```
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.

Section TRANSF.

Variable transf : expr -> expr.

Fixpoint transf_s (s : stmt) : stmt :=
  match s with
  | Snop =>
      Snop
  | Sset x e =>
      Sset x (transf e)
  | Salloc x e1 e2 =>
      Salloc x (transf e1) (transf e2)
  | Swrite x e1 e2 =>
      Swrite x (transf e1) (transf e2)
  | Scall x f es =>
      Scall x f (List.map transf es)
  | Sifelse e p1 p2 =>
      Sifelse (transf e) (transf_s p1) (transf_s p2)
  | Swhile e p1 =>
      Swhile (transf e) (transf_s p1)
  | Sseq p1 p2 =>
      Sseq (transf_s p1) (transf_s p2)
  | Sincall p e x sr =>
      Sincall (transf_s p) (transf e) x sr
  end.

Definition transf_f (f: func) : func :=
  match f with
  | Func name params body ret =>
      Func name params (transf_s body) (transf ret)
  end.

Fixpoint transf_env (e : env) : env :=
  match e with
  | nil => nil
  | f :: fs => transf_f f :: transf_env fs
  end.

Definition transf_p (p : prog) : prog :=
  match p with
  | Prog funcs main ret =>
      Prog (transf_env funcs)
           (transf_s main)
           (transf ret)
  end.

End TRANSF.

(** common helper for recursive transf *)
Fixpoint transf_e (transf : expr -> expr)
  (e : expr) : expr :=
  match e with
  | Eval v =>
      transf (Eval v)
  | Evar x =>
      transf (Evar x)
  | Eop1 op e1 =>
      transf (Eop1 op (transf_e transf e1))
```



```
  | Eop2 op e1 e2 =>
      transf (Eop2 op (transf_e transf e1)
                      (transf_e transf e2))
  | Elen e1 =>
      transf (Elen (transf_e transf e1))
  | Eidx e1 e2 =>
      transf (Eidx (transf_e transf e1)
                   (transf_e transf e2))
  end.
```