**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 Flex/Bison综合实验1

报告时间 2021年3月27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**

[实验1基本计算器 1](#_Toc68253370)

[1.1 实验目的 1](#_Toc68253371)

[1.2 实验内容 1](#_Toc68253372)

[1.3 实验要求 1](#_Toc68253373)

[1.4 代码文件分析 1](#_Toc68253374)

[1.4.1 fb3-1.h文件分析 1](#_Toc68253375)

[1.4.2 fb3-1.l文件分析 2](#_Toc68253376)

[1.4.3 fb3-1.y文件分析 2](#_Toc68253377)

[1.5 节点分析 3](#_Toc68253378)

[1.5.1 普通节点 3](#_Toc68253379)

[1.5.2 数值节点 4](#_Toc68253380)

[1.6 实验步骤 4](#_Toc68253381)

[1.6.1 Windows环境 4](#_Toc68253382)

[1.6.2 Linux环境 6](#_Toc68253383)

[1.7 分析抽象语法树构建过程 8](#_Toc68253384)

[1.8 实验总结 9](#_Toc68253385)

[实验2高级计算器 10](#_Toc68253386)

[2.1 实验目的 10](#_Toc68253387)

[2.2 实验内容 10](#_Toc68253388)

[2.3 实验要求 10](#_Toc68253389)

[2.4 高级计算器功能 10](#_Toc68253390)

[2.5 代码文件分析 11](#_Toc68253391)

[2.5.1 fb3-2.h文件分析 11](#_Toc68253392)

[2.5.2 fb3-2.y文件分析 12](#_Toc68253393)

[2.5.3 fb3-2.l文件分析 12](#_Toc68253394)

[2.5.4 fb3-2funcs.c文件分析 13](#_Toc68253395)

[2.6 节点分析 15](#_Toc68253396)

[2.6.1 比较运算节点 15](#_Toc68253397)

[2.6.2 符号节点 16](#_Toc68253398)

[2.6.3 形式参数列表节点 16](#_Toc68253399)

[2.6.4 语句列表节点 16](#_Toc68253400)

[2.6.5 内置函数节点 17](#_Toc68253401)

[2.6.6 用户自定义函数节点 17](#_Toc68253402)

[2.6.7 控制语句节点 17](#_Toc68253403)

[2.6.8 引用节点 18](#_Toc68253404)

[2.6.9 赋值节点 18](#_Toc68253405)

[2.7 实验步骤 18](#_Toc68253406)

[2.7.1 Windows环境 18](#_Toc68253407)

[2.7.2 Linux环境 22](#_Toc68253408)

[2.8 分析抽象语法树构建过程 23](#_Toc68253409)

[2.9 实验总结 26](#_Toc68253410)

[实验3修改高级计算器 27](#_Toc68253411)

[3.1 实验目的 27](#_Toc68253412)

[3.2 实验内容 27](#_Toc68253413)

[3.3 实验要求 27](#_Toc68253414)

[3.4 高级计算器功能 27](#_Toc68253415)

[3.5 相关代码修改 28](#_Toc68253416)

[3.6 实验步骤 30](#_Toc68253417)

[3.6.1 Windows环境 30](#_Toc68253418)

[3.6.2 Linux环境 31](#_Toc68253419)

[3.7 分析抽象语法树构建过程 33](#_Toc68253420)

[3.8 实验总结 37](#_Toc68253421)

[实验4可视化计算器 38](#_Toc68253422)

[4.1 实验目的 38](#_Toc68253423)

[4.2 实验内容 38](#_Toc68253424)

[4.3 高级计算器功能 38](#_Toc68253425)

[4.4 实现工具 38](#_Toc68253426)

[4.5 整体思路 39](#_Toc68253427)

[4.6 设计原理 39](#_Toc68253428)

[4.6.1 基于Flex&Bison编译器整体原理 39](#_Toc68253429)

[4.6.2 C#调用后端实现与前端的交互 41](#_Toc68253430)

[4.7 实现过程 42](#_Toc68253431)

[4.7.1 利用Flex和Bison实现词法分析和语法分析 42](#_Toc68253432)

[4.7.2 编写并修改前端代码 44](#_Toc68253433)

[4.8 设计成果展示 47](#_Toc68253434)

[4.9 实验心得与总结 49](#_Toc68253435)

[4.10 实验补充—python语言的dll文件调用 49](#_Toc68253436)

# 实验1基本计算器

## 1.1 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。

## 1.2 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。

## 1.3 实验要求

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

(2) 阅读fb3-1.y、fb3-1.l、fb3-1funcs.c、fb3-1.h文件；

(3) 撰写实验报告，结合实验结果，如(1+2)-(2\*6)、1+2-3\*2/5，给出抽象语法树的构建过程；

(4) 提交报告和实验代码。

## 1.4 代码文件分析

### 1.4.1 fb3-1.h文件分析

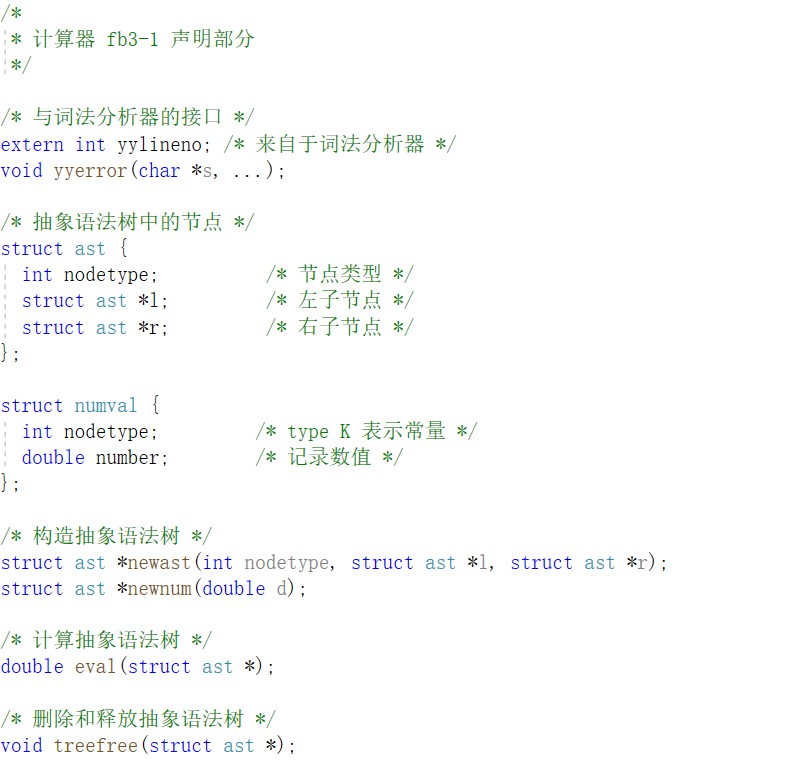


图1 fb3-1.h文件

### 1.4.2 fb3-1.l文件分析

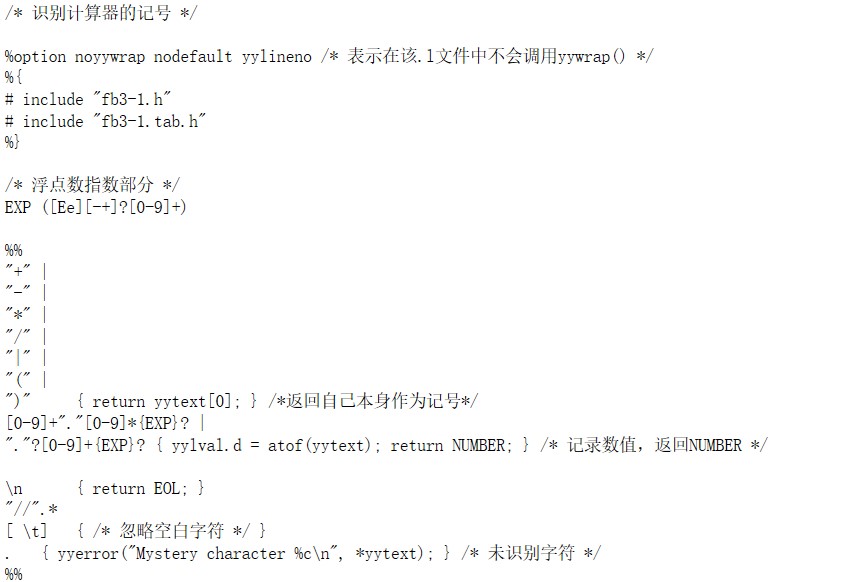


图2 fb3-1.l文件

### 1.4.3 fb3-1.y文件分析



图3 fb3-1.y文件1

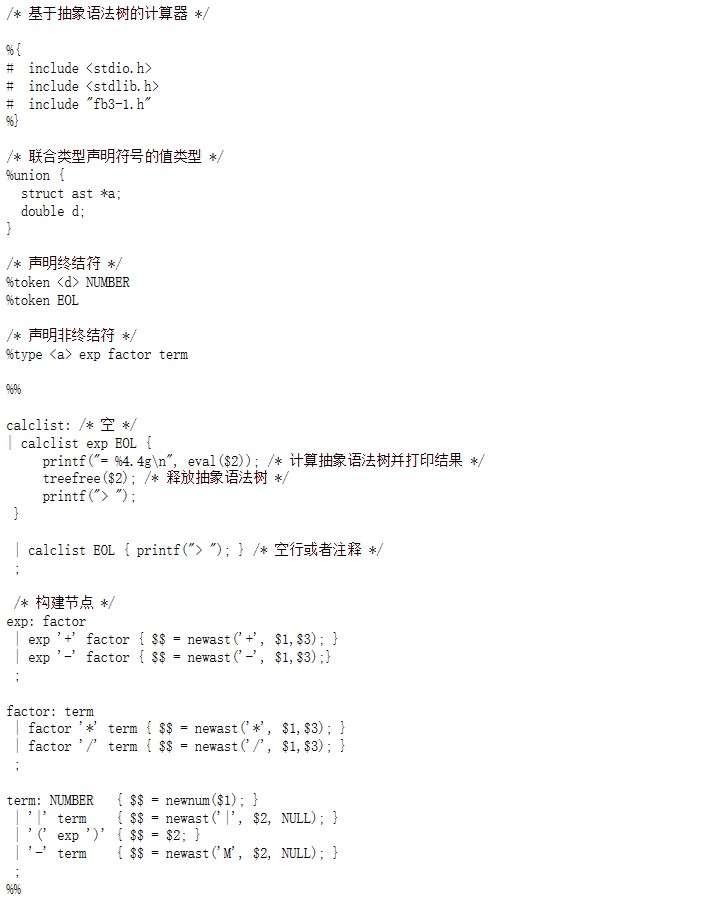


图4 fb3-1.y文件2

## 1.5 节点分析

### 1.5.1 普通节点

普通节点包含有自身节点类型值以及其左右子节点：

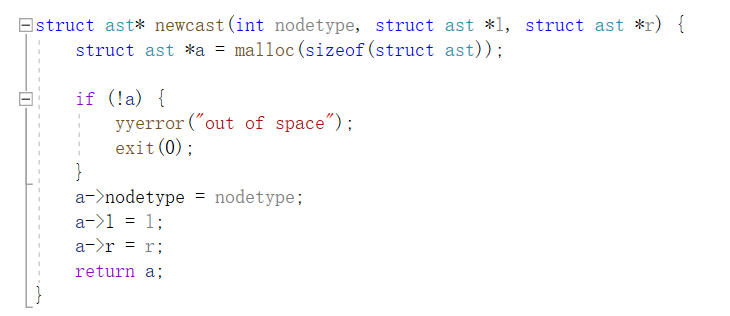


图5 构建普通节点

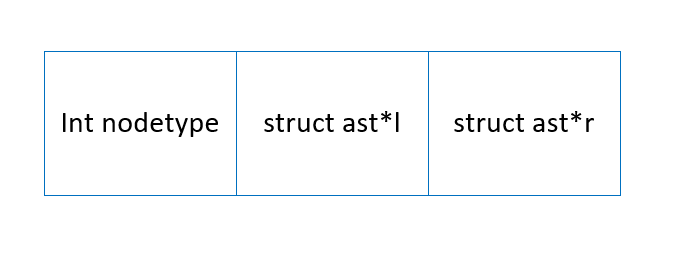


图6 普通节点类型

### 1.5.2 数值节点

数值节点的节点类型为’K’，自身有double类型的数值：

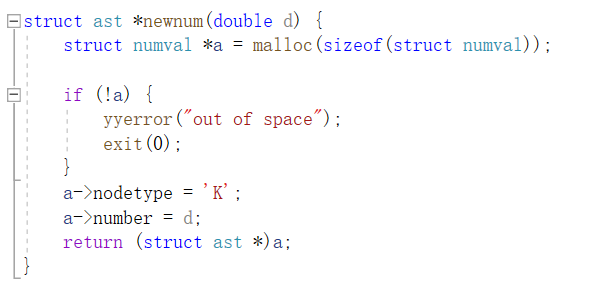


图7 构建数值节点

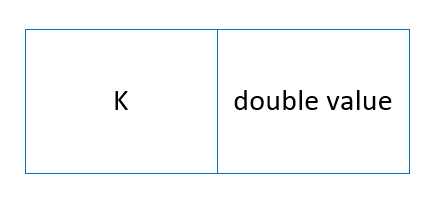


图8 数值节点类型

## 1.6 实验步骤

### 1.6.1 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 fb3-1.y、flex -ofb3-1.lex.c fb3-1.l，生成fb3-1.tab.h、fb3-1.tab.c和fb3-1.lex.c文件。

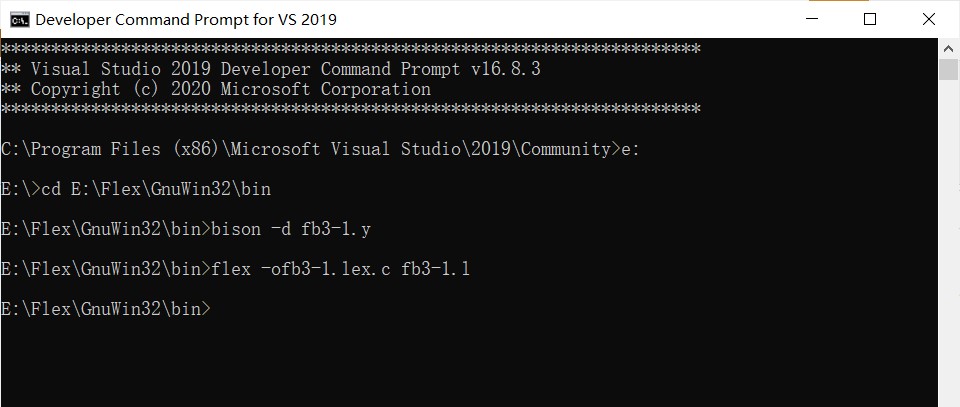


图9 输入命令

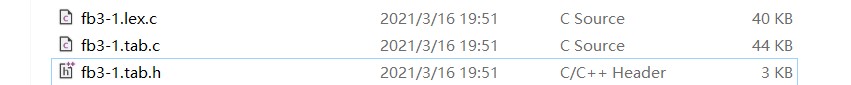


图10 生成文件

(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”fb3-1.tab.exe” fb3-1.tab.c fb3-1.lex.c fb3-1funcs.c -lm生成其对应的fb3-1.tab.exe文件。

