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：实验六 Flex/Bison 综合实验二

学生姓名：陈柏翰

学生学号：02140385

专业班级：计算机科学与技术 2014-4 班

任课教师：张博老师

Flex/Bison 综合实验二

一 实验内容

使用 flex 和 bison 开发了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。

二 实验要求

编写 Flex/Bison 源文件，实现 C 语言的语法分析功能，最后上机调试。

要求编写一个测试程序：

> 首先自定义两个函数 sq 和 avg，sq 函数使用 Newton 方法来迭代计算平方根；avg 函数计算两个数值的平均值。
>
> 利用定义好的函数进行计算，得到计算结果并显示出来。

三 结合实验结果，如 sqrt(10)和 sq(10)，给出抽象语法树的构建过程。

当我们向高级计算器输入 sqrt（10）和 sq（10）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。

以上即是计算的基本过程。

以下结合代码进行详细的分析。

首先对 sqrt（10）进行词法分析。

"sqrt" { yylval.fn = B_sqrt; return FUNC; }

将计算后的结果赋值给 yylval.fn，并且返回一个 token，告诉语法分析器，词法分析的结果是 FUNC.

Exp：...

| FUNC '(' explist ')' { $$ = newfunc($1, $3); }

| NAME '(' explist ')' { $$ = newcall($1, $3); }

计算器由以上这两条规则来处理 FUNC 标识的内置函数（即 sqrt）和 NAME 标识的用户自定义函数。

explist: exp

 | exp ',' explist { $$ = newast('L', $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。

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

 | NAME ',' symlist { $$ = newsymlist($1, $3); }

而 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。

然后我们看到建立语法树节点的两个函数 newfunc（ ）newast（ ）

```
struct ast *
newfunc(int functype, struct ast *l)
{
  struct fncall *a = malloc(sizeof(struct fncall));
  if(!a) {
    yyerror("out of space");
    exit(0);
```

```
  }
  a->nodetype = 'F';
  a->l = l;
  a->functype = functype;
  return (struct ast *)a;
}
```

Sqrt（10）运行结果为：节点类型 F；l 为传入的参数 l；funtype 为传入的参数 funtype。

此处的节点类型 F 用于 eval（ ）的计算

```
case 'F': v = callbuiltin((struct fncall *)a); break;
```

其中的 callbuiltin（ ）如下

```
static double
callbuiltin(struct fncall *f)
{
  enum bifs functype = f->functype;
  double v = eval(f->l);
 switch(functype) {
 case B_sqrt:
   return sqrt(v);
 case B_exp:
   return exp(v);
 case B_log:
   return log(v);
```

```
 case B_print:
   printf("= %4.4g\n", v);
   return v;
 default:
   yyerror("Unknown built-in function %d", functype);
   return 0.0;
 }
}
```

根据相应的 functype 调用相应的函数块计算函数结果

例如 sqrt（10）即调用内置函数 sqrt（ ）

```
struct ast *
newcall(struct symbol *s, struct ast *l)
{
  struct ufncall *a = malloc(sizeof(struct ufncall));
  if(!a) {
    yyerror("out of space");
    exit(0);
  }
  a->nodetype = 'C';
  a->l = l;
  a->s = s;
  return (struct ast *)a;
```

