```
Last login: Mon Apr  9 13:41:07 on ttys008
carbon:$ c2
carbon:$ cd public-class-repo/Sample\ Programs/Sec_10_3-35pm/
carbon:$ utop
                                                                    
Welcome to utop version 2.0.2 (using OCaml version 4.06.0)
                                                                    

Type #utop_help for help about using utop.

-( 15:49:51 )-< command 0 >-------------------{ counter: 0 }-
utop # #use "interpreter.ml";;
File "interpreter.ml", line 53, characters 5-6:
Error: Syntax error
-( 15:49:51 )-< command 1 >-------------------{ 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
  | 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>
-( 15:49:54 )-< command 2 >-------------------{ counter: 0 }-
utop # read_number ;;
```

```
- : unit -> int = <fun>
-( 15:50:02 )-< command 3 >-----------------{ counter: 0 }-
utop # read_number () ;;
Enter an integer value:
55
- : int = 55
-( 15:50:10 )-< command 4 >-----------------{ counter: 0 }-
utop # write_number ;;
- : int -> unit = <fun>
-( 15:50:18 )-< command 5 >-----------------{ counter: 0 }-
utop # write_number 44 ;;
44
- : unit = ()
-( 15:50:24 )-< command 6 >-----------------{ 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
  | 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 95, characters 2-102:
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"))))
-( 15:50:28 )-< command 7 >-----------------{ counter: 0 }-
utop # exec program_assign [] ;;
- : state = [("y", Int 3); ("x", Int 1)]
-( 15:54:02 )-< command 8 >-----------------{ counter: 0 }-
utop # exec program_seq [] ;;
Exception: Match_failure ("interpreter.ml", 95, 2).
-( 15:54:25 )-< 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
  | 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 95, characters 2-249:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
ReadNum _
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"))))
-( 15:55:04 )-< command 10 >----------------{ counter: 0 }-
utop # exec program_seq [] ;;
9
- : state = [("z", Int 9); ("y", Int 3); ("x", Int 1)]
-( 15:58:12 )-< command 11 >----------------{ counter: 0 }-
```

```
utop # #use "interpreter.ml";;
File "interpreter.ml", line 110, characters 0-3:
Error: Syntax error: ')' expected
File "interpreter.ml", line 105, characters 5-6:
Error: This '(' might be unmatched
-( 15:58:13 )-< command 12 >----------------{ 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
  | 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
File "interpreter.ml", line 96, characters 2-431:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
ReadNum _
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")))
-( 16:01:06 )-< command 13 >----------------{ counter: 0 }-
utop # exec program_ifthen_simple_1 [] ;;
```

```
10
- : state = [("y", Int 10)]
-( 16:01:20 )-< command 14 >-----------------{ 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
  | 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
File "interpreter.ml", line 96, characters 2-431:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
ReadNum _
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))))))))
-( 16:01:31 )-< command 15 >-----------------{ counter: 0 }-
utop # exec program_ifthen_simple_2 [] ;;
- : state = [("y", Int 6); ("y", Int 2); ("y", Int 0)]
-( 16:02:25 )-< command 16 >-----------------{ counter: 0 }-
utop # #use "interpreter.ml";;
File "interpreter.ml", line 119, characters 0-3:
Error: Syntax error: ')' expected
File "interpreter.ml", line 114, characters 5-6:
Error: This '(' might be unmatched
-( 16:02:31 )-< 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
  | 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
File "interpreter.ml", line 101, characters 5-51:
Error: This expression has type state -> state
       but an expression was expected of type
         state = (string * value) list
-( 16:09:54 )-< command 18 >-----------------{ 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
  | 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
File "interpreter.ml", line 114, characters 5-153:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Int _
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")))))
-( 16:10:00 )-< command 19 >----------------{ counter: 0 }-
utop # exec program_while [] ;;
Enter an integer value:
2
0
1
1
- : state =
[("i", Int 2); ("sum", Int 1); ("i", Int 1);
 ("sum", Int 0); ("sum", Int 0); ("i", Int 0);
 ("x", Int 2)]
-( 16:10:25 )-< command 20 >----------------{ 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)]
-( 16:10:49 )-< command 21 >----------------{ 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
  | 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
File "interpreter.ml", line 114, characters 5-244:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
Int _
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")))))
```

```
─( 16:10:55 )─< command 22 >──────────────────{ 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)]
─( 16:15:10 )─< command 23 >──────────────────────────────────{ counter: 0 }─
utop #
```

Add	And	Arg	Array	ArrayLabels	Assert_failure	Assign	Bigarray	Bool	Buffe