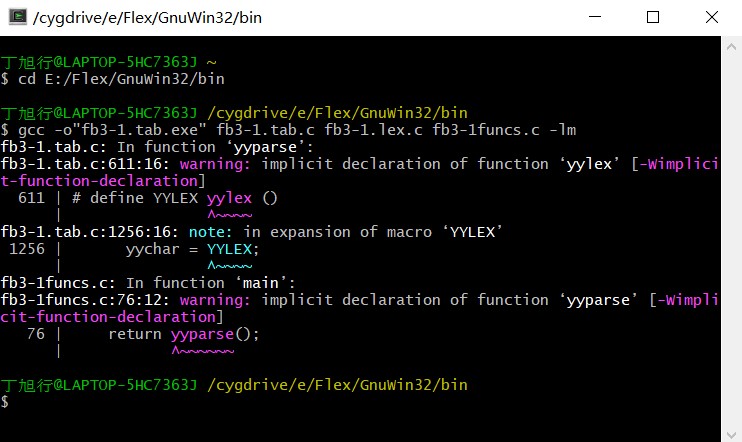


图11 输入命令

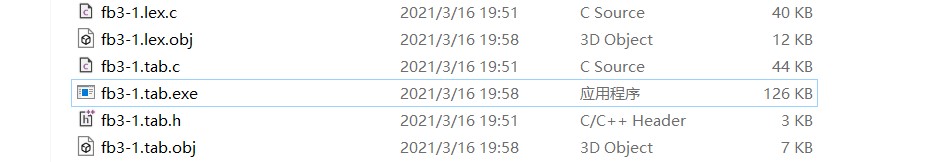


图12 生成文件

(3) 再回到命令窗口输入fb3-1.tab.exe运行fb3-1.tab.exe文件，输入计算式示例，回车后输出计算结果。

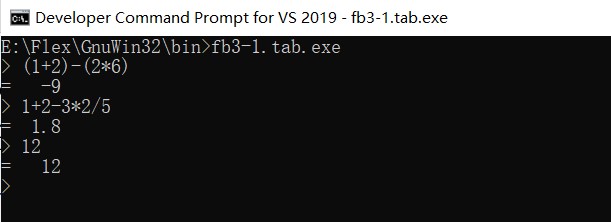


图13 运行程序及结果显示

### 1.6.2 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 fb3-1.y、flex -ofb3-1.lex.c fb3-1.l命令生成fb3-1.tab.c、fb3-1.tab.h和fb3-1.lex.c文件。

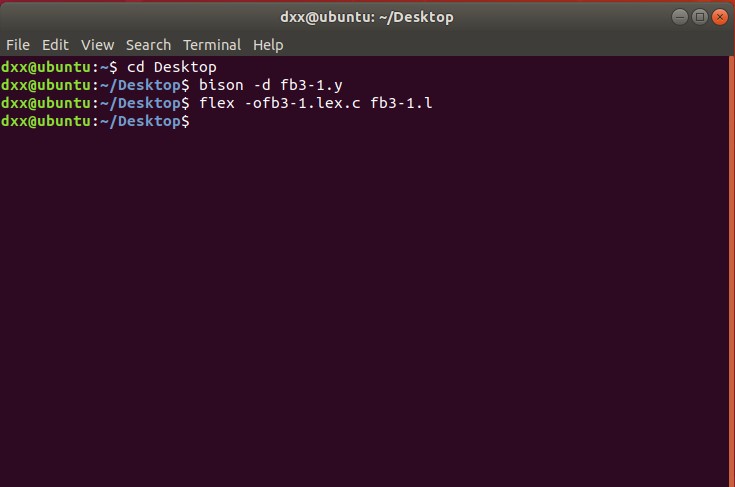


图14 输入命令

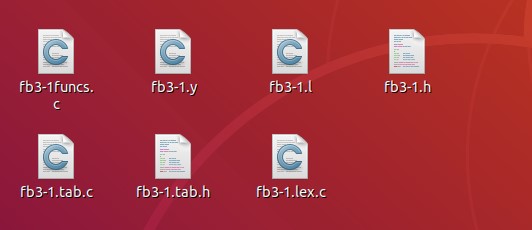


图15 生成文件

(2) 继续输入命令cc fb3-1.tab.c fb3-1.lex.c fb3-1funcs.c -lm进行编译工作，生成out文件。

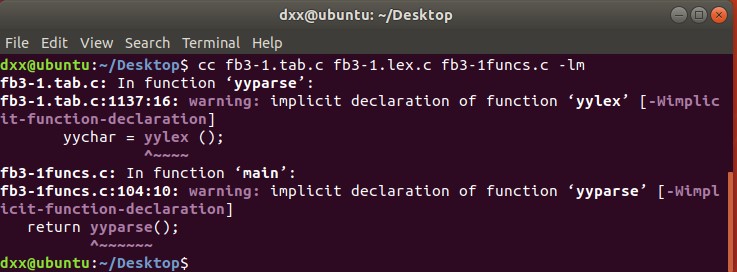


图16 输入命令

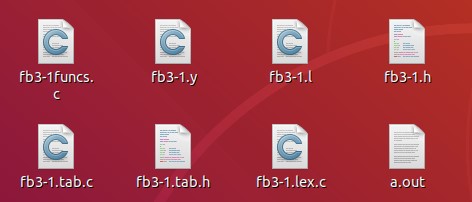


图17 生成文件

(3) 在命令窗口输入./a.out，调用生成文件，输入计算式示例，回车后输出计算结果。

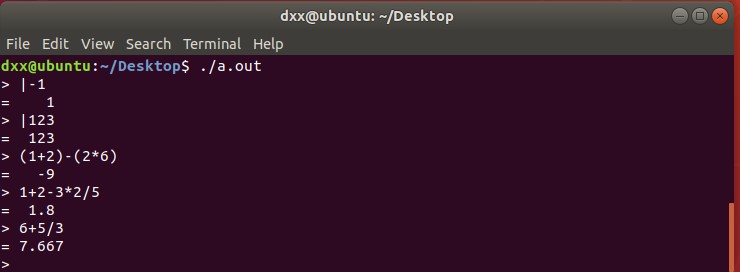


图18 运行程序及结果显示

## 1.7 分析抽象语法树构建过程

根据运行结果，现以“(1+2)-(2\*6)”为例分析其抽象语法树的构建过程：

(1) 读入‘(’，语法分析器接受词法分析器传来的‘(’，并移进。

(2) 读入1，语法分析器接受词法分析器传来的NUMBER，并归约为term，构建节点类型为‘K’的子节点，进而归约为factor，最后归约为exp。

(3) 读入‘+’，语法分析器接受词法分析器传来的‘+’，并移进。

(4) 读入2，语法分析器接受词法分析器传来的NUMBER，并归约为term，构建节点类型为‘K’的子节点，进而归约为factor。最后exp ‘+’ factor归约为exp，构建节点类型为‘+’的子节点，(2)(4)中的‘K’子节点成为其左右子节点。

(5) 读入‘)’，语法分析器接受词法分析器传来的‘)’，‘(’ exp ‘)’归约为term，进而归约为factor，最后归约为exp。

(6) 读入‘-’，语法分析器接受词法分析器传来的‘-’，并移进。

(7) 读入‘(’，语法分析器接受词法分析器传来的‘(’，并移进。

(8) 读入2，语法分析器接受词法分析器传来的NUMBER，并归约为term，构建节点类型为‘K’的子节点，进而归约为factor。

(9) 读入‘\*’，语法分析器接受词法分析器传来的‘\*’，并移进。

(10) 读入6，语法分析器接受词法分析器传来的NUMBER，并归约为term，构建节点类型为‘K’的子节点。最后factor ‘\*’ term归约为factor，构建节点类型为‘\*’的子节点，(8)(10)中的‘K’子节点成为其左右子节点，之后factor归约为exp。

(11) 读入‘)’，语法分析器接受词法分析器传来的‘)’，‘(’ exp ‘)’归约为term，进而归约为factor。之后exp ‘-’ factor归约为exp，构建节点类型为‘-’的子节点，(4)(10)中的‘+’子节点和‘\*’子节点成为其左右子节点。

(12) 读入‘\n’，语法分析器接受词法分析器传来的‘EOL’，最后calclist exp EOL归约为calclist开始符号，开始计算结果并打印。

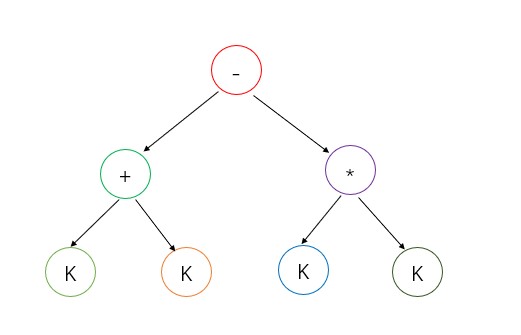


图19 抽象语法树

## 1.8 实验总结

在本次实验中使用Flex和Bison开发了一个具有基本功能的桌面计算器，支持四则运算，取绝对值，括号和负号，使它成为一个虽然短小但是具有现实意义的编译器。通过阅读《Flex&Bison》第三章，学习了抽象语法树的用法，掌握了抽象语法树的构造过程。但是，在实验的过程中还是遇到了一个小问题，VS的cl编译器无法执行-lm命令连接数字库，所以我改用了Cygwin的gcc编译器，最后成功生成了exe文件。由于有了上次实验阅读大段代码的经验，本次的代码理解起来更加的轻松，也为以后的实验打下了基础。

# 实验2高级计算器

## 2.1 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。

## 2.2 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。

## 2.3 实验要求

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

(2) 阅读fb3-2.y、fb3-2.l、fb3-2funcs.c、fb3-2.h；

(3) 增加新的内置函数pow(a,n)计算一个数a的n次方；

(4) 撰写实验报告，结合实验结果，如sqrt(10)和sq(10)，给出抽象语法树的构建过程；

(5) 提交报告和实验代码。

## 2.4 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)

## 2.5 代码文件分析

### 2.5.1 fb3-2.h文件分析

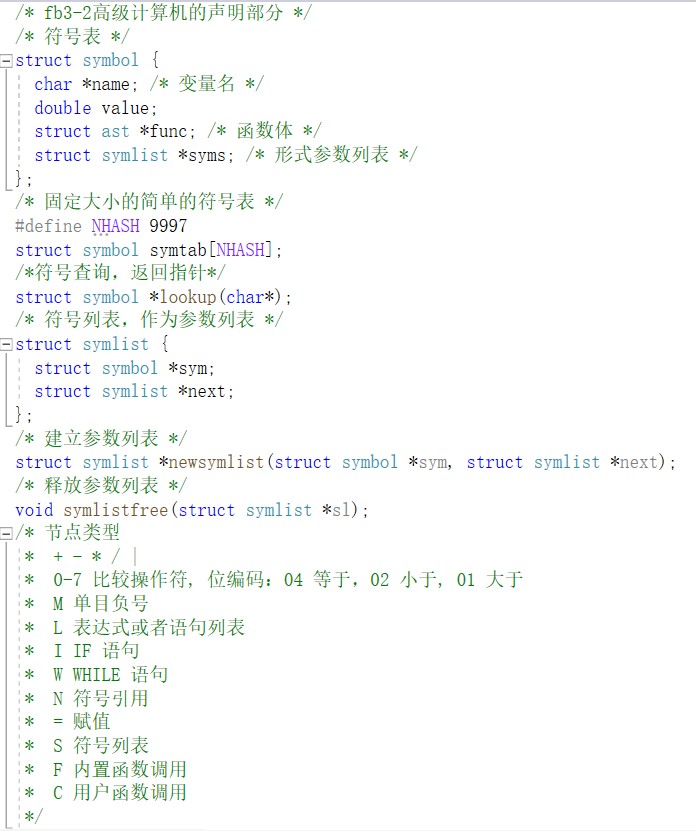


图20 fb3-2.h文件1 图21 fb3-2.h文件2



图22 fb3-2.h文件3

### 2.5.2 fb3-2.y文件分析

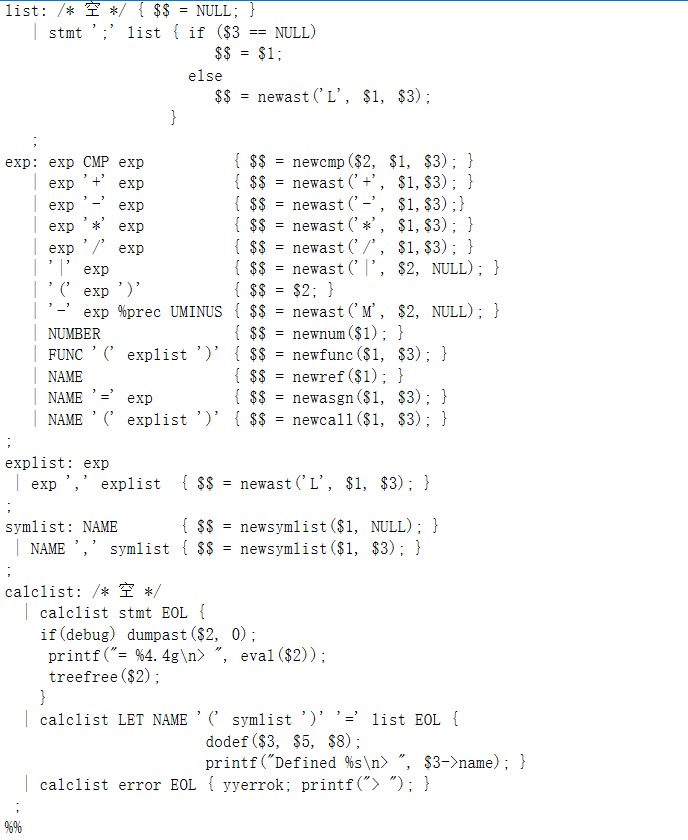
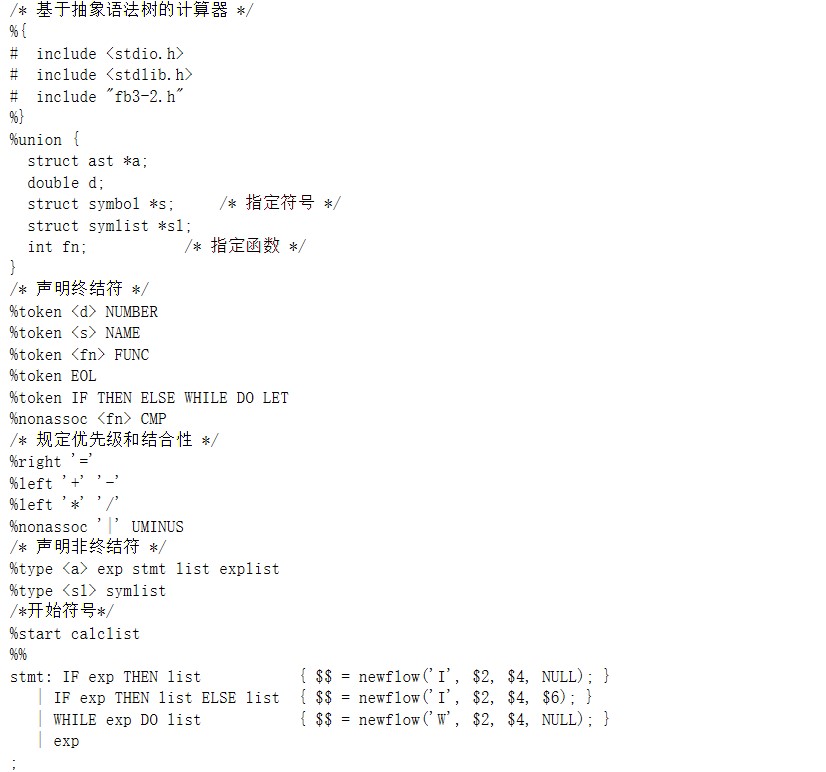


