
AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF THE UGC ACT, 1956

# PHOTO EDITOR

## 22AIE101

## PROBLEM SOLVING AND C PROGRAMMING

## END SEMESTER PROJECT

**PRESENTED BY**

1. Arya Palackal Shijish [CB.EN.U4AIE22008]
2. Ganesh Sundhar S [CB.EN.U4AIE22017]
3. Praneeth T V S [CB.EN.U4AIE22054]
4. Subhashini Sudhakar [CB.EN.U4AIE22061]

CLASS : Batch A

DEPARTMENT : Centre for excellence in Computational Engineering and Networking

INSTITUTION : Amrita Vishwa Vidhyapeetham, Coimbatore

# ACKNOWLEDGEMENT

This project has been possible due to the sincere and dedicated efforts of many. First of all, we would like to thank the dean of our college for giving us the opportunity to get involved in a project and express our skills. We thank our C Programming teacher, Dr.Aswathy, for her guidance and support without which this project would have been impossible. Last but not least, we thank our parents and our classmates who encouraged us throughout the project.

# ABSTRACT

This is a Report on the Project made for the subject “Problem Solving and C Programming” on the topic “Photo Editor”. The main agenda of this project is to create a basic photo editor app within the Visual Studio(VS) code platform and run it as an executable(.exe) file. The salient features available in our Photo Editor app are :

- Cropping an Image
- Extract the RGB Components of the Image
- Convert the Image to a Grayscale image
- Convert the image to Black and White Image
- Convert the image to Sepia colour format
- Interchange the components of the image

In this project, all the images that we are going to convert can be in any format. But we should specify the name along with the format in the code for this to work. For example, if we have an image named “Mr.Bean” in the jpg format, we should specify the name as “Mr.Bean.jpg”. In this particular project, we are using <stdio.h> and <stdlib.h> standard libraries along with some header source files available on the internet to import and export images to C.

The <stdio.h> library is used for functions like ‘printf’ and ‘scanf’ while the <stdlib.h> library is used for functions like ‘malloc’, etc. The ‘printf’ function is used for printing our desired texts into the output window. The ‘scanf’ function is used to obtain the variables from the user. The ‘malloc’ function is used for allocating the amount of memory specified inside the parenthesis. The ‘malloc’ function also provides us with a pointer that contains the address of the memory location allocated by it.

The various header files used are listed below :

- stb_image
- stb_image_write
- stb_image_resize

The ‘stb_image’ header file is used for importing images while the ‘stb_image_write’ header file is used for exporting the images.

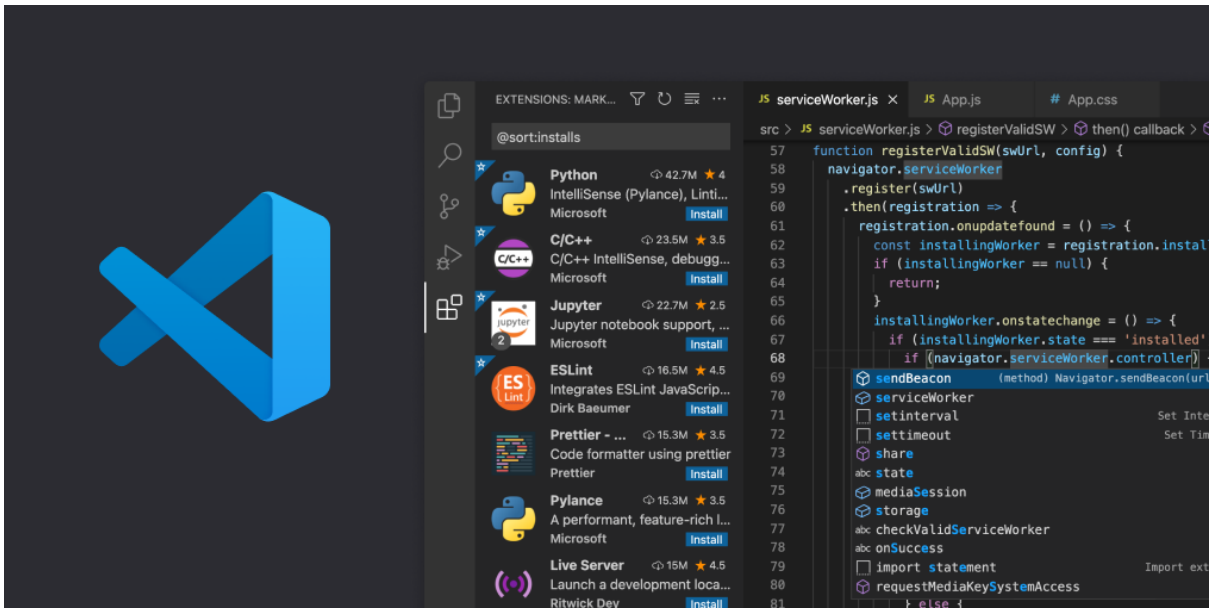
All the image processing calculations done here are based on single-dimensional arrays and not on multidimensional arrays. The stb_image_write header file has functions to convert the single-dimensional matrix to the respective image with the help of image size and the number of channels. The various cases of image conversions specified can be handled using a ‘switch case’ flow control statement available in C Language.

Finally, the executable(.exe) file can be created by using the ‘Run Build Task’ icon present under the ‘Terminal’ tab of VS Code platform. This will provide us with an executable file with the Application type that can convert an image to our desired image.

# CHAPTER 1: INTRODUCTION

The C Language is one of the most primitive and low-level programming languages that are available today with the lowest level being the machine language. Higher-level programming languages are built based on the C Language. The C Language has its own compiler which compiles the code into machine language which further converts it into binary code for the computer to understand.

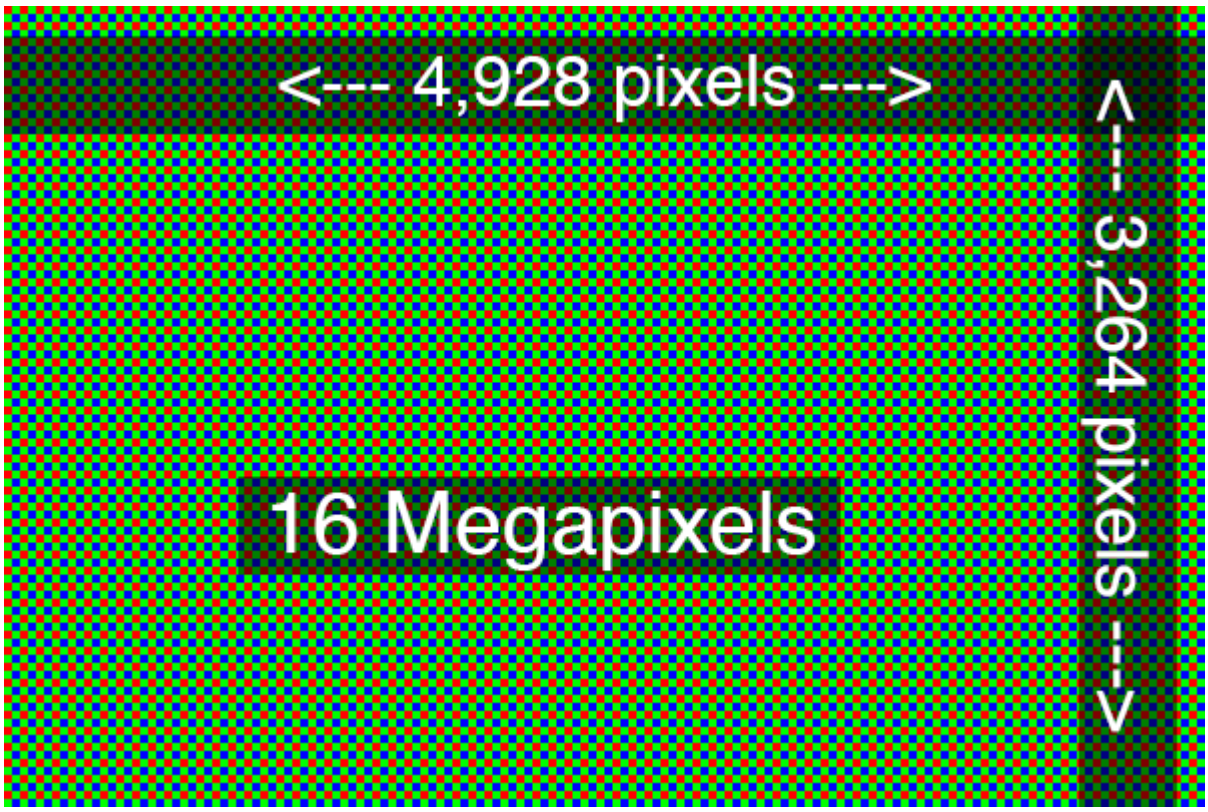
There are various C Compilers and C programming platforms available on the internet. The particular compiler used in this project is the MinGW Compiler and this project is made in the visual studio(VS) code platform. The VS Code platform supports a lot of programming languages like C, MATLAB, java, python, and html.




