**中国矿业大学计算机学院**

**2018 级本科生课程设计报告**

课程名称 系统软件开发实践

实验名称 实验3-3修改高级计算器

报告时间 2021年3月24日

学生姓名 丁旭行

学 号 14184501

专 业 计算机科学与技术

任课教师 徐东红

**目 录**

[**1 实验目的 1**](#_Toc67470814)

[**2 实验内容 1**](#_Toc67470815)

[**3 实验要求 1**](#_Toc67470816)

[**4 高级计算器功能 1**](#_Toc67470817)

[**5 相关代码修改 2**](#_Toc67470818)

[**6 实验步骤 4**](#_Toc67470819)

[**6.1 Windows环境 4**](#_Toc67470820)

[**6.2 Linux环境 5**](#_Toc67470821)

[**7 分析抽象语法树构建过程 7**](#_Toc67470822)

[**8 实验总结 11**](#_Toc67470823)

# 1 实验目的

使用Flex和Bison开发一个具有全部功能的桌面计算器。

# 2 实验内容

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，但是具有现实意义的编译器。重点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。

# 3 实验要求

(1) 阅读《Flex&Bison》第三章，学习抽象语法树”Abstract Syntax Trees”；

(2) 根据习题1的要求，修改fb3-2相关代码，保存为fb3-3，完成实验；

(3) 撰写实验报告，结合实验结果，说明相关的修改代码，给出抽象语法树的构建过程；

(4) 提交报告和实验代码。

# 4 高级计算器功能

* 变量命名；
* 实现赋值功能；
* 实现比较表达式(大于、小于、等于等等)；
* 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。

(注：此处实现功能均是在基本计算器功能基础之上追加的功能)

# 5 相关代码修改

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以及循环语句部分，同样需要可以支持“{……}”的括弧定义。

