**Multimedia project**

**Doctor : Hamada Ibrahim**

**Names of team:**

1. Nada Khalaf
2. Samaa Abd Eltwab

**Content:**

Run length

Huffman

Adaptive Huffman

LWZ

Arithmetic

**Run Length coding:**

Run Length is a type of lossless compression or RLE.

This method is mainly used for reducing the size of image

files, though there are now better, more modern options

available.

**Example:**

BBBBBBBBBYYBBBWBBYYBBWWWBBBBWWWWBB

BWWWWWGGGGGGGGGGGGGGGGGGGGGGGG

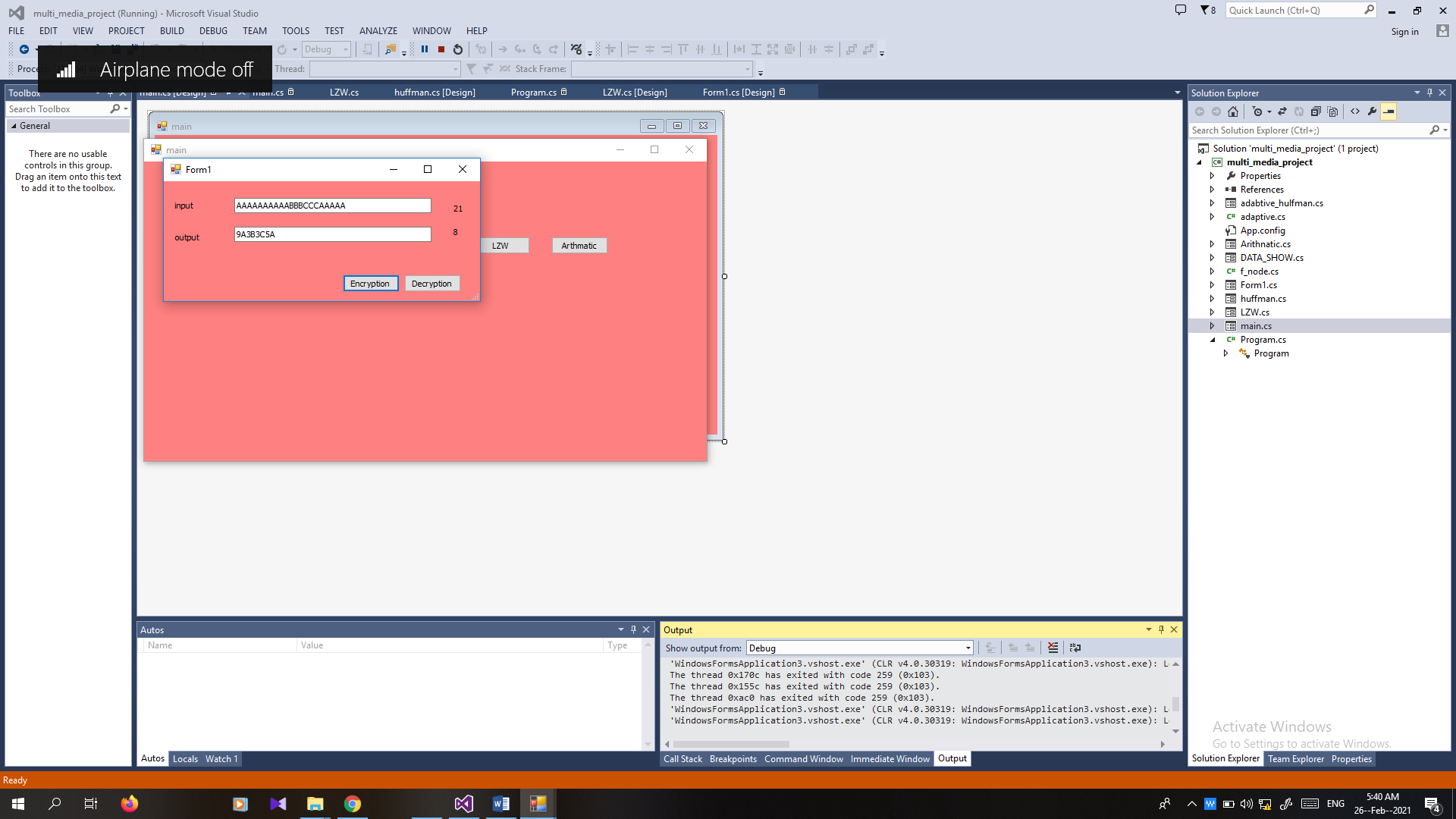
**The answer:**

so how would RLE reduce this file without reducing

the quality of the image?

