

J Compiler

Using flex and bison

CSIE, NDHU 2018

高英皓

The problem description.

Use flex and bison to implement a front end (including a lexical analyzer and a syntax recognizer) of the compiler for the j programming language, which is a simplified version of Java and specially designed as a compiler construction project by Professor Chung Yung.

Program Highlight

```
|
| { printf("***** Parsing failed!\n"); }
|
;

mainc : CLASS ID LBP PUB STATIC VOID MAIN LP STR LSP RSP ID RP LBP stmts RBP RBP
| { printf("MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp Statemet* rbp\n"); }
;

cdcls : cdcl cdcls
| { printf("(for ClassDecl*) cdcls : cdcl cdcls\n"); }
| { printf("(for ClassDecl*) cdcls : \n"); }
;

cdcl : CLASS ID LBP vdcls mdcls RBP
| { printf("ClassDecl -> class id lbp VarDecl* MethodDecl* rbp\n"); }
;

vdcls : vdcl vdcls
| { printf("(for VarDecl*) vdcls : vdcl vdcls\n"); }
| { printf("(for VarDecl*) vdcls : \n"); }
;

vdcl : type ID SEMI
| { printf("VarDecl -> Type id semi\n"); }
;

mdcls : mdcl mdcls
| { printf("(for MethodDecl*) mdcls : mdcl mdcls\n"); }
| { printf("(for MethodDecl*) mdcls : \n"); }
;

mdcl : PUB type ID LP formals RP LBP vdcls stmts RETURN exp SEMI RBP
| { printf("MethodDecl -> public Type id lp FormalList rp lbp Statements* return Exp semi rbp\n"); }
;

formals : type ID frest
| { printf("FormalList -> Type id FormalRest*\n"); }
| { printf("FormalList -> \n"); }
;

frest : COMMA type ID frest
| { printf("FormalRest -> comma Type id FormalRest\n"); }
| { printf("FormalRest -> \n"); }
;
```

Program Listing:

J_lex.l

```
1  %{
2  #include "j_lex.h"
3  #include "j_parse.h"
4  %}
5
6  ID [A-Za-z][A-Za-z0-9_]*
7  LIT [0-9][0-9]*
8  NONNL [^\n]
9
10 %%
11
12 class {return CLASS;}
13 public {return PUB;}
14 static {return STATIC;}
15 String {return STR;}
16 void {return VOID;}
17 main {return MAIN;}
18 System.Out.println {return PRINT;}
19 int {return INT;}
20 boolean {return BOOLEAN;}
21 true {return TRUE;}
22 false {return FALSE;}
23 if {return IF;}
24 else {return ELSE;}
25 while {return WHILE;}
26 new {return NEW;}
27 return {return RETURN;}
28 this {return THIS;}
29 "&&" {return AND;}
30 "||" {return OR;}
31 "<" {return LT;}
32 "<=" {return LE;}
33 "==" {return EQ;}
34 "+" {return ADD;}
35 "-" {return MINUS;}
36 "*" {return TIMES;}
37 "(" {return LP;}
38 ")" {return RP;}
39 "[" {return LSP;}
40 "]" {return RSP;}
41 "{" {return LBP;}
42 "}" {return RBP;}
43 "," {return COMMA;}
44 "." {return SEMI;}
45 "=" {return ASSIGN;}
46 "." {return DOT;}
47 "!" {return NOT;}
48 "/*" {NONNL}* {}
49
50
51 {LIT} {return LIT;}
52 {ID} {return ID;}
53
54 [ \t\n] {}
55 . {}
56
57 %%
58
59 int yywrap() {return 1;}
60
61 /*
62 void print_lex( int t ) {
63     switch( t ) {
64         case ID: printf("ID(%s)\n", name);
65         break;
66         case LIT: printf("LIT(%d)\n", val);
67         break;
68         case AND: printf("AND\n");
69         break;
70         case LT: printf("LT\n");
71         break;
72         case ADD: printf("ADD\n");
73         break;
74         case MINUS: printf("MINUS\n");
75         break;
76         case TIMES: printf("TIMES\n");
77         break;
78         case LP: printf("LP\n");
79         break;
80         case RP: printf("RP\n");
81         break;
82         case LSP: printf("LSP\n");
83         break;
84         case RSP: printf("RSP\n");
85         break;
86         case LBP: printf("LBP\n");
87         break;
88         case RBP: printf("RBP\n");
89         break;
90         case SEMI: printf("SEMI\n");
91         break;
92         case COMMA: printf("COMMA\n");
93         break;
94         case ASSIGN: printf("ASSIGN\n");
95         break;
96         case DOT: printf("DOT\n");
97         break;
98         case COMMENT: printf("COMMENT (should be skipped)\n");
99         break;
100         case CLASS: printf("CLASS\n");
101         break;
102         case PUB: printf("PUBLIC\n");
103         break;
104         case STATIC: printf("STATIC\n");
105         break;
106         case VOID: printf("VOID\n");
107         break;
108         case MAIN: printf("MAIN\n");
109         break;
110         case INT: printf("INT\n");
111         break;
112         case IF: printf("IF\n");
113         break;
114         case ELSE: printf("ELSE\n");
115         break;
116         case WHILE: printf("WHILE\n");
117         break;
118         case NEW: printf("NEW\n");
119         break;
120         case RETURN: printf("RETURN\n");
121         break;
122         case THIS: printf("THIS\n");
123         break;
124         default: printf("***** lexical error!!!");
125     }
126 }
127 */
```

