Software Foundations: VC

August 4, 2026

Contents

Chapter 1

Library VC.sumarray

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "sumarray.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 17%positive.
Definition ___builtin_annot_intval : ident := 18%positive.
Definition ___builtin_bswap : ident := 2%positive.
Definition ___builtin_bswap16 : ident := 4%positive.
Definition ___builtin_bswap32 : ident := 3%positive.
Definition ___builtin_bswap64 : ident := 1%positive.
Definition ___builtin_cls : ident := 26%positive.
Definition ___builtin_clsl : ident := 27%positive.
Definition ___builtin_clsll : ident := 28%positive.
Definition ___builtin_clz : ident := 5%positive.
```

```
Definition ___builtin_clzl : ident := 6%positive.
Definition ___builtin_clzll : ident := 7%positive.
Definition ___builtin_ctz : ident := 8%positive.
Definition ___builtin_ctzl : ident := 9%positive.
Definition ___builtin_ctzll : ident := 10%positive.
Definition ___builtin_debug : ident := 35%positive.
Definition ___builtin_expect : ident := 25%positive.
Definition ___builtin_fabs : ident := 11%positive.
Definition ___builtin_fabsf : ident := 12%positive.
Definition ___builtin_fmadd : ident := 29%positive.
Definition ___builtin_fmax : ident := 33%positive.
Definition ___builtin_fmin : ident := 34%positive.
Definition ___builtin_fmsub : ident := 30%positive.
Definition ___builtin_fnmadd : ident := 31%positive.
Definition ___builtin_fnmsub : ident := 32%positive.
Definition ___builtin_fsqrt : ident := 13%positive.
Definition ___builtin_membar : ident := 19%positive.
Definition ___builtin_memcpy_aligned : ident := 15%positive.
Definition ___builtin_sel : ident := 16%positive.
Definition ___builtin_sqrt : ident := 14%positive.
Definition ___builtin_unreachable : ident := 24%positive.
Definition ___builtin_va_arg : ident := 21%positive.
Definition ___builtin_va_copy : ident := 22%positive.
Definition ___builtin_va_end : ident := 23%positive.
Definition ___builtin_va_start : ident := 20%positive.
Definition ___compcert_i64_dtos : ident := 47%positive.
Definition ___compcert_i64_dtou : ident := 48%positive.
Definition ___compcert_i64_sar : ident := 59%positive.
Definition ___compcert_i64_sdiv : ident := 53%positive.
Definition ___compcert_i64_shl : ident := 57%positive.
Definition ___compcert_i64_shr : ident := 58%positive.
Definition ___compcert_i64_smod : ident := 55%positive.
Definition ___compcert_i64_smulh : ident := 60%positive.
Definition ___compcert_i64_stod : ident := 49%positive.
Definition ___compcert_i64_stof : ident := 51%positive.
Definition ___compcert_i64_udiv : ident := 54%positive.
Definition ___compcert_i64_umod : ident := 56%positive.
Definition ___compcert_i64_umulh : ident := 61%positive.
Definition ___compcert_i64_utod : ident := 50%positive.
Definition ___compcert_i64_utof : ident := 52%positive.
Definition ___compcert_va_composite : ident := 46%positive.
Definition ___compcert_va_float64 : ident := 45%positive.
```

```
Definition ___compcert_va_int32 : ident := 43%positive.
Definition ___compcert_va_int64 : ident := 44%positive.
Definition _a : ident := 36%positive.
Definition _four : ident := 41%positive.
Definition _i : ident := 38%positive.
Definition _main : ident := 42%positive.
Definition _n : ident := 37%positive.
Definition _s : ident := 39%positive.
Definition _sumarray : ident := 40%positive.
Definition _t'1 : ident := 62%positive.

Definition f_sumarray := {|
  fn_return := tuint;
  fn_callconv := cc_default;
  fn_params := ((_a, (tptr tuint)) :: (_n, tint) :: nil);
  fn_vars := nil;
  fn_temps := ((_i, tint) :: (_s, tuint) :: (_t'1, tuint) :: nil);
  fn_body :=
(Ssequence
  (Sset _i (Econst_int (Int.repr 0) tint))
  (Ssequence
    (Sset _s (Econst_int (Int.repr 0) tint))
    (Ssequence
      (Swhile
        (Ebinop Olt (Etempvar _i tint) (Etempvar _n tint) tint)
        (Ssequence
          (Ssequence
            (Sset _t'1
              (Ederef
                (Ebinop Oadd (Etempvar _a (tptr tuint)) (Etempvar _i tint)
                  (tptr tuint)) tuint))
            (Sset _s
              (Ebinop Oadd (Etempvar _s tuint) (Etempvar _t'1 tuint) tuint)))
          (Sset _i
            (Ebinop Oadd (Etempvar _i tint) (Econst_int (Int.repr 1) tint)
              tint))))
      (Sreturn (Some (Etempvar _s tuint))))))
|}.

Definition v_four := {|
  gvar_info := (tarray tuint 4);
  gvar_init := (Init_int32 (Int.repr 1) :: Init_int32 (Int.repr 2) ::
                Init_int32 (Int.repr 3) :: Init_int32 (Int.repr 4) :: nil);
  gvar_readonly := false;
```

```
  gvar_volatile := false
|}.
Definition f_main := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := nil;
  fn_vars := nil;
  fn_temps := ((_s, tuint) :: (_t'1, tuint) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Ssequence
      (Scall (Some _t'1)
        (Evar _sumarray (Tfunction ((tptr tuint) :: tint :: nil) tuint
                                   cc_default))
        ((Evar _four (tarray tuint 4)) :: (Econst_int (Int.repr 4) tint) ::
         nil))
      (Sset _s (Etempvar _t'1 tuint))))
    (Sreturn (Some (Ecast (Etempvar _s tuint) tint))))
  (Sreturn (Some (Econst_int (Int.repr 0) tint))))
|}.
Definition composites : list composite_definition :=
nil.
Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
   Gfun(External (EF_runtime "__compcert_va_int32"
                             (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
     ((tptr tvoid) :: nil) tuint cc_default)) ::
 (___compcert_va_int64,
   Gfun(External (EF_runtime "__compcert_va_int64"
                             (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
     ((tptr tvoid) :: nil) tulong cc_default)) ::
 (___compcert_va_float64,
   Gfun(External (EF_runtime "__compcert_va_float64"
                             (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
     ((tptr tvoid) :: nil) tdouble cc_default)) ::
 (___compcert_va_composite,
   Gfun(External (EF_runtime "__compcert_va_composite"
                             (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                               cc_default)) ((tptr tvoid) :: tulong :: nil)
     (tptr tvoid) cc_default)) ::
 (___compcert_i64_dtos,
```

```
  Gfun(External (EF_runtime "__compcert_i64_dtos"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tlong cc_default)) ::
(___compcert_i64_dtou,
  Gfun(External (EF_runtime "__compcert_i64_dtou"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tulong cc_default)) ::
(___compcert_i64_stod,
  Gfun(External (EF_runtime "__compcert_i64_stod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tlong :: nil) tdouble cc_default)) ::
(___compcert_i64_utod,
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
(___compcert_i64_stof,
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
(___compcert_i64_utof,
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
```

```
  Gfun(External (EF_runtime "__compcert_i64_shl"
                  (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                    cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                  (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                    cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                  (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                    cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                  (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                    cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                  (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                    cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                  (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
                  (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
                  (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                  (mksignature (AST.Xint16unsigned :: nil)
                    AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                  (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
```

```
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctz,
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_ctzl,
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctzll,
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_fabs,
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_fabsf,
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
    (tfloat :: nil) tfloat cc_default)) ::
(___builtin_fsqrt,
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_sqrt,
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_memcpy_aligned,
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
```

```
(___builtin_sel,
  Gfun(External (EF_builtin "__builtin_sel"
                  (mksignature (AST.Xbool :: nil) AST.Xvoid
                    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
                  (mksignature (AST.Xptr :: nil) AST.Xvoid
                    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                  (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                    cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                  (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                  (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
  Gfun(External (EF_builtin "__builtin_va_arg"
                  (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                    cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                  (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                    cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                  (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                  (mksignature nil AST.Xvoid cc_default)) nil tvoid
```

```
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmadd,
  Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmsub,
  Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmax,
```

```
    Gfun(External (EF_builtin "__builtin_fmax"
                          (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                            cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
  (___builtin_fmin,
    Gfun(External (EF_builtin "__builtin_fmin"
                          (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                            cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
  (___builtin_debug,
    Gfun(External (EF_external "__builtin_debug"
                          (mksignature (AST.Xint :: nil) AST.Xvoid
                            {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
  (_sumarray, Gfun(Internal f_sumarray)) :: (_four, Gvar v_four) ::
  (_main, Gfun(Internal f_main)) :: nil).

Definition public_idents : list ident :=
(_main :: _four :: _sumarray :: ___builtin_debug :: ___builtin_fmin ::
  ___builtin_fmax :: ___builtin_fnmsub :: ___builtin_fnmadd ::
  ___builtin_fmsub :: ___builtin_fmadd :: ___builtin_clsll ::
  ___builtin_clsl :: ___builtin_cls :: ___builtin_expect ::
  ___builtin_unreachable :: ___builtin_va_end :: ___builtin_va_copy ::
  ___builtin_va_arg :: ___builtin_va_start :: ___builtin_membar ::
  ___builtin_annot_intval :: ___builtin_annot :: ___builtin_sel ::
  ___builtin_memcpy_aligned :: ___builtin_sqrt :: ___builtin_fsqrt ::
  ___builtin_fabsf :: ___builtin_fabs :: ___builtin_ctzll ::
  ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll :: ___builtin_clzl ::
  ___builtin_clz :: ___builtin_bswap16 :: ___builtin_bswap32 ::
  ___builtin_bswap :: ___builtin_bswap64 :: ___compcert_i64_umulh ::
  ___compcert_i64_smulh :: ___compcert_i64_sar :: ___compcert_i64_shr ::
  ___compcert_i64_shl :: ___compcert_i64_umod :: ___compcert_i64_smod ::
  ___compcert_i64_udiv :: ___compcert_i64_sdiv :: ___compcert_i64_utof ::
  ___compcert_i64_stof :: ___compcert_i64_utod :: ___compcert_i64_stod ::
  ___compcert_i64_dtou :: ___compcert_i64_dtos :: ___compcert_va_composite ::
  ___compcert_va_float64 :: ___compcert_va_int64 :: ___compcert_va_int32 ::
  nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 2

Library VC.reverse

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "reverse.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 20%positive.
Definition ___builtin_annot_intval : ident := 21%positive.
Definition ___builtin_bswap : ident := 5%positive.
Definition ___builtin_bswap16 : ident := 7%positive.
Definition ___builtin_bswap32 : ident := 6%positive.
Definition ___builtin_bswap64 : ident := 4%positive.
Definition ___builtin_cls : ident := 29%positive.
Definition ___builtin_clsl : ident := 30%positive.
Definition ___builtin_clsll : ident := 31%positive.
Definition ___builtin_clz : ident := 8%positive.
```

```
Definition ___builtin_clzl : ident := 9%positive.
Definition ___builtin_clzll : ident := 10%positive.
Definition ___builtin_ctz : ident := 11%positive.
Definition ___builtin_ctzl : ident := 12%positive.
Definition ___builtin_ctzll : ident := 13%positive.
Definition ___builtin_debug : ident := 38%positive.
Definition ___builtin_expect : ident := 28%positive.
Definition ___builtin_fabs : ident := 14%positive.
Definition ___builtin_fabsf : ident := 15%positive.
Definition ___builtin_fmadd : ident := 32%positive.
Definition ___builtin_fmax : ident := 36%positive.
Definition ___builtin_fmin : ident := 37%positive.
Definition ___builtin_fmsub : ident := 33%positive.
Definition ___builtin_fnmadd : ident := 34%positive.
Definition ___builtin_fnmsub : ident := 35%positive.
Definition ___builtin_fsqrt : ident := 16%positive.
Definition ___builtin_membar : ident := 22%positive.
Definition ___builtin_memcpy_aligned : ident := 18%positive.
Definition ___builtin_sel : ident := 19%positive.
Definition ___builtin_sqrt : ident := 17%positive.
Definition ___builtin_unreachable : ident := 27%positive.
Definition ___builtin_va_arg : ident := 24%positive.
Definition ___builtin_va_copy : ident := 25%positive.
Definition ___builtin_va_end : ident := 26%positive.
Definition ___builtin_va_start : ident := 23%positive.
Definition ___compcert_i64_dtos : ident := 54%positive.
Definition ___compcert_i64_dtou : ident := 55%positive.
Definition ___compcert_i64_sar : ident := 66%positive.
Definition ___compcert_i64_sdiv : ident := 60%positive.
Definition ___compcert_i64_shl : ident := 64%positive.
Definition ___compcert_i64_shr : ident := 65%positive.
Definition ___compcert_i64_smod : ident := 62%positive.
Definition ___compcert_i64_smulh : ident := 67%positive.
Definition ___compcert_i64_stod : ident := 56%positive.
Definition ___compcert_i64_stof : ident := 58%positive.
Definition ___compcert_i64_udiv : ident := 61%positive.
Definition ___compcert_i64_umod : ident := 63%positive.
Definition ___compcert_i64_umulh : ident := 68%positive.
Definition ___compcert_i64_utod : ident := 57%positive.
Definition ___compcert_i64_utof : ident := 59%positive.
Definition ___compcert_va_composite : ident := 53%positive.
Definition ___compcert_va_float64 : ident := 52%positive.
```

```
Definition ___compcert_va_int32 : ident := 50%positive.
Definition ___compcert_va_int64 : ident := 51%positive.
Definition _h : ident := 43%positive.
Definition _head : ident := 2%positive.
Definition _list : ident := 1%positive.
Definition _main : ident := 49%positive.
Definition _p : ident := 40%positive.
Definition _r : ident := 48%positive.
Definition _reverse : ident := 47%positive.
Definition _s : ident := 41%positive.
Definition _sumlist : ident := 44%positive.
Definition _t : ident := 42%positive.
Definition _tail : ident := 3%positive.
Definition _three : ident := 39%positive.
Definition _v : ident := 46%positive.
Definition _w : ident := 45%positive.
Definition _t'1 : ident := 69%positive.
Definition _t'2 : ident := 70%positive.

Definition v_three := {|
  gvar_info := (tarray (Tstruct _list noattr) 3);
  gvar_init := (Init_int32 (Int.repr 1) :: Init_space 4 ::
                Init_addrof _three (Ptrofs.repr 16) ::
                Init_int32 (Int.repr 2) :: Init_space 4 ::
                Init_addrof _three (Ptrofs.repr 32) ::
                Init_int32 (Int.repr 3) :: Init_space 4 ::
                Init_int64 (Int64.repr 0) :: nil);
  gvar_readonly := false;
  gvar_volatile := false
|}.

Definition f_sumlist := {|
  fn_return := tuint;
  fn_callconv := cc_default;
  fn_params := ((_p, (tptr (Tstruct _list noattr))) :: nil);
  fn_vars := nil;
  fn_temps := ((_s, tuint) :: (_t, (tptr (Tstruct _list noattr))) ::
               (_h, tuint) :: nil);
  fn_body :=
(Ssequence
  (Sset _s (Econst_int (Int.repr 0) tint))
  (Ssequence
    (Sset _t (Etempvar _p (tptr (Tstruct _list noattr))))
    (Ssequence
```

```
    (Swhile
      (Etempvar _t (tptr (Tstruct _list noattr)))
      (Ssequence
        (Sset _h
          (Efield
            (Ederef (Etempvar _t (tptr (Tstruct _list noattr)))
              (Tstruct _list noattr)) _head tuint))
        (Ssequence
          (Sset _t
            (Efield
              (Ederef (Etempvar _t (tptr (Tstruct _list noattr)))
                (Tstruct _list noattr)) _tail
              (tptr (Tstruct _list noattr))))
          (Sset _s
            (Ebinop Oadd (Etempvar _s tuint) (Etempvar _h tuint) tuint)))))
    (Sreturn (Some (Etempvar _s tuint))))))
|}.
Definition f_reverse := {|
  fn_return := (tptr (Tstruct _list noattr));
  fn_callconv := cc_default;
  fn_params := ((_p, (tptr (Tstruct _list noattr))) :: nil);
  fn_vars := nil;
  fn_temps := ((_w, (tptr (Tstruct _list noattr))) ::
               (_t, (tptr (Tstruct _list noattr))) ::
               (_v, (tptr (Tstruct _list noattr))) :: nil);
  fn_body :=
(Ssequence
  (Sset _w (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid)))
  (Ssequence
    (Sset _v (Etempvar _p (tptr (Tstruct _list noattr))))
    (Ssequence
      (Swhile
        (Etempvar _v (tptr (Tstruct _list noattr)))
        (Ssequence
          (Sset _t
            (Efield
              (Ederef (Etempvar _v (tptr (Tstruct _list noattr)))
                (Tstruct _list noattr)) _tail (tptr (Tstruct _list noattr))))
          (Ssequence
            (Sassign
              (Efield
                (Ederef (Etempvar _v (tptr (Tstruct _list noattr)))
```

```
                    (Tstruct _list noattr)) _tail
                  (tptr (Tstruct _list noattr)))
                (Etempvar _w (tptr (Tstruct _list noattr))))
              (Ssequence
                (Sset _w (Etempvar _v (tptr (Tstruct _list noattr))))
                (Sset _v (Etempvar _t (tptr (Tstruct _list noattr)))))))))
      (Sreturn (Some (Etempvar _w (tptr (Tstruct _list noattr))))))))
|}.
Definition f_main := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := nil;
  fn_vars := nil;
  fn_temps := ((_r, (tptr (Tstruct _list noattr))) :: (_s, tuint) ::
               (_t'2, tuint) :: (_t'1, (tptr (Tstruct _list noattr))) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Ssequence
      (Scall (Some _t'1)
        (Evar _reverse (Tfunction ((tptr (Tstruct _list noattr)) :: nil)
                                  (tptr (Tstruct _list noattr)) cc_default))
        ((Evar _three (tarray (Tstruct _list noattr) 3)) :: nil))
      (Sset _r (Etempvar _t'1 (tptr (Tstruct _list noattr)))))
    (Ssequence
      (Ssequence
        (Scall (Some _t'2)
          (Evar _sumlist (Tfunction ((tptr (Tstruct _list noattr)) :: nil)
                                    tuint cc_default))
          ((Etempvar _r (tptr (Tstruct _list noattr))) :: nil))
        (Sset _s (Etempvar _t'2 tuint)))
      (Sreturn (Some (Ecast (Etempvar _s tuint) tint)))))
  (Sreturn (Some (Econst_int (Int.repr 0) tint))))
|}.
Definition composites : list composite_definition :=
(Composite _list Struct
    (Member_plain _head tuint ::
     Member_plain _tail (tptr (Tstruct _list noattr)) :: nil)
    noattr :: nil).
Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
    Gfun(External (EF_runtime "__compcert_va_int32"
```

```
                    (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
    ((tptr tvoid) :: nil) tuint cc_default)) ::
(___compcert_va_int64,
  Gfun(External (EF_runtime "__compcert_va_int64"
                    (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
    ((tptr tvoid) :: nil) tulong cc_default)) ::
(___compcert_va_float64,
  Gfun(External (EF_runtime "__compcert_va_float64"
                    (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
    ((tptr tvoid) :: nil) tdouble cc_default)) ::
(___compcert_va_composite,
  Gfun(External (EF_runtime "__compcert_va_composite"
                    (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                      cc_default)) ((tptr tvoid) :: tulong :: nil)
    (tptr tvoid) cc_default)) ::
(___compcert_i64_dtos,
  Gfun(External (EF_runtime "__compcert_i64_dtos"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tlong cc_default)) ::
(___compcert_i64_dtou,
  Gfun(External (EF_runtime "__compcert_i64_dtou"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tulong cc_default)) ::
(___compcert_i64_stod,
  Gfun(External (EF_runtime "__compcert_i64_stod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tlong :: nil) tdouble cc_default)) ::
(___compcert_i64_utod,
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
(___compcert_i64_stof,
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
(___compcert_i64_utof,
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
```

```
                    cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                    (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
```

```
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
("___builtin_bswap32",
  Gfun(External (EF_builtin "__builtin_bswap32"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
("___builtin_bswap16",
  Gfun(External (EF_builtin "__builtin_bswap16"
                    (mksignature (AST.Xint16unsigned :: nil)
                      AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
("___builtin_clz",
  Gfun(External (EF_builtin "__builtin_clz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
("___builtin_clzl",
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
("___builtin_clzll",
  Gfun(External (EF_builtin "__builtin_clzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
("___builtin_ctz",
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
("___builtin_ctzl",
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
("___builtin_ctzll",
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
("___builtin_fabs",
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
("___builtin_fabsf",
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
```

```
    (tfloat :: nil) tfloat cc_default)) ::
(___builtin_fsqrt,
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_sqrt,
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_memcpy_aligned,
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
(___builtin_sel,
  Gfun(External (EF_builtin "__builtin_sel"
                    (mksignature (AST.Xbool :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                      cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
```

```
  Gfun(External (EF_builtin "__builtin_va_arg"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                    (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
```

```
                    (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                    AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
 (___builtin_fnmadd,
   Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
 (___builtin_fnmsub,
   Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
 (___builtin_fmax,
   Gfun(External (EF_builtin "__builtin_fmax"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
 (___builtin_fmin,
   Gfun(External (EF_builtin "__builtin_fmin"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
 (___builtin_debug,
   Gfun(External (EF_external "__builtin_debug"
                    (mksignature (AST.Xint :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
 (_three, Gvar v_three) :: (_sumlist, Gfun(Internal f_sumlist)) ::
 (_reverse, Gfun(Internal f_reverse)) :: (_main, Gfun(Internal f_main)) ::
 nil).

Definition public_idents : list ident :=
(_main :: _reverse :: _sumlist :: _three :: ___builtin_debug ::
 ___builtin_fmin :: ___builtin_fmax :: ___builtin_fnmsub ::
 ___builtin_fnmadd :: ___builtin_fmsub :: ___builtin_fmadd ::
 ___builtin_clsll :: ___builtin_clsl :: ___builtin_cls ::
 ___builtin_expect :: ___builtin_unreachable :: ___builtin_va_end ::
 ___builtin_va_copy :: ___builtin_va_arg :: ___builtin_va_start ::
 ___builtin_membar :: ___builtin_annot_intval :: ___builtin_annot ::
```

```
  ___builtin_sel :: ___builtin_memcpy_aligned :: ___builtin_sqrt ::
  ___builtin_fsqrt :: ___builtin_fabsf :: ___builtin_fabs ::
  ___builtin_ctzll :: ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll ::
  ___builtin_clzl :: ___builtin_clz :: ___builtin_bswap16 ::
  ___builtin_bswap32 :: ___builtin_bswap :: ___builtin_bswap64 ::
  ___compcert_i64_umulh :: ___compcert_i64_smulh :: ___compcert_i64_sar ::
  ___compcert_i64_shr :: ___compcert_i64_shl :: ___compcert_i64_umod ::
  ___compcert_i64_smod :: ___compcert_i64_udiv :: ___compcert_i64_sdiv ::
  ___compcert_i64_utof :: ___compcert_i64_stof :: ___compcert_i64_utod ::
  ___compcert_i64_stod :: ___compcert_i64_dtou :: ___compcert_i64_dtos ::
  ___compcert_va_composite :: ___compcert_va_float64 ::
  ___compcert_va_int64 :: ___compcert_va_int32 :: nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 3

Library VC.append

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "append.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 20%positive.
Definition ___builtin_annot_intval : ident := 21%positive.
Definition ___builtin_bswap : ident := 5%positive.
Definition ___builtin_bswap16 : ident := 7%positive.
Definition ___builtin_bswap32 : ident := 6%positive.
Definition ___builtin_bswap64 : ident := 4%positive.
Definition ___builtin_cls : ident := 29%positive.
Definition ___builtin_clsl : ident := 30%positive.
Definition ___builtin_clsll : ident := 31%positive.
Definition ___builtin_clz : ident := 8%positive.
```

```
Definition ___builtin_clzl : ident := 9%positive.
Definition ___builtin_clzll : ident := 10%positive.
Definition ___builtin_ctz : ident := 11%positive.
Definition ___builtin_ctzl : ident := 12%positive.
Definition ___builtin_ctzll : ident := 13%positive.
Definition ___builtin_debug : ident := 38%positive.
Definition ___builtin_expect : ident := 28%positive.
Definition ___builtin_fabs : ident := 14%positive.
Definition ___builtin_fabsf : ident := 15%positive.
Definition ___builtin_fmadd : ident := 32%positive.
Definition ___builtin_fmax : ident := 36%positive.
Definition ___builtin_fmin : ident := 37%positive.
Definition ___builtin_fmsub : ident := 33%positive.
Definition ___builtin_fnmadd : ident := 34%positive.
Definition ___builtin_fnmsub : ident := 35%positive.
Definition ___builtin_fsqrt : ident := 16%positive.
Definition ___builtin_membar : ident := 22%positive.
Definition ___builtin_memcpy_aligned : ident := 18%positive.
Definition ___builtin_sel : ident := 19%positive.
Definition ___builtin_sqrt : ident := 17%positive.
Definition ___builtin_unreachable : ident := 27%positive.
Definition ___builtin_va_arg : ident := 24%positive.
Definition ___builtin_va_copy : ident := 25%positive.
Definition ___builtin_va_end : ident := 26%positive.
Definition ___builtin_va_start : ident := 23%positive.
Definition ___compcert_i64_dtos : ident := 51%positive.
Definition ___compcert_i64_dtou : ident := 52%positive.
Definition ___compcert_i64_sar : ident := 63%positive.
Definition ___compcert_i64_sdiv : ident := 57%positive.
Definition ___compcert_i64_shl : ident := 61%positive.
Definition ___compcert_i64_shr : ident := 62%positive.
Definition ___compcert_i64_smod : ident := 59%positive.
Definition ___compcert_i64_smulh : ident := 64%positive.
Definition ___compcert_i64_stod : ident := 53%positive.
Definition ___compcert_i64_stof : ident := 55%positive.
Definition ___compcert_i64_udiv : ident := 58%positive.
Definition ___compcert_i64_umod : ident := 60%positive.
Definition ___compcert_i64_umulh : ident := 65%positive.
Definition ___compcert_i64_utod : ident := 54%positive.
Definition ___compcert_i64_utof : ident := 56%positive.
Definition ___compcert_va_composite : ident := 50%positive.
Definition ___compcert_va_float64 : ident := 49%positive.
```

```
Definition ___compcert_va_int32 : ident := 47%positive.
Definition ___compcert_va_int64 : ident := 48%positive.
Definition _append : ident := 43%positive.
Definition _append2 : ident := 46%positive.
Definition _curp : ident := 45%positive.
Definition _head : ident := 2%positive.
Definition _list : ident := 1%positive.
Definition _main : ident := 66%positive.
Definition _retp : ident := 44%positive.
Definition _t : ident := 41%positive.
Definition _tail : ident := 3%positive.
Definition _u : ident := 42%positive.
Definition _x : ident := 39%positive.
Definition _y : ident := 40%positive.
Definition _t'1 : ident := 67%positive.
Definition _t'2 : ident := 68%positive.
Definition _t'3 : ident := 69%positive.

Definition f_append := {|
  fn_return := (tptr (Tstruct _list noattr));
  fn_callconv := cc_default;
  fn_params := ((_x, (tptr (Tstruct _list noattr))) ::
                (_y, (tptr (Tstruct _list noattr))) :: nil);
  fn_vars := nil;
  fn_temps := ((_t, (tptr (Tstruct _list noattr))) ::
               (_u, (tptr (Tstruct _list noattr))) :: nil);
  fn_body :=
(Sifthenelse (Ebinop Oeq (Etempvar _x (tptr (Tstruct _list noattr)))
                (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid)) tint)
  (Sreturn (Some (Etempvar _y (tptr (Tstruct _list noattr)))))
  (Ssequence
    (Sset _t (Etempvar _x (tptr (Tstruct _list noattr))))
    (Ssequence
      (Sset _u
        (Efield
          (Ederef (Etempvar _t (tptr (Tstruct _list noattr)))
            (Tstruct _list noattr)) _tail (tptr (Tstruct _list noattr))))
      (Ssequence
        (Swhile
          (Ebinop One (Etempvar _u (tptr (Tstruct _list noattr)))
            (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid)) tint)
          (Ssequence
            (Sset _t (Etempvar _u (tptr (Tstruct _list noattr))))
```

```
            (Sset _u
              (Efield
                (Ederef (Etempvar _t (tptr (Tstruct _list noattr)))
                  (Tstruct _list noattr)) _tail
                (tptr (Tstruct _list noattr))))))
        (Ssequence
          (Sassign
            (Efield
              (Ederef (Etempvar _t (tptr (Tstruct _list noattr)))
                (Tstruct _list noattr)) _tail (tptr (Tstruct _list noattr)))
            (Etempvar _y (tptr (Tstruct _list noattr))))
          (Sreturn (Some (Etempvar _x (tptr (Tstruct _list noattr)))))))))
|}.
Definition f_append2 := {|
  fn_return := (tptr (Tstruct _list noattr));
  fn_callconv := cc_default;
  fn_params := ((_x, (tptr (Tstruct _list noattr))) ::
                (_y, (tptr (Tstruct _list noattr))) :: nil);
  fn_vars := ((_x, (tptr (Tstruct _list noattr))) :: nil);
  fn_temps := ((_retp, (tptr (tptr (Tstruct _list noattr)))) ::
               (_curp, (tptr (tptr (Tstruct _list noattr)))) ::
               (_t'3, (tptr (Tstruct _list noattr))) ::
               (_t'2, (tptr (Tstruct _list noattr))) ::
               (_t'1, (tptr (Tstruct _list noattr))) :: nil);
  fn_body :=
(Ssequence
  (Sassign (Evar _x (tptr (Tstruct _list noattr)))
    (Etempvar _x (tptr (Tstruct _list noattr))))
  (Ssequence
    (Sset _retp
      (Eaddrof (Evar _x (tptr (Tstruct _list noattr)))
        (tptr (tptr (Tstruct _list noattr)))))
    (Ssequence
      (Sset _curp
        (Eaddrof (Evar _x (tptr (Tstruct _list noattr)))
          (tptr (tptr (Tstruct _list noattr)))))
      (Ssequence
        (Sloop
          (Ssequence
            (Ssequence
              (Sset _t'3
                (Ederef (Etempvar _curp (tptr (tptr (Tstruct _list noattr))))
```

```
                    (tptr (Tstruct _list noattr))))
                (Sifthenelse (Ebinop One
                                (Etempvar _t'3 (tptr (Tstruct _list noattr)))
                                (Ecast (Econst_int (Int.repr 0) tint)
                                  (tptr tvoid)) tint)
                  Sskip
                  Sbreak))
              (Ssequence
                (Sset _t'2
                  (Ederef (Etempvar _curp (tptr (tptr (Tstruct _list noattr))))
                    (tptr (Tstruct _list noattr))))
                (Sset _curp
                  (Eaddrof
                    (Efield
                      (Ederef (Etempvar _t'2 (tptr (Tstruct _list noattr)))
                        (Tstruct _list noattr)) _tail
                      (tptr (Tstruct _list noattr)))
                    (tptr (tptr (Tstruct _list noattr)))))))
            Sskip)
          (Ssequence
            (Sassign
              (Ederef (Etempvar _curp (tptr (tptr (Tstruct _list noattr))))
                (tptr (Tstruct _list noattr)))
              (Etempvar _y (tptr (Tstruct _list noattr))))
            (Ssequence
              (Sset _t'1
                (Ederef (Etempvar _retp (tptr (tptr (Tstruct _list noattr))))
                  (tptr (Tstruct _list noattr))))
              (Sreturn (Some (Etempvar _t'1 (tptr (Tstruct _list noattr))))))))))
|}.

Definition composites : list composite_definition :=
(Composite _list Struct
    (Member_plain _head tint ::
     Member_plain _tail (tptr (Tstruct _list noattr)) :: nil)
    noattr :: nil).

Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
    Gfun(External (EF_runtime "__compcert_va_int32"
                    (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
      ((tptr tvoid) :: nil) tuint cc_default)) ::
 (___compcert_va_int64,
    Gfun(External (EF_runtime "__compcert_va_int64"
```

```
                    (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
    ((tptr tvoid) :: nil) tulong cc_default)) ::
(___compcert_va_float64,
  Gfun(External (EF_runtime "__compcert_va_float64"
                    (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
    ((tptr tvoid) :: nil) tdouble cc_default)) ::
(___compcert_va_composite,
  Gfun(External (EF_runtime "__compcert_va_composite"
                    (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                      cc_default)) ((tptr tvoid) :: tulong :: nil)
    (tptr tvoid) cc_default)) ::
(___compcert_i64_dtos,
  Gfun(External (EF_runtime "__compcert_i64_dtos"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tlong cc_default)) ::
(___compcert_i64_dtou,
  Gfun(External (EF_runtime "__compcert_i64_dtou"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tulong cc_default)) ::
(___compcert_i64_stod,
  Gfun(External (EF_runtime "__compcert_i64_stod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tlong :: nil) tdouble cc_default)) ::
(___compcert_i64_utod,
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
(___compcert_i64_stof,
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
(___compcert_i64_utof,
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
```

```
                    cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                    (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
```

```
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                    (mksignature (AST.Xint16unsigned :: nil)
                      AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctz,
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_ctzl,
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctzll,
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_fabs,
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_fabsf,
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
    (tfloat :: nil) tfloat cc_default)) ::
(___builtin_fsqrt,
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
```

```
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_sqrt,
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_memcpy_aligned,
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
(___builtin_sel,
  Gfun(External (EF_builtin "__builtin_sel"
                    (mksignature (AST.Xbool :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                      cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
  Gfun(External (EF_builtin "__builtin_va_arg"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
```

```
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                    (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmadd,
```

```
    Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
  (___builtin_fnmsub,
    Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
  (___builtin_fmax,
    Gfun(External (EF_builtin "__builtin_fmax"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
  (___builtin_fmin,
    Gfun(External (EF_builtin "__builtin_fmin"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
  (___builtin_debug,
    Gfun(External (EF_external "__builtin_debug"
                    (mksignature (AST.Xint :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
  (_append, Gfun(Internal f_append)) ::
  (_append2, Gfun(Internal f_append2)) :: nil).

Definition public_idents : list ident :=
(_append2 :: _append :: ___builtin_debug :: ___builtin_fmin ::
  ___builtin_fmax :: ___builtin_fnmsub :: ___builtin_fnmadd ::
  ___builtin_fmsub :: ___builtin_fmadd :: ___builtin_clsll ::
  ___builtin_clsl :: ___builtin_cls :: ___builtin_expect ::
  ___builtin_unreachable :: ___builtin_va_end :: ___builtin_va_copy ::
  ___builtin_va_arg :: ___builtin_va_start :: ___builtin_membar ::
  ___builtin_annot_intval :: ___builtin_annot :: ___builtin_sel ::
  ___builtin_memcpy_aligned :: ___builtin_sqrt :: ___builtin_fsqrt ::
  ___builtin_fabsf :: ___builtin_fabs :: ___builtin_ctzll ::
  ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll :: ___builtin_clzl ::
  ___builtin_clz :: ___builtin_bswap16 :: ___builtin_bswap32 ::
  ___builtin_bswap :: ___builtin_bswap64 :: ___compcert_i64_umulh ::
```

```
  ___compcert_i64_smulh :: ___compcert_i64_sar :: ___compcert_i64_shr ::
  ___compcert_i64_shl :: ___compcert_i64_umod :: ___compcert_i64_smod ::
  ___compcert_i64_udiv :: ___compcert_i64_sdiv :: ___compcert_i64_utof ::
  ___compcert_i64_stof :: ___compcert_i64_utod :: ___compcert_i64_stod ::
  ___compcert_i64_dtou :: ___compcert_i64_dtos :: ___compcert_va_composite ::
  ___compcert_va_float64 :: ___compcert_va_int64 :: ___compcert_va_int32 ::
  nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 4

Library VC.stack

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "stack.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 22%positive.
Definition ___builtin_annot_intval : ident := 23%positive.
Definition ___builtin_bswap : ident := 7%positive.
Definition ___builtin_bswap16 : ident := 9%positive.
Definition ___builtin_bswap32 : ident := 8%positive.
Definition ___builtin_bswap64 : ident := 6%positive.
Definition ___builtin_cls : ident := 31%positive.
Definition ___builtin_clsl : ident := 32%positive.
Definition ___builtin_clsll : ident := 33%positive.
Definition ___builtin_clz : ident := 10%positive.
```

```
Definition ___builtin_clzl : ident := 11%positive.
Definition ___builtin_clzll : ident := 12%positive.
Definition ___builtin_ctz : ident := 13%positive.
Definition ___builtin_ctzl : ident := 14%positive.
Definition ___builtin_ctzll : ident := 15%positive.
Definition ___builtin_debug : ident := 40%positive.
Definition ___builtin_expect : ident := 30%positive.
Definition ___builtin_fabs : ident := 16%positive.
Definition ___builtin_fabsf : ident := 17%positive.
Definition ___builtin_fmadd : ident := 34%positive.
Definition ___builtin_fmax : ident := 38%positive.
Definition ___builtin_fmin : ident := 39%positive.
Definition ___builtin_fmsub : ident := 35%positive.
Definition ___builtin_fnmadd : ident := 36%positive.
Definition ___builtin_fnmsub : ident := 37%positive.
Definition ___builtin_fsqrt : ident := 18%positive.
Definition ___builtin_membar : ident := 24%positive.
Definition ___builtin_memcpy_aligned : ident := 20%positive.
Definition ___builtin_sel : ident := 21%positive.
Definition ___builtin_sqrt : ident := 19%positive.
Definition ___builtin_unreachable : ident := 29%positive.
Definition ___builtin_va_arg : ident := 26%positive.
Definition ___builtin_va_copy : ident := 27%positive.
Definition ___builtin_va_end : ident := 28%positive.
Definition ___builtin_va_start : ident := 25%positive.
Definition ___compcert_i64_dtos : ident := 61%positive.
Definition ___compcert_i64_dtou : ident := 62%positive.
Definition ___compcert_i64_sar : ident := 73%positive.
Definition ___compcert_i64_sdiv : ident := 67%positive.
Definition ___compcert_i64_shl : ident := 71%positive.
Definition ___compcert_i64_shr : ident := 72%positive.
Definition ___compcert_i64_smod : ident := 69%positive.
Definition ___compcert_i64_smulh : ident := 74%positive.
Definition ___compcert_i64_stod : ident := 63%positive.
Definition ___compcert_i64_stof : ident := 65%positive.
Definition ___compcert_i64_udiv : ident := 68%positive.
Definition ___compcert_i64_umod : ident := 70%positive.
Definition ___compcert_i64_umulh : ident := 75%positive.
Definition ___compcert_i64_utod : ident := 64%positive.
Definition ___compcert_i64_utof : ident := 66%positive.
Definition ___compcert_va_composite : ident := 60%positive.
Definition ___compcert_va_float64 : ident := 59%positive.
```

```
Definition ___compcert_va_int32 : ident := 57%positive.
Definition ___compcert_va_int64 : ident := 58%positive.
Definition _cons : ident := 1%positive.
Definition _exit : ident := 43%positive.
Definition _free : ident := 42%positive.
Definition _i : ident := 46%positive.
Definition _main : ident := 56%positive.
Definition _malloc : ident := 41%positive.
Definition _n : ident := 51%positive.
Definition _newstack : ident := 45%positive.
Definition _next : ident := 3%positive.
Definition _p : ident := 44%positive.
Definition _pop : ident := 49%positive.
Definition _pop_and_add : ident := 55%positive.
Definition _push : ident := 48%positive.
Definition _push_increasing : ident := 52%positive.
Definition _q : ident := 47%positive.
Definition _s : ident := 54%positive.
Definition _st : ident := 50%positive.
Definition _stack : ident := 4%positive.
Definition _t : ident := 53%positive.
Definition _top : ident := 5%positive.
Definition _value : ident := 2%positive.
Definition _t'1 : ident := 76%positive.
Definition _t'2 : ident := 77%positive.

