A Flexible Video Stream Converter

Timothy Bohr, Rainer Dömer

Technical Report CECS-08-13
October 5, 2008

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-2625, USA
(949) 824-8919

tbohr@uci.edu

A Flexible Video Stream Converter

Timothy Bohr, Rainer Dömer

Technical Report CECS-08-13
October 5, 2008

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-2625, USA
(949) 824-8919

tbohr@uci.edu

Abstract

This report describes the purpose and the use of a flexible video stream converter program which is capable of performing various image manipulation operations on YUV-encoded video streams. This program was developed in support of a larger project that strives to make a more versitile and efficient programming environment for video processing on embedded devices such as mobile phones. The described YUVconverter program assists this project by producing test video streams for evaluating the embedded applications. The converter is able to read and edit YUV video input streams with operations, such as mirroring, black and white conversion and scaling, allowing the producton of controlled test video files.

Contents

List of Figures

A Flexible Video Stream Converter

Timothy Bohr, Rainer Dömer
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-2625, USA
tbohr@uci.edu

Abstract

This report describes the purpose and the use of a flexible video stream converter program which is capable of performing various image manipulation operations on YUV-encoded video streams. This program was developed in support of a larger project that strives to make a more versitile and efficient programming environment for video processing on embedded devices such as mobile phones. The described YUVconverter program assists this project by producing test video streams for evaluating the embedded applications. The converter is able to read and edit YUV video input streams with operations, such as mirroring, black and white conversion and scaling, allowing the producton of controlled test video files.

1 Introduction

The described project in this report is part of an overall research project in the area of embedded systems design. The specific topic of his research is ”Result-Oriented System-Level Modeling for Efficient Design of Embedded Systems”, which addresses the creation and optimization of the system model itself for effective use in existing system design processes rather than the traditional method of focusing largely on simulation and synthesis from a given model [4]. Just like a high quality architectural blueprint leads to a high quality building, only a ”good” model of an embedded system will lead to a successful implementation of an embedded application. Embedded systems range from smart home appliances to video-enabled mobile phones, from real-time automotive applications to communication satellites, and from portable multi-media components to reliable medical devices [7].

Embedded computer systems are around us everyday, ranging from reliable medical devices to real-time automotive applications to video-enabled mobile phones. The desire to produce more capable phones and video devices has motivated researchers and industry-partners to develop data compression algorithms to enable the transmission of video through networks. This effort is not without technical challenges. The video-enabled devices need to handle various temporal and special video formats that exist around the world. This need has been generated because the consumer product manufacturers have built video formatting processors or codecs to meet the specifications requested by different people and governments over the years. Now that the web has brought populations closer together, researchers, developers, and manufacturers of video processing systems and products need to handle many different formats ”out of the box”.

Embedded computing systems have gained a tremendous amount of functionality and processing power and, at the same time, can now be integrated into a Multi-Processor System-on-Chip (MPSoC). The design of MPSoCs, however, faces great challenges due to the huge complexity of these systems. The goal of the overall project is to optimize the modeling of embedded systems such that targeted properties of the intended product can be quickly and precisely predicted, and the system can be efficiently implemented based on its abstract model. This includes the use of an adequate model of computation, a systematic analysis of system models using well-defined metrics, the identification of essential properties and proper abstraction levels, and the development of efficient modeling techniques and guidelines.

1.1 Need for Video Converter

For the success of the overall project, a driver application is essential. This application needs to demonstrate the feasibility and benefits of result-oriented system-level modeling techniques of the overall research. The project team is using the Advanced Video Coding (AVC) standard H.264 as the driving application. H.264, also known as MPEG-4, is an advanced standard for video compression [3]. Its free availability and high complexity makes it an ideal, industry-sized example for our system modeling.

In order to effectively evaluate the performance of firmware with different H.264 processing algorithms on embedded processors, the developed flexible and sharable converter is necessary. This converter is written in C and generates digital video streams for use as input and output data with varying degrees of complexity. Using this program, a variety of edited streams will be created having attributes which include, black and white, negative image, edited frame resolutions, and black and white pixelization.

2 A Flexible Video Converter

In the following sections, we will describe our video converter in detail.

2.1 General Program Flow

When discussing the use and implementation of the video converter program, it is first necessary to discuss the general flow of of the program. Our program converts a standard ".yuv" video stream to either an edited stream or a picture of a single frame, Figure 1. When running the program, the user specifies the input ".yuv" video stream of a 4:2:0 format which is to be edited. The file is then read into the program and whether or not the user desires, the program will output another ".yuv" 4:2:0 format stream or a ".ppm" image.

Figure 1 shows the general flow, from input to output of the YUVconverter.

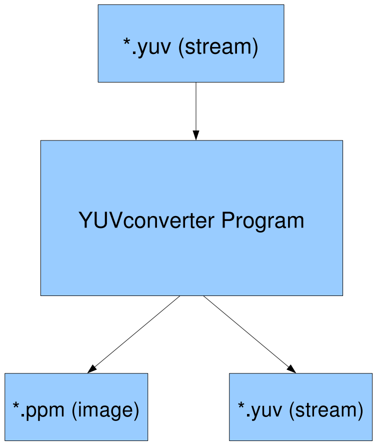


Figure 1: General Flow of YUVconverter

2.2 Internal Program Flow

A view of the internal flow of the program can be seen in Figure 2.

The YUV video stream is read in frame by frame to minimize the amount of memory required during program run time. For optimal memory allocation, we use dynamic memory functions. Once a frame is read in, it is converted and formated to regular RGB arrays having individual values for each pixel location. The conversion was done by applying a formula to corresponding YUV values [8].

Following this, the program enters a loop to apply the desired edits. These edits redefine the RGB values for each pixel. With the edits completed, the program either resizes the frame or passes these values straight to a save function.

Resolution editing is done by building a second set of arrays from original RGB values with desired output dimensions. Using RGB arrays, whether it is the resized or initial set, the program finally writes a ".ppm" file with these values, or converts back to YUV and appends current frame data to the ".yuv" stream being saved. When a video stream is being created the program will run through the described process for each frame, until all the frames desired are saved.

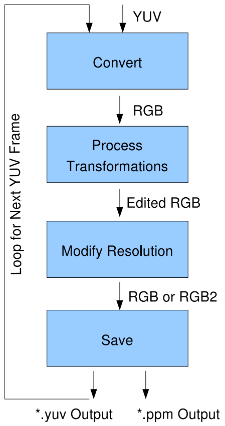


Figure 2: Internal Flow of YUVconverter

3 User Manual for YUVconverter

In the following sections, we will describe the features, usage and limitations of the YUVconverter program in detail.

3.1 Features

The converter has a set of supported features and edits. These features are executed one of two ways. The converter can either create a snapshot of any frame in the input video stream, outputting a ”.ppm” image, or the program can create a video stream from a specified initial frame to a desired final frame.