图23 fb3-2.y文件1 图24 fb3-2.y文件2

### 2.5.3 fb3-2.l文件分析

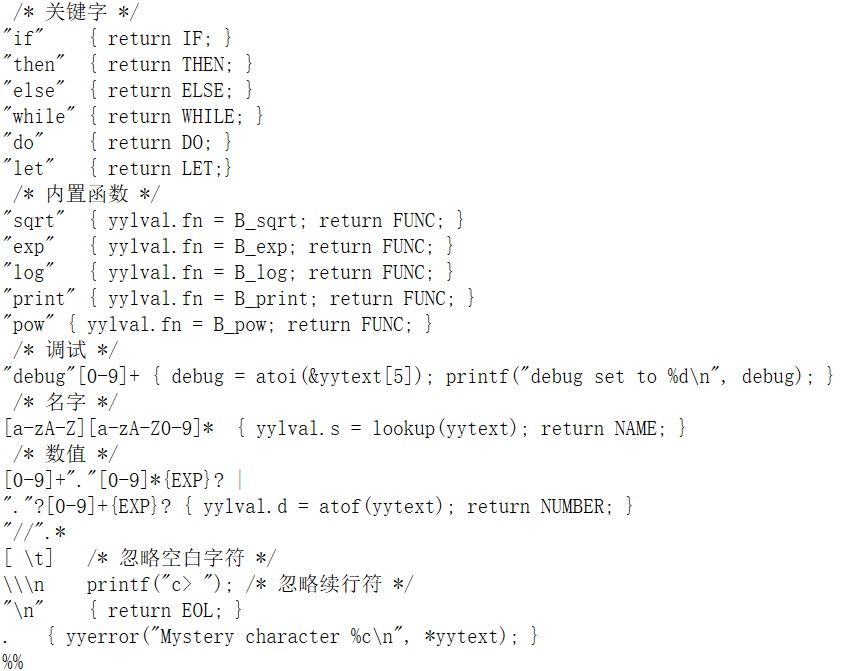
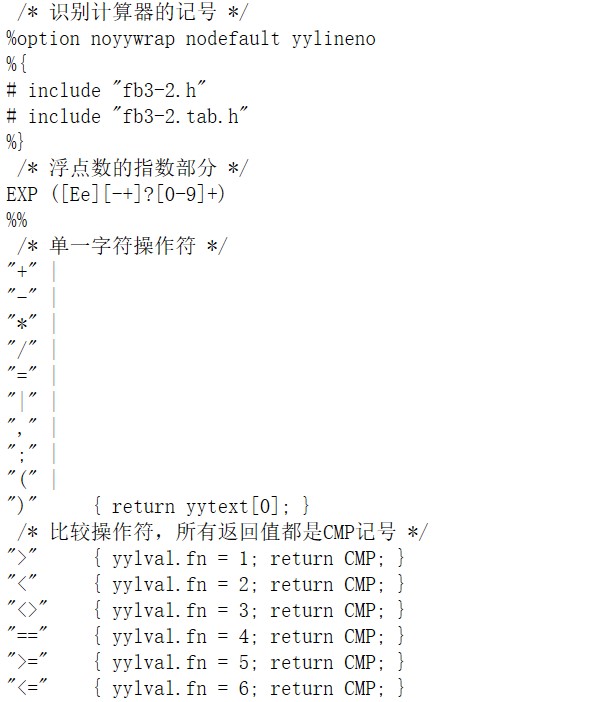


图25 fb3-2.l文件1 图26 fb3-2.l文件2

### 2.5.4 fb3-2funcs.c文件分析

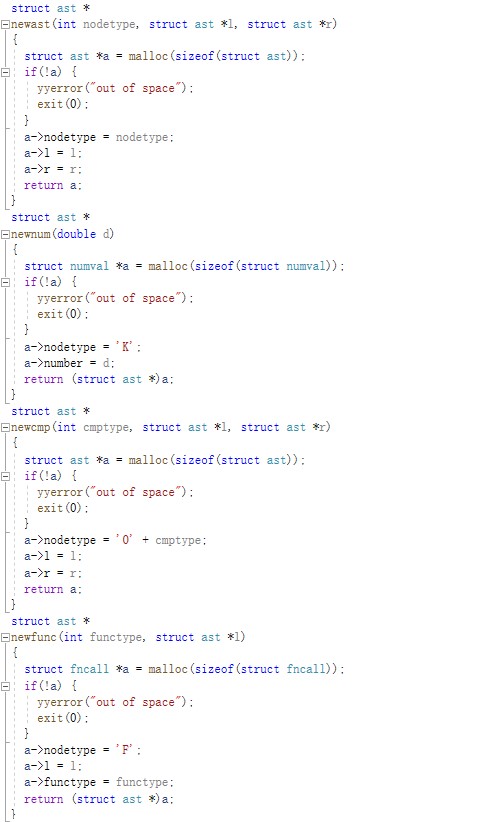


图27 fb3-2funcs.c文件1 图28 fb3-2funcs.c文件2

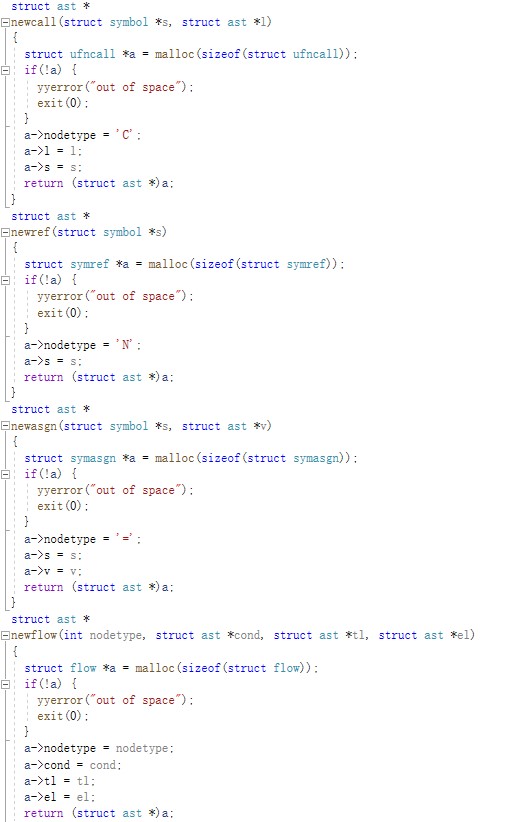


图29 fb3-2funcs.c文件3 图30 fb3-2funcs.c文件4

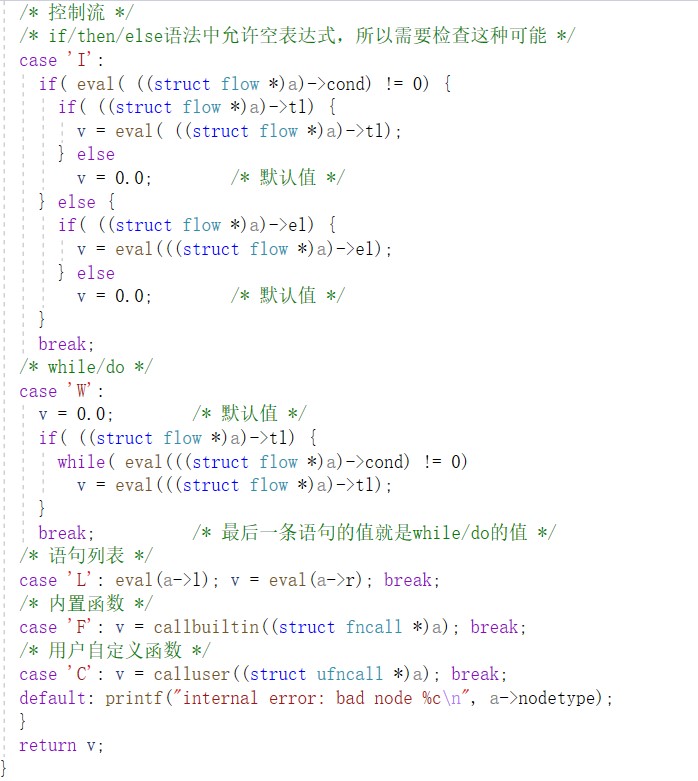
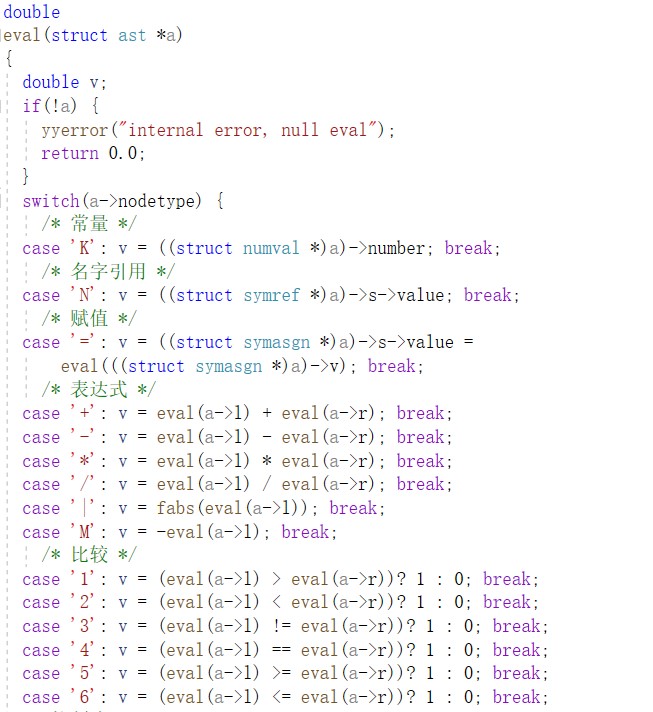


图31 fb3-2funcs.c文件5 图32 fb3-2funcs.c文件6

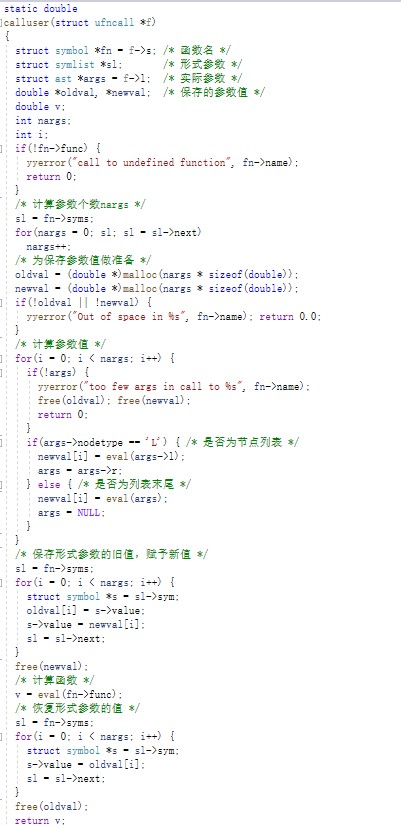
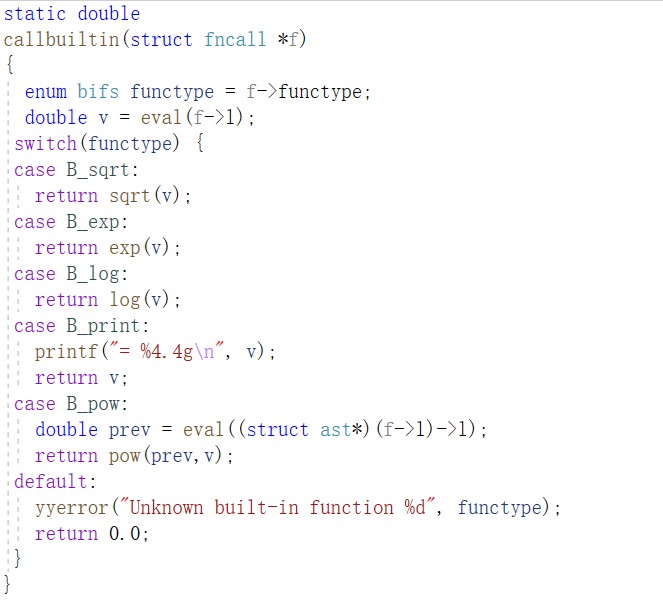


图33 fb3-2funcs.c文件7 图34 fb3-2funcs.c文件8

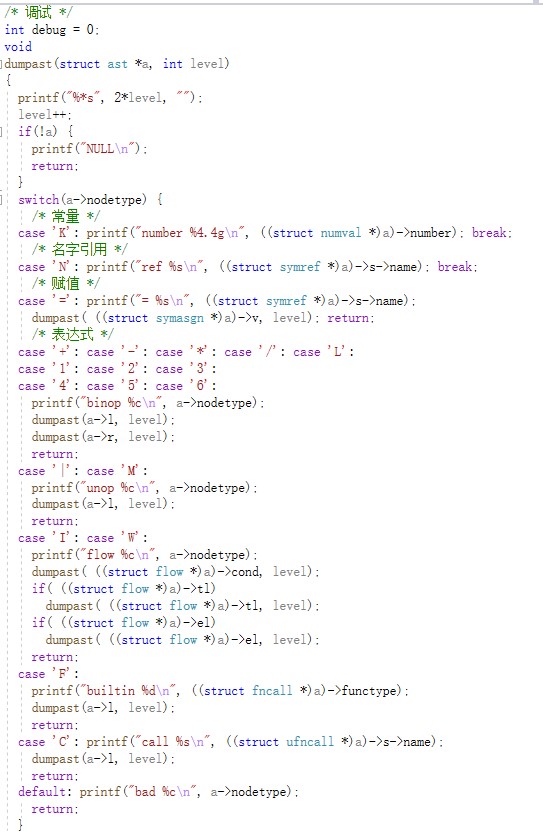
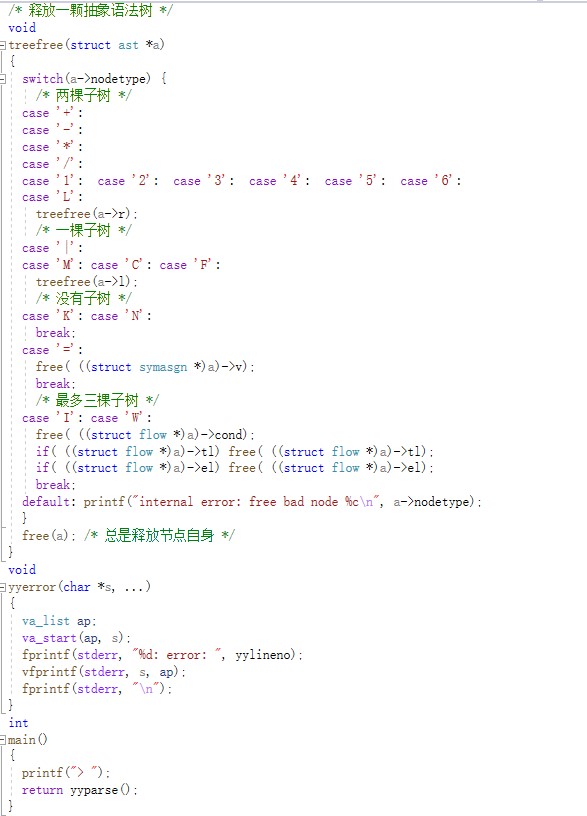


