

浙江大学

本科实验报告

课程名称：编译原理

姓名：谢文想

学院：竺可桢学院

专业：计算机科学与技术

学号：3190105734

指导教师：李莹

2022年3月4日

利用 YACC 生成中缀表示的计算器

实验目的

了解 YACC 处理二义性的方法。

实验要求

生成如下文法表示的表达式对应的计算器：

$\text{exp} \rightarrow \text{exp} + \text{exp} \mid \text{exp} - \text{exp} \mid \text{exp} * \text{exp} \mid \text{exp} / \text{exp} \mid \text{exp} ^ \text{exp} \mid - \text{exp} \mid (\text{exp}) \mid \text{NUM}$

对于输入的中缀表达式，要给出结果。如 $3 + (4 * 5)$ 结果应为 23。要求能连续处理若干个数学表达式，直到输入结束或文件结束。

代码分析

语法分析 lab2.y

```
1  %{
2  #include <stdio.h>
3  #include <math.h>
4
5  int mypow(int x, int y);
6  int yylex(void);
7  int yyparse(void);
8  void yyerror(const char *);
9  %}
10
11 %token NUMBER '(' ')'
12 %left '+' '-'
13 %left '*' '/'
14 %right '^'
15
16 %%
17 program:
18 program expr '\n' { printf("%d\n", $2); }
19 | program '\n'
20 |
21 ;
22
23 expr:
24 NUMBER { $$ = $1; }
25 | '(' expr ')' { $$ = $2; }
26 | expr '^' expr { $$ = mypow($1, $3); }
27 | expr '*' expr { $$ = $1 * $3; }
28 | expr '/' expr { $$ = $1 / $3; }
29 | expr '+' expr { $$ = $1 + $3; }
30 | expr '-' expr { $$ = $1 - $3; }
31 | '-' expr { $$ = -$2; }
32 ;
33 %%
34
35 void yyerror(const char *s) {
36     printf("%s\n", s);
37 }
38
```

```

39 int mypow(int x, int y)
40 {
41     int ret = 1;
42     for(int i = 0; i < y; i++){
43         ret *= x;
44     }
45     return ret;
46 }
47
48 int main(void) {
49     yyparse();
50     return 0;
51 }

```

yacc文件定义与lex十分相似，分别以%{ }% %% %%分界。

%}和%%这一段看作预定义标记部分，%token NUMBER 定义声明了一个标记，%left 表示左结合，%right 表示右结合。最后列出的定义拥有最高的优先权，因此幂最后定义，乘法和除法比加法和减法后定义，+ - * / 所有这四个算术符都是左结合的，^是右结合的，运用这个我们可以消除文法的歧义。

```

1 program:
2 program expr '\n' { printf("%d\n", $2); }
3 | program '\n'
4 |
5 ;

```

输入可以是空串，使当程序一开始就接收到EOF不致于发生错误，也可以是一行，这一行可以是简单的回车也可以是一个表达式。

```

1 expr:
2 NUMBER { $$ = $1; }
3 | '(' expr ')' { $$ = $2; }
4 | expr '^' expr { $$ = mypow($1, $3); }
5 | expr '*' expr { $$ = $1 * $3; }
6 | expr '/' expr { $$ = $1 / $3; }
7 | expr '+' expr { $$ = $1 + $3; }
8 | expr '-' expr { $$ = $1 - $3; }
9 | '-' expr { $$ = -$2; }
10 ;

```

表达式可以派生为如上这些运算，“\$1”代表式中的第一个成员，“\$2”代表第二个，后面的以此类推，“\$\$”表示缩小后的堆栈顶部。以 expr '+' expr 为例，把对应两个表达式的值相加，弹出内容栈中的三个成员 expr '+' expr，然后把得到的和压入堆栈中。

词法分析 lab2.l

```

1 %{
2 #include "lab2.tab.h"
3 #include <math.h>
4 void yyerror(const char *);
5
6 %}
7
8 %%
9 [0-9]+          { yylval = atoi(yytext); return NUMBER; }

```

```

10  [-+*/^()\n]      return *yytext;
11  [ \t]             ;
12  [q]               return 0;
13  .                 yyerror("无效字符");
14  %%
15
16  int yywrap(void) {
17      return 1;
18  }

```

[0-9]+ 表示NUMBER，yylval返回对应的值（字符串转整型），[-+*/^()\n] 直接返回文本，[\t] 用于去除\t和空格，q用来结束输入（文本输入不需要q）。

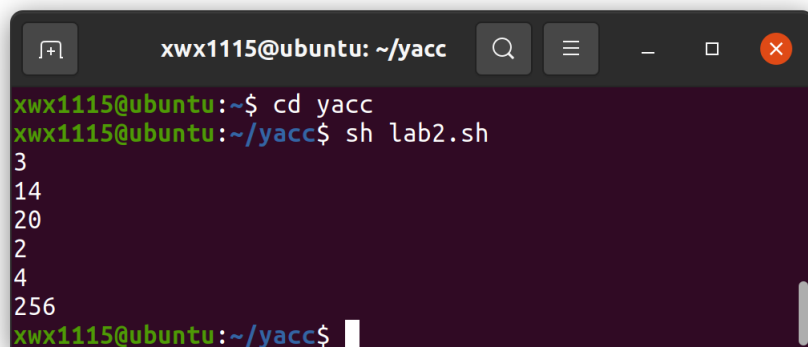
实验结果

我对以下文本进行测试，最后得到的结果如下图：

```

1  1+2
2  2+3*4
3  (2+3)*4
4  (-2+3)*(-6/(-3))
5  (-2+3)*(-6/(-3))^2
6  2^2^3

```




```

xwx1115@ubuntu: ~/yacc
xwx1115@ubuntu:~$ cd yacc
xwx1115@ubuntu:~/yacc$ sh lab2.sh
3
14
20
2
4
256
xwx1115@ubuntu:~/yacc$

```

转换成命令行输入（q结束输入）：



```

xwx1115@ubuntu:~/yacc$ ./parser
1+2
3
2+3*4
14
(2+3)*4
20
(-2+3)*(-6/(-3))
2
(-2+3)*(-6/(-3))^2
4
2^2^3
256
q
xwx1115@ubuntu:~/yacc$

```