CS4481B Assignment 1

**Copy of Code:**

**main.c, full program includes all 3 functions written for the assignment**

/\* CS4481B Assignment 1

 Name: Jameel Kaba

 UWO/GAUL ID: jkaba

 Student #: 250796017

 \*/

// Include header files

#include <stdlib.h>

#include <string.h>

#include "libpnm.h"

// Structures for each image type

struct PBM\_Image pbmImage;

struct PGM\_Image pgmImage;

struct PPM\_Image ppmImage;

// Constants, lots of constants

// image dimensions

int width;

int height;

// Is the image raw or ascii, used for saving

bool raw = false;

char filename[1024];

int x;

int y;

int color;

// colorType, used for representing color

char \*colorType[] = {"R","G", "B"};

// max color value

float c = 255.0;

// calculated slope value

float slope;

// shade/intensity of color (used for changing color)

float shade;

/\* PROGRAM 1: PBM IMAGE \*/

int CreatePBM(){

    // Nested loops to set the whole image to black

    for(x = 0; x < width; x++){

        for(y = 0; y < height; y++){

            pbmImage.image[y][x] = 1;

        }

    }

    // Second set of nested loops to create a white square

//in the middle

    for(x = width/4; x < (width\*3/4); x++){

        for(y = height/4; y < (height\*3/4); y++){

            pbmImage.image[y][x] = 0;

        }

    }

    // Calculate the slope

    slope = (float)((height\*(3/4)) - (height/4)) / (float)((width\*(3/4)) - (width/4));

    // If the slope is larger than 1

// then we loop over the x values (width)

    if(slope <= 1.0){

        for(x = width/4; x <= (width\*3/4); x++){

            // Calculate y

            y = slope \* x;

            // -1 is included in the calculations/pixel placement

// to fix rounding error so that pixels go to the end

// rather than stop 1 short

            // Black line which goes from Bottom Right to Top Left

            pbmImage.image[height-y-1][width-x-1] = 1;

            // Black line which goes from Bottom Left to Top Right

            pbmImage.image[height-y-1][x] = 1;

        }

    }

    // Slope is smaller than 1

// so we need to loop over y values (height)

    else{

        for(y = height/4; y <= (height\*3/4); y++){

            // Calculate x

            x = y / slope;

            // -1 is included in the calculations/pixel placement

// to fix rounding error so that pixels go to the end

// rather than stop 1 short

            // Black line which goes from Bottom Right to Top Left

            pbmImage.image[height-y-1][width-x-1] = 1;

            // Black line which goes from Bottom Left to Top Right

            pbmImage.image[height-y-1][x] = 1;

        }

    }

    // Return 0, go back to caller

    return 0;

}

/\* PROGRAM 2: PGM IMAGE \*/

int CreatePGM(){

    // loop counter

    int i;

    // Nested loops to set the whole image to black

    for(x = 0; x < width; x++){

        for(y = 0; y < height; y++){

            pgmImage.image[y][x] = 0;

        }

    }

    // Second set of nested loops to set the middle to white

    for(x = width/4; x < (width\*(3/4)); x++){

        for(y = height/4; y < (height\*(3/4)); y++){

            pgmImage.image[y][x] = 255;

        }

    }

    // Calculate slope

    slope = (float)((height\*(3/4)) - (height/4)) / (float)((width\*(3/4)) - (width/4));

    // If the slope is larger than 1

    if(slope <= 1.0){

        // Set the color intensity / next shade

        shade = (255.0 / width) \* 4.0;

        // Loop from corner to the center

        for(x = width/4; x <= (width/2); x++){

            // Calculate y

            y = slope \* x;

            // Loop over x, to set the values

// for the top and bottom of the inside

            for(i = x; i < width - x; i++){

                //Set the color for the top and bottom to be the same

                pgmImage.image[y][i] = (unsigned char)c;

                // -1 is included in the calculation

// to handle rounding error

                pgmImage.image[height-y-1][i] = (unsigned char)c;

            }

            // Loop over y, to set the values

// for the left and right of the inside

            for(i = y; i < height - y; i++){

                // Set the color for the left and right to be the same

                pgmImage.image[i][x] = (unsigned char)c;

                pgmImage.image[i][width-x] = (unsigned char)c;

            }

            // Adjust the color so that it gets closer to black

            c -= shade;

        }

    }

    // Slope is less than 1

    else{

        // Set the color intensity / next shade

        shade = (255.0 / height) \* 4.0;

        //Loop from the top, to center

        for(y = height/4; y <= (height/2); y++){

            // Calculate x

            x = y / slope;

            // Loop over x values to set the values

// for the top and bottom of the inside

            for(i = x; i < width - x; i++){

                pgmImage.image[y][i] = (unsigned char)c;

                pgmImage.image[height-y][i] = (unsigned char)c;

            }

            // Loop over y values to set the values

// for the left and right of the inside

            for(i = y; i < height - y; i++){

                pgmImage.image[i][x] = (unsigned char)c;

                // -1 included in calculation to

// handle the rounding error

                pgmImage.image[i][width-x-1] = (unsigned char)c;

            }

            // Adjust the color so that it gets closer to black

            c -= shade;

        }

    }

    // Return 0, go back to caller

    return 0;

}

/\* PROGRAM 3: PPM IMAGE \*/

int CreatePPM(){

    // Nested loops to set the image to white

    for(x = 0; x < width; x++){

        for(y = 0; y < height; y++){

            for(color = RED; color <= BLUE; color++){

                ppmImage.image[y][x][color] = 255;

            }

        }

    }

    // Set the intensity / next shade

    shade = 255.0 / (height / 2);

    // Set color value to 0

    c = 0.0;

    // Going from the top to the center

    for(y = 0; y < height/2; y++){

        // Inside loops, setting RGB values

// so that they eventually are equal (white)

        // Red loop / first third (red goes from red to white)

        for(x = 0; x < width/3; x++){

            // Set Green and Blue value to current color value

            ppmImage.image[y][x][GREEN] = (unsigned char)c;

            ppmImage.image[y][x][BLUE] = (unsigned char)c;

        }

        // Green loop / second third (green goes from white to green)

        for(x = width/3; x < width/3 \* 2; x++){

            // Set Red and Blue to be

// 255 - current color value since reverse fade

            ppmImage.image[y][x][RED] = (unsigned char)(255 - c);

            ppmImage.image[y][x][BLUE] = (unsigned char)(255 - c);

        }

        // Blue loop / last third (blue goes from blue to white)

        for(x = width/3 \* 2; x < width; x++){

            // Set Red and Green to be current color value

            ppmImage.image[y][x][RED] = (unsigned char)c;

            ppmImage.image[y][x][GREEN] = (unsigned char)c;

        }

        // Increase the color so it becomes closer to white

        c += shade;

