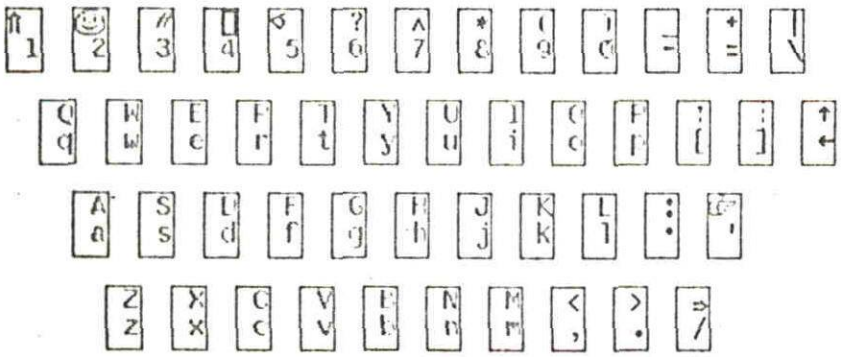
SMALLTALK -- Version 7-16-74

SYSDEFS - TABLE OF CONTENTS


A: addto - 8 again - 2 apply - 7
atom - 3
B: base8 - 5 button - 11
C: classprint - 25 core - 6 cr - 7
D: dclear - 16 dir - 23 disp - 6/26
dispframe - 14 dmove - 16 dmovevec - 16
done - 2 do - 3 directory - 21
DRIBBLE - 26 dsoff - 7 dson - 7
E: edit - 11 error - 10 eq - 5
ev - 4 eval - 7
F: falseclas - 3/4 file - 18 filin - 1/26
filout - 26 fix - 16 for - 3
G: GET - 3
H:
I: if - 3 isnew - 1 is - 7
ISIT - 5
J:
K: kbd - 6/25 kbck - 11
L: leech - 3
M: mem - 6 mouse - 6 mx - 6 my - 6
N: nil - 5 null - 5 number - 5/7/21
nprint - 7
O: obset - 8
P: pshow - 8 PUT - 3 print - 1
Q:
R: read - 1/26 reed - 8 redo - 16
repeat - 2
S: sequence - 8 show - 8/25 showev - 25
sp - 7 stop - 1 string - 4/7
substr - 7
T: true - 7 TTY - 6 type - 26
TURTLE COMMANDS - 5
U: USER 3/20
V: vecmod - 7 vector - 4/7 vectorize - 5
veced - 11
W:
X:
Y:
Z:

SPECIAL CHARACTERS:
: - 2 ◁ - 2 ⇑ - 2
☞ - 2 - - 5 # - 1


note: page numbers are absolute given the following ordering of files: NEWDEFS ERR SHORTEDIT DISPLAY FILESMALL NREADER NMAP.



```
xplot

□k draw 200 16!

□do 128 (N print, disp←N, sp.)!
1⇒ 2~ 3% 4} 5{ 6@ 7    9; 10! 11□ 12" 13
 14& 15 16 17 18∘ 19┐ 20□ 21∑ 22ᴹ 23∨ 24∧ 25≥ 26≤ 27≠
28 29 30π 31£ 32  33ñ 34é 35# 36□ 37∅ 38∧ 39' 40( 41)
42* 43+ 44, 45- 46. 47/ 480 491 502 513 524 535 546 557
568 579 58: 59• 60< 61= 62> 63≥ 64☺ 65A 66B 67C 68D 69E
70F 71G 72H 73I 74J 75K 76L 77M 78N 79O 80P 81Q 82R 83S
84T 85U 86V 87W 88X 89Y 90Z 91[ 92\ 93] 94↑ 95← 96 97a
98b 99c 100d 101e 102f 103g 104h 105i 106j 107k 108l
109m 110n 111o 112p 113q 114r 115s 116t 117u 118v 119w
120x 121y 122z 123; 124| 125; 126? 127 128
□xplot 'pix' 1!
```

```
to class x y ()
to number x y :: nprint ()
to vector x y :: substr ()
to printing (CODE 38). printing ←.
to atom x y (CODE 29)
to string x y :: substr ()
to float x y :: nprint ()
to falseclass x y (isnew)
to isnew (CODE 5)
←false←falseclass.
←(TITLE USER DO SIZE CODE SELF AREC GLOB MESS RETN CLAS
 length eval or and mod chars error
←.,/·:-[]?|↑⍋#□◁ ←=|||*+⊃<>)
```

*'DONT EDIT ABOVE HERE!--These classes and atoms mentioned early to guarantee addresses for machine code.*

*On NOVAs, printing 0 turns off listing, printing 1 turns it back on again.*

```
to isnew (null instance?("instance←allocate permsize.
      instance[0]←class. !true)
!false).
```

## *BOOTSTRAPPING MAGIC.*

*HEREWITH A SOMEWHAT WHIMSICAL ANNOTATED VERSION OF SYSDEFS. ANNOTATIONS ARE IN ITALICS. WHILE IT IS HOPED THAT THIS WILL PROVIDE SOME ELUCIDATION OF THE CODE ESCAPES, OBSCURITIES WILL NO DOUBT PERSIST . THE ANNOTATIONS ARE INTENDED TO BE BUT DIMLY LIGHTED MARKERS ON THE ROAD TO TRUE ILLUMINATION.'*

```
to print (◁..)
```

*':x.Print its address in octal.*
*Printing goes to the same place as CODE 20.*
*This is used primarily for bootstrapping.*
*All system classes will print themselves.'*

```
to read (CODE 2)
```

*'Read keyboard input into a vector. This is almost identical in function to the SMALLTALK read routine, except that DOIT is signalled by <CR> at zero-th parenthesis level, and single-quote strings are ignored. It is only available in Nova versions.'*

```
to fillin x (:x. CODE 16 x)
```

*'Patches the keyboard input to be read from the file specified. This is only for Novas, and the filename can~t have an extension. fillin"foo will read from FOO. and revert to the keyboard upon EOF.'*

```
to stop (CODE 31)
```

*'Writes the entire SMALLTALK workspace out to the file VMEM (on Novas only). Thus when SMALLTALK is next started on that Nova, it will begin with that workspace. It is good to keep a backup copy of VMEM for this reason.'*

```
to # (:#)
```

*'Returns a REFERENCE to its argument~s binding.'*

```
printing 0. print #number. print #atom. print #vector. print #string.
print #false. print #class. print #float. printing 0.
```

## 'MESSAGE HANDLING'

```
to ↕ (CODE 18)

    '     to ↕ name

    %" ?(↕"name nil ?(!name←caller message quotefetch)
                    (!caller message quotefetch)

    Fetch the next thing in the message stream unevaluated
    and bind it to the name if one is there.

    %# ?(↕"name nil ?(!name←caller message referencefetch)
                    (!caller message referencefetch)

    Fetch the reference to next thing in the message stream
    and bind it to the name if one is there.

        (↕"name nil ?(!name← caller message evalfetch)
             !caller message evalfetch)

    Fetch the next thing in the message stream evaluated
    and bind it to the name if one is there.'

to ◁ (CODE 17)

    '↕"token. token←caller.message.code[caller.message.pc]?
         (caller.message.pc←caller.message.pc+1. !true) !false.
    That is, if a match for the token is found in the message, then
    gobble it up and return true, else return false.'

to ⇃ (CODE 13)

    '↕x. then do a return, and apply x to any further message.
    Note that in (... !x←3. "y←y-2), the assignment to y will never
    happen, since ! causes a return.'

to ☞ (CODE 9)

    '!↕". That is, get the next thing in the message
    stream unevalled and active return it (which
    causes it to be applied to the message).'
```

## 'CONTROL CLASSES'

```
to repeat token (↕#token. CODE 1)

    'repeat (token eval) Not a true apply to "eval,
    and therefore token MUST be a vector.'

to done x (◁with⇒(↕x. CODE 25) CODE 25)

    'done causes a pop out of the nearest enclosing repeat , for, or do.
    "done with val" will cause the repeat to have value val'

to again (CODE 6)

    'repeat ("active←active caller. eq active.
    class #repeat?(done)). That is, redo the most
    recent repeat, for, or do loop.'
```

```
to if exp (:exp⇒(◁then⇒(:exp. ◁else⇒(:↱. exp)exp)error ↱(no then⇖))
          ◁then⇒(:↱.  ◁else⇒(:exp) false)error ↱(no then⇖))

    'The ALGOL "if ... then ... else ..."'

to for token step stop var start exp (
    :↱var. (◁←⇒(:start.)↱start←1).
    (◁to⇒(:stop.)↱stop←start.)
    (◁by⇒(:step.)↱stop←1.)
    ◁do. :#exp. CODE 24)

    'An Algol-like "for". Note the default values if
    "←","to","by",etc., are omitted.
    CODE 24 means --repeat(exp eval).
    This implies "done" and "again" will work,
    which is correct.'

to do N exp (:N. :#exp. for N to N do (exp eval.))

    '"N" is made available and contains the current
    iteration number.'
```

