



中国矿业大学
CHINA UNIVERSITY OF MINING AND TECHNOLOGY

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 2020.04.27

学 号

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	实验内容	1
2	实验5-1	1
2.1	实验目标	1
2.2	配置与修改代码	1
2.2.1	Windows	1
2.2.2	Ubuntu	2
2.3	分析源代码	3
2.3.1	fb3-1.y	3
2.3.2	fb3-1.l	4
2.3.3	fb3-1funcs.c	5
2.3.4	fb3-1.h	7
2.4	实验过程	7
2.4.1	Windows	7
2.4.2	Ubuntu	8
2.4.3	抽象语法树构建	9
2.4.4	移进规约过程	11
2.5	实验感想	12
3	实验5-2	12
3.1	实验目标	12
3.2	配置与修改代码	12
3.3	分析源代码	12
3.3.1	fb3-2.y	12
3.3.2	fb3-2.l	14
3.3.3	fb3-2funcs.c	15
3.3.4	fb3-2.h	16
3.4	实验过程	17
3.4.1	Windows	17
3.4.2	Ubuntu	19
3.4.3	抽象语法树构建	19
3.5	实验感想	19
4	实验5-3	20
4.1	实验目标	20
4.2	修改思路	20
4.3	实验过程	24
4.3.1	Windows	24
4.3.2	Ubuntu	25
4.3.3	抽象语法树构建	26
4.4	实验感想	26

5	实验5-4	26
5.1	实验目标	26
5.2	配置与修改代码	26
5.2.1	开发环境	26
5.2.2	基本思路	27
5.2.3	代码修改	27
5.3	实验过程	28
5.3.1	Windows	28
5.3.2	Ubuntu	29
5.4	实验演示	30
5.5	扩展功能	32
5.6	实验感想	34

实验五 Flex&Bison高级计算器

1 实验内容

阅读《Flex/Bison.pdf》，使用Flex和Bison开发了具有全部功能的桌面计算器。

- a) 支持变量；
- b) 实现赋值功能；
- c) 实现比较表达式（大于、小于等）；
- d) 实现if/then/else和do/while的流程控制；
- e) 用户可以自定义函数；
- f) 简单的错误恢复机制。

2 实验5-1

2.1 实验目标

- 1) 阅读《Flex&Bison》第三章 P47 P60，学习抽象语法树；
- 2) 阅读fb3-1.y、fb3-1.l、fb3-1funcs.c、fb3-1.h；
- 3) 撰写实验报告，结合实验结果，给出移近/规约过程，及抽象语法树的构建过程，如 $(1 + 2) - (2 * 6)$ 、 $1 + 2 - 3 * 2/5$ 。
- 4) 提交报告和实验代码。

2.2 配置与修改代码

2.2.1 Windows

atof()函数会扫描参数nptr字符串，跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，而再遇到非数字或字符串结束时(' \0')才结束转换，并将结果返回，str字符串可包含正负号、小数点或E(e)来表示指数部分。

```
fb3-1.l(27): warning C4477: "printf": 格式字符串 "%f" 需要类型 "double" 的参数，但可变参数
3 拥有了类型 "int"
fb3-1.tab.c
fb3-1funcs.c
正在生成代码...
```

图 2.1: Windows测试yylext输出

但是这里出现了问题，在实验过程中，总是出现返回转换为0的情况，为了检测此问题，在此语句附近加上`printf("yytext: \t%f\n", atof(yytext));`；

来测试输出结果，如图2.1所示，发现报：格式字符串“%f”需要类型“double”的参数，但可变参数 1 拥有了类型“int”的错误。

由此，我认为是在Windows环境下，使用bison编译，atof()函数会被定义为返回int的函数，所以，在fb3-1.1文件C语言声明部分对 atof()函数的返回值进行声明为float，如图2.2所示。

```
# include "fb3-1.h"
# include "fb3-1.tab.h"
double atof (const char* str);
```

图 2.2: Windows修改代码方式1

sscanf()函数作用是从字符串读取格式化输入，这里可以通过此函数将 yytext 中的内容以长浮点型读取到yylval.d中。

修改fb3-1.1代码中的第27行，将yylval.d=atof(yytext);修改为sscanf(yytext,"%lf",&yylval.d);，如图2.3所示。

```
"."?[0-9]+[EXP]? { yylval.d = atof(yytext); return NUMBER; }
改为
"."?[0-9]+[EXP]? { sscanf(yytext,"%lf",&yylval.d); return NUMBER; }
```

图 2.3: Windows修改代码方式2

注：本次Windows环境实验使用方式1对代码进行修改。

2.2.2 Ubuntu

在 Ubuntu 环境下进行实验，发现存在部分警告，如图2.4所示，原因是使用了隐式定义的yylex()与yyparse()函数。

```
yuan@ubuntu:~/桌面/System Software/ex5$ bison -d fb3-1.y
yuan@ubuntu:~/桌面/System Software/ex5$ flex fb3-1.l
yuan@ubuntu:~/桌面/System Software/ex5$ cc lex.yy.c fb3-1.tab.c fb3-1funcs.c -lm
fb3-1.tab.c: In function 'yyparse':
fb3-1.tab.c:1137:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
    yychar = yylex ();
fb3-1funcs.c: In function 'main':
fb3-1funcs.c:104:10: warning: implicit declaration of function 'yyparse' [-Wimplicit-function-declaration]
    return yyparse();
```

图 2.4: Ubuntu环境警告

修改fb3-1.y文件，在第14行C语言声明部分加入extern int yylex();代码，如图2.5所示。

修改fb3-1funcs.c文件，在第14行C语言声明部分加入extern int yyparse();代码，如图2.6所示。

```

/* calculator with AST */
%{
# include <stdio.h>
# include <stdlib.h>
# include "fb3-1.h"
extern int yylex();
%}

```

图 2.5: 修改fb3-1.y

```

/*
 * helper functions for fb3-1
 */
# include <stdio.h>
# include <stdlib.h>
# include <stdarg.h>
# include "fb3-1.h"
extern int yyparse();

```

图 2.6: 修改fb3-1funcs.c

2.3 分析源代码

2.3.1 fb3-1.y

```

1  %{
2  # include <stdio.h>
3  # include <stdlib.h>
4  # include "fb3-1.h"
5  %} // C代码声明部分。
6
7  %union {
8      struct ast *a;
9      double d;
10 } // 定义一个结构体，包含一个抽象语法树结构与一个浮点型数。
11
12 /* declare tokens */
13 %token <d> NUMBER // 定义NUMBER为浮点型
14 %token EOL // 定义一行结束
15
16 %type <a> exp factor term // 定义此表达式为抽象语法树结构
17
18 %%
19 // 以下为语义规则
20 calclist: /* nothing */
21 | calclist exp EOL {
22     printf("= %4.4g\n", eval($2)); // 以实数类型输出$2
23     treefree($2); // 从内存中释放节点
24     printf("> ");
25 }
26
27 | calclist EOL { printf("> "); } /* blank line or a comment */
28 ;
29
30 exp: factor // 表达式为因子，或者表达式加因子，或者表达式减因子

```

```

31 | exp '+' factor { $$ = newast('+', $1,$3); }
32 | exp '-' factor { $$ = newast('-', $1,$3);}
33 ;
34
35 factor: term // 因子是term, 或者因子乘term, 或者因子除term
36 | factor '*' term { $$ = newast('*', $1,$3); }
37 | factor '/' term { $$ = newast('/', $1,$3); }
38 ;
39
40 // term对应一个数字, 或者取绝对值, 或者括号和表达式, 或者取负
41 term: NUMBER { $$ = newnum($1); }
42 | '|' term { $$ = newast('|', $2, NULL); }
43 | '(' exp ')' { $$ = $2; }
44 | '-' term { $$ = newast('M', $2, NULL); }
45 ;
46 %%

```

2.3.2 fb3-1.1

```

1 %option noyywrap nodefault yylineno
2 %{
3 # include "fb3-1.h"
4 # include "fb3-1.tab.h"
5 double atof(const char* str);
6 %} // 声明部分
7
8 /* float exponent */
9 EXP      ([Ee] [-+]?[0-9]+) // 匹配科学计数法结尾
10
11 %%
12 "+" |
13 "-" |
14 "*" |
15 "/" |
16 "|" |
17 "(" |
18 ")" { return yytext[0]; } // 匹配以上字符输出
19 [0-9]+ "." [0-9]* {EXP}? | // 匹配浮点型实数, 赋值给yylval.d
20 "."? [0-9]+ {EXP}? { yyval.d = atof(yytext); return NUMBER; }
21
22 \n { return EOL; } \\ 匹配换行符, 输出EOL
23 "//".*
24 [ \t] { /* ignore white space */ } \\ 匹配空格和换行符, 不做任何操作

```



```
25 .      { yyerror("Mystery character %c\n", *yytext); }  \\ 错误信息输出
26 %%
```

