```
# ========================================================================

DIR = parser
FILES = Makefile parser.y scanner.l

# ========================================================================

parser: parser.o scanner.o
        gcc -O -o parser parser.o scanner.o -lfl

parser.o:       parser.c
        gcc -O -c parser.c

parser.c:       parser.y
        bison -d parser.y
        mv parser.tab.c parser.c
        mv parser.tab.h parser.h

scanner.o:      scanner.c
        gcc -O -c scanner.c

scanner.c:      scanner.l
        flex scanner.l
        mv lex.yy.c scanner.c

# ========================================================================

install:        parser
        install parser /usr/local/bin

# ========================================================================

pdf:    $(FILES)
        a2ps --medium=A4 --line-numbers=1 $(FILES) -o $(DIR).ps
        ps2pdf -sPAPERSIZE=a4 $(DIR).ps $(DIR).pdf
        @rm -f $(DIR) $(DIR).ps

# ========================================================================

clean:
        @rm -rf .*~ *~ parser parser.c parser.h scanner.c *.o *.ps *.pdf

# ========================================================================
```

```
/* ============================================================================ */

%{

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

/* ============================================================================ */

extern FILE *yyin;
extern char *yytext;
extern int source_line_no;
/* ============================================================================ */

int yyerror(char *message);

%}

/* ============================================================================ */

%start program

%token VOID INT IF ELSE WHILE RETURN INPUT OUTPUT
%token PLUS MINUS MULTIPLY DIVIDE LT LE GT GE EQ NE
%token ASSIGN COMMA SEMICOLON LPAR RPAR LBRACE RBRACE LBRACKET RBRACKET
%token ID NUM UNDEFINED
//...

%%

/* ============================================================================ */

program
  : var_declaration_list fun_declaration_list
;

var_declaration_list
  : var_declaration_list var_declaration
  | empty
;

fun_declaration_list
    : fun_declaration_list fun_declaration
    | fun_declaration
;

var_declaration
    : type_specifier var SEMICOLON
    | type_specifier var LBRACKET num RBRACKET SEMICOLON
;

type_specifier
    : INT
    | VOID
;

var
    : ID
;

num
    : NUM
;

```



```
fun_declaration
    : type_specifier var LPAR params RPAR LBRACE local_declarations statement_li
st RBRACE
;

params
    : param_list
    | VOID
;


param_list
    : param_list COMMA param
    | param
;



param
    : type_specifier var
    | type_specifier var LBRACKET RBRACKET
;


local_declarations
    : local_declarations var_declaration
    | empty
;


statement_list
    : statement_list statement
    | empty
;


statement
    : compound_stmt
    | expression_stmt
    | selection_stmt
    | iteration_stmt
    | funcall_stmt
    | return_stmt
    | input_stmt
    | output_stmt
;


compound_stmt
    : LBRACE statement_list RBRACE
;


expression_stmt
    : expression SEMICOLON
    | SEMICOLON
;


expression
    : var ASSIGN expression
    | var LBRACKET expression RBRACKET ASSIGN expression
    | simple_expression
;