首先修改fb3-2.l代码完成词法分析器对‘{‘和’}’的读取识别，并另存为fb3-3.l文件。

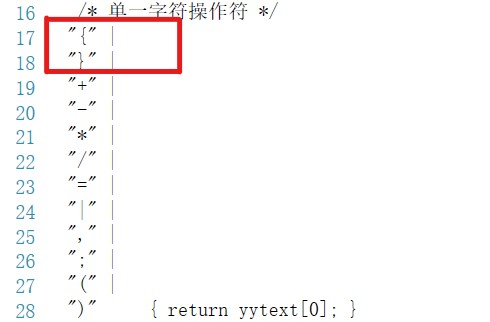


图1 fb3-3.l修改代码

观察fb3-2.y文件中原有的语句定义，考虑对涉及的控制选择语句、关键语句list部分和编写用户自定义函数部分进行如下的修改，并另存为fb3-3.y文件。

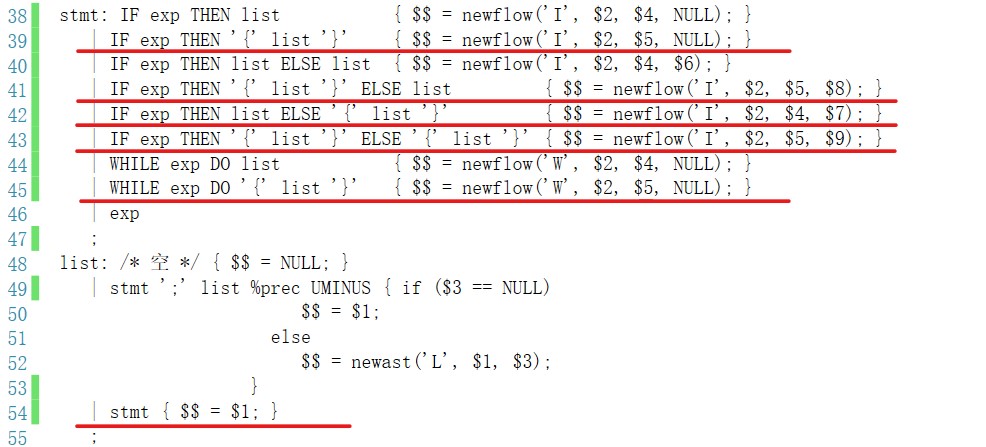


图2 fb3-3.y修改代码1

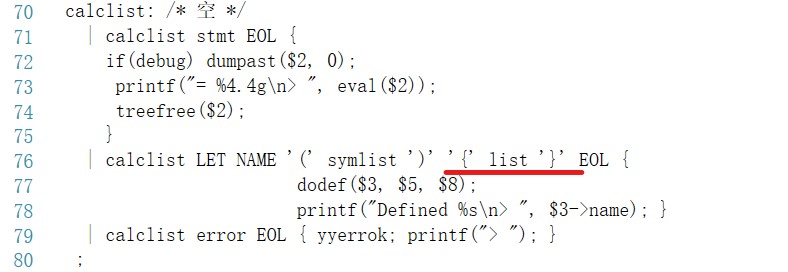


图3 fb3-3.y修改代码2

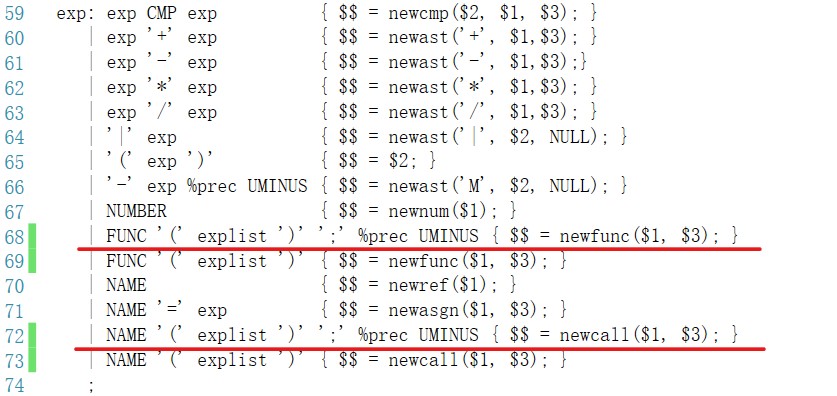


图4 fb3-3.y修改代码3

代码修改完成后，在编译运行时提示如下移进归约冲突。

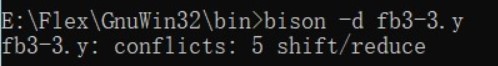


图5 移进归约冲突

针对移进规约冲突，可以在bison编译时通过-v命令生成状态机描述文件，打开.output文件，具体查找对应的移进归约冲突，下图为部分冲突说明。

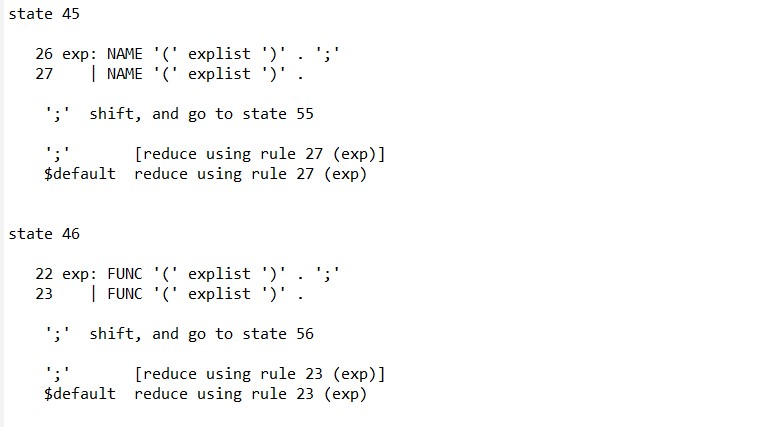


图6 部分具体冲突说明

以上图为例，语法分析器在接收到‘)‘后，面临下一个字符’;’，此时前面所有已经读入的字符可以进行归约也可以继续读入’;’。bison对于移进归约冲突默认以移进为主，此时我们发现移进’;’正是我们想要的结果，所有虽然有移进规约冲突，但是无需进行特殊修改。