Definition f_newstack := {|
  fn_return := (tptr (Tstruct _stack noattr));
  fn_callconv := cc_default;
  fn_params := nil;
  fn_vars := nil;
  fn_temps := ((_p, (tptr (Tstruct _stack noattr))) ::
                (_t'1, (tptr tvoid)) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _malloc (Tfunction (tulong :: nil) (tptr tvoid) cc_default))
      ((Esizeof (Tstruct _stack noattr) tulong) :: nil))
    (Sset _p
      (Ecast (Etempvar _t'1 (tptr tvoid)) (tptr (Tstruct _stack noattr)))))
  (Ssequence
    (Sifthenelse (Eunop Onotbool (Etempvar _p (tptr (Tstruct _stack noattr)))
```

```
                    tint)
        (Scall None (Evar _exit (Tfunction (tint :: nil) tvoid cc_default))
          ((Econst_int (Int.repr 1) tint) :: nil))
        Sskip)
      (Ssequence
        (Sassign
          (Efield
            (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
              (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr)))
          (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid)))
        (Sreturn (Some (Etempvar _p (tptr (Tstruct _stack noattr)))))))))
|}.
Definition f_push := {|
  fn_return := tvoid;
  fn_callconv := cc_default;
  fn_params := ((_p, (tptr (Tstruct _stack noattr))) :: (_i, tint) :: nil);
  fn_vars := nil;
  fn_temps := ((_q, (tptr (Tstruct _cons noattr))) :: (_t'1, (tptr tvoid)) ::
                (_t'2, (tptr (Tstruct _cons noattr))) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _malloc (Tfunction (tulong :: nil) (tptr tvoid) cc_default))
      ((Esizeof (Tstruct _cons noattr) tulong) :: nil))
    (Sset _q
      (Ecast (Etempvar _t'1 (tptr tvoid)) (tptr (Tstruct _cons noattr)))))
  (Ssequence
    (Sifthenelse (Eunop Onotbool (Etempvar _q (tptr (Tstruct _cons noattr)))
                    tint)
      (Scall None (Evar _exit (Tfunction (tint :: nil) tvoid cc_default))
        ((Econst_int (Int.repr 1) tint) :: nil))
      Sskip)
    (Ssequence
      (Sassign
        (Efield
          (Ederef (Etempvar _q (tptr (Tstruct _cons noattr)))
            (Tstruct _cons noattr)) _value tint) (Etempvar _i tint))
      (Ssequence
        (Ssequence
          (Sset _t'2
            (Efield
```

```
                (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
                  (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr))))
            (Sassign
              (Efield
                (Ederef (Etempvar _q (tptr (Tstruct _cons noattr)))
                  (Tstruct _cons noattr)) _next (tptr (Tstruct _cons noattr)))
              (Etempvar _t'2 (tptr (Tstruct _cons noattr)))))
          (Sassign
            (Efield
              (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
                (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr)))
            (Etempvar _q (tptr (Tstruct _cons noattr))))))))
|}.
Definition f_pop := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := ((_p, (tptr (Tstruct _stack noattr))) :: nil);
  fn_vars := nil;
  fn_temps := ((_q, (tptr (Tstruct _cons noattr))) :: (_i, tint) ::
                (_t'1, (tptr (Tstruct _cons noattr))) :: nil);
  fn_body :=
(Ssequence
  (Sset _q
    (Efield
      (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
        (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr))))
  (Ssequence
    (Ssequence
      (Sset _t'1
        (Efield
          (Ederef (Etempvar _q (tptr (Tstruct _cons noattr)))
            (Tstruct _cons noattr)) _next (tptr (Tstruct _cons noattr))))
      (Sassign
        (Efield
          (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
            (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr)))
        (Etempvar _t'1 (tptr (Tstruct _cons noattr)))))
    (Ssequence
      (Sset _i
        (Efield
          (Ederef (Etempvar _q (tptr (Tstruct _cons noattr)))
            (Tstruct _cons noattr)) _value tint))
```

```
      (Ssequence
        (Scall None
          (Evar _free (Tfunction ((tptr tvoid) :: nil) tvoid cc_default))
          ((Etempvar _q (tptr (Tstruct _cons noattr))) :: nil))
        (Sreturn (Some (Etempvar _i tint)))))))
|}.
Definition f_push_increasing := {|
  fn_return := tvoid;
  fn_callconv := cc_default;
  fn_params := ((_st, (tptr (Tstruct _stack noattr))) :: (_n, tint) :: nil);
  fn_vars := nil;
  fn_temps := ((_i, tint) :: nil);
  fn_body :=
(Ssequence
  (Sset _i (Econst_int (Int.repr 0) tint))
  (Swhile
    (Ebinop Olt (Etempvar _i tint) (Etempvar _n tint) tint)
    (Ssequence
      (Sset _i
        (Ebinop Oadd (Etempvar _i tint) (Econst_int (Int.repr 1) tint) tint))
      (Scall None
        (Evar _push (Tfunction
                      ((tptr (Tstruct _stack noattr)) :: tint :: nil) tvoid
                      cc_default))
        ((Etempvar _st (tptr (Tstruct _stack noattr))) ::
         (Etempvar _i tint) :: nil))))))
|}.
Definition f_pop_and_add := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := ((_st, (tptr (Tstruct _stack noattr))) :: (_n, tint) :: nil);
  fn_vars := nil;
  fn_temps := ((_i, tint) :: (_t, tint) :: (_s, tint) :: (_t'1, tint) :: nil);
  fn_body :=
(Ssequence
  (Sset _i (Econst_int (Int.repr 0) tint))
  (Ssequence
    (Sset _s (Econst_int (Int.repr 0) tint))
    (Ssequence
      (Swhile
        (Ebinop Olt (Etempvar _i tint) (Etempvar _n tint) tint)
        (Ssequence
```

```
            (Ssequence
              (Scall (Some _t'1)
                (Evar _pop (Tfunction ((tptr (Tstruct _stack noattr)) :: nil)
                                      tint cc_default))
                ((Etempvar _st (tptr (Tstruct _stack noattr))) :: nil))
              (Sset _t (Etempvar _t'1 tint)))
            (Ssequence
              (Sset _s
                (Ebinop Oadd (Etempvar _s tint) (Etempvar _t tint) tint))
              (Sset _i
                (Ebinop Oadd (Etempvar _i tint) (Econst_int (Int.repr 1) tint)
                  tint)))))
        (Sreturn (Some (Etempvar _s tint))))))
|}.
Definition f_main := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := nil;
  fn_vars := nil;
  fn_temps := ((_st, (tptr (Tstruct _stack noattr))) :: (_i, tint) ::
               (_t, tint) :: (_s, tint) :: (_t'2, tint) ::
               (_t'1, (tptr (Tstruct _stack noattr))) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Ssequence
      (Scall (Some _t'1)
        (Evar _newstack (Tfunction nil (tptr (Tstruct _stack noattr))
                                   cc_default)) nil)
      (Sset _st (Etempvar _t'1 (tptr (Tstruct _stack noattr)))))
    (Ssequence
      (Scall None
        (Evar _push_increasing (Tfunction
                                  ((tptr (Tstruct _stack noattr)) :: tint ::
                                   nil) tvoid cc_default))
        ((Etempvar _st (tptr (Tstruct _stack noattr))) ::
         (Econst_int (Int.repr 10) tint) :: nil))
      (Ssequence
        (Ssequence
          (Scall (Some _t'2)
            (Evar _pop_and_add (Tfunction
                                  ((tptr (Tstruct _stack noattr)) :: tint ::
```

```
                                        nil) tint cc_default))
                ((Etempvar _st (tptr (Tstruct _stack noattr))) ::
                 (Econst_int (Int.repr 10) tint) :: nil))
              (Sset _s (Etempvar _t'2 tint)))
            (Sreturn (Some (Etempvar _s tint))))))
   (Sreturn (Some (Econst_int (Int.repr 0) tint))))
|}.

Definition composites : list composite_definition :=
(Composite _cons Struct
    (Member_plain _value tint ::
     Member_plain _next (tptr (Tstruct _cons noattr)) :: nil)
    noattr ::
 Composite _stack Struct
    (Member_plain _top (tptr (Tstruct _cons noattr)) :: nil)
    noattr :: nil).

Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
    Gfun(External (EF_runtime "__compcert_va_int32"
                    (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
      ((tptr tvoid) :: nil) tuint cc_default)) ::
 (___compcert_va_int64,
    Gfun(External (EF_runtime "__compcert_va_int64"
                    (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
      ((tptr tvoid) :: nil) tulong cc_default)) ::
 (___compcert_va_float64,
    Gfun(External (EF_runtime "__compcert_va_float64"
                    (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
      ((tptr tvoid) :: nil) tdouble cc_default)) ::
 (___compcert_va_composite,
    Gfun(External (EF_runtime "__compcert_va_composite"
                    (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                      cc_default)) ((tptr tvoid) :: tulong :: nil)
      (tptr tvoid) cc_default)) ::
 (___compcert_i64_dtos,
    Gfun(External (EF_runtime "__compcert_i64_dtos"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
      (tdouble :: nil) tlong cc_default)) ::
 (___compcert_i64_dtou,
    Gfun(External (EF_runtime "__compcert_i64_dtou"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
      (tdouble :: nil) tulong cc_default)) ::
 (___compcert_i64_stod,
```

```
  Gfun(External (EF_runtime "__compcert_i64_stod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tlong :: nil) tdouble cc_default)) ::
(___compcert_i64_utod,
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
(___compcert_i64_stof,
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
(___compcert_i64_utof,
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
```

```
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                    (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                    (mksignature (AST.Xint16unsigned :: nil)
                      AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctz,
```

```
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_ctzl,
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctzll,
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_fabs,
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_fabsf,
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
    (tfloat :: nil) tfloat cc_default)) ::
(___builtin_fsqrt,
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_sqrt,
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_memcpy_aligned,
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
(___builtin_sel,
  Gfun(External (EF_builtin "__builtin_sel"
                    (mksignature (AST.Xbool :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
```

```
                    (mksignature (AST.Xptr :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                      cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
  Gfun(External (EF_builtin "__builtin_va_arg"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                    (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
```

```
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmadd,
  Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmsub,
  Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmax,
  Gfun(External (EF_builtin "__builtin_fmax"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
    cc_default)) ::
(___builtin_fmin,
  Gfun(External (EF_builtin "__builtin_fmin"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
```

```
      cc_default)) ::
  (___builtin_debug,
    Gfun(External (EF_external "__builtin_debug"
                        (mksignature (AST.Xint :: nil) AST.Xvoid
                          {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
  (_malloc, Gfun(External EF_malloc (tulong :: nil) (tptr tvoid) cc_default)) ::
  (_free, Gfun(External EF_free ((tptr tvoid) :: nil) tvoid cc_default)) ::
  (_exit,
    Gfun(External (EF_external "exit"
                        (mksignature (AST.Xint :: nil) AST.Xvoid cc_default))
      (tint :: nil) tvoid cc_default)) ::
  (_newstack, Gfun(Internal f_newstack)) :: (_push, Gfun(Internal f_push)) ::
  (_pop, Gfun(Internal f_pop)) ::
  (_push_increasing, Gfun(Internal f_push_increasing)) ::
  (_pop_and_add, Gfun(Internal f_pop_and_add)) ::
  (_main, Gfun(Internal f_main)) :: nil).

Definition public_idents : list ident :=
(_main :: _pop_and_add :: _push_increasing :: _pop :: _push :: _newstack ::
  _exit :: _free :: _malloc :: ___builtin_debug :: ___builtin_fmin ::
  ___builtin_fmax :: ___builtin_fnmsub :: ___builtin_fnmadd ::
  ___builtin_fmsub :: ___builtin_fmadd :: ___builtin_clsll ::
  ___builtin_clsl :: ___builtin_cls :: ___builtin_expect ::
  ___builtin_unreachable :: ___builtin_va_end :: ___builtin_va_copy ::
  ___builtin_va_arg :: ___builtin_va_start :: ___builtin_membar ::
  ___builtin_annot_intval :: ___builtin_annot :: ___builtin_sel ::
  ___builtin_memcpy_aligned :: ___builtin_sqrt :: ___builtin_fsqrt ::
  ___builtin_fabsf :: ___builtin_fabs :: ___builtin_ctzll ::
  ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll :: ___builtin_clzl ::
  ___builtin_clz :: ___builtin_bswap16 :: ___builtin_bswap32 ::
  ___builtin_bswap :: ___builtin_bswap64 :: ___compcert_i64_umulh ::
  ___compcert_i64_smulh :: ___compcert_i64_sar :: ___compcert_i64_shr ::
  ___compcert_i64_shl :: ___compcert_i64_umod :: ___compcert_i64_smod ::
  ___compcert_i64_udiv :: ___compcert_i64_sdiv :: ___compcert_i64_utof ::
  ___compcert_i64_stof :: ___compcert_i64_utod :: ___compcert_i64_stod ::
  ___compcert_i64_dtou :: ___compcert_i64_dtos :: ___compcert_va_composite ::
  ___compcert_va_float64 :: ___compcert_va_int64 :: ___compcert_va_int32 ::
  nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 5

Library VC.strlib

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "strlib.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 17%positive.
Definition ___builtin_annot_intval : ident := 18%positive.
Definition ___builtin_bswap : ident := 2%positive.
Definition ___builtin_bswap16 : ident := 4%positive.
Definition ___builtin_bswap32 : ident := 3%positive.
Definition ___builtin_bswap64 : ident := 1%positive.
Definition ___builtin_cls : ident := 26%positive.
Definition ___builtin_clsl : ident := 27%positive.
Definition ___builtin_clsll : ident := 28%positive.
Definition ___builtin_clz : ident := 5%positive.
```

```
Definition ___builtin_clzl : ident := 6%positive.
Definition ___builtin_clzll : ident := 7%positive.
Definition ___builtin_ctz : ident := 8%positive.
Definition ___builtin_ctzl : ident := 9%positive.
Definition ___builtin_ctzll : ident := 10%positive.
Definition ___builtin_debug : ident := 35%positive.
Definition ___builtin_expect : ident := 25%positive.
Definition ___builtin_fabs : ident := 11%positive.
Definition ___builtin_fabsf : ident := 12%positive.
Definition ___builtin_fmadd : ident := 29%positive.
Definition ___builtin_fmax : ident := 33%positive.
Definition ___builtin_fmin : ident := 34%positive.
Definition ___builtin_fmsub : ident := 30%positive.
Definition ___builtin_fnmadd : ident := 31%positive.
Definition ___builtin_fnmsub : ident := 32%positive.
Definition ___builtin_fsqrt : ident := 13%positive.
Definition ___builtin_membar : ident := 19%positive.
Definition ___builtin_memcpy_aligned : ident := 15%positive.
Definition ___builtin_sel : ident := 16%positive.
Definition ___builtin_sqrt : ident := 14%positive.
Definition ___builtin_unreachable : ident := 24%positive.
Definition ___builtin_va_arg : ident := 21%positive.
Definition ___builtin_va_copy : ident := 22%positive.
Definition ___builtin_va_end : ident := 23%positive.
Definition ___builtin_va_start : ident := 20%positive.
Definition ___compcert_i64_dtos : ident := 59%positive.
Definition ___compcert_i64_dtou : ident := 60%positive.
Definition ___compcert_i64_sar : ident := 71%positive.
Definition ___compcert_i64_sdiv : ident := 65%positive.
Definition ___compcert_i64_shl : ident := 69%positive.
Definition ___compcert_i64_shr : ident := 70%positive.
Definition ___compcert_i64_smod : ident := 67%positive.
Definition ___compcert_i64_smulh : ident := 72%positive.
Definition ___compcert_i64_stod : ident := 61%positive.
Definition ___compcert_i64_stof : ident := 63%positive.
Definition ___compcert_i64_udiv : ident := 66%positive.
Definition ___compcert_i64_umod : ident := 68%positive.
Definition ___compcert_i64_umulh : ident := 73%positive.
Definition ___compcert_i64_utod : ident := 62%positive.
Definition ___compcert_i64_utof : ident := 64%positive.
Definition ___compcert_va_composite : ident := 58%positive.
Definition ___compcert_va_float64 : ident := 57%positive.
```

```
Definition ___compcert_va_int32 : ident := 55%positive.
Definition ___compcert_va_int64 : ident := 56%positive.
Definition ___stringlit_1 : ident := 53%positive.
Definition _buf : ident := 52%positive.
Definition _c : ident := 39%positive.
Definition _d : ident := 40%positive.
Definition _d1 : ident := 49%positive.
Definition _d2 : ident := 50%positive.
Definition _dest : ident := 42%positive.
Definition _example_call_strcpy : ident := 54%positive.
Definition _i : ident := 37%positive.
Definition _j : ident := 45%positive.
Definition _main : ident := 74%positive.
Definition _src : ident := 43%positive.
Definition _str : ident := 36%positive.
Definition _str1 : ident := 47%positive.
Definition _str2 : ident := 48%positive.
Definition _strcat : ident := 46%positive.
Definition _strchr : ident := 41%positive.
Definition _strcmp : ident := 51%positive.
Definition _strcpy : ident := 44%positive.
Definition _strlen : ident := 38%positive.
Definition _t'1 : ident := 75%positive.
Definition _t'2 : ident := 76%positive.
Definition _t'3 : ident := 77%positive.

Definition v___stringlit_1 := {|
  gvar_info := (tarray tschar 6);
  gvar_init := (Init_int8 (Int.repr 72) :: Init_int8 (Int.repr 101) ::
                Init_int8 (Int.repr 108) :: Init_int8 (Int.repr 108) ::
                Init_int8 (Int.repr 111) :: Init_int8 (Int.repr 0) :: nil);
  gvar_readonly := true;
  gvar_volatile := false
|}.

Definition f_strlen := {|
  fn_return := tulong;
  fn_callconv := cc_default;
  fn_params := ((_str, (tptr tschar)) :: nil);
  fn_vars := nil;
  fn_temps := ((_i, tulong) :: (_t'1, tschar) :: nil);
  fn_body :=
(Ssequence
  (Sset _i (Ecast (Econst_int (Int.repr 0) tint) tulong))
```

```
  (Sloop
    (Ssequence
      Sskip
      (Ssequence
        (Sset _t'1
          (Ederef
            (Ebinop Oadd (Etempvar _str (tptr tschar)) (Etempvar _i tulong)
              (tptr tschar)) tschar))
        (Sifthenelse (Ebinop Oeq (Etempvar _t'1 tschar)
                               (Econst_int (Int.repr 0) tint) tint)
          (Sreturn (Some (Etempvar _i tulong)))
          Sskip)))
    (Sset _i
      (Ebinop Oadd (Etempvar _i tulong) (Econst_int (Int.repr 1) tint)
        tulong))))
|}.
Definition f_strchr := {|
  fn_return := (tptr tschar);
  fn_callconv := cc_default;
  fn_params := ((_str, (tptr tschar)) :: (_c, tint) :: nil);
  fn_vars := nil;
  fn_temps := ((_i, tulong) :: (_d, tschar) :: (_t'1, tschar) :: nil);
  fn_body :=
(Ssequence
  (Sset _i (Ecast (Econst_int (Int.repr 0) tint) tulong))
  (Sloop
    (Ssequence
      Sskip
      (Ssequence
        (Ssequence
          (Sset _t'1
            (Ederef
              (Ebinop Oadd (Etempvar _str (tptr tschar)) (Etempvar _i tulong)
                (tptr tschar)) tschar))
          (Sset _d (Ecast (Etempvar _t'1 tschar) tschar)))
        (Ssequence
          (Sifthenelse (Ebinop Oeq (Etempvar _d tschar) (Etempvar _c tint)
                                 tint)
            (Sreturn (Some (Ebinop Oadd (Etempvar _str (tptr tschar))
                                       (Etempvar _i tulong) (tptr tschar))))
            Sskip)
          (Sifthenelse (Ebinop Oeq (Etempvar _d tschar)
```

```
                                (Econst_int (Int.repr 0) tint) tint)
                    (Sreturn (Some (Econst_int (Int.repr 0) tint)))
                    Sskip))))
      (Sset _i
        (Ebinop Oadd (Etempvar _i tulong) (Econst_int (Int.repr 1) tint)
          tulong))))
|}.

Definition f_strcpy := {|
  fn_return := (tptr tschar);
  fn_callconv := cc_default;
  fn_params := ((_dest, (tptr tschar)) :: (_src, (tptr tschar)) :: nil);
  fn_vars := nil;
  fn_temps := ((_i, tulong) :: (_d, tschar) :: (_t'1, tschar) :: nil);
  fn_body :=
(Ssequence
  (Sset _i (Ecast (Econst_int (Int.repr 0) tint) tulong))
  (Sloop
    (Ssequence
      Sskip
      (Ssequence
        (Ssequence
          (Sset _t'1
            (Ederef
              (Ebinop Oadd (Etempvar _src (tptr tschar)) (Etempvar _i tulong)
                (tptr tschar)) tschar))
          (Sset _d (Ecast (Etempvar _t'1 tschar) tschar)))
        (Ssequence
          (Sassign
            (Ederef
              (Ebinop Oadd (Etempvar _dest (tptr tschar))
                (Etempvar _i tulong) (tptr tschar)) tschar)
            (Etempvar _d tschar))
          (Sifthenelse (Ebinop Oeq (Etempvar _d tschar)
                                (Econst_int (Int.repr 0) tint) tint)
            (Sreturn (Some (Etempvar _dest (tptr tschar))))
            Sskip))))
      (Sset _i
        (Ebinop Oadd (Etempvar _i tulong) (Econst_int (Int.repr 1) tint)
          tulong))))
|}.

Definition f_strcat := {|
  fn_return := (tptr tschar);
```

```
  fn_callconv := cc_default;
  fn_params := ((_dest, (tptr tschar)) :: (_src, (tptr tschar)) :: nil);
  fn_vars := nil;
  fn_temps := ((_i, tulong) :: (_j, tulong) :: (_d, tschar) ::
                (_t'2, tschar) :: (_t'1, tschar) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Sset _i (Ecast (Econst_int (Int.repr 0) tint) tulong))
    (Sloop
      (Ssequence
        Sskip
        (Ssequence
          (Ssequence
            (Sset _t'2
              (Ederef
                (Ebinop Oadd (Etempvar _dest (tptr tschar))
                  (Etempvar _i tulong) (tptr tschar)) tschar))
            (Sset _d (Ecast (Etempvar _t'2 tschar) tschar)))
          (Sifthenelse (Ebinop Oeq (Etempvar _d tschar)
                               (Econst_int (Int.repr 0) tint) tint)
            Sbreak
            Sskip)))
      (Sset _i
        (Ebinop Oadd (Etempvar _i tulong) (Econst_int (Int.repr 1) tint)
          tulong))))
  (Ssequence
    (Sset _j (Ecast (Econst_int (Int.repr 0) tint) tulong))
    (Sloop
      (Ssequence
        Sskip
        (Ssequence
          (Ssequence
            (Sset _t'1
              (Ederef
                (Ebinop Oadd (Etempvar _src (tptr tschar))
                  (Etempvar _j tulong) (tptr tschar)) tschar))
            (Sset _d (Ecast (Etempvar _t'1 tschar) tschar)))
          (Ssequence
            (Sassign
              (Ederef
                (Ebinop Oadd (Etempvar _dest (tptr tschar))
```

```
                    (Ebinop Oadd (Etempvar _i tulong) (Etempvar _j tulong)
                      tulong) (tptr tschar)) tschar) (Etempvar _d tschar))
                (Sifthenelse (Ebinop Oeq (Etempvar _d tschar)
                                (Econst_int (Int.repr 0) tint) tint)
                  (Sreturn (Some (Etempvar _dest (tptr tschar))))
                  Sskip)))))
        (Sset _j
          (Ebinop Oadd (Etempvar _j tulong) (Econst_int (Int.repr 1) tint)
            tulong))))))
|}.
Definition f_strcmp := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := ((_str1, (tptr tschar)) :: (_str2, (tptr tschar)) :: nil);
  fn_vars := nil;
  fn_temps := ((_i, tulong) :: (_d1, tschar) :: (_d2, tschar) ::
                (_t'1, tint) :: (_t'3, tschar) :: (_t'2, tschar) :: nil);
  fn_body :=
(Ssequence
  (Sset _i (Ecast (Econst_int (Int.repr 0) tint) tulong))
  (Sloop
    (Ssequence
      Sskip
      (Ssequence
        (Ssequence
          (Sset _t'3
            (Ederef
              (Ebinop Oadd (Etempvar _str1 (tptr tschar))
                (Etempvar _i tulong) (tptr tschar)) tschar))
          (Sset _d1 (Ecast (Etempvar _t'3 tschar) tschar)))
        (Ssequence
          (Ssequence
            (Sset _t'2
              (Ederef
                (Ebinop Oadd (Etempvar _str2 (tptr tschar))
                  (Etempvar _i tulong) (tptr tschar)) tschar))
            (Sset _d2 (Ecast (Etempvar _t'2 tschar) tschar)))
          (Ssequence
            (Sifthenelse (Ebinop Oeq (Etempvar _d1 tschar)
                            (Econst_int (Int.repr 0) tint) tint)
              (Sset _t'1
                (Ecast
```

```
                    (Ebinop Oeq (Etempvar _d2 tschar)
                      (Econst_int (Int.repr 0) tint) tint) tbool))
                  (Sset _t'1 (Econst_int (Int.repr 0) tint)))
                (Sifthenelse (Etempvar _t'1 tint)
                  (Sreturn (Some (Econst_int (Int.repr 0) tint)))
                  (Sifthenelse (Ebinop Olt (Etempvar _d1 tschar)
                                 (Etempvar _d2 tschar) tint)
                    (Sreturn (Some (Eunop Oneg (Econst_int (Int.repr 1) tint)
                                     tint)))
                    (Sifthenelse (Ebinop Ogt (Etempvar _d1 tschar)
                                   (Etempvar _d2 tschar) tint)
                      (Sreturn (Some (Econst_int (Int.repr 1) tint)))
                      Sskip)))))))
    (Sset _i
      (Ebinop Oadd (Etempvar _i tulong) (Econst_int (Int.repr 1) tint)
        tulong))))
|}.

Definition f_example_call_strcpy := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := nil;
  fn_vars := ((_buf, (tarray tschar 10)) :: nil);
  fn_temps := ((_t'1, tschar) :: nil);
  fn_body :=
(Ssequence
  (Scall None
    (Evar _strcpy (Tfunction ((tptr tschar) :: (tptr tschar) :: nil)
                    (tptr tschar) cc_default))
    ((Evar _buf (tarray tschar 10)) ::
     (Evar ___stringlit_1 (tarray tschar 6)) :: nil))
  (Ssequence
    (Sset _t'1
      (Ederef
        (Ebinop Oadd (Evar _buf (tarray tschar 10))
          (Econst_int (Int.repr 0) tint) (tptr tschar)) tschar))
    (Sreturn (Some (Etempvar _t'1 tschar)))))
|}.

Definition composites : list composite_definition :=
nil.

Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
   Gfun(External (EF_runtime "__compcert_va_int32"
```

```
                    (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
    ((tptr tvoid) :: nil) tuint cc_default)) ::
(___compcert_va_int64,
  Gfun(External (EF_runtime "__compcert_va_int64"
                    (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
    ((tptr tvoid) :: nil) tulong cc_default)) ::
(___compcert_va_float64,
  Gfun(External (EF_runtime "__compcert_va_float64"
                    (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
    ((tptr tvoid) :: nil) tdouble cc_default)) ::
(___compcert_va_composite,
  Gfun(External (EF_runtime "__compcert_va_composite"
                    (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                      cc_default)) ((tptr tvoid) :: tulong :: nil)
    (tptr tvoid) cc_default)) ::
(___compcert_i64_dtos,
  Gfun(External (EF_runtime "__compcert_i64_dtos"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tlong cc_default)) ::
(___compcert_i64_dtou,
  Gfun(External (EF_runtime "__compcert_i64_dtou"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tulong cc_default)) ::
(___compcert_i64_stod,
  Gfun(External (EF_runtime "__compcert_i64_stod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tlong :: nil) tdouble cc_default)) ::
(___compcert_i64_utod,
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
(___compcert_i64_stof,
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
(___compcert_i64_utof,
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
```

```
                    cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) :: (___stringlit_1, Gvar v___stringlit_1) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                    (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
```

```
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                    (mksignature (AST.Xint16unsigned :: nil)
                      AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctz,
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_ctzl,
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctzll,
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_fabs,
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_fabsf,
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
```

```
    (tfloat :: nil) tfloat cc_default)) ::
(___builtin_fsqrt,
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_sqrt,
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_memcpy_aligned,
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
(___builtin_sel,
  Gfun(External (EF_builtin "__builtin_sel"
                    (mksignature (AST.Xbool :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                      cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
```

```
  Gfun(External (EF_builtin "__builtin_va_arg"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                    (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
```

```
                    (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                    AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
 (___builtin_fnmadd,
   Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
 (___builtin_fnmsub,
   Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
 (___builtin_fmax,
   Gfun(External (EF_builtin "__builtin_fmax"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
 (___builtin_fmin,
   Gfun(External (EF_builtin "__builtin_fmin"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
 (___builtin_debug,
   Gfun(External (EF_external "__builtin_debug"
                    (mksignature (AST.Xint :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
 (_strlen, Gfun(Internal f_strlen)) :: (_strchr, Gfun(Internal f_strchr)) ::
 (_strcpy, Gfun(Internal f_strcpy)) :: (_strcat, Gfun(Internal f_strcat)) ::
 (_strcmp, Gfun(Internal f_strcmp)) ::
 (_example_call_strcpy, Gfun(Internal f_example_call_strcpy)) :: nil).

Definition public_idents : list ident :=
(_example_call_strcpy :: _strcmp :: _strcat :: _strcpy :: _strchr ::
 _strlen :: ___builtin_debug :: ___builtin_fmin :: ___builtin_fmax ::
 ___builtin_fnmsub :: ___builtin_fnmadd :: ___builtin_fmsub ::
 ___builtin_fmadd :: ___builtin_clsll :: ___builtin_clsl :: ___builtin_cls ::
 ___builtin_expect :: ___builtin_unreachable :: ___builtin_va_end ::
 ___builtin_va_copy :: ___builtin_va_arg :: ___builtin_va_start ::
```

```
  ___builtin_membar :: ___builtin_annot_intval :: ___builtin_annot ::
  ___builtin_sel :: ___builtin_memcpy_aligned :: ___builtin_sqrt ::
  ___builtin_fsqrt :: ___builtin_fabsf :: ___builtin_fabs ::
  ___builtin_ctzll :: ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll ::
  ___builtin_clzl :: ___builtin_clz :: ___builtin_bswap16 ::
  ___builtin_bswap32 :: ___builtin_bswap :: ___builtin_bswap64 ::
  ___compcert_i64_umulh :: ___compcert_i64_smulh :: ___compcert_i64_sar ::
  ___compcert_i64_shr :: ___compcert_i64_shl :: ___compcert_i64_umod ::
  ___compcert_i64_smod :: ___compcert_i64_udiv :: ___compcert_i64_sdiv ::
  ___compcert_i64_utof :: ___compcert_i64_stof :: ___compcert_i64_utod ::
  ___compcert_i64_stod :: ___compcert_i64_dtou :: ___compcert_i64_dtos ::
  ___compcert_va_composite :: ___compcert_va_float64 ::
  ___compcert_va_int64 :: ___compcert_va_int32 :: nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 6

Library VC.hash

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "hash.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 23%positive.
Definition ___builtin_annot_intval : ident := 24%positive.
Definition ___builtin_bswap : ident := 8%positive.
Definition ___builtin_bswap16 : ident := 10%positive.
Definition ___builtin_bswap32 : ident := 9%positive.
Definition ___builtin_bswap64 : ident := 7%positive.
Definition ___builtin_cls : ident := 32%positive.
Definition ___builtin_clsl : ident := 33%positive.
Definition ___builtin_clsll : ident := 34%positive.
Definition ___builtin_clz : ident := 11%positive.
```

```
Definition ___builtin_clzl : ident := 12%positive.
Definition ___builtin_clzll : ident := 13%positive.
Definition ___builtin_ctz : ident := 14%positive.
Definition ___builtin_ctzl : ident := 15%positive.
Definition ___builtin_ctzll : ident := 16%positive.
Definition ___builtin_debug : ident := 41%positive.
Definition ___builtin_expect : ident := 31%positive.
Definition ___builtin_fabs : ident := 17%positive.
Definition ___builtin_fabsf : ident := 18%positive.
Definition ___builtin_fmadd : ident := 35%positive.
Definition ___builtin_fmax : ident := 39%positive.
Definition ___builtin_fmin : ident := 40%positive.
Definition ___builtin_fmsub : ident := 36%positive.
Definition ___builtin_fnmadd : ident := 37%positive.
Definition ___builtin_fnmsub : ident := 38%positive.
Definition ___builtin_fsqrt : ident := 19%positive.
Definition ___builtin_membar : ident := 25%positive.
Definition ___builtin_memcpy_aligned : ident := 21%positive.
Definition ___builtin_sel : ident := 22%positive.
Definition ___builtin_sqrt : ident := 20%positive.
Definition ___builtin_unreachable : ident := 30%positive.
Definition ___builtin_va_arg : ident := 27%positive.
Definition ___builtin_va_copy : ident := 28%positive.
Definition ___builtin_va_end : ident := 29%positive.
Definition ___builtin_va_start : ident := 26%positive.
Definition ___compcert_i64_dtos : ident := 69%positive.
Definition ___compcert_i64_dtou : ident := 70%positive.
Definition ___compcert_i64_sar : ident := 81%positive.
Definition ___compcert_i64_sdiv : ident := 75%positive.
Definition ___compcert_i64_shl : ident := 79%positive.
Definition ___compcert_i64_shr : ident := 80%positive.
Definition ___compcert_i64_smod : ident := 77%positive.
Definition ___compcert_i64_smulh : ident := 82%positive.
Definition ___compcert_i64_stod : ident := 71%positive.
Definition ___compcert_i64_stof : ident := 73%positive.
Definition ___compcert_i64_udiv : ident := 76%positive.
Definition ___compcert_i64_umod : ident := 78%positive.
Definition ___compcert_i64_umulh : ident := 83%positive.
Definition ___compcert_i64_utod : ident := 72%positive.
Definition ___compcert_i64_utof : ident := 74%positive.
Definition ___compcert_va_composite : ident := 68%positive.
Definition ___compcert_va_float64 : ident := 67%positive.
```

```
Definition ___compcert_va_int32 : ident := 65%positive.
Definition ___compcert_va_int64 : ident := 66%positive.
Definition _b : ident := 58%positive.
Definition _buckets : ident := 6%positive.
Definition _c : ident := 50%positive.
Definition _cell : ident := 1%positive.
Definition _copy_string : ident := 53%positive.
Definition _count : ident := 3%positive.
Definition _exit : ident := 43%positive.
Definition _get : ident := 59%positive.
Definition _h : ident := 57%positive.
Definition _hash : ident := 51%positive.
Definition _hashtable : ident := 5%positive.
Definition _i : ident := 49%positive.
Definition _incr : ident := 63%positive.
Definition _incr_list : ident := 62%positive.
Definition _incrx : ident := 64%positive.
Definition _key : ident := 2%positive.
Definition _main : ident := 84%positive.
Definition _malloc : ident := 42%positive.
Definition _n : ident := 48%positive.
Definition _new_cell : ident := 55%positive.
Definition _new_table : ident := 54%positive.
Definition _next : ident := 4%positive.
Definition _p : ident := 52%positive.
Definition _r : ident := 61%positive.
Definition _r0 : ident := 60%positive.
Definition _s : ident := 47%positive.
Definition _strcmp : ident := 46%positive.
Definition _strcpy : ident := 45%positive.
Definition _strlen : ident := 44%positive.
Definition _table : ident := 56%positive.
Definition _t'1 : ident := 85%positive.
Definition _t'2 : ident := 86%positive.
Definition _t'3 : ident := 87%positive.
Definition _t'4 : ident := 88%positive.
Definition _t'5 : ident := 89%positive.
Definition _t'6 : ident := 90%positive.