```

```
132 simple_expression
133     : additive_expression relop additive_expression
134     | additive_expression
135 ;
136
137
138
139 relop
140     : LT
141     | LE
142     | GT
143     | GE
144     | EQ
145     | NE
146 ;
147
148
149 additive_expression
150     : additive_expression addop term
151     | term
152 ;
153
154
155 addop
156     : PLUS
157     | MINUS
158 ;
159
160
161 term
162     : term mulop factor
163     | factor
164 ;
165
166
167 mulop
168     : MULTIPLY
169     | DIVIDE
170 ;
171
172
173 factor
174     : LPAR expression RPAR
175     | var
176     | var LBRACKET expression RBRACKET
177     | num
178     | PLUS num
179     | MINUS num
180 ;
181
182 selection_stmt
183     : IF LPAR expression RPAR statement ELSE statement
184 ;
185
186 iteration_stmt
187     : WHILE LPAR expression RPAR statement
188 ;
189
190 funcall_stmt
191     : var ASSIGN call
192     | var LBRACKET expression RBRACKET ASSIGN call
193     | call
194 ;
195
196
197 call
```



```
198     : var LPAR args RPAR
199 ;
200
201 args
202     : arg_list
203     | empty
204 ;
205
206 arg_list
207     : arg_list COMMA expression
208     | expression
209 ;
210
211 return_stmt
212     : RETURN SEMICOLON
213     | RETURN expression SEMICOLON
214 ;
215
216 input_stmt
217     : INPUT var SEMICOLON
218     | INPUT var LBRACKET expression RBRACKET SEMICOLON
219 ;
220
221 output_stmt
222     : OUTPUT expression SEMICOLON
223 ;
224
225 empty
226     :
227 ;
228
229
230
231 //...
232
233
234 %%
235
236 /* ========================================================================== */
237
238 int yyerror(char *message)
239 {
240   fprintf(stderr, "line %d: %s at \"%s\"\n", source_line_no, message, yytext);
241 }
242
243 /* ========================================================================== */
244
245 int main(int argc, char *argv[])
246 {
247   if(argc != 2) {
248     fprintf(stderr, "usage: parser file\n");
249     exit(1);
250   }
251   yyin = fopen(argv[1], "r");
252   if(yyin == NULL) {
253     fprintf(stderr, "%s: %s\n", argv[1], strerror(errno));
254     exit(1);
255   }
256   yyparse();
257
258   return 0;
259 }
260
261 /* ========================================================================== */
```

```
/*=========================================*/

%{

#include "parser.h"
#include <string.h>

/* ========================================================================== */

#define ACCEPT(x) return(x)
#define ACCEPT_LEX(x) \
{ \
        yytext[yyleng] ='\0'; \
        lex = malloc(yyleng + 1); \
        strcpy(lex, yytext); \
        return(x); \
}
int source_line_no = 1;

/* ========================================================================== */

char *lex;                      /* current lexeme of ID or NUM */

/* ========================================================================== */

%}

digit                           [0-9]
letter                          [a-zA-Z]

%%

"void"                          ACCEPT(VOID);
"int"                           ACCEPT(INT);


"if"                            ACCEPT(IF);
"else"                          ACCEPT(ELSE);
"while"                         ACCEPT(WHILE);
"return"                        ACCEPT(RETURN);
"input"                         ACCEPT(INPUT);
"output"                        ACCEPT(OUTPUT);


"+"                                     ACCEPT(PLUS);
"−"                                     ACCEPT(MINUS);
"*"                                     ACCEPT(MULTIPLY);

"/"                                     ACCEPT(DIVIDE);
"<"                                     ACCEPT(LT);
"<="                            ACCEPT(LE);
">"                                     ACCEPT(GT);
">="                            ACCEPT(GE);
"=="                            ACCEPT(EQ);
"!="                            ACCEPT(NE);

"="                                     ACCEPT(ASSIGN);
","                                     ACCEPT(COMMA);
";"                                     ACCEPT(SEMICOLON);
"("                                     ACCEPT(LPAR);
")"                                     ACCEPT(RPAR);

"{"                                     ACCEPT(LBRACE);
"}"                                     ACCEPT(RBRACE);
"["                                     ACCEPT(LBRACKET);
"]"                                     ACCEPT(RBRACKET);
```



```


" "       ;
"\t"      ;
"\r"      ;
"\n"      source_line_no++;


"/*"     comments();
"//"      line_comments();



({letter}|_)({letter}|{digit}|_)*       ACCEPT_LEX(ID);
{digit}{digit}*                         ACCEPT_LEX(NUM);

.                                       ACCEPT(UNDEFINED);

%%

/* ========================================================================== */
void comments(){
        int c;
        for(;;){
                while((c=input())!='*' && c!=EOF){
                        if(c=='\n')
                                source_line_no++;
                        continue;
                }

                if(c=='*')
                {
                        while((c=input()) == '*'){
                                if(c=='\n')
                                        source_line_no++;
                        }
                        if(c=='/')
                                break;
                }
        }
}
void line_comments(){
        int c;
        for(;;){
                while((c=input())!='\n' && c!=EOF)
                ;
                if(c=='\n'){
                        source_line_no++;
                        break;
                }
        }
}

/* ========================================================================== */


```