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

报告时间 2022.3.15

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	

目录

三、 综合实验 3

1 实验目的

阅读《flex&Bison》第三章。使用 flex 和 Bison 开发一个具有全部功能的计算器，包括如下功能：

1) 支持变量；
2) 实现复制功能；
3) 实现比较表达式（大于小于等）；
4) 实现 if/then/else 和 do/while 流程控制；
5) 用户可以自定义函数；
6) 简单的错误恢复机制。

2 实验内容

10)阅读《Flex&Bison》第三章 P79 习题 1，学习抽象语法树

11)修改 fb3-2 的相关代码，实现特定函数，并保存为 fb3-3

12)撰写实验报告，结合实验结果，给出抽象语法树的构建过程

3 实验步骤

本次实验的重点与前两次综合实验略有不同，前两次实验考察词法、语法分析过程并给出源代码；而本次实验则侧重于编译前端的最后一部分——语法制导翻译。

实验中，我们需要修改特定文法或是语法制导翻译模式，使计算器支持特定的语法。

3.1 文法修改

实验要求我们修改计算器自定义函数的语句 let，使之支持**大括号**作为函数体作用范围的区域；同时在 if/else 等选择判断分支语句处，也要支持**大括号**的使用。

首先观察 let 的语法分析段，如下：

```
1. calclist: /* nothing */
2.   | calclist stmt EOL {
3.     if(debug) dumpast($2, 0);
4.      printf("= %4.4g\n> ", eval($2));
5.      treefree($2);
6.     }
7.   | calclist LET NAME '(' symlist ')' '=' list EOL {
```

```
8.                      dodef($3, $5, $8);
9.                      printf("Defined %s\n> ", $3->name); }
10.
11.   | calclist error EOL { yyerrok; printf("> "); }
12.  ;
```

行 7 给出了 let 的产生式。为了使 let 后支持大括号，我们去掉行 7 中的'='，并修改 dodef($3, $5, $7)。接下来对文法的修改思路就很清晰了——修改非终结符list，使之支持大括号结构。

参考资料中的《ANSI C grammar (Yacc).pdf》，我们对 list 产生式修改为如下结构：

```
1. list: '{' list '}' {$$ = $2;}
2.     | '{' list stmt '}' {$$ = newast('L', $2, $3);}
3.     | stmt {$$ = $1;}
4.     | exp {$$ = $1;}
```

最后，由于计算器本身不支持{}这样的终结符，我们在词法分析中，增加左右大括号的词法匹配模式：

```
1. "{" |
2. "}" |
3. { return yytext[0]; }
```

至此，我们的计算器在文法上已经支持大括号结构了。编译结果如下：

```
D:\Here\recently\系统软件开发实践\综合实验\3-3\test>bison -d fb3-2.y
fb3-2.y: conflicts: 3 shift/reduce
```

图 1 第一次编译结果

经过我们修改的文法，存在 3 个移进/归约冲突。我们使用 bison -v 命令，生成日志文件，查看具体的移进/归约冲突发生的代码段。

3.2 移进/归约冲突的解决

在使用 bison -v 指令后，得到 fb3-2.output 日志文件。从中可以看出，识别活前缀的 DFA 在状态 5、50、51 发成了移进/归约冲突。

```
State 5 conflicts: 1 shift/reduce
State 50 conflicts: 1 shift/reduce
State 51 conflicts: 1 shift/reduce
```

图 2 移进/归约冲突的产生位置

♦ 状态 5 冲突的解决

日志文件中，对状态 5 的描述如下。

```
state 5

    19 exp: NAME .
    20    | NAME . '=' exp
    21    | NAME . '(' explist ')'

     '='  shift, and go to state 16
     '('  shift, and go to state 17

     '('       [reduce using rule 19 (exp)]
     $default  reduce using rule 19 (exp)
```

当自底向上的语法分析归约出文法符号 NAME 时，如果输入串的下一个符号是'('，此时识别活前缀的 DFA 既可以移进'('转为状态 17，又可以使用产生式$exp \rightarrow NAME$归约。即移进/归约冲突。

我们将 NAME 的优先级降低，优先移进归约左括号后的表达式，从而解决冲突。

在 Yacc 文件的声明区，加入%nonassoc LOWER，并将左括号的改为左结合性%left '('后，将产生式$exp \rightarrow NAME$更改为$exp \rightarrow NAME$ %prec LOWER，即可解决冲突。

♦ 状态 50 冲突的解决

日志文件中，对状态 50 的描述如下：

```
state 50

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

    ELSE  shift, and go to state 57

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

当符号栈中已经存在了归约出的 IF exp THEN list，且输入串的下一位是 ELSE 时，此时识 DFA 既可以移进 ELSE 转为状态 57，又可以使用产生式$stmt \rightarrow IF\ exp\ THEN\ list$归约。即移进/归约冲突。

根据 ELSE 与最近的 IF 匹配的原则，我们选择降低归约操作的优先级，优先移进操

作即可解决冲突。

我们将上述产生式修改为$stmt \rightarrow IF\ exp\ THEN\ list$ %prec LOWER，并为符号 ELSE 加入左结合性，冲突解决。

♦ 状态 51 冲突的解决

日志文件中，对状态 50 的描述如下：

```
1. state 51
2.
3.     4 stmt: exp . ';'
4.     8 list: exp .
5.     9 exp: exp . CMP exp
6.     10    | exp . '+' exp
7.     11    | exp . '-' exp
8.     12    | exp . '*' exp
9.     13    | exp . '/' exp
10.
11.    CMP  shift, and go to state 26
12.    '+'  shift, and go to state 27
13.    '-'  shift, and go to state 28
14.    '*'  shift, and go to state 29
15.    '/'  shift, and go to state 30
16.    ';'  shift, and go to state 31
17.
18.    '-'      [reduce using rule 8 (list)]
19.    $default  reduce using rule 8 (list)
```

当符号栈中已经存在了归约出的 exp，且输入串的下一位是'-'时，此时识 DFA 既可以移进'-'转为状态 28，又可以使用产生式$list \rightarrow exp$归约。即移进/归约冲突。

解决冲突的办法仍然是降低归约的优先级，为'-'增加左结合性即可。

4 实验结果

4.1 Windows 系统下的实验结果

```
D:\Here\recently\系统软件开发实践\综合实验\3-3\code>fb3-3
> 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 系统下实验结果

4.2 Linux 系统下的实验结果

```
chen@chen:~/桌面/chen/e7$ ./fb3-3
> 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);
8: error: syntax error
> 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 系统下实验结果

5 实验总结

本次实验总体上较为简单，根据编译原理课程上所学的语法制导翻译相关内容，以及参考书《ANSI C grammar (Yacc).pdf》，可以比较轻松的完成文法的修改；通过 Yacc 的冲突发生时的.output 日志文件，可以快速定位移进/归约冲突的位置，并给出合理的解决方案。通过本次实验，对 Flex 与 Bison 的使用更加熟练，加深了对计算器源码的理解。受益匪浅。