Definition f_hash := {|
  fn_return := tuint;
  fn_callconv := cc_default;
  fn_params := ((_s, (tptr tschar)) :: nil);
```

```
  fn_vars := nil;
  fn_temps := ((_n, tuint) :: (_i, tulong) :: (_c, tint) :: nil);
  fn_body :=
(Ssequence
  (Sset _n (Econst_int (Int.repr 0) tint))
  (Ssequence
    (Sset _i (Ecast (Econst_int (Int.repr 0) tint) tulong))
    (Ssequence
      (Sset _c
        (Ederef
          (Ebinop Oadd (Etempvar _s (tptr tschar)) (Etempvar _i tulong)
            (tptr tschar)) tschar))
      (Ssequence
        (Swhile
          (Etempvar _c tint)
          (Ssequence
            (Sset _n
              (Ebinop Oadd
                (Ebinop Omul (Etempvar _n tuint)
                  (Econst_int (Int.repr 65599) tuint) tuint)
                (Ecast (Etempvar _c tint) tuint) tuint))
            (Ssequence
              (Sset _i
                (Ebinop Oadd (Etempvar _i tulong)
                  (Econst_int (Int.repr 1) tint) tulong))
              (Sset _c
                (Ederef
                  (Ebinop Oadd (Etempvar _s (tptr tschar))
                    (Etempvar _i tulong) (tptr tschar)) tschar)))))
        (Sreturn (Some (Etempvar _n tuint)))))))
|}.
Definition f_copy_string := {|
  fn_return := (tptr tschar);
  fn_callconv := cc_default;
  fn_params := ((_s, (tptr tschar)) :: nil);
  fn_vars := nil;
  fn_temps := ((_n, tulong) :: (_p, (tptr tschar)) :: (_t'2, (tptr tvoid)) ::
               (_t'1, tulong) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
```

```
      (Evar _strlen (Tfunction ((tptr tschar) :: nil) tulong cc_default))
      ((Etempvar _s (tptr tschar)) :: nil))
    (Sset _n
      (Ebinop Oadd (Etempvar _t'1 tulong) (Econst_int (Int.repr 1) tint)
        tulong)))
  (Ssequence
    (Ssequence
      (Scall (Some _t'2)
        (Evar _malloc (Tfunction (tulong :: nil) (tptr tvoid) cc_default))
        ((Etempvar _n tulong) :: nil))
      (Sset _p (Etempvar _t'2 (tptr tvoid))))
    (Ssequence
      (Sifthenelse (Eunop Onotbool (Etempvar _p (tptr tschar)) tint)
        (Scall None (Evar _exit (Tfunction (tint :: nil) tvoid cc_default))
          ((Econst_int (Int.repr 1) tint) :: nil))
        Sskip)
      (Ssequence
        (Scall None
          (Evar _strcpy (Tfunction ((tptr tschar) :: (tptr tschar) :: nil)
                                   (tptr tschar) cc_default))
          ((Etempvar _p (tptr tschar)) :: (Etempvar _s (tptr tschar)) :: nil))
        (Sreturn (Some (Etempvar _p (tptr tschar))))))))
|}.
Definition f_new_table := {|
  fn_return := (tptr (Tstruct _hashtable noattr));
  fn_callconv := cc_default;
  fn_params := nil;
  fn_vars := nil;
  fn_temps := ((_i, tint) :: (_p, (tptr (Tstruct _hashtable noattr))) ::
                 (_t'1, (tptr tvoid)) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _malloc (Tfunction (tulong :: nil) (tptr tvoid) cc_default))
      ((Esizeof (Tstruct _hashtable noattr) tulong) :: nil))
    (Sset _p
      (Ecast (Etempvar _t'1 (tptr tvoid)) (tptr (Tstruct _hashtable noattr)))))
  (Ssequence
    (Sifthenelse (Eunop Onotbool
                   (Etempvar _p (tptr (Tstruct _hashtable noattr))) tint)
      (Scall None (Evar _exit (Tfunction (tint :: nil) tvoid cc_default))
```

```
          ((Econst_int (Int.repr 1) tint) :: nil))
        Sskip)
      (Ssequence
        (Ssequence
          (Sset _i (Econst_int (Int.repr 0) tint))
          (Sloop
            (Ssequence
              (Sifthenelse (Ebinop Olt (Etempvar _i tint)
                                       (Econst_int (Int.repr 109) tint) tint)
                Sskip
                Sbreak)
              (Sassign
                (Ederef
                  (Ebinop Oadd
                    (Efield
                      (Ederef (Etempvar _p (tptr (Tstruct _hashtable noattr)))
                        (Tstruct _hashtable noattr)) _buckets
                      (tarray (tptr (Tstruct _cell noattr)) 109))
                    (Etempvar _i tint) (tptr (tptr (Tstruct _cell noattr))))
                  (tptr (Tstruct _cell noattr)))
                (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid))))
            (Sset _i
              (Ebinop Oadd (Etempvar _i tint) (Econst_int (Int.repr 1) tint)
                tint))))
        (Sreturn (Some (Etempvar _p (tptr (Tstruct _hashtable noattr)))))))
|}.
Definition f_new_cell := {|
  fn_return := (tptr (Tstruct _cell noattr));
  fn_callconv := cc_default;
  fn_params := ((_key, (tptr tschar)) :: (_count, tint) ::
                (_next, (tptr (Tstruct _cell noattr))) :: nil);
  fn_vars := nil;
  fn_temps := ((_p, (tptr (Tstruct _cell noattr))) ::
               (_t'2, (tptr tschar)) :: (_t'1, (tptr tvoid)) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _malloc (Tfunction (tulong :: nil) (tptr tvoid) cc_default))
      ((Esizeof (Tstruct _cell noattr) tulong) :: nil))
    (Sset _p
      (Ecast (Etempvar _t'1 (tptr tvoid)) (tptr (Tstruct _cell noattr)))))
```

```
  (Ssequence
    (Sifthenelse (Eunop Onotbool (Etempvar _p (tptr (Tstruct _cell noattr)))
                   tint)
      (Scall None (Evar _exit (Tfunction (tint :: nil) tvoid cc_default))
        ((Econst_int (Int.repr 1) tint) :: nil))
      Sskip)
    (Ssequence
      (Ssequence
        (Scall (Some _t'2)
          (Evar _copy_string (Tfunction ((tptr tschar) :: nil) (tptr tschar)
                               cc_default))
          ((Etempvar _key (tptr tschar)) :: nil))
        (Sassign
          (Efield
            (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
              (Tstruct _cell noattr)) _key (tptr tschar))
          (Etempvar _t'2 (tptr tschar))))
      (Ssequence
        (Sassign
          (Efield
            (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
              (Tstruct _cell noattr)) _count tuint) (Etempvar _count tint))
        (Ssequence
          (Sassign
            (Efield
              (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                (Tstruct _cell noattr)) _next (tptr (Tstruct _cell noattr)))
            (Etempvar _next (tptr (Tstruct _cell noattr))))
          (Sreturn (Some (Etempvar _p (tptr (Tstruct _cell noattr))))))))))
|}.
Definition f_get := {|
  fn_return := tuint;
  fn_callconv := cc_default;
  fn_params := ((_table, (tptr (Tstruct _hashtable noattr))) ::
                (_s, (tptr tschar)) :: nil);
  fn_vars := nil;
  fn_temps := ((_h, tuint) :: (_b, tuint) ::
               (_p, (tptr (Tstruct _cell noattr))) :: (_t'2, tint) ::
               (_t'1, tuint) :: (_t'4, (tptr tschar)) :: (_t'3, tuint) ::
               nil);
  fn_body :=
(Ssequence
```

```
(Ssequence
  (Scall (Some _t'1)
    (Evar _hash (Tfunction ((tptr tschar) :: nil) tuint cc_default))
    ((Etempvar _s (tptr tschar)) :: nil))
  (Sset _h (Etempvar _t'1 tuint)))
(Ssequence
  (Sset _b
    (Ebinop Omod (Etempvar _h tuint) (Econst_int (Int.repr 109) tint)
      tuint))
  (Ssequence
    (Sset _p
      (Ederef
        (Ebinop Oadd
          (Efield
            (Ederef (Etempvar _table (tptr (Tstruct _hashtable noattr)))
              (Tstruct _hashtable noattr)) _buckets
            (tarray (tptr (Tstruct _cell noattr)) 109)) (Etempvar _b tuint)
          (tptr (tptr (Tstruct _cell noattr))))
        (tptr (Tstruct _cell noattr))))
    (Ssequence
      (Swhile
        (Etempvar _p (tptr (Tstruct _cell noattr)))
        (Ssequence
          (Ssequence
            (Ssequence
              (Sset _t'4
                (Efield
                  (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                    (Tstruct _cell noattr)) _key (tptr tschar)))
              (Scall (Some _t'2)
                (Evar _strcmp (Tfunction
                                ((tptr tschar) :: (tptr tschar) :: nil)
                                tint cc_default))
                ((Etempvar _t'4 (tptr tschar)) ::
                  (Etempvar _s (tptr tschar)) :: nil)))
            (Sifthenelse (Ebinop Oeq (Etempvar _t'2 tint)
                           (Econst_int (Int.repr 0) tint) tint)
              (Ssequence
                (Sset _t'3
                  (Efield
                    (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                      (Tstruct _cell noattr)) _count tuint))
```

```
                    (Sreturn (Some (Etempvar _t'3 tuint))))
                  Sskip))
            (Sset _p
              (Efield
                (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                  (Tstruct _cell noattr)) _next
                (tptr (Tstruct _cell noattr))))))
        (Sreturn (Some (Econst_int (Int.repr 0) tint)))))))
|}.
Definition f_incr_list := {|
  fn_return := tvoid;
  fn_callconv := cc_default;
  fn_params := ((_r0, (tptr (tptr (Tstruct _cell noattr)))) ::
                (_s, (tptr tschar)) :: nil);
  fn_vars := nil;
  fn_temps := ((_p, (tptr (Tstruct _cell noattr))) ::
               (_r, (tptr (tptr (Tstruct _cell noattr)))) :: (_t'2, tint) ::
               (_t'1, (tptr (Tstruct _cell noattr))) ::
               (_t'4, (tptr tschar)) :: (_t'3, tuint) :: nil);
  fn_body :=
(Ssequence
  (Sset _r (Etempvar _r0 (tptr (tptr (Tstruct _cell noattr)))))
  (Sloop
    (Ssequence
      Sskip
      (Ssequence
        (Sset _p
          (Ederef (Etempvar _r (tptr (tptr (Tstruct _cell noattr))))
            (tptr (Tstruct _cell noattr))))
        (Ssequence
          (Sifthenelse (Eunop Onotbool
                              (Etempvar _p (tptr (Tstruct _cell noattr))) tint)
            (Ssequence
              (Ssequence
                (Scall (Some _t'1)
                  (Evar _new_cell (Tfunction
                                    ((tptr tschar) :: tint ::
                                     (tptr (Tstruct _cell noattr)) :: nil)
                                    (tptr (Tstruct _cell noattr)) cc_default))
                  ((Etempvar _s (tptr tschar)) ::
                   (Econst_int (Int.repr 1) tint) ::
                   (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid)) ::
```

```
                    nil))
                (Sassign
                  (Ederef (Etempvar _r (tptr (tptr (Tstruct _cell noattr))))
                    (tptr (Tstruct _cell noattr)))
                  (Etempvar _t'1 (tptr (Tstruct _cell noattr)))))
                (Sreturn None))
              Sskip)
            (Ssequence
              (Ssequence
                (Sset _t'4
                  (Efield
                    (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                      (Tstruct _cell noattr)) _key (tptr tschar)))
                (Scall (Some _t'2)
                  (Evar _strcmp (Tfunction
                                  ((tptr tschar) :: (tptr tschar) :: nil) tint
                                  cc_default))
                  ((Etempvar _t'4 (tptr tschar)) ::
                   (Etempvar _s (tptr tschar)) :: nil)))
              (Sifthenelse (Ebinop Oeq (Etempvar _t'2 tint)
                             (Econst_int (Int.repr 0) tint) tint)
                (Ssequence
                  (Ssequence
                    (Sset _t'3
                      (Efield
                        (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                          (Tstruct _cell noattr)) _count tuint))
                    (Sassign
                      (Efield
                        (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                          (Tstruct _cell noattr)) _count tuint)
                      (Ebinop Oadd (Etempvar _t'3 tuint)
                        (Econst_int (Int.repr 1) tint) tuint)))
                  (Sreturn None))
                Sskip)))))
      (Sset _r
        (Eaddrof
          (Efield
            (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
              (Tstruct _cell noattr)) _next (tptr (Tstruct _cell noattr)))
          (tptr (tptr (Tstruct _cell noattr)))))))
|}.
```

```
Definition f_incr := {|
  fn_return := tvoid;
  fn_callconv := cc_default;
  fn_params := ((_table, (tptr (Tstruct _hashtable noattr))) ::
                (_s, (tptr tschar)) :: nil);
  fn_vars := nil;
  fn_temps := ((_h, tuint) :: (_b, tuint) :: (_t'1, tuint) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _hash (Tfunction ((tptr tschar) :: nil) tuint cc_default))
      ((Etempvar _s (tptr tschar)) :: nil))
    (Sset _h (Etempvar _t'1 tuint)))
  (Ssequence
    (Sset _b
      (Ebinop Omod (Etempvar _h tuint) (Econst_int (Int.repr 109) tint)
        tuint))
    (Scall None
      (Evar _incr_list (Tfunction
                         ((tptr (tptr (Tstruct _cell noattr))) ::
                          (tptr tschar) :: nil) tvoid cc_default))
      ((Ebinop Oadd
         (Efield
           (Ederef (Etempvar _table (tptr (Tstruct _hashtable noattr)))
             (Tstruct _hashtable noattr)) _buckets
           (tarray (tptr (Tstruct _cell noattr)) 109)) (Etempvar _b tuint)
         (tptr (tptr (Tstruct _cell noattr)))) ::
       (Etempvar _s (tptr tschar)) :: nil))))
|}.
Definition f_incrx := {|
  fn_return := tvoid;
  fn_callconv := cc_default;
  fn_params := ((_table, (tptr (Tstruct _hashtable noattr))) ::
                (_s, (tptr tschar)) :: nil);
  fn_vars := nil;
  fn_temps := ((_h, tuint) :: (_b, tuint) ::
               (_p, (tptr (Tstruct _cell noattr))) ::
               (_t'3, (tptr (Tstruct _cell noattr))) :: (_t'2, tint) ::
               (_t'1, tuint) :: (_t'6, (tptr tschar)) :: (_t'5, tuint) ::
               (_t'4, (tptr (Tstruct _cell noattr))) :: nil);
  fn_body :=
```

```
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _hash (Tfunction ((tptr tschar) :: nil) tuint cc_default))
      ((Etempvar _s (tptr tschar)) :: nil))
    (Sset _h (Etempvar _t'1 tuint)))
  (Ssequence
    (Sset _b
      (Ebinop Omod (Etempvar _h tuint) (Econst_int (Int.repr 109) tint)
        tuint))
    (Ssequence
      (Sset _p
        (Ederef
          (Ebinop Oadd
            (Efield
              (Ederef (Etempvar _table (tptr (Tstruct _hashtable noattr)))
                (Tstruct _hashtable noattr)) _buckets
              (tarray (tptr (Tstruct _cell noattr)) 109)) (Etempvar _b tuint)
            (tptr (tptr (Tstruct _cell noattr))))
          (tptr (Tstruct _cell noattr))))
      (Ssequence
        (Swhile
          (Etempvar _p (tptr (Tstruct _cell noattr)))
          (Ssequence
            (Ssequence
              (Ssequence
                (Sset _t'6
                  (Efield
                    (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                      (Tstruct _cell noattr)) _key (tptr tschar)))
                (Scall (Some _t'2)
                  (Evar _strcmp (Tfunction
                                  ((tptr tschar) :: (tptr tschar) :: nil)
                                  tint cc_default))
                  ((Etempvar _t'6 (tptr tschar)) ::
                    (Etempvar _s (tptr tschar)) :: nil)))
              (Sifthenelse (Ebinop Oeq (Etempvar _t'2 tint)
                             (Econst_int (Int.repr 0) tint) tint)
                (Ssequence
                  (Ssequence
                    (Sset _t'5
                      (Efield
```

```
                    (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                      (Tstruct _cell noattr)) _count tuint))
                (Sassign
                  (Efield
                    (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                      (Tstruct _cell noattr)) _count tuint)
                  (Ebinop Oadd (Etempvar _t'5 tuint)
                    (Econst_int (Int.repr 1) tint) tuint)))
              (Sreturn None))
            Sskip))
        (Sset _p
          (Efield
            (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
              (Tstruct _cell noattr)) _next
            (tptr (Tstruct _cell noattr))))))
  (Ssequence
    (Ssequence
      (Sset _t'4
        (Ederef
          (Ebinop Oadd
            (Efield
              (Ederef
                (Etempvar _table (tptr (Tstruct _hashtable noattr)))
                (Tstruct _hashtable noattr)) _buckets
              (tarray (tptr (Tstruct _cell noattr)) 109))
            (Etempvar _b tuint) (tptr (tptr (Tstruct _cell noattr))))
          (tptr (Tstruct _cell noattr))))
      (Scall (Some _t'3)
        (Evar _new_cell (Tfunction
                          ((tptr tschar) :: tint ::
                            (tptr (Tstruct _cell noattr)) :: nil)
                          (tptr (Tstruct _cell noattr)) cc_default))
        ((Etempvar _s (tptr tschar)) ::
         (Econst_int (Int.repr 1) tint) ::
         (Etempvar _t'4 (tptr (Tstruct _cell noattr))) :: nil)))
    (Sassign
      (Ederef
        (Ebinop Oadd
          (Efield
            (Ederef
              (Etempvar _table (tptr (Tstruct _hashtable noattr)))
              (Tstruct _hashtable noattr)) _buckets
```

```
                    (tarray (tptr (Tstruct _cell noattr)) 109))
                  (Etempvar _b tuint) (tptr (tptr (Tstruct _cell noattr))))
                (tptr (Tstruct _cell noattr)))
              (Etempvar _t'3 (tptr (Tstruct _cell noattr)))))))))
|}.

Definition composites : list composite_definition :=
(Composite _cell Struct
    (Member_plain _key (tptr tschar) :: Member_plain _count tuint ::
     Member_plain _next (tptr (Tstruct _cell noattr)) :: nil)
    noattr ::
 Composite _hashtable Struct
    (Member_plain _buckets (tarray (tptr (Tstruct _cell noattr)) 109) :: nil)
    noattr :: nil).

Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
    Gfun(External (EF_runtime "__compcert_va_int32"
                    (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
      ((tptr tvoid) :: nil) tuint cc_default)) ::
 (___compcert_va_int64,
    Gfun(External (EF_runtime "__compcert_va_int64"
                    (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
      ((tptr tvoid) :: nil) tulong cc_default)) ::
 (___compcert_va_float64,
    Gfun(External (EF_runtime "__compcert_va_float64"
                    (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
      ((tptr tvoid) :: nil) tdouble cc_default)) ::
 (___compcert_va_composite,
    Gfun(External (EF_runtime "__compcert_va_composite"
                    (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                      cc_default)) ((tptr tvoid) :: tulong :: nil)
      (tptr tvoid) cc_default)) ::
 (___compcert_i64_dtos,
    Gfun(External (EF_runtime "__compcert_i64_dtos"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
      (tdouble :: nil) tlong cc_default)) ::
 (___compcert_i64_dtou,
    Gfun(External (EF_runtime "__compcert_i64_dtou"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
      (tdouble :: nil) tulong cc_default)) ::
 (___compcert_i64_stod,
    Gfun(External (EF_runtime "__compcert_i64_stod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
```

```
    (tlong :: nil) tdouble cc_default)) ::
(___compcert_i64_utod,
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
(___compcert_i64_stof,
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
(___compcert_i64_utof,
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
```

```
                    cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                    (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                    (mksignature (AST.Xint16unsigned :: nil)
                      AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctz,
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
```

```
    (tuint :: nil) tint cc_default)) ::
(___builtin_ctzl,
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctzll,
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_fabs,
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_fabsf,
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
    (tfloat :: nil) tfloat cc_default)) ::
(___builtin_fsqrt,
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_sqrt,
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_memcpy_aligned,
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
(___builtin_sel,
  Gfun(External (EF_builtin "__builtin_sel"
                    (mksignature (AST.Xbool :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
```

```
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                      cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
  Gfun(External (EF_builtin "__builtin_va_arg"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                    (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
```

```
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmadd,
  Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmsub,
  Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmax,
  Gfun(External (EF_builtin "__builtin_fmax"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
    cc_default)) ::
(___builtin_fmin,
  Gfun(External (EF_builtin "__builtin_fmin"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
    cc_default)) ::
(___builtin_debug,
```

```
    Gfun(External (EF_external "__builtin_debug"
                        (mksignature (AST.Xint :: nil) AST.Xvoid
                          {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
  (_malloc, Gfun(External EF_malloc (tulong :: nil) (tptr tvoid) cc_default)) ::
  (_exit,
    Gfun(External (EF_external "exit"
                        (mksignature (AST.Xint :: nil) AST.Xvoid cc_default))
      (tint :: nil) tvoid cc_default)) ::
  (_strlen,
    Gfun(External (EF_external "strlen"
                        (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
      ((tptr tschar) :: nil) tulong cc_default)) ::
  (_strcpy,
    Gfun(External (EF_external "strcpy"
                        (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xptr
                          cc_default)) ((tptr tschar) :: (tptr tschar) :: nil)
      (tptr tschar) cc_default)) ::
  (_strcmp,
    Gfun(External (EF_external "strcmp"
                        (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xint
                          cc_default)) ((tptr tschar) :: (tptr tschar) :: nil)
      tint cc_default)) :: (_hash, Gfun(Internal f_hash)) ::
  (_copy_string, Gfun(Internal f_copy_string)) ::
  (_new_table, Gfun(Internal f_new_table)) ::
  (_new_cell, Gfun(Internal f_new_cell)) :: (_get, Gfun(Internal f_get)) ::
  (_incr_list, Gfun(Internal f_incr_list)) ::
  (_incr, Gfun(Internal f_incr)) :: (_incrx, Gfun(Internal f_incrx)) :: nil).

Definition public_idents : list ident :=
(_incrx :: _incr :: _incr_list :: _get :: _new_cell :: _new_table ::
  _copy_string :: _hash :: _strcmp :: _strcpy :: _strlen :: _exit ::
  _malloc :: ___builtin_debug :: ___builtin_fmin :: ___builtin_fmax ::
  ___builtin_fnmsub :: ___builtin_fnmadd :: ___builtin_fmsub ::
  ___builtin_fmadd :: ___builtin_clsll :: ___builtin_clsl :: ___builtin_cls ::
  ___builtin_expect :: ___builtin_unreachable :: ___builtin_va_end ::
  ___builtin_va_copy :: ___builtin_va_arg :: ___builtin_va_start ::
  ___builtin_membar :: ___builtin_annot_intval :: ___builtin_annot ::
  ___builtin_sel :: ___builtin_memcpy_aligned :: ___builtin_sqrt ::
  ___builtin_fsqrt :: ___builtin_fabsf :: ___builtin_fabs ::
  ___builtin_ctzll :: ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll ::
  ___builtin_clzl :: ___builtin_clz :: ___builtin_bswap16 ::
```

```
  ___builtin_bswap32 :: ___builtin_bswap :: ___builtin_bswap64 ::
  ___compcert_i64_umulh :: ___compcert_i64_smulh :: ___compcert_i64_sar ::
  ___compcert_i64_shr :: ___compcert_i64_shl :: ___compcert_i64_umod ::
  ___compcert_i64_smod :: ___compcert_i64_udiv :: ___compcert_i64_sdiv ::
  ___compcert_i64_utof :: ___compcert_i64_stof :: ___compcert_i64_utod ::
  ___compcert_i64_stod :: ___compcert_i64_dtou :: ___compcert_i64_dtos ::
  ___compcert_va_composite :: ___compcert_va_float64 ::
  ___compcert_va_int64 :: ___compcert_va_int32 :: nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 7

Library VC.hints

```
Require Import VST.floyd.proofauto.
Require Import VST.floyd.library.

Ltac verif_stack_free_hint1 :=
match goal with
⊢ semax ?D (PROPx _ (LOCALx ?Q (SEPx ?R)))
                (Ssequence
                      (Scall _ (Evar ?free (Tfunction [tptr tvoid] tvoid cc_default))
                      (Etempvar ?i _ :: _)) _) _ ⇒
  match Q with context [temp i ?q] ⇒
   match R with context [data_at _ ?t _ q] ⇒
        unify (Maps.PTree.get free (glob_specs D)) (Some library.free_spec');
        idtac "When doing forward_call through this call to" free
  "you need to supply a WITH-witness of type (type*val*globals) and you need to supply a proof that"
    q "<>nullval. Look in your SEP clauses for 'data_at _" t "_" q"', which will be useful for both."
"Regarding the proof, assert_PROP(...) will make use of the fact that data_at cannot be a nullval."
"Regarding the witness, you should look at the funspec declared for" free
"to see what will be needed; look in Verif_stack.v at free_spec_example."
"But in particular, for the type, you can use the second argument of the data_at, that is," t
".";
  match goal with a : globals ⊢ _ ⇒
    idtac "Regarding the 'globals', you have" a ": globals above the line."
   end
  end end
end.

Ltac verif_stack_malloc_hint1_aux D R c :=
  match c with
```

```
  | Ssequence ?c1 _ ⇒ verif_stack_malloc_hint1_aux D R c1
  | Scall _ (Evar ?malloc
                  (Tfunction [tuint] (tptr tvoid) cc_default))
                 (cons (Esizeof ?t _) nil) ⇒
      match R with context [mem_mgr ?gv] ⇒
          idtac "try 'forward_call (" t "," gv ")'"
     end
  end.

Ltac verif_stack_malloc_hint1 :=
match goal with ⊢ semax ?D (PROPx _ (LOCALx _ (SEPx ?R))) ?c _ ⇒
     verif_stack_malloc_hint1_aux D R c
end.

Ltac vc_special_hint :=
  first
    [ verif_stack_free_hint1
    | verif_stack_malloc_hint1
    ];
  idtac "THAT WAS NOT A STANDARD VST HINT, IT IS SPECIAL FOR THE VC
VOLUME OF SOFTWARE FOUNDATIONS."
  "STANDARD VST HINTS WOULD BE AS FOLLOWS: ".

Ltac hint_special ::= try vc_special_hint.
```

Chapter 8

Library VC.stdlib

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "stdlib.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 17%positive.
Definition ___builtin_annot_intval : ident := 18%positive.
Definition ___builtin_bswap : ident := 2%positive.
Definition ___builtin_bswap16 : ident := 4%positive.
Definition ___builtin_bswap32 : ident := 3%positive.
Definition ___builtin_bswap64 : ident := 1%positive.
Definition ___builtin_cls : ident := 26%positive.
Definition ___builtin_clsl : ident := 27%positive.
Definition ___builtin_clsll : ident := 28%positive.
Definition ___builtin_clz : ident := 5%positive.
```

```
Definition ___builtin_clzl : ident := 6%positive.
Definition ___builtin_clzll : ident := 7%positive.
Definition ___builtin_ctz : ident := 8%positive.
Definition ___builtin_ctzl : ident := 9%positive.
Definition ___builtin_ctzll : ident := 10%positive.
Definition ___builtin_debug : ident := 35%positive.
Definition ___builtin_expect : ident := 25%positive.
Definition ___builtin_fabs : ident := 11%positive.
Definition ___builtin_fabsf : ident := 12%positive.
Definition ___builtin_fmadd : ident := 29%positive.
Definition ___builtin_fmax : ident := 33%positive.
Definition ___builtin_fmin : ident := 34%positive.
Definition ___builtin_fmsub : ident := 30%positive.
Definition ___builtin_fnmadd : ident := 31%positive.
Definition ___builtin_fnmsub : ident := 32%positive.
Definition ___builtin_fsqrt : ident := 13%positive.
Definition ___builtin_membar : ident := 19%positive.
Definition ___builtin_memcpy_aligned : ident := 15%positive.
Definition ___builtin_sel : ident := 16%positive.
Definition ___builtin_sqrt : ident := 14%positive.
Definition ___builtin_unreachable : ident := 24%positive.
Definition ___builtin_va_arg : ident := 21%positive.
Definition ___builtin_va_copy : ident := 22%positive.
Definition ___builtin_va_end : ident := 23%positive.
Definition ___builtin_va_start : ident := 20%positive.
Definition ___compcert_i64_dtos : ident := 44%positive.
Definition ___compcert_i64_dtou : ident := 45%positive.
Definition ___compcert_i64_sar : ident := 56%positive.
Definition ___compcert_i64_sdiv : ident := 50%positive.
Definition ___compcert_i64_shl : ident := 54%positive.
Definition ___compcert_i64_shr : ident := 55%positive.
Definition ___compcert_i64_smod : ident := 52%positive.
Definition ___compcert_i64_smulh : ident := 57%positive.
Definition ___compcert_i64_stod : ident := 46%positive.
Definition ___compcert_i64_stof : ident := 48%positive.
Definition ___compcert_i64_udiv : ident := 51%positive.
Definition ___compcert_i64_umod : ident := 53%positive.
Definition ___compcert_i64_umulh : ident := 58%positive.
Definition ___compcert_i64_utod : ident := 47%positive.
Definition ___compcert_i64_utof : ident := 49%positive.
Definition ___compcert_va_composite : ident := 43%positive.
Definition ___compcert_va_float64 : ident := 42%positive.
```

```
Definition ___compcert_va_int32 : ident := 40%positive.
Definition ___compcert_va_int64 : ident := 41%positive.
Definition _exit : ident := 38%positive.
Definition _free : ident := 37%positive.
Definition _main : ident := 59%positive.
Definition _malloc : ident := 36%positive.
Definition _placeholder : ident := 39%positive.

Definition f_placeholder := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := nil;
  fn_vars := nil;
  fn_temps := nil;
  fn_body :=
(Sreturn (Some (Econst_int (Int.repr 0) tint)))
|}.

Definition composites : list composite_definition :=
nil.

Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
   Gfun(External (EF_runtime "__compcert_va_int32"
                     (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
     ((tptr tvoid) :: nil) tuint cc_default)) ::
 (___compcert_va_int64,
   Gfun(External (EF_runtime "__compcert_va_int64"
                     (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
     ((tptr tvoid) :: nil) tulong cc_default)) ::
 (___compcert_va_float64,
   Gfun(External (EF_runtime "__compcert_va_float64"
                     (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
     ((tptr tvoid) :: nil) tdouble cc_default)) ::
 (___compcert_va_composite,
   Gfun(External (EF_runtime "__compcert_va_composite"
                     (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                       cc_default)) ((tptr tvoid) :: tulong :: nil)
     (tptr tvoid) cc_default)) ::
 (___compcert_i64_dtos,
   Gfun(External (EF_runtime "__compcert_i64_dtos"
                     (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
     (tdouble :: nil) tlong cc_default)) ::
 (___compcert_i64_dtou,
   Gfun(External (EF_runtime "__compcert_i64_dtou"
```

```
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tulong cc_default)) ::
("___compcert_i64_stod",
  Gfun(External (EF_runtime "__compcert_i64_stod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tlong :: nil) tdouble cc_default)) ::
("___compcert_i64_utod",
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
("___compcert_i64_stof",
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
("___compcert_i64_utof",
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
("___compcert_i64_sdiv",
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
("___compcert_i64_udiv",
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
("___compcert_i64_smod",
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
("___compcert_i64_umod",
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
("___compcert_i64_shl",
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
("___compcert_i64_shr",
  Gfun(External (EF_runtime "__compcert_i64_shr"
```

```
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                    (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                    (mksignature (AST.Xint16unsigned :: nil)
                      AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
```

```
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
("___builtin_ctz",
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
("___builtin_ctzl",
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
("___builtin_ctzll",
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
("___builtin_fabs",
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
("___builtin_fabsf",
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
    (tfloat :: nil) tfloat cc_default)) ::
("___builtin_fsqrt",
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
("___builtin_sqrt",
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
("___builtin_memcpy_aligned",
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
("___builtin_sel",
  Gfun(External (EF_builtin "__builtin_sel"
                    (mksignature (AST.Xbool :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
```

```
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                      cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
  Gfun(External (EF_builtin "__builtin_va_arg"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                    (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
```

```
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmadd,
  Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmsub,
  Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmax,
  Gfun(External (EF_builtin "__builtin_fmax"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
    cc_default)) ::
(___builtin_fmin,
```

```
    Gfun(External (EF_builtin "__builtin_fmin"
                          (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                            cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
  (___builtin_debug,
    Gfun(External (EF_external "__builtin_debug"
                          (mksignature (AST.Xint :: nil) AST.Xvoid
                            {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
  (_malloc, Gfun(External EF_malloc (tulong :: nil) (tptr tvoid) cc_default)) ::
  (_free, Gfun(External EF_free ((tptr tvoid) :: nil) tvoid cc_default)) ::
  (_exit,
    Gfun(External (EF_external "exit"
                          (mksignature (AST.Xint :: nil) AST.Xvoid cc_default))
      (tint :: nil) tvoid cc_default)) ::
  (_placeholder, Gfun(Internal f_placeholder)) :: nil).

Definition public_idents : list ident :=
(_placeholder :: _exit :: _free :: _malloc :: ___builtin_debug ::
  ___builtin_fmin :: ___builtin_fmax :: ___builtin_fnmsub ::
  ___builtin_fnmadd :: ___builtin_fmsub :: ___builtin_fmadd ::
  ___builtin_clsll :: ___builtin_clsl :: ___builtin_cls ::
  ___builtin_expect :: ___builtin_unreachable :: ___builtin_va_end ::
  ___builtin_va_copy :: ___builtin_va_arg :: ___builtin_va_start ::
  ___builtin_membar :: ___builtin_annot_intval :: ___builtin_annot ::
  ___builtin_sel :: ___builtin_memcpy_aligned :: ___builtin_sqrt ::
  ___builtin_fsqrt :: ___builtin_fabsf :: ___builtin_fabs ::
  ___builtin_ctzll :: ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll ::
  ___builtin_clzl :: ___builtin_clz :: ___builtin_bswap16 ::
  ___builtin_bswap32 :: ___builtin_bswap :: ___builtin_bswap64 ::
  ___compcert_i64_umulh :: ___compcert_i64_smulh :: ___compcert_i64_sar ::
  ___compcert_i64_shr :: ___compcert_i64_shl :: ___compcert_i64_umod ::
  ___compcert_i64_smod :: ___compcert_i64_udiv :: ___compcert_i64_sdiv ::
  ___compcert_i64_utof :: ___compcert_i64_stof :: ___compcert_i64_utod ::
  ___compcert_i64_stod :: ___compcert_i64_dtou :: ___compcert_i64_dtos ::
  ___compcert_va_composite :: ___compcert_va_float64 ::
  ___compcert_va_int64 :: ___compcert_va_int32 :: nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 9

Library VC.stdlib2

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "stdlib2.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 17%positive.
Definition ___builtin_annot_intval : ident := 18%positive.
Definition ___builtin_bswap : ident := 2%positive.
Definition ___builtin_bswap16 : ident := 4%positive.
Definition ___builtin_bswap32 : ident := 3%positive.
Definition ___builtin_bswap64 : ident := 1%positive.
Definition ___builtin_cls : ident := 26%positive.
Definition ___builtin_clsl : ident := 27%positive.
Definition ___builtin_clsll : ident := 28%positive.
Definition ___builtin_clz : ident := 5%positive.
```

```
Definition ___builtin_clzl : ident := 6%positive.
Definition ___builtin_clzll : ident := 7%positive.
Definition ___builtin_ctz : ident := 8%positive.
Definition ___builtin_ctzl : ident := 9%positive.
Definition ___builtin_ctzll : ident := 10%positive.
Definition ___builtin_debug : ident := 35%positive.
Definition ___builtin_expect : ident := 25%positive.
Definition ___builtin_fabs : ident := 11%positive.
Definition ___builtin_fabsf : ident := 12%positive.
Definition ___builtin_fmadd : ident := 29%positive.
Definition ___builtin_fmax : ident := 33%positive.
Definition ___builtin_fmin : ident := 34%positive.
Definition ___builtin_fmsub : ident := 30%positive.
Definition ___builtin_fnmadd : ident := 31%positive.
Definition ___builtin_fnmsub : ident := 32%positive.
Definition ___builtin_fsqrt : ident := 13%positive.
Definition ___builtin_membar : ident := 19%positive.
Definition ___builtin_memcpy_aligned : ident := 15%positive.
Definition ___builtin_sel : ident := 16%positive.
Definition ___builtin_sqrt : ident := 14%positive.
Definition ___builtin_unreachable : ident := 24%positive.
Definition ___builtin_va_arg : ident := 21%positive.
Definition ___builtin_va_copy : ident := 22%positive.
Definition ___builtin_va_end : ident := 23%positive.
Definition ___builtin_va_start : ident := 20%positive.
Definition ___compcert_i64_dtos : ident := 44%positive.
Definition ___compcert_i64_dtou : ident := 45%positive.
Definition ___compcert_i64_sar : ident := 56%positive.
Definition ___compcert_i64_sdiv : ident := 50%positive.
Definition ___compcert_i64_shl : ident := 54%positive.
Definition ___compcert_i64_shr : ident := 55%positive.
Definition ___compcert_i64_smod : ident := 52%positive.
Definition ___compcert_i64_smulh : ident := 57%positive.
Definition ___compcert_i64_stod : ident := 46%positive.
Definition ___compcert_i64_stof : ident := 48%positive.
Definition ___compcert_i64_udiv : ident := 51%positive.
Definition ___compcert_i64_umod : ident := 53%positive.
Definition ___compcert_i64_umulh : ident := 58%positive.
Definition ___compcert_i64_utod : ident := 47%positive.
Definition ___compcert_i64_utof : ident := 49%positive.
Definition ___compcert_va_composite : ident := 43%positive.
Definition ___compcert_va_float64 : ident := 42%positive.
```

```
Definition ___compcert_va_int32 : ident := 40%positive.
Definition ___compcert_va_int64 : ident := 41%positive.
Definition _a : ident := 61%positive.
Definition _b : ident := 62%positive.
Definition _c : ident := 63%positive.
Definition _cell : ident := 60%positive.
Definition _d : ident := 64%positive.
Definition _exit : ident := 38%positive.
Definition _free : ident := 37%positive.
Definition _freelist : ident := 67%positive.
Definition _main : ident := 59%positive.
Definition _malloc : ident := 36%positive.
Definition _n : ident := 68%positive.
Definition _p : ident := 69%positive.
Definition _placeholder : ident := 39%positive.
Definition _pool : ident := 65%positive.
Definition _pool_index : ident := 66%positive.
Definition _pp : ident := 70%positive.
Definition _t'1 : ident := 71%positive.
Definition _t'2 : ident := 72%positive.
Definition _t'3 : ident := 73%positive.
Definition _t'4 : ident := 74%positive.

Definition v_pool := {|
  gvar_info := (tarray (Tstruct _cell noattr) 80000);
  gvar_init := (Init_space 2560000 :: nil);
  gvar_readonly := false;
  gvar_volatile := false
|}.

Definition v_pool_index := {|
  gvar_info := tint;
  gvar_init := (Init_int32 (Int.repr 0) :: nil);
  gvar_readonly := false;
  gvar_volatile := false
|}.

Definition v_freelist := {|
  gvar_info := (tptr (Tstruct _cell noattr));
  gvar_init := (Init_int64 (Int64.repr 0) :: nil);
  gvar_readonly := false;
  gvar_volatile := false
|}.

