

中国矿业大学计算机学院

2019 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 2022.2.25

学生姓名 王杰永

学 号 03190886

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	

Flex 实验一

1 实验内容.....	1
2 Flex 环境搭建	1
2.1 Windows 环境下 Flex 的配置	1
2.2 Linux 环境下 Flex 的配置	1
3 Flex 源码分析	2
3.1 lex1.l 源码分析	2
3.2 lex2.l 源码分析	3
4 实验结果及分析.....	4
4.1 lex1 结果分析	4
4.2 lex2 结果分析	5
5 实验总结.....	5

1 实验内容

- 1) 阅读《Flex/Bison.pdf》第一章，第二章，掌握 Flex 基础知识。
- 2) 利用 Flex 设计一个词法扫描器，用于统计输入文件中的字符数，单词数和行数。

2 Flex 环境搭建

2.1 Windows 环境下 Flex 的配置

首先以管理员权限打开资料中的 flex-2.5.4a-1.exe。

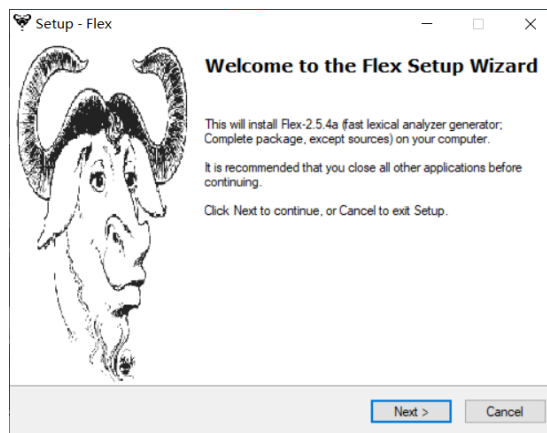


图 1 Flex 的安装

点击 Next，接受许可协议，选择安装路径，即可安装成功。将安装好的 flex 根目录下的 bin 文件夹加入系统环境变量，以便在任何路径下都可以使用 flex 命令。

2.2 Linux 环境下 Flex 的配置

我选用的 Linux 发行版是 ubuntu18.04，可以使用 apt 很方便的安装 flex。

打开终端，使用命令 `sudo apt install flex`，输入管理员密码，等待安装完成。

```
chen@chen: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
chen@chen:~$ sudo apt install flex  
[sudo] chen 的密码:  
正在读取软件包列表... 完成  
正在分析软件包的依赖关系树  
正在读取状态信息... 完成  
将会同时安装下列软件:  
  libfl-dev libfl2 libsigsegv2 m4  
建议安装:  
  bison build-essential flex-doc m4-doc  
下列【新】软件包将被安装:  
  flex libfl-dev libfl2 libsigsegv2 m4  
升级了 0 个软件包，新安装了 5 个软件包，要卸载 0 个软件包，有 25 个软件包未被升级。  
需要下载 545 kB 的归档。  
解压缩后会消耗 1,511 kB 的额外空间。  
您希望继续执行吗？ [Y/n]
```

图 2 ubuntu 安装 flex

3 Flex 源码分析

3.1 lex1.l 源码分析

lex1 要求利用 Flex 设计一个词法扫描器，用于统计输入文件中的字符数，单词数和行数。其源代码如下。

```
1. /*word count*/
2. %{
3.     int chars = 0;
4.     int words = 0;
5.     int lines = 0;
6. }%
7. %%
8. [^ \t\n]+ {words++; chars += yyleng;}
9. \n {
10.     chars++;
11.     lines++;
12. }
13. . {
14.     chars++;
15. }
16. %%
17. int main(int argc, char **argv){
18.     yylex();
19.     printf("%d\t%d\t%d\n", chars, words, lines);
20.     return 0;
21. }
22.
23. int yywrap(){
24.     return 1;
25. }
```