    }

    // Reset color to black

    c = 0.0;

    // Going from the center, down to the bottom

    for(y = height/2; y < height; y++){

        // Black loop / first half (black to white)

        for(x = 0; x < width/2; x++){

            // Set Red, Green, and Blue to all be current color

            ppmImage.image[y][x][RED] = (unsigned char)c;

            ppmImage.image[y][x][GREEN] = (unsigned char)c;

            ppmImage.image[y][x][BLUE] = (unsigned char)c;

        }

        // White loop / second half (white to black)

        for(x = width/2; x < width; x++){

            // Reverse color (-255) applied to Red, Green, and Blue

            ppmImage.image[y][x][RED] = (unsigned char)(255 - c);

            ppmImage.image[y][x][GREEN] = (unsigned char)(255 - c);

            ppmImage.image[y][x][BLUE] = (unsigned char)(255 - c);

        }

        // Increase the color so it becomes closer to white

        c += shade;

    }

    // Return 0, go back to caller

    return 0;

}

/\* MAIN METHOD

    This method performs the initial setup

before going to the functions

    This method also converts PPM Images to PGM Images

 \*/

int main(int argc, char\*\* argv){

    // Boolean values used to determine which image to make

    bool pbm = false;

    bool pgm = false;

    bool ppm = false;

    char extension[1024];

    // If there are not enough arguments then return an error message

    if(argc != 6){

        // Message to let the user know what arguments are needed

        printf("Error: This program requires five arguments\n");

        printf("1: Image type (1 for pbm, 2 for pgm, 3 for ppm)\n");

        printf("2: Width of image. (pbm and pgm must be a multiple of 4, ppm a multiple of 6)\n");

        printf("3: Height of image. (must be a multiple of 4)\n");

        printf("4: Output image name\n");

        printf("5: Image format code (0 for ASCII, 1 for Raw)\n");

        // Stop the program as there was an error

        return 0;

    }

    // Switch case to determine what image to make

    switch (argv[1][0]) {

        // If it is 1, we are making a PBM

        case '1':

            pbm = true;

            break;

        // If it is 2, we are making a PGM

        case '2':

            pgm = true;

            break;

        // If it is 3, we are making a PPM

        case '3':

            ppm = true;

            break;

        // If the number is something else,

// return an error message and stop the program

        default:

            printf("Error: Image type should be either 1(pbm), 2(pgm), or 3(ppm)\n");

            return 0;

    }

    // The width is given as the second argument

    width = atoi(argv[2]);

    // The height is given as the third argument

    height = atoi(argv[3]);

    // The filename is given as the fourth argument

    strcpy(filename, argv[4]);

    // Switch case for fifth argument

// to determine whether the image is ascii or raw

    switch (argv[5][0]) {

        // If it is 0, then the image will be saved as an ascii image

        case '0':

            raw = false;

            break;

        // If it is 1, then the image will be saved as a raw image

        case '1':

            raw = true;

            break;

        // If it is something else,

// then return an error message and stop the program

        default:

            printf("Error: Image format code should be either 0 (ASCII) or 1(raw)\n");

            return 0;

    }

    // If pbm is true, we are making a pbm image

    if(pbm == true){

        // Print statement to let user know what image is being made

        printf("PBM image\n");

        // Width check to ensure that width is proper

        if(width < 4 || width % 4 != 0){

            // If the check fails,

// then return an error message and stop the program

            printf("Error: Width for pbm must be a multiple of 4\n");

            return 0;

        }

        // Height check to ensure that height is proper

        if(height < 4 || height % 4 != 0){

            // If the check fails,

// then return an error message and stop the program

            printf("Error: Height for pbm must be a multiple of 4\n");

            return 0;

        }

        // Try to create enough memory for the image

        if(create\_PBM\_Image(&pbmImage, width, height) != 0){

            // If memory couldn't be allocated

// return an error message and stop the program

            printf("Error: Memory problem creating the PBM Image\n");

            return 0;

        }

        // Print statement to let user know image dimensions

        printf("Creating PBM image %d W x %d H\n", width, height);

        // Create PBM Image function (Refer to program 1)

        CreatePBM();

        // Add .pbm file extension to the filename

        strcat(filename, ".pbm");

        // Try to save the image

        if(save\_PBM\_Image(&pbmImage, filename, raw) != 0){

            // If the image couldn't be saved

// return an error message and stop the program

            printf("An Error occured while trying to save the PBM image to the requested filename\n");

            return 0;

        }

        // If the user wanted to save the image as raw

        if(raw == true){

            // Print message letting the user

// know what the raw image is saved as

            printf("PBM Raw image saved as: %s\n", filename);

        }

        else{

            // Print message letting the user

// know what the ascii image is saved as

            printf("PBM ASCII image saved as: %s\n", filename);

        }

        // Free memory

        free\_PBM\_Image(&pbmImage);

    }

    // If pgm is true, we are making a PGM image

    else if(pgm == true){

        // Print statement to let the user

// know what image is being made

        printf("PGM Image\n");

        // Width check, to ensure that the width is proper

        if(width < 4 || width % 4 != 0){

            // If the width check failed,

// return an error message and stop the program

            printf("Error: Width for pgm image must be a multiple of 4\n");

            return 0;

        }

        // Height check, to ensure that the height is proper

        if(height < 4 || height % 4 != 0){

            // If the height check failed,

// return an error message and stop the program

            printf("Error: Height for pgm image must be a multiple of 4\n");

            return 0;

        }

        // Try to create enough memory for the image

        if(create\_PGM\_Image(&pgmImage, width, height, 255) != 0){

            // If memory could not be allocated,

// return an error message and stop the program

            printf("Error: Memory problem creating the PGM Image\n");

            return 0;

        }

        // Print out the dimensions of the image

        printf("Creating PGM image %d W x %d H\n", width, height);

        // Create PGM Function (Refer to program 2)

        CreatePGM();

        // Add .pgm file extension to the file name

        strcat(filename, ".pgm");

        // Try to save the PGM Image

        if(save\_PGM\_Image(&pgmImage, filename, raw) != 0){

            // If the image could not be saved,

// return an error message and stop the program

            printf("An Error occured while trying to save the PGM image to the requested filename\n");

            return 0;

        }

        // If the user wanted to save the image as raw

        if(raw == true){

            // Print message letting the user

// know what the raw image is saved as

            printf("PGM Raw image saved as: %s\n", filename);

        }

        else{

            // Print message letting the user

// know what the ascii image is saved as

            printf("PGM ASCII image saved as: %s\n", filename);

        }

        // Free memory

        free\_PGM\_Image(&pgmImage);

    }

    // If ppm is true, we are making a PPM Image

    else if(ppm == true){

        // Print statement to let the user

// know what image is being made

        printf("PPM Image\n");

