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：实验五 Flex/Bison 综合实验一

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 综合实验一

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

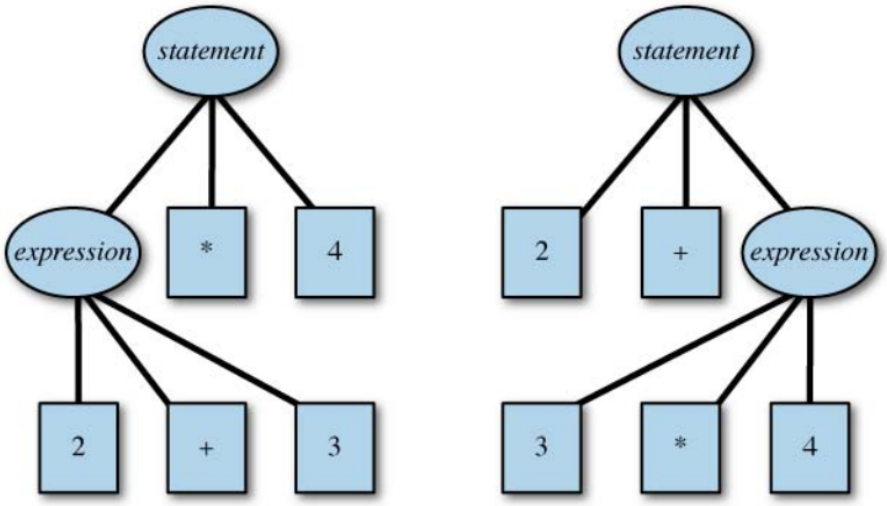
编写 Flex/Bison 源文件，实现 C 语言的语法分析功能，最后上机调试。

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 抽象语法树的构建过程

结合以下对 fb3-1.y、fb3-1.l、 fb3-1funcs.c、 fb3-1.h 四个文件的阅读，现对例如 1+2-3*2/5 的算式进行分析。



如上图所示，根节点为 statement，然后三个子节点分别为 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。

四 分析 fb3-1.y

```
%union {                                        定义一个结构体
  struct ast *a;
  double d;
}

/* declare tokens */
%token <d> NUMBER                               定义 token
%token EOL

%type <a> exp factor term

%%

calclist: /* nothing */
| calclist exp EOL {
     printf("= %4.4g\n", eval($2));
     treefree($2);                              从内存中释放节点
     printf("> ");                              打印>
 }
```

```
| calclist EOL { printf("> "); } /* blank line or a comment */
;

exp: factor                                  factor 是加数或减数
| exp '+' factor { $$ = newast('+', $1,$3); }   建立两个左右子节点
| exp '-' factor { $$ = newast('-', $1,$3);     建立两个左右子节点
;

factor: term                                 term 是乘数或除数
| factor '*' term { $$ = newast('*', $1,$3); }  建立两个左右子节点
| factor '/' term { $$ = newast('/', $1,$3); }  建立两个左右子节点
;

term: NUMBER   { $$ = newnum($1); }          建立一个新数字
| '|' term    { $$ = newast('|', $2, NULL); }  取绝对值 建立子节点
| '(' exp ')' { $$ = $2; }                   加括号
| '-' term    { $$ = newast('M', $2, NULL); }  取负数 建立子节点
;
%%
```

五　分析 fb3-1.l

```
/* float exponent */
EXP     ([Ee][-+]?[0-9]+)                    表达式 EXP 为 E 或 e 加减数值


%%
"+" |
"-" |
"*" |
"/" |
"|" |
"(" |
")"     { return yytext[0]; }                          匹配符号返回 yytext
[0-9]+"."[0-9]*{EXP}? |                                匹配到整数带小数
"."?[0-9]+{EXP}? { sscanf(yytext,"%lf",&yylval.d); return NUMBER; }
                                                       或者匹配到纯小数
                                                  将匹配到的字符串转换成浮点型
                                                       返回 NUMBER
\n      { return EOL; }                                匹配到换行符 返回 EOL
"//".*
[ \t]   { /* ignore white space */ }                   忽视跳过制表符
.       { yyerror("Mystery character %c\n", *yytext); }
%%
```

