

用 Yacc 设计语法分析器 1

一、实验目的：

学习如何使用 Yacc 设计一个语法分析器，并与用 lex 写的词法分析器链接起来。

二、实验内容：

使用 yacc 为课程设计实验 1 所给的语言写一个语法分析器（你可以重新设计该语言的文法，但不能改变语言）。其中，词法分析使用课程设计实验 2 中已完成的词法分析器（即，你需要将本实验的语法分析器和实验 2 的词法分析器链接起来）。

三、实验要求：

1. 输入为实验 1 所给语言写的源程序文件；输出为屏幕显示语法分析是否成功。
2. 在语法分析中不能出现任何的冲突（移进-归约或归约-归约冲突），或者虽然有冲突，但是你能说清楚冲突是如何解决的。
3. 在 cygwin 下用 flex, bison 和 gcc 工具将实验调试通过，并且你写的语法分析器至少应该能通过例子 parser0 中 testcases 目录下的 test0.p 和 test1.p 两个测试例的测试。

四、具体实现

Yacc 文件

1. 定义段部分：

```

1  %{
2  #include <stdio.h>
3  #include "util.h"
4  #include "errmsg.h"
5
6  int yylex(void); /* function prototype */
7
8  /* 该函数显示错误信息s，显示时包含了错误发生的位置。*/
9  void yyerror(char *s)
10 {
11     EM_error(EM_tokPos, "%s", s);
12 }
13
14
15 %}
16
17
18 /* 定义属性值栈的类型 */
19 %union {
20     int ival;
21     double fval;
22     string sval;
23 }
24
25 /* 定义各个终结符，以及他们的属性值的类型 */
26 %token <sval> ID /* id */
27 %token <ival> INT /*整型数*/
28 %token <fval> FLOAT /*浮点数*/
29 %token INTEGER REAL /*两种类型名：整型、实型*/
30 %token
31     COMMA COLON SEMICOLON LPAREN RPAREN PERIOD /* 符号 , : ; ( ) . */
32     PROGRAM BEGINN END VAR IF WHILE DO /* 关键字：PROGRAM BEGIN END VAR IF WHILE DO */
33     THEN ELSE /* 关键字：THEN ELSE */
34     ASSIGN EQ NEQ LT LE GT GE /* 符号 := = <> < <= > >= */
35     PLUS MINUS TIMES DIVIDE /* 符号 + = * / */
36 %start program /* 开始符号为program */
37
38 %debug /* 允许跟踪错误 */

```

- 属性值栈和实验三一样，属性值可能是整型数，浮点型数，字符串。其中 `string` 在此是 `char*` 的“别名”，在 `util.h` 头文件中有定义。
- 在定义终结符时，有一个 `BEGINN`，实际对应的是 `BEGIN` 关键字，之所以多加一个 `N` 是因为 `Yacc` 和 `Lex` 中 `BEGIN` 是有定义的，所以我们不能定义，就用 `BEGINN` 代替，在测试样例中也应做出相应的修改。

2. 规则段：

```

49
50 program :    PROGRAM ID SEMICOLON block
51           ; /* 程序 */
52
53 block    :    vardec BEGINN stmtslist END PERIOD
54           ; /* 分程序 */
55
56 vardec   :    VAR declist
57           ; /* 变量说明 */
58
59 declist  :    idlist COLON type SEMICOLON declist_other_part
60           ; /* 变量说明表 */
61
62 declist_other_part : declist
63                       |
64                       ;
65
66 type     :    INTEGER
67           | REAL
68           ; /* 类型 */
69
70 idlist   :    ID
71           | ID COMMA idlist
72           ; /* 变量表 */

