

# Binary Image Compression using Run-Length-Encoding (RLE)

+

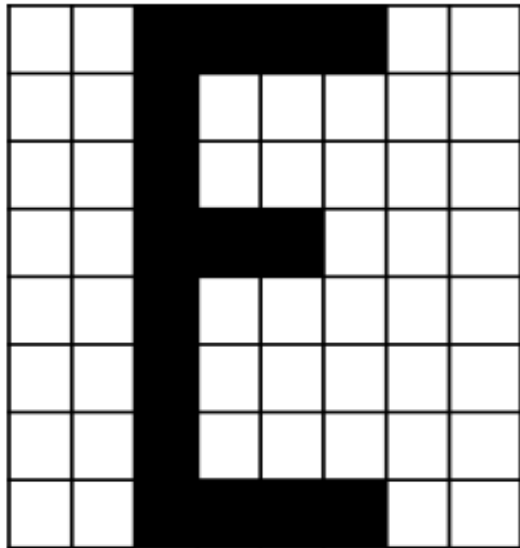
Author: Amalia Filimon

# Project Requirements

- + Implement a function that compresses a binary image using Run-Length-Encoding.
- + Save the compressed image as a binary file, which must contain the dimensions of the image.
- + Also implement the decompression function, which reconstructs the original image from the binary file.
- + Finally, compute the compression ratio for various input images.

# What is Run-Length Encoding (RLE)?

- + Run-length encoding (RLE) is a data compression technique used to reduce the size of repeated characters or data sequences.
- + It replaces consecutive repetitions of a symbol or group of symbols with a single occurrence and a counter indicating how many times it is repeated.
- + For example, the string "AAAABBBCCDAA" becomes "4A3B2C1D2A".



1	1	0	0	0	0	1	1
1	1	0	1	1	1	1	1
1	1	0	1	1	1	1	1
1	1	0	0	0	1	1	1
1	1	0	1	1	1	1	1
1	1	0	1	1	1	1	1
1	1	0	1	1	1	1	1
1	1	0	1	1	1	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1



w	w	b	b	b	b	w	w
w	w	b	w	w	w	w	w
w	w	b	w	w	w	w	w
w	w	b	w	w	w	w	w
w	w	b	b	b	w	w	w
w	w	b	w	w	w	w	w
w	w	b	w	w	w	w	w
w	w	b	w	w	w	w	w
w	w	b	w	w	w	w	w
w	w	b	b	b	b	w	w



2w4b2w	6 Bytes
2w1b5w	6 Bytes
2w1b5w	6 Bytes
2w3b3w	6 Bytes
2w1b5w	6 Bytes
2w1b5w	6 Bytes
2w1b5w	6 Bytes
2w4b2w	6 Bytes
<b>Total bytes</b>	<b>48</b>

# Compression Function

To implement the compression function for binary images using RLE:

- + A binary image contains only black and white pixels
- + The image is traversed row by row, pixel by pixel
- + For each sequence of identical pixels, we store:
  - + A count of consecutive pixels
  - + The color (W for white / B for black)
- + The result is saved in a **binary output file**.
  - + For example, "**WWWWW**" becomes "**5W**" in the output

# Compression Input

- + The function receives a binary image (black & white) as input
- + Some of the input images provided to the function include:

**B l a c k**

**BINARY  
IMAGE  
COMPRESSION**

**W H I T E**

# Binarization of the Input Image

- + Although the previous images appear to be strictly black and white, they may contain values close to white, such as 254, 250 or 253 and values close to black such as 1, 2, or 5, which the human eye cannot distinguish from pure white (255) or pure black (0).
- + However, the compression function will not work correctly unless the pixels in the input image have values of exactly 255 and 0.
- + Therefore, it is necessary to use a binarization function that converts a grayscale image into a strictly binary image (white=255, black=0) to ensure that the compression function operates correctly.

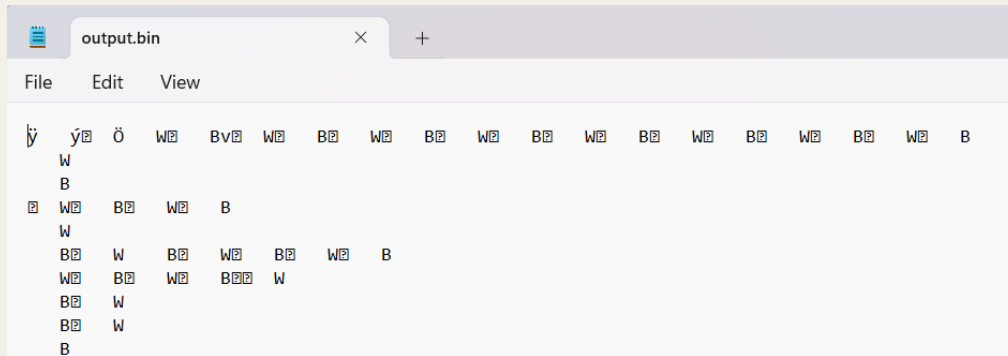


# Compression output

The output is a binary file that contains:

- + Image dimensions (rows and columns) which will help in decompression
- + RLE-encoded pixel data

This file cannot be viewed directly, as the data is stored in binary format.