## 'INITIALIZING SYSTEM CLASSES'

```
    'Here are the main kludges which remain from
    the time when we really didn~t understand classes
    very well, but wanted a working SMALLTALK.
    PUT and GET are two of the principle actions of class
    class. The new verson of SMALLTALK will have
    class as a class with these actions intensional.'

to PUT x y z (:#x. :y. :z. CODE 12)

    'The first argument MUST be an atom which is bound
    to a class table. The third argument is installed
    in the value side of that table corresponding to the
    name (atom) which was the second argument.'

to GET x y (:#x. :y. CODE 28)

    'If "x" is a class table then the binding of
    the atom in "y" will be fetched.'

to leech field bits : ptr (
    lsnew⇒(:ptr)
    CODE 27)

    'Lets you subscript any instance
    a[0] gives you the class, a[1] gives the first field, etc.
    a[2] gives you the pointer; a[2]& returns the BITS in an integer
    a[2]←foo will dereference count previous contents,
    but a[2]&←foo will not.'

PUT USER ↱TITLE ↱USER
PUT falseclass ↱TITLE ↱false

PUT atom ↱DO ↱(CODE 29

    '%←?(:x.  !x -- Lookup SELF and replace its value by x.)
    %eval?(! -- Lookup the binding of SELF)
    %←?(!SELF←)
    %chars?(! -- printname of SELF (a string))'

    ◁is⇒(ISIT eval)
```

```
⊲print⊳(disp←SELF chars) )

'Done this way (PUT used rather than using "to") because
we wanted to know where the system classes are.
Hence the initial "to atom x y ( )" , for example,
in "Bootstrapping Magic" followed by the
behavior here.'

'THE SMALLTALK EVALUATOR!!!!!!!!'

    to ev (repeat (cr read eval print.)). ☞cr←6

'Later in the boot there will be "to cr(disp←13)",
which will actually do carriage returns when
a display frame has been instanced.'

PUT falseclass ☞DO ☞(CODE 11

'%?? (:".)
%or? (!:)
%and? (:.)
%<? (:.)
%=? (:.)
%>? (:.)'

⊲is⊳(ISIT eval)
⊲print⊳(☞false print) )

PUT vector ☞DO ☞(CODE 3 ⊳(⋂substr SELF x GLOB MESS)

'isnew?(Allocate vector of length :.
    Fill vector with nils.)
%[?(:x. %].
    (%←?(:y. !y -- store y into xth element. )
    ! xth element) )
%length?(! length of string or vector)
%eval?("pc←0. repeat
    (null SELF["pc←pc+1]?(done)
    "val←SELF[pc] eval)
    !val) sort of...'

⊲is⊳(ISIT eval)
⊲print⊳(disp←40. for x to SELF length
    (disp←32. SELF[x] print). disp←41) )

PUT string ☞DO ☞(CODE 3 ⊳(⋂substr SELF x GLOB MESS)

'isnew?(Allocate string of length :.
    Fill string with 0377s.)
%[?(:x. %].
    (%←?(:y. !y -- store y into xth element. )
    ! xth element) )
%length?(! length of string or vector)'

⊲is⊳(ISIT eval)
⊲print⊳(disp←39. disp←SELF. disp←39)
⊲=⊳(:y is string⊳(SELF length=y length⊳(
    for x to SELF length (SELF[x]=y[x]⊳() ⋂false)) ⋂false)
    ⋂false)
⊲+⊳(:y is string⊳(☞x←SELF[1 to SELF length+y length].
    ⋂x[SELF length+1 to x length]←y[1 to y length])
    error ☞(string not found))
⊲fill⊳(☞x←1. repeat(
    10=SELF[x]←disp←kbd⊳(done)
    ☞x←x+(SELF[x]=8⊳(x=1⊳(0)-1)1)
    x>SELF length⊳(done))) )
```

```
to vectorize reader (↑reader ← sequence :reader 0. ⇑reed)
      'Makes a vector out of a string a la read.'

PUT number ↑DO ↑(CODE 4

      '%+?(!val+:)
      %-?(!val-:)
      %*?(!val*:)
      %/?(!val/:)
      %<?(!val<:)
      %=?(!val=:
      %>?(!val>:)
      %&?(%+?(!val OR :)
            %-?(!val XOR :)
            %*?(!val AND :)
            %/?(!val LSHIFT :))'

      ◁is⇒(ISIT eval)
      ◁print⇒(SELF>0⇒(nprint SELF)
            SELF=0⇒(disp←060)
            disp←055. nprint 0-SELF) )

      'For floating point stuff see FLOAT'

to - (0-:)

      'An often used abbreviation.'

to base8 i x s (:x. ↑s←string 6. for i to 6
      (s[7-i] ← 060 + x ^* 7. ↑x ← x ^/ -3). ⇑s)

      'Returns a string containing the octal representation (unsigned)
      of its integer argument.'


↑ISIT ← ↑(◁?⇒(⇑TITLE) ⇑TITLE=:↑).

to nil x (#x)

      'nil is an "unbound pointer", which is used
      to fill vectors and tables.'

to null x (:x. 1 CODE 37)

      'Null returns true if its message is "nil",
      otherwise false.'

to eq x (CODE 15)

      '(!:x is-identical-to :) - compare 2 SMALLTALK pointers.'
```

## 'TURTLE COMMANDS'

turn

```
to right (CODE 21)
to go (CODE 22)
to goto (CODE 36)
to TURTX x (:x. CODE 23)
to ink x (:x. CODE 32 x)
to penup (TURTX 0)
to pendn (TURTX 1)
to home (TURTX 2)
to up (TURTX 3)
to erase (TURTX 4)
```

```
to black (ink 0-1)
to white (ink 0)
to turtle z,i pen ink dir x xf y yf dx dxf dy dyf (
    isnow⇒(0 CODE 38)
    knows⇒(↑(:↑.)eval)
    1 CODE 39 :z. 0 CODE 39 ↑z)
```

'0: *Load SELF from THEturtle state.*
1: *Load THEturtle state from SELF.*

## *UTILITIES'*

```
to mem x y (:x. CODE 26)
    'to mem x y (:x. %←?(!core/mem x ←:)!core/mem x)
    mem loads integers from and stores them into real core.
    Tee hee...
    mem 0430 ← 0  ;set alto clock to zero
    mem 0430  ;read the clock
    for i to 16 (mem 0430←1 ← cursor[i])  ;put new bits into cursor
    mem 0424 ← mem 0425 ← 0.  ;reset mouse x and y to 0.
    mem 0102 ← 0.  ;disconnect cursor from mouse
    mem 0426 ← x. mem 0427 ← y.  ;move the cursor
    mem 0100 ← 0177.  ;make DEL the interrupt char (instead of ESC).
    mem 0420.  ;get pointer to display control block
    mem 0177034.  ;reads the first of 4 keyboard input words.'

to mouse x (:x. CODE 35)

    ' 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.'
to mx (↑mouse 8)
to my (↑mouse 9)

to core ((mem 59)-mem 58)

    'Returns the amount of space left in your Smalltalk.'
to kbd (0 CODE 20)

    'Waits until a key is struck.
    Returns an ascii code when a key is struck on the keyboard.
    Use to kbck (1 CODE 20) to return true if kbd has a character,
    otherwise false.
    Used in multiprocessing.'

to disp x i (
    ◁←⇒(:x is string⇒(for i to x length (TTY←x[i])) TTY←x)
    ◁clear⇒() ◁sub⇒(:x eval))

    'This disp is used for bootstrapping.
    Later in these definitions (READER)it will
    be restored to an instance of "display frame."'

to TTY (0 CODE 20)

    'TTY←(integer) will print an ascii  on the Nova tty.
    On altos, TTY  prints in little error window at bottom
    of screen.'
```

```
to dsoff (mem 272←0)

    'Turns display off by storing 0 in display control block ptr.
    Speeds up Alto Smalltalk by factor of 2.'

to dson (mem 272 ← mem 62)

    'Turns display back on by refreshing display
    control block pointer.'

to apply x y (:#x. ◁to⇒(:y. CODE 10) CODE 10)

    'Causes its argument to be applied to the message stream of the
    caller, or, in the case of apply foo to ‹vector›, to that vector.
    Note that only the message is changed, and that the caller is
    not bypassed in any global symbol lookup. Dan knows that
    this really needs to also allow ‹contexts› and ‹returns› to be
    specified.'

to cr (disp←13). to sp (disp←32)

☞true←☞true
☞eval←☞eval
to is ( ◁?⇒(⋂☞untyped):☞. ⋂false)

    'These are used to handle messages to classes which
    can't answer queston invoking "is", "eval", etc.'

to t nprint substr (ev). t 'prevent -to- from making these global.'

to nprint digit n (:n=0⇒()
    ☞digit←n mod 10. nprint n/10. disp←060+digit)
PUT number ☞nprint #nprint.

    'Prints (non-neg) integers in decimal
    with leading zeroes suppressed!'

to substr op byte s lb ub s2 lb2 ub2 (
    :#s. :lb. :ub. :MESS. ☞GLOB←ub. 'tee hee'
    :ub. (◁]⇒() error ☞(missing right bracket))
    ☞byte ← ☞lb2 ← ☞ub2 ← 1.
    ◁find⇒ (☞op ← (◁first⇒(1) ◁last⇒(2) 1) + (◁non⇒(2) 0). :byte. CODE 40)
    ◁←⇒ (◁all⇒ (:byte. ☞op←0. CODE 40)
        :#s2. ☞op←5.
        ◁[⇒ (:lb2. ◁to. :ub2. ◁]. CODE 40)
            ☞ub2←9999. CODE 40)
    ☞op ← 6. ☞ub2 ← ub+1-lb.
    ☞s2 ← (s is string⇒(string ub2) vector ub2). CODE 40).
PUT string ☞substr #substr.
PUT vector ☞substr #substr.
done

    'substr takes care of copying, moving and searching
    within strings and vectors. It first gets its father (string/vector)
    and the lower bound, and then proceeds to fetch the rest of the
    message from above. Some examples:
        "(a b c d e)[2 to 3] -> (b c)
        "(a b c d e)[1 to 5] find "c -> 3
        "(a b c d e)[1 to 5] find "x -> 0
    See vecmod for more examples. String syntax is identical.'

to vecmod new end old posn ndel nins ins (☞end←10000.
    :old. :posn. :ndel. :ins.
    ☞nins←(ins is vector⇒(ins length-1) null ins⇒(0) 1).
    ☞new ← old[1 to old length+nins-ndel].
```

```
      (ins is vector⇒(new[posn to end] ← ins[1 to nins]) new[posn]←ins).
      new[posn+nins to end] ← old[posn+ndel to end].
      ⇑new)

      'Vecmod makes a copy of old vector with ndel elements deleted
      beginning at posn.  If ins is a vector, its elements are inserted
      at the same place.  It is the heart of edit.'

to addto func v w (:#func. :w. ☞v←GET func ☞DO. null v⇒(error ☞(no code))
      PUT func ☞DO vecmod v v length 0 w)

      'Addto appends code to a class definition.'
```