2.3.3 fb3-1funcs.c

```
1  # include <stdio.h>
2  # include <stdlib.h>
3  # include <stdarg.h>
4  # include "fb3-1.h"
5
6  // 声明一个结构体，两个操作数
7  struct ast *
8  newast(int nodetype, struct ast *l, struct ast *r){
9      struct ast *a = malloc(sizeof(struct ast));
10
11      if(!a) {
12          yyerror("out of space");
13          exit(0);
14      }
15      a->nodetype = nodetype;
16      a->l = l;
17      a->r = r;
18      return a;
19  }
20
21  // 声明一个结构体，一个操作数
22  struct ast *
23  newnum(double d){
24      struct numval *a = malloc(sizeof(struct numval));
25
26      if(!a) {
27          yyerror("out of space");
28          exit(0);
29      }
30      a->nodetype = 'K';
31      a->number = d;
32      return (struct ast *)a;
33  }
34
35  double eval(struct ast *a){
36      double v;
37
38      switch(a->nodetype) { // 判断结点类型，做相应的计算
```

```
39         case 'K': v = ((struct numval *)a)->number; break;
40
41         case '+': v = eval(a->l) + eval(a->r); break;
42         case '-': v = eval(a->l) - eval(a->r); break;
43         case '*': v = eval(a->l) * eval(a->r); break;
44         case '/': v = eval(a->l) / eval(a->r); break;
45         case '|': v = eval(a->l); if(v < 0) v = -v; break;
46         case 'M': v = -eval(a->l); break;
47         default: printf("internal error: bad node %c\n", a->nodetype);
48     }
49     return v;
50 }
51
52 void treefree(struct ast *a){
53     switch(a->nodetype) {
54
55         /* 两个子树, 释放右支 */
56         case '+':
57         case '-':
58         case '*':
59         case '/':
60             treefree(a->r);
61
62         /* 一个子树, 释放左支 */
63         case '|':
64         case 'M':
65             treefree(a->l);
66
67         /* 没有子树, 释放根节点 */
68         case 'K':
69             free(a);
70             break;
71
72         default: printf("internal error: free bad node %c\n", a->nodetype);
73     }
74 }
75
76 void yyerror(char *s, ...){ // 输出错误信息
77     va_list ap;
78     va_start(ap, s);
79
80     fprintf(stderr, "%d: error: ", yylineno);
```

```
81     fprintf(stderr, s, ap);
82     fprintf(stderr, "\n");
83 }
84
85 int main(){
86     printf("> ");
87     return yyparse();
88 }
```

2.3.4 fb3-1.h

```
1 // 与词法分析器的接口
2 extern int yylineno; /* from lexer */
3 void yyerror(char *s, ...);
4
5 // 抽象语法树中的节点
6 /* nodes in the Abstract Syntax Tree */
7 struct ast {
8     int nodetype;
9     struct ast *l;
10    struct ast *r;
11 };
12
13 struct numval {
14     int nodetype; /* type K */
15     double number;
16 };
17
18 // 建立一个抽象语法树
19 struct ast *newast(int nodetype, struct ast *l, struct ast *r);
20 struct ast *newnum(double d);
21
22 // 计算抽象语法树
23 double eval(struct ast *);
24
25 // 删除与释放抽象语法树
26 void treefree(struct ast *);
```

2.4 实验过程

2.4.1 Windows

将实验代码放在实验目录ex5-1下，打开Developer Command Prompt for VS

2019, 实验命令同前几次实验, 此处不再赘述。命令如下:

```
bison -d fb3-1.y
flex fb3-1.1
cl lex.yy.c fb3-1.tab.c fb3-1funcs.c
```

输入上述命令后, 结果如图2.7所示。



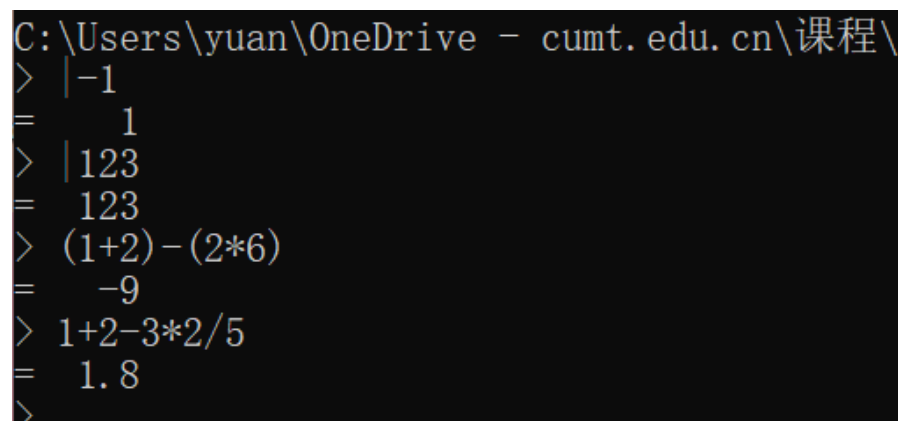
```
C:\Users\yuan\OneDrive - cumt.edu.cn\课程\2019-2020-2 系统软件开发实践
fb3-1funcs.c
用于 x86 的 Microsoft (R) C/C++ 优化编译器 19.24.28316 版
版权所有 (C) Microsoft Corporation. 保留所有权利。

lex.yy.c
fb3-1.tab.c
fb3-1funcs.c
正在生成代码...
Microsoft (R) Incremental Linker Version 14.24.28316.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:lex.yy.exe
lex.yy.obj
fb3-1.tab.obj
fb3-1funcs.obj
```

图 2.7: Windows编译结果

然后输入`lex.yy.exe`, 进行实验测试, 依次输入`| -1`、`| 123`、`(1+2)-(2*6)`、`1 + 2 - 3 * 2 / 5`, 结果如图2.8所示。



```
C:\Users\yuan\OneDrive - cumt.edu.cn\课程\
> | -1
= 1
> | 123
= 123
> (1+2)-(2*6)
= -9
> 1+2-3*2/5
= 1.8
>
```

图 2.8: Windows实验结果

2.4.2 Ubuntu

将实验代码放在实验目录`ex5-1`下, 打开终端, 输入如下命令:

```
bison -d fb3-1.y
flex fb3-1.1
cc lex.yy.c fb3-1.tab.c fb3-1funcs.c
```

输入上述命令后, 结果如图2.9所示。

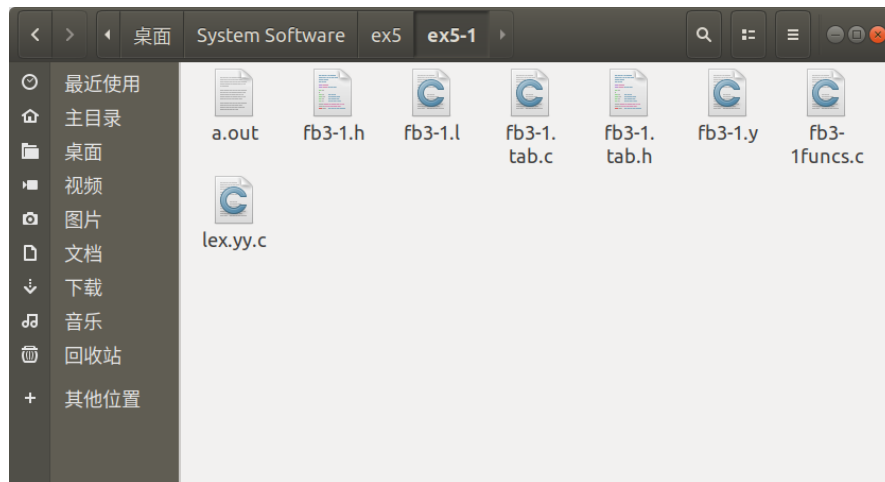


图 2.9: Ubuntu编译结果

然后输入`./a.out`，进行实验测试，依次输入 $| - 1$ 、 $| 123$ 、 $(1 + 2) - (2 * 6)$ 、 $1 + 2 - 3 * 2 / 5$ ，结果如图2.8所示。