图35 fb3-2funcs.c文件9 图36 fb3-2funcs.c文件10

## 2.6 节点分析

关于普通节点ast和数值节点numval的分析与实验五报告内分析相同在此不再花费篇幅赘述。

### 2.6.1 比较运算节点

比较运算节点与普通节点相似，但其类型为内置的比较运算符的编号值，其左右子节点指向其参与比较的两个表达式：

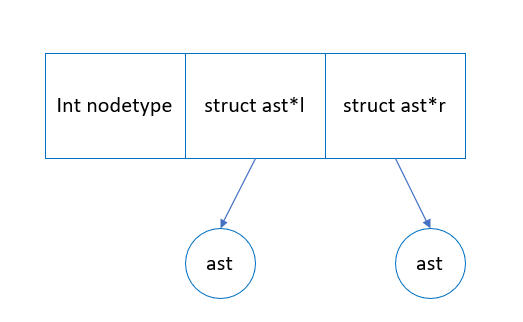


图37 比较运算节点类型

### 2.6.2 符号节点

词法分析器对于用户输入的标识符会在其符号表中寻找是否已存在同名符号，若为存在则为其新建一个符号节点，否则直接引用已存在符号节点。符号节点包含标识符名称、数值、参数列表节点和普通节点：

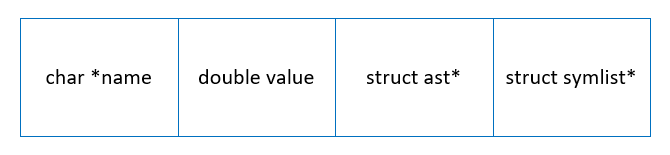


图38 符号节点类型

### 2.6.3 形式参数列表节点

形式参数列表节点包含一个符号节点指针，其指向第一个形式参数，还包含一个形式参数列表节点的指针，其指向下一个形式参数列表：

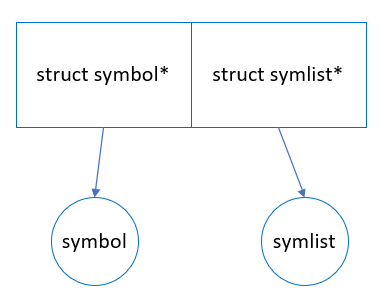


图39 形式参数列表节点类型

### 2.6.4 语句列表节点

语句列表节点和普通节点相似，只是其节点类型为’L’，左右子节点指向相邻的两条语句：

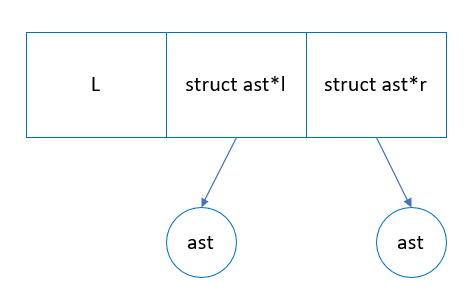


图40 语句列表节点类型

### 2.6.5 内置函数节点

内置函数节点类型为’F’，内含指明函数类型的整型数和指向参数的指针：

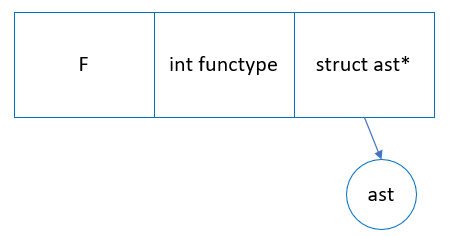


图41 内置函数节点类型

### 2.6.6 用户自定义函数节点

用户自定义函数节点的节点类型为’C’，内含指向符号节点的指针来寻找函数名和指向参数的指针：

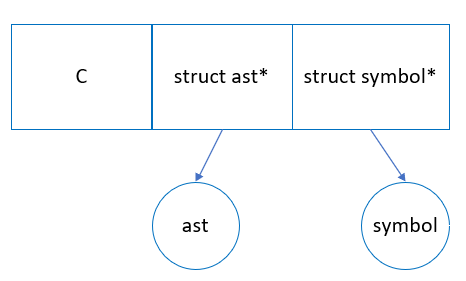


图42 用户自定义函数节点类型

### 2.6.7 控制语句节点

控制语句节点的节点类型可以为’I’或者’W’，分别对应的if\_then\_else语句和while\_do语句，其包含三个指向子节点的指针，分别指向条件、分支操作1、分支操作2，当然分支操作2只有在else情况下不为空：

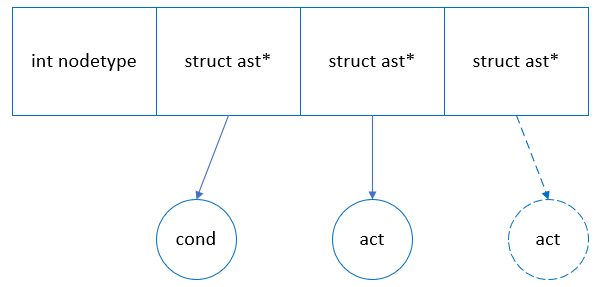


图43 控制语句节点类型

### 2.6.8 引用节点

引用节点类型为’N’，内含指向一个符号节点的指针，即是对其的引用：

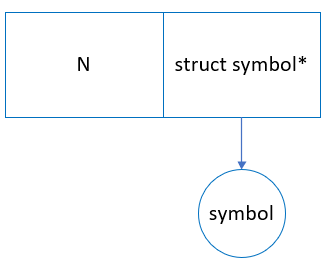


图44 引用节点类型

### 2.6.9 赋值节点

赋值节点类型为’=’，内含指向一个符号节点的指针，即是对其的赋值，还有一个指向普通节点的指针，其结果值赋值给符号节点代表的符号：

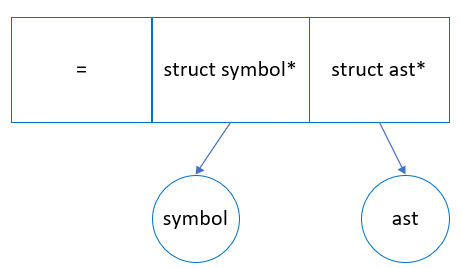


图45 赋值节点类型

## 2.7 实验步骤

### 2.7.1 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 fb3-2.y、flex -ofb3-2.lex.c fb3-2.l，生成fb3-2.tab.h、fb3-2.tab.c和fb3-2.lex.c文件。

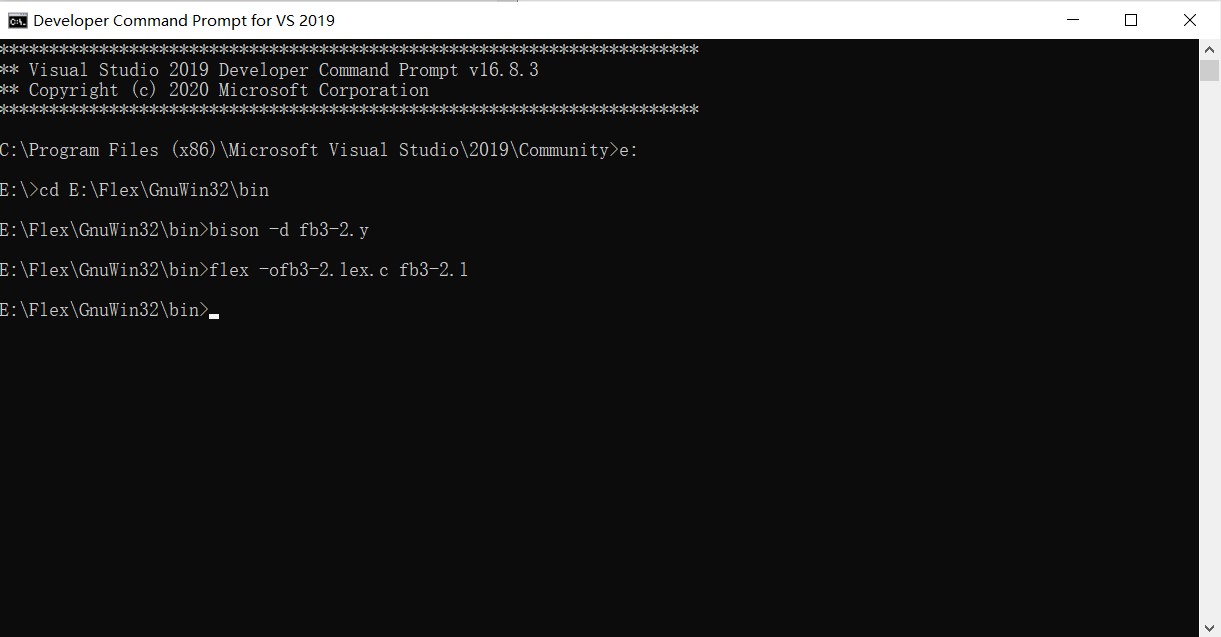


图46 输入命令

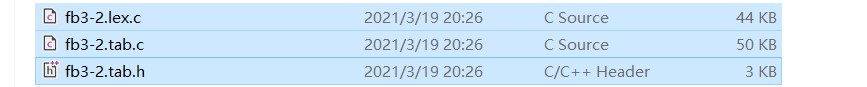


图47 生成文件

(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”fb3-2.tab.exe” fb3-2.tab.c fb3-2.lex.c fb3-2funcs.c -lm生成其对应的fb3-2.tab.exe文件。

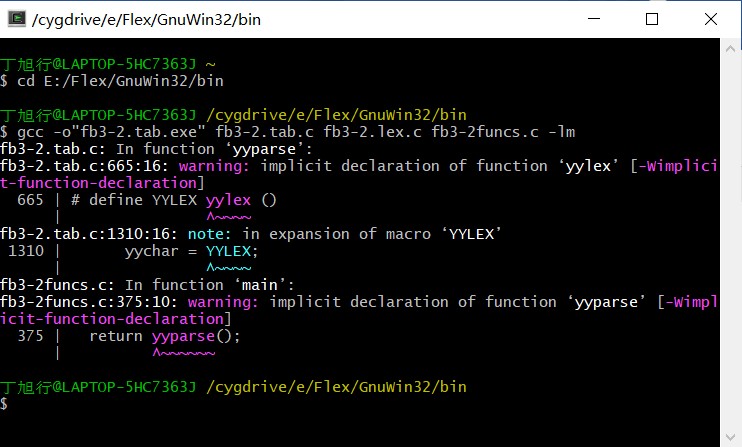


图48 输入命令

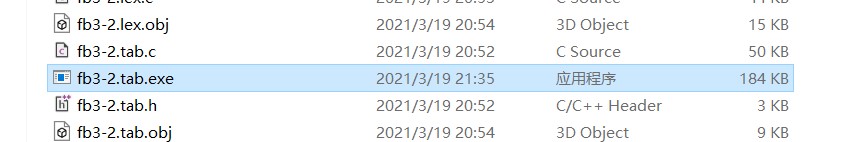


图49 生成文件

(3) 再回到命令窗口输入fb3-2.tab.exe运行fb3-2.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。

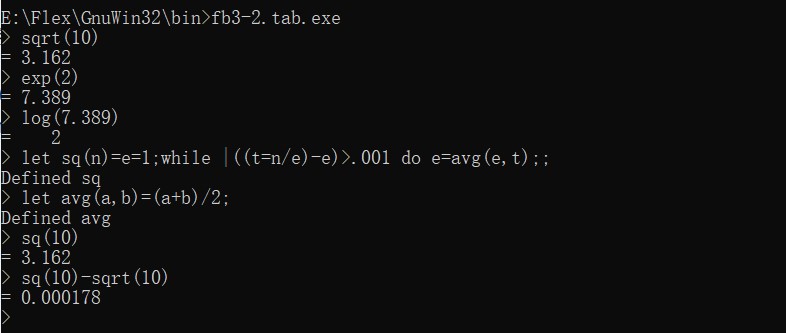


图50 运行程序及结果显示

(4) 为了新增一个新的内置函数pow(a,n)来计算一个数a的n次方，则需要增加一些代码。在fb3-2.h声明文件内增加pow(a,n)内置函数标识；在fb3-2.l文件中增加对字符串“pow”的识别，返回FUNC；在fb3-2funcs.c文件中对于内置函数的计算函数中增加对于pow(a,n)函数的处理代码。

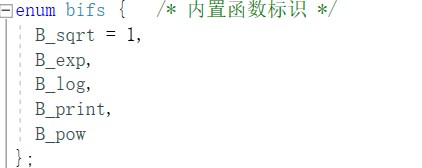


图51 fb3-2.h声明文件改动

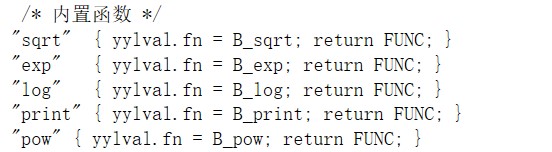


图52 fb3-2.l文件代码改动

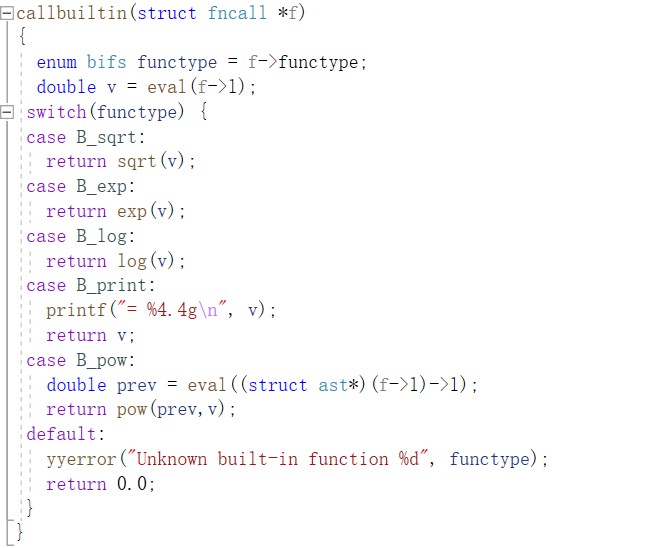


图53 fb3-2funcs.c文件代码改动

(5) 代码修改后，重复上述步骤重新生成exe文件并运行测试pow(a,n)函数。

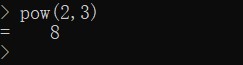


图54 测试内置函数

### 2.7.2 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 fb3-2.y、flex -ofb3-2.lex.c fb3-2.l命令生成fb3-2.tab.c、fb3-2.tab.h和fb3-2.lex.c文件。

