```
Last login: Fri Mar 31 15:34:55 on ttys015
carbon:SamplePrograms$ cd Sec_10_3\:35pm/
carbon:Sec_10_3:35pm$ utop
```

```
        Welcome to utop version 1.14 (using OCaml version 4.01.0)!

Type #utop_help for help about using utop.

─( 18:00:00 )─< command 0 >──────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
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>
─( 16:01:36 )─< command 1 >──────────────────────────────────────{ counter: 0 }─
utop # eval (Add (Var "x", Value (Int 3))) ;;
- : environment -> value = <fun>
─( 16:01:39 )─< command 2 >──────────────────────────────────────{ counter: 0 }─
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
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
  | While of expr * stmt
  | IfThen of expr * stmt
  | WriteNum of expr
  | Seq of stmt * stmt
val program_1 : stmt =
  Seq (Assign ("x", Value (Int 2)),
```

```
   Seq (Assign ("y", Add (Var "x", Value (Int 3))),
    Seq (Assign ("z", Add (Var "y", Value (Int 2))), WriteNum (Var "z"))))
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
-( 16:01:53 )-< command 3 >-------------------------------------------{ counter: 0 }-
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
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
  | While of expr * stmt
  | IfThen of expr * stmt
  | WriteNum of expr
  | Seq of stmt * stmt
val program_1 : stmt =
  Seq (Assign ("x", Value (Int 2)),
   Seq (Assign ("y", Add (Var "x", Value (Int 3))),
    Seq (Assign ("z", Add (Var "y", Value (Int 2))), WriteNum (Var "z"))))
File "interpreter.ml", line 89, characters 2-61:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
(While (_, _)|IfThen (_, _)|WriteNum _|Seq (_, _))
val exec : stmt -> state -> state = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
-( 16:13:33 )-< command 4 >-------------------------------------------{ counter: 0 }-
utop # exec (Assign ("x", Value (Int 3))) [] ;;
- : state = [("x", Int 3)]
-( 16:16:53 )-< command 5 >-------------------------------------------{ counter: 0 }-
utop # exec (Assign ("x", Add( Value (Int 3), Var "y"))) [ ("y", Int 5)] ;;
- : state = [("x", Int 8); ("y", Int 5)]
-( 16:17:11 )-< command 6 >-------------------------------------------{ counter: 0 }-
utop # exec (Assign ("x", Add( Value (Int 3), Var "y"))) [ ( "x", Int 8); ("y", I
nt 5)] ;;
- : state = [("x", Int 8); ("x", Int 8); ("y", Int 5)]
-( 16:17:40 )-< command 7 >-------------------------------------------{ counter: 0 }-
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
```

```
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
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
  | While of expr * stmt
  | IfThen of expr * stmt
  | WriteNum of expr
  | Seq of stmt * stmt
val program_1 : stmt =
  Seq (Assign ("x", Value (Int 2)),
   Seq (Assign ("y", Add (Var "x", Value (Int 3))),
    Seq (Assign ("z", Add (Var "y", Value (Int 2))), WriteNum (Var "z"))))
File "interpreter.ml", line 89, characters 2-103:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
(While (_, _)|IfThen (_, _)|WriteNum _)
val exec : stmt -> state -> state = <fun>
val read_number : unit -> int = <fun>
val write_number : int -> unit = <fun>
-( 16:18:17 )-< command 8 >-------------------------------------------------{ counter: 0 }-
utop # exec (Seq (Assign ("x", Int 3), Assign ("x", Add (Var "x", Value (Int 4)))
)) [] ;;
Error: The variant type expr has no constructor Int
-( 16:21:31 )-< command 9 >-------------------------------------------------{ counter: 0 }-
utop # exec (Seq (Assign ("x", Value (Int 3)), Assign ("x", Add (Var "x", Value (
Int 4))))) [] ;;
- : state = [("x", Int 7); ("x", Int 3)]
-( 16:22:31 )-< command 10 >------------------------------------------------{ counter: 0 }-
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
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
  | While of expr * stmt
```

```
  | IfThen of expr * stmt
  | WriteNum of expr
  | Seq of stmt * stmt
val program_1 : stmt =
  Seq (Assign ("x", Value (Int 2)),
   Seq (Assign ("y", Add (Var "x", Value (Int 3))),
    Seq (Assign ("z", Add (Var "y", Value (Int 2))), WriteNum (Var "z"))))
val write_number : int -> unit = <fun>
File "interpreter.ml", line 95, characters 30-44:
Error: This expression has type unit but an expression was expected of type
         state = (string * value) list
-( 16:22:31 )-< command 11 >-------------------------------------{ counter: 0 }-
utop # #use "interpreter.ml";;
type value = Int of int | Bool of bool
type expr =
    Add of expr * expr
  | Mul of expr * expr
  | Sub of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | Var of string
  | Value of value
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
  | While of expr * stmt
  | IfThen of expr * stmt
  | WriteNum of expr
  | Seq of stmt * stmt
val program_1 : stmt =
  Seq (Assign ("x", Value (Int 2)),
   Seq (Assign ("y", Add (Var "x", Value (Int 3))),
    Seq (Assign ("z", Add (Var "y", Value (Int 2))), WriteNum (Var "z"))))
val write_number : int -> unit = <fun>
File "interpreter.ml", line 94, characters 18-110:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
Bool _
File "interpreter.ml", line 91, characters 2-214:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
(While (_, _)|IfThen (_, _))
val exec : stmt -> state -> state = <fun>
val read_number : unit -> int = <fun>
-( 16:23:52 )-< command 12 >-------------------------------------{ counter: 0 }-
utop # exec program_1 ;;
- : state -> state = <fun>
-( 16:24:15 )-< command 13 >-------------------------------------{ counter: 0 }-
utop # exec program_1 [] ;;
7
```

```
- : state = [("z", Int 7); ("y", Int 5); ("x", Int 2)]
─( 16:24:21 )─< command 14 >──────────────────────────────────────{ counter: 0 }─
utop #
```

Add	And	Arg	Arith_status	Array	ArrayLabels	Assert_failure	Assign	Big_int	Bigarr