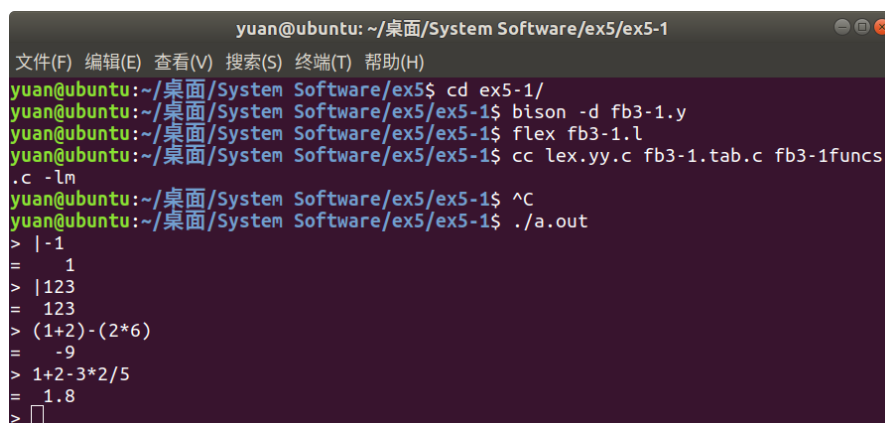


图 2.10: Ubuntu实验结果

2.4.3 抽象语法树构建

$(1 + 2) - (2 * 6)$ 抽象语法树如下图所示：

2.4.4 移进规约过程

以 $(1 + 2) - (2 * 6)$ 为例分析移进规约过程

栈	输入	动作
#	$(1+2)-(2*6)\#$	移进
#($1+2)-(2*6)\#$	移进
#(1	$+2)-(2*6)\#$	规约 $term \rightarrow NUMBER$
#(term	$+2)-(2*6)\#$	规约 $factor \rightarrow term$
#(factor	$+2)-(2*6)\#$	规约 $exp \rightarrow factor$
#(exp	$+2)-(2*6)\#$	移进
#(exp+	$2)-(2*6)\#$	移进
#(exp+2	$)-(2*6)\#$	规约 $term \rightarrow NUMBER$
#(exp+term	$)-(2*6)\#$	规约 $factor \rightarrow term$
#(exp+factor	$)-(2*6)\#$	规约 $exp \rightarrow exp + factor$
#(exp	$)-(2*6)\#$	移进
#(exp)	$-(2*6)\#$	规约 $term \rightarrow (exp)$
#term	$-(2*6)\#$	规约 $factor \rightarrow term$
#factor	$-(2*6)\#$	规约 $exp \rightarrow factor$
#exp	$-(2*6)\#$	移进
#exp-	$(2*6)\#$	移进
#exp-($2*6)\#$	移进
#exp-(2	$*6)\#$	规约 $term \rightarrow NUMBER$
#exp-(term	$*6)\#$	规约 $factor \rightarrow term$
#exp-(factor	$*6)\#$	移进
#exp-(factor*	$6)\#$	移进
#exp-(factor*6	$)\#$	规约 $term \rightarrow NUMBER$
#exp-(factor*term	$)\#$	规约 $factor \rightarrow factor * term$
#exp-(factor	$)\#$	规约 $exp \rightarrow factor$
#exp-(exp	$)\#$	移进
#exp-(exp)	$\#$	规约 $term \rightarrow (exp)$
#exp-term	$\#$	规约 $factor \rightarrow term$
#exp-factor	$\#$	规约 $exp \rightarrow exp - factor$
#exp	$\#$	接受

2.5 实验感想

这次实验是一个简单的计算器，可以完成加减乘除等基本计算，通过分析代码，逐渐掌握此计算器的原理。

在Windows环境下进行实验的过程中，总是出现返回转换为0的情况，经过查阅资料与本地的调试，发现了问题，atof()函数会被定义为返回int的函数，所以，在声明部分对atof()函数的返回值进行声明为float，就可以解决此问题。还有一种方法是用sscanf()函数，这个可以从字符串读取格式化输入，也能解决问题。

在Ubuntu环境下的实验只有警告，在声明部分显示定义函数的参数和返回值类型就可以解决。

3 实验5-2

3.1 实验目标

- 1) 阅读《Flex&Bison》第三章 P60-P79，学习抽象语法树；
- 2) 阅读fb3-2.y、fb3-2.l、fb3-2funcs.c、fb3-2.h；
- 3) 使用内置函数 sqrt(n)、exp(n)，log(n)；
- 4) 定义函数sq(n)、avg(a,b)，用于计算平方根；
- 5) 撰写实验报告，结合实验结果，给出抽象语法树的构建过程。

3.2 配置与修改代码

本次实验Windows环境与Ubuntu环境所出现的问题同实验5-1，修改方法一样，但是在编译的过程中会出现math相关的函数未定义的引用问题，查阅资料，发现在Ubuntu下需要指定函数库的位置，在编译时加上-lm命令即可。

3.3 分析源代码

3.3.1 fb3-2.y

fb3-2.y相对于fb3-1.y有以下修改

```
1 %union {
2     struct ast *a;
3     double d;
4     struct symbol *s;          /* 哪一个符号 */
5     struct symlist *sl;
6     int fn;                   /* 哪一个方法 */
```



```

7  }
8
9  %token <s> NAME // 定义NAME为符号结构
10 %token <fn> FUNC // 定义FUNC为方法
11
12 %token IF THEN ELSE WHILE DO LET // 定义关键字的标识
13
14 %nonassoc <fn> CMP
15 %right '='
16 %left '+' '-'
17 %left '*' '/'
18 %nonassoc '|' UMINUS // 一些优先级声明
19
20 stmt: IF exp THEN list { $$ = newflow('I', $2, $4, NULL); }
21      | IF exp THEN list ELSE list { $$ = newflow('I', $2, $4, $6); }
22      | WHILE exp DO list { $$ = newflow('W', $2, $4, NULL); }
23      | exp // 定义语句规则, if-then、if-then-else、while-do或者exp
24 ;
25
26 list: /* nothing */ { $$ = NULL; }
27      | stmt ';' list { if ($3 == NULL)
28                          $$ = $1;
29                          else
30                          $$ = newast('L', $1, $3);
31                      } // 定义过个语句以;分割
32 ;
33
34 exp: exp CMP exp { $$ = newcmp($2, $1, $3); }
35      | exp '+' exp { $$ = newast('+', $1,$3); }
36      | exp '-' exp { $$ = newast('-', $1,$3); }
37      | exp '*' exp { $$ = newast('*', $1,$3); }
38      | exp '/' exp { $$ = newast('/', $1,$3); }
39      | '|' exp { $$ = newast('|', $2, NULL); }
40      | '(' exp ')' { $$ = $2; }
41      | '-' exp %prec UMINUS { $$ = newast('M', $2, NULL); }
42      | NUMBER { $$ = newnum($1); }
43      | FUNC '(' explist ')' { $$ = newfunc($1, $3); }
44      | NAME { $$ = newref($1); }
45      | NAME '=' exp { $$ = newasgn($1, $3); }
46      | NAME '(' explist ')' { $$ = newcall($1, $3); } // 定义表达式规则
47 ;
48

```

```

49 explist: exp
50   | exp ',' explist { $$ = newast('L', $1, $3); } // 定义表达式列表
51 ;
52 symlist: NAME      { $$ = newsymlist($1, NULL); }
53   | NAME ',' symlist { $$ = newsymlist($1, $3); } // 定义符号列表
54 ;
55
56 calclist: /* nothing */
57   | calclist stmt EOL {
58       if(debug) dumpast($2, 0);
59       printf("= %4.4g\n> ", eval($2));
60       treefree($2);
61   }
62   | calclist LET NAME '(' symlist ')' '=' list EOL {
63       dodef($3, $5, $8);
64       printf("Defined %s\n> ", $3->name); }
65   | calclist error EOL { yyerrok; printf("> "); }
66   // 顶层规则，识别表达式与函数声明
67 ;

```

3.3.2 fb3-2.1

fb3-2.1相对于fb3-1.1有以下修改

```

1  "=" |
2  ",", |
3  ";" { return yytext[0]; } // 定义单字符操作
4
5  ">" { yylval.fn = 1; return CMP; }
6  "<" { yylval.fn = 2; return CMP; }
7  "<>" { yylval.fn = 3; return CMP; }
8  "==" { yylval.fn = 4; return CMP; }
9  ">=" { yylval.fn = 5; return CMP; }
10 "<=" { yylval.fn = 6; return CMP; } // 定义一些比较运算
11
12 "if" { return IF; }
13 "then" { return THEN; }
14 "else" { return ELSE; }
15 "while" { return WHILE; }
16 "do" { return DO; }
17 "let" { return LET; } // 定义一些关键字
18
19 "sqrt" { yylval.fn = B_sqrt; return FUNC; }

```

```

20 "exp"    { yylval.fn = B_exp; return FUNC; }
21 "log"    { yylval.fn = B_log; return FUNC; }
22 "print"  { yylval.fn = B_print; return FUNC; } // 定义一些函数
23
24 [a-zA-Z][a-zA-Z0-9]* { yylval.s = lookup(yytext); return NAME; } // 定义关键字标识符