We are going to use this platform to type our code and compile them into an executable(.exe) file. This will provide us with a simple photo editor application. We can use this to edit images to our desired formats from the list available.

### *What is an Image ?*

An image is an organized collection of pixels. It can be thought of as a 2-dimensional(2D) array of pixels. An image can be either a colour image or a Grayscale image. Each pixel represents a particular colour. It is the combination of many small pixels that provides us with the image. We may have seen the quality of the cameras of our smartphones mentioned as 64MP or 108MP. What this MP specifies is the number of pixels. 1MP stands for 1 Million Pixels.




The above figure shows us the pixel count of a 16MP image.

### *What is the difference between a Colour image and a grayscale image?*

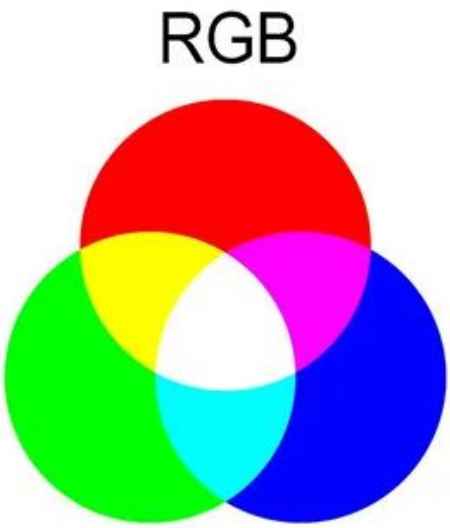
The major difference between a colour image and a grayscale image is that a colour image has 3 components ,i.e. Red, Green, and Blue while the grayscale image has only one component ,i.e. light intensity.

If the light intensity is a grayscale image is 0%, it means there is no light. So, it will be dark and the colour will be black. When the light intensity is 100%, it means there is very bright light. So, the colour will be white. The intensities are represented in a scale from 0-255. As there are 256 values totally, it would take 8 bits to store these values entirely. So, each pixel in a grayscale image occupies 8 bits of memory and is in the form of unsigned integer.

In a colour image, the principle is the same. But, instead of White light, we will use 3 Colour lights ,i.e. Red, Green and Blue(RGB). This shows us that each pixel has 3 different light intensities of the RGB Colours. So, if all three are 0 then there is no light and the colour is black. Here, each colour intensity is represented as 8 bit unsigned integers because each can have a value ranging from 0-255. So, the total space occupied by each pixel of a colour image is 24 bits(8 x 3 = 24).

## *What are channels in an image ?*

Channels are the various values that are used to represent each pixel of the image. According to our previous observations a grayscale image may have 1 channel and a colour image may have 3 channels. But this is not always true. A grayscale image may have a 2nd channel and a colour image may have a 4th channel. So what do these extra channels represent? This represents the opacity or transparency of each pixel of the image.




## *Black and white Image*

A Black and white image is an image which contains only black and white pixels. It has no other type of pixels which are intermediate in colour.

### *Sepia Toning*

The sepia toning changes the appearance of the image by changing any image to a brown toned image. This is mostly used for antique photography purposes.

### *RGB Components*

This particular filter will separate out the RGB Components of a colour image.  If it separates the Red component, it means that the value of Green and Blue component is set to 0.

# CHAPTER 2 : METHADOLOGY

## 2.1 LIBRARIES USED

The various standard libraries used in this project are as follow :

- ✓ <stdio.h>
- ✓ <stdlib.h>

```
#include <stdio.h>
#include <stdlib.h>
```

The <stdio.h> library is used for the following commands :

**1) Printf :**

The printf() function is used to print our desired text into the output window from our main function. The main function is the basic function used in all the C programs. Only those codes which are written inside the main function runs. This is the syntax of C Language.

- ➢ Code :

```
printf("Loaded image with a width of %dpx, a height of %dpx and %d channels\n", width, height, channels);
```

- ➢ Output :

```
Loaded image with a width of 1280px, a height of 720px and 3 channels
```

Here, the %d used is a format specifier for integers. The various format specifiers used in C Language are as follows :

| Format specifier | Meaning |
|---|---|
| %c | Print a single character |
| %i, %d | printf a decimal integer |
| %u | print an unsigned decimal integer |
| %x | print an unsigned hexadecimal integer using a,b,c,d,e,f |
| %X | printf an unsigned hexadecimal integers using A,B,C,D,E,F |
| %o | print an unsigned octal integer |
| %f | print a floating point number |
| %e | print the floating point number in the exponential format |
| %E | Same as the %e. but it is print E for exponent |
| %g | print the floating point number in float %f or %e format whichever is shorter. |
| %% | print the % sign. |
| %s | print the string |
| %p | print the pointer |

2) **Scanf :**

The scanf() function is used to obtain different values from the user and assign them to a variable using the ampersand sign(&). The syntax for using the scanf function is to :

- Open parenthesis
- Specify the format specifiers inside double quotes(“ ”).
- Mention the variable names using ampersand(&) which will provide us the address of that variable.
- Close it with a semicolon(;).

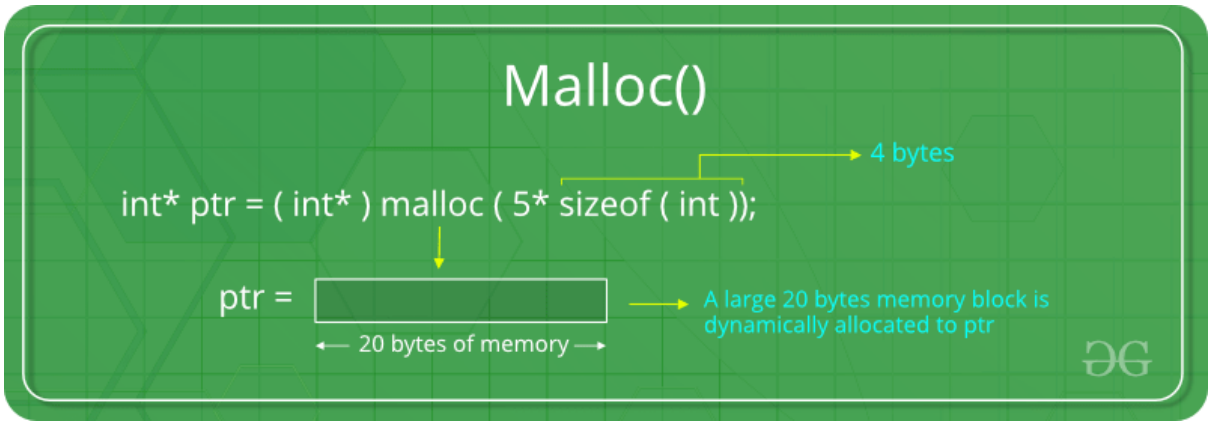
➢ Code :

```
printf("Enter the starting coordinates : ");
scanf("%d %d",&a,&b);
```

➢ Output :

```
Enter the starting coordinates : 300 300
```

The <stdlib.h> library is used for the malloc() function. The malloc() function allows the compiler to dynamically allocate memory from the heap of memory for a particular object. It returns the pointer pointing to the first element of the heap as the output. The word ‘malloc’ stands for memory allocation. The amount of memory to be allocated is specified inside the parenthesis.




The above image shows us the syntax and usage of the malloc function.

## 2.2 HEADER FILES

The various header files used in this project are listed below :

- stb_image
- stb_image_write
- stb_image_resize

The various functions used from those header files are :

- **stbi_load()**

  This function loads the image specified inside it. It also returns the width, height and number of channels in the loaded image. It returns a pointer of type unsigned character as the output. We should also specify the format of the image that is to be loaded.

  For example, if we want to load a image named ‘sample’ in ‘jpg’ format, and to get the width, height, and number of channels we should use the following syntax.

```
int width,height,channels;

unsigned char *img = stbi_load("sample.jpg",&width,&height,&channels,0);
```

- **stbi_write()**

  This function writes the image with the name and format specified. There are different functions for each image format. The various functions are

  - stbi_write_jpg()
  - stbi_write_png()

  The syntax of this function is the image name with format to be given as output, the width, height, number of channels. The fourth value given for a jpg image is set to 100 while for a png image it is width*channels. We should also mention the pointer pointing to the output image to write it to our folder.

