"

<Robson> MDEFS.
November 4, 1976.

Annotations are in italics.
Font 1 of your User.Cm should be a Smalltalk font.
"

"BOOTSTRAPPING"

```
☞true ← ☞true.
☞eval ← ☞eval.
☞print ← ☞print.
☞repeat←iterator.
install 0 repeat 2.

☞denter←☞(
    ☞i ← 0.
    repeat do
        (i<nm length⇒
            (☞i←i+1. cdict enter nm[i] flag+i)
        done))

☞csize←force2 23.
☞v1←vector csize.
☞v2←vector csize.
☞cdict←table.
install csize cdict 0.
install v1 cdict 1.
install v2 cdict 2.

☞nm←☞(a i j k oc). ☞flag←¯1. denter eval.

☞nm←☞(hook statichain classdict classvars initcode mdict runcode instsize tempsize
title ftype freelist)
☞flag ← 037. denter eval.

install cdict class 2.
install cdict vlclsclass 2.

"class[classdict]←vlclsclass[classdict]←table 18 classinit (
    (a i j)
    (hook statichain classdict classvars initcode runcode instsize tempsize)
    ()())
⟨see later for table's classinit clause⟩"

☞ISIT ← ☞(↞?⇒(⇑CLASS's title) ⇑eq CLASS's title 8)

class understands ☞'s  ☞(☞a←8. ↞←⇒(⇑a←:)⇑a eval)
class understands ☞print  ☞(title print)
class understands ☞is ☞(ISIT eval)
class's title ← ☞class
class'stempsize ← 5.

☞csize←force2 9.
☞v1←vector csize.
☞v2←vector csize.
```

```
☞cdict←table.
install csize cdict 0.
install v1 cdict 1.
install v2 cdict 2.

☞nm←☞(a i j). ☞flag←¯1.
denter eval.

☞nm←☞(length hashpart valpart).☞flag ← 037.
denter eval.

☞nm←☞(flags).☞flag←077.
denter eval.

table'S classdict ← cdict.
table'S classvars ← vector 1.
table'S title ← ☞table.
table evals (☞flags←☞(0 037 077). flags[1]←¯1)

table'S initcode ← ☞(
        ☞length← force2 :.
        ☞hashpart←vector length.
        ☞valpart←vector length).

table understands ☞classinit  ☞(☞a←:.
            for i to 3 do
              (for j to a[i] length do
                  (SELF enter a[i][j] flags[i]+j ))
            for j to a[4] length do
              (SELF enter a[4][j] 0777)).
table understands ☞'S  ☞(☞a←⍠. ↞←⇒(⇑a←:)⇑a eval)
table understands ☞is  ☞(ISIT eval)

"table[classdict]←table 9 classinit (
    (a i j)
    (length hashpart valpart)
    (flags)( ))"


                                "SYSTEM CLASSES"

iterator'S (☞initcode ← ☞(
    (↞for⇒
        (☞stop←:.
        ☞start←☞step←1)
    ↞of⇒
        (☞var←(↞valueof⇒(:)⍠)
        ☞start←(↞from⇒(:)1)
        ☞stop←(↞to⇒(:)start)
        ☞step←(↞by⇒(:)1)
        )
    ☞step←0)
    ⇑SELF))
☞cdict ← table 11.
☞nm←☞(iloc iofst exp).☞flag←¯1.
denter eval.
☞nm←☞(var start step stop).☞flag←037.
denter eval.
```

```
iterator's classdict ← cdict.
iterator's title ← ☞iterator.
iterator understands ☞done ☞((☜with⇒(☞iloc←:))CODE 19)
iterator understands ☞again ☞(CODE 20)

☞for ← class.
for's (☞statichain←program itself)
☞cdict ← table 2.
☞nm←☞(start).☞flag←¯1.
denter eval.
for's title ← ☞for.
for's classdict ← cdict.
for's instsize←0.
for's tempsize←1.
for's ftype←1.
for's initcode ← ☞(⇑iterator of valueof ⊡
                    from ☞start←(☜←⇒(:)1)
                    to (☜to⇒(:)start)
                    by (☜by⇒(:)1))

function's classdict←table 8.
function's classdict classinit ☞((a i)
          (fcode fdict ftempsize fstatichain fvars)()()).
function's initcode←☞(
    ☞ftempsize ← (☞a←⊡) length.
    ☞fdict←table ftempsize+(ftempsize+1)/2.
    for i to ftempsize do (fdict enter a[i] i-1).
    ☞fcode←⊡.
    ☞fstatichain←program itself)
function's title ← ☞function.

arec understands ☞'s   ☞(☞a←⊡. ☜←⇒(⇑a←:)⇑a eval)
arec's classdict ← table 17.
arec's classdict classinit ☞(
(a)
(extra resume tempext temp0 temp1 temp2 temp3 arinst ardictionary
code pc max flag caller global message)
()()).
arec's title ← ☞arec.

string understands ☞is     ☞(ISIT eval)
string understands ☞atomize ☞(CODE 8)
string understands ☞print    ☞(
                    0 = ☞x ← SELF[1 to 9999] find first 39⇒
                        (disp←39. disp←SELF. disp←39)
                    SELF[1 to x - 1] print.
                    SELF[x + 1 to SELF length] print)
string understands ☞+ ☞(☞y←:.
                     y is string⇒(☞x←SELF[1 to SELF length+y length].
                        ⇑x[SELF length + 1 to x length]←y[1 to y length])
                      error ☞StringNotFound chars)
string understands ☞= ☞(
                    ☞y←:.
                    y is string⇒
                        (SELF length = y length⇒
                            (for x to SELF length do (SELF[x] = y[x]⇒() ⇑false))
                         ⇑false)
                    ⇑false)
```

```
string'S classdict ←table 5.
string'S classdict classinit ☞((x y)()()(disp)).
string'S title ← ☞string.

vector understands ☞is      ☞(ISIT eval)
vector understands ☞print   ☞(disp←40. SELF sprint. disp←41)
vector understands ☞sprint ☞(
                   for x to SELF length do (disp←32. SELF[x] print)
                   )
vector understands ☞map ☞(☞y←:.
                      for x to SELF length do
                          (apply SELF[x] to y in GLOB))
vector understands ☞+ ☞(☞y←:.
                   y is vector⇒(☞x←SELF[1 to SELF length+y length].
                     ⇑x[SELF length + 1 to x length]←y[1 to y length])
                    error ☞VectorNotFound chars)
vector understands ☞bitsin  ☞(☞x←:. CODE 68)
vector understands ☞smash   ☞(☞x←:. ☞y←:. CODE 69)
vector'S classdict ←table 5.
vector'S classdict classinit ☞((x y)()()(disp)).
vector'S title ← ☞vector.

atom'S classdict ← table 3.
atom'S classdict classinit ☞((x)()()(disp)).
atom'S (☞initcode←☞(CODE 9))
atom understands   ☞chars  ☞(CODE 10)
atom understands   ☞is       ☞(ISIT eval)
atom understands   ☞print  ☞(disp←SELF chars)
atom understands   ☞=    ☞(⇑eq SELF :)
atom understands   ☞swap   ☞(☞x← apply SELF to ☞(eval) in GLOB.
      apply SELF to ☞(←:) in GLOB. ⇑x)
atom'S title ← ☞atom.

number understands ☞print   ☞(SELF>0⇒(nprint SELF)
                              SELF=0⇒(disp←060)
                              disp←025. nprint 0-SELF)
number understands ☞is   ☞(ISIT eval)
number understands ☞| ☞(☞x←:. ⇑x*SELF/x)
number'S classdict ←table 5.
number'S classdict classinit ☞((x y)()()(disp)).
number'S title ← ☞number.