Count each of the blocks of color and you get:

9B 2Y 3B 1W 2B 2Y 2B 3W 4B 5W 3B 5W 24G

****

**Huffman’s Algorithm:**

1. Create a terminal node for each *ai* *o*, with probability *p*(*ai*)

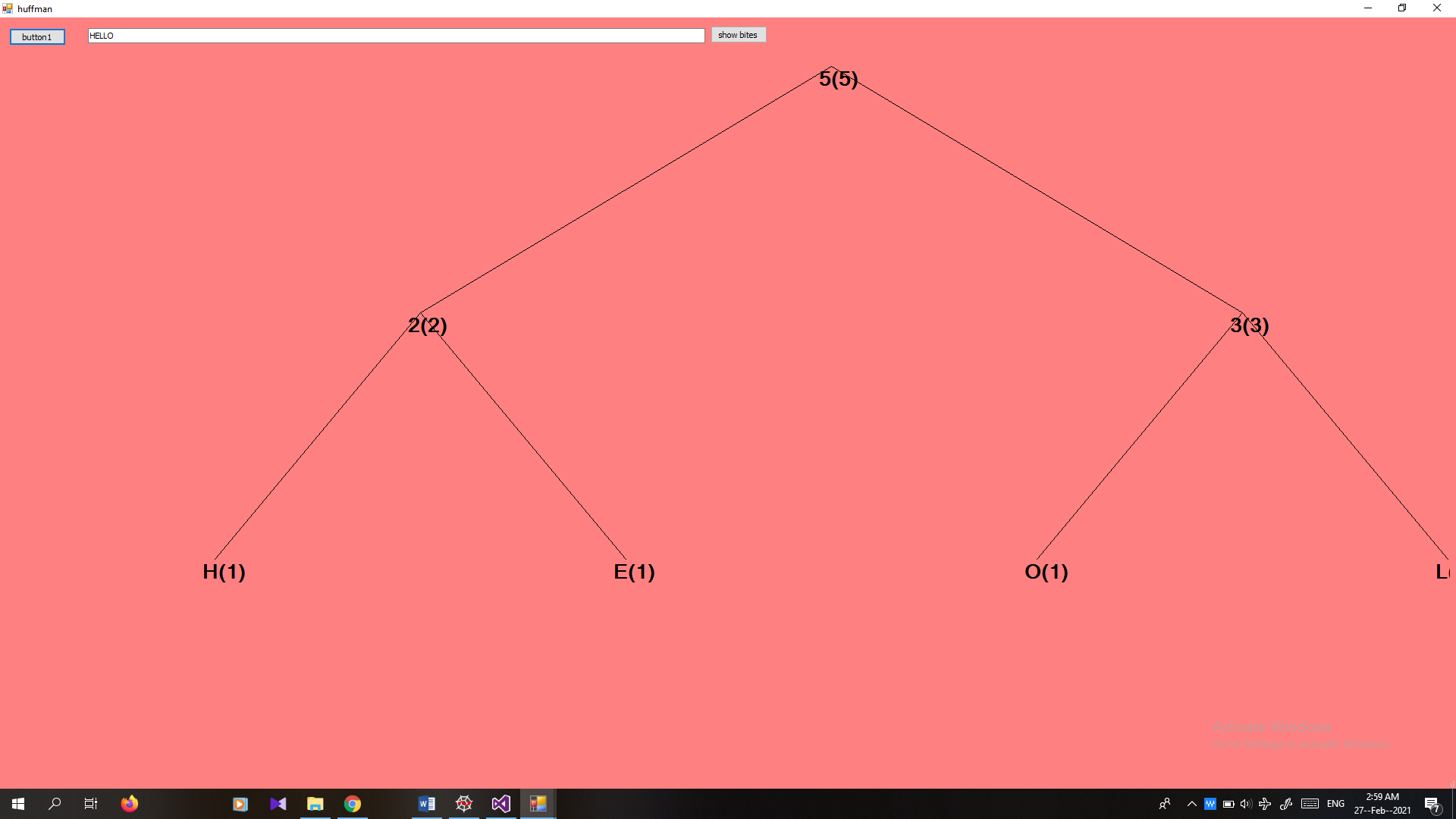
and let *S* = the set of terminal nodes.