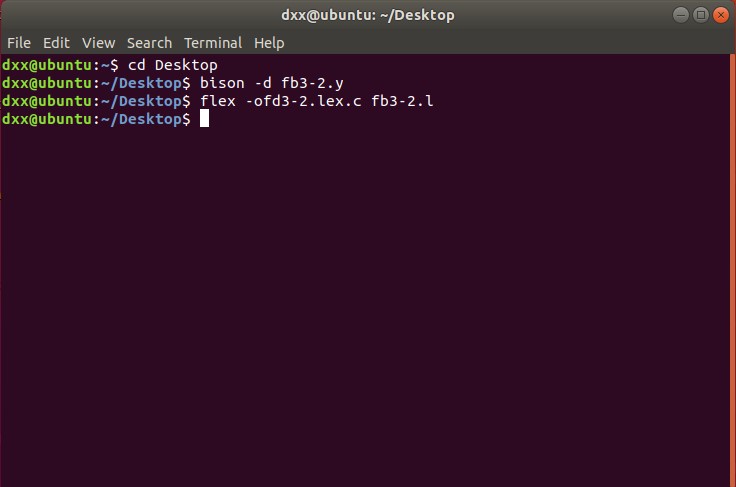


图55 输入命令



图56 生成文件

(2) 继续输入命令cc fb3-2.tab.c fb3-2.lex.c fb3-2funcs.c -lm进行编译工作，生成out文件。

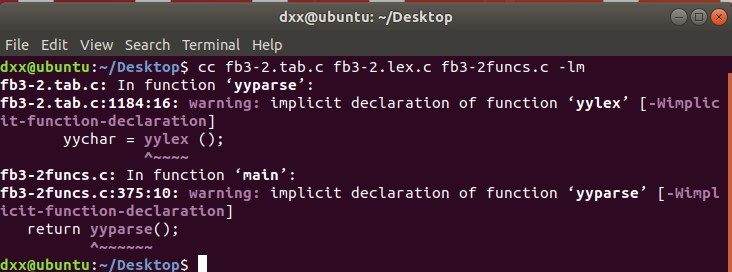


图57 输入命令



图58 生成文件

(3) 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。

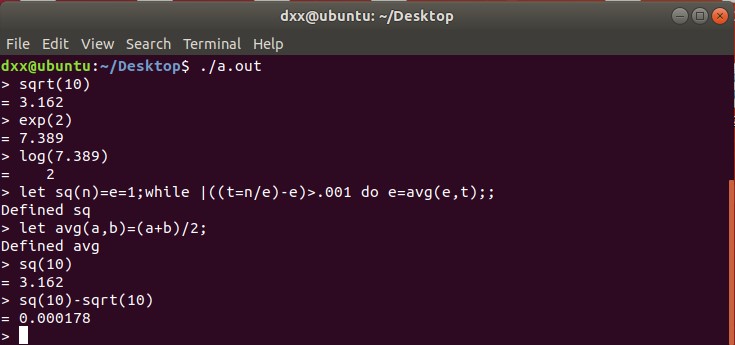


图59 运行程序及结果显示

(4) 为了新增一个新的内置函数pow(a,n)来计算一个数a的n次方，其具体过程与Windows环境下一致，在此不再赘述，下图为测试结果：

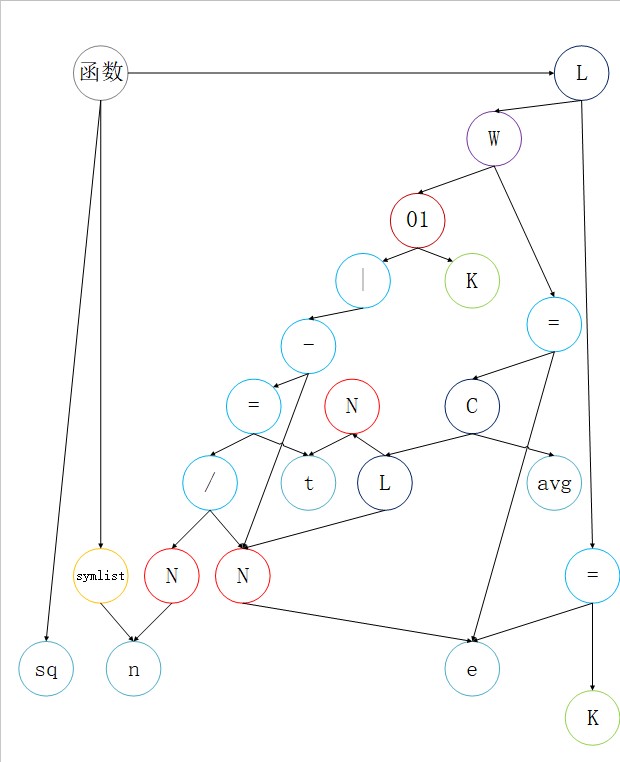


图60 测试内置函数

## 2.8 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’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。
16. 读入‘=’，语法分析器接受词法分析器传来的‘=’，并移进。
17. 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。
18. 读入‘/’，语法分析器接受词法分析器传来的‘/’，并移进。
19. 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。
20. 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。
21. 读入‘-’，语法分析器接受词法分析器传来的‘-’，并移进。
22. 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。
23. 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。
24. 读入‘>’，语法分析器接受词法分析器传来的CMP，并移进。
25. 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。
26. 读入‘do’，语法分析器接受词法分析器传来的‘DO’，并移进。
27. 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。
28. 读入‘=’，语法分析器接受词法分析器传来的‘=’，并移进。
29. 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。
30. 读入‘(’，语法分析器接受词法分析器传来的‘(’，并移进。
31. 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。
32. 读入‘,’，语法分析器接受词法分析器传来的‘,’，并移进。
33. 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。
34. 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。
35. 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最后WHILE exp DO list规约为stmt构建类型为‘W’的节点。
36. 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，最后stmt ’;‘ list规约为list构建类型为’L‘的节点。
37. 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。



图表 61 抽象语法树1

根据运行结果，现以“sqrt(10)”为例分析其抽象语法树的构建过程：

1. 读入‘sqrt’，语法分析器接受词法分析器传来的‘FUNC’，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，进而归约为explist。
4. 读入‘)’，语法分析器接受词法分析器传来的‘)’，FUNC ‘(’ explist ‘)’归约为exp，并构建节点类型为‘F’的子节点，(3)中的‘K’类型子节点成为其左子节点，最后再归约为stmt。
5. 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。

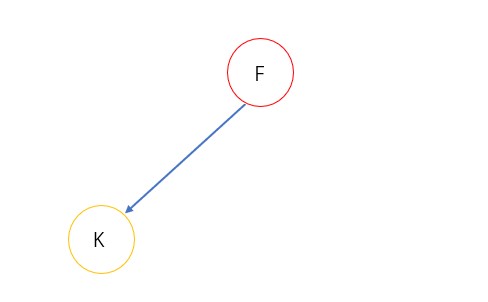


图62 抽象语法树2

## 2.9 实验总结

在本次实验中使用Flex和Bison在基本计算器的基础上添加变量命名、实现赋值、比较运算、流程控制、内置函数、用户自定义函数和简单错误恢复机制功能，使其成为一个具有全部功能的桌面计算器，虽然短小但是极具现实意义。通过阅读《Flex&Bison》第三章，加深学习抽象语法树的用法，熟练地掌握了抽象语法树的构造过程。由于有了上次实验的经验，本次对于代码的阅读以及为增加新内置函数pow(a,n)而修改增加代码变得简单了起来。但是，在实验的过程中还是遇到了一些编译错误的问题，但通过搜索相关资料，这些问题都得以解决。通过本次实验开发了一个具有全部功能的高级计算器，深刻理解了抽象语法树的用法以及Flex和Bison的代码语法，也激起了我的学习兴趣，相信以后的实验会更加顺利。