```

3.3.3 fb3-2funcs.c

fb3-2funcs.c主要是实现一些预定的结构或者方法，fb3-2funcs.c相对于fb3-1funcs.c有以下修改

```

1 static unsigned symhash(char *sym){
2     unsigned int hash = 0;
3     unsigned c;
4
5     while(c = *sym++) hash = hash*9 ^ c;
6
7     return hash;
8 } // 符号哈希值
9
10 struct symbol * lookup(char* sym){
11     struct symbol *sp = &symtab[symhash(sym)%NHASH];
12     int scount = NHASH;          /* how many have we looked at */
13
14     while(--scount >= 0) {
15         if(sp->name && !strcmp(sp->name, sym)) { return sp; }
16
17         if(!sp->name) {          /* new entry */
18             sp->name = strdup(sym);
19             sp->value = 0;
20             sp->func = NULL;
21             sp->syms = NULL;
22             return sp;
23         }
24         if(++sp >= symtab+NHASH) sp = symtab; /* try the next entry */
25     }
26     yyerror("symbol table overflow\n");
27     abort(); /* tried them all, table is full */
28 } // 查找符号
29
30 struct ast * newcmp(int cmptype, struct ast *l, struct ast *r){
31     struct ast *a = malloc(sizeof(struct ast));
32     if(!a) {

```

```

33         yyerror("out of space");
34         exit(0);
35     }
36     a->nodetype = '0' + cmptype;
37     a->l = l;
38     a->r = r;
39     return a;
40 } // 比较表达式抽象语法树结点

```

3.3.4 fb3-2.h

fb3-2.h相对于fb3-1.h有以下修改

```

1 // 定义符号表结构及其相关方法
2 struct symbol {                                /* a variable name */
3     char *name;
4     double value;
5     struct ast *func;                          /* stmt for the function */
6     struct symlist *syms; /* list of dummy args */
7 };
8
9 /* simple symtab of fixed size */
10 #define NHASH 9997
11 struct symbol symtab[NHASH];
12
13 struct symbol *lookup(char*);
14
15 /* list of symbols, for an argument list */
16 struct symlist {
17     struct symbol *sym;
18     struct symlist *next;
19 };
20
21 struct symlist *newsymlist(struct symbol *sym, struct symlist *next);
22 void symlistfree(struct symlist *sl);
23
24 enum bifs {                                     // 预定义函数
25     B_sqrt = 1,
26     B_exp,
27     B_log,
28     B_print
29 };
30

```

```
31 struct fncall {          // 调用预定义函数
32     int nodetype;          /* type F */
33     struct ast *l;
34     enum bifs functype;
35 };
36
37 struct ufncall {          // 调用用户定义函数
38     int nodetype;          /* type C */
39     struct ast *l;          /* list of arguments */
40     struct symbol *s;
41 };
42
43 struct flow {
44     int nodetype;          /* type I or W */
45     struct ast *cond;       /* condition */
46     struct ast *tl;         /* then or do list */
47     struct ast *el;         /* optional else list */
48 };
49
50 struct numval { // 数值
51     int nodetype;          /* type K */
52     double number;
53 };
54
55 struct symref { // 符号查找
56     int nodetype;          /* type N */
57     struct symbol *s;
58 };
59
60 struct symasgn { // 符号赋值
61     int nodetype;          /* type = */
62     struct symbol *s;
63     struct ast *v;          /* value */
64 };
```

3.4 实验过程

3.4.1 Windows

将实验代码放在实验目录ex5-2下，打开Developer Command Prompt for VS 2019，输入如下命令，结果如图3.1所示：

```
bison -d fb3-2.y
```

```
flex -ofb3-2.lex.c fb3-2.1
cl fb3-2.tab.c fb3-2.lex.c fb3-2funcs.c
```



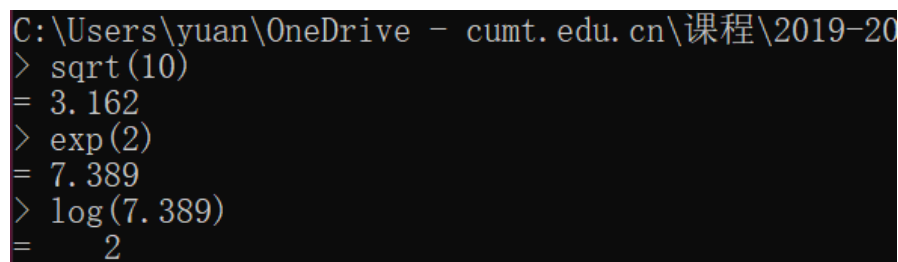
```
C:\Users\yuan\OneDrive - cumt.edu.cn\课程\2019-2020-2 系统软件开发实践\报告\05
fb3-2funcs.c
用于 x86 的 Microsoft (R) C/C++ 优化编译器 19.24.28316 版
版权所有 (C) Microsoft Corporation。保留所有权利。

fb3-2.tab.c
fb3-2.lex.c
fb3-2funcs.c
正在生成代码...
Microsoft (R) Incremental Linker Version 14.24.28316.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:fb3-2.tab.exe
fb3-2.tab.obj
fb3-2.lex.obj
fb3-2funcs.obj
```

图 3.1: Windows编译结果

然后输入fb3-2.tab.exe, 进行实验测试, 依次输入 $\sqrt{10}$ 、 $\exp(2)$ 、 $\log(7.389)$, 结果如图3.2所示。

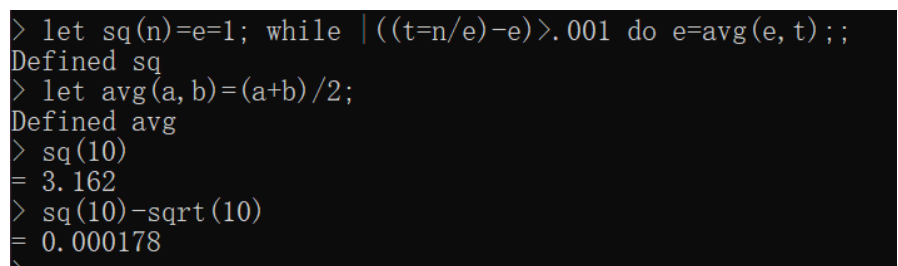


```
C:\Users\yuan\OneDrive - cumt.edu.cn\课程\2019-2020
> sqrt(10)
= 3.162
> exp(2)
= 7.389
> log(7.389)
= 2
```

图 3.2: Windows实验结果1

然后再依次输入如下代码, 进行函数自定义, 结果如图3.3所示。

```
let sq(n)=e=1; while |((t=n/e)-e)>.001 do e=avg(e,t);;
let avg(a,b)=(a+b)/2;
sq(10)
sq(10)-sqrt(10)
```



```
> let sq(n)=e=1; while |((t=n/e)-e)>.001 do e=avg(e,t);;
Defined sq
> let avg(a,b)=(a+b)/2;
Defined avg
> sq(10)
= 3.162
> sq(10)-sqrt(10)
= 0.000178
>
```

图 3.3: Windows实验结果2

3.4.2 Ubuntu

将实验代码放在实验目录ex5-2下，打开终端，输入如下命令，结果如图3.4所示：

```
bison -d fb3-2.y
flex -ofb3-2.lex.c fb3-2.l
cc fb3-2.tab.c fb3-2.lex.c fb3-2funcs.c -lm
```

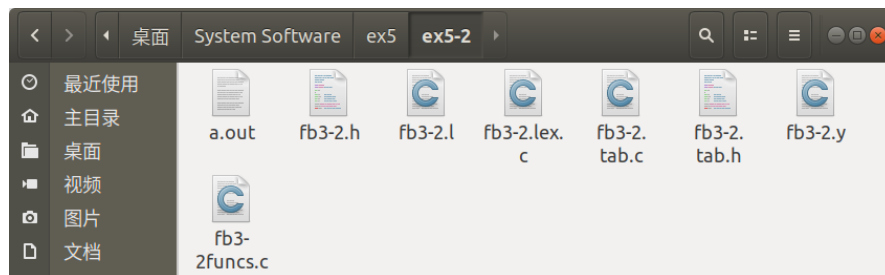


图 3.4: Ubuntu编译结果

然后输入./，进行实验测试，依次输入 $\sqrt{10}$ 、 $\exp(2)$ 、 $\log(7.389)$ ，结果如图3.5所示。

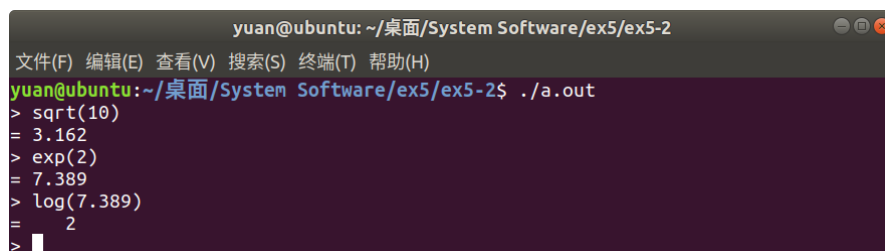


图 3.5: Ubuntu实验结果1

然后再依次输入如下代码，进行函数自定义，结果如图3.6所示。

```
let sq(n)=e=1; while |((t=n/e)-e)>.001 do e=avg(e,t);;
let avg(a,b)=(a+b)/2;
sq(10)
sq(10)-sqrt(10)
```

