```
(* FILE:   inilib.joy *)

LIBRA

    _inilib == true;


(* - - - - -  I N P U T   O U T P U T  - - - - *)

    newline == '\n putch;
    putln == put newline;
    space == '\032 putch;
    bell == '\007 putch;
(* this is now a primitive in raw Joy:
    putchars == [putch] step;
*)
    putstrings == [putchars] step;

    ask == "Please " putchars putchars newline get;

(* - - - - -   O P E R A T O R S   - - - - - *)

    dup2 ==  dupd dup swapd;
    pop2 == pop pop;
    newstack == [] unstack;
    truth == true;
    falsity == false;
    to-upper == ['a &gt;=] [32 -] [] ifte;
    to-lower == ['a &lt; ] [32 +] [] ifte;
    boolean == [logical] [set] sequor;
    numerical == [integer] [float] sequor;
    swoncat == swap concat;

(* date and time *)

    weekdays ==
      [ "Monday" "Tuesday" "Wednesday" "Thursday" "Friday"
          "Saturday" "Sunday" ];
    months ==
      [ "JAN" "FEB" "MAR" "APR" "MAY" "JUN"
        "JUL" "AUG" "SEP" "OCT" "NOV" "DEC" ];
    localtime-strings ==
      time localtime
      [ [ 0 at 'd 4 4 format           ]
        [ 1 at pred months of               ]
        [ 2 at 'd 2 2 format           ]
        [ 3 at 'd 2 2 format           ]
        [ 4 at 'd 2 2 format           ]
        [ 5 at 'd 2 2 format           ]
        [ 6 at [] ["true"] ["false"] ifte    ]
        [ 7 at 'd 5 5 format           ]
        [ 8 at pred weekdays of        ] ]
      [i] map
      popd;
    today ==
      localtime-strings
      [ [8 at] [" "] [2 at] ["-"] [1 at] ["-"] [0 at rest rest] ]
      [i] map
      popd
      "" [concat] fold;
    now ==
      localtime-strings
      3 drop
```

```
      [ [0 at] [":"] [1 at] [":"] [2 at] ]
      [i] map
      popd
      "" [concat] fold;
    show-todaynow ==
      today putchars space now putchars newline;

(* program operators *)

    conjoin == [[false] ifte] cons cons;
    disjoin == [ifte] cons [true] swons cons;
    negate == [[false] [true] ifte] cons;

(* - - - - -  C O M B I N A T O R S  - - - - - *)

    sequor == [pop true] swap ifte;
    sequand == [pop false] ifte;
    dipd == [dip] cons dip;
    dip2 == [dip] cons dip;
    dip3 == [dip2] cons dip;
    call == [] cons i;
    i2 == [dip] dip i;
    nullary2 == [nullary] cons dup i2 swapd;
(* this is now a primitive in raw Joy:
    unary2 == [unary  ] cons dup i2;
*)
    repeat == dupd swap [i] dip2 while;
    forever == maxint swap times;

(* library inclusion *)

    verbose == false;
    libload ==
      [ '_ swons intern body null ]
      [ ".joy" concat include ]
      [ [ verbose ]
        [ putchars "  is already loaded\n" putchars ]
        [ pop ]
        ifte ]
      ifte;
    basic-libload ==
      "agglib" libload
      "seqlib" libload
      "numlib" libload;
    special-libload ==
      "mtrlib" libload
      "tutlib" libload
      "lazlib" libload
      "lsplib" libload
      "symlib" libload;

    all-libload == basic-libload special-libload;

    INILIB == "inilib.joy - the initial library, assumed everywhere\n".
                                        (* end LIBRA *)

"inilib  is loaded\n" putchars.