}

Sq（10）运行结果为：节点类型 C；l 为传入的参数 l；s 为传入的参数 s。

此处的节点类型 F 用于 eval（ ）的计算

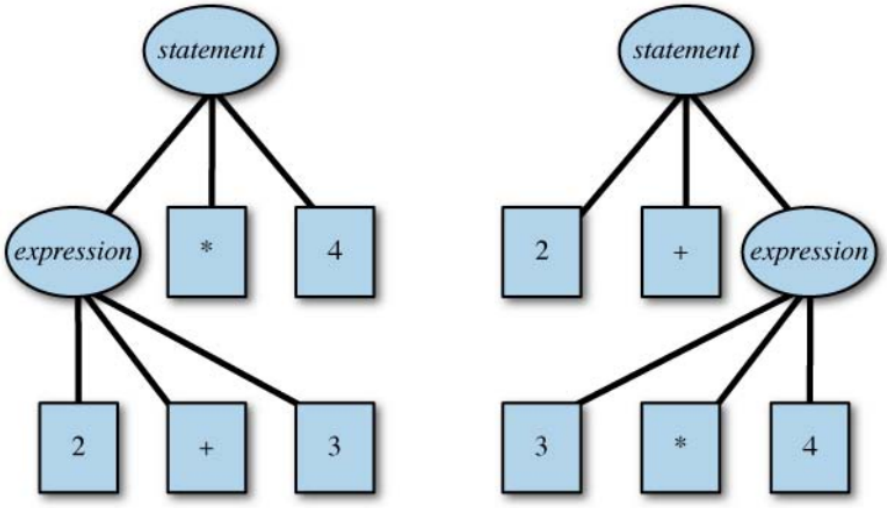
case 'C': v = calluser((struct ufncall *)a); break;

四 抽象语法树的构建过程

本次实验制作的高级计算器，在实验五的基础上，添加了如下几个方面：

1. 对变量进行命名和赋值。
2. 引入比较表达式
3. 实现了一些基本的流程控制（if then else do while）
4. 支持内置函数和用户自定义函数

结合以下对 fb3-2.y、fb3-2.l、 fb3-2funcs.c、 fb3-2.h 四个文件的阅读，现对这个高级计算器进行分析。



在实验报告五中，我们已经分析到，语法分析器一旦创建了一棵抽象语法树，它就可以直接编写递归程序来遍历整棵树。我们为每个表达式创建抽象语法树，然后计算这棵抽象语法树，打印运算结果，最后释放抽象语法树。

根节点起始符号为 statement，然后调用 newast（ ）创建三个子节点分别为 1、+、expression，然后再对 expression 建立三个子节点分别为 2、-、expression，再对 expression 建立三个子节点分别为 3、*、expression，最后 expression 建立最后三个子节点为 2、/、5。然后自下往上计算，得到结果。

在计算结果时采用 depth-first 深度优先的算法，首先先递归访问每个节点的所有子树，然后再访问节点本身。

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

对表达式的计算过程，在以上已经描述。

计算器的核心是 eval，它用于计算抽象语法树。

其中比较表达式根据比较的结果返回 1 或者 0。

在计算例如 if、then、else 的选择分支时，他会通过计表达式的抽象与法树的值来决定选择哪一个分支继续。

在计算例如 do、while 的循环时，eval 会循环地计算条件表达式的值，然后建立语句的抽象与法树，直到表达式不再成立为止。

关于内置函数的实现：计算器先确定用户调用了什么函数，然后再调用相应的代码来执行即可。

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：

1. 计算实参的值
2. 保存形参的值，将实参赋值给形参。
3. 执行函数体，构建抽象语法树。
4. 将形参恢复原来的值
5. 返回函数的返回值。

五 分析 fb3-2.y

```
%union {
  struct ast *a;
  double d;
  struct symbol *s;          /* which symbol */
  struct symlist *sl;
  int fn;                    /* which function */
}
```

这个版本的%union 定义了许多符号值

```
/* declare tokens */
%token <d> NUMBER
%token <s> NAME
%token <fn> FUNC
%token EOL

%token IF THEN ELSE WHILE DO LET
```

定义了 6 个记号，分别为 IF THEN ELSE WHILE DO LET 的保留字

```
%nonassoc <fn> CMP
%right '='
%left '+' '-'
%left '*' '/'
%nonassoc '|' UMINUS
```