# 实验3修改高级计算器

## 3.1 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。

## 3.2 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。

## 3.3 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。

## 3.4 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)

## 3.5 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。

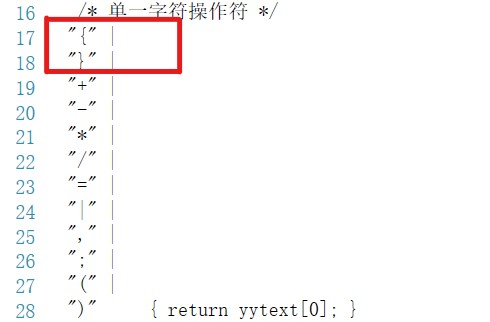


图63 fb3-3.l修改代码

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。

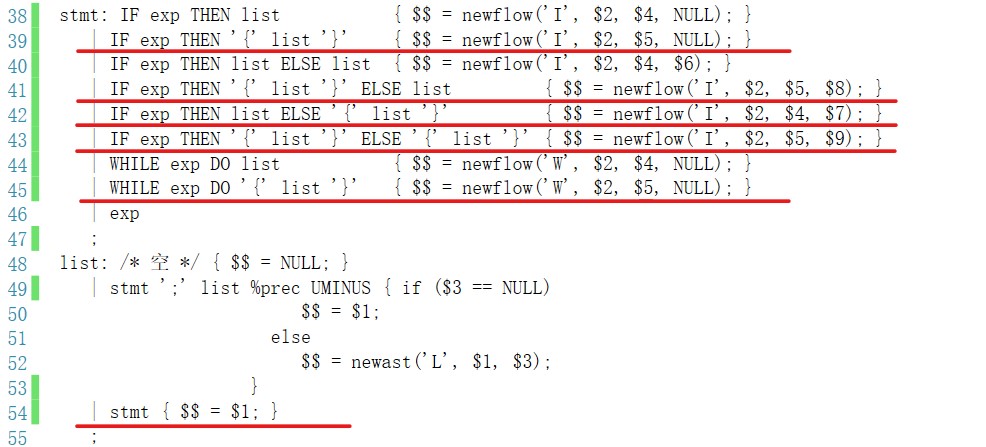


图64 fb3-3.y修改代码1

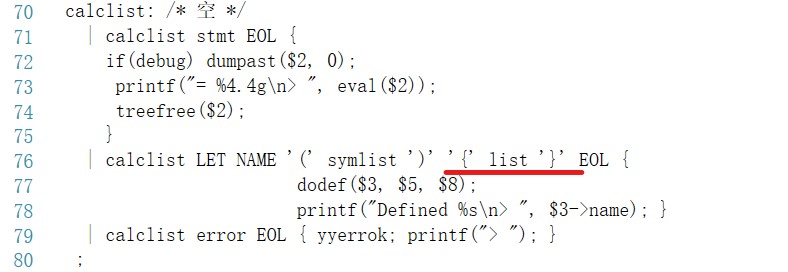


图65 fb3-3.y修改代码2



图66 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。

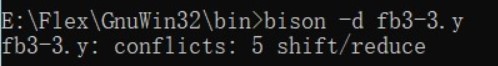


图67 移进归约冲突

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。

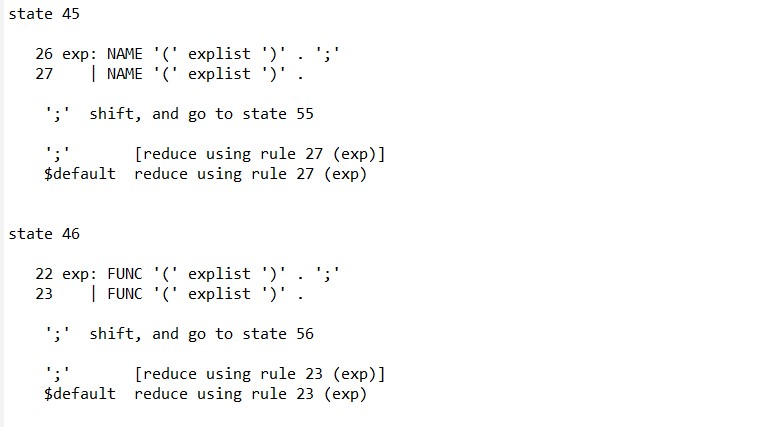


图68 部分具体冲突说明

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。

## 3.6 实验步骤

### 3.6.1 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。

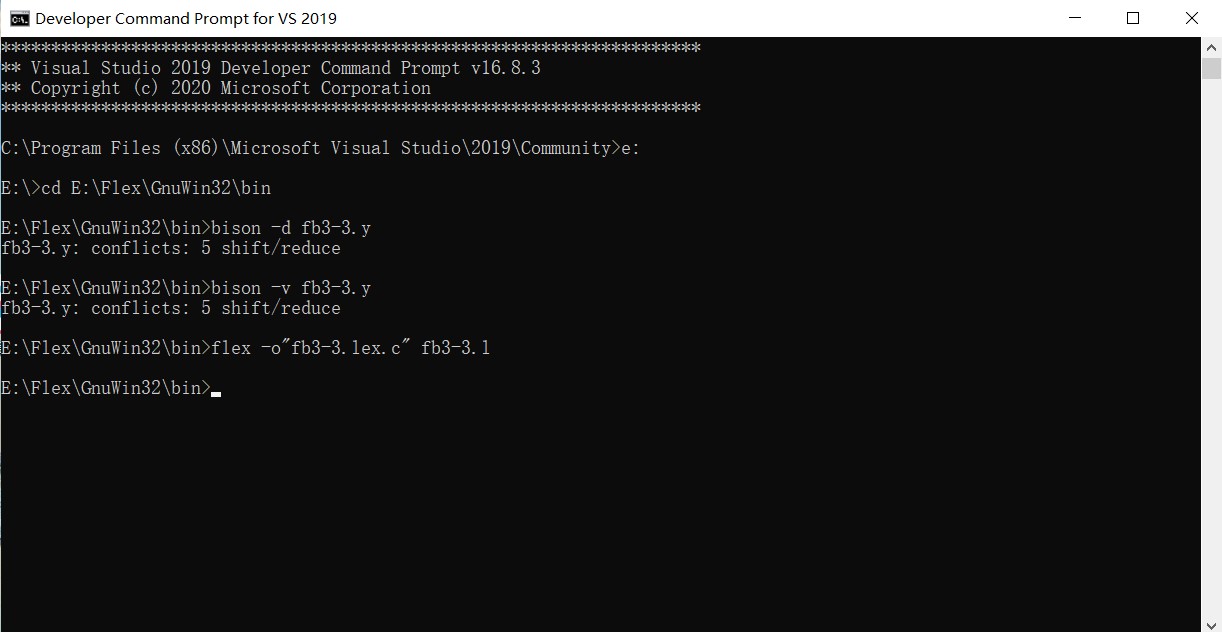


图69 输入命令

(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。

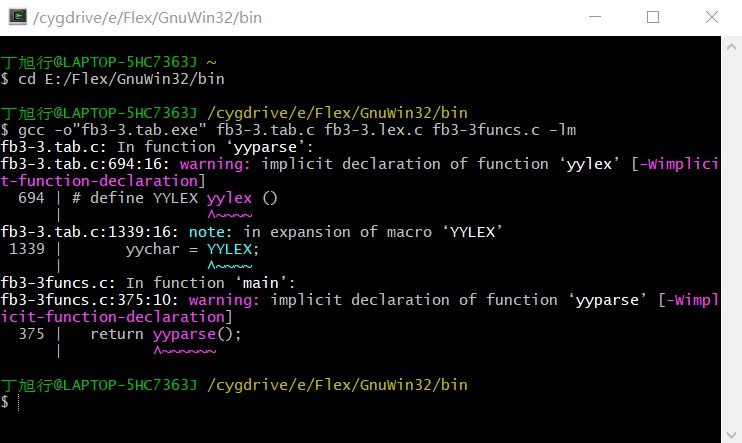


图8 输入命令



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。

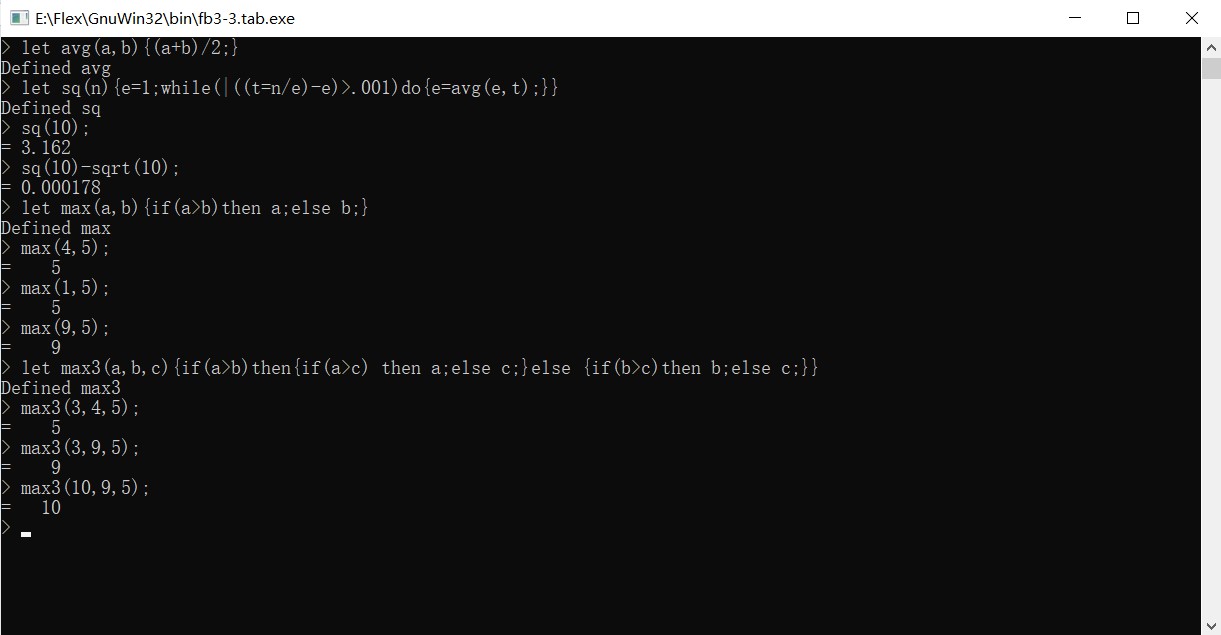


图10运行程序及结果显示

### 3.6.2 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。

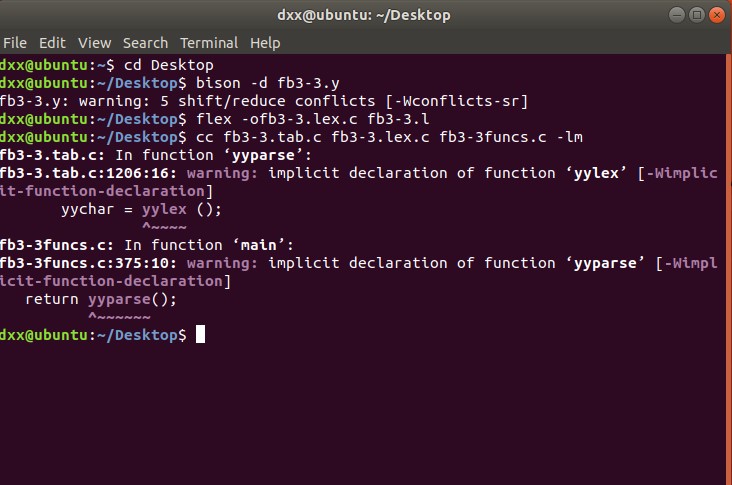


图11 输入命令



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。

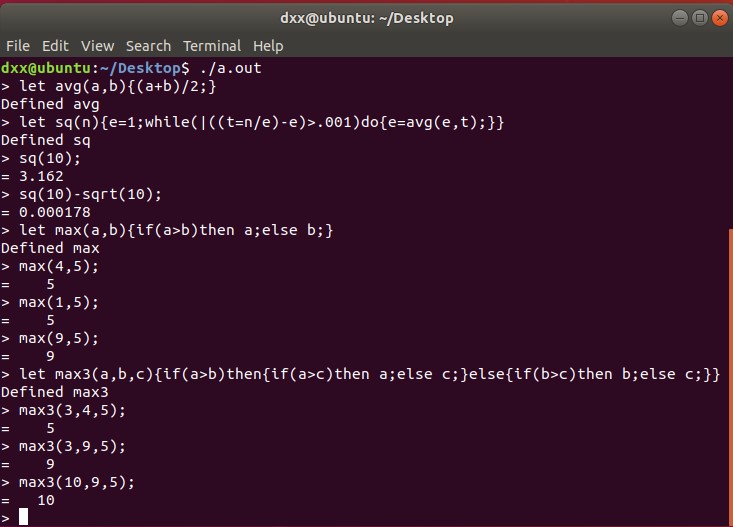


图13 运行程序及结果显示

## 3.7 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。

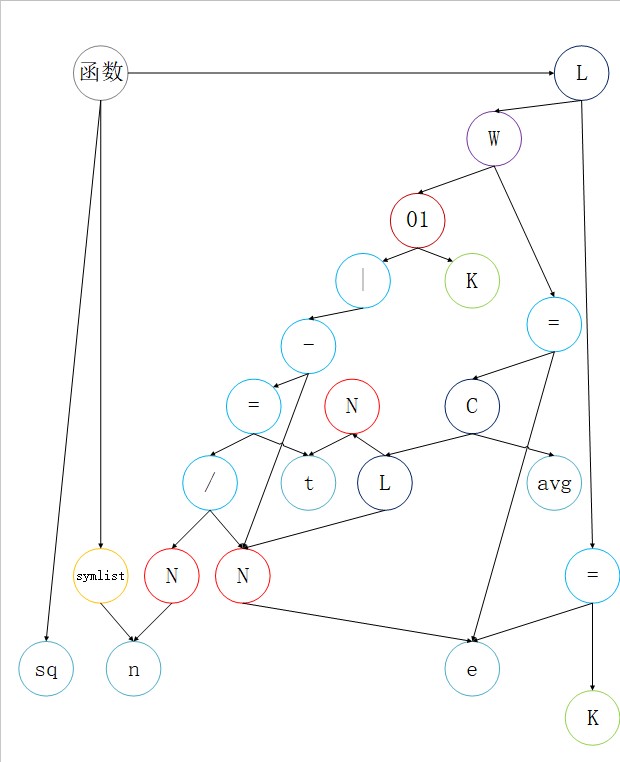


图14 抽象语法树1

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。

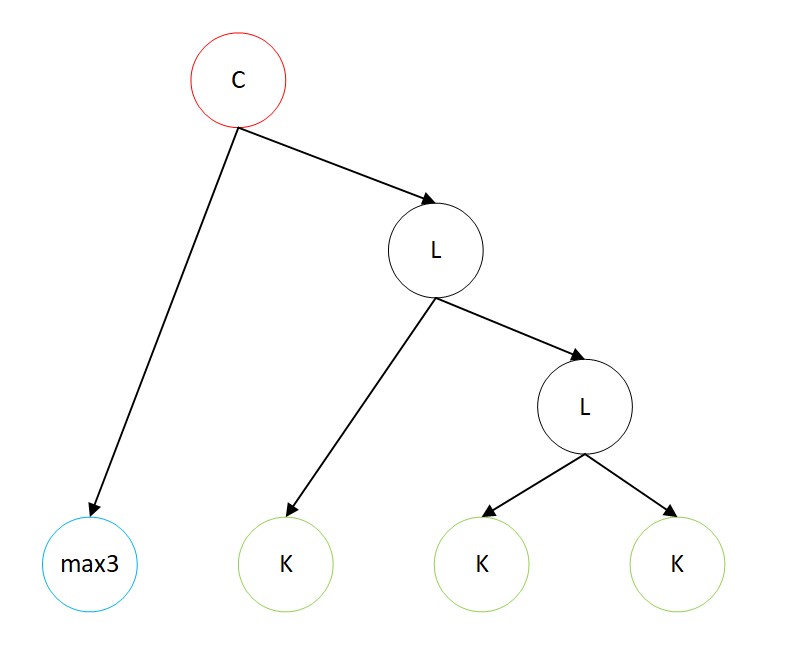


图15 抽象语法树2

## 3.8 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。

# 实验4可视化计算器

## 4.1 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并借助其他编程语言实现可视化界面。

## 4.2 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。

## 4.3 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)

## 4.4 实现工具

使用C#Windows窗体制作计算器的可视化界面，后端使用Flex和Bison对前端接受的表达式进行词法和语法分析处理，返回前端结果值，开发一个可视化的可编程计算器。

实验环境：Visual Studio 2019

编程语言：C#

## 4.5 整体思路

首先利用Flex和Bison构造一个可以识别一定语法和功能的编译器，再利用Visual Studio 2019生成其编译函数的dll文件，用C#调用dll文件中的主函数，使其读取in.txt的内容进行分析，这里in.txt中的内容也是由前端接受用户输入保存到in.txt中的。分析完之后，主函数将结果输出到out.txt文件中，然后由前端读取out.txt中的内容，进行结果展示。

但在实验途中发现，在Visual Studio 2019中生成dll文件始终报错，在经过用百度查阅各种资料，并多次尝试无果后放弃原来的思路，改换直接用C#调用编译生成的可执行的fb3-3.tab.exe文件，使其读取in.txt的内容进行分析，分析完之后将结果输出到out.txt文件中。

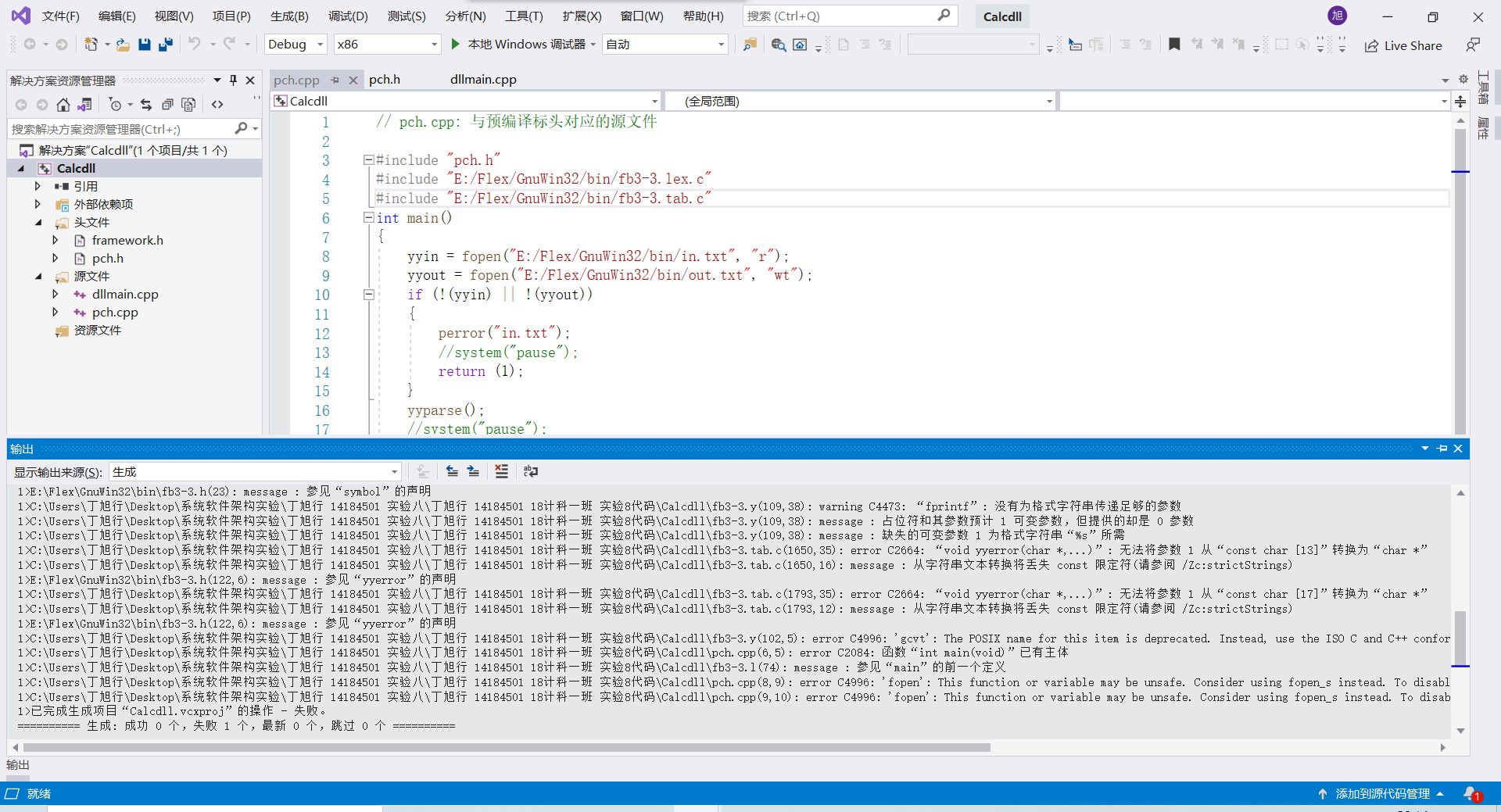


图70 封装dll文件失败

## 4.6 设计原理

### 4.6.1 基于Flex&Bison编译器整体原理

Flex是一个自动化工具，可以按照定义好的规则自动生成一个C函数yylex()，称为扫描器(Scanner)。这个函数可以将输入按照定义好的规则分析文本串中的字符，找到符合规则的一些字符序列后，就执行在规则中定义好的动作(Action)。例如可以在规则中定义：如果遇到一个换行字符’\n’，那么就把行计数器的值加一。

Bison是一种通用目的的分析器生成器，它可以将LALR(1)上下文无关文法的描述转化成分析该文法的C程序。整体过程图如下：

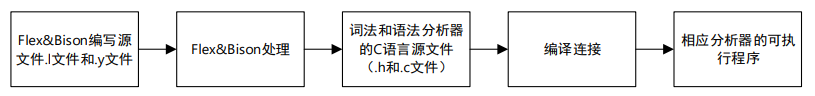


图71 生成过程

进行词法分析和语法分析的整体过程：

1. flex把输入流分解为若干片段记号，也就是所谓的终结符，即语法树的叶子节点，而bison则分析这些记号并基于逻辑进行组合。利用移进规约原理以及深度优先构造语法树，树的建立顺序是自底向上的，flex中先给词法单元建立了对应的结点，在语法树中属于叶子结点，在bison中，由词法单元对应的叶子结点规约出新的父节点，又由新的父节点和其他结点规约到新的父节点，若匹配则调用相应的抽象语法树构造函数自底向上建立语法树，如果规约到起始语法单元就表示整棵语法树都被建立好了。
2. 计算树由多个节点构成，每个节点都有一个nodetype，遍历树的代码使用这个变量来判断当前访问的节点，譬如内置函数使用fncall节点，用户自定义函数使用ufcall节点，流程控制表达式if/then/else和while/do使用flow节点，它包含控制表达式、then分支或者do语句体以及可选的else分支。
3. 每个计算树都有相应的值，计算器的核心是函数eval，它计算语法分析器中构造的计算树，对于if/then/else而言，它的值就是所选的分支的值；while/do的值则是do语句列表的最后一条语句的值。
4. 用户自定义函数：函数定义包括函数名、虚拟参数列表和代表函数体的计算树。当函数被定义时，参数列表和计算树的根节点地址将被简单地保存到符号表中函数名对应的条目中。在调用自定义函数时，就可以通过这两个地址，将虚拟参数改为实际传入的参数，然后再用eval()函数计算计算树的结果，计算完毕，再将虚拟参数改回原来的值。