        // Width check to ensure width

// is proper (note: ppm needs a multiple of 6)

        if(width < 6 || width % 6 != 0){

            // If width check fails,

// return an error message and stop the program

            printf("Error: Width for ppm image must be a multiple of 6\n");

            return 0;

        }

        // Height check to ensure height is proper

        if(height < 4 || height % 4 != 0){

            // If height check fails,

// return an error message and stop the program

            printf("Error: Height for ppm image must be a multiple of 4\n");

            return 0;

        }

        // Try to allocate memory for PPM Image

        if(create\_PPM\_Image(&ppmImage, width, height, 255) != 0){

            // If memory could not be allocated,

// return an error message and stop the program

            printf("Error: Memory problem creating the PPM Image\n");

            return 0;

        }

        // Print out dimensions for the image

        printf("Creating PPM image %d W x %d H\n", width, height);

        // Create PPM Function (Refer to program 3)

        CreatePPM();

        // Add .ppm file extension to the file name

        strcat(filename, ".ppm");

        // Try to save the PPM Image

        if(save\_PPM\_Image(&ppmImage, filename, raw) != 0){

            // If the image could not be saved,

// then return an error message and stop the program

            printf("An Error occured while trying to save the PPM image to the requested filename\n");

            return 0;

        }

        // If the user wanted to save the image as raw

        if(raw == true){

            // Print message letting the user

// know what the raw image is saved as

            printf("PPM Raw image saved as: %s\n", filename);

        }

        else{

            // Print message letting the user

// know what the ascii image is saved as

            printf("PPM ASCII image saved as: %s\n", filename);

        }

        // convert and store PPM into 3 PGM through copy\_PPM\_to\_PGM

        printf("Copying PPM Image to PGM Format\n");

        // Allocate memory for the pgm conversion of the ppm image

        if(create\_PGM\_Image(&pgmImage, width, height, 255) != 0){

            // If memory could not be allocated,

// then return an error message and stop the program

            printf("Error: Memory problem creating the PGM Image Copy\n");

            return 0;

        }

        // Print out PGM copy dimensions

        printf("Creating PGM image copy %d W x %d H\n", width, height);

        // Converting ppm into 3 pgm, converting one color at a time

        for(color = RED; color <= BLUE; color++){

            // Copy PPM to PGM with current color from the loop

            if(copy\_PPM\_to\_PGM(&ppmImage, &pgmImage, color) != 0){

                // If there was an error converting,

// return an error message and stop the program

                printf("Error: converting from PPM to PGM\n");

                return 0;

            }

            // Filename setup for the copies

            strcpy(filename, argv[4]);

            strcat(filename, ".ppm.");

            strcat(filename, colorType[color]);

            strcat(filename, ".copy2PGM");

            strcat(filename, ".pgm");

            // Try to save the converted pgm image

            if(save\_PGM\_Image(&pgmImage, filename, raw) != 0){

                // If the PGM cannot be saved,

// return an error message and stop the program

                printf("An Error occured while trying to save the converted PGM image to the requested filename\n");

                return 0;

            }

            // If the user wanted to save the image as raw

            if(raw == true){

                // Print message letting the user

// know what the raw image is saved as

                printf("PGM Raw image saved as: %s\n", filename);

            }

            else{

                // Print message letting the user

// know what the ascii image is saved as

                printf("PGM ASCII image saved as: %s\n", filename);

            }

        }

        // Free memory used for the converted image

        free\_PGM\_Image(&pgmImage);

        // Free memory used for the PPM image

        free\_PPM\_Image(&ppmImage);

    }

    // Program reached end successfully, return 0

    return 0;

}

**Program 1 (PBM)**

**Flowchart 1:**

Initialize constants/variables

Error Handling

Set image type

Is Image type PBM / type 1

Other image type, refer to other flow charts

N

Y

Add file extension

Error Handling

save\_PBM\_image

create\_PBM\_image

Print out filename

CreatePBM

Y > height

While y starts at 0, and goes until height

While x starts at 0, and goes until width

0 ≤ x < width

x > width 0 ≤ y <height

Set the current pixel to black

Y > 3/4

While y goes from 1/4 to 3/4 of height

While x goes from 1/4 to 3/4 of width

1/4 ≤ x < 3/4

X > width 0 ≤ y ≤ height

Calculate slope

Set the current pixel to white

Is slope ≤ 1

N

While y goes from 1/4 to 3/4 of height

Y

Y > 3/4

While x goes from 1/4 to 3/4 of width

X > 3/4

0 ≤ y < height

Calculate x by manipulating slope

0 ≤ x < width

Calculate y by manipulating slope

Draw pixel for diagonal line

Draw pixel for diagonal line

**Required images:**

**Image width 120 height 4**

****

How image was generated: main 1 120 4 binary\_120\_4\_ascii 0

Parameters:

* Program name: main
* Image type: 1
* Width: 120
* Height: 4
* Output image name: binary\_120\_4\_ascii
* Image format: 0

Filename: binary\_120\_4\_ascii.pbm

Description: You might be wondering, why is there just a black box? Well, there actually is a white box which of half width and height, and there is an x made of two diagonal lines of which the pixels touch each other. The reason for why you can’t see the white box with an x is as follows.

When drawing the x, the program will go from one corner of the white box to the halfway point. Along the way to the half way point, every pixel along the row is colored black. Once we reach the halfway point, we go diagonally across one column. From there we continue on to the other corner, filling in every pixel as black along the way. At this point, we have a white box, of which half of the top, and the opposite half on the bottom are filled in. To complete the x, we need to draw the second diagonal line. This is done using the process mentioned previously. If the first diagonal line is not colored in, then we can see the inverse from the second diagonal line being colored in. So, when we combine the two diagonal lines together, we end up with the white box being filled in.

**Image width 4 height 120**

****

How image was generated: main 1 4 120 binary\_4\_120\_ascii 0

Parameters:

* Program name: main
* Image type: 1
* Width: 4
* Height: 120
* Output image name: binary\_4\_120\_ascii
* Image format: 0

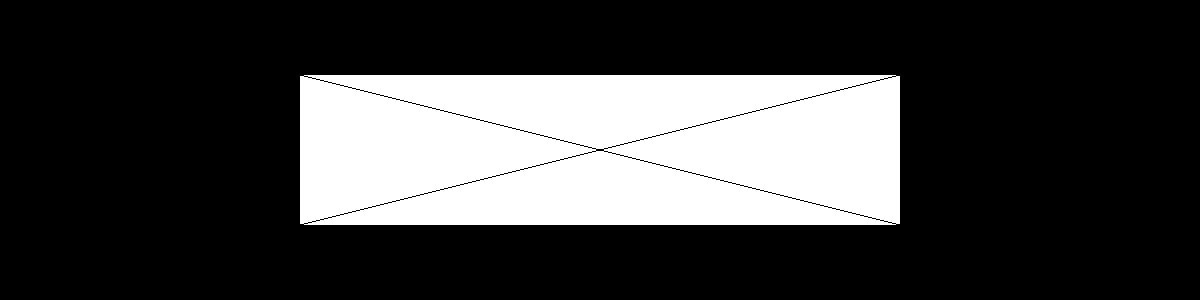
Filename: binary\_4\_120\_ascii.pbm

Description: Like the previous image, this one as well is also a black box. The reasoning is the same as above, as there is a white box of half width and height inside, just that it is being covered up by the two diagonal lines of which the x is comprised of.

