```
Last login: Fri Mar  3 13:12:08 on ttys019
carbon:~$ c2
carbon:17_Fall_2041$ cd carbon-repos/repo-score100/Hwk_04
carbon:Hwk_04$ 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 "eval.ml" ;;
type expr =
    Add of expr * expr
  | Sub of expr * expr
  | Mul of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | If of expr * expr * expr
  | Id of string
  | Let of string * expr * expr
  | LetRec of string * expr * expr
  | App of expr * expr
  | Lambda of string * expr
  | Value of value
and value =
    Int of int
  | Bool of bool
  | Closure of string * expr * environment
  | Ref of value ref
and environment = (string * value) list
val lookup : string -> environment -> value = <fun>
val freevars : expr -> string list = <fun>
val eval : environment -> expr -> value = <fun>
val evaluate : expr -> value = <fun>
val i0 : expr = Value (Int 0)
val i1 : expr = Value (Int 1)
val i2 : expr = Value (Int 2)
val i3 : expr = Value (Int 3)
val i4 : expr = Value (Int 4)
val a1 : expr = Add (Value (Int 2), Value (Int 4))
val m1 : expr = Mul (Add (Value (Int 2), Value (Int 4)), Value (Int 3))
val e1 : expr = Add (Add (Value (Int 2), Value (Int 4)), Id "x")
val e2 : expr =
  Mul (Add (Add (Value (Int 2), Value (Int 4)), Id "x"),
   Add (Add (Value (Int 2), Value (Int 4)), Id "x"))
val inc : expr = Lambda ("n", Add (Id "n", Value (Int 1)))
val two : expr = App (Lambda ("n", Add (Id "n", Value (Int 1))), Value (Int 1))
val sumToBody : expr =
  If (Eq (Id "n", Value (Int 0)), Value (Int 0),
   Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1)))))
val sumTo : expr =
  LetRec ("sumTo",
   Lambda ("n",
    If (Eq (Id "n", Value (Int 0)), Value (Int 0),
     Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1)))))),
   Id "sumTo")
val dummy : value = Int 999
val sumToRef : value ref = {contents = Int 999}
val sumToV : value =
  Closure ("n",
   If (Eq (Id "n", Value (Int 0)), Value (Int 0),
    Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
   [("sumTo", Ref {contents = Int 999})])
val sumTo4 : expr =
  App
   (Value
     (Closure ("n",
       If (Eq (Id "n", Value (Int 0)), Value (Int 0),
        Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
       [("sumTo",
         Ref
          {contents =
            Closure ("n",
             If (Eq (Id "n", Value (Int 0)), Value (Int 0),
              Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
             [("sumTo",
               Ref
                {contents =
                  Closure ("n",
                   If (Eq (Id "n", Value (Int 0)), Value (Int 0),
                    Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1)))))
,
                   [("sumTo",
                     Ref
                      {contents =
                        Closure ("n",
                         If (Eq (Id "n", Value (Int 0)), Value (Int 0),
                          Add (Id "n",
                           App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
                         [("sumTo",
                           Ref
                            {contents =
                              Closure ("n",
                               If (Eq (Id "n", Value (Int 0)), Value (Int 0),
                                Add (Id "n",
                                 App (Id "sumTo", Sub (Id "n", Value (Int 1)))))
,
                               [("sumTo",
                                 Ref
                                  {contents =
                                    Closure ("n",
                                     If (Eq (Id "n", Value (Int 0)),
                                      Value (Int 0),
                                      Add (Id "n",
                                       App (Id "sumTo",
                                        Sub (Id "n", Value (Int 1))))),
                                     [("sumTo",
                                       Ref
                                        {contents =
                                          Closure ("n",
                                           If (Eq (Id "n", Value (Int 0)),
                                            Value (Int 0),
                                            Add (Id "n",
                                             App (Id "sumTo",
                                              Sub (Id "n", Value (Int 1))))),
                                           [("sumTo",
                                             Ref
                                              {contents =
                                                Closure ("n",
                                                 If
                                                  (Eq (Id "n", Value (Int 0)),
                                                  Value (Int 0),
                                                  Add (Id "n",
                                                   App (Id "sumTo",
                                                    Sub (Id "n", Value (Int 1)))
)),
                                                 [("sumTo",
                                                   Ref
                                                    {contents =
                                                      Closure ("n",
                                                       If
                                                        (Eq (Id "n",
                                                          Value (Int 0)),
                                                        Value (Int 0),
                                                        Add (Id "n",
                                                         App (Id "sumTo",
                                                          Sub (Id "n",
                                                           Value (Int 1))))),
                                                       [("sumTo",
                                                         Ref
                                                          {contents =
                                                            Closure ("n",
                                                             If
                                                              (Eq (Id "n",
                                                                Value (Int 0)),
                                                              Value (Int 0),
                                                              Add (Id "n",
                                                               App (Id "sumTo",
                                                                Sub (
                                                                 Id "n",
                                                                 Value (Int 1)))
)),
                                                             [("sumTo", ...);
                                                              ...])});
                                                        ...])});
                                                  ...])});
                                            ...])});
                                      ...])});
                                ...])});
```

```
                              ...])});
                        ...])});
                  ...])});
            ...])),
       ...)
