Run-Length Encoding (RLE)

Run-length encoding (RLE) is one of the simplest data compression methods. Consider the example in which we have represented an MxN image whose top half s totally white, and bottom half is totally black. That example was a primitive attempt to encode the image using RLE. The principle of RLE is to exploit the repeating values in a source. The algorithm counts the consecutive repetition amount of a symbol and uses that value to represent the **run**. This simple principle works best on certain source types in which repeated data values are significant. Black-White document images, cartoon images, etc. are quite suitable for RLE.

Actually, RLE may be used on any kind of source regardless of its content, but its compression efficiency changes significantly depending on the above types of data are used or not.

As another application suitable for RLE, we can mention text files which contains multiple spaces for indention and formatting paragraphs, tables and charts.

Principle :
As indicated above, basic RLE principle is that the run of characters is replaced with the number of the same characters and a single character. Examples may be helpful to understand it better.
Ex. 8:
Consider a text source: R T A A A A S D E E E E E
The RLE representation is: R T *4A S D *5E
This example also shows how to distinguish whether a symbol corresponds to the daa value or its repetition count (called run). Each repeating bunch of characters is replaced with three symbols: **an indicator (*), number of characters**, and the **character itself**. We need the indication of he redundant character * to separate between the encoding of a repeating cluster and a single character. In the above example, if there is no repetition around a character, it is encoded as itself.

In the above example, it is important to realize that the encoding process is effective only if there are sequences of 4 or more repeating characters because three characters are used to conduct RLE. For example, coding two repeating characters would lead to expansion and coding three repeating characters wouldn't cause compression or expansion since we represent a repetitive cluster with at least three symbols.

The decoding process is easy: If there aren't control characters (*) the coded symbol just corresponds to the original symbol, and if control character occurred then it must be replaced with characters in a defined number of times. It can be noticed that the process of decoding control characters don't lead to any special procedures.

The RLE symbolism and details are not unique. There are many different run-length encoding schemes. The above example has just been used to demonstrate the basic principle of RLE encoding. In a particular case the implementation of RLE depends on what type of data is being compressed.

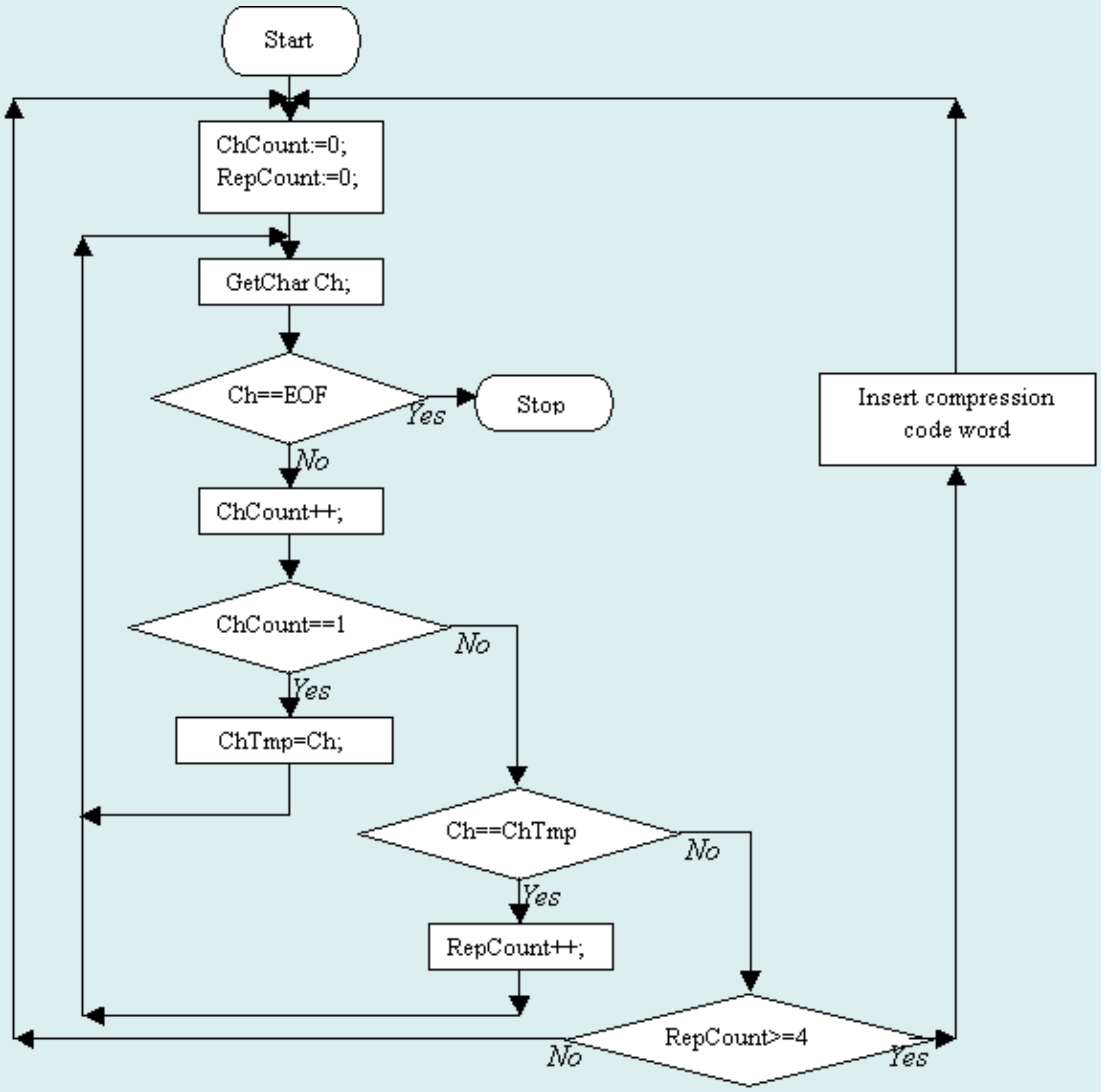
Nevertheless the encoder applies a three symbol encoding strategy to represent a repetitive cluster:

