# 预备工作 3 熟悉语法分析器辅助构造工具

高钰洋崔立骁

October 2020

# 目录

# 1 作业解析

## 1.1 作业目的

通过本次实验，希望同学们能够熟悉并掌握 Bison 的使用。结合课上所学知识增进对词法分析过程的了解，学会 Bison 程序的编写，巩固所学知识，提高实践能力。Bison 环境的配置请参考之前提供的实验指导书。

## 1.2 作业思路

根据作业要求，可以看出前两步要求返回简单表达式计算的值，而第三步要求要求实现中缀表达式到后缀表达式的转换，因此返回的是字符串类型的表达式。因此建议同学们分写两个程序。第一步作业要求的修改重点在于定义单词类别去替代运算符及整数，将字符流转化为一个一个的 token。第二步是在第一步基础上，加入对更复杂单词的识别代码。第三步作业要求中缀转后缀，重点在于修改翻译模式和返回值，类比识别多位十进制整数完成实现识别标识符，并添加有关标识符的上下文无关文法。

## 1.3 简单的 Yacc 程序讲解

一个实现简单表达式计算的 Yacc 程序如下：

```
1  %{
2  /*************************************************************************
3  hw3.y
4  YACC file
5  Date: 2020/10/01
6  lixiao cui <cuilx@nbjl.nankai.edu.cn>
7  *************************************************************************/
8  #include <stdio.h>
9  #include <stdlib.h>
10 #ifndef YYSTYPE
11 #define YYSTYPE double
12 #endif
13 int yylex();
14 extern int yyparse();
15 FILE* yyin;
16 void yyerror(const char* s);
17 %}
18
19 %left '+' '-'
20 %left '*' '/'
21 %right UMINUS
22
23 %%
24
25
26 lines   :       lines expr '\n' { printf("%f\n", $2); }
27         |       lines '\n'
```

```
28          |
29          ;
30
31  expr    :       expr '+' expr   { $$ = $1 + $3; }
32          |       expr '-' expr   { $$ = $1 - $3; }
33          |       expr '*' expr   { $$ = $1 * $3; }
34          |       expr '/' expr   { $$ = $1 / $3; }
35          |       '(' expr ')'    { $$ = $2; }
36          |       '-' expr %prec UMINUS   { $$ = -$2; }
37          |       NUMBER
38          ;
39
40  NUMBER  :       '0'                     { $$ = 0.0; }
41          |       '1'                     { $$ = 1.0; }
42          |       '2'                     { $$ = 2.0; }
43          |       '3'                     { $$ = 3.0; }
44          |       '4'                     { $$ = 4.0; }
45          |       '5'                     { $$ = 5.0; }
46          |       '6'                     { $$ = 6.0; }
47          |       '7'                     { $$ = 7.0; }
48          |       '8'                     { $$ = 8.0; }
49          |       '9'                     { $$ = 9.0; }
50          ;
51
52  %%
53
54  // programs section
55
56  int yylex()
57  {
58          // place your token retrieving code here
59          return getchar();
60  }
61
62  int main(void)
63  {
64          yyin = stdin;
65          do {
66                  yyparse();
67          } while(!feof(yyin));
68          return 0;
69  }
70  void yyerror(const char* s) {
71          fprintf(stderr, "Parse error: %s\n", s);
72          exit(1);
73  }
```

文件的开头部分称为定义段，用于添加所需的头文件，函数定义以及全局变量等，注释部分可添加描述 yacc 文件信息。其中 YYSTYPE 用于确定 $$ 变量类型，这里由于需要返回的是简单表达式计算结果，暂时确定为 double 类型。接下来需要声明运算符的左右结合性及优先级，按照优先级由低到高的顺序声明，同一级的运算符放在同一行。这里定义了三种优先级，由低到高分别为加减、乘除以及取负。

