```
Last login: Mon Apr  9 13:41:05 on ttys007
carbon:$ pwd
/Users/evw
carbon:$ c2
carbon:$ cd public-class-repo/Sample\ Programs/Sec_01_1-25pm/
carbon:$ utop
────────┬─────────────────────────────────────────────────────────────┬────
        │ Welcome to utop version 2.0.2 (using OCaml version 4.06.0)! │
        └─────────────────────────────────────────────────────────────┘

Type #utop_help for help about using utop.

─( 13:41:22 )─< command 0 >──────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Val of value
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Mod of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | Not of expr
  | And of expr * expr
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | Seq of stmt * stmt
  | ReadNum of string
  | WriteNum of expr
val program_assign : stmt =
  Seq (Assign ("x", Val (Int 1)), Assign ("y", Add (Var "x", Val (Int 2))))
val program_seq : stmt =
  Seq (Assign ("x", Val (Int 1)),
   Seq (Assign ("y", Add (Var "x", Val (Int 2))),
    Seq (Assign ("z", Mul (Var "y", Val (Int 3))), WriteNum (Var "z"))))
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
val exec : stmt -> state -> state = <fun>
─( 13:41:22 )─< command 1 >──────────────────────────────────────{ counter: 0 }─
utop # read_number;;
- : unit -> int = <fun>
─( 13:41:27 )─< command 2 >──────────────────────────────────────{ counter: 0 }─
utop # read_number () ;;
Enter an integer value:
```

```
55
- : int = 55
─( 13:41:48 )─< command 3 >──────────────────────────────────────{ counter: 0 }─
utop # write_number 44 ;;
44
- : unit = ()
─( 13:42:00 )─< command 4 >──────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Val of value
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Mod of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | Not of expr
  | And of expr * expr
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | Seq of stmt * stmt
  | ReadNum of string
  | WriteNum of expr
File "interpreter.ml", line 99, characters 2-103:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
(ReadNum _|WriteNum _)
val exec : stmt -> state -> state = <fun>
val program_assign : stmt =
  Seq (Assign ("x", Val (Int 1)), Assign ("y", Add (Var "x", Val (Int 2))))
val program_seq : stmt =
  Seq (Assign ("x", Val (Int 1)),
   Seq (Assign ("y", Add (Var "x", Val (Int 2))),
    Seq (Assign ("z", Mul (Var "y", Val (Int 3))), WriteNum (Var "z"))))
─( 13:42:10 )─< command 5 >──────────────────────────────────────{ counter: 0 }─
utop # program_assign ;;
- : stmt =
Seq (Assign ("x", Val (Int 1)), Assign ("y", Add (Var "x", Val (Int 2))))
─( 13:45:43 )─< command 6 >──────────────────────────────────────{ counter: 0 }─
utop # exec program_assing [] ;;
Error: Unbound value program_assing
Hint: Did you mean program_assign?