```

- 这部分对应的文法是：
 - 程序→PROGRAM <标识符> ; <分程序>
 - <分程序>→<变量说明> BEGIN <语句表> END.
 - <变量说明>→VAR <变量说明表>;
 - <变量说明表>→<变量表>: <类型> | <变量表>: <类型>; <变量说明表>
 - <类型>→INTEGER | REAL
 - <变量表>→<变量> | <变量>, <变量表>
- 其中有所修改，如：“<变量说明>→VAR <变量说明表>;”该条文法是有分号的，但观察到变量说明表的定义，发现这个分号如果分开写的话并不好实现，而且可以合并并在变量说明表中，即：
 - <变量说明>→VAR <变量说明表>
 - <变量说明表>→<变量表>: <类型>; | <变量表>: <类型>; <变量说明表>
- 但是如果这样的话对于变量说明表的定义，容易引起移进-规约冲突，根据观察使用提取左因子的方法解决冲突，最后的文法改成：
 - 程序→PROGRAM <标识符> ; <分程序>
 - <分程序>→<变量说明> BEGIN <语句表> END.
 - <变量说明>→VAR <变量说明表>
 - <变量说明表>→<变量表>: <类型> ; <其他变量说明部分>
 - <其他变量说明部分>→<变量说明表> | <空>
 - <类型>→INTEGER | REAL
 - <变量表>→<变量> | <变量>, <变量表>

74	stmtslist:	stmts	101	term	:	factor
75		stmts SEMICOLON stmtslist	102			term TIMES factor
76		; /* 语句表 */	103			term DIVIDE factor
77			104			; /* 项 */
78	stmts	:	105	factor	:	ID
79		asstmts	106			constant
80		constmts	107			LPAREN aritexp RPAREN
81		whilestmts	108			; /* 因式 */
82		comstmts	109			
83		; /* 语句 */	110	relaexp	:	aritexp LT aritexp
84	asstmts	:	111			aritexp LE aritexp
85		ID ASSIGN aritexp	112			aritexp GT aritexp
86		; /* 赋值语句 */	113			aritexp GE aritexp
87	constmts:	IF relaexp THEN stmts ELSE stmts	114			aritexp EQ aritexp
88		; /* 条件语句 */	115			aritexp NEQ aritexp
89			116			; /* 关系表达式 */
90	whilestmts:	WHILE relaexp DO stmts	117			
91		; /* WHILE语句 */	118	constant:	:	INT
92			119			FLOAT
93	comstmts:	BEGINN stmtslist END	120			; /* 常量 */
94		; /* 复合语句 */	121			
95						
96	aritexp	:				
97		term				
98		aritexp PLUS term				
99		aritexp MINUS term				
		; /* 算术表达式 */				

这部分对应的文法是：

- <语句表>→<语句> | <语句>; <语句表>
- <语句>→<赋值语句> | <条件语句> | <WHILE 语句> | <复合语句>
- <赋值语句>→<变量> := <算术表达式>
- <条件语句>→IF <关系表达式> THEN <语句> ELSE <语句>
- <WHILE 语句>→WHILE <关系表达式> DO <语句>
- <复合语句>→BEGIN <语句表> END
- <算术表达式>→<项> | <算术表达式> + <项> | <算术表达式> - <项>
- <项>→<因式> | <项> * <因式> | <项> / <因式>
- <因式>→<变量> | <常数> | (<算术表达式>)
- <关系表达式>→<算术表达式> <关系符> <算术表达式>
- <变量>→<标识符>
- <标识符>→<标识符><字母> | <标识符><数字> | <字母>
- <常数>→<整数> | <浮点数>

实验结果

```

/cygdrive/e/编译原理2016年秋季学期/课程设计/44
LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/44
$ make
gcc -c parsetest.c
parsetest.c: 在函数'main'中:
parsetest.c:30:68: 警告: 隐式声明函数'exit' [-Wimplicit-function-declaration]
    if (argc!=2) {fprintf(stderr,"usage: parser0.exe inputfile \n"); exit(1);}
                                                                    ^
parsetest.c:30:68: 警告: 隐式声明与内建函数'exit'不兼容
parsetest.c:30:68: 附注: include '<stdlib.h>' or provide a declaration of 'exit'
bison -dv parser.y
gcc -c parser.tab.c
gcc -c lex.yy.c
gcc -c errormsg.c
errormsg.c: 在函数'EM_reset'中:
errormsg.c:79:2: 警告: 隐式声明函数'yyrestart' [-Wimplicit-function-declaration]
    yyrestart(yyin);//0
    ^
gcc -c util.c
gcc -o parser0 parsetest.o parser.tab.o lex.yy.o errormsg.o util.o

LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/44
$ make test00
./parser0.exe testcases/test1.p

Parsing begin: testcases/test1.p

Parse Successful!

Parsing end: testcases/test1.p

LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/44
$