Definition f_malloc := {|
```

```
  fn_return := (tptr tvoid);
  fn_callconv := cc_default;
  fn_params := ((_n, tulong) :: nil);
  fn_vars := nil;
  fn_temps := ((_p, (tptr (Tstruct _cell noattr))) :: (_t'1, tint) ::
                (_t'4, (tptr (Tstruct _cell noattr))) :: (_t'3, tint) ::
                (_t'2, (tptr (Tstruct _cell noattr))) :: nil);
  fn_body :=
(Ssequence
  (Sifthenelse (Ebinop Ogt (Etempvar _n tulong)
                 (Esizeof (Tstruct _cell noattr) tulong) tint)
    (Sreturn (Some (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid))))
    Sskip)
  (Ssequence
    (Ssequence
      (Sset _t'2 (Evar _freelist (tptr (Tstruct _cell noattr))))
      (Sifthenelse (Etempvar _t'2 (tptr (Tstruct _cell noattr)))
        (Ssequence
          (Sset _p (Evar _freelist (tptr (Tstruct _cell noattr))))
          (Ssequence
            (Sset _t'4
              (Efield
                (Ederef (Etempvar _p (tptr (Tstruct _cell noattr)))
                  (Tstruct _cell noattr)) _a (tptr (Tstruct _cell noattr))))
            (Sassign (Evar _freelist (tptr (Tstruct _cell noattr)))
              (Etempvar _t'4 (tptr (Tstruct _cell noattr))))))
        (Ssequence
          (Sset _t'3 (Evar _pool_index tint))
          (Sifthenelse (Ebinop Olt (Etempvar _t'3 tint)
                         (Econst_int (Int.repr 80000) tint) tint)
            (Ssequence
              (Ssequence
                (Sset _t'1 (Evar _pool_index tint))
                (Sassign (Evar _pool_index tint)
                  (Ebinop Oadd (Etempvar _t'1 tint)
                    (Econst_int (Int.repr 1) tint) tint)))
              (Sset _p
                (Ebinop Oadd
                  (Evar _pool (tarray (Tstruct _cell noattr) 80000))
                  (Etempvar _t'1 tint) (tptr (Tstruct _cell noattr)))))
            (Sset _p (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid)))))))
    (Sreturn (Some (Ecast (Etempvar _p (tptr (Tstruct _cell noattr)))
```

```
                        (tptr tvoid))))))
|}.
Definition f_free := {|
  fn_return := tvoid;
  fn_callconv := cc_default;
  fn_params := ((_p, (tptr tvoid)) :: nil);
  fn_vars := nil;
  fn_temps := ((_pp, (tptr (Tstruct _cell noattr))) ::
               (_t'1, (tptr (Tstruct _cell noattr))) :: nil);
  fn_body :=
(Ssequence
  (Sset _pp (Etempvar _p (tptr tvoid)))
  (Ssequence
    (Sifthenelse (Ebinop Oeq (Etempvar _pp (tptr (Tstruct _cell noattr)))
                             (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid)) tint)
      (Sreturn None)
      Sskip)
    (Ssequence
      (Ssequence
        (Sset _t'1 (Evar _freelist (tptr (Tstruct _cell noattr))))
        (Sassign
          (Efield
            (Ederef (Etempvar _pp (tptr (Tstruct _cell noattr)))
              (Tstruct _cell noattr)) _a (tptr (Tstruct _cell noattr)))
          (Etempvar _t'1 (tptr (Tstruct _cell noattr)))))
      (Sassign (Evar _freelist (tptr (Tstruct _cell noattr)))
        (Etempvar _pp (tptr (Tstruct _cell noattr)))))))
|}.
Definition f_exit := {|
  fn_return := tvoid;
  fn_callconv := cc_default;
  fn_params := ((_n, tint) :: nil);
  fn_vars := nil;
  fn_temps := nil;
  fn_body :=
(Sloop Sskip Sskip)
|}.
Definition composites : list composite_definition :=
(Composite _cell Struct
    (Member_plain _a (tptr (Tstruct _cell noattr)) ::
     Member_plain _b (tptr (Tstruct _cell noattr)) ::
     Member_plain _c (tptr (Tstruct _cell noattr)) ::
```

```
    Member_plain _d (tptr (Tstruct _cell noattr)) :: nil)
  noattr :: nil).

Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
   Gfun(External (EF_runtime "__compcert_va_int32"
                     (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
     ((tptr tvoid) :: nil) tuint cc_default)) ::
 (___compcert_va_int64,
   Gfun(External (EF_runtime "__compcert_va_int64"
                     (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
     ((tptr tvoid) :: nil) tulong cc_default)) ::
 (___compcert_va_float64,
   Gfun(External (EF_runtime "__compcert_va_float64"
                     (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
     ((tptr tvoid) :: nil) tdouble cc_default)) ::
 (___compcert_va_composite,
   Gfun(External (EF_runtime "__compcert_va_composite"
                     (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                       cc_default)) ((tptr tvoid) :: tulong :: nil)
     (tptr tvoid) cc_default)) ::
 (___compcert_i64_dtos,
   Gfun(External (EF_runtime "__compcert_i64_dtos"
                     (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
     (tdouble :: nil) tlong cc_default)) ::
 (___compcert_i64_dtou,
   Gfun(External (EF_runtime "__compcert_i64_dtou"
                     (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
     (tdouble :: nil) tulong cc_default)) ::
 (___compcert_i64_stod,
   Gfun(External (EF_runtime "__compcert_i64_stod"
                     (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
     (tlong :: nil) tdouble cc_default)) ::
 (___compcert_i64_utod,
   Gfun(External (EF_runtime "__compcert_i64_utod"
                     (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
     (tulong :: nil) tdouble cc_default)) ::
 (___compcert_i64_stof,
   Gfun(External (EF_runtime "__compcert_i64_stof"
                     (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
     (tlong :: nil) tfloat cc_default)) ::
 (___compcert_i64_utof,
   Gfun(External (EF_runtime "__compcert_i64_utof"
```

```
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
```

```
  Gfun(External (EF_builtin "__builtin_bswap64"
                    (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                    (mksignature (AST.Xint16unsigned :: nil)
                      AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctz,
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_ctzl,
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctzll,
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_fabs,
  Gfun(External (EF_builtin "__builtin_fabs"
```

```
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
("___builtin_fabsf",
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
    (tfloat :: nil) tfloat cc_default)) ::
("___builtin_fsqrt",
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
("___builtin_sqrt",
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
("___builtin_memcpy_aligned",
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
("___builtin_sel",
  Gfun(External (EF_builtin "__builtin_sel"
                    (mksignature (AST.Xbool :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
("___builtin_annot",
  Gfun(External (EF_builtin "__builtin_annot"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
("___builtin_annot_intval",
  Gfun(External (EF_builtin "__builtin_annot_intval"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                      cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
("___builtin_membar",
  Gfun(External (EF_builtin "__builtin_membar"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
```

```
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
  Gfun(External (EF_builtin "__builtin_va_arg"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                    (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
```

```
                    AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
  (___builtin_fmsub,
    Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
  (___builtin_fnmadd,
    Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
  (___builtin_fnmsub,
    Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
  (___builtin_fmax,
    Gfun(External (EF_builtin "__builtin_fmax"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
  (___builtin_fmin,
    Gfun(External (EF_builtin "__builtin_fmin"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
  (___builtin_debug,
    Gfun(External (EF_external "__builtin_debug"
                    (mksignature (AST.Xint :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
  (_pool, Gvar v_pool) :: (_pool_index, Gvar v_pool_index) ::
  (_freelist, Gvar v_freelist) :: (_malloc, Gfun(Internal f_malloc)) ::
  (_free, Gfun(Internal f_free)) :: (_exit, Gfun(Internal f_exit)) :: nil).

Definition public_idents : list ident :=
(_exit :: _free :: _malloc :: _freelist :: _pool_index :: _pool ::
  ___builtin_debug :: ___builtin_fmin :: ___builtin_fmax ::
```

```
  ___builtin_fnmsub :: ___builtin_fnmadd :: ___builtin_fmsub ::
  ___builtin_fmadd :: ___builtin_clsll :: ___builtin_clsl :: ___builtin_cls ::
  ___builtin_expect :: ___builtin_unreachable :: ___builtin_va_end ::
  ___builtin_va_copy :: ___builtin_va_arg :: ___builtin_va_start ::
  ___builtin_membar :: ___builtin_annot_intval :: ___builtin_annot ::
  ___builtin_sel :: ___builtin_memcpy_aligned :: ___builtin_sqrt ::
  ___builtin_fsqrt :: ___builtin_fabsf :: ___builtin_fabs ::
  ___builtin_ctzll :: ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll ::
  ___builtin_clzl :: ___builtin_clz :: ___builtin_bswap16 ::
  ___builtin_bswap32 :: ___builtin_bswap :: ___builtin_bswap64 ::
  ___compcert_i64_umulh :: ___compcert_i64_smulh :: ___compcert_i64_sar ::
  ___compcert_i64_shr :: ___compcert_i64_shl :: ___compcert_i64_umod ::
  ___compcert_i64_smod :: ___compcert_i64_udiv :: ___compcert_i64_sdiv ::
  ___compcert_i64_utof :: ___compcert_i64_stof :: ___compcert_i64_utod ::
  ___compcert_i64_stod :: ___compcert_i64_dtou :: ___compcert_i64_dtos ::
  ___compcert_va_composite :: ___compcert_va_float64 ::
  ___compcert_va_int64 :: ___compcert_va_int32 :: nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 10

Library VC.stack2

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "stack2.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 17%positive.
Definition ___builtin_annot_intval : ident := 18%positive.
Definition ___builtin_bswap : ident := 2%positive.
Definition ___builtin_bswap16 : ident := 4%positive.
Definition ___builtin_bswap32 : ident := 3%positive.
Definition ___builtin_bswap64 : ident := 1%positive.
Definition ___builtin_cls : ident := 26%positive.
Definition ___builtin_clsl : ident := 27%positive.
Definition ___builtin_clsll : ident := 28%positive.
Definition ___builtin_clz : ident := 5%positive.
```

```
Definition ___builtin_clzl : ident := 6%positive.
Definition ___builtin_clzll : ident := 7%positive.
Definition ___builtin_ctz : ident := 8%positive.
Definition ___builtin_ctzl : ident := 9%positive.
Definition ___builtin_ctzll : ident := 10%positive.
Definition ___builtin_debug : ident := 35%positive.
Definition ___builtin_expect : ident := 25%positive.
Definition ___builtin_fabs : ident := 11%positive.
Definition ___builtin_fabsf : ident := 12%positive.
Definition ___builtin_fmadd : ident := 29%positive.
Definition ___builtin_fmax : ident := 33%positive.
Definition ___builtin_fmin : ident := 34%positive.
Definition ___builtin_fmsub : ident := 30%positive.
Definition ___builtin_fnmadd : ident := 31%positive.
Definition ___builtin_fnmsub : ident := 32%positive.
Definition ___builtin_fsqrt : ident := 13%positive.
Definition ___builtin_membar : ident := 19%positive.
Definition ___builtin_memcpy_aligned : ident := 15%positive.
Definition ___builtin_sel : ident := 16%positive.
Definition ___builtin_sqrt : ident := 14%positive.
Definition ___builtin_unreachable : ident := 24%positive.
Definition ___builtin_va_arg : ident := 21%positive.
Definition ___builtin_va_copy : ident := 22%positive.
Definition ___builtin_va_end : ident := 23%positive.
Definition ___builtin_va_start : ident := 20%positive.
Definition ___compcert_i64_dtos : ident := 44%positive.
Definition ___compcert_i64_dtou : ident := 45%positive.
Definition ___compcert_i64_sar : ident := 56%positive.
Definition ___compcert_i64_sdiv : ident := 50%positive.
Definition ___compcert_i64_shl : ident := 54%positive.
Definition ___compcert_i64_shr : ident := 55%positive.
Definition ___compcert_i64_smod : ident := 52%positive.
Definition ___compcert_i64_smulh : ident := 57%positive.
Definition ___compcert_i64_stod : ident := 46%positive.
Definition ___compcert_i64_stof : ident := 48%positive.
Definition ___compcert_i64_udiv : ident := 51%positive.
Definition ___compcert_i64_umod : ident := 53%positive.
Definition ___compcert_i64_umulh : ident := 58%positive.
Definition ___compcert_i64_utod : ident := 47%positive.
Definition ___compcert_i64_utof : ident := 49%positive.
Definition ___compcert_va_composite : ident := 43%positive.
Definition ___compcert_va_float64 : ident := 42%positive.
```

```
Definition ___compcert_va_int32 : ident := 40%positive.
Definition ___compcert_va_int64 : ident := 41%positive.
Definition _a : ident := 61%positive.
Definition _b : ident := 62%positive.
Definition _c : ident := 63%positive.
Definition _cell : ident := 60%positive.
Definition _cons : ident := 71%positive.
Definition _d : ident := 64%positive.
Definition _exit : ident := 38%positive.
Definition _free : ident := 37%positive.
Definition _freelist : ident := 67%positive.
Definition _i : ident := 78%positive.
Definition _main : ident := 59%positive.
Definition _malloc : ident := 36%positive.
Definition _n : ident := 68%positive.
Definition _newstack : ident := 77%positive.
Definition _next : ident := 73%positive.
Definition _p : ident := 69%positive.
Definition _placeholder : ident := 39%positive.
Definition _pool : ident := 65%positive.
Definition _pool_index : ident := 66%positive.
Definition _pop : ident := 81%positive.
Definition _pp : ident := 70%positive.
Definition _push : ident := 80%positive.
Definition _q : ident := 79%positive.
Definition _stack : ident := 74%positive.
Definition _surely_malloc : ident := 76%positive.
Definition _top : ident := 75%positive.
Definition _value : ident := 72%positive.
Definition _t'1 : ident := 82%positive.
Definition _t'2 : ident := 83%positive.

Definition f_surely_malloc := {|
  fn_return := (tptr tvoid);
  fn_callconv := cc_default;
  fn_params := ((_n, tulong) :: nil);
  fn_vars := nil;
  fn_temps := ((_p, (tptr tvoid)) :: (_t'1, (tptr tvoid)) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _malloc (Tfunction (tulong :: nil) (tptr tvoid) cc_default))
```

```
      ((Etempvar _n tulong) :: nil))
    (Sset _p (Etempvar _t'1 (tptr tvoid))))
  (Ssequence
    (Sifthenelse (Eunop Onotbool (Etempvar _p (tptr tvoid)) tint)
      (Scall None (Evar _exit (Tfunction (tint :: nil) tvoid cc_default))
        ((Econst_int (Int.repr 1) tint) :: nil))
      Sskip)
    (Sreturn (Some (Etempvar _p (tptr tvoid))))))
|}.

Definition f_newstack := {|
  fn_return := (tptr (Tstruct _stack noattr));
  fn_callconv := cc_default;
  fn_params := nil;
  fn_vars := nil;
  fn_temps := ((_p, (tptr (Tstruct _stack noattr))) ::
               (_t'1, (tptr tvoid)) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _surely_malloc (Tfunction (tulong :: nil) (tptr tvoid)
                                      cc_default))
      ((Esizeof (Tstruct _stack noattr) tulong) :: nil))
    (Sset _p
      (Ecast (Etempvar _t'1 (tptr tvoid)) (tptr (Tstruct _stack noattr)))))
  (Ssequence
    (Sassign
      (Efield
        (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
          (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr)))
      (Ecast (Econst_int (Int.repr 0) tint) (tptr tvoid)))
    (Sreturn (Some (Etempvar _p (tptr (Tstruct _stack noattr)))))))
|}.

Definition f_push := {|
  fn_return := tvoid;
  fn_callconv := cc_default;
  fn_params := ((_p, (tptr (Tstruct _stack noattr))) :: (_i, tint) :: nil);
  fn_vars := nil;
  fn_temps := ((_q, (tptr (Tstruct _cons noattr))) :: (_t'1, (tptr tvoid)) ::
               (_t'2, (tptr (Tstruct _cons noattr))) :: nil);
  fn_body :=
(Ssequence
```

```
  (Ssequence
    (Scall (Some _t'1)
      (Evar _surely_malloc (Tfunction (tulong :: nil) (tptr tvoid)
                                      cc_default))
      ((Esizeof (Tstruct _cons noattr) tulong) :: nil))
    (Sset _q
      (Ecast (Etempvar _t'1 (tptr tvoid)) (tptr (Tstruct _cons noattr)))))
  (Ssequence
    (Sassign
      (Efield
        (Ederef (Etempvar _q (tptr (Tstruct _cons noattr)))
          (Tstruct _cons noattr)) _value tint) (Etempvar _i tint))
    (Ssequence
      (Ssequence
        (Sset _t'2
          (Efield
            (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
              (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr))))
        (Sassign
          (Efield
            (Ederef (Etempvar _q (tptr (Tstruct _cons noattr)))
              (Tstruct _cons noattr)) _next (tptr (Tstruct _cons noattr)))
          (Etempvar _t'2 (tptr (Tstruct _cons noattr)))))
      (Sassign
        (Efield
          (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
            (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr)))
        (Etempvar _q (tptr (Tstruct _cons noattr)))))))
|}.

Definition f_pop := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := ((_p, (tptr (Tstruct _stack noattr))) :: nil);
  fn_vars := nil;
  fn_temps := ((_q, (tptr (Tstruct _cons noattr))) :: (_i, tint) ::
                (_t'1, (tptr (Tstruct _cons noattr))) :: nil);
  fn_body :=
(Ssequence
  (Sset _q
    (Efield
      (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
        (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr))))
```

```
  (Ssequence
    (Ssequence
      (Sset _t'1
        (Efield
          (Ederef (Etempvar _q (tptr (Tstruct _cons noattr)))
            (Tstruct _cons noattr)) _next (tptr (Tstruct _cons noattr))))
      (Sassign
        (Efield
          (Ederef (Etempvar _p (tptr (Tstruct _stack noattr)))
            (Tstruct _stack noattr)) _top (tptr (Tstruct _cons noattr)))
        (Etempvar _t'1 (tptr (Tstruct _cons noattr)))))
    (Ssequence
      (Sset _i
        (Efield
          (Ederef (Etempvar _q (tptr (Tstruct _cons noattr)))
            (Tstruct _cons noattr)) _value tint))
      (Ssequence
        (Scall None
          (Evar _free (Tfunction ((tptr tvoid) :: nil) tvoid cc_default))
          ((Etempvar _q (tptr (Tstruct _cons noattr))) :: nil))
        (Sreturn (Some (Etempvar _i tint)))))))
|}.

Definition composites : list composite_definition :=
(Composite _cons Struct
    (Member_plain _value tint ::
     Member_plain _next (tptr (Tstruct _cons noattr)) :: nil)
    noattr ::
 Composite _stack Struct
    (Member_plain _top (tptr (Tstruct _cons noattr)) :: nil)
    noattr :: nil).

Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
    Gfun(External (EF_runtime "__compcert_va_int32"
                      (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
      ((tptr tvoid) :: nil) tuint cc_default)) ::
 (___compcert_va_int64,
    Gfun(External (EF_runtime "__compcert_va_int64"
                      (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
      ((tptr tvoid) :: nil) tulong cc_default)) ::
 (___compcert_va_float64,
    Gfun(External (EF_runtime "__compcert_va_float64"
                      (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
```

```
    ((tptr tvoid) :: nil) tdouble cc_default)) ::
(___compcert_va_composite,
  Gfun(External (EF_runtime "__compcert_va_composite"
                    (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                      cc_default)) ((tptr tvoid) :: tulong :: nil)
    (tptr tvoid) cc_default)) ::
(___compcert_i64_dtos,
  Gfun(External (EF_runtime "__compcert_i64_dtos"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tlong cc_default)) ::
(___compcert_i64_dtou,
  Gfun(External (EF_runtime "__compcert_i64_dtou"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tulong cc_default)) ::
(___compcert_i64_stod,
  Gfun(External (EF_runtime "__compcert_i64_stod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tlong :: nil) tdouble cc_default)) ::
(___compcert_i64_utod,
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
(___compcert_i64_stof,
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
(___compcert_i64_utof,
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
```

```
                    cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                    (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                    (mksignature (AST.Xint16unsigned :: nil)
```

```
                    AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctz,
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_ctzl,
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctzll,
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_fabs,
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_fabsf,
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
    (tfloat :: nil) tfloat cc_default)) ::
(___builtin_fsqrt,
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_sqrt,
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
```

```
(___builtin_memcpy_aligned,
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                  (mksignature
                    (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                    AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
(___builtin_sel,
  Gfun(External (EF_builtin "__builtin_sel"
                  (mksignature (AST.Xbool :: nil) AST.Xvoid
                    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
                  (mksignature (AST.Xptr :: nil) AST.Xvoid
                    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                  (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                    cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                  (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                  (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
  Gfun(External (EF_builtin "__builtin_va_arg"
                  (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                    cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                  (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                    cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
```

```
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmadd,
  Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
```

```
  (___builtin_fnmsub,
    Gfun(External (EF_builtin "__builtin_fnmsub"
                          (mksignature
                            (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                            AST.Xfloat cc_default))
      (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
  (___builtin_fmax,
    Gfun(External (EF_builtin "__builtin_fmax"
                          (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                            cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
  (___builtin_fmin,
    Gfun(External (EF_builtin "__builtin_fmin"
                          (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                            cc_default)) (tdouble :: tdouble :: nil) tdouble
      cc_default)) ::
  (___builtin_debug,
    Gfun(External (EF_external "__builtin_debug"
                          (mksignature (AST.Xint :: nil) AST.Xvoid
                            {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
  (_malloc, Gfun(External EF_malloc (tulong :: nil) (tptr tvoid) cc_default)) ::
  (_free, Gfun(External EF_free ((tptr tvoid) :: nil) tvoid cc_default)) ::
  (_exit,
    Gfun(External (EF_external "exit"
                          (mksignature (AST.Xint :: nil) AST.Xvoid cc_default))
      (tint :: nil) tvoid cc_default)) ::
  (_surely_malloc, Gfun(Internal f_surely_malloc)) ::
  (_newstack, Gfun(Internal f_newstack)) :: (_push, Gfun(Internal f_push)) ::
  (_pop, Gfun(Internal f_pop)) :: nil).

Definition public_idents : list ident :=
(_pop :: _push :: _newstack :: _surely_malloc :: _exit :: _free :: _malloc ::
  ___builtin_debug :: ___builtin_fmin :: ___builtin_fmax ::
  ___builtin_fnmsub :: ___builtin_fnmadd :: ___builtin_fmsub ::
  ___builtin_fmadd :: ___builtin_clsll :: ___builtin_clsl :: ___builtin_cls ::
  ___builtin_expect :: ___builtin_unreachable :: ___builtin_va_end ::
  ___builtin_va_copy :: ___builtin_va_arg :: ___builtin_va_start ::
  ___builtin_membar :: ___builtin_annot_intval :: ___builtin_annot ::
  ___builtin_sel :: ___builtin_memcpy_aligned :: ___builtin_sqrt ::
  ___builtin_fsqrt :: ___builtin_fabsf :: ___builtin_fabs ::
  ___builtin_ctzll :: ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll ::
```

```
  ___builtin_clzl :: ___builtin_clz :: ___builtin_bswap16 ::
  ___builtin_bswap32 :: ___builtin_bswap :: ___builtin_bswap64 ::
  ___compcert_i64_umulh :: ___compcert_i64_smulh :: ___compcert_i64_sar ::
  ___compcert_i64_shr :: ___compcert_i64_shl :: ___compcert_i64_umod ::
  ___compcert_i64_smod :: ___compcert_i64_udiv :: ___compcert_i64_sdiv ::
  ___compcert_i64_utof :: ___compcert_i64_stof :: ___compcert_i64_utod ::
  ___compcert_i64_stod :: ___compcert_i64_dtou :: ___compcert_i64_dtos ::
  ___compcert_va_composite :: ___compcert_va_float64 ::
  ___compcert_va_int64 :: ___compcert_va_int32 :: nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 11

Library VC.triang2

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "triang2.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 17%positive.
Definition ___builtin_annot_intval : ident := 18%positive.
Definition ___builtin_bswap : ident := 2%positive.
Definition ___builtin_bswap16 : ident := 4%positive.
Definition ___builtin_bswap32 : ident := 3%positive.
Definition ___builtin_bswap64 : ident := 1%positive.
Definition ___builtin_cls : ident := 26%positive.
Definition ___builtin_clsl : ident := 27%positive.
Definition ___builtin_clsll : ident := 28%positive.
Definition ___builtin_clz : ident := 5%positive.
```

```
Definition ___builtin_clzl : ident := 6%positive.
Definition ___builtin_clzll : ident := 7%positive.
Definition ___builtin_ctz : ident := 8%positive.
Definition ___builtin_ctzl : ident := 9%positive.
Definition ___builtin_ctzll : ident := 10%positive.
Definition ___builtin_debug : ident := 35%positive.
Definition ___builtin_expect : ident := 25%positive.
Definition ___builtin_fabs : ident := 11%positive.
Definition ___builtin_fabsf : ident := 12%positive.
Definition ___builtin_fmadd : ident := 29%positive.
Definition ___builtin_fmax : ident := 33%positive.
Definition ___builtin_fmin : ident := 34%positive.
Definition ___builtin_fmsub : ident := 30%positive.
Definition ___builtin_fnmadd : ident := 31%positive.
Definition ___builtin_fnmsub : ident := 32%positive.
Definition ___builtin_fsqrt : ident := 13%positive.
Definition ___builtin_membar : ident := 19%positive.
Definition ___builtin_memcpy_aligned : ident := 15%positive.
Definition ___builtin_sel : ident := 16%positive.
Definition ___builtin_sqrt : ident := 14%positive.
Definition ___builtin_unreachable : ident := 24%positive.
Definition ___builtin_va_arg : ident := 21%positive.
Definition ___builtin_va_copy : ident := 22%positive.
Definition ___builtin_va_end : ident := 23%positive.
Definition ___builtin_va_start : ident := 20%positive.
Definition ___compcert_i64_dtos : ident := 44%positive.
Definition ___compcert_i64_dtou : ident := 45%positive.
Definition ___compcert_i64_sar : ident := 56%positive.
Definition ___compcert_i64_sdiv : ident := 50%positive.
Definition ___compcert_i64_shl : ident := 54%positive.
Definition ___compcert_i64_shr : ident := 55%positive.
Definition ___compcert_i64_smod : ident := 52%positive.
Definition ___compcert_i64_smulh : ident := 57%positive.
Definition ___compcert_i64_stod : ident := 46%positive.
Definition ___compcert_i64_stof : ident := 48%positive.
Definition ___compcert_i64_udiv : ident := 51%positive.
Definition ___compcert_i64_umod : ident := 53%positive.
Definition ___compcert_i64_umulh : ident := 58%positive.
Definition ___compcert_i64_utod : ident := 47%positive.
Definition ___compcert_i64_utof : ident := 49%positive.
Definition ___compcert_va_composite : ident := 43%positive.
Definition ___compcert_va_float64 : ident := 42%positive.
```

```
Definition ___compcert_va_int32 : ident := 40%positive.
Definition ___compcert_va_int64 : ident := 41%positive.
Definition _a : ident := 61%positive.
Definition _b : ident := 62%positive.
Definition _c : ident := 63%positive.
Definition _cell : ident := 60%positive.
Definition _cons : ident := 71%positive.
Definition _d : ident := 64%positive.
Definition _exit : ident := 38%positive.
Definition _free : ident := 37%positive.
Definition _freelist : ident := 67%positive.
Definition _i : ident := 78%positive.
Definition _main : ident := 59%positive.
Definition _malloc : ident := 36%positive.
Definition _n : ident := 68%positive.
Definition _newstack : ident := 77%positive.
Definition _next : ident := 73%positive.
Definition _p : ident := 69%positive.
Definition _placeholder : ident := 39%positive.
Definition _pool : ident := 65%positive.
Definition _pool_index : ident := 66%positive.
Definition _pop : ident := 81%positive.
Definition _pop_and_add : ident := 86%positive.
Definition _pp : ident := 70%positive.
Definition _push : ident := 80%positive.
Definition _push_increasing : ident := 83%positive.
Definition _q : ident := 79%positive.
Definition _s : ident := 85%positive.
Definition _st : ident := 82%positive.
Definition _stack : ident := 74%positive.
Definition _surely_malloc : ident := 76%positive.
Definition _t : ident := 84%positive.
Definition _top : ident := 75%positive.
Definition _triang : ident := 87%positive.
Definition _value : ident := 72%positive.
Definition _t'1 : ident := 88%positive.
Definition _t'2 : ident := 89%positive.

Definition f_push_increasing := {|
  fn_return := tvoid;
  fn_callconv := cc_default;
  fn_params := ((_st, (tptr (Tstruct _stack noattr))) :: (_n, tint) :: nil);
  fn_vars := nil;
```

```
  fn_temps := ((_i, tint) :: nil);
  fn_body :=
(Ssequence
  (Sset _i (Econst_int (Int.repr 0) tint))
  (Swhile
    (Ebinop Olt (Etempvar _i tint) (Etempvar _n tint) tint)
    (Ssequence
      (Sset _i
        (Ebinop Oadd (Etempvar _i tint) (Econst_int (Int.repr 1) tint) tint))
      (Scall None
        (Evar _push (Tfunction
                      ((tptr (Tstruct _stack noattr)) :: tint :: nil) tvoid
                      cc_default))
        ((Etempvar _st (tptr (Tstruct _stack noattr))) ::
         (Etempvar _i tint) :: nil)))))
|}.
Definition f_pop_and_add := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := ((_st, (tptr (Tstruct _stack noattr))) :: (_n, tint) :: nil);
  fn_vars := nil;
  fn_temps := ((_i, tint) :: (_t, tint) :: (_s, tint) :: (_t'1, tint) :: nil);
  fn_body :=
(Ssequence
  (Sset _i (Econst_int (Int.repr 0) tint))
  (Ssequence
    (Sset _s (Econst_int (Int.repr 0) tint))
    (Ssequence
      (Swhile
        (Ebinop Olt (Etempvar _i tint) (Etempvar _n tint) tint)
        (Ssequence
          (Ssequence
            (Scall (Some _t'1)
              (Evar _pop (Tfunction ((tptr (Tstruct _stack noattr)) :: nil)
                           tint cc_default))
              ((Etempvar _st (tptr (Tstruct _stack noattr))) :: nil))
            (Sset _t (Etempvar _t'1 tint)))
          (Ssequence
            (Sset _s
              (Ebinop Oadd (Etempvar _s tint) (Etempvar _t tint) tint))
            (Sset _i
              (Ebinop Oadd (Etempvar _i tint) (Econst_int (Int.repr 1) tint)
```

```
                    tint)))))
        (Sreturn (Some (Etempvar _s tint))))))
|}.

Definition f_triang := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := ((_n, tint) :: nil);
  fn_vars := nil;
  fn_temps := ((_st, (tptr (Tstruct _stack noattr))) :: (_i, tint) ::
                (_t, tint) :: (_s, tint) :: (_t'2, tint) ::
                (_t'1, (tptr (Tstruct _stack noattr))) :: nil);
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _newstack (Tfunction nil (tptr (Tstruct _stack noattr))
                                 cc_default)) nil)
    (Sset _st (Etempvar _t'1 (tptr (Tstruct _stack noattr)))))
  (Ssequence
    (Scall None
      (Evar _push_increasing (Tfunction
                                ((tptr (Tstruct _stack noattr)) :: tint ::
                                 nil) tvoid cc_default))
      ((Etempvar _st (tptr (Tstruct _stack noattr))) :: (Etempvar _n tint) ::
       nil))
    (Ssequence
      (Ssequence
        (Scall (Some _t'2)
          (Evar _pop_and_add (Tfunction
                                ((tptr (Tstruct _stack noattr)) :: tint ::
                                 nil) tint cc_default))
          ((Etempvar _st (tptr (Tstruct _stack noattr))) ::
           (Etempvar _n tint) :: nil))
        (Sset _s (Etempvar _t'2 tint)))
      (Sreturn (Some (Etempvar _s tint))))))
|}.

Definition composites : list composite_definition :=
nil.

Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
   Gfun(External (EF_runtime "__compcert_va_int32"
                     (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
```

```
    ((tptr tvoid) :: nil) tuint cc_default)) ::
(___compcert_va_int64,
  Gfun(External (EF_runtime "__compcert_va_int64"
                    (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
    ((tptr tvoid) :: nil) tulong cc_default)) ::
(___compcert_va_float64,
  Gfun(External (EF_runtime "__compcert_va_float64"
                    (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
    ((tptr tvoid) :: nil) tdouble cc_default)) ::
(___compcert_va_composite,
  Gfun(External (EF_runtime "__compcert_va_composite"
                    (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                      cc_default)) ((tptr tvoid) :: tulong :: nil)
    (tptr tvoid) cc_default)) ::
(___compcert_i64_dtos,
  Gfun(External (EF_runtime "__compcert_i64_dtos"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tlong cc_default)) ::
(___compcert_i64_dtou,
  Gfun(External (EF_runtime "__compcert_i64_dtou"
                    (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
    (tdouble :: nil) tulong cc_default)) ::
(___compcert_i64_stod,
  Gfun(External (EF_runtime "__compcert_i64_stod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tlong :: nil) tdouble cc_default)) ::
(___compcert_i64_utod,
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
(___compcert_i64_stof,
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
(___compcert_i64_utof,
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
```

```
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                    (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
```

```
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                    (mksignature (AST.Xint16unsigned :: nil)
                      AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
  Gfun(External (EF_builtin "__builtin_clzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctz,
  Gfun(External (EF_builtin "__builtin_ctz"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_ctzl,
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctzll,
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_fabs,
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_fabsf,
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
    (tfloat :: nil) tfloat cc_default)) ::
```

```
(___builtin_fsqrt,
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_sqrt,
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_memcpy_aligned,
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
(___builtin_sel,
  Gfun(External (EF_builtin "__builtin_sel"
                    (mksignature (AST.Xbool :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                      cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
  Gfun(External (EF_builtin "__builtin_va_arg"
```

```
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                    (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
```

```
                    AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmadd,
  Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmsub,
  Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmax,
  Gfun(External (EF_builtin "__builtin_fmax"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
    cc_default)) ::
(___builtin_fmin,
  Gfun(External (EF_builtin "__builtin_fmin"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
    cc_default)) ::
(___builtin_debug,
  Gfun(External (EF_external "__builtin_debug"
                    (mksignature (AST.Xint :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tint :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(_newstack,
  Gfun(External (EF_external "newstack"
                    (mksignature nil AST.Xptr cc_default)) nil
    (tptr (Tstruct _stack noattr)) cc_default)) ::
(_push,
  Gfun(External (EF_external "push"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default))
    ((tptr (Tstruct _stack noattr)) :: tint :: nil) tvoid cc_default)) ::
(_pop,
  Gfun(External (EF_external "pop"
                    (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
```

```
      ((tptr (Tstruct _stack noattr)) :: nil) tint cc_default)) ::
  (_push_increasing, Gfun(Internal f_push_increasing)) ::
  (_pop_and_add, Gfun(Internal f_pop_and_add)) ::
  (_triang, Gfun(Internal f_triang)) :: nil).

