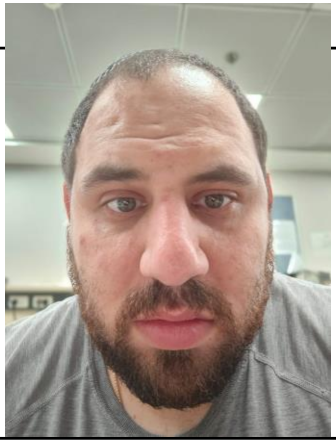
Course: Image Processing 31651
Assignment #21
Pixel to Pixel Operations (Part 1) – version 2

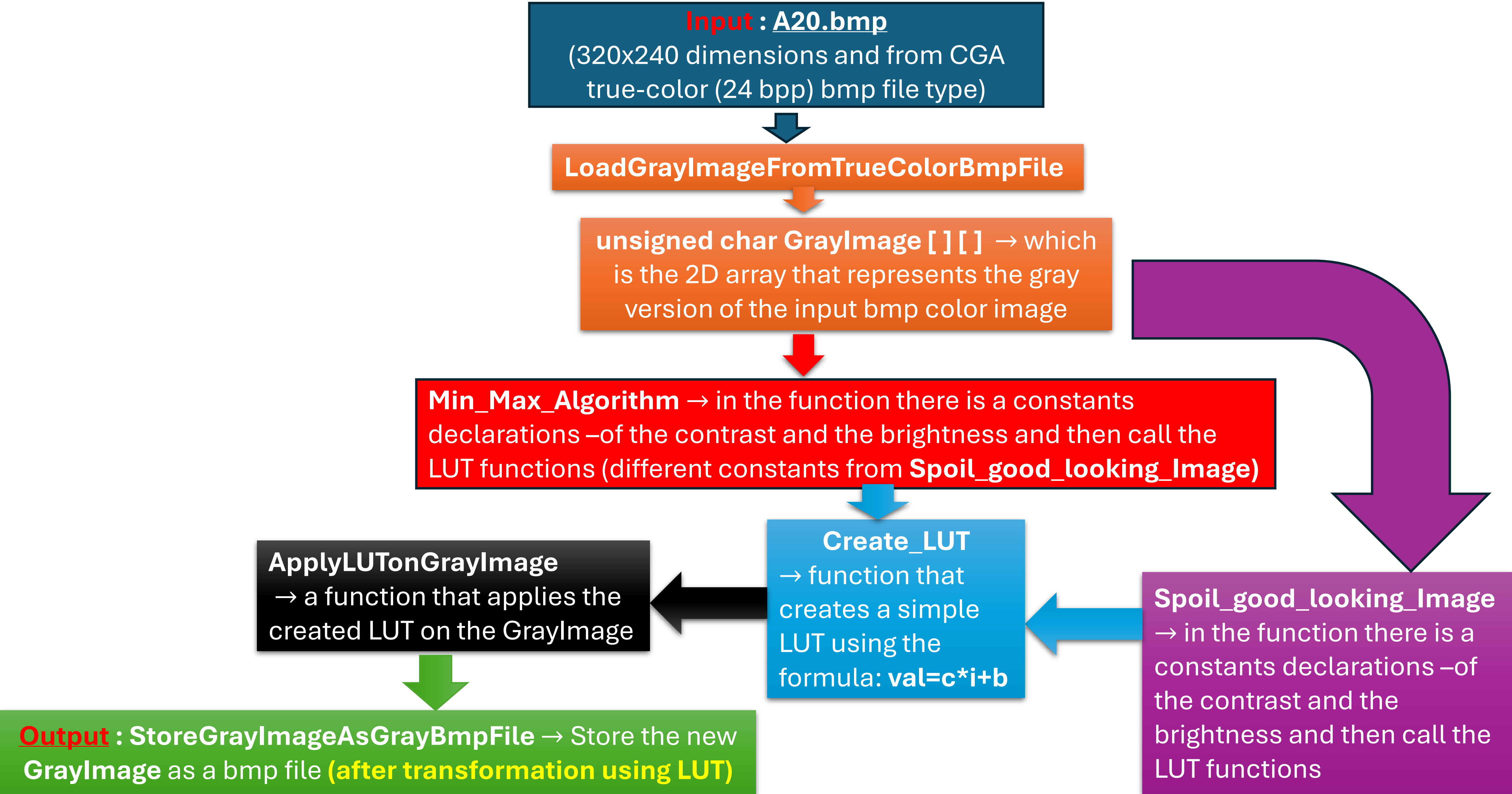
	ID (4 last digits)	Shorten Name	Photo of the student
Student #1	**1950**	shienfeld	
Student #2	**2210**	pony	
Student #3	**7939**	akimov	

Assignment #21: Process set of Gray Images

In the file ImProcInPlainC.h defined numerical values of:
NUMBER_OF_ROWS and NUMBER_OF COLUMNS

#	**Image Description**
1	Find in the Internet Color PHOTO of a set of beautiful fruits and vegetables (at least 6 different types). *(Joke about Vovochka – 5 animals at Africa)* ***Photo must have natural and non-trivial background*** Resize the image as CGA true color (24 bpp) bmp file. A20.bmp Convert color image A20.bmp to gray image A20gray.bmp Spoil "good looking image": Calculate LUT for the **contrast** = 0.22+0.023*B+0.003*C ; **brightness** = (23 + F*3 +G) (B, C ..– from short ID) Apply LUT to A20gray.BMP. **Pointers must be used to access the pixels**. Store processed Image as A211.bmp
2	Apply MIN-MAX algorithm to A211.bmp. **The algorithm must use pointers to access the pixels**. Store processed Image as A212.bmp
	For report use YOUR OWN STYLE. From YOUR Power Point report customer must understand what was done, how and what are the results (Lecturer may later ask to add slides) **Done: 21.1 21.2** **REMOVE ALL RED when the items are ready** **Important hint: Fill report pages while working.** **Do not complain "not enough time" in case you did not follow THIS rule.**