The below image represents the syntax of the stbi_write() functions.

```
stbi_write_jpg("Crop.jpg", a, b, channels, ima_si, 100);
stbi_write_png("PsrCrop.png", (m-a+1), (n-b+1), channels, ima_si, (m-a+1) * channels);
```

- **free**

  The images are represented using various pointers in our program. So, the pointers as such may change the values of the pixels of the image. So, we should remove those pointers that we used. This is done with the help of the free() function. For the input image, it is stbi_image_free() while for the generated images it is free().

```
stbi_image_free(img);
free(ima_si);
```

- **Define and include lists**

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image/stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image/stb_image_write.h"
```

## 2.3 CROPPING AN IMAGE

The crop function built in this project crops an image by getting the coordinates of the top left pixel and bottom right pixel of the output image. A user can decide those values through estimation where the height and width are printed through the print statement after importing the image. It then creates another image with the dimension specified using the coordinates and pastes the value required in it. Finally, it writes the image to our working folder using the stbi_write() function.

## *Pseudocode*

1. Specify the various variables that we are going to use, i.e. width, height and channels. We also use variables c and d for storing the pixel value in each loop. Specify the values of c and d as -1. Also, initialise 4 variables a, b, m, and n to store the values of the coordinates entered by the user.
2. Initialise an unsigned char data type to store the data value of input image as pointer by using the stbi_load() function.
3. Use an if loop to check if the input image is NULL. If so, print a text showing an error message.
4. Next, print the width and height of the loaded image for the user.
5. Next, get the values of the coordinates from the user. Let it be
6. Next, initialise two size_t data types to store the size of images. This is a data type used to represent unsigned integers which in turn are used to represent the size in bits.
7. Use one for the size *(m-a+1)*(n-b+1)*channels*. This is used to represent our final image. This size is determined by the coordinates, where (a,b) is the top left coordinate while (m,n) is the bottom right coordinate.
8. Use the other one for the size *width*(n+1)*channels*. This is used to run the for loop. *n+1* is used because we should run the loop for *n+1* pixels along the vertical line(height) so that the entire part that we require is covered.
9. Next, initialise an unsigned char pointer datatype to point to the heap of memory allocated by the malloc function. Inside the parenthesis use the size value of the output image.
10. Next, initialise a for loop with

    **Initialisation** – unsigned char pointer pointing to both the input image and output image

    **Condition** – value of pointer pointing to original image not equal to original image + second allocated size value.

    **Updation** – pointer pointing to original image + number of channels.
11. Inside the for loop, increment the value of c with 1. Also do modulus of c with width so that whenever the value of c hits the width of the image, it will be resetted back to 0.
12. Whenever c is equal to 0, increment d by 1. This will create a function where c and d store the pixel value.
13. Now, use an if loop to print those values to our new image whenever the value of c and d equals the pixel value that we need.
14. Inside the if loop, provide the value at address of the pointer pointing to the output image equal to the input image for each 3 channel of the colour image. Inside that, use another if loop and check if the number of channels is 4. Is yes, include that value also in the output image.
15. Next, close the for loop.
16. Now, write the output image to our working folder by using the stbi_write() function by using the proper syntax.
17. Now, free the input and output images using the free() function.

## 2.4 EXTRACTING THE RGB COMPONENTS

We can extract the RGB Components of an image by setting the values of one component as such and making the value of other components 0. This will provide us with an output image that has only a single component. The value 0 indicates that there is no intensity for other components.

### *Pseudocode*

1. Specify the various variables that we are going to use. Specify the image's width, height, and channels under the integer datatype.
2. Initialise one unsigned char type pointer to point to the imported image and give its value using the stbi_load() function.
3. Next, use an if loop to check if the image got imported correctly.
4. Initialise a size_t datatype to represent the size of the output image. As the size of both the input and output images are same in this case, we can use one single variable. Set its value to *width*height*channels*.
5. Initialise an unsigned char pointer data type to point to the heap of memory allocated by using the malloc() function. Specify the space to be allocated as image size inside the malloc function.
6. Use an if loop to check if the space allocation is proper.
7. Initialise a for loop with the following conditions :

   **Initialisation** – unsigned char pointer pointing to both the input image and output image

   **Condition** – value of pointer pointing to original image not equal to original image + image size created.

   **Updation** – pointer pointing to original image and the output image + number of channels in each respectively.
8. Inside the for loop, for
   a. **Red** – Give the value of red as such and make values of others 0
   b. **Green** - Give the value of green as such and make values of others 0
   c. **Blue** - Give the value of blue as such and make values of others 0
9. Next, use an if loop inside to check if the number of channels is 4. If yes, make the value in new image same as that of the old image.
10. Close the for loop.
11. Now, write the output image to our working folder by using the stbi_write() function by using the proper syntax.
12. Free the input and output images using the free() function.

    This will provide us with separate images for each component, i.e. Red, Green and Blue.

## 2.5 GRAYSCALE CONVERSION

A grayscale image contains only one component while a colour image has 3 components. So, how do we convert a colour image to a grayscale one? Most of us may pause at this question. One possible way is to take the average of the 3 components and place as one components. But the formula which we are using here is

*0.299*Red + 0.587*Green + 0.114*Blue*

This is referred to as the NTSC Formula.

### *Pseudocode*

1. Specify the various variables that we are going to use. Specify the image's width, height, and channels under the integer datatype.
2. Initialise one unsigned char type pointer to point to the imported image and give its value using the stbi_load() function.
3. Next, use an if loop to check if the image got imported correctly.
4. Initialise two size_t datatypes to represent the size of the input and output images.
5. For the output image, use a **Ternary operator*** to check if the number of channels is 4 or not. If it is 4, set the number of gray channels to 2, or else to 1.
6. Initialise an unsigned char pointer data type to point to the starting point of the heap of memory allocated by the malloc() function. Specify the space to be allocated as output image size inside the malloc function.
7. Use an if loop to check if the space allocation is proper.
8. Initialise a for loop with the following conditions :

   **Initialisation** – unsigned char pointer pointing to both the input image and output image

   **Condition** – value of pointer pointing to original image not equal to original image + image size created.

   **Updation** – pointer pointing to original image and the output image + number of channels in each respectively.
9. Inside the for loop, mention the value at address of pointer pointing to the output grayscale image using the formula *0.299*Red + 0.587*Green + 0.114*Blue*. Represent each component with the respective pointers.
10. Next, use an if loop inside to check if the number of channels is 4. If yes, make the value in new image same as that of the old image.
11. Close the for loop.
12. Now, write the output image to our working folder by using the stbi_write() function by using the proper syntax.
13. Free the input and output images using the free() function.

This will provide us with an grayscale image corresponding to our colour image.

***Ternary operator** : It is a conditional operator which is a shortened form of the if-else statement. Its syntax is condition ? value if TRUE : value if FALSE.

## 2.6 BLACK AND WHITE CONVERSION

A Black and White image is an image that contains only black and white pixels. A pixel can either be black or white, but it cannot have an intermediate shade like a grayscale image. We can convert an image to a black and white image by setting the pixel value as black if the average of the 3 components in a colour image is less than 128 which is half of 256. If it is greater than or equal to 128, we can set it to white.

Let avg represent the average of those 3 components. If avg is greater than or equal to 128, the pixel will be white. Else, it will be black.

### *Pseudocode*

1. Specify the various variables that we are going to use. Specify the image's width, height, channels and a variable n to store the average of the three components under the integer datatype.
2. Initialise one unsigned char type pointer to point to the imported image and give its value using the stbi_load() function.
3. Next, use an if loop to check if the image got imported correctly.
4. Initialise two size_t datatypes to represent the size of the input and output images.
5. For the output image, use a Ternary operator to check if the number of channels is 4 or not. If it is 4, set the number of BandW channels to 2, or else to 1.
6. Initialise an unsigned char pointer data type to point to the starting point of the heap of memory allocated by the malloc() function. Specify the space to be allocated as output image size inside the malloc function.
7. Use an if loop to check if the space allocation is proper.
8. Initialise a for loop with the following conditions :

   **Initialisation** – unsigned char pointer pointing to both the input image and output image

   **Condition** – value of pointer pointing to original image not equal to original image + image size created.

   **Updation** – pointer pointing to original image and the output image + number of channels in each respectively.
9. Inside the for loop, write the equation of n as *n = (Red+Green+Blue)/2*.
10. Next, use an if loop to check if the value of n is greater than or equal to 128. If yes, make the value of the pixel as white(255). Or else, set the value to black(0).
11. Next, use an if loop to check if the number of channels is 4. If yes, make the value in new image same as that of the old image.
12. Close the for loop.
13. Now, write the output image to our working folder by using the stbi_write() function by using the proper syntax.
14. Free the input and output images using the free() function.

This will provide us with an black and white image corresponding to our input image.

## 2.7 SEPIA TONING

Sepia is a reddish brown colour named after the brown pigment derived from the inc sac of the cuttlefish sepia. The sepia toning process changes the colour format of the image to entire brownish colour. This will provide us an antique finish. This filter is used in movies, images made to give us an olden fashion, etc.

The sepia coefficients for conversion from colour to sepia is given below :

Red = min(0.393 * Red + 0.769 * Green + 0.189 *Blue, 255)

Green = min(0.349 * Red + 0.686 * Green + 0.168 *Blue, 255)

Blue = min(0.272 * Red + 0.534* Green + 0.131 *Blue, 255)

### *Pseudocode*

1. Specify the various variables that we are going to use. Specify the image's width, height, and channels under the integer datatype.
2. Initialise one unsigned char type pointer to point to the imported image and give its value using the stbi_load() function.
3. Next, use an if loop to check if the image got imported correctly.
4. Initialise a size_t datatype to represent the size of the output image. As the size of both the input and output images are same in this case, we can use one single variable. Set its value to *width*height*channels*.
5. Initialise an unsigned char pointer data type to point to the heap of memory allocated using the malloc() function. Specify the space to be allocated as image size inside the malloc function.
6. Use an if loop to check if the space allocation is proper.
7. Initialise a for loop with the following conditions :

   **Initialisation** – unsigned char pointer pointing to both the input image and output image

   **Condition** – value of pointer pointing to original image not equal to original image + image size created.

   **Updation** – pointer pointing to original image and the output image + number of channels in each respectively.
8. Inside the for loop, for
   a. **Red** – min(0.393 * Red + 0.769 * Green + 0.189 *Blue, 255)
   b. **Green** - min(0.349 * Red + 0.686 * Green + 0.168 *Blue, 255)
   c. **Blue** - = min(0.272 * Red + 0.534* Green + 0.131 *Blue, 255)
9. Next, use an if loop inside to check if the number of channels is 4. If yes, make the value in new image same as that of the old image.
10. Close the for loop.
11. Now, write the output image to our working folder by using the stbi_write() function by using the proper syntax.
12. Free the input and output images using the free() function.

This will provide us with an output sepia toned image.

## 2.8 INTER-CHANGING COMPONENTS :

The process of interchanging various components of the image is just setting the value of one component to the other and vice versa. This will give us some interesting outputs. Some outputs can even make us laugh.

### *Pseudocode*

1. Specify the various variables that we are going to use. Specify the image's width, height, and channels under the integer datatype.
2. Initialise one unsigned char type pointer to point to the imported image and give its value using the stbi_load() function.
3. Next, use an if loop to check if the image got imported correctly.
4. Initialise a size_t datatype to represent the size of the output image. As the size of both the input and output images are same in this case, we can use one single variable. Set its value to *width*height*channels*.
5. Initialise an unsigned char pointer data type to point to the heap of memory allocated using the malloc() function. Specify the space to be allocated as image size inside the malloc function.
6. Use an if loop to check if the space allocation is proper.
7. Initialise a for loop with the following conditions :

   **Initialisation** – unsigned char pointer pointing to both the input image and output image

   **Condition** – value of pointer pointing to original image not equal to original image + image size created.

   **Updation** – pointer pointing to original image and the output image + number of channels in each respectively.
8. Inside the for loop, for
   a. **Red – Green :** Set the value of the red component to green and green component to red.
   b. **Red – Blue:** Set the value of the red component to blue and blue component to red.
   c. **Green – Blue :** Set the value of the green component to blue and blue component to green.
   d. **Red -> Blue, Blue -> Green, Green -> Red :** Set the value of Red to Blue, Blue to Green and Green to Red.
   e. **Red -> Green, Green -> Blue, Blue -> Red :** Set the value of Red to Green, Green to Blue and Blue to Red.
9. Next, use an if loop inside to check if the number of channels is 4. If yes, make the value in new image same as that of the old image.
10. Close the for loop.
11. Now, write the output image to our working folder by using the stbi_write() function by using the proper syntax.
12. Free the input and output images using the free() function.

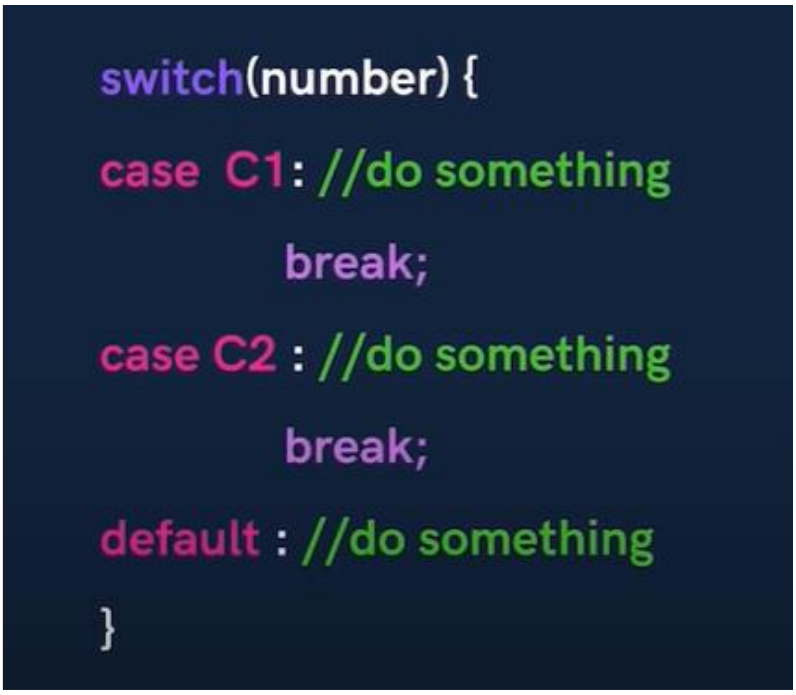
This will provide us with an interesting image as the output.

## 2.9 SWITCH-CASE

We have done different types of image processing and conversions. Now, how to include these as an single code? This can be done using the switch-case conditional statement.

***Pseudocode***

1. First, initialise a variable called *case* to store the case number of the image.
2. Print all those conversion that we have into the output window and give an number for each. Ask the user to enter the number of the conversion needed.
3. Store this value in the *case* variable.
4. Use this as the value for switch-case conditional statement and paste the various codes did according to the case numbers.
5. Don't forget to include a break statement at the end to break from the switch-case conditional statement.
6. Also print the default case as "Enter the correct case number from those available".
7. Use nested switches the same way if needed.




## 2.10 EXECUTABLE FILE

Click the 'Run Build Task' option available in the 'Terminal' pane of the Visual Studio Code application to create a executable file for our code.

# CHAPTER 3 : SOURCE CODE