Definition public_idents : list ident :=
(_triang :: _pop_and_add :: _push_increasing :: _pop :: _push :: _newstack ::
  ___builtin_debug :: ___builtin_fmin :: ___builtin_fmax ::
  ___builtin_fnmsub :: ___builtin_fnmadd :: ___builtin_fmsub ::
  ___builtin_fmadd :: ___builtin_clsll :: ___builtin_clsl :: ___builtin_cls ::
  ___builtin_expect :: ___builtin_unreachable :: ___builtin_va_end ::
  ___builtin_va_copy :: ___builtin_va_arg :: ___builtin_va_start ::
  ___builtin_membar :: ___builtin_annot_intval :: ___builtin_annot ::
  ___builtin_sel :: ___builtin_memcpy_aligned :: ___builtin_sqrt ::
  ___builtin_fsqrt :: ___builtin_fabsf :: ___builtin_fabs ::
  ___builtin_ctzll :: ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll ::
  ___builtin_clzl :: ___builtin_clz :: ___builtin_bswap16 ::
  ___builtin_bswap32 :: ___builtin_bswap :: ___builtin_bswap64 ::
  ___compcert_i64_umulh :: ___compcert_i64_smulh :: ___compcert_i64_sar ::
  ___compcert_i64_shr :: ___compcert_i64_shl :: ___compcert_i64_umod ::
  ___compcert_i64_smod :: ___compcert_i64_udiv :: ___compcert_i64_sdiv ::
  ___compcert_i64_utof :: ___compcert_i64_stof :: ___compcert_i64_utod ::
  ___compcert_i64_stod :: ___compcert_i64_dtou :: ___compcert_i64_dtos ::
  ___compcert_va_composite :: ___compcert_va_float64 ::
  ___compcert_va_int64 :: ___compcert_va_int32 :: nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 12

Library VC.main2

```
From Stdlib Require Import String List ZArith.
From compcert Require Import Coqlib Integers Floats AST Ctypes Cop Clight Clightdefs.
Import Clightdefs.ClightNotations.
Local Open Scope Z_scope.
Local Open Scope string_scope.
Local Open Scope clight_scope.

Module Info.
  Definition version := "3.16".
  Definition build_number := "".
  Definition build_tag := "".
  Definition build_branch := "".
  Definition arch := "aarch64".
  Definition model := "default".
  Definition abi := "apple".
  Definition bitsize := 64.
  Definition big_endian := false.
  Definition source_file := "main2.c".
  Definition normalized := true.
End Info.

Definition ___builtin_annot : ident := 17%positive.
Definition ___builtin_annot_intval : ident := 18%positive.
Definition ___builtin_bswap : ident := 2%positive.
Definition ___builtin_bswap16 : ident := 4%positive.
Definition ___builtin_bswap32 : ident := 3%positive.
Definition ___builtin_bswap64 : ident := 1%positive.
Definition ___builtin_cls : ident := 26%positive.
Definition ___builtin_clsl : ident := 27%positive.
Definition ___builtin_clsll : ident := 28%positive.
Definition ___builtin_clz : ident := 5%positive.
```

```
Definition ___builtin_clzl : ident := 6%positive.
Definition ___builtin_clzll : ident := 7%positive.
Definition ___builtin_ctz : ident := 8%positive.
Definition ___builtin_ctzl : ident := 9%positive.
Definition ___builtin_ctzll : ident := 10%positive.
Definition ___builtin_debug : ident := 35%positive.
Definition ___builtin_expect : ident := 25%positive.
Definition ___builtin_fabs : ident := 11%positive.
Definition ___builtin_fabsf : ident := 12%positive.
Definition ___builtin_fmadd : ident := 29%positive.
Definition ___builtin_fmax : ident := 33%positive.
Definition ___builtin_fmin : ident := 34%positive.
Definition ___builtin_fmsub : ident := 30%positive.
Definition ___builtin_fnmadd : ident := 31%positive.
Definition ___builtin_fnmsub : ident := 32%positive.
Definition ___builtin_fsqrt : ident := 13%positive.
Definition ___builtin_membar : ident := 19%positive.
Definition ___builtin_memcpy_aligned : ident := 15%positive.
Definition ___builtin_sel : ident := 16%positive.
Definition ___builtin_sqrt : ident := 14%positive.
Definition ___builtin_unreachable : ident := 24%positive.
Definition ___builtin_va_arg : ident := 21%positive.
Definition ___builtin_va_copy : ident := 22%positive.
Definition ___builtin_va_end : ident := 23%positive.
Definition ___builtin_va_start : ident := 20%positive.
Definition ___compcert_i64_dtos : ident := 44%positive.
Definition ___compcert_i64_dtou : ident := 45%positive.
Definition ___compcert_i64_sar : ident := 56%positive.
Definition ___compcert_i64_sdiv : ident := 50%positive.
Definition ___compcert_i64_shl : ident := 54%positive.
Definition ___compcert_i64_shr : ident := 55%positive.
Definition ___compcert_i64_smod : ident := 52%positive.
Definition ___compcert_i64_smulh : ident := 57%positive.
Definition ___compcert_i64_stod : ident := 46%positive.
Definition ___compcert_i64_stof : ident := 48%positive.
Definition ___compcert_i64_udiv : ident := 51%positive.
Definition ___compcert_i64_umod : ident := 53%positive.
Definition ___compcert_i64_umulh : ident := 58%positive.
Definition ___compcert_i64_utod : ident := 47%positive.
Definition ___compcert_i64_utof : ident := 49%positive.
Definition ___compcert_va_composite : ident := 43%positive.
Definition ___compcert_va_float64 : ident := 42%positive.
```

```
Definition ___compcert_va_int32 : ident := 40%positive.
Definition ___compcert_va_int64 : ident := 41%positive.
Definition _a : ident := 61%positive.
Definition _b : ident := 62%positive.
Definition _c : ident := 63%positive.
Definition _cell : ident := 60%positive.
Definition _cons : ident := 71%positive.
Definition _d : ident := 64%positive.
Definition _exit : ident := 38%positive.
Definition _free : ident := 37%positive.
Definition _freelist : ident := 67%positive.
Definition _i : ident := 78%positive.
Definition _main : ident := 59%positive.
Definition _malloc : ident := 36%positive.
Definition _n : ident := 68%positive.
Definition _newstack : ident := 77%positive.
Definition _next : ident := 73%positive.
Definition _p : ident := 69%positive.
Definition _placeholder : ident := 39%positive.
Definition _pool : ident := 65%positive.
Definition _pool_index : ident := 66%positive.
Definition _pop : ident := 81%positive.
Definition _pop_and_add : ident := 86%positive.
Definition _pp : ident := 70%positive.
Definition _push : ident := 80%positive.
Definition _push_increasing : ident := 83%positive.
Definition _q : ident := 79%positive.
Definition _s : ident := 85%positive.
Definition _st : ident := 82%positive.
Definition _stack : ident := 74%positive.
Definition _surely_malloc : ident := 76%positive.
Definition _t : ident := 84%positive.
Definition _top : ident := 75%positive.
Definition _triang : ident := 87%positive.
Definition _value : ident := 72%positive.
Definition _t'1 : ident := 88%positive.

Definition f_main := {|
  fn_return := tint;
  fn_callconv := cc_default;
  fn_params := nil;
  fn_vars := nil;
  fn_temps := ((_t'1, tint) :: nil);
```

```
  fn_body :=
(Ssequence
  (Ssequence
    (Scall (Some _t'1)
      (Evar _triang (Tfunction (tint :: nil) tint cc_default))
      ((Econst_int (Int.repr 10) tint) :: nil))
    (Sreturn (Some (Etempvar _t'1 tint))))
  (Sreturn (Some (Econst_int (Int.repr 0) tint))))
|}.

Definition composites : list composite_definition :=
nil.

Definition global_definitions : list (ident × globdef fundef type) :=
((___compcert_va_int32,
   Gfun(External (EF_runtime "__compcert_va_int32"
                     (mksignature (AST.Xptr :: nil) AST.Xint cc_default))
     ((tptr tvoid) :: nil) tuint cc_default)) ::
 (___compcert_va_int64,
   Gfun(External (EF_runtime "__compcert_va_int64"
                     (mksignature (AST.Xptr :: nil) AST.Xlong cc_default))
     ((tptr tvoid) :: nil) tulong cc_default)) ::
 (___compcert_va_float64,
   Gfun(External (EF_runtime "__compcert_va_float64"
                     (mksignature (AST.Xptr :: nil) AST.Xfloat cc_default))
     ((tptr tvoid) :: nil) tdouble cc_default)) ::
 (___compcert_va_composite,
   Gfun(External (EF_runtime "__compcert_va_composite"
                     (mksignature (AST.Xptr :: AST.Xlong :: nil) AST.Xptr
                       cc_default)) ((tptr tvoid) :: tulong :: nil)
     (tptr tvoid) cc_default)) ::
 (___compcert_i64_dtos,
   Gfun(External (EF_runtime "__compcert_i64_dtos"
                     (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
     (tdouble :: nil) tlong cc_default)) ::
 (___compcert_i64_dtou,
   Gfun(External (EF_runtime "__compcert_i64_dtou"
                     (mksignature (AST.Xfloat :: nil) AST.Xlong cc_default))
     (tdouble :: nil) tulong cc_default)) ::
 (___compcert_i64_stod,
   Gfun(External (EF_runtime "__compcert_i64_stod"
                     (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
     (tlong :: nil) tdouble cc_default)) ::
 (___compcert_i64_utod,
```

```
  Gfun(External (EF_runtime "__compcert_i64_utod"
                    (mksignature (AST.Xlong :: nil) AST.Xfloat cc_default))
    (tulong :: nil) tdouble cc_default)) ::
(___compcert_i64_stof,
  Gfun(External (EF_runtime "__compcert_i64_stof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tlong :: nil) tfloat cc_default)) ::
(___compcert_i64_utof,
  Gfun(External (EF_runtime "__compcert_i64_utof"
                    (mksignature (AST.Xlong :: nil) AST.Xsingle cc_default))
    (tulong :: nil) tfloat cc_default)) ::
(___compcert_i64_sdiv,
  Gfun(External (EF_runtime "__compcert_i64_sdiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_udiv,
  Gfun(External (EF_runtime "__compcert_i64_udiv"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_smod,
  Gfun(External (EF_runtime "__compcert_i64_smod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umod,
  Gfun(External (EF_runtime "__compcert_i64_umod"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___compcert_i64_shl,
  Gfun(External (EF_runtime "__compcert_i64_shl"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_shr,
  Gfun(External (EF_runtime "__compcert_i64_shr"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tulong :: tint :: nil) tulong cc_default)) ::
(___compcert_i64_sar,
  Gfun(External (EF_runtime "__compcert_i64_sar"
                    (mksignature (AST.Xlong :: AST.Xint :: nil) AST.Xlong
                      cc_default)) (tlong :: tint :: nil) tlong cc_default)) ::
(___compcert_i64_smulh,
```

```
  Gfun(External (EF_runtime "__compcert_i64_smulh"
                  (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                    cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___compcert_i64_umulh,
  Gfun(External (EF_runtime "__compcert_i64_umulh"
                  (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                    cc_default)) (tulong :: tulong :: nil) tulong
    cc_default)) ::
(___builtin_bswap64,
  Gfun(External (EF_builtin "__builtin_bswap64"
                  (mksignature (AST.Xlong :: nil) AST.Xlong cc_default))
    (tulong :: nil) tulong cc_default)) ::
(___builtin_bswap,
  Gfun(External (EF_builtin "__builtin_bswap"
                  (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap32,
  Gfun(External (EF_builtin "__builtin_bswap32"
                  (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tuint cc_default)) ::
(___builtin_bswap16,
  Gfun(External (EF_builtin "__builtin_bswap16"
                  (mksignature (AST.Xint16unsigned :: nil)
                    AST.Xint16unsigned cc_default)) (tushort :: nil) tushort
    cc_default)) ::
(___builtin_clz,
  Gfun(External (EF_builtin "__builtin_clz"
                  (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_clzl,
  Gfun(External (EF_builtin "__builtin_clzl"
                  (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_clzll,
  Gfun(External (EF_builtin "__builtin_clzll"
                  (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctz,
  Gfun(External (EF_builtin "__builtin_ctz"
                  (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tuint :: nil) tint cc_default)) ::
(___builtin_ctzl,
```

```
  Gfun(External (EF_builtin "__builtin_ctzl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_ctzll,
  Gfun(External (EF_builtin "__builtin_ctzll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tulong :: nil) tint cc_default)) ::
(___builtin_fabs,
  Gfun(External (EF_builtin "__builtin_fabs"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_fabsf,
  Gfun(External (EF_builtin "__builtin_fabsf"
                    (mksignature (AST.Xsingle :: nil) AST.Xsingle cc_default))
    (tfloat :: nil) tfloat cc_default)) ::
(___builtin_fsqrt,
  Gfun(External (EF_builtin "__builtin_fsqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_sqrt,
  Gfun(External (EF_builtin "__builtin_sqrt"
                    (mksignature (AST.Xfloat :: nil) AST.Xfloat cc_default))
    (tdouble :: nil) tdouble cc_default)) ::
(___builtin_memcpy_aligned,
  Gfun(External (EF_builtin "__builtin_memcpy_aligned"
                    (mksignature
                      (AST.Xptr :: AST.Xptr :: AST.Xlong :: AST.Xlong :: nil)
                      AST.Xvoid cc_default))
    ((tptr tvoid) :: (tptr tvoid) :: tulong :: tulong :: nil) tvoid
    cc_default)) ::
(___builtin_sel,
  Gfun(External (EF_builtin "__builtin_sel"
                    (mksignature (AST.Xbool :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    (tbool :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
(___builtin_annot,
  Gfun(External (EF_builtin "__builtin_annot"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid
                      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
    ((tptr tschar) :: nil) tvoid
    {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
```

```
(___builtin_annot_intval,
  Gfun(External (EF_builtin "__builtin_annot_intval"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xint
                      cc_default)) ((tptr tschar) :: tint :: nil) tint
    cc_default)) ::
(___builtin_membar,
  Gfun(External (EF_builtin "__builtin_membar"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_va_start,
  Gfun(External (EF_builtin "__builtin_va_start"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_va_arg,
  Gfun(External (EF_builtin "__builtin_va_arg"
                    (mksignature (AST.Xptr :: AST.Xint :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: tuint :: nil) tvoid
    cc_default)) ::
(___builtin_va_copy,
  Gfun(External (EF_builtin "__builtin_va_copy"
                    (mksignature (AST.Xptr :: AST.Xptr :: nil) AST.Xvoid
                      cc_default)) ((tptr tvoid) :: (tptr tvoid) :: nil) tvoid
    cc_default)) ::
(___builtin_va_end,
  Gfun(External (EF_builtin "__builtin_va_end"
                    (mksignature (AST.Xptr :: nil) AST.Xvoid cc_default))
    ((tptr tvoid) :: nil) tvoid cc_default)) ::
(___builtin_unreachable,
  Gfun(External (EF_builtin "__builtin_unreachable"
                    (mksignature nil AST.Xvoid cc_default)) nil tvoid
    cc_default)) ::
(___builtin_expect,
  Gfun(External (EF_builtin "__builtin_expect"
                    (mksignature (AST.Xlong :: AST.Xlong :: nil) AST.Xlong
                      cc_default)) (tlong :: tlong :: nil) tlong cc_default)) ::
(___builtin_cls,
  Gfun(External (EF_builtin "__builtin_cls"
                    (mksignature (AST.Xint :: nil) AST.Xint cc_default))
    (tint :: nil) tint cc_default)) ::
(___builtin_clsl,
  Gfun(External (EF_builtin "__builtin_clsl"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
```

```
    (tlong :: nil) tint cc_default)) ::
(___builtin_clsll,
  Gfun(External (EF_builtin "__builtin_clsll"
                    (mksignature (AST.Xlong :: nil) AST.Xint cc_default))
    (tlong :: nil) tint cc_default)) ::
(___builtin_fmadd,
  Gfun(External (EF_builtin "__builtin_fmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmsub,
  Gfun(External (EF_builtin "__builtin_fmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmadd,
  Gfun(External (EF_builtin "__builtin_fnmadd"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fnmsub,
  Gfun(External (EF_builtin "__builtin_fnmsub"
                    (mksignature
                      (AST.Xfloat :: AST.Xfloat :: AST.Xfloat :: nil)
                      AST.Xfloat cc_default))
    (tdouble :: tdouble :: tdouble :: nil) tdouble cc_default)) ::
(___builtin_fmax,
  Gfun(External (EF_builtin "__builtin_fmax"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
    cc_default)) ::
(___builtin_fmin,
  Gfun(External (EF_builtin "__builtin_fmin"
                    (mksignature (AST.Xfloat :: AST.Xfloat :: nil) AST.Xfloat
                      cc_default)) (tdouble :: tdouble :: nil) tdouble
    cc_default)) ::
(___builtin_debug,
  Gfun(External (EF_external "__builtin_debug"
                    (mksignature (AST.Xint :: nil) AST.Xvoid
```

```
                                {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|}))
      (tint :: nil) tvoid
      {|cc_vararg:=(Some 1); cc_unproto:=false; cc_structret:=false|})) ::
 (_triang,
   Gfun(External (EF_external "triang"
                        (mksignature (AST.Xint :: nil) AST.Xint cc_default))
      (tint :: nil) tint cc_default)) :: (_main, Gfun(Internal f_main)) ::
 nil).

Definition public_idents : list ident :=
(_main :: _triang :: ___builtin_debug :: ___builtin_fmin ::
 ___builtin_fmax :: ___builtin_fnmsub :: ___builtin_fnmadd ::
 ___builtin_fmsub :: ___builtin_fmadd :: ___builtin_clsll ::
 ___builtin_clsl :: ___builtin_cls :: ___builtin_expect ::
 ___builtin_unreachable :: ___builtin_va_end :: ___builtin_va_copy ::
 ___builtin_va_arg :: ___builtin_va_start :: ___builtin_membar ::
 ___builtin_annot_intval :: ___builtin_annot :: ___builtin_sel ::
 ___builtin_memcpy_aligned :: ___builtin_sqrt :: ___builtin_fsqrt ::
 ___builtin_fabsf :: ___builtin_fabs :: ___builtin_ctzll ::
 ___builtin_ctzl :: ___builtin_ctz :: ___builtin_clzll :: ___builtin_clzl ::
 ___builtin_clz :: ___builtin_bswap16 :: ___builtin_bswap32 ::
 ___builtin_bswap :: ___builtin_bswap64 :: ___compcert_i64_umulh ::
 ___compcert_i64_smulh :: ___compcert_i64_sar :: ___compcert_i64_shr ::
 ___compcert_i64_shl :: ___compcert_i64_umod :: ___compcert_i64_smod ::
 ___compcert_i64_udiv :: ___compcert_i64_sdiv :: ___compcert_i64_utof ::
 ___compcert_i64_stof :: ___compcert_i64_utod :: ___compcert_i64_stod ::
 ___compcert_i64_dtou :: ___compcert_i64_dtos :: ___compcert_va_composite ::
 ___compcert_va_float64 :: ___compcert_va_int64 :: ___compcert_va_int32 ::
 nil).

Definition prog : Clight.program :=
  mkprogram composites global_definitions public_idents _main Logic.I.
```

Chapter 13

Library VC.Preface

13.1 Preface

13.2 Welcome

Here's a good way to build formally verified correct software:

- Write your program in an expressive language with a good proof theory (the Gallina language embedded in Rocq's logic).
- Prove it correct in Rocq.
- Extract it to ML and compile it with an optimizing ML compiler.

Unfortunately, for some applications you cannot afford to use a higher-order garbage-collected functional programming language such as Gallina or ML. Perhaps you are writing an operating-system kernel, or a bit-shuffling cryptographic primitive, or the runtime system and garbage-collector of your functional language! In those cases, you might want to use a low-level imperative systems programming language such as C.

But you still want your OS, or crypto, or GC, to be correct! So you should use machine-checked program verification in Rocq. For that purpose, you can use *Verifiable C*, a program logic and proof system for C.

What is a program logic? One example of a program logic is the Hoare logic that you studied in the *Programming Language Foundations* volume of this series. (If you have not done so already, study the Hoare and Hoare2 chapters of that volume, and do the exercises.)

Verifiable C is based on a 21st-century version of Hoare logic called *higher-order impredicative concurrent separation logic*. Back in the 20th century, computer scientists discovered that Hoare Logic was not very good at verifying programs with pointer data structures; so *separation logic* was developed. Hoare Logic was clumsy at verifying concurrent programs, so *concurrent separation logic* was developed. Hoare Logic could not handle higher-order object-oriented programming patterns or function-closures, so *higher-order impredicative program logics* were developed.

This electronic book is Volume 5 of the *Software Foundations* series, which presents the mathematical underpinnings of reliable software. The principal novelty of *Software Foundations* is that it is one hundred percent formalized and machine-checked: the entire text is literally a script for Rocq. It is intended to be read alongside an interactive session with Rocq. All the details in the text are fully formalized in Rocq, and the exercises are designed to be worked using Rocq.

Before studying this volume, you should be a competent user of Rocq:

- Study *Software Foundations Volume 1* (Logical Foundations), and do the exercises!
- Study the Hoare and Hoare2 chapters of *Software Foundations Volume 2* (Programming Language Foundations), and do the exercises!
- Study the Sort, SearchTree, and ADT chapters of *Software Foundations Volume 3* (Verified Functional Algorithms), and do the exercises!

You will also need a working knowledge of the C programming language.

13.3 Practicalities

13.3.1 System Requirements

Rocq runs on Windows, Linux, and OS X. The Preface of Volume 1 describes the Rocq installation you will need. This edition was built with Rocq 9.0.0.

You will need VST 2.16 installed. You can do that either by installing it as part of the standard "Coq Platform" that is released with each new version of Coq, or using opam (the package is named coq-vst). At the end of this chapter is a test to make sure you have the right version of VST installed.

Note: As of January 2026, there is not yet a Coq Platform release for Rocq 9. Install using opam as directed here.

IF YOU USE OPAM, you can install Rocq and RocqIDE using the instructions at https://rocq-prover.org/opam-using.html and then continue with,

- opam update (*as necessary*)
- opam install coq-vst.2.16 (*this will take 30 minutes or more*)

You do not need to install CompCert clightgen to do the exercises in this volume. But if you wish to modify and reparse the .c files, or verify C programs of your own, install the CompCert verified optimizing C compiler. You can get CompCert in the standard "Coq Package" or by opam (the package is named coq-compcert).

13.3.2 Downloading the Rocq Files

A tar file containing the full sources for the "release version" of this book (as a collection of Rocq scripts and HTML files) is available at {https://softwarefoundations.cis.upenn.edu}.

(If you are using the book as part of a class, your professor may give you access to a locally modified version of the files, which you should use instead of the release version.)

13.3.3 Installation

Unpack the *vc.tgz* tar file into a vc directory. In this *vc* directory, the *make* command builds it.

13.3.4 Exercises

Each chapter includes exercises. Each is marked with a "star rating," which can be interpreted as follows:

- One star: easy exercises that underscore points in the text and that, for most readers, should take only a minute or two. Get in the habit of working these as you reach them.
- Two stars: straightforward exercises (five or ten minutes).
- Three stars: exercises requiring a bit of thought (ten minutes to half an hour).
- Four and five stars: more difficult exercises (half an hour and up).

Please do not post solutions to the exercises in any public place: Software Foundations is widely used both for self-study and for

university courses. Having solutions easily available makes it much less useful for courses, which typically have graded homework assignments. The authors especially request that readers not post solutions to the exercises anyplace where they can be found by search engines.

13.3.5 Recommended Citation Format

If you want to refer to this volume in your own writing, please do so as follows:

@book {Appel:SF5, author = {Andrew W. Appel, Lennart Beringer, and Qinxiang Cao}, editor = {Benjamin C. Pierce}, title = "Verifiable C", series = "Software Foundations", volume = "5", year = "2026", publisher = "Electronic textbook", note = {Version 2.0, \URL{http://softwarefoundations.cis.upenn.edu} }, }

13.3.6 For Instructors and Contributors

If you plan to use these materials in your own course, you will undoubtedly find things you'd like to change, improve, or add. Your contributions are welcome! Please see the *Preface* to *Logical Foundations* for instructions.

13.4 Thanks

Development of the *Software Foundations* series has been supported, in part, by the National Science Foundation under the NSF Expeditions grant 1521523, *The Science of Deep Specification*.

13.5 Check that we have the right version of VST

```
Require Import Stdlib.Strings.String.
Open Scope string.
Definition release_needed := "2.16".
Require Import VST.veric.version. Goal release = release_needed.
reflexivity ||
let need := constr:(release_needed) in let need := eval hnf in need in
let rel := constr:(release) in let rel := eval hnf in rel in
fail "The wrong version of VST is installed. You have VST version"
rel "but this version of 'Software Foundations Volume 5: Verifiable C'"
"demands version" need ". If possible, install VST version" need
"using the Coq Platform or using opam. Or, if not From Stdlib Platform or opam,"
"for instructions about building VST from source and accessing that version, see the README
file in this directory."
"Or, instead of installing VST" need ","
"if you want to proceed using VST version" rel ","
"then edit the Definition release_needed in Preface.v".
Abort.
```

```
From Stdlib Require Import ZArith.
Local Open Scope Z_scope.
```

```
Require VC.stack. Goal VC.stack.Info.bitsize = VST.veric.version.bitsize.
reflexivity ||
let b1 := constr:(VST.veric.version.bitsize) in let b1 := eval compute in b1 in
let b2 := constr:(VC.stack.Info.bitsize) in let b2 := eval compute in b2 in
fail "Your installed VST is configured to verify C programs compiled with"
  b1 "bit pointers,"
 "but the .c files in the local directory have been clightgen'ed as if compiled with"
 b2 "bit pointers."
"You can fix this by executing the shell command, "
"bash cfiles-bitsize" b1 " to install the properly configured clightgen outputs."
"It is not necessary to have clightgen installed".
Abort.
```

Chapter 14

Library VC.Verif_sumarray

14.1 Verif_sumarray: Introduction to Verifiable C

14.1.1 Verified Software Toolchain

The Verified Software Toolchain is a toolset for proving the functional correctness of C programs, with

- a *program logic* called Verifiable C, based on separation logic.
- a *proof automation system* called VST-Floyd that assists you in applying the program logic to your program.
- a soundness proof in Rocq, guaranteeing that whatever properties you prove about your program will actually hold in any execution of the C source-language operational semantics. And this proof *composes* with the correctness proof of the CompCert verified optimizing C compiler, so you can also get a guarantee about the behavior of the assembly language program.

This volume of *Software Foundations* teaches you how to use Verifiable C and VST-Floyd to prove C programs correct. In the process you'll learn some key concepts of Hoare Logic and Separation Logic. This book does *not* cover VST's soundness proof (which is described in the book *Program Logics for Certified Compilers* *Appel* 2014 (in Bib.v)).

14.1.2 How to use this textbook

The first two chapters (this one and *Verif_reverse*) are a feature-by-feature introduction to Verifiable C, demonstrated on two example C programs: adding up an array and reversing a linked list. These chapters are best understood if you step through them in Rocq, where you can see the proof goals at each stage; they are less useful to read in HTML. These two chapters closely follow the first 48 mini-chapters of the *Verifiable C Reference Manual*,

VC.pdf, that is distributed with VST – and you can find a copy distributed with this volume of *Software Foundations*. The first two chapters have no exercises.

This *Verifiable C* volume of *Software Foundations* is self-contained, so you should not need to look things up in the reference manual *VC.pdf*. But to use features of Verifiable C beyond what's needed for this textbook, *VC.pdf* can be very useful. The words SEE ALSO suggest which chapters of the reference manual cover the features discussed in this text.

The remaining 7 chapters are *mainly* exercises. The best way to learn is by doing it yourself – so each chapter presents a little C program, and guides you through verifying it yourself. The "capstone exercise" is the verification of a hash table with external chaining.

14.1.3 A C program to add up an array

Here is a little C program, *sumarray.c*:

14.1.4 Workflow

SEE ALSO: VC.pdf Chapter 3 (*Workflow*), Chapter 4 (*Verifiable C*), Chapter 5 (*clightgen*), Chapter 6 (*ASTs*)

To verify a C program, such as *sumarray.c*, use the CompCert front end to parse it into an Abstract Syntax Tree (AST). For all the chapters in this volume of *Software Foundations* we've done that for you, so you don't have to install clightgen; but generally what you would do is,

clightgen -normalize sumarray.c

You would have installed *clightgen* as part of the CompCert tools, by mentioning the -clightgen option when you run ./configure when building CompCert.

The output of *clightgen* would be a file *sumarray.v* that contains the Rocq inductive data structure describing the syntax trees of the source program. You can open *sumarray.v* in the current directory and inspect it.

14.1.5 Let's verify!

SEE ALSO: VC.pdf Chapter 8 (*Functional model, API spec*)

This file, *Verif_sumarray.v*, contains a *specification* of the functional correctness of the program *sumarray.c*, followed by a proof that the program satisfies its specification.

For larger programs, one would typically break this down into three or more files:

- Functional model (often in the form of a Rocq function)
- API specification
- Function-body correctness proofs, one per file.

Make sure you have the right version of VST installed

```
Require VC.Preface.
```

Standard boilerplate

Every API specification begins with the same standard boilerplate; the only thing that changes is the name of the program – in this case, *sumarray*.

```
Require Import VST.floyd.proofauto.
Require Import VC.sumarray.
#[export] Instance CompSpecs : compspecs. make_compspecs prog. Defined.
Definition Vprog : varspecs. mk_varspecs prog. Defined.
```

The first line imports Verifiable C and its *Floyd* proof-automation library. The second line imports the AST of the program to be verified. The third line processes all the struct and union definitions in the AST, and the fourth line processes global variable declarations.

Functional model

To prove correctness of *sumarray.c*, we start by writing a *functional model* of adding up a sequence. We can use a list-fold to express the sum of all the elements in a list of integers:

```
Definition sum_Z : list Z → Z := fold_right Z.add 0.
```

Then we prove properties of the functional model: in this case, how *sum_Z* interacts with list append.

```
Lemma sum_Z_app:
  ∀ a b, sum_Z (a++b) = sum_Z a + sum_Z b.
Proof.
  intros. induction a; simpl; lia.
Qed.
```

The data types used in a functional model can be any kind of mathematics at all, as long as we have a way to relate them to the integers, tuples, and sequences used in a C program. But the mathematical integers *Z* and the 32-bit modular integers *Int.int* are often relevant. Notice that this functional spec does not depend on *sumarray.v* or on anything in the Verifiable C libraries. This is typical, and desirable: the functional model is about mathematics, not about C programming.

14.1.6 API spec for the sumarray.c program

The Application Programmer Interface (API) of a C program is expressed in its header file: function prototypes and data-structure definitions that explain how to call upon the modules' functionality. In Verifiable C, an *API specification* is written as a series of *function specifications* (*funspec*s) corresponding to the function prototypes.

```
Definition sumarray_spec : ident × funspec :=
DECLARE _sumarray
 WITH a: val, sh : share, contents : list Z, size: Z
 PRE [ tptr tuint, tint ]
  PROP (readable_share sh; 0 ≤ size ≤ Int.max_signed;
        Forall (fun x ⇒ 0 ≤ x ≤ Int.max_unsigned) contents)
  PARAMS (a; Vint (Int.repr size))
  SEP (data_at sh (tarray tuint size) (map Vint (map Int.repr contents)) a)
 POST [ tuint ]
  PROP () RETURN (Vint (Int.repr (sum_Z contents)))
  SEP (data_at sh (tarray tuint size) (map Vint (map Int.repr contents)) a).
```

This *DECLARE* statement has type *ident*×*funspec*. That is, it associates the name of a function (the identifier *_sumarray*) with a function-specification. The identifier *_sumarray* comes directly from the C program, as parsed by *clightgen*. If you are curious, you can look in *sumarray.v* (the output of *clightgen*) for `Definition` *_sumarray* := .... Later in *sumarray.v*, you can see `Definition` *f_sumarray* that is the C-language function body (represented as a syntax tree).

A function is specified by its *precondition* and its *postcondition*. The *WITH* clause quantifies over Rocq values that may appear in both the precondition and the postcondition. The precondition has access to the function parameters (in this case *a* and *size*) and the postcondition has access to the return value (*sum_Z contents*).

Function preconditions, postconditions, and loop invariants are *assertions* about the state of variables and memory at a particular program point. In an assertion *PROP*(*P*) *LOCAL*(*Q*) *SEP*(*R*), the propositions in the sequence *P* are all of Rocq type `Prop`. They describe things that are true independent of program state. In the precondition above, the statement $0 \le size \le Int.max\_signed$ is true *just within the scope of the quantification of the variable size*; that variable is bound by *WITH*, and spans the *PRE* and *POST* assertions.

The *LOCAL* clause, describing what's in C local variables, takes different forms depending on context:

- In a function-precondition, we write PROP/PARAMS/SEP, that is, the *PARAMS* lists the values of C function parameters (in order).
- In a function-postcondition, we write *RETURN*(*v*) to indicate the return value of the function.
- Within a function body (in assertions and invariants) we write *LOCAL* to describe the values of local variables (including parameters).

Whether it is PARAMS or RETURN or LOCAL, we are talking about the *values* contained in parameters or local variables. In general, a C scalar variable holds something of type *val*; this type is defined by CompCert as, `Print` *val*.

In an assertion *PROP*(*P*) *LOCAL*(*Q*) *SEP*(*R*), the *SEP* conjuncts *R* are *spatial assertions* in separation logic. In our example precondition, there's just one SEP conjunct, a *data_at* assertion saying that at address *a* in memory, there is a data structure of type

array*size* of unsigned int;

with access-permission *sh*, and the contents of that array is the sequence *map Vint* (*map Int.repr contents*).

The postcondition is introduced by *POST* [*tuint*], indicating that this function returns a value of type *unsigned int*. There are no *PROP* statements in this postcondition–no forever-true facts hold now, that weren't already true on entry to the function.

RETURN(*v*) gives the return value *v*; *RETURN*() for void functions.

The postcondition's *SEP* clause mentions all the spatial resources from the precondition, minus ones that have been freed (deallocated), plus ones that have been malloc'd (allocated).

So, overall, the specification for *sumarray* is this: "At any call to sumarray, there exist values *a*, *sh*, *contents*, *size* such that *sh* gives at least read-permission; *size* is representable as a nonnegative 32-bit signed integer; function-parameter *_a* contains value *a* and *_n* contains the 32-bit representation of *size*; and there's an array in memory at address *a* with permission *sh* containing *contents*. The function returns a value equal to *sum_int*(*contents*), and leaves the array in memory unaltered."

Function specification for main()

The function-spec for *main* has a special form, which we discuss below in the section called *Global variables and main*. In particular, its precondition is defined using *main_pre*.

```
Definition main_spec :=
 DECLARE _main
  WITH gv : globals
  PRE [] main_pre prog tt gv
  POST [ tint ]
     PROP()
     RETURN (Vint (Int.repr (1+2+3+4)))
     SEP(TT).
```

This postcondition says we have indeed added up the global array *four*.

Integer overflow

In Verifiable C's signed integer arithmetic, you must prove (if the system cannot prove automatically) that no overflow occurs. For unsigned integers, arithmetic is treated as modulo-2^n (where n is typically 32 or 64), and overflow is not an issue. The function *Int.repr*: *Z* → *int* truncates mathematical integers into 32-bit integers by taking the (sign-extended) low-order 32 bits. *Int.signed*: *int* → *Z* injects back into the signed integers.

The *sumarray* program uses unsigned arithmetic for *s* and the array contents; it uses signed arithmetic for *i*.

The postcondition guarantees that the value returned is *Int.repr* (*sum_Z contents*). But what if the sum of all the *s* is larger than 2^32, so the sum doesn't fit in a 32-bit signed integer? Then *Int.unsigned*(*Int.repr* (*sum_Z contents*)) $\neq$ *sum_Z contents*. In general, for a claim about *Int.repr*(x) to be *useful* one also needs to know that $0 \leq x \leq$ *Int.max_unsigned* or *Int.min_signed* $\leq x \leq$ *Int.max_signed*. The caller of *sumarray* will probably need to prove $0 \leq$ *sum_Z contents* $\leq$ *Int.max_unsigned* in order to make much use of the postcondition.

14.1.7 Packaging the Gprog and Vprog

SEE ALSO: VC.pdf Chapter 9 (*Proof of the sumarray program*)

To prove the correctness of a whole program,

- 1. Collect the function-API specs together into *Gprog*.
- 2. Prove that each function satisfies its own API spec (with a *semax_body* proof).
- 3. Tie everything together with a *semax_func* proof.

The first step is easy:

```
Definition Gprog := [sumarray_spec; main_spec].
```

What's in *Gprog* are the funspecs that we built using *DECLARE*. (In multi-module programs we would also include imported funspecs.)

In addition to *Gprog*, the API spec contains *Vprog*, the list of global-variable type-specs. This was computed automatically by the *mk_varspecs* tactic, in the "boilerplate" code above.

```
Print Vprog.
Print varspecs.
```

That is, for each C language global variable, *Vprog* gives its name and its C-language type.

14.1.8 Proof of the sumarray program

Now comes the proof that *f_sumarray*, the body of the *sumarray*() function, satisfies *sumarray_spec*, in global context (*Vprog*,*Gprog*). `Lemma` *body_sumarray*: *semax_body Vprog Gprog f_sumarray sumarray_spec*.

Here, *f_sumarray* is the actual function body (AST of the C code) as parsed by *clightgen*; you can read it in *sumarray.v*. You can read *body_sumarray* as claiming: In the context of *Vprog* and *Gprog*, the function body *f_sumarray* satisfies its specification *sumarray_spec*. We need the context in case the sumarray function refers to a global variable (*Vprog* provides the variable's type) or calls a global function (*Gprog* provides the function's API spec).

Now, the proof of *body_sumarray*.

```
Proof.
```

If you are reading this as a static document, you should consider switching to your favorite Rocq development environment, in which you can step through the rest of this chapter, tactic by tactic, and examine the proof state at each point.

start_function

SEE ALSO: VC.pdf Chapter 10 (*start_function*)

The predicate *semax_body* states the Hoare triple of the function body, *Delta* ⊢ {*Pre*} *c* {*Post*}, where *Pre* and *Post* are taken from the *funspec*, *c* is the body of the function, and the type-context *Delta* is calculated from the global type-context overlaid with the parameter- and local-types of the function.

To prove this, we begin with the tactic *start_function*, which takes care of some simple bookkeeping and expresses the Hoare triple to be proved.

start_function.

Some of the assumptions you now see above the line are,

- *a*, *sh*, *contents*, *size*, taken directly from the WITH clause of *sumarray_spec*;
- *Delta_specs*, the context in which Floyd's proof tactics will look up the specifications of global functions;
- *Delta*, the context in which Floyd will look up the types of local and global variables;
- *SH*,*H*,*H0*, taken exactly from the *PROP* clauses of *sumarray_spec*'s precondition.

There are also two *abbreviations* above the line, *POSTCONDITION* and *MORE_COMMANDS*, discussed below.

Forward symbolic execution

SEE ALSO: VC.pdf Chapter 11 (*forward*).

We do Hoare logic proof by forward symbolic execution. At the beginning of this function body, our proof goal is a Hoare triple about the statement (*i*=0; ...*more commands*...). In a forward Hoare logic proof of {*P*}(*i*=0;...*more*...){*R*} we might first apply the sequence rule,

{P}(i=0;){Q} {Q}(...more...){R}

{P}(i=0;...more...){R}

assuming we could derive some appropriate assertion *Q*. For many kinds of statements (assignments, return, break, continue) *Q* is derived automatically by the *forward* tactic, which applies a strongest-postcondition style of proof rule. Let us now apply the *forward* tactic:

forward.

Look at the precondition of the current proof goal, that is, the second argument of *semax*; it has the form *PROP*(...) *LOCAL*(...) *SEP*(...). That precondition is also the *postcondition*

of i=0;. It's much like the *precondition* of i=0; except for one change: we now know that i is equal to 0, which is expressed in the *LOCAL* part as *temp _i* (*Vint* (*Int.repr* 0)).

```
Check 0. Check (Int.repr 0). Check (Vint (Int.repr 0)). Check (temp _i (Vint (Int.repr
0))).
```

abbreviate, MORE_COMMANDS, POSTCONDITION

When doing forward symbolic execution (forward Floyd/Hoare proof) through a large function, you don't usually want to see the entire function-body in your proof subgoal. Therefore the system abbreviates some things for you, using the magic of Rocq's implicit arguments.

```
Check @abbreviate.
About abbreviate.
```

We see here that *abbreviate* is just the identity function, with *both* of its arguments implicit!

To examine the actual contents of MORE_COMMANDS, just do this:

```
unfold abbreviate in MORE_COMMANDS.
```

or alternately, `subst MORE_COMMANDS; unfold abbreviate.`

Similarly, to see the POSTCONDITION, just do,

```
unfold abbreviate in POSTCONDITION.
```

Hint

In any VST proof state, the *hint* tactic will print a suggestion (if it can) that will help you make progress in the proof. In stepping through the case study in this chapter, insert *hint* at any point to see what it says.

```
hint.
```

Then delete the hints! (They slow down replay of your proof.)

The hint here suggests using *abbreviate_semax*, which will undo the `unfold abbreviate` that we did above. Really this is optional; if we don't do *abbreviate_semax*, the next *forward* tactic will do it for us.

```
abbreviate_semax.
hint.
```

This time, the hint suggests that we try 'forward'.

Forward through another assignment statement.

```
forward.
```

The *forward* tactic works on assignment statements, break, continue, and return.

While loops, forward_while

SEE ALSO: VC.pdf Chapter 13 (*if, while, for*) and Chapter 14 (*while loops*).

To do symbolic execution through a *while* loop, use the *forward_while* tactic; you must supply a loop invariant. *forward_while*

```
(EX i: Z,
  PROP (0 ≤ i ≤ size)
  LOCAL (temp _a a;
         temp _i (Vint (Int.repr i));
         temp _n (Vint (Int.repr size));
         temp _s (Vint (Int.repr (sum_Z (sublist 0 i contents)))))
  SEP (data_at sh (tarray tuint size) (map Vint (map Int.repr contents)) a)).
```

A loop invariant is an assertion, almost always in the form of an existential quantifier, *EX*...*PROP*(...)*LOCAL*(...)*SEP*(...). Each iteration of the loop has a state characterized by a different value of some iteration variable(s), the *EX* binds that value.

forward_while leaves four subgoals; here we label them with the - bullet. - *hint*.

The first subgoal is to prove that the current assertion (precondition) entails the loop invariant.

Proving separation-logic entailments

SEE ALSO: VC.pdf Chapter 15 (*PROP LOCAL SEP*) and Chapter 16 (*Entailments*)

This proof goal is an *entailment*, *ENTAIL Delta*, *P* |– *Q*, meaning "in context *Delta*, any state that satisfies *P* will also satisfy *Q*."