2. Select nodes *x* and *y* in *S* with the two smallest probabilities.

3. Replace *x* and *y* in *S* by a node with probability *p*(*x*) + *p*(*y*).

Also, create a node in the tree which is the parent of *x* and *y*.

1. Repeat (2)-(3) until |*S*| = 1.



**Adaptive Huffman :**

The basic Huffman algorithm has been extended, for the following reasons:

(a) The previous algorithms require the statistical knowledge which is often not available (e.g., live audio, video).

(b) Even when it is available, it could be a heavy overhead especially when many tables had to be sent when a non-order0 model is used, i.e. taking into account the impact of the previous symbol to the probability of the current symbol (e.g., "qu" often come together, ...).

The solution is to use adaptive algorithms. As an example, the Adaptive Huffman Coding is examined below. The idea is however applicable to other adaptive compression algorithms.

ENCODER DECODER

------- -------

Initialize\_model(); Initialize\_model();

while ((c = getc (input)) != eof) while ((c = decode (input)) != eof)

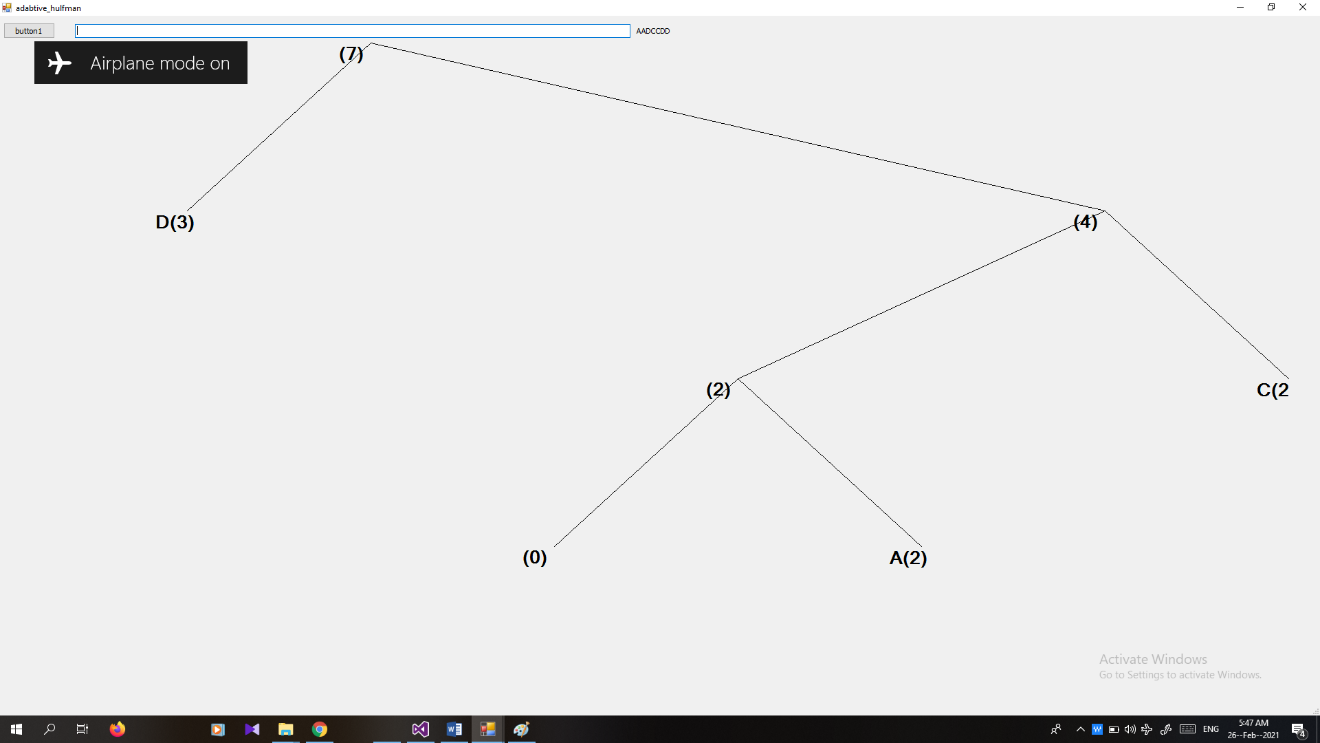
{ {

encode (c, output); putc (c, output);

