KENDRIYA VIDYALAYA NO.3 NAL

COMPUTER SCIENCE PROJECT

Image Editor

Academic Year 2012-13

Submitted To:

Mr.Hansraj Saini

PGT Computer Science

Submitted By :

Diptanshu Kakwani , XII Sci

Divyanshu Kakwani , XII Sci

TABLE OF CONTENTS

ACKNOWLEDGEMENT

The project received a great help from our elder brother, Chitresh Kakwani, a veteran computer engineer. He helped us in choosing the right format (BMP), debugging the program and making the program compiler-independent.

Another great contributor to this project was our elder sister, Kumudini Kakwani, a to-be computer engineer. She gave us the outline of the contrast algorithm and inspected our other algorithms .

Our family kept encouraging us throughout the painstaking development period of one month so that we never lost hope.They provided us constant feedback to improve the program.

Many of our queries were resolved by our computer teacher, Mr. Hansraj Saini. Sir too assisted us in making the project and its documentation.

Among our friends, Hardik and Harshul suggested us some effects and ways to improve the interface of the program.

We owe a lot to all the contributors of this project who made this project a success.

Regards

Diptanshu and Divyanshu

COMPUTER SCIENCE PROJECT

IMAGE EDITOR

this certificate is awarded to:

Diptanshu Kakwani & Divyanshu Kakwani

in recognition of

Mr. Hansraj Saini

Mr. Hansraj Saini PGT Comp Sci KV NO. 3 NAL	External	Mr. U.S. Vijay Principal KV NO.3 NAL	DATE

INTRODUCTION

The image editor first takes a 24 bit-per-pixel bitmap image as input. It then displays the details of the image followed by the menu. The user is then required to enter his/her choice. Depending on the choice, the program manipulates the image data and writes the image in the "output.bmp" file.

The program is capable of adding following effects:

1. Convert to Grayscale
2. Change Brightness
3. Add Contrast
4. Boost the color
5. Pixelate part of the image
6. Crop
7. Resize
8. Convert to its Negative

The program reinforces the following C++ concepts:

1. Classes and Objects
2. Inheritance
3. File Handling
4. Dynamic Memory Allocation

The code is 342 lines long. It is developed in dev-cpp and compiled through MinGW GCC 4.7.0 32-bit .The program is platform dependent and works only in windows.

DIGITAL IMAGES

A digital image is a numeric representation (normally binary) of a two-dimensional image. Digital Images are electronic snapshots taken of a scene or scanned from documents, such as photographs, manuscripts, printed texts, and artwork. The digital image is sampled and mapped as a grid of dots or picture elements (pixels). Each pixel is assigned an RGB value (Red, Green, Blue), which is represented in binary code (zeros and ones). The binary digits ("bits") for each pixel are stored in a sequence by a computer and often reduced to a mathematical representation (compressed). The bits are then interpreted and read by the computer to produce an analog version for display or printing.

Raster

Raster images have a finite set of digital values, called picture elements or pixels. The digital image contains a fixed number of rows and columns of pixels. Pixels are the smallest individual element in an image, holding quantized values that represent the brightness of a given color at any specific point.

Typically, the pixels are stored in computer memory as a raster image or raster map, a two-dimensional array of small integers. These values are often transmitted or stored in a compressed form.

Vector

Vector images resulted from mathematical geometry (vector). In mathematical terms, a vector consists of point that has both direction and length.

Often, both raster and vector elements will be combined in one image; for example, in the case of a billboard with text (vector) and photographs (raster).

Pixel resolution

The set of two positive integer numbers, where the first number is the number of pixel columns (width) and the second is the number of pixel rows (height).

IMAGE FORMATS

Image file formats are standardized means of organizing and storing digital images. Image files are composed of digital data in one of these formats that can be rasterized for use on a computer display or printer. An image file format may store data in uncompressed, compressed, or vector formats. Once rasterized, an image becomes a grid of pixels, each of which has a number of bits to designate its color equal to the color depth of the device displaying it.