class'S initcode ← ☞(
   ☞title ← AREC'S caller'S arinst.
   (title is atom⇒
      (☞oc←apply title to ☞(nfeval itself) in GLOB.)
    ☞title ← nil. ☞oc ← false)
   ☞a←o.
   ☞i ← a[1]length+a[2]length+a[3]length+a[4]length.
   (i=0⇒()
    ☞classdict ← table i+(i+1)/2.
    classdict classinit a)
   ☞initcode←o.
   ☞j←o.
   (null j⇒() ☞mdict←table (4*j length)/3.
    for i to j length by 2 do (mdict enter j[i] j[i+1]))
    (☞runcode←o.
     null runcode⇒()
```

```
    runcode length=0⇒(☞runcode←nil))
  ☞classvars←vector a[3]length.
  ☞instsize←a[2]length. ☞tempsize←a[1]length.
  ☞j←true.
  (oc is class ⇒
      (instsize≠oc's instsize⇒(☞j←false)
       for i to instsize do
          (☞k ← oc's classdict lookup a[2][i].
           k is false⇒(☞j←false. done)
           k ≠ classdict lookup a[2][i]⇒(☞j←false. done))
      )
   ☞j←false)
  j⇒
      (oc's classdict ← classdict.
       oc's initcode ← initcode.
       oc's mdict ← mdict.
       oc's runcode ← runcode.
       oc's tempsize ← tempsize.
       oc's classvars ← classvars.
       AREC's caller's resume ← nil)
  (oc is class⇒
      (oc's initcode←oc's mdict←oc's classdict←nil.
       oc's (☞classvars ← SELF itself).
       oc's runcode ← ☞(error ☞ObsoleteInstance chars)))
  ☞hook←ordinaryhook.
  ☞ftype←1.
  ☞statichain←program itself)

☞is ← function () (↯?⇒(⇑☞untyped)§⇑false)

☞apply ← function (act mes glb cal)(
  ☞act←:.
  (↯to⇒(☞mes←:))
  (↯in⇒(☞glb←:))
  (↯from⇒(☞cal←:))
  CODE 5)

☞disp ← ☞TTY ← function () (CODE 3 SUB 0)

☞fill ← function (i str) (
  ☞str ← :.
  for i to str length do
    (TTY ← str[i] ← TTY)
  ⇑str)

☞{ ← function (set) (
        ☞set←stream of vector 10.
        repeat do (↯}⇒(⇑set contents) set ← :))

☞float ← class ((x y)(fp1 fp2 fp3)()(disp))(CODE 40)(
  ipart          (CODE 41)
  fpart          (CODE 42)
  +              (CODE 43)
  -              (CODE 44)
  *              (CODE 45)
  /              (CODE 46)
  <              (CODE 47)
  =              (CODE 48)
```

```
    ≤           (CODE 49)
    >           (CODE 50)
    ≠           (CODE 51)
    ≥           (CODE 52)
    min         (CODE 64)
    max         (CODE 65)
    sqrt        (☞x ← 0.00000001. CODE 66)
    is          (ISIT eval)
        )

float's ftype←0.
float 0

float understands ☞ipow ☞(
    ☞x←:. x=0⇒(⇑1.0) x=1⇒()
    x>1⇒(1=x mod 2⇒(⇑SELF*(SELF*SELF) ipow x/2)
        ⇑(SELF*SELF) ipow x/2)
    ⇑1.0/SELF ipow 0-x)

float understands ☞epart ☞(☞x←:. SELF<x⇒(⇑0) SELF<x*x⇒(⇑1)
    ⇑(☞y←2*SELF epart x*x)+(SELF/x ipow y) epart x)

float understands ☞print ☞(SELF=0.0⇒(disp←48. disp←46. disp←48)
        SELF<0.0⇒(disp←025. fprint -SELF)
        fprint SELF)

☞fprint ← class ( (n p q s) () () (disp)) (
    ☞n←:.
    (n<0⇒(☞p← -(10.0/n) epart 10.0)☞p←n epart 10.0)
    ☞n←fprintfuzz + n/10.0 ipow p.
    (n≥10.0⇒(☞p←p+1. ☞n←n/10.0))
    (☞s ← fprintfuzz * 2.
     6>p>¯4⇒
        (☞q←0.
         p<0⇒(disp←fprintz[1 to 1-p])
         ☞s ← s * 10.0 ipow p)
     ☞q←p. ☞p←0)
    iterator for 9 do
        (disp←48+n ipart.
         ☞p←p-1.
         ☞n←10.0 * n fpart.
         p<0⇒
            ((p=¯1⇒(disp←46))
             n<☞s←10.0*s⇒(done)))
    (p=¯1⇒(disp←48))
    q≠0⇒(disp ← 0145. q print))

☞fprintfuzz ← 5.0*10.0 ipow ¯9.
☞fprintz←fill string 4
0.00


☞denter←☞nm←☞flag←☞cdict←nil.

☞break ← function () (CODE 1 SUB :)

☞mem←function (x) (☞x←:.CODE 11)
```

```
withpointer←function (name value exp) (
        name←:. for. value←:. do. exp←:.
        CODE 36.
        apply name to (←CODE 37) in GLOB.
        apply exp to (eval) in GLOB.
        apply name to (←nil) in GLOB.
        CODE 38.)

nprint ← class ((digit n)()()(disp)) (
                n←:.
                n=0⇒()
                digit←n mod 10.
                nprint n/10.
                disp←060+digit)

-←function () (↑:*¯1)

substr←class ((op byte s lb ub s2 lb2 ub2 ins d1 d2)()()()) (
    ub←:.(]⇒()error NoRightBracket chars)
    byte←lb2←ub2←1.
     find⇒(op←(first⇒(1)last⇒(2)1)+(non⇒(2)0).
        byte←:. CODE 2 SUB 0)
     ←⇒  (all⇒(byte←:.op←0. CODE 2 SUB 0)
        s2←⍬.
        s2←apply s2 to (eval) in GLOB.
        op←5.
        [⇒(lb2←:. to. ub2←:. ]. CODE 2 SUB 0)
        ub2←9999. CODE 2 SUB 0)
     replace⇒
        (ins←:. d1←lb. d2←ub.
         ub2 ← s length+ins length-1+d2-d1.
         s2 ← (s is string⇒(string ub2) vector ub2).
         lb ← 1. ub2 ← ub ← d1-1.
         op ← 6. CODE 2 SUB 1.
         lb ← d2+1. ub ← s length.
         lb2 ← d1+ins length. ub2 ← s2 length.
         CODE 2 SUB 1.
         ub ← ins length. ub=0⇒(↑s2) s ← ins. lb ← 1.
         lb2 ← d1.CODE 2 SUB 0)
    op←6. ub2←ub+1-lb.
    s2←(s is string⇒(string ub2) vector ub2)
    CODE 2 SUB 0)

obset ← class((i input)(vec size end)()())(
                end←0. vec←vector size←4) (
    add         ((size=end←end+1⇒(vec←vec[1 to size←size+10]))
                vec[end]←:)
    ←           (0=vec[1 to end] find first (input←:)⇒
                        (SELF add input))
    delete  (0=i←vec[1 to end] find first input←:⇒(↑false)
                vec[i to end]←vec[i+1 to end+1]. end←end-1)
    unadd (input←vec[end]. vec[end]←nil.end←end-1. ↑input)
    vec     (↑vec[1 to end])
    map         (input←:. i←0.
                 repeat do (end<i←i+1⇒(done)
                                each←vec[i].
                                apply input to (eval) in GLOB)  ↑false)
    print       (SELF map (each print. sp))
```

```
    is          (ISIT eval)
    )

☞stream ← class ((in)(i s l)()(disp))(
    ☞s ← ( ↞of⇒ (:) string 10).
    ☞i ← ( ↞from⇒ ((:)-1) 0).
    ☞l ← ( ↞to⇒ (:) s length)
    )(
    ←       (CODE 54)
        " (i=l ⇒ (☞l ← 2*l. ☞s ← s[1 to l])) ☞i ← i+1. ⇑s[i] ← : "
    contents  (CODE 55)
        " ⇑s[1 to i] "
    next        (CODE 56)
        " i=l ⇒ (⇑0) ☞i ← i+1. ⇑s[i] "
    reset       (☞i ← 0)
    is      (ISIT eval)
    end         (⇑i = l)
    print       ((i > 0⇒ (s[1 to i] print)).
                disp ← 1.
                l < i + 1⇒() s[i + 1 to l] print)
    's      (☞in←▫. ↞←⇒(⇑in←:)⇑in eval)
    )()
```