```

- 使用 makefile 文件编译后，产生警告不影响实验结果，忽略，然后对 test1.p 文件测试（样例给的 makefile 文件中是对 test0.p 文件测试，在此对 makefile 的测试文件语句进行了修改。）结果如上，显示分析开始→分析成功→分析结束。
- 在 parsetest.c 文件中作如下修改后在编译执行，即能各种语法分析时的栈的状态，如下图所示。

```

10  /* 语法分析函数，如果分析成功，返回0；分析失败，返回1。
11  * 参数：输入文件名fname，输出文件名output，
12  * 返回值：整数0或1
13  */
14  int parse(string fname) {
15      int yy ;
16      EM_reset(fname, "");
17      EM_yydebug(); /*将本行注释去掉则运行时进入错误跟踪状态。*/
18      yy = yyparse();
19      if (yy == 0) { /* parsing worked */
20          printf("\nParse Successful!\n");
21          return 0;
22      }
23      else return 1;
24  }
```

```
/cygdrive/e/编译原理2016年秋季学期/课程设计/44
LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/44
$ make test00
./parser0.exe testcases/test1.p

Parsing begin: testcases/test1.p
Starting parse
Entering state 0
Reading a token: Next token is token PROGRAM ()
Shifting token PROGRAM ()
Entering state 1
Reading a token: Next token is token ID ()
Shifting token ID ()
Entering state 3
Reading a token: Next token is token SEMICOLON ()
Shifting token SEMICOLON ()
Entering state 5
Reading a token: Next token is token VAR ()
Shifting token VAR ()
Entering state 6
Reading a token: Next token is token ID ()
Shifting token ID ()
Entering state 9
Reading a token: Next token is token COMMA ()
Shifting token COMMA ()
Entering state 13
Reading a token: Next token is token ID ()
Shifting token ID ()
Entering state 9
Reading a token: Next token is token COMMA ()
Shifting token COMMA ()
Entering state 13
Reading a token: Next token is token ID ()
Shifting token ID ()
Entering state 9
Reading a token: Next token is token COLON ()
Reducing stack by rule 9 (line 70):
    $1 = token ID ()
-> $$ = nterm idlist ()
Stack now 0 1 3 5 6 9 13 9 13
Entering state 25
Reducing stack by rule 10 (line 71):
    $1 = token ID ()
    $2 = token COMMA ()
    $3 = nterm idlist ()
-> $$ = nterm idlist ()
Stack now 0 1 3 5 6 9 13
Entering state 25
Reducing stack by rule 10 (line 71):
    $1 = token ID ()
    $2 = token COMMA ()
    $3 = nterm idlist ()
-> $$ = nterm idlist ()
Stack now 0 1 3 5 6
Entering state 11
```

由于篇幅问题，不对整个分析过程进行截图。

```
/cygdrive/e/编译原理2016年秋季学期/课程设计/44
$3 = nterm stmtslist ()
-> $$ = nterm stmtslist ()
Stack now 0 1 3 5 8 12 20 42 20 42
Entering state 60
Reducing stack by rule 12 (line 75):
  $1 = nterm stmts ()
  $2 = token SEMICOLON ()
  $3 = nterm stmtslist ()
-> $$ = nterm stmtslist ()
Stack now 0 1 3 5 8 12 20 42
Entering state 60
Reducing stack by rule 12 (line 75):
  $1 = nterm stmts ()
  $2 = token SEMICOLON ()
  $3 = nterm stmtslist ()
-> $$ = nterm stmtslist ()
Stack now 0 1 3 5 8 12
Entering state 19
Next token is token END ()
Shifting token END ()
Entering state 41
Reading a token: Next token is token PERIOD ()
Shifting token PERIOD ()
Entering state 59
Reducing stack by rule 2 (line 53):
  $1 = nterm vardec ()
  $2 = token BEGINN ()
  $3 = nterm stmtslist ()
  $4 = token END ()
  $5 = token PERIOD ()
-> $$ = nterm block ()
Stack now 0 1 3 5
Entering state 7
Reducing stack by rule 1 (line 50):
  $1 = token PROGRAM ()
  $2 = token ID ()
  $3 = token SEMICOLON ()
  $4 = nterm block ()
-> $$ = nterm program ()
Stack now 0
Entering state 2
Reading a token: Now at end of input.
Shifting token $end ()
Entering state 4
Stack now 0 2 4
Cleanup: popping token $end ()
Cleanup: popping nterm program ()

Parse Successful!

Parsing end: testcases/test1.p
LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/44
$
```