The video converter supports various image manipulation opperations that can be applied to either the output .ppm image or the .yuv stream. To illustrate the effect of these operations, we will use frame 40 of the stream ”coastguard” [1].

Figure 3 shows the original unmodified frame 40 from the ”coastguard” stream. The picture was extracted from the stream using our YUV converter with the frame option `-f`. The actual command line call for this is as follows:

```
YUV coastguard.yuv -f 40
```

The features and usage of the features are listed bellow:

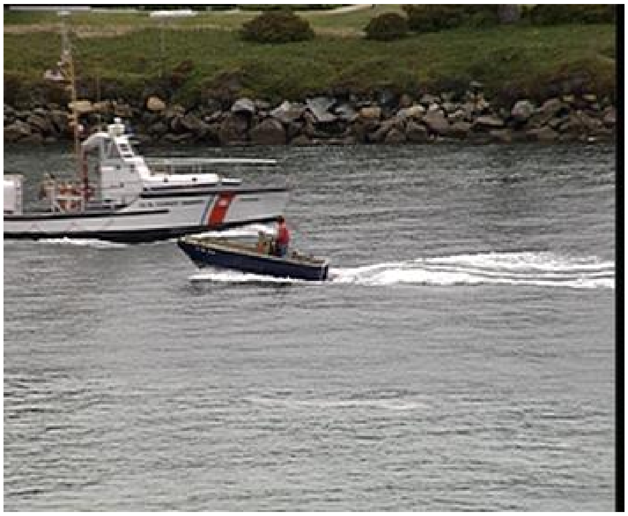


Figure 3: Original Example Frame

3.1.1 Negative conversion

The YUVconverter program can convert an input frame into a negative image using option `-n`. This operation replaces each RGB value with the difference of their value and the max color value, (255).

Figure 4 shows the negative image of the original frame in Figure 3.

3.1.2 Black and white

The YUVconverter program can convert an input frame into a black and white image using option `-bw`. This operation replaces each RGB value with the average of the RGB values at the corresponding pixel location.

Figure 5 shows the black and white image of the original frame in Figure 3, when `-bw` is applied.

3.1.3 Horizontal flip

The operation, (`-hf`), creates a horizantally mirrored image of the input frame. This is done by reassigning pixel values at corresponding possitions.

Figure 6 displayes the horizontal flip operation on Figure 3.

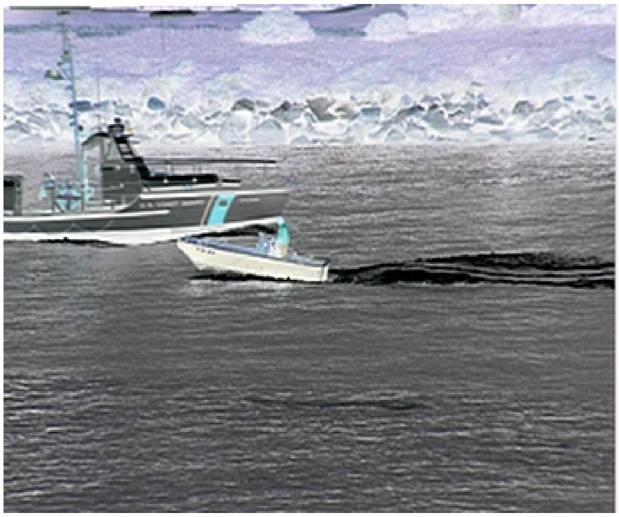


Figure 4: Example video frame after Negative Operation

3.1.4 Vertical flip

The operation, (`-vf`), is nearly the same as ”horizontal flip” except the image is mirrored vertically.

Figure 7 displayes the vertical flip operation applied on Figure 3.

3.1.5 Noise

The option, (`-noise`), is follow by a percentage which indicates the amount of black and white pixelation to the input frame at random pixel positions.

Figure 8 displayes the noise operation applied on Figure 3.

3.1.6 Frame selection

The option, (`-f`), is follow by a frame number corresponding to the initial frame of the input ”.yuv” stream desired to read in. If another number is not entered after the initial frame, the program will create a ”.ppm” image of the specified frame. If another number is entered, this will indicate the frame to which a video stream should be created. This is done by reading one frame in at a time until all frames have been and saved. Through this process it is possible to create a video stream that displayes the frames of the input stream in the opposite order, playing it ”backwards”.

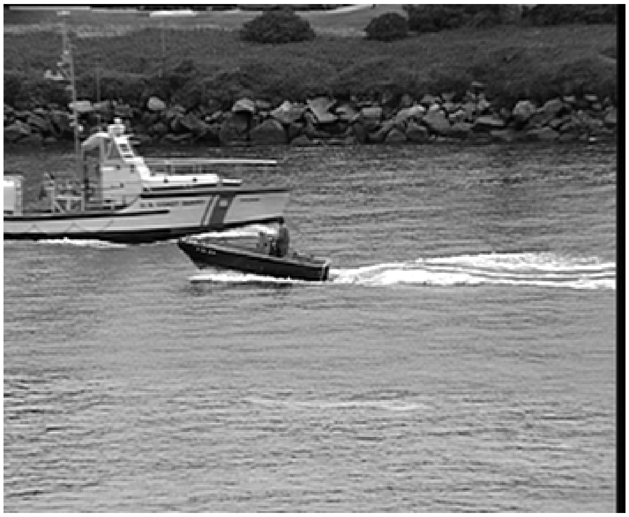


Figure 5: Example video frame after Black and White

3.1.7 Step

The option, (`-s`), allows the user to specify a number which corresponds to which frames are read into the program. A number must be entered after the option that states the ratio of input video frames per output frame. Thus, the result of entering a number greater than one is skipping over some input frames, making the video run faster. In contrast, if a number less than one is entered, input frames will be used more than once creating multiple output frames. In turn, the video will play slower.

3.1.8 Input resolution

When the option, (`-r`), is entered, it must be followed by two numbers, defining the height and width of the input stream. If this option is not entered, the input stream is assumed to have default dimensions, (352 x 288). Note that, if the resolution specified does not match the resolution of the video, the output will be comletely scattered and visually undiscernable. As a result also the output will have default dimensions.