以上是优先级声明列表

```
%%
exp: exp CMP exp          { $$ = newcmp($2, $1, $3); }
   | exp '+' exp          { $$ = newast('+', $1,$3); }
   | exp '-' exp          { $$ = newast('-', $1,$3);}
   | exp '*' exp          { $$ = newast('*', $1,$3); }
   | exp '/' exp          { $$ = newast('/', $1,$3); }
   | '|' exp              { $$ = newast('|', $2, NULL); }
   | '(' exp ')'          { $$ = $2; }
   | '-' exp %prec UMINUS { $$ = newast('M', $2, NULL); }
   | NUMBER               { $$ = newnum($1); }
   | FUNC '(' explist ')' { $$ = newfunc($1, $3); }
   | NAME                 { $$ = newref($1); }
   | NAME '=' exp         { $$ = newasgn($1, $3); }
   | NAME '(' explist ')' { $$ = newcall($1, $3); }
;
explist: exp
 | exp ',' explist  { $$ = newast('L', $1, $3); }
;
symlist: NAME       { $$ = newsymlist($1, NULL); }
 | NAME ',' symlist { $$ = newsymlist($1, $3); }
;
calclist: /* nothing */                                  顶层规则
  | calclist stmt EOL {
```

```
  if(debug) dumpast($2, 0);
   printf("= %4.4g\n> ", eval($2));                    计算抽象与法树并打印
   treefree($2);                                       释放抽象语法树
  }
 | calclist LET NAME '(' symlist ')' '=' list EOL {
                    dodef($3, $5, $8);
                    printf("Defined %s\n> ", $3->name); }

 | calclist error EOL { yyerrok; printf("> "); }      错误恢复
```