Image file compression

There are two types of image file compression algorithms: lossless and lossy.

Lossless compression algorithms reduce file size while preserving a perfect copy of the original uncompressed image. Lossless compression generally, but not exclusively, results in larger files than lossy compression. Lossless compression should be used to avoid accumulating stages of re-compression when editing images.

Lossy compression algorithms preserve a representation of the original uncompressed image that may appear to be a perfect copy, but it is not a perfect copy. Oftentimes lossy compression is able to achieve smaller file sizes than lossless compression. Most lossy compression algorithms allow for variable compression that trades image quality for file size.

Major graphic file formats:

Raster formats:

JPEG/JFIF	JPEG 2000	Exif	TIFF
GIF	BMP	PNG	RAW

THE BMP FORMAT

The BMP file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows and OS/2 operating systems.

The BMP file format is capable of storing 2D digital images of arbitrary width, height, and resolution, both monochrome and color, in various color depths, and optionally with data compression, alpha channels, and color profiles.

File structure

The bitmap image file consists of fixed-size structures (headers) as well as variable-size structures appearing in a predetermined sequence.

BITMAP FILE HEADER

Offset#	Size	Purpose
0000_h	2 bytes	The header field used to identify the BMP & DIB file is 0x42 0x4D in hexadecimal, same as BM in ASCII. The following entries are possible: • **BM** – Windows 3.1x, 95, NT, ... etc.
0002_h	4 bytes	the size of the BMP file in bytes
0006_h	2 bytes	reserved; actual value depends on the application that creates the image
0008_h	2 bytes	reserved; actual value depends on the application that creates the image
$000A_h$	4 bytes	The offset, i.e. starting address, of the byte where the bitmap image data (pixel array) can be found.

DIB header (bitmap information header)

Offset #	Size	Purpose
0Eh	4	the size of this header (40 bytes)
12h	4	The bitmap width in pixels (signed integer).
16h	4	The bitmap height in pixels (signed integer).
1Ah	2	The number of color planes being used. Must be set to 1.
1Ch	2	The number of bits per pixel, which is the color depth of the image. Typical values are 1, 4, 8, 16, 24 and 32.
1Eh	4	The compression method being used. See the next table for a list of possible values.
22h	4	The image size. This is the size of the raw bitmap data (see below), and should not be confused with the file size.
26h	4	The horizontal resolution of the image. (pixel per meter, signed integer)
2Ah	4	The vertical resolution of the image. (pixel per meter, signed integer)
2Eh	4	The number of colors in the color palette, or 0 to default to 2^n.
32h	4	The number of important colors used, or 0 when every color is important; generally ignored.

Pixel array (bitmap data)

The pixel array is a block of 32-bit DWORDs, that describes the image pixel by pixel. Normally pixels are stored "upside-down" with respect to normal image raster scan order, starting in the lower left corner, going from left to right, and then row by row from the bottom to the top of the image.

Pixel format

The 24-bit pixel (24bpp) format supports 16,777,216 distinct colors and stores 1 pixel value per 3 bytes. Each pixel value defines the red, green and blue samples of the pixel (8.8.8.0.0 in RGBAX notation). Specifically in the order (blue, green and red, 8-bits per each sample).

Padding

Padding bytes (not necessarily 0) must be appended to the end of the rows in order to bring up the length of the rows to a multiple of four bytes. When the pixel array is loaded into memory, each row must begin at a memory address that is a multiple of 4. This address/offset restriction is mandatory only for Pixel Arrays loaded in memory. For file storage purposes, only the size of each row must be a multiple of 4 bytes while the file offset can be arbitrary.
A 24-bit bitmap with Width=1, would have 3 bytes of data per row (blue, green, red) and 1 byte of padding, while Width=2 would have 2 bytes of padding, Width=3 would have 3 bytes of padding, and Width=4 would not have any padding at all.

