Simple Huffman coder

```
PROC insert.new.node( VAR new.node, VALUE weight.of.new.node,
                      VAR left.limit, VALUE right.limit        )

PROC construct.tree(VALUE probability[])

  PROC construct.leaves
    -- build the leaves of the tree

  PROC construct.other.nodes
    -- join pairs of subtrees until only one tree remains

  PROC invert.representation
    -- set parent[] and representative[]

  -- construct.tree

PROC encode.character(CHAN output, VALUE ch)
  -- Transmit the encoding of ch along output

PROC decode.character(CHAN input, VAR ch)

PROC copy.encoding(CHAN source, end.of.source, sink)
  -- Read characters from source, sending their encodings along
  -- sink, until a signal is received along end.of.source.

PROC copy.decoding(CHAN source, sink)
  -- Read a bit stream from source, decoding it into characters
  -- and send these along sink until end.of.message is decoded
```

```
DEF bits.in.character    = 8,
    number.of.characters = 1 << bits.in.character,
    number.of.codes      = number.of.characters,
    character.mask       = NOT ((NOT 0) << bits.in.character) :



DEF root = 0,  size.of.tree = (2 * number.of.codes) - 1 :



VAR  eldest[size.of.tree],
     parent[size.of.tree],
     character[size.of.tree],
     representative[number.of.characters] :




PROC insert.new.node( VAR new.node, VALUE weight.of.new.node,
                      VAR left.limit, VALUE right.limit       ) =
  VAR weight.limit :
  SEQ
    IF
      IF node = [left.limit FOR right.limit - left.limit]
        weight[node] <= weight.of.new.node
          weight.limit := node
      TRUE
        weight.limit := right.limit
    SEQ node = [left.limit FOR weight.limit - left.limit]
      SEQ
        character[node - 1] := character[node]
        eldest[node - 1]    := eldest[node]
        weight[node - 1]    := weight[node]
    left.limit := left.limit - 1
    new.node := weight.limit - 1
    weight[new.node] := weight.of.new.node             :
```

```
PROC construct.tree(VALUE probability[]) =
  VAR left.limit, right.limit, weight[size.of.tree] :

  PROC construct.leaves =
    -- build the leaves of the tree
    DEF minimum.character = - (number.of.characters / 2) :
    SEQ ch = [minimum.character FOR number.of.characters]
      VAR new.node :
      SEQ
        insert.new.node(new.node, probability[ch /\ character.mask],
                                             left.limit, right.limit)
        eldest[new.node]    := root
        character[new.node] := ch          :

  PROC construct.other.nodes =
    -- join pairs of subtrees until only one tree remains
    WHILE (right.limit - left.limit) <> 1
      VAR new.node :
        SEQ
          right.limit := right.limit - 2
          insert.new.node(new.node,
                          weight[right.limit] + weight[right.limit+1],
                                              left.limit, right.limit)
          eldest[new.node] := right          :

  PROC invert.representation =
    -- set parent[] and representative[]
    SEQ node = [root FOR size.of.tree]
      IF
        eldest[node] = root
          representative[character[node] /\ character.mask] := node
        eldest[node] <> root
          SEQ child = [eldest[node] FOR 2]
            parent[child] := node                                  :

SEQ
  left.limit   := size.of.tree + 1
  right.limit  := size.of.tree + 1
  -- here  left.limit = (size.of.tree + 1)
  -- and   (right.limit - left.limit) = 0
  construct.leaves
  -- here  left.limit = number.of.code
  -- and   (right.limit - left.limit) = number.of.codes
  construct.other.nodes
  -- here  left.limit = root
  -- and   (right.limit - left.limit) = 1
  invert.representation                    :
```

```
PROC encode.character(CHAN output, VALUE ch) =
  -- Transmit the encoding of ch along output
  DEF size.of.encoding = number.of.codes - 1 :
  VAR encoding[size.of.encoding], length, node :
  SEQ
    length := 0
    node   := representative[ch /\ character.mask]
    WHILE node <> root
      SEQ
        encoding[length] := node - eldest[parent[node]]
        length := length + 1
        node   := parent[node]
    SEQ i = [1 FOR length]
      output ! encoding[length - i]              :
```

```
PROC decode.character(CHAN input, VAR ch) =
  VAR node :
  SEQ
    node := root
    WHILE eldest[node] <> root
      VAR bit :
      SEQ
        input ? bit
        node := eldest[node] + bit
    ch := character[node]                :
```

```
DEF probability = TABLE[ ... ] :
   -- indexed by [0 FOR number.of.characters]
```

```
DEF end.of.message = -1 :
```

```
PROC copy.encoding(CHAN source, end.of.source, sink) =
  -- Read characters from source, sending their encodings along
  -- sink, until a signal is received along end.of.source.
  VAR more.characters.expected :
  SEQ
    construct.tree(probability)
    more.characters.expected := TRUE
    WHILE more.characters.expected
      VAR ch :
      ALT
        source ? ch
          encode.character(sink, ch)
        end.of.source ? ANY
          more.characters.expected := FALSE
    encode.character(sink, end.of.message)          :


PROC copy.decoding(CHAN source, sink) =
  -- Read a bit stream from source, decoding it into characters
  -- and send these along sink until end.of.message is decoded
  VAR more.characters.expected :
  SEQ
    construct.tree(probability)
    more.characters.expected := TRUE
    WHILE more.characters.expected
      VAR ch :
      SEQ
        decode.character(source, ch)
        IF
          ch <> end.of.message
            sink ! ch
          ch = end.of.message
            more.characters.expected := FALSE    :
```