(* END   inilib.joy *)
```

```
(* FILE:  usrlib.joy  -  if it exists, then it is loaded by default *)
```

```
LIBRA

    RAWJOY1 == "the primitives of the Joy1 system\n";
    _usrlib == true;

                                    (* personalise:
    myname == "Abigail Aardvark";
    myphone == 12345678;
                                    etc *)

HIDE
    returned == "\007\nReturned to Joy\n" putchars
IN
                                    (* unix:
    unix == true;
    control-eof == 'D;
    terminal == "/dev/tty";
    ls == "ls -la" system;
    editor == "vi ";
    escape ==
      "\nTo return to Joy, type:   exit\n" putchars
      "csh" system
      returned;
                                    etc *)

                                    (* vms: *)
    vms == true;
    control-eof == 'Z;
    terminal == "tt:";
    dir == "DIR/DATE" system returned;
    editor == "TECO ";
    escape ==
      "\nTo return to Joy, hit  Control-" putchars
      control-eof putch  '\n putch
      "@tt:" system
      returned;
                                    (* etc *)

    edit ==
      dup editor swap concat system
      dup "Including  " putchars putchars '\n putch
      include
      returned;

    find-in ==
      [ [ [ [unix] first body null not ]
          " " swap concat concat "grep " swap concat system ]
        [ [ [vms] first body null not ]
          swap " " swap concat concat "SEARCH " swap concat system ]
        [ "unknown operating system for  find-in\n" putchars ] ]
      cond
      returned;
    standard-setting == 1 setautoput 1 setundeferror;
    USRLIB == "usrlib.joy - (personal) user library\n"

END.                                (* end HIDE and LIBRA *)

                                    (* demo:
"library"   "*.joy"  find-in.
                                    etc *)

"usrlib  is loaded\n"  putchars.
```

```
standard-setting.

"inilib.joy" include.
                    (* assuming inilib.joy was included:  *)
"agglib" libload.

DEFINE verbose == true. (* Example of over-riding  inilib.joy *)

(* END  usrlib.joy *)
```

```
(* FILE:   lsplib.joy *)

    basic-libload.

LIBRA

    _lsplib == true;

(*   - - -            L I S P   I N T E R P R E T E R            - - - *)

(* REFS: SICP p ???, Henderson p 39, p 101 *)

(* - - - - -                    E V A L                       - - - - - *)

eval ==                                          (*  env  exp           *)
(*
        dup2
        swap
        "eval: env = " putchars put newline
        "      exp = " putchars put newline
*)
    [ list ]

    [                                            (*  eval-compound!     *)
      unswons                                    (*  env args fun       *)
      [ [ QUOTE
          first ]
        [ LAMBDA
          dupd cons [CLOSURE] swoncat ]
        [ IF                                     (*  env  [[i] [t] [e]]  *)
          uncons [eval] dip                      (*  env  e-i [[t] [e]]  *)
          swap                                   (*  env  [[t] [e]] e-i  *)
          [ null ]
          [ pop second]                          (*  env  [e]            *)
          [ pop first ]                          (*  env  [t]            *)
          ifte eval ]
        [ DEF                                    (* env      [name body] *)
          uncons first swap                      (* env      body   name *)
          [ eval ]                               (* env    e-b           *)
          dip                                    (* env    e-b   name    *)
          dup                                    (* env    e-b   n    n  *)
          [ [[] cons] unary2                     (* env   [e-b] [n]      *)
            swons                                (* env   [[n] e-b]      *)
            swons ]                              (* [[[n] e-b] env]      *)
          dip ]                                  (* new-env           n *)
        [ DEFUN                                  (* e  [name vars body]  *)
          uncons                                 (* e      n   [vs b]    *)
          [LAMBDA] swoncat                       (* e      n [L vs b]    *)
          [] cons cons                           (* e     [n [L vs b]]   *)
          [DEF] swoncat                          (* e   [D n [L vs b]]   *)
          eval ]
        [ (* DEFAULT *)
```

```
        swons [eval] map unswons              (*  env  ev-args ev-fun *)
        apply ] ]
    case ]