DATA STRUCTURE

A Class **imgheader**

Having Data Memebers:

- ✓ char format[2]
- ✓ int size
- ✓ short int reserved1
- ✓ short int reserved2
- ✓ int start_pos
- ✓ int header_size
- ✓ int width
- ✓ int height
- ✓ short int color_planes
- ✓ short int bpp
- ✓ int comp_method
- ✓ int image_size
- ✓ int hres
- ✓ int vres
- ✓ int colors_in_palette
- ✓ int imp_colors

And Member Functions:

- ✓ Void readHeader()
- ✓ Void writeHeader()

A Class **Image**

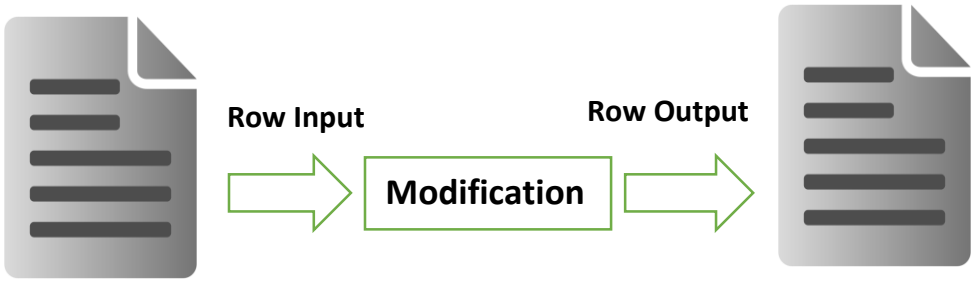
Having Data Members:

- ✓ int padding
- ✓ int row_size
- ✓ char file_name[100]
- ✓ unsigned char* row
- ✓ int i
- ✓ int j

And Member Functions:

- ✓ Void read()
- ✓ Void write()
- ✓ Void negative()
- ✓ Void grayscale()
- ✓ Void brightness(float intensity)
- ✓ Void crop(int hcut,int vcut)
- ✓ Image()
- ✓ ~Image()
- ✓ Void resize(int n)
- ✓ Void pixelate(float intensity,int y1,int y2,int x1,int x2)
- ✓ Void color_boost(float intensity)
- ✓ Void contrast(float intensity)
- ✓ Void display_info()
- ✓ Void developers()

ALGORITHMS



Input File Output File

Colour Boost(float intensity)

```
writeHeader();
i=0;
For  i<height:
        Read one row from the file to row[];
        j=0;
        For  j<( row_size -padding):
                row[j]+=row[j]/(100-intensity_of_color_boost);
                if(row[j]>255)      row[j]=255;
                j=j+1;
        Write the modified row[]  in the output file;
        i=i+1;
```

Grayscale

```
writeHeader();
i=0;
For  i<height:
        Read one row from the file to row[];
        j=0;
        For  j<(row_size-padding):
                average=(row[j]+row[j+1]+row[j+2])/3;
                row[j]=row[j+1]=row[j+2]=average;
                j=j+3;
        Write the modified row[]  in the output file;
        i=i+1;
```

Resize(int n) // Factor by which the image is to be reduced

```
width=width/n;
height=height/n;
new_ padding=(4-((new_width*3)%4))%4;
size=(new_height)*(new_width)*3+54+new_padding*new_height;
 image_size=new_size-54;
new_row_size=width*3+new_padding;
unsigned char* new_row=new unsigned char[new_row_size];
writeHeader();

 i=0;
 For  i<height:
        Read one row from the file to row[];
        j=0;
        while(j<new_row_size){
              if(j<new_row_size-padding)   new_row[j]=row[n*(j- (j%3) )+(j%3)];
              else   new_row[j]=0;
            j=j+1;
       Write the modified row[]  in the output file;
       Move the seekg pointer (n-1)*row_size ahead so as to skip (n-1)  rows;
       i=i+1;
 delete new_row;
```