# 6 实验步骤

## 6.1 Windows环境

(1) 打开Developer Command Prompt for VS 2019命令窗口，依次输入e:、cd E:\Flex\GnuWin32\bin进入安装目录下，再依次输入bison -d fb3-3.y、flex -ofb3-3.lex.c fb3-3.l，生成fb3-3.tab.h、fb3-3.tab.c和fb3-3.lex.c文件。

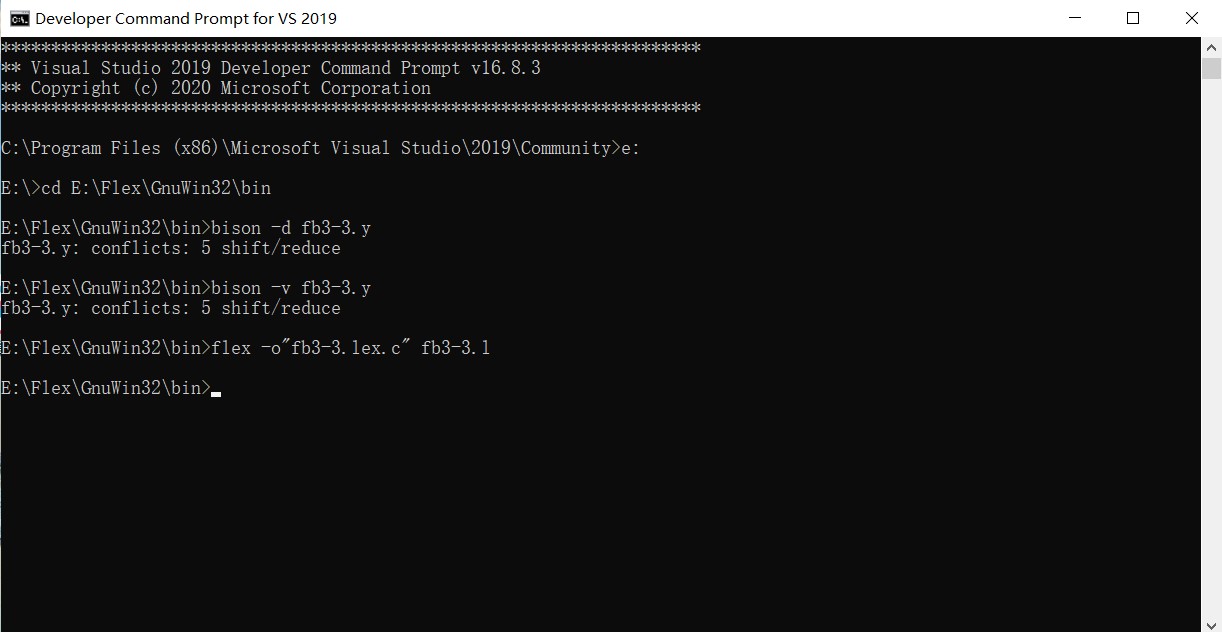


图7 输入命令

(2) 打开Cygwin命令窗口，输入cd E:/Flex/GnuWin32/bin进入所在目录，再输入命令gcc -o”fb3-3.tab.exe” fb3-3.tab.c fb3-3.lex.c fb3-3funcs.c -lm生成其对应的fb3-3.tab.exe文件。

