4481 Assignment #1
Paul Salvatore 250668447

***** CODE *****

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "libpnm.h"


void program_1(int width, int height, char* image_name, int
image_format);
void program_2(int width, int height, char* image_name, int
image_format);
void program_3(int width, int height, char* image_name, int
image_format);


int main(int argc, char *argv[]){

     // Checking that the appropriate number of args were passed
     if(argc != 6){
          printf("Please ensure you have entered five arguments.
You entered: %d\n", argc - 1);
          exit(0);
     }

     // Converting args to their proper types
     int code = atoi(argv[1]);
     int width = atoi(argv[2]);
     int height = atoi(argv[3]);
     char* image_name = argv[4];
     bool image_format = atoi(argv[5]);

     // Checking that the height conforms to specifications
     if (height % 4 != 0 || height < 4) {
          printf("Please enter a valid height.\n");
          exit(0);
     }

     // Checking that the width and code conform to
specifications
     if(code == 1 || code == 2){
          if (width % 4 != 0 || width < 4){
               printf("Please enter a valid width.\n");
               exit(0);
          }
     } else if (code == 3){
```

```
          if (width % 6 != 0 || width < 6){
               printf("Please enter a valid width.\n");
               exit(0);
          }
     } else{
          printf("Please ensure you have entered a valid
code.\n");
          exit(0);
     }

     // Checking that the image format and code conform to
specifications
     if (image_format != 0 && image_format != 1){
          printf("Please ensure you have entered a valid
format.\n");
          exit(0);
     }

     // Running the appropriate program, as specified by the
code
     if(code == 1){
          program_1(width, height, image_name, image_format);
     } else if (code == 2){
          program_2(width, height, image_name, image_format);
     } else {
          program_3(width, height, image_name, image_format);
     }

     return 0;
}


// Determines the min of two numbers
int min(int a, int b){
     return a > b ? b : a;
}


// Determines the max of two numbers
int max(int a, int b){
     return a > b ? a : b;
}


void program_1(int width, int height, char* image_name, bool
image_format){
```

4481 Assignment #1
Paul Salvatore 250668447

```
	// Initalize a pbm image and input it's parameters
	struct PBM_Image *pbm_image = malloc(sizeof(struct 
PBM_Image));
	create_PBM_Image(pbm_image, width, height);

	// Building the outer rectangle
	for(int i = 0; i < height; i++){
		for(int j = 0; j < width; j++){

			// If we are outside the middle rectangle, we 
make the region black, otherwise we make the inner rectangle 
white.
			if(j < width * 0.25 || j >= width * 0.75 || i < 
height * 0.25 || i >= height * 0.75){
				pbm_image->image[i][j] = BLACK;
			} else{
				pbm_image->image[i][j] = WHITE;

			}
		}
	}


	/*** Building the "x" ***/


	// X and Y starting positions for the inner rectngle
	const int START_Y = (int)(0.25 * height);
	const int START_X = (int)(0.25 * width);

	// The width and height of the inner rectangle
	const int SIZE_Y = (int)(0.5 * height);
	const int SIZE_X = (int)(0.5 * width);

	// The X and Y end positions for the inner rectangle
	const float BOUNDARY_Y = 0.75 * height;
	const float BOUNDARY_X = 0.75 * width;

	// Current position variables
	int cur_y = 0;
	int cur_x = 0;

	// When finish when we have traversed enough x and y 
positions to cover the entire inner rectangle's width and height
	while(cur_y < SIZE_Y || cur_x < SIZE_X){

		// We find the current position and make it black
```

```
			pbm_image->image[START_Y + min(SIZE_Y, cur_y)][START_X
+ min(SIZE_X, cur_x)] = BLACK;

			// We do the same for a lower line which will run
bottom to top, rather than top to bottom
			pbm_image->image[START_Y + min(SIZE_Y, cur_y)][START_X
+ SIZE_X - min(SIZE_X, cur_x) - 1] = BLACK;

			// Booleans for determining when to move to the right
or down by increasing our position variables
			bool inc_y = false;
			bool inc_x = false;

			// If the percentage that we have traversed y is less
than or equal to the percentage that
			// we have traversed x, than we want to move down
since we have more vertical space to
			// finish traversing than horizontal
			if ((cur_y + 1.0) / (BOUNDARY_Y - 1) <= (cur_x + 1.0)
/ (BOUNDARY_X - 1)){
				inc_y = true;
			}

			// If the percentage that we have traversed y is
greater than or equal to the percentage that
			// we have traversed x, than we want to move right
since we have more horizontal space to
			// finish traversing than vertical
			if ((cur_y + 1.0) / (BOUNDARY_Y - 1) >= (cur_x + 1.0)
/ (BOUNDARY_X - 1)) {
				inc_x = true;
			}

			// Perform the movements by increasing the position
variables
			if(inc_y){
				cur_y++;
			}
			if(inc_x){
				cur_x++;
			}
		}

