编译原理课程实践 1：LEX 词法分析器

王靖康
515030910059
网络空间安全学院
wangjksjtu_01@sjtu.edu.cn

Abstract

实验基于 Linux 平台 LEX 的一个开源实现 Flex，实现了一个简易 LEX 词法分析器。该词法分析器按照要求对用户输入的脚本文件（只含英文）进行分析处理，实现了进制转换和程序注释内容小写化两个特定功能。通过作业和相关参考资料的学习，理解并熟悉了 LEX 基本结构框架，掌握了词法分析器的构造流程，能够熟练利用 Flex 工具快速实现高性能词法分析器。

1 LEX 基础

LEX 是一个**词法分析器的自动生成系统**。输入是一个文本文件，即 LEX 源程序，通常用 *.l* 表示。该文件通过对正规表达式的定义和相应的处理动作来完成符号串的识别和应采取的行动。输出是一个包含定义过程的 C 语言源代码文件，对该文件进行编译，可得到具有相应功能的词法分析器程序。

LEX 的工作原理是将源程序中的正规式表示为相应的有穷确定自动机 (DFA)。值得说明的是，LEX 生成的 C 代码是较为高效的，这依赖于正规表达式的高效匹配，且本文仅使用 Linux 平台的一个 LEX 开源实现 flex 来完成词法分析器的构建。

1.1 LEX 源程序

LEX 源程序包括三部分，**定义部分**，**规则部分**以及**辅助程序**。这三部分的由顶行的两个连续百分号 %% 分割，其中第一个 %% 不可以省略。以下通过对与 Linux 程序 wc 相同功能的 LEX 源程序的分析，介绍源程序的输入框架与各部分的语法与功能

如图 1 所示 (Figure 1)，源程序定义部分为 C 程序源代码，定义了字符，单词和行计数变量。C 程序代码需要用 %{ 和 %} 指定代码头部和尾部。该部分可以为变量的声明定义或函数定义，会被直接复制到生成的 C 文件中。除此之外，正规表达式的名字也可以在该部分定义（附录）。

源程序第二部分是规则部分，是由一连串的含有 C 语言的正规表达式组成。当匹配到对应的正规表达式时，这些 C 语言代码将会被执行。这些对应的代码被称为动作 (action)，实现对匹配字符串的处理。值得说明的是，当字符串可被多个正规表达式匹配时，取最长匹配项，若长度相同，则取最上的正规表达式。图 1 的程序就是利用了该性质实现了字符，单词和行数的统计。

辅助程序部分为一段 C 程序代码，用于在规则部分被调用且不在任何地方被定义的辅助程序，可能包含一个主程序。该部分会被直接复制到生成的 C 文件中。

fb1-2.l

```
/* just like Unix wc */
%{
    int chars = 0;
    int words = 0;
    int lines = 0;
%}

%%

[^ \t\n\r\f\v]+     { words++; chars += strlen(yytext);}
\n                  { chars++; lines++;}
.                   { chars++;}

%%

int main(int argc, char **argv)
{
    yylex();
    printf("%d %d %d\n", lines, words, chars);
}
```

Figure 1: 类 Linux 系统 wc 程序 LEX 源程序

1.2 正规表达式

正规表达式是 LEX 中的核心，允许匹配单个字符或字符串，应用十分广泛。常见的正则表达式元字符约定如表 1 所示。另外，有很多在线正则表达式调试匹配器可以帮助我们完成分析，已达到我们想要的功能。在实现作业中规定的源程序时，我使用了 Regular Expressions101 网站(https://regex101.com)提供的服务 (如图 2)，节约了正则表达式的构建时间，同时也保证了在多组测试数据上的通过。