```c
#include<stdio.h>
#include<stdlib.h>

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image/stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image/stb_image_write.h"

int main(void)
{
    int conversion;
    printf("Welcome to our Project...\n");
    printf("This program is a basic photo editor inside the Visual Studio Code
Platform.\n");
    printf("\nThis program has the following features :");

    printf("\n 1) Crop an image.");
    printf("\n 2) Extract the RGB Components of the image.");
    printf("\n 3) Convert the image to a greyscale image.");
    printf("\n 4) Convert the image to a Black and White image.");
    printf("\n 5) Convert the image to sepia colour format.");
    printf("\n 6) Interchange the components of the image.");

    printf("\nTo use those conversions, write the number of the conversion you
need : ");
    scanf("%d",&conversion);

    switch(conversion){
        case 1 :     {int width,height,channels,a,b,i,j,c=-1,d=-1,m,n;

    unsigned char *img = stbi_load("project.png",&width,&height,&channels,0);
    if(img == NULL){
        printf("Error in loading image.\n");
    }
    printf("Its width is %d and height is %d\n",width,height);
    printf("Enter the starting coordinates : ");
    scanf("%d %d",&a,&b);
    printf("Enter the ending coordinates : ");
    scanf("%d %d",&m,&n);

    size_t im_si = (m-a+1)*(n-b+1)*channels;
    size_t img_siz = (width)*(n+1)*channels;

    unsigned char *ima_si = malloc(im_si);
    if(ima_si == NULL){
        printf("Error in allocation.");
    }
```

