```
to number x y ()
to class x y ()
to vector x y ()
to atom x y ()
to string x y ()
to false x y ()
'Dont edit above here!'
to printing (CODE 38). printing 0.
to print (%.?() CODE 0)
to # (:#)
print #number. print #atom. print #vector. print #string.
print #false. print #class.

to : (CODE 18)
to % (CODE 17)
to ! (CODE 13)
to " (CODE 9)

"(TITLE USER DO SIZE CODE SELF or and mod rem
 false print chars ref
 ".,/;:~[]~|↑!#$%}←={}*+?<>)

to PUT x y z (:#x. :y. :z. CODE 12)
PUT atom "DO "(CODE 29
        %is?(%atom?() :". !false)
        %print?(disp←SELF chars) )
PUT false "DO "(CODE 11
        %is?(%false?(!1) :".)
        %print?("false print) )
PUT number "DO "(CODE 4
        %is?(%number?() :". !false)
        %print?(disp←nprint SELF 10)
        %base?(!nprint SELF :x) )
PUT vector "DO "(CODE 3
        %is?(%vector?() :". !false)
        %print?(disp←40. for x to SELF length
                (disp←32. SELF[x] print). disp←41) )
PUT string "DO "(CODE 3
        %is?(%string?() :". !false)
        %print?(disp←39. disp←SELF. disp←39) )
PUT USER "TITLE "USER

to repeat x (:#x. CODE 1)
to done x (%with?(:x. CODE 25) CODE 25)
to again (CODE 6)
to ev (repeat (read eval print cr.))
"eval←"eval
to read (CODE 2)
to nprint r n i s :: pname (%knows?(ev):n. :r. "i←18.
        (n<0?("s←1. "n←0-n)"s←0)
        repeat (pname["i←i-1] ← 48+n-r*"n←n/r. n=0?(done))
        (s=1?(pname["i←i-1]←45)). !sscan pname[i to 17])
nprint knows
"pname←string 17. done

to right (CODE 21)
to go (CODE 22)
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 : pen ink dir x xf y yf dx dxf dy dyf (
        isnew?(0 CODE 39)
        %knows?(!(".)eval)
        1 CODE 39 :z. 0 CODE 39 !z)

to filin x (:x. CODE 16 x)
to STOP (CODE 8)
to GET x y (:#x. :y. CODE 28)
to mem x y (:x. CODE 26)
to - (0-:)
to trace (CODE 34)
to mouse x (:x. CODE 35)
to core ((mem 59)-mem 58)
to ov (CODE 15)
to isnew (CODE 5)
to null x (:x. 1 CODE 37)
to error (print :. CODE 14)
to eq x y (:x. :y. 1 CODE 33)
to kbd (CODE 20)
to disp x i (%←.
        :x is string?(for i to x length (TTY←x[i]))
        TTY←x)
to TTY (CODE 20)
to dsoff (mem 272←0)
to dson (mem 272 ← mem 62)
to apply x y (:#x. %to?(:y. CODE 10) CODE 10)
to cr (disp←13).  to sp (disp←32)
to whoot (CODE 27)
to is (:". !false)
to nil x (#x)

to if exp (:exp?(%then?(:exp. %else?(:". exp)exp)error "(no then!))
        %then?(:".   %else?(:exp) false)error "(no then!))
to for x step stop var start exp (
        :"var. (%=?(:start.)"start←1.)
        (%to?(:stop.)"stop←start.)
        (%by?(:step.)"step←1.)
        %do. :#exp. CODE 24)
to do N exp (:N. :#exp. for N to N do (exp eval.))

to dragon i (:i=0?(go 5)
        i>0?(dragon i-1. right 90. dragon 1-i)
        dragon -i+1. right -90. dragon i+1)

'editor needs cleanup - use sscan to copy, use read for commands,
        and thus eliminate ugly numeric input at cost of needing command accpt.'
to edit func t char :: ed1 edsearch edmatch eddel edins edup edpush (
        %knows?(ev)
        "char←0. :#func. "t←GET func "DO.
        null t ? (error"(no code)) PUT func "DO ed1 t. "done)
edit knows
to ed1 ptr n back(:ptr. "back←false. repeat(
        (char=120?(done with ptr) char=63?() (char=0?()cr). for n to ptr length-1
                (disp←32.ptr[n] is vector?("$ print)ptr[n] print)) "n←0. cr.