"THE SMALLTALK READ ROUTINE"

```
☞kbd ← function () (⇑kmap[TTY])
☞kbck ← function () (CODE 3 SUB 1)
☞kmap ← string 0377.
    for i←0200 to 0377 do (kmap[i] ← 0177) "ILLEGAL CHARS"
    for i←001 to 0177 do (kmap[i] ← i) "REGULAR ASCII 'S"

                        "CHAR -- KEYBOARD"
kmap[0341]←kmap[0301]←kmap[0274]←kmap[0254]←01.
                                    "≤ -- ↑A or ↑SHF A or ↑, or ↑SHF,"
kmap[0342]←kmap[0302]←kmap[0236]←kmap[0237]←kmap[037]←kmap[036]←02.
                                "  -- ↑B or ↑SHF B or any TOP BLANK KEY"
kmap[0272]←kmap[0273]←03. "▫ -- ↑; or ↑SHF;"
kmap[0344]←kmap[0304]←04. "↑D  /done/ "
kmap[0345]←kmap[0305]←kmap[023]←05. "▯ -- ↑E or ↑SHF E or ↑SHF ESC"
kmap[0346]←kmap[0306]←kmap[0262]←06. "@ -- ↑F or ↑SHF F or ↑2"
kmap[0347]←kmap[0307]←07. "' -- ↑G or ↑SHF G"
kmap[0353]←kmap[0313]←kmap[0245]←013. "Ω -- ↑K or ↑SHF K or ↑SHF 5"
kmap[0356]←kmap[0316]←kmap[0275]←016. "≠ -- ↑N or ↑SHF N or ↑="
kmap[0357]←kmap[0317]←kmap[0242]←kmap[0247]←017.
        "comment char -- ↑O or ↑SHF O or ↑' or ↑SHF '"
kmap[0360]←kmap[0320]←kmap[0271]←020. "➔( -- ↑P or ↑SHF P or ↑9"
kmap[0361]←kmap[0321]←kmap[0261]←021. "! -- ↑Q or ↑SHF Q or ↑1"
kmap[0362]←kmap[0322]←kmap[0300]←kmap[0276]←kmap[0256]←022.
        "≥ -- ↑R or ↑SHF R or ↑SHF 2 or ↑. or ↑SHF ."
kmap[0363]←kmap[0323]←023.          "'s -- ↑S or ↑SHF S"
kmap[0364]←kmap[0324]←024.          "Ω -- ↑T or ↑SHF T"
kmap[0365]←kmap[0325]←kmap[0255]←kmap[0140]←025.
        "▬ -- ↑U or ↑SHF U or ↑- or SHF -"
kmap[0366]←kmap[0326]←kmap[0265]←026. "% -- ↑V or ↑SHF V or ↑5"
kmap[0367]←kmap[0327]←kmap[0376]←027. "˅ -- ↑W or ↑SHF W or ↑SHF 6"
kmap[0370]←kmap[0330]←kmap[0246]←030. "_ -- ↑X or ↑SHF X or ↑SHF7"
kmap[0371]←kmap[0331]←kmap[0277]←031. "➘ -- ↑Y or ↑SHF Y or ↑SHF/"
```

```
kmap[0372]←kmap[0332]←032.  " -- ↑Z  or ↑SHF Z"
kmap[0333]←kmap[0264]←033. "$ -- ↑[ or ↑4"
kmap[0334]←kmap[0267]←034. "& -- ↑\ or ↑7"
kmap[0335]←kmap[0375]←035. " § -- ↑] or ↑SHF ]"
kmap[0337]←kmap[0336]←kmap[0222]←kmap[0212]←kmap[022]←kmap[012]←036.
                "▌ -- ALL LF 'S or ↑ ← or ↑SHF←"
kmap[0257]←0176.          "? -- SHF6 or ↑/"
kmap[0263]←043.          "# -- SHF3 or ↑3"
kmap[0270]←052.          "* -- SHF8 or ↑8"
kmap[0220]←kmap[0210]←kmap[020]←010. "ALL BS 'S"
kmap[0225]←kmap[0215]←kmap[025]←015. "ALL CR 'S"
kmap[0240]←kmap[0230]←kmap[030]←040. "ALL SP 'S"


☞read ← class ((scanner)()
    (read1 tablscan rdnum mknum rdstr rdcomment rtb1 type
     combit crbit sepbit letbit digbit qtbit atbits)
    ()) (
    ☞of⇒(☞scanner ← :.
          (scanner is string⇒(☞scanner ← stream of scanner))
          ☞scanner ← tablscan scanner type.
          ⇑read1 rtb1)
    ⇑disp read) () ()

"Initialization of class variables of class read.  This is done in two parts to avoid an
overflow problem in the bootstrap reader."

read evals (

☞tablscan ← class (
                    (mask)
                    (source type seq isfil nxtchr)
                    ()
                    (rdtb rbuf scanner rdnum rdstr rdcomment
                     read1 rtb1 atbits crbit sepbit flag)
                    ) (
    ☞source ← :. ☞type ← :.
    ☞seq ← stream.
    (source is file⇒(☞isfil ← 1))
    SELF skip)(
    next    (CODE 7)
        "☞mask ← :.
         mask=0⇒
           (☞t ← string 1.  t[1] ← nxtchr.
            ☞nxtchr ← source next.  ⇑t atomize)
         seq reset.  repeat do
           (0=nxtchr⇒(done)
            0=mask&type[nxtchr+1]⇒(done)
            seq ← nxtchr.  ☞nxtchr ← source next)
         ⇑seq contents"
    skip    (☞nxtchr ← source next)
    read    (repeat do (rdtb[nxtchr + 1] eval))
    end     (⇑source end)
    )().

☞rdnum ← class (
                (sign base n fs)
                ()
```

```
                ()
                (nxtchr scanner mknum digbit flag rdnum)
                ) (
  ☞sign ← (nxtchr=025⇒(scanner skip. ¯1)1).
  ☞base ← (nxtchr=060⇒(8)10).
   ☞n←mknum scanner next digbit base. ☞flag←false.
   056=nxtchr⇒
       (scanner skip.
        ☞fs←scanner next digbit.
        0=fs length⇒
           (☞flag←true.
            ⇑sign*n ipart)
        ☞n←n+(mknum fs 10)/10.0 ipow fs length.
        nxtchr=0145⇒
           (scanner skip.
            ⇑n*(10.0 ipow rdnum)*sign)
        ⇑n*sign)
  ⇑sign*n ipart)()().

☞mknum ← function (str base n i) (
  ☞str ← :.
  ☞base ← :.
  ☞n ← 0.0.
  for i to str length do
   (☞n ← (n*base) + str[i]-060)
  ⇑n).

☞rdstr ← class ((t)()()(scanner nxtchr seq qtbit rdstr)) (scanner skip.
    ☞t←scanner next qtbit.
       "At this point the next character is either a string quote or a 0.  If it's a 0, check to
       see if it really belongs in the string, or if the end of the file has been reached
       because of an unmatched string quote."
    nxtchr=0⇒
       (scanner end⇒(error ☞UnmatchedStringQuote chars)
        seq←0. ⇑seq contents+rdstr)
       "It was a string quote -- check for a pair of quotes, indicating a quote mark which
       is part of the string."
    scanner skip.
    nxtchr=047⇒(seq←047. ⇑seq contents+rdstr)
    ⇑t)()().

☞rdcomment ← class (()()()(scanner nxtchr combit rdcomment)) (
     scanner skip. scanner next combit.
       "At this point the next character is either a comment char or a 0.  If it's a 0, check
       to see if it really is part of the comment, or if the end of the file has been reached
       because of an unmatched comment char."
     nxtchr=0⇒
       (scanner end⇒(error ☞UnmatchedCommentChar chars)
        rdcomment. ⇑)
    scanner skip. ⇑) () ().

)

read evals (

☞rtb1 ← vector 256.
☞type ← string 256.
type[1 to 256]←all 0.
```

```
☞combit←2*☞crbit←2*☞sepbit←2*☞letbit←2*☞digbit←2*☞qtbit←1.
☞atbits ← letbit + digbit.

☞read1 ← class ((n v i j)()()(type rtb1)) (
    ☞n ← :. ☞v ← :.
    repeat do
        (☞i ← :.
         for j←1+i to 1+(↜to⇒(:)i) do (type[j]←type[j]⊟. rtb1[j]←v)
         ↜and⇒() done))()().

read1 combit+crbit+qtbit ☞(rbuf←scanner next 0) 0 to 0377.
read1 combit ☞(rdcomment) 017. "comments"
read1 letbit ☞(rbuf←scanner next atbits atomize) 0101 to 0132 and 0141 to 0172.
    "letters"
read1 digbit ☞(rbuf←rdnum. flag⇒(rbuf←☞.)) 060 to 071. "digits"
read1 0 ☞(rbuf←rdnum. flag⇒(rbuf←☞.)) 025. "high-minus"
read1 crbit 0 015. "cr"
read1 sepbit ☞(scanner next sepbit) 011 and 014 and 015 and 040.
    "tab, LF, FF, CR, blank"
read1 qtbit ☞(rbuf ← rdstr) 047. "string-quote"
read1 0 ☞(scanner next crbit) 032. "↑Z (for BRAVO)"
read1 0 ☞(scanner skip. rbuf ← (read1 rtb1) eval) 020. "eval-paren"
read1 0 ☞(scanner skip. rbuf ← read1 rtb1.) 050. "left-paren"
read1 0 ☞(scanner skip. done) 051. "right-paren"
read1 0 ☞(done) 036. "DOIT"
read1 combit+qtbit ☞(done) 0. "null"
        "Null counts as string quote and comment char, as well as DOIT, so that the read
        routine can check for unmatched string quotes and comment chars."

☞read1 ← class ((rbuf rdtb flag)()()(scanner)) (
   ☞rdtb ← :.
   ☞rbuf ← stream of vector 10.
   scanner read.
   ⇑rbuf contents)()().

)
```