21.0 - flow chart of the code



Input : A20.bmp (320x240 dimensions and from CGA true-color (24 bpp) bmp file type)

LoadGrayImageFromTrueColorBmpFile

unsigned char GrayImage [ ] [ ] → which is the 2D array that represents the gray version of the input bmp color image

Min_Max_Algorithm → in the function there is a constants declarations –of the contrast and the brightness and then call the LUT functions (different constants from **Spoil_good_looking_Image)**

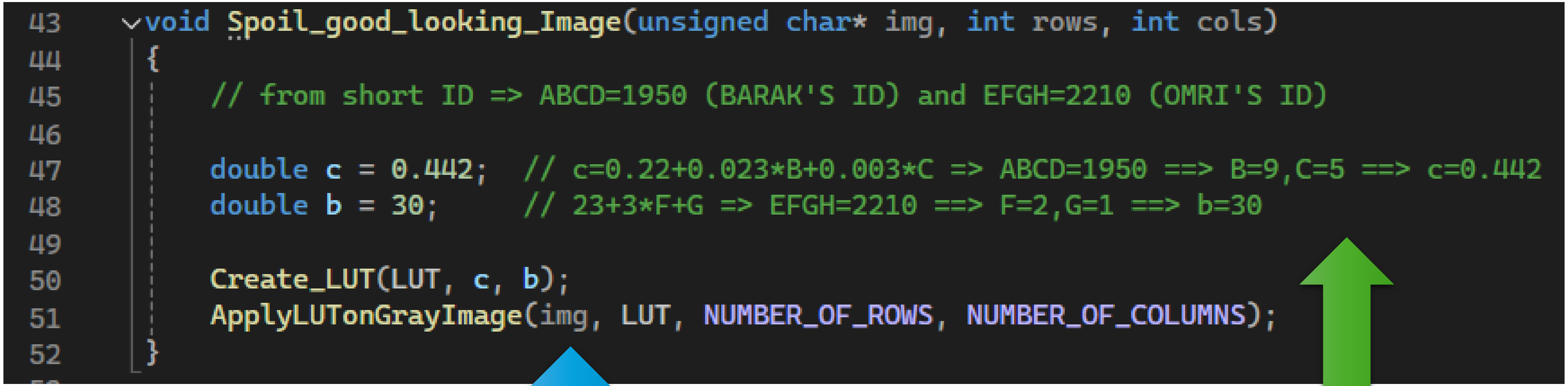
Spoil_good_looking_Image → in the function there is a constants declarations –of the contrast and the brightness and then call the LUT functions

Create_LUT → function that creates a simple LUT using the formula: **val=c*i+b**

ApplyLUTonGrayImage → a function that applies the created LUT on the GrayImage

Output : **StoreGrayImageAsGrayBmpFile** → Store the new **GrayImage** as a bmp file **(after transformation using LUT)**

21.1 – "Spoil_good_looking_Image" function

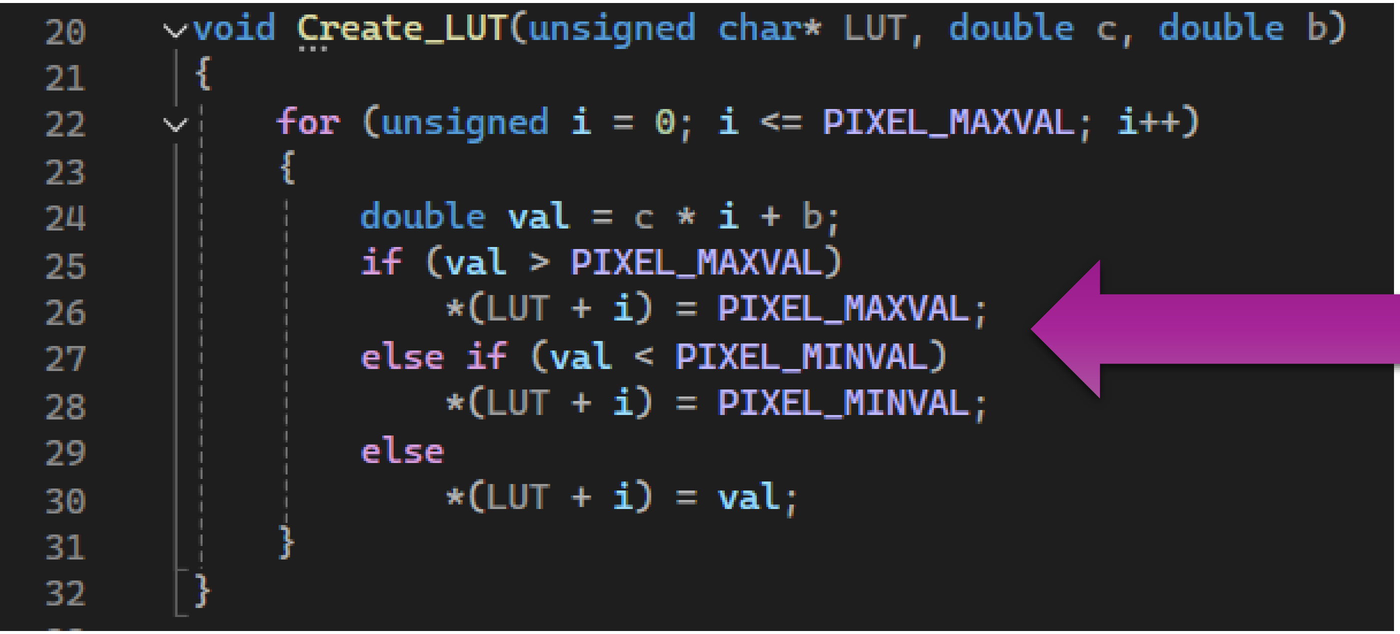


```
void Spoil_good_looking_Image(unsigned char* img, int rows, int cols)
{
    // from short ID => ABCD=1950 (BARAK'S ID) and EFGH=2210 (OMRI'S ID)

    double c = 0.442;  // c=0.22+0.023*B+0.003*C => ABCD=1950 ==> B=9,C=5 ==> c=0.442
    double b = 30;     // 23+3*F+G => EFGH=2210 ==> F=2,G=1 ==> b=30

    Create_LUT(LUT, c, b);
    ApplyLUTonGrayImage(img, LUT, NUMBER_OF_ROWS, NUMBER_OF_COLUMNS);
}
```