update\_model (c); update\_model (c);

} }

}



Dictionary based coding (LWZ):

Dictionary-based algorithms do not encode single symbols as variable-length bit strings; they encode variable-length strings of symbols as single tokens

-The tokens form an index into a phrase dictionary

- If the tokens are smaller than the phrases they replace, compression occurs.

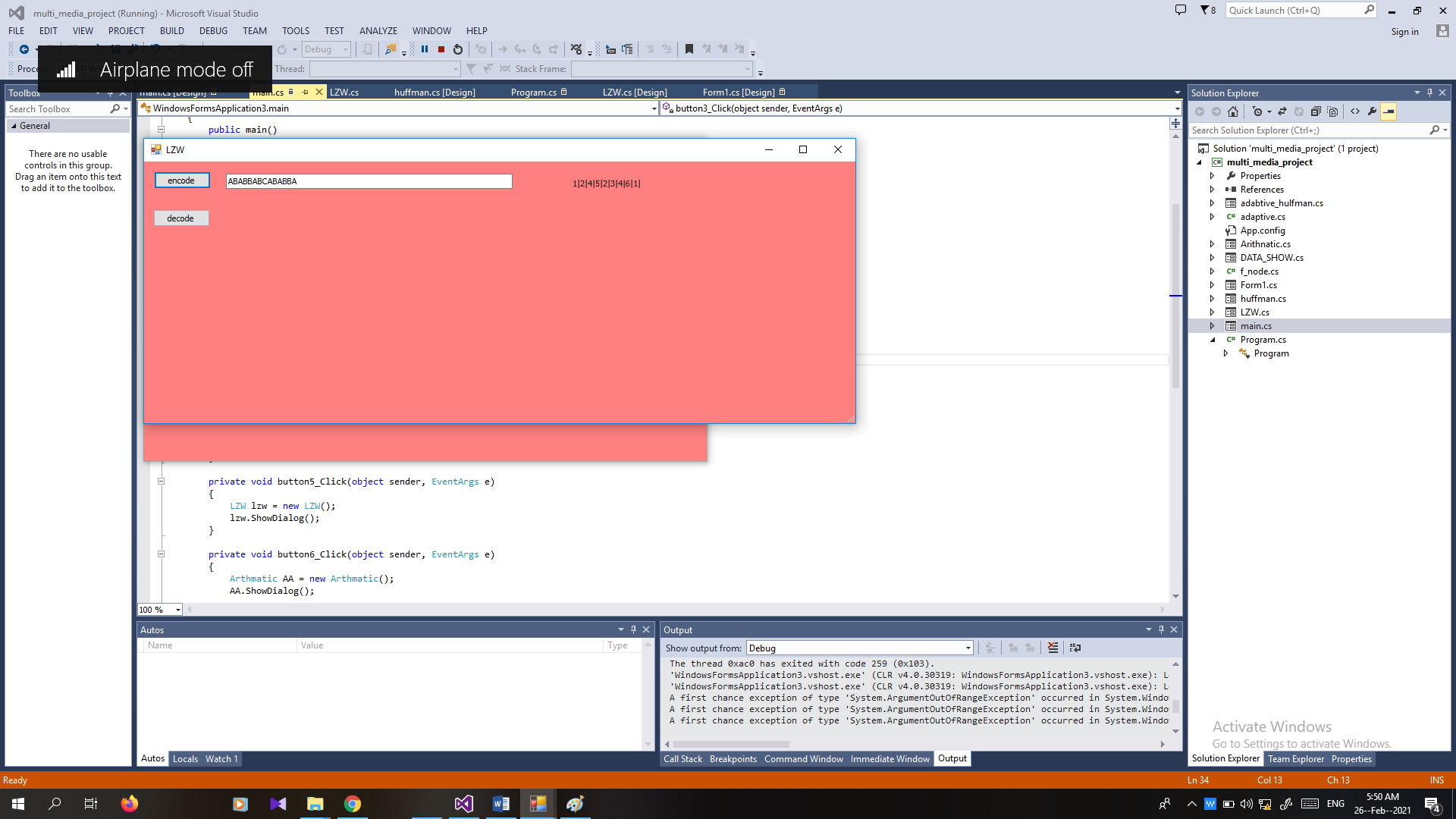
- Dictionary-based compression is easier to understand because it uses a strategy that programmers are familiar with-> using indexes into databases to retrieve information from large amounts of storage.