		// Save the image and clear allocated memory
		save_PBM_Image(pbm_image, image_name, image_format);
		free_PBM_Image(pbm_image);
		free(pbm_image);
```

```
}


void program_2(int width, int height, char* image_name, int
image_format){

     // Initalize a pgm image and input it's parameters
     struct PGM_Image *pgm_image = malloc(sizeof(struct
PGM_Image));
     create_PGM_Image(pgm_image, width, height, MAX_GRAY_VALUE);

     // Building the outer rectangle
     for(int i = 0; i < height; i++){
          for(int j = 0; j < width; j++){

               // If we are outside the middle rectangle, we
make the region black, otherwise we make the inner rectangle
white.
               if(j < width * 0.25 || j >= width * 0.75 || i <
height * 0.25 || i >= height * 0.75){
                    pgm_image->image[i][j] = 0;
               } else{
                    pgm_image->image[i][j] = MAX_GRAY_VALUE;

               }
          }
     }


     /*** Building the gradients ***/


     // X and Y starting positions for the inner rectngle
     const int START_Y = (int)(0.25 * height);
     const int START_X = (int)(0.25 * width);

     // The width and height of the inner rectangle
     const int SIZE_Y = (int)(0.5 * height);
     const int SIZE_X = (int)(0.5 * width);

     // The X and Y end positions for the inner rectangle
     const float BOUNDARY_Y = 0.75 * height;
     const float BOUNDARY_X = 0.75 * width;

     // The step sizes with which each pixel of the triangles
will advance
     const float STEP_Y = 255/((SIZE_Y)/2.0);
```

4481 Assignment #1
Paul Salvatore 250668447

```
	const float STEP_X = 255/((SIZE_X)/2.0 - 1);

	// Current position variables
	int cur_y = 0;
	int cur_x = 0;

	// Current pixel intensity variables
	float cur_x_gray = MAX_GRAY_VALUE;
	float cur_y_gray = MAX_GRAY_VALUE;

	// We only need to traverse the first quarter quadrant of 
the image since this will be mirrored
	// SO once each of our position variables pass this section 
we can finish
	while(cur_y < SIZE_Y/2 || cur_x < SIZE_X/2){

		// Drawing an "x" similar to program one, taking the 
average of the tow neighboring triangles
		pgm_image->image[START_Y + min(SIZE_Y/2, 
cur_y)][START_X + min(SIZE_X/2, cur_x)] = (int)((cur_x_gray + 
cur_y_gray) / 2.0);
		pgm_image->image[(int)BOUNDARY_Y - min(SIZE_Y/2, 
cur_y) - 1][START_X + min(SIZE_X/2, cur_x)] = (int)((cur_x_gray 
+ cur_y_gray) / 2.0);

		// Repeating drawing the "x" for the other half
		pgm_image->image[START_Y + min(SIZE_Y/2, 
cur_y)][START_X + SIZE_X - min(SIZE_X/2, cur_x) - 1] = 
(int)((cur_x_gray + cur_y_gray) / 2.0);
		pgm_image->image[(int)BOUNDARY_Y - min(SIZE_Y/2, 
cur_y) - 1][START_X + SIZE_X - min(SIZE_X/2, cur_x) - 1] = 
(int)((cur_x_gray + cur_y_gray) / 2.0);

		// Booleans for determining when to move to the right 
or down by increasing our position variables
		bool inc_y = false;
		bool inc_x = false;

		// If the percentage that we have traversed y is less 
than or equal to the percentage that
		// we have traversed x, than we want to move down 
since we have more vertical space to
		// finish traversing than horizontal
		if ((cur_y + 1.0) / (BOUNDARY_Y - 1) <= (cur_x + 1.0) 
/ (BOUNDARY_X - 1)){
			inc_y = true;
```

```
                // Vertical triangles, drawing horizontal lines
when we know that x boundary is finalized
                for(int j = START_X + cur_x + 1; j < BOUNDARY_X -
min(SIZE_X, cur_x) - 1; j++){

                     pgm_image->image[START_Y + min(SIZE_Y,
cur_y)][j] = (int)cur_y_gray;
                     pgm_image->image[(int)BOUNDARY_Y - cur_y -
1][j] = (int)cur_y_gray;

                }

                // Alter the colour of the next set of pixels in
the top and bottom triangles
                // The minimum value these triangles can have is
0
                cur_y_gray = cur_y_gray - STEP_Y;
           }

           // If the percentage that we have traversed y is
greater than or equal to the percentage that
           // we have traversed x, than we want to move right
since we have more horizontal space to
           // finish traversing than vertical
           if ((cur_y + 1.0) / (BOUNDARY_Y - 1) >= (cur_x + 1.0)
/ (BOUNDARY_X - 1)) {
                inc_x = true;