## 'READING SMALLTALK'

```
to sequence : str ptr (
      ◇scan⇒(CODE 41⇒(⇑rPar))
      ◇pos⇒(◇←⇒(⇑:ptr)⇑ptr)
      ◇fill⇒(str fill. ☞ptr←0)
      ◇is⇒(ISIT eval)
      isnew⇒(:str. :ptr) )

      'Sequences are used to build the main read routine.
      CODE 41 advances pointer to the end of the next token in string,
      and returns the internal representation of that token.'

to reed v s : t ownv lPar rPar subreed (◇knows⇒(ev)
      ☞v←ownv. ☞s←v length. ⇑subreed 1 1)
reed knows
to subreed i j t (:i.
      for j ← :j to s (☞t←reader scan.
            rPar=t⇒(v[j]←nil. ⇑v[i to j])
            lPar=t⇒(☞t←subreed j j. v[j]←t)
            v[j]←t)
      ☞v←v[1 to ☞s ← s + (s>core⇒(50)s)]. ⇑subreed i j)
☞ownv←vector 200.
☞lPar←(sequence (string 1) fill 0) scan
(
☞rPar←(sequence (string 1) fill 0) scan
)
done

to obset i input : vec size end (
      ◇←⇒(0=vec[1 to end] find first :input⇒
            (end<size⇒(vec[☞end←end+1]←input)error ☞(obset full)))
      ◇delete⇒(0=☞i←vec[1 to end] find first :input⇒()
            vec[i to end]←vec[i+1 to end+1]. ☞end←end-1)
      ◇print⇒(SELF map ☞(vec[i] print. sp))
      ◇map⇒(:input. for i to end (input eval))
      ◇is⇒(ISIT eval)
      isnew⇒(☞end←0. ☞vec←vector :size+1)
      )
```

## 'SHOWING SMALLTALK'

```
      'A more grandiose "show" follows after class dispframe
      (see READER) is defined.'

to show func t (
      :#func. ☞t←GET func ☞DO.
      null t ⇒ (⇑☞(no code)) pshow t 0.)
to pshow ptr dent i t :: x tabin index (◇knows⇒(ev)
```

```
    :ptr :dent.
    (ptr length>4⇒(tabin dent)) disp←40.
    for i to ptr length-1
        (@t ← ptr[i].
        t is vector ⇒(pshow t dent+3.
            i=ptr length-1⇒()
            @. = @x←ptr[i+1]⇒()
            x is vector⇒()
            tabin dent)
        i=1 ⇒(t print)
        0<@x←index @(. , ' [ ] ⇒) t⇒
            (x=1⇒(t print. ptr[i+1] is vector⇒() tabin dent) t print)
        0=index @(: @ # ⋔ [ ◇ ⇒ ∧) ptr[i-1]⇒(disp←32. t print)
        t print)
    disp←41)
pshow knows
to tabin n i (disp←13. for i to n (disp←32))
.to index op byte s lb ub s2 lb2 ub2 (
    :s. :byte. @op←@lb←@s2←@lb2←@ub2←1. @ub←9999. CODE 40)
done
    'A piece of substr which runs faster.'

'NOVA FILE MAINTENANCE -- DP0:SYSDEFS.AN  DP1:SYSDEFS.AN '
printing 1. stop.
```

```
printing 0.
to error adr ptr arec class :: c shocode find sub (
      (0=☞adr←mem 0105⇒(◇knows⇒(ev ⋂) dson. :ptr))
      ☞arec←leech AREC.
      disp sub ☞((0=adr⇒(ptr print)
            mem 0105←0. repeat (☞ptr←mem adr.
                  0=disp←ptr ∧/ -8⇒(done)
                  0=disp←ptr ∧* 255⇒(done) ☞adr←adr+1))
      cr. c. ev))
error knows
to c class code cpc (
      null arec[5]⇒(.) ☞arec←leech arec[5]. ☞class←arec[0].
      (GET class ☞TITLE) print. ☞: print.
      find arec[1]∧ GET class ☞DO ⇒ (shocode)
      find arec[1]∧ arec[6] ⇒ (shocode).
      )
to shocode i (
      for i←1 to code length
            (i<cpc-5⇒(disp←056) i>cpc+5⇒(disp←056)
            sp. (i=cpc⇒(disp←1))
            code[i] is vector⇒(☞□ print) code[i] print).
      )
to find adr vec vadr i ( 'a tree search in vec for the address adr'
      ☞adr←t. ☞i←leech tvec.
      vec is vector is false⇒(⋂false)
      ☞vadr←(leech i)[1]∧ +1.
      (adr>vadr⇒(adr<vadr+vec length+1⇒
            (☞cpc ← adr-vadr. ☞i←0. ☞code←vec. ⋂true)))
      ☞i←0. for i to vec length
            (vec[i] is vector⇒(find adr vec[i]⇒(⋂true)))
      ⋂false)
to sub disp (☞disp ← GET USER ☞disp. (:) eval)
done
printing 1.
```

```
printing 0.

to kbck (1 CODE 20)
      'Returns true if the keyboard has been hit'

to button n (↑in=mouse 7)
      'Returns true if that pattern is being held down'
```