J_parse.y

```

1 %include <stdio.h>
2 %include <stdlib.h>
3 %include "j_lex.h"
4 %include "j_parse.h"
5 %}
6
7 %token CLASS PUB STATIC
8 %left AND OR
9 %left LT LE EQ
10 %left ADD MINUS
11 %left TIMES
12 %token LBP RBP LSP RSP LP RP
13 %token INT NUM BOOLEAN
14 %token TRUE FALSE NOT
15 %token IF ELSE
16 %token WHILE PRINT
17 %token ASSIGN
18 %token VOID MAIN STR
19 %token RETURN
20 %token SEMI COMMA
21 %token THIS NEW DOT
22 %token ID LIT
23 %token COMMENT
24
25 %%
26 prog : mainc cdds
27 { printf("Program -> MainClass ClassDed\n");
28   printf("Parsed OK!\n"); }
29
30 |
31 { printf("***** Parsing failed!\n"); }
32 ;
33
34 stmts : stmt stmts
35 { printf("(for Statement*) stmts : stmt stmts\n"); }
36
37 |
38 { printf("(for Statement*) stmts : \n"); }
39 ;
40
41 mainc : CLASS ID LBP PUB STATIC VOID MAIN LP STR LSP RSP ID RP LBP stmts RBP RBP
42 { printf("MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp stmts rbp rbp\n"); }
43 ;
44
45 cdds : cdd cdds
46 { printf("(for ClassDed*) cdds : cdd cdds\n"); }
47
48 |
49 { printf("(for ClassDed*) cdds : \n"); }
50 ;
51
52 cdd : CLASS ID LBP vdds mdds RBP
53 { printf("ClassDed -> class id lbp MethodDed* rbp\n"); }
54 ;
55
56 vdds : vdd vdds
57 { printf("(for VarDed*) vdds : vdd vdds\n"); }
58
59 |
60 { printf("(for VarDed*) vdds : \n"); }
61 ;
62
63 vdd : type ID SEMI
64 { printf("VarDed -> Type id semi\n"); }
65 ;
66
67 mdds : mdd mdds
68 { printf("(for MethodDed*) mdds : mdd mdds\n"); }
69
70 |
71 { printf("(for MethodDed*) mdds : \n"); }
72 ;
73
74 mdd : PUB type ID LP formals RP LBP stmts RETURN exp SEMI RBP
75 { printf("MethodDed -> public Type id lp FormalList rp lbp Statements* return Exp semi rbp\n"); }
76 ;
77
78 formals : type ID frest
79 { printf("FormalRest -> Type id FormalRest\n"); }
80
81 |
82 { printf("FormalList -> \n"); }
83 ;
84
85 frest : COMMA type ID frest
86 { printf("FormalRest -> comma Type id FormalRest\n"); }
87
88 |
89 { printf("FormalRest -> \n"); }
90 ;
91
92 type : INT LSP RSP
93 { printf("type -> int lsp rsp\n"); }
94
95 | BOOLEAN
96 { printf("type -> BOOLEAN\n"); }
97
98 | INT
99 { printf("type -> INT\n"); }
100
101 | ID
102 { printf("type -> ID\n"); }
103 ;
104
105 stmt : LBP stmts RBP
106 { printf("stmt -> LBP stmts RBP\n"); }
107
108 | IF LP exp RP stmts ELSE stmts
109 { printf("stmt -> IF LP exp RP stmts ELSE stmts\n"); }
110
111 | WHILE LP exp RP stmts
112 { printf("stmt -> WHILE LP exp RP stmts\n"); }
113
114 | PRINT LP exp RP SEMI
115 { printf("stmt -> PRINT LP exp RP SEMI\n"); }
116
117 | ID ASSIGN exp SEMI
118 { printf("stmt -> ID ASSIGN exp SEMI\n"); }
119
120 | ID LSP exp RSP ASSIGN exp SEMI
121 { printf("stmt -> ID LSP exp RSP EQ exp SEMI\n"); }
122
123 | vdd
124 { printf("VOCL\n"); }
125 ;
126
127 relop : ADD
128 { printf("ADD"); }
129
130 | MINUS
131 { printf("MINUS"); }
132
133 | TIMES
134 { printf("TIMES"); }
135
136 | OR
137 { printf("OR"); }
138
139 | AND
140 { printf("AND"); }
141
142 | LT
143 { printf("LT"); }
144
145 | LE
146 { printf("LE"); }
147
148 | EQ
149 { printf("EQ"); }
150 ;
151
152 exp : exp relop exp
153 { printf("relop"); }
154
155 | ID LSP exp RSP
156 { printf("LSP exp RSP"); }
157
158 | ID LP explst RP
159 { printf("LP explst RP"); }
160
161 | LP exp RP
162 { printf("LP exp RP"); }
163
164 | LIT
165 { printf("LIT"); }
166
167 | TRUE
168 { printf("TRUE"); }
169
170 | FALSE
171 { printf("FALSE"); }
172
173 | ID
174 { printf("ID"); }
175
176 | THIS
177 { printf("THIS"); }
178
179 | NEW INT LSP exp RSP
180 { printf("NEW INT LSP exp RSP"); }
181
182 | NEW ID LP RP
183 { printf("NEW ID LP RP"); }
184
185 | NOT exp
186 { printf("NOT"); }
187 ;
188
189 explsts : explst explsts
190 { printf("(for ExprRest*) explsts : explst explsts\n"); }
191
192 |
193 { printf("(for ExprRest*) explsts : \n"); }
194 ;
195
196 explst : exp explsts
197 { printf("ExpRest -> Exp RspRest* \n"); }
198 ;
199
200 exprest : COMMA exp
201 { printf("ExpRest -> comma Exp \n"); }
202 ;
203
204 %%
205
206 int yyerror(char *s)
207 {
208   printf("%s\n", s);
209   return 1;
210 }

