中国矿业大学计算机学院

系统软件开发实践报告

课程名称 系统软件开发实践

实验名称 实验七 Flex/Bison 综合实验三

学生姓名 胡钧耀

学　　号 06192081

专业班级 计算机科学与技术 2019-4 班

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（七） Flex/Bison 综合实验三

7.1 实验要求与目标

使用 flex 和 bison 开发一个具有全部功能的桌面计算器，能够支持变量，过程，循环和条件表达式，使它成为一个虽然短小但具有现实意义的编译器。

学习抽象语法树的用法，它具有强大而简单的数据结构来表示分析结果。

7.2 实验内容

阅读 *Flex/Bison.pdf* 第三章。使用 Flex 和 Bison 开发了一个具有全部功能的桌面计算器：支持变量；实现赋值功能；实现比较表达式（大于、小于等）；实现 if/then/else 和 do/while 的流程控制；用户可以自定义函数；简单的错误恢复机制。修改 *fb3-2* 的相关代码，实现自定义函数，并保存为 *fb3-3*。

请在高级计算器中尝试一些不同的语法。在前例中，函数 sq 不得不使用两个分号来分别结束 while 循环和 let 语句，这有些臃肿。你可以修改这个语法使它变得更为直观吗？如果你可以添加结束符号至条件语句 if/then/else/fi 和循环语句 while/do/done，你能够使语句列表的语法更加灵活吗？

7.3 源码修改思路

首先把所有源文件复制到新目录下，将文件名中的 *3-2* 均修改为 *3-3*。

7.3.1 *fb3-3.l*

首先需要把对应的引用头文件进行修改，修改部分已加粗。

```
%{
# include "fb3-3.h"
# include "fb3-3.tab.h"
%}
```

在 *fb3-3.l* 中增加识别{和}的匹配模式，该匹配模式与其他四则运算一样，直接返回该符号。修改部分已加粗。

```
%%/* single character ops */
"+" | "-" | "*" | "/" |……| "{" | "}"     { return yytext[0]; }
```

7.3.2 *fb3-3.y*

首先需要把对应的引用头文件进行修改，修改部分已加粗。

```
%{
#  include <stdio.h>
#  include <stdlib.h>
#  include "fb3-3.h"
int yylex();
%}
```

在实验的要求中，自定义函数部分 let 需要能够支持{……}的函数段落定义结构，同时在选择判断分支语句部分，同样需要可以支持{……}的括弧定义。

观察 *fb3-3.y* 文件中原有的语句定义，考虑对涉及的关键语句 list 部分进行如下的修改，也就是在 bison 语法的 let、if、while 规则相应位置添加花括号。

let 语句修改如下，修改部分已加粗。

```
calclist: /* nothing */
  ……
  | calclist LET NAME '(' symlist ')' '{' list '}' EOL {
                       dodef($3, $5, $8);
                       printf("Defined %s\n> ", $3->name); }
  ……
 ;
%%
```

if 和 while 语句修改如下，修改部分已加粗。由于 if/then 语句和 if/then/else 语句 then、else 后面的语句块既可以加花括号，也可以不加，因此复制一份，两种模式均可以识别。

```
stmt: IF exp THEN list           { $$ = newflow('I', $2, $4,
NULL); }
   | IF exp THEN list ELSE list  { $$ = newflow('I', $2, $4,
$6); }
   | WHILE exp DO list           { $$ = newflow('W', $2, $4,
NULL); }
   | IF exp THEN '{' list '}'
{ $$ = newflow('I', $2, $5, NULL); }
   | IF exp THEN '{' list '}' ELSE '{' list '}'
{ $$ = newflow('I', $2, $5, $9); }
   | WHILE exp DO '{' list '}'
{ $$ = newflow('W', $2, $5, NULL); }
   | exp
;
```

由于加了花括号的语句可以不加分号，因此修改 list 语法，修改部分已加粗。

```
list: /* nothing */ { $$ = NULL; }
   | ……
   | stmt {$$ = $1; }
   ;
```

在每个可计算的表达式后面都要加分号，例如 sqrt(10);，因此修改 calclist,

修改部分已加粗。

```
calclist: /* nothing */
  | calclist stmt ';' EOL {……}
  |……
 ;
%%
```

运行如下代码查看指定的规则是否有移进规约冲突，发现有冲突，于是保存冲突的地方并进行修改。

```
bison -d fb3-3.y
bison -v fb3-3.y
```

```
state 49

    9 list: stmt . ';' list
   10     | stmt .

    ';'  shift, and go to state 57

    ';'       [reduce using rule 10 (list)]
    $default  reduce using rule 10 (list)

state 50

    1 stmt: IF exp THEN list .
    2     | IF exp THEN list . ELSE list

    ELSE  shift, and go to state 58

    ELSE      [reduce using rule 1 (stmt)]
    $default  reduce using rule 1 (stmt)

state 62

    4 stmt: IF exp THEN '{' list '}' .
    5     | IF exp THEN '{' list '}' . ELSE '{' list '}'

    ELSE  shift, and go to state 68

    ELSE      [reduce using rule 4 (stmt)]
    $default  reduce using rule 4 (stmt)
```