## 'THE SMALLTALK EDITOR - -'

```
to edit func t (:#func.
      ☞t←GET func ☞DO.
      null t ⇒ (↑☞(no code))
      ◁title⇒ ((veced classprint func header) eval)
      PUT func ☞DO veced t.
      ↑☞edited)

      'Edit picks up a code vector, makes sure it is not empty
      and calls veced to edit the code body.  If you say edit foo title,
      veced will edit the header as well, and the changed form will be
      evalled upon exit to redefine the function, title and all.

      Veced can be used on any vector, and is used by FIX as well
      as EDIT.  It creates two new windows within the default DISP
      which exists when it is called.  One is used for a menu of
      commands, the other becomes the new default window DISP.
      The new default is passed to an intermediary; and the newly
      edited vector is returned.'

(to veced back newdisp  menu x ll  menuwidth menulen menuvec
ed ed1 edpush edtarget gettwo bugin  (
      ◁knows⇒(ov)
      ☞back←false.
      disp fclear.
      disp param (☞menu←dispframe winx+winwd-menuwidth menuwidth winy winht
                  string 70.
      mem 0425 ← winy + 105.
      for x to menulen do (menu←32. menu ← menuvec[x] chars.  menu←13)
            ☞newdisp ← dispframe winx winwd-menuwidth winy winht
                  string buf length - 100)
      tx. ☞x ← ed newdisp x.
      disp show.
      ↑x)  )


veced knows

☞menuwidth ← 64.
☞menuvec  ← ☞( Add Insert Replace Delete Move Up Push Enter Leave Exit)
☞menulen ← 10.

to ed disp (:disp. apply ed1)
      'Ed is used to make DISP a new local.'

to ed1 ptr l n nrun command temp i (
      ☞command ← 0.
      :ptr.
      repeat(
            ☞l←ptr length.
            back⇒(done with ptr)
            mem 0424 ← menu param (winx + winwd/2).
            (command=126⇒(disp←126)
                  disp clear
                  for n to l-1
```

```
                (disp←32.
                ptr[n] is vector⇒(disp←044)
                ptr[n] print))
        cr. disp←052.

        ☞command ← (bugin menu 2*menulen)/2.
        menuvec[command] print.
        mem 0424 ← disp param (winx + winwd/2).

        ☞(
        (☞ptr←vecmod ptr 1 0 read)
        (☞ptr←vecmod ptr edtarget 0 read)
        (gettwo. ☞ptr←vecmod ptr n nrun read)
        (gettwo. ☞ptr←vecmod ptr n nrun nil)
        (gettwo. ☞temp ← ptr[n to n+nrun]
              temp[nrun + 1] ← nil.
              ☞i←edtarget.
              ☞ptr←vecmod ptr n nrun nil.
              (i>n ⇒ (☞i←i-nrun))
              ☞ptr←vecmod ptr 1 0 temp)
        (ptr[☞n←edtarget] is vector ⇒
                (☞ptr←vecmod ptr n 1 ptr[n])
                ☞command ← 126)
        (gettwo. edpush)
        ( ptr[☞n←edtarget] is vector ⇒
                (ptr[n]←ed1 ptr[n])
                ☞command ← 126)
        (done with ptr)
        (☞back←true. done with ptr)
              ) [command] eval.
        )
    )
    'The heart of ED1 is a vector, containing as its elements
    code vectors. The giant vector is indexed to get the particular
    piece of program, and it is sent the message EVAL. Note that
    the order of the segments in ED1 should match the order of the
    atom names in MENUVEC.'


to edpush ins (☞ins←vector 2.
    ins[1]← ptr[n to n+nrun]. ins[1][nrun+1]←nil.
    ☞ptr←vecmod ptr n nrun ins)

to gettwo t1 n2 (☞n←edtarget. ☞n2←edtarget.
    ☞nrun ← 1+n2-n.
    nrun<1⇒(☞n←n2. ☞nrun←2-nrun))

to bugin someframe max index(
    :someframe.
    ☞max ← 1+:.
    repeat ( repeat (button 0 ⇒ (repeat (
                button 7 ⇒(disp sub ☞(ev))
                button 4 ⇒(done)
                button 1 ⇒ (done))
            done)
        )
        0<(☞index←someframe mfindt mouse 8 mouse 9) < max ⇒
            (∩index)
        )
    )

to edtarget targ( ☞targ←bugin disp 1.
    disp ← 056. ∩(targ-1)

done
printing 1.
```

printing 6.

## 'TEXT DISPLAY ROUTINES'

'Display frames are declared with five parameters. They are a left x, a width, a top y, a height, and a string. Hence -- "yourframe←dispframe 16 256 16 256 string 400. -- gets you an area on the upper left portion of the display that starts at x,y 16,16 and is 256 bits(raster units) wide and 256 bits high. The string (buf) serves as the text buffer, and is altered by ← and scrolling.

[N.B. X-s and width-s are driven to a mulitiple of 16 value in the machine code. X-s resolve to the floor (e.g. if x is 15 going in, it will be 0 on the display. Width-s resolve to the ceiling (e.g. if a width goes in as 249, it will come out on the display as 256. )This restriction may be relieved in the future.]

There are actually two entities associated with display frames--frames and windows. Currently both are given the same dimensions upon declaration (see isnew).

The four instance variables defining the window are "winx", "winwd", "winy", and "winht". The boundaries of this rectangle are intersected with the physical display. The window actually used by the machine language will reduce the size of the window, if necessary, to be confined by the physical display. Clipping and scrolling are done on the basis of window boundaries. If a character is in the window it will be displayed. If a string or character cause overflow of the bottom of the window, scrolling will occur.

The four instance variables defining the frame are "frmx", "frmwd", "frmy", and "frmht". This rectangle may be smaller or larger than its associated window as well as the physical display. Frame boundaries are the basis for word-wraparound. (Presently, if frmy+ frmht will cause overflow of the window bottom[winx+winht], frmht will get changed to a height consonant with the bottom of the window. This has been done to manage scrolling, but may get changed as we get a better handle on the meaning of frames and windows.).

"Buf" is the string buffer associated with any given instance of dispframe. This is the string that is picked on the way to microcode scan conversion. When scrolling occurs, the first line of characters, according to frame boundaries, is stripped out and the remainder of the buffer mapped back into itself. If a "←" message would overflow this buffer, then scrolling will occur until the input fits.

"Last" is a "buf" subscript, pointing to the current last character in the buffer. That is, the last character resulting from a "←".

"Lstln" also points into the buffer at the character that begins the last line of text in the frame. It is a starting point for scan conversion in the "←" call.

"Mark" is set by dread (see below) and points to the character in the buffer which represents the last

```
    prompt output by SMALLTALK; reading begins there.
    Mark is updated by scrolling, so that it tracks
    the characters.  One could detect scrolling by
    watching mark.

    "Charx" and "chary"  reflect right x and top y of
    the character pointed to by "last".

    The "reply" variable in the instance may be helpful in controlling
    things.  When the reply is 0, it means everything should be OK.
    That is, there was intersection between the window and display and
    intersection between the window and the frame.  When reply is 1,
    there was no intersection between the window and the display.
    A 2 reply means no intersection between window and frame.
    A 11 means that the frame height has been increased in order to
    accomodate the input.  A 12 means the bottom of the window
    (i.e. window x + window height) has been overflowed --hence that
    scrolling took place.  A 13 means that both 11 and 12 are true.'


(to dispframe input
    : winx winwd winy winht frmx frmwd frmy frmht buf
    last mark lstln charx chary reply editor
    : sub frame dread reread (
◁ ← ▷(6 CODE 51)

    'is. s is number ? (append this ascii char)
        s is string ?(append string)
        error.'

◁param▷(⍎(!☞.)oval)

    'Allows access to instance variables. For example,
        yourframe param ("winx+32)
    will alter the value of window x in the
    instance of dispframe called "yourframe".'

◁show▷(4 CODE 51 3 CODE 51)

    'Show clears the intersection of window and frame (see fclear ,
    below) and displays buf from the beginning through last.
    A handy way to clean up a cluttered world.'

◁fclear▷(4 CODE 51)

    'Fclear clears the intersection of the window and frame.
    Hence if the frame is defined as smaller than the window,
    only the frame area will be cleared.  If the frame is defined
    as larger than the window, only the window area will be cleared,
    since that space is in fact your "window" on that frame.'

◁wclear▷(5 CODE 51)

    'Wclear clears the intersection of a  window and the physical
    display.'

◁scroll▷(2 CODE 51)

    'Scroll removes the top line of text from the frame~s
    string buffer, and moves the text up one line.'

◁clear▷(1 CODE 51)

    'Clear does an fclear and sets the "last" pointer into the
    string buffer to 0 and "lstln" to 1.  It has the effect of
    cleaning out the string buffer as well as clearing the
```

```
        frame area.'

⊲mfindc ⊃(7 CODE 51)

        'Returns the number of the character
        corresponding to the x and y passed.  Sample call --
              "a←yourframe mfindc mouse 8 mouse 9.
        A -1 return means x,y after end of string.
        A -2 return means x,y not in frame.'

⊲mfindt ⊃(6 CODE 51)

        'Like mfindc except returns a token number. Tokens
        in this context are taken to be separated
        by a blank or carriage return.  Hopefully, it will soon
        be smarter about multiple spaces and/or cr-s. A sample call--
              "variable←yourframe mfindt mouse 8 mouse 9.
        -1 and -2 returns same as for mfindc.'

⊲read⊃(⍐dread)

        'Makes a code vector out of keyboard input.
        See dread below.'

⊲reread⊃(⍐reread 1)

        'Used by redo and fix.  Goes back n(its argument),
        prompts and does a read from there.
        See reread below.  '

⊲sub⊃(☞input ← sub 1. SELF show. ⍐input)

        'Evals its argument in a sub-window.  Used by fix and
        shift-esc.  See sub below.'

⊲knows⊃(ev)

        'Whilst at the KEYBOARD, one can say "yourframe knows(DOIT)"
        and get a copy of the evaluator in the context of that instance
        of dispframe.  Allows access to instance variables without going
        through the param path.  '

⊲frame ⊃ (apply frame)

        'Draws a border of the given color
        around the frame.  E.g.,
              yourframe frame -1.'

lsnew ⊃ (☞winx←1frmx. ☞winwd←1frmwd. ☞chary←☞winy←1frmy.
        ☞winht←1frmht. 1buf. ☞lstln←1.
        ☞mark←☞last←☞charx←☞reply←0. frame -1)  ))

dispframe knows
to dread reader t flag (
        disp←20. DRIBBLE flush. ☞flag←false. ☞mark←last.
        repeat (050>disp←☞t←kbd⊃(
              t=010⊃(last<mark⊃(disp←buf[last+1])
              'Backspace only up to prompt.'
                    buf[last+1]=047⊃(☞flag←flag is false))
                    'Backspace out of string flips flag.'
              t=012⊃(flag⊃() done)
              'DOIT checks if in a string.'
              t=047⊃(☞flag←flag is false)
              'Flag is true if in a string'
              t=023⊃(sub ☞(ev). ☞last←last-1. disp show)
              'Shift-Esc make sub-eval.'
```

```
        ))
    disp←13. ☞reader ← sequence buf mark. ⍎read)
to sub disp (
    ☞disp←dispframe winx+48 winwd-64 winy+14 winht-28 string 300.
    disp clear. (:)eval)

    'Opens a sub-frame, and evals its argument
    in that context.'

to frame a t (☞a←turtle. 'save turt' ink t. penup goto winx-1 winy-1
    pendn goto ☞t←winx+winwd winy-1 goto t ☞t←winy+winht
    goto winx-1 t goto winx-1 winy-1. a 1 'restore turt')

    'Draws a line around the frame, one unit outside.'

to reread n i p reader (
    ☞p←mark. for i to :n
        (☞p←buf[1 to p-1] find last 20.
        p<1⇒(done))
    i<n+1⇒(error ☞(no code))
    ☞reader ← sequence buf p. ⍎read)

    'Counts back n prompts (n is integer arg)
    and then does a read from there.'

done

to dclear (CODE 52)

    'This function takes four parameters -- x width y height,
    and clears the display rectangle thus defined. X will be
    taken to closest multiple of 16 [floor] unless already
    a multiple of 16. Also true for width, except
    taken to ceiling.'

to dmove (CODE 53)

    'This function takes six parameters -- source x width
    source y height destination x destination y. It takes the
    source rectangle (x and width mod 16-d as in dclear) and
    moves it to the destination x and y. Clipping will occur on
    display boundaries. The source will remain intact unless
    it overlaps with the destination, in which case the over-
    lapping portion of the destination wins.'

to dmovec (CODE 54)

    'Dmovec takes the same parameters as dmove, but
    in addition clears the non-intersecting source material.
    It is the general case of what happens on the display
    screen during a scroll, i.e. scrolling could be accomplished
    by saying disp param (dmovec winx winwd winy+fontheight
    winht-fontheight winx winy). A sample call --
    dmovec 0 256 0 256 256 256. This will move
    whatever is in the upper left hand corner of the
    display to x,y 256,256 -- and then erase the
    source area.'


to redo (disp param (☞last←mark-2). (disp reread :) eval. disp show.)

    'Causes re-evaluation of the input typed n prompts
    before this. Setting last←mark-2 makes the redo
    statement and its prompt disappear with a disp show.'

to fix vec (disp param (☞last←mark-2). ☞vec←disp reread :.
```

```
(disp sub (veced vec)) eval)

'Like redo, except that the previous input is given
to the editor in a subwindow.  When editing is done,
the resulting code is evalled before returning.'

'Nova file maintenance.   DP0:DISPLAY.AN  DP1:DISPLAY.AN '
printing 1
```

