CISC 1110 (Science Section)
Brooklyn College
Professor Langsam

Assignment #5

File Compression – RLE Encoding[1]

Run-length encoding (RLE) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs: for example, relatively simple graphic images such as icons, line drawings, and animations.

For example, consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. Let us take a hypothetical single scan line, with B representing a black pixel and W representing white:

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWW
WWWWWWWWWWBWWWWWWWWWWWWW

If we apply the run-length encoding (RLE) data compression algorithm to the above hypothetical scan line, we get the following:

W12B1W12B3W24B1W14

Interpret this as twelve W's, one B, twelve W's, three B's, etc.

The run-length code represents the original 67 characters in only 18. Of course, the actual format used for the storage of images is generally binary rather than ASCII characters like this, but the principle remains the same.

However, consider the following sequence of characters:

AAAABCD

Using our algorithm we get the following:

A4B1C1D1

which is longer than the original! One solution is to only encode a run if *two* or more bytes are repeated. Thus we would encode the sequence as:

AA2BCD

[1] http://en.wikipedia.org/wiki/Run-length_encoding

We interpret this string to mean 2 ‘A’s followed by 2 additional ‘A’s followed by a single ‘B’, ‘C’ and ‘D’.

An algorithm for doing so is as follows[2]:

Encoder

Read two bytes, if they are equal output both of them, and then count how many bytes equal to the first you have; output this value, and continue encoding. Of course you have to discard the repeated bytes.

Note that the value can't be greater than 255, since a character (or byte) can not represent an unsigned integer greater than 255). For the purposes of this program we will assume that there are no repeats greater than 255. (For *extra credit*, you may suggest and program a solution.)

If the bytes were not equal, then output the first, make the second the first, and get the next byte as a second, and start again.

Pseudocode

```
Get two bytes
Loop
Are they equal?
Yes
        Output both of them
        Count how many bytes repeated we have
        Output that value
        Get next two bytes.
        Repeat.
No
        Output the first byte
        Put the second, as first
        Get a byte for the second one
        Repeat.
```

[2] http://www.arturocampos.com/ac_rle.html

Decoder

Pseudocode

```
Get one byte, put it to the output file, and now it's the 'last' byte.
Loop
Get one byte
Is the current byte equal to last?
Yes
        Now get another byte, this is 'counter'
        Put current byte in the output file
        Copy 'counter' times the 'last' byte to the output file
        Put last copied byte in 'last' (or leave it alone)
        Repeat.
No
        Put current byte to the output file
        Now 'last' is the current byte
        Repeat.
```

Write a program that implements the *Encoder* and *Decoder* algorithms. The encoder function should read a file (defined in the main) and write an encoded file having the same name but with the extension .enc. The decoder function should read a file with the extension .enc and produce a file with the extension .txt.

Test you program on a file called input.txt which contains the following:

aaaaaabcddccc

The file input.enc should then contain the sequence:

aa4bcdd0cc1

Additional data files will be posted on the course website.

Be sure your program is neatly formatted and commented as discussed in class. All output is to go to both the console (screen) as well as to a file.