该计算器能区分语句 stmt 和表达式 exp。语句在这里指的是一个控制流程的语句或者其判断的表达式。

六　分析 fb3-2.l

单一字符操作符返回值都是它本身

比较操作符返回值是 CMP 的记号，以数值的不同作为区分

关键字返回的是对应的 token 记号

七 分析 fb3-2funcs.c

计算器的核心是 eval，它用于计算抽象语法树。

其中比较表达式根据比较的结果返回 1 或者 0。

在计算例如 if、then、else 的选择分支时，他会通过计表达式的抽象与法树的值来决定选择哪一个分支继续。

在计算例如 do、while 的循环时，eval 会循环地计算条件表达式的值，然后建立语句的抽象与法树，直到表达式不再成立为止。

八 分析 fb3-2.h

在这个计算其中，每个符号都可以 有一个变量和一个用户自定义函数。

Value 用来保存符号的值；func 指向抽象语法树；sym 指向形参的链表。

本计算器定义了许多节点类型：ast、fncall 内置函数、ufncall 用户自定义函数、flow 流程控制表达式。

内置的函数共三个：sqrt（ ）exp（ ）log（ ）。

九 实验步骤

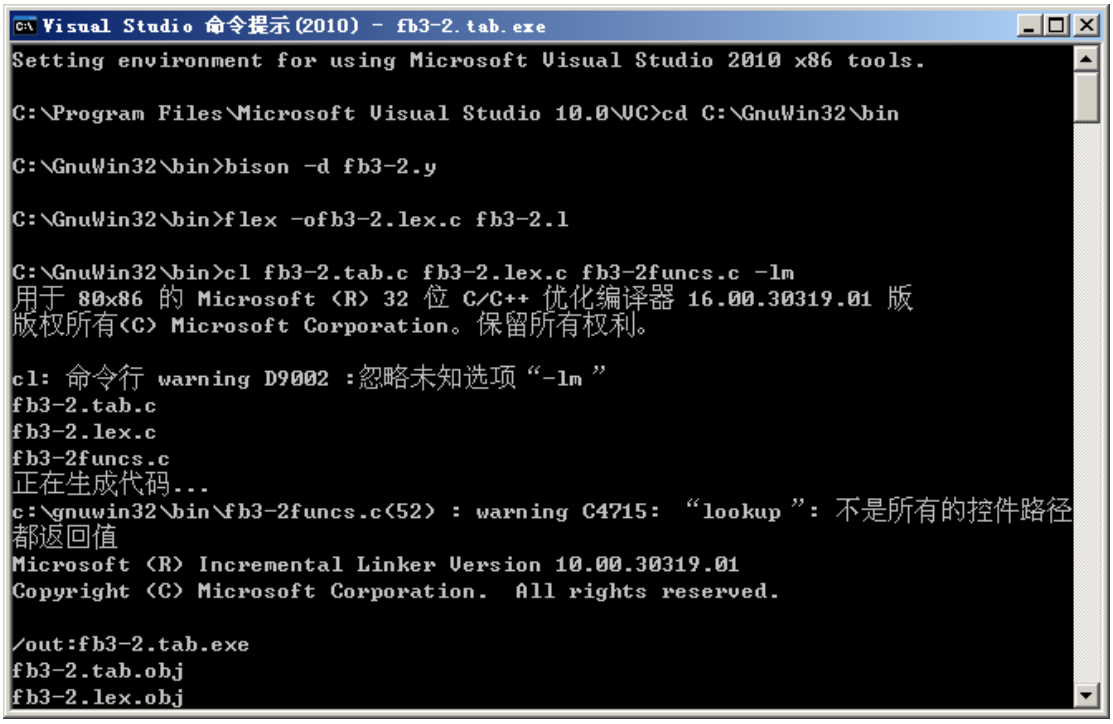
VS 命令提示行，里**执行**以下命令：

bison -d fb3-2.y

flex -ofb3-2.lex.c fb3-2.l

cl fb3-2.tab.c fb3-2.lex.c fb3-2funcs.c -lm

生成可执行文件: fb3-2.tab.exe



执行计算任务：

使用内置函数 sqrt(n)、 exp(n) ，log(n)

C:\GnuWin32\bin>fb3-2.tab.exe

> sqrt(10)

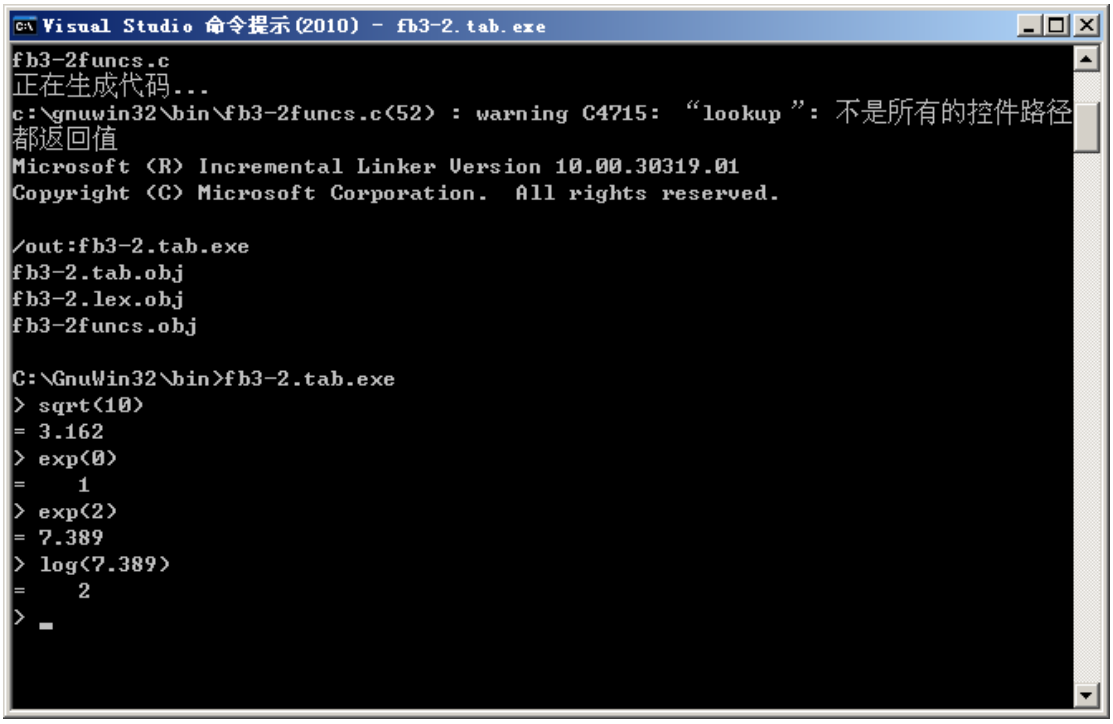
= 3.162

> exp(2)

