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 胡钧耀

学　　号 06192081

专业班级 计算机科学与技术 2019-4 班

任课教师 张博

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

实验（五） Flex/Bison 综合实验一

5.1 实验要求与目标

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具有现实意义的编译器。

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。

5.2 实验内容

阅读 *Flex/Bison.pdf* 第三章。使用 Flex 和 Bison 开发了一个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。

5.3 不同环境的配置和使用

5.3.1 Windows

根据以往经验，为防止报警，需要修改代码，首先是给 *fb3-1funcs.c* 加入 yyparse 方法的显式声明。

```
int yyparse();
```

其次是要在 *fb3-1.y* 给加入 yylex 方法的显式声明。

```
int yylex();
```

执行如下代码，可生成计算器 *3-1.exe*。

```
bison -d fb3-1.y
flex -ofb3-1.lex.c fb3-1.l
gcc -o cal3-1.exe fb3-1.tab.c fb3-1.lex.c fb3-1funcs.c -lm
```

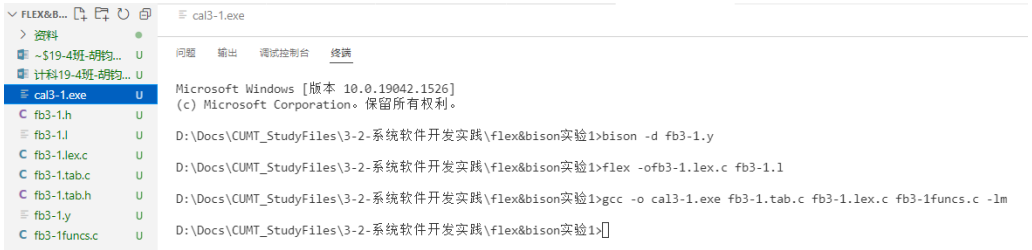

图 1 Windows 环境生成计算器 *3-1.exe*

调用该计算器，验证结果如下，与真实情况一致，表明计算器无问题。

```
D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\flex&bison实验1>cal3-1.exe
> |-1
=    1
> |123
=  123
> (1+2)-(2*6)
=   -9
> 1+2-3*2/5
=  1.8
>
```

图 2 Windows 计算器计算结果

5.3.2 Linux

直接把在 Windows 修改好的四个源文件复制到 Linux 系统中，输入如下代码，并进

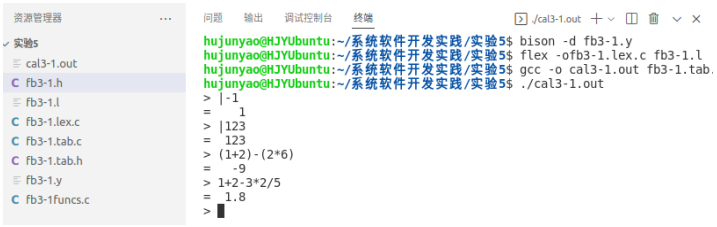

图 3 Linux 生成计算器 *3-1.out* 及运算结果

5.4 分析源码

5.4.1 *fb3-1.y*

首先引用一些必要的头文件，避免让 yylex 和 yyparse 等方法报错或报警。

```
%{
#  include <stdio.h>
#  include <stdlib.h>
#  include "fb3-1.h"
int yylex();
%}
```

定义了一个结构体，包括一个抽象语法树结构体指针和一个数字。

```
%union {struct ast *a;double d;}
```

声明了一些终结符号，一个是 NUMBER 数字，一个是 EOL（End of Line）行结束符号。还声明了一些非终结符号，如 exp 表达式，factor 系数，trem 句子。

```
/* declare tokens */
%token <d> NUMBER
%token EOL
%type <a> exp factor term
```

