```
%token<row> NOT
%token<row> PLUS
%token<row> MINUS
%token<row> MULT
%token<row> DIV
%token<row> MOD
%token<row> EQUAL
%token<row> NOT_EQUAL
%token<row> LT
%token<row> LE
%token<row> GT
%token<row> GE
%token<row> ASSIGNMENT
%token<row> AND
%token<row> OR
%token<row> L_BRACKET
%token<row> R_BRACKET
%token<row> L_SQUARE_BRACKET
%token<row> R_SQUARE_BRACKET
%token<row> L_PARANTHESIS
%token<row> R_PARANTHESIS
%token<row> COMMA
%token<row> APOSTROPHE
%token<row> COLON
%token<row> PERIOD
%token<row> IF
%token<row> ELSE
%token<row> WHILE
%token<row> FOR
%token<row> PRINT
%token<row> READ
%token<row> ARRAY
%token<row> TRUE
%token<row> FALSE
%token<row> IDENTIFIER
%token<row> INT
%token<row> BOOL
%token<row> STRING
%token<row> CHAR
%token<row> INT_CONSTANT
%token<row> BOOL_CONSTANT
%token<row> STRING_CONSTANT
%token<row> CHAR_CONSTANT
%token<row> STOP

%{
  #include "include.h"
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>

  static struct row_entry* cons(const char* name, struct row_entry*
first_child) {
    struct row_entry* answer = malloc(sizeof(struct row_entry));
    answer->name = strdup(name);
    answer->first_child = first_child;
    answer->next_sibling = NULL;
    return answer;
  }

  static void display(struct row_entry* node, int count_tabs) {
    if(node == NULL) return;
    for(int i = 0; i < count_tabs; i++) {
      printf("\t");
    }
    printf("%s\n", node->name);
    display(node->first_child, count_tabs + 1);
    display(node->next_sibling, count_tabs);
  }
  static void free_row_entry(struct row_entry* node) {
    if(node == NULL) return;
    free_row_entry(node->first_child);
    free_row_entry(node->next_sibling);
    free(node->name);
    free(node);
  }
  extern int yylex();
  extern int yylex_destroy();
%}

%union {
  struct row_entry* row;
}

%type<row> program
%type<row> statement
%type<row> type
%type<row> array_type
%type<row> decl_statement
%type<row> var_decl
%type<row> assignment_statement
%type<row> input
%type<row> output
%type<row> if_statement
%type<row> while_statement
%type<row> for_statement
%type<row> expression
%type<row> term
%type<row> factor
%type<row> stmtlist
%type<row> simple_statement
%type<row> struct_statement
%type<row> operator
%type<row> relation
%type<row> comp_condition
%type<row> condition
%type<row> simple_condition
%type<row> string_expression

%%
accept: program                              { display($1, 0);
free_row_entry($1); }

program: stmtlist STOP                        { $$ = cons("program", $1);
$1->next_sibling = $2; }
      ;
```

```
stmtlist: stmtlist statement                        { $$ =
cons("stmtlist", $1); $1->next_sibling = $2; }
       | statement                              { $$ = cons("stmtlist",
$1); }
       ;

statement: simple_statement  { $$ = cons("statement", $1); }
        | struct_statement   { $$ = cons("statement", $1); }
        ;

simple_statement:  decl_statement APOSTROPHE                { $$ =
cons("simple_statement", $1); $1->next_sibling = $2; }
         | assignment_statement APOSTROPHE        { $$ =
cons("simple_statement", $1); $1->next_sibling = $2; }
         | input APOSTROPHE                       { $$ =
cons("simple_statement", $1); $1->next_sibling = $2; }
         | output APOSTROPHE                      { $$ =
cons("simple_statement", $1); $1->next_sibling = $2; }
         ;

type: INT                                         { $$ = cons("type",
$1); }
    | STRING                                         { $$ = cons("type",
$1); }
    | BOOL                                        { $$ = cons("type",
$1); }
    | CHAR                                        { $$ = cons("type",
$1); }
    ;

array_type: type L_SQUARE_BRACKET INT_CONSTANT R_SQUARE_BRACKET { $$ =
cons("array_decl", $1); $1->next_sibling = $2; $2->next_sibling = $3; $3-
>next_sibling = $4; }
          ;

decl_statement: type IDENTIFIER                 { $$ =
cons("decl_statement", $1); $1->next_sibling = $2; }
              | array_type IDENTIFIER           { $$ =
cons("decl_statement", $1); $1->next_sibling = $2; }
              ;

var_decl: type IDENTIFIER { $$ = cons("var_decl", $1); $1->next_sibling =
$2; }
        ;

assignment_statement: IDENTIFIER ASSIGNMENT expression          { $$ =
cons("assignment_statement", $1); $1->next_sibling = $2; $2->next_sibling
= $3; }
                    | IDENTIFIER ASSIGNMENT comp_condition       { $$ =
cons("assignment_statement", $1); $1->next_sibling = $2; $2->next_sibling
= $3; }
                    | IDENTIFIER ASSIGNMENT string_expression   { $$ =