CTRL	COUNT	CHAR

Here, the following terminology is defined:

- CTRL - control character which is used to indicate compression
- COUNT- number of counted characters in stream of the same characters
- CHAR - repeating characters

The chart below illustrates a typical RLE encoder for text files. Similar algorithms and charts may be adopted for image, audio, etc.



RLE algorithms are practically used in various image compression techniques like the well known BMP, PCX, TIFF, and is also used in PDF file format. Furthermore, RLE also exists as separate compression technique and there is also a file format called RLE (in various brands).

One of the companies that incorporates RLE is **CompuServe**. The file format is threefore CompuServe RLE, which was standardized in the 80's and it was aimed to compress for 1-bit (black and white) images.

Note: The standards are usually patented. They contain specific headers that indicate the technique and the name of the standard. For example, in CompuServe RLE, header part indicates or represents the Graphic Mode Control, which indicates that it is a CompuServe image. The standard initial header symbol sequence is : ASCII ESC (HEX 1B), ASCII G(HEX 47) and the third character is ASCII H (HEX 48) or M (HEX 4D). Third character represents resolution (out of two possible graphics modes : high resolution graphic mode (256 x 192 pixels) represented by <H> and medium resolution graphic mode (128 x 96 pixels) represented by <M>). Therefore, typically, if a file starts with <ESC><G><H> or <ESC><G><M>, then it is a CompuServe RLE image.

After header sequence, the application can identify that it is a CompuServe RLE image. Now the data sequence starts.

Basic data sequence consists of a pair of run length encoded ASCII characters. The first number represents number of the background (off) pixels and the second character is the number of foreground (on) pixels. Each number of a pair represents the count number of pixels plus 32 decimal, i.e. from each number 32 is substracted and that number represents how many next pixels will be turned on or turned off depending on what number of pair we observe. Usually the value 126 is used as the highest possible value, because of some limitations in some old computer terminals. This limitation leads us to the conclusion that in each byte we can denote repetition of at most 94 pixels (126 - 32). For example pair <D><'> (HEX: 44 27, DECIMAL 68 39) means next 68 (decimal) pixels are turned off and then 39 (decimal) pixels are turned on.

As you see, there is no control character (like '*') between the encoded symbols because there are only two distinct source levels: 0 and 1. For more general data types (color o gray-level images), CompuServe RLE is modified to incorporate the control chaacter.

Finally, the file formats must also define where the input data ends. The ending sequence for RLE standard consists of three characters <ESC><G><N>.

