```
%{
#include <stdio.h>
#include <stdlib.h>
#define YYDEBUG 1

#define TIP_INT 1
#define TIP_REAL 2
#define TIP_CAR 3

double stiva[20];
int sp;

void push(double x)
{ stiva[sp++]=x; }

double pop()
{ return stiva[--sp]; }

%}

%union {
        int l_val;
        char *p_val;
}

%token of
%token than
%token let
%token be
```

```
%token show
%token read
%token is
%token START
%token END
%token if
%token else
%token while
%token then
%token for
%token do
%token until
%token done
%token PROGRAM
%token CONST
%token VAR

%token ID
%token <p_val> number
%token <p_val> str
%token <p_val> bool
%token list

%token lower
%token greater
%token '='
%token '(' ')'
%token '{' '}'
%token '[' ']'
```

```
%token ':'
%token ';'
%token ','

%left sum diff '+' '-'
%left multiple div '*' '/'
%left or
%left and
%left not

%type <l_val> expr_stat factor_stat constanta
%%
prog_sursa:     PROGRAM ID ';' bloc '.'
                ;
bloc:           sect_const sect_var instr_comp
                ;
sect_const:     /* empty */
                | CONST lista_const
                ;
lista_const:    decl_const
                | lista_const decl_const
                ;
sect_var:       /* empty */
                | VAR lista_var
                ;
lista_var:      decl_var
                | lista_var decl_var
                ;
decl_const:     ID '=' {sp=0;} expr_stat ';'          {
```

```
                printf("*** %d %g ***\n", $4, pop());
                                        }
                ;
decl_var:       lista_id ':' tip ';'
                ;
lista_id:ID
                | lista_id ',' ID
                ;
tip:            tip_simplu
                ;
tip_simplu:     bool
                | str
                | number
                ;
expr_stat:      factor_stat
                | expr_stat '+' expr_stat{
                        if($1==TIP_REAL || $3==TIP_REAL) $$=TIP_REAL;
                        else if($1==TIP_CAR) $$=TIP_CAR;
                                else $$=number;
                        push(pop()+pop());
                                                }
                | expr_stat '-' expr_stat {
                        if($1==TIP_REAL || $3==TIP_REAL) $$=TIP_REAL;
                        else if($1==TIP_CAR) $$=TIP_CAR;
                                else $$=number;
                        push(-pop()+pop());
                                                }
                | expr_stat '*' expr_stat{
                        if($1==TIP_REAL || $3==TIP_REAL) $$=TIP_REAL;
```

```
                else if($1==TIP_CAR) $$=TIP_CAR;
                        else $$=number;
                push(pop()*pop());
                                        }
        | expr_stat '/' expr_stat
        | expr_stat diff expr_stat
        | expr_stat multiple expr_stat
        | expr_stat sum expr_stat
        | expr_stat div expr_stat
        ;
factor_stat:    ID              {}
        | constanta
        | '(' expr_stat ')'{$$ = $2;}
        ;
constanta:      number{
                $$ = TIP_INT;
                push(atof($1));
                        }
        | str   {
                $$ = TIP_CAR;
                push((double)$1[0]);
                        }
        ;
instr_comp:     START lista_instr END
        ;
lista_instr:    instr
        | lista_instr ';' instr
        ;
instr:          /* empty */
```

```
                | instr_atrib
                | instr_if
                | instr_while
                | instr_comp
                | instr_read
                | instr_print
                ;
instr_atrib:    variabila '=' expresie
                ;
variabila:      ID
                | ID '[' expresie ']'
                | ID '.' ID
                ;
expresie:       factor
                | expresie '+' expresie
                | expresie '-' expresie
                | expresie '/' expresie
                | expresie '%' expresie
                | expresie multiple of expresie
                | expresie div expresie
                | expresie diff of expresie
                ;
factor:         ID
                | constanta {}
                | ID '(' lista_expr ')'
                | '(' expresie ')'
                | ID '[' expresie ']'
                | ID '.' ID
                ;
```

```
lista_expr:     expresie
                | lista_expr ',' expresie
                ;
instr_if: if conditie then instr ramura_else
                ;
ramura_else:    /* empty */
                else instr
                ;
conditie:       expr_logica
                | expresie op_rel expresie
                ;
expr_logica:    factor_logic
                | expr_logica and expr_logica
                | expr_logica or expr_logica
                ;
factor_logic:   '(' conditie ')'
                | not factor_logic
                ;
op_rel:         '='
                | '<'
                | '>'
                | greater
                | lower
                ;
instr_while:    while conditie do instr
                ;
instr_print:    show '(' lista_elem ')'
                ;
lista_elem:     element
```

```
                    | lista_elem ',' element
                    ;
element:            expresie
                    | list
                    ;
instr_read:         read '(' lista_variab ')'
                    ;
lista_variab:       variabila
                    | lista_variab ',' variabila
                    ;

%%

yyerror(char *s)
{
  printf("%s\n", s);
}

extern FILE *yyin;

main(int argc, char **argv)
{
  if(argc>1) yyin = fopen(argv[1], "r");
  if((argc>2)&&(!strcmp(argv[2],"-d"))) yydebug = 1;
  if(!yyparse()); fprintf(stderr,"\tO.K.\n");
}
```