3.4.3 抽象语法树构建

$\sqrt{10}$ 抽象语法树如图3.7所示， $\exp(2)$ 抽象语法树如图3.8所示，

3.5 实验感想

经过这次实验我对flex与bison如何进行计算器的运算有了更加清晰的认识，在自定义函数的时候，首先分析了代码如何工作，虽然有一些还不明白，但是

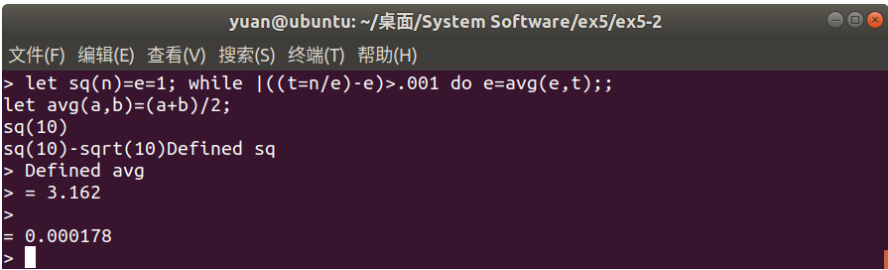


图 3.6: Ubuntu实验结果2

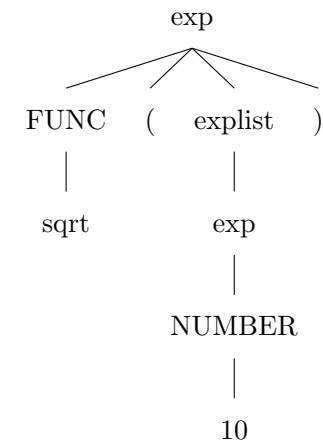


图 3.7: sqrt(10)抽象语法树

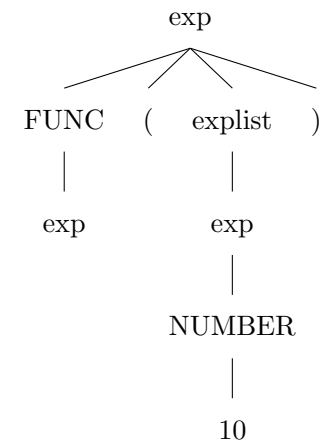


图 3.8: exp(2)抽象语法树

经过实际的实验操作，明白了绝大部分的代码，这对于我接下来的实验很有帮助。

4 实验5-3

4.1 实验目标

- 1) 阅读《Flex&Bison》第三章 P79习题1，学习抽象语法树。
- 2) 修改fb3-2的相关代码，实现以下自定义函数，并保存为fb3-3。
- 3) 撰写实验报告，结合实验结果，给出抽象语法树的构建过程。

4.2 修改思路

根据实验要求，这次是修改定义函数的形式，如：let avg(a,b)=(a+b)/2;修改为let avg(a,b) { (a+b)/2; }，最明显特征是更加贴近代码的格式。在文件fb3-3.y中，原来定义方式如图4.1所示，修改为图4.2所示的代码。

原来list规则定义如图4.3所示，修改为如图4.4所示。
添加了{与}符号，需要修改fb3-3.l文件中的符号定义，由图4.5修改为图4.6所示。


```

calclist: /* nothing */
| calclist stmt EOL {
    if(debug) dumpast($2, 0);
    printf("= %4.4g\n> ", eval($2));
    treefree($2);
}
| calclist LET NAME '(' symlist ')' '=' list EOL {
    dodef($3, $5, $8);
    printf("Defined %s\n> ", $3->name); }

```

图 4.1: 原来函数定义方式

```

calclist: /* nothing */
| calclist stmt EOL {
    if(debug) dumpast($2, 0);
    printf("= %4.4g\n> ", eval($2));
    treefree($2);
}
| calclist LET NAME '(' symlist ')' list EOL {
    dodef($3, $5, $7);
    printf("Defined %s\n> ", $3->name); }
| calclist error EOL { yyerrok; printf("> "); }

```

图 4.2: 修改后的函数定义方式

```

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

```

图 4.3: 原来list定义

```

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

```

图 4.4: 修改后的list定义

```

19 %%
20 /* single character ops */
21 "+" |
22 "-" |
23 "*" |
24 "/" |
25 "=" |
26 "|" |
27 ";" |
28 "," |
29 "(" |
30 ")" { return yytext[0]; }

```

图 4.5: 原来符号定义

```

20 /* single character ops */
21 "+" |
22 "-" |
23 "*" |
24 "/" |
25 "=" |
26 "|" |
27 ";" |
28 "," |
29 "(" |
30 ")" |
31 "{" |
32 "}" { return yytext[0]; }

```

图 4.6: 修改后的符号定义

编译bison代码，发现报移进规约冲突，如图4.7所示，然后通过命令查看冲突所在的位置，如图4.8所示。

```
C:\Users\yuan\OneDrive - cumt.edu.cn\课程\2019-2020-2
fb3-3.y: conflicts: 3 shift/reduce
```

图 4.7: 移进规约冲突

```
6 State 5 conflicts: 1 shift/reduce
7 State 49 conflicts: 1 shift/reduce
8 State 50 conflicts: 1 shift/reduce
```

图 4.8: 移进规约冲突位置

错误1如图4.9所示，当移进NAME后，遇到(时程序不能判断是否进行规约或者移进，所以这里需要修改优先级，将NAME改为没有结合性(改为左结合性)。在定义部分增加%nonassoc LOWER与%left '(', 然后修改代码，如图4.10所示。

```
154 state 5
155
156 19 exp: NAME .
157 20 | NAME . '=' exp
158 21 | NAME . '(' explist ')'
159
160 '=' shift, and go to state 16
161 '(' shift, and go to state 17
162
163 '(' [reduce using rule 19 (exp)]
164 $default reduce using rule 19 (exp)
```

图 4.9: 移进规约冲突位置1

```
62 exp: exp CMP exp { $$ = newcmp($2, $1, $3); }
63 | exp '+' exp { $$ = newast('+', $1,$3); }
64 | exp '-' exp { $$ = newast('-', $1,$3); }
65 | exp '*' exp { $$ = newast('*', $1,$3); }
66 | exp '/' exp { $$ = newast('/', $1,$3); }
67 | '|' exp { $$ = newast('|', $2, NULL); }
68 | '(' exp ')' { $$ = $2; }
69 | '-' exp %prec UMINUS { $$ = newast('M', $2, NULL); }
70 | NUMBER { $$ = newnum($1); }
71 | FUNC '(' explist ')' { $$ = newfunc($1, $3); }
72 | NAME %prec LOWER { $$ = newref($1); }
73 | NAME '=' exp { $$ = newasgn($1, $3); }
74 | NAME '(' explist ')' { $$ = newcall($1, $3); }
75 ;
```

图 4.10: 移进规约冲突位置1修改

错误2如图4.11所示，当移进list后，遇到ELSE时程序不能判断是否进行规约或者移进，所以这里需要修改优先级，提高stmt:IF exp THEN list的优先级，如图4.12所示。

错误3如图4.13所示，当移进-后，遇到exp时程序不能判断是否进行规约或者移进，所以这里需要修改优先级，提高Istmt:exp的优先级，如图4.14所示。

```

716 state 49
717
718     1 stmt: IF exp THEN list .
719       2   | IF exp THEN list . ELSE list
720
721       ELSE shift, and go to state 56
722
723       ELSE [reduce using rule 1 (stmt)]
724       $default reduce using rule 1 (stmt)

```

图 4.11: 移进规约冲突位置2

```

49 stmt: IF exp THEN list %prec LOWER { $$ = newflow('I', $2, $4, NULL); }
50 | IF exp THEN list ELSE list { $$ = newflow('I', $2, $4, $6); }
51 | WHILE exp DO list { $$ = newflow('W', $2, $4, NULL); }
52 | exp
53 ;

```

图 4.12: 移进规约冲突位置2修改

```

725 state 50
726
727     4 stmt: exp .
728     8 list: exp . ';'
729     9 exp: exp . CMP exp
730    10 | exp . '+' exp
731    11 | exp . '-' exp
732    12 | exp . '*' exp
733    13 | exp . '/' exp
734
735    CMP shift, and go to state 26
736    '+' shift, and go to state 27
737    '-' shift, and go to state 28
738    '*' shift, and go to state 29
739    '/' shift, and go to state 30
740    ';' shift, and go to state 57
741
742    '-' [reduce using rule 4 (stmt)]
743    $default reduce using rule 4 (stmt)

```

图 4.13: 移进规约冲突位置3

```

49 stmt: IF exp THEN list %prec LOWER { $$ = newflow('I', $2, $4, NULL); }
50 | IF exp THEN list ELSE list { $$ = newflow('I', $2, $4, $6); }
51 | WHILE exp DO list { $$ = newflow('W', $2, $4, NULL); }
52 | exp %prec LOWER
53 ;

