```
Require Import List.
Require Import String.
Require Import ZArith.

Open Scope list_scope.
Open Scope string_scope.
Open Scope Z_scope.

Require Import ImpSyntax.
Require Import ImpCommon.

Inductive eval_unop : op1 -> val -> val -> Prop :=
| eval_neg :
    forall i,
      eval_unop Oneg (Vint i)
                (Vint (Z.opp i))
| eval_not :
    forall b,
      eval_unop Onot (Vbool b)
                (Vbool (negb b)).

Inductive eval_binop : op2 -> val -> val -> val -> Prop :=
| eval_add_i :
    forall i1 i2,
      eval_binop Oadd (Vint i1) (Vint i2)
                 (Vint (Z.add i1 i2))
| eval_add_s :
    forall s1 s2,
      eval_binop Oadd (Vstr s1) (Vstr s2)
                 (Vstr (String.append s1 s2))
| eval_sub :
    forall i1 i2,
      eval_binop Osub (Vint i1) (Vint i2)
                 (Vint (Z.sub i1 i2))
| eval_mul :
    forall i1 i2,
      eval_binop Omul (Vint i1) (Vint i2)
                 (Vint (Z.mul i1 i2))
| eval_div :
    forall i1 i2,
      i2 <> 0 ->
      eval_binop Odiv (Vint i1) (Vint i2)
                 (Vint (Z.div i1 i2))
| eval_mod :
    forall i1 i2,
      i2 <> 0 ->
      eval_binop Omod (Vint i1) (Vint i2)
                 (Vint (Z.modulo i1 i2))
| eval_eq :
    forall v1 v2,
      eval_binop Oeq v1 v2
                 (Vbool (imp_eq v1 v2))
| eval_lt :
    forall i1 i2,
      eval_binop Olt (Vint i1) (Vint i2)
                 (Vbool (imp_lt i1 i2))
| eval_le :
    forall i1 i2,
      eval_binop Ole (Vint i1) (Vint i2)
                 (Vbool (imp_le i1 i2))
| eval_conj :
    forall b1 b2,
      eval_binop Oconj (Vbool b1) (Vbool b2)
                 (Vbool (andb b1 b2))
| eval_disj :
    forall b1 b2,
      eval_binop Odisj (Vbool b1) (Vbool b2)
                (Vbool (orb b1 b2)).
```

```
Print string.

Inductive eval_e (s : store) (h : heap) :
  expr -> val -> Prop :=
| eval_val :
    forall v,
      eval_e s h (Eval v) v
| eval_var :
    forall x v,
      lkup s x = Some v ->
      eval_e s h (Evar x) v
| eval_op1 :
    forall op e v v',
      eval_e s h e v ->
      eval_unop op v v' ->
      eval_e s h (Eop1 op e) v'
| eval_op2 :
    forall op e1 e2 v1 v2 v',
      eval_e s h e1 v1 ->
      eval_e s h e2 v2 ->
      eval_binop op v1 v2 v' ->
      eval_e s h (Eop2 op e1 e2) v'

(**
  TODO

  Please write the rules for Elen and Eidx.

  You may want to use helpers from ImpCommon.v
  and check out the definition of [string]:

<<
    Inductive string : Type :=
    | EmptyString : string
    | String : Ascii.ascii -> string -> string.
>>

  My solution has 4 rules.
*)
.

Inductive evals_e (s : store) (h : heap) :
  list expr -> list val -> Prop :=
| evals_nil :
    evals_e s h nil nil
| evals_cons :
    forall e es v vs,
      eval_e s h e v ->
      evals_e s h es vs ->
      evals_e s h (e :: es) (v :: vs).

Inductive eval_s :
  store -> heap -> stmt -> store -> heap -> Prop :=
| eval_nop :
    forall s h,
      eval_s
        s h Snop
        s h
| eval_set :
    forall s h x e v,
      eval_e s h e v ->
      eval_s
        s h (Sset x e)
        (update s x v) h

(**
  TODO

  Please write the rules for Salloc and Swrite.
  You may want to use helpers from ImpCommon.v.
*)


| eval_ifelse_t :
    forall s h e p1 p2 s' h',
      eval_e s h e (Vbool true) ->
      eval_s
        s h p1
        s' h' ->
      eval_s
        s h (Sifelse e p1 p2)
        s' h'
| eval_ifelse_f :
    forall s h e p1 p2 s' h',
      eval_e s h e (Vbool false) ->
      eval_s
        s h p2
        s' h' ->
      eval_s
        s h (Sifelse e p1 p2)
        s' h'
| eval_while_t :
    forall s1 h1 e p s2 h2 s3 h3,
      eval_e s1 h1 e (Vbool true) ->
      eval_s
        s1 h1 p
        s2 h2 ->
      eval_s
        s2 h2 (Swhile e p)
        s3 h3 ->
      eval_s
        s1 h1 (Swhile e p)
        s3 h3
| eval_while_f :
    forall s h e p,
      eval_e s h e (Vbool false) ->
      eval_s
        s h (Swhile e p)
        s h
| eval_seq :
    forall s1 h1 p1 s2 h2 p2 s3 h3,
      eval_s
        s1 h1 p1
        s2 h2 ->
      eval_s
        s2 h2 p2
        s3 h3 ->
      eval_s
        s1 h1 (Sseq p1 p2)
        s3 h3.
```