接着是各种模式的匹配以及对应的语法动作。

calclist→calclist exp EOL | calclist EOL。$2 表示打印第二个符号（也就是最后得到的表达式 exp），g 说明根据数值不同选择浮点数、十进制计数法%f 或浮点数、e 计数法%e。4.4 说明如果是小数，宽度 4，小数点后保留 4 位。treefree 表示对 exp 这棵抽象语法树进行释放（类似 delete）。这里对空行和注释的处理时也只输出一个空行（以类似命令行的‘>’作为开头导引符号）。

```
%%
calclist: /* nothing */
| calclist exp EOL {
     printf("= %4.4g\n", eval($2));
     treefree($2);
     printf("> ");
 }

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

exp→factor | exp + factor | exp - factor。factor 是加法或减法，两种情况都是建立两个左右子节点，用+或者-符号连接。

```
exp: factor
 | exp '+' factor { $$ = newast('+', $1,$3); }
 | exp '-' factor { $$ = newast('-', $1,$3);}
 ;
```

factor→term | factor * term| factor / term。term 是乘法或除法，两种情况都是建立两个左右子节点，用*或者/符号连接。

```
factor: term
 | factor '*' term { $$ = newast('*', $1,$3); }
 | factor '/' term { $$ = newast('/', $1,$3); }
 ;
```

term→NUMBER | ‘|’ term | (exp) | - term。四种情况分别是：建立新的数字、取绝对值建立子节点、加括号、取负数连接该数字建立子节点。

```
term: NUMBER   { $$ = newnum($1); }
 | '|' term    { $$ = newast('|', $2, NULL); }
 | '(' exp ')' { $$ = $2; }
 | '-' term    { $$ = newast('M', $2, NULL); }
 ;
%%
```

5.4.2 *fb3-1.l*

flex 的选项影响最终生成的词法分析器的属性和行为。这些选项可以在运行 flex 命令时在终端输入，也可以在.l 文件中使用%option 指定。noyywrap 表示在该.l 文件中不会调用 yywrap()，假设生成的扫描器只扫描单个文件。nodefault 表示不使用默认规则。yylineno 表示记录符号所在行号。

```
%option noyywrap nodefault yylineno
```

引用一些必要的头文件，避免报错报警。

```
%{ # include "fb3-1.h" # include "fb3-1.tab.h" %}
```

定义了一种科学计数法的指数表达式。

```
/* float exponent */
EXP ([Ee][-+]?[0-9]+)
```

遇到符号，返回自己本身 yytext[0]。

```
"+" | "-" | "*" | "/" | "|" | "(" | ")"  { return yytext[0]; }
```

匹配到整数、小数、科学计数，把 yytext 转为浮点数，然后返回 NUMBER。

```
[0-9]+"."[0-9]*{EXP}? | "."?[0-9]+{EXP}?
{ yylval.d = atof(yytext); return NUMBER; }
```

匹配到换行符，返回 EOL 行末标识。其他的则是处理空格、制表符等无关符号，其他符号则作为未知符号进入 yyerror 处理操作。

```
\n      { return EOL; }
"//".*
[ \t]   { /* ignore white space */ }
.   { yyerror("Mystery character %c\n", *yytext); }
%%
```

5.4.3 *fb3-1funcs.c*

引用一些必要的头文件，避免报错报警。

```
#  include <stdio.h>
#  include <stdlib.h>
#  include <stdarg.h>
#  include "fb3-1.h"
int yyparse();
```

定义了建立新的子节点的函数 newast，参数是节点类型 nodetype、左值指针 l、右值指针 r，返回值是该抽象树根节点的指针。其操作主要是把新的根节点 a 和左节点 l 和右节点 r 连接起来。

```
struct ast *
newast(int nodetype, struct ast *l, struct ast *r)
{
  struct ast *a = malloc(sizeof(struct ast));

  if(!a) {
    yyerror("out of space");
    exit(0);
  }
  a->nodetype = nodetype;
  a->l = l;
  a->r = r;
  return a;
}
```

定义了建立新的子节点的函数 newnum，参数是数字 d，返回值是该抽象树根节点的指针。其操作主要是把新的数字根节点 a 的结构体标注上对应的节点类型 K 和数字的值 d。