3.1.9 Output resolution

The option, (`-r2`), when followed by height and width dimensions, defines the resolution of the output file. When this option is entered, the program enters another function which defines a frame of the desired

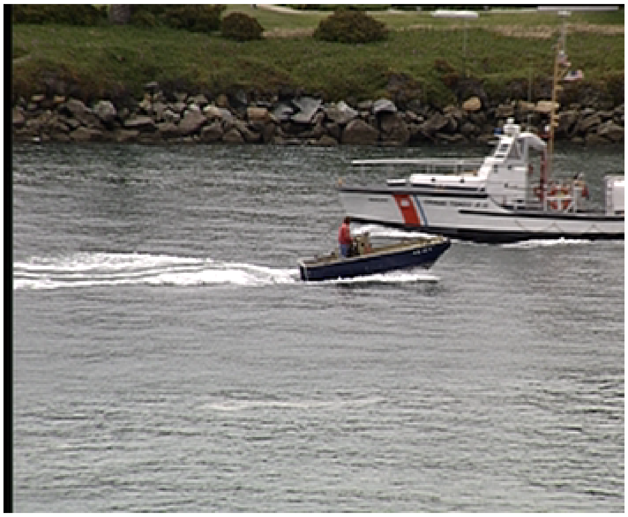


Figure 6: Example video frame after Horizonal Mirroring

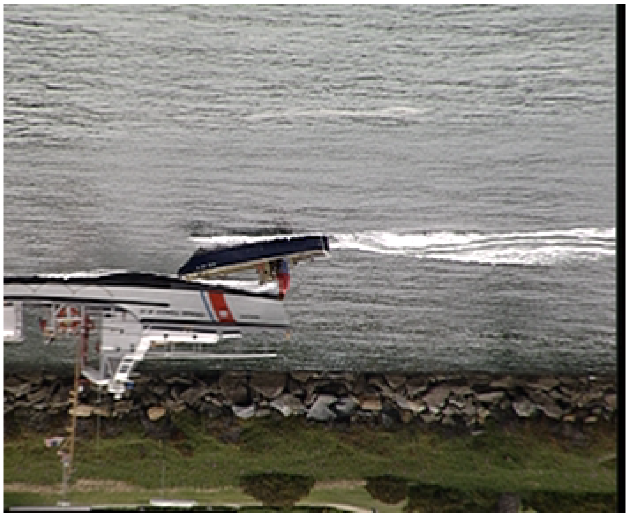


Figure 7: Example video frame after Vertical Flip

output resolution from the edited input frame. Thus, images can be scaled to any desired output size.

3.1.10 Input file name

The option, (`-i`), is entered before the input file name. By entering this option it tells the program what file to read in. It should not be assumed that when this option is used it is not necessary to enter a output file name because all that is being defined is the input file name. If this option is not entered, the program will automatically look for a base name at the second possition on the command. The input file would then constructed by appending ”_cif.yuv” to the base name.

3.1.11 Output file name

The option, (`-o`), is entered before the output file name without the file type ending. When the file is being saved, the appropriate ending is added to the output file name, (.ppm or .yuv).

3.2 Usage

Figure 9 presents all the possible options to enter on the command line. In the example call, the result would be a stream created from file, ”coastguard_cif.yuv”, running backwards from the 20th to the 14th frame, skipping every other input frame. The output would also be enlarged to 500x500 pixels and be the negative of the input.

3.3 Limitations

The current implementation of the YUVconverter has a few limitations to its uses. First, the program does not have an imbedded ”.ppm” to ”.jpg” converter. This makes viewing output frames more difficult because applications which display ”.ppm” images are less common than those which can handle ”.jpg” files.

Second, at this point the converter can only handle 4:2:0 YUV format type, which negelects to address 4:2:2 and 4:4:4 file formats.

Third, in order read in a YUV file correctly it is necessary to know the resolution of the video stream.

4 Summary

The YUVconverter program reads in a 4:2:0 type YUV video stream and produces various outputs. The program is able to create a negative, black and white, both horizontally and virtically fliped, and noisy image frames. Some other possible edits include, resolution adjustment and frame skiping and addition.

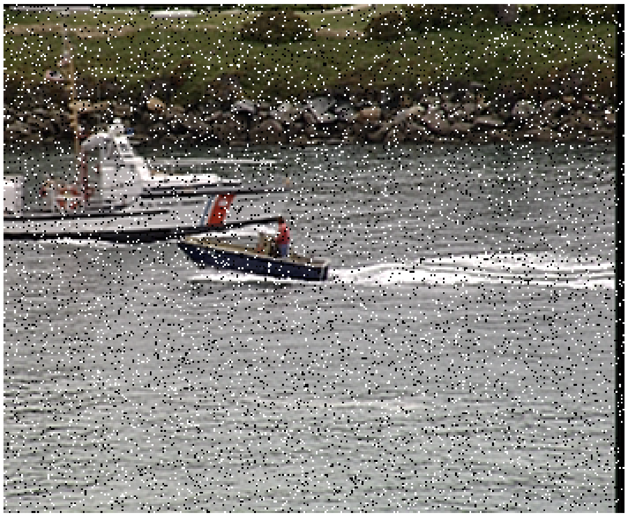
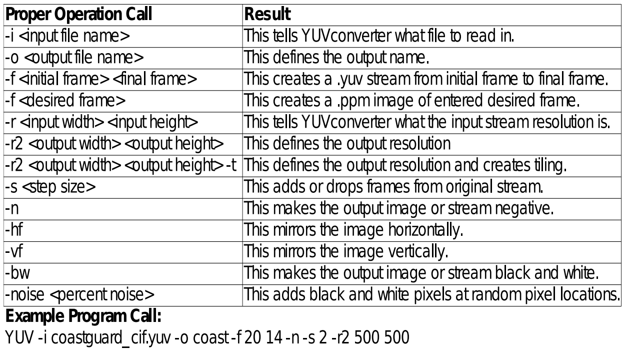


Figure 8: Example video frame after Noise Operation



Proper Operation Call	Result
-i <input file name>	This tells YUVconverter what file to read in.
-o <output file name>	This defines the output name.
-f <initial frame> <final frame>	This creates a .yuv stream from initial frame to final frame.
-f <desired frame>	This creates a .ppm image of entered desired frame.
-r <input width> <input height>	This tells YUVconverter what the input stream resolution is.
-r2 <output width> <output height>	This defines the output resolution
-r2 <output width> <output height> -t	This defines the output resolution and creates tiling.
-s <step size>	This adds or drops frames from original stream.
-n	This makes the output image or stream negative.
-hf	This mirrors the image horizontally.
-vf	This mirrors the image vertically.
-bw	This makes the output image or stream black and white.
-noise <percent noise>	This adds black and white pixels at random pixel locations.

Example Program Call:

```
YUV -i coastguard_cif.yuv -o coast -f 20 14 -n -s 2 -r2 500 500
```

Figure 9: Operations Chart

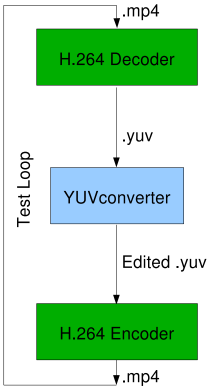


Figure 10: Test Loop

4.1 Future Work

With the converter completed, testing on the H.264 encoder and decoder are now possible. Test loops are to be done, checking the efficiency of the coding process. Cycles using varying stream complexities will be used to find the best implementation for the chips.

The test loop that will be conducted on the H.264 decoder can be seen in Figure 10 using a test ".mp4" type video, the designed H.264 decoder will convert the file to a ".yuv" stream. This stream will then be run in i the YUVconverter, applying desired edits and outputting another ".yuv" stream. Following this, the edited file will then be encoded by the H.264 encoder completing one test loop.