When drawing the x, the program will go from one corner of the white box to the halfway point. Along the way to the half way point, every pixel along the column is colored black. Once we reach the halfway point, we go diagonally across one column. From there we continue on to the other corner, filling in every pixel as black along the way. At this point, we have a white box, of which half of the left side, and the opposite half on the right side are filled in. To complete the x, we need to draw the second diagonal line. This is done using the process mentioned previously. If the first diagonal line is not colored in, then we can see the inverse from the second diagonal line being colored in. So, when we combine the two diagonal lines together, we end up with the white box being filled in, and are left with a black box.

**Test Cases:**

**Horizontal Rectangle**

****

How image was generated: main 1 1200 300 binary\_1200\_300\_ascii 0

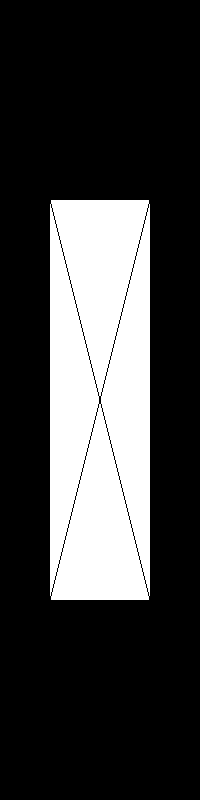
Parameters:

* Program name: main
* Image type: 1
* Width: 1200
* Height 300
* Output image name: binary\_1200\_300\_ascii
* Image format: 0

Filename: binary\_1200\_300\_ascii.pbm

Description: PBM image with a width of 1200 and a height of 300. This 1200 by 300 image is a horizontal rectangle. The white box in the middle is half the width and half the height of the entire image. The black x in the middle is two diagonal lines of which the pixels are connected from corner to corner. In each row, the diagonal line consists of 4 pixels in width per row before moving to next row.

**Vertical Rectangle (On next page, description the page after)**

****

How image was generated: main 1 200 800 binary\_200\_800\_ascii 0

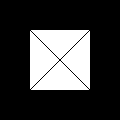
Parameters:

* Program name: main
* Image type: 1
* Width: 200
* Height 800
* Output image name: binary\_200\_800\_ascii
* Image format: 0

Filename: binary\_200\_800\_ascii.pbm

Description: PBM image with a width of 200 and a height of 800. This 200 by 800 image is a vertical rectangle. The white box in the middle is half the width and half the height of the whole image. The black x in the middle is two diagonal lines of which the pixels are connected from corner to corner. Each line goes vertically down 4 pixels before moving diagonally to the next column.

**Generated Square**

****

How image was generated: main 1 120 120 binary\_120\_120\_raw 1

Parameters:

* Program name: main
* Image type: 1
* Width: 120
* Height: 120
* Output image name: binary\_120\_120\_raw
* Image format: 1

Filename: binary\_120\_120\_raw.pbm

Description: PBM image with width and height of 120, making it a square. The white box in the middle is half of both the width and height. The two diagonal lines in the middle go from corner to corner. Each line goes diagonally across 1 pixel at a time until it reaches the opposing corner.

**Error Testing (Testing to see if error’s generate when needed)**

**Image type -1**

How error was generated: main -1 120 120 image 0

Output:

*Error: Image type should be either 1(pbm), 2(pgm), or 3(ppm)*

Reason: Because the image type is an invalid option, the user is presented with the error message above.

**Having more than required arguments**

How error was generated: main 1 120 120 image 0 120

Output:

*Error: This program requires five arguments*

*1: Image type (1 for pbm, 2 for pgm, 3 for ppm)*

*2: Width of image. (pbm and pgm must be a multiple of 4, ppm a multiple of 6)*

*3: Height of image. (must be a multiple of 4)*

*4: Output image name*

*5: Image format code (0 for ASCII, 1 for Raw)*

Reason: Because there are more than 5 arguments, the program will send out the argument error, telling the user that they need 5 arguments not more.

**Image height -10**

How error was generated: main 1 120 -10 image 0

Output:

*PBM image*

*Error: Height for pbm must be a multiple of 4*

Reason: Height is not a multiple of 4, as such it fails the height check and the error above is generated.

**Image format code is 3**

How error was generated: main 1 120 120 image 3

Output:

*Error: Image format code should be either 0 (ASCII) or 1(raw)*

Reason: Image format code is an invalid option, as such the error message above is presented to the user.

**Program 2 (PGM)**

**Flowchart 2:**

Initialize constants/variables

Error Handling

Set image type

Is Image type PGM / type 2

Other image type, refer to other flow charts

N

Y

Add file extension

Error Handling

save\_PGM\_image

create\_PGM\_image

Print out filename

CreatePGM

Y > height

While y starts at 0, and goes until height

While x starts at 0, and goes until width

0 ≤ x < width

x > width 0 ≤ y <height

Set the current pixel to black

Y > 3/4

While y goes from 1/4 to 3/4 of height

While x goes from 1/4 to 3/4 of width

1/4 ≤ x < 3/4

X > width 0 ≤ y ≤ height

Calculate slope

Set the current pixel to white

Is slope ≤ 1

N

Set the shade of gray

Y

Set the shade of gray

While y goes from h/4 to h/2

Y > h/2

While x goes from w/4 to w/2

X > w/2 h/4 ≤ y < h/2

Calculate x by manipulating slope

w/4 ≤ x ≤ w/2

Calculate y by manipulating slope

While i goes from x to w-x

While i goes from x to w-x

i < w-x

i < w-x

Set the color for top and bottom line

Set the color for top and bottom line

i > w-x i > w-x

While i goes from y to h-y

While i goes from y to h-y

i < h-y i < h-y

Set the color for left and right line

Set the color for left and right line

i > h-y i > h-y

Adjust color shade

Adjust color shade

**Required images:**

**Image width 120 height 4**

****

How image was generated: main 2 120 4 gray\_120\_4\_ascii 0

Parameters:

* Program name: main
* Image type: 2
* Width: 120
* Height: 4
* Output image name: gray\_120\_4\_ascii
* Image format: 0

Filename: gray\_120\_4\_ascii.pgm

Description: For this case where the width is 120 and height is 4. The white rectangle is of width 60 and height of 2. The white box goes from white at the edges, to black in the center. Because there is only a height of 4, the box goes from white to black from the sides and not from top to bottom. As, since there is only 2 pixels, the box would either have to be all of the same color to use all 4 sides. At the edges of the rectangle we start at white, and each pixel that we go further to the center we go through different shades of gray before ultimately reaching black. Because we only have 2 pixels worth of height to work with, we take advantage of the width to reach our color goal. Going by height would not work as there would not be a center location. So both top and bottom would be white, because we are trying to have the same shade all around, we can’t have the top being white and bottom black or vice versa. Due to the larger width, it is possible to have a center point where we can have black and so side to side is the way to go.