"FILES AND DIRECTORIES"

"

A file is found in a directory ('dirinst') by its file name ('fname'), and has a one 'page', 512 character string ('sadr'). 'rvec' is an optional vector of disk addresses used for random page access.

☞fi ← ⟨directory⟩ file ⟨string⟩ old -- finds an old file named ⟨string⟩ in ⟨directory⟩ or returns false if does not exist or a disk error occurs.

☞fi ← ⟨directory⟩ file ⟨string⟩ new -- creates a new file or returns false if it already exists. if neither old or new is specified, an existing file named ⟨string⟩ will be found or a new file created. if ⟨directory⟩ is not specified, the current default directory is used.

⟨directory⟩ file ⟨string⟩ delete -- deletes a file from a directory and deallocates its pages. do not delete the system directory (SysDir.) or bittable (DiskDescriptor.), or any directories you create.

⟨directory⟩ file ⟨string⟩ rename ⟨string⟩ -- renames file named by first string in

⟨directory⟩ with second string. currently not implemented for directory files.

⟨directory⟩ file ⟨string⟩ load -- loads a previously 'saved' memory image (Swat format) and compacted RAM, thereby destroying your current state.

⟨directory⟩ file ⟨string⟩ save -- saves your Smalltalk memory and RAM.

'leader' and 'curadr' are the alto disk addresses of page 0 and the current page of the file, respectively. 'bytec' is a character index into 'sadr'.

'dirty' = 2 if any label block numbers ('nextp' thru 'sn2') have been changed, 1 if 'sadr' has been changed, 0 if the current page is clean. the user need not worry about this unless (s)he deals directly with the label or 'sadr'. it might be noted here that multiple instances of the same file do not know of each others activities or sadr's.

'status' is normally 0, 1 if end occurred with the last 'set'; a positive number (machine language pointer to offending disk command block (dcb)) signals a disk error.

the next 8 numbers are the alto disk label block. 'nextp' and 'backp' are the forward and backward alto address pointers. 'lnused' is currently unused. 'numch' is number of characters on the current page, numch must be 512, except on the last page. 'pagen' is the current page number. page numbers are non-negative numbers, and the format demands that the difference in consecutive page numbers is 1. normal file access starts at page 1, although all files possess page 0 (the 'leader' page). 'version' numbers > 1 are not implemented. 'sn1' and 'sn2' are the unique 2-word serial number for the file.

