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 年 2 月 22 日

学生姓名 胡钧耀

学　　号 06192081

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

实验（一） 利用 Flex 构造编译器

1.1 实验目的

用 Flex 设计扫描器程序，计算一个文件中的字符数，单词数和行数。

1.2 Flex 的安装与配置

1.2.1 Windows 环境

安装 *flex-2.5.4a-1.exe*。安装后，把安装路径加入环境变量。

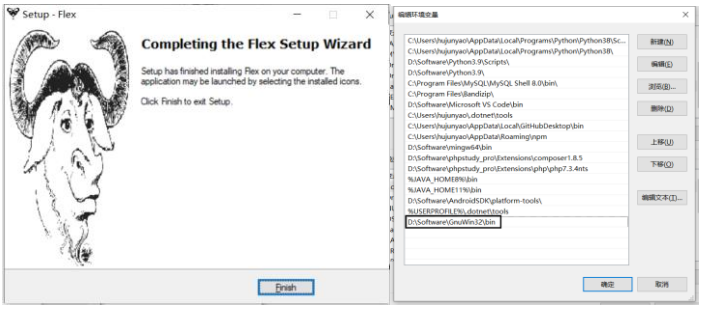

图 1 Flex 的安装，并将其加入环境变量

1.2.2 Linux 环境

在 VMware Workstation Pro 运行 Ubuntu 操作系统，使用命令行工具安装 *flex*。

```
sudo apt install flex
```

图 2 Linux 命令行安装 flex

1.3 分析 Flex 代码

1.3.1 lex1.l

```
%{
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;
}
```

该文件一共有分为三段，每段中间使用%%进行分隔，这三段分别是声明全局变量、需要匹配的各种模式和补充的以 C 语言为基础的函数。

首先是全局声明，这里定义三种变量，分别是字符数量、单词数量和行数。全局声明使用%{int a}%类似 C 语言的格式加上外层标记即可。

其次是匹配模式，一行是一种匹配模式，由正则表达式及其匹配后 C 语言动作构成。这里有三种动作：第一，遇到回车时，行数字符数都自增 1；第二，遇到不是制表符和回车时，单词数自增 1，字符数增加当前匹配模式的长度，也就是单词长度；第三，遇到除了回车的任意字符，都进行字符数自增 1 的操作。

再就是 C 语言主函数。其中出现两个函数。第一个是 yylex 函数，函数开始分析输入的文件，这由 lex 自动生成。第二个是 yywrap 函数，这一函数在文件（或输入）的末尾调用。如果函数的返回值是 1，就停止解析。因此它可以用来解析多个文件。代码可以写在第三段，这就能够解析多个文件。方法是使用 yyin 文件指针（见上表）指向不同的文件，直到所有的文件都被解析。最后，yywrap 可以返回 1 来表示解析的结束。printf 输出字符数、单词数、行数。

1.3.2 lex2.l

```
%{
int wordCount = 0;
%}
chars [A-Za-z\_\'\.\"]
numbers ([0-9])+
delim [" "\n\t]
whitespace {delim}+
words {chars}+
%%
{words} { wordCount++; /*increase the word count by one*/ }
{whitespace} { /* donothing*/ }
{numbers} { /* one may want to add some processing here*/ }
%%
void main()
{
yylex(); /* start theanalysis*/
```

```
printf(" No of words:%d\n", wordCount);
}
int yywrap()
{
return 1;
}
```

文件一共有四段，每段中间使用%%进行分隔，这四段分别是声明全局变量、声明正则表的式的别称、需要匹配的各种模式和补充的以 C 语言为基础的函数。

首先是全局声明，这里定义一种变量，是单词数量。全局声明使用%{int a}%类似 C 语言的格式加上外层标记即可。

其次是 lex 标记声明（正则别名命名），下面一一介绍：chars 匹配单字符（大小写、下划线、单双引号、点），numbers 匹配数字（0、123 这些，而不是仅匹配一位数字），delim 匹配双引号、空格（**注意：这里不能理解为单个空格**）、单个回车、单个制表符，whitespace 则是匹配一个或者多个 delim（也就是一个或多个空格、回车、制表符的混合形式），words 则是匹配一个或者多个 chars（也就是多个大小写、下划线、单双引号、点的混合形式）。

然后是三种动作：第一，遇到 words 时，wordCount 单词计数自增 1；但是对于另外两个，虽然写出了需要匹配空白和数字，但是都未进行任何操作，也就是仅仅进行单词的计数。