六 分析 fb3-1funcs.c

```
struct ast *                                          声明结构体
newast(int nodetype, struct ast *l, struct ast *r)    建立新的子节点函数
{                                                     其中包括三个参数
  struct ast *a = malloc(sizeof(struct ast));

  if(!a) {                                            如果 a 为 0
    yyerror("out of space");                          报错
    exit(0);
  }
  a->nodetype = nodetype;      a 指向第一个参数 节点类型 并赋值给 a
  a->l = l;                    a 指向第二个参数 左子节点 并赋值给 a
  a->r = r;                    a 指向第三个参数 右子节点 并赋值给 a
  return a;                                           返回 a
}

struct ast *
newnum(double d)                                      建立一个新的数字函数
{
  struct numval *a = malloc(sizeof(struct numval));

  if(!a) {                                            如果 a 为 0
    yyerror("out of space");                          报错
```

```
    exit(0);
  }
  a->nodetype = 'K';                    a 指向节点类型 并设置为字符 K
  a->number = d;                        a 指向数字 并将 d 的值赋值给 a
  return (struct ast *)a;               返回 a
}

double
eval(struct ast *a)                                  计算值的函数
{
  double v;

  switch(a->nodetype) {                              当 a 指向节点类型时
  case 'K': v = ((struct numval *)a)->number; break;
                                        若是字符 K 则是数字 值为其中的数值
  case '+': v = eval(a->l) + eval(a->r); break;      若是+ 则做加法运算
  case '-': v = eval(a->l) - eval(a->r); break;      若是- 则做减法运算
  case '*': v = eval(a->l) * eval(a->r); break;      若是* 则做乘法运算
  case '/': v = eval(a->l) / eval(a->r); break;      若是/ 则做除法运算
  case '|': v = eval(a->l); if(v < 0) v = -v; break;  若是| 则取其绝对值
  case 'M': v = -eval(a->l); break;                  若是- 则取反
  default: printf("internal error: bad node %c\n", a->nodetype);
  }                                     否则 报错 匹配不到相应的节点类型
```

```
  return v;                                          返回计算后的值 v
}
```

七 分析 fb3-1.h

```
  struct ast {                                       定义结构体 ast
  int nodetype;                                      其中包括 int 的节点类型
  struct ast *l;                                     包括结构体的左子节点
  struct ast *r;                                     包括结构体的右子节点
};

struct numval {
  int nodetype;                  /* type K */
  double number;
};

/* build an AST */                                   建立抽象语法树
struct ast *newast(int nodetype, struct ast *l, struct ast *r);  建立子节点
struct ast *newnum(double d);                        建立新的数字节点

/* evaluate an AST */
double eval(struct ast *);
```