Pay attention to 'c' and 'b' values – corresponding the IDs of the group

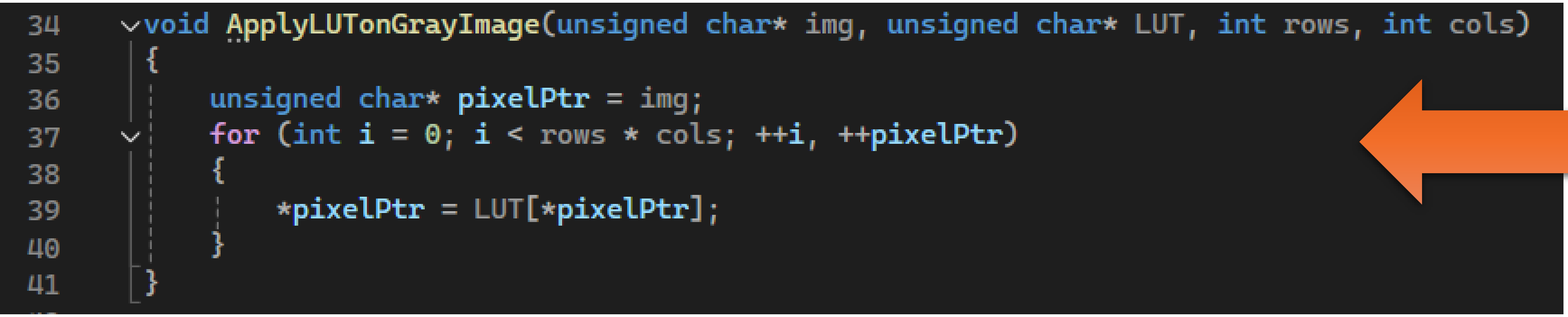
After 'c' and 'b' declarations – create using them a simple LUT and then apply the LUP on the gray image (img [ ] [ ])

21.2 – “Create_LUT” and “ApplyLUTonGrrayImage” functions



```
void Create_LUT(unsigned char* LUT, double c, double b)
{
    for (unsigned i = 0; i <= PIXEL_MAXVAL; i++)
    {
        double val = c * i + b;
        if (val > PIXEL_MAXVAL)
            *(LUT + i) = PIXEL_MAXVAL;
        else if (val < PIXEL_MINVAL)
            *(LUT + i) = PIXEL_MINVAL;
        else
            *(LUT + i) = val;
    }
}
```

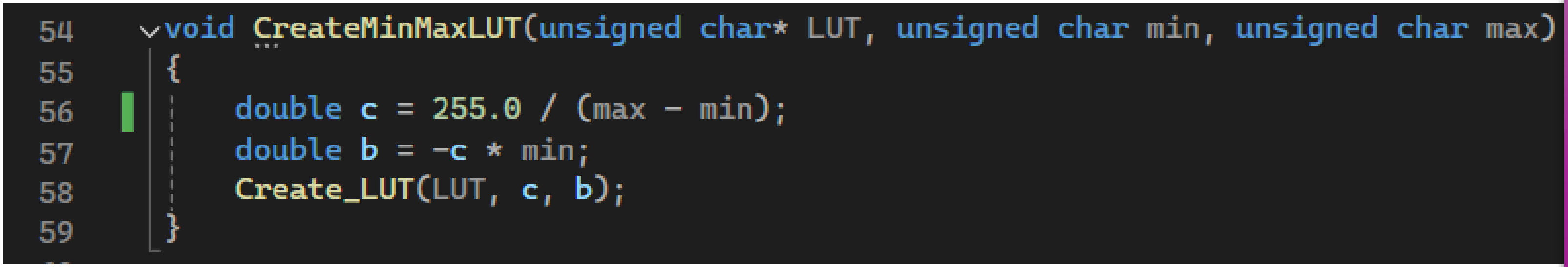
Create_LUT is a very simple function for creating a LUT using the contrast (c) and brightness (b) constants.



```
void ApplyLUTonGrayImage(unsigned char* img, unsigned char* LUT, int rows, int cols)
{
    unsigned char* pixelPtr = img;
    for (int i = 0; i < rows * cols; ++i, ++pixelPtr)
    {
        *pixelPtr = LUT[*pixelPtr];
    }
}
```