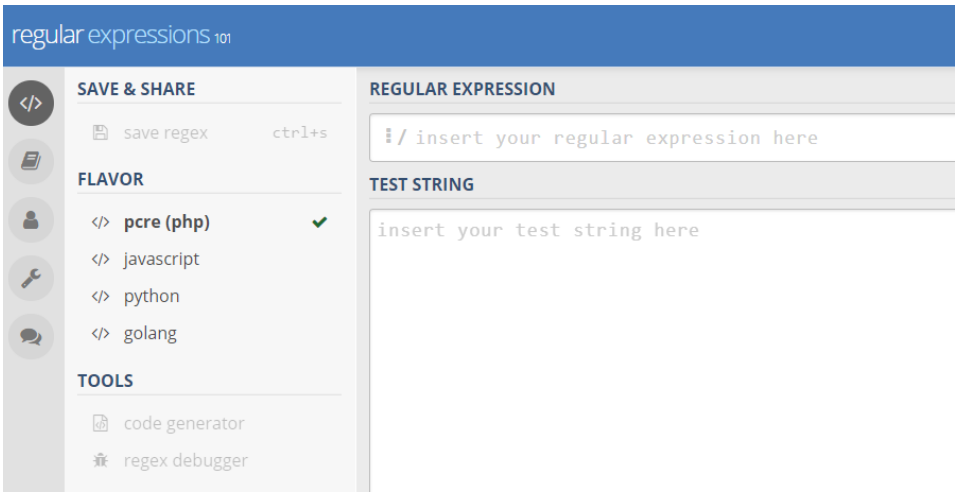

Figure 2: 在线正则表达式分析器

Table 1: LEX 正规表达式约定

格式	匹配意义
A	字符 A
.	除了新行之外的其他字符
"("	字符 (
$a*$	a 的闭包，即 a 的零次或多次重复，这里 a 是正规表达式
$a+$	a 的正闭包，即 a 的一次或多次重复式
$a?$	一个可选的 a
$a\|b$	a 或 b
$[abc]$	字符 a,b,c 中的任何一个
$[a-z]$	字符 a-c 中的任何一个
$[\wedge abc]$	除了字符 a,b 外任何一个其他字符
$a\{2,4\}$	字符 a 的 2 次到 4 次重复
$a\{2,\}$	字符 a 的 2 次以上重复
$\{xxx\}$	名字为 xxx 的正规表达式

Table 2: Flex 命令选项

命令选项	意义
$-f$	生成快速词法分析器
$-i$	生成大小写不敏感的分析器
$-s$	取消默认规则
$-t$	不产生 *lex.yy.c* 文件，将生生成的词法分析程序输出到标准输出上
$-v$	将 Flex 生成的词法分析器的有关统计结果打印到标准错误输出中
$-T$	将 Flex 中间过程输出到 stderr 中

1.3 Flex

Flex 是 Linux 平台上 LEX 的一个开源实现，可以从多种 Linux 包管理器中直接获得。例如 Arch Linux 中 **sudo pacman -S flex**, Debian 或者 Ubuntu 中 **sudo apt-get install flex**。常见的 Flex 命令选项如表 2 所示。详细命令可使用 **man flex** 查询

2 实验环境搭建

本文实验环境基于 Ubuntu16.04，也通过了 Arch Linux 环境测试。由于系统之前已经安装 gcc 编译器（系统默认安装），故省略 gcc 相关安装步骤。具体的安装和环境测试步骤为

$ sudo apt-get install flex
$ gcc –version
$ flex –version

本文实验环境，如表 3 所示。

Table 3: 实验环境配置

环境	版本
操作系统	Ubuntu-16.04 （测试：Arch linux)
gcc 编译器	gcc (Ubuntu 5.4.0-6ubuntu1 16.04.4) 5.4.0 20160609
flex	flex 2.6.0

3 正规表达式构建

正规表达式的构建和对应动作函数是 LEX 源程序的核心部分，该章节详细说明了本次作业需要构建的正则表达式和对应的动作函数。

本次程序的任务为对用户输入的 python 脚本文件（只含英文）进行分析处理，完成进制转换和注释字母小写化两个功能。具体功能表述为：1) 进制转换：在不影响代码的运行结果的前提下，对脚本分析，将代码中的非十进制数转化为十进制数; 2）注释小写化：# 用于单行注释，假设三引号仅用于多行注释，将注释中的大写字母全部转化为小写字母。因此，基于上述要求，我构建的正规表达式有以下两部分：