### 4.6.2 C#调用后端实现与前端的交互

DLL(Dynamic Link Library)文件为动态链接库文件，又称“应用程序拓展”，是软件文件类型。在Windows中，许多应用程序并不是一个完整的可执行文件，它们被分割成一些相对独立的动态链接库，即DLL文件，放置于系统中。当我们执行某一个程序时，相应的DLL文件就会被调用。按照最开始的思路使用C#调用dll文件，我查询了关于生成dll文件以及C#调用dll文件的很多资料，其中最有用的一篇为：<https://blog.csdn.net/weixin_34976988/article/details/99625533>

但由于封装dll文件始终出问题，并且没有找到解决方式，所以改用C#直接调用flex和bison的编译器执行文件，为此我也查询了很多资料，这是其中一篇较为有用的文章：<https://blog.csdn.net/dark_tone/article/details/100190740>

本系统的总体结构图如图所示，首先C#前端将获取待分析的语句以及操作，送到int.txt文件中，之后调用fb3-3.tab.exe文件读入内容并计算返回结果到out.txt文件，最后前端读取返回内容并显示：

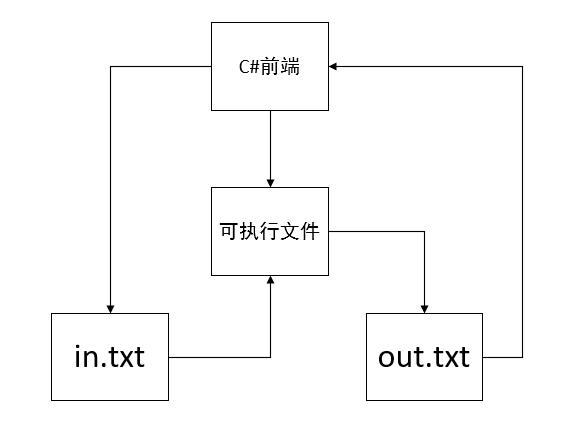


图72 总体结构图

## 4.7 实现过程

### 4.7.1 利用Flex和Bison实现词法分析和语法分析

这里可以直接使用上次实验完成的flex和bison代码，但是需要注意：前几次实验命令的输入为键盘输入，这次我们需要修改输入输出流来完成文件的输入输出。这里需要明白编译器工作的几个原理：

1. .l文件中的第三段即用户代码段，是会被原样拷贝到.lex.c的文件中的，可以定义一些辅助函数或代码供扫描器yylex()调用，或者调用扫描器，一般来说就是 main()函数，这一部分是可选的，如果没有的话，Flex会在.lex.c里默认生成。
2. Flex的输出是.lex.c文件，它包含扫描例程yylex()、一些用来存放匹配的token的table和一些附加的例程和宏。yylex()被调用后，它从输入文件yyin扫描tokens，直到遇到EOF时返回。其中yyin是一个文件指针，指向的是作为输入的文件，它在缺省条件下是stdin，为标准输入，可以将其改为到指定的文件来获取输入，yyout同理。
3. yyin的原理在YY\_INPUT宏中有定义：YY\_INPUT(buf,result,max\_size)是最多将max\_size个字符放在字符数据缓冲区，返回一个整型变量，其值要么是读入的字符数，要么是常量YY\_NULL(Unix下是0)指示EOF。扫描器通过yyin一次读一个数据块而不是用getc()读一个个字符，从而提高了效率。当扫描器从 YY\_INPUT收到EOF，它会检查yywrap()函数。如果该函数返回false即0，则假定该函数已经执行并设置了yyin指向另一个输入文件，扫描继续进行；如果该函数返回true非0，扫描器终止，返回0给它的调用者。.l文件中使用了%option noyywrap(相当于返回1)。

基于如上认知，将yyin,yyout修改为指定的文件来输入输出，而不再依赖于用户在键盘的输入和显示器输出。

1. 在.l文件中在用户代码段添加自己定义的主函数如下：

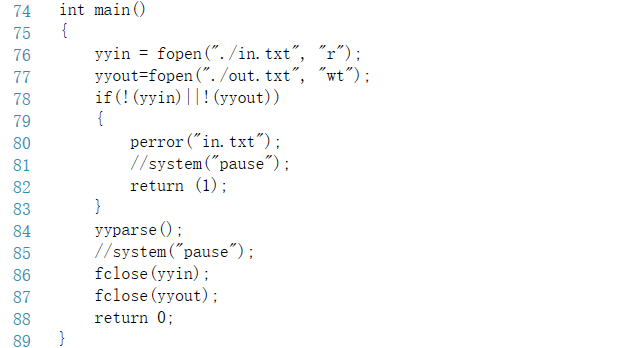


图73 .l文件修改

1. 这里只是定了输入的来源和最终结果输出的位置，需要修改.y文件将函数的输出写入到yyout里，在.y 文件里对输出进行修改如下：

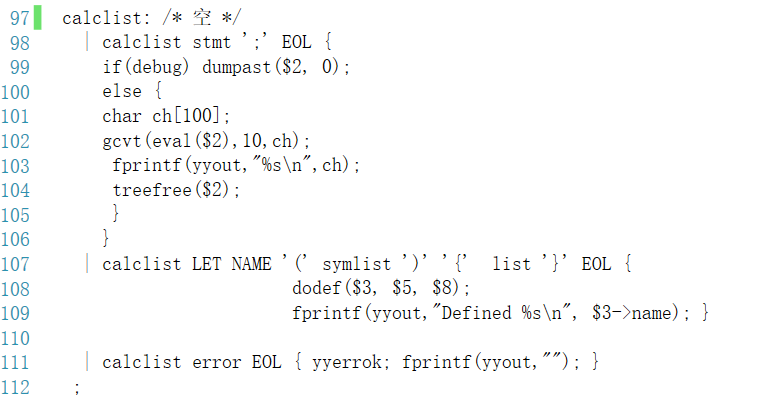


图74 .y文件修改

1. 为了执行.l文件里的主函数需要将fb3-3funcs.c中定义的主函数注释掉，防止冲突：

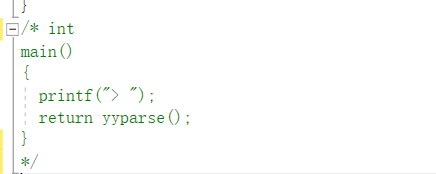


图75 fb3-3funcs.c文件修改

1. 修改后重新编译，生成可执行文件fb3-3.tab.exe：

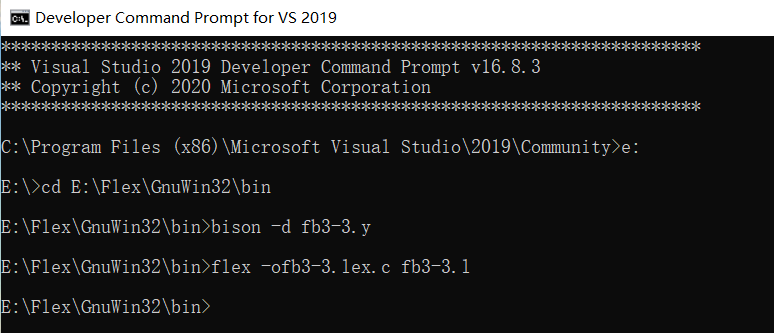


图76 输入命令1

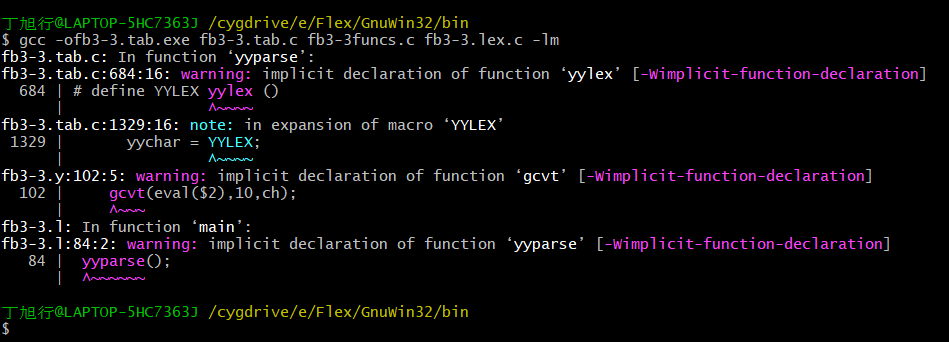


图77 输入命令2

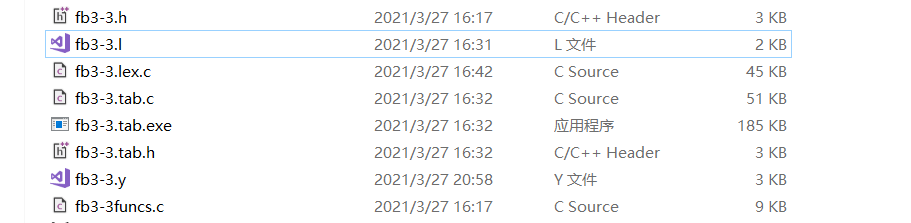


图78 生成文件

1. 在in.txt文件中输入计算表达式，之后运行生成的fb3-3.tab.exe文件，可在out.txt文件中显示计算结果，说明编译测试成功：

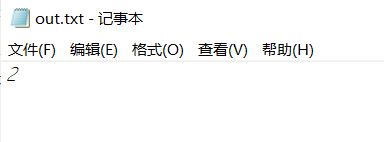
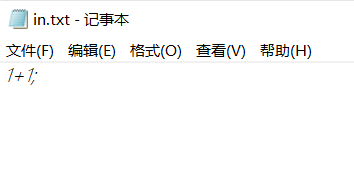


图79 测试成功

### 4.7.2 编写并修改前端代码

(1) 对于前端，我在之前做过的C#实现的基本计算器基础上进行二次开发，做了相应的修改，添加新的功能按钮以及新的窗体。



图80 项目目录结构

(2) Visual Studio 2019的C#中的窗体类提供了友好的UI制作界面，可以通过拖动工具箱里的工具进行UI绘制并自动生成代码，同时可以设置点击事件。

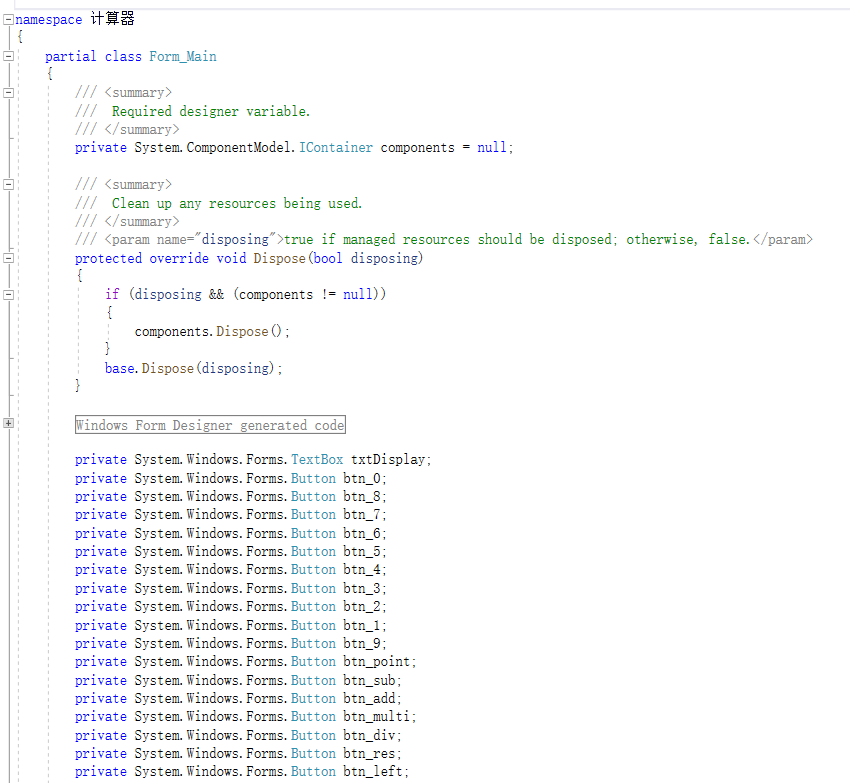


图81 主窗体部分代码

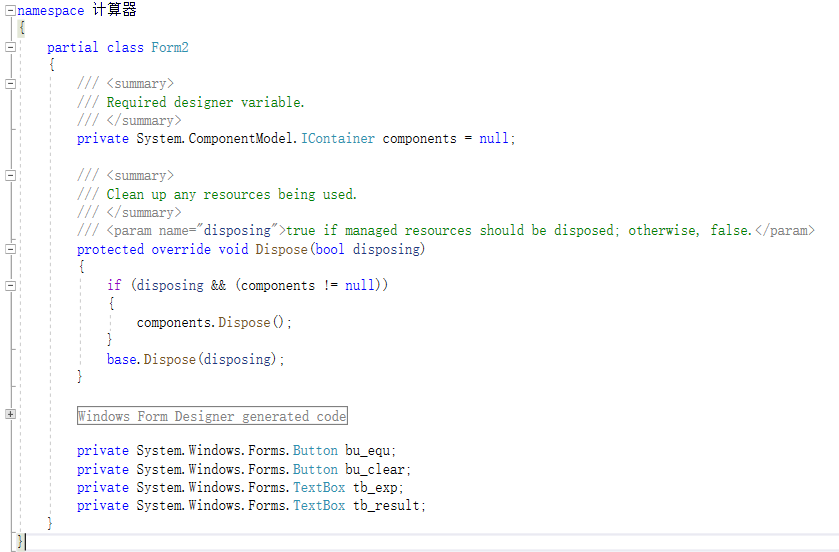


图82 副窗体代码

(3) 用户可以在基本计算器界面进行点击操作，也可点击高级版按钮进入高级计算器界面，在下方区域输入指令，点击”=”按钮即可在上方区域返回结果。

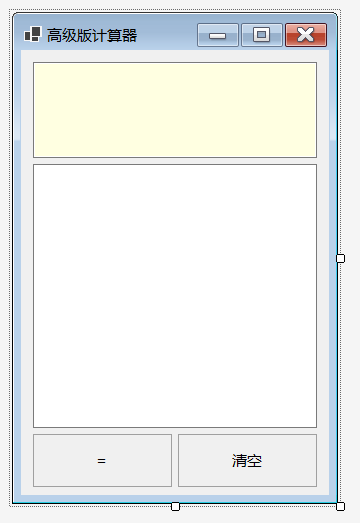
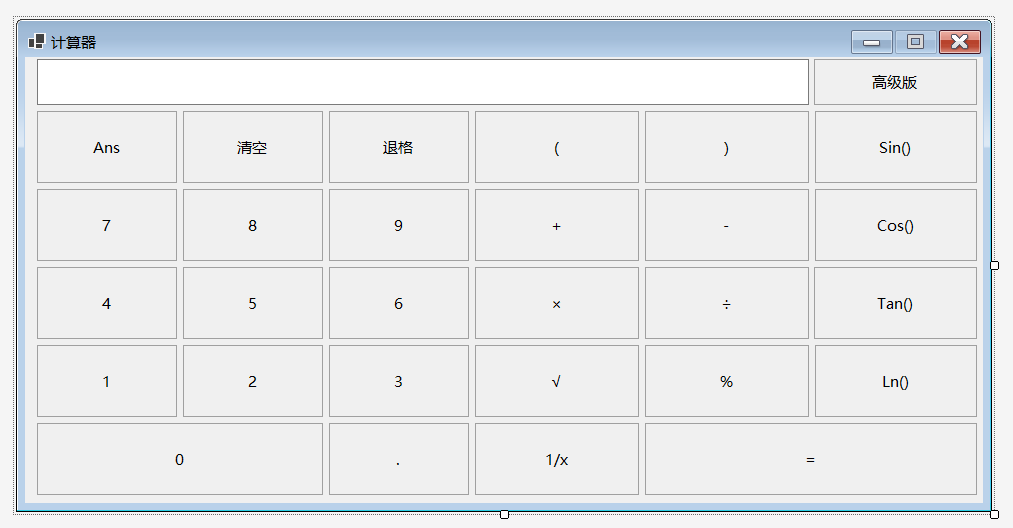