```

Main.c

```

1 #include <stdio.h>
2 #include "j_lex.h"
3 #include "j_parse.h"
4
5 char name[16];
6 int val;
7
8 int main(int argc, char *argv[]) {
9     int t;
10
11     yyin = fopen(argv[1], "r");
12     yyparse();
13 }
14

```

J_parse.h

```
37 # define YYTOKENTYPE
38 /* Put the tokens into the symbol table, so that GDB and other debuggers
39    know about them. */
40 enum yytokentype {
41     CLASS = 258,
42     PUB = 259,
43     STATIC = 260,
44     OR = 261,
45     AND = 262,
46     EQ = 263,
47     LE = 264,
48     LT = 265,
49     MINUS = 266,
50     ADD = 267,
51     TIMES = 268,
52     LBP = 269,
53     RBP = 270,
54     LSP = 271,
55     RSP = 272,
56     LP = 273,
57     RP = 274,
58     INT = 275,
59     NUM = 276,
60     BOOLEAN = 277,
61     TRUE = 278,
62     FALSE = 279,
63     NOT = 280,
64     IF = 281,
65     ELSE = 282,
66     WHILE = 283,
67     PRINT = 284,
68     ASSIGN = 285,
69     VOID = 286,
70     MAIN = 287,
71     STR = 288,
72     RETURN = 289,
73     SEMI = 290,
74     COMMA = 291,
75     THIS = 292,
76     NEW = 293,
77     DOT = 294,
78     ID = 295,
79     LIT = 296,
80     COMMENT = 297
81 };
82 #endif
83
84
85
86 #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
87 typedef int YYSTYPE;
88 # define YYSTYPE_IS_TRIVIAL 1
89 # define YYSTYPE YYSYNTAX_TYPE /* obsolescent; will be withdrawn */
90 # define YYSTYPE_IS_DECLARED 1
91 #endif
92
93 extern YYSTYPE yylval;
```