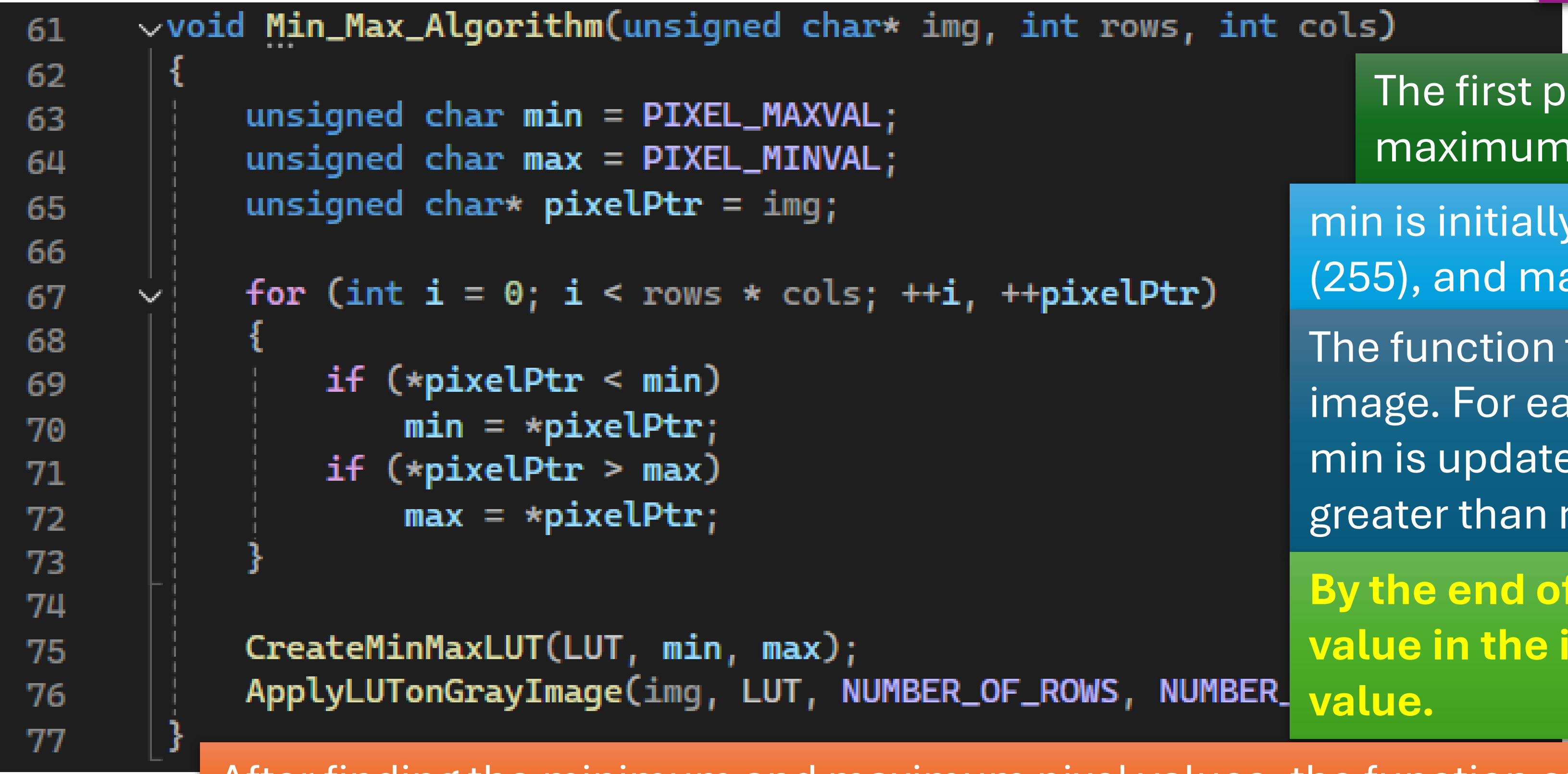
After defining the LUT – we need to apply the LUT of the original gray image (img [ ] [ ])

21.3 – “CreateMinMaxLUT” and “Min_Max_Algorithm” functions



```
void CreateMinMaxLUT(unsigned char* LUT, unsigned char min, unsigned char max)
{
    double c = 255.0 / (max - min);
    double b = -c * min;
    Create_LUT(LUT, c, b);
}
```

Another LUT creation function – now for the min_max_algorithm.
The max value is 255 and the min value is 0 (in grayscale). **Now, the ‘c’ and the ‘b’ are defined differently** then they were defined for the Spoil_good_looking_Image algorithm



```
void Min_Max_Algorithm(unsigned char* img, int rows, int cols)
{
    unsigned char min = PIXEL_MAXVAL;
    unsigned char max = PIXEL_MINVAL;
    unsigned char* pixelPtr = img;

    for (int i = 0; i < rows * cols; ++i, ++pixelPtr)
    {
        if (*pixelPtr < min)
            min = *pixelPtr;
        if (*pixelPtr > max)
            max = *pixelPtr;
    }

    CreateMinMaxLUT(LUT, min, max);
    ApplyLUTonGrayImage(img, LUT, NUMBER_OF_ROWS, NUMBER_
}
```

The first part of the function finds the minimum and maximum pixel values in the image.

min is initially set to the maximum possible pixel value (255), and max is set to the minimum possible pixel value

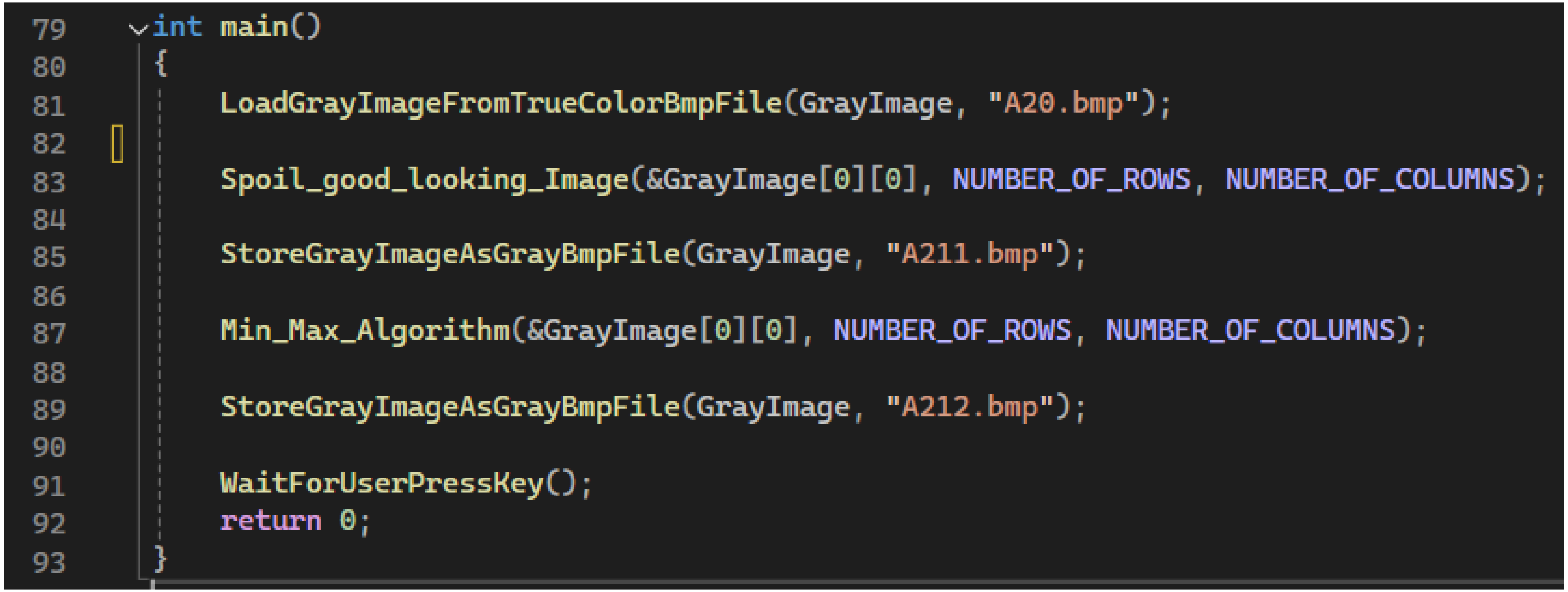
The function then iterates through each pixel in the image. For each pixel : If the pixel value is less than min, min is updated to this pixel value. If the pixel value is greater than max, max is updated to this pixel value.

