```
open Ast

let rec tabs i = match i with
    0 -> ""
  | x -> "\t" ^ tabs (x - 1)

let string_of_op op = match op with
    Add     -> "+"
  | Sub     -> "-"
  | Mult    -> "*"
  | Div     -> "/"
  | Equal   -> "=="
  | Neq     -> "!="
  | Less    -> "<"
  | Leq     -> "<="
  | Greater -> ">"
  | Geq     -> ">="
  | And     -> "&&"
  | Or      -> "||"
  | Concat  -> "^"

let rec string_of_t t = match t with
    Int        -> "int"
  | StringType -> "string"
  | Bool       -> "bool"
  | Card       -> "Card"
  | CardEntity -> "CardEntity"
  | ListType   -> "list"

let string_of_scope scope = match scope with
    Global  -> "#"
  | Local   -> ""
  | Entity  -> "$"

let string_of_vardec v = match v with
    id -> "var " ^ id

let rec string_of_varexp v = match v with
  | VarExp(id, s) -> string_of_scope s ^ id
  | GetIndex(id, s, e) -> string_of_scope s ^ id ^ "[" ^ string_of_expr e ^ "]"
and string_of_expr expr = match expr with
    Null -> "null"
  | Variable(v) -> string_of_varexp v
  | IntLiteral(i)    -> string_of_int i
  | StringLiteral(s) -> "\"" ^ s ^ "\""
  | BoolLiteral(b)   -> string_of_bool b
  | CardLiteral(c)   -> c
  | ListLiteral(el) ->
      "[" ^ String.concat ", " (List.map string_of_expr el) ^ "]"
  | Binop(e1, o, e2) ->
      "(" ^ string_of_expr e1 ^ " "
      ^ string_of_op o ^ " " ^ string_of_expr e2 ^ ")"
  | Rand(e) -> "~" ^ string_of_expr e
  | Assign(v, e) -> string_of_varexp v ^ " = " ^ string_of_expr e
  | ListLength(e) -> "|" ^ string_of_expr e ^ "|"
  | GetType(e) -> "@(" ^ string_of_expr e ^ ")"
  | Append(e1, e2) -> string_of_expr e1 ^ " :: " ^ string_of_expr e2
  | Transfer(v, e) -> string_of_varexp v ^ " <- " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""

let rec string_of_stmt t stmt = tabs t ^ match stmt with
    Break -> "break;\n"
  | Print(expr) -> "<< " ^ string_of_expr expr ^ ";\n"
  | Read(var) -> ">> " ^ string_of_varexp var ^ ";\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n"
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n"
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ") {\n" ^
```

```
      String.concat "" (List.map (string_of_stmt (t+1)) s1) ^
      tabs t ^ "} else {\n" ^
      String.concat "" (List.map (string_of_stmt (t+1)) s2) ^
      tabs t ^ "}\n"
(*| For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ "; " ^ string_of_expr e2 ^ "; " ^
      string_of_expr e3  ^ ") {\n" ^
      String.concat "" (List.map (string_of_stmt (t+1)) s) ^
      tabs t ^ "}\n" *)
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") {\n" ^
      String.concat "" (List.map (string_of_stmt (t+1)) s) ^
      tabs t ^ "}\n"
  | Nostmt -> ""

let string_of_strdecl id =
  "\t" ^ id ^ ";\n"

let string_of_vdecl v =
  "\t" ^ string_of_vardec v ^ ";\n"

let string_of_fdecl fdecl =
  fdecl.fname ^ "(" ^
  String.concat ", " (List.map string_of_vardec fdecl.formals) ^ ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map (string_of_stmt 1) fdecl.body) ^
  "}\n"

let string_of_sdecl_1 sname strings =
  sname ^ "\n{\n" ^
  String.concat "" (List.map string_of_strdecl strings) ^
  "}\n"

let string_of_sdecl_2 sname vars =
  sname ^ "\n{\n" ^
  String.concat "" (List.map string_of_vdecl vars) ^
  "}\n"

let string_of_sdecl_3 sname vars body =
  sname ^ "\n{\n" ^
  String.concat "" (List.map string_of_vdecl vars) ^
  String.concat "" (List.map (string_of_stmt 1) body) ^
  "}\n"

let string_of_program (spec, funcs) =
  string_of_sdecl_1 "Include" spec.incl.includes ^ "\n" ^
  string_of_sdecl_1 "CardEntities" spec.cent.entities ^ "\n" ^
  string_of_sdecl_2 "Globals" spec.glob.globals ^ "\n" ^
  string_of_sdecl_3 "Start" spec.strt.slocals spec.strt.sbody ^ "\n" ^
  string_of_sdecl_3 "PlayOrder" spec.play.plocals spec.play.pbody ^ "\n" ^
  string_of_sdecl_3 "WinningCondition" spec.wcon.wlocals spec.wcon.wbody
  ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)

let string_of_include_file (funcs) =
  String.concat "\n" (List.map string_of_fdecl funcs)
```