val sumToWith : expr =
  Lambda ("i",
   LetRec ("sTW",
    Lambda ("n",
     If (Eq (Id "n", Value (Int 0)), Id "i",
      Add (Id "n", App (Id "sTW", Sub (Id "n", Value (Int 1)))))),
    Id "sTW"))
val sT100 : expr =
  App
   (Lambda ("i",
     LetRec ("sTW",
      Lambda ("n",
       If (Eq (Id "n", Value (Int 0)), Id "i",
        Add (Id "n", App (Id "sTW", Sub (Id "n", Value (Int 1)))))),
      Id "sTW")),
   Value (Int 1000))
val st4 : expr =
  App
   (App
     (Lambda ("i",
       LetRec ("sTW",
        Lambda ("n",
         If (Eq (Id "n", Value (Int 0)), Id "i",
          Add (Id "n", App (Id "sTW", Sub (Id "n", Value (Int 1)))))),
        Id "sTW")),
     Value (Int 1000)),
   Value (Int 4))
val add : expr = Lambda ("x", Lambda ("y", Add (Id "x", Id "y")))
val inc' : expr =
  App (Lambda ("x", Lambda ("y", Add (Id "x", Id "y"))), Value (Int 1))
val five : expr =
  App (App (Lambda ("x", Lambda ("y", Add (Id "x", Id "y"))), Value (Int 1)),
   Value (Int 4))
─( 13:22:37 )─< command 1 >──────────────────────────────────────{ counter: 0 }─
utop # evaluate (App (sumTo, Value (Int 3))) ;;
- : value = Int 6
─( 13:22:42 )─< command 2 >──────────────────────────────────────{ counter: 0 }─
utop # evaluate (App (sumTo, Value (Int 6))) ;;
- : value = Int 21
─( 13:23:23 )─< command 3 >──────────────────────────────────────{ counter: 0 }─
utop # #quit ;;
carbon:Hwk_04$ c


carbon:Hwk_04$
```

```
carbon:Hwk_04$ 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 "eval.ml" ;;
type expr =
    Add of expr * expr
  | Sub of expr * expr
  | Mul of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | If of expr * expr * expr
  | Id of string
  | Let of string * expr * expr
  | LetRec of string * expr * expr
  | App of expr * expr
  | Lambda of string * expr
  | Value of value
and value =
    Int of int
  | Bool of bool
  | Closure of string * expr * environment
  | Ref of value ref
and environment = (string * value) list
val lookup : string -> environment -> value = <fun>
val freevars : expr -> string list = <fun>
val eval : environment -> expr -> value = <fun>
val evaluate : expr -> value = <fun>
val i0 : expr = Value (Int 0)
val i1 : expr = Value (Int 1)
val i2 : expr = Value (Int 2)
val i3 : expr = Value (Int 3)
val i4 : expr = Value (Int 4)
val a1 : expr = Add (Value (Int 2), Value (Int 4))
val m1 : expr = Mul (Add (Value (Int 2), Value (Int 4)), Value (Int 3))
val e1 : expr = Add (Add (Value (Int 2), Value (Int 4)), Id "x")
val e2 : expr =
  Mul (Add (Add (Value (Int 2), Value (Int 4)), Id "x"),
   Add (Add (Value (Int 2), Value (Int 4)), Id "x"))
val inc : expr = Lambda ("n", Add (Id "n", Value (Int 1)))
val two : expr = App (Lambda ("n", Add (Id "n", Value (Int 1))), Value (Int 1))
val sumToBody : expr =
  If (Eq (Id "n", Value (Int 0)), Value (Int 0),
   Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1)))))
val sumTo : expr =
  LetRec ("sumTo",
   Lambda ("n",
    If (Eq (Id "n", Value (Int 0)), Value (Int 0),
     Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1)))))),
   Id "sumTo")
val dummy : value = Int 999
val sumToRef : value ref = {contents = Int 999}
val sumToV : value =
  Closure ("n",
   If (Eq (Id "n", Value (Int 0)), Value (Int 0),
    Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
   [("sumTo", Ref {contents = Int 999})])
val sumTo4 : expr =
  App
   (Value
     (Closure ("n",
       If (Eq (Id "n", Value (Int 0)), Value (Int 0),
        Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
       [("sumTo",
         Ref
          {contents =
            Closure ("n",
             If (Eq (Id "n", Value (Int 0)), Value (Int 0),
              Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
             [("sumTo",
               Ref
                {contents =
                  Closure ("n",
                   If (Eq (Id "n", Value (Int 0)), Value (Int 0),
                    Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1)))))
,
                   [("sumTo",
                     Ref
                      {contents =
                        Closure ("n",
                         If (Eq (Id "n", Value (Int 0)), Value (Int 0),
                          Add (Id "n",
                           App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
                         [("sumTo",
                           Ref
                            {contents =
                              Closure ("n",
                               If (Eq (Id "n", Value (Int 0)), Value (Int 0),
                                Add (Id "n",
                                 App (Id "sumTo", Sub (Id "n", Value (Int 1)))))
,
                               [("sumTo",
                                 Ref
                                  {contents =
                                    Closure ("n",
                                     If (Eq (Id "n", Value (Int 0)),