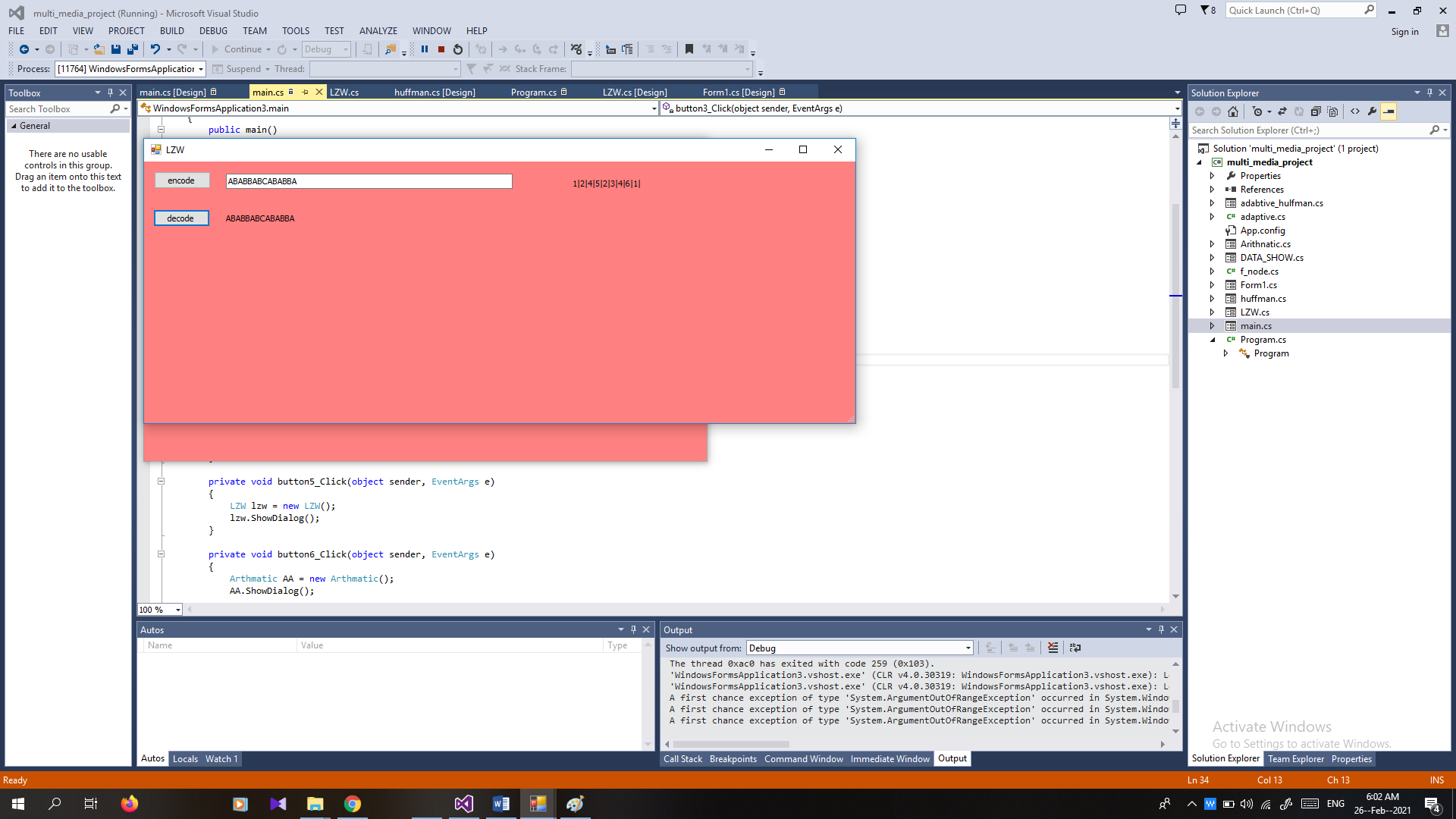
- Telephone number

Example:

