

1 ( Huffman Tree Builder: field addr @!  
2 ( Huffman Tree Builder: ?ACTIVE 2\*H+1 2\*H  
3 ( Huffman Tree Builder: HUFFARRAY H:  
4 ( Huffman Tree Builder: DOWN/OP DESCEND  
5 ( Huffman Tree Builder: variables  
6 ( Huffman Tree Builder: LEAST2  
7 ( Huffman Tree Builder: CODEH1 CODEH2  
8 ( Huffman Tree Builder: H1>H2  
9 ( Huffman Tree Builder: HUFFMAN  
10 ( Huffman Tree Builder: HUFFCODE  
11 ( Huffman Branch Table: variables  
12 ( Huffman Branch Table: ?LEAF >2LEAF >0LEAF  
13 ( Huffman Branch Table: >LEAF >RLEAF  
14 ( Huffman Branch Table: DESCEND2  
15 ( Huffman Branch Table: BRANCH-TABLE  
16 ( Huffman Branch Table: [ .BTABLE] .BTABLE  
17 ( Huffman Encoder: variables string-buffers  
18 ( Huffman Encoder: ENCODE-TRACE  
19 ( Huffman Encoder: CHAR>RC\$  
20 ( Huffman Encoder: DELTA-PTRS RCODE>HB\$  
21 ( Huffman Encoder: DATA\$>HB\$ >HUFF

1

4

```

0 ( Huffman Tree Builder: field addr @.!          3/12/85 AWK ) ( Huffman Tree Builder: DOWN/OP DESCEND          3/12/85 AWK )
1 V= HUFFCNT      \ count of nodes (incl. leaves) generated
2 V= HUFFADDR     \ address of huffman tree table
3 V= NODE         \ address of current node in table
4
5 : HCOUNT ( -- addr ) NODE @ 1+ ; \ freq cnt field
6 : PARENT ( -- addr ) NODE @ 5+ ; \ parent ptr field
7 : LCHILD ( -- addr ) NODE @ 7+ ; \ left(0) child ptr field
8 : RCHILD ( -- addr ) NODE @ 9+ ; \ right(1) child ptr field
9 : HCODE ( -- addr ) NODE @ 11+ ; \ reverse code field
10
11 : @HCODE ( -- d ) HCODE D@ ; \ fetch reverse code
12 : !HCODE ( -- d ) HCODE D! ; \ store reverse code
13 : @HCOUNT ( -- d ) HCOUNT D@ ; \ fetch node frequency count
14 : !HCOUNT ( -- d ) HCOUNT D! ; \ store node frequency count
15
: DOWN/OP ( n -- )
?DUP IF NODE @ >V NODE ! 'RECURSE EXECUTE V) NODE !
ELSE 'OPERATION EXECUTE THEN ;
\ descend one level in the tree or perform link operation
: DESCEND ( -- )
LCHILD @ DUP DOWN/OP
IF RCHILD @ DOWN/OP THEN ;
\ recursive descend called by, and calls, DOWN/OP
' DESCEND ' 'RECURSE !

```

2

5

```

0 ( Huffman Tree Builder: ?ACTIVE 2*H+1 2*H          3/12/85 AWK ) ( Huffman Tree Builder: variables          3/12/85 AWK )
1
2 V= #UNLINKED    \ number of nodes waiting to linked into tree
3
4 : ?UNLINKED ( -- f ) PARENT @ 0= DUP #UNLINKED +! ;
5
6 : 2*H+1 ( -- ) @HCODE 2DUP D+ 1. D+ !HCODE ;
7 : 2*H ( -- ) @HCODE 2DUP D+ !HCODE ;
8 \ add a 1 or a 0 to the reverse code
9
10
11
12
13
14
15
V= H1      \ active node w/lowest freq count
V= H2      \ active node w/2nd lowest freq count
2V= LEAST  \ lowest freq count
2V= 2NDLEAST \ 2nd lowest freq count

```

3

6

```

0 ( Huffman Tree Builder: HUFFARRAY H:          3/12/85 AWK ) ( Huffman Tree Builder: LEAST2          3/12/85 AWK )
1
2 : HUFFARRAY ( n -- )
3 HERE HUFFADDR ! HERE SWAP 30 * DUP ALLOT 0 FILL HUFFCNT OFF ;
4 \ allot space for an array capable of holding a binary tree of
5 \ n elements (i.e. 2n-1 nodes)
6
7 : H: ( -- ) \ <char> <freq>
8 HUFFADDR @ HUFFCNT @ 15 * + NODE ! \ HUFFARRAY row address
9 [COMPILE] ASCII NODE @ C! \ store ascii char value
10 BL WORD NUMBER !HCOUNT \ store count #
11 1. !HCODE \ store code place-holder
12 1 HUFFCNT +! ; \ inc huffcnt
13 \ compile word for character & frequency w/which it occurs.
14 \ HUFFARRAY must be executed first.
15
: LEAST2 ( -- )
9999999. 2DUP LEAST D! 2NDLEAST D! #UNLINKED OFF
HUFFCNT @ 0 DO HUFFADDR @ I 15 * + NODE ! ?UNLINKED
IF @HCOUNT 2DUP LEAST D@ D<
IF LEAST DUP D@ 2NDLEAST D! D! H1 @ H2 ! NODE @ H1 !
ELSE 2DUP 2NDLEAST D@ D<
IF 2NDLEAST D! NODE @ H2 ! ELSE 2DROP THEN
THEN
LOOP ;
\ try to find 2 unlinked nodes (if so, w/least counts).
\ #UNLINKED contains count of unlinked nodes after execution of
\ LEAST2.