                                      Value (Int 0),
                                      Add (Id "n",
                                       App (Id "sumTo",
                                        Sub (Id "n", Value (Int 1))))),
                                     [("sumTo",
                                       Ref
                                        {contents =
                                          Closure ("n",
                                           If (Eq (Id "n", Value (Int 0)),
                                            Value (Int 0),
                                            Add (Id "n",
                                             App (Id "sumTo",
                                              Sub (Id "n", Value (Int 1))))),
                                           [("sumTo",
                                             Ref
                                              {contents =
                                                Closure ("n",
                                                 If
                                                  (Eq (Id "n", Value (Int 0)),
                                                  Value (Int 0),
                                                  Add (Id "n",
                                                   App (Id "sumTo",
                                                    Sub (Id "n", Value (Int 1)))
)),
                                                 [("sumTo",
                                                   Ref
                                                    {contents =
                                                      Closure ("n",
                                                       If
                                                        (Eq (Id "n",
                                                          Value (Int 0)),
                                                        Value (Int 0),
                                                        Add (Id "n",
                                                         App (Id "sumTo",
                                                          Sub (Id "n",
                                                           Value (Int 1))))),
                                                       [("sumTo",
                                                         Ref
                                                          {contents =
                                                            Closure ("n",
                                                             If
                                                              (Eq (Id "n",
                                                                Value (Int 0)),
                                                              Value (Int 0),
                                                              Add (Id "n",
                                                               App (Id "sumTo",
                                                                Sub (
                                                                 Id "n",
                                                                 Value (Int 1)))
)),
                                                             [("sumTo", ...);
                                                              ...])});
                                                        ...])});
                                                  ...])});
                                            ...])});
                                      ...])});
                                ...])});
                          ...])});
                    ...])});
              ...])});
        ...])),
   ...)
val sumToWith : expr =
  Lambda ("i",
   LetRec ("sTW",
    Lambda ("n",
     If (Eq (Id "n", Value (Int 0)), Id "i",
      Add (Id "n", App (Id "sTW", Sub (Id "n", Value (Int 1)))))),
    Id "sTW"))
val sT100 : expr =
  App
   (Lambda ("i",
     LetRec ("sTW",
      Lambda ("n",
       If (Eq (Id "n", Value (Int 0)), Id "i",
        Add (Id "n", App (Id "sTW", Sub (Id "n", Value (Int 1)))))),
      Id "sTW")),
   Value (Int 1000))
val st4 : expr =
  App
   (App
     (Lambda ("i",
       LetRec ("sTW",
        Lambda ("n",
         If (Eq (Id "n", Value (Int 0)), Id "i",
          Add (Id "n", App (Id "sTW", Sub (Id "n", Value (Int 1)))))),
        Id "sTW")),
     Value (Int 1000)),
   Value (Int 4))
val add : expr = Lambda ("x", Lambda ("y", Add (Id "x", Id "y")))
val inc' : expr =
  App (Lambda ("x", Lambda ("y", Add (Id "x", Id "y"))), Value (Int 1))
val five : expr =
  App (App (Lambda ("x", Lambda ("y", Add (Id "x", Id "y"))), Value (Int 1)),
   Value (Int 4))
─( 13:28:29 )─< command 1 >──────────────────────────────────────{ counter: 0 }─
```

```
utop # evalute (App (sumTo, Value (Int 6))) ;;
Error: Unbound value evalute
Did you mean evaluate?
