中国矿业大学计算机学院

2017级本科生课程设计报告

**课程名称** 系统软件开发实践

**报告时间** 2020年2月28日

**学生姓名** 陆玺文

**学 号** 03170908

**专 业** 计算机科学与技术

**任课教师** 张博

成绩考核

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 编号 | 课程教学目标 | | | 占比 | | 得分 |
| 1 | **目标1：**针对编译器中词法分析器软件要求，能够分析系统需求，并采用FLEX脚本语言描述单词结构。 | | | 15% | |  |
| 2 | **目标2：**针对编译器中语法分析器软件要求，能够分析系统需求，并采用Bison脚本语言描述语法结构。 | | | 15% | |  |
| 3 | **目标3：**针对计算器需求描述，采用Flex/Bison设计实现高级解释器，进行系统设计，形成结构化设计方案。 | | | 30% | |  |
| 4 | **目标4：**针对编译器软件前端与后端的需求描述，采用软件工程进行系统分析、设计和实现，形成工程方案。 | | | 30% | |  |
| 5 | **目标5：**培养独立解决问题的能力,理解并遵守计算机职业道德和规范，具有良好的法律意识、社会公德和社会责任感。 | | | 10% | |  |
| 总成绩 | | | | | |  |
| 指导教师 | |  | 评阅日期 | |  | |

目 录

[1、 实验一 词法分析器 1](#_Toc33777317)

[1.1 实验内容 1](#_Toc33777318)

[1.2 环境配置与代码验证 1](#_Toc33777319)

[1.2.1 Windows下使用Flex与Bison的集成开发环境 1](#_Toc33777320)

[1.2.2 Linux环境下代码验证 3](#_Toc33777321)

[1.3 Lex代码分析 3](#_Toc33777322)

[1.3.1 程序组成 3](#_Toc33777323)

[1.3.2 模式匹配原理与规则 4](#_Toc33777324)

[1.3.3 变量与函数 5](#_Toc33777325)

[1.3.4 程序输出结果分析 5](#_Toc33777326)

[1.4 实验感悟 7](#_Toc33777327)

# 实验一 词法分析器

## 实验内容

1. 阅读《Flex/Bison.pdf》第一、二章，掌握Flex基础知识。
2. 利用Flex设计一个词法扫描器，用于统计输入文本文件中的字符数、单词数和行数。

## 环境配置与代码验证

实验一中lex1与lex2代码均已给出，在Windows和Linux操作系统环境下分别安装配置Flex的编译环境并运行。

### Windows下使用Flex与Bison的集成开发环境

#### 环境安装

在《编译原理》课程学习过程中，使用过Flex与Bison的集成开发环境（下载地址： <https://download.csdn.net/download/weixin_42577438/11862600> ）。开发界面如图 1‑1所处。

