```
(* Simple binary operators *)
type op =
  Add | Sub | Mult | Div | Equal | Neq | Less |
  Leq | Greater | Geq | And | Or | Concat

(* Simple indicator of scope for variables found in expressions *)
type scope =
    Global (* "Global" scope, which means that this var is a global variable *)
  | Local  (* "Local" scope, which means that this var is a local variable *)
  | Entity (* "Entity" scope, which means that this var is a CardEntity *)

(* Simple indicator of type for variable declarations *)
type t =
    Int (* "int" type *)
  | StringType (* "string" type *)
  | Bool (* "boolean" type *)
  | Card (* "Card" reference type *)
  | CardEntity (* "CardEntity" reference type *)
  | ListType (* "list" type. elements of a list can be expressions. *)

(* Variable used in an expression, contains the id and scope of the variable *)
(* Also can be a "GetIndex", which is a list dereference with string being *)
(* the list variable, and expr being the expression within the brackets. *)
type varexp =
    VarExp of string * scope (* Used by interpreter to store CardEntity refs *)
  | GetIndex of string * scope * expr

(* The expression type *)
and expr =
    Null (* The null type, comes from the "null" keyword *)
  | Variable of varexp
  | IntLiteral of int (* An "int" literal. Needs to be coerced to int in interpreter *
)
  | StringLiteral of string (* A "string" literal *)
  | BoolLiteral of bool (* A "bool" literal. Needs to be coerced to bool in interprete
r *)
  | CardLiteral of string (* A "card" reference literal, e.g. H2, DQ, S10 *)
  | ListLiteral of expr list (* The list literal, whose items can each be *)
                             (* expressions, so type checking needs to occur *)
                             (* in the interpreter. *)
  | Binop of expr * op * expr
  | Rand of expr (* The random operator, e.g. ~1, ~(a + b). The interpreter *)
                 (* needs to check that its expression evaluates to "int" *)
  | Assign of varexp * expr (* Assignment of an expression to a variable *)
  | Append of expr * expr (* Appending to a list variable *)
  | GetType of expr (* Returns the string description of the type of expr *)
  | ListLength of expr (* Should evaluate to the length of a list *)
  | Transfer of varexp * expr (* The transfer operator, e.g $player1 <- H1. *)
                              (* The interpreter needs to check that the lhs *)
                              (* evaluates to CardEntity and rhs evaluates *)
                              (* to Card *)
  | Call of string * expr list (* The function call where string is its id *)
                               (* and the list of expressions is the list of *)
                               (* actual arguments *)
  | Noexpr (* Could appear in the "for(;;)" or the "return" construction *)

type stmt =
    Break (* Should break out of the current for/while loop *)
  | Print of expr (* Prints out the contents of expr. The interpreter should *)
                  (* checks that the expr evaluates to a string *)
  | Read of varexp (* Reads in standard input into the variable *)
  | Expr of expr
  | Return of expr (* Returns from the current function. Interpreter should *)
                   (* check that expr evaluates to the return type of the *)
                   (* function. *)
  | If of expr * stmt list * stmt list (* If statement. the expr should *)
                                       (* evaluate to bool type. The first *)
                                       (* stmt list is executed when the *)
                                       (* expr is true, otherwise execute *)
```

```
                                               (* the second stmt list *)
(* | For of expr * expr * expr * stmt list Expressions are, in order, the *)
                                                  (* initialization, truth condition *)
                                                  (* and finally update step *)
  | While of expr * stmt list (* As long as expr is true, execute stmt list *)
                              (* indefinitely. *)
  | Nostmt

(* Standard function declaration *)
type func_decl = {
    fname : string;
    formals : string list;
    locals : string list;
    body : stmt list;
  }

(* Include declaration, behavior is undefined for now... *)
type incl_decl = {
    includes : string list;
  }

(* Card Entity declaration, contains a list of names for the entities *)
type cent_decl = {
    entities : string list;
  }

(* Global variable declaration, contains a list of variable declarations *)
type glob_decl = {
    globals : string list;
  }

(* Start declaration. Executed once at the beginning of the interpretation. *)
(* Should be able to break out with "return" *)
type strt_decl = {
    slocals : string list;
    sbody : stmt list;
  }

(* Play declaration. Executed indefinitely as long as WinCondition returns *)
(* null. Should be able to break out with "return" *)
type play_decl = {
    plocals : string list;
    pbody : stmt list;
  }

(* WinCondition declaration. Executed before each Play execution. Has a *)
(* return type of List (containing CardEntities) *)
type wcon_decl = {
    wlocals : string list;
    wbody : stmt list;
  }

(* Special declarations. Contains each of the above special declarations. *)
type spec_decl = {
    incl : incl_decl;
    cent : cent_decl;
    glob : glob_decl;
    strt : strt_decl;
    play : play_decl;
    wcon : wcon_decl;
  }

(* The program. Contains the special declarations and function declarations *)
type program = spec_decl * func_decl list
```