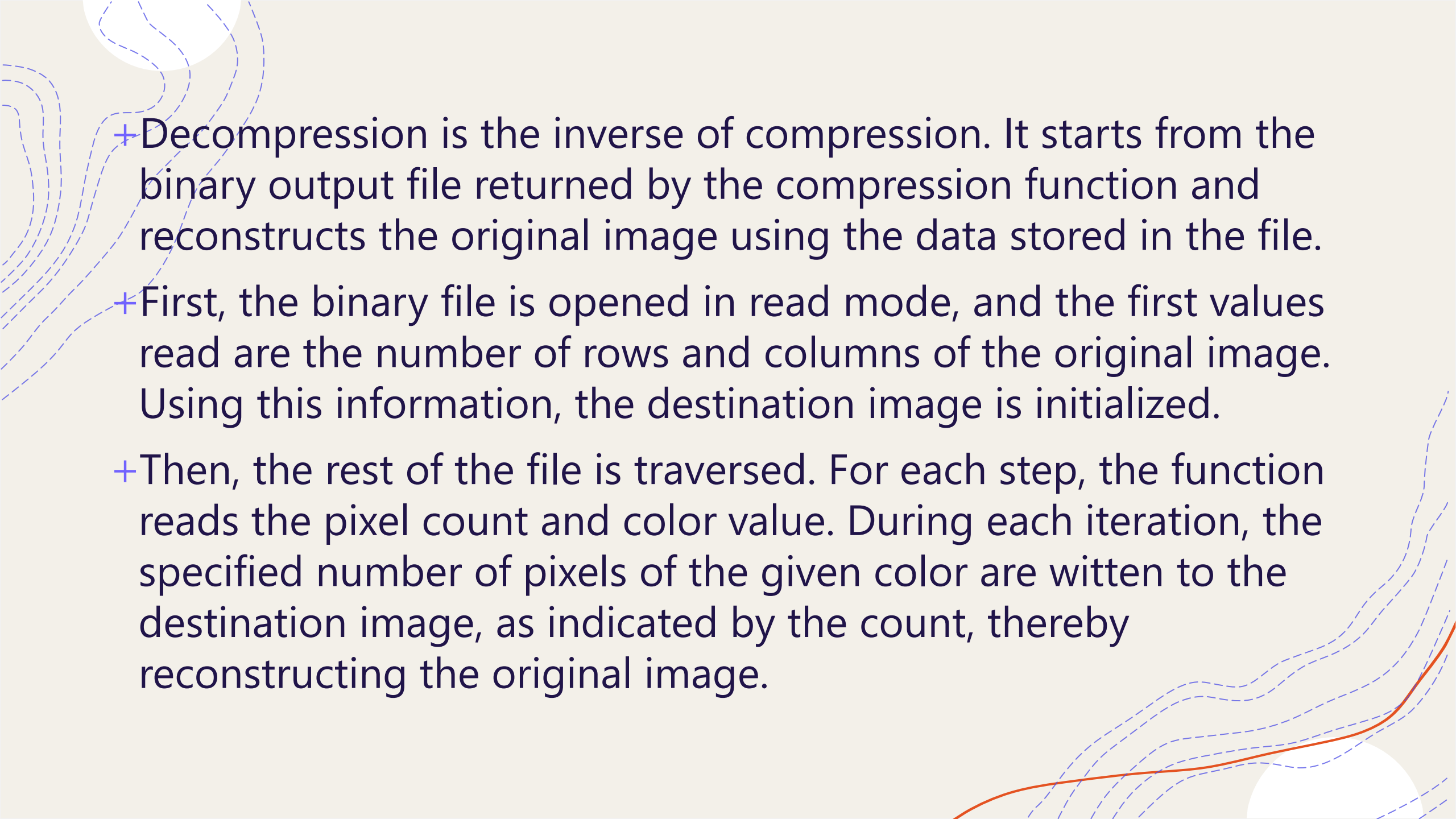
# Decompression Function

```
function decompress()
```

```
+ open file in binary read mode ("Images/output.bin")  
+ rows <- 0  
+ cols <- 0  
+ read value of rows from file  
+ read value of cols from file  
+ dest <- Mat_<uchar>(rows, cols)  
+ color <- 'N'  
+ count <- 0  
+ iterator <- pointer to each pixel in the destination image
```

```
+ while able to read count from file  
+   read color value from file  
+   while count > 0 do  
+     if color is 'W' then  
+       set value of iterator to 255  
+     else if color is 'B' then  
+       set value of iterator to 0  
+     end if  
+     increment iterator  
+     decrement count  
+   end while  
+ end while  
+ close file  
+ display image dest after decompression  
+ end function
```



- 
- + Decompression is the inverse of compression. It starts from the binary output file returned by the compression function and reconstructs the original image using the data stored in the file.
  - + First, the binary file is opened in read mode, and the first values read are the number of rows and columns of the original image. Using this information, the destination image is initialized.
  - + Then, the rest of the file is traversed. For each step, the function reads the pixel count and color value. During each iteration, the specified number of pixels of the given color are written to the destination image, as indicated by the count, thereby reconstructing the original image.

# Decompression I/O

- + **Input:** Binary file created by the compression function
- + **Output:** An image identical to the one that was originally compressed



# Compression Ratio

## **BINARY IMAGE COMPRESSION**

- + The compression ratio represents the ratio between the size of the original image file before compression and the size of the resulting binary file after compression
- + Example: for the image on the left:
  - + Original image file size: 389.694 bytes
  - + Compressed binary file size: 17.273 bytes
  - + Compressed ratio: 22

# Bibliografie

- + [https://en.wikipedia.org/wiki/Run-length\\_encoding](https://en.wikipedia.org/wiki/Run-length_encoding)
- + Source of the input images: manually created in Microsoft Word by the author