- python 程序中各种进制数字的识别（2 进制，8 进制，16 进制）
- python 程序注释段字符串（假设三引号只用于多行注释）

3.1 各进制数字

由于 python 程序中除十进制外仅有三种进制的数字：2 进制、8 进制和 16 进制。且各种进制匹配后需要进行的处理不同（即动作函数不同），故为简便容易理解起见，决定对其分开处理。

二进制数：python 中的二进制数的格式为 $0b$ 或 $0B$ 加二进制串，例如 $0b10100$ 和 $0B1110$。容易知道，正规式为 $0(b|B)(0|1)+$，有限状态机为 t 图 3(a)。所以依据表 1 的规则，可得正则表达式为 $0[bB][0-1]+$。

八进制数：：python 中的八进制数的格式为 0,$0O$ 或 $0o$ 加八进制串，例如 0135,$0O710$ 和 $0o177$。容易知道，正规式为 $0((o|O)(0|1|2|3|4|5|6|7)+)|((0|1|2|3|4|5|6|7)+)$，有限状态机为图 3(b)，其中 Σ_1 表示字符集合 [0-7]，所以依据表 1 的规则，可得正则表达式为 $(0[0-9]+)|(0[oO][0-7]+)$。

十六进制数：python 中的八进制数的格式为,$0x$ 或 $0X$ 加十六进制串，例如 $0x1AF$ 和 $0XD$。容易知道，正规式为 $0(x|X)(0|1|2|3|4|5|6|7|9|A|a|B|b|C|c|D|d|E|e|F|f)+$，有限状态机为图 3(c), 其中 Σ_2 表示字符集合 [0-9A-Fa-f]。所以依据表 1 的规则，可得正则表达式为 $0[xX][0-9a-fA-F]+$。

3.2 注释段字符串

python 注释段字符串分为两种，单行注释和多行注释。单行注释由 # 引起，注释到行末结束。多行注释由三引号引起，到下一个三引号终结（注意单双引号要匹配）。由上述描述，可得有限状态机为图 4（这里将三引号看成一个符号，为了容易理解，该状态机并不规范）。所以依据表 1 的规则，可得正则表达式为 $((\backslash"\backslash"\backslash")[\wedge\backslash"]^*(\backslash"\backslash"\backslash"))|((''')[\wedge\backslash"]^*(''''))|(\#.*\backslash n)$。