**Image width 4 height 120**

****

How image was generated: main 2 4 120 gray\_4\_120\_ascii 0

Parameters:

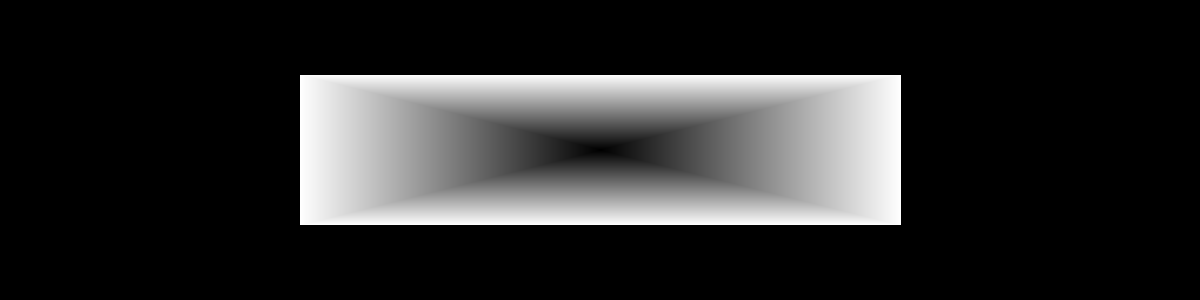
* Program name: main
* Image type: 2
* Width: 4
* Height: 120
* Output image name: gray\_4\_120\_ascii
* Image format: 0

Filename: gray\_4\_120\_ascii.pgm

Description: For this case where the width is 4 and height is 120. The white rectangle is of width 2 and height of 60. The white box goes from white at the edges, to black in the center. Because there is only a width of 4, the box goes from white to black from top to bottom and not from side to side. As, since there is only 2 pixels, the box would either have to be all of the same color to use all 4 sides. At the edges of the rectangle we start at white, and each pixel that we go further to the center we go through different shades of gray before ultimately reaching black. Because we only have 2 pixels worth of width to work with, we take advantage of the height to reach our color goal. Going by width would not work as there would not be a center location. So both left and right would be white, because we are trying to have the same shade all around, we can’t have the left being white and right black or vice versa. Because we have a larger amount of height, we can have a center point for where we will have black.

**Test Cases:**

**Horizontal Rectangle**

****

How image was generated: main 2 1200 300 gray\_1200\_300\_ascii 0

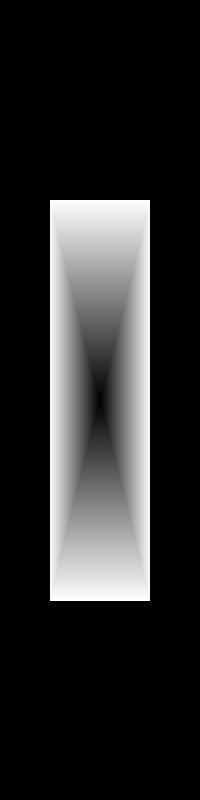
Parameters:

* Program name: main
* Image type: 2
* Width: 1200
* Height: 300
* Output image name: gray\_1200\_300\_ascii
* Image format: 0

Filename: gray\_1200\_300\_ascii.pgm

Description: This is a horizontal rectangle representation of the black box with a white box that gets darker and darker inside until it reaches black in the center. The width is 1200 and height is 300, the width and height of the white box is half of the black box. At the edge of the white box the color is pure white, and as we get closer and closer to the center we end up going through darker shades of gray before ultimately hitting black in the center. The inner rectangle can be thought of as 4 triangles, of which they meet at the center and form a pyramid.

**Vertical Rectangle (Image on next page, description on the page following)**

****

How image was generated: main 2 200 800 gray\_200\_800\_ascii 0

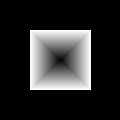
Parameters:

* Program name: main
* Image type: 2
* Width: 200
* Height: 800
* Output image name: gray\_200\_800\_ascii
* Image format: 0

Filename: gray\_200\_800\_ascii.pgm

Description: This is the vertical rectangle representation of a black box with a white box that gets darker and darker inside until it reaches black in the center. The width is 200 and height is 800, the width and height of the white box is half of the black box. At the edge of the white box the color is pure white, and as we get closer and closer to the center we end up going through darker shades of gray before ultimately hitting black in the center. The inner rectangle can be thought of as 4 triangles, of which they meet at the center and form a pyramid.

**Generated Square**

****

How image was generated: main 2 120 120 gray\_120\_120\_ascii 0

Parameters:

* Program name: main
* Image type: 2
* Width: 120
* Height: 120
* Output image name: gray\_120\_120\_ascii
* Image format: 0

Filename: gray\_120\_120\_ascii.pgm

Description: This is the square representation of a black box with a white box that gets darker and darker inside until it reaches black in the center. The width is 120 and height is 120, the width and height of the white box is half of the black box. At the edge of the white box the color is pure white, and as we get closer and closer to the center we end up going through darker shades of gray before ultimately hitting black in the center. The inner rectangle can be thought of as 4 triangles, of which they meet at the center and form a pyramid.

**Error Testing (Testing to ensure error checks work)**

**Image type replaced with a space**

How error was generated: main 120 120 image 1

Output:

*Error: This program requires five arguments*

*1: Image type (1 for pbm, 2 for pgm, 3 for ppm)*

*2: Width of image. (pbm and pgm must be a multiple of 4, ppm a multiple of 6)*

*3: Height of image. (must be a multiple of 4)*

*4: Output image name*

*5: Image format code (0 for ASCII, 1 for Raw)*

Reason: Because the image type is replaced with a space, it is considered as if it was not there, and so only 4 arguments are present. Due to this the error above is presented as there are not enough arguments.

**Image width of 0**

How error was generated: main 2 0 120 image 1

Output:

*PGM Image*

*Error: Width for pgm image must be a multiple of 4*

Reason: Program realizes we are trying to make a PGM image, but the width check fails as the width is not a multiple of 4, and so the error message is displayed.