                // Side-ways triangles, drawing vertical lines
when we know that y boundary is finalized
                for(int j = START_Y + cur_y + 1; j <= BOUNDARY_Y
-  min(SIZE_Y, cur_y) - 1; j++){

                     pgm_image->image[j][START_X + min(SIZE_X,
cur_x)] = (int)cur_x_gray;
                     pgm_image->image[j][(int)BOUNDARY_X - cur_x
- 1] = (int)cur_x_gray;

                }

                // Alter the colour of the next set of pixels in
the left and right triangles
                // The minimum value these triangles can have is
0
                cur_x_gray = cur_x_gray - STEP_X;
           }
```

```
          // Perform the movements by increasing the position
variables
          if(inc_y){
               cur_y++;
          }
          if(inc_x){
               cur_x++;
          }
     }

     // Save the image and clear allocated memory
     save_PGM_Image(pgm_image, image_name, image_format);
     free_PGM_Image(pgm_image);
     free(pgm_image);
}


void program_3(int width, int height, char* image_name, int
image_format){

     // Initalize a pgm image and input it's parameters
     struct PPM_Image *ppm_image = malloc(sizeof(struct
PPM_Image));
     create_PPM_Image(ppm_image, width, height, MAX_GRAY_VALUE);

     // The height of one gradient region
     const int SIZE_Y = (int)(0.5 * height);

     // The width of gradient regions on the top of the image
     const int WIDTH_THIRD = (int)(width/3);

     // The width of gradient regions on the bottom of the image
     const int WIDTH_HALF = (int)(width/2);

     // The floating point change each step must take to create
the gradient
     const float STEP_PX = 255.0/(SIZE_Y - 1);


     /*** Top-left: red to white ***/


     // Red starts fully saturated with the the other values
non-present
     // These other values gradually increase until we reach
white (255, 255, 255)
     float cur_value = 0;
```

```
	for(int y = 0; y < SIZE_Y; y++){
		for(int x = 0; x < WIDTH_THIRD; x++){
			ppm_image->image[y][x][0] = 255;
			ppm_image->image[y][x][1] = (int)cur_value;
			ppm_image->image[y][x][2] = (int)cur_value;
		}
		cur_value += STEP_PX;
	}


	/*** Top-middle: white to green ***/


	// All colours start fully saturated to make white (255,
255, 255).
	// Red and blue gradually decrease until we reach green (0,
255, 0)
	cur_value = MAX_GRAY_VALUE;
	for(int y = 0; y < SIZE_Y; y++){
		for(int x = WIDTH_THIRD; x < 2*WIDTH_THIRD; x++){
			ppm_image->image[y][x][0] = (int)cur_value;
			ppm_image->image[y][x][1] = 255;
			ppm_image->image[y][x][2] = (int)cur_value;
		}
		cur_value -= STEP_PX;
	}


	/*** Top-right: blue to white ***/


	// Blue starts fully saturated with the the other values
non-present
	// These other values gradually increase until we reach
white (255, 255, 255)
	cur_value = 0;
	for(int y = 0; y < SIZE_Y; y++){
		for(int x = 2*WIDTH_THIRD; x < 3*WIDTH_THIRD; x++){
			ppm_image->image[y][x][0] = (int)cur_value;
			ppm_image->image[y][x][1] = (int)cur_value;
			ppm_image->image[y][x][2] = 255;
		}
		cur_value += STEP_PX;
	}