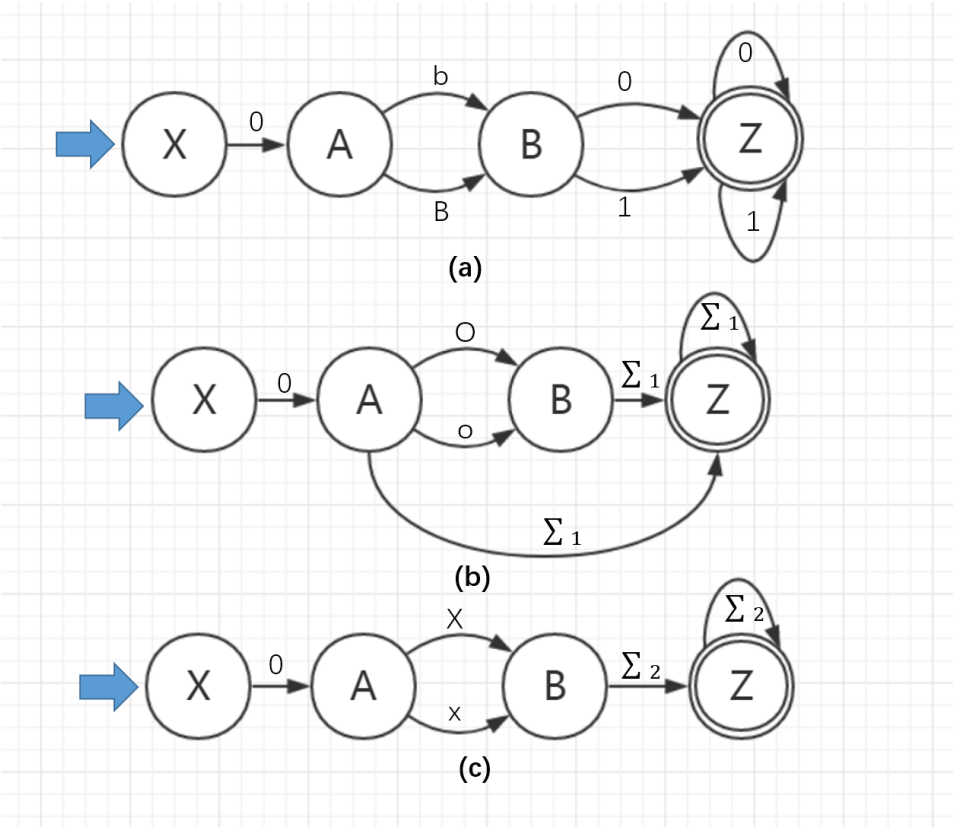

Figure 3: 不同进制 python 数字串的有限状态自动机

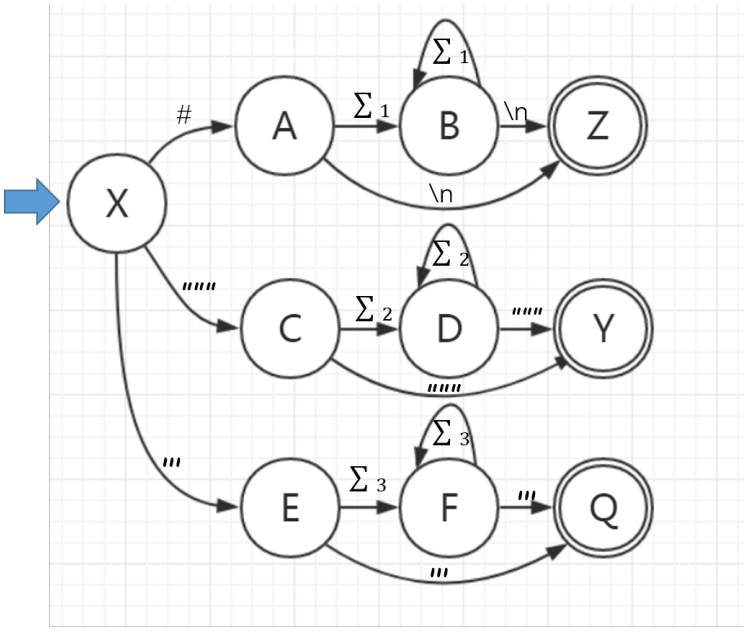

Figure 4: 注释段字符串的有限状态自动机

4 LEX 程序运行实例

本文完整代码可见附录（Appendix），词法分析器程序运行结果如图 5 所示（包含 flex 产生 c 文件，编译 c 文件，运行测试文件，对比输入输出）。

```
wangjk@asus-wjk:~/programs/lex/lab$ cat test.py && flex flex.l && \
> cc lex.yy.c -lfl && ./a.out < test.py
#!/usr/bin/env python

'''Main  FUCTION'''
def  main():
    a = (0o21 + 0x1c) * 2   # 0o21 equals 17
    b = 0b1001 * 0O37   # 0b1001 eqUALs 9
    c = 0XA1 - 55   # 0XA1 equals 161
    d = 0101   # 0101 EQUALS 65
    print a + b - c - d
'''END'''

if   __name__  ==  '__main__':
    """
    CALL
    main function
    """
    main()


#!/usr/bin/env python

'''main  fuction'''
def  main():
    a = (17 + 28) * 2   # 0o21 equals 17
    b = 9 * 31   # 0b1001 equals 9
    c = 161 - 55   # 0xa1 equals 161
    d = 65   # 0101 equals 65
    print a + b - c - d
'''end'''

if   __name__  ==  '__main__':
    """
    call
    main function
    """
    main()
```

