```
%{ open Ast %}

%token SEMI LBRACK RBRACK LPAREN RPAREN LBRACE RBRACE BAR COMMA
%token PLUS PLUSEQ MINUS MINUSEQ TIMES TIMESEQ DIVIDE DIVIDEEQ PLUSTWO MINUSTWO
%token TILDE TRANSFER ASSIGN EQ NEQ LT LEQ GT GEQ AND OR CONCAT APPEND
%token GLOBALVAR ENTITYVAR
%token PRINT READ
%token RETURN IF ELSE FOR WHILE BREAK CONTINUE
%token BOOL INT STRING CARD CARDENTITY LIST VAR
%token CARDENTITIES GLOBALS INCLUDE PLAY START WINCONDITION
%token NULL GETTYPE
%token <bool> TRUE FALSE
%token <int> INTLITERAL
%token <string> STRINGLITERAL
%token <string> CARDLITERAL
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%left CONCAT
%left AND OR
%right ASSIGN PLUSEQ MINUSEQ TIMESEQ DIVIDEEQ APPEND
%right READ
%right PRINT
%left TRANSFER
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left PLUSTWO MINUSTWO
%left BOOL INT STRING CARD CARDENTITY LIST VAR
%left TILDE GETTYPE
%left GLOBALVAR ENTITYVAR

%start program
%type <Ast.program> program

%%

program:
  sdecl funcs_list { $1, List.rev $2 }

sdecl:
    INCLUDE LBRACE idecl_list RBRACE
    CARDENTITIES LBRACE cdecl_list RBRACE
    GLOBALS LBRACE vdecl_list RBRACE
    START LBRACE vdecl_list stmt_list RBRACE
    PLAY LBRACE vdecl_list stmt_list RBRACE
    WINCONDITION LBRACE vdecl_list stmt_list RBRACE
    { { incl = { includes = List.rev $3 };
        cent = { entities = List.rev $7 };
        glob = { globals = List.rev $11 };
        strt = { slocals = List.rev $15;
                 sbody = List.rev $16 };
        play = { plocals = List.rev $20;
                 pbody = List.rev $21 };
        wcon = { wlocals = List.rev $25;
                 wbody = List.rev $26 } } }

funcs_list:
    /* nothing */    { [] }
  | funcs_list fdecl { $2 :: $1 }

fdecl:
    ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
    { { fname = $1;
        formals = $3;
        locals = List.rev $6;
```

```
        body = List.rev $7 } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
    vdecl                   { [$1] }
  | formal_list COMMA vdecl { $3 :: $1 }

idecl_list:
    /* nothing */    { [] }
  | idecl_list idecl { $2 :: $1 }

idecl:
    STRINGLITERAL SEMI { $1 }

cdecl_list:
    /* nothing */    { [] }
  | cdecl_list cdecl { $2 :: $1 }

cdecl:
    ID SEMI { $1 }

vdecl_list:
    /* nothing */         { [] }
  | vdecl_list vdecl SEMI { $2 :: $1 }

vdecl:
    VAR ID  { $2 }

stmt_list:
    /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI { Expr($1) }
  | PRINT expr SEMI { Print($2) }
  | READ var SEMI { Read($2) }
  | BREAK SEMI { Break }
  | RETURN expr_opt SEMI { Return($2) }
  | IF LPAREN expr RPAREN LBRACE stmt_list RBRACE /* %prec NOELSE */
      { If($3, $6, []) }
  | IF LPAREN expr RPAREN LBRACE stmt_list RBRACE ELSE LBRACE stmt_list RBRACE
      { If($3, $6, $10) }
/*  | FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN */
/*      LBRACE stmt_list RBRACE */
/*      { For($3, $5, $7, $10) } */
  | WHILE LPAREN expr RPAREN LBRACE stmt_list RBRACE { While($3, $6) }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    NULL             { Null }
  | TRUE             { BoolLiteral(true) }
  | FALSE            { BoolLiteral(false) }
  | INTLITERAL       { IntLiteral($1) }
  | CARDLITERAL      { CardLiteral($1) }
  | STRINGLITERAL    { StringLiteral($1) }
  | var              { Variable($1) }
  | TILDE expr       { Rand($2) }
  | GETTYPE expr     { GetType($2) }
  | expr PLUS     expr { Binop($1, Add,   $3) }
  | expr MINUS    expr { Binop($1, Sub,   $3) }
  | expr TIMES    expr { Binop($1, Mult,  $3) }
  | expr DIVIDE   expr { Binop($1, Div,   $3) }
  | expr EQ       expr { Binop($1, Equal, $3) }
```

```
  | expr NEQ      expr { Binop($1, Neq,    $3) }
  | expr LT       expr { Binop($1, Less,   $3) }
  | expr LEQ      expr { Binop($1, Leq,    $3) }
  | expr GT       expr { Binop($1, Greater,  $3) }
  | expr GEQ      expr { Binop($1, Geq,    $3) }
  | expr AND      expr { Binop($1, And,    $3) }
  | expr OR       expr { Binop($1, Or,     $3) }
  | expr CONCAT   expr { Binop($1, Concat,$3) }
  | var PLUSEQ    expr { Assign($1, Binop(Variable($1), Add, $3)) }
  | var MINUSEQ   expr { Assign($1, Binop(Variable($1), Sub, $3)) }
  | var TIMESEQ   expr { Assign($1, Binop(Variable($1), Mult, $3)) }
  | var DIVIDEEQ  expr { Assign($1, Binop(Variable($1), Div, $3)) }
  | var PLUSTWO
      { Assign($1, Binop(Variable($1), Add, IntLiteral(1))) }
  | var MINUSTWO
      { Assign($1, Binop(Variable($1), Sub, IntLiteral(1))) }
  | var ASSIGN expr { Assign($1, $3) }
  | expr APPEND expr { Append($1, $3) }
  | BAR expr BAR { ListLength($2) }
  | var TRANSFER expr { Transfer($1, $3) }
  | LBRACK list_opt RBRACK { ListLiteral($2) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }

var:
    ID                                { VarExp($1, Local) }
  | GLOBALVAR ID                      { VarExp($2, Global) }
  | ENTITYVAR ID                      { VarExp($2, Entity) }
  | ID LBRACK expr RBRACK             { GetIndex($1, Local, $3) }
  | GLOBALVAR ID LBRACK expr RBRACK   { GetIndex($2, Global, $4) }
  | ENTITYVAR ID LBRACK expr RBRACK   { GetIndex($2, Entity, $4) }

list_opt:
    /* nothing */ { [] }
  | list_  { List.rev $1 }

list_:
    expr            { [$1] }
  | list_ COMMA expr { $3 :: $1 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```