Negative

```
writeHeader();
i=0;
For  i<height:
        Read one row from the file to row[];
        j=0;
        For  j<( row_size -padding):
              row[j]=255-row[j];
              j=j+1;
        Write the modified row[]  in the output file;
        i=i+1;
```

Pixelation(float intensity,int y1,int y2,int x1,int x2)

// coordinates = (x1,y1) (x1,y2) (x2,y1) (x2,y2)

```
x1*=3 , x2*=3;
writeHeader();
avg[3]={0,0,0};
i=0,j=0;
For  i<height:
        Read one row from the file to row[];
        j=x1;
        If(i lies between y1 & y2)
              For  j<x2:
                  for(int n=0;n<3*intensity;n++)  { avg[(j+n)%3]+=row[j+n]/intensity;}
                   for(int n=0;n<3*intensity;n++) {   row[j+n]=avg[(j+n)%3];    }
                    j+=3*intensity;
                   avg[0]=avg[1]=avg[2]=0;
                   j=j+1;
        Write the modified row[]  in the output file;
        i=i+1;
```

Contrast(float intensity) // Darker pixel gets darker, brighter gets brighter

```
writeHeader();
i=0;
For  i<height:
        Read one row from the file to row[];
        j=0;
        For  j<( row_size -padding):
              if(row[j]>=150)
                    if((row[j]+(row[j]-150)/intensity)>255)   row[j]=255;
                    else  row[j]+=(row[j]-150)/intensity;
              if(row[j]<150)
                    if((row[j]-(150-row[j])/intensity)>0)  row[j]-=(150-row[j])/intensity;
                    else row[j]=0;
              j=j+1;
        Write the modified row[]  in the output file;
        i=i+1;
```

Brightness(float intensity)

```
writeHeader();
i=0;
For  i<height:
        Read one row from the file to row[];
        j=0;
        For  j<( row_size -padding):
                if(row[j]+intensity>255)     row[j]=255;
                 else if(row[j]+intensity<0) row[j]=0;
                else                                 row[j]+=intensity;
                j=j+1;
        Write the modified row[]  in the output file;
        i=i+1;
```

Crop (int hcut,int vcut) // horizontal cut and vertical cut

```
width=width/n;
height=height/n;
new_ padding=(4-((new_width*3)%4))%4;
size=(new_height)*(new_width)*3+54+new_padding*new_height;
image_size=new_size-54;
new_row_size=width*3+new_padding;
unsigned char* new_row=new unsigned char[new_row_size];
writeHeader();
Move the seekg pointer vcut*row_size ahead so as to skip vcut  rows;
 i=0;
 For  i<new_height:
        Read one row from the file to row[];
        j=hcut*3;
        while(j<new_row_size+hcut*3){
            if(j<row_size-padding)            new_row[j-hcut*3]=row[j];
            else       new_row[j]=0;
            j=j+1;
        Write the modified row[]  in the output file;
        i=i+1;
delete new_row;
```