	/*** Bottom-left: black to white ***/
```

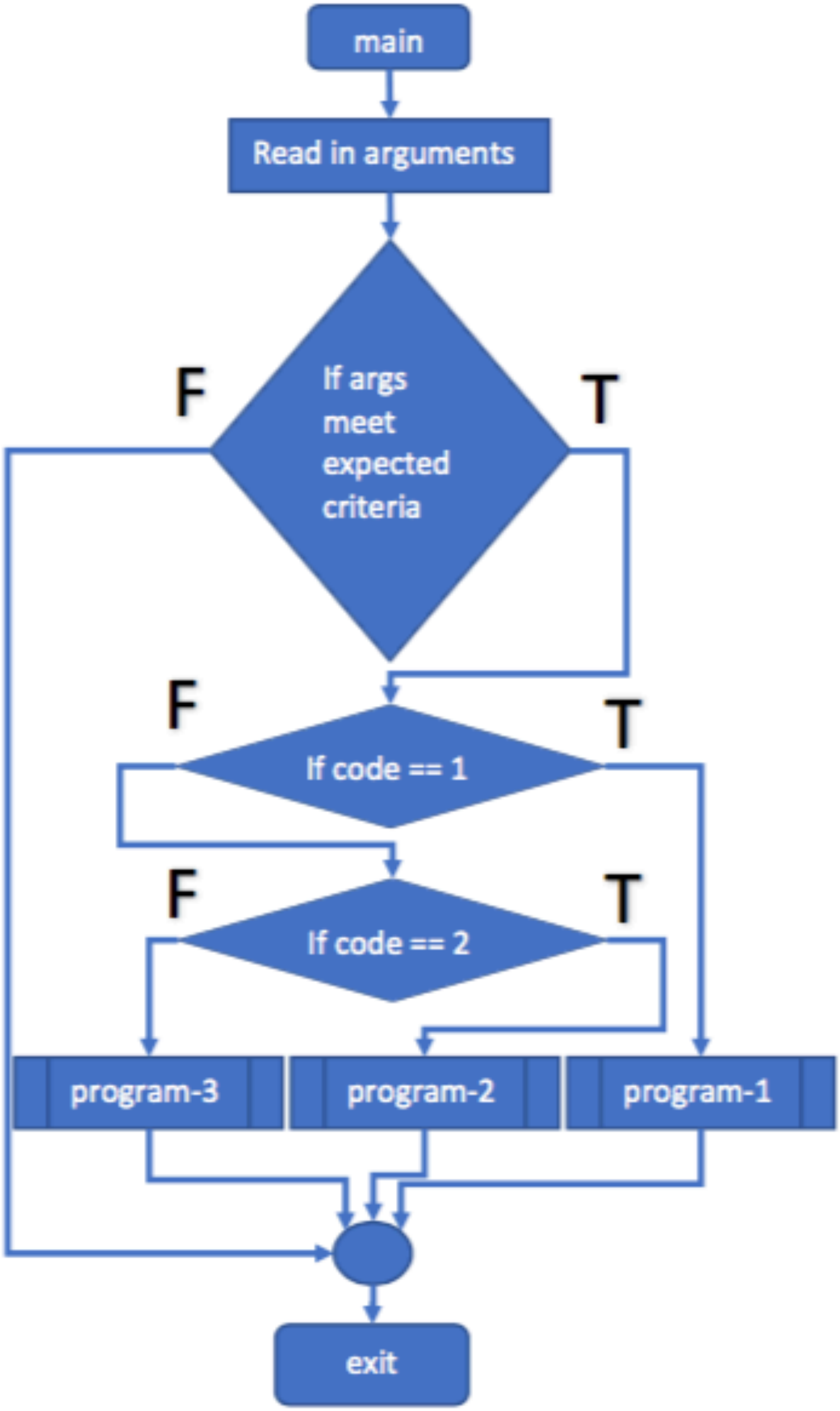
4481 Assignment #1
Paul Salvatore 250668447

```
	// Start at black and gradually increase each value until
we are at white
	cur_value = 0;
	for(int y = SIZE_Y; y < 2*SIZE_Y; y++){
		for(int x = 0; x < WIDTH_HALF; x++){
			ppm_image->image[y][x][0] = (int)cur_value;
			ppm_image->image[y][x][1] = (int)cur_value;
			ppm_image->image[y][x][2] = (int)cur_value;
		}
		cur_value += STEP_PX;
	}

	/*** Bottom-right: white to black ***/

	// Start at white and gradually decrease each value until
we are at black
	cur_value = MAX_GRAY_VALUE;
	for(int y = SIZE_Y; y < 2*SIZE_Y; y++){
		for(int x = WIDTH_HALF; x < 2*WIDTH_HALF; x++){
			ppm_image->image[y][x][0] = (int)cur_value;
			ppm_image->image[y][x][1] = (int)cur_value;
			ppm_image->image[y][x][2] = (int)cur_value;
		}
		cur_value -= STEP_PX;
	}

	/*** Copy ppm image to three pgm images ***/

	struct PGM_Image *pgm_image_1 = malloc(sizeof(struct
PGM_Image));
	struct PGM_Image *pgm_image_2 = malloc(sizeof(struct
PGM_Image));
	struct PGM_Image *pgm_image_3 = malloc(sizeof(struct
PGM_Image));

	copy_PPM_to_PGM(ppm_image, pgm_image_1, 0);  // red
	copy_PPM_to_PGM(ppm_image, pgm_image_2, 1); // green
	copy_PPM_to_PGM(ppm_image, pgm_image_3, 2); // blue