```
struct ast *
newnum(double d)
{
  struct numval *a = malloc(sizeof(struct numval));

  if(!a) {
    yyerror("out of space");
    exit(0);
  }
  a->nodetype = 'K';
  a->number = d;
  return (struct ast *)a;
}
```

定义了计算表达式数值的函数 eval，这容易使人联想到 Python 的 eval 方法。参数是抽象语法树的根结点指针 a，返回的是该抽象语法树的浮点数值结果。其操作主要是根据 case 语句分析出不同的 nodetype 要进行不同的处理，例如类型 K 标识根节点是数字，就可以直接输出对应的数值 v，如果类型是+，就要把 a 指针的左节点的 eval 方法返回值和右节点的 eval 方法的返回值进行相加，把这个结果 v 返回。不是所列出的类型，就要报错匹配不到对应的节点类型。

```
double
eval(struct ast *a)
{
  double v;
  switch(a->nodetype) {
  case 'K': v = ((struct numval *)a)->number; break;
  case '+': v = eval(a->l) + eval(a->r); break;
  case '-': v = eval(a->l) - eval(a->r); break;
  case '*': v = eval(a->l) * eval(a->r); break;
  case '/': v = eval(a->l) / eval(a->r); break;
  case '|': v = eval(a->l); if(v < 0) v = -v; break;
  case 'M': v = -eval(a->l); break;
  default: printf("internal error: bad node %c\n", a-
>nodetype);
  }
  return v;
}
```

定义了删除抽象语法树的函数 treefree，是一种递归函数，要删除本语法树首先要删除其所有子节点。

```
void
treefree(struct ast *a)
{
  switch(a->nodetype) {
    /* two subtrees */
  case '+':
  case '-':
  case '*':
  case '/':
    treefree(a->r);
    /* one subtree */
  case '|':
  case 'M':
    treefree(a->l);
    /* no subtree */
  case 'K':
    free(a);
    break;
  default: printf("internal error: free bad node %c\n", a-
>nodetype);
  }
}
```

出错处理。输出错误及其行号。

```
void
yyerror(char *s, ...)
{
  va_list ap;
  va_start(ap, s);

  fprintf(stderr, "%d: error: ", yylineno);
  vfprintf(stderr, s, ap);
  fprintf(stderr, "\n");
}
```

主函数，先输出行引导符号，再调用 yyparse 进行分析。

```
int
main()
{
  printf("> ");
  return yyparse();
}
```

5.4.4 *fb3-1.h*

引用一些必要的头文件，避免报错报警。

```
/* interface to the lexer */
extern int yylineno; /* from lexer */
void yyerror(char *s, ...);
```

定义抽象语法树结构体，包括当前节点类型，以及左右子树的指针。

```
/* nodes in the Abstract Syntax Tree */
struct ast {
  int nodetype;
  struct ast *l;
  struct ast *r;
};
```

定义数字节点的结构体，包括当前节点类型 K，以及数字节点的值。

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

在.*h* 文件声明需要建立的下面几个函数，这些函数已经在.*c* 文件中编写。