SOURCE CODE

```
#include<iostream>
#include<fstream>
#include<windows.h>
#include<string.h>
using namespace std;
ifstream ifile;
ofstream ofile("output.bmp",ios::binary);
class ImgHeader{
        public:
          char format[2];
         int size;
         short int reserved1;
         short int reserved2;
         int start_pos;
         int header_size;
         int width;
         int height;
         short int color_planes;
         short int bpp;
         int comp_method;
         int image_size;
         int hres;
         int vres;
         int colors_in_palette;
         int imp_colors;
         void readHeader(){
        ifile.read((char*)&format,2);
        ifile.read((char*)&size,4);
        ifile.read((char*)&reserved1,2);
        ifile.read((char*)&reserved2,2);
```

```
        ifile.read((char*)&start_pos,4);
        ifile.read((char*)&header_size,4);
        ifile.read((char*)&width,4);
        ifile.read((char*)&height,4);
        ifile.read((char*)&color_planes,2);
        ifile.read((char*)&bpp,2);
        ifile.read((char*)&comp_method,4);
        ifile.read((char*)&image_size,4);
        ifile.read((char*)&hres,4);
        ifile.read((char*)&vres,4);
        ifile.read((char*)&colors_in_palette,4);
        ifile.read((char*)&imp_colors,4);
              }
         void writeHeader(){
        ofile.write((char*)&format,2);
        ofile.write((char*)&size,4);
        ofile.write((char*)&reserved1,2);
        ofile.write((char*)&reserved2,2);
        ofile.write((char*)&start_pos,4);
        ofile.write((char*)&header_size,4);
        ofile.write((char*)&width,4);
        ofile.write((char*)&height,4);
        ofile.write((char*)&color_planes,2);
        ofile.write((char*)&bpp,2);
        ofile.write((char*)&comp_method,4);
        ofile.write((char*)&image_size,4);
        ofile.write((char*)&hres,4);
        ofile.write((char*)&vres,4);
        ofile.write((char*)&colors_in_palette,4);
        ofile.write((char*)&imp_colors,4);
         }
};
```

```
class image:public ImgHeader{
        public:
                char file_name[100];
                int padding;
        int row_size;
      unsigned char* row;
          int i,j;
        image(){
                developers();
                cout<<"\nEnter image file name(a 24 bit bmp file)\t";
                cin>>file_name;
                ifile.open(file_name,ios::binary);
                if(!ifile){  //If File doesnt exist,exit the program
                 cerr<<"\n\n\tFile Not Found!";
                 exit(17);
                 }
                readHeader();
                padding=(4-((width*3)%4))%4;
       row_size=width*3+padding;
       row=new unsigned char[row_size];
                display_info();
                if(!(format[0]=='B'&& format[1]=='M') || bpp!=24){
                 cerr<<"\n\n\t\tInvalid Format !!";
                 exit(3);
                 }
        }
        ~image(){
                delete row;
        }
        void grayscale();
        void negative();
        void brightness(float intensity);
```

```
        void pixelate(float intensity,int y1,int y2,int x1,int x2);
        void color_boost(float intensity);
        void contrast(float intensity);
        void crop(int hcut,int vcut);
        void resize(int n);
        void display_info(){
         cout<<"\nThe size of the image is :  "<<size/1024.0<<"  Kilobytes(KB)";
         cout<<"\nThe width of the image is :  "<<width<<"  Pixels";
         cout<<"\nThe height of the image is :  "<<height<<"  Pixels\n\n";
      }
        void developers(){
         cout<<"\n\n\tDevelopers of the program:\n \n"
                  <<"\tDivyanshu Kakwani,XII Sci,120017\n"
                  <<"\tDiptanshu Kakwani,XII Sci,120016\n";
       }
};
void image::grayscale(){
        writeHeader();
        unsigned char avg;
   int i(0),j(0);
   while(i++<height){
       ifile.read((char*)row,row_size);
       j=0;
       while(j<row_size-padding){
          avg=(row[j]+row[j+1]+row[j+2])/3;
          row[j]=avg;
          row[j+1]=avg;
          row[j+2]=avg;
          j=j+3;
       }
       ofile.write((char*)row,row_size);
   }
```

```
}
void image::negative(){
            writeHeader();
            i=0,j=0;
    while(i++<height){
        ifile.read((char*)row,row_size);
        j=0;
        while(j<row_size-padding){
            row[j]=255-row[j];
            j=j+1;
        }
        ofile.write((char*)row,row_size);
    }
}
void image::brightness(float intensity){
            writeHeader();
            i=0,j=0;
    while(i++<height){
            ifile.read((char*)row,row_size);
        j=0;
        while(j<row_size-padding){
              if(row[j]+intensity>255)  row[j]=255;
              else if(row[j]+intensity<0)row[j]=0;
             else row[j]+=intensity;
             j=j+1;
        }
        ofile.write((char*)row,row_size);
    }
}
void image::pixelate(float intensity,int y1,int y2,int x1,int x2){
   writeHeader();
   x1*=3, x2*=3;
```

```
    i=0,j=0;
    unsigned char avg[3]={0,0,0};
    while(i++<height){
        ifile.read((char*)row,row_size);
        j=x1;
        if(i>y1 &&i<y2){
            while(j<x2){
                for(int n=0;n<3*intensity;n++)  { avg[(j+n)%3]+=row[j+n]/intensity;}
                for(int n=0;n<3*intensity;n++) {   row[j+n]=avg[(j+n)%3];    }
                 j+=3*intensity;
                 avg[0]=avg[1]=avg[2]=0;
            }