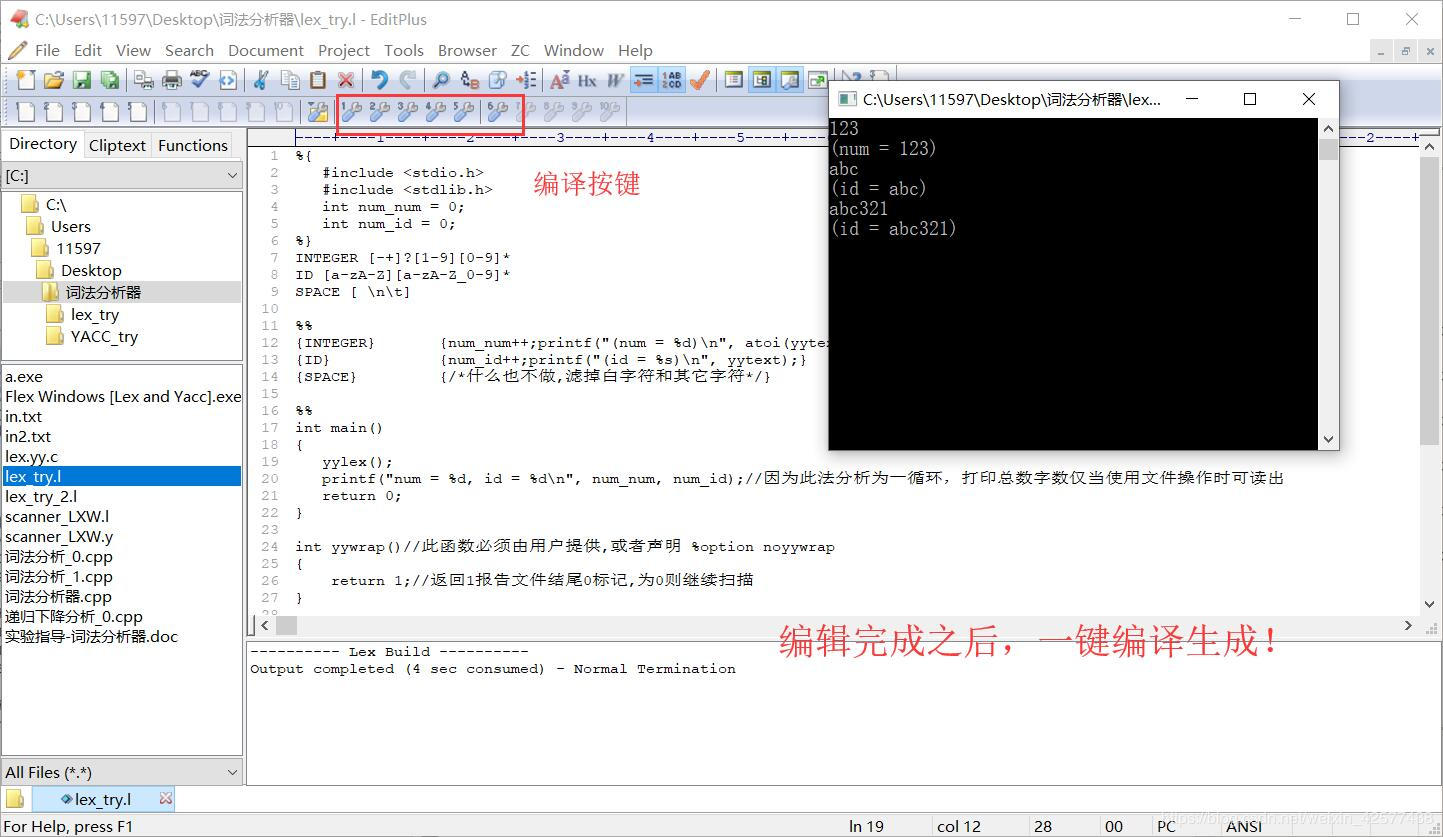


图 ‑1 开发环境界面

#### 软件使用

编辑完文本后，保存为.l后缀格式文件，接下来执行如下编译执行顺序即可：

1. 点击 Lex File Compile（形状为：扳手1）
2. 点击 Lex Build （形状为：扳手2）
3. 点击 Execute exe directly （形状为：扳手5）

在个人博客上有发布过关于该软件的使用，以及之后与Yacc联系使用的更加详细博文（地址：<https://blog.csdn.net/weixin_42577438/article/details/102544803>）。

#### 代码验证

代码 ‑1 test1.l 代码

|  |  |
| --- | --- |
|  | **%{** |
|  | **int** nchar**,**nword**,**nline**;** |
|  | **%}** |
|  |  |
|  | **%%** |
|  | \n **{**nline**++;**nchar**++;}** |
|  | **[^** \t\n**]+** **{**nword**++,**nchar**+=**yyleng**;}** |
|  | **.** **{**nchar**++;}** |
|  |  |
|  | **%%** |
|  | **void** main**()** |
|  | **{** |
|  | yylex**();** |
|  | printf**("%d \t %d \t %d \n",** nchar**,**nword**,**nline**);** |
|  | **}** |
|  | **int** yywrap**()** |
|  | **{** |
|  | **return** 1**;** |
|  | **}** |