/* delete and free an AST */

void treefree(struct ast *);　　释放抽象语法树的内存空间

八 实验步骤

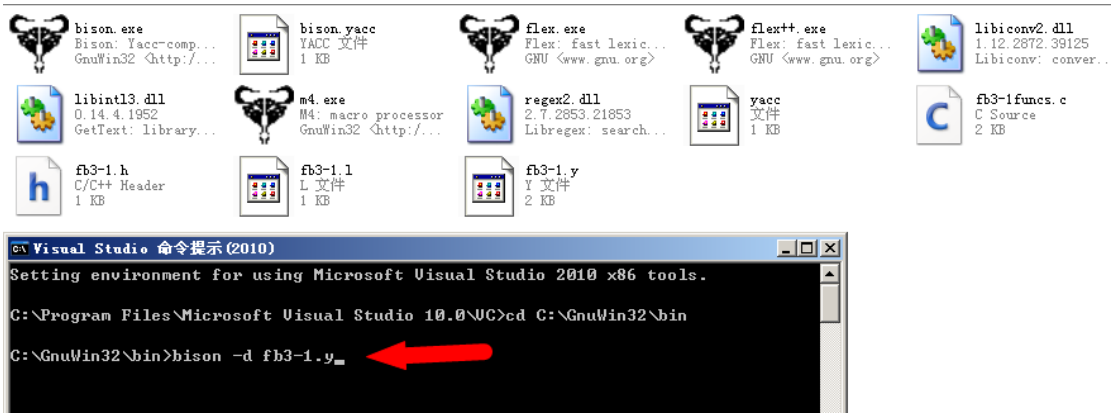
VS 命令提示行，里执行以下命令：

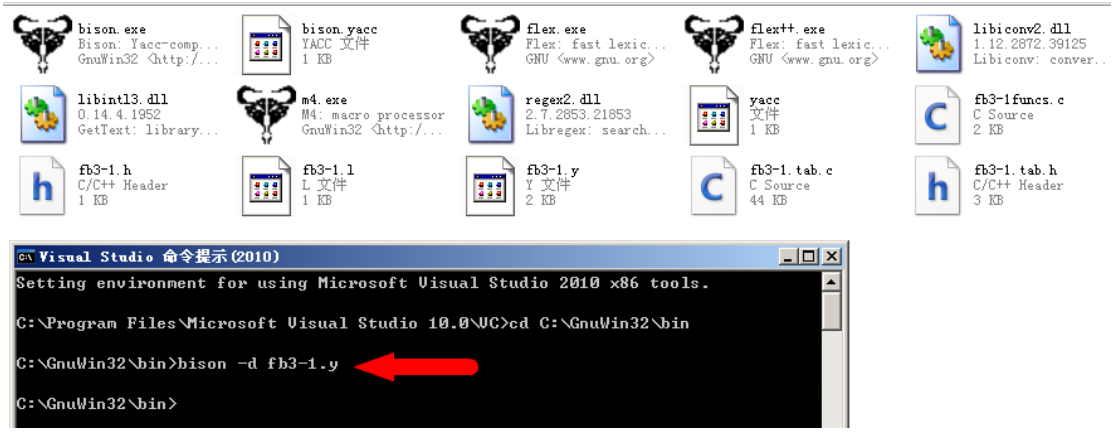
bison -d fb3-1.y

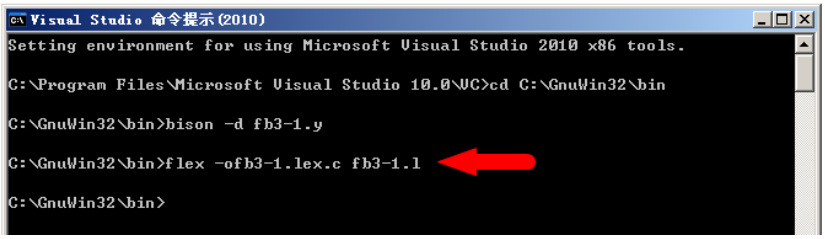
flex -ofb3-1.lex.c fb3-1.l

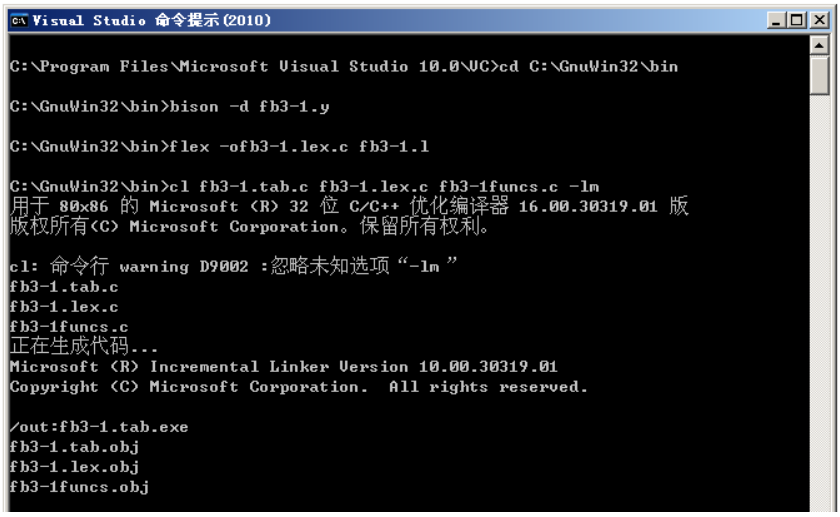
cl fb3-1.tab.c fb3-1.lex.c fb3-1funcs.c -lm

生成可执行文件: fb3-1.tab.exe









执行计算任务：

C:\GnuWin32\bin>fb3-1.tab.exe

> |-1

= 1

> |123

= 123

> (1+2)-(2*6)

= -9

> 1+2-3*2/5

= 1.8

>

九 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。