        repeat (47<("char←disp←kbd)<58 ? ("n←(char-48)+10*n)
                char=45?("back←1)done)
```

```
        (char=115 ? (edsearch)  )
        (back?("n←ptr length - n."back←false))
        char=108 ? (done with ptr)
        char=120 ? (done with ptr)
        (n<1?("char←63).
        char=101 ? (ptr length>n?(ptr[n] is vector?(ptr[n]←ed1 ptr[n])
                        disp←"char←63).
        char=117 ? (edup)
        char=114 ? (ptr[n]←read[1])
        "n←n-1
        char=112 ? (edpush)
        char=105 ? (edins)
        char=100 ? (eddel)
        disp←"char←63) )
to edsearch target i ("target←read[1]. (n=0?("n←1)).
        back?(for i=ptr length to 1 by -1 (edmatch))
        for i to ptr length (edmatch) )
to edmatch (eq target (ptr[i] is vector?("$)ptr[i]) ?
        (0="n←n-1?(cr. "n←i print. "char←disp←kbd. "back←false. done)) )
to edins t1 t2 ("t1←read. "t2←vector t1 length + ptr length-1.
        do n (t2[N]←ptr[N]).
        do t1 length-1 (t2[n+N]←t1[N]).
        do ptr length-n (t2[n+t1 length+N-1]←ptr[N+n]).
        "ptr←t2.)
to eddel t1 t2 ("t2←vector ptr length - "t1←(disp←kbd)-48.
        do n (t2[N]←ptr[N]).
        do ptr length-t1+n (t2[N+n]←ptr[N+n+t1]).
        "ptr←t2.)
to edpush t1 t2 t3 (
        ((n+("t2←(disp←kbd)-48))>ptr length-1?("t2←ptr length-n+1))
        "t1←vector ptr length+1-t2.  do n (t1[N]←ptr[N]).
        t1[n+1]←"t3←vector t2+1.  do t2 (t3[N]←ptr[n+N]).
        do ptr length-n+t2 (t1[n+1+N]←ptr[n+N+t2]). "ptr←t1)
to edup t1 t2 (ptr[n] is vector ? (
        "t1←vector ptr length+ptr[n] length-2.  do n-1 (t1[N]←ptr[N]).
        "t2←ptr[n].  do t2 length-1 (t1[n+N-1]←t2[N]).
        do ptr length-n (t1[N+n+t2 length-2]←ptr[n+N]). "ptr←t1)
        disp←char←63)
done
to addto func v w (:#func. :w. "v←GET func "DO.
        null v?(error "(no code)). PUT func "DO catcode v w.)
to catcode v w x (:v. :w. "x←sscan v[1 to v length+w length -1].
        !sscan x[v length to x length]←w[1 to w length])

to sscan op byte s lb ub s2 lb2 ub2 (
"lb2 ← "ub2 ← 1.
:#s. %[. :lb. %to. :ub. %].
%find? ("op ← (%first?(1) %last?(2) 1) + (%non?(2) 0). :byte. CODE 40)
%←?  (%all? (:byte. "op←0. CODE 40)
       :#s2.%[:lb2.%to.:ub2.%]. "byte ← 0. "op ← 5. CODE 40)
"op ← 6. "byte ← 0. "ub2 ← ub+1-lb.
"s2 ← (s is string?(string ub2)vector ub2).  CODE 40)

to cons : head tail (
        %tail?(%←?(!:tail)!tail)
        %head?(%←?(!:head)!head)
        (isnew?(:head. :tail))
        %print?(disp←40. head print. disp←46. tail print. disp←41))
to sequence : str ptr (
        %scan?(CODE 41?(!rpar))
        %pos?(%←?(!:ptr)!ptr)
```

```
        %fill?(for ptr to str length
                (10=str[ptr]←disp←kbd?(done))"ptr←0)
        isnew?(:str. "ptr←0) )

to reed v s : : ownv lpar rpar subreed growreed (%knows?(ev)
        "v←ownv. "s←v length. !subreed 1)
reed knows
to subreed i j t ("j←:i.
        repeat("t←reader scan.
                rpar=t?(v[j]←nil. done with sscan v[i to j])
                lpar=t?(v[j]←subreed j. s < "j←j+1?(growreed))
                v[j]←t. s < "j←j+1?(growreed) ))
to growreed ("v←sscan v[1 to "s←s+ (200>core?(50)s)] )
"ownv←vector 200.
"reader←sequence string 2. reader fill.
()  "lpar←reader scan. "rpar←reader scan.
done

printing 1 STOP "(2/22 Smalltalk)
```