**Image height not there**

How error was generated: main 2 120 image 1

Output:

*Error: This program requires five arguments*

*1: Image type (1 for pbm, 2 for pgm, 3 for ppm)*

*2: Width of image. (pbm and pgm must be a multiple of 4, ppm a multiple of 6)*

*3: Height of image. (must be a multiple of 4)*

*4: Output image name*

*5: Image format code (0 for ASCII, 1 for Raw)*

Reason: Because height is replaced with spacing, the program recognizes that there are not enough arguments, and so prompts the initial error message.

**Image format code is -2**

How error was generated: main 2 120 120 image -2

Output:

*Error: Image format code should be either 0 (ASCII) or 1(raw)*

Reason: Because the image format code was neither 0 or 1, the program informs the user of their invalid choice and what needs to be done to fix this problem.

**Program 3 (PPM)**

**Flowchart 3: (Refer to copy of main.c for code)**

Initialize constants/variables

Error Handling

Set image type

Is Image type PPM / type 3

Other image type, refer to other flow charts

N

Y

Add file extension

Error Handling

save\_PPM\_image

create\_PPM\_image

Print out filename

CreatePPM

Print to let user know about PPM to PGM conversion

create\_PGM\_image

Loop through each RGB

RGB Each finish

Loop through red, green, and blue

copy\_PPM\_to\_PGM

Add file extension

save\_PGM\_image

Print out filename

Loop through image, set every pixel white

Not every pixel is white

Set pixel to white

Every pixel is white

Set color shade

Set initial color to 0

Loop through the top half of image

Finished looping through top half

Not finished looping through top half

Loop through first third of columns

i < 1/3

Set the color for the red part of image

i = 1/3

Loop through second third of columns

Set the color for the green part of image

i < 2/3

i = 2/3

Loop through last third of columns

i < 3/3

Set the color for the blue part of image

i = 3/3

Increase the shade of color

Set color to 0

Loop through the bottom half of image

Finish Looping

not finished

Loop through first half of bottom columns

Set the color for the black part of image

i < 1/2

i=1/2

Loop through second half of bottom columns

i < 1/1

Set the color for the white part of image

i=1/1

Increase the shade of color

**Required images:**

**\*\*Comments on the conversion from PPM to PGM are located after the required images and there descriptions**

**Image width 120 height 4**

****

How image was generated: main 3 120 4 color\_120\_4\_ascii 0

Parameters:

* Program name: main
* Image type: 3
* Width: 120
* Height: 4
* Output image name: color\_120\_4\_ascii
* Image format: 0

Filename: color\_120\_4\_ascii.ppm

Description: PPM image of height 4 and width 120. The top 2 pixels are broken into 3 sections. Each section is 40 pixels long. The first section is red, second green, and blue being the third. The red and blue sections follow the pattern of darker on the top pixel and lighter on the bottom. While the Green part is the reverse, with light being on top and dark being on the bottom. For the second half, the image is broken into 2 parts. The first part is black on top, and grey on the bottom. While for the second half it is white on top and grey on the bottom.

The reason why the colors don’t go from one extreme to the other is as follows. Because the height of the image is 4, the image cannot go from all of 1 color to complete white. Instead for the case of red and blue, it goes from all blue to a middle lighter shade of blue. For green, because it goes from white to dark green, it starts at white and goes to a middle lighter shade of green. For the black, and white sections on the bottom of the image it is the same idea. However, because black goes to white, and white goes to black, the bottom pixel for both sections are identical. If you were to increase the height of the image, you can see more clearly how each part changes shades to go to it’s appropriate end color.

**Conversion to PGM through copy\_PPM\_to\_PGM**

****

Filename: color\_120\_4\_ascii.ppm.R.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the red component has become white.

****

Filename: color\_120\_4\_ascii.ppm.G.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the green component has become white.

****

Filename: color\_120\_4\_ascii.ppm.B.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the blue component has become white.

**Image width 6 height 120**

****

How image was generated: main 3 6 120 color\_6\_120\_ascii 0

Parameters:

* Program name: main
* Image type: 3
* Width: 6
* Height: 120
* Output image name: color\_6\_120\_ascii
* Image format: 0

Filename: color\_6\_120\_ascii.ppm

Description: This PPM image has a width of 6, and a height of 120. The reason as to why the PPM image needs to have a width of at least 6 is simple. It’s due to the top half having 3 portions, and the bottom having 2. The minimum required width needed would be 6 pixels as that is the first common number between the two.

The image is described as follows. The red and blue parts go from pure red and blue to white as they get lighter and lighter the further down it goes. For green, the reverse is true as it starts at white and becomes a darker shade of green as it gets further down. The bottom half is split into two parts: black going to white, and white going to black. Halfway through both sides are the same color shade.

**Conversion to PGM through copy\_PPM\_to\_PGM**

****

Filename: color\_6\_120\_ascii.ppm.R.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the red component has become white.

****

Filename: color\_6\_120\_ascii.ppm.G.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the green component has become white.

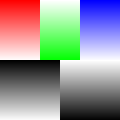
****

Filename: color\_6\_120\_ascii.ppm.B.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the blue component has become white.

**Comments in regards to the converted pictures**

**\*\* For the images used below, refer to the test case on generated square for more information in regards to how the image was generated**

****

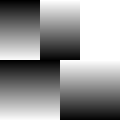
Here we have a square version of the PPM image. As you can see it goes from red to white, white to green, and blue to white from the top to halfway down. For the second half it goes from black to white, and white to black. When converting the image into a PGM image, it is done 1 color at a time starting with Red.

****

Here is the red PGM copy of the PPM image, as you can see the red component has become pure white. While the green component now matches with the white to black, and likewise the blue component matches with the black to white. This is due to the conversion, as when converting into gray scale, this is done 1 color at a time. However, this is because we are removing this 1 color. As such, the red component of the image has been removed. Now you might be wondering, that if red was removed then how can we still have the other shades. Well, it’s due to the type of image we’re converting it into. PGM requires a max gray value, and so whatever the intensities of the remaining colors are, they are converted into the appropriate gray color based on the max gray value.

****

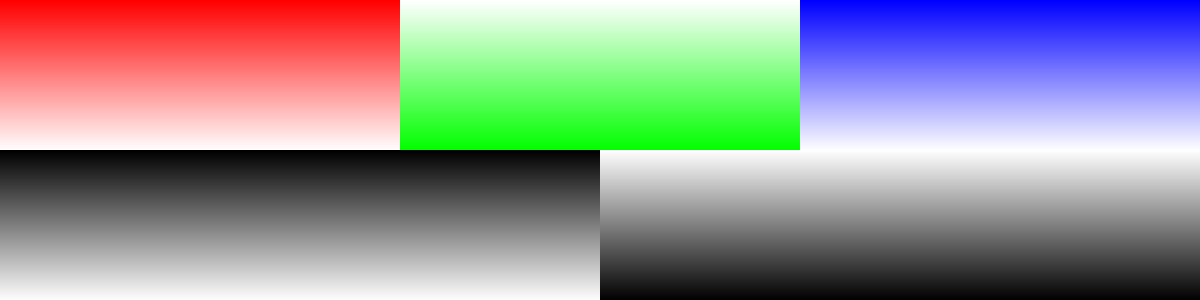
Here is the Green PGM copy of the PPM image, as you can see the green component has become pure white. While both the red and blue components match the black to white component. This is due to the conversion. Because red and blue are both going from dark to white, when converted into gray scale they become black going into white. The green component is white due to the color being removed from the image.