  [                                           (* eval-atomic          *)
    [ [numerical] [string] sequor ]
      [ ]                                     (* self-evaluating      *)
      [                                       (* lookup!              *)
        dupd swap                             (*  env  v  env         *)
        [                                     (*  member?             *)
          [ null ]
          [ true ]                            (*  fake               *)
          [ first first in ]                  (*  really ?           *)
          ifte ]
        [ [ null ]                            (*  was fake ?         *)
          [ pop ]                             (*  self-evaluating    *)
          [                                   (*  search really      *)
            first unswons rolldown            (*  [e1..] [v1..] v    *)
            [ [first] dip = ]
            [ pop pop first ]
            [ [ [rest] unary2] dip ]
            tailrec ]
          ifte ]
        [ rest ]
        tailrec ]                             (* end lookup!          *)
      ifte ]                                  (* end eval-atomic      *)

  ifte;                                       (* end eval             *)

(* - - - - -                  A P P L Y                    - - - - - *)

apply ==
(*
    dup2
    "apply: fun = " putchars putln
    "       args = " putchars putln
*)
    [ list ]

    [                                         (* apply-compound       *)
      unswons
      [ [ CLOSURE                             (* args [e v body]      *)
          unswons call swons
          uncons swapd uncons                 (* dissect the closure  *)
          [ swap cons                         (* build new frame      *)
            swons ]                           (* install new frame    *)
          dip eval                            (* new-env body         *)
          popd ]                              (* restore old env      *)
        [ "apply: unknown procedure type -\n"
          putchars abort ] ]
      case ]
    [                                         (* apply-atomic         *)
      [ [ CAR first first ]
        [ CDR first rest ]
        [ CONS uncons first cons ]
        [ EQ uncons first equal ]
        [ ATOM first leaf ]
        [ NULL first null ]
        [ LIST (* do nothing *) ]
        [ (* try Joy: *)
          [i] dip call ] ]
      case ]

    ifte;                                     (* end apply            *)
```

```
(* - - - - -                      L I B                        - - - - - *)

lib0 ==
        [
          [ [ FOLDR ]
            [ CLOSURE lib0 [lis ini bin]
              IF [NULL lis] ini
                 [bin [CAR lis]
                      [FOLDR [CDR lis] ini bin] ] ] ]
          [ [ FOLDL ]
            [ CLOSURE lib0 [lis ini bin]
              IF [NULL lis] ini
                 [FOLDL [CDR lis]
                        [bin [CAR lis] ini]
                        bin ] ] ]
          [ [ FOLDR2 ]
            [ CLOSURE lib0 [l1 l2 ini tern]
              IF [or [NULL l1] [NULL l2]] ini
                 [ tern [CAR l1] [CAR l2]
                        [FOLDR2 [CDR l1] [CDR l2] ini tern] ] ] ]
          [ [ RECFOLDR ]
            [ CLOSURE lib0 [x y bin]
              IF [ATOM x]
                 [bin x y]
                 [IF [NULL x]
                     y
                     [RECFOLDR [CAR x]
                               [RECFOLDR [CDR x]
                                         y
                                         bin]
                               bin] ] ] ]
          (* other definitions could go here, candidates are:
                LINREC  BINREC  Y                                      *)
            ];

(* - - - - -              L I S P  (read-eval-print)          - - - - - *)

l-prompt == "L: ";

lisp ==
        [ "\nLisp interpreter\n"
          "\t\tTo include the Lisp library, type\n"
          "\t\t\t[ include  \"OK\"  \"lsplib.lsp\" ]\n"
          "GO\n\n" ]
        putstrings
        lib0                                          (* load lib0    *)
        l-prompt putchars get
        [ "EXIT" = not ]
        [ eval putln
          l-prompt putchars get ]
        while
        pop pop
        "exit from Lisp interpreter\n" putchars;

    LSPLIB == "lsplib.joy - (eval-apply) Lisp interpreter\n".

                                                      (* end LIBRA    *)

"lsplib  is loaded\n" putchars.

(* END   lsplib.joy *)
```