LZW compression used to encode/decode a GIF file by Bob Montgomery [73357,3140]

ENCODER
Consider the following input data stream in a 4 color (A, B, C, D) system. We will build a table of codes which represent strings of colors. Each time we find a new string, we will give it the next code, and break it into a prefix string and a suffix color. The symbols \ or --- represent the prefix string, and / represents the suffix color. The first 4 entries in the table are the 4 colors with codes 0 thru 3, each of which represents a single color. The next 2 codes (4 and 5) are the clear code and the end of image code. The first available code to represent a string of colors is 6. Each time we find a new code, we will send the prefix for that code to the output data stream.

```
A B A B A B A B B B A B A B A  A  C  D A C D A D  C A B A A A B A B .....
\/\/---/-----/\/---/-------/\/ \/ \ /\/---/---/\ /\/-----/---/---/
6 7   8     9 10 11      12 13 14 15 16  17 18 19 20   21  22  23     Code
    6    8      10    8               14  16        8    13  7       Prefix
```

The encoder table is built from input data stream. Always start with the suffix of last code, and keep getting colors until you have a new combination.

```
Color     Code      Prefix    Suffix    String    Output
 A        0                             -
 B        1                             -
 C        2                             -
 D        3                             -
Clear     4                             -
End       5                             -
 A \                A         A                   First color is a special case.
 B /  \   6         A         B         AB        A
 A |  /   7         B         A         BA        B
 B |
 A /  |   8         6         A         ABA       6
 B    |
 A    |
 B \  /   9         8         B         ABAB      8
 B /  |   10        B         B         BB        B
 B    |
 A |  /   11        10        A         BBA       10
 B |
 A |
 B |
 A /  \   12        9         A         ABABA     9
 A \  /   13        A         A         AA        A
 C /  \   14        A         C         AC        A
 D \  /   15        C         D         CD        C
 A /  |   16        D         A         DA        D
 C    |
 D |  /   17        14        D         ACD       14
 A |
 D /  \   18        16        D         DAD       16
 C \  /   19        D         C         DC        D
 A /  |   20        C         A         CA        C
 B    |
 A    |
 A |  /   21        8         A         ABAA      8
 A |
 B /  |   22        13        B         AAB       13
 A    |
 B    /   23        7         B         BAB       7
```

The resultant output stream is: A B 6 8 B 10 9 A A C D 14 16 D C 8 ....
The GIF encoder starts with a code length of 2+1=3 bits for 4 colors, so when the code reaches 8 we will have to increase the code size to 4 bits. Similarly, when the code gets to 16 we will have to increse the code size to 5 bits, etc. If the code gets to 13 bits, we send a clear code and start over. See GIFENCOD.GIF for a flow diagram of the encoding process. This uses a tree method to search if a new string is already in the table, which is much simpler, faster, and easier to understand than hashing.

===

DECODER

We will now see if we can regenerate the original data stream and duplicate the table looking only at the output data stream generated by the encoder on the previous page. The output data stream is:

```
          A B 6 8 B 10 9 A A C D 14 16 D C 8 ....
```

The docoding process is harder to see, but easier to implement, than the encoding process. The data is taken in pairs, and a new code is assigned to each pair. The prefix is the left side of the pair, and the suffix is the color that the right side of the pair decomposes to from the table. The decomposition is done by outputing the suffix of the code, and using the prefix as the new code. The process repeats until the prefix is a single color, and it is output too. The output of the decomposition is pushed onto a stack, and then poped off the stack to the display, which restores the original order that the colors were seen by the encoder. We will go thru the first few entries in detail, which will hopefully make the process clearer.

The first pair is (A B), so the prefix of code 6 is A and the suffix is B, and 6 represents the string AB. The color A is sent to the display.

The 2nd pair is (B 6), so the prefix of code 7 is B and the suffix is the the last color in the decomposition of code 6. Code 6 decomposes into BA, so code 7 = BA, and has a suffix A. The color B is sent to the display.

The 3rd pair is (6 8) and the next code is 8. How can we decompose 8. We know that the prefix of code 8 is 6, but we don't know the suffix. The answer is that we use the suffix of the prefix code; A in this case since the suffix of 6 is A. Thus, code 8 = ABA and has a suffix A. We decompose 6 to get BA, which becomes AB when we pop it off the stack to the display.

The 4th pair is (8 B), so code 9 has a prefix of 8 and a suffix of B, and code 9 = ABAB. We output ABA to the stack, and pop it off to the display as ABA.

The 5th pair is (B 10) and the next code is 10. The prefix of code 10 is B and the suffix is B (since the prefix is B). Code 10 = BB, and we output the prefix B to the display.

The 6th pair is (10 9) and the next code is 11. Thus the prefix of code 11 is 10 and the suffix is the last color in the decomposition of 9, which is A. Thus code 11 = BBA, And we output BB to the display.

So far, we have output the correct colors stream A B AB ABA B BB to the display, and have duplicated the codes 6 thru 11 in the encoder table. This process is repeated for the whole data stream to reconstruct the original color stream and build a table identical to the one built by the encoder. We start the table with codes 0-5 representing the 4 colors, the clear code, and the end code. When we get to code 8, we must increse the code size to 5 bits, etc. See GIFDECOD.GIF for a flow diagram of the decoding process.

I Hope this helps take some of the mystery out of LZW compression, which is really quite easy once you 'see' it.   Bob Montgomery