	// Save all of the images
	save_PPM_Image(ppm_image, image_name, image_format);
```

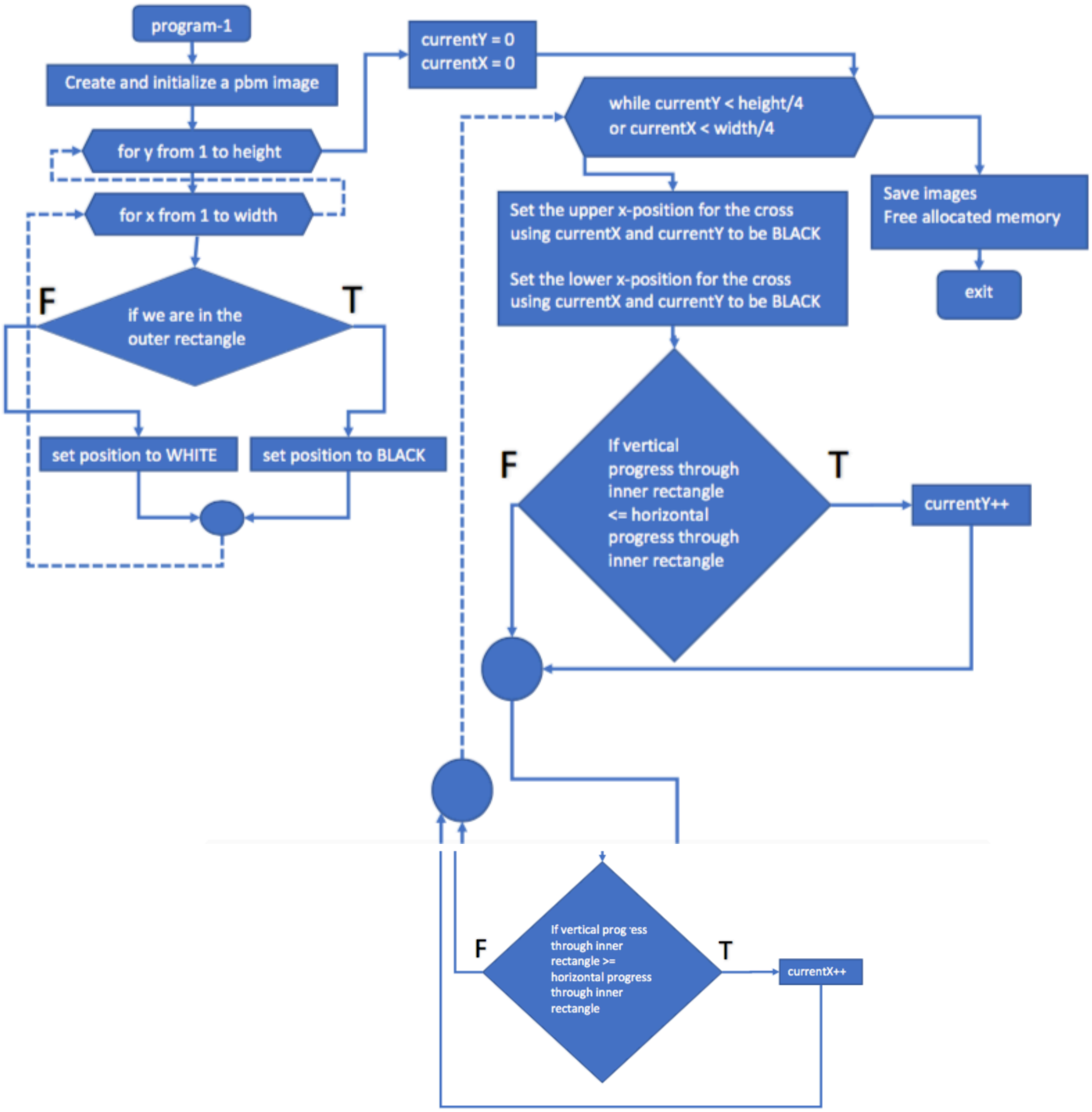
```
	save_PGM_Image(pgm_image_1, strcat(image_name,
"TO_PGM_RED.pgm"), image_format);
	save_PGM_Image(pgm_image_2, strcat(image_name,
"TO_PGM_GREEN.pgm"), image_format);
	save_PGM_Image(pgm_image_3, strcat(image_name,
"TO_PGM_BLUE.pgm"), image_format);