By the end of these loops, min holds the smallest pixel value in the image, and max holds the largest pixel value.

After finding the minimum and maximum pixel values, the function creates a LUT to normalize the pixel values.

Finally, the LUT is applied to the image to adjust the pixel values.

21.4 – the “main” function

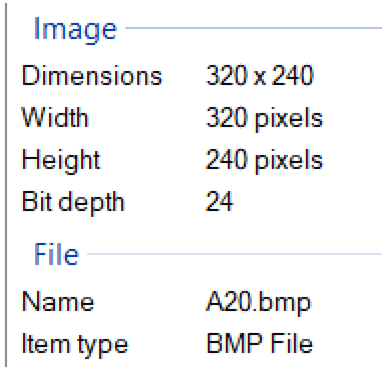


```
int main()
{
    LoadGrayImageFromTrueColorBmpFile(GrayImage, "A20.bmp");

    Spoil_good_looking_Image(&GrayImage[0][0], NUMBER_OF_ROWS, NUMBER_OF_COLUMNS);

    StoreGrayImageAsGrayBmpFile(GrayImage, "A211.bmp");

    Min_Max_Algorithm(&GrayImage[0][0], NUMBER_OF_ROWS, NUMBER_OF_COLUMNS);

    StoreGrayImageAsGrayBmpFile(GrayImage, "A212.bmp");

    WaitForUserPressKey();
    return 0;
}
```

Apply the two algorithms on the A20.bmp original image

21.5 – demonstration using profiler – part 1/5

A20.bmp (original color image)



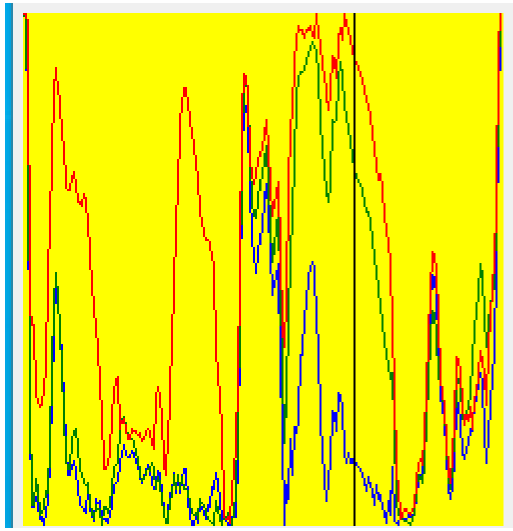
Image

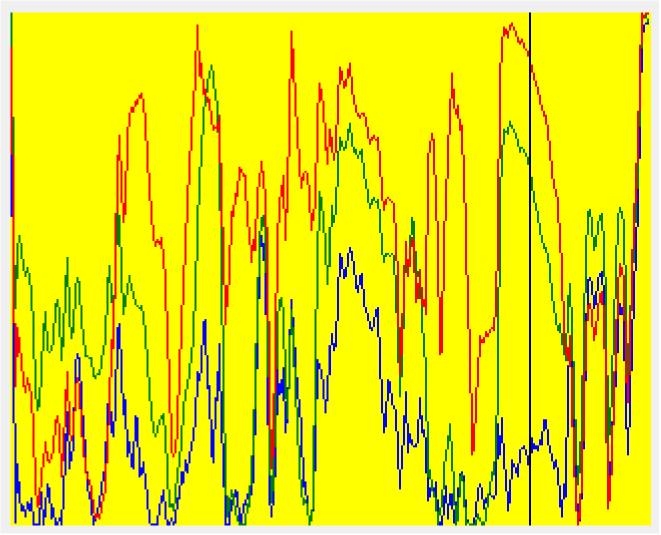
Dimensions	320 x 240
Width	320 pixels
Height	240 pixels
Bit depth	24

File

Name	A20.bmp
Item type	BMP File

we found on the internet a good-looking photograph consisting variety of fruits and vegetables. We resized the image to 320x240 dimensions and made sure it’s from CGA true-color (24 bpp) bmp file type.