然后我们定义规则段，即主体的上下文无关文法，扩展了翻译模式，大括号内的部分即为语义动作。$$ 代表产生式左部的属性值，$\$n$ 为产生式右

部第 $n$ 个语法符号的属性值，**注意运算符等终结符也计算其中，即，不只非终结符有属性值，终结符也有属性值**。

yylex 函数部分作用是实现词法分析功能，也就是本次作业着重修改的部分。该部分可以使用独立的词法分析工具生成（如 flex），但在本次作业中，我们要求大家手动编码实现该部分。

## 1.4 表达式计算

首先我们为每个单词定义一种单词类别（作业第一步），以加号为例，用定义名称为 ADD 的 token 表示加号，首先需要声明该 token（第 2 行），然后在规则段涉及加号的位置替换为 ADD（第 7 行），最后在词法分析函数 yylex 写明加号的情况如何处理——将原来简单返回语法分析程序单个字符作为单词的方式修改为识别出各单词（如加号）、将单词类别记号（如 ADD）返回语法分析程序。

```
1  ...
2  %token ADD
3  %left ADD '-'
4  %left '*' '/'
5  %right UMINUS
6  ...
7  expr    :       expr ADD expr   { $$ = $1 + $3; }
8  ...
9  int yylex()
10 {
11         // place your token retrieving code here
12         int t;
13         t=getchar();
14         if(t=='+')
15             return ADD;
16         else
17             return t;
18 }
19 ...
```

其余运算符及整数请同学们举一反三实现。

之后修改实现的是识别忽略空白字符以及识别多位十进制数字（作业第二步），我们可以对空格、制表符、回车和数字分情况处理。需要注意的是以下几点：

a) 回车在原 yacc 程序中语法分析中 lines 的产生式中，由于要实现忽略回车，因此可使用分号替换 lines 产生式中的 '$\backslash n$'，这样在测试程序时，简单表达式以分号结束而不是以回车结束，这样就可以实现由词法分析忽略回车。

b) 在 yacc 中由于声明的 token 都会有一个互不冲突的整数值，因此不需要在 yacc 中进行对 NUMBER 这种 token 进行显式的宏定义，只声明其为 token 即可。

c) yylval 是 Yacc 自动定义的全局变量，类型为 YYSTYPE，在本次作业中定义为 double。yylval 的使用贯穿词法分析和语法分析，可视为符号表的简化原型（同学们可以进一步思考符号表的构造）。删除之前 yacc 文件中 NUMBER 对应的上下文无关文法及翻译模式，将 NUMBER 视为终结符，添加语义动作，将 yylval 的值赋给 NUMBER 的属性值。下文中给出的例子中，NUMBER 的语义动作为 $$ = $1。这里涉及到 yylval 的用法问题，yacc 会将 yylval 隐式赋值给终结符的属性值（此处赋值给 NUMBER 的属性值 $1），所以在语义动作中不需要显式地执行 $$ = yylval 的操作。如果进一步简化这里的语义动作，甚至可以什么都不写，yacc 会自动执行 $$ = $1（对产生式右部只有单个语法符号的情况）。

d) ungetc 函数作用是将读出的多余字符再次放回到缓冲区去，下一次读数字符进行下一个单词的识别时，会再次读出来的。由于判断处理多位十进制数字时，需要读到最新一位为非数字时停止，但这个非数字字符已被读出，因此需要再次放回缓冲区，所以同学们写代码时不要忽略。