Ex. 9: A typical CompuServe RLE file (all numbers are in ASCII HEX format):

```
1B 47 48 7E 20 7E 20 7E 20 7E 20 ....
. . . . . . . . . . . .
. 41 36 . . . . . . . . .
. . . . . . . . . . . .
. . . 07 1B 47 4E
```

1B 47 48 - is header <ESC><G><H> and represents high resolution, first data sequence pair 20hex, 7Ehex means that first 94dec pixels are all turned on, the second data sequence is the same so second 94d pixels are also turned on (the first 188d pixels are turned on so far), and so on. Then somewhere in the file pairs 41h 36h occurs which means that next 33d pixels are turned off and after that 22d pixels are turned off, etc. Last four character are the ending sequence which was described above.

You may want to experiment with some real image data. Click here to download samples of CompuServe RLE images

Another standard that uses RLE is **MS Windows standard for RLE** file format. MS Windows standard for RLE have the same file format (in terms of headers, etc.) as the well-known BMP file format, but it's RLE format is defined only for 4-bit and 8-bit colour images.

4bit RLE: Compression sequence consists of two bytes, first byte (if not zero) determines number of pixels which will be drawn on the screen. The second byte specifies two colors, high-order 4 bits (upper 4 bits) specifies the first color, low-order 4bits specifies the second color. This means that 1st, 3rd and other odd pixels will be in drawn in the color specified by high-order bits, while 2nd, 4th and other even pixels will be in drawn in the color specified by low-order bits. If first byte is zero then the second byte specifies escape code which describes some specific behavior such as:

- 0 : End-of-line
- 1 : End-of-Rle(Bitmap)
- 2 : Following two bytes defines offset in x and y direction (x is right,y is up). The skipped pixels get color zero.
- >=3 : When expanding (decoding), the following >=3 nibbles (nibble means 4bits) are just copied from compressed file, file/memory pointer must be on 16bit boundary so adequate number of zeros follows

Ex. 10 : Example for 4bit RLE:

Compressed data	Expanded data
06 52	5 2 5 2 5 2
08 1B	1 B 1 B 1 B 1 B
00 06 83 14 34	8 3 1 4 3 4
00 02 09 06	Move 9 positions right and 6 up
00 00	End-of -line
04 22	2 2 2 2
00 01	End-of-RLE(Bitmap)

8bit RLE: Compressed sequence is also formed from 2 bytes (like the 4 bit RLE). the first byte (if not zero) is a number of consecutive pixels which are in color specified by the second byte. Same as in 4bit RLE if the first byte is zero the second byte defines escape code, escape codes 0, 1, 2, have same meaning as described in 4bit RLE.

Ex. 11: Examples for 8bit RLE

Compressed data	Expanded data
06 52	

	52 52 52 52 52 52
08 1B	1B 1B 1B 1B 1B 1B 1B 1B
00 03 83 14 34	83 14 34
00 02 09 06	Move 9 positions right and 6 up
00 00	End-of -line
04 2A	2A 2A 2A 2A
00 01	End-of-RLE(Bitmap)

Detailed description about MS Windows BMP file format (RLE is special case of BMP) is available here.

Other RLE file formats:

RLE scheme which will be described here is being used in PDF and TIFF file format. In this case, RLE encoded data consists of compression sequences. One compression sequence starts with number n (byte), this byte may be followed by 1 to 128 bytes (so these 2 to 129 bytes form one compression sequence). If n is between 0 and 127, then following n+1 (1 to 128) bytes are just copied during decompression. If n is between 129 and 255, then the byte value which follows n is copied 256-(n-1) (which is between 2 and 128) times in the decompressed file. If 128 occurs then it indicates the end of compressed data.

This scheme is similar to the PackBits encoding standard which is used in Macintosh systes.

Ex. 12 : For the above description you can consider the following example:

Compressed data-hex format	Decompressed data-hex format
07 A4 56 C9 90 E5 F1 DB 32	A4 56 C9 E5 F1 DB 32
02 23 A1 56	23 A1 56
FE 12	12 12 12
FC 6C	6C 6C 6C 6C 6C

Please check the following resources for further detailed descriptions on RLE:

- Detailed description of Utah RLE
- Another useful link about Utah RLE - file format description
- DjVu compression technology - RLE file format description- specified by AT&T
- Portable Document Format Reference Manual Version 1.3 November 16, 1999
- Unofficial CompuServe RLE specification
- Specification of MS Windows BMP