In this case, the right-hand-side of this entailment is existentially quantified; it says: there exists a value *i* such that (among other things) *temp _i* (*Vint* (*Int.repr i*)), that is, the C variable *_i* contains the value *i*. But the left-hand-side of the entailment says *temp _i* (*Vint* (*Int.repr* 0)), that is, the C variable *_i* contains 0.

This is analogous to the following situation:

```
Set Nested Proofs Allowed.
Goal ∀ (f: Z→Z) (x: Z), f(x)=0 → ∃ i:Z, f(x)=i.
  intros.
```

To prove such a goal, one uses Rocq's "exists" tactic to demonstrate a value for *i*: ∃ 0.

```
  auto.
Qed.
```

In a separation logic entailment, one can prove an *EX* on the right-hand side by using the *Exists* tactic to demonstrate a value for the quantified variable: *Exists* 0.

Notice that *i* has now been replaced with 0 on the right side.

To prove entailments, we usually use the *entailer*! tactic to simplify the entailment as much as possible–or in many cases, to prove it entirely.

entailer!.

In this case, it solves entirely; in other cases, entailer! leaves subgoals for you to prove.

Type-checking the loop test

- *hint*.

The second subgoal of *forward_while* is always to prove that the loop-test expression can evaluate without crashing–that is, all the variables it references exist and are initialized, it doesn't divide by zero, et cetera.

We call this a "type-checking condition", the predicate *tc_expr*. In this case, it's the while-loop test $i<n$ that must execute, so we see *tc_expr Delta* (! ($\_i < \_n$)) on the right-hand side of the entailment.

Very often, these *tc_expr* goals solve automatically by *entailer*!. *entailer*!.

and indeed, this subgoal is solved.

Proving that the loop body preserves the loop invariant

- *hint*.

The third subgoal of *forward_while* is to prove that the loop body preserves the loop invariant. We must forward-symbolic-execute through the loop body.

SEE ALSO: VC.pdf Chapter 17 (*Array subscripts*)

Examine the proof goal at the beginning of the loop body. Above the line is the variable i, introduced automatically by *forward_while* from the existential *EX* i:*Z* in the loop invariant.

The first C command in the loop body is the array subscript, $\_x = a[\_i]$; . In order to prove this statement, the *forward* tactic needs to be able to prove that i is within bounds of the array. When we try *forward*, it fails:

Fail forward.

SEE ALSO: VST.pdf, Chapter "assert_PROP"

The required information to prove *Zlength contents* = *size* comes from the *precondition* of the current *semax* goal. In the precondition, we have

data_at sh (tarray tuint size) (map Vint (map Int.repr contents)) a

The *data_at* predicate always enforces that the "contents" list for an array is exactly the same length as the size of the array.

To make use of precondition facts in an assertion, use *assert_PROP*.

assert_PROP (*Zlength contents* = *size*). {

The proof goal is an entailment, with the current precondition on the left, and the proposition to be proved on the right. As usual, to prove an entailment, we use the *entailer*! tactic to simplify the proof goal:

entailer!.

Indeed, *entailer*! has done almost all the work. If you want to see how *entailer*! did it, undo the last step and use these two tactics: *go_lower*. *saturate_local*. The job of *go_lower* is to process the PROP and LOCAL parts of the entailment; and *saturate_local* derives all

the propositional facts derivable from the *mpred*s on the left-hand-side, and puts those facts above the line. In this case, above the line is, *Zlength* (*unfold_reptype* (*map Vint* (*map Int.repr contents*))) = *size* which is the fact we need. *hint*.

The *hint* suggests that *list_solve* solves this goal, *Zlength contents* = *Zlength* (*map Vint* (*map Int.repr contents*)). Indeed, *list_solve* knows a lot of things about the interaction of list operators: *Zlength*, *map*, *sublist*, etc.

Or, we can solve the goal "by hand": `do 2 rewrite` *Zlength_map*. `reflexivity`.

}

hint.

Now that we have *Zlength contents* = *size* above the line, we can go *forward* through the array-subscript statement. *forward*.

Now *forward* through the rest of the loop body. *forward*. *forward*.

SEE ALSO: VC.pdf Chapter 18 (*At the end of the loop body*)

We have reached the end of the loop body, and it's time to prove that the *current precondition* (which is the postcondition of the loop body) entails the loop invariant. *Exists* (*i*+1).

entailer!.

`f_equal. f_equal.`

Here the proof goal is,

sum_Z (sublist 0 (i + 1) contents) = sum_Z (sublist 0 i contents) + Znth i contents

We will prove this in stages:

sum_Z (sublist 0 (i + 1) contents) = sum_Z (sublist 0 i contents ++ sublist i (i+1) contents) = sum_Z (sublist 0 i contents) + sum_Z (sublist i (i+1) contents) = sum_Z (sublist 0 i contents) + sum_Z (Znth i contents :: nil) = sum_Z (sublist 0 i contents) + Znth i contents

`rewrite` (*sublist_split* 0 *i* (*i*+1)) `by` *lia*.

`rewrite` *sum_Z_app*. `rewrite` (*sublist_one i*) `by` *lia*.

`simpl`. *lia*.

After the loop, our precondition is the conjunction of the loop invariant and the negation of the loop test.

SEE ALSO: VC.pdf Chapter 19 (*Returning from a function*)

- *hint*.

You can always go *forward* through a `return` statement. The resulting proof goal is an entailment, that the current precondition implies the function's postcondition.

forward.

Here we prove that the postcondition of the function body entails the postcondition demanded by the function specification. *entailer*!.

hint.

`autorewrite with` *sublist* `in` *⊢.

hint.

`autorewrite with` *sublist*.

hint.
reflexivity.
Qed.

14.1.9 Global variables and main()

SEE ALSO: VC.pdf Chapter 20 (*Global variables and main*) Definition *four_contents* := [1; 2; 3; 4].

Lemma *body_main*: *semax_body Vprog Gprog f_main main_spec*.
Proof.
  start_function.

C programs may have extern global variables, either with explicit initializers or implicitly initialized to zero. Because they live in memory, they need to be described by a separation logic predicate, a "resource" that gets passed from one function to another via the SEP part of funspec preconditions and postconditions. Initially, all the global-variable resources are passed into the *main* function, as its precondition. The built-in operator *main_pre* calculates this precondition of *main* by examining all the global declarations of the program.

In this program, there is one global variable,
unsigned four4 = {1,2,3,4};
and we can see its SEP assertion in the precondition of the current proof goal:
data_at Ews (tarray tuint 4) (map Vint *Int.repr* 1; *Int.repr* 2; *Int.repr* 3; *Int.repr* 4) (gv

Chapter 15

Library VC.Verif_reverse

15.1 Verif_reverse: Linked lists in Verifiable C

This chapter demonstrates some more features of Verifiable C. There are no exercises in this chapter.

15.1.1 Running Example

Here is a little C program, *reverse.c*:

SEE ALSO VC.pdf Chapter 50 (*Proof of the reverse program*)

As usual, we import the Verifiable C system *VST.floyd.proofauto*, then the program to be verified, in this case *reverse*. Then we give the standard boilerplate definitions of *CompSpecs* and *Vprog*.

```
Require VC.Preface. Require Import VST.floyd.proofauto.
Require Import VC.reverse.
#[export] Instance CompSpecs : compspecs. make_compspecs prog. Defined.
Definition Vprog : varspecs. mk_varspecs prog. Defined.
```

15.1.2 Inductive definition of linked lists

Tstruct _list noattr is the AST (abstract syntax tree) description of the C-language type `struct list`. We will be using this a lot, so we make an abbreviation for it, t_list:

```
Definition t_list := Tstruct _list noattr.
```

We will define a separation-logic predicate, *listrep sigma p*, to describe the concept that the address *p* in memory is a linked list that represents the mathematical sequence *sigma*. Here, *sigma* is a list of *val*, which is C's "value" type: integers, pointers, floats, etc.

```
Fixpoint listrep (sigma: list val) (p: val) : mpred :=
 match sigma with
 | h::hs ⇒
```

```
    EX y:val, data_at Tsh t_list (h,y) p × listrep hs y
| nil ⇒
    !! (p = nullval) && emp
end.
```

This says, if *sigma* has head *h* and tail *hs*, then there is a cons cell at address *p* with components (*h*,*y*). This cons cell is described by *data_at Tsh t_list* (*h*,*y*) *p*. Separate from that, at address *y*, there is the representation of the rest of the list, *listrep hs y*. The memory footprint for *listrep* (*h*::*hs*) *p* contains the first cons cell at address *p*, and the rest of the cons cells in the list starting at address *y*.

But if *sigma* is *nil*, then *p* is the null pointer, and the memory footprint is empty (*emp*). The fact *p*=*nullval* is a pure proposition (Rocq `Prop`); we inject this into the assertion language (Rocq *mpred*) using the !! operator.

Because !!*P* (for a proposition *P*) does not specify any footprint (whether empty or otherwise), we do not use the separating conjunction × to combine it with *emp*; !!*P* has no *spatial* specification to separate from. Instead, we use the ordinary conjunction &&.

Now, we want to prevent the `simpl` tactic from automatically unfolding *listrep*. This is a design choice that you might make differently, in which case, leave out the *Arguments* command.

```
Arguments listrep sigma p : simpl never.
```

15.1.3 Hint databases for spatial operators

Whenever you define a new spatial operator–a definition of type *mpred* such as *listrep*–it's useful to populate two hint databases.

- The *saturate_local* hint is a lemma that extracts pure propositional facts from a spatial fact.
- The *valid_pointer* hint is a lemma that extracts a valid-pointer fact from a spatial lemma.

  Consider this proof goal:

```
Lemma data_at_isptr_example1:
  ∀ (h y p : val) ,
    data_at Tsh t_list (h,y) p |– !! isptr p.
Proof.
intros.
```

isptr p means *p* is a non-null pointer, not NULL or Vundef or a floating-point number:

```
Print isptr.
entailer!.
Qed.

Lemma data_at_isptr_example2:
```

```
  ∀ (h y p : val) ,
    data_at Tsh t_list (h,y) p |– !! isptr p.
Proof.
intros.
```

Let's look more closely at how *entailer*! solves this goal. First, it finds all the pure propositions `Prop` that it can deduce from the *mpred* conjuncts on the left-hand side, and puts them above the line. *saturate_local*.

The *saturate_local* tactic uses a Hint database (also called *saturate_local*) to look up the individual conjuncts on the left-hand side (this particular entailment has just one conjunct). `Print` *HintDb saturate_local*.

In this case, the new propositions above the line are labeled *H* and *H0*.

Next, if the proof goal has just a proposition !!*P* on the right, *entailer*! throws away the left-hand-side and tries to prove *P*. (This is rather aggressive, and can sometimes lose information, that is, sometimes *entailer*! will turn a provable goal into an unprovable goal.)

```
apply prop_right.
```

It happens that *field_compatible* _ _ *p* implies *isptr p*, `Check` *field_compatible_isptr*.

So therefore, *field_compatible_isptr* solves the goal. `eapply` *field_compatible_isptr*; `eauto`.

Now you have some insight into how *entailer*! works. `Qed`.

But when you define a new spatial predicate *mpred* such as *listrep*, the *saturate_local* tactic does not know how to deduce `Prop` facts from the *listrep* conjunct:

```
Lemma listrep_facts_example:
 ∀ sigma p,
    listrep sigma p |– !! (isptr p ∨ p=nullval).
Proof.
intros.
entailer!.
```

Here, *entailer*! threw away the left-hand-side and left an unprovable goal. Let's see why.

```
Abort.

Lemma listrep_facts_example:
 ∀ sigma p,
    listrep sigma p |– !! (isptr p ∨ p=nullval).
Proof.
intros.
```

First *entailer*! would use *saturate_local* to see (from the Hint database) what can be deduced from *listrep sigma p*. *saturate_local*.

But *saturate_local* did not add anything above the line. That's because there's no Hint in the Hint database for *listrep*. Therefore we must add one. The conventional name for such a lemma is *f_local_facts*, if your new predicate is named *f*. `Abort`.

```
Lemma listrep_local_facts:
```

∀ *sigma p*,
listrep sigma p |–
!! (*is_pointer_or_null p* ∧ (*p*=*nullval* ↔ *sigma*=*nil*)).

For each spatial predicate `Definition` *f*(_): *mpred*, there should be *one* "local fact", a lemma of the form *f*(_) |– !! _. On the right hand side, put all the propositions you can derive from *f*(_). In this case, we know:

- *p* is either a pointer or null (it's never *Vundef*, or *Vfloat*, or a nonzero *Vint*).
- *p* is null, if and only if *sigma* is nil.

`Proof`.
`intros`.
We will prove this entailment by induction on sigma *revert p*; `induction` *sigma*; `intros` *p*.
- In the base case, *sigma* is nil. We can unfold the definition of *listrep* to see what that means. `unfold` *listrep*.

Now we have an entailment with a proposition *p*=*nullval* on the left. To move that proposition above the line, we could do *Intros*, but it's easier just to call on *entailer*! to see how it can simplify (and perhaps partially solve) this entailment goal: *entailer*!.
`split`; `auto`.
- In the inductive case, we can again unfold the definition of *listrep* (*a*::*sigma*); but then it's good to fold *listrep sigma*. Replace the semicolon ; with a period in the next line, to see why. `unfold` *listrep*; `fold` *listrep*.

Warning! Sometimes *entailer*! is too aggressive. If we use it here, it will throw away the left-hand side because it doesn't understand how to look inside an EXistential quantitier. The exclamation point ! is a warning that *entailer*! can turn a provable goal into an unprovable goal. Uncomment the next line and see what happens. Then put the comment marks back.

The preferred way to handle *EX y*:*t* on the left-hand-side of an entailment is to use *Intros y*. Uncomment this to try it out, then put the comment marks back.

A less agressive entailment-reducer is *entailer* without the exclamation point. This one never turns a provable goal into an unprovable goal. Here it will Intro the EX-bound variable y. *entailer*.

Should you use *entailer*! or *entailer* in ordinary proofs? Usually *entailer*! is best: it's faster, and it does more work for you. Only if you find that *entailer*! has gone into a dead end, should you use *entailer* instead.

Here it is safe to use *entailer*! *entailer*!.

Notice that entailer! has put several facts above the line: *field_compatible t_list* [] *p* and *value_fits t_list* (*a*,*y*) come from the *saturate_local* hint database, from the *data_at* conjunct; and *is_pointer_or_null y* and *y*=*nullval* ↔ *sigma*=[] come from the the *listrep* conjunct, using the induction hypothesis *IHsigma*.

Now, let's split the goal and take the two cases separately. `split`; `intro`.

```
 +
  clear - H H2.
  subst p.
```
It happens that *field_compatible _ _ p* implies *isptr p*, `Check` *field_compatible_isptr*.

The predicate isptr excludes the null pointer, `Print` *isptr*.

```
Print nullval.
```
Therefore *H* is a contradiction. We can proceed with, `Check` *field_compatible_nullval*.

```
  eapply field_compatible_nullval; eauto.
 +
   inversion H2.
Qed.
```
Now we add this lemma to the Hint database called *saturate_local* #[*export*] `Hint Resolve` *listrep_local_facts* : *saturate_local*.

Valid pointers, and the *valid_pointer* Hint database

In the C language, you can do a pointer comparison such as *p*!=*NULL* or *p*==*q* only if *p* is a *valid pointer*, that is, either NULL or actually pointing within an allocated object. One way to prove that *p* is valid is if, for example, *data_at Tsh t_list* (*h*,*y*) *p*, meaning that *p* is pointing at a list cell. There is a hint database *valid_pointer* from which the predicate *valid_pointer p* can be proved automatically. For example:

```
Lemma struct_list_valid_pointer_example:
  ∀ h y p,
    data_at Tsh t_list (h,y) p |– valid_pointer p.
Proof.
  intros.
  auto with valid_pointer.
Qed.
```
However, the hint database does not know about user-defined separation-logic predicates (*mpred*) such as *listrep*; for example:

```
Lemma listrep_valid_pointer_example:
 ∀ sigma p,
    listrep sigma p |– valid_pointer p.
Proof.
 intros.
 auto with valid_pointer.
```
Notice that `auto with...` did not solve the proof goal `Abort`.

Therefore, we should prove the appropriate lemma, and add it to the Hint database.

```
Lemma listrep_valid_pointer:
  ∀ sigma p,
    listrep sigma p |– valid_pointer p.
```

```
Proof.
 intros.
```
The main point is to unfold listrep. `unfold` *listrep*.

Now we can prove it by case analysis on sigma; we don't even need induction. `destruct` *sigma*; `simpl`.

- The nil case is easy: *hint*.
  entailer!.
- The cons case *Intros y*.

Now this solves using the Hint database *valid_pointer*, because the *data_at Tsh t_list* (*v*,*y*) *p* on the left is enough to prove the goal. `auto with` *valid_pointer*.

```
Qed.
```

Now we add this lemma to the Hint database #[*export*] `Hint Resolve` *listrep_valid_pointer* : *valid_pointer*.

15.1.4 Specification of the *reverse* function.

A *funspec* characterizes the precondition required for calling the function and the postcondition guaranteed by the function. `Definition` *reverse_spec* : *ident* × *funspec* :=

```
 DECLARE _reverse
  WITH sigma : list val, p: val
  PRE [ tptr t_list ]
     PROP () PARAMS (p) SEP (listrep sigma p)
  POST [ (tptr t_list) ]
    EX q:val,
     PROP () RETURN (q) SEP (listrep(rev sigma) q).
```

- The WITH clause says, there is a value *sigma*: *list val* and a value *p*: *val*, visible in both the precondition and the postcondition.
- The PREcondition says,
  - There is one function-parameter, whose C type is "pointer to struct list"
  - PARAMS: The parameter contains the Rocq value *p*;
  - SEP: in memory at address *p* there is a linked list representing *sigma*.
- The POSTcondition says,
  - the function returns a value whose C type is "pointer to struct list"; and
  - there exists a value *q*: *val*, such that
  - RETURN: the function's return value is *q*
  - SEP: in memory at address *q* there is a linked list representing *rev sigma*.

The global function spec characterizes the preconditions/postconditions of all the functions that your proved-correct program will call. Normally you include all the functions here, but in this tutorial example we include only one. `Definition` *Gprog* : *funspecs* := [*reverse_spec*].

15.1.5 Proof of the *reverse* function

For each function definition in the C program, prove that the function-body (in this case, f_reverse) satisfies its specification (in this case, reverse_spec). `Lemma` *body_reverse*: *semax_body Vprog Gprog f_reverse reverse_spec*.
`Proof`.

The start_function tactic "opens up" a semax_body proof goal into a Hoare triple.

start_function.

As usual, the current assertion (precondition) is derived from the PRE clause of the function specification, *reverse_spec*, and the current command *w*=0; ...*more*... is the function body of *f_reverse*.

The first statement (command) in the function-body is the assignment statement *w*=*NULL*;, where *NULL* is a C #*define* that exands to "cast 0 to void-pointer", (*void* ×)0, here ugly-printed as (*tptr tvoid*)(0). To apply the separation-logic assignment rule to this command, simply use the tactic *forward* :

forward.

The new *semax* judgment is for the rest of the function body *after* the command *w*=*NULL*. The precondition of this *semax* is actually the postcondition of the *w*=*NULL* statement. It's much like the precondition of *w*=*NULL*, but contains the additional LOCAL fact, *temp _w* (*Vint* (*Int.repr* 0)), that is, the variable *_w* contains *nullval*.

We can view the Hoare-logic proof of this program as a "symbolic execution", where the symbolic states are assertions. We can symbolically execute the next command by saying *forward* again.

forward.

Examine the precondition, and notice that now we have the additional fact, *temp _v p*.

We cannot take the next step using *forward* ...

Fail forward.

... because the next command is a *while* loop.

15.1.6 The loop invariant

To prove a while-loop, you must supply a loop invariant, such as
(EX s1 ... PROP(...)LOCAL(...)SEP(...).

forward_while
(*EX s1*: *list val*, *EX s2* : *list val*,

```
  EX w: val, EX v: val,
   PROP (sigma = rev s1 ++ s2)
   LOCAL (temp _w w; temp _v v)
   SEP (listrep s1 w; listrep s2 v)).
```

The forward_while tactic leaves four subgoals, which we mark with - (the Rocq "bullet")

```
-
hint.
```

On the left-hand side of this entailment is the precondition (that we had already established by forward symbolic execution to this point) for the entire while-loop. On the right-hand side is the loop invariant, that we just gave to the *forward_while* tactic. Because the right-hand side has four existentials, a good proof strategy is to choose values for them, using the *Exists* tactic.

```
Exists (@nil val) sigma nullval p.
```

Now we have a quantifier-free proof goal; let us see whether *entailer*! can solve some parts of it.

```
entailer!.
```

Indeed, the *entailer*! did a fine job. What's left is a property of our user-defined *listrep* predicate: *emp* |– *listrep* [] *nullval*.

```
unfold listrep.
```

Now that the user-defined predicate is unfolded, *entailer*! can solve the residual entailment.

```
entailer!.
```

```
-
hint.
```

The second subgoal of *forward_while* is to prove that the loop-test condition can execute without crashing. Consider, for example, the C-language while loop, *while* (*a*[*i*]>0) ..., where the value of *i* might exceed the bounds of the array. Then this would be a "buffer overrun", and is "undefined behavior" ("stuck") in the C semantics. We must prove that, given the current precondition (in this case, the loop invariant), the loop test is not "undefined behavior." This proof goal takes the form, *current-precondition* |– *tc_expr Delta e*, where *e* is the loop-test expression. You can pronounce *tc_expr* as "type-check expression", since the Verifiable C type-checker ensures that such expressions are safe (sometimes with a subgoal for you to prove).

Fortunately, in most cases the *entailer*! solves *tc_expr* goals completely automatically:

```
entailer!.
```

```
-
hint.
```

As usual in any Hoare logic (including Separation Logic), we need to prove that the loop body preserves the loop invariant, more precisely,

- {Inv /\ Test} body {Inv}

where Test is the loop-test condition. Here, the loop-test condition in the original C code is (*v*), and its manifestation above the line is the hypothesis *HRE*: *isptr v*, meaning that *v* is a (non-null) pointer.

The loop invariant was *EX s1*:_, *EX s2*:_, *EX w*:_, *EX v*:_, ..., and here all the existentially quantified variables on the left side of the entailment have been moved above the line: *s1*, *s2*: *val* and *w*,*v*: *val*.

The PROP part of the loop invariant was *sigma* = *rev s1* ++ *s2*, and it has also been moved above the line, as hypothesis *H*.

So now we would like to do forward-symbolic execution through the four assignment statements in the loop body.

Fail forward.

But we cannot go forward through *t*=*v*→*tail*; because that would require a SEP conjunct in the precondition of the form *data_at sh t_list* (_,_) *v*, and there is no such conjunct. Actually, there is such a conjunct, but it is hiding inside *listrep s2 v*. That is, there is such a conjunct as long as *s2* is not *nil*. Let's do case analysis on s2:

```
destruct s2 as [ | h r].
```

+

Suppose *s2*=*nil*. If we unfold *listrep* . . . `unfold listrep at 2.`

then we learn that *v*=*nullval*. To move this fact (or any proposition) from the precondition to above-the-line, we use *Intros*:

Intros.

Now, above the line, we have *v*=*nullval* and *isptr v*; this is a contradiction.

`subst`. *contradiction*.

+

Suppose *s2*=*h*::*r*. We can unfold/fold the *listrep* conjunct for *h*::*r*; if you don't remember why we do `unfold/fold`, then replace the semicolon (between the fold and the unfold) with a period and see what happens.

`unfold` *listrep* `at` 2; `fold` *listrep*.

By the definition of *listrep*, at address *v* there must exist a value *y* and a list cell containing (h,y). So let us move *y* above the line:

Intros y.

Now we have the appropriate SEP conjuncts to be able to go *forward* through the loop body

forward. *forward*. *forward*. *forward*.

At the end of loop body; we must reestablish the loop invariant. The left-hand-side of this entailment is the current assertion (after the loop body); the right-hand side is simply our

loop invariant. (Unfortunately, the *forward_while* tactic has "uncurried" the existentials into a single EX that binds a 4-tuple.) Since the proof goal is a complicated-looking entailment, let's see if *entailer*! can simplify it a bit:

entailer!.

Now, we can provide new values for *s1*,*s2*,*w*,*v* to instantiate the four existentials; these are, respectively, *h*::*s1*,*r*,*v*,*y*.

Exists (*h*::*s1*,*r*,*v*,*y*).

Again, we have a complicated-looking entailment; we ask *entailer*! to reduce it some more.

entailer!.
× `simpl. rewrite` *app_ass*. `auto.`
× `unfold` *listrep* `at` 3; `fold` *listrep*.
Exists w. *entailer*!.

-

As usual in any Hoare logic (including Separation Logic), the postcondition of a while-loop is {Inv /\ not Test}, where Inv is the loop invariant and Test is the loop test. Here, all the EXistentials and PROPs of the loop invariant have been moved above the line as *s1*,*s2*,*w*,*v*,*HRE*,*H*.

We can always go *forward* through a `return` statement: *forward*.

```
Exists w; entailer!.
rewrite (proj1 H1) by auto.
unfold listrep at 2; fold listrep.
entailer!.
rewrite app_nil_r, rev_involutive.
auto.
Qed.
```

15.1.7 Why separation logic?

If we review our functional correctness proof for *reverse.c*, it may not be obvious why we need separation logic at all. Let's take a close look.

First, we build "separation" into the definition of *listrep*. The following is our definition:

Fixpoint listrep (sigma: list val) (p: val) : mpred := match sigma with | h::hs => EX y:val, data_at Tsh t_list (h,y) p * listrep hs y | nil => !! (p = nullval) && emp end.

In the nonempty list case, the head element is described by

data_at Tsh t_list (h,y) p

which is separated (by the separating conjunction *) from the rest of the list

listrep hs y.

This separation ensures that no address could be used more than once in a linked list. For example, considering a linked list of length at least 2,

listrep (a :: b :: l) x.

We know that there must be two addresses y and z such that
data_at Tsh t_list (a,y) x * data_at Tsh t_list (b,z) y * listrep l z.

The "separating conjunction" $\times$ tells us that x and y must be different! Formally, we can prove the following two lemmas:

```
Lemma listrep_len_ge2_fact: ∀ (a b x: val) (l: list val),
  listrep (a :: b :: l) x |–
  EX y: val, EX z: val,
       data_at Tsh t_list (a,y) x ×
       data_at Tsh t_list (b,z) y ×
       listrep l z.
Proof.
  intros.
  unfold listrep; fold listrep.
  Intros y z.
  Exists y z.
  cancel.
Qed.

Lemma listrep_len_ge2_address_different: ∀ (a b x y z: val) (l: list val),
  data_at Tsh t_list (a,y) x ×
  data_at Tsh t_list (b,z) y ×
  listrep l z |–
  !! (x ≠ y).
Proof.
  intros.
```

To prove that the addresses are different, we do case analysis first. If $x = y$, we use the following theorem: `Check data_at_conflict.`

It says that we can derives address anti-aliasing from the "separation" defined by $\times$. If $x \neq y$, the right side is already proved.

```
  destruct (Val.eq x y); [| apply prop_right; auto].
  subst x.
  sep_apply (data_at_conflict Tsh t_list (a, y)).
  + auto.
  + entailer!.
Qed.
```

Actually, even the property $x \neq y$ is not strong enough! We need to know that x does not overlap with *any field* of record y, for example (in C notation) x != &($y \rightarrow tail$) and &($x \rightarrow tail$) != y. Otherwise, when storing into $y \rightarrow tail$, we couldn't know that $x \rightarrow head$ is not altered.

Without separation logic, we could still define *listrep'* using extra clauses for address anti-aliasing. For example, a length-3 linked list *listrep* (a :: b :: c :: *nil*) x can be: exists y and z, such that (a, y) is stored at x, (b, z) is stored at y, (c, *nullval*) is stored at z and x, y and z are different from each other. In general, that assertion will be quadratically long (as a function of the length of the linked list). Then, to make sure $x \rightarrow head$ is not at the

same address as $y \rightarrow tail$, we'd need even more assertions.

In our program correctness proof, we do (implicitly) use the fact that different *SEP* clauses describe disjoint heaplets. Here is an intermediate step in the proof of *body_reverse*.

(We rarely state intermediate proof goals such as this one. We do it here to illustrate a point about separating conjunction.)

```
Lemma body_reverse_step: ∀
  {Espec : OracleKind}
  (sigma : list val)
  (s1 : list val)
  (h : val)
  (r : list val)
  (w v : val)
  (HRE : isptr v)
  (H : sigma = rev s1 ++ h :: r)
  (y : val),
  semax (func_tycontext f_reverse Vprog Gprog nil)
    (PROP ( )
     LOCAL (temp _t y; temp _w w; temp _v v)
     SEP (listrep s1 w; data_at Tsh t_list (h, y) v; listrep r y))
    (Ssequence
       (Sassign
          (Efield
             (Ederef (Etempvar _v (tptr (Tstruct _list noattr)))
                     (Tstruct _list noattr))
             _tail (tptr (Tstruct _list noattr)))
          (Etempvar _w (tptr (Tstruct _list noattr))))
       (Ssequence (Sset _w (Etempvar _v (tptr (Tstruct _list noattr))))
                  (Sset _v (Etempvar _t (tptr (Tstruct _list noattr)))))))
    (normal_ret_assert
       (PROP ( )
        LOCAL (temp _v y; temp _w v; temp _t y)
        SEP (listrep s1 w; data_at Tsh t_list (h, w) v; listrep r y))).
Proof.
  intros.
  abbreviate_semax.
```

Now, our proof goal is:
semax Delta (PROP ( ) LOCAL (temp

Chapter 16

Library VC.Verif_stack

16.1 Verif_stack: Stack ADT implemented by linked lists

Here is a little C program, *stack.c*

16.1.1 Let's verify!

```
Require VC.Preface. Require Import VST.floyd.proofauto.
Require Import VST.floyd.library.
Require Import VC.stack.
#[export] Instance CompSpecs : compspecs. make_compspecs prog. Defined.
Definition Vprog : varspecs. mk_varspecs prog. Defined.
Require Import VC.hints.
```

16.1.2 Malloc and free

When you use C's malloc/free library, you write p=*malloc*(n); to get a pointer p to a block of n bytes; when you're done with that block, you call *free*(p) to dispose of it. How does the *free* function know how many bytes to dispose?

The answer is, the malloc/free library puts an extra "header" field just before address p, so really you get this:

+————+ | header | +————+ p–>| zero | +————+ | one | +————+ | two | +————+

where in this case, *header*=3.

In separation logic, we can describe this as

- *malloc_token Ews p* × *data_at Ews* (*Tstruct _mystruct noattr*) (*zero*,*one*,*two*) *p*

where *malloc_token Ews p* describes this picture:

+————+ | header | +————+ p–>

Of course, the malloc/free library might have a different way of "remembering" the size that p points to, so its representation of *malloc_token* is *not necessarily* a word at offset -1. Therefore, clients of the malloc/free library treat *malloc_token* as an abstract predicate. Now, the function-specifications of malloc and free are something like this:

```
Definition malloc_spec_example :=
 DECLARE _malloc
 WITH t:type, gv: globals
 PRE [ tuint ]
    PROP (0 ≤ sizeof t ≤ Int.max_unsigned;
          complete_legal_cosu_type t = true;
          natural_aligned natural_alignment t = true)
    PARAMS (Vint (Int.repr (sizeof t)))
    SEP (mem_mgr gv)
 POST [ tptr tvoid ] EX p:_,
    PROP ()
    RETURN (p)
    SEP (mem_mgr gv;
           if eq_dec p nullval then emp
           else (malloc_token Ews t p × data_at_ Ews t p)).
Definition free_spec_example :=
 DECLARE _free
 WITH t: type, p:val, gv: globals
 PRE [ tptr tvoid ]
     PROP ()
     PARAMS (p)
     SEP (mem_mgr gv; malloc_token Ews t p; data_at_ Ews t p)
 POST [ Tvoid ]
     PROP () RETURN () SEP (mem_mgr gv).
```

If your source program says *malloc*(*sizeof*(t)), your *forward_call* should supply (as a WITH-witness) the C type t. Malloc may choose to return NULL, in which case the SEP part of the postcondition is *emp*, or it may return a pointer, in which case you get *data_at_ Ews t p*, and as a free bonus you get a *malloc_token Ews t p*. But don't lose that *malloc_token*! You will need to supply it later to the *free* function when you dispose of the object.

The SEP predicate *data_at_ Ews t p* is an *uninitialized* structure of type t. It is equivalent to, *data_at Ews t* (*default_val t*) p. The *default_val* is basically a struct or array full of *Vundef* values.

16.1.3 Specification of linked lists

This is much like the linked lists in *Verif_reverse*.

```
Fixpoint listrep (il: list Z) (p: val) : mpred :=
```

```
match il with
| i::il' ⇒ EX y: val,
          malloc_token Ews (Tstruct _cons noattr) p ×
          data_at Ews (Tstruct _cons noattr) (Vint (Int.repr i),y) p ×
          listrep il' y
| nil ⇒ !! (p = nullval) && emp
end.
```

Proof automation for user-defined separation predicates works better if you disable automatic simplification, as follows: `Arguments listrep il p : simpl never`.

As usual, we should populate the Hint databases *saturate_local* and *valid_pointer*

Exercise: 1 star, standard (stack_listrep_properties) `Lemma listrep_local_prop: ∀ il p, listrep il p |–`

```
        !! (is_pointer_or_null p ∧ (p=nullval ↔ il=nil)).
```

See if you can remember how to prove this; or look again at *Verif_reverse* to see how it's done. *Admitted*.

```
#[export] Hint Resolve listrep_local_prop : saturate_local.

Lemma listrep_valid_pointer:
  ∀ il p,
    listrep il p |– valid_pointer p.
```

See if you can remember how to prove this; or look again at *Verif_reverse* to see how it's done. *Admitted*.

```
#[export] Hint Resolve listrep_valid_pointer : valid_pointer.
```

□

16.1.4 Specification of stack data structure

Our stack data structure looks like this:

+———–+ | token | +———–+ +——— p–>| top——+—q–>| linked list... +———–+ +———

The stack object *p* points to a header node with one field *top* (plus a malloc token); the *contents* of the *top* field is some pointer *q* that points to a linked list.

```
Definition stack (il: list Z) (p: val) :=
 EX q: val,
  malloc_token Ews (Tstruct _stack noattr) p ×
  data_at Ews (Tstruct _stack noattr) q p ×
  listrep il q.

Arguments stack il p : simpl never.
```

Exercise: 1 star, standard (stack_properties) `Lemma stack_local_prop: ∀ il p, stack il p |– !! (isptr p).`

```
  Admitted.
#[export] Hint Resolve stack_local_prop : saturate_local.

Lemma stack_valid_pointer:
  ∀ il p,
   stack il p |– valid_pointer p.
   Admitted.
#[export] Hint Resolve stack_valid_pointer : valid_pointer.
   □
```

16.1.5 Function specifications for the stack operations

```
Definition newstack_spec : ident × funspec :=
 DECLARE _newstack
 WITH gv: globals
 PRE [ ]
    PROP () PARAMS() GLOBALS(gv) SEP (mem_mgr gv)
 POST [ tptr (Tstruct _stack noattr) ]
    EX p: val, PROP ( ) RETURN (p) SEP (stack nil p; mem_mgr gv).

Definition push_spec : ident × funspec :=
 DECLARE _push
 WITH p: val, i: Z, il: list Z, gv: globals
 PRE [ tptr (Tstruct _stack noattr), tint ]
    PROP (Int.min_signed ≤ i ≤ Int.max_signed)
    PARAMS (p; Vint (Int.repr i)) GLOBALS(gv)
    SEP (stack il p; mem_mgr gv)
 POST [ tvoid ]
    PROP ( ) RETURN () SEP (stack (i::il) p; mem_mgr gv).

Definition pop_spec : ident × funspec :=
 DECLARE _pop
 WITH p: val, i: Z, il: list Z, gv: globals
 PRE [ tptr (Tstruct _stack noattr) ]
    PROP ()
    PARAMS (p) GLOBALS(gv)
    SEP (stack (i::il) p; mem_mgr gv)
 POST [ tint ]
    PROP ( ) RETURN (Vint (Int.repr i)) SEP (stack il p; mem_mgr gv).
```

Putting all the funspecs together:

```
Definition Gprog : funspecs :=
        ltac:(with_library prog [
                    newstack_spec; push_spec; pop_spec
 ]).
```

16.1.6 Proofs of the function bodies

An *Abstract Data Type* (ADT) is a type provided with a *representation* and a set of *operations*. Clients of the ADT never see the representation, they only call upon the operations. Implementations of the operations do need to manipulate the representation directly.

In this case, *stack* is our ADT. The operations are *newstack*, *push*, and *pop*. Clients of these operations see only *stack il p*, where *il* is the list of values that the client has pushed onto the stack, and *p* is the client's "handle", the address of the representation of the stack. The client does not know whether the abstract list *il* is represented in C data structures by a singly linked list, a doubly linked list, an array, or some other data structure. The client *never unfolds* the Definition *stack*.

The operations *newstack*, *push*, *pop* are implemented in C, and they directly manipulate (in this case) a singly linked list. In proving the correctness of *newstack*, *push*, *pop*, we need to know the representation. Therefore,

Hint: At the beginning of *body_pop*, of *body_push*, and of *body_newstack*, the first thing you should do is `unfold stack in *`.

Exercise: 2 stars, standard (body_pop) `Lemma` *body_pop*: *semax_body Vprog Gprog f_pop pop_spec*.
`Proof`.
start_function.
Admitted.
□

Exercise: 2 stars, standard (body_push) `Lemma` *body_push*: *semax_body Vprog Gprog f_push push_spec*.
`Proof`.
start_function.
forward_call (*Tstruct _cons noattr*, *gv*).
Admitted.
□

Exercise: 2 stars, standard (body_newstack) `Lemma` *body_newstack*: *semax_body Vprog Gprog f_newstack newstack_spec*.
`Proof`.
start_function.
Admitted.
□

Chapter 17

Library VC.Verif_triang

17.1 Verif_triang: A client of the stack functions

```
Require VC.Preface. Require Import VST.floyd.proofauto.
Require Import VST.floyd.library.
Require Import VC.stack.
#[export] Instance CompSpecs : compspecs. make_compspecs prog. Defined.
Definition Vprog : varspecs. mk_varspecs prog. Defined.
```

Here are some functions (in *stack.c*) that are clients of the stack ADT. First, push the numbers 1,2,...,n onto a stack, then pop the numbers off the stack and add them up. This computes the nth triangular number, 1+2+...+n = n(n+1)/2.

void push_increasing (struct stack *st, int n) { int i; i=0; while (i<n) { i++; push(st,i); } }

int pop_and_add (struct stack *st, int n) { int i=0; int t, s=0; while (i<n) { t=pop(st); s += t; i++; } return s; }

int main (void) { struct stack *st; int i,t,s; st = newstack(); push_increasing(st, 10); s = pop_and_add(st, 10); return s; }

Let's verify this program!

