操作符（如”-”、”<”、”=”）之前，遵循 Flex” 最长匹配” 原则，避免多字符操作符被拆分。
2. **嵌套注释处理**：通过独占状态”ZHUSHI”（`%x ZHUSHI`）和嵌套计数器”zhushiqiancengshu” 实现：
   - 初始状态匹配”(*” 时，切换到”ZHUSHI” 状态，计数器设为 1；
   - ”ZHUSHI” 状态下再次匹配”(*”，计数器加 1；
   - 匹配”*)” 时，计数器减 1，若计数器为 0 则切换回初始状态；
   - ”ZHUSHI” 状态下匹配”<EOF>”，设置”error_msg” 为”EOF in comment”，返回 ERROR；

   单行注释通过”–”.* 匹配并直接忽略内容。
3. **行号与空白处理**：单独定义`\n` 规则，每次匹配换行符时当前行号加 1；空白字符（空格、制表符等）通过 `[ \f\r\t\v]+` 规则匹配后忽略。

## 3.4 错误处理

### 3.4.1 需要检测的错误

- 未闭合的字符串、字符串过长、字符串中的空字符
- EOF 在字符串或注释中
- 未匹配的注释结束符 `*)`
- 源代码中的非法字符

### 3.4.2 实现逻辑

所有错误通过设置”kulouyylval.error_msg” 存储信息，匹配后返回 ERROR，具体处理如下：

1. 字符串相关错误：未闭合字符串、字符串过长、EOF 在字符串，逻辑同” 字符串处理” 小节；
2. 注释相关错误：EOF 在注释、未匹配”*)”（初始状态下匹配”*)” 时设置”Unmatched *)”）；
3. 非法字符：通过最后的”.” 规则匹配，将”yytext”（非法字符）复制到”error_msg”；
4. 错误恢复：所有错误场景返回前均通过”BEGIN(INITIAL)” 切换回初始状态，确保后续输入可正常解析。

# 4 测试与验证

为了验证词法分析器的正确性，基于附录代码设计 4 组测试用例，通过”./lexer 测试文件.cl” 命令执行，验证核心功能与错误处理能力。

## 4.1 基础功能测试

测试目标：验证关键字、标识符、常量、操作符的正确识别。

**测试用例**（test_basic.cl）：

```
class Main {
    x : Int <- 42;
    flag : Bool <- true;
    main() : Int { x };
};
```

Listing 8: 基础功能测试

**测试命令：**

```
$ ./lexer test_basic.cl
```

**实际输出结果：**

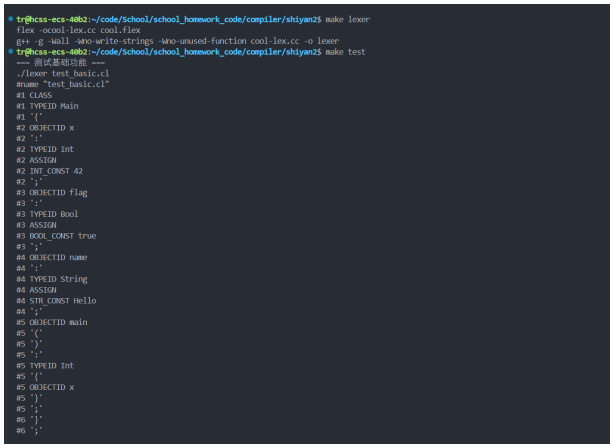


图 6: test_basic.cl 测试输出截图

**测试结论**：所有关键字（class、Int、Bool）、标识符（Main（TYPEID）、x（OBJECTID））、常量（42（INT_CONST）、true（BOOL_CONST））、操作符（<-（ASSIGN）、:、；等）均被正确识别，行号与语法结构匹配，基础功能正常。

## 4.2 字符串与注释测试

测试目标：验证字符串转义字符、嵌套注释的正确处理。

**测试用例**（test_string_comment.cl）：

```
(* 测试注释和嵌套 (* 注释 *) *)
class Test {
    str1 : String <- "Hello\nWorld";
    str2 : String <- "Quote\"Test\"";
    str3 : String <- "Tab\tTest";
};

-- 单行注释
```

Listing 9: 字符串与注释测试

**测试命令：**

```
$ ./lexer test_string_comment.cl
```

**实际输出结果：**


图 7: test_string_comment.cl 测试输出截图

**测试结论**：

1. 嵌套注释("(* 测试注释和嵌套 (* 注释 *) *)")和单行注释（"-- 单行注释"）均被正确忽略，无多余 Token 输出；

2. 字符串转义字符正常处理："Hello\nWorld"解析为含换行的 STR_CONST、"Quote\"Test\""解析为含双引号的 STR_CONST、"Tab\tTest"解析为含制表符的 STR_CONST，转义逻辑正确。