图83基本计算器 图84高级版计算器

(4) 对于新增窗体的按钮以及基本计算器的按钮进行代码编写与更改，使其能够完成相应功能，并可以把用户输入信息输去到in.txt文件中，并能够调用fb3-3.tab.exe文件对in.txt文件进行解析，然后从out.txt文件中读取最终结果返回到窗体显示区域。

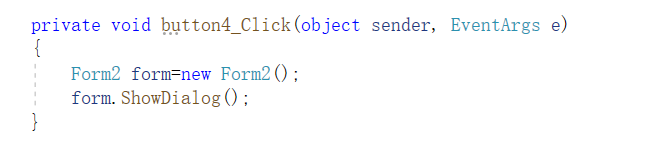


图85 “高级版”按钮

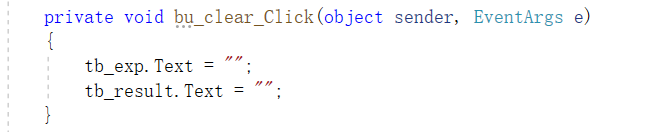


图86 “清空”按钮

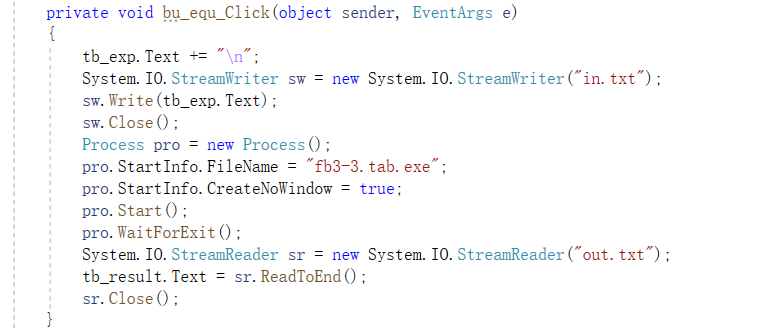


图87 “=”按钮

其中Process类进行调用exe文件，StreamWriter与StreamReader分别用来完成对in.txt文件的写入和对out.txt文件的读取。

## 4.8 设计成果展示

计算器前端运行界面如图：



图88 基本计算器

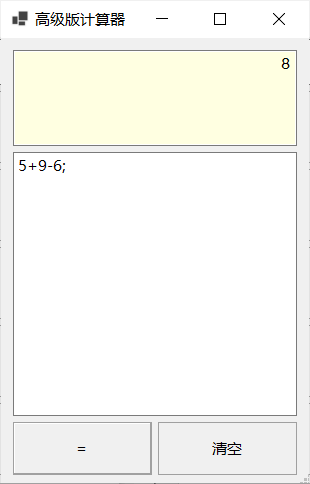


图89 高级版计算器

其中基本计算器与普通计算器无异，可以实现简单的四则运算、部分函数、以及部分非常规运算，通过点击操作在显示框内显示输入的计算公式，点击”=”按钮即可返回结果到显示框。高级版计算器可满足高级计算器的功能包括控制语句、变量命名、自定义函数等，用户在下方输入语句，点击”=”按钮后返回信息显示在上方区域。”清空”按钮可以清除上方和下方的内容。

高级版计算器功能展示：

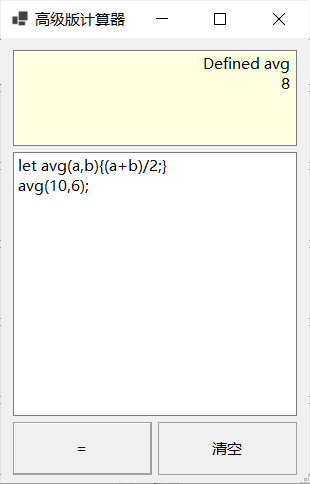
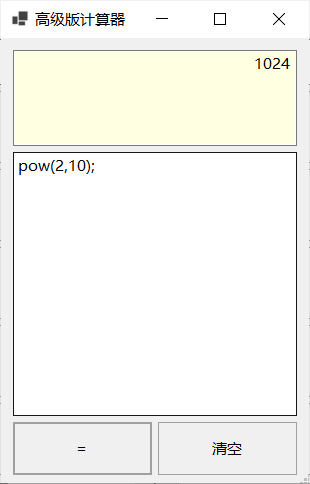


图90 内置函数 图91 自定义函数

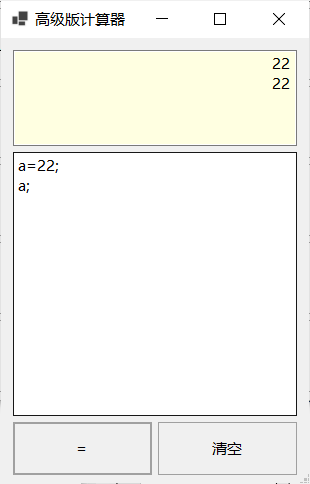
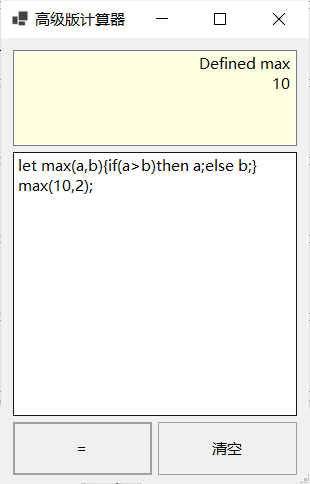


图92 控制语句 图93 变量命名

## 4.9 实验心得与总结

最终实现了这个高级计算器的可视化工作，开发了一个能够支持变量、过程、循环和条件表达式、提供标准函数、用户可自定义函数的多功能桌面计算器。通过本次的实验，加深了我对词法分析、语法分析及语义分析的理解，更加熟悉掌握了flex和bison结合构建编译器的原理，巩固了软件开发相关知识，同时增强了我的查错能力、代码能力以及查阅资料的能力，我可以说已经能够成功地运用一些工具来做一些实际的东西了，可谓是收获颇多！

但是仍旧有些许的不足，比如对编译器的封装dll文件未能实现，程序的一些错误处理不够完善，在所使用的开发工具及语言上没有太大创新仍旧使用的是以前接触过的语言，这些都有待于去提高和改进。希望未来自己的能力可以变得更强！

## 4.10 实验补充—python语言的dll文件调用

通过与同学间的讨论，得知可以通过gcc命令生成dll文件，命令如图：

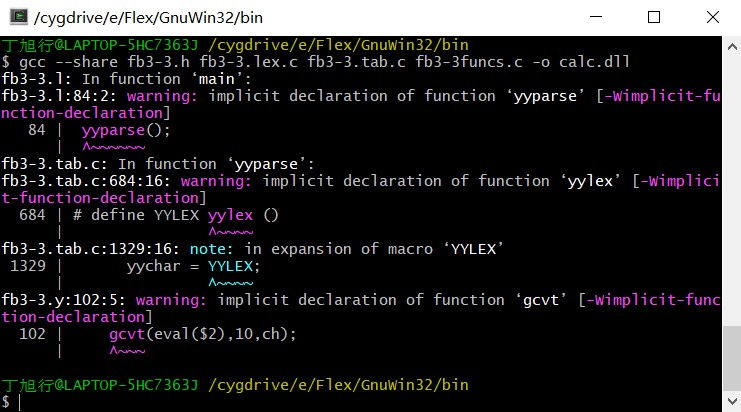


图94 gcc命令生成dll文件

但是因为没有匹配的lib文件，导致dll文件无法在C#环境下使用，所以我更换了编程语言使用python调用dll文件内的函数。我在码云上找到了使用pyqt5制作的科学计算器<https://gitee.com/ngc13009/scientific_calculator>，并在此基础上进行二次开发。

我在原本的科学计算器上添加高级版模块，点击弹出高级计算器子窗口，用户可以在此窗口输入表达式，点击“=“后返回结果值。

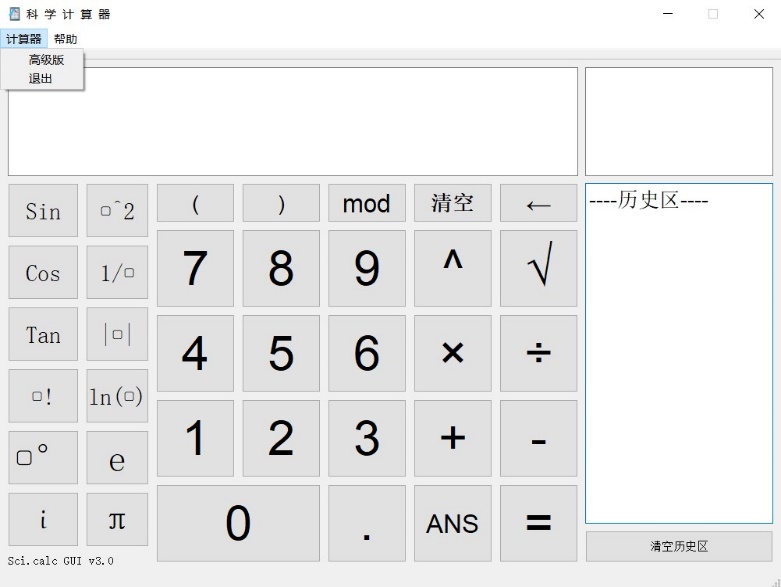


图95 高级版模块按钮 图96 高级计算器子窗口

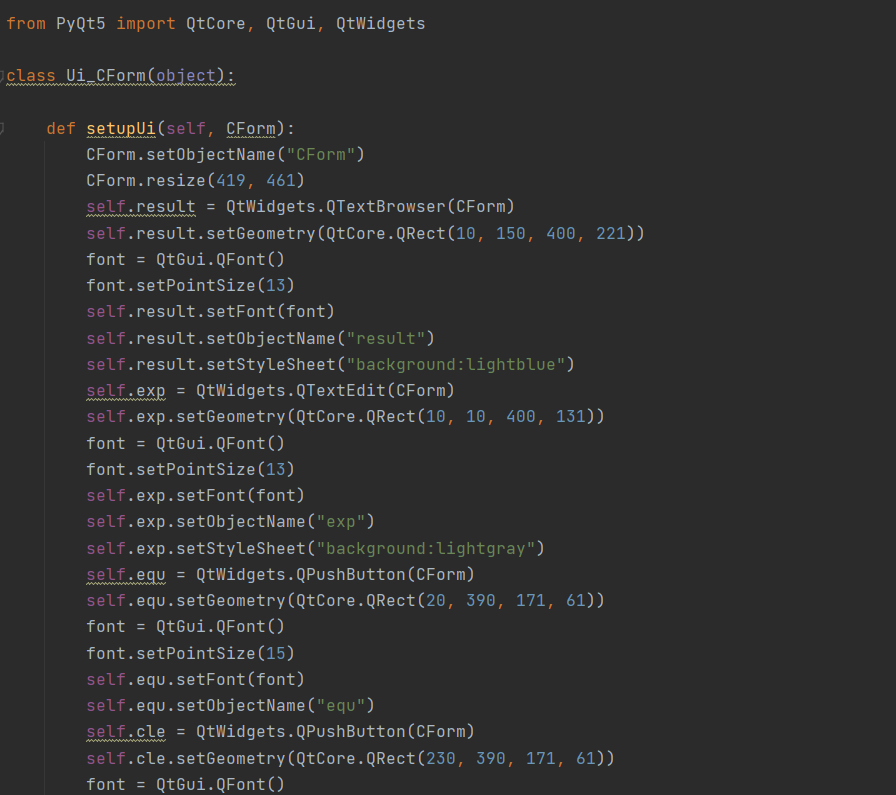


图97 子窗体部分代码

通过查找资料了解到了python调用dll文件的方法，其中最有价值的一篇文章是<https://www.php.cn/python-tutorials-422498.html，> 通过引入ctypes库，生成dll对象，进而调用对象内提供的公共函数：



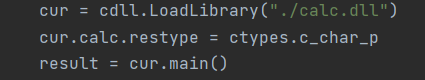
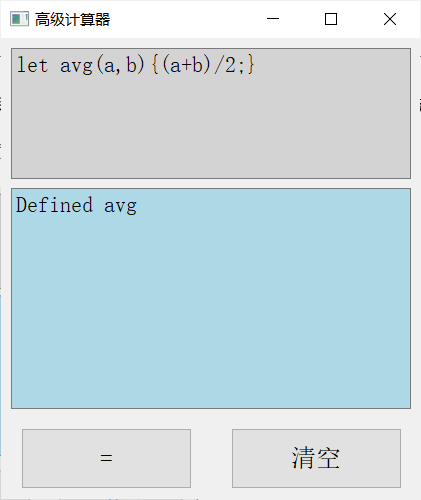
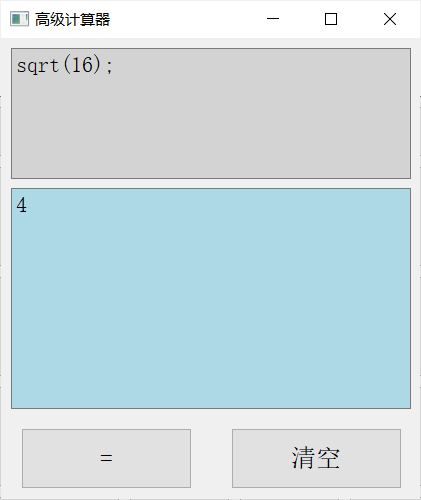


图98 关键代码

此计算器系统的总体架构与C#环境下的计算器架构一致，工作流程也相仿，在此不再赘述。最后运行效果如图：



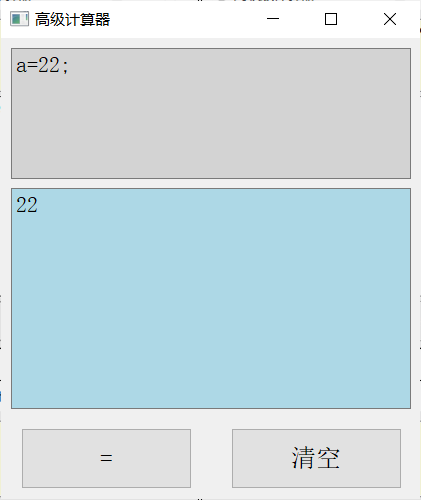
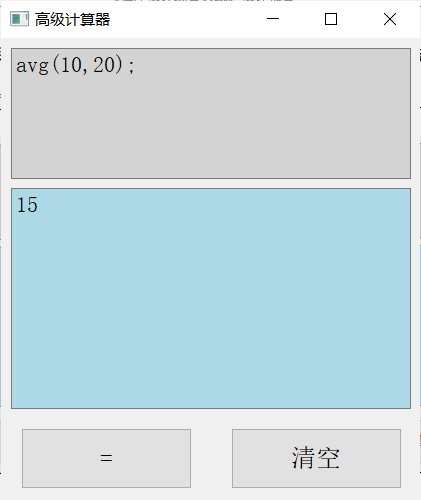


图99 运行效果图