─( 13:28:34 )─< command 2 >──────────────────────────────────────{ counter: 0 }─
utop # evaluate (App (sumTo, Value (Int 6))) ;;
- : value = Int 21
─( 13:28:53 )─< command 3 >──────────────────────────────────────{ counter: 0 }─
utop # #quit ;;
carbon:Hwk_04$ c


carbon:Hwk_04$ 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 "eval.ml";;
type expr =
    Add of expr * expr
  | Sub of expr * expr
  | Mul of expr * expr
  | Div of expr * expr
  | Lt of expr * expr
  | Eq of expr * expr
  | And of expr * expr
  | If of expr * expr * expr
  | Id of string
  | Let of string * expr * expr
  | LetRec of string * expr * expr
  | App of expr * expr
  | Lambda of string * expr
  | Value of value
and value =
    Int of int
  | Bool of bool
  | Closure of string * expr * environment
  | Ref of value ref
and environment = (string * value) list
val lookup : string -> environment -> value = <fun>
val freevars : expr -> string list = <fun>
val eval : environment -> expr -> value = <fun>
val evaluate : expr -> value = <fun>
val i0 : expr = Value (Int 0)
val i1 : expr = Value (Int 1)
val i2 : expr = Value (Int 2)
val i3 : expr = Value (Int 3)
val i4 : expr = Value (Int 4)
val a1 : expr = Add (Value (Int 2), Value (Int 4))
val m1 : expr = Mul (Add (Value (Int 2), Value (Int 4)), Value (Int 3))
val e1 : expr = Add (Add (Value (Int 2), Value (Int 4)), Id "x")
val e2 : expr =
  Mul (Add (Add (Value (Int 2), Value (Int 4)), Id "x"),
   Add (Add (Value (Int 2), Value (Int 4)), Id "x"))
val inc : expr = Lambda ("n", Add (Id "n", Value (Int 1)))
val two : expr = App (Lambda ("n", Add (Id "n", Value (Int 1))), Value (Int 1))
val sumToBody : expr =
  If (Eq (Id "n", Value (Int 0)), Value (Int 0),
   Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1)))))
val sumTo : expr =
  LetRec ("sumTo",
   Lambda ("n",
    If (Eq (Id "n", Value (Int 0)), Value (Int 0),
     Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1)))))),
   Id "sumTo")
val sumToN_expr : expr =
  LetRec ("sumToN",
   Lambda ("n",
    If (Eq (Id "n", Value (Int 0)), Value (Int 0),
     Add (Id "n", App (Id "sumToN", Sub (Id "n", Value (Int 1)))))),
   Id "sumToN")
val dummy : value = Int 999
val sumToRef : value ref = {contents = Int 999}
val sumToV : value =
  Closure ("n",
   If (Eq (Id "n", Value (Int 0)), Value (Int 0),
    Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
   [("sumTo", Ref {contents = Int 999})])
val sumTo4 : expr =
  App
   (Value
     (Closure ("n",
       If (Eq (Id "n", Value (Int 0)), Value (Int 0),
        Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
       [("sumTo",
         Ref
          {contents =
            Closure ("n",
             If (Eq (Id "n", Value (Int 0)), Value (Int 0),
              Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
             [("sumTo",
               Ref
                {contents =
                  Closure ("n",
                   If (Eq (Id "n", Value (Int 0)), Value (Int 0),
                    Add (Id "n", App (Id "sumTo", Sub (Id "n", Value (Int 1)))))
,
                   [("sumTo",
                     Ref
                      {contents =
                        Closure ("n",
                         If (Eq (Id "n", Value (Int 0)), Value (Int 0),
                          Add (Id "n",
                           App (Id "sumTo", Sub (Id "n", Value (Int 1))))),
                         [("sumTo",
                           Ref
```

```
                            {contents =
                              Closure ("n",
                               If (Eq (Id "n", Value (Int 0)), Value (Int 0),
                                Add (Id "n",
                                 App (Id "sumTo", Sub (Id "n", Value (Int 1)))))
,
                               [("sumTo",
                                 Ref
                                  {contents =
                                    Closure ("n",
                                     If (Eq (Id "n", Value (Int 0)),
                                      Value (Int 0),
                                      Add (Id "n",
                                       App (Id "sumTo",
                                        Sub (Id "n", Value (Int 1))))),
                                     [("sumTo",
                                       Ref
                                        {contents =
                                          Closure ("n",
                                           If (Eq (Id "n", Value (Int 0)),
                                            Value (Int 0),
                                            Add (Id "n",
                                             App (Id "sumTo",
                                              Sub (Id "n", Value (Int 1))))),
                                           [("sumTo",
                                             Ref
                                              {contents =
                                                Closure ("n",
                                                 If