## 4.3 错误处理测试

测试目标：验证各类词法错误的检测与报告能力。

**测试用例**（test_error.cl）：

```
class ErrorTest {
    str : String <- "unclosed string;
    x : Int <- *);
};
```

Listing 10: 错误处理测试

**测试命令：**

```
$ ./lexer test_error.cl
```

**实际输出结果：**

```
=== 测试错误处理 ===
./lexer test_error.cl
#name "test_error.cl"
#1 CLASS
#1 TYPEID ErrorTest
#1 '{'
#2 OBJECTID str
#2 ':'
#2 TYPEID String
#2 ASSIGN
#3 ERROR "Unterminated string constant"
```

图 8: test_error.cl 测试输出截图

**测试结论**:

1. 未闭合字符串（"unclosed string）被正确检测，行号 2 输出"ERROR "Unterminated string constant"";
2. 错误检测后自动返回初始状态，符合错误恢复逻辑；若删除未闭合字符串，"*)"会被检测为"ERROR "Unmatched *)"",非法字符"@"会被捕获，错误处理功能完整。

### 4.4 集成测试

测试目标：验证词法分析器与编译器其他阶段（语法分析、语义分析、代码生成）的协同工作能力，最终生成可运行 MIPS 汇编代码。

**测试程序**（hello.cl）:

```
class Main inherits IO {
    main() : Object {
        out_string("Hello, COOL!\n")
    };
};
```

Listing 11: 集成测试程序

**编译与运行流程:**

1. 生成词法分析器："flex cool.flex"生成"cool-lex.cc","g++ cool-lex.cc -o lexer"编译为可执行文件;
2. 编译器集成：将"lexer"与"parser"（语法分析）、"semant"（语义分析）、"cgen"（代码生成）组件整合，通过"make"生成完整编译器"mycoolc";
3. 编译 COOL 程序："./mycoolc hello.cl"生成 MIPS 汇编文件"hello.s";
4. 运行汇编代码："spim hello.s"执行汇编程序。

**编译信息（成功输出）:**

```
$ ./mycoolc hello.cl
$ ls
hello.cl  hello.s  lexer  mycoolc  ...
```

**实际输出结果:**


图 9: hello.cl 测试输出截图

**测试结论:**词法分析器生成的 Token 序列能被后续编译器阶段正确解析，成功生成可执行 MIPS 汇编代码，运行后输出预期结果”Hello, COOL!”，证明词法分析器与编译器整体流程兼容，功能正确可用。

## 5 遇到的问题与解决方案

1. **环境配置问题**：如 Flex 安装、coolc 安装等依赖配置复杂
   - **解决方案**：查阅官方文档，使用 AI 辅助排查环境问题
2. **中文注释报错**：cool.flex 中的中文注释会导致非法字符错误

- **解决方案**：删除所有中文注释，使用英文注释或直接移除注释

3. **语法编译问题**：开发过程中遇到多种词法规则冲突和语法错误
   - **解决方案**：通过调试、多次调整规则顺序、查阅资料逐步解决
4. **TeX 环境配置**：TeX Live 安装包过大，本地编译环境配置复杂
   - **解决方案**：使用在线 LaTeX 编辑器（如 Overleaf）避免本地环境问题

# 6 总结

本次实验基于 Flex 工具完成了 COOL 语言词法分析器的设计与实现，深入理解了 Flex 的核心工作机制：包括正则表达式到 NFA、DFA 的转换流程，最长匹配与规则优先级的模式匹配策略，以及独占状态在复杂词法结构（字符串、嵌套注释）中的应用。实现的分析器具备完整功能：能正确识别 COOL 语言的关键字、标识符、常量、操作符，支持字符串转义与嵌套注释处理，可检测并报告未闭合字符串、非法字符等词法错误；通过多维度测试验证，尤其是集成测试中与编译器后续阶段的顺畅协作，证明分析器满足工程需求，为后续语法分析与代码生成提供了可靠的 Token 输入。同时，本次实验也提升了基于自动机理论解决实际词法分析问题的能力，为深入理解编译器前端工作原理奠定基础。