We plan to initially conduct tests on a program that simulates chip function, allowing cheap and efficient design alteration. Data will be collected and plotted on a chart displaying the relationship between effort and performance.

5 Acknowledgments

This work has ben supported in part by a Summer Undergraduate Research Program (SURP)/Undergraduate Research Opportunities Program (UROP) Fellowship over the Summer quarter 2008. Based on the research proposal titled *"Research and Developement of a Flexible Converter for Digital Processing"* (proposal code 02843s1), the SURP fellowship was awarded to Timothy Bohr under the supervision of Rainer Doemer. The authors thank the UROP Office at UC Irvine for this support.

References

[1] CIPR SIF Sequences. `http://www.cipr.rpi.edu/resource/sequences/sif.html`.

[2] P. J. Deitel and H. M. Deitel. *C – How to Program.* Prentice Hall, Upper Saddle River, New Jersey, 2007.

[3] H.264/MPEG–4 AVC. `http://en.wikipedia.org/wiki/H.264`.

[4] EECS Assistant Professor Receives CAREER Award. http://www.eng.uci.edu/node/1387.

[5] A. Kelly and I. Pohl. *A Book on C: Programming in C*. Addison – Wesley, 1998.

[6] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Upper Saddle River, New Jersey, 1988.

[7] Result–Oriented System–Level Modeling for Efficient Design on Embedded Systems. http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0747523, 2008.

[8] Converting Between YUV and RGB. http://msdn.microsoft.com/en-us/library/ms893078.aspx.

A Appendix

The following Section A.1 lists the source code of the video converter described in this report. The listed source file follows ANSI-C coding guidelines and should be portable to any ANSI-C compliant programming environment. This code has been successfully compiled and used on Linux Fedora Core 4 and Mac OS.

A.1 Source Code for YUVconverter

The following listing shows the source code for YUVconverter.

```
/*YUVconverter, version 1.0*/
/*Author: Timothy Bohr*/
/*09/16/2008*/
/*Copyright Timothy Bohr*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <assert.h>

/* global definitions */
#define WIDTH 352 /*default width*/
#define HEIGHT 288 /*default height*/

/*frame structure definitions*/
typedef struct rgbframe {
    unsigned char *R;
    unsigned char *G;
    unsigned char *B;
    }RGBframe;

typedef struct yuvframe {
    unsigned char *Y;
    unsigned char *U;
    unsigned char *V;
    }YUVframe;

/*read frame from a file*/
int ReadFrame(YUVframe *YUV, const char *fname, int *iframe, int fframe, unsigned int width,
unsigned int height, float s);

/*convert YUV to RGB*/
int YUVconverter(RGBframe *RGB, YUVframe *YUV, unsigned int width, unsigned int height);

/*convert RGB back to YUV*/
int YUVreconverter(RGBframe *RGB, YUVframe *YUV, unsigned int width, unsigned int height);

/*save a converted frame*/
int SaveFrame(RGBframe *RGB, const char *fname, unsigned int width, unsigned int height);

/*save a yuv stream*/
```

```
int SaveYUV(RGBframe *RGB, RGBframe *RGB2, YUVframe *YUV, YUVframe *YUV2, const char *fname,
                const char *fin , unsigned int width, unsigned int height, int width2,
                int height2 , int *iframe , unsigned int fframe , int n, int h, int v, int bw,
                float s , int noise , int degree , int tile );

/*create nagative image*/
void Negative(RGBframe *RGB, unsigned int width, unsigned int height);

/* flip image horizontally */
void HFlip(RGBframe *RGB, unsigned int width, unsigned int height);

/* flip image vertically */
void VFlip(RGBframe *RGB, unsigned int width, unsigned int height);

/*create black and white*/
void BW(RGBframe *RGB, unsigned int width, unsigned int height);

/*add noise to frames*/
void AddNoise(RGBframe *RGB, int degree, int width, int height);

/*adjust to output resolution*/
void ADJres(RGBframe *RGB, RGBframe *RGB2, int width, int height, int width2, int height2);

/*create tiling in desired output resolution*/
void Tile(RGBframe *RGB, RGBframe *RGB2, int width, int height, int width2, int height2);

/*print possible options for program*/
void printoptions(void);


/*entering main function*/
int main( int argc, char *argv[])
{
  /*defining local variables*/

  char *fin = NULL; /*input file name*/
  char *fout = NULL; /*output file name*/
  int E = 0; /*possible error return*/
  int iframe = 0, fframe = (-1); /*frame numbers*/
  unsigned int height = HEIGHT, width = WIDTH; /*dimensions*/
  unsigned int height2, width2; /*dimensions of output stream*/
  int x = 0; /*parameter*/
  int n = 0; /*flag for negative*/
  int h = 0; /*flag for horizontal flip*/
  int v = 0; /*flag for vertical flip*/
  float s = 1; /*flag for skipping or multiplying frames*/
  int bw = 0; /*flag for black and white*/
  int noise = 0; /*flag for adding noise*/
  int degree; /*the percent noise*/
  int tile = 0; /*flag for tiling*/
  unsigned int size; /*contains size of pointers*/