“ABABBABCABABBA”

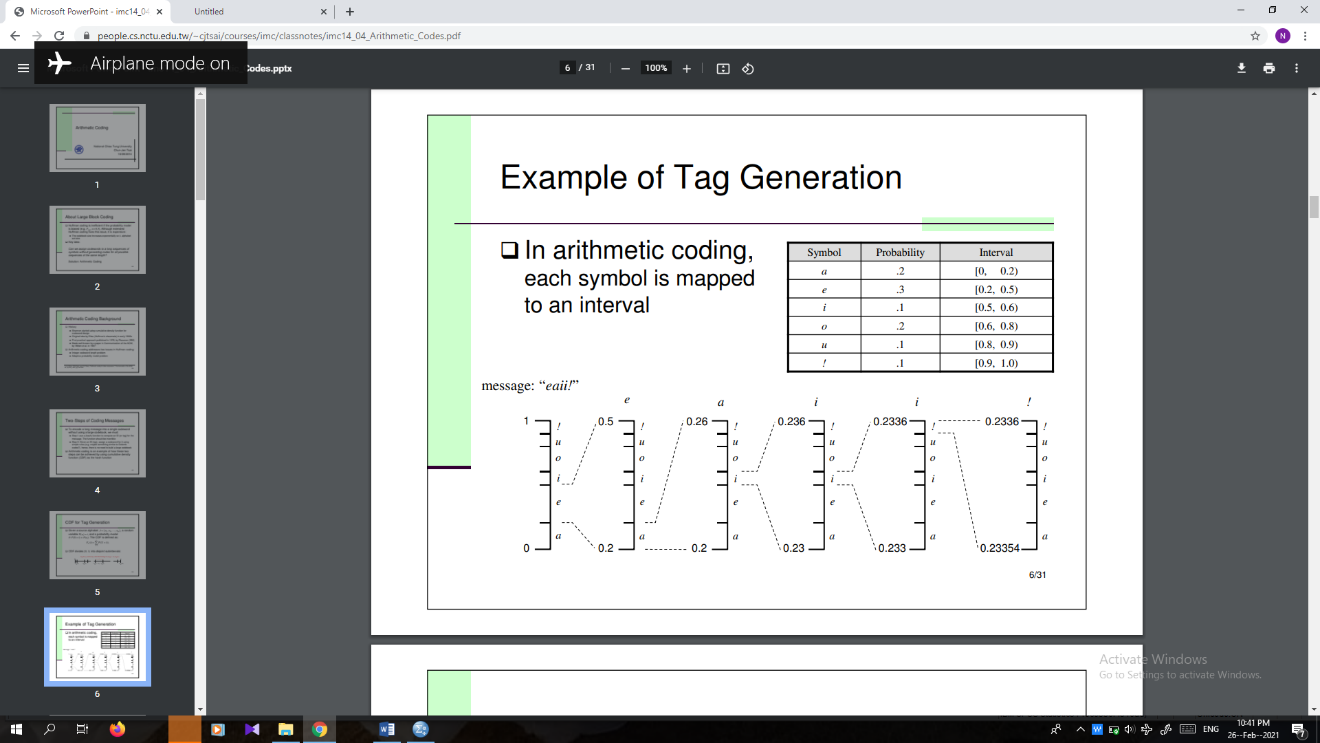
When LWZ applied the answer will be “124523461”