再就是 C 语言主函数。其中出现两个函数。第一个是 yylex 函数，函数开始分析输入的文件，这由 lex 自动生成。第二个是 yywrap 函数，这一函数在文件（或输入）的末尾调用。如果函数的返回值是 1，就停止解析。因此它可以用来解析多个文件。代码可以写在第三段，这就能够解析多个文件。方法是使用 yyin 文件指针（见上表）指向不同的文件， 直到所有的文件都被解析。最后，yywrap 可以返回 1 来表示解析的结束。printf 函数输出单词的计数。

1.4 Windows 环境下使用 Flex 编译、运行源程序

1.4.1 lex1.l

打开 Developer Command Prompt for VS 2022 命令行，运行如下代码，可调用 *flex.exe*。生成 *lex1.yy.c* 文件。

```
cd D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\实验 1
flex.exe -o"lex1.yy.c" lex1.l
```

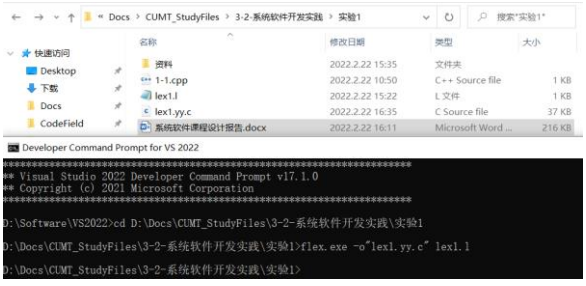

图 3 Windows 命令行运行生成.c 文件

在命令窗口输入如下代码，调用 VS 的编译器 *cl.exe*，进而生成 *lex1.yy.exe* 和 *lex1.yy.obj* 文件。

```
cl lex1.yy.c
```

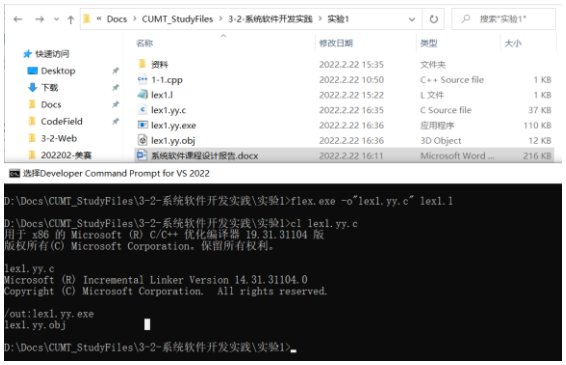

图 4 Windows 命令行生成.obj 和.exe 文件

本实验 *1-1.cpp* 样例代码如下。**注意：return 后面还有一行是空行。**

```
#include iostream
using namespace std
int main
cout "Hello! "<<endl
cout "Welcome to c++! " endl
return

```

调用 *lex1.yy.exe,*对 *1-1.cpp* 进行词法分析。

```
lex1.yy.exe < 1-1.cpp

结果:
104    17    6
```

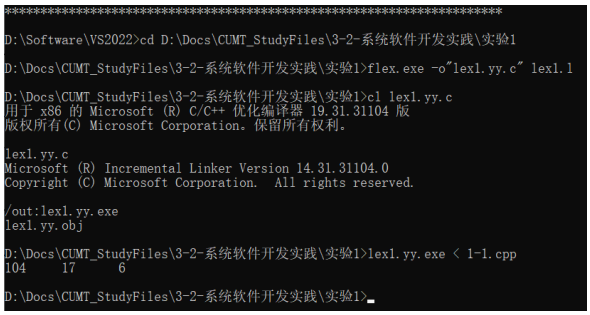

图 5 Windows 环境 lex1 运行结果

1.4.2 lex2.l

如同上述步骤，打开 Developer Command Prompt for VS 2022 命令行，运行如下代码。包括：调用 *flex.exe*，生成 *lex2.yy.c* 文件。调用 VS 的编译器 *cl.exe*，进而生成 *lex2.yy.exe* 和 *lex2.yy.obj* 文件。调用 *lex2.yy.exe*,对 *1-1.cpp* 进行词法分析。

```
cd D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\实验 1
flex.exe -o"lex2.yy.c" lex2.l
cl lex2.yy.c
lex2.yy.exe < 1-1.cpp

结果：
#!<<++! No of words:16
```