完成识别空白符、多位十进制整数后的代码大致如下：

```
1  ...
2  %{
3  ...
4  %}
5
6  %token NUMBER
7  %left '+' '-'
8  %left '*' '/'
9  %right UMINUS
10
11 %%
12
13
14 lines   :       lines expr ';'  { printf("%f\n", $2); }
15         |       lines ';'
16         |
17         ;
18
19 expr    :       expr '+' expr   { $$ = $1 + $3; }
20         |       expr '-' expr   { $$ = $1 - $3; }
21         |       expr '*' expr   { $$ = $1 * $3; }
22         |       expr '/' expr   { $$ = $1 / $3; }
23         |       '(' expr ')'    { $$ = $2; }
```

```
        |       '-' expr %prec UMINUS    { $$ = -$2; }
        |       NUMBER{ $$ = $1; }
        ;

%%

// programs section

int yylex()
{
        // place your token retrieving code here
        int t;
        while (1) {
                t = getchar();
                if (t == ' ' || t == '\t' || t == '\n'){
                    //do nothing
                }else if (isdigit(t)) {
                        yylval = 0;
                        while (isdigit(t)) {
                                yylval = yylval * 10 + t - '0';
                                t = getchar();
                        }
                        ungetc(t, stdin);
                        return NUMBER;
                }else {
                    return t;
                }
        }
}
...
```

完成前两步骤作业，生成并编译（为了调试方便，同学们可以自行编写 makefile）。

```
[clx@nbjl compiler_hw3]$ bison -d hw3.y
[clx@nbjl compiler_hw3]$ ls
hw3.tab.c  hw3.tab.h  hw3.y
[clx@nbjl compiler_hw3]$ gcc -o hw3 hw3.tab.c
```

效果如下 (测试表达式为 2+128/4 和 32-3*5):

```
[clx@nbjl compiler_hw3]$ ./hw3
2+128/4 ;
34.000000
32- 3*5;
17.000000
```

## 1.5 中缀表达式转后缀表达式

由于需要返回的是一个后缀表达式，是一个字符串，因此 YYSTYPE 需要声明为 char*, 而词法分析函数中不仅需要分析运算符，多位十进制整数，空白字符，还需要识别标识符 ID。而对于多位十进制整数，不再需要得到整数值，而是需要得到整数对应的字符串。对于多位十进制整数，当读到一个字符为整数字符时，连续读接下来的数字字符直至读到不为数字字符的

字符，调用 ungetc 函数将读出的非数字字符放回缓冲区，将读到的若干数字字符存为一个字符串, 最后需要在字符串的最后添加结束符\0, 将这个字符串的地址赋给 **yylval**。NUMBER 的翻译模式执行动作就修改为将 yylval 的值拷贝给 NUMBER, 这里需要注意的是，这里翻译模式的代码中并未出现 yylval，却依然完成了赋值操作，这是因为 yacc 与 lex 默认将 yylval 的值赋给了识别出的标识符，也就是例子中的 $1, 所以这里的 strcpy($$,$1) 也等价于 strcpy($$,yylval)。ID 与 NUMBER 思路类似，声明一个名为 ID 的 token 和一个全局的字符串变量 **yylval**, 用于赋给 ID 值，由于标识符是有数字、字母和下划线组成的，而第一个字符不可以为数字，因此思路为当读到一个字符为字母或下划线时，连续读接下来的字符，直到出现一个不是数字或字母或下划线的字符停止，调用 ungetc 函数将读出的非数字非字母非下划线字符放回缓冲区，将读到的若干字符存为一个字符串，将这个字符串的地址赋给 **yylval**。ID 的翻译模式执行动作就修改为将 $1 的值赋给 ID。对于加减乘除、括号、负号、空白字符的词法分析部分不需修改，但语法分析部分需要将计算值修改成字符串的拷贝和连接。