第 2 行至第 6 行被 `%{` 与 `%}` 包裹，属于全局声明，其代码会复制到生成的 `c` 代码的最前端。在这里定义三个变量分别用于统计字符数，单词数与行数。

在第 7 行使用 `%%` 分割，其下的行 8 至行 15 属于词法分析模式匹配段，用于定义词法规则及匹配到特定规则后执行的动作。

行 8 定义了遇到非空格非换行符非制表符之外其他任意串（即单词）的动作——单词数加一，字符数增加匹配到的串的长度。

行 9-12 定义了遇到换行符后的动作——字符数与代码行数均加一。

行 13-15 定义了遇到任意字符的动作——字符数加 1。由于正则匹配的原则，被上两条匹配到的串不会被该规则匹配。

最后，在行 16 的%%下的部分，会直接填补到生成的 c 文件的末尾。主函数中调用了 flex 词法分析函数的入口 yylex()，并在分析完成后输出全局变量的值。

3.2 lex2.l 源码分析

lex2 仍然是要求统计单词数，其源代码如下。

```
1. %{
2.     int wordCount = 0;
3. }
4. chars [A-Za-z\_\'\".]
5. numbers ([0-9])+
6. delim [" "\n\t]
7. whitespace {delim}+
8. words {chars}+
9. %%
10. {words} {wordCount++;}
11. {whitespace} {/*donothing*/}
12. {numbers} {/*one may want to add some processing here*/}
13. . {/*donothing*/}
14. %%
15. void main(){
16.     yylex();
17.     printf("%d\n", wordCount);
18. }
19. int yywrap(){
20.     return 1;
21. }
```

行 4-8 定义了一些正则表达式的变量。

行 4 定义变量 **chars**，指代所有大小写字母以及下划线单引号双引号和小数点之一。

行 5 定义变量 **numbers**，指代任意长度大于 1 的数字串。

行 6 定义变量 **delim**，指代空格、换行符、制表符中的一个。

行 7 定义变量 **whitespace**，指代任意长度大于 1 的 **delim**。

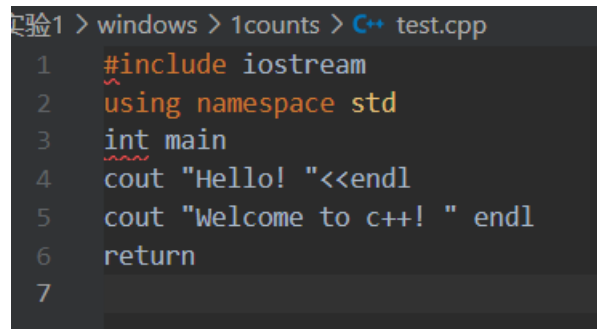
行 8 定义变量 **words**，指代任意长度大于 1 的 **chars**。

最后，在行 10，当词法分析过程中匹配到 **words**，则进行单词数加一的动作。

4 实验结果及分析

4.1 lex1 结果分析

用于 lex1 词法分析的示例文本如下图。



```
实验1 > windows > 1counts > C++ test.cpp
1  #include iostream
2  using namespace std
3  int main
4  cout "Hello! "<<endl
5  cout "Welcome to c++! " endl
6  return
7
```