17.1.1 Proofs with integers

The natural numbers have arithmetic axioms that are not very nice. For example, you might expect that *a*-*b*+*b*=*a*, but that's not true: `Lemma nat_sub_add_yuck`:

```
  ¬ (∀ a b: nat, a-b+b=a)%nat.
Proof.
intros.
intro.
specialize (H 0 1)%nat.
simpl in H. inversion H.
Qed.
```

This just shows that if the negative numbers did not exist, it would be necessary to construct them! In reasoning about programs, as in many other kinds of mathematics, we should use the integers. In Rocq the type is called *Z*. `Lemma` *Z_sub_add_ok*:
∀ *a b* : *Z*, *a*-*b*+*b*=*a*.
`Proof. intros. lia. Qed.`

The Z type does have an inductive definition . . . `Print` *Z*.

Let's consider a recursive function on Z, the function that turns 5 into the list 5::4::3::2::1::*nil*. In the natural numbers, that's easy to define:

```
Fixpoint decreasing_nat (n: nat) : list nat :=
 match n with S n' ⇒ n :: decreasing_nat n' | O ⇒ nil end.
```

But in the integers *Z*, we cannot simply pattern-match on successor ... *Fail* `Fixpoint`

```
decreasing_Z (n: Z) : list Z :=
 match n with Z.succ n' ⇒ n :: decreasing_Z n' | 0 ⇒ nil end.
```

... because Z.succ is a function, not a constructor.

There are two ways we might define a function to produce a decreasing list of Z. First, we might use *Z.of_nat* and *Z.to_nat*:

```
Fixpoint decreasing_Z1_aux (n: nat) : list Z :=
  match n with
  | S n' ⇒ Z.of_nat n :: decreasing_Z1_aux n'
  | O ⇒ nil
  end.
Definition decreasing_Z1 (n: Z) : list Z :=
      decreasing_Z1_aux (Z.to_nat n).
```

This will work, but in doing proofs the frequent conversion between Z and nat will be awkward. If possible, we'd like to stay in the integers as much as possible. So here's another way:

```
Check Z_gt_dec.

Function decreasing (n: Z) {measure Z.to_nat n}:=
 if Z_gt_dec n 0 then n :: decreasing (n-1) else nil.
Proof.
```

When you define a Function, you must provide a `measure`, that is, a function from your argument-type (in this case Z) to the natural numbers, and then you must prove that each recursive call within the function body decreases the measure. In this case, there's only one recursive call, so there's just one proof obligation: show that if *n*>0 then *Z.to_nat* (*n*-1) < *Z.to_nat n*. *lia*.
`Defined.`

Exercise: 2 stars, standard (Zinduction) Rocq's standard induction principle for Z is not the one we usually want, so let us define a more natural induction scheme: `Lemma` *Zinduction*: ∀ (*P*: *Z* → `Prop`),

```
  P 0 →
  (∀ i, 0 < i → P (i-1) → P i) →
  ∀ n, 0 ≤ n → P n.
Proof.
intros.
rewrite ← (Z2Nat.id n) in * by lia.
set (j := Z.to_nat n) in *. clearbody j.
Check inj_S. Print Z.succ. Admitted.
```
□

A theorem about the nth triangular number

```
Definition add_list: list Z → Z := fold_right Z.add 0.
```

Exercise: 2 stars, standard (add_list_decreasing) Theorem: the sum of the list (*n*)::(*n*-1):: ... :: 2::1 is *n**(*n*+1)/2.

```
Lemma add_list_decreasing_eq_alt: ∀ n,
  0 ≤ n →
  (2 × (add_list (decreasing n)))%Z = (n × (n+1))%Z.
Proof.
  intros.
  pattern n; apply Zinduction.
  - reflexivity.
  - intros.
```

WARNING! When using functions defined by `Function`, don't unfold them! Temporarily remove the brackets from the next line to see what happens!

Instead of unfolding *decreasing* we use the equation that Rocq automagically defines for the Function. Try the command `Search decreasing.` to see all the reasoning principles that Rocq defined for the new `Function`. We will use this one: `Check decreasing_equation.`

```
   rewrite decreasing_equation.
```

during the proof of this lemma, you may find the *ring_simplify* tactic useful. Read about it in the Rocq reference manual. Basically, it takes formulas with multiplication and addition, and simplifies them. But you can do this without *ring_simplify*, using just ordinary rewriting with lemmas about Z.add and Z.mul. *Admitted*.

```
Lemma add_list_decreasing_eq: ∀ n,
  0 ≤ n →
  add_list (decreasing n) = n × (n+1) / 2.
Proof.
  intros.
  apply Z.div_unique_exact.
   Admitted.
```

□

Definitions copied from Verif_stack.v

We repeat here some material from Verif_stack.v. Normally we would break the .c file into separate modules, and do our Verifiable C proofs in separate modules; but for this example we leave out the modules. Just skip down to "End of the material repeated from Verif_stack.v".

Specification of linked lists in separation logic

```
Fixpoint listrep (il: list Z) (p: val) : mpred :=
 match il with
 | i::il' ⇒ EX y: val,
          malloc_token Ews (Tstruct _cons noattr) p ×
          data_at Ews (Tstruct _cons noattr) (Vint (Int.repr i),y) p ×
          listrep il' y
 | nil ⇒ !! (p = nullval) && emp
 end.

Lemma listrep_local_prop: ∀ il p, listrep il p |–
          !! (is_pointer_or_null p ∧ (p=nullval ↔ il=nil)).
Proof.
induction il; intro; simpl.
entailer!. intuition.
Intros y.
entailer!.
split; intros. subst.
eapply field_compatible_nullval; eauto.
inversion H3.
Qed.
#[export] Hint Resolve listrep_local_prop : saturate_local.

Lemma listrep_valid_pointer:
  ∀ il p,
   listrep il p |– valid_pointer p.
Proof.
   Admitted.
#[export] Hint Resolve listrep_valid_pointer : valid_pointer.
```

Specification of stack data structure

```
Definition stack (il: list Z) (p: val) :=
 EX q: val,
  malloc_token Ews (Tstruct _stack noattr) p ×
  data_at Ews (Tstruct _stack noattr) q p × listrep il q.

Lemma stack_local_prop: ∀ il p, stack il p |– !! (isptr p).
Proof.
```

```
  Admitted.
#[export] Hint Resolve stack_local_prop : saturate_local.

Lemma stack_valid_pointer:
  ∀ il p,
   stack il p |– valid_pointer p.
Proof.
  Admitted.
#[export] Hint Resolve stack_valid_pointer : valid_pointer.

Definition newstack_spec : ident × funspec :=
 DECLARE _newstack
 WITH gv: globals
 PRE [ ]
    PROP () PARAMS() GLOBALS(gv) SEP (mem_mgr gv)
 POST [ tptr (Tstruct _stack noattr) ]
    EX p: val, PROP ( ) RETURN (p) SEP (stack nil p; mem_mgr gv).

Definition push_spec : ident × funspec :=
 DECLARE _push
 WITH p: val, i: Z, il: list Z, gv: globals
 PRE [ tptr (Tstruct _stack noattr), tint ]
    PROP (Int.min_signed ≤ i ≤ Int.max_signed)
    PARAMS (p; Vint (Int.repr i)) GLOBALS(gv)
    SEP (stack il p; mem_mgr gv)
 POST [ tvoid ]
    PROP ( ) RETURN() SEP (stack (i::il) p; mem_mgr gv).

Definition pop_spec : ident × funspec :=
 DECLARE _pop
 WITH p: val, i: Z, il: list Z, gv: globals
 PRE [ tptr (Tstruct _stack noattr) ]
    PROP ()
    PARAMS (p) GLOBALS(gv)
    SEP (stack (i::il) p; mem_mgr gv)
 POST [ tint ]
    PROP ( ) RETURN (Vint (Int.repr i)) SEP (stack il p; mem_mgr gv).
```

(End of the material repeated from Verif_stack.v)

17.1.2 Specification of the stack-client functions

Spend a few minutes studying these funspecs, and compare to the implementations in stack.c, until you understand why these might be appropriate specifications.

```
Definition push_increasing_spec :=
 DECLARE _push_increasing
```

```
  WITH st: val, n: Z, gv: globals
  PRE [ tptr (Tstruct _stack noattr), tint ]
    PROP (0 ≤ n ≤ Int.max_signed)
    PARAMS (st; Vint (Int.repr n)) GLOBALS(gv)
    SEP (stack nil st; mem_mgr gv)
  POST [ tvoid ]
    PROP() RETURN() SEP (stack (decreasing n) st; mem_mgr gv).
Definition pop_and_add_spec :=
  DECLARE _pop_and_add
  WITH st: val, il: list Z, gv: globals
  PRE [ tptr (Tstruct _stack noattr), tint ]
    PROP (Zlength il ≤ Int.max_signed;
          Forall (Z.le 0) il;
          add_list il ≤ Int.max_signed)
    PARAMS (st; Vint (Int.repr (Zlength il))) GLOBALS(gv)
    SEP (stack il st; mem_mgr gv)
  POST [ tint ]
    PROP()
    RETURN (Vint (Int.repr (add_list il)))
    SEP (stack nil st; mem_mgr gv).
Definition main_spec :=
  DECLARE _main
  WITH gv: globals
  PRE [ ] main_pre prog tt gv
  POST [ tint ]
    PROP( ) RETURN (Vint (Int.repr 55)) SEP( TT ).
```

Putting all the funspecs together

```
Definition Gprog : funspecs :=
        ltac:(with_library prog [
                    newstack_spec; push_spec; pop_spec;
                    push_increasing_spec; pop_and_add_spec; main_spec
 ]).
```

17.1.3 Proofs of the stack-client function-bodies

Exercise: 3 stars, standard (body_push_increasing) `Lemma body_push_increasing: semax_body Vprog Gprog f_push_increasing push_increasing_spec.`

```
  Admitted.
```

□

Exercise: 2 stars, standard (add_list_lemmas) `Lemma add_list_app:`

```
  ∀ al bl, add_list (al++bl) = add_list al + add_list bl.
   Admitted.

Lemma add_list_nonneg:
 ∀ il,
  Forall (Z.le 0) il →
  0 ≤ add_list il.
   Admitted.
   □
```

Exercise: 2 stars, standard (add_list_sublist_bounds) `Lemma add_list_sublist_bounds:`

```
 ∀ lo hi K il,
  0 ≤ lo ≤ hi →
  hi ≤ Zlength il →
  Forall (Z.le 0) il →
  0 ≤ add_list il ≤ K →
  0 ≤ add_list (sublist lo hi il) ≤ K.
Proof.
```

Hint: you don't need induction. Useful lemmas are, *sublist_same*, *sublist_split*, *add_list_nonneg*, *add_list_app*, *Forall_sublist*, and use the *hint* tactic to learn when the *list_solve* tactic will be useful. *Admitted*.

□

Exercise: 3 stars, standard (add_another) Suppose we have a list *il* of integers, *il* = [5;4;3;2;1], with *Znth* 0 *il* = 5, *Znth* 4 *il* = 1, and *Zlength il* = 5, and we want to add them all up, 5+4+3+2+1=15. Suppose we've already added up the first *i* of them (let *i*=2 for example), that is, 5+4=9, and we want to add the next one, that is, the *ith* one. That is, we want to add 9+3. How do we know that won't overflow the range of C-language signed integer arithmetic?

The proof goes: Every element of the list is nonnegative; the whole list adds up to a number <= Int.max_signed; and any sublist of an all-nonnegative list adds up to less-or-equal to the total of the whole list.

```
Lemma add_another:
 ∀ il,
   Forall (Z.le 0) il →
   add_list il ≤ Int.max_signed →
  ∀ i : Z,
  0 ≤ i < Zlength il →
  Int.min_signed ≤ Int.signed (Int.repr (add_list (sublist 0 i il))) +
                   Int.signed (Int.repr (Znth i il)) ≤ Int.max_signed.
Proof.
intros.
assert (0 ≤ add_list il). {
```

```
  admit.
}
 assert (0 ≤ add_list (sublist 0 i il) ≤ Int.max_signed). {
  admit.
 }
 assert (H4: 0 ≤ add_list (sublist 0 (i+1) il) ≤ Int.max_signed). {
  admit.
 }
 assert (0 ≤ Znth i il ≤ Int.max_signed). {
  replace (Znth i il) with (add_list (sublist i (i+1) il)).
  -
    admit.
  -
    admit.
}
```

Next: *Int.signed (Int.repr (add_list (sublist 0 i il))) = add_list (sublist 0 i il)*. To prove that, we'll use *Int.signed_repr*: `Check Int.signed_repr.`

```
rewrite Int.signed_repr by rep_lia.
```

rep_lia is just like *lia*, but it also knows the numeric values of representation-related constants such as Int.min_signed. `rewrite Int.signed_repr by rep_lia.`

```
rewrite (sublist_split 0 i (i+1)) in H4 by list_solve.
rewrite add_list_app in H4.
rewrite sublist_len_1 in H4 by list_solve.
simpl in H4.
rep_lia.
```

Admitted.

□

Exercise: 3 stars, standard (body_pop_and_add) `Lemma body_pop_and_add: semax_body Vprog Gprog f_pop_and_add pop_and_add_spec.`

```
Proof.
start_function.
forward.
forward.
forward_while (EX i:Z,
    PROP(0 ≤ i ≤ Zlength il)
    LOCAL (temp _st st;
           temp _i (Vint (Int.repr i));
           temp _n (Vint (Int.repr (Zlength il)));
          gvars gv)
    SEP (stack (sublist i (Zlength il) il) st; mem_mgr gv)).
```

+
admit.
+
entailer!.
+
forward_call (*st*, *Znth i il*, *sublist* (*i*+1) (*Zlength il*) *il*, *gv*).

This *forward_call* couldn't quite figure out the "Frame" for the function call. That is, it couldn't match up *stack* (*sublist i* (*Zlength il*) *il*) *st* with

stack (Znth i il :: sublist (i + 1) (Zlength il) il) st.

You have to help, by doing some rewrites with *sublist_split*, *sublist_len_1* that prove

sublist i (Zlength il) il = Znth i il :: sublist (i+1) (Zlength il) il.

When you've rewritten the goal into,

stack (Znth i il :: sublist (i + 1) (Zlength il) il) st |– stack (Znth i il :: sublist (i + 1) (Zlength il) il) st * fold_right_sepcon Frame

then just do *cancel*. *admit*.

And now we are ready to go forward through the C statement *_s* = *_s* + *_t*; *Fail forward*.

oops! we can't go *forward* through *_s* = *_s* + *_t*; because we forgot to mention *temp _s* in the loop invariant! Time to start over.

By the way, this statement *_s* = *_s* + *_t* is exactly where *forward* will ask you to prove a subgoal in which you can use lemma *add_another*. `Abort`.

Into this lemma, paste in the failed proof just above, but adjust the loop invariant: add a LOCAL assertion for *_s*. `Lemma` *body_pop_and_add*: *semax_body Vprog Gprog f_pop_and_add pop_and_add_spec*.
`Proof`.

Hint: choose the loop invariant for *temp _s* ??? in such a way that you can make use of Lemma add_another. *Admitted*.

□

Exercise: 3 stars, standard (body_main) `Lemma` *body_main*: *semax_body Vprog Gprog f_main main_spec*.
`Proof`.
start_function.

We assume that *triang.c* is linked with an implementation of malloc/free. That assumption is expressed by the *create_mem_mgr* axiom, which we can *sep_apply* here. On the other hand, if we want a complete verified system including libraries, then instead of importing *floyd.library* we would actually link with a malloc/free implementation, but that's beyond the scope of this chapter. *sep_apply* (*create_mem_mgr gv*).

You can see that this has produced the SEP conjunct *mem_mgr gv*, which is useful to satisfy the precondition of *newstack*, *push*, *pop*, etc. Now you can finish this proof.

Admitted.

□

Chapter 18

Library VC.Verif_append1

18.1 Verif_append1: List segments

Here is a little C program, *append.c*

```
Require VC.Preface.

Require Import VST.floyd.proofauto.
Require Import VC.append.
#[export] Instance CompSpecs : compspecs. make_compspecs prog. Defined.
Definition Vprog : varspecs. mk_varspecs prog. Defined.
```

18.1.1 Specification of the *append* function.

Here we just copy what we have defined in *Verif_reverse*

```
Definition t_list := Tstruct _list noattr.

Fixpoint listrep (sigma: list val) (p: val) : mpred :=
 match sigma with
 | h::hs ⇒
     EX y:val,
       data_at Tsh t_list (h,y) p × listrep hs y
 | nil ⇒
     !! (p = nullval) && emp
 end.

Arguments listrep sigma p : simpl never.
```

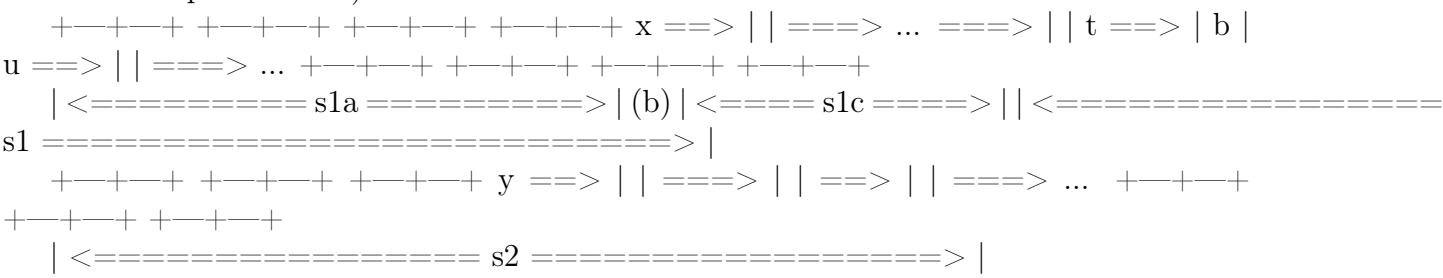
Then we can easily describe the functionality of this *append*.

```
Definition append_spec :=
 DECLARE _append
  WITH x: val, y: val, s1: list val, s2: list val
  PRE [ tptr t_list , tptr t_list]
```

```
    PROP()
    PARAMS (x; y)
    SEP (listrep s1 x; listrep s2 y)
  POST [ tptr t_list ]
    EX r: val,
    PROP()
    RETURN(r)
    SEP (listrep (s1++s2) r).
Definition Gprog : funspecs := [ append_spec ].
```

18.1.2 List segments.

When verifying this program, a critical step is to figure out a correct loop invariant. If we try to simulate this program, especially the loop in it, we may want a loop invariant which can be illustrated by the following diagram. (The following diagram is best demonstrated with a monospaced font.)



To describe this loop invariant, we need a separation logic predicate to describe the partial linked list from address x to address t with contents $s1a$. This must be a new predicate different from *listrep* because *listrep* describes linked lists ending with *NULL* which is not the case here. We call this new predicate *lseg*, pronounced "list-segment".

```
Fixpoint lseg (contents: list val) (x z: val) : mpred :=
  match contents with
  | nil ⇒ !! (x = z) && emp
  | h::hs ⇒ EX y:val, data_at Tsh t_list (h, y) x × lseg hs y z
  end.

Arguments lseg contents x z : simpl never.
```

Now, we can prove some useful properties about *lseg*.

Exercise: 1 star, standard (singleton_lseg) `Lemma singleton_lseg: ∀ (a: val) (x y: val),`
`  data_at Tsh t_list (a, y) x |– lseg [a] x y.`
`Proof.`
`  Admitted.`
□

It is critical to observe that a partial linked list defined by *lseg s x y* may have a loop. For example, the following diagram does satisfy *lseg* [*a*; *b*] *x y*. (The following diagram is best demonstrated with a monospaced font.)

+—+—+ +—+—+ x ==> | a | y ==> | b | y ====+ +—+—+ +- -+—+ | ^ | | | +==========+

We can prove this formally.

```
Lemma lseg_maybe_loop: ∀ (a b x y: val),
  data_at Tsh t_list (a, y) x × data_at Tsh t_list (b, y) y
  |– lseg [a; b] x y.
Proof.
  intros.
  unfold lseg.
  Exists y.
  Exists y.
  entailer!.
Qed.
```

Is our definition of *lseg* wrong? The answer is no because a loopy *lseg* cannot connect to a nonempty linked list. For instance, we can prove that

data_at Tsh t_list (a, y) x * data_at Tsh t_list (b, y) y * listrep *c* y

will lead to a contradication. Here, the first two separating conjuncts build a loopy *lseg* and the third separating conjunct is a nonempty *listrep*.

```
Lemma loopy_lseg_not_bad: ∀ (a b c x y: val),
  data_at Tsh t_list (a, y) x × data_at Tsh t_list (b, y) y × listrep [c] y
    |– FF.
Proof.
  intros.
  unfold listrep.
  Intros u.
  subst.
Check (data_at_conflict Tsh t_list (c, nullval)).
  sep_apply (data_at_conflict Tsh t_list (c, nullval)).
  + auto.
  + entailer!.
Qed.
```

Important note! The proof above demonstrates the use of the *sep_apply* tactic. Step through that part of the proof to see what *sep_apply* does.

Now we can prove the following theorems about partial linked lists and complete linked lists.

Exercise: 1 star, standard (lseg_lseg) `Lemma lseg_lseg: ∀ (s1 s2: list val) (x y z: val), lseg s1 x y × lseg s2 y z |– lseg (s1 ++ s2) x z.`

```
Proof.
    Admitted.
    □
```

Exercise: 1 star, standard (lseg_list) `Lemma lseg_list: ∀ (s1 s2: list val) (x y: val),`
`  lseg s1 x y × listrep s2 y |– listrep (s1 ++ s2) x.`

```
Proof.
    Admitted.
    □
```

Is it possible to define *lseg* in a different way so that loopy situations can be banned? Yes. We discuss this near the end of the chapter.

18.1.3 Proof of the *append* function

Before verifying the functional correctness of *append*, we still need to add lemmas to hint databases for separation logic predicates. Readers may copy proofs from *Verif_reverse* or just skip down to "End of the material".

```
Lemma listrep_local_facts:
  ∀ sigma p,
    listrep sigma p |–
    !! (is_pointer_or_null p ∧ (p=nullval ↔ sigma=nil)).
Proof.
    Admitted.
#[export] Hint Resolve listrep_local_facts : saturate_local.

Lemma listrep_valid_pointer:
  ∀ sigma p,
    listrep sigma p |– valid_pointer p.
Proof.
    Admitted.
#[export] Hint Resolve listrep_valid_pointer : valid_pointer.
```

(End of the material repeated from Verif_reverse.v)

In C programs, we test whether the head pointer of a linked list is null to determine whether that list is empty or not. Thus, from a separating conjunct *listrep contents x*, it is useful to prove *contents* = *nil* (or *contents* ≠ *nil*) when knowing that *x* = *nullval* (or *x* ≠ *nullval*). The following two lemmas state such correlation. They will be used several times in the C function *append*'s correctness proof.

Exercise: 1 star, standard (listrep_null) `Lemma listrep_null: ∀ contents x,`
`  x = nullval →`
`  listrep contents x = !! (contents=nil) && emp.`

```
Proof.
```

Hint: One way to prove $P=Q$, where P and Q are *mpred*s, is to apply *pred_ext* and then prove $P|-Q$ and $Q|-P$. *Admitted*.

□

Exercise: 1 star, standard (listrep_nonnull) `Lemma` *listrep_nonnull*: ∀ *contents x*,
x ≠ *nullval* →
listrep contents x =
EX h: *val*, *EX hs*: *list val*, *EX y*:*val*,
!! (*contents* = *h* :: *hs*) && *data_at Tsh t_list* (*h*, *y*) *x* × *listrep hs y*.
`Proof`.

Again, *pred_ext* will be useful here. *Admitted*.

□

Now, let's prove this *append* function correct.

Exercise: 3 stars, standard (body_append) `Lemma` *body_append*: *semax_body Vprog Gprog f_append append_spec*.
`Proof`.
start_function.
forward_if. -

This if-then branch handles the cases in which *x* is null. In other words, *s1* should be *nil*. We can easily derive this by *listrep_null*. The rest of the proof in this branch is left as an exercise. `rewrite` (*listrep_null _ x*) `by auto`.

admit.

-

This time, we know that *x* is not null; thus *s1* should be nonempty. `rewrite` (*listrep_nonnull _ x*) `by auto`.

Intros h r u.
forward. *forward*.

After symbolically executing two assignment commands, we arrive at the while loop. As mentioned above, we can verify it using the following loop invariant. *forward_while*

```
( EX s1a: list val, EX b: val, EX s1c: list val, EX t: val, EX u: val,
  PROP (s1 = s1a ++ b :: s1c)
  LOCAL (temp _x x; temp _t t; temp _u u; temp _y y)
  SEP (lseg s1a x t;
       data_at Tsh t_list (b, u) t;
       listrep s1c u;
       listrep s2 y))%assert.
+
  Exists (@nil val) h r x u.
  subst s1. entailer!. unfold lseg; entailer!.
+
  entailer!.
```

+

We know u is not null from the fact that the loop condition is true. Thus we can represent $s1c$ in the form of $(c :: s1d)$. `clear` h r u $H0$; `rename` $u0$ *into* u.

`rewrite` (*listrep_nonnull* _ u) `by auto`.

Intros c $s1d$ z.

forward. *forward*.

In the end of the loop body, we need to re-establish the loop invariant. At this point, the memory layout can be illustrated by the following diagram.

(The following diagram is best demonstrated with a monospaced font.)

new t new u | | | | +—+—+ +—+—+ +—+—+ +—+—+ x ==> ... ===> | | t ==> | b | u ==> | c | z ==> | | ===> ... +—+—+ +—+—+ +—+—+ +—+—+

| <====== s1a ======> | (b) (c) | <==== s1d ====> | | <========== new s1a =========> | (new b) | <== new s1c ==> |

+—+—+ +—+—+ +—+—+ y ==> | | ===> | | ==> | | ===> ... +—+—+ +—+—+ +—+—+ | <================== s2 ==================> |

Clearly, $s1a$ ++ b :: nil should be the new value of $s1a$; c should be the new value of b; $s1d$ should be the new value of $s1c$; u should be the new value of t; and z should be the new value of u. The next command instantiates the existentially quantified variables in our loop invariant accordingly.

Exists $((s1a$ ++ b :: $nil),c,s1d,u,z)$. `unfold` *fst*, *snd*.

As usual, we try *entailer*! to solve this proof goal. This time, *entailer*! does not solve it directly. Instead, two simplified proof goals are left. Their proofs are left for the reader, using *app_assoc*, *singleton_lseg* and *lseg_lseg*. *entailer*!.

× *admit*.

× *admit*.

+

After exiting the loop, the loop condition must be false, i.e. u is the null pointer. Thus $s1c = nil$ and $s1 = s1a$ ++ $[b]$. `clear` h r u $H0$; `rename` $u0$ *into* u.

`rewrite` (*listrep_null* $s1c$) `by auto`.

Intros.

`subst` $s1c$.

The rest of the proof is standard. Hint, *singleton_lseg*, *lseg_lseg* and/or *lseg_list* may be useful. *admit*.

Admitted.

□

18.1.4 Additional exercises: more proofs about list segments

For verifying the C function *append*, it is enough to have only three separation logic proof rules about *lseg*: *singleton_lseg*, *lseg_lseg* and *lseg_list*. The following exercises are other important properties of *lseg*.

Exercise: 1 star, standard: (lseg2listrep) `Lemma` *lseg2listrep*: ∀ *s x*,
  lseg s x nullval |– *listrep s x*.
`Proof`.
  Admitted.
  □

Exercise: 1 star, standard: (listrep2lseg) `Lemma` *listrep2lseg*: ∀ *s x*,
  listrep s x |– *lseg s x nullval*.
`Proof`.
  Admitted.
  □

`Corollary` *lseg_listrep_equiv*: ∀ *s x*,
  lseg s x nullval = *listrep s x*.
`Proof`.
  `intros`.
  `apply` *pred_ext*.
  + `apply` *lseg2listrep*.
  + `apply` *listrep2lseg*.
`Qed`.

Exercise: 2 stars, standard: (lseg_lseg_inv) `Lemma` *lseg_lseg_inv*: ∀ *s1 s2 x z*,
  lseg (*s1* ++ *s2*) *x z* |– *EX y*: *val*, *lseg s1 x y* × *lseg s2 y z*.
`Proof`.
  Admitted.
  □

Exercise: 2 stars, standard: (loopy_lseg_no_connection) `Lemma` *loopy_lseg_no_connection*:
∀ *s1 s2 x y z*,
  s1 ≠ *nil* →
  s2 ≠ *nil* →
  x = *y* →
  lseg s1 x y × *lseg s2 y z* |– *FF*.
`Proof`.
  Admitted.
  □

18.1.5 Additional exercises: loop-free list segments

In the following exercise, try to redo the proof above using a different partial-linked-list predicate.

We have mentioned that the *lseg* predicate allows a loopy structure, which is quite counterintuitive. Here is an alternate definition that prohibits loops:

```
Fixpoint nt_lseg (contents: list val) (x z: val) : mpred :=
  match contents with
  | nil ⇒ !! (x = z) && emp
  | h::hs ⇒ EX y:val, !! (x ≠ z)
                    && data_at Tsh t_list (h, y) x × nt_lseg hs y z
  end.

Arguments nt_lseg contents x z : simpl never.
```

Here, "nt" means no-touch.

The difference between *nt_lseg* and *lseg* is the extra proposition $x \neq z$ in the nonempty situation. This extra clause in *nt_lseg* prevents loop structures.

The proof theories of *nt_lseg* and *lseg* are a bit different as well. The following diagram shows that the counterpart of *lseg_lseg* is not valid!

+—+—+ +—+—+ +—+—+ +—+—+ x ==> | a | u ==> | b | y ==> | c | v ==> | d |
u ===+ +—+—+ +- -+—+ +—+—+ +—+—+ | ^ | | | +==========

In this example, both [*a*; *b*] and [*c*; *d*] are stored in loop-free partial linked lists but it is not true for their concatenation. In general, if (*nt_lseg s1 x y*) and (*nt_lseg s2 y z*) describe two loop-free partial linked lists, the assertion

(nt_lseg s1 x y * nt_lseg s2 y z)

cannot ensure that the structure is loop free. Specifically, the address *z* may be used in (*nt_lseg s1 x y*). In other words,

nt_lseg s1 x y * nt_lseg s2 y z |-/- nt_lseg (s1 ++ s2) x z

For *nt_lseg*, the following proof rules are useful.

Exercise: 2 stars, standard, optional (nt_lseg) `Lemma` *singleton_nt_lseg*: ∀ (*contents*: *list val*) (*a x y*: *val*),

```
  data_at Tsh t_list (a, y) x × listrep contents y |–
  nt_lseg [a] x y × listrep contents y.
Proof.
```

Admitted.

□

Exercise: 2 stars, standard, optional (singleton_nt_lseg') `Lemma` *singleton_nt_lseg'*: ∀ (*a b x y z*: *val*),

```
  data_at Tsh t_list (a, y) x × data_at Tsh t_list (b, z) y |–
  nt_lseg [a] x y × data_at Tsh t_list (b, z) y.
Proof.
```

Admitted.

□

Exercise: 2 stars, standard, optional (nt_lseg_nt_lseg) `Lemma` *nt_lseg_nt_lseg*: ∀ (*s1 s2*: *list val*) (*a x y z u*: *val*),

nt_lseg s1 x y × *nt_lseg s2 y z* × *data_at Tsh t_list* (*a*, *u*) *z* |–
nt_lseg (*s1* ++ *s2*) *x z* × *data_at Tsh t_list* (*a*, *u*) *z*.
Proof.

Hint: This lemma illustrates the most classic case where aggressive *cancel* can turn a provable goal into an unprovable goal. For that reason, you may need to use *entailer* rather than *entailer*! at one point. *Admitted*.

□

Exercise: 2 stars, standard, optional (nt_lseg_list) Lemma *nt_lseg_list*: ∀ (*s1 s2*: *list val*) (*x y*: *val*),
nt_lseg s1 x y × *listrep s2 y* |– *listrep* (*s1* ++ *s2*) *x*.
Proof.

Admitted.

□

Now, we will use *nt_lseg* instead of *lseg* in the loop invariant to prove *body_append*.

Exercise: 3 stars, standard, optional (body_append_alter1) Lemma *body_append_alter1*: *semax_body Vprog Gprog f_append append_spec*.
Proof.
start_function.
forward_if. -
rewrite (*listrep_null* _ *x*) by auto.
admit.
-
rewrite (*listrep_nonnull* _ *x*) by auto.
Intros h r u.
forward. *forward*. Now use *forward_while* to verify this while loop. Remember, *forward_while* will generate four proof goals: current precondition implies loop invariant; loop test is safe to execute; loop body preserves invariant; and the correctness of after-loop commands. *Admitted*.

□

Chapter 19

Library VC.Verif_append2

19.1 Verif_append2: Magic wand, partial data structure

19.1.1 Separating Implication

Separating implication is another separation logic operator. It is written as -* in Verifiable C. Because of its shape, it is usually called "magic wand". The following `Locate` command and `Check` command show this notation definition and its typing information.

```
Require VC.Preface.
Require Import VST.floyd.proofauto.
Locate "-*".
Check wand.
```

In separation logic, a heaplet (piece of memory) m satisfies P -* Q if and only if: for any possible heaplet n, if n and m are disjoint and n satisfies P, then the combination of n and m will satisfy Q.

The most important proof rule for separating implication is the adjoint property. It says, $P \times Q$ derives R if and only if P derives Q-*R. This rule is called *wand_sepcon_adjoint* in Verifiable C.

```
Check wand_sepcon_adjoint.
```

Because of this property, we also call -* a right adjoint of $\times$. In propositional logic, implication $\rightarrow$ is a right adjoint of conjunction $\wedge$.

```
Lemma implies_and_adjoint:
  ∀ P Q R : Prop, (P ∧ Q → R) ↔ (P → (Q → R)).
Proof. intuition. Qed.
```

This intrinsic similarity gives -* the name "separating implication". The following are two other important properties of -*; we can easily find their counterparts about propositional-logic "implication".

Proof rules for separating implication:

```
Check wand_derives.

Check modus_ponens_wand.
```

Now, we learn to use the adjoint property to prove other separation-logic rules about -*. We will start from an easy one.

```
Lemma wand_trivial: ∀ P Q: mpred, P |– Q -* (P × Q).
Proof.
  intros.
  rewrite ← wand_sepcon_adjoint.
  apply derives_refl.
Qed.
```

Then, we will reprove the modus ponens rule for -* and × from the adjoint property.

```
Lemma modus_ponens_wand_from_adjoint: ∀ P Q : mpred, P × (P -* Q) |– Q.
Proof.
  intros.
  rewrite sepcon_comm.
  rewrite → wand_sepcon_adjoint.
  apply derives_refl.
Qed.
```

Now prove *wand_derives* using *wand_sepcon_adjoint* and *modus_ponens_wand*. You can use other proof rules about ×, such as *sepcon_derives*. Also, the tactic *sep_apply* may be useful.

Exercise: 2 stars, standard: (wand_derives) `Lemma wand_derives_from_adjoint_and_modus_ponens`

```
  ∀ P P' Q Q' : mpred,
    P' |– P → Q |– Q' → P -* Q |– P' -* Q'.
Proof.
    Admitted.
```

□

Theorem *wand_frame_ver* is the counterpart of implication's transitivity. As we will see, it allows "vertical composition" of wand frames.

```
Check wand_frame_ver.
```

Prove it by *wand_sepcon_adjoint* and *sep_apply* (*modus_ponens_wand* ...)

Exercise: 2 stars, standard: (wand_frame_ver) `Lemma wand_frame_ver_from_adjoint_and_modus_p`

```
  ∀ P Q R : mpred, (P -* Q) × (Q -* R) |– P -* R.
Proof.
    Admitted.
```

□

More exercises: prove that *emp* -* *emp* and *emp* are equivalent.

Exercise: 3 stars, standard: (emp_wand_emp) `Lemma emp_wand_emp_right: emp |– emp -* emp.`

```
Proof.
  Admitted.

Lemma emp_wand_emp_left: emp -* emp |– emp.
Proof.
  Admitted.
```

□

19.1.2 List segments by magic wand

```
Require Import VC.append.
Require Import VC.Verif_append1.
```

In *Verif_append1*, we recursively defined a new separation logic predicate: list segment. That predicate describes a heaplet that contains a partial linked list.

In this chapter, we learn a different way of describing partial data structures–we use magic wand together with quantifiers.

This is a natural idea. Using linked lists as an example, adding a linked list to the tail of a partial linked list (or a list segment) will result in a complete linked list from the head. Thus, a partial linked list can be described by "the added list -* the complete list". Formally:

```
Definition wlseg (contents: list val) (x y: val) : mpred :=
  ALL tail: list val, listrep tail y -* listrep (contents ++ tail) x.
```

Here, "w" in "wlseg" represents "wand".

This definition is very different from *lseg* and is beautifully simple, and it generalizes nicely to other data structures such as trees.

Let's prove some basic properties of *wlseg*. The following lemmas show how a wand expression can be introduced (*emp_wlseg* and *singleton_wlseg*), how a wand expression can be eliminated (*wlseg_list*) and how two wand expressions can merge (*wlseg_wlseg*).

There are two logical operators in this definition, -* and the universal quantifier. Previously, we have learned how to prove properties about × and -*. To prove properties about universal quantifiers, we will use *allp_left* and *allp_right*.

```
Check allp_left.
Check allp_right.
```

The first property of *wlseg* is that we can introduce *wlseg* from *emp*.

```
Lemma emp_wlseg: ∀ (x: val),
  emp |– wlseg [] x x.
Proof.
  intros.
  unfold wlseg.
  apply allp_right; intro tail.
```

```
  rewrite ← wand_sepcon_adjoint.
  rewrite emp_sepcon.
  simpl app.
  apply derives_refl.
Qed.
```

Next, we show that two *wlseg* predicates can be merged into one.

```
Lemma wlseg_wlseg: ∀ (s1 s2: list val) (x y z: val),
  wlseg s2 y z × wlseg s1 x y |– wlseg (s1 ++ s2) x z.
Proof.
 intros.
 unfold wlseg.
```

First, extract the universally quantified variable *tail* on the right side. `apply allp_right; intro tail.`

Next, instantiate the first quantified *tail0* on the left with *tail*. `rewrite → wand_sepcon_adjoint.`

```
 apply (allp_left _ tail).
 rewrite ← wand_sepcon_adjoint.
```

Then, instantiate the other quantified *tail0* on the left with *s2 ++ tail*. `rewrite sepcon_comm, → wand_sepcon_adjoint.`

```
 apply (allp_left _ (s2 ++ tail)).
 rewrite ← wand_sepcon_adjoint, sepcon_comm.
```

Finally, complete the proof with wand_frame_ver. `rewrite ← app_assoc.`

```
 apply wand_frame_ver.
Qed.
```

This theorem *wlseg_wlseg* shares the same form with *lseg_lseg*. In fact, properties about *lseg* and *wlseg* are very similar. The following exercises are to prove the counterparts of *singleton_lseg* and *lseg_list*.