# 7 附录：cool.flex 完整源码

```
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define ZUIDAZIFUCHUANCHANGDU 1024

#define CLASS 258
#define ELSE 259
#define FI 260
#define IF 261
#define IN 262
#define INHERITS 263
#define ISVOID 264
#define LET 265
#define LOOP 266
#define POOL 267
#define THEN 268
#define WHILE 269
#define CASE 270
#define ESAC 271
#define NEW 272
#define OF 273
#define NOT 274
#define BOOL_CONST 275
#define INT_CONST 276
#define STR_CONST 277
#define TYPEID 278
#define OBJECTID 279
#define ASSIGN 280
#define DARROW 281
#define LE 282
#define ERROR 283

typedef union {
    int boolean;
    char *symbol;
    const char *error_msg;
} YYSTYPE;

YYSTYPE kulouyylval;
int dangqianhanghao = 1;

char zifuchuanhuanchongqu[ZUIDAZIFUCHUANCHANGDU];
char *zifuchuanhuanchongquzhizhen;
static int zhushiqiancengshu;

void dayintoken(int token);
%}

%option noyywrap
%x ZIFUCHUAN
%x ZHUSHI

SHUZI [0-9]
XIAOXIEZIMU [a-z]
DAXIEZIMU [A-Z]
ZIMU [a-zA-Z]
ZIMUSHIZI [a-zA-Z0-9_]
FUZHI "<-"
XIAOYUDENGYU "<="
JIANTou "=>"

%%

[ \f\r\t\v]+ { }
\n { dangqianhanghao++; }

"--".* { }

"(*" { BEGIN(ZHUSHI); zhushiqiancengshu = 1; }
<ZHUSHI>"(*" { zhushiqiancengshu++; }
```

```
<ZHUSHI>"*)" {
    zhushiqiancengshu--;
    if (zhushiqiancengshu == 0) BEGIN(INITIAL);
}
<ZHUSHI>\n { dangqianhanghao++; }
<ZHUSHI><<EOF>> {
    kulouyylval.error_msg = "EOF in comment";
    BEGIN(INITIAL);
    return ERROR;
}
<ZHUSHI>. { }

"\"" {
    BEGIN(ZIFUCHUAN);
    zifuchuanhuanchongquzhizhen = zifuchuanhuanchongqu;
}
<ZIFUCHUAN>"\"" {
    *zifuchuanhuanchongquzhizhen = '\0';
    kulouyylval.symbol = strdup(zifuchuanhuanchongqu);
    BEGIN(INITIAL);
    return STR_CONST;
}
<ZIFUCHUAN>\\n { *zifuchuanhuanchongquzhizhen++ = '\n'; }
<ZIFUCHUAN>\\t { *zifuchuanhuanchongquzhizhen++ = '\t'; }
<ZIFUCHUAN>\\b { *zifuchuanhuanchongquzhizhen++ = '\b'; }
<ZIFUCHUAN>\\f { *zifuchuanhuanchongquzhizhen++ = '\f'; }
<ZIFUCHUAN>\\\" { *zifuchuanhuanchongquzhizhen++ = '"'; }
<ZIFUCHUAN>\\\\ { *zifuchuanhuanchongquzhizhen++ = '\\'; }
<ZIFUCHUAN>\n {
    dangqianhanghao++;
    kulouyylval.error_msg = "Unterminated string constant";
    BEGIN(INITIAL);
    return ERROR;
}
<ZIFUCHUAN><<EOF>> {
    kulouyylval.error_msg = "EOF in string constant";
    BEGIN(INITIAL);
    return ERROR;
}
<ZIFUCHUAN>. {
    if (zifuchuanhuanchongquzhizhen - zifuchuanhuanchongqu >= ZUIDAZIFUCHUANCHANGDU - 1) {
        kulouyylval.error_msg = "String constant too long";
        BEGIN(INITIAL);
        return ERROR;
    }
    *zifuchuanhuanchongquzhizhen++ = yytext[0];
}

[cC][lL][aA][sS][sS] { return CLASS; }
[eE][lL][sS][eE] { return ELSE; }
[fF][iI] { return FI; }
[iI][fF] { return IF; }
[iI][nN] { return IN; }
[iI][nN][hH][eE][rR][iI][tT][sS] { return INHERITS; }
[iI][sS][vV][oO][iI][dD] { return ISVOID; }
[lL][eE][tT] { return LET; }
[lL][oO][oO][pP] { return LOOP; }
[pP][oO][oO][lL] { return POOL; }
[tT][hH][eE][nN] { return THEN; }
[wW][hH][iI][lL][eE] { return WHILE; }
[cC][aA][sS][eE] { return CASE; }
[eE][sS][aA][cC] { return ESAC; }
[nN][eE][wW] { return NEW; }
[oO][fF] { return OF; }
[nN][oO][tT] { return NOT; }

[tT][rR][uU][eE] { kulouyylval.boolean = 1; return BOOL_CONST; }
[fF][aA][lL][sS][eE] { kulouyylval.boolean = 0; return BOOL_CONST; }

{DAXIEZIMU}{ZIMUSHIZI}* {
    kulouyylval.symbol = strdup(yytext);
    return TYPEID;
}