```

7

10

```

0 ( Huffman Tree Builder: CODEH1 CODEH2          3/12/85 AWK ) ( Huffman Tree Builder: HUFFCODE          3/12/85 AWK )
1
2 : CODEH1 ( -- )                                : HUFFCODE ( -- )
3 ' 2+H ' OPERATION ! H1 @ NODE ! DESCEND :      HUFFCNT @ 0 DO
4                                              HUFFADDR @ I 15 * + NODE !
5 \ add 1 to reverse code of leaves under H1      NODE @ C@ ?DUP
6                                              IF CR EMIT SPACE
7 : CODEH2 ( -- )                                BINARY @HCODE
8 ' 2+H+1 ' OPERATION ! H2 @ NODE ! DESCEND :    <# #S #> OVER + SWAP 1+ SWAP 1- DO I C@ EMIT -1 +LOOP
9                                              DECIMAL THEN
10 \ add 0 to reverse code of leaves under H2      LOOP ;
11
12                                              \ reverses the reverse code, and eliminates the place holder to
13                                              \ display the true Huffman code for each of the characters in
14                                              \ the tree.
15

```

8

11

```

0 ( Huffman Tree Builder: H1><H2          3/12/85 AWK ) ( Huffman Branch Table: variables          3/22/85 AWK )
1
2 : H1><H2 ( -- )                                V= BTADDR          \ address of branch table
3 H1 @ NODE ! @HCOUNT H2 @ NODE ! @HCOUNT D+ \ sum counts V= BTCNT          \ node count
4 HUFFADDR @ HUFFCNT @ 15 * + NODE ! !HCOUNT \ ! parent count V= NODE-OFFSET \ branch offset register in # of dbl bytes
5 H1 @ LCHILD ! H2 @ RCHILD !                  \ ! child ptrs ' NOP C= 'RECURSE2 \ pfa of recursive routine (see below)
6 NODE @ DUP H1 @ NODE ! PARENT !              \ ! parent ptr
7 H2 @ NODE ! PARENT !                          \ ! parent ptr
8 1 HUFFCNT +! ;                                \ inc huffcnt
9
10 \ link the nodes whose addresses are stored in H1 and H2. the
11 \ new node has a freq. count equal to the sum of its children's.
12
13
14
15

```

9

12

```

0 ( Huffman Tree Builder: HUFFMAN          3/12/85 AWK ) ( Huffman Branch Table: ?LEAF >2LEAF >OLEAF          3/22/85 AWK )
1
2 : CODING-TREE ( -- )                                : ?LEAF ( addr -- f ) \ f= 1: leaf 0: not a leaf
3 BEGIN LEAST2 #UNLINKED @ 1 >                                NODE @ >R NODE ! LCHILD @ RCHILD @ OR 0= R> NODE ! ;
4 WHILE CODEH1 CODEH2 H1><H2                                \ is this node a leaf?
5 REPEAT ;
6
7 \ while the number of active nodes is at least 2, link the two : >2LEAF ( -- )
8 \ with the lowest frequency counts, when only one active node LCHILD @ C@ C, RCHILD @ C@ C, ;
9 \ remains, it is the super-node, and the tree is complete. \ store the 2 leaves (chars) into the branch table
10
11 : >OLEAF ( -- )
12 HERE >V 0 C, 129 C,          \ sav lft addr & store rt offset
13 RCHILD @ 'RECURSE2 EXECUTE \ descend right branch
14 NODE-OFFSET @ 128 OR V> C! \ rcl left addr & store offset
15 LCHILD @ 'RECURSE2 EXECUTE ; \ descend left branch
\ no leaves -- recursively descend both branches

```

13

16

```

0 ( Huffman Branch Table: >LLEAF >RLEAF          3/22/85 AWK ) ( Huffman Branch Table: [.BTABLE] .BTABLE          3/22/85 AWK )
1
2 : >LLEAF ( -- )                                : (.BTABLE) ( addr -- )
3 LCHILD @ C@ C, 129 C, \ store left leaf & right offset  DUP C@ DUP 128 AND
4 RCHILD @ 'RECURSE2 EXECUTE ; \ descend right branch    IF 127 AND 2* 3 .R SPACE ELSE
5 \ left branch is leaf -- store it & descend right branch  DUP 2 SPACES EMIT 48 58 WITHIN 10 * 32 + EMIT THEN 1+ SPACE
6                                                         C@ DUP 128 AND
7 : >RLEAF ( -- )                                IF 127 AND 2* 3 .R SPACE ELSE
8 129 C, RCHILD @ C@ C, \ store left offset & right leaf  DUP 2 SPACES EMIT 48 58 WITHIN 10 * 32 + EMIT THEN SPACE ;
9 LCHILD @ 'RECURSE2 EXECUTE ; \ descend left branch
10 \ right branch is leaf -- store it & descend left branch : .BTABLE ( -- )
11                                                         BT CNT @ 0 DO I 8 MOD 0=
12 \ for all of the above >xLEAF words, a branch offset is IF CR THEN
13 \ differentiated from a character by bit 7 (128) being set BT ADDR @ I + (.BTABLE) 4 SPACES
14 \ as the branch flag.                                  2 +LOOP ;
15                                                         \ display routine for the branch table