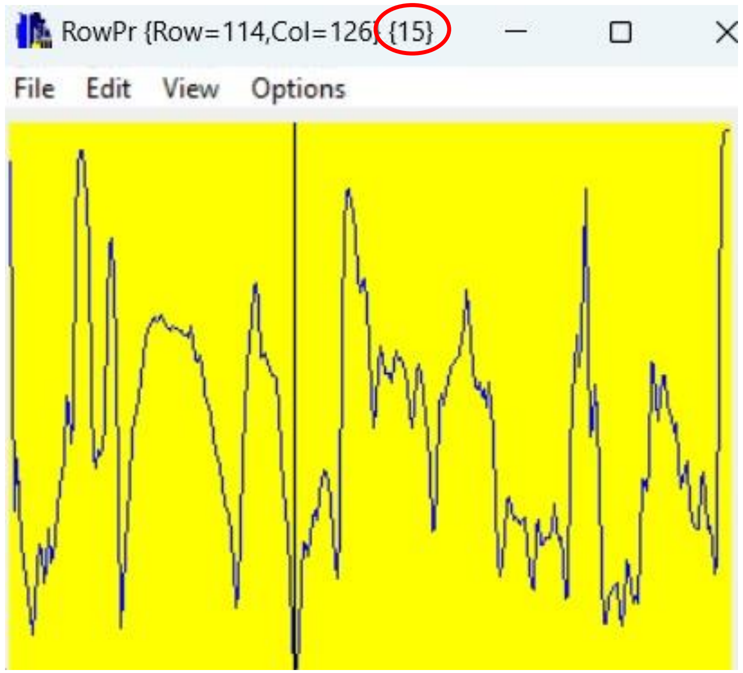
21.6 – demonstration using profiler – part 2/5

A20gray.bmp

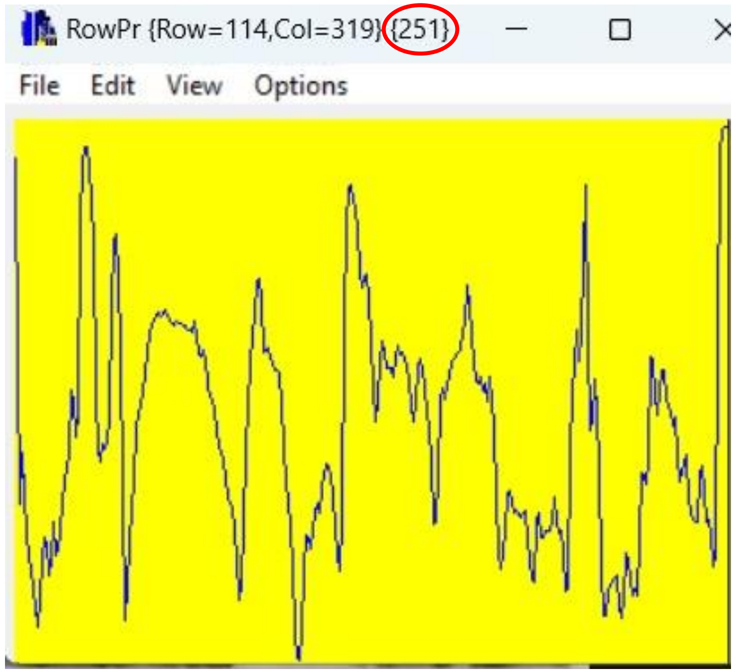
Here we can see the gray version of the original color image we provided.

Important conclusion: **we see that the image consists approximately 92% of the grayscale range (according to the profilers) because highest point is 251 and the lowest is 15.**

After applying the spoiling as well as the min-max algorithm we will examine that point's grayscale and the profilers to see the effects on them.



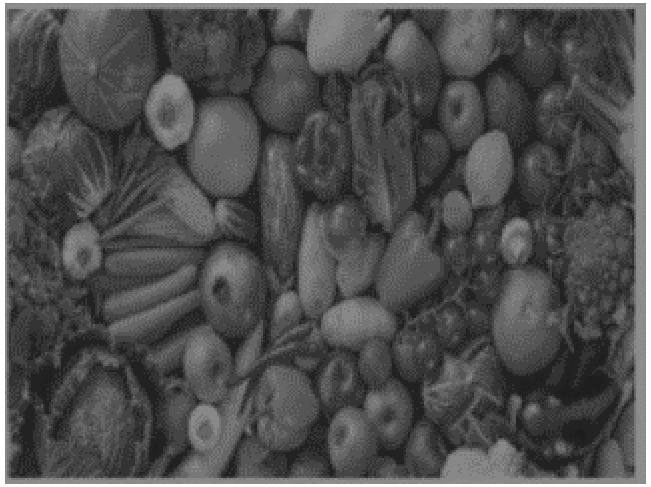
RowPr {Row=114,Col=126} {15}



RowPr {Row=114,Col=319}{251}

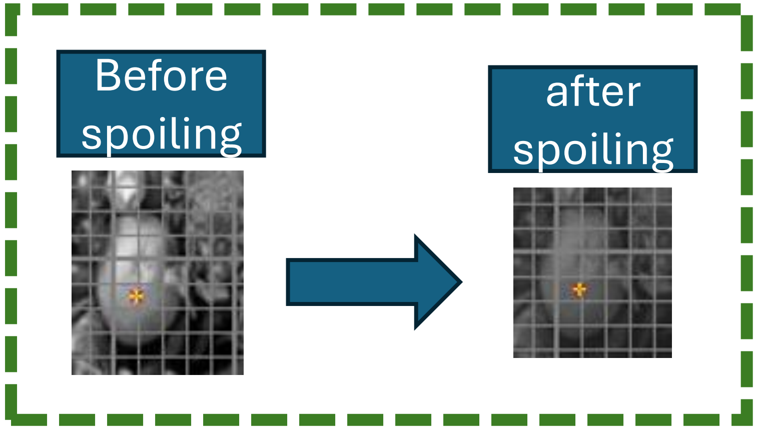
21.8 – demonstration using profiler – part 3/5