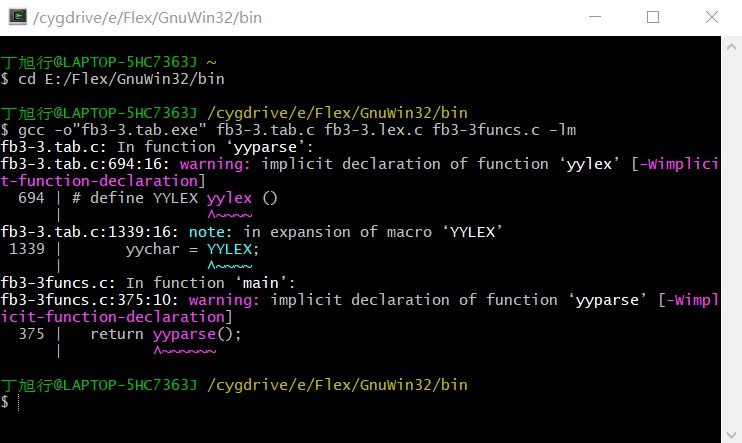


图8 输入命令

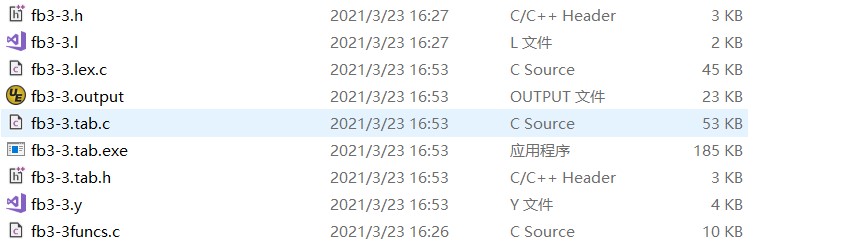


图9生成文件

(3) 再回到命令窗口输入fb3-3.tab.exe运行fb3-3.tab.exe文件，输入计算式示例，回车后执行计算任务并输出计算结果。

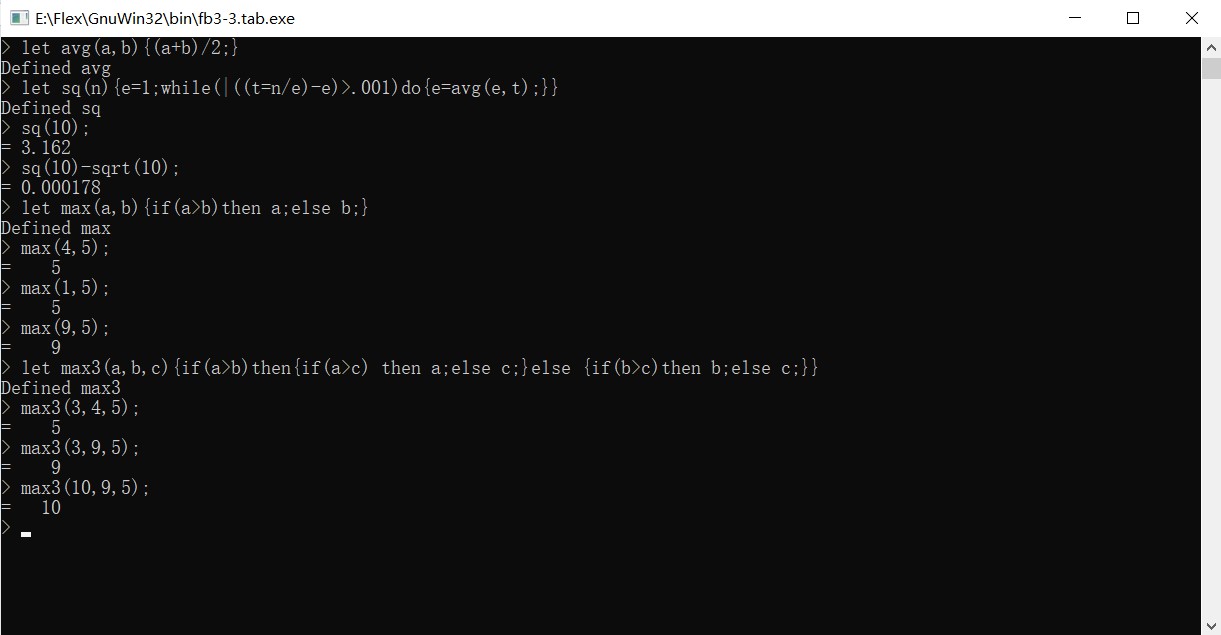


图10运行程序及结果显示

## 6.2 Linux环境

(1) 打开命令窗口，输入cd Desktop命令进入桌面目录，再依次输入bison -d fb3-3.y、flex -ofb3-3.lex.c fb3-3.l命令生成fb3-3.tab.c、fb3-3.tab.h和fb3-3.lex.c文件。