|  |  |
| --- | --- |

Image Processing Package   
(Assembly Project)  
2017-2018

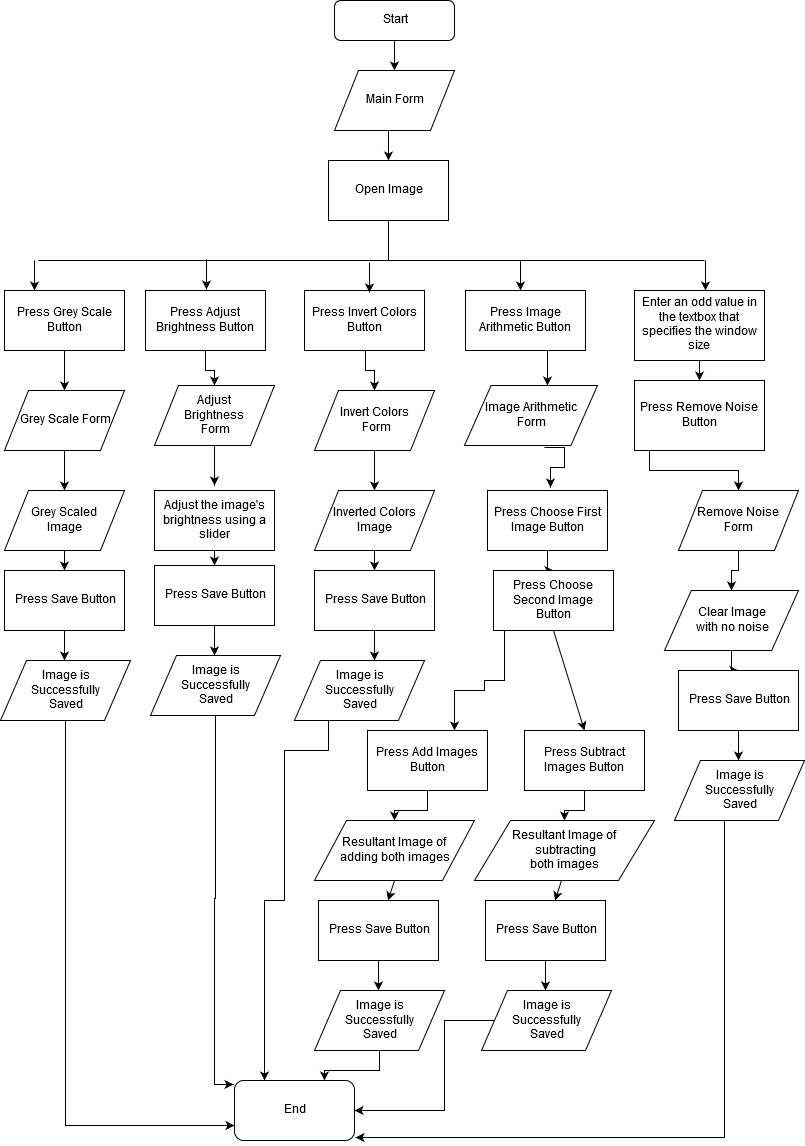
**TABLE OF CONTENTS**

Flow Chart of the whole Project …………….…………………………………….…..… 3

Procedure Hierarchy …………….……………………….……………………………….…..… 4

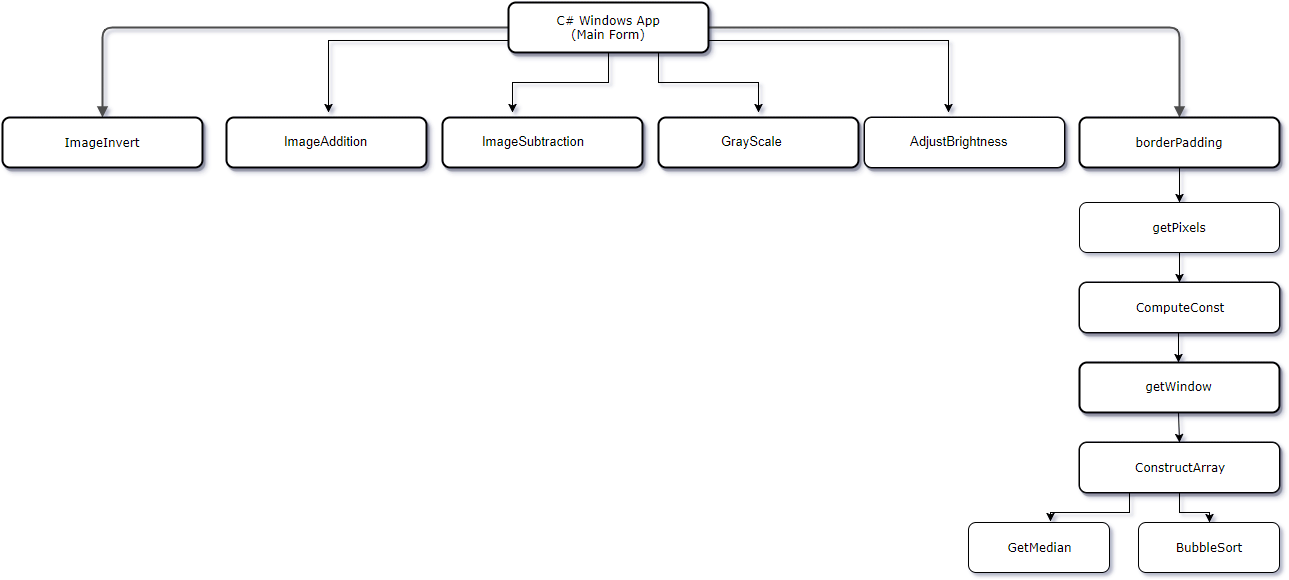
Brief Description for each procedure ……………………………………………. 5