printing 0

# 'THE TRUTH ABOUT FILES

*FILESMALL -- SMALLTALK file system 7/74*

*a file is found in a directory ("dirinst") by its file name ("fname"), and has a one "page", 512 character string ("sadr").*

*"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 (SYS.STAT.), or any directories you create.*

*<directory> file <string> load loads a previously "saved" SMALLTALK virtual memory, thereby destroying your current state.*

*<directory> file <string> save saves SMALLTALK virtual memory.*

*"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" is (should be) 1 if any label block integers ("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 eof occurred with the last "set"; a positive number (machine language pointer to offending disk command block (dcb)) signals a disk error.*

*the next 8 integers 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 integers, 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. bits in "sn1" flag directories and random files.*

*the class function "ncheck" checks that file names are strings, contain no "illegal" characters, and terminate with a period .'*

```
(to file : dirinst fname sadr leader curadr bytec dirty status nextp
      backp lnused numch pagen version sn1 sn2 : ncheck illegal (

    ◁←▷ (17 CODE 50)

        'fi←<integer>, <string>, or <file> --
        :x is string? (for i to x length (SELF←x[i]))
        x is file? (repeat (x eof? (done) SELF←x next))
        (numch<"bytec←bytec+1?
          (SELF set to write (pagen←bytec/512) bytec mod 512))
```

```
    sadr[bytec]+x &* 0377'

◁next▷    ((◁into▷ (16)

        'fi next into <string> -- read a string
        for i to :x length(x[1]+SELF next).!x'

      ◁word▷ (◁+▷ (7)

        'fi next word+<integer> -- write integer,
        possibly increment pointer to word boundary.
        (0=bytec &* 1? ( ) "bytec+bytec+1)
        SELF + :x/256. SELF + x mod 256.'

         6)

        'fi next word -- read an integer
        (0=bytec &* 1? ( ) "bytec+bytec+1)
        !(SELF next*256) + SELF next'

      ◁char. 0) CODE 50)

        'fi next or fi next char -- read a character
        (numch<"bytec+bytec+1?
         (SELF set to read (pagen+bytec/512)
          bytec mod 512? ( ) !0)) !sadr[bytec]'

◁eof▷     (10 CODE 50)

    'fi eof -- return false if end of file has not
    occurred. nextp=0? (bytec<numch? (!false))!false'

◁flush▷   (12 CODE 50)

    'fi flush -- dirty=0? ( ) write current page'

◁set▷     (◁to. (◁eof▷(13)

        'fi set to eof -- set file pointer to end
        of file. SELF set to read 037777 0'

      ◁write▷(5)

        'set to write <integer> <integer> -- set
        file pointer to :spage :schar. if current page
        is dirty, or "rewind", "set to eof" or page change
        occurs, flush current page. read pages until
        pagen=spage. allocate new pages after eof if
        necessary (-1 512 is treated as start of next
        page, i.e. pagen+1 0). "bytec+schar'

    ◁read. 4) CODE 50)

        'same as "write" except stop at eof'

◁skipnext▷ (18 CODE 50)

    'fi skipnext <integer> -- set character pointer
    relative to current position. (useful for skipping
    rather than reading, or for reading and backing up,
    but "eof" may not work if "bytec" points off the current
    page) "bytec + bytec + :offset'

◁is▷ (ISIT eval)

◁pages▷   (20 CODE 50)
```

'fi pages <integer> ... <integer> -- 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, copy "sadr" to a buffer before the i/o call, and copy it back to "sadr" after the call. diskaddress (=-1 yields "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". perform i/o call. return false if error occurs'

⟨readseq⇒ (21 CODE 50)

'transfer words from a file to memory :adr. :count. for i←adr to adr+count-1 ( mem I ← SELF next word)'

⟨writeseq⇒ (22 CODE 50)

'...from memory to a file...(SELF next word←mem I)'

⟨print⇒ (disp ← fname) 'file prints its name'

⟨rewind⇒ (11 CODE 50)

'fi rewind -- reposition to beginning of file SELF set 1 0'

⟨clear⇒ ()

'disp clear -- imitates dispframes for filout'

⟨evals⇒ (⋔(:@) eval)

⟨knows⇒ (ev)

⟨remove⇒ (dirinst evals(filesopen)delete SELF)

'remove file from filesopen list of directory'

⟨close⇒ (dirinst evals(bitinst flush). SELF remove. SELF flush. ⋔@closed)

'fi close or "fi←fi close (necessary to release instance) -- flush bittable and current page.'