```

图 4.14: 移进规约冲突位置3修改

4.3 实验过程

4.3.1 Windows

打开实验工具，实验命令同上一轮的，结果如图4.15所示。

```
C:\Users\yuan\OneDrive - cumt.edu.cn\课程\2019-2020-2 系统软件开发实践\报告
d fb3-3.1 && cl lex.yy.c fb3-3.tab.c fb3-3funcs.c
用于 x86 的 Microsoft (R) C/C++ 优化编译器 19.25.28614 版
版权所有 (C) Microsoft Corporation。保留所有权利。

lex.yy.c
fb3-3.tab.c
fb3-3funcs.c
正在生成代码...
Microsoft (R) Incremental Linker Version 14.25.28614.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:lex.yy.exe
lex.yy.obj
fb3-3.tab.obj
fb3-3funcs.obj
```

图 4.15: windows编译结果

分别操作步骤1、2、3，结果如图4.16、4.17、4.18所示。

```
C:\Users\yuan\OneDrive - cumt.edu.cn\课程\2019-2020-2 系统软件开发实践\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
```

图 4.16: windows步骤1

```
> 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
```

图 4.17: windows步骤2

```
> 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 4.18: windows步骤3

4.3.2 Ubuntu

打开终端，输入相应命令，结果如图4.19所示。

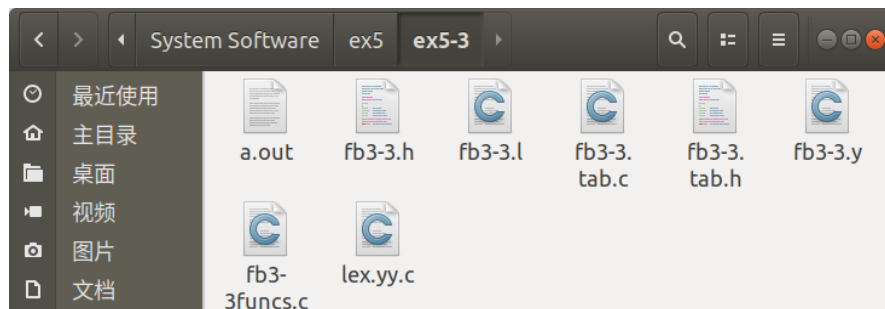


图 4.19: Ubuntu编译结果

分别操作步骤1、2、3，结果如图4.20、4.21、4.22所示。

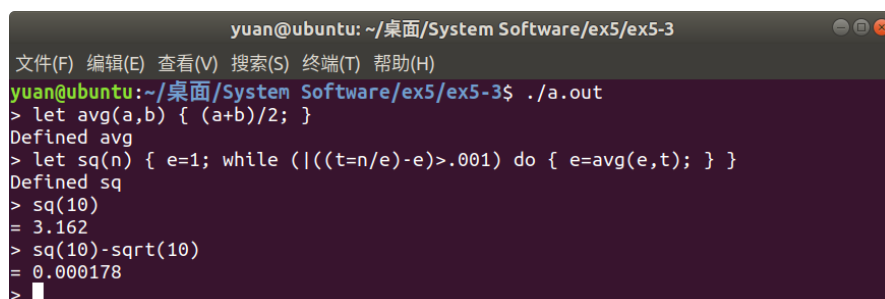


图 4.20: Ubuntu步骤1

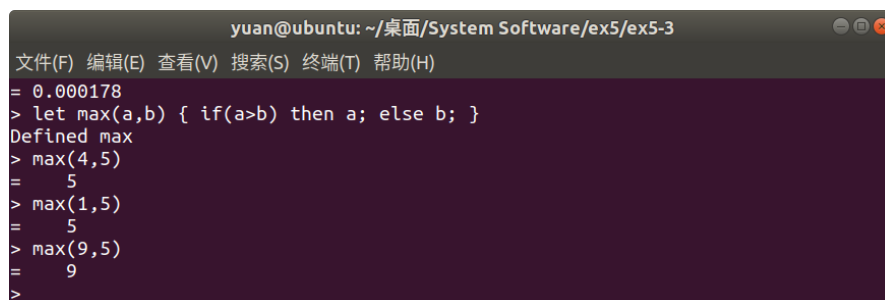


图 4.21: Ubuntu步骤2

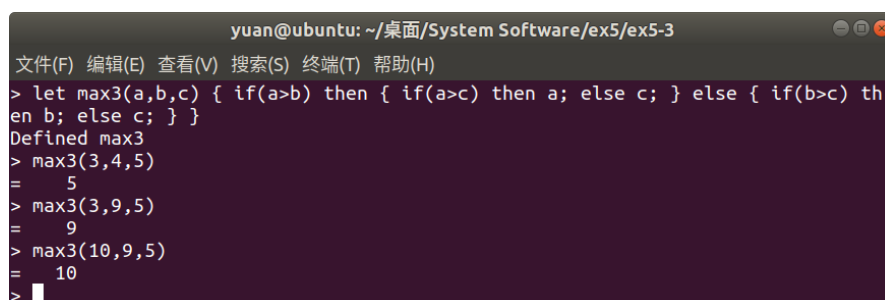
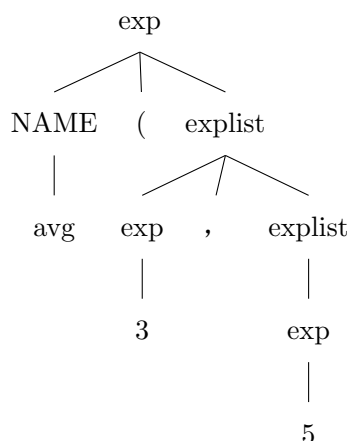
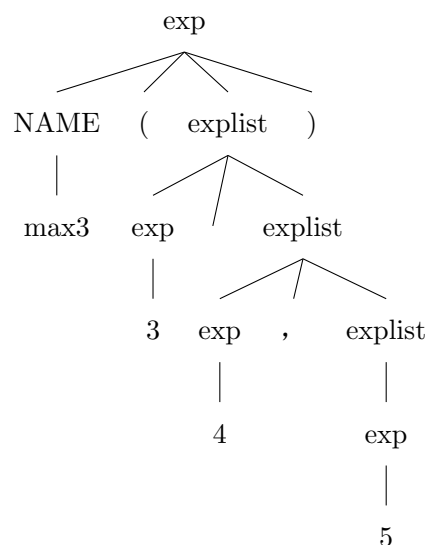


图 4.22: Ubuntu步骤3

4.3.3 抽象语法树构建

图 4.23: *avg*(3,5)抽象语法树图 4.24: *max3*(3,4,5)抽象语法树

4.4 实验感想

这次实验比较麻烦，原因是编写好规则，编译没问题，但是定义函数的时候会出错。为了检测问题，又去学习了怎样输出调试日志的方法，最后发现是自己规则编写有一些问题，当编写好之后，后面就是解决移进规约问题了，因为有了前面的基础，解决很快。总之，这次实验锻炼了很多，不仅仅是代码编写，也对于我理解编译的知识也有很大帮助，希望接下来的实验能再提高。

5 实验5-4

5.1 实验目标

使用Flex和Bison开发了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；

5.2 配置与修改代码

5.2.1 开发环境

- 环境平台：Windows，Ubuntu

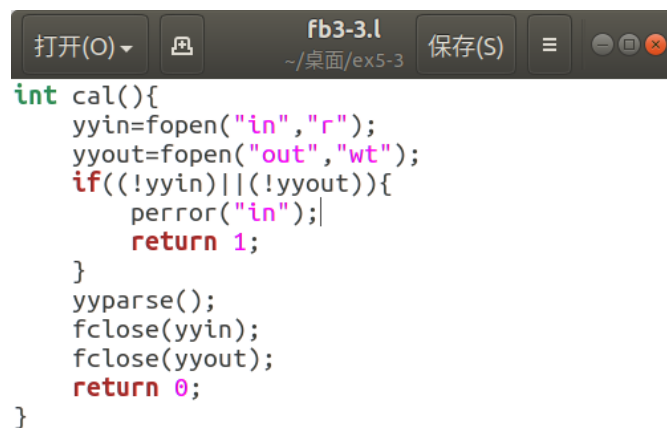
- 编程语言：python（Windows32位，Ubuntu64位），vs2019

5.2.2 基本思路

首先在修改flex与bison文件，建立一个与前端交互的接口（见代码修改部分），然后相应的开发环境下编译动态链接文件，windows编译dll文件，ubuntu编译so文件，计算器的可视化界面使用tkinter模块，交互后端接口使用ctypes模块。

5.2.3 代码修改

Ubuntu环境下配置较为简单，修改fb3-3.l文件，在最下方增加一个cal()计算函数，作用是从in文件读取输入的文本，经过分析计算，将结果写入out文件，如图5.1所示。



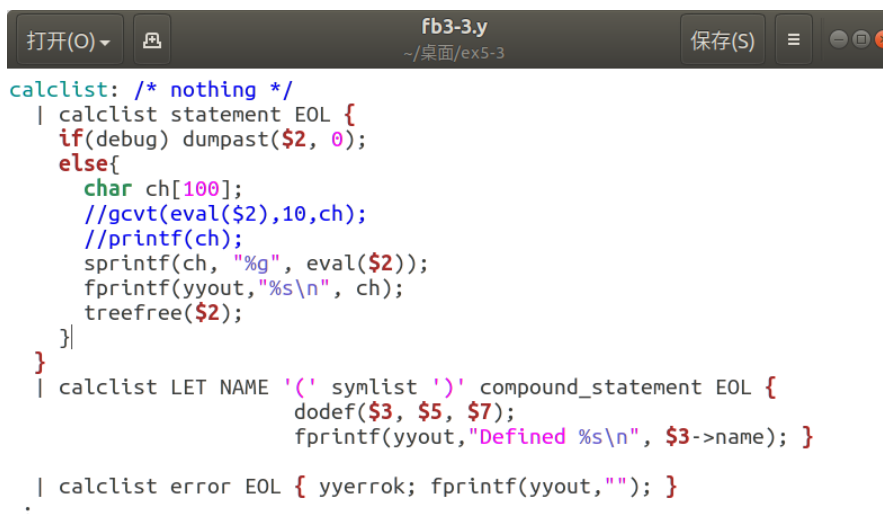
```