```

```
/*declaring structures*/
RGBframe RGB = {NULL, NULL, NULL};
YUVframe YUV = {NULL, NULL, NULL};
RGBframe RGB2 = {NULL, NULL, NULL};
YUVframe YUV2 = {NULL, NULL, NULL};

/*declaring pointers*/
RGBframe * RGBptr = NULL;
YUVframe * YUVptr = NULL;
RGBframe * RGB2ptr = NULL;
YUVframe * YUV2ptr = NULL;


/*entering while loop to check options entered*/
while(x < argc)
 {if(0 == strcmp(&argv[x][0], ”–i”))
     {if(x < argc - 1)
         {fin = (char *)malloc(sizeof(char) * (strlen(&argv[x+1][0]) + 1));
         strcpy( fin, argv[x+1]);
         }/*fi*/
     else
         {printf(”Missing␣argument␣for␣input␣name!”);
         return 5;
         }/*esle*/
     x += 2;
     continue;
     }/*fi*/
 if(0 == strcmp(&argv[x][0], ”–o”))
     {if(x < argc - 1)
         {fout = (char *)malloc(sizeof(char) * (strlen(&argv[x+1][0]) + strlen(”.ppm”) + 1));
         strcpy( fout, argv[x+1]);
         }/*fi*/
     else
         {printf(”Missing␣argument␣for␣output␣name!”);
         return 5;
         }/*esle*/
     x += 2;
     continue;
     }/*fi*/
 if(0 == strcmp(argv[x], ”–f”))
     {if(argc < (x + 1) || 0 == isdigit(argv[x+1][0]))
         {printf(”\nDesired␣frame␣not␣entered!\n”);
         printoptions();
         return 5;
         }/*fi*/
     else
         {iframe = atoi(argv[x+1]);
         }/*esle*/
     if(argc > (x + 2) && 0 != isdigit(argv[x+2][0]))
         {fframe = atoi(argv[x+2]);
         x++;
         }/*fi*/
```

```
        x += 2;
        continue;
        }/*fi*/
    if(0 == strcmp(&argv[x][0], ”–r”))
        {if(argc < (x + 1) || 0 == isdigit(argv[x+1][0]))
            {printf(”\nInput␣width␣was␣not␣entered!\n”);
            printoptions();
            return 5;
            }/*fi*/
        else
            {width = atoi(argv[x+1]);
            }/*esle*/
        if(argc < (x + 2) || 0 == isdigit(argv[x+2][0]))
            {printf(”\nInput␣height␣was␣not␣entered!\n”);
            printoptions();
            return 5;
            }/*fi*/
        else
            {height = atoi(argv[x+2]);
            }/*esle*/
        x += 3;
        continue;
        }/*fi*/
    if(0 == strcmp(&argv[x][0], ”–r2”))
        {if(argc < (x + 1) || 0 == isdigit(argv[x+1][0]))
            {printf(”\nOutput␣width␣was␣not␣entered!\n”);
            printoptions();
            return 5;
            }/*fi*/
        else
            {width2 = atoi(argv[x+1]);
            }/*esle*/
        if(argc < (x + 2) || 0 == isdigit(argv[x+2][0]))
            {printf(”\nOutput␣height␣was␣not␣entered!\n”);
            printoptions();
            return 5;
            }/*fi*/
        else
            {height2 = atoi(argv[x+2]);
            }/*esle*/
        if((x + 3) < argc && 0 == strcmp(&argv[x + 3][0], ”–t”))
            {tile = 1;
            x++;
            }/*fi*/
        x += 3;
        continue;
        }/*fi*/
    if(0 == strcmp(&argv[x][0], ”–s”))
        {if(argc < (x + 1))
            {printf(”Missing␣step␣size␣entry!”);
            return 5;
            }/*fi*/
```

```
          s = atof(argv[x+1]);
          x += 2;
          continue;
          }/*fi*/
      if(0 == strcmp(&argv[x][0], ”–n”))
          {n = 1;
          x++;
          continue;
          }/*fi*/
      if(0 == strcmp(&argv[x][0], ”–hf”))
          {h = 1;
          x++;
          continue;
          }/*fi*/
      if(0 == strcmp(&argv[x][0], ”–vf”))
          {v = 1;
          x++;
          continue;
          }/*fi*/
      if(0 == strcmp(&argv[x][0], ”–bw”))
          {bw = 1;
          x++;
          continue;
          }/*fi*/
      if(0 == strcmp(&argv[x][0], ”–noise”))
          {if(argc < (x + 1) || 0 == isdigit(argv[x+1][0]))
              {printf(”Missing␣degree␣noise!\n”);
              return 5;
              }/*fi*/
          degree = atoi(argv[x+1]);
          noise = 1;
          x += 2;
          continue;
          }/*fi*/
      if(0 == strcmp(&argv[x][0], ”–h”))
          { printoptions();
          return 0;
          }/*fi*/
      x++;
      }/*elihw*/

     /*allocate memory*/
     size = width * height * sizeof(unsigned char);

    YUV.Y = (unsigned char *)malloc(size);
    YUV.U = (unsigned char *)malloc(size/4);
    YUV.V = (unsigned char *)malloc(size/4);

    RGB.R = (unsigned char *)malloc(size);
    RGB.G = (unsigned char *)malloc(size);
    RGB.B = (unsigned char *)malloc(size);