	// Free all allocated memory
	free_PPM_Image(ppm_image);
	free_PGM_Image(pgm_image_1);
	free_PGM_Image(pgm_image_2);
	free_PGM_Image(pgm_image_3);
	free(ppm_image);
	free(pgm_image_1);
	free(pgm_image_2);
	free(pgm_image_3);
}
```

4481 Assignment #1
Paul Salvatore 250668447

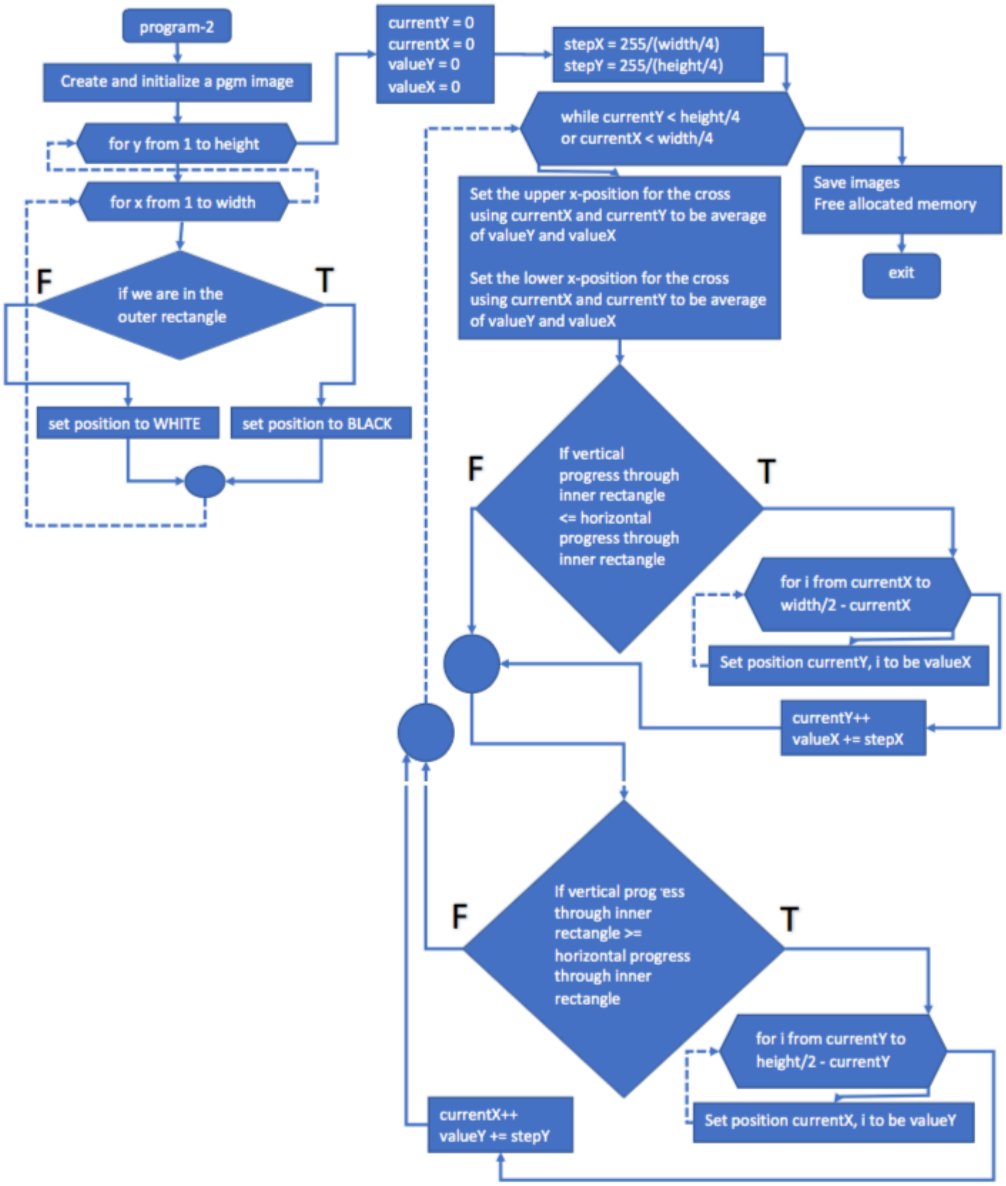
**** FLOWCHARTS ****



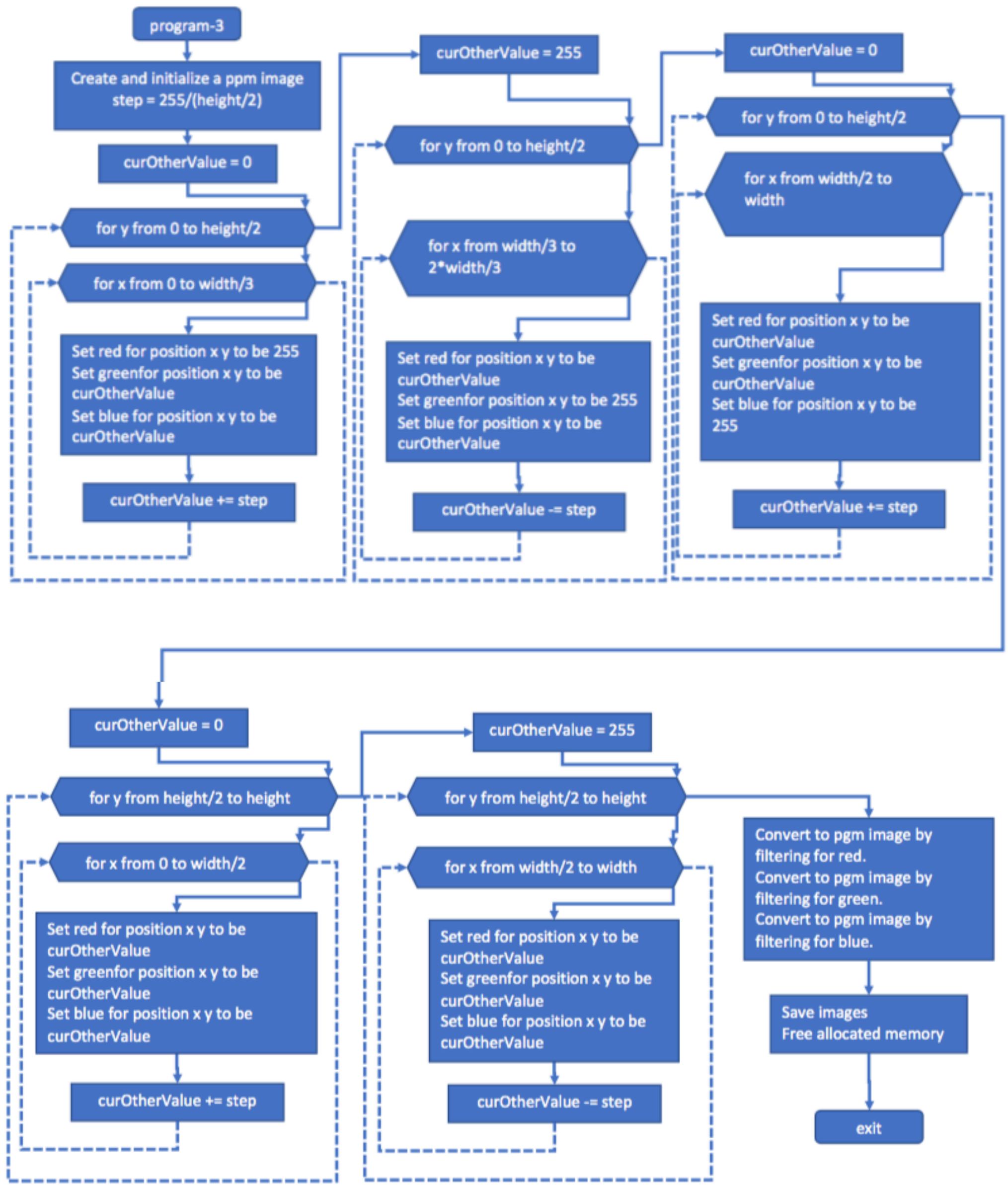
4481 Assignment #1
Paul Salvatore 250668447



4481 Assignment #1
Paul Salvatore 250668447



4481 Assignment #1
Paul Salvatore 250668447



4481 Assignment #1
Paul Salvatore 250668447

**** IMAGES ****

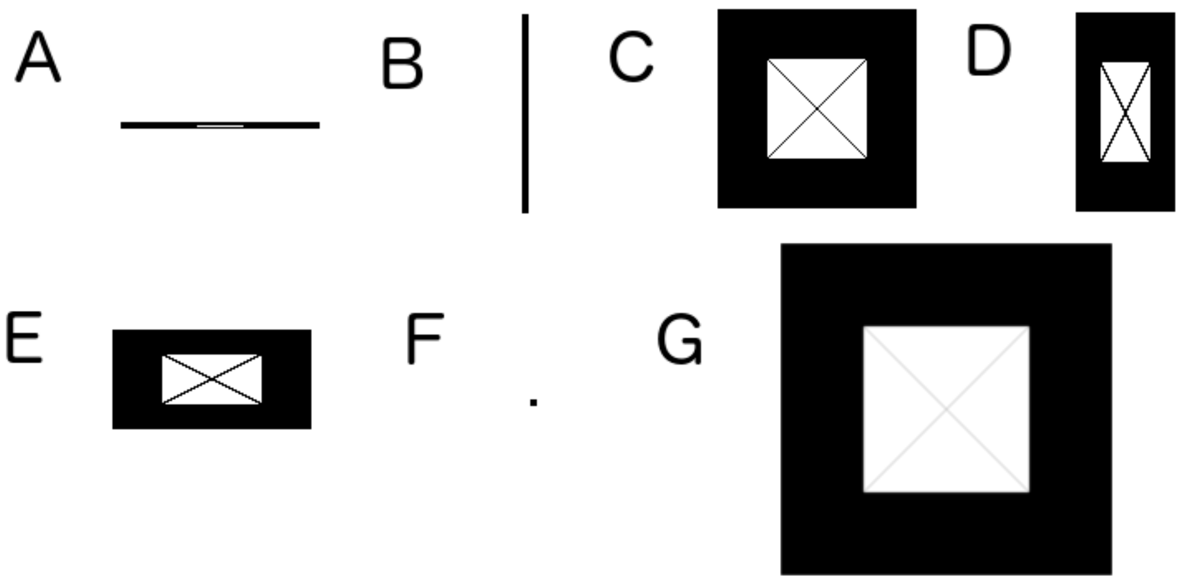


Figure 1. Test cases for pbm images
A test case for program-1, which creates an outer black rectangle and a centered, inner white rectangle half the size of the entire image, which is crossed by a solid black line from each top corner to the opposite bottom corner.

Shared parameters:
Parameters: type_code=1, format_code=0