Figure 5: LEX 程序运行实例

另外，本文测试了 LEX 词法分析器的效率。对于在图 1 源程序，本文将其与 Linux 标准程序 wc 进行对比，在 rockyou 口令集上进行了测试（数据量较大，以体现性能对比）。实验显示，LEX 生成的类 wc 程序效率仅仅略低于标准程序，这也说明 LEX 词法分析器具有高效性。

```
wangjk@asus-wjk:~/programs/lex$ time wc ../pwrnn/rockyou.random
 14341265  14438937 139842217 ../pwrnn/rockyou.random

real    0m1.675s
user    0m1.648s
sys     0m0.020s
wangjk@asus-wjk:~/programs/lex$ time ./wc < ../pwrnn/rockyou.random
14341265 14438943 139842217

real    0m1.959s
user    0m1.928s
sys     0m0.020s
```

Figure 6: wc 程序性能比较

5 结论

- 词法分析程序的功能是从左至右扫描源程序字符串，根据语言的词法规则识别出各类单词符号。构造词法分析程序主要有两种方法：手工和利用自动生成攻击 LEX 自动生成，本文采用后者。
- 实验基于 Linux 平台 LEX 的一个开源实现 Flex，实现了一个简易 LEX 词法分析器。该词法分析器按照要求对用户输入的脚本文件（只含英文）进行分析处理，实现了进制转换和程序注释内容小写化两个特定功能。
- 正规表达式的构建是核心，写正规式、构造有限状态机等有助于该过程的完成。同时，适当的正则表达式工具将会大大缩短该过程的时间。
- 通过性能对比实验，可以看出 LEX 除了具有简洁性外还具有高效性，其生成的 C 程序具有较高的效率。

Acknowledgments

非常感谢张老师和助教精心讲解编译原理相关知识并设计这个课程实践作业。通过这次作业，我体会到了 LEX 生成的词法分析器的简洁高效，同时更进一步加深了对正规式（正则表达式）以及有限状态自动机的理解。最后，感谢 OREILLY 图书 A Guide to Lex & Yacc 和在线网站 Regular Expressions101 的帮助。

References

[1] 刘铭, 徐兰芳, 骆婷. (2011) 编译原理（第三版）, 电子工业出版社

[2] Thomas Niemann. A Guide to Lex & Yacc, E-papers

[3] John.R.Levine. (2009) flex & bison, OREILLY

[4] Regular Expressions101. https://regex101.com/

Appendix

1. 感想和体会

本次作业的布置对于加深课本第三章内容（词法分析与有穷自动机）的理解非常有帮助，也加深了学习的兴趣（课堂知识的实践，编写效率很高的类 wc 程序等）。同时，Lex 和 Yacc 工具（之后第四章）的使用使我能够接触到现代编译器的构建过程之中，即自动生成词法、语法分析器，这是十分有实际意义的。同时，助教上传的作业指导手册 pdf 十分美观整洁，目测是使用了 LaTeX，这也激励我使用 LaTeX 完成了本报告的书写！从中我也学会了一些中文 LaTeX 的知识。

优点: 作业任务清晰，参考资料详尽，结合课堂紧密。

建议：建议同时加入 C 语言字符串匹配的训练（即手工方式实现一词法分析器）。

2. 词法分析器 LEX 源代码

```
%option noyywrap
%{
#include <stdio.h>
```

```
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

void lower(char* yytext) {
    int i = 0;
    while (yytext[i]) {
        putchar(tolower(yytext[i]));
        ++i;
    }
}

%}

digit_8_type0    0[0-9]+
digit_8_type1    0[oO][0-7]+
digit_2          0[bB][0-1]+
digit_16         0[xX][0-9a-fA-F]+
sline_comment    #.*\n
mline_comment    ((\"\"\")[^\"]*(\"\"\"))|(('''')[^\"]*(''''))

%%

{digit_16}          { printf("%d", (int)strtol(yytext, NULL, 16)); }
{digit_8_type0}     { printf("%d", (int)strtol(yytext, NULL, 8)); }
{digit_8_type1}     { printf("%d", (int)strtol(yytext + 2, NULL, 8)); }
{digit_2}           { printf("%d", (int)strtol(yytext + 2, NULL, 2)); }
{mline_comment}     { lower(yytext); }
{sline_comment}     { lower(yytext); }

%%

void main(void)
{
    yylex();
}
```