         }

        ofile.write((char*)row,row_size);
    }
}
void image::color_boost(float intensity){
          writeHeader();
    i=0,j=0;
    while(i++<height){
        ifile.read((char*)row,row_size);
        j=0;
        while(j<row_size-padding){
            if(row[j]+row[j]/(100-intensity)>255)row[j]=255;
                                    else if(row[j]+row[j]/(100-intensity)<0) row[j]=0;
                                    else row[j]+=row[j]/(100-intensity);
            j++;
         }
        ofile.write((char*)row,row_size);
    }
}
```

```
void image::contrast(float intensity){
        writeHeader();
    i=0,j=0;
    while(i++<height){
            ifile.read((char*)row,row_size);
        j=0;
        while(j<row_size-padding){
            if(row[j]>=150)
            {           if((row[j]+(row[j]-150)/intensity)>255)
                row[j]=255;
            else  row[j]+=(row[j]-150)/intensity;
            }
            if(row[j]<150){
                if((row[j]-(150-row[j])/intensity)>0)
                    row[j]-=(150-row[j])/intensity;
                else row[j]=0;
            }
            j++;
        }
        ofile.write((char*)row,row_size);
    }
}
void image::crop(int hcut,int vcut){
    width=width-2*hcut;
    height=height-2*vcut;
    int new_padding=(4-((width*3)%4))%4;
    size=(height)*(width)*3+54+new_padding*height;
    int image_size=size-54;
    int new_row_size=width*3+new_padding;
    writeHeader();
    i=0,j=0;
    unsigned char* new_row=new unsigned char[new_row_size];
```

```
    ifile.seekg(vcut*row_size,ios::cur);
    while(i++<height){
        ifile.read((char*)row,row_size);
        j=hcut*3;
        while(j<new_row_size+hcut*3){
           if(j<row_size-padding)
                                        new_row[j-hcut*3]=row[j];
                                else    new_row[j]=0;
           j=j+1;
        }
    ofile.write((char*)new_row,new_row_size);
    }
    delete new_row;
}
void image::resize(int n){
    width=width/n;
    height=height/n;
    int new_padding=(4-((width*3)%4))%4;
    int size=(height)*(width)*3+54+new_padding*height;
    int image_size=size-54;
    int new_row_size=width*3+new_padding;
    writeHeader();
    i=0,j=0;
    unsigned char* new_row=new unsigned char[new_row_size];
    while(i++<height){
        ifile.read((char*)row,row_size);
        j=0;
        while(j<new_row_size){
         if(j<new_row_size-padding)
                                        new_row[j]=row[n*(j- (j%3) )+(j%3)];
                                else    new_row[j]=0;
             j++;
```

```
        }
        ofile.write((char*)new_row,width*3+new_padding);
        ifile.seekg((n-1)*row_size,ios::cur);
    }
    delete new_row;
}
int main(){
        system("TITLE DIPDIV's Image Editor");
        system("color 3F");
        cout<<"\t\t\tDIPDIV's Image editor\t\t\n";
        image img;    // Show developers' info
    int choice;
    cout<<"\n\t Press:\n";
    cout<<"\t 1 To convert image into grayscale\n";
    cout<<"\t 2 To convert into negative\n";
    cout<<"\t 3 To crop the image\n";
    cout<<"\t 4 To increase or decrease the brightness of image\n";
    cout<<"\t 5 To resize the image\n";
    cout<<"\t 6 To add contrast \n";
    cout<<"\t 7 To pixelate the image\n";
    cout<<"\t 8 To boost the color\n";
    cin>>choice;
    switch(choice){
        case 1:
                img.grayscale();
                break;
        case 2:
                img.negative();
                break;
        case 3:
                int h,v;
                cout<<"\n Enter the no of pixels you want to remove from left and right each \t";
```

```
        cin>>h;
        cout<<"\n Enter the no of pixels you want to remove from top and bottom each \t";
        cin>>v;
        img.crop(h,v);
        break;
case 4:
        int degree;
        cout<<"\nEnter the intensity of brightness(-255.0 to 255.0)\t";
        cin>>degree;
        img.brightness(degree);
        break;
case 5:
        int n;
        cout<<"\n By what fraction do you want to reduce image size (positive integer)\t";
        cin>>n;
        img.resize(n);
        break;
case 6:
        int intensity;
        cout<<"\n Enter the intensity(>0),the smaller the better\t";
        cin>>intensity;
        img.contrast(intensity);
        break;
case 7:
        int y1,y2,x1,x2,power;
        cout<<"\nEnter the coordinates of the part you want to pixelate\n";
        cout<<"\n Enter no of pixels from bottom: \t";
        cin>>y1;
        cout<<"\nEnter no of pixels from top: \t";
        cin>>y2;
        cout<<"\n Enter no of pixels from left: \t";
        cin>>x1;
```

```
                cout<<"\nEnter no of pixels from right: \t";
                cin>>x2;
                cout<<"\nEnter the intensity of pixelation(integer)(>1)\t";
                cin>>power;
                img.pixelate(power,y1,y2,x1,x2);
                break;
        case 8:
                int boost_power;
                cout<<"\n Enter the intensity of boost(0 to 100.0 for positive boost and >100.0 for negative
boost)\t";
                cin>>boost_power;
                img.color_boost(boost_power);
                break;
        default:
                cout<<"Invalid Choice";
                exit(0);
    }
     ifile.close();
    ofile.close();
    cout<<"\n\tSuccees!!\n";
    system("\"\"output.bmp\"\"");
    system("Pause");
    return 0;
}
```