```

```
/*checking for error allocating memory*/
if(!RGB.R || !RGB.G || !RGB.B || !YUV.Y || !YUV.U || !YUV.V)
 {printf(”Out␣of␣memory!”);
 return 20;
 }/*fi*/

if(width2 != 0 || height2 != 0)
 {
 /*Redifine the size necessary to allocate*/
 size = width2 * height2 * sizeof(unsigned char);

 /*allocating memory for resizing*/
 YUV2.Y = (unsigned char *)malloc(size);
 YUV2.U = (unsigned char *)malloc(size/4);
 YUV2.V = (unsigned char *)malloc(size/4);

 RGB2.R = (unsigned char *)malloc(size);
 RGB2.G = (unsigned char *)malloc(size);
 RGB2.B = (unsigned char *)malloc(size);

 /*checking for error allocating memory*/
 if(!RGB2.R || !RGB2.G || !RGB2.B || !YUV2.Y || !YUV2.U || !YUV2.V)
     {printf(”Out␣of␣memory!”);
     return 20;
     }/*fi*/
 }/*fi*/

/*creating pointers to structures*/
RGBptr = &RGB;
YUVptr = &YUV;
RGB2ptr = &RGB2;
YUV2ptr = &YUV2;

/*checking for missing file names*/
if(argc < 2)
 {printf(”Missing␣base␣name␣argument!”);
 return 20;
 }/*fi*/

/*defining File names if base name entered*/
if(fin == NULL)
 {fin = (char *)malloc(sizeof(char) * (strlen(&argv[1][0]) + strlen(”_cif.yuv”) + 1));
 strcpy(fin, argv[1]);
 strcat( fin, ”_cif.yuv”);
 }/*fi*/
if(fout == NULL)
 {fout = (char *)malloc(sizeof(char) * (strlen(&argv[1][0]) + strlen(”.ppm”) + 1));
 strcpy(fout, argv[1]);
 }/*fi*/


/*creating a YUV stream*/
```

```
if(fframe != (-1))
 {
 /*print parameters*/
 printf(”Initial␣frame:␣%d\n”, iframe);
 printf(”Final␣frame:␣%d\n”, fframe);
 /*printing for originally sized frame*/
 if(width2 != 0 || height2 != 0)
     {printf(”Width2:␣%d\n”, width2);
     printf(”Height2:␣%d\n”, height2);
     }/*fi*/
 /*printing for resized frame*/
 else
     {printf(”Width:␣%d\n”, width);
     printf(”Height:␣%d\n”, height);
     }/*esle*/

 /*appending proper ending to input string*/
 strcat( fout, ”.yuv”);

 SaveYUV(RGBptr, RGB2ptr, YUVptr, YUV2ptr, fout, fin, width, height, width2, height2,
          &iframe, fframe, n, h, v, bw, s, noise, degree, tile);
 }/*fi*/

/*creating a sigle PPM image*/
else
 {
 /*defining appropriate ending of .ppm*/
 strcat( fout, ”.ppm”);

 /*reading in frame and checking for error*/
 E = ReadFrame(YUVptr, fin, &iframe, fframe, width, height, s);
 if (E != 0)
     {return 10;
     }/*fi*/

 /*Printing parameters*/
 printf(”Frame:␣%d\n”, iframe);

 if(width2 != 0 || height2 != 0)
     {printf(”Width2:␣%d\n”, width2);
     printf(”Height2:␣%d\n”, height2);
     }/*fi*/
 else
     {printf(”Width:␣%d\n”, width);
     printf(”Height:␣%d\n”, height);
     }/*esle*/

 /*converting YUV -> RGB*/
 YUVconverter(RGBptr, YUVptr, width, height);

 /*applying desired edits*/
     if(n == 1)
```

```
             {Negative(RGBptr, width, height);
             }/*fi*/
         if(h == 1)
             {HFlip(RGBptr, width, height);
             }/*fi*/
         if(v == 1)
             {VFlip(RGBptr, width, height);
             }/*fi*/
         if(bw == 1)
             {BW(RGBptr, width, height);
             }/*fi*/
         if(noise == 1)
             {AddNoise(RGBptr, degree, width, height);
             }/*fi*/

     /*saving RGB to ppm and checking for error*/
     if(width2 != 0 || height2 != 0)
         {if(tile == 1)
             {Tile(RGBptr, RGB2ptr, width, height, width2, height2);
             }/*fi*/
         else
             {ADJres(RGBptr, RGB2ptr, width, height, width2, height2);
             }/*esle*/
         E = SaveFrame(RGB2ptr, fout, width2, height2);
         if (E != 0)
             {return 10;
             }/*fi*/
         }/*fi*/
     else
         {E = SaveFrame(RGBptr, fout, width, height);
         if (E != 0)
             {return 10;
             }/*fi*/
         }/*esle*/

     }/*esle*/

    /*freeing up memory*/
    free(YUV.Y);
    free(YUV.U);
    free(YUV.V);
    free(RGB.R);
    free(RGB.G);
    free(RGB.B);

    YUV.Y = NULL;
    YUV.U = NULL;
    YUV.V = NULL;
    RGB.R = NULL;
    RGB.G = NULL;
    RGB.B = NULL;

    free(fin);
    free(fout);

    if(width2 != 0 || height2 != 0)
     {free(YUV2.Y);
     free(YUV2.U);
     free(YUV2.V);
     free(RGB2.R);
     free(RGB2.G);
     free(RGB2.B);

     YUV2.Y = NULL;
     YUV2.U = NULL;
     YUV2.V = NULL;
     RGB2.R = NULL;
     RGB2.G = NULL;
     RGB2.B = NULL;
     }/*fi*/

    printf("Conversion␣successfully␣done!\n");

    /*terminating program*/
    return 0;
}

int SaveFrame(RGBframe *RGB, const char *fname, unsigned int width, unsigned int height)
{
    /*defining local variables*/
    FILE *File;
    int i,j;


     /*opening stream*/
     File = fopen(fname, "w");

     /*checking for possible error*/
     if (!File)
        {printf("\nCan␣not␣open␣file␣\"%s\"␣for␣writing!\n", fname);
         return 1;
        }/*fi*/

     /*writing file information*/
     fprintf( File, "P6\n");
     fprintf( File, "%di␣%d\n", width, height);
     fprintf( File, "255\n");


     /*allocating pixel values to stream*/
     for( j = 0; j < height; j++)
        {for( i = 0; i < width; i++)
            {
         fputc(RGB->R[i + width * j], File);
```

```
          fputc(RGB->G[i + width * j], File);
          fputc(RGB->B[i + width * j], File);
              }/*rof*/
         }/*rof*/


     /*checking for error*/
     if (ferror(File))
         {
          printf("\nFile␣error␣while␣writing␣to␣file!\n");
          return 2;
         }/*fi*/

     /*closing stream and terminating function*/
     fclose( File );
     printf("%s␣was␣saved␣successfully.␣\n", fname);

    /*terminating read*/
    return 0;
}

int SaveYUV(RGBframe *RGB, RGBframe *RGB2, YUVframe *YUV, YUVframe *YUV2, const char *fname,
               const char *fin, unsigned int width, unsigned int height, int width2, int height2,
               int *iframe, unsigned int fframe, int n, int h, int v, int bw, float s, int noise,
               int degree, int tile)
{
    /*defining local variables*/
    FILE *File;
    int pixel;
    int E = 0; /*error report*/
    int cut = 0; /*flag for break loop*/

    /*opening stream*/
    File = fopen(fname, "w");

    /*checking for possible error*/
    if (!File)
     {
      printf("\nCan␣not␣open␣file␣\"%s\"␣for␣writing!\n", fname);
      return 1;
     }/*fi*/

    while(cut != 1)
     {if(*iframe == fframe)
          {cut = 1;
          }/*fi*/

     E= ReadFrame(YUV, fin, iframe, fframe, width, height, s);
     if(E != 0)
         {return 1;
         }/*fi*/

