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

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

Require Import StructTactics.
Require Import ImpSyntax.
Require Import ImpCommon.
Require Import ImpEval.
Require Import ImpInterp.
Require Import ImpInterpNock.
Require Import ImpInterpProof.
Require Import ImpInterpNockProof.
Require Import ImpVerif.
Require Import ImpSemanticsFacts.

Import ListNotations.

Definition sum_body :=
  (Sseq
  (Sset "i" (Eval (Vint 0)))
  (Sseq
  (Sset "result" (Eval (Vint 0)))
  (Swhile (Eop2 Olt (Evar "i") (Elen (Evar "a")))
    (Sseq
    (Sset "result" (Eop2 Oadd (Evar "result") (Eidx (Evar "a") (Evar "i"))))
    (Sset "i" (Eop2 Oadd (Evar "i") (Eval (Vint 1)))))))).

Definition sum_ret :=
  (Evar "result").

Definition sum_func :=
  (Func "sum" ("a" :: nil)
      sum_body
      sum_ret).

Definition main_body :=
(Sseq
(Scall "n" "read_int" (nil))
(Sseq
(Sifelse (Eop2 Olt (Evar "n") (Eval (Vint 0)))
  (Sset "n" (Eop1 Oneg (Evar "n")))
  Snop)
(Sseq
(Salloc "a" (Evar "n") (Eval (Vint 0)))
(Sseq
(Sset "i" (Eval (Vint 0)))
(Sseq
(Swhile (Eop2 Olt (Evar "i") (Evar "n"))
  (Sseq
  (Swrite "a" (Evar "i") (Eop2 Osub (Evar "n") (Evar "i")))
  (Sset "i" (Eop2 Oadd (Evar "i") (Eval (Vint 1))))))
(Scall "result" "sum" ((Evar "a") :: nil)))))))).

Definition answer := (Evar "result").

Definition main :=
(Prog [sum_func]
      main_body
      answer).

(** Carefully designed to match control flow of loop body above. *)
Fixpoint sum_Zlist (acc : Z) (l : list Z) : Z :=
  match l with
  | [] => acc
  | z :: l => sum_Zlist (acc + z) l
```



```
  end.

Lemma skipn_none :
  forall A (l : list A),
    skipn (List.length l) l = [].
Proof. induction l; simpl; auto using f_equal. Qed.

Lemma skipn_nth_error_unroll :
  forall A n (l : list A) a,
    nth_error l n = Some a ->
    skipn n l = a :: skipn (S n) l.
Proof.
  induction n; destruct l; simpl; intros; try congruence.
  erewrite IHn by eauto. destruct l; auto.
Qed.

Lemma sum_total_spec :
  forall env s h a_val contents,
    lkup s "a" = Some (Vaddr a_val) ->
    array_at a_val contents h ->
    after env s h sum_body (fun _ => False) (fun s' h' =>
      h' = h /\
      lkup s' "result" = Some (Vint (sum_Zlist 0 contents))).
Proof.
  intros.
  unfold sum_body.

  apply after_seq.
  eapply after_set; eauto.

  apply after_seq.
  eapply after_set; eauto.

  apply after_while
  with (I := fun s0 h0 =>
             h0 = h /\
             lkup s0 "a" = Some (Vaddr a_val) /\
             exists i_val acc,
               lkup s0 "i" = Some (Vint i_val) /\
               0 <= i_val <= Zlength contents /\
               lkup s0 "result" = Some (Vint acc) /\
               sum_Zlist acc (skipn (Z.to_nat i_val) contents) = sum_Zlist 0 con
tents)
       (f := fun s0 =>
             match lkup s0 "i" with
             | Some (Vint i) =>
               Z.to_nat (Zlength contents - i)
             | _ => 0%nat
             end).
  - (** I -> condition safety *)
    intuition.
    break_exists. break_and.
    subst.
    eauto 10 using array_at_read_length.
  - (** precondition *)
    intuition.
    exists 0, 0.
    rewrite !lkup_update_neq by discriminate.
    rewrite !lkup_update_same.
    intuition.
    rewrite Zlength_correct.
    zify. omega.
  - (** body obligation *)
    intuition. subst.
    break_exists.
    break_and.
    break_eval_expr.
    repeat find_rewrite.
    repeat find_injection.
```

```
    apply after_seq.
    eapply after_set.
    eauto 10 using array_at_read_length, array_at_read_nth with *.

    eapply after_set.
    eauto.

    rewrite !lkup_update_neq by discriminate.
    rewrite !lkup_update_same.
    rewrite !lkup_update_neq by discriminate.
    rewrite !lkup_update_same.
    intuition.
    + do 2 eexists.
      intuition eauto.
      omega.
      omega.
      erewrite skipn_nth_error_unroll with (a := nth (Z.to_nat i1) contents 0) i
n *|-.
      * cbn [sum_Zlist] in *.
        etransitivity; [|eauto].
        f_equal.
        f_equal.
        zify. rewrite !Z2Nat.id by omega. omega.
      * apply nth_nth_error.
        rewrite Zlength_correct in *.
        zify.
        rewrite Z2Nat.id in * by auto.
        omega.
    + rewrite Zlength_correct in *.
      zify. rewrite !Z2Nat.id in * by omega. omega.
- (** postcondition *)
  intuition.
  break_exists. break_and. subst.
  break_eval_expr.
  repeat find_rewrite.
  repeat find_injection.
  assert (i1 = Zlength contents) by omega.
  subst.
  rewrite Zlength_correct in *.
  rewrite Nat2Z.id in *.
  rewrite skipn_none in *.
  cbn [sum_Zlist] in *.
  congruence.
Qed.
```