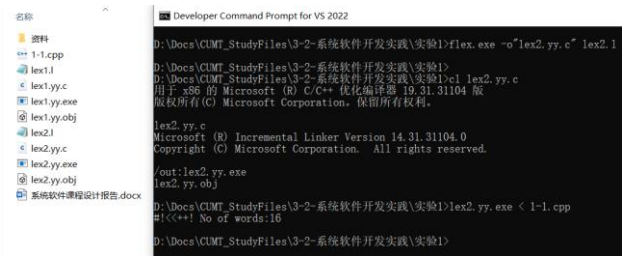

图 6 Windows 环境 lex2 运行结果

1.5 Linux 环境下使用 Flex 编译、运行源程序

1.5.1 lex1.l

类似上述步骤，在文件所在当前文件夹下打开命令行，运行如下代码。包括：调用 *flex.exe*，生成 *lex1.yy.c* 文件。调用 g++编译器，对 *lex1.yy.c* 文件进行编译，生成 *lex1.out* 可执行程序。调用 *lex1.out*,对 *1-1.cpp*（**注意：这个文件是从 Windows 直接复制的，在复制后，会发现最后一行空行丢失**，具体原因见 1.8.3）进行词法分析（**注意：需要将上述文件的 void 改成 int**，不修改的方法见 1.8.2）。

```
flex -o"lex1.yy.c" lex1.l
g++ -o"lex1.out" lex1.yy.c
./lex1.out < 1-1.cpp

结果：
110    17    6
```

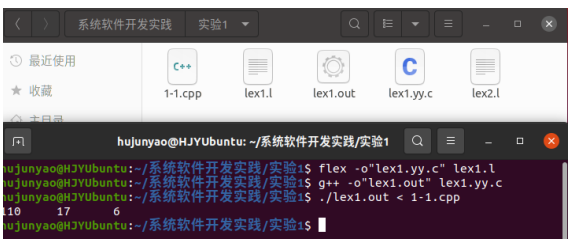

图 7 Linux 环境 lex1 运行结果

1.5.2 lex2.l

类似上述步骤，在文件所在当前文件夹下打开命令行，运行如下代码。包括：调用 *flex.exe*，生成 *lex2.yy.c* 文件。调用 g++编译器，对 *lex2.yy.c* 文件进行编译，生成 *lex2.out* 可执行程序。调用 *lex2.out,*对 *1-1.cpp* 进行词法分析（**注意：同样需要将上述文件的 void 改成 int**，不修改的方法见 1.8.2）。

```
flex -o"lex2.yy.c" lex2.l
g++ -o"lex2.out" lex2.yy.c
./lex2.out < 1-1.cpp

结果：
 No of Words:16
```

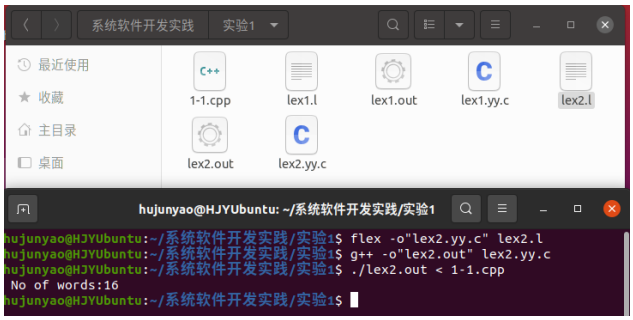

图 8 Linux 环境 lex2 运行结果

1.6 编译结果分析

1.6.1 lex1.yy.c

摘录部分代码如下。该文件在 360 行声明了字符型指针变量 yytext；在 364 行定义了 3 个全局变量 nchar, nword, nline，对应于源程序的第一部分。

```
#define REJECT reject_used_but_not_detected
#define yymore() yymore_used_but_not_detected
#define YY_MORE_ADJ 0
#define YY_RESTORE_YY_MORE_OFFSET
char *yytext;
#line 1 "lex1.l"
#define INITIAL 0
#line 2 "lex1.l"
int nchar, nword, nline;
#line 366 "lex1.yy.c"
```

在第 601-620 行的前三个 case 分别对应于 flex 源程序第二部分的三个匹配模式，最后一个 case 是如果没有识别到，就输出没有匹配到的那段信息。

```
case 1:
YY_RULE_SETUP
#line 5 "lex1.l"
{ nline++; nchar++; }
    YY_BREAK
case 2:
YY_RULE_SETUP
#line 6 "lex1.l"
{ nword++, nchar += yyleng; }
    YY_BREAK
case 3:
YY_RULE_SETUP
#line 7 "lex1.l"
{ nchar++; }
    YY_BREAK
case 4:
YY_RULE_SETUP
#line 8 "lex1.l"
ECHO;
    YY_BREAK
```