                                                  (Eq (Id "n", Value (Int 0)),
                                                  Value (Int 0),
                                                  Add (Id "n",
                                                   App (Id "sumTo",
                                                    Sub (Id "n", Value (Int 1)))
)),
                                                 [("sumTo",
                                                   Ref
                                                    {contents =
                                                      Closure ("n",
                                                       If
                                                        (Eq (Id "n",
                                                          Value (Int 0)),
                                                        Value (Int 0),
                                                        Add (Id "n",
                                                         App (Id "sumTo",
                                                          Sub (Id "n",
                                                           Value (Int 1))))),
                                                       [("sumTo",
                                                         Ref
                                                          {contents =
                                                            Closure ("n",
                                                             If
                                                              (Eq (Id "n",
                                                                Value (Int 0)),
                                                              Value (Int 0),
                                                              Add (Id "n",
                                                               App (Id "sumTo",
                                                                Sub (
                                                                 Id "n",
                                                                 Value (Int 1)))
)),
                                                             [("sumTo", ...);
                                                              ...])});
                                                        ...])});
                                                  ...])});
                                            ...])});
                                      ...])});
                                ...])});
                          ...])});
                    ...])});
              ...])});
        ...])),
   ...)
val sumToWith : expr =
  Lambda ("i",
   LetRec ("sTW",
    Lambda ("n",
     If (Eq (Id "n", Value (Int 0)), Id "i",
      Add (Id "n", App (Id "sTW", Sub (Id "n", Value (Int 1)))))),
    Id "sTW"))
val sT100 : expr =
  App
   (Lambda ("i",
     LetRec ("sTW",
      Lambda ("n",
       If (Eq (Id "n", Value (Int 0)), Id "i",
        Add (Id "n", App (Id "sTW", Sub (Id "n", Value (Int 1)))))),
      Id "sTW")),
   Value (Int 1000))
val st4 : expr =
  App
   (App
     (Lambda ("i",
       LetRec ("sTW",
        Lambda ("n",
         If (Eq (Id "n", Value (Int 0)), Id "i",
          Add (Id "n", App (Id "sTW", Sub (Id "n", Value (Int 1)))))),
        Id "sTW")),
     Value (Int 1000)),
   Value (Int 4))
val add : expr = Lambda ("x", Lambda ("y", Add (Id "x", Id "y")))
val inc' : expr =
  App (Lambda ("x", Lambda ("y", Add (Id "x", Id "y"))), Value (Int 1))
val five : expr =
  App (App (Lambda ("x", Lambda ("y", Add (Id "x", Id "y"))), Value (Int 1)),
   Value (Int 4))
─( 15:38:34 )─< command 1 >──────────────────────────────────────{ counter: 0 }─
utop # evaluate ;;
```

```
- : expr -> value = <fun>
─( 15:38:39 )─< command 2 >──────────────────────────────────────{ counter: 0 }─
utop # evaluate (App (sumToN, Value (Int 6))) ;;
Error: Unbound value sumToN
Did you mean sumTo, sumTo4 or sumToV?
─( 15:38:46 )─< command 3 >──────────────────────────────────────{ counter: 0 }─
utop # #quit;;
carbon:Hwk_04$ pwd
/project/evw/Teaching/17_Fall_2041/carbon-repos/repo-score100/Hwk_04
carbon:Hwk_04$ c

carbon:Hwk_04$ 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 >──────────────────────────────────────( 15:39:41 )─<
command 0 >─────────────────────────────────────{ counter: 0 ─(─(─(─(─(─(─(─(─(─(─(─(─(─(─(─(─(─(
─(─( 15:39:41 )─< command 0 >──────────────────────────────────────{ counter: 0
─( 15:39:41 )─< command 0 >──────────────────────────────────────{ counter: 0 }─
utop #
┌───┬────────────┬─────┬───────────┬──────────────┬───────┬────────┬──────┬────┐
│Arg│Arith_status│Array│ArrayLabels│Assert_failure│Big_int│Bigarray│Buffer│Call│
└───┴────────────┴─────┴───────────┴──────────────┴───────┴────────┴──────┴────┘
───┘
└───┴────────────┴─────┴───────────┴──────────────┴───────┴────┘ray│Buffer│Call│
└───┴────────────┴─────┴───────────┴──────────────┴───────┴────────┴──────┴────┘
```