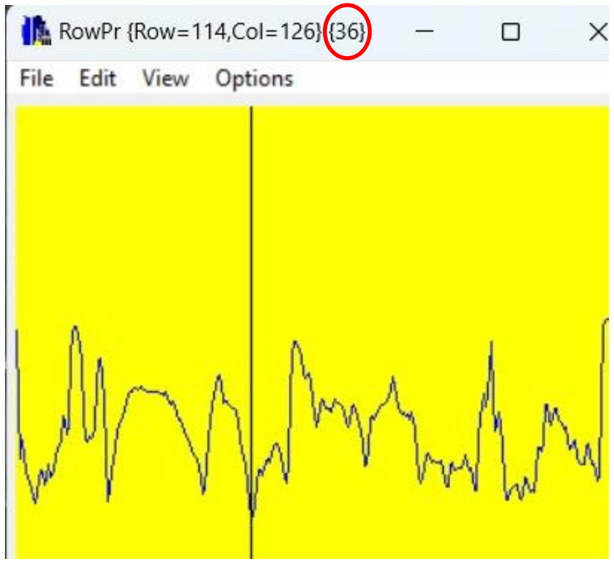
A211.bmp

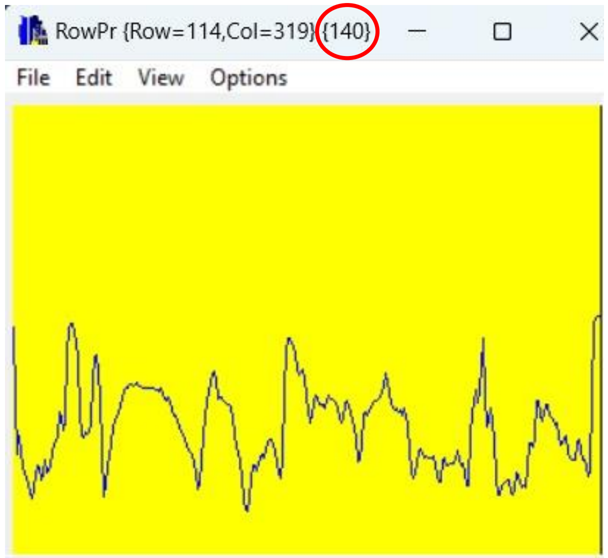


We can see that spoiling the image truly ruins the image. The white parts of the original gray image got ruined and became dark – makes the image almost impossible to work with and to comprehend the true components (in that case – the fruits and vegetables) in the original gray image. (see example in the green dashed rect below)

Important conclusion: **we see that the image consists approximately 40% of the grayscale range (according to the profilers) because highest point is 140 and the lowest is 36. That is a major decrease of the grayscale usage – and the effect is clearly seen – spoiling and ruining the image. (lowering the usage of the grayscale range)**




RowPr {Row=114,Col=126}{36}


RowPr {Row=114,Col=319}{140}

21.7 – demonstration using profiler – part 4/5

A212.bmp

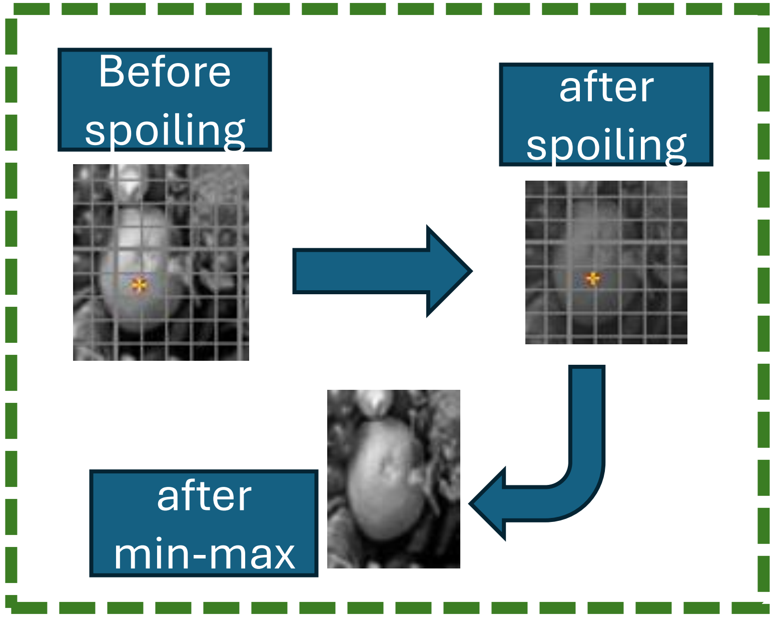
We can see the min-max algorithm result – the original gray image was pretty much restored.

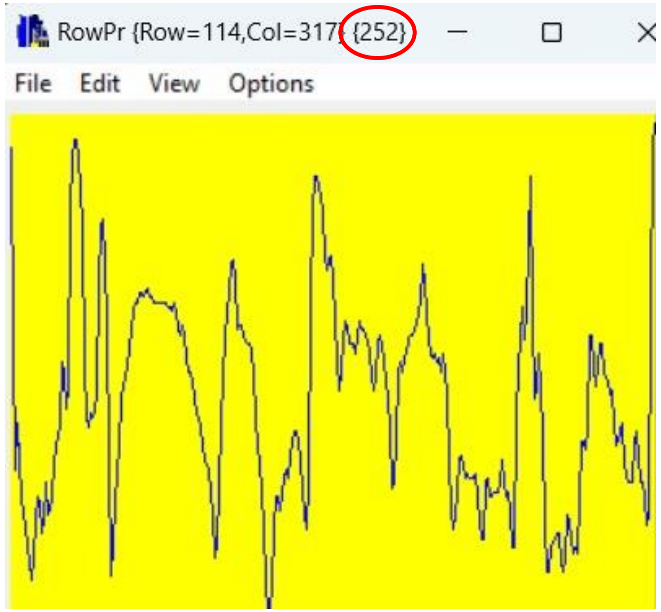
Closer look at the original white components in the image reveal us that the restoration succeeded in that task.