MAKEFILE

```
1 # 6
2 jparse: main.o j_parse.o j_lex.o
3     gcc -o jparse main.o j_parse.o j_lex.o
4
5 debug:
6     bison -d --report=all -o j_parse.c j_parse.y
7
8 # 1
9 j_parse.c: j_parse.y
10     bison -d -o j_parse.c j_parse.y
11
12 j_parse.h: j_parse.y
13     bison -d -o j_parse.c j_parse.y
14
15 # 2
16 j_parse.o: j_parse.c j_lex.h j_parse.h
17     gcc -c -o j_parse.o j_parse.c
18
19 # 3
20 j_lex.c: j_lex.l
21     flex -oj_lex.c j_lex.l
22 # 4
23 j_lex.o: j_lex.c j_lex.h j_parse.h
24     gcc -c -o j_lex.o j_lex.c
25
26 # 5
27 main.o: main.c j_lex.h j_parse.h
28     gcc -c -o main.o main.c
29
30 clean:
31     rm *.o j_lex.c j_parse.c j_parse.h jparse
32
```

Test and Result:

```
C:\Users\ndhucsie\Desktop\J-Compiler-master>jparse.exe test1.j
NEW ID LP RPDOTLIT<for ExpRest*> exprests :
ExpList -> Exp RxpRest*
LP explist RPrelopstmt -> PRINT LP exp RP SEMI
<for Statement*> stmts :
<for Statement*> stmts : stmt stmts
MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp st
mts rbp rbp
<for VarDecl*> vdcls :
type -> INT
type -> INT
FormalRest ->
FormalList -> Type id FormalRest*
type -> INT
VarDecl -> Type id semi
UDCL
IDLTLITrelopLITstmt -> ID ASSIGN exp SEMI
<for Statement*> stmts :
<for Statement*> stmts : stmt stmts
IDTIMESTHISDOTIDMINUSLITrelop<for ExpRest*> exprests :
ExpList -> Exp RxpRest*
LP explist RPrelopLP exp RPrelopstmt -> ID ASSIGN exp SEMI
<for Statement*> stmts :
<for Statement*> stmts : stmt stmts
stmt -> IF LP exp RP stmts ELSE stmts
<for Statement*> stmts :
<for Statement*> stmts : stmt stmts
<for Statement*> stmts : stmt stmts
IDMethodDecl -> public Type id lp FormalList rp lbp Statements* return Exp semi
rbp
<for MethodDecl*> mdcls :
<for MethodDecl*> mdcls : mdcl mdcls
ClassDecl -> class id lbp MethodDecl* rbp
```

test1.j

```
1 class Factorial {
2     public static void main ( String[] a) {
3         System.out.println(new Fac().ComputeFac(10));
4     }
5 }
6
7 // Fac
8 class Fac {
9     public int ComputeFac ( int num ) {
10         int num_aux;
11         if (num < 1)
12             num_aux = 1;
13         else
14             num_aux = num * (this.ComputeFac( num-1 ));
15         return num_aux;
16     }
17 }
```

test2.j

```
1 class Factorial {  
2     public static void main ( String[] a) {  
3         System.Out.println( 10 );  
4     }  
5 }
```

Discussion:

在實作這次的程式之前，都要把所有 ERROR 和 Warning 全部處理，不論是 C/C++ 或 Python 等等。這次 BISON 中 shift/reduce 和 reduce/reduce 的 Warning 可以 expect，真是覺得通體舒暢。這次除了更熟練 BISON 和 FLEX，也更了解正規語言的細節。最後感謝助教上機的時候不斷解決我的疑惑，尤其是系統環境變數的地方，如果不做修改，甚至連 BISON 都沒辦法編譯。