Exercise: 2 stars, standard: (singleton_wlseg) `Lemma singleton_wlseg: ∀ (a: val) (x y: val),`

```
  data_at Tsh t_list (a, y) x |– wlseg [a] x y.
Proof.
```

Admitted.

□

Exercise: 2 stars, standard: (wlseg_list) `Lemma wlseg_list: ∀ (s1 s2: list val) (x y: val),`

```
  wlseg s1 x y × listrep s2 y |– listrep (s1 ++ s2) x.
Proof.
```

Admitted.

□

19.1.3 Proof of the *append* function by *wlseg*

Now, we are ready to reprove the correctness for the C program *append*. This time, we will use *wlseg* to write the loop invariant.

Exercise: 3 stars, standard: (body_append_alter2) `Lemma` *body_append_alter2*: *semax_body Vprog Gprog f_append append_spec*.
`Proof`.
start_function.
forward_if. -
`rewrite` (*listrep_null _ x*) `by auto`.
admit.
-
`rewrite` (*listrep_nonnull _ x*) `by auto`.
Intros h r u.
forward. *forward*. Here, we use *wlseg* to represent a list segment. *forward_while*
(*EX s1a*: *list val*, *EX b*: *val*, *EX s1c*: *list val*, *EX t*: *val*, *EX u*: *val*,
PROP (*s1 = s1a ++ b :: s1c*)
LOCAL (*temp _x x*; *temp _t t*; *temp _u u*; *temp _y y*)
SEP (*wlseg s1a x t*;
data_at Tsh t_list (*b*, *u*) *t*;
listrep s1c u;
listrep s2 y))%`assert`.
+
To derive a loop invariant from the current assertion, the key point is to introduce *wlseg*. You may find *emp_wlseg* helpful here. *admit*.
+
entailer!.
+
Step forward through the loop body; along the way you'll need to do other transformations on the current assertion, to uncover opportunities to step *forward*. At the end of the loop body, you need to prove that a list segment for *s1a* and a singleton cell for *b* forms a longer list segment, whose contents is *s1a ++ b :: nil*. You may find *singleton_wlseg* and *wlseg_wlseg* useful there. *admit*.
+
After you symbolicly execute the return command, you need to establish one single linked list with contents *s1a ++ b :: s2* from a list segment for *s1a*, a singleton cell for *b* and another linked list for *s2*. You may find *singleton_wlseg* and *wlseg_list* useful there. *admit*.
Admitted.
□

19.1.4 The general idea: magic wand as frame

Let's review the proof script above. Before the loop, we first derive *wlseg* [] *x x* from *emp*. After every iteration of the loop body, we merge a piece of singleton list segment *wlseg* [*b*] *t u* into it. When exiting the loop, we get *wlseg* [*s1a*] *x t* where *s1* = *s1a* ++ [*b*]. Eventually, this list segment is merged with a tail *listrep* ([*b*] ++ *s2*) *t*, which results in *listrep* (*s1* ++ *s2*) *x*.

From where the list segment is introduced in the proof, to where the list segment is eliminated by merging, the C program never modifies the submemory described by that *wlseg* predicate. In separation logic, such a separating conjunct is called a "frame". Thus, the general idea here is to use a wand expression to describe a partial data structure and this wand expression will act as a frame in program verification.

In the proof of *body_append_alter2*, we only need four of *wlseg*'s properties: *emp_wlseg*, *singleton_wlseg*, *wlseg_wlseg* and *wlseg_list*. They are used to introduce, merge and eliminate *wlseg* predicates. Here are some general patterns beyond these specific rules.

```
Lemma wandQ_frame_elim_mpred: ∀ {A: Type} (P Q: A → mpred) (a: A),
  (ALL x : A, P x -* Q x) × P a |– Q a.
Proof.
  intros.
  rewrite → wand_sepcon_adjoint.
  apply (allp_left _ a).
  apply derives_refl.
Qed.
```

"ver" in the name of the next lemma stands for "vertical composition" of wand frames. One wand-frame is nested inside another.

```
Lemma wandQ_frame_ver_mpred: ∀ {A: Type} (P Q R: A → mpred),
  (ALL x : A, P x -* Q x) × (ALL x: A, Q x -* R x) |– ALL x: A, P x -* R x.
Proof.
  intros.
  apply allp_right; intro a.
  rewrite → wand_sepcon_adjoint.
  apply (allp_left _ a).
  rewrite ← wand_sepcon_adjoint.
  rewrite sepcon_comm, → wand_sepcon_adjoint.
  apply (allp_left _ a).
  rewrite ← wand_sepcon_adjoint, sepcon_comm.
  apply wand_frame_ver.
Qed.
```

19.1.5 Case study: list segments for linked list box

In the following exercise, you are going to apply the magic-wand-as-frame method on a slightly different data structure.

Consider the following C function, *append2*.

struct list * append2 (struct list * x, struct list * y) { struct list **retp, **curp; retp = & x; curp = & x; while (*curp != NULL) { curp = & ((*curp) -> tail); } *curp = y; return *retp; }

In comparison, this is *append*.

struct list *append (struct list *x, struct list *y) { struct list *t, *u; if (x==NULL) return y; else { t = x; u = t->tail; while (u!=NULL) { t = u; u = t->tail; } t->tail = y; return x; } }

In *append*, u always equals $t \to tail$ after every iteration. When exiting the loop, the value of u is always null; that is not important. More important is the address from whence the null value is loaded. A new value will be stored into that location in memory. The program variable t is used to remember that address.

The C function *append2* implements linked-list append in an alternative way. In this function, *curp*'s value is not an address in the linked list. Instead, it records where a linked list address is stored in memory. Specifically, when *curp* points the head pointer x, the value of *curp* is the address of x. When *curp* points to some intermediate linked list node, the value of *curp* is the predecessor node's *tail* field address. Using this implementation, we do not need to test whether x is null in the beginning.

The following separation logic predicate defines this data structure.

```
Definition t_list_box := tptr t_list.

Definition listboxrep (contents: list val) (x: val) :=
  EX y: val, data_at Tsh t_list_box y x × listrep contents x.

Definition lbseg (contents: list val) (x y: val) :=
  ALL tail: list val, listboxrep tail y -* listboxrep (contents ++ tail) x.
```

Previously, we have shown that we can introduce, eliminate and merge wand expressions by proving *emp_wlseg*, *singleton_wlseg*, *wlseg_list* and *wlseg_wlseg*. Now, your task is to prove *lbseg*'s properties. Hint: proving *wlseg*'s properties and proving *lbseg*'s properties should be very similar.

Exercise: 1 star, standard: (emp_lbseg) Introducing a wand expression, *lbseg*, from *emp*.

```
Lemma emp_lbseg: ∀ (x: val),
  emp |– lbseg [] x x.
Proof.
  Admitted.
```

□

Exercise: 2 stars, standard: (lbseg_lbseg) Merging two wand expressions.

```
Lemma lbseg_lbseg: ∀ (s1 s2: list val) (x y z: val),
  lbseg s2 y z × lbseg s1 x y |– lbseg (s1 ++ s2) x z.
Proof.
  Admitted.
```

□

Exercise: 2 stars, standard: (listbox_lbseg) Eliminating a wand expression. Lemma *listbox_lbseg*: ∀ (*s1 s2*: *list val*) (*x y*: *val*),
lbseg s1 x y × *listboxrep s2 y* |– *listboxrep* (*s1* ++ *s2*) *x*.
Proof.
Admitted.
□

19.1.6 Comparison and connection: *lseg* vs. *wlseg*

We have demonstrated two different approaches to define a separation logic predicate for list segments. In *Verif_append1* we define it using recursive definition over the list. In this chapter, we use a quantifed magic wand expression. It is natural to ask: what is the relation between *lseg* and *wlseg*? Are they equivalent to each other? They following theorems can offer a brief answer.

First of all, recursive defined *lseg* is a logically stronger predicate than *wlseg*. Lemma *lseg2wlseg*: ∀ *s x y*, *lseg s x y* |– *wlseg s x y*.
Proof.
intros.
unfold *wlseg*.
apply *allp_right*; intros *tail*.
rewrite ← *wand_sepcon_adjoint*.
sep_apply (*lseg_list s tail x y*).
apply *derives_refl*.
Qed.

In some special cases, *wlseg* derives *lseg* as well. Lemma *wlseg2lseg_nullval*: ∀ *s x*, *wlseg s x nullval* |– *lseg s x nullval*.
Proof.
intros.
unfold *wlseg*.
apply *allp_left* with (@*nil val*).
unfold *listrep* at 1.
rewrite *prop_true_andp* by auto.
entailer!.
rewrite *app_nil_r*.

The proof goal now has the form: a wand expression derives some wand-free assertion. Usually, this is a tough task because there is no good way to eliminate magic wand on left side. But this proof goal is special. We can add an extra separating conjunct *emp* to the left side and use *modus_ponens_wand* to eliminate wand. rewrite ← (*emp_sepcon* (*emp* -* *listrep s x*)).
sep_apply (*modus_ponens_wand emp* (*listrep s x*)).

Then, easy! `apply` *listrep2lseg*.
`Qed`.

Combining these two lemmas above together, we know that *wlseg*-to-null equals *lseg*.

```
Lemma wlseg_nullval: ∀ s x, wlseg s x nullval = lseg s x nullval.
Proof.
  intros.
  apply pred_ext.
  + apply wlseg2lseg_nullval.
  + apply lseg2wlseg.
Qed.

Corollary wlseg_listrep_equiv: ∀ s x, wlseg s x nullval = listrep s x.
Proof.
  intros.
  rewrite wlseg_nullval, lseg_listrep_equiv.
  reflexivity.
Qed.
```

However, *wlseg* does not derive *lseg* in general. As mentioned above, to eliminate magic wand on the left side is hard. When $y \neq nullval$, we cannot instantiate the universally quantified variable *tail* inside (*wlseg s x y*) to get the form *emp* -* _. The following is a counterexample of the general entailment from *wlseg* to *lseg*. On one hand, it is obvious that *data_at_ Tsh t_list y* |-/- *lseg* [*a*] *x y*. On the other hand, *data_at_ Tsh t_list y* |– *wlseg* [*a*] *x y*. See the following theorem:

```
Lemma wlseg_weird: ∀ a x y,
  data_at_ Tsh t_list y |– wlseg [a] x y.
Proof.
  intros.
  unfold wlseg.
  apply allp_right; intros s.
  rewrite ← wand_sepcon_adjoint.
  destruct s.
  + unfold listrep at 1.
    entailer!.
    destruct H as [H _].
    contradiction.
  + unfold listrep at 1; fold listrep.
    Intros u.
    sep_apply (data_at_conflict Tsh t_list (default_val t_list) (v, u) y); auto.
    entailer!.
Qed.
```

Chapter 20

Library VC.Verif_strlib

20.1 Verif_strlib: String functions

In this chapter we show how to prove the correctness of C programs that use null-terminated character strings.

Here are some functions from the C standard library, *strlib.c*

20.2 Standard boilerplate

```
Require VC.Preface. Require Import VST.floyd.proofauto.
Require Import VC.strlib.
#[export] Instance CompSpecs : compspecs. make_compspecs prog. Defined.
Definition Vprog : varspecs. mk_varspecs prog. Defined.
Require Import VC.hints. Require Import Stdlib.Strings.Ascii.
```

20.3 Representation of null-terminated strings.

Rocq represents a string as a list-like Inductive of Ascii characters: `Locate string. Print string.`

The C programming language represents a *character* as a byte, that is, an 8-bit signed or unsigned integer. In Rocq represent the 8-bit integers using the *byte* type. `Print byte.`

`Search byte.`

We can convert a Rocq string to a list of bytes:

```
Fixpoint string_to_list_byte (s: string) : list byte :=
  match s with
  | EmptyString ⇒ nil
  | String a s' ⇒ Byte.repr (Z.of_N (Ascii.N_of_ascii a))
                                        :: string_to_list_byte s'
```

```
  end.
Definition Hello := "Hello"%string.
Definition Hello' := string_to_list_byte Hello.

Eval simpl in string_to_list_byte Hello.

Section StringDemo.

  Variable p : val.
```

To describe a single byte in memory, we can use *data_at* with the signed-character type: `Print tschar. Check (data_at Tsh tschar (Vint (Int.repr 72)) p).`

This *data_at Tsh tschar (Vint (Int.repr 72)) p* is an *mpred*, that is, a memory predicate in separation logic. It says, at address *p* in memory there is a sequence of bytes whose length is appropriate for type *tschar*; that is, one byte. The contents of this sequence of bytes (one byte) is a representation of the integer 72. The ownership share (access permission) of memory at address *p* is the "top share" *Tsh*

```
Check (data_at Tsh tschar (Vbyte (Byte.repr 72))).
```

We can express the same thing using *Vbyte (Byte.repr 72)*. In fact, *Vbyte* is not a primitive CompCert value, it is a definition: `Print Vbyte.`

```
Goal Vbyte (Byte.repr 72) = Vint (Int.repr 72).
Proof. reflexivity. Qed.

Goal Vbyte (Byte.repr 72) = Vint (Int.repr 72).
Proof.
  unfold Vbyte.
  rewrite Byte.signed_repr.
  auto.
  rep_lia.
Qed.
```

The C programming language represents a string of length *k* as an array of nonnull characters (bytes), terminated by a null character. We represent this in separation logic using the *cstring* memory-predicate:

`Locate cstring. Print cstring.`

Here is an example of a *cstring* predicate that says, At address *p* there is a null-terminated string representing "Hello".

```
Check (cstring Tsh Hello' p).
```

By unfolding the definition of *cstring* this is equivalent to,

```
Check (
  !! (~ In Byte.zero Hello') &&
  data_at Tsh (tarray tschar (5 + 1)) (map Vbyte (Hello'++[Byte.zero])) p ).
```

This says, no element of the list *Hello'* is equal to *Byte.zero*. In memory at address *p* there is an array of 6 bytes, whose contents are the contents of *Hello'* with a *Byte.zero* appended at the end.

Sometimes we know that there is a null-terminated string inside an array of length n. That is, there are k nonnull characters (where $k<n$), followed by a null character, followed by n-(k+1) uninitialized (or don't-care) characters. We represent this with the *cstringn* predicate. `Print` *cstringn*.

`Check` (*cstringn Tsh Hello'* 10 *p*).

`End` *StringDemo*.

20.4 Reasoning about the contents of C strings

In separation logic proofs about C strings, we often find proof goals similar to the one exemplified by this lemma: `Lemma` *demonstrate_cstring1*:

```
 ∀ i contents
   (H: ¬ In Byte.zero contents)
   (H0: Znth i (contents ++ [Byte.zero]) ≠ Byte.zero)
   (H1: 0 ≤ i ≤ Zlength contents),
   0 ≤ i + 1 < Zlength (contents ++ [Byte.zero]).
Proof.
intros.
```

A null-terminated string is an array of characters with three parts:

- The contents of the string, none of which is the '\0' character;
- The null termination character, equal to Byte.zero;
- the remaining garbage in the array, after the null.

When processing a string, you should maintain three kinds of assumptions above the line:

- Hypothesis *H* above the line says that none of the

contents is the null character;

- Hypothesis *H0* typically comes from a loop test, *s*[*i*]!=0
- *H1* typically comes from a loop invariant: suppose a

a loop iteration variable *_i* (with value *i*) is traversing the array. We expect that loop to go up to but no farther than the null character, that is, one past the contents.

The *cstring* tactic processes all three of these hypotheses to conclude that $i <$ *Zlength contents*. `assert` (*H7*: $i <$ *Zlength contents*) `by` *cstring*.

But actually, *cstring* tactic will prove any rep_lia consequence of that fact. For example:

```
clear H7.
autorewrite with sublist.
```

```
cstring.
Qed.
```

Here is another demonstration. When your loop on the string contents reaches the end, the loop test $s[i]!=0$ is false, so therefore $s[i]=0$.

```
Lemma demonstrate_cstring2:
 ∀ i contents
    (H: ¬ In Byte.zero contents)
    (H0: Znth i (contents ++ [Byte.zero]) = Byte.zero)
    (H1: 0 ≤ i ≤ Zlength contents),
    i = Zlength contents.
Proof.
intros.
```

Hypothesis *H0* expresses that the loop test determined $s[i]=0$. The *cstring* can then prove that $i = Zlength\ contents$.

```
  cstring.
Qed.
```

20.5 Function specs

strlen(*s*) returns the length of the string *s*.

```
Definition strlen_spec :=
 DECLARE _strlen
  WITH sh: share, s : list byte, str: val
  PRE [ tptr tschar ]
    PROP (readable_share sh)
    PARAMS (str)
    SEP (cstring sh s str)
  POST [ size_t ]
    PROP ()
    RETURN (Vptrofs (Ptrofs.repr (Zlength s)))
    SEP (cstring sh s str).
```

20.5.1 A digression about size_t

Vptrofs? Ptrofs.repr? What's that?

Programmers use *size_t* when writing C programs that are portable to 64-bit and 32-bit installations; this typedef stands for whatever size of unsigned (long) integer is the same size as a pointer. In Verifiable C, size_t is the corresponding C unsigned type:

```
Print size_t.
Print Vptrofs.
Search Ptrofs.int Int64.int.
Search Ptrofs.int Int.int.
```

VST-Floyd has proof automation tactics that try to help you by applying these lemmas where appropriate. For example, in the proofs in this chapter, you don't have to do much special to deal with the Vptrofs.

End of digression about size_t

strcpy(*dest*,*src*) copies the string *src* to the array *dest*. `Definition` *strcpy_spec* :=
DECLARE _strcpy
WITH wsh: *share*, *rsh*: *share*, *dest* : *val*, *n* : *Z*, *src* : *val*, *s* : *list byte*
PRE [*tptr tschar*, *tptr tschar*]
PROP (*writable_share wsh*; *readable_share rsh*; *Zlength s* < *n*)
PARAMS (*dest*; *src*)
SEP (*data_at_ wsh* (*tarray tschar n*) *dest*; *cstring rsh s src*)
POST [*tptr tschar*]
PROP ()
RETURN (*dest*)
SEP (*cstringn wsh s n dest*; *cstring rsh s src*).

strcmp(*s1*,*s2*) compares strings s1 and s2 for lexicographic order. This funspec is an underspecification of the actual behavior, in that it specifies equality testing only. `Definition` *strcmp_spec* :=
DECLARE _strcmp
WITH sh1: *share*, *sh2*: *share*, *str1* : *val*, *s1* : *list byte*, *str2* : *val*,
s2 : *list byte*
PRE [*tptr tschar*, *tptr tschar*]
PROP (*readable_share sh1*; *readable_share sh2*)
PARAMS (*str1*; *str2*)
SEP (*cstring sh1 s1 str1*; *cstring sh2 s2 str2*)
POST [*tint*]
EX i : *int*,
PROP (`if` *Int.eq_dec i Int.zero* `then` *s1* = *s2* `else` *s1* ≠ *s2*)
RETURN (*Vint i*)
SEP (*cstring sh1 s1 str1*; *cstring sh2 s2 str2*).

`Definition` *Gprog* : *funspecs* := [*strlen_spec*; *strcpy_spec*; *strcmp_spec*].

20.6 Proof of the *strlen* function

Exercise: 2 stars, standard (body_strlen) `Lemma` *body_strlen*: *semax_body Vprog Gprog f_strlen strlen_spec*.
`Proof`.
start_function.
Look at the proof goal below the line. We have the assertion,
PROP ( ) LOCAL (temp

Chapter 21

Library VC.Hashfun

21.1 Hashfun: Functional model of hash tables

This C program, *hash.c*, implements a hash table with external chaining. See {https://www.cs.princeton.edu
for an introduction to hash tables.

/* First, access a few standard-library functions */

21.1.1 A functional model

Before we prove the C program correct, we write a functional program that models its behavior as closely as possible. The functional program won't be (average) constant time per access, like the C program, because it takes linear time to get the nth element of a list, while the C program can subscript an array in constant time. But we are not worried about the execution time of the functional program; only that it serve as a model for specifying the C program.

```
Require Import VST.floyd.functional_base.

#[export] Instance EqDec_string: EqDec (list byte) := list_eq_dec Byte.eq_dec.

Fixpoint hashfun_aux (h: Z) (s: list byte) : Z :=
 match s with
 | nil ⇒ h
 | c :: s' ⇒
       hashfun_aux ((h × 65599 + Byte.signed c) mod Int.modulus) s'
end.

Definition hashfun (s: list byte) := hashfun_aux 0 s.

Definition hashtable_contents := list (list (list byte × Z)).

Definition N := 109.
Lemma N_eq : N = 109.
Proof. reflexivity. Qed.
#[export] Hint Rewrite N_eq : rep_lia.
```

```
Global Opaque N.

Definition empty_table : hashtable_contents :=
  Zrepeat nil N.

Fixpoint list_get (s: list byte) (al: list (list byte × Z)) : Z :=
  match al with
 | (k,i) :: al' ⇒ if eq_dec s k then i else list_get s al'
 | nil ⇒ 0
 end.

Fixpoint list_incr (s: list byte) (al: list (list byte × Z))
                : list (list byte × Z) :=
  match al with
 | (k,i) :: al' ⇒ if eq_dec s k
                         then (k, i +1)::al'
                         else (k,i)::list_incr s al'
 | nil ⇒ (s, 1)::nil
 end.

Definition hashtable_get (s: list byte) (contents: hashtable_contents) : Z :=
  list_get s (Znth (hashfun s mod (Zlength contents)) contents).

Definition hashtable_incr (s: list byte) (contents: hashtable_contents)
                         : hashtable_contents :=
  let h := hashfun s mod (Zlength contents)
  in let al := Znth h contents
  in upd_Znth h contents (list_incr s al).
```

Exercise: 2 stars, standard (hashfun_inrange) `Lemma hashfun_inrange: ∀ s, 0 ≤ hashfun s ≤ Int.max_unsigned.`

```
Proof.
    Admitted.
```

□

Exercise: 1 star, standard (hashfun_get_unfold) `Lemma hashtable_get_unfold:`

```
 ∀ sigma (cts: list (list (list byte × Z) × val)),
  hashtable_get sigma (map fst cts) =
    list_get sigma (Znth (hashfun sigma mod (Zlength cts)) (map fst cts)).
Proof.
    Admitted.
```

□

Exercise: 2 stars, standard (Zlength_hashtable_incr) `Lemma Zlength_hashtable_incr:`

```
 ∀ sigma cts,
        0 < Zlength cts →
```

```
      Zlength (hashtable_incr sigma cts) = Zlength cts.
Proof.
  Admitted.
#[export] Hint Rewrite Zlength_hashtable_incr using list_solve : sublist.
  □
```

Exercise: 3 stars, standard (hashfun_snoc) `Lemma Int_repr_eq_mod:`
`∀ a, Int.repr (a mod Int.modulus) = Int.repr a.`

```
Proof.
Print Int.eqm. Search Int.eqm. Admitted.
```

Use *Int_repr_eq_mod* in the proof of *hashfun_aux_snoc*.

```
Lemma hashfun_aux_snoc:
  ∀ sigma h lo i,
    0 ≤ lo →
    lo ≤ i < Zlength sigma →
  Int.repr (hashfun_aux h (sublist lo (i + 1) sigma)) =
  Int.repr (hashfun_aux h (sublist lo i sigma) × 65599
                                    + Byte.signed (Znth i sigma)).
Proof.
  Admitted.

Lemma hashfun_snoc:
  ∀ sigma i,
    0 ≤ i < Zlength sigma →
  Int.repr (hashfun (sublist 0 (i + 1) sigma)) =
  Int.repr (hashfun (sublist 0 i sigma) × 65599 + Byte.signed (Znth i sigma)).
Proof.
  Admitted.
  □
```

21.1.2 Functional model satisfies the high-level specification

The purpose of a hash table is to implement a finite mapping, (a finite function) from keys to values. We claim that the functional model (*empty_table*, *hashtable_get*, *hashtable_incr*) correctly implements the appropriate operations on the abstract data type of finite functions.

We formalize that statement by defining a Module Type:

```
Module Type COUNT_TABLE.
 Parameter table: Type.
 Parameter key : Type.
 Parameter empty: table.
 Parameter get: key → table → Z.
 Parameter incr: key → table → table.
```

```
 Axiom gempty: ∀ k,
       get k empty = 0.
 Axiom gss: ∀ k t,
      get k (incr k t) = 1+(get k t).
 Axiom gso: ∀ j k t,
      j ≠ k → get j (incr k t) = get j t.
End COUNT_TABLE.
```

This means: in any `Module` that satisfies this `Module Type`, there's a type *table* of count-tables, and operators *empty*, *get*, `set` that satisfy the axioms *gempty*, *gss*, and *gso*.

A "reference" implementation of COUNT_TABLE

Exercise: 2 stars, standard (FunTable) It's easy to make a slow implementation of *COUNT_TABLE*, using functions.

```
Module FunTable <: COUNT_TABLE.
 Definition table: Type := nat → Z.
 Definition key : Type := nat.
 Definition empty: table := fun k ⇒ 0.
 Definition get (k: key) (t: table) : Z := t k.
 Definition incr (k: key) (t: table) : table :=
    fun k' ⇒ if Nat.eqb k' k then 1 + t k' else t k'.
 Lemma gempty: ∀ k, get k empty = 0.
   Admitted.
 Lemma gss: ∀ k t, get k (incr k t) = 1+(get k t).
   Admitted.
 Lemma gso: ∀ j k t, j ≠ k → get j (incr k t) = get j t.
   Admitted.
End FunTable.
```

□

Demonstration that hash tables implement COUNT_TABLE

Exercise: 3 stars, standard (IntHashTable) Now we make a "fast" implementation using hash tables. We put "fast" in quotes because, unlike the imperative implementation, the purely functional implementation takes linear time, not constant time, to select the the i'th bucket. That is, *Znth i al* takes time proportional to *i*. But that is no problem, because we are not using *hashtable_get* and *hashtable_incr* as our real implementation; they are serving as the *functional model* of the fast implementation in C.

```
Module IntHashTable <: COUNT_TABLE.
 Definition hashtable_invariant (cts: hashtable_contents) : Prop :=
  Zlength cts = N ∧
  ∀ i, 0 ≤ i < N →
```

```
                list_norepet (map fst (Znth i cts))
                ∧ Forall (fun s ⇒ hashfun s mod N = i) (map fst (Znth i cts)).
 Definition table := sig hashtable_invariant.
 Definition key := list byte.

 Lemma empty_invariant: hashtable_invariant empty_table.
 Proof.
   Admitted.

Lemma incr_invariant: ∀ k cts,
      hashtable_invariant cts → hashtable_invariant (hashtable_incr k cts).
Proof.
   Admitted.

 Definition empty : table := exist _ _ empty_invariant.
 Definition get : key → table → Z :=
                   fun k tbl ⇒ hashtable_get k (proj1_sig tbl).
 Definition incr : key → table → table :=
       fun k tbl ⇒ exist _ _ (incr_invariant k _ (proj2_sig tbl)).

 Theorem gempty: ∀ k, get k empty = 0.
 Proof.
   Admitted.

 Theorem gss: ∀ k t, get k (incr k t) = 1 + (get k t).
 Proof.
   Admitted.

 Theorem gso: ∀ j k t,
      j ≠ k → get j (incr k t) = get j t.
Proof.
   Admitted.
   □

End IntHashTable.
```

Chapter 22

Library VC.Verif_hash

22.1 Verif_hash: Correctness proof of hash.c

In this chapter we will prove that the C program, hash.c, correctly implements the functional model in hashfun.v. That proof, composed with the proof in hashfun.v that the functional model behaves correctly as a key-value map with string keys, demonstrates the correct behavior of hash.c.

The usual boilerplate `Require VC.Preface. Require Import VST.floyd.proofauto.`
`Require Import VST.floyd.library.`
`Require Import VC.hash.`
`#[export] Instance CompSpecs : compspecs. make_compspecs prog. Defined.`
`Definition Vprog : varspecs. mk_varspecs prog. Defined.`
`Require Import VC.hints.`

Now we import some VST libraries that don't come standard with VST.floyd.proofauto.

`Require Import VST.msl.wand_frame.`
`Require Import VST.msl.iter_sepcon.`
`Require Import VST.floyd.reassoc_seq.`
`Require Import VST.floyd.field_at_wand.`

Now we import the functional model. `Require Import VC.Hashfun.`

22.2 Function specifications

Imports from the C string library (see *Verif_strlib*)

```
Definition strcmp_spec :=
 DECLARE _strcmp
  WITH str1 : val, s1 : list byte, str2 : val, s2 : list byte
  PRE [ tptr tschar, tptr tschar ]
```

```
    PROP ()
    PARAMS (str1; str2)
    SEP (cstring Ews s1 str1; cstring Ews s2 str2)
  POST [ tint ]
   EX i : int,
    PROP (if Int.eq_dec i Int.zero then s1 = s2 else s1 ≠ s2)
    RETURN (Vint i)
    SEP (cstring Ews s1 str1; cstring Ews s2 str2).
Definition strcpy_spec :=
 DECLARE _strcpy
  WITH dest : val, n : Z, src : val, s : list byte
  PRE [ tptr tschar, tptr tschar ]
    PROP (Zlength s < n)
    PARAMS (dest; src)
    SEP (data_at_ Ews (tarray tschar n) dest; cstring Ews s src)
  POST [ tptr tschar ]
    PROP ()
    RETURN (dest)
    SEP (cstringn Ews s n dest; cstring Ews s src).
Definition strlen_spec :=
 DECLARE _strlen
  WITH s : list byte, str: val
  PRE [ tptr tschar ]
    PROP ( )
    PARAMS (str)
    SEP (cstring Ews s str)
  POST [ size_t ]
    PROP ()
    RETURN (Vptrofs (Ptrofs.repr (Zlength s)))
    SEP (cstring Ews s str).
```

String functions: copy, hash

```
Definition copy_string_spec : ident × funspec :=
 DECLARE _copy_string
 WITH s: val, sigma : list byte, gv: globals
 PRE [ tptr tschar ]
    PROP ()
    PARAMS (s) GLOBALS(gv)
    SEP (cstring Ews sigma s; mem_mgr gv)
 POST [ tptr tschar ]
    EX p: val,
```

```
        PROP ( ) RETURN (p)
        SEP (cstring Ews sigma s;
             cstring Ews sigma p;
             malloc_token Ews (tarray tschar (Zlength sigma + 1)) p;
             mem_mgr gv).
Definition hash_spec : ident × funspec :=
  DECLARE _hash
  WITH s: val, contents : list byte
  PRE [ tptr tschar ]
          PROP ()
          PARAMS (s)
          SEP (cstring Ews contents s)
  POST [ tuint ]
         PROP ()
         RETURN (Vint (Int.repr (hashfun contents)))
         SEP (cstring Ews contents s).
```

Data structures for hash table

Some abbreviations for C struct types that we use frequently `Definition tcell := Tstruct _cell noattr.`
`Definition thashtable := Tstruct _hashtable noattr.`

A *list_cell* has four parts:

- a linked list *cons* cell with a key-pointer *kp*, integer *count*, and pointer to the *next* element of the linked list;
- the key-pointer points to a string (null-terminated array of char) containing the key;
- the cons cell was obtained by *malloc*, and must be freeable by *free*, and so there's a *malloc_token* giving that capability;
- the key-string also has a *malloc-token* so that it can be freed

```
Definition list_cell (key: list byte) (count: Z) (next: val) (p: val): mpred :=
 EX kp: val, cstring Ews key kp
               × malloc_token Ews (tarray tschar (Zlength key + 1)) kp
               × data_at Ews tcell (kp,(Vint (Int.repr count), next)) p
               × malloc_token Ews tcell p.
Definition list_cell_local_facts:
  ∀ key count next p, list_cell key count next p |– !! isptr p.
Proof. intros. unfold list_cell. Intros kp. entailer!. Qed.
#[export] Hint Resolve list_cell_local_facts: saturate_local.
```

```
Definition list_cell_valid_pointer:
  ∀ key count next p, list_cell key count next p |– valid_pointer p.
Proof. intros. unfold list_cell. Intros kp. entailer!. Qed.
#[export] Hint Resolve list_cell_valid_pointer: valid_pointer.
```

Exercise: 1 star, standard (listcell_fold) `Lemma listcell_fold: ∀ key kp count p' p,`
`cstring Ews key kp`
`× malloc_token Ews (tarray tschar (Zlength key + 1)) kp`
`× data_at Ews tcell (kp, (Vint (Int.repr count), p')) p`
`× malloc_token Ews tcell p`
`|– list_cell key count p' p.`

```
Proof.
    Admitted.
    □
```

```
Fixpoint listrep (sigma: list (list byte × Z)) (x: val) : mpred :=
 match sigma with
 | (s,c)::hs ⇒ EX y: val, list_cell s c y x × listrep hs y
 | nil ⇒
     !! (x = nullval) && emp
 end.
```

Exercise: 2 stars, standard (listrep_hints) `Lemma listrep_local_prop: ∀ sigma p, listrep sigma p |–`
`!! (is_pointer_or_null p ∧ (p=nullval ↔ sigma=nil)).`

```
Proof.
    Admitted.
#[export] Hint Resolve listrep_local_prop : saturate_local.
```

```
Lemma listrep_valid_pointer:
  ∀ sigma p,
    listrep sigma p |– valid_pointer p.
Proof.
    Admitted.
#[export] Hint Resolve listrep_valid_pointer : valid_pointer.
    □
```

```
Lemma listrep_fold: ∀ key count p' p al,
  list_cell key count p' p × listrep al p' |– listrep ((key,count)::al) p.
Proof. intros. simpl. Exists p'. cancel. Qed.
```

A *listbox* is a pointer to a pointer to a cons cell. `Definition listboxrep al r :=`
`EX p:val, data_at Ews (tptr tcell) p r × listrep al p.`

```
Definition uncurry {A B C} (f: A → B → C) (xy: A×B) : C :=
  f (fst xy) (snd xy).
```

```
Definition hashtable_rep (contents: hashtable_contents) (p: val) : mpred :=
  EX bl: list (list byte × Z) × val),
    !! (contents = map fst bl) &&
    malloc_token Ews thashtable p ×
    data_at Ews thashtable (map snd bl) p
    × iter_sepcon (uncurry listrep) bl.
```

Exercise: 2 stars, standard (hashtable_rep_hints) `Lemma hashtable_rep_local_facts:`

```
∀ contents p,
 hashtable_rep contents p |– !! (isptr p ∧ Zlength contents = N).
   Admitted.
#[export] Hint Resolve hashtable_rep_local_facts : saturate_local.

Lemma hashtable_rep_valid_pointer: ∀ contents p,
 hashtable_rep contents p |– valid_pointer p.
   Admitted.
#[export] Hint Resolve hashtable_rep_valid_pointer : valid_pointer.
```

□

Function specifications for hash table

```
Definition new_table_spec : ident × funspec :=
 DECLARE _new_table
 WITH gv: globals
 PRE [ ]
   PROP()
   PARAMS() GLOBALS(gv)
   SEP(mem_mgr gv)
 POST [ tptr thashtable ]
   EX p:val, PROP()
      RETURN (p)
      SEP(hashtable_rep empty_table p; mem_mgr gv).

Definition new_cell_spec : ident × funspec :=
 DECLARE _new_cell
 WITH s: val, key: list byte, count: Z, next: val, gv: globals
 PRE [ tptr tschar, tint, tptr tcell ]
   PROP()
   PARAMS(s; Vint (Int.repr count); next) GLOBALS(gv)
   SEP(cstring Ews key s; mem_mgr gv)
 POST [ tptr tcell ]
   EX p:val, PROP()
      RETURN(p)
      SEP(list_cell key count next p; cstring Ews key s;
```

```
        mem_mgr gv).
Definition get_spec : ident × funspec :=
 DECLARE _get
 WITH p: val, contents: hashtable_contents, s: val, sigma : list byte
 PRE [ tptr (Tstruct _hashtable noattr), tptr tschar ]
    PROP ()
    PARAMS (p; s)
    SEP (hashtable_rep contents p; cstring Ews sigma s)
 POST [ tuint ]
    PROP ( )
    RETURN (Vint (Int.repr (hashtable_get sigma contents)))
    SEP (hashtable_rep contents p; cstring Ews sigma s).
Definition incr_list_spec : ident × funspec :=
 DECLARE _incr_list
 WITH r0: val, al: list (list byte × Z), s: val,
      sigma : list byte, gv: globals
 PRE [ tptr (tptr tcell), tptr tschar ]
    PROP (list_get sigma al < Int.max_unsigned)
    PARAMS (r0; s) GLOBALS(gv)
    SEP (listboxrep al r0; cstring Ews sigma s; mem_mgr gv)
 POST [ tvoid ]
      PROP ( ) RETURN ()
      SEP (listboxrep (list_incr sigma al) r0;
           cstring Ews sigma s; mem_mgr gv).
Definition incr_spec : ident × funspec :=
 DECLARE _incr
 WITH p: val, contents: hashtable_contents, s: val,
      sigma : list byte, gv: globals
 PRE [ tptr (Tstruct _hashtable noattr), tptr tschar ]
    PROP (hashtable_get sigma contents < Int.max_unsigned)
    PARAMS (p; s) GLOBALS(gv)
    SEP (hashtable_rep contents p; cstring Ews sigma s; mem_mgr gv)
 POST [ tvoid ]
      PROP ( ) RETURN ()
      SEP (hashtable_rep (hashtable_incr sigma contents) p;
           cstring Ews sigma s; mem_mgr gv).
```

Putting all the funspecs together

```
Definition Gprog : funspecs :=
        ltac:(with_library prog [
                    strcmp_spec; strcpy_spec; strlen_spec; hash_spec;
                    new_cell_spec; copy_string_spec; get_spec; incr_spec;
```

incr_list_spec
]).

22.3 Proofs of the functions *hash*, *copy_string*, *new_cell*

Before attempting to prove *body_hash*, do *Verif_strlib* at least through *body_strlen*.

Exercise: 3 stars, standard (body_hash) `Lemma` *body_hash*: *semax_body Vprog Gprog f_hash hash_spec*.
`Proof`.
start_function.
`unfold` *cstring* `in` *.

In the PROP part of your loop invariant, you'll want to maintain $0 \leq i \leq$ *Zlength contents*. In the LOCAL part of your loop invariant, try to use something like

temp