代码 1‑1即为使用的示例代码lex1.l，在Windows环境下的运行结果如图 1‑2所示（104 17 6）。

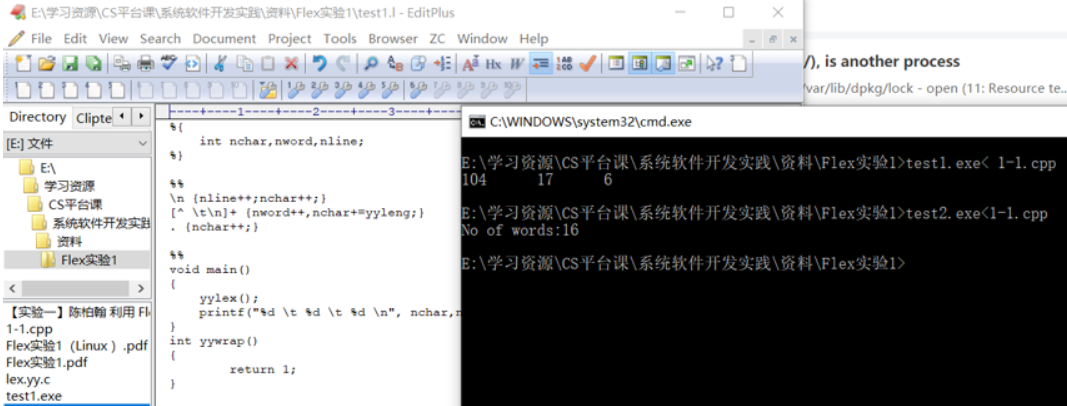


图 ‑2 test1.l 编译后运行结果

示例代码lex2.l在Windows环境下的运行结果在图 1‑2中也展现出来了（No of words: 16）。

### Linux环境下代码验证

所使用的虚拟机系统为Ubuntu19.04，在终端执行指令“apt install flex”安装完成flex。

#### 代码验证

在编辑完成代码文件后，使用命令`flex test1.l`进行编译。使用命令`cc -o parser lex.yy.c`进行链接，使用命令`./parser < file.txt`进行运行，最终运行结果如图 1‑3所示。

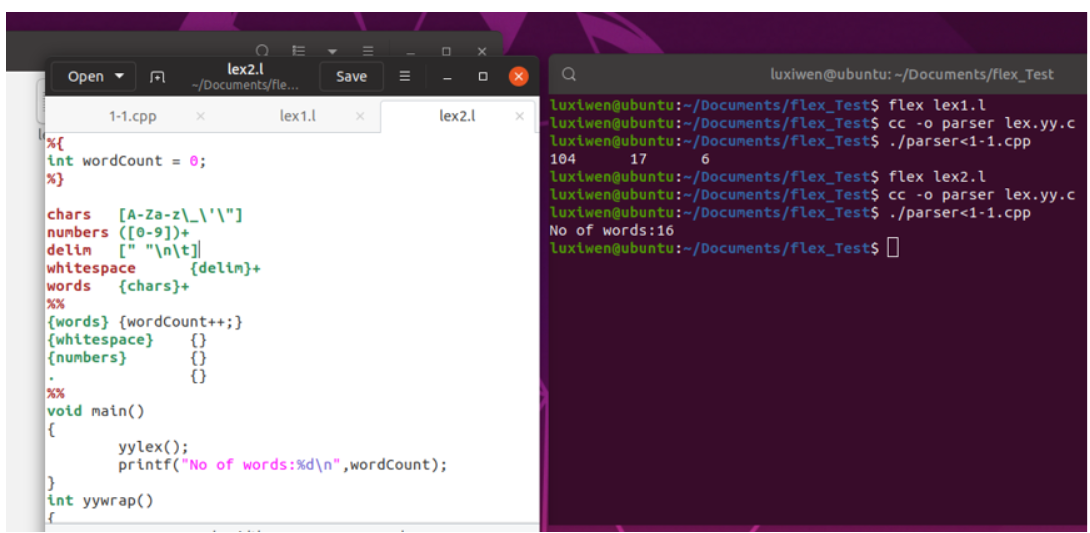


图 ‑3 Linux下实验一运行结果

其中lex1.l运行结果为：104 17 6，lex2.l运行结果为：No of words: 16。

## Lex代码分析

在《编译原理》课程中学习到，单词匹配或者说模式识别的核心技术为构造出相应文法的识别**自动机**，而正规式与自动机之间具有形式上的联系，Flex作为词法分析器，本质上完成了自动机的构造。通过所预定义的正规式，Flex可以很好的帮助构建相应的字母**转移矩阵**，来完成识别。在我的个人博客中，关于自动机完成识别有过相应练习（地址：<https://blog.csdn.net/weixin_42577438/article/details/101872798> ）。

### 程序组成

Flex程序总体上可以分为三段，段与段之间以%%分界。以代码 1‑1为例，行1~3为第一段，用作C和Lex的全局声明部分，分别定义了三个全局变量；行5~10为第二段，定义了相应的模式（正规式），以及识别之后的执行动作；行11~19为第三段，补充了所要执行的C函数。