int cal(){
    yyin=fopen("in","r");
    yyout=fopen("out","wt");
    if((!yyin)||(!yyout)){
        perror("in");|
        return 1;
    }
    yyparse();
    fclose(yyin);
    fclose(yyout);
    return 0;
}

```

图 5.1: Ubuntu修改1

还需要修改fb3-3.y文件，在最下方将原来的printf输出改为fprintf输出，作用是格式化输出到一个流文件中，如图5.2所示。



```

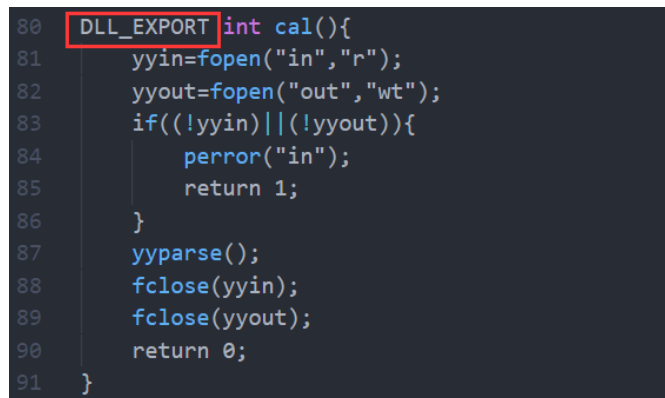
calclist: /* nothing */
| calclist statement EOL {
    if(debug) dumpast($2, 0);
    else{
        char ch[100];
        //gcvf(eval($2),10,ch);
        //printf(ch);
        sprintf(ch, "%g", eval($2));
        fprintf(yyout, "%s\n", ch);
        treefree($2);
    }
}
| calclist LET NAME '(' symlist ')' compound_statement EOL {
    dodef($3, $5, $7);
    fprintf(yyout, "Defined %s\n", $3->name); }
| calclist error EOL { yyerrok; fprintf(yyout, ""); }
:

```

图 5.2: Ubuntu修改2

对于windows环境，需要在此基础上再次进行修改，不然在使用ctypes读取动态链接文件时，无法解析到cal()函数，具体操作如下。在fb3-3.1的声明部分加入如下代码，然后在cal()函数的返回值左侧加入DLL_EXPORT声明，将cal()声明为外部可调用函数，如图5.3所示。

```
#ifdef _MSC_VER
    #define DLL_EXPORT __declspec( dllexport )
#else
    #define DLL_EXPORT
#endif
```



```
80  DLL_EXPORT int cal(){
81      yyin=fopen("in","r");
82      yyout=fopen("out","wt");
83      if(!yyin||!yyout){
84          perror("in");
85          return 1;
86      }
87      yyparse();
88      fclose(yyin);
89      fclose(yyout);
90      return 0;
91  }
```

图 5.3: Windows附加修改

5.3 实验过程

5.3.1 Windows

首先进行dll动态链接库的编译，打开Developer Command Prompt for VS 2019，进入实验目录，输入`cl /LDd lex.yy.c fb3-3funcs.c fb3-3.tab.c`，会发现生成了lex.yy.dll和一些obj文件，如图5.4所示。



```
C:\Users\yuan\OneDrive - cumt.edu.cn\课程\2019-2020-2 系统软件开发实践\报告\05实验五\
ab.c
用于 x86 的 Microsoft (R) C/C++ 优化编译器 19.25.28614 版
版权所有 (C) Microsoft Corporation。保留所有权利。

lex.yy.c
fb3-3funcs.c
fb3-3.tab.c
正在生成代码...
Microsoft (R) Incremental Linker Version 14.25.28614.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:lex.yy.dll
/dll
/implib:lex.yy.lib
lex.yy.obj
fb3-3funcs.obj
fb3-3.tab.obj
正在创建库 lex.yy.lib 和对象 lex.yy.exp
```

图 5.4: Windows编译dll

python简单调用dll中的cal()函数，如下。


```
from ctypes import *

so = cdll.LoadLibrary("./libcalc.dll")
so.cal()
```

运行编写好的cal.py文件，显示界面如图5.5所示。

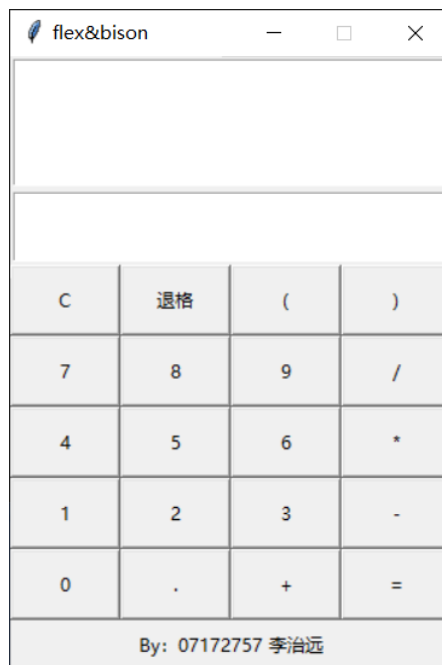


图 5.5: Windows计算器界面

5.3.2 Ubuntu

新建一个Makefile文件在实验目录下，输入下述内容，打开终端，进入实验目录，输入make，进行so文件编译，结果如图5.6所示。

```
CC = clang
FLAGS = -Wall -std=gnu99 -g -lm
libcalc.so: fb3-3.tab.c lex.yy.c fb3-3funcs.c
    ${CC} ${FLAGS} -fPIC -shared $^ -o $@
libcalc.a: lex.yy.o fb3-3.tab.o fb3-3funcs.o
    ar -rcs $@ $^
    ranlib $@
fb3-3.tab.o: fb3-3.tab.c
    ${CC} ${FLAGS} $^ -c -o $@
fb3-3.tab.c: fb3-3.y
    bison -d $^
lex.yy.o: lex.yy.c
    ${CC} ${FLAGS} $^ -c -o $@
fb3-3funcs.o: fb3-3funcs.c
```