图 1 三个移进规约冲突结果

根据之前 bison 实验二的研究，对于移进规约冲突的解决办法通常是指定优先级，`%nonassoc` 意味着没有依赖关系，经常在连接词中和`%prec` 一起使用，用于指定一个规则的优先级。

```
%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE
……
A %prec LOWER_THAN_ELSE ……
B ELSE ……
```

上面的修改方案，使得 LOWER_THAN_ELSE 的优先级小于 ElSE，同时语法第一句的优先级被指定为了 LOWER_THAN_ELSE，这样当冲突发生时，编器将先移进，后规约。

根据这种方法，可以制定以下修改策略。

第一个冲突是分号的移进规约冲突。只要规定一个比分号优先级低的即可。修改部分已加粗。

```
%nonassoc LOWER_THAN_SEMICOLON
%nonassoc ';'
```

```
list: /* nothing */ { $$ = NULL; }
   | stmt ';' list { if ($3 == NULL)
                   $$ = $1;
                      else
         $$ = newast('L', $1, $3);
                    }
   | stmt %prec LOWER_THAN_SEMICOLON {$$ = $1; }
   ;
```

第二个和第三个是 ELSE 的移进规约冲突。只要规定一个比 ELSE 优先级低的即可。修改部分已加粗。

```
%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE

stmt: IF exp THEN list %prec LOWER_THAN_ELSE
{ $$ = newflow('I', $2, $4, NULL); }
   | IF exp THEN list ELSE list
{ $$ = newflow('I', $2, $4, $6); }
   | WHILE exp DO list
{ $$ = newflow('W', $2, $4, NULL); }
   | IF exp THEN '{' list '}' %prec LOWER_THAN_ELSE
{ $$ = newflow('I', $2, $5, NULL); }
   | IF exp THEN '{' list '}' ELSE '{' list '}'
{ $$ = newflow('I', $2, $5, $9); }
   | WHILE exp DO '{' list '}'
{ $$ = newflow('W', $2, $5, NULL); }
   | exp
;
```

再次编译，发现语法已经没有移进规约冲突了，程序可以正常运行。

```
PS D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验3> bison -d fb3-3.y
fb3-3.y: conflicts: 3 shift/reduce
PS D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验3> bison -v fb3-3.y
fb3-3.y: conflicts: 3 shift/reduce
PS D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验3> bison -d fb3-3.y
PS D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验3> bison -v fb3-3.y
PS D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验3>
```

图 2 移进规约冲突已消除

7.3.3 *fb3-3funcs.c*

修改引用头文件，修改部分已加粗。

```
#  include "fb3-3.h"
```

7.4 不同环境运行结果

7.4.1 Windows

执行如下代码，可生成计算器 *cal3-3.exe*。调用该计算器，验证结果如下，与真实情况一致，表明计算器无问题。

```
bison -d fb3-3.y
flex -ofb3-3.lex.c fb3-3.l
gcc -o cal3-3.exe fb3-3.tab.c fb3-3.lex.c fb3-3funcs.c -lm
```

```
D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验3>bison -d fb3-3.y

D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验3>flex -ofb3-3.lex.c fb3-3.l

D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验3>gcc -o cal3-3.exe fb3-3.tab.c fb3-
3.lex.c fb3-3funcs.c -lm

D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验3>cal3-3.exe
> let avg(a,b) { (a+b)/2; }
Defined avg
> let sq(n) { e=1; while (|((t=n/e)-e)>.001) do { e=avg(e,t); } }
Defined sq
> sq(10);
= 3.162
> sq(10)-sqrt(10);
= 0.000178
> let max(a,b) { if(a>b) then a; else b; }
Defined max
> max(4,5);
=   5
> max(1,5);
=   5
> max(9,5);
=   9
> let max3(a,b,c) { if(a>b) then { if(a>c) then a; else c; } else { if(b>c) then b; else c; } }
Defined max3
> max3(3,4,5);
=   5
> max3(3,9,5);
=   9
> max3(10,9,5);
=  10
>
```