****

Here is the Blue PGM copy of the PPM image, as you can see the blue component has become pure white. While both the red and green have been converted into grayscale. Because red goes from dark to white, when converted it becomes a match with the black going to white part in the bottom. Likewise for green, how it goes from white to dark, it ends up matching the white into black component. Blue has been removed from the image and as such it appears as white.

**Test Cases:**

**Horizontal Rectangle**

****

How image was generated: main 3 1200 300 color\_1200\_300\_ascii 0

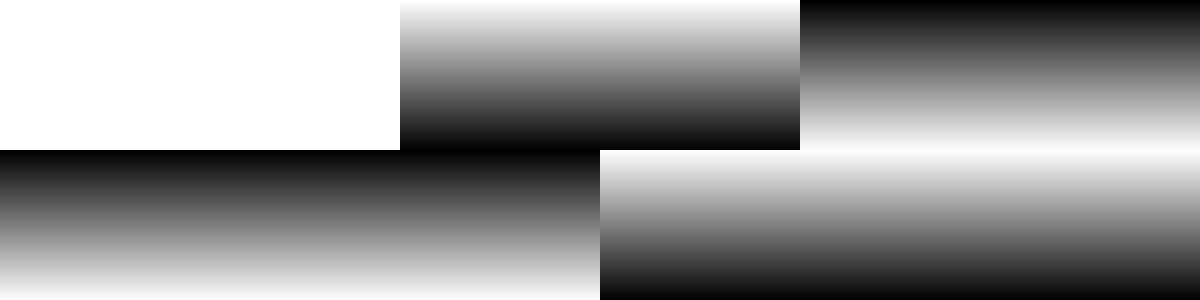
Parameters:

* Program name: main
* Image type: 3
* Width: 1200
* Height: 300
* Output image name: color\_1200\_300\_ascii 0
* Image format: 0

Filename: color\_1200\_300\_ascii.ppm

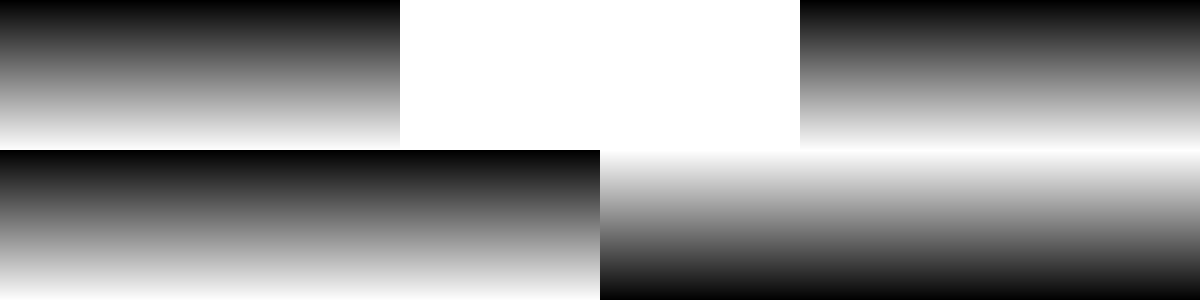
Description: PPM image of height 1200 and width 300. The top is broken into 3 sections while the bottom is split in half. On top the first section is red, second green, and blue being the third. The red and blue sections follow the pattern of darker on the top pixel and lighter on the bottom. While the Green part is the reverse, with light being on top and dark being on the bottom. For the bottom half the first part starts as black on top, and ends at white on the bottom, getting lighter as it goes. While for the second half it is white on top and black on the bottom, getting darker as it goes.

**Conversion to PGM through copy\_PPM\_to\_PGM**

****

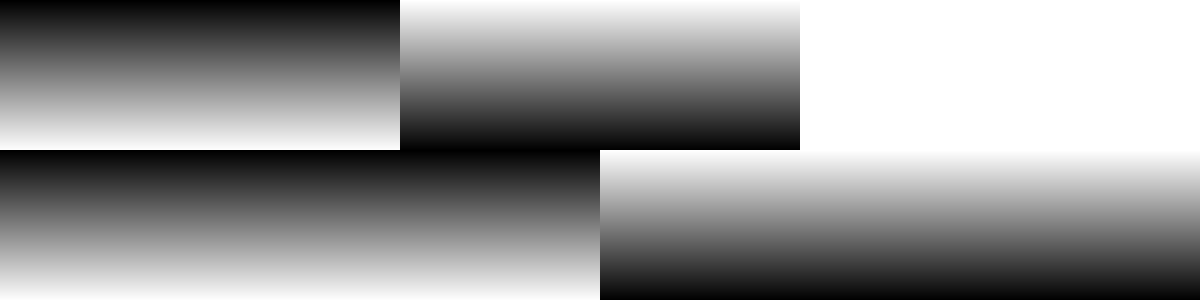
Filename: color\_1200\_300\_ascii.ppm.R.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the red component has become white.

****

Filename: color\_1200\_300\_ascii.ppm.G.copy2PGM.pgm

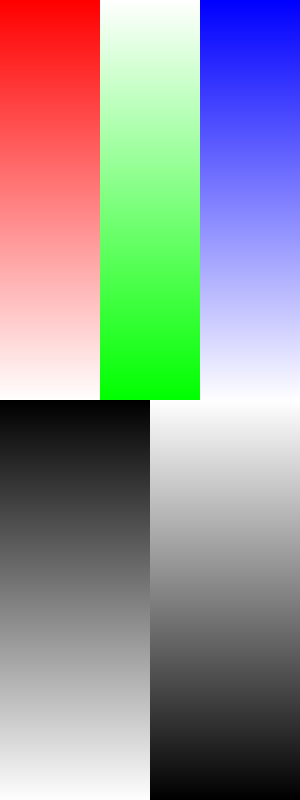
Description: PGM copy of the PPM image above. In this image, the green component has become white.

****

Filename: color\_1200\_300\_ascii.ppm.B.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the blue component has become white.