```

```
─( 13:46:05 )─< command 7 >──────────────────────────────────────────{ counter: 0 }─
utop # exec program_assign [] ;;
- : state = [("y", Int 3); ("x", Int 1)]
─( 13:46:18 )─< command 8 >──────────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Val of value
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Mod of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | Not of expr
  | And of expr * expr
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | Seq of stmt * stmt
  | ReadNum of string
  | WriteNum of expr
File "interpreter.ml", line 105, characters 30-45:
Error: This expression has type unit but an expression was expected of type
         state = (string * value) list
─( 13:46:23 )─< command 9 >──────────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Val of value
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Mod of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | Not of expr
  | And of expr * expr
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | Seq of stmt * stmt
  | ReadNum of string
  | WriteNum of expr
val exec : stmt -> state -> state = <fun>
val program_assign : stmt =
  Seq (Assign ("x", Val (Int 1)), Assign ("y", Add (Var "x", Val (Int 2))))
val program_seq : stmt =
  Seq (Assign ("x", Val (Int 1)),
   Seq (Assign ("y", Add (Var "x", Val (Int 2))),
    Seq (Assign ("z", Mul (Var "y", Val (Int 3))), WriteNum (Var "z"))))
─( 13:49:32 )─< command 10 >──────────────────────────────────────{ counter: 0 }─
utop # exec program_seq [] ;;
9
- : state = [("z", Int 9); ("y", Int 3); ("x", Int 1)]
─( 13:50:41 )─< command 11 >──────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Val of value
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Mod of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | Not of expr
  | And of expr * expr
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | Seq of stmt * stmt
  | ReadNum of string
  | WriteNum of expr
  | IfThen of expr * stmt
val exec : stmt -> state -> state = <fun>
val program_assign : stmt =
  Seq (Assign ("x", Val (Int 1)), Assign ("y", Add (Var "x", Val (Int 2))))
val program_seq : stmt =
  Seq (Assign ("x", Val (Int 1)),
   Seq (Assign ("y", Add (Var "x", Val (Int 2))),
    Seq (Assign ("z", Mul (Var "y", Val (Int 3))), WriteNum (Var "z"))))
val program_ifthen_simple_1 : stmt =
  Seq (Assign ("y", Val (Int 10)),
   IfThen (Lt (Var "y", Val (Int 15)), WriteNum (Var "y")))
─( 13:50:48 )─< command 12 >────────────────────────────────────{ counter: 0 }─
utop # exec program_ifthen_simple_1 [] ;;
10
- : state = [("y", Int 10)]
─( 13:54:04 )─< command 13 >────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Val of value
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Mod of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | Not of expr
  | And of expr * expr
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | Seq of stmt * stmt
  | ReadNum of string
  | WriteNum of expr
  | IfThen of expr * stmt
val exec : stmt -> state -> state = <fun>
val program_assign : stmt =
  Seq (Assign ("x", Val (Int 1)), Assign ("y", Add (Var "x", Val (Int 2))))
val program_seq : stmt =
  Seq (Assign ("x", Val (Int 1)),
   Seq (Assign ("y", Add (Var "x", Val (Int 2))),
    Seq (Assign ("z", Mul (Var "y", Val (Int 3))), WriteNum (Var "z"))))
val program_ifthen_simple_1 : stmt =
  Seq (Assign ("y", Val (Int 10)),
   IfThen (Lt (Var "y", Val (Int 15)), WriteNum (Var "y")))
val program_ifthen_simple_2 : stmt =
  Seq (Assign ("y", Val (Int 0)),
   Seq
    (IfThen (Eq (Var "y", Val (Int 0)),
      Assign ("y", Add (Var "y", Val (Int 2)))),
    Seq
```

```
     (IfThen (Not (Lt (Var "y", Val (Int 4))),
       Assign ("y", Add (Var "y", Val (Int 3)))),
     IfThen (Lt (Var "y", Val (Int 10)),
      Assign ("y", Add (Var "y", Val (Int 4)))))))
─( 13:54:34 )─< command 14 >─────────────────────────────────────{ counter: 0 }─
utop # exec program_ifthen_simple_2 [] ;;
Stack overflow during evaluation (looping recursion?).