### 模式匹配原理与规则

#### 模式匹配原理

由自动机原理知，正则式可以对应构造出DFA转移矩阵，因而我们需要在Flex程序的第二段中，给定出需要的识别正则式，进而在Lex帮助下完成单次扫描识别。

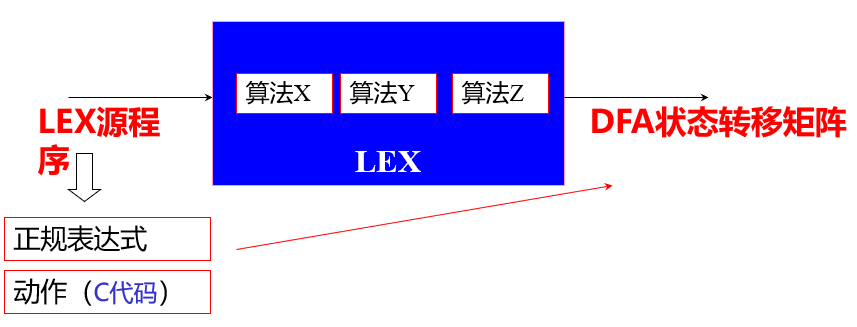


图 ‑4 Flex原理图

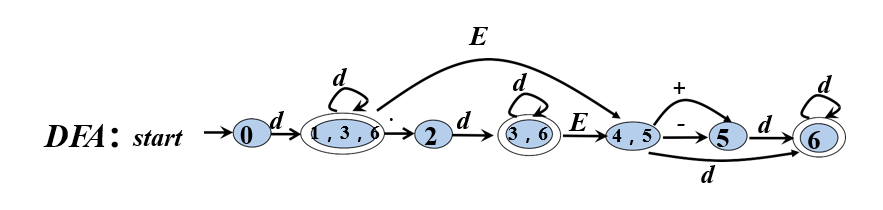


图 ‑5 自动机识别原理（无符号整数示例）

Flex的源程序（\*.yy.c）在此原理的基础上，分为两个部分：①状态转移矩阵（DFA）②控制执行部分。

#### 模式匹配规则

匹配的规则即相应的正则表达式规则，在参考书《Flex与Bison》P25中可以查阅到许多正规表达式的规则。以代码 1‑1中所使用到的部分符合为例：

表格 1 部分匹配规则（正则式规则）

|  |  |
| --- | --- |
| ^ | 匹配行首，也被用于方括号中表示补集。 |
| [a-z] | 字符类，可以匹配方括号中的任意一个字符。 |
| \ | 用来表示元字符本身和一部分常用的C语言转义序列。 |
| \* | 匹配领个或多个紧接在前面的表达式。 |
| + | 匹配一个或多个紧接在前面的表达式。 |
| “…” | 引号中的字符将基于字面意义被解释。 |

### 变量与函数

在Flex中有一些内置的变量与函数，可以帮助我们更加便捷的使用。在参考书《Flex与Bison》第5章中，可以查阅到许多Flex内置的变量与函数。总的来说，当模式匹配时执行的C代码可以包含一条返回语句，它将从yylex()返回相应的值给调用方，通常该调用方法也就是yacc所生成的语法分析器。

以代码 1‑1中使用到的函数和变量为例：

yyleng，当词法分析器匹配一个记号时，记号的文本被存放在以空字符结束的字符串yytext中，它的长度是在yyleng中。

Yyless(n)，可以推回记号的前n个字符。

Yymore()，使得lex把下一个记号也添加到当前记号中。

Yywrap()，当词法分析器到达文件末尾时，可以选择性的调用例程yywrap()来了解下一步操作。如果yywrap()返回0，词法分析器继续分析；返回1，词法分析器将返回一个零记号来表明文件结束。

### 程序输出结果分析

代码 1‑1，的输出结果为104 17 6 表明，1-1.cpp中包含有104个字符，17个单词，6行。模式匹配的优先级自上而下，因而识别到换行符则行数加一，识别到非空格非制表符非换行符的字符时，单词数加一，识别到其他字符使，字符数加一。

