```
%{
open Pp
%}

%token <int> INT
%token <string> IDENT
%token <string> STRING
%token <bool> BOOL
%token PLUS MINUS TIMES DIV
%token LESSGREATER LESSEQUAL GREATEREQUAL LESS GREATER EQUAL EQUALEQUAL
%token COLON COLONEQUAL COMMA SEMI SEMISEMI DOT
%token BEGIN END LPAREN RPAREN LBRACKET RBRACKET
%token IF THEN ELSE WHILE  DO
%token VAR FUNCTION PROCEDURE PROGRAM
%token WRITE WRITELN READ ALLOC
%token INTEGER BOOLEAN ARRAY OF TYPE

/* lowest precedence */
%left LESSGREATER GREATEREQUAL LESSEQUAL LESS GREATER EQUAL EQUALEQUAL
%left PLUS MINUS
%left TIMES DIV
%nonassoc UMINUS

/* highest precedence */

/* the entry point */
%start main
%type <Pp.program> main
%%
main:
  PROGRAM vars definitions body endtok
    { {global_vars = $2; definitions = $3; main = $4 } }
;

endtok:
  SEMISEMI {()}
| DOT      {()}
;

vars:
  VAR args2                      { $2 }
|                                { [] }
;

args2:
  declvar SEMI args2 { $1 @ $3 }
|                    { [] }
;

idents:
  IDENT COMMA idents {$1 :: $3 }
| IDENT              { [ $1 ] }
;

declvar:
  idents COLON type_expr           { List.map (fun x -> x, $3) $1 }
;
definitions:
  fonction SEMI definitions             { $1 :: $3 }
|                                       { [] }
;
fonction:
  FUNCTION IDENT LPAREN args RPAREN COLON type_expr SEMI vars body
    { $2, { arguments = $4; result = Some $7; local_vars = $9; body = $10 } }
| PROCEDURE IDENT LPAREN args RPAREN  SEMI vars body
    { $2, { arguments = $4; result = None; local_vars = $7; body = $8 } }
;
args:
  declvar args1                         { $1 @ $2 }
|                                       { [] }
;
```

```
args1:
  SEMI declvar args1                    { $2 @ $3 }
|                                       { [] }
;
instruction:
  IDENT COLONEQUAL expression           { Set ($1, $3) }
| BEGIN bloc END                        { Sequence $2 }
| IF expression THEN instruction ELSE instruction
                                        { If ($2, $4, $6) }
| WHILE expression DO instruction       { While ($2, $4) }
| IDENT LPAREN arguments RPAREN         { Procedure_call ($1, $3) }
| READ LPAREN IDENT RPAREN              { Read_int ($3) }
| WRITE LPAREN expression RPAREN        { Write_int ($3) }
| WRITELN LPAREN expression RPAREN      { Writeln_int ($3) }
| array_expression LBRACKET expression RBRACKET  COLONEQUAL expression
                                        { Seti ($1, $3, $6) }
;

body:
  BEGIN bloc END                        { $2 }
;

bloc:
  instruction bloc1                     { $1 :: $2 }
|                                       { [] }
;

bloc1:
  SEMI instruction bloc1                { $2 :: $3 }
|                                       { [] }
;

arguments:
  expression  arguments1                { $1 :: $2 }
|                                       { [] }
;

arguments1:
  COMMA expression arguments1           { $2 :: $3 }
|                                       { [] }
;


expression:
  LPAREN expression RPAREN              { $2 }
| INT                                   { Int $1 }
| MINUS expression %prec UMINUS         { Bin (Minus, Int 0, $2) }
| BOOL                                  { Bool $1 }
| IDENT                                 { Get $1 }
| call_expression                       { $1 }
| array_expression LBRACKET expression RBRACKET
                                        { Geti ($1, $3) }
| expression PLUS expression            { Bin (Plus, $1, $3) }
| expression MINUS expression           { Bin (Minus, $1, $3) }
| expression TIMES expression           { Bin (Times, $1, $3) }
| expression DIV expression             { Bin (Div, $1, $3) }
| expression LESS expression            { Bin (Lt, $1, $3) }
| expression LESSEQUAL expression       { Bin (Le, $1, $3) }
| expression GREATER expression         { Bin (Gt, $1, $3) }
| expression GREATEREQUAL expression    { Bin (Ge, $1, $3) }
| expression EQUAL expression           { Bin (Eq, $1, $3) }
| expression LESSGREATER expression     { Bin (Ne, $1, $3) }
;

array_expression:
  IDENT { Get $1 }
| array_expression LBRACKET expression RBRACKET { Geti ($1, $3) }
| call_expression { $1 }
| LPAREN expression RPAREN { $2 }

;
```

```
call_expression:
  | IDENT LPAREN arguments RPAREN       { Function_call ($1, $3) }
  | ALLOC LPAREN expression
          COLON type_expr RPAREN        { Alloc ($3,$5) }

type_expr:
    INTEGER                              { Integer }
  | BOOLEAN                              { Boolean }
  | ARRAY OF type_expr
                                         { Array ($3) }
;
```