─( 13:55:29 )─< command 15 >─────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Val of value
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Mod of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | Not of expr
  | And of expr * expr
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | Seq of stmt * stmt
  | ReadNum of string
  | WriteNum of expr
  | IfThen of expr * stmt
val exec : stmt -> state -> state = <fun>
val program_assign : stmt =
  Seq (Assign ("x", Val (Int 1)), Assign ("y", Add (Var "x", Val (Int 2))))
val program_seq : stmt =
  Seq (Assign ("x", Val (Int 1)),
   Seq (Assign ("y", Add (Var "x", Val (Int 2))),
    Seq (Assign ("z", Mul (Var "y", Val (Int 3))), WriteNum (Var "z"))))
val program_ifthen_simple_1 : stmt =
  Seq (Assign ("y", Val (Int 10)),
   IfThen (Lt (Var "y", Val (Int 15)), WriteNum (Var "y")))
val program_ifthen_simple_2 : stmt =
  Seq (Assign ("y", Val (Int 0)),
   Seq
    (IfThen (Eq (Var "y", Val (Int 0)),
      Assign ("y", Add (Var "y", Val (Int 2)))),
    Seq
     (IfThen (Not (Lt (Var "y", Val (Int 4))),
       Assign ("y", Add (Var "y", Val (Int 3)))),
     IfThen (Lt (Var "y", Val (Int 10)),
      Assign ("y", Add (Var "y", Val (Int 4)))))))
─( 13:55:32 )─< command 16 >────────────────────────────────────{ counter: 0 }─
utop # exec program_ifthen_simple_2 [] ;;
- : state = [("y", Int 6); ("y", Int 2); ("y", Int 0)]
─( 13:55:55 )─< command 17 >────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Val of value
  | Var of string
  | Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Mod of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | Not of expr
  | And of expr * expr
type environment = (string * value) list
val lookup : string -> (string * 'a) list -> 'a = <fun>
val eval : expr -> environment -> value = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
type state = environment
type stmt =
    Assign of string * expr
  | Seq of stmt * stmt
  | ReadNum of string
  | WriteNum of expr
  | IfThen of expr * stmt
  | While of expr * stmt
val exec : stmt -> state -> state = <fun>
val program_assign : stmt =
  Seq (Assign ("x", Val (Int 1)), Assign ("y", Add (Var "x", Val (Int 2))))
val program_seq : stmt =
  Seq (Assign ("x", Val (Int 1)),
   Seq (Assign ("y", Add (Var "x", Val (Int 2))),
    Seq (Assign ("z", Mul (Var "y", Val (Int 3))), WriteNum (Var "z"))))
val program_ifthen_simple_1 : stmt =
  Seq (Assign ("y", Val (Int 10)),
   IfThen (Lt (Var "y", Val (Int 15)), WriteNum (Var "y")))
val program_ifthen_simple_2 : stmt =
  Seq (Assign ("y", Val (Int 0)),
   Seq
    (IfThen (Eq (Var "y", Val (Int 0)),
      Assign ("y", Add (Var "y", Val (Int 2)))),
    Seq
     (IfThen (Not (Lt (Var "y", Val (Int 4))),
        Assign ("y", Add (Var "y", Val (Int 3)))),
      IfThen (Lt (Var "y", Val (Int 10)),
       Assign ("y", Add (Var "y", Val (Int 4)))))))
val program_while : stmt =
  Seq (ReadNum "x",
   Seq (Assign ("i", Val (Int 0)),
    Seq (Assign ("sum", Val (Int 0)),
     Seq
      (While (Lt (Var "i", Var "x"),
        Seq (WriteNum (Var "i"),
         Seq (Assign ("sum", Add (Var "sum", Var "i")),
          Assign ("i", Add (Var "i", Val (Int 1)))))),
      WriteNum (Var "sum")))))
─( 13:55:57 )─< command 18 >─────────────────────────────────────{ counter: 0 }─
utop # exec program_while []  ;;
Enter an integer value:
5
0
1
2
3
4
10
- : state =
[("i", Int 5); ("sum", Int 10); ("i", Int 4); ("sum", Int 6); ("i", Int 3);
 ("sum", Int 3); ("i", Int 2); ("sum", Int 1); ("i", Int 1); ("sum", Int 0);
 ("sum", Int 0); ("i", Int 0); ("x", Int 5)]
─( 14:03:27 )─< command 19 >─────────────────────────────────────{ counter: 0 }─
utop #
┌───┬───┬───┬─────┬───────────┬──────────────┬──────┬────────┬────┬──────┬─────┐
│Add│And│Arg│Array│ArrayLabels│Assert_failure│Assign│Bigarray│Bool│Buffer│Bytes│
└───┴───┴───┴─────┴───────────┴──────────────┴──────┴────────┴────┴──────┴─────┘
```