⟨shorten⇒ (⟨to. ⟨here⇒ (SELF shorten pagen bytec) 14 CODE 50)

'fi shorten to <integer> <integer> -- shorten a file SELF set to read :spage :schar. "x←nextp. "nextp←0. "numch←schar. "dirty←1. deallocate x and successors'

isnew⇒ ((@←fname←ncheck (:)⇒ () error @(bad file name))
((@←dirinst←curdir) is directory⇒ ()
(@←dirinst←directory evals(defdir))is directory⇒
(dirinst open) error@(illegal directory))

'set directory instance for file. if curdir is not a directory (null global value because file was not called from the context of a directory instance), use the default directory'

```
        ◁delete⇒ (15 CODE 50. ⇑☞deleted)

              'delete a file (see intro)'

        ☞sadr← string 512.
        (◁new⇒(dirinst evals(filinst)is file⇒(3)19)
         ◁old⇒(2)1) CODE 50.

              'allocate string buffer. find an old file or
              add a new entry (see intro). directories have
              a special "sn1". machine code may return false'

        ◁load⇒     (8 CODE 50)
        ◁save⇒     (9 CODE 50.
              directory evals(☞'curdir←0.☞defdir←
                   ☞dp0←directory dirname))

              'load returns via "save". virtual memory on
              file should have no active files or directories;
              dp0 is reinitialized upon load. how to reopen
              other files (e.g. DRIBBLE)?'

        dirinst evals(filesopen)←SELF

              'file puts itself into the filesopen list
              of its directory' ) ))
file knows
to nchock i str (
     (:str is string⇒ (0<str length<255⇒ () ⇑false) ⇑false)
     for i to str length (str[i]<33⇒ (⇑false)
      0<illegal[1 to 99] find str[i]⇒ (⇑false))
     str[str length]=46⇒ (⇑str) ⇑str+☞'. chars)

'check that file name is a string and of proper length and contains
no illegal characters. if name does not end with a period, append one'

(☞illegal←string 11) fill
[]().,:;>*#☞
done


'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"). the top level, "distinguished node"
of the directory structure is the system directory "dp0" (see "directory
knows" below if you also want "dp1"). dp0 knows the disk number ("dirinst")
and the true identity of the bittable. 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.

"dl ← <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 a directory, i.e. file was not called
in the context of a directory, but globally'

(to directory ch : dirinst bitinst filinst filesopen : dirname bitname
     curdir defdir (
     ◁file⇒     (SELF open. ⇑apply file)
```

```
        'di file <string>... -- open directory. create file
        instance (see file intro)'

    ◁directory▷ (SELF open. ⇑apply directory)

        'di directory <string>... -- open directory. create
        directory instance'

    ◁open▷    (☞curdir←SELF. filinst is file▷ ()
        (bitinst)0▷ (☞bitinst←dirinst evals(bitinst).
        ☞filinst←file filinst new) ☞filinst←file filinst old.
        ☞bitinst←(dirinst is directory▷ (dirinst evals(bitinst))
         file bitname old))
        filinst evals(sn1)>-1▷ (error☞(bad directory sn))
        dirinst is directory▷ (dirinst evals (filesopen)←SELF))

            'di open -- (normally not user-called since access
            to the directory always reopens it) initialize
            directory file and bittable instances. directory
            (except for "top level") puts itself into filesopen
            list of its directory'

    ◁is▷ (ISIT eval)

    ◁print▷    (disp←0133. filesopen print. disp←0135)

            'di or di print. --print the filesopen list'

    ◁list▷    (SELF open. (◁after▷ (filinst snt (:) (:)) filinst rewind).
        repeat (filinst eof▷ (cr. done)  0=1024 ^*
        ☞ch←filinst next word▷ (ch<2▷ () filinst skipnext 2*ch-1)
        filinst skipnext 10.
        disp ← filinst next into string filinst next. sp) )

            'di list -- print the entry names contained in filinst'

    ◁flush▷ (filinst is file▷ (filesopen map ☞(vec[i] flush).
        ☞filesopen←obset 10. ☞filinst←filinst evals(fname).
        bitinst flush. ☞bitinst←-1))

            'di flush -- (normally not user-called). if
            directory is open, flush it by flushing all files
            and directories in its filesopen list. store
            file name in "filinst", and "old" in bitinst so
            that it can be reopened'

    ◁close▷    ((dirinst is directory▷ (dirinst evals(filesopen)
            delete SELF))  SELF flush. ⇑☞closed)

            'di close (e.g. dp0 close) or "di←di close
            (to release instance) --close a directory by
            deleting it from the filesopen list of its
            directory (except for the system directory)
            and flushing it. this is currently the easiest
            way to regain space by closing unwanted file
            instances -- beware of multiple instances of
            the same file. this is also the officially
            recognized way to change disk packs without
            rebooting (replaces "drelease")'

    ◁use▷    (☞dofdir←SELF) 'di use -- change the default directory'

    ◁evals▷    (⇑(:☞) eval)

    ◁knows▷  (ev)
```

```
    lsnow⇒    (↑filesopen←obset 10. ↑dirinst←curdir.
        dirname=:filinst⇒ (↑bitinst←-1. ↑curdir←SELF)
        ↑bitinst←(⇦new⇨(1) ⇦old. -1). SELF open)

            'default directory will not work here unless
            "curdir" is "unset" upon exit. only thing
            special is that the system directory is not
            opened the first time through' ))
directory knows
(↑dirname←string 7) fill
SYSDIR.
(↑bitname←string 9) fill
SYS.STAT.

    'names for the system directory and bittable'

↑curdir←0. ↑defdir←↑dp0←directory dirname.

    'create the system directory instance (the initial default)
    on disk 0 in a "closed" state. to initialize a second disk:
    directory knows
    "curdir←1. "dp1←directory dirname.
    done'
done
↑curdir←nil.
        'to prevent "curdir has no value, I was in
        file..." when default is desired'

to dir (dp0 list after 5 2)

    '...for a partial directory listing, starting after
    a page and byte specification. numbers may change between
    versions or if directory is "compressed"'

'some "fillin-able" disk utilities

dskutils.
    dskstat    prints the number of free/used pages on your disk
    e.g. dskstat

    rename renames an existing file. this function is not too
    speedy or smart -- the new file name should not already be in
    use, and neither the old or new file names are checked for
    correctness or ending dots.
    e.g. rename ⟨string = old file name⟩ ⟨string = new file name⟩
    (if there is sufficient demand, rename will be done right)

    spool copies files from default directory into file PT. interim
    measure for DPRINTing files stored in subdirectories, or for putting
    more files (separated by their file names) onto fewer sheets of paper.
    e.g. spool ⟨vector of ⟨string = file name⟩s⟩

    purge deletes files from default directory
    e.g. purge ⟨vector of ⟨string = file name⟩s⟩

    undribble closes DRIBBLE file and redefines it as a function.
    e.g. undribble

type.types a file, 512 characters at a time (=1 disk page),
    into "disp". ⟨space⟩ increments the page number; ⟨return⟩ exits;
    any other character decrements the page number.
    e.g. type ⟨file⟩ or ⟨string = file name⟩


install.    "installs" boot file at a designated disk location by
```

```
    copying page 1 of the boot there if the page is free. if
    the page belongs to another file, the serial number
    of that file is printed, and you must type <doit> to
    overwrite; anything else to abort.
    e.g. install <file> or <string = file name> at
         <string = "uppercase bootkeys"> or <integer = disk address>

    boot a file (via software). alleviates holding down keys.
    an oft-used boot file should be installed, see below.
    e.g. boot <file> or <string = file name>
       booter <integer = disk address of page 1 of boot file>

doit.fix.  puts <doit>s at ends of lines when ( ) count is 0. for
    bringing text files from Novas or elsewhere.
    e.g. doit <file> or <string = file name>


xplot.     dumps a screen bitmap onto a file (86 pages) for printing
    on an XGP with XPLOT, e.g. xplot <file> or <string = file name>'

' to [ i x s (:x. "s+string 6. for i to 6(
    s[7-i]+48+x &* 7. "x + x &/ -3). !s)
to bl i s n (:s. :n. for i=s to s+n-1 (([mem i)print.sp.))
    temporary debugging aids'
```

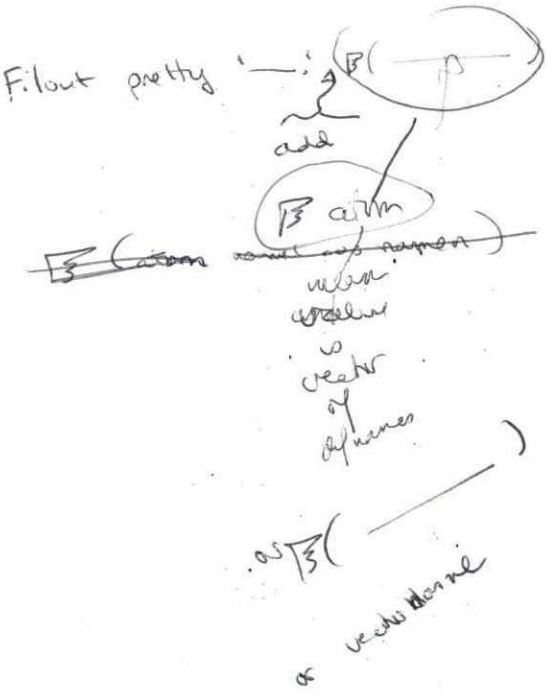
printing 1

printing 0.