the function 'ncheck' checks that file names contain alphabetic or 'legal' characters or digits, and end with a period."

```
☞file ← class (
                (x y)
                (dirinst fname sadr rvec leader curadr bytec dirty status nextp
                backp lnused numch pagen version sn1 sn2)
                ()
                (disp curdir)
                ) (
        ☞fname←ncheck :.
        fname is false⇒(error ☞BadFileName chars)
        (null ☞dirinst ← curdir⇒
                (☞dirinst ← directory evals (defdir). dirinst open.)).
                "Set director instance for file. If 'curdir' is nil because file was not
                called from the context of a directory instance, use the default
                directory"
        ☞exists⇒ (CODE 4 SUB 22. ⇑fname)
                "return false is file name does not occur in the directory"
        ☞delete⇒ (CODE 4 SUB 15. ⇑☞deleted)
                "delete a file (see intro) "
        ☞sadr ← (☞using⇒ (:) string 512).
                "set up file string buffer"
        ☞rename⇒ (☞x ← ncheck :.
                x is false⇒(error ☞BadNewName chars ⇑nil)
                file x exists⇒ (error ☞NameAlreadyInUse chars)
                CODE 4 SUB 2. ☞fname ← x. CODE 4 SUB 21.
                SELF set 0 12. SELF ← fname length.
                SELF ← fname. SELF flush. ⇑fname)
                "check that the new name is not already in use. lookup the original
                file and change its name in its directory, and in its leader page"
```

```
        ☙load⇒ (CODE 4 SUB 2. CODE 4 SUB 8)
        (☙old⇒ (CODE 4 SUB 2)
        ☙new⇒ (dirinst'S filinst is file⇒
                (CODE 4 SUB 3) CODE 4 SUB 19)CODE 4 SUB 1)
        ☙save⇒ (SELF set to write 256 0.  SELF reset.
        dirinst close.  CODE 4 SUB 9)
        ☙intostring⇒(⇑SELF intostring)
        dirinst remember SELF
        ) (
←               (☞x ← :. CODE 4 SUB 17)
        " fi ← ⟨number⟩ or ⟨string⟩ "
next            (☙word⇒ (☙←⇒ (☞x ← :. CODE 4 SUB 7) CODE 4 SUB 6)
        " fi next word ← ⟨number⟩ -- possibly increment pointer to word boundary,
        write number.
        fi next word -- read a number "
                ☙into⇒ (☞x ← :. CODE 4 SUB 16)CODE 4 SUB 23)
        " fi next into ⟨string⟩ -- read a string
        fi next -- read a character.  If end of file, then return 0. "
set         (☙to.
                    ☙end⇒(CODE 4 SUB 13)
                    ☙write⇒(☞x ← :. ☞y ← :. CODE 4 SUB 5)
                    ☙read. ☞x ← :. ☞y ← :. CODE 4 SUB 4)
        " fi set to end -- set file pointer to end of file

        fi set to write ⟨number⟩ ⟨number⟩ -- ☞spage←:. ☞schar←:.  schar≥512
        ⇒ (☞spage ← spage + schar/512. ☞schar ← schar mod 512).  Set file
        pointer to spage schar.  If current page is dirty, or 'reset, 'set to end', or
        page change occurs, flush current page, read rvec[spage] or pages
        sequentially until pagen=spage.  Allocate new pages after end if necessary.
        spage=¯1 is interpreted as pagen, i.e. current page.  ☞bytec ← schar.

        fi set to read ⟨number⟩ ⟨number⟩ -- same as write, except stop at end of
        file. "
skipnext        (☞x ← :. CODE 4 SUB 18)
        " fi skipnext ⟨number⟩  -- set character pointer relative to current position
        (useful for skipping rather than reading, or for reading and backing up).
        SELF set to read pagen bytec+: "
end             (CODE 4 SUB 10)
        " return false if end of file has not occurred "
'S          (⇑⍬ eval)
flush           (CODE 4 SUB 12)
        " fi flush -- dirty=0 ⇒ ( )  write current page "
is              (ISIT eval)
remove          (dirinst forget SELF)
        " remove file from filesopen list of directory "
close           ((☙shortened⇒
                    (SELF shorten to pagen bytec)
                    dirinst'S bitinst flush.
                    SELF flush)
                SELF remove. ⇑fname)
        " fi close or ☞fi←fi close (if fi is global) -- flush bittable and current
        page, remove instance from filesopen list of directory "
shorten         (☙to.  ☙here⇒ (SELF shorten pagen bytec)
            ☞x←:. ☞y←:. CODE 4 SUB 14)
        " fi shorten to ⟨number⟩ ⟨number⟩ -- shorten a file.  SELF set to read
        (☞spage←:) (☞schar←:). ☞x←nextp. ☞nextp←0. ☞numch←bytec.
        ☞dirty←2.  SELF flush.  deallocate succeeding pages "
reopen      (CODE 4 SUB 31. dirinst remember SELF)
```

```
        " doesn't look it up unless necessary "
    print       (disp ← fname)
    reset       (CODE 4 SUB 11)
        " fi reset -- reposition to beginning of file.  SELF set 1 0 "
    intostring  (SELF set to end.
                    ☞x ← string bytec + 512 * pagen - 1.
                    SELF reset. ⇑SELF next into x)
        " fi intostring -- return a string containing contents of fi "
    random      (SELF set to end. ☞rvec ← vector pagen.
                    for x to rvec length do (SELF set x 0. rvec[x] ← curadr))
        " fi random -- initialize a random access vector to be used in fi set ... New
        pages appended to the file will not be randomly accessed "
    pages       (☞x←{: : : : : :}. CODE 4 SUB 20)
        " fi pages ⟨number⟩ ... ⟨number⟩ -- out of the same great tradition as
        'mem' comes the power to do potentially catastrophic direct disk i/o (not
        for the faint-hearted). ☞coreaddress←:. ☞diskaddress←:.
        ☞diskcommand←:. ☞startpage←:. ☞numberofpages←:.
        ☞coreincrement←:. if ~1 = coreaddress, use 'sadr'. diskaddress (=-1 uses
        'curadr') and diskcommand are the alto disk address and command.
        startpage is relevant if label checking is performed. numberofpages is the
        number of disk pages to process. coreincrement is usually 0 (for writing in
        same buffer) or 256 for using consecutive pages of core. use label block
        from instance of 'fi'. copy label block from instance. perform i/o call.
        copy 'curadr' and label block into instance. "
    copyto      (☞x←:.
                    repeat do
                    (x←sadr[bytec+1 to numch].
                    nextp=0⇒(done)
                    SELF to pagen+1 0)
                    x shorten to here)
        " fi copyto ⟨file⟩ -- copy SELF into new file "
    )

☞ncheck ← function (str i x) (☞str←:.
    (str is string⇒(0 < str length < 255⇒() ⇑false) ⇑false)
    for i to str length do
        (☞x ← str[i].
        0140 < x < 0173⇒ ("lowercase")
         057 < x <  072⇒ ("digit")
           0 < legal[1 to 6] find x⇒ ("legal")
        0100 < x < 0133⇒ ("uppercase")
        ⇑false)
    x=056⇒(⇑str) ⇑str+ ☞.chars)
        "Check that the file name is a proper length string containing only lower/upper
        case letters, digits, and legal characters.  If name does not end with a period,
        append one."

☞legal ← fill string 6
+-()⇑⇒.


"
A directory is found in a directory ('dirinst'), has a bittable file ('bitinst') for allocating
new pages, a file of file entries ('filinst' -- file names, disk addresses etc.), and a list of
currently open files ('filesopen' which is an 'obset'). each file must ask its directory for
the bittable when page allocation is necessary, and the system directory (via its local
directory) for the disk number.
```

☞di ← ⟨directory⟩ directory ⟨string⟩ old/new

currently, ⟨directory⟩ and old or new must be specified.