OUTPUT

```
                    DIPDIV's Image editor

Developers of the program:

Divyanshu Kakwani,XII Sci
Diptanshu Kakwani,XII Sci

Enter image file name(a 24 bit bmp file)        input.bmp

The size of the image is :  14400.1  Kilobytes(KB)
The width of the image is :  2560  Pixels
The height of the image is :  1920  Pixels


        Press:
        1 To convert image into grayscale
        2 To convert into negative
        3 To crop the image
        4 To increase or decrease the brightness of image
        5 To resize the image
        6 To add contrast
        7 To pixelate the image
        8 To boost the color
```

Menu

Input Image (2560x1920) 14.0 MB

Pixelated Image (2560x1920) 14.0 MB

Grayscale (2560x1920) 14.0 MB

Negative (2560x1920) 14.0 MB

Bright (2560x1920) 14.0 MB

Color Boost (2560x1920) 14.0 MB

Contrast (2560x1920) 14.0 MB

Resize (853x640) 1.56 MB

Crop (2160x520) 3.21 MB

ENHANCEMENTS

1. The Image Editor currently operates only on 24 bpp BMP images. It can be extended to operate on other formats as well.
2. The program's interface can be changed to make it more user friendly.
3. There is still a plethora of effects that can be added in the program.
4. The program can be easily modified to add more than one effect at once.
5. The algorithms are self-devised and are not standard ones. They can be made more sophisticated.

BIBILIOGRAPHY

http://www.wikipedia.org/

http://stackoverflow.com/

http://www.cplusplus.com/doc/tutorial/