cons("assignment_statement", $1); $1->next_sibling = $2; $2->next_sibling
= $3; }
                    ;
```

```
expression: expression operator term    { $$ = cons("expression", $1);
$1->next_sibling = $2; $2->next_sibling = $3; }
          | term                          { $$ = cons("expression", $1);
}
          ;

term: term operator factor      { $$ = cons("term", $1); $1->next_sibling
= $2; $2->next_sibling = $3; }
    | factor                    { $$ = cons("term", $1); }
    ;

factor: L_PARANTHESIS expression R_PARANTHESIS { $$ = cons("factor", $1);
$1->next_sibling = $2; $2->next_sibling = $3; }
        | IDENTIFIER                           { $$ = cons("factor", $1);
}
        | INT_CONSTANT                             { $$ = cons("factor",
$1); }
        ;

string_expression: string_expression PLUS STRING_CONSTANT  { $$ =
cons("string_expression", $1); $1->next_sibling = $2; $2->next_sibling =
$3; }
                | STRING_CONSTANT                         { $$ =
cons("string_expression", $1); }
                ;

input: READ L_PARANTHESIS IDENTIFIER R_PARANTHESIS       { $$ =
cons("input", $1); $1->next_sibling = $2; $2->next_sibling = $3; $3-
>next_sibling = $4; }
     ;

output: PRINT L_PARANTHESIS IDENTIFIER R_PARANTHESIS   { $$ =
cons("output", $1); $1->next_sibling = $2; $2->next_sibling = $3; $3-
>next_sibling = $4; }
      | PRINT L_PARANTHESIS string_expression R_PARANTHESIS { $$ =
cons("output", $1); $1->next_sibling = $2; $2->next_sibling = $3; $3-
>next_sibling = $4; }
      ;

struct_statement: if_statement { $$ = cons("struct_statement", $1); }
        | while_statement { $$ = cons("struct_statement", $1); }
        | for_statement { $$ = cons("struct_statement", $1); }
        ;

if_statement: IF comp_condition COLON stmtlist COLON        { $$ =
cons("if_statement", $1); $1->next_sibling = $2; $2->next_sibling = $3;
$3->next_sibling = $4; $4->next_sibling = $5; }
            | IF comp_condition COLON stmtlist COLON ELSE COLON stmtlist
COLON  { $$ = cons("if_statement", $1); $1->next_sibling = $2; $2-
>next_sibling = $3; $3->next_sibling = $4; $4->next_sibling = $5; $5-
>next_sibling = $6; $6->next_sibling = $7; $7->next_sibling = $8; $8-
>next_sibling = $9; }
            ;

while_statement: WHILE comp_condition COLON stmtlist COLON  { $$ =
cons("while_statement", $1); $1->next_sibling = $2; $2->next_sibling =
$3; $3->next_sibling = $4; $4->next_sibling = $5; }
                ;
```

```
for_statement: FOR L_PARANTHESIS var_decl APOSTROPHE assignment_statement
APOSTROPHE assignment_statement R_PARANTHESIS COLON stmtlist COLON  { $$
= cons("for_statement", $1); $1->next_sibling = $2; $2->next_sibling =
$3; $3->next_sibling = $4; $4->next_sibling = $5; $5->next_sibling = $6;
$6->next_sibling = $7; $7->next_sibling = $8; $8->next_sibling = $9; $9-
>next_sibling = $10; $10->next_sibling = $11; }
             ;

comp_condition: L_PARANTHESIS condition AND comp_condition R_PARANTHESIS
{ $$ = cons("comp_condition", $1); $1->next_sibling = $2; $2-
>next_sibling = $3; $3->next_sibling = $4; $4->next_sibling = $5; }
            | L_PARANTHESIS condition OR comp_condition R_PARANTHESIS
{ $$ = cons("comp_condition", $1); $1->next_sibling = $2; $2-
>next_sibling = $3; $3->next_sibling = $4; $4->next_sibling = $5; }
            | condition     { $$ = cons("condition", $1); }
            ;

condition: NOT simple_condition     { $$ = cons("condition", $1); $1-
>next_sibling = $2;}
        | simple_condition          { $$ = cons("condition", $1); }
        ;

simple_condition: expression relation expression    { $$ =
cons("simple_condition", $1); $1->next_sibling = $2; $2->next_sibling =
$3; }
                ;

operator: PLUS              { $$ = cons("operator", $1); }
        | MINUS             { $$ = cons("operator", $1); }
        | MULT              { $$ = cons("operator", $1); }
        | DIV               { $$ = cons("operator", $1); }
        | MOD               { $$ = cons("operator", $1); }
        ;

relation: EQUAL             { $$ = cons("relation", $1); }
        | NOT_EQUAL         { $$ = cons("relation", $1); }
        | LT                { $$ = cons("relation", $1); }
        | LE                { $$ = cons("relation", $1); }
        | GT                { $$ = cons("relation", $1); }
        | GE                { $$ = cons("relation", $1); }
        ;
%%

int yyerror(char *s) {
  fprintf(stderr, "Error: %s\n", s);
  return 0;
}

int main() {
  yyparse();
  yylex_destroy();
  return 0;
}
```