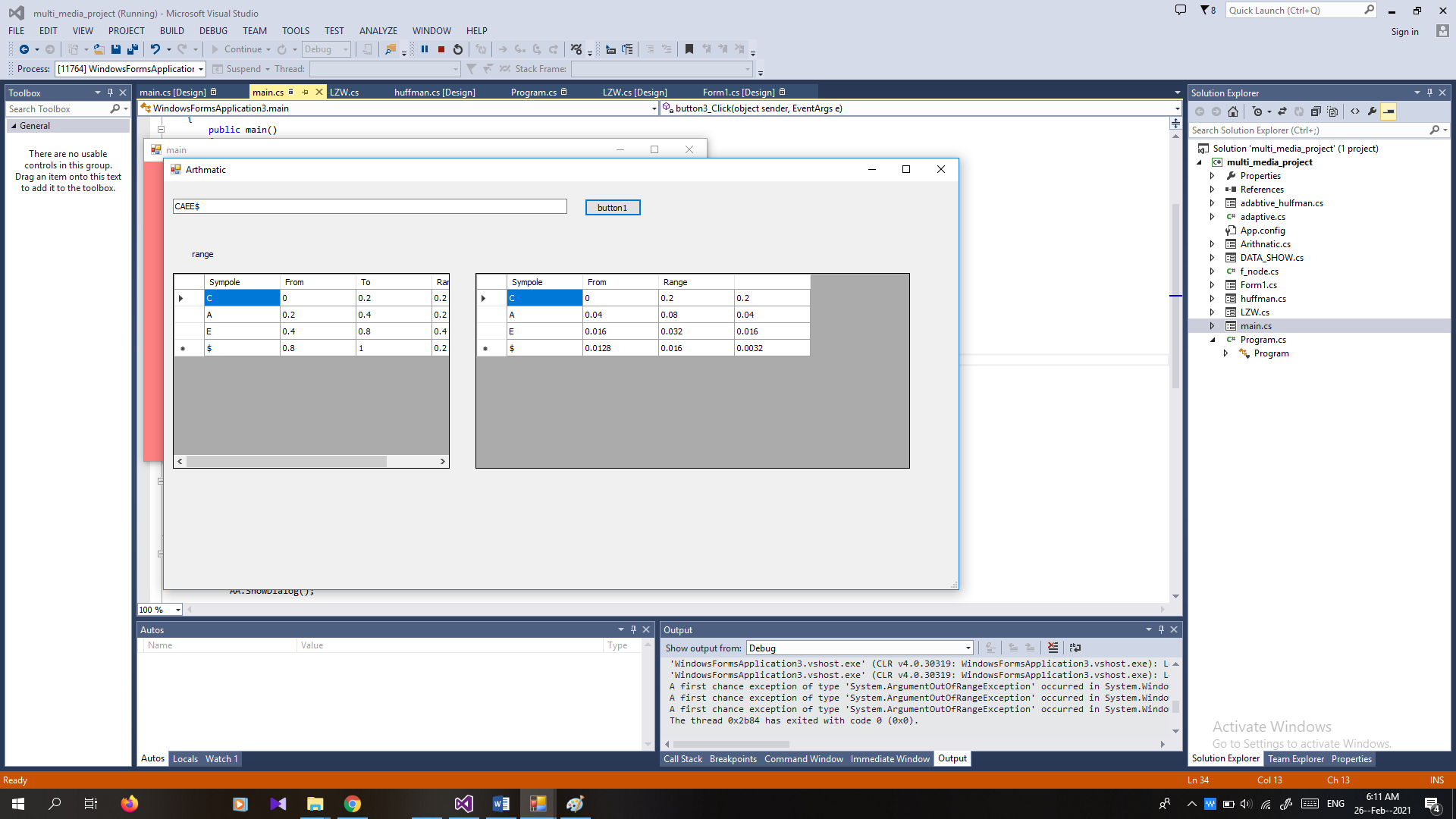
**decode**:

**Arithmetic:**

-To encode a long message into a single codeword without using a large codebook, we must Step I: use a (hash) function to compute an ID (or tag) for the message. The function should be invertible Step II: Given an ID (tag), assign a codeword for it using simple rules (e.g. maybe something similar to Golomb codes?), hence, there is no need to build a large codebook

- Arithmetic coding is an example of how these two steps can be achieved by using cumulative density function (CDF) as the hash function

An example “CAEE$”

****