'dirname' is the system directory name and 'bitname' is the bittable name. 'curdir' is a class variable bound to the last directory instance 'opened', and provides information 'who called you' (i.e. CALLER) to a file or directory. 'defdir' is a default directory, initially set to dp0, which is invoked when 'curdir' fails to be a directory, i.e. file was not called in the context of a directory, but globally.
"

```
☞directory ← class ((name exp hd)(dirinst bitinst filinst filesopen)(defdir
curdir)(disp)
        )(
        ☞filesopen ← obset.
         ↞device⇒(☞dirinst ← :. ☞filinst ← dirname.
              ☞bitinst ← (1<dirinst⇒(bitname + ☞two chars) bitname))
           ☞dirinst ← curdir. ☞filinst ← :. ☞bitinst ← false.
           ↞new⇒(SELF open new) ↞old. SELF open
              "Store the directory file name in 'filinst'. Subdirectories use 'curdir' as
              their directory, are flagged by 'false bitinst', and then opened. For system
              directories, dirinst is a number indicating disk number, and bitinst contains
              the file name for the appropriate bittable. The directory is not immediately
              opened."
        )(
        file        (SELF open. ⇑apply file)
              "di file ⟨string⟩ -- open directory. create file instance (see file intro) "
        open        (☞curdir ← SELF. filinst is file⇒ ()
                       bitinst⇒(☞filinst ← file filinst old. ☞bitinst ← file bitinst old)
                       ☞filinst ← (↞new⇒(file filinst new) file filinst old).
                       ☞bitinst ← dirinst's bitinst. dirinst remember SELF)
              "di open -- normally not user-called, since access to the directory always
              opens it. Initialize directory file and bittable instances. A 'subdirectory'
              uses the bittable of its system directory and puts itself into that filesopen
              list."
        is          (ISIT eval)
        remember    (filesopen ← :)
        forget      (filesopen delete :)
        print       (disp←0133. filesopen print. disp←0135)
        map         (SELF open. ☞exp ← :. filinst reset.
                     repeat do (filinst end⇒ (done)
                        0 = 1024 *☞hd← filinst next word⇒
                             (filinst skipnext 0 max 2*1023*hd-1)
                        filinst skipnext 10.
                        ☞name ← filinst next into string filinst next.
                        filinst skipnext (2*hd*1023) - 13+name length.
                        exp eval))
              "di map ⟨vector⟩ -- evaluate a vector for each file name"
        list        (SELF map ☞(disp←name. cr))
              "di list -- print the entry names contained in filinst"
        flush       (filesopen map ☞(each flush))
        close       ((filinst is file⇒ (SELF flush. ☞filesopen←obset.
                          ☞filinst ← filinst's fname.
                          dirinst is directory⇒ (dirinst forget SELF. ☞bitinst ← false)
                          ☞bitinst ← bitinst's fname)). ⇑☞closed)
              "di close (e.g. dp0 close) or ☞di ← di close (to release instance) -- close a
              directory by closing all files and directories in its filesopen list and
              deleting it from the filesopen list of its directory. This is currently one way
```

```
        to regain space by closing unwanted file instances, and to change disk
        packs."
    use  (☞defdir ← SELF)
        "di use -- change the default directory"
    's       (⇑⁸₈ eval)
    free     (SELF open. ☞exp ← 0.
              bitinst set to 1  609*dirinst /¯1.
              iterator for 609 do (255 = ☞name ← bitinst next⇒ (☞exp ← exp + 8)
                  name = 0⇒.()
                  repeat do (☞exp ← exp + name *1.
                      0 = ☞name ← name /¯1⇒(done))). ⇑4872 - exp)
        "return the number of free pages in this directory"
    directory  (SELF open. ⇑apply directory))

☞dirname ← fill string 7
SysDir.
☞bitname ← fill string 15
DiskDescriptor.

☞dp0 ← directory device 0
☞curdir ← nil.
dp0 use

☞filin ← class ((fi)()()()) (☞fi←:.(fi is string⇒(☞fi← file fi old⇒()⇑false)).
    repeat do (fi end⇒(done) cr. dsoff. read of fi eval print. dson) fi close)
```

"GRAPHICAL THINGS"

```
☞point ← class ((t)(x y)()())(CODE 23)(
    x        (CODE 24)
    y        (CODE 25)
    ≦        (CODE 26)
    =        (CODE 27)
    +        (CODE 28)
    -        (CODE 29)
    /        (☞t←:.⇑point x/t y/t)
    <        (☞t←:. x<t x⇒(y<t y⇒(⇑SELF)⇑false)⇑false)
    max      (☞t←:. ⇑point x max t x y max t y)
    min      (☞t←:. ⇑point x min t x y min t y)
    is       (ISIT eval)
    print    (☞point print sp x print sp y print)
    )

☞rectangle ← class ((t a b)(origin extent)()())(
    ☞origin ← :.
    ☞extent ← :)(
  has        (☞t ← :.
              ⇑origin ≦ t < origin + extent)
  paint      (CODE 15)
  comp       (SELF paint 14 ¯1)
  inset      (⇑rectangle origin+☞t←: extent-t+:)
  's         (☞t ← ⁸₈.↯←⇒(⇑t←:)⇑t eval)
  is         (ISIT eval)
  moveto     (☞origin ← :)
  moveby     (☞origin←origin+:)
  growto     (☞extent←:-origin)
  center     (⇑origin+extent/2)
```

```
drag        (CODE 17)
dragto      (SELF drag 0 ☞t←:. ☞origin←t)
outline     (☞a←turtle at origin - point 1 1.
             a wide 2.
             a's ink ← (⇙with⇒(:) "3)
             for t to 2 do
                (a turn 90 go extent x + 2 turn 90 go extent y + 2))
print       (☞rectangle print. sp.
             origin print sp extent print)
makebuff    (☞a ← string 2*extent y*☞t←(extent x+15)/16.
             withpointer ☞b for a
                ☞(BLT 0 0 b t 0 0 extent x extent y
                    mem 58 STDISPLAY's displaywidth origin x origin y.)
             ⇑a)
loadbuff    (☞t ← (extent x+15)/16.☞a ← :.
             withpointer ☞b for a
                ☞(BLT 0 0 mem 58 STDISPLAY's displaywidth
                    origin x origin y extent x extent y b t 0 0.))
)()
```

"

class rectangle's paint message is used as follows:

```
☞source ← rectangle upleft widthhheight.    'some rectangle'
☞dest ← point x y.              'some point'
source paint 0 dest.        'copies source to dest pt'
source paint 4 dest.        'copies complement of source to dest pt'
source paint 8 dest gray.   'source brushes gray to dest'
source paint 12 gray.       'fills source with gray'
```

In each case, the effective destination is a rectangle of same size as the source, located with upper left at the point dest. Gray is specified by an integer which gets folded into a 4x4 rectangle to form a pattern which then gets replicated throughout the area being painted. The folding is:

```
A B C D -->  ------     0153727 (binary 1101 0111 1101 0111)
       | A |      makes 1101
       | B |            0111
       | C |            1101
       | D |            0111   (dark gray)
       ------
```

The 'brushing' works by painting gray into the destination wherever the source is black (1), and leaving the destination alone (transparency) wherever the source is white (0).

For each of the 4 operations above, there are 4 modes, which are selected by adding 0-3 to the basic code. These mean:

0 *Store source into destination (paint)*
1 *OR source into destination (merge)*
2 *XOR source into destination (invert)*
3 *AND complement of source into destination (erase)*

"

☞BLT ← function () (CODE 14)

"

BLT allows tranfers from any part of memory to any other, with no bounds checking. It allows one to transfer the bits of a rectangle into a SMALLTALK

string (of the right size!) whence it may of course be further moved to and from the disk, ether, etc. The messages are as follows:

BLT operation code, gray,
dest base addr, dest raster, dest x, dest y, width, height,
source base addr, source raster, source x, source y.