继续输入命令cc fb3-3.tab.c fb3-3.lex.c fb3-3funcs.c -lm进行编译工作，生成out文件。

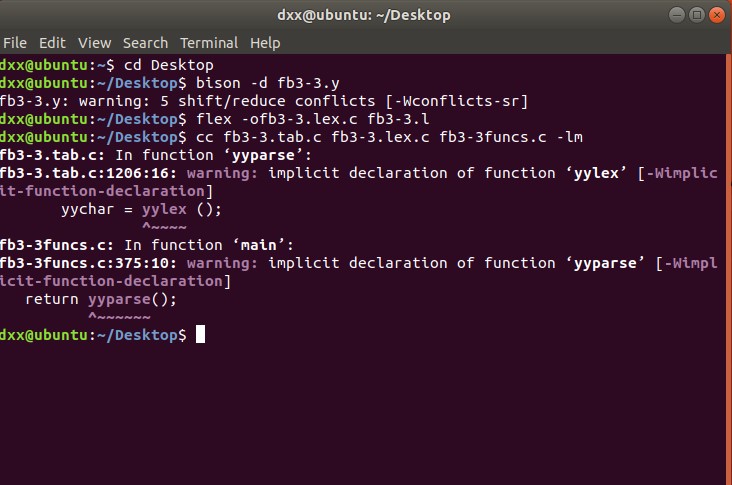


图11 输入命令



图12生成文件

(2) 在命令窗口输入./a.out，调用生成文件，输入计算式示例，回车后执行计算任务并输出计算结果。

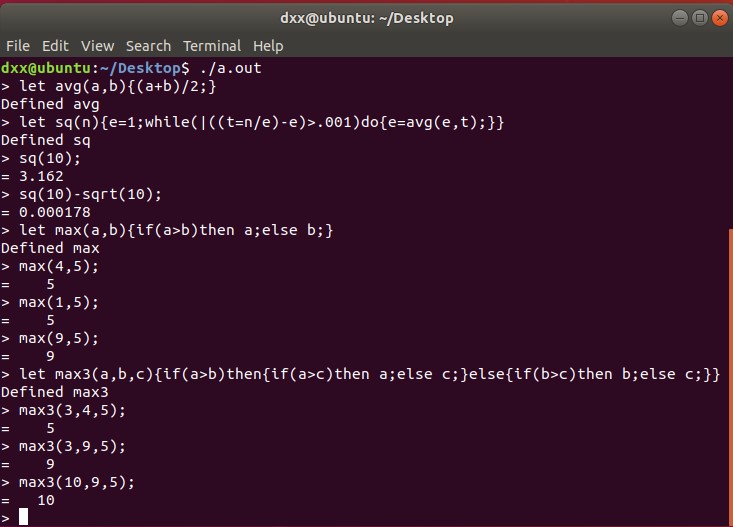


图13 运行程序及结果显示

# 7 分析抽象语法树构建过程

根据运行结果，现以“let sq(n){e=1; while (|((t=n/e)-e)>.001) do {e=avg(e,t);}}”为例分析其抽象语法树的构建过程：