(see example in the green dashed rect below)

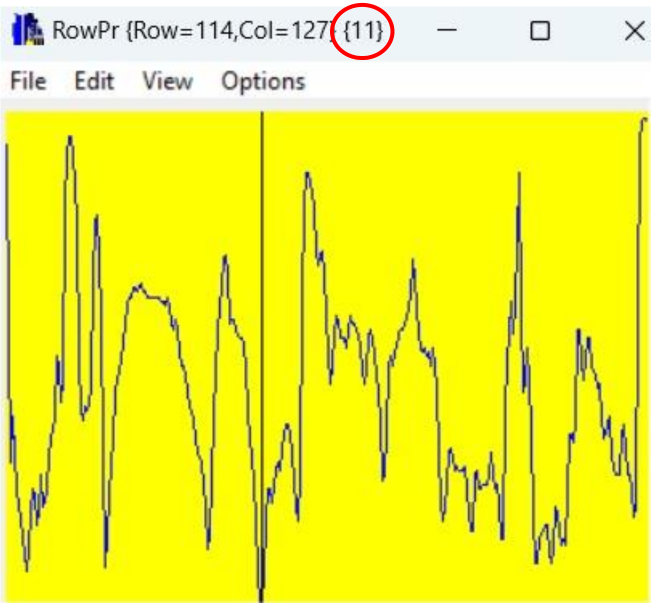
Important conclusion: **we see that the image consists approximately 94% of the grayscale range (according to the profilers) because highest point is 252 and the lowest is 11.**

It means that the usage of the grayscale range for that specific image is even higher than for the original one (94% > 92%)





RowPr {Row=114,Col=317} {252}



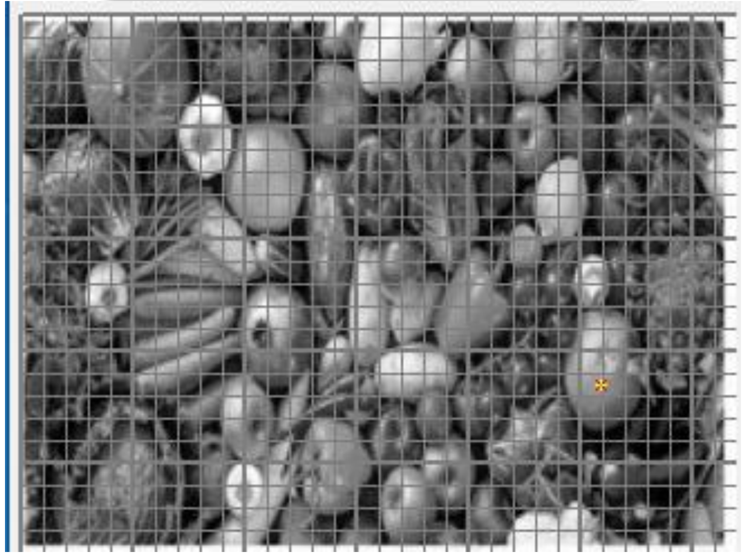
RowPr {Row=114,Col=127} {11}

21.8 –conclusions

1. we can see that the min-max algorithm succeed to fix the spoiling of the gray image.
2. We saw that even major spoiling of the gray image can be restored using min-max algorithm. (in our specific image even **better** range from the original image)
3. We understood that the purpose of the min-max algorithm is to use all the range of the grayscale.

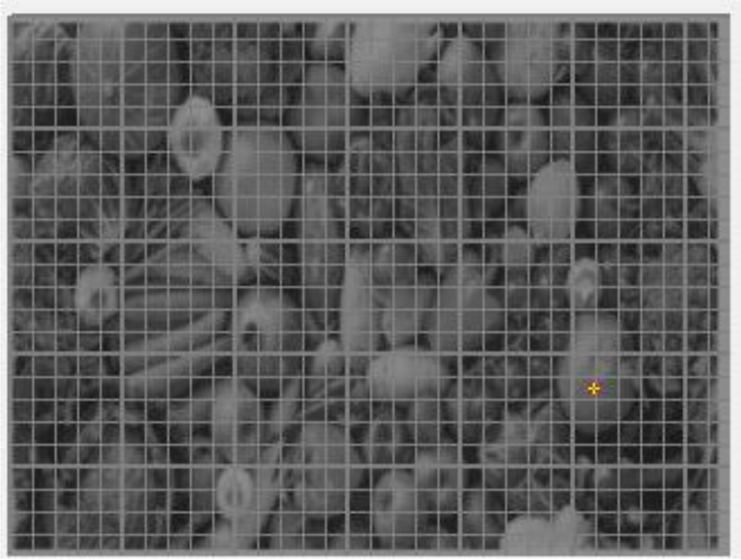
Original gray image

Grayscale range is 92%



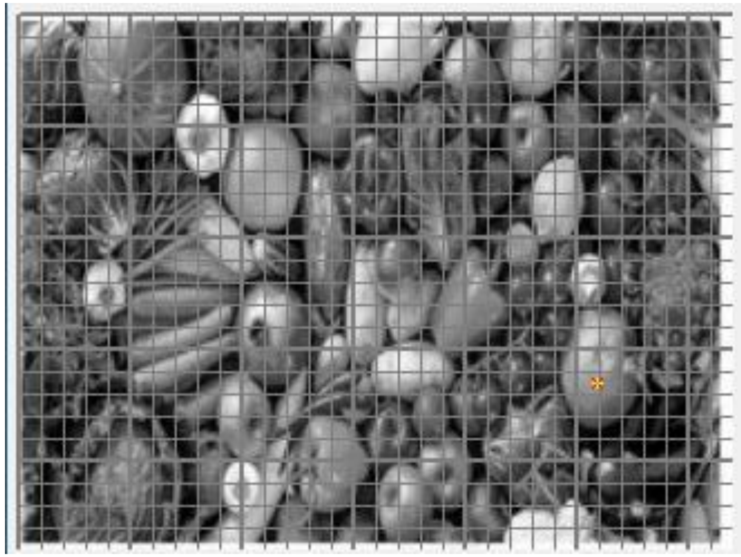
With spoiled_Image algorithm

Grayscale range is 40%



With min_max_algorithm

Grayscale range is 94%



21.9 What did we learned

1) We learned how to implement algorithms and functions that deals with image properties such as: contrast & brightness.

2) We learned how to use the minmax algorithm to “fix” distorted images.

3) We used methods involving LUT which is an essential concept in Image Processing