```
    for(unsigned char *p = img,*pg=ima_si;p!=img_siz+img;p+=channels) {
        c+=1;
        c=c%(width);
        if(c==0){
            d+=1;
        }

        if(c>=a && c<=m && d>=b && d<=n){
        *pg = *p;
        *(pg+1) = *(p+1);
        *(pg+2) = *(p+2);

        if(channels==4){
            *(pg+3) = *(p+3);
        }

        pg+=channels;

        }

    }

    //stbi_write_jpg("Crop.jpg", a, b, channels, ima_si, 100);
    stbi_write_png("Crop.png", (m-a+1), (n-b+1), channels, ima_si, (m-a+1) *
channels);

    stbi_image_free(img);
    free(ima_si);}

    break;

    case 2 :

    {
    int number;
    printf("What component you need?\n");
    printf("Red - 1, Green - 2, Blue - 3 : ");
    scanf("%d",&number);

    switch(number){
        case 1 :
        {
            int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
```

```
        exit(1);
    }

    size_t img_size = width * height * channels;

    unsigned char *gray_img = malloc(img_size);
    if(gray_img == NULL) {
        printf("Unable to allocate memory for the gray image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = gray_img; p != img + img_size; p +=
channels, pg += channels) {
        *pg = *(p);
        *(pg+1) = (0);
        *(pg+2) = (0);

        if(channels == 4){
            *(pg+3) = *(p+3);
        }

    }
    //stbi_write_jpg("Red.jpg", width, height, channels, gray_img, 100);
    stbi_write_png("Red.png", width, height, channels, gray_img, width *
channels);
    }
    break;

    case 2 :
    {
        int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }

    size_t img_size = width * height * channels;

    unsigned char *gray_img = malloc(img_size);
    if(gray_img == NULL) {
        printf("Unable to allocate memory for the gray image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = gray_img; p != img + img_size; p +=
channels, pg += channels) {
```

```
        *pg = (0);
        *(pg+1) = *(p+1);
        *(pg+2) = (0);

        if(channels == 4){
            *(pg+3) = *(p+3);
        }

    }
    //stbi_write_jpg("Green.jpg", width, height, channels, gray_img, 100);
    stbi_write_png("Green.png", width, height, channels, gray_img, width *
channels);
    }
    break;

    case 3:
    {
        int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }

    size_t img_size = width * height * channels;

    unsigned char *gray_img = malloc(img_size);
    if(gray_img == NULL) {
        printf("Unable to allocate memory for the gray image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = gray_img; p != img + img_size; p +=
channels, pg += channels) {
        *pg = (0);
        *(pg+1) = (0);
        *(pg+2) = *(p+2);

        if(channels == 4){
            *(pg+3) = *(p+3);
        }

    }
    //stbi_write_jpg("Blue.jpg", width, height, channels, gray_img, 100);
    stbi_write_png("Blue.png", width, height, channels, gray_img, width *
channels);
    }
```

```
    break;

    default : printf("Enter the correct case number for Red,Green or
Blue...");
    }
    }

    break;


    case 3:
    {
        int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }

    size_t img_size = width * height * channels;
    int gray_channels = channels == 4 ? 2 : 1;
    size_t gray_img_size = width * height * gray_channels;

    unsigned char *gray_img = malloc(gray_img_size);
    if(gray_img == NULL) {
        printf("Unable to allocate memory for the gray image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = gray_img; p != img + img_size; p +=
channels, pg += gray_channels) {
        *pg = (uint8_t)((*p)*0.299 + *(p + 1)*0.587 + *(p + 2)*0.114);
        if(channels == 4) {
            *(pg + 1) = *(p + 3);
        }
    }

    //stbi_write_jpg("Grayscale.jpg", width, height, gray_channels, gray_img,
100);
    stbi_write_png("Grayscale.png", width, height, gray_channels, gray_img,
width * gray_channels);

    stbi_image_free(img);
    free(gray_img);
    }
    break;

    case 5 :
```

```
    {
        int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }

    size_t img_size = width * height * channels;

    unsigned char *sepia_img = malloc(img_size);
    if(sepia_img == NULL) {
        printf("Unable to allocate memory for the sepia image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = sepia_img; p != img + img_size; p +=
channels, pg += channels) {
        *pg       = (uint8_t)fmin(0.393 * *p + 0.769 * *(p + 1) + 0.189 * *(p
+ 2), 255.0);
        *(pg + 1) = (uint8_t)fmin(0.349 * *p + 0.686 * *(p + 1) + 0.168 * *(p
+ 2), 255.0);
        *(pg + 2) = (uint8_t)fmin(0.272 * *p + 0.534 * *(p + 1) + 0.131 * *(p
+ 2), 255.0);
        if(channels == 4) {
            *(pg + 3) = *(p + 3);
        }
    }

    //stbi_write_jpg("Sepia.jpg", width, height, channels, sepia_img, 100);
    stbi_write_png("Sepia.png", width, height, channels, sepia_img, width *
channels);

    stbi_image_free(img);
    free(sepia_img);
    }
    break;

    case 4 :
    {
    int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }
```

```
    size_t img_size = width * height * channels;
    int bchannels= channels == 4 ? 2 : 1;
    size_t newimg_siz = width*height*bchannels;

    unsigned char *grey_img = malloc(newimg_siz);
    if(grey_img == NULL){
        printf("Error in allocation.");
    }

    int n;
    for(unsigned char *a = img,*b = grey_img;a!=
img+img_size;a+=channels,b+=bchannels){
            n = (*a + *(a+1) + *(a+2))/3;

            if(n>=128){
                *b = 255;
            }
            else{
                *b = 0;
            }

            if(channels == 4){
                *(b+1) = *(a+3);
            }
    }


    stbi_write_jpg("Black_and_White.jpg",width,height,bchannels,grey_img,100);
    //stbi_write_png("Black_and_White.png", width, height, bchannels,
grey_img, width * bchannels);


    stbi_image_free(img);
    free(grey_img);
    }
    break;

    case 6 :
    {
        int casenum;
        printf("1)Red <-> Green");
        printf("\n2)Red <-> Blue");
        printf("\n3)Blues <-> Green");

        printf("\n4)Red -> Blue,Blue -> Green, Green -> Red");
        printf("\n5)Red -> Green,Green -> Blue, Blue -> Red");

        printf("\nEnter the case number : ");
```

```
        scanf("%d",&casenum);

        switch(casenum){
            case 1 :
                {
        int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }

    size_t img_size = width * height * channels;

    unsigned char *gray_img = malloc(img_size);
    if(gray_img == NULL) {
        printf("Unable to allocate memory for the gray image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = gray_img; p != img + img_size; p +=
channels, pg += channels) {
        *pg = *(p+1);
        *(pg+1) = *(p);
        *(pg+2) = *(p+2);

        if(channels == 4){
            *(pg+3) = *(p+3);
        }

    }
    //stbi_write_jpg("RG.jpg", width, height, channels, gray_img, 100);
    stbi_write_png("RG.png", width, height, channels, gray_img, width *
channels);
    }
    break;

    case 2 :
                {
        int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }
```

```
    size_t img_size = width * height * channels;

    unsigned char *gray_img = malloc(img_size);
    if(gray_img == NULL) {
        printf("Unable to allocate memory for the gray image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = gray_img; p != img + img_size; p +=
channels, pg += channels) {
        *pg = *(p+2);
        *(pg+1) = *(p+1);
        *(pg+2) = *(p);

        if(channels == 4){
            *(pg+3) = *(p+3);
        }

    }
    //stbi_write_jpg("RB.jpg", width, height, channels, gray_img, 100);
    stbi_write_png("RB.png", width, height, channels, gray_img, width *
channels);
    }
    break;


    case 3 :
                {
        int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }

    size_t img_size = width * height * channels;

    unsigned char *gray_img = malloc(img_size);
    if(gray_img == NULL) {
        printf("Unable to allocate memory for the gray image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = gray_img; p != img + img_size; p +=
channels, pg += channels) {
        *pg = *(p);
        *(pg+1) = *(p+2);
        *(pg+2) = *(p+1);
```

```
        if(channels == 4){
            *(pg+3) = *(p+3);
        }

    }
    //stbi_write_jpg("BG.jpg", width, height, channels, gray_img, 100);
    stbi_write_png("BG.png", width, height, channels, gray_img, width *
channels);
    }
    break;

    case 4 :