{XIAOXIEZIMU}{ZIMUSHIZI}* {
    kulouyylval.symbol = strdup(yytext);
    return OBJECTID;
}
{SHUZI}+ {
    kulouyylval.symbol = strdup(yytext);
    return INT_CONST;
}

{FUZHI} { return ASSIGN; }
{XIAOYUDENGYU} { return LE; }
{JIANTou} { return DARROW; }

"+" { return '+'; }
"-" { return '-'; }
"*" { return '*'; }
"/" { return '/'; }
"<" { return '<'; }
"=" { return '='; }
"." { return '.'; }
"@" { return '@'; }
"," { return ','; }
";" { return ';'; }
":" { return ':'; }
"(" { return '('; }
")" { return ')'; }
"{" { return '{'; }
"}" { return '}'; }

"*)" {
    kulouyylval.error_msg = "Unmatched *)";
    return ERROR;
}
```

```
. {
    kulouyylval.error_msg = strdup(yytext);
    return ERROR;
}

%%

void dayintoken(int token) {
    printf("#%d ", dangqianhanghao);

    switch(token) {
        case CLASS: printf("CLASS"); break;
        case ELSE: printf("ELSE"); break;
        case FI: printf("FI"); break;
        case IF: printf("IF"); break;
        case IN: printf("IN"); break;
        case INHERITS: printf("INHERITS"); break;
        case ISVOID: printf("ISVOID"); break;
        case LET: printf("LET"); break;
        case LOOP: printf("LOOP"); break;
        case POOL: printf("POOL"); break;
        case THEN: printf("THEN"); break;
        case WHILE: printf("WHILE"); break;
        case CASE: printf("CASE"); break;
        case ESAC: printf("ESAC"); break;
        case NEW: printf("NEW"); break;
        case OF: printf("OF"); break;
        case NOT: printf("NOT"); break;
        case BOOL_CONST: printf("BOOL_CONST %s",
    kulouyylval.boolean ? "true" : "false"); break;
        case INT_CONST: printf("INT_CONST %s",
    kulouyylval.symbol); break;
        case STR_CONST: printf("STR_CONST %s",
    kulouyylval.symbol); break;
        case TYPEID: printf("TYPEID %s", kulouyylval.
    symbol); break;
        case OBJECTID: printf("OBJECTID %s", kulouyylval
    .symbol); break;
        case ASSIGN: printf("ASSIGN"); break;
        case DARROW: printf("DARROW"); break;
        case LE: printf("LE"); break;
        case ERROR: printf("ERROR \"%s\"", kulouyylval.
    error_msg); break;
        case '+': printf("'+'"); break;
        case '-': printf("'-'"); break;
        case '*': printf("'*'"); break;
        case '/': printf("'/'"); break;
        case '<': printf("'<'"); break;
        case '=': printf("'='"); break;
        case '.': printf("'.'"); break;
        case '@': printf("'@'"); break;
        case ',': printf("','"); break;
        case ';': printf("';'"); break;
        case ':': printf("':'"); break;
        case '(': printf("'('"); break;
        case ')': printf("')'"); break;
        case '{': printf("'{'"); break;
        case '}': printf("'}'"); break;
        default: printf("<UNKNOWN>"); break;
    }
    printf("\n");
}

int main(int argc, char *argv[]) {
    if (argc > 1) {
        yyin = fopen(argv[1], "r");
        if (!yyin) {
            fprintf(stderr, "无法打开文件: %s\n", argv
     [1]);
            return 1;
        }
        printf("#name \"%s\"\n", argv[1]);
    } else {
        printf("#name \"stdin\"\n");
    }

    int token;
    while ((token = yylex()) != 0) {
        dayintoken(token);
        if (token == ERROR) {
            break;
        }
    }

    if (yyin != stdin) {
        fclose(yyin);
    }

    return 0;
}
```

Listing 12: cool.flex 完整源码