## 'BOOTSTRAPPING REVISITED'

```
to classprint fn a b i j k flags clsv clsm arecv arecm instv instm code (
      :#fn. @code ← GET fn @DO. null code⇒(@(no code))
      @a←leach #fn. @b←vector 1. @b←leach b. @clsm←@arecm←@instm←0.
      @k←a[1]^. @clsv←vector k. @arecv←vector k. @instv←vector k.

      'Pull symbols out of class table'
      for i←4 to 4+2*k by 2   'k is no. dbl entries -1, here'
            (@k←a[i]^.
            k=-1⇒(again). @flags ← k^/-14. '0=class, 2=arec, 3=inst'
            flags=0⇒(0-@(DO TITLE SIZE) [1 to 3] find a[i]⇒
                  (clsv[@clsm←clsm+1] ← a[i]))
            b[2]^ ← k^*03777. @j←a[i+1]^.
            (flags=2⇒(arecv[j-6] ← b[2]. arecm<j-6⇒(@arecm←j-6))
                  instv[j+1] ← b[2]. instm<j+1⇒(@instm←j+1))
            )

      'Now make up input form.'
      @a ← vector 6+arecm+instm+clsm.
      a[1] ← @to. a[2] ← GET fn @TITLE.
      a[3 to @j←2+arecm] ← arecv.
      (0<instm+clsm⇒ (a[@j←j+1]←@:. a[j+1 to @j←j+instm] ← instv.
            0<clsm⇒ (a[@j←j+1]←@:. a[j+1 to @j←j+clsm] ← clsv)))
      ◇header⇒(a[j+1]←code. ∩a)
      for i to j (a[i] print. disp←32)
      showpretty⇒(pshow code 3) code print)

to show showpretty (@showpretty←true. showev (:@))

to showev shAtom shVal (:shAtom. cr.
      (shAtom is atom⇒
            (@shVal ← shAtom eval.
            (null GET shVal @DO⇒
                  (@@ print. shAtom print. @← print.
                  (shVal is vector⇒ (@@ print))
                  shVal print. @. print)
            classprint shVal))
      shAtom print)
      disp←10.)

'Keyboard fix to translate and dribble'

to kbd : : translation (◇knows⇒(ev))
kbd knows
      @translation ← string 0377.
      for i←001 to 0177(translation[i]←translation[0200+i] ← i)
      translation[0200]←translation[0233]← 040. 'ctl null and esc'
      to t i (translation[:i]←translation[0200+i]←t)
      t 0020 0010     'SHIFT BS'
      t 0021 0011     'SHIFT TAB'
      t 0037 0040     'SHIFT UP'
      t 0036 0040     'UP'
      t 0035 0040     'SHIFT DOWN'
      t 0034 0040     'DOWN'
      t 0030 0040     'SHIFT SPACE'
      t 0025 0015     'SHIFT RETURN'
      t 0027 0010     'SHIFT DEL->BS'
      t 0177 0010     'DEL->BS'
      t 0032 0040     'SHIFT INS'
      t 0031 0040     'INS'
      t 0022 0012     'SHIFT LF'
done
```

```
PUT kbd ☞DO ☞(DRIBBLE ← translation[TTY])
to DRIBBLE (◁← ⇒(:)◁flush⇒()◁set⇒(◁to. ◁eof))

to filout disp flist i showpretty (☞showpretty ← ◁pretty.
      dsoff.(:disp is file⇒ () ☞disp←file disp⇒ () error ☞(file error))
      (◁add⇒(disp set to eof))
      (null :flist⇒(defs map ☞(showev vec[i]. cr))
      (flist is atom⇒ (showev flist. ☞flist←flist eval))
      for i to flist length-1 (showev flist[i]. cr))
      disp shorten to here. disp close. dson.)

      'Filout basically does a show in an environment
      where the display is replaced by a file.
      filout pretty <file> or <string = file name> add <vector>
      if "pretty" is used, the text representation is neater but takes
      longer to generate. if "add" is used, function definitions are
      appended to the file. if <vector> is not specified, "defs" is used.'

to filin f reader : : read fseq bridgit ckend (◁evals⇒((:☞)eval)
      (dsoff. :f is file⇒ () ☞f←file f old⇒ () error ☞(file does not exist))
      ☞reader←fseq f evals (sadr).
      ev. f close .)
☞t←#read. filin evals (☞read←#t. ev) 'ie filin knows...'
to fseq t : str ptr stop bridge str2 eof end (
      ◁scan⇒(ptr>stop⇒((bridge⇒(☞ptr←ptr-stop. ☞stop←end-75) bridgit)
                  ☞bridge←bridge is false. ☞t←str. ☞str←str2. ☞str2←t.
                  ⇑SELF scan)
            CODE 41⇒(1=☞eof←eof+1⇒(⇑☞done) eof=2⇒(⇑rPar) error ☞(file end)))
      ◁skip⇒(☞ptr←ptr+1)
      isnew⇒(:str. ☞bridge←false. ☞end←512. ☞str2←string 150.
            ☞stop←end-75. ☞ptr←☞eof←0. ckend))
to bridgit (str2[1 to end]←str[ptr+1 to end].
      ☞stop←end-ptr. f evals (nextp←0⇒(sadr[1]←0) f set to -1 512. ckend)
      str2[stop+1 to end]←str[1 to end]. ☞ptr←0)
to ckend (f evals  'puts a null at eof'
      (numch<512⇒(sadr[numch+1]←0)) )
to read t (dsoff. ☞t←reed. reader skip. dson. ⇑t) done

      'Filin basically does a read-eval-print loop, but
      its context causes reed to use a file buffer
      ("reader←fseq . . .) instead of the usual keyboard
      buffer in a dispframe.'

to t fool fname ::x y z (◁knows⇒(ev)
      ☞stop←0. ☞disp←dispframe 16 480 511 168 string 520.
      disp←x. to read (⇑disp read). ☞defs ← obset 100.
      ☞fname ← read[1]. (null fname⇒() ☞DRIBBLE←file fname chars+y.
      DRIBBLE set to eof. do 20 (DRIBBLE ← 052))
      disp←z. ☞fool←#to. to to toAtm (CODE 19 defs←toAtm. toAtm)
      PUT USER ☞DO ☞(cr read eval print.). ☞t←0.)
t knows
☞x←(string 36) fill
[ 7/15] Hi, please type your name...
☞y←(string 6) fill
.drib.
☞z←(string 20) fill
Welcome to SMALLTALK
done

      't is called to set up a display frame, and defs -- the list
      of newly defined functions. it also sets up a dribble file (if
      a name is given), and then self-destructs to save space.

Nova file maintenance -- DP0:READER.AN  DP1:READER.AN '
to type f t (☞f←file:t old⇒(f remove. ☞t←string 30.
      repeat(f eof⇒(done) disp←f next string into t)))
```



```
printing 1. ☞printing+6.
PUT USER ☞DO ☞(t). stop
```

```
ALTO            CY     000016   .FIND 000162   TTPUT 001613   .FORX 004541
PAGE0           RP     000017   .CACT 000163   ERR   001614   FORK  004606
READ            DWIDE  000040   .LALO 000164   MKTOK 001665   PICK  004612
EVAL            .LOAD  000070   .EVAL 000165   QSLAS 001674   STIK  004622
SSCAN           .STOR  000071   .FECH 000166   QUEST 001774   GBYT  004626
CODE2           TLMEM  000072   .PEEK 000167   QSIZE 002104   PBYT  004635
UTIL            BHMEM  000073   .LARG 000170   QLEN  002157   QERR  004651
FUNCS           SELF   000074   .ERET 000171   QAROW 002174   INDX  004675
COMM            VALUE  000075   .INDX 000172   QAND  002203   DISP  004715
CODE            DX     000076   .DISP 000173   QAREC 002257   LINST 004731
MEM             RADIX  000077   .SARG 000174   QMOD  002264   LNST  004734
RCHAR           ATTN   000100   .SGET 000175   QLESS 002360   SINST 004745
TURT            .COMP  000101   .SPUT 000176   U TRAC1 002    INTN  004771
OPEN            MSMSK  000102   .MUL  000177   EVAL  002404   MXNUM 005013
GTINS           ERLOC  000105   .DIV  000200   ARET  002421   ISIT  005031
MOD16           TOKEN  000106   .LAL4 000201   QRETN 002455   USRX  005067
CKPAR           EMPTY  000107   .REFI 000202   QOR   002457   GET   005102
SETUP           SCLAS  000110   .REFD 000203   ERET  002461   GADDR 005200
MAZUR           ATCLS  000111   .SVAL 000204   EFIND 002477   PUT   005201
LNOUT           LSCLA  000112   .SVLI 000205   MESSX 002620   FIND  005232