                {
        int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }

    size_t img_size = width * height * channels;

    unsigned char *gray_img = malloc(img_size);
    if(gray_img == NULL) {
        printf("Unable to allocate memory for the gray image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = gray_img; p != img + img_size; p +=
channels, pg += channels) {
        *pg = *(p+1);
        *(pg+1) = *(p+2);
        *(pg+2) = *(p);

        if(channels == 4){
            *(pg+3) = *(p+3);
        }

    }
    //stbi_write_jpg("RGB1.jpg", width, height, channels, gray_img, 100);
    stbi_write_png("RGB1.png", width, height, channels, gray_img, width *
channels);
    }
    break;
```

```
    case 5 :

                {
        int width, height, channels;
    unsigned char *img = stbi_load("project.png", &width, &height, &channels,
0);
    if(img == NULL) {
        printf("Error in loading the image\n");
        exit(1);
    }

    size_t img_size = width * height * channels;

    unsigned char *gray_img = malloc(img_size);
    if(gray_img == NULL) {
        printf("Unable to allocate memory for the gray image.\n");
        exit(1);
    }

    for(unsigned char *p = img, *pg = gray_img; p != img + img_size; p +=
channels, pg += channels) {
        *pg = *(p+2);
        *(pg+1) = *(p);
        *(pg+2) = *(p+1);

        if(channels == 4){
            *(pg+3) = *(p+3);
        }

    }
    //stbi_write_jpg("RGB2.jpg", width, height, channels, gray_img, 100);
    stbi_write_png("RGB2.png", width, height, channels, gray_img, width *
channels);
    }
    break;


    default : printf("Enter the correct case number for conversion..");

        }

    }
    default : printf("Enter the correct case number for image conversion...");


    }

}
```

# CHAPTER 4 : RESULTS

## 4.2 TEST 1

***Original Image***

***Cropping Image***

Application :

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 1
Its width is 1280 and height is 960
Enter the starting coordinates : 200 175
Enter the ending coordinates : 1000 900
```

Output :

***Extracting RGB Components***

Application :

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 2
What component you need?
Red - 1, Green - 2, Blue - 3 :
```

Output :

***Greyscale***

Application:

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 3
```

Output :

***Black and White***

Application :

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 4
```

Output :

***Sepia Toning***

Application :

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 5
```

Output :

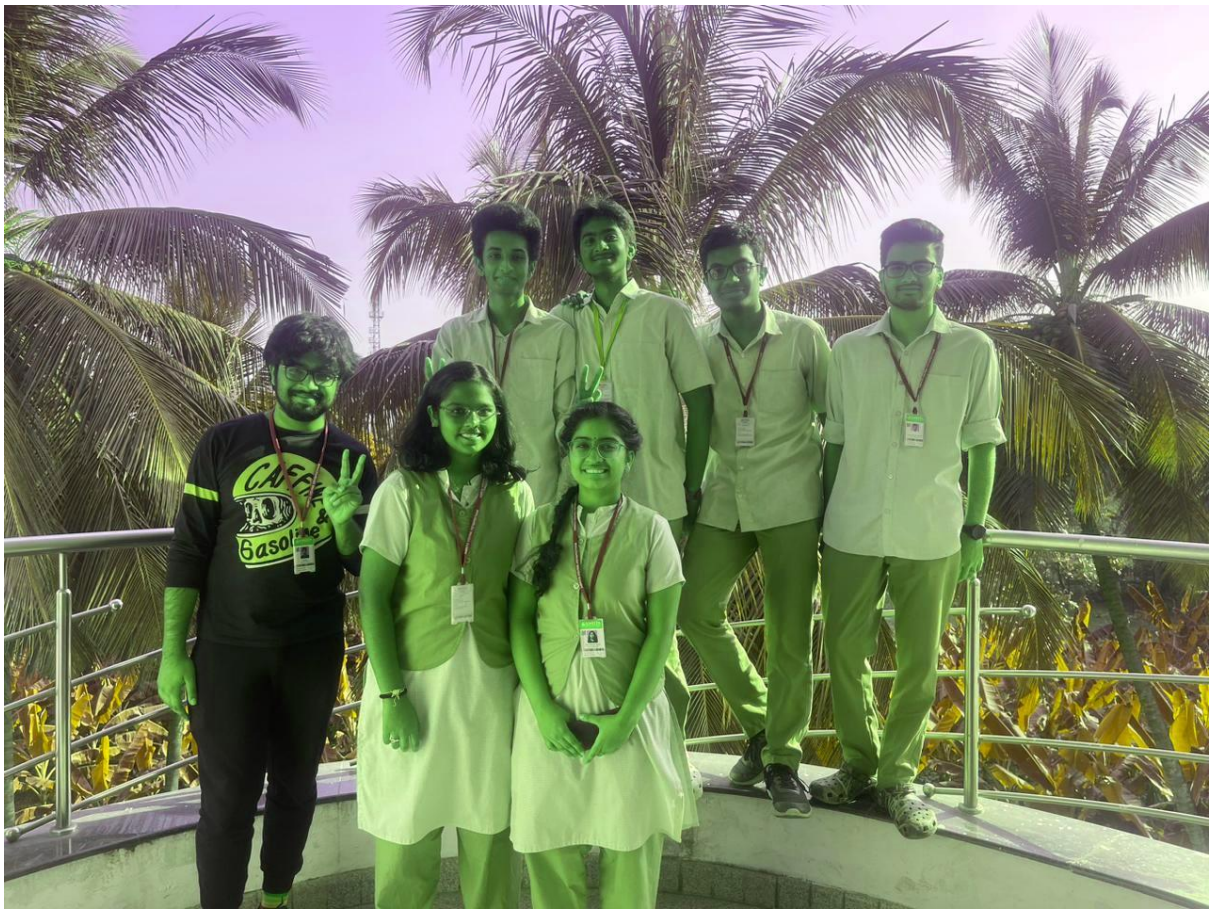
***Interchanging Components***

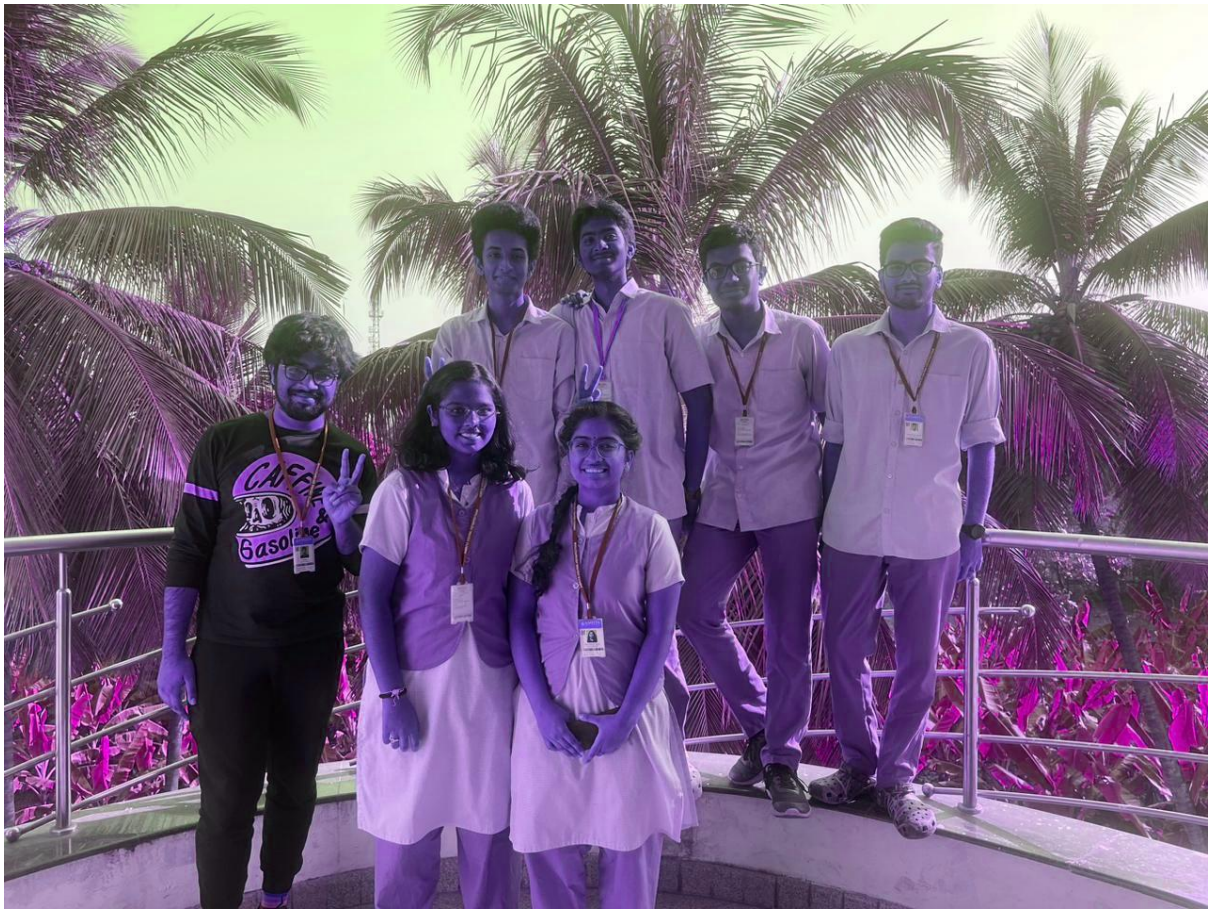
Application :