1. 读入‘let’，语法分析器接受词法分析器传来的‘LET’，并移进。
2. 读入‘sq’，语法分析器接受词法分析器传来的‘NAME’，构建其符号节点，并移进。
3. 读入‘(’，语法分析器接受词法分析器传来的‘(’，并移进。
4. 读入’n’，语法分析器接受词法分析器传来的’NAME’， 构建其符号节点，并归约为symlist，构建节点类型为‘S’的参数列表节点。
5. 读入‘)’，语法分析器接受词法分析器传来的‘)’，并移进。
6. 读入‘{’，语法分析器接受词法分析器传来的‘{’，并移进。
7. 读入’e’，语法分析器接受词法分析器传来的’NAME’，构建其符号节点，并移进。
8. 读入‘=’，语法分析器接受词法分析器传来的‘=’，并移进。
9. 读入1，语法分析器接受词法分析器传来的NUMBER，并归约为exp，构建节点类型为‘K’的子节点，此时NAME ‘=’ exp归约为exp，构建节点类型为’=’的子节点，(7)中符号节点成为其符号节点，(9)中’K’类型节点为其值节点。最后exp归约为stmt。
10. 读入‘;’，语法分析器接受词法分析器传来的‘;’，并移进。
11. 读入‘while’，语法分析器接受词法分析器传来的‘WHILE’，并移进。
12. 读入‘(’，语法分析器接受词法分析器传来的‘(’，并移进。
13. 读入‘|’，语法分析器接受词法分析器传来的‘|’，并移进。
14. 读入‘(’，语法分析器接受词法分析器传来的‘(’，并移进。
15. 读入‘(’，语法分析器接受词法分析器传来的‘(’，并移进。
16. 读入’t’，语法分析器接受词法分析器传来的’NAME’，构建其符号节点，并移进。
17. 读入‘=’，语法分析器接受词法分析器传来的‘=’，并移进。
18. 读入’n’，语法分析器接受词法分析器传来的’NAME’， 构建其符号节点，并归约为exp，构建类型为’N’的符号引用节点，(17)中符号节点为其符号节点。
19. 读入‘/’，语法分析器接受词法分析器传来的‘/’，并移进。
20. 读入’e’，语法分析器接受词法分析器传来的’NAME’， 构建其符号节点，并归约为exp，构建类型为’N’的符号引用节点，(19)中符号节点为其符号节点。之后exp ‘/’ exp归约为exp，构建类型为’/’的子节点，(17)(19)中的‘N‘类型节点为其子节点。最后NAME ‘=’ exp归约为exp，构建类型为’=’的子节点，(15)中符号节点成为其符号节点，(19)中’/’类型节点为其值节点。
21. 读入‘)’，语法分析器接受词法分析器传来的‘)’，并移进。之后’(’ exp ‘)’归约为exp。
22. 读入‘-’，语法分析器接受词法分析器传来的‘-’，并移进。
23. 读入’e’，语法分析器接受词法分析器传来的’NAME’，构建其符号节点，并移进。之后归约为exp，构建类型为‘N‘的符号引用节点，(22)中符号节点为其符号节点。最后exp ’-‘ exp归约为exp，构建类型为’-‘的子节点，(19)中的’=‘节点和(22)中的’N’节点为其子节点。
