Input and output routines

```
PROC write.string(CHAN output, VALUE string[])
  -- Write the characters of the string[] to the output

PROC write.signed(CHAN output, VALUE n, field.width)
  -- Write a signed decimal representation of n to the output,
  -- right justified to occupy field.width character spaces

PROC read.signed(CHAN input, VAR n, ok)
  -- Read an (optionally signed) decimal numeral from the input
  -- returning the corresponding value in n, and TRUE or FALSE
  -- in ok according as the conversion worked or not

PROC read.line(CHAN keyboard, screen, VAR s[])
  -- Construct a string in s[] from the printable characters
  -- read from keyboard and echoed to screen. The string
  -- finishes at a carriage return.
```

```
PROC write.string(CHAN output, VALUE string[]) =
  -- Write the characters of the string[] to the output
  SEQ character.number = [1 FOR string[BYTE 0]]
    output ! string[BYTE character.number]         :
```

```
PROC write.signed(CHAN output, VALUE n, field.width) =
  -- Write a signed decimal representation of n to the output,
  -- right justified to occupy field.width character spaces
  VAR tens, width :         -- tens will be a signed power of ten
  SEQ
    IF
      n >= 0
        SEQ
          tens  := -1
          width :=  1       -- count a minimum of one digit
      n < 0
        SEQ
          tens  :=  1
          width :=  2       -- count a sign and a minimum of one digit

    WHILE (n / tens) <= (-10)   -- set tens so that  0<(-(n/tens))<=10
      SEQ                       -- or, if n = 0 then tens = -1
        tens  := 10 * tens
        width := width + 1

    WHILE width < field.width    -- pad with spaces to field.width
      SEQ
        output ! '*s'
        width := width + 1

    IF                           -- output a sign for negative n
      n >= 0
        SKIP
      n < 0
        output ! '-'

    WHILE tens <> 0 -- output the digits of n, most significant first
      SEQ
        output ! '0' - ((n / tens) \ 10)
        tens := tens / 10                   :
```

```
PROC read.signed(CHAN input, VAR n, ok) =
  -- Read an (optionally signed) decimal numeral from the input
  -- returning the corresponding value in n, and TRUE or FALSE in
  -- ok according as the conversion worked or not

  DEF min = NOT ((NOT 0) >> 1), max = (NOT 0) >> 1 :
  DEF otherwise = TRUE :

  VAR ch, sign :
  SEQ

    input ? ch
    WHILE ch = '*s'                -- skip leading spaces
      input ? ch

    IF
      (ch = '+') OR (ch = '-')     -- read a possible sign
        SEQ
          sign := ch
          input ? ch
      (ch <> '+') AND (ch <> '-')
        sign := '+'

    WHILE ch = '*s'                -- skip any spaces after the sign
      input ? ch

    n := 0
    ok := ('0' <= ch) AND (ch <= '9')  -- check for digits

    WHILE ('0' <= ch) AND (ch <= '9')  -- and read a sequence of them
      SEQ
        IF
          (sign = '+') AND (n <= ((max - (ch - '0')) / 10))
            n := (10 * n) + (ch - '0')
          (sign = '-') AND (((min + (ch - '0')) / 10) <= n)
            n := (10 * n) - (ch - '0')
          otherwise
            ok := FALSE            -- number out of representable range
        input ? ch                             :
```

```
PROC read.line(CHAN keyboard, screen, VAR s[]) =
  -- Construct a string in s[] from the printable characters
  -- read from keyboard and echoed to screen. The string
  -- finishes at a carriage return.

  DEF control    = NOT ((NOT 0) << 5),
      otherwise  = TRUE,
      backspace  = control /\ 'H',
      bell       = control /\ 'G',
      cancel     = control /\ 'U',
      delete     = NOT ((NOT 0) << 7),
      max.length = NOT ((NOT 0) << 8) :

  SEQ
    s[BYTE 0] := 0       -- byte zero contains the length of the string
    WHILE s[BYTE s[BYTE 0]] <> '*C'
      VAR ch :
      keyboard ? ch
      IF
        ('*S' <= ch) AND (ch < delete) AND
                                 (s[BYTE 0] < (max.length - 1))
          SEQ                           -- 'printable' characters are
            screen ! ch                 --     echoed
            s[BYTE 0] := s[BYTE 0] + 1
            s[BYTE s[BYTE 0]] := ch     --    and added to the string
        ch = '*C'
          SEQ                           -- carriage return
            s[BYTE 0] := s[BYTE 0] + 1  --    is added to the string
            s[BYTE s[BYTE 0]] := ch     --    and terminates the loop
        (ch = backspace) AND (s[BYTE 0] > 0)
          SEQ                           -- backspace
            screen ! backspace ; '*S'   --   overwrites the last
            screen ! backspace          --   character echoed
            s[BYTE 0] := s[BYTE 0] - 1  --   and removes it from the
                                        --   string
        ch = cancel
          WHILE s[BYTE 0] > 0           -- cancel
            SEQ                         --   backspaces over the
                                        --   whole of the line
              screen ! backspace ; '*S' ; backspace
              s[BYTE 0] := s[BYTE 0] - 1
        otherwise                       -- anything else is an error
          screen ! bell                 :
```