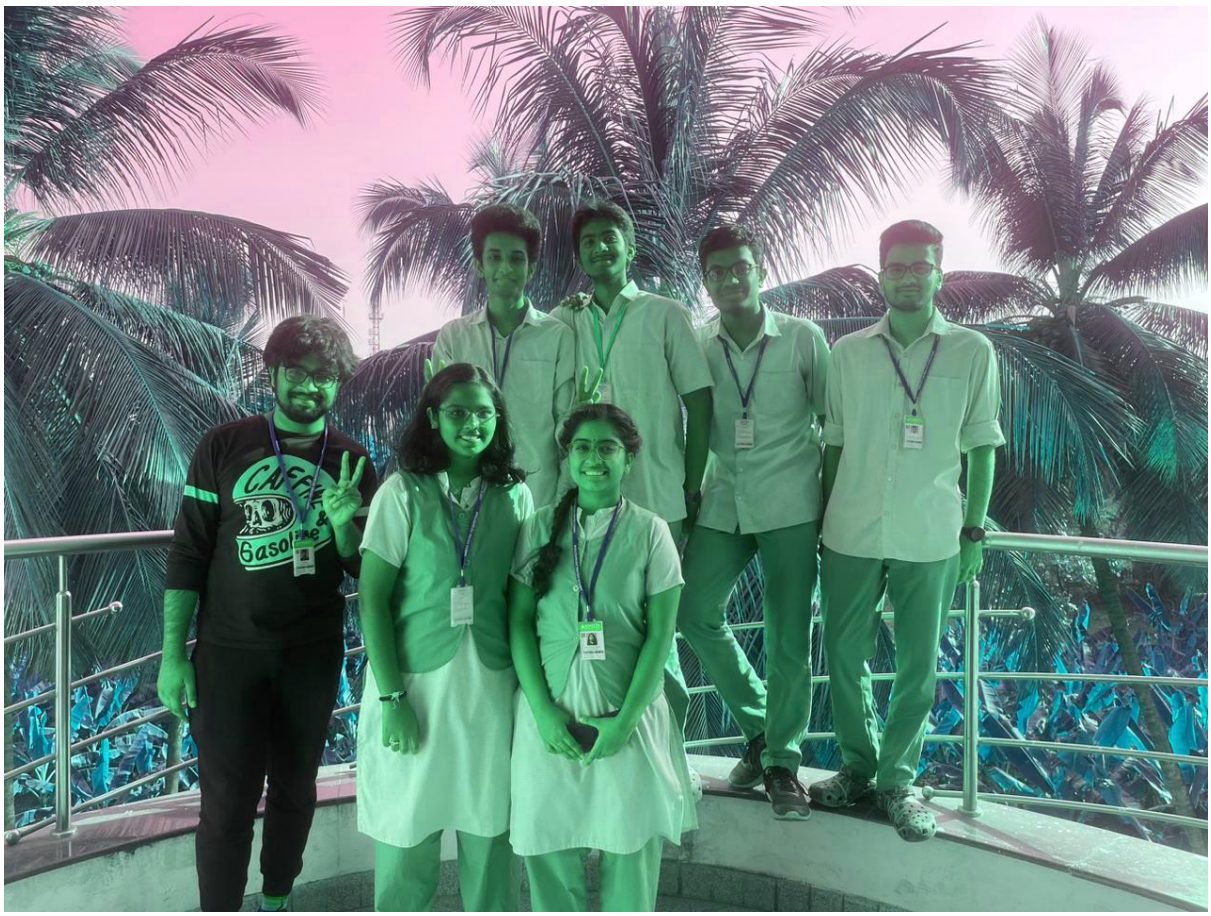
```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 6
1)Red <-> Green
2)Red <-> Blue
3)Blues <-> Green
4)Red -> Blue,Blue -> Green, Green -> Red
5)Red -> Green,Green -> Blue, Blue -> Red
Enter the case number :
```

Output :







## 4.2 TEST 2

***Original Image***

## *Cropping Image*

Application :

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 1
Its width is 768 and height is 1024
Enter the starting coordinates : 250 300
Enter the ending coordinates : 500 900
```

Output :

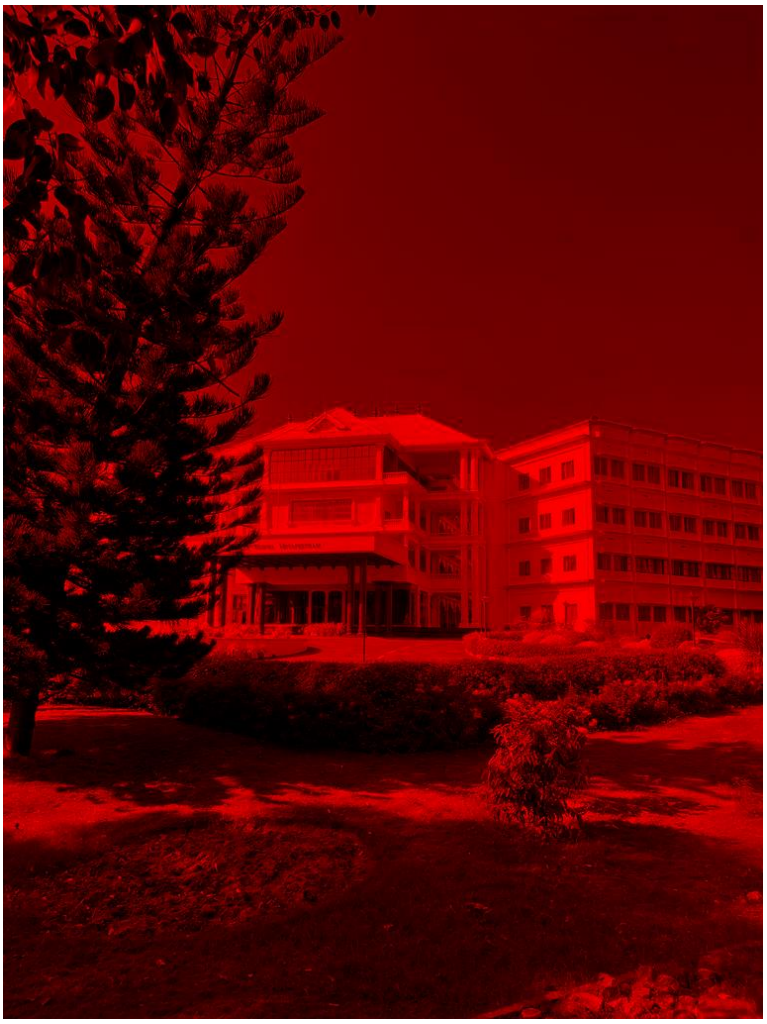
## *Extracting RGB Components*

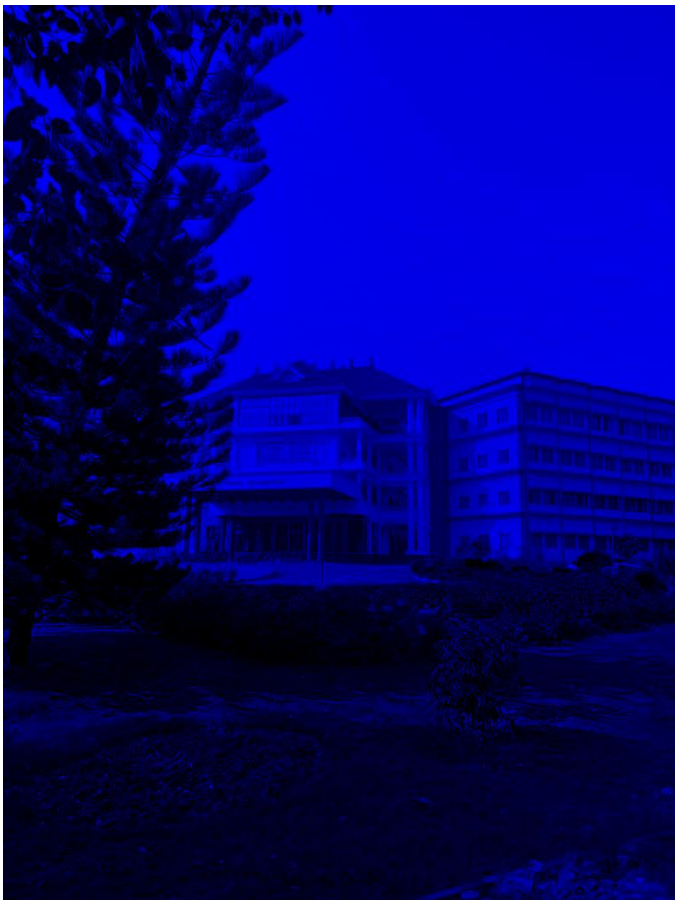
Application :

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 2
What component you need?
Red - 1, Green - 2, Blue - 3 :
```