24. 读入‘)’，语法分析器接受词法分析器传来的‘)’，并移进。之后’(’ exp ‘)’归约为exp。’|‘ exp归约为exp构建类型为’|’的子节点，(22)中的’-‘节点为其子节点。
25. 读入‘>’，语法分析器接受词法分析器传来的CMP，并移进。
26. 读入.001，语法分析器接受词法分析器传来的NUMBER，并归约为exp，构建节点类型为‘K’的子节点。exp CMP exp归约为exp构建类型为‘01‘的子节点，(23)中的’|‘节点(25)中的’K‘节点为其子节点。
27. 读入‘)’，语法分析器接受词法分析器传来的‘)’，并移进，之后’(’ exp ‘)’归约为exp。
28. 读入‘do’，语法分析器接受词法分析器传来的‘DO’，并移进。
29. 读入‘{’，语法分析器接受词法分析器传来的‘{’，并移进。
30. 读入’e’，语法分析器接受词法分析器传来的’NAME’，构建其符号节点，并移进。
31. 读入‘=’，语法分析器接受词法分析器传来的‘=’，并移进。
32. 读入’avg’，语法分析器接受词法分析器传来的’NAME’，构建其符号节点，并移进。
33. 读入‘(’，语法分析器接受词法分析器传来的‘(’，并移进。
34. 读入’e’，语法分析器接受词法分析器传来的’NAME’，构建其符号节点，并移进，再归约为exp，构建节点类型为‘N’的符号引用节点。
35. 读入‘,’，语法分析器接受词法分析器传来的‘,’，并移进。
36. 读入’t’，语法分析器接受词法分析器传来的’NAME’，构建其符号节点，并移进，再归约为exp，构建节点类型为‘N’的符号引用节点，最后归约为explist。之后exp ‘,‘ explist归约为explist构建类型为’L‘的子节点，(31)(33)中的’N‘类型节点为其子节点。
37. 读入‘)’，语法分析器接受词法分析器传来的‘)’，并移进。NAME ’(‘ explist ‘)’归约为exp，构建类型为‘C’的节点，(29)中符号节点(33)中‘L‘节点为其子节点。最后NAME ‘=’ exp归约为exp构建类型为’=‘的节点，进而归约为stmt。
38. 读入‘;’，语法分析器接受词法分析器传来的‘;’，并移进。stmt ‘;’ list归约为list。
39. 读入‘}’，语法分析器接受词法分析器传来的‘}’，并移进。之后WHILE exp DO ‘{’ list ‘}’规约为stmt构建类型为‘W’的节点。紧接着归约为list，最后stmt ‘;’ list归约为list，构建类型为‘L’的节点。
40. 读入‘}’，语法分析器接受词法分析器传来的‘}’，并移进。
41. 读入‘\n’，语法分析器接受词法分析器传来的‘EOL’，最后calclist LET NAME ’(’ symlist ’)‘ ’{‘ list ‘}’ EOL归约为calclist开始符号，定义一个函数给到(2)中的符号节点。