* Brightness Adjustment …………………………………………………..... 5
* Grayscale convertor ……………………………………….……………………..… 7
* Image arithmetic …………………………………………….……………….. 8
* image addition
* image subtraction
* image invert
* Noise removal using median filter .…………………….…………….. 9

****

**Flow Chart**

**Procedure Hierarchy**

****

**Brightness Adjustment:**

This function in the project enables the user to adjust the brightness of any image he selects.

After the conversion of the image to 1D array of pixels and then decomposing its elements into three arrays each has one color component of the RGB image.

The AdjustBrightness procedure receives these three arrays alongside with the arrays’ size (only one variable as the three of them are of equal sizes) and the brightness value inserted by the user using a slider so that it can be added to or subtracted from the elements of the three arrays to increase or decrease the brightness of the image.

This procedure uses four registers EAX, EBX, ECX and ESI and five variables brightnessValue\_RA, arrSize\_RA, red\_RA, blue\_RA and green\_RA.

The EAX register is used to hold the brightness value (brightnessValue\_RA) which will be added to or subtracted from the elements of the three arrays to increase or decrease the brightness of the image.

The EBX register is used to hold the value of each component of each array in its specified loop as each array is modified in a separate loop.

The ECX register is used to hold the array size to control the loops.

The ESI register is used to hold the OFFSET of each array (initialized before each loop by red\_RA, blue\_RA and green\_RA).

The red\_RA is a variable of type PTR DWORD as it is a pointer (holds the offset) to the array of the red components of the pixels.

The blue\_RA is a variable of type PTR DWORD as it is a pointer (holds the offset) to the array of the blue components of the pixels.

The green\_RA is a variable of type PTR DWORD as it is a pointer (holds the offset) to the array of the green components of the pixels.

It is also important to notice that each of the three arrays that holds a component of all the pixels of the RGB image are of type int so that the red\_RA, blue\_RA and the green\_RA variables are of type PTR DWORD.

The function starts with initializing the EAX register with the brightness value sent to the procedure as a parameter and then continues its way to adjust the red color first.

The ESI register is initialized by the red\_RA variable and then the EBX register is initialized with zero to remove any previous value in the register and the ECX is initialized with the array size to enter the loop to firstly move the first element of the array using indirect memory addressing by dereferencing the ESI register to EBX.

Then the EAX register is added to the EBX register and then the value contained inside the EBX register is compared to 255 (the maximum intensity that can be assigned to a color), if the value is greater than 255 then this value should be modified to be set to 255 and if the value is smaller than 255 it should be tested and compared against 0 to make sure that the value is valid (The valid value must be between 0 and 255 inclusive).

The value contained in the EBX, then, should be compared against 0, if the value is greater than 0 then this value is valid, and it should be left as it is and continue the loop safely to the next element in the array.

If the value is smaller than 0 then this value should be modified and assigned to 0 (the minimum intensity that can be assigned to a color) and then continue the loop safely to the next element in the array.

This process is the same for the arrays that hold the green and the blue components of all the pixels of the RGB image.

**Grayscale convertor:**

This function in the project enables the user to convert the image to its gray scale.

When converting an RGB image to grayscale, we have to take the RGB values for each pixel so, first we convert the image to 1D array of pixels and then decomposing its elements into three arrays each has one color component of the RGB image after that we make as output a single value reflecting the brightness of that pixel. One such approach is to take the average of the contribution from each channel:

**(R+B+C)/3.**

**Procedure Parameters and Returns considered on the commented block which is before it In the code.**

**Image arithmetic:**

**1.image addition**

this procedure takes as input two identically sized images and produces as output a third image of the same size as the first two, in which each pixel value is the sum of the values of the corresponding pixel from each of the two input images.

If the image format being used only supports, say 8-bit integer pixel values, then it is very easy for the result of the addition to be greater than the maximum allowed pixel value. The overflowing pixel values might just be set to the maximum allowed value, an effect known as **SATURATION**.

**2.image subtraction**

this procedure takes as input two identically sized images and produces as output a third image of the same size as the first two whose pixel values are simply those of the first image minus the corresponding pixel values from the second image.

The same as Image Addition the result of Image Subtraction may be less than Zero then often such pixels are just set to zero (*i.e.* black typically) according to the SATURATION method, or If the operator calculates absolute differences and the two input images use the same pixel value type, then it is impossible for the output pixel values to be outside the range that may be represented by the input pixel type and so this problem does not arise. This is one good reason for using absolute differences.

**3.image invert**

This procedure takes as input an Image and produces its photographic negative dark areas, in the input image become light and light areas become dark.

The resulting value for each pixel is the input value subtracted from 255