= 7.389

> log(7.39)

= 2



定义函数 sq(n)、avg(a, b)，用于计算平方根

C:\GnuWin32\bin>fb3-2.tab.exe

> 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;

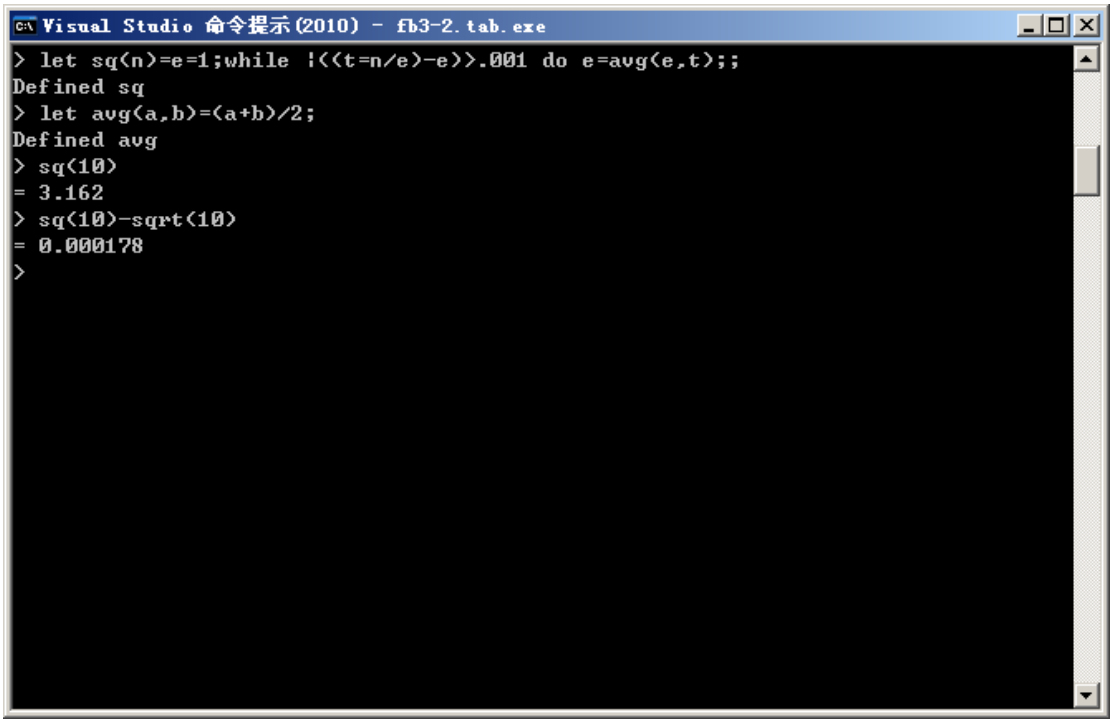
Defined avg

> sq(10)

= 3.162

> sq(10)-sqrt(10)

= 0.000178



增加新的内置函数 pow(a,n)计算一个数 a 的 n 次方

十 实验总结

1. 你在编程过程中遇到了哪些难题？你是怎么克服的？

   在实验过程中，对于语法树不是特别理解。

   我先是自己上网搜索语法树相关的资料，后来通过老师的帮助，顺利解决了所有的困难，完成了实验。

2. 你对你的程序的评价？

   本次实验完成的程序是一个简单的语法和语法分析器，和之前的实验相比增加了语法分析的环节，我认为在语法分析器的编程上，还有很多东西要学习与实践。

3. 你的收获有哪些

   我进一步掌握了 Yacc 与 Lex 基本使用方法，此外还学习了在 LINUX 系统下完成该实验的方法。并且在老师的答疑下解决了一些重点难点，受益匪浅。