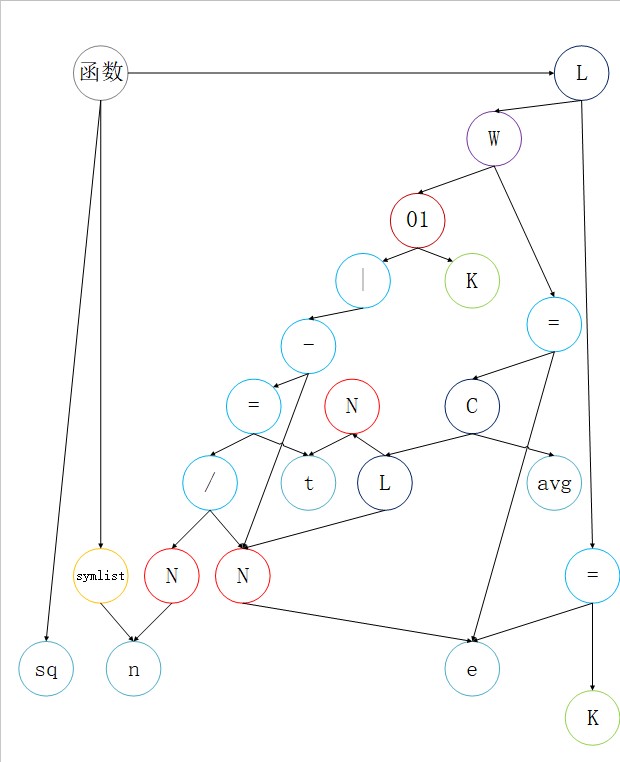


图14 抽象语法树1

根据运行结果，现以“max3(10,9,5);”为例分析其抽象语法树的构建过程：

1. 读入‘max3’，语法分析器接受词法分析器传来的‘NAME’， 构建其符号节点并移进。
2. 读入‘(’，语法分析器接受词法分析器传来的‘(’，并移进。
3. 读入10，语法分析器接受词法分析器传来的NUMBER，并归约为exp，构建节点类型为‘K’的子节点。
4. 读入‘,’，语法分析器接受词法分析器传来的‘,’，并移进。
5. 读入9，语法分析器接受词法分析器传来的NUMBER，并归约为exp，构建节点类型为‘K’的子节点。
6. 读入‘,’，语法分析器接受词法分析器传来的‘,’，并移进。
7. 读入5，语法分析器接受词法分析器传来的NUMBER，并归约为exp，构建节点类型为‘K’的子节点，之后归约为explist。紧接着exp ‘,’ explist归约为explist，构建节点类型为‘L’的节点，(5)中的‘K’节点和(7)中的‘K’节点为其子节点。最后exp ‘,’ explist又归约为explist，构建类型为‘L’的节点，(3)中的‘K’类型节点和(7)中的‘L’类型节点为其子节点。
8. 读入‘)’，语法分析器接受词法分析器传来的‘)’，并移进。
9. 读入‘;’，语法分析器接受词法分析器传来的‘;’，并移进。NAME ‘(’ explist ‘)’ ‘;’归约为exp，构建类型为‘C’的节点，(1)中符号节点为其符号节点，(7)中‘L’为其参数列表节点。最后exp归约为stmt。
10. 读入‘\n’，语法分析器接受词法分析器传来的‘EOL’，最后calclist stmt EOL归约为calclist开始符号，开始计算结果并打印结果数值。

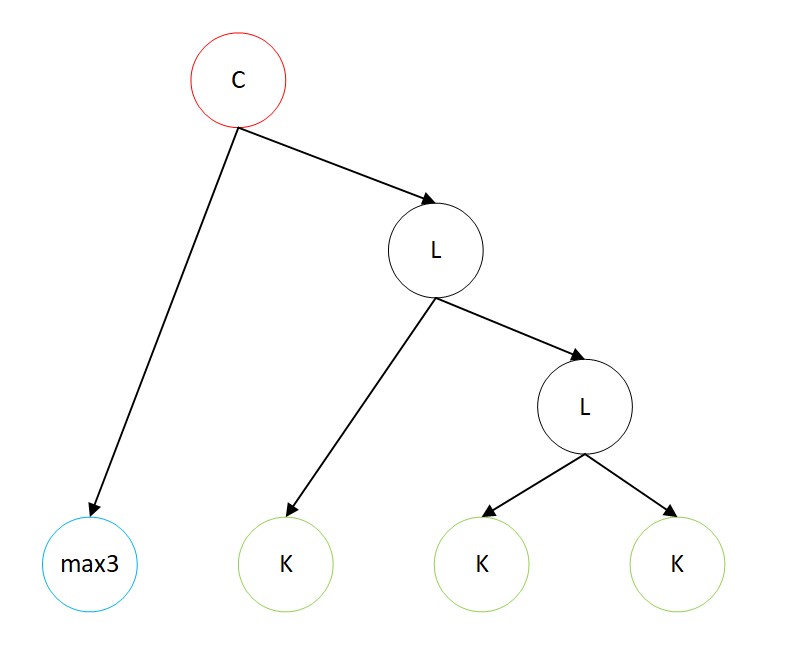


图15 抽象语法树2

# 8 实验总结

在本次实验中使用Flex和Bison在高级计算器的基础上对代码进行修改以满足比较合理化的编码规则，使对其的使用更加规范简便。通过阅读《Flex&Bison》第三章，加深对抽象语法树的用法的学习，熟练地掌握了抽象语法树的构造过程。虽然通过对代码的修改完成了规定的要求，但是修改过后的代码存在一定的移进归约冲突，即便bison默认的移进解决方式与我们的中心思想一致，能够化解冲突的发生，但是其文法本身还是存在问题，需要进一步修改文法结构，还有待提升。通过本次修改代码、改进程序，深刻理解了抽象语法树的用法以及Flex和Bison的代码语法，修改成功的成就感让我对后面的实验更加有信心。