Test2.l代码如

代码 1‑2所示，主要目的忽略掉空白符之后计算单词数目，最终输出结果为：No of words: 16 。

代码 1‑2 test2.y文件

|  |  |
| --- | --- |
|  | **%{** |
|  | **int** wordCount **=** 0**;** |
|  | **%}** |
|  |  |
|  | chars **[**A**-**Za**-**z\\_\'\.\"] |
|  | numbers **([**0**-**9**])+** |
|  | delim **[“** “\n\t**]** |
|  | whitespace **{**delim**}+** |
|  | words **{**chars**}+** |
|  |  |
|  | **%%** |
|  | **{**words**}** **{**wordCount**++;**printf**("chars:%s\n",**yytext**);}** |
|  | **{**delim**}** **{**printf**("delim:%s\n",**yytext**);}** |
|  | **{**whitespace**}** **{**printf**("whitespace:%s\n",**yytext**);}** |
|  | **{**numbers**}** **{}** |
|  | **.** **{}** |
|  | **%%** |
|  | **void** main**()** |
|  | **{** |
|  | yylex**();** |
|  | printf**("No of words:%d\n",** wordCount**);** |
|  | **}** |
|  | **int** yywrap**()** |
|  | **{** |
|  | **return** 1**;** |
|  | **}** |

代码2中的**关键问题**在于，行7中使用了引号意在表示空白字符本身，然而在Flex中被理解成了匹配引号，所以源文件中的”被当成空白符忽略掉了。为了进一步验证这个问题，打印出所有匹配的单词和delim，同时修改行7为[ \t\n]，结果如图 1‑6所示。

可以看到在去掉delim中的引号后，引号可以被正确语义的识别到chars中，猜想是正确的。

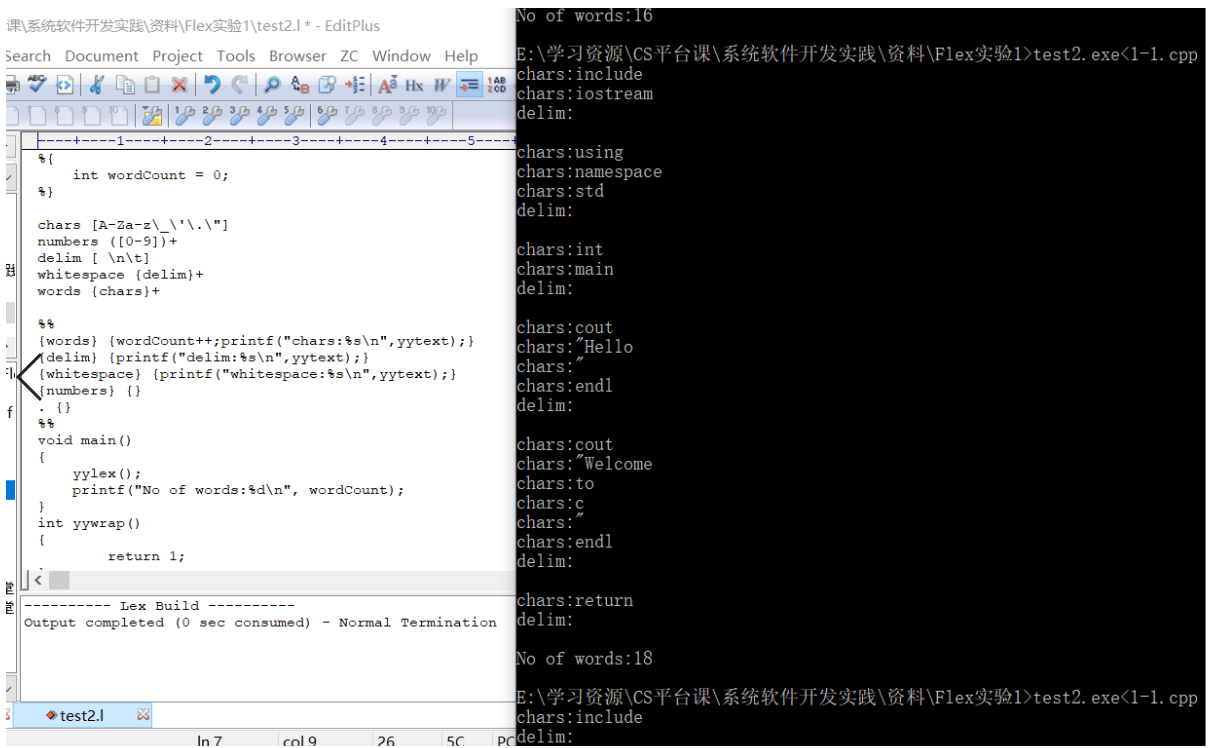


图 ‑6 引号匹配探索

## 实验感悟

上学期《编译原理》课程中初步接触过Flex和Bison，这一次的实验更好的去熟练了Flex的运用，同时对于函数与变量有了更加深入的理解。

实验过程中发现了引号下空格不能被仅理解成空格，算是意外收获。