The operation code is as in the 'paint' message. All messages must be present, though some may be ignored, depending on operation. Be sure to terminate the messages with a period. The base address is the starting core location of the screen or of a string. The base address of the display is in mem 58. Since in the Ooze environment the core location of a string is normally not constant, the 'withpointer' function must be used to lock the string in core and provide a pointer to it while BLT is being executed. See rectangle's 'makebuff' and 'loadbuff' for examples. 'Raster' is the number of words of memory between the start of each scan line. For strings, it is 0; for the screen, it is given by STDISPLAY's displaywidth. A final caution: Ooze currently doesn't know that a string written into by BLT is dirty.
"

```
☞textframe ← class (
                (input)
                (window frame last justtabspace scanmode reply buf font)
                (defont)
                ())(
                ☞window←☞frame←:.
                ☞reply←☞scanmode←0. ☞justtabspace←0010004.
                ☞buf←(↜with⇒(:)string 0)
                ☞last←buf length.
                ☞font ←
                    (↜font⇒(☞input←:.input is string⇒(input) defont)defont))(
's              (☞input ← ⍰.↜←⇒(⇑input←:)⇑input eval)
show            (↜of⇒(☞buf←:. ☞last←(↜to⇒(:) buf length)
                        CODE 16 SUB 0)
                 CODE 16 SUB 0)
charofpt        (CODE 16 SUB 1)
ptofchar        (CODE 16 SUB 2)
comp            (window paint 14 ¯1)
put             (☞buf←:. ☞last←buf length. ↜at.
                 window's extent←point 1000 1000. window's origin←:.
                 window's extent←point
                    (SELF ptofchar last + 1 x)-window's origin x
                    SELF fontheight.
                 SELF show)
fontheight      (⇑font[12] + font[14])
scrolln         (☞input←:.
                 ⇑SELF charofpt
                        frame's origin+point frame's extent x SELF fontheight*input)
is              (ISIT eval)
justify         (↜off⇒(☞justtabspace ← justtabspace ⊓077777)
                    ↜on. ☞justtabspace ← justtabspace ⊔0100000)
tab             (☞input ← :. ☞justtabspace ← (justtabspace ⊓0100577) +
                                                              (input ⁄8))
space           (☞justtabspace ← (justtabspace ⊓0177400) + :))


☞dispframe ← class ((n t)(strm text)()())(
                ☞text ← textframe :.
```

```
                    ☞n ← text fontheight.
                    ☞t ← text's frame's extent y.
                    text's frame's extent y ← n*t/n.
                    ↜nosetup⇒()
                    SELF clear.
                    SELF outline)(
←           (☞t ← :. (t = 8⇒(strm's (☞i ← i - 1))strm←t.) SELF show)
show        (repeat do
                (text show of strm's s to ☞t ← strm's i.
                 ☞n ← text's reply.
                 n is point⇒(done)
                 n ≥ t⇒(done)
                 t < ☞n←text scrolln 2 ⇒(done).
                 strm's s←strm's s[n+1 to strm's i] strm's i←strm's i-n))
sub         (☞n ← :.
                 indisp
                    dispframe text's frame inset ☞t←point 10 7 t
                    (n eval).
             SELF show)
read        (SELF←20.
             ☞n ← 0.
             repeat do
                (4=☞t←kbd⇒(⇑☞(done))
                 t=8⇒(n>0⇒(☞n←n-1. strm's (☞i←i-1). kbck⇒() SELF show)
                     SELF show)
                 t=30⇒(strm←30. SELF←13.
                         ⇑read of strm's s[strm's i-n+1 to strm's i-1])
                 strm ← t. ☞n←n+1. kbck⇒() SELF show))
clear       (text's window paint 12 0. ☞strm←stream)
outline     (text's frame outline)
moveby      (text's frame moveby :)
's          (⇑⚬ eval)
is          (ISIT eval)
    )


☞turtle ← class(
                (var)
                (pen ink width dir x xf y yf frame)
                ()
                ()
                )(
                ☞ink ← ¯3. ☞pen ← ☞width ← 1. ☞dir←270.
                ☞frame ← (↜frame⇒(:) STDISPLAY rectangle)
                ☞x ← ☞y ← ☞xf ← ☞yf ← 0.
                SELF place (↜at⇒(:) frame's extent / 2)
                )(
go          (CODE 30)
goto        (CODE 31)
turn        (CODE 32)
←           (CODE 33)
place       (☞pen←0. SELF goto :. ☞pen←1. ⇑SELF)
's          (☞var←⚬. ↜←⇒(⇑var ← :) ⇑var  eval)
wide        (☞width←:.⇑SELF)
pendn       (☞pen ← 1. ⇑SELF)
penup       (☞pen ← 0. ⇑SELF)
black       (☞ink ← ¯3. ⇑SELF)
```

```
    white       (☞ink ← ¯1. ⇑SELF)
    xor         (☞ink ← ¯2. ⇑SELF)
    is          (ISIT eval)
    home        (SELF place frame's extent / 2. ☞dir←270. ⇑SELF)
    erase       (frame paint 12 0. ⇑SELF)
    up          (☞dir ← 270. ⇑SELF)
        )

☞mouse ← class((x)()()()) (☞x←:. CODE 12)
        "
        x = 0-7 are a map on the mouse buttons. E.g. (4=mouse 4) comes back true if the
        top mouse button is depressed, (1=mouse 1)) comes back true if bottom mouse
        button depressed, (7=mouse 7)) comes back true if all three mouse buttons
        depressed, etc. Mouse 8 returns the x coordinate of the mouse and mouse 9 returns
        the y coordinate.
        "
☞mx ← function () (⇑mouse 8)
☞my ← function () (⇑mouse 9)
☞mp ← function () (⇑mouse 10)

☞dsoff ← function () (mem 073 ← 30.)
        "Shortens display to speed up Smalltalk"

☞dson ← function () (mem 073 ← STDISPLAY's displayheight/2.)
        "Undoes dsoff"


☞indisp ← function (disp) (☞disp←:.⇑8 eval)

☞stringof ← function () (
    ☞stringofVariable←:.
    ⇑indisp stream (stringofVariable print. disp contents))

☞ev←class (()()()(disp)) (
    repeat do (cr. disp←stringof apply read to ☞(eval) in GLOB))
```

"THE SMALLTALK SCHEDULER"

```
☞ProcessList←class((a b)(ArecStack StartCode Next Last)()())(
    (↜head⇒(☞Next←☞Last←SELF) ☞Next←:.☞Last←:)
    ☞StartCode ← :.
    )(
    remove      (Last'sNext ← Next. Next'sLast←Last. ⇑Next)
    suspend     (☞ArecStack←:)
    resume      (⇑ArecStack)
    cycle       (⇑Next)
    newcell     (☞Next←ProcessList Next SELF :.
                    Next'sNext'sLast←Next. ⇑Next)
    restart     (☞ArecStack←nil.
                    (program) startwith StartCode)
    count       (☞a←1. ☞b←Next.
                 repeat do (eq SELF b⇒(⇑a) ☞a←a+1. ☞b←b'sNext))
    singular    (⇑eq SELF Next)
    's          (☞a←8.↜←⇒(⇑a←:)⇑a eval)
    is              (ISIT eval))

☞Scheduler←class((action)
```

```
    (CurrentProcess BackgroundProcess)()())(
☞CurrentProcess←ProcessList head ☞BackgroundProcess ← :)(
block       (CurrentProcess suspend AREC'scaller.
                ☞CurrentProcess ← CurrentProcess cycle.
                AREC'scaller ← CurrentProcess resume)
terminate (CurrentProcess singular➔
                    (CurrentProcess'sStartCode ← BackgroundProcess.
                     CurrentProcess restart)
                ☞CurrentProcess ← CurrentProcess remove.
                AREC'scaller ← CurrentProcess resume)
newproc   (CurrentProcess suspend AREC'scaller.
                ☞CurrentProcess←CurrentProcess newcell :.
                CurrentProcess restart)
restart     (CurrentProcess restart)
processes     (⇑CurrentProcess count)
is              (ISIT eval)
setBackground   (☞BackgroundProcess ← :))

☞escape ← function () (SCHED restart)

program'sclassdict enter ☞code 0.
program'smdict ← table 4.
program understands ☞startwith ☞(☞code←:.
        AREC'scaller←AREC'sglobal←AREC'smessage←nil.
        code eval).

☞interrupthandler←class((i j)(iproc icall icode Stack)(actions)())()(
    setaction  (actions[:]←:)
    run     (actions[icode] eval. CODE 59)
    enable (CODE 60)
    disable     (CODE 61))
interrupthandler evals (☞actions←vector 15.
    for i ← 1 to 15 do (actions[i]←☞())
    CODE 62)
(interrupthandler) setaction 1 ☞(☞Stack←stackclimber iproc.
                            repeat do (cr. read eval print))
        " this interrupt happens when the user types ↑C "

(interrupthandler) setaction 2 ☞(☞j ← rectangle point 35*16 0 point 16 16.
    j comp.
    ☞i ← file OozeSaveFile old.  i set to end.
    i set to write i'spagen + 30 max mem 0112 0.  i close.
    j comp.)
        "
        Add more pages to ooze.sv file for more ooze space.  The number of additional
        pages ooze needs is contained in mem 0112.  If this is less than 30, 30 pages are
        added to prevent adding 1 page numerous times.  The rightmost rectangle at the
        top of the screen is complemented while adding pages.
        "
☞OozeSaveFile ← fill string 8
ooze.sv.


(interrupthandler) setaction 3 ☞(☞j ← rectangle point 27*16 0 point 16 16.
    j comp. CODE 63.  j comp)
        "
        Save to core image so that Ooze can recover after a crash.  The next-to-rightmost
        rectangle at the top of the screen is complemented during a save.
```

```
"

(interrupthandler) setaction 4 ☞()
"
```