1510-1517 行，为 flex 源程序第三部分对应的 C 源程序。

```
void main()
{
yylex();
printf("%d\t%d\t%d\n", nchar, nword, nline);
}
int yywrap()
{
return 1;
}
```

1.6.2 lex2.yy.c

摘录部分代码如下。该文件在 365 行声明了字符型指针变量 yytext；在 369 行定义了 1 个全局变量 wordCount，对应于源程序的第一部分。

```
#define REJECT reject_used_but_not_detected
#define yymore() yymore_used_but_not_detected
#define YY_MORE_ADJ 0
#define YY_RESTORE_YY_MORE_OFFSET
char *yytext;
#line 1 "lex2.l"
#define INITIAL 0
```

```
#line 2 "lex2.l"
int wordCount = 0;
#line 371 "lex2.yy.c"
```

在第 606-625 行的前三个 case 分别对应于 flex 源程序第三部分的三个匹配模式，最后一个 case 是如果没有识别到，就输出没有匹配到的那段信息。

```
case 1:
YY_RULE_SETUP
#line 10 "lex2.l"
{ wordCount++; /*increase the word count by one*/ }
    YY_BREAK
case 2:
YY_RULE_SETUP
#line 11 "lex2.l"
{ /* donothing*/ }
    YY_BREAK
case 3:
YY_RULE_SETUP
#line 12 "lex2.l"
{ /* one may want to add some processing here*/ }
    YY_BREAK
case 4:
YY_RULE_SETUP
#line 13 "lex2.l"
ECHO;
    YY_BREAK
```

1.7 输出结果分析

1.7.1 lex1.l

首先基于 Windows 系统进行手工分析，结果如表格所示，与图 5 程序运行结果一致，验证正确。（**注意：对于结果出现#!<<++!等字符**，见 1.8.4）

表 1 手工验证 Windows 系统 lex1 输出结果

匹配	字符	单词	行	匹配	字符	单词	行
#include	8	1		"<<endl	7	1	
[space]	1			\n	1		1
iostream	8	1		cout	4	1	
\n	1		1	[space]	1		
using	5	1		"Welcome	8	1	
[space]	1			[space]	1		
namespace	9	1		to	2	1	
[space]	1			[space]	1		

匹配	字符	单词	行	匹配	字符	单词	行
std	3	1		c++!	4	1	
\n	1		1	[space]	1		
int	3	1		"	1	1	
[space]	1			[space]	1		
main	4	1		endl	4	1	
\n	1		1	\n	1		1
cout	4	1		return	6	1	
[space]	1			\n	1		1
"Hello!	7	1		合计	104	17	6
[space]	1						

基于 Linux 进行的手工分析可以在表 1 基础上分析，共出现 6 个\n，应该会读取到六个\r\n，对于 Windows 的结果就应该给字符数加 6，其他不变，即字符数 110，单词数 17，行数 16，该结果与图 7 程序运行结果一致，验证正确。

1.7.2 lex2.l

首先基于 Windows 系统进行手工分析，结果如表格所示，与图 6 程序运行结果一致，验证正确。这里考虑双引号如何识别，究竟是 words 还是 whitespace，应该根据文件中出现顺序先后考虑优先级，也就是先 words，后 whitespace。在进行正则匹配的时候，也要注意采取的是最大匹配。

表 2 手工验证 Windows 系统 lex2 输出结果

模式匹配	单词数	模式匹配	单词数	模式匹配	单词数
#	未匹配	\n		to	1
include	1	cout	1	[space]	
[space]		[space]"		c	1
iostream	1	Hello	1	+	未匹配
\n		!	未匹配	+	未匹配
using	1	[space]"		!	未匹配
[space]		<	未匹配	[space]"[space]	
namespace	1	<	未匹配	endl	1
[space]		endl	1	\n	
std	1	\n		return	1
\n		cout	1	\n	
int	1	[space]"		合计	16
[space]		Welcome	1		
main	1	[space]			

基于 Linux 进行的手工分析可以在基础上分析，会多出现 6 个\n，应该会读取到六个\r\n，但是这对计算 wordCount 没有任何影响，也应该是 16，该结果与图 8 程序运行结果一致，验证正确。

1.8 注意事项

1.8.1 命令行的选择

Developer PowerShell for VS 2022 不支持输入含“<”的命令。需要使用 Developer Command Prompt for VS 2022。