**Vertical Rectangle (Image on following page, description following)**

****

How image was generated: main 3 300 800 color\_300\_800\_ascii 0

Parameters:

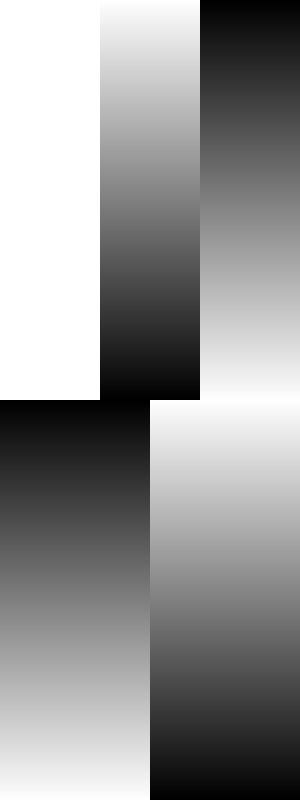
* Program name: main
* Image type: 3
* Width: 300
* Height: 800
* Output image name: color\_300\_800\_ascii
* Image format: 0

Filename: color\_300\_800\_ascii.ppm

Description: PPM image of height 300 and width 800. The top is broken into 3 sections while the bottom is split in half. On top the first section is red, second green, and blue being the third. The red and blue sections follow the pattern of darker on the top pixel and lighter on the bottom. While the Green part is the reverse, with light being on top and dark being on the bottom. For the bottom half the first part starts as black on top, and ends at white on the bottom, getting lighter as it goes. While for the second half it is white on top and black on the bottom, getting darker as it goes.

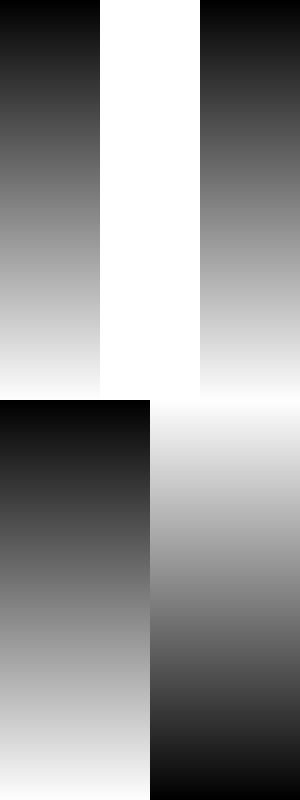
**Conversion to PGM through copy\_PPM\_to\_PGM**

**(Images on following pages, with descriptions following)**

****

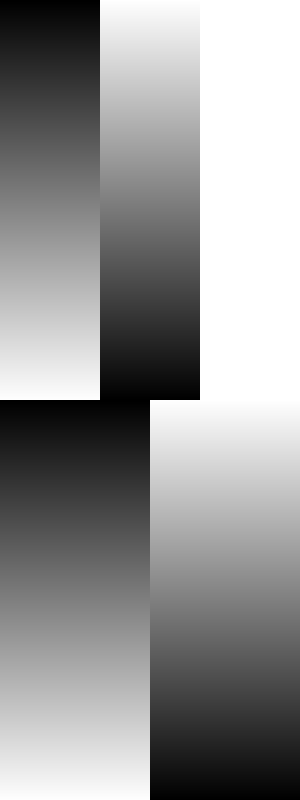
Filename: color\_300\_800\_ascii.ppm.R.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the red component has become white.

****

Filename: color\_300\_800\_ascii.ppm.G.copy2PGM.pgm

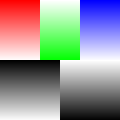
Description: PGM copy of the PPM image above. In this image, the green component has become white.

****

Filename: color\_300\_800\_ascii.ppm.B.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the blue component has become white.

**Generated Square**

****

How image was generated: main 3 120 120 color\_120\_120\_ascii 0

Parameters:

* Program name: main
* Image type: 3
* Width: 120
* Height: 120
* Output image name: color\_120\_120\_ascii
* Image format: 0

Filename: color\_120\_120\_ascii.ppm

Description: PPM image of 120 by 120 square. The top is broken into 3 sections while the bottom is split in half. On top the first section is red, second green, and blue being the third. The red and blue sections follow the pattern of darker on the top pixel and lighter on the bottom. While the Green part is the reverse, with light being on top and dark being on the bottom. For the bottom half the first part starts as black on top, and ends at white on the bottom, getting lighter as it goes. While for the second half it is white on top and black on the bottom, getting darker as it goes.

**Conversion to PGM through copy\_PPM\_to\_PGM**

****

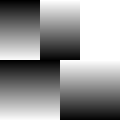
Filename: color\_120\_120\_ascii.ppm.R.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the red component has become white.

****

Filename: color\_120\_120\_ascii.ppm.G.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the green component has become white.

****

Filename: color\_120\_120\_ascii.ppm.B.copy2PGM.pgm

Description: PGM copy of the PPM image above. In this image, the blue component has become white.

**Error Testing (Tests to ensure that error checks work)**

**Wrong image type (0)**

How error was generated: main 0 120 120 image 0

Output:

*Error: Image type should be either 1(pbm), 2(pgm), or 3(ppm)*

Reason: Due to the image code not being 1, 2, or 3 the program returns an error telling the user that 0 is an invalid option.

**Image Width of 4**

How error was generated: main 3 4 120 image 0

Output:

*PPM Image*

*Error: Width for ppm image must be a multiple of 6*

Reason: Program gets far enough to see it is a PPM image we are trying to make. Due to the width being 4, it fails the width check as for PPM width needs to be a multiple of 6.

**Image Height of 6**

How error was generated: main 3 120 6 image 1

Output:

*PPM Image*

*Error: Height for ppm image must be a multiple of 4*

Reason: Program gets far enough to see that we are making a PPM image, unfortunately due to the height not being a multiple of 4, the height check fails and so the program returns an error.

**No output name**

How error was generated: main 3 120 120  1

Output:

*Error: This program requires five arguments*

*1: Image type (1 for pbm, 2 for pgm, 3 for ppm)*

*2: Width of image. (pbm and pgm must be a multiple of 4, ppm a multiple of 6)*

*3: Height of image. (must be a multiple of 4)*

*4: Output image name*

*5: Image format code (0 for ASCII, 1 for Raw)*

Reason: Because the output name is not there, the 1 used for the format is considered as the filename, and so the program thinks there is no format code. Either way, there are not enough arguments and so the program returns the initial error.

**No image format code**

How error was generated: main 3 120 120 image

Output:

*Error: This program requires five arguments*

*1: Image type (1 for pbm, 2 for pgm, 3 for ppm)*

*2: Width of image. (pbm and pgm must be a multiple of 4, ppm a multiple of 6)*

*3: Height of image. (must be a multiple of 4)*

*4: Output image name*

*5: Image format code (0 for ASCII, 1 for Raw)*

Reason: Because the last argument is missing, the general error message is displayed telling the user that there is not enough arguments.