```
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.
Require Import ImpEval.

Inductive cont : Type :=
| Kstop : cont
| Kseq : stmt -> cont -> cont
| Kcall : expr -> string -> store -> cont -> cont.

Inductive kstep (env : env) :
  store -> heap -> stmt -> cont ->
  store -> heap -> stmt -> cont -> Prop :=
| kstep_set :
    forall s h x e v k,
      eval_e s h e v ->
      kstep env
        s h (Sset x e) k
        (update s x v) h Snop k
| kstep_alloc :
    forall s h x e1 e2 i v k,
      eval_e s h e1 (Vint i) ->
      eval_e s h e2 v ->
      0 <= i ->
      kstep env
        s h (Salloc x e1 e2) k
        (update s x (Vaddr (zlen h))) (alloc h i v) Snop k
| kstep_write :
    forall s h x e1 e2 a i v l h' k,
      lkup s x = Some (Vaddr a) ->
      read h a = Some (Vint l) ->
      eval_e s h e1 (Vint i) ->
      eval_e s h e2 v ->
      0 <= i < l ->
      write h (Zsucc (a + i)) v = Some h' ->
      kstep env
        s h (Swrite x e1 e2) k
        s h' Snop k
| kstep_call_internal :
    forall s h x f es params body ret vs s' k,
      locate env f = Some (Func f params body ret) ->
      evals_e s h es vs ->
      updates store_0 params vs = Some s' ->
      kstep env
        s h (Scall x f es) k
        s' h body (Kcall ret x s k)
| kstep_call_external :
    forall s h x f es vs h' v' k,
      locate env f = None ->
      evals_e s h es vs ->
      extcall_spec f vs h v' h' ->
      kstep env
        s h (Scall x f es) k
        (update s x v') h' Snop k
| kstep_ifelse_t :
    forall s h e p1 p2 k,
      eval_e s h e (Vbool true) ->
      kstep env
        s h (Sifelse e p1 p2) k
        s h p1 k
| kstep_ifelse_f :
    forall s h e p1 p2 k,
```



```
      eval_e s h e (Vbool false) ->
      kstep env
        s h (Sifelse e p1 p2) k
        s h p2 k
| kstep_while_t :
    forall s h e p k,
      eval_e s h e (Vbool true) ->
      kstep env
        s h (Swhile e p) k
        s h p (Kseq (Swhile e p) k)
| kstep_while_f :
    forall s h e p k,
      eval_e s h e (Vbool false) ->
      kstep env
        s h (Swhile e p) k
        s h Snop k
| kstep_seq_nop :
    forall s h p k,
      kstep env
        s h Snop (Kseq p k)
        s h p k
| kstep_seq :
    forall s h p1 p2 k,
      kstep env
        s h (Sseq p1 p2) k
        s h p1 (Kseq p2 k)
| kstep_incall_ret :
    forall s h ret lr sr v k,
      eval_e s h ret v ->
      kstep env
        s h Snop (Kcall ret lr sr k)
        (update sr lr v) h Snop k
| kstep_incall :
    forall s h p ret lr sr k,
      kstep env
        s h (Sincall p ret lr sr) k
        s h p (Kcall ret lr sr k).

Inductive kstep_star (env : env) :
  store -> heap -> stmt -> cont ->
  store -> heap -> stmt -> cont -> Prop :=
| kstep_star_refl :
    forall s h p k,
      kstep_star env
        s h p k
        s h p k
| kstep_star_l :
    forall s1 h1 p1 k1 s2 h2 p2 k2 s3 h3 p3 k3,
      kstep env
        s1 h1 p1 k1
        s2 h2 p2 k2 ->
      kstep_star env
        s2 h2 p2 k2
        s3 h3 p3 k3 ->
      kstep_star env
        s1 h1 p1 k1
        s3 h3 p3 k3.

Inductive ksteps_p : prog -> val -> Prop :=
| steps_prog :
    forall funcs main ret s' h' v,
      kstep_star funcs
        store_0 heap_0 main Kstop
        s' h' Snop Kstop ->
      eval_e s' h' ret v ->
      ksteps_p (Prog funcs main ret) v.
```