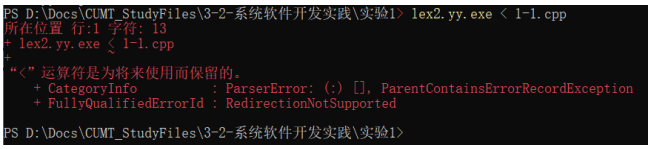

图 9 选择合适的命令行工具

1.8.2 编译软件的选择

对于 Ubuntu 系统可能提示.*l* 文件错误，不支持“void main”的语法。要么手动修改 void 为 int，要么不使用 g++，要么更换为 gcc 编译器。

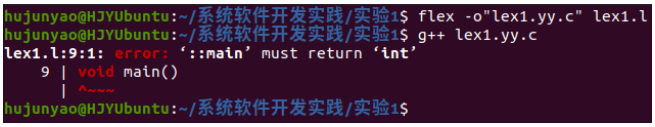

图 10 选择合适的编译器

在不修改 *1-1.cpp* 中的 void 的情况下，修改命令行如下，其他步骤相同。

```
flex -o"lex2-cc.yy.c" lex2.l
g++ -o"lex2-cc.out" lex2-cc.yy.c
./lex2-cc.out < 1-1.cpp
```

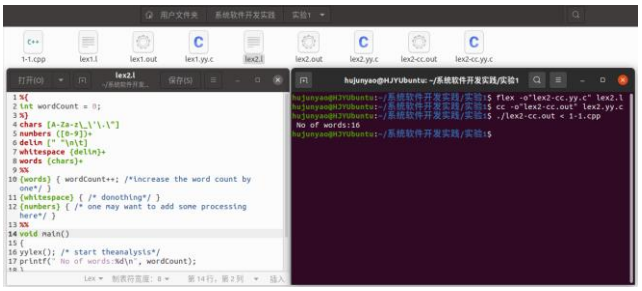

图 11 使用 gcc 编译的结果

关于 void main()

在 C 和 C++中，不接收任何参数也不返回任何信息的函数原型为“void foo(void);”。虽然在一些编译器中，void main 可以通过编译(如 vc6)，但并非所有编译器都支持 void main，因为标准中从来没有定义过 void main。g++3.2 中如果 main 函数的返回值不是 int 类型，就根本通不过编译。而 gcc3.2 则会发出警告。关于 C 语言 main 函数为什么可以没有返回值，可以认为是现代编译器、链接器和标准库“近人情”的妥协。

1.8.3 LF 与 CRLF

把 Windows 下最后一行是空行的文件（*1-1.cpp*）直接复制到 Linux 系统，显示最后一行无空行。把 Linux 下最后一行是空行的文件（*1-1.cpp（复件）*）直接复制到 Windows 系统，显示最后两行都是空行。

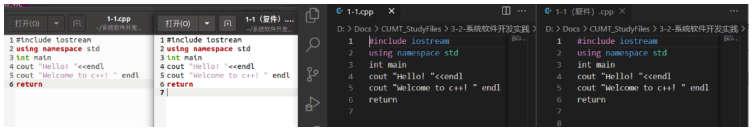

图 12 LF 与 CRLF

关于 CR，LF 与 CRLF

CR：Carriage Return，对应 ASCII 中转义字符\r，表示回车（MacIntosh）

LF：Linefeed，对应 ASCII 中转义字符\n，表示换行（Unix/Linux/Mac OS X）

CRLF：Carriage Return & Linefeed，\r\n，表示回车并换行（Windows）

1.8.4 第二部分实验的输出问题

在 Windows 中还会把不能匹配的那些字符给输出出来，如#等。如果想去掉，需要在*.l* 文件中的模式匹配部分中添加如下代码，它的作用是，匹配到除\n 外任意字符，但不做任何处理。这一步可以匹配以上别的规则匹配不了的字符，没有此匹配则 lex 会将匹配不了的字符输出到屏幕

```
. {}
```

1.9 实验感悟

本次实验，我学习了在 Windows 和 Linux 两种操作系统的环境下将*.l* 文件编译成*.yy.c* 源文件，并编译、链接成*.exe* 或者*.out* 可执行程序，使用此程序对*.cpp* 代码进行词法分析。

通过本次实验，我复习了正则表达式语法，学会了编写简单的 lex 源程序，并学会了在 Windows 和 Linux 两种操作系统环境下使用 flex 生成词法分析器，输出行号、字符数、单词数。同时在编写程序到了一些问题，查询相关网络资源也知道了不同编译器还有不同操作系统之间的微小差别，虽然很小，但对运行结果也有一定的影响，这值得深入考虑。