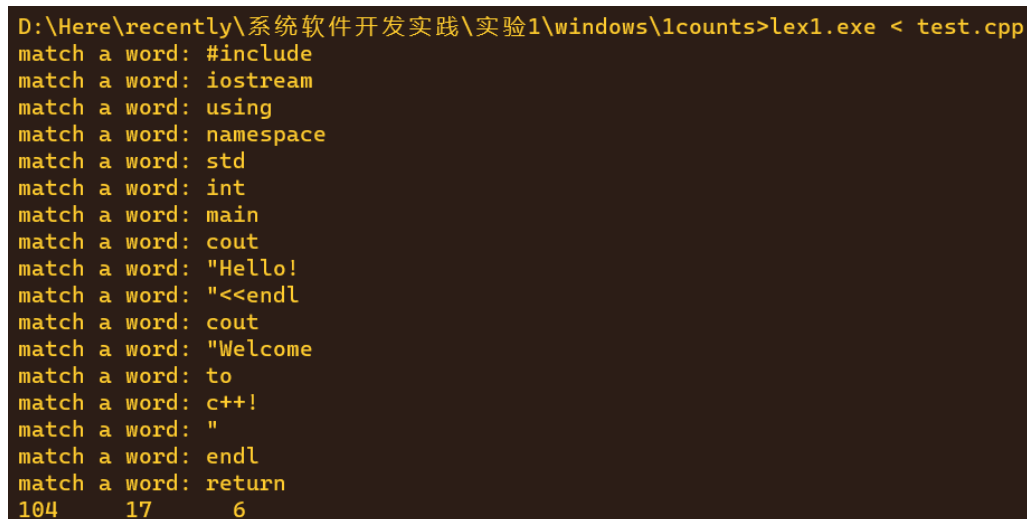
图 3 用于 lex1 词法分析的文本

从源码中可以看出，源码以空格、换行符和制表符作为单词的分隔符，因此#include会作为一个单词。在源码中的单词匹配模式处，加入如下语句

```
printf("match a word: %s\n",yytext)
```

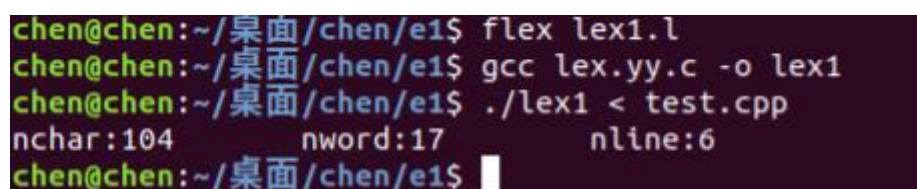
最终可以看出是哪 17 个单词被匹配了。

Windows 与 Linux 系统的运行结果如图。



```
D:\Here\recently\系统软件开发实践\实验1\windows\1counts>lex1.exe < test.cpp
match a word: #include
match a word: iostream
match a word: using
match a word: namespace
match a word: std
match a word: int
match a word: main
match a word: cout
match a word: "Hello!
match a word: "<<endl
match a word: cout
match a word: "Welcome
match a word: to
match a word: c++!
match a word: "
match a word: endl
match a word: return
104      17      6
```

图 4 Windows 系统下 lex1 的运行结果



```
chen@chen:~/桌面/chen/e1$ flex lex1.l
chen@chen:~/桌面/chen/e1$ gcc lex.yy.c -o lex1
chen@chen:~/桌面/chen/e1$ ./lex1 < test.cpp
nchar:104      nword:17      nline:6
chen@chen:~/桌面/chen/e1$
```

图 5 Linux 系统下 lex1 的运行结果

4.2 lex2 结果分析

lex2 源程序在 Windows 与 Linux 系统下的运行结果如下图。

```
D:\Here\recently\系统软件开发实践\实验1\windows\1counts>flex lex2.l
D:\Here\recently\系统软件开发实践\实验1\windows\1counts>gcc lex.yy.c -o lex2.exe
D:\Here\recently\系统软件开发实践\实验1\windows\1counts>lex2 < test.cpp
16
```

图 6 Windows 系统下 lex2 的运行结果

```
chen@chen:~/桌面/chen/e1$ flex lex2.l
chen@chen:~/桌面/chen/e1$ gcc lex.yy.c -o lex2
chen@chen:~/桌面/chen/e1$ ./lex2 < test.cpp
16
chen@chen:~/桌面/chen/e1$
```

图 7 Linux 系统下 lex2 的运行结果

5 实验总结

通过本次实验，初步掌握了 Flex 的.l 文件的基本构成，学习了各种 flex 允许的各种正则表达式的规则，加深了正则表达式的理解并得以更加熟练的运用正则表达式去解决实际问题。受益匪浅。