```
struct ast *newast(int nodetype, struct ast *l, struct ast *r);
struct ast *newnum(double d);
double eval(struct ast *);
void treefree(struct ast *);
```

5.5 分析移进规约过程及构建抽象语法树

5.5.1 移进规约分析

对于(1+2)-2*6 分析如下。

表 1 对(1+2)-2*6 的移进规约分析

栈	输入	动作
#	(1+2)-(2*6)#	移进
# (	1+2)-(2*6)#	移进
# (1	+2)-(2*6)#	规约 term→NUMBER
# (term	+2)-(2*6)#	规约 factor→term
# (factor	+2)-(2*6)#	规约 exp→factor
# (exp	+2)-(2*6)#	移进
# (exp +	2)-(2*6)#	移进
# (exp + 2	)-(2*6)#	规约 term→NUMBER
# (exp + term	)-(2*6)#	规约 factor→term
# (exp + factor	)-(2*6)#	规约 exp→exp + factor
# (exp	)-(2*6)#	移进
# (exp)	-(2*6)#	规约 term→(exp)
# term	-(2*6)#	规约 factor→term
# factor	-(2*6)#	规约 exp→factor
# exp	-(2*6)#	移进
# exp -	(2*6)#	移进
# exp - (	2*6)#	移进
# exp - (2	*6)#	规约 term→NUMBER
# exp - (term	*6)#	规约 factor→term
# exp - (factor	*6)#	移进
# exp - (factor *	6)#	移进
# exp - (factor * 6	)#	规约 term→NUMBER
# exp - (factor * term	)#	规约 factor→factor＊ term
# exp - (factor	)#	规约 exp→factor
# exp - (exp	)#	移进
# exp - (exp)	#	规约 term→(exp)
# exp - term	#	规约 factor→term
# exp - factor	#	规约 exp→exp − factor
# exp	#	接受

对于 1+2-3*2/5 分析如下。

表 2 对 1+2-3*2/5 的移进规约分析

栈	输入	动作
#	1+2-3*2/5#	移进
# 1	+2-3*2/5#	规约 term→NUMBER
# term	+2-3*2/5#	规约 factor→term
# factor	+2-3*2/5#	规约 exp→factor
# exp	+2-3*2/5#	移进
# exp +	2-3*2/5#	移进
# exp + 2	-3*2/5#	规约 term→NUMBER
# exp + term	-3*2/5#	规约 factor→term
# exp + factor	-3*2/5#	规约 exp→exp + factor
# exp	-3*2/5#	移进
# exp -	3*2/5#	移进
# exp - 3	*2/5#	规约 term→NUMBER
# exp - term	*2/5#	规约 factor→term
# exp - factor	*2/5#	移进
# exp – factor *	2/5#	移进
# exp – factor * 2	/5#	规约 term→NUMBER
# exp – factor * term	/5#	规约 factor→factor＊ term
# exp – factor	/5#	移进
# exp – factor /	5#	移进
# exp – factor / 5	#	规约 term→NUMBER
# exp – factor / term	#	规约 factor→factor＊ term
# exp – factor	#	规约 exp→exp − factor
# exp	#	接受

5.5.2 构建抽象语法树

构建示例如下所示。

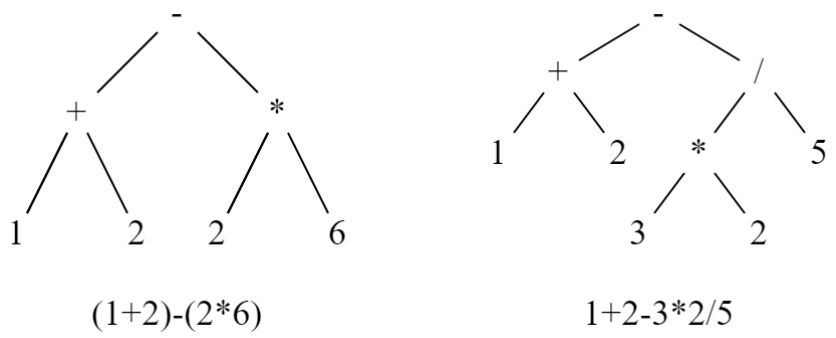

图 4 构建本实验部分抽象语法树

5.6 实验总结

5.6.1 atof 函数报错

虽然自己没有遇到这个错误，但是室友遇到了该问题。在 Windows 环境下，对 *fb3-1.l* 引入头文件<stdlib.h>，因为将字符串转换为浮点型的 atof 函数是在<stdlib.h>中声明的。

5.6.2 评价与收获

这一次实验体验了使用 flex 和 bison 联合编写简单计算器并工作的步骤，对于所学的关于编译的知识有了更加进一步的了解。助人为乐，不仅能帮助他人解决问题，也能在交流中增长自己的见识，在开源的时代更是不能闭门造车，有好的想法或者是不懂的疑问最好可以和大家交流交流，说不定会有新的发现。