图 3 Windows 环境生成计算器 *3-3.exe* 及其计算结果

7.4.2 Linux

直接把在 Windows 修改好的四个源文件复制到 Linux 系统中，输入如下代码，并进行调用，验证结果正确。

```
bison -d fb3-3.y
flex -ofb3-3.lex.c fb3-3.l
gcc -o cal3-3.out fb3-3.tab.c fb3-3.lex.c fb3-3funcs.c -lm
```

```
hujunyao@HJYUbuntu:~/系统软件开发实践/实验7$ bison -d fb3-3.y
hujunyao@HJYUbuntu:~/系统软件开发实践/实验7$ flex -ofb3-3.lex.c fb3-3.l
hujunyao@HJYUbuntu:~/系统软件开发实践/实验7$ gcc -o cal3-3.out fb3-3.ta
b.c fb3-3.lex.c fb3-3funcs.c -lm
hujunyao@HJYUbuntu:~/系统软件开发实践/实验7$ ./cal3-3.out
> let avg(a,b) { (a+b)/2; }
Defined avg
> let sq(n) { e=1; while (|((t=n/e)-e)>.001) do { e=avg(e,t); } }
Defined sq
> sq(10);
= 3.162
> sq(10)-sqrt(10);
= 0.000178
> let max(a,b) { if(a>b) then a; else b; }
Defined max
> max(4,5);
=    5
> max(1,5);
=    5
> max(9,5);
=    9
> let max3(a,b,c) { if(a>b) then { if(a>c) then a; else c; } else { if(
b>c) then b; else c; } }
Defined max3
> max3(3,4,5);
=    5
> max3(3,9,5);
=    9
> max3(10,9,5);
=   10
>
```

图 4 Linux 生成计算器 *3-3.out* 及运算结果

7.5 构建抽象语法树

7.5.1 avg(3,5)构建过程

当我们向计算器输入 avg(3,5)时，先是用词法分析器对其进行分析，判断出是内置函数或自定义函数，返回相应的 token 告诉语法分析器。

语法分析器收到词法分析结果，建立相应的语法树节点，并且设置好节点的参数。在 eval 函数计算结果时，根据节点的类型，调用对应的函数代码，计算出正确的结果。

以下结合代码进行详细的分析。

首先将计算后的结果在内建函数里面查找，发现没有，则是自定义函数。计算器由这条规则来处理 NAME 标识的用户自定义函数。

```
Exp：  NAME '(' explist ')' { $$ = newcall($1, $3); }
```

其中的 explist 定义了一个表达式列表，它用于创建抽象语法树来作为函数调用所需要的实参。

```
explist: exp
```

symlist 定义了一个符号列表，创建了这些符号的链表用于函数定义时所带的形参。建立语法树节点的两个函数 newfunc、newast。

```
symlist: NAME { $$ = newsymlist($1, NULL); }
```

本次实验制作的计算器的重点是利用抽象语法树实现了流程的控制和用户自定义函数的功能。

关于用户自定义函数：用户自定义的函数包括函数名、形参列表、函数体

当自定义函数被调用时：计算实参的值、保存形参的值，将实参赋值给形参、执行函数体，构建抽象语法树、将形参恢复原来的值、返回函数的返回值。

7.5.2 avg(3,5)构建结果

构建示例如下所示。

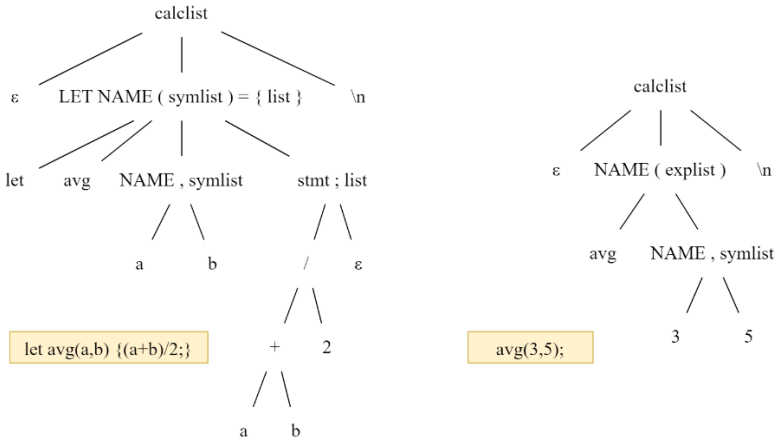

图 5 构建本实验部分抽象语法树

7.6 实验总结

7.6.1 问题

本次实验在修改源代码时，遇到了移入规约冲突，联想到之前实验中遇到的问题，查看实验报告文档就知道了应该如何去修改代码，使用 bison 编译时可以使用`-v` 参数，生成*.output* 文件以查看冲突产生原因，并通过定义优先级解决冲突。

7.6.2 评价

本次实验总体来说比较简单，可以通过对于所需要文法的改进支持，增加了括号匹配，改进了分号的使用。同时，在面对移进规约冲突时，通过观察状态机描述文件，联想到之前的文档已经解决过类似的问题，最后举一反三，解决了这个问题，进一步增强了用 flex 与 bison 设计文法并工作的能力，同时很好的回顾了 bison 实验二的内容。

7.6.3 收获

通过对函数调用是传来的实参列表对应的抽象语法树深入分析，我了解了实参列表的结构以及计算参数值的方法，因此可以找到每个参数并计算其值。本次实验我也更加理解了 flex 和 bison 的工作原理，明白了如何有效解决移入规约冲突。在语法分析器编程上，还有很多东西要实践。也要多写报告，多写博客，记录下自己遇到的问题，这样在以后遇到或者同学同事遇到相同的问题的时候，就能很快找到解决的办法，事半功倍，提高自己编码的效率。