In the user interface, code is installed for this interrupt to ask the user if this is an infinite loop because between 128 and 256 objects of the same class were just created. This is a warning only. User may turn off further warnings with ☞GrowthAlarm ← ☞off.

```
"


☞timers←class((time icode)()()())(
    ☇reset⇒(CODE 57)
    ☇timeout⇒(☞time←:.☞icode←:.CODE 58))
```

"THE ERROR HANDLER"

```
☞error←class((ptr adr S err)()()())(
        dson. ☞disp←topdisp.
        disp←☞ERROR chars.
        ☞S←stackclimber AREC'Scaller.
        (0=☞adr←mem 0111⇒(dson. cr. disp←:)
         ☞err←stream.
         mem 0111←0. err←0377* mem adr.
         for adr←adr+1 to adr+(mem adr)/-9 do
            (☞ptr←mem adr.
             err←ptr/-8. err←ptr* 0377)
        cr. disp←err contents. cr)
        S display.
        repeat do (cr read eval print).
        SCHED restart)

☞stackclimber←class((scidx pstr)(acode apc amax arec)(disp)())(
    ☞arec←:.SELF setup.☞disp←topdisp
    )(
      caller
                (null arec'S caller⇒
                    (☞no print. sp. ☞caller print)
                ☞arec←arec'S caller.
                SELF setup.
                ☇noprint⇒()
                SELF display)
    display
                (null acode⇒(☞no print. sp. ☞code print)
                 (apc=1⇒()iterator for 3 do (☞. print))
                 for scidx←1 max apc-6 to apc-1 do
                    (sp.acode[scidx] is vector⇒(disp←36)
                     acode[scidx] print)
                 sp. disp ← 031.
                 for scidx←1 max apc to amax min apc+5 do
                    (sp.acode[scidx] is vector⇒(disp←36)
                     acode[scidx] print)
                 (amax<apc+5⇒()iterator for 3 do (☞. print)))
    setup       (☞acode←arec'S code.
                ☞apc←arec'S pc.
                ☞amax←arec'S max)
    evals       (⇑apply ⁸ to ☞(eval) in (arec'Sflag = 0⇒(arec)arec'Sglobal))
```

```
's      (⇑⁰₀ eval)
is      (ISIT eval))
```

"DISPLAY SETUP"

```
☞displaysetup ← class (
        (i)
        (displaycolor displaywidth displayinset displayheight displayrectangle)
        (MINDISPLAYWIDTH MAXDISPLAYWIDTH
        MINDISPLAYHEIGHT MAXDISPLAYHEIGHT
        black white initflag)
        ()
        )(
        SELF set : : : :.
        )(

set     (☞displaywidth ← :. ☞displayheight ← :.
        ☞displayinset ← :. ☞displaycolor ← :.
        (displaycolor = 0 ⇒(☞displaycolor ← white) ☞displaycolor ← black)
        ☞displaywidth ←
            ((displaywidth max MINDISPLAYWIDTH)
                    min MAXDISPLAYWIDTH)|2.
        ☞displayheight ←
            ((displayheight max MINDISPLAYHEIGHT)
                    min MAXDISPLAYHEIGHT)|2.
        ☞displayinset ← (displayinset min MAXDISPLAYINSET) max 0.

        for i ← displayinset to 0 by ¯1 do (
            (displaywidth + displayinset) ≤ (MAXDISPLAYWIDTH+1) ⇒
                (done)
                    ☞displayinset ← displayinset - 1)
        ☞displayinset ← displayinset ⍁ 8.
        (initflag⇒()☺erase.)
        ☞displayrectangle ←
        rectangle point 0 0 point (displaywidth * 16) displayheight.
        CODE 53.
        ☞☺ ← turtle frame displayrectangle.
        ☞displayinset ← displayinset ⍁ ¯8.
        (☟savedisp⇒(☟at⇒(disp's text's frame moveto :)
            disp's text's (☞window ← ☞frame ←
            displayrectangle inset point 2 2 point 2 2)))
        initflag ⇒(☞initflag ← false.) ☺erase. disp outline. disp show
        )
rectangle (⇑displayrectangle)
's      (☞i←⁰₀. ☟←⇒(⇑i ← :) ⇑i eval)
is      (ISIT eval)
refresh (SELF set displaywidth displayheight displayinset displaycolor)
restart (SELF set MAXDISPLAYWIDTH MAXDISPLAYHEIGHT 0 white)
    )


☞cr←class (()()()(disp))(disp←13) "carriage return"
☞sp←class (()()()(disp))(disp←32) "space"

☞t←class ((fool )()()()) (displaysetup evals (
        ☞MINDISPLAYWIDTH ← 12. ☞MAXDISPLAYWIDTH ← 36.
        ☞MINDISPLAYHEIGHT ← 56. ☞MAXDISPLAYHEIGHT ← 808.
```

```
        ☞MAXDISPLAYINSET ← 24. ☞black ← 040000. ☞white ← 0.
        ☞initflag ← true.
        ☞STDISPLAY ←
            displaysetup MAXDISPLAYWIDTH 750 1 white)
    ☺erase. mem 64 ← 227.
    mem 040←070. mem 041←040000. mem 042←0. mem 043←10.
    mem 044←0. mem 045←040000. mem 046←0. mem 047←1.
    mem 0420 ← 040. mem 070 ← 044.
    textframe evals (☞defont ← file fontname intostring).
    ☞disp ← ☞topdisp ← dispframe rectangle point 2 580 point 572 168.
    disp ← version.
    program's runcode←☞(SCHED restart)
    ☞SCHED←Scheduler ☞(cr read eval print)
    SCHED restart)

☞version←fill string 39
Welcome to the wide open spaces of OOZE

☞fontname←fill string 12
ST10.STRIKE.
☞fill ← function () (program's (☞runcode←☞(t) CODE 22))
fill
```