PS. 为了看起来美观避免歧义，在每个运算符、ID 和 NUMBER 之间都用空格分开。

下面我们给出加法操作的代码，其余的减法乘法除法括号等操作的请同学们自行编写添加。

```
1
2  %{
3  /*************************************************************************
4  expr.y
5  YACC file
6  Date: 2020/10/01
7  gyy <gyy@nbjl.nankai.edu.cn>
8  *************************************************************************/
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #ifndef YYSTYPE
13 #define YYSTYPE char*
14 #endif
15 char idStr[50];
16 char numStr[50];
17 int yylex();
18 extern int yyparse();
19 FILE* yyin;
20 void yyerror(const char* s);
21 %}
22 %token NUMBER
23 %token ID
24 %token ADD
25 %left ADD MINUS
26 %left MUL DIV
27 %right UMINUS
```

```
28
29  %%
30
31
32  lines   :       lines expr '\n' { printf("%s\n", $2); }
33          |       lines '\n'
34          |
35          ;
36
37  expr    :       expr ADD expr   { $$ = (char *)malloc(50*sizeof(char)); strcpy($$,$1);
38                                    strcat($$,$3); strcat($$,"+"); }
39          |       NUMBER          { $$ = (char *)malloc(50*sizeof(char));
40                                    strcpy($$, $1);strcat($$," ");}
41          |       ID              { $$ = (char *)malloc(50*sizeof(char));
42                                    strcpy($$, $1);strcat($$," ");}
43          ;
44
45
46
47  %%
48
49  // programs section
50
51  int yylex()
52  {
53          // place your token retrieving code here
54          int t;
55          while (1) {
56                  t = getchar();
57                  if (t == ' ' || t== '\t')
58                          ;
59                  else if (( t >= '0' && t <= '9' )) {
60                          int ti=0;
61                          while (( t >= '0' && t <= '9' )) {
62                                  numStr[ti]=t;
63                                  t = getchar();
64                                  ti++;
65                          }
66                          numStr[ti]='\0';
67                          yylval=numStr;
68                          ungetc(t, stdin);
69                          return NUMBER;
70                  }
71                  else if (( t >= 'a' && t <= 'z' ) || ( t >= 'A' && t<= 'Z' ) || ( t == '_' ) ){
72                          int ti=0;
73                          while (( t >= 'a' && t <= 'z' ) || ( t >= 'A' && t<= 'Z' )
74                          || ( t == '_') || ( t >= '0' && t <= '9' ) ) {
75                                  idStr[ti]=t;
76                                  ti++;
77                                  t = getchar();
78                          }
79                          idStr[ti]='\0';
80                          yylval=idStr;
81                          ungetc(t, stdin);
82                          return ID;
83                  }
84                  else if(t=='+')
85                  return ADD;
86                  else { return t; }
87          }
88  }
89
90  int main(void)
```

```
91  {
92          yyin = stdin;
93
94          do {
95                  yyparse();
96          } while(!feof(yyin));
97          return 0;
98
99  }
100 void yyerror(const char* s) {
101         fprintf(stderr, "Parse error: %s\n", s);
102         exit(1);
103 }
```

作业第三步要求修改后效果如下

```
1  [gyy@409147 bisonflex]$ ./a.out
2  >>> a_5+b78*(ccc-890)
3  >>> a_5 b78 ccc 890 - * +
```

## 1.6 思考——符号表

本部分鼓励学有余力的同学们进行探究实践，不做硬性要求。

### 1.6.1 思考题要求

在第二步要求的基础上，实现功能更强的词法分析和语法分析程序，使之能支持变量，修改词法分析程序，能识别变量（标识符）和 “=” 符号，修改语法分析器，使之能分析、翻译 “a=2” 形式的（或更复杂的，“a= 表达式”）赋值语句，当变量出现在表达式中时，能正确获取其值进行计算（未赋值的变量取 0）。当然，这些都需要实现符号表功能。

### 1.6.2 思考题提示

可以使用数组，vector，map 等数据结构实现符号表。标识符的实现已在作业第三步要求实现过程中讲解，“=” 的识别可类比别的运算符，语法分析部分需要添加相应赋值语句，修改上下文无关文法及翻译模式。

Makefile

```
h0: hw3_0.y
    bison -d hw3_0.y
    gcc hw3_0.tab.c -o hw3_0
h1: hw3_1.y
    bison -d hw3_1.y
    gcc hw3_1.tab.c -o hw3_1
h2: hw3_2.y
    bison -d hw3_2.y
    gcc hw3_2.tab.c -o hw3_2
h3: hw3_3.y
    bison -d hw3_3.y
    gcc hw3_3.tab.c -o hw3_3

```