ALFNT           NCLAS  000113   .SSLF 000206   GLOBX 002621   SGET  005302
PSTRG           MASTE  000114   .LINS 000207   QRPAR 002644   SPUT  005344
BCLMV           .QUES  000115   .SINS 000210   QCODE 002662   CACT  005376
CLREM           .QSIZ  000116   .ISIT 000211   FETCH 002753   SRETN 005451
SETXY           .QLAR  000117   .FALS 000212   EVTK  002756   SVAL  005523
SCAN            .QPER  000120   .SSCA 000213   BOUND 003153   SVLI  005536
UPDAT           .QDO   000121   .SMF  000214   GNST  003205   IVAL  005547
NEWLN           HOLD0  000122   .GNB  000215   GINS  003207   U FTR   005
INSEC           HOLD1  000123   .PNB  000216   GARG  003217   U FLD   005
CKSTF           HOLD2  000124   .GARG 000217   QMPER 003230   REFD  005613
DOMOD           HOLD3  000125   QLPAR 000240   QPER  003270   REFI  005643
CLRWF           HOLD4  000126   QCHAR 000270   SSCOD 003275   FPCLA 005710
DSPCH           HOLD5  000127   QGLOB 000350   SSCAN 003341   LALO  006155
SCROL           C2     000130   QTITL 000513   QGRTR 003370   LAL4  006224
BKSPC           C3     000131   QDO   000540   QEROR 003407   LOCS  006315
SMFCO           C4     000132   QMACH 000624   SMF   003545   CODES 006322
SMFIL           C5     000133   QHYPH 000664   SMS   003574   U PRIN1 006
SMALL           RBMSK  000134   QEQ   000764   QNOEV 003614   U READ1 006
SMLOO           LBMSK  000135   QEVAL 000774   PPW   003616   U USERL 006
SMDIO           RCMSK  000136   DWIDB 001000   PNW   003620   U STOP  006
DSKIO           MXATM  000137   START 001001   PTW   003621   U FILIN 006
LAST            .IVAL  000140   PSTAD 001001   GPW   003642   U WVMEM 006
NMAX 060345     .SRET  000141   QSELF 001011   GNW   003644   U CLK   006
ZMAX 000220     .PUT   000142   QTO   001100   GTW   003645   U TRACE 006
CSZE            .GET   000143   XYBIT 001163   QPLUS 003654   U PRING 006
EST             .NEXT  000144   QRBRA 001164   GPB   003657   U FPCOD 006
SST             .GBYT  000145   PRTON 001205   GNB   003661   U SPLOT 006
                .PBYT  000146   KBCK  001210   GTB   003662   RPT1  006417
DBITS           .PICK  000147   QUOTE 001210   PPB   003702   AGAIN 006425
DTABE           .STIK  000150   KBINT 001232   PNB   003704   DONE1 006427
FX      000005  .ERR   000151   TTGET 001242   PTB   003705   NUM1  006473
FW      000006  .QERR  000152   DWITE 001250   MSAVE 004000   RFALS 006712
FY      000007  .TPUT  000153   QSTAR 001250   FREE  004004   QUOT1 006753
FH      000010  QLBRA  000154   QCLAS 001261   TOPLE 004010   ATOM1 006756
BF      000011  .TGET  000154   QCOLN 001350   SINTB 004012   PUT1  007035
LS      000012  .ARET  000155   MOUSE 001354   STRVE 004057   EMPT1 007051
MK      000013  .INTN  000156   QMESS 001432   KYBD  004352   MEM1  007127
LL      000014  .LMPC  000157   SIO1  001535   APLY1 004374   GET1  007156
CX      000015  .AMPC  000160   DOMUL 001564   LEECH 004420   FET1  007173
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| MAT1 | 007257 | WINX | 012220 | CLEAR | 016265 | IRCOP | 021003 |
| TO1 | 007276 | WNWD | 012221 | STCLR | 016355 | RICOP | 021005 |
| EQ1 | 007471 | WINY | 012222 | CLRX | 016357 | RFPTR | 021063 |
| NULL | 007503 | WNHT | 012223 | CLRWD | 016360 | SNST | 021104 |
| APRET | 007507 | FRTBL | 012224 | CLRY | 016361 | GFCH | 021136 |
| ISNEW | 007522 | FRMX | 012224 | CLRHT | 016362 | GFWD | 021175 |
| MKINS | 007553 | FRWD | 012225 | INDEX | 016375 | GFSTR | 021206 |
| LCL | 007603 | FRMY | 012226 | STBCL | 016425 | PFCH | 021233 |
| LPC | 007606 | FRHT | 012227 | STBLT | 016426 | PFWD | 021317 |
| LMO | 007611 | BUF | 012230 | BMVCL | 016452 | PFSTR | 021371 |
| LME | 007614 | BFLN | 012231 | BLKMV | 016453 | DSKAD | 021527 |
| LGL | 007617 | LAST | 012232 | CLREM | 016644 | DSKSE | 021556 |
| LRE | 007622 | MARK | 012233 | STSXY | 016767 | ALLOC | 021575 |
| LIN | 007625 | LSLN | 012234 | SETX | 016771 | SHORT | 021635 |
| LAO | 007630 | CHRX | 012235 | SETY | 016773 | DALLO | 021656 |
| SCL | 007633 | CHRY | 012236 | SETXY | 017002 | GBDSK | 021743 |
| SPC | 007636 | RPLY | 012237 | CHR | 017026 | LODM | 021770 |
| SMO | 007641 | WNBT | 012240 | CONTB | 017033 | SAVM | 021772 |
| SME | 007644 | FRBT | 012241 | CTB0 | 017033 | PAGIO | 022037 |
| SGL | 007647 | FRRT | 012242 | CTB1 | 017034 | EOF | 022155 |
| SRE | 007652 | GTINS | 012244 | CTB2 | 017035 | FCLAS | 022201 |
| STIN | 007655 | MOD16 | 012333 | SCAN | 017036 | DELET | 022213 |
| SAO | 007660 | RELTB | 012352 | HITSW | 017073 | LOOKU | 022242 |
| LARG | 007663 | REALX | 012352 | WNOVR | 017074 | ELOOK | 022443 |
| SARG | 007667 | RELWD | 012353 | UPDAT | 017101 | ENTER | 022453 |
| SSELF | 007703 | REALY | 012354 | NEWLN | 017166 | CRR | 022631 |
| AMPC | 007711 | RELHT | 012355 | INSEC | 017231 | CRW | 022632 |
| APC | 007716 | CKPAR | 012357 | LSTSP | 017300 | CCR | 022633 |
| LMPC | 007732 | MZTBL | 012440 | LSTC | 017301 | CCW | 022634 |
| NEXT | 007740 | MLBD | 012441 | QFLG | 017322 | CWW | 022635 |
| PEEK | 007752 | MRBD | 012442 | QPT1 | 017323 | SREW | 022636 |
| LOADE | 007777 | LNTBL | 012444 | QPT2 | 017324 | SREOF | 022637 |
| STORE | 010004 | LLBD | 012445 | CRCHK | 017326 | SCLOS | 022643 |
| VM2CR | 010016 | LRBD | 012446 | DOSPC | 017400 | SRBYT | 022653 |
| CR2VM | 010033 | INTBL | 012447 | CKSPC | 017417 | SWBYT | 022661 |
| ST2CO | 010075 | INX | 012447 | MDDIM | 017505 | DSKRW | 023232 |
| GCHAR | 010124 | INWD | 012450 | MDINC | 017511 | WDSKO | 023267 |
| RIGHT | 010760 | INY | 012451 | MDFIX | 017526 | DSKIO | 023374 |
| GO | 011111 | INHT | 012452 | PCLER | 017563 | DSKBU | 023673 |
| DPLOT | 011267 | OPNSW | 012455 | WCLER | 017564 | PATCH | 024273 |
| INK | 011344 | SETUP | 012457 | FCLER | 017576 | MEMSI | 024333 |
| TTLC | 011352 | .MGWD | 012540 | DSPCH | 017617 | XYC1 | 024334 |
| TSTAT | 011370 | .CRCK | 012541 | BFCON | 017625 | XYC0 | 024341 |
| GOTO | 011575 | .DOSP | 012542 | MOOD | 020000 | MEMBI | 024345 |
| BTMAP | 012032 | .CKSP | 012543 | STRMD | 020061 | DSIZE | 052400 |
| DSPTB | 012033 | MAZUR | 012551 | MSEX | 020152 | PHIAD | 060445 |
| DWDTH | 012037 | OUTBL | 012606 | MSEY | 020153 | SINT1 | 177600 |
| OPEN | 012066 | OUTX | 012606 | MCHR | 020154 | | |
| SVALL | 012117 | OUTWD | 012607 | MTOK | 020155 | | |
| SVACS | 012120 | OUTY | 012610 | MCHRS | 020156 | | |
| RSALL | 012123 | OUTHT | 012611 | MTOKS | 020157 | | |
| RSACS | 012124 | .LGWD | 012614 | MSESW | 020160 | | |
| WIDTH | 012127 | .LSCN | 012623 | GOTIT | 020161 | | |
| GETHT | 012171 | CURX | 012640 | GOTMS | 020220 | | |
| GETWD | 012173 | LNOUT | 012642 | STSCL | 020321 | | |
| GTMES | 012203 | FHTGH | 012764 | SCROL | 020327 | | |
| .BF | 012216 | FWDTH | 012765 | BKSPC | 020452 | | |
| INSTB | 012217 | FSTRT | 012766 | FCODE | 020530 | | |
| INSTC | 012217 | MAZX | 016171 | RVCOP | 020777 | | |