- 设置对分析栈观察，可以看出各个词法单元进栈以及根据文法规则规约的情况。由于篇幅问题，不对这个过程进行截图。
- 下图是分析出错的情况，对测试文件 test1.p 文件修改，把第一个 BEGIN 关键字去掉。其中第一幅图是不显示分析栈的情况，第二幅图是分析过程错误跟踪状态。

```
LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/44
$ make test00
./parser0.exe testcases/test1.p

Parsing begin: testcases/test1.p
testcases/test1.p:5.5: syntax error

Parsing end: testcases/test1.p
LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/44
$ !
```



```
/cygdrive/e/编译原理2016年秋季学期/课程设计/44
Shifting token REAL ()
Entering state 27
Reducing stack by rule 8 (line 67):
    $1 = token REAL ()
-> $$ = nterm type ()
Stack now 0 1 3 5 6 11 14 28 43 11 14
Entering state 28
Reading a token: Next token is token SEMICOLON ()
Shifting token SEMICOLON ()
Entering state 43
Reading a token: Next token is token ID ()
Shifting token ID ()
Entering state 9
Reading a token: Next token is token ASSIGN ()
Reducing stack by rule 9 (line 70):
    $1 = token ID ()
-> $$ = nterm idlist ()
Stack now 0 1 3 5 6 11 14 28 43 11 14 28 43
Entering state 11
Next token is token ASSIGN ()
testcases/test1.p:5.5: syntax error
Error: popping nterm idlist ()
Stack now 0 1 3 5 6 11 14 28 43 11 14 28 43
Error: popping token SEMICOLON ()
Stack now 0 1 3 5 6 11 14 28 43 11 14 28
Error: popping nterm type ()
Stack now 0 1 3 5 6 11 14 28 43 11 14
Error: popping token COLON ()
Stack now 0 1 3 5 6 11 14 28 43 11
Error: popping nterm idlist ()
Stack now 0 1 3 5 6 11 14 28 43
Error: popping token SEMICOLON ()
Stack now 0 1 3 5 6 11 14 28
Error: popping nterm type ()
Stack now 0 1 3 5 6 11 14
Error: popping token COLON ()
Stack now 0 1 3 5 6 11
Error: popping nterm idlist ()
Stack now 0 1 3 5 6
Error: popping token VAR ()
Stack now 0 1 3 5
Error: popping token SEMICOLON ()
Stack now 0 1 3
Error: popping token ID ()
Stack now 0 1
Error: popping token PROGRAM ()
Stack now 0
Cleanup: discarding lookahead token ASSIGN ()
Stack now 0

Parsing end: testcases/test1.p
LittleSec@DESKTOP-5CJHD10 /cygdrive/e/编译原理2016年秋季学期/课程设计/44
$
```

- 对于不跟踪的情况，分析出错会告知第几行出错，删除的第一个 BEGIN 是在第 4 行，图中显示是分析到第五行时出错。即实验结果正确。
- 而对于跟踪的情况就需要细心看完分析过程，通过栈的状态找出错误。

五、心得与体会

1. 通过对参考例子的分析进一步了解 Yacc 的作用，对语法分析器有进一步的认识。
2. 实验只需要对 Yacc 文件修改，其他文件都是参考例子已经提供好的了，尤其是 Lex 文件，这样就减轻了大量的工作。在实验前应当认真仔细的看 readme 文档，这样就知道 Lex 文件不需要自己写。一开始 Lex 文件是自己写的，经常出错。实际上 Yacc 文件中定义的终结符也正是 Lex 中所用的，所以既然给出了这部分，说明 Lex 文件已经有了，不需要自己写。

3. 实验前通过看 `readme` 文档清楚了其他 C 源程序的作用，通过大致的浏览可以知道这些源程序的具体功能，有利于实验的开展，例如 `parsetest.c` 对错误追踪的函数。
4. 在语法分析中不能出现任何的冲突（移进-归约或归约-归约冲突），或者虽然有冲突，但是需要想办法解决冲突。例如对变量说明表这个文法的修改，实际上不修改的话除了不直观，还会有冲突。
5. 进一步掌握了 Yacc 与 Lex 的联合使用。