```

```
/*converting YUV -> RGB*/
YUVconverter(RGB, YUV, width, height);

/*applying desired edits*/
if(n == 1)
   {Negative(RGB, width, height);
   }/*fi*/
if(h == 1)
   {HFlip(RGB, width, height);
   }/*fi*/
if(v == 1)
    {VFlip(RGB, width, height);
    }/*fi*/
if(bw == 1)
    {BW(RGB, width, height);
    }/*fi*/
if(noise == 1)
    {AddNoise(RGB, degree, width, height);
    }/*fi*/

/*incorporating resizing*/
if(width2 != 0 || height2 != 0)
    {if(tile == 1)
        {
        /*define RGB2 from RGB in tiles*/
        Tile(RGB, RGB2, width, height, width2, height2);
        }/*fi*/
    else
        {
        /*define RGB2 from RGB for resizing*/
        ADJres(RGB, RGB2, width, height, width2, height2);
        }/*esle*/

    YUVreconverter(RGB2, YUV2, width2, height2);

    /*allocating pixel values to stream*/

    for( pixel = 0; pixel < height2 * width2; pixel++)
        {fputc(YUV2->Y[pixel], File);
        }/*rof*/
    for( pixel = 0; pixel < (height2 / 2) * (width2 / 2); pixel++)
        {fputc(YUV2->U[pixel], File);
        }/*rof*/
    for( pixel = 0; pixel < (height2 / 2) * (width2 / 2); pixel++)
        {fputc(YUV2->V[pixel], File);
        }/*rof*/

    }/*fi*/

else
    {
    /*reconvert edited RGB -> YUV*/
```

```
            YUVreconverter(RGB, YUV, width, height);

            /*allocating pixel values to stream*/

            for( pixel = 0; pixel < height * width; pixel++)
                 {fputc(YUV->Y[pixel], File);
                 }/*rof*/
            for( pixel = 0; pixel < height/2*width/2; pixel++)
                 {fputc(YUV->U[pixel], File);
                 }/*rof*/
            for( pixel = 0; pixel < height/2*width/2; pixel++)
                 {fputc(YUV->V[pixel], File);
                 }/*rof*/

            }/*esle*/
      }/*elihw*/


    /*checking for error*/
    if (ferror(File))
      {
       printf("\nFile␣error␣while␣writing␣to␣file!\n");
       return 2;
      }/*fi*/

    /*closing stream and terminating function*/
    fclose( File );
    printf("%s␣was␣saved␣successfully.␣\n", fname);

    return 0;
}

int ReadFrame(YUVframe *YUV, const char *fname, int *iframe, int fframe, unsigned int width,
                    unsigned int height, float s)
{
    /*defining local variables*/
    FILE *File;
    int pixel;
    static float step;
    static int count = 0;

    /*opening file stream*/
    File = fopen(fname, "r");

    /*checking error*/
    if (!File)
         {
      printf("\nCan␣not␣open␣file␣\"%s\"␣for␣reading!\n", fname);
      return 1;
         }/*fi*/

    printf("step␣=␣%f,␣s␣=␣%f␣fframe␣=␣%d␣and␣iframe␣=␣%d␣in␣read\n", step, s, fframe, *iframe );
```

```

/*define YUV arrays*/
/*find desired frame*/
if(iframe > 0)
 {fseek(File, 1.5 * (*iframe) * width * height, SEEK_SET);
 }/*fi*/

for( pixel = 0; pixel < height * width; pixel++)
    {
 YUV->Y[pixel] = fgetc(File);
    }/*rof*/
 assert(pixel == (height * width));
for( pixel = 0; pixel < height/2*width/2; pixel++)
    {
 YUV->U[pixel] = fgetc(File);
    }/*rof*/
 assert(pixel == (height * width / 4));
for( pixel = 0; pixel < height/2*width/2; pixel++)
    {
 YUV->V[pixel] = fgetc(File);
    }/*rof*/
 assert(pixel == (height * width / 4));

/*checking for error*/
if (ferror(File))
    {
 printf("\nFile␣error␣while␣reading␣from␣file!\n");
 return 2;
    }/*fi*/

printf("%s␣was␣read␣successfully!\n", fname);

if(count == 0)
 {step = *iframe;
 }/*fi*/

/*dealing with following frame determination*/
if(step > (fframe - s) && step < (fframe + s))
 {*iframe = fframe;
 count = (-1);
 }/*fi*/

if(fframe > *iframe && fframe != -1)
 {step += s;
 }/*fi*/

if(fframe < *iframe && fframe != -1)
 {step -= s;
 }/*fi*/

if(count != (-1))
 {*iframe = step + 0.5;
```

```
     }/*fi*/

    count++;

    /*closing stream and terminating*/
    fclose(File);

    return 0;
}

int YUVconverter(RGBframe *RGB, YUVframe *YUV, unsigned int width, unsigned int height)
{
    /*defining local variables*/

    int C, D, E; /*variables in conversion formulae*/
    int count = 0; /*pixel number in RGB and Y pointers*/
    int r, g, b; /*temporary variables*/
    int reset = 1; /*flag for recounting a row for U and V pointers*/
    int slow_count = 0; /*counter to establish UV pixel*/
    int width_count = 0; /*counter for reset*/

    while(count < height * width)
     {if(width_count == width)
          {reset += 1;
          width_count = 0;
          }/*fi*/

     if(reset == 2)
          {slow_count = slow_count - width;
          reset = 0;
          }

     assert((slow_count/2) < (width * height/4));

     C = (int)YUV->Y[count] - 16;
     D = (int)YUV->U[slow_count/2] - 128;
     E = (int)YUV->V[slow_count/2] - 128;


     /*defining intermediary variables*/
     r = (298 * C + 409 * E + 128) >> 8;
     g = (298 * C - 100 * D - 208 * E + 128) >> 8;
     b = (298 * C + 516 * D + 128) >> 8;

     /*passing intermediary values to global pointers*/
     RGB->R[count] = (unsigned char)r;
     RGB->G[count] = (unsigned char)g;
     RGB->B[count] = (unsigned char)b;

     /*checking for byte overflow and if so redefining to either 0 or 255*/
     if(r < 0)
        {RGB->R[count] = 0;}
```

```
      if(r > 255)
          {RGB->R[count] = 255;}
      if(g < 0)
          {RGB->G[count] = 0;}
      if(g > 255)
          {RGB->G[count] = 255;}
      if(b < 0)
          {RGB->B[count] = 0;}
      if(b > 255)
          {RGB->B[count] = 255;}

      /*incrementing counters*/
      slow_count++;
      count++;
      width_count++;

      }/*elihw*/

      assert(count == (width * height));

    /*terminating function*/
    return 0;
}

int YUVreconverter(RGBframe *RGB, YUVframe *YUV, unsigned int width, unsigned int height)
{
    /*defining local variables*/
    int i, j;
    int y, u, v;
    int count = 0;

    /*going through for loop to convert each pixel*/
    for( j = 0; j < height; j++)
        {for( i = 0; i < width; i++)
          {
     /*defining intermediary y*/
      y = (( 66 * RGB->R[i + width * j] + 129 * RGB->G[i + width * j] +
           25 * RGB->B[i + width * j] + 128) >> 8) + 16;

      /*passing intermediary values to global pointers*/
      YUV->Y[count] = (unsigned char)y;

      /*checking for byte overflow and if so redefining to either 0 or 255*/
      if(y < 0)
          {YUV->Y[count] = 0;
          }/*fi*/
      if(y > 255)
          {YUV->Y[count] = 255;
          }/*fi*/
      count++;
          }/*rof*/
       }/*rof*/
```

```

   /*reinitializing counter*/
   count = 0;

   /*going through for loop to convert each pixel*/
   for( j = 0; j < height; j+=2)
      {for( i = 0; i < width; i+=2)