```

    ${CC} ${FLAGS} $^ -c -o $@
lex.yy.c: fb3-3.1
    flex $^
clear:
    rm *.o
```



图 5.6: Ubuntu编译so

运行编写好的cal.py文件，显示界面如图5.7所示。



图 5.7: Ubuntu计算器界面

5.4 实验演示

简单计算，输入 $1 + 2 - (3 * 6)$ ，得到结果-15，如图5.8所示。定义变量a，并赋值计算，输入 $a = 1, a + 2$ ，得到结果3，如图5.9所示。



图 5.8: 计算器演示1

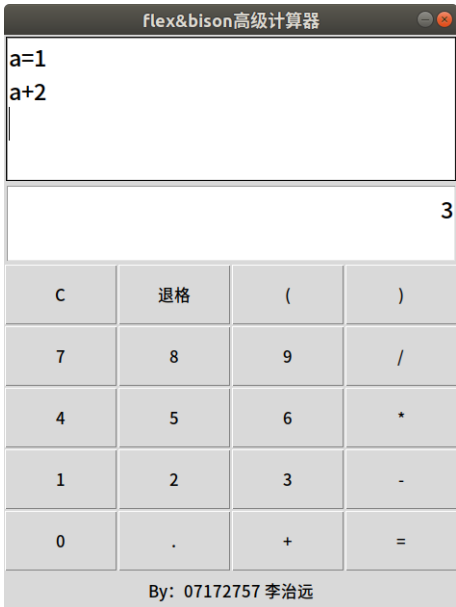


图 5.9: 计算器演示2

进行比较运算，输入 $a < 2$ ，得到1 (true)，结果如图5.10所示。进行if-then-else运算，输入 $if(a = 1)then\{a = 2;\}else\{a = 0;\}$ 得到结果为0，如图5.11所示。

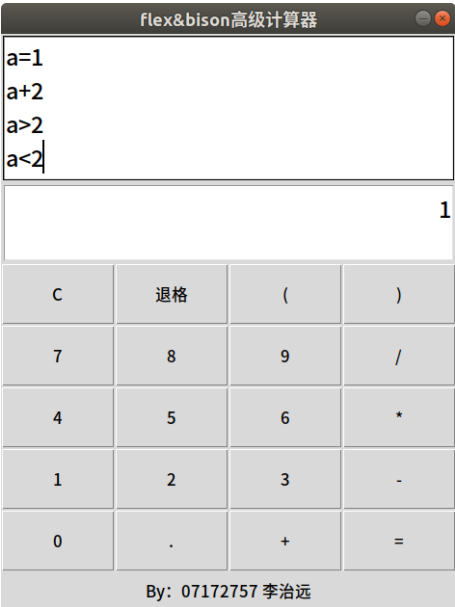


图 5.10: 计算器演示3

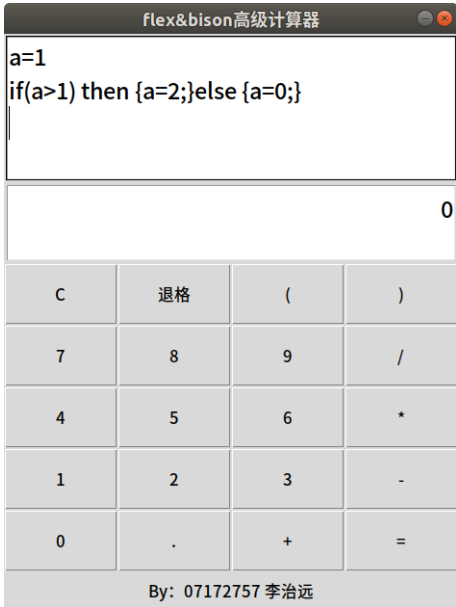


图 5.11: 计算器演示4

定义函数，输入 $let \quad avg(a,b)\{(a + b)/2;\}$ ，定义一个avg函数，然后输入 $let \quad sq(n)\{e = 1; while(|((t = n/e) - e) > .001)do\{e = avg(e,t);\}$ ，最后输入 $sq(10);$ ，得到结果3.16246，如图5.12所示。

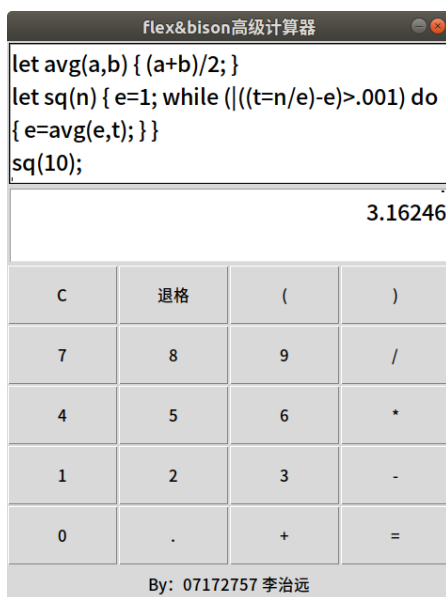


图 5.12: 计算器演示5

5.5 扩展功能

在以上的基础上，增加画函数图的功能。

修改fb3-3.y文件，增加 `calclist: calclist C statement EOL {draw($3);}` 规则，并声明C为token。修改fb3-3.l，在keywords部分增加 `"c" { return C; }`，在 built in functions 部分增加 `"sin" { yylval.fn = B_sin; return FUNC; }`，修改fb3-3.h文件，在enum bifs内部增加B_sin，修改fb3-3funcs.c文件，在最后面增加下述代码。

```
void draw(struct ast *a){
    struct ast *b, *c;
    double x, y, dx = 1.0 / 16, dy = 1.0 / 16;
    const char *pFileName = "out.txt";
    FILE* fp = NULL; // 文件指针
    fopen_s(&fp, pFileName, "a+");
    for (y = 1.0; y >= -1.0; y -= dy){
        for (x = 0.0; x < 6.28; x += dx){
            struct symbol *sp = &syntab[symhash("x") % NHASH];
            c = newnum(x);
            b = newasgn(sp, c);
            eval(b);
            char p[2] = {" *"[fminf(fabs(eval(a) - y), fabs(y)) < dy / 2]};
            fputs(p, fp);

            putchar(" *"[fminf(fabs(eval(a) - y), fabs(y)) < dy / 2]);
        }
    }
}
```

```
fputs("\n", fp);
putchar('\n');
}
fclose(fp);
}
```

编译进行实验，输入c sin(x)，如图5.13所示，按下等于号，结果如图5.14-5.15所示。



图 5.13: 计算器演示6

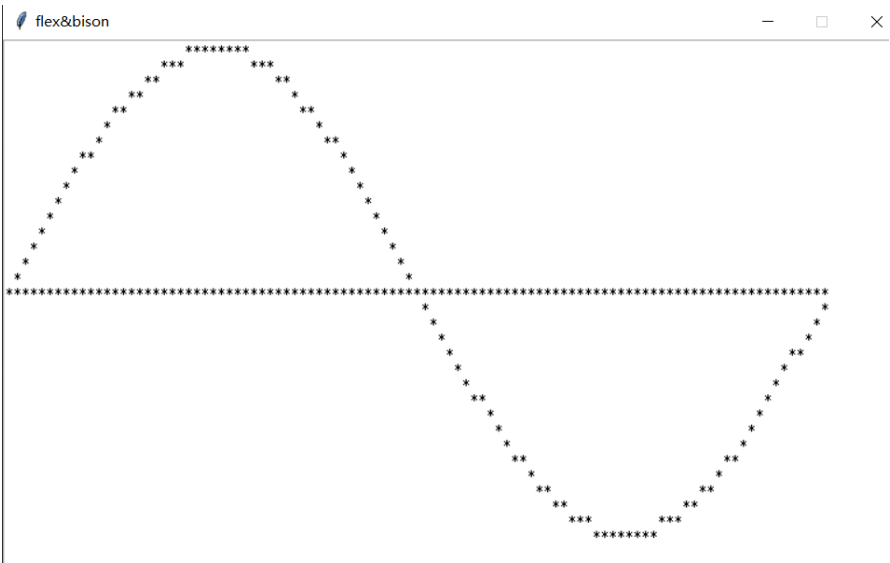


图 5.14: 演示结果1

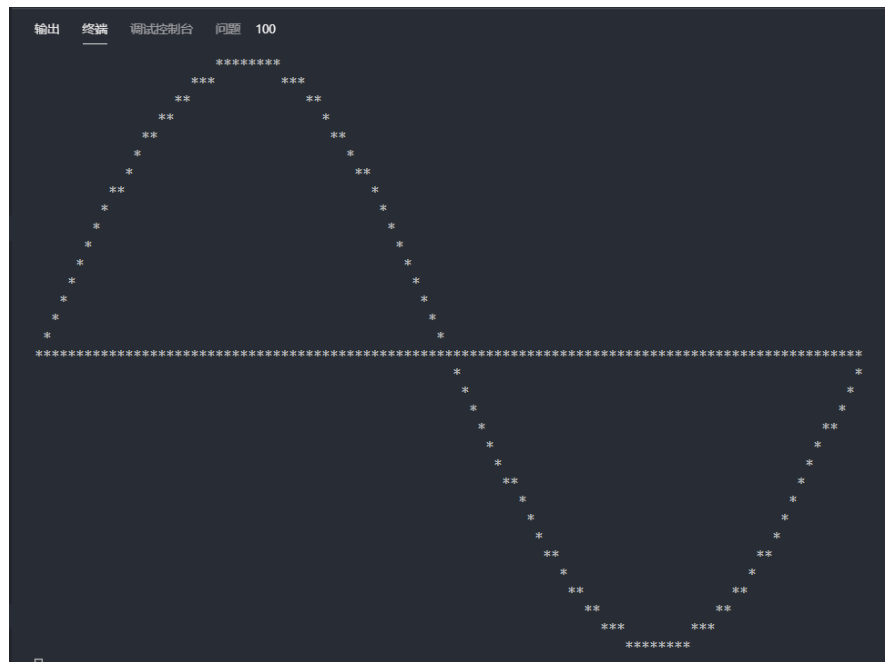


图 5.15: 演示结果2