```

14

17

```

0 ( Huffman Branch Table: DESCEND2          3/22/85 AWK ) ( Huffman Encoder: variables string-buffers          3/29/85 AWK )
1
2 : DESCEND2 ( addr -- )                        V= HB$-PTR \ pointer to current byte in HB$
3 2 BT CNT +! \ inc table count                V= D$-OFF \ offset of current char in D$
4 NODE @ >V NODE ! \ save node addr            V= HBIT \ current bit of HB$ + HB$-PTR
5 1 NODE-OFFSET DUP @ >V ! \ save & set node offset V= 0$ADDR \ pointer to original string address
6 LCHILD @ ?LEAF
7 IF RCHILD @ ?LEAF                            CREATE HB$ 257 ALLOT \ buffer for compressed string
8 IF >2LEAF ELSE >LLEAF THEN                  \ b0 is compressed count
9 ELSE RCHILD @ ?LEAF                          \ b1 is uncompressed count
10 IF >RLEAF ELSE >OLEAF THEN
11 THEN V> NODE-OFFSET +! \ rcl & inc node offset CREATE D$ 256 ALLOT \ holding space for data string
12 V> NODE ! ; \ rcl node addr
13 \ recursive descend finds # of leaves & descends accordingly CREATE RC$ 33 ALLOT \ reverse code string
14
15 ' DESCEND2 ' 'RECURSE2 !

```

15

18

```

0 ( Huffman Branch Table: BRANCH-TABLE          3/22/85 AWK ) ( Huffman Encoder: ENCODE-TRACE          3/29/85 AWK )
1
2 : BRANCH-TABLE ( -- )
3 HERE BT ADDR ! NODE-OFFSET OFF BT CNT OFF
4 HUFFADDR @ HUFFCNT @ 1- 15 * + DESCEND2 ;
5
6 \ builds the branch table, one call to the recursive descend
7 \ function scans the entire tree.
8
9 : HUFFMAN ( -- ) CODING-TREE BRANCH-TABLE ;
10
11
12
13
14
15

```

19

```

0 ( Huffman Encoder: CHAR>RC$ 3/29/85 AWK )
1
2 : CHAR>RC$ ( b -- )
3 BINARY RC$ OFF
4 HUFFADDR @ HUFFCNT @ 15 * OVER + SWAP DO
5 I C@ OVER =
6 IF I 11 + D@ <# #5 #>
7 DUP RC$ C! RC$ 1+ SWAP CMOVE
8 LEAVE
9 THEN
10 15 +LOOP DECIMAL DROP
11 RC$ C@ 0= ABORT" Couldn't find next character." ;
12
13 \ find char b's reverse code, convert it to a string, and move
14 \ it to the reverse string code string buffer.
15

```

20

```

0 ( Huffman Encoder: DELTA-PTRS RCODE>HB$ 5/02/85 AWK )
1
2 : DELTA-PTRS ( -- )
3 HBIT @ DUP 128 = HB$-PTR +! 2/ DUP 0= 128 * + HBIT ! ;
4
5 \ change current bit value and compressed string pointer.
6
7 : RCODE>HB$ ( -- )
8 RC$ COUNT OVER + 1- SWAP 1+ SWAP DO
9 HB$ HB$-PTR @ + DUP C@ I C@ 49 = HBIT @ * OR SWAP C!
10 DELTA-PTRS
11 -1 +LOOP ;
12
13 \ convert reverse code to forward code while moving it to the
14 \ compressed string buffer.
15

```

21

```

0 ( Huffman Encoder: DATA$>HB$ >HUFF 5/02/85 AWK )
1
2 : DATA$>HB$ ( -- )
3 D$ COUNT OVER + SWAP DO
4 I C@ CHAR>RC$ RCODE>HB$
5 LOOP
6 HB$-PTR @ HB$ C! ;
7 \ move data buffer Huffman buffer while compressing
8
9 : >HUFF ( addr cnt -- addr' cnt' )
10 OVER SWAP HB$ OFF HB$ 1+ C! 1- D$ $!
11 HB$ 2+ 255 0 FILL
12 128 HBIT ! 2 HB$-PTR !
13 DATA$>HB$
14 1- HB$ OVER $! COUNT ;
15 \ translate string into Huffman compressed string

```