        {
    /*defining intermediary u and v*/
    u = (( (-38) * RGB->R[i + width * j] - 74 * RGB->G[i + width * j] +
        112 * RGB->B[i + width * j] + 128) >> 8) + 128;
    v = (( 112 * RGB->R[i + width * j] - 94 * RGB->G[i + width * j] -
        18 * RGB->B[i + width * j] + 128) >> 8) + 128;

    /*passing intermediary values to global pointers*/
    YUV->U[count] = (unsigned char)u;
    YUV->V[count] = (unsigned char)v;

    /*checking for byte overflow and if so redefining to either 0 or 255*/
    if(u < 0)
       {YUV->U[count] = 0;}
    if(u > 255)
       {YUV->U[count] = 255;}
    if(v < 0)
       {YUV->V[count] = 0;}
    if(v > 255)
       {YUV->V[count] = 255;}

    count++;
       }/*rof*/
     }/*rof*/

   printf("reconversion done!\n");

   /*terminating function*/
   return 0;
}

void ADJres(RGBframe *RGB, RGBframe *RGB2, int width, int height, int width2, int height2)
{
   int i, j;
   float scalex = (float)width2 / (float)width;
   float scaley = (float)height2 / (float)height;

   for (j=0; j < height2; j++)
    {for (i=0; i < width2; i++)
        {assert((i + width2 * j) < height2 * width2);
        assert(((int)(i/scalex) + width * (int)(j/scaley)) < width * height);
        assert((i < width2) && (j < height2));

        RGB2->R[i + width2 * j] = RGB->R[(int)(i/scalex) + width * (int)(j/scaley)];
        RGB2->G[i + width2 * j] = RGB->G[(int)(i/scalex) + width * (int)(j/scaley)];
        RGB2->B[i + width2 * j] = RGB->B[(int)(i/scalex) + width * (int)(j/scaley)];
        }/*rof*/
    }/*rof*/
}

void Tile(RGBframe *RGB, RGBframe *RGB2, int width, int height, int width2, int height2)
{
   int i, j, x, y;
   assert(width > 0);
   assert(height > 0);

   for (j=0, y=0; j < height2; j++, y++)
    {for (i=0, x=0; i < width2; i++, x++)
        {RGB2->R[i + width2 * j] = RGB->R[x + width * y];
        RGB2->G[i + width2 * j] = RGB->G[x + width * y];
        RGB2->B[i + width2 * j] = RGB->B[x + width * y];

        if(x + 1 == width)
            {x = 0;
            }/*fi*/
        if(y + 1 == height)
            {y = 0;
            }/*fi*/

        }/*rof*/
    }/*rof*/
}

/* reverse image color */
void Negative(RGBframe *RGB, unsigned int width, unsigned int height)
{
   /*defining local variables*/
   int i=0, j=0;

   /*redefining pixels*/
   for (i=0; i < width; i++)
    {for (j=0; j < height; j++)
        {RGB->R[i + width * j] = 255 - RGB->R[i + width * j];
        RGB->G[i + width * j] = 255 - RGB->G[i + width * j];
        RGB->B[i + width * j] = 255 - RGB->B[i + width * j];
        }/*rof*/
    }/*rof*/

   /*displaying completion*/
   printf("\"Negative\"␣is␣done!\n");
}

/* flip image horizontally */
void HFlip(RGBframe *RGB, unsigned int width, unsigned int height)
{
   /*defining local variables*/
   int i, j, temp;
```

```

    /*redefining pixels*/
    for (j=0; j < height; j++)
     {for (i=0; i < width/2; i++)
         {temp = RGB->R[i + width * j];
         RGB->R[i + width * j] = RGB->R[(width - i - 1) + width * j];
         RGB->R[(width - i - 1) + width * j] = temp;

         temp = RGB->G[i + width * j];
         RGB->G[i + width * j] = RGB->G[(width - i - 1) + width * j];
         RGB->G[(width - i - 1) + width * j] = temp;

         temp = RGB->B[i + width * j];
         RGB->B[i + width * j] = RGB->B[(width - i - 1) + width * j];
         RGB->B[(width - i - 1) + width * j] = temp;
         }/*rof*/
     }/*rof*/

    /*desplaying completion*/
    printf(”\”Horizontal␣Flip\”␣is␣done!\n”);
}

void VFlip(RGBframe *RGB, unsigned int width, unsigned int height)
{
    /*defining local variables*/
    int i, j, temp;

    /*redifining pixels*/
    for (j=0; j < height/2; j++)
     {for (i=0; i < width; i++)
        {temp = RGB->R[i + width * j];
        RGB->R[i + width * j] = RGB->R[i + width * (height - j - 1)];
        RGB->R[i + width * (height - j - 1)] = temp;

        temp = RGB->G[i + width * j];
        RGB->G[i + width * j] = RGB->G[i + width * (height - j - 1)];
        RGB->G[i + width * (height - j - 1)] = temp;

        temp = RGB->B[i + width * j];
        RGB->B[i + width * j] = RGB->B[i + width * (height - j - 1)];
        RGB->B[i + width * (height - j - 1)] = temp;
        }/*rof*/
     }/*rof*/

    /*displaying completion*/
    printf(”\”Vertical␣Flip\”␣is␣done!\n”);
}

void BW(RGBframe *RGB, unsigned int width, unsigned int height)
{
    /*defining local variables*/
    int i, j, avg;

    /*redefining pixels*/
    for (j=0; j < height; j++)
     {for (i=0; i < width; i++)
         {avg = (RGB->R[i + width * j] + RGB->G[i + width * j] + RGB->G[i + width * j]) / 3;

         RGB->R[i + width * j] = avg;
         RGB->G[i + width * j] = avg;
         RGB->B[i + width * j] = avg;
         }/*rof*/
     }/*rof*/

    /*displaying completion*/
    printf("\"Black␣&␣White\"␣is␣done!\n");
}

void AddNoise(RGBframe *RGB, int degree, int width, int height)
{
    int count;
    int pixel;

    srand(time(0));

    unsigned int size = width * height;

    count = (degree * size) / 100;

    while(count > 0)
     {pixel = rand() % size;

     if(count % 2 == 1)
         {RGB->R[pixel] = 0;
         RGB->G[pixel] = 0;
         RGB->B[pixel] = 0;
         }/*fi*/

     else
         {RGB->R[pixel] = 255;
         RGB->G[pixel] = 255;
         RGB->B[pixel] = 255;
         }/*esle*/
     count--;
     }/*elihw*/

    printf("\"Add␣Noise\"␣operation␣done!\n");
}

void printoptions(void)
{
    printf("\nFormat␣on␣command␣line␣is:\n"
     "YUV␣<base␣file␣name>␣<options...>\n"
     "\nPossible␣options␣include:\n"
```

```
        ”–i␣<input␣file>\t\t\t\tto␣change␣input␣file␣name\n”
        ”–o␣<output␣file>\t\t\tto␣change␣output␣file␣name\n”
        ”–f␣<initial␣frame>␣<final␣frame>\tto␣create␣a␣YUV␣stream␣from␣”
        ”designated␣initial␣frame␣to␣final␣frame\n”
        ”–f␣<frame>\t\t\t\tto␣create␣a␣ppm␣from␣the␣frame␣selected\n”
        ”–r␣<width>␣<height>\t\t\tto␣designate␣input␣file␣reselution.␣”
        ”Default␣is␣352␣x␣288\n”
        ”–r2␣<width>␣<height>␣<–t>\t\tto␣designate␣output␣file␣resolution.␣”
        ”Default␣is␣input␣resolution.␣Possibly␣add␣tiling\n”
        ”–s␣<step␣size>\t\t\t\tto␣determine␣how␣many␣frames␣desired␣per␣frame␣”
        ”in␣the␣input␣stream\n”
        ”–n\t\t\t\t\tto␣activate␣the␣conversion␣to␣negative\n”
        ”–hf\t\t\t\t\tto␣activate␣horizontal␣flip\n”
        ”–vf\t\t\t\t\tto␣activate␣vertical␣flip\n”
        ”–bw\t\t\t\t\tto␣activate␣the␣conversion␣to␣black␣and␣white\n”
        ”–noise␣<percent␣noise>\t\t\tto␣cause␣a␣percentage␣of␣white␣and␣black␣”
        ”pixelation\n”);
}
```