Output :





***Greyscale***

Application:

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 3
```

Output :

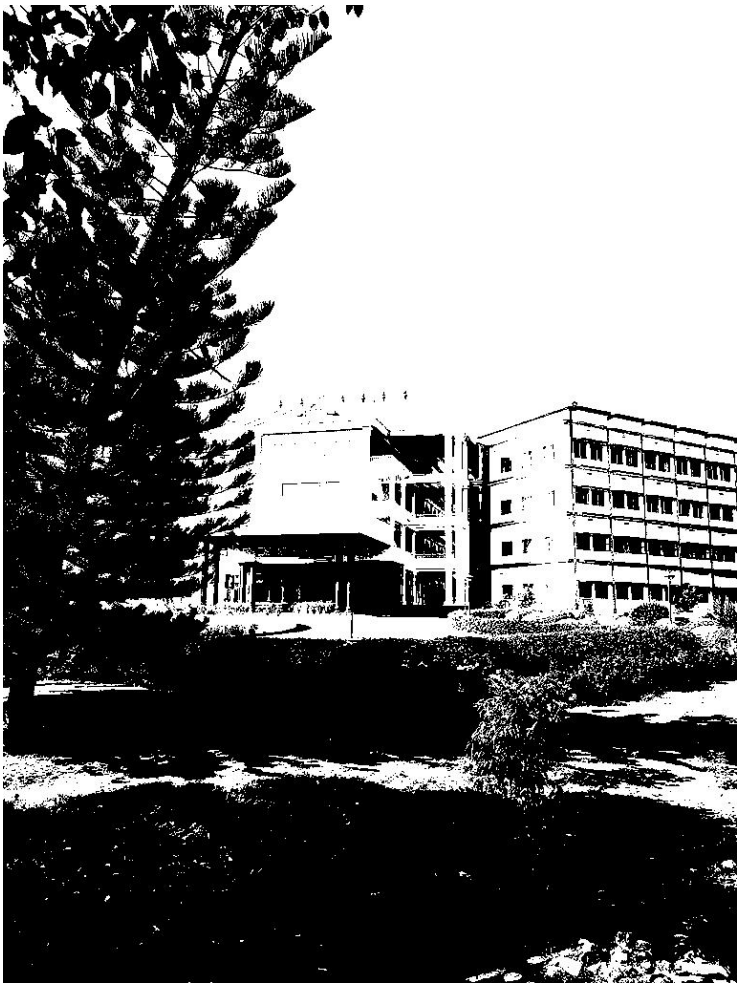
***Black and White***

Application :

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 4
```

Output :



***Sepia Toning***

Application :

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 5
```

Output :

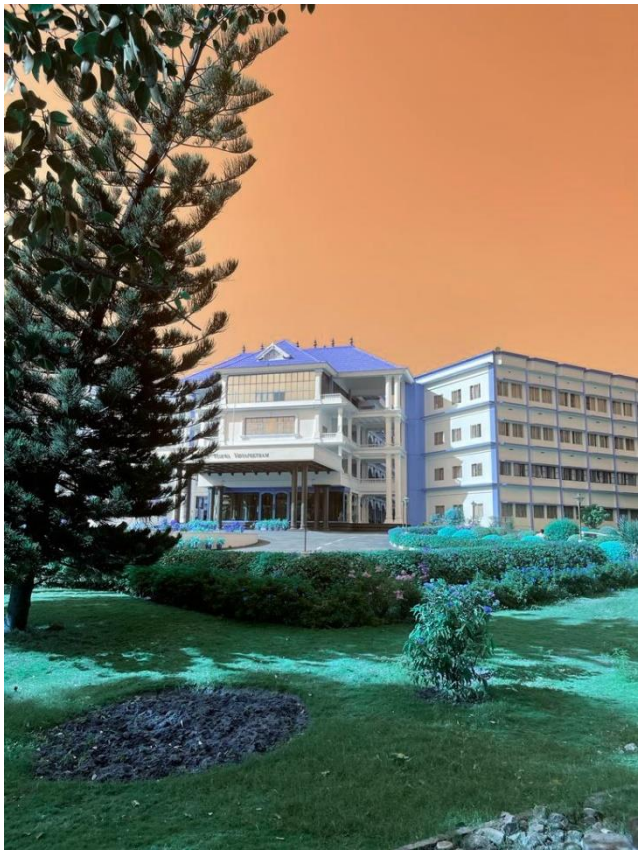
***Interchanging Components***

Application :

```
Welcome to our Project...
This program is a basic photo editor inside the Visual Studio Code Platform.

This program has the following features :
 1) Crop an image.
 2) Extract the RGB Components of the image.
 3) Convert the image to a greyscale image.
 4) Convert the image to a Black and White image.
 5) Convert the image to sepia colour format.
 6) Interchange the components of the image.
To use those conversions, write the number of the conversion you need : 6
1)Red <-> Green
2)Red <-> Blue
3)Blues <-> Green
4)Red -> Blue,Blue -> Green, Green -> Red
5)Red -> Green,Green -> Blue, Blue -> Red
Enter the case number :
```

Output :



Remarks : Alien Planet

# CHAPTER 5 : CONCLUSION

In tis project, we saw about basic image processing like cropping, grayscale conversion, black and white conversion, etc. At last, we also saw some interesting image conversions. From this we can understand that the basic photo edits that we do today have very complex codes behind. With this basic knowledge about image processing, we can do a lot more by extending our ideas. We can also do video processing with this basic idea. Image processing is also one major area of development under Artificial Intelligence(AI). So, we conclude that Image Processing has various useful applications in the upcoming future and learning this will prove useful later on.

# CHAPTER 6 : REFERENCES


**Image 1 :**

https://www.google.com/search?rlz=1C1RXQR_enIN1028IN1028&sxsrf=AJOqlzUUBG5HYFBywvH_buOk4ytJlpGoaQ:1675665194588&q=low+quality+image&tbm=isch&sa=X&ved=2ahUKEwix9L2lo4D9AhXW6jgGHSdDDPEQ0pQJegQIEhAB&biw=1920&bih=961&dpr=1#imgrc=fuFQOVROkwfokM

**Image 2 :**

https://www.google.com/search?sxsrf=AJOqlzVHaVVaY-jI5OlW6YkuGpxW-ue3XA:1675665670503&q=channels+in+an+image&tbm=isch&sa=X&ved=2ahUKEwiyzbWIpYD9AhVKasAKHdQqC_QQ0pQJegQIDxAB#imgrc=7BxsmdgpFgVPqM

**Image 3 :**

https://www.google.com/search?rlz=1C1RXQR_enIN1028IN1028&sxsrf=AJOqlzXd-vL5VlLN6Ssn6aWb2kRzI1p46g:1675668205495&q=format+specifiers+in+c&tbm=isch&sa=X&ved=2ahUKEwien5nBroD9AhUMiFwKHc7zCEgQ0pQJegQIDhAB&biw=1920&bih=961&dpr=1#imgrc=g0_4P4rEzrQ-AM

**Image 4 :**

https://www.google.com/search?sxsrf=AJOqlzVMiqf1ExhB4oqTrXyHaAJabPP0AQ:1675673210701&q=malloc+in+c&tbm=isch&sa=X&ved=2ahUKEwj71O6TwYD9AhWpwjgGHTySBeEQ0pQJegQIDBAB&biw=1920&bih=961&dpr=1#imgrc=6r5i2SUioOcMoM

**Header Files :**

https://github.com/sol-prog/tutorial_stb-image_library_examples