a) 120x4_testCase.pbm
Specific Parameters: width=120, height=4, image_name=120x4_testCase.pbm

b) 4x120_testCase.pbm
Specific Parameters: width=4, height=120, image_name=4x120_testCase.pbm

c) 120x120_testCase.pbm
Specific Parameters: width=120, height=120, image_name=120x120_testCase.pbm

d) 60x120_testCase.pbm
Specific Parameters: width=60, height=120, image_name=60x120_testCase.pbm

e) 120x60_testCase.pbm
Specific Parameters: width=120, height=60, image_name=120x60_testCase.pbm

4481 Assignment #1
Paul Salvatore 250668447

f) 4x4_testCase.pbm
Specific Parameters: width=4, height=4,
image_name=4x4_testCase.pbm

g) 1200x1200_testCase.pbm
Specific Parameters: width=1200, height=1200,
image_name=1200x1200_testCase.pbm

4481 Assignment #1
Paul Salvatore 250668447

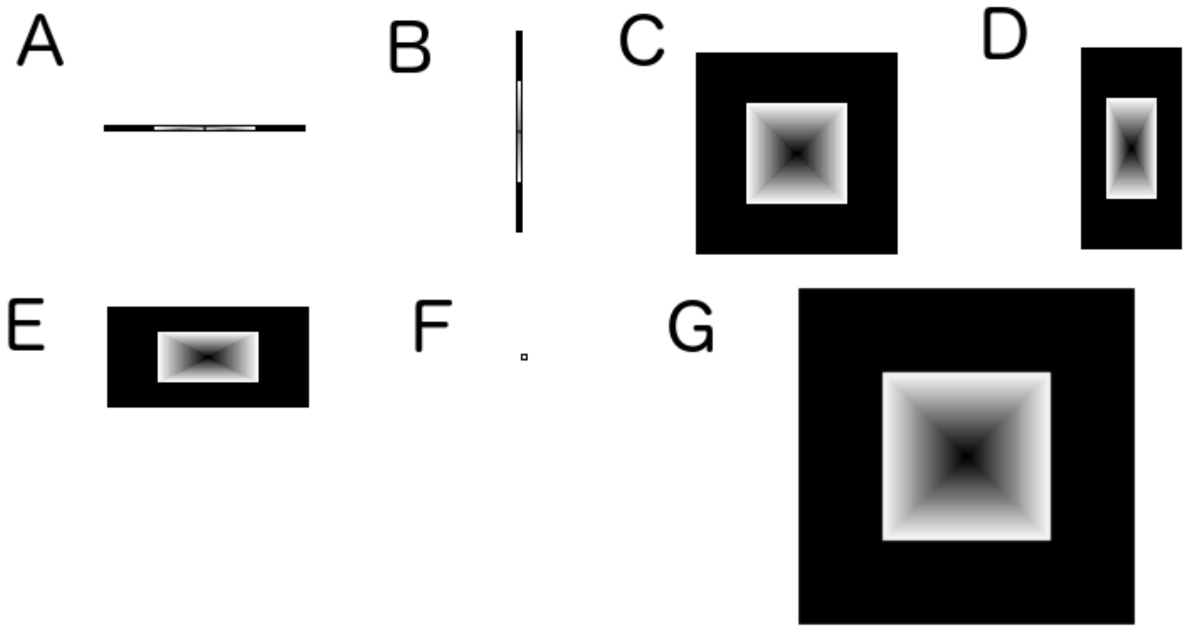


Figure 2. Test cases for pgm images
A test case for program-2, which creates an outer black rectangle and a centered, inner white rectangle half the size of the entire image, which is comprised of four triangles each with a consistent gradient from white at their base to black at their peak (the center of the image).

Shared parameters:
Parameters: type_code=2, format_code=0

a) 120x4_testCase.pgm
Specific Parameters: width=120, height=4, image_name=120x4_testCase.pgm

b) 4x120_testCase.pgm
Specific Parameters: width=4, height=120, image_name=4x120_testCase.pgm

c) 120x120_testCase.pgm
Specific Parameters: width=120, height=120, image_name=120x120_testCase.pgm

d) 60x120_testCase.pgm
Specific Parameters: width=60, height=120, image_name=60x120_testCase.pgm

e) 120x60_testCase.pgm
Specific Parameters: width=120, height=60, image_name=120x60_testCase.pgm

4481 Assignment #1
Paul Salvatore 250668447

f) 4x4_testCase.pgm
Specific Parameters: width=4, height=4,
image_name=4x4_testCase.pgm

g) 1200x1200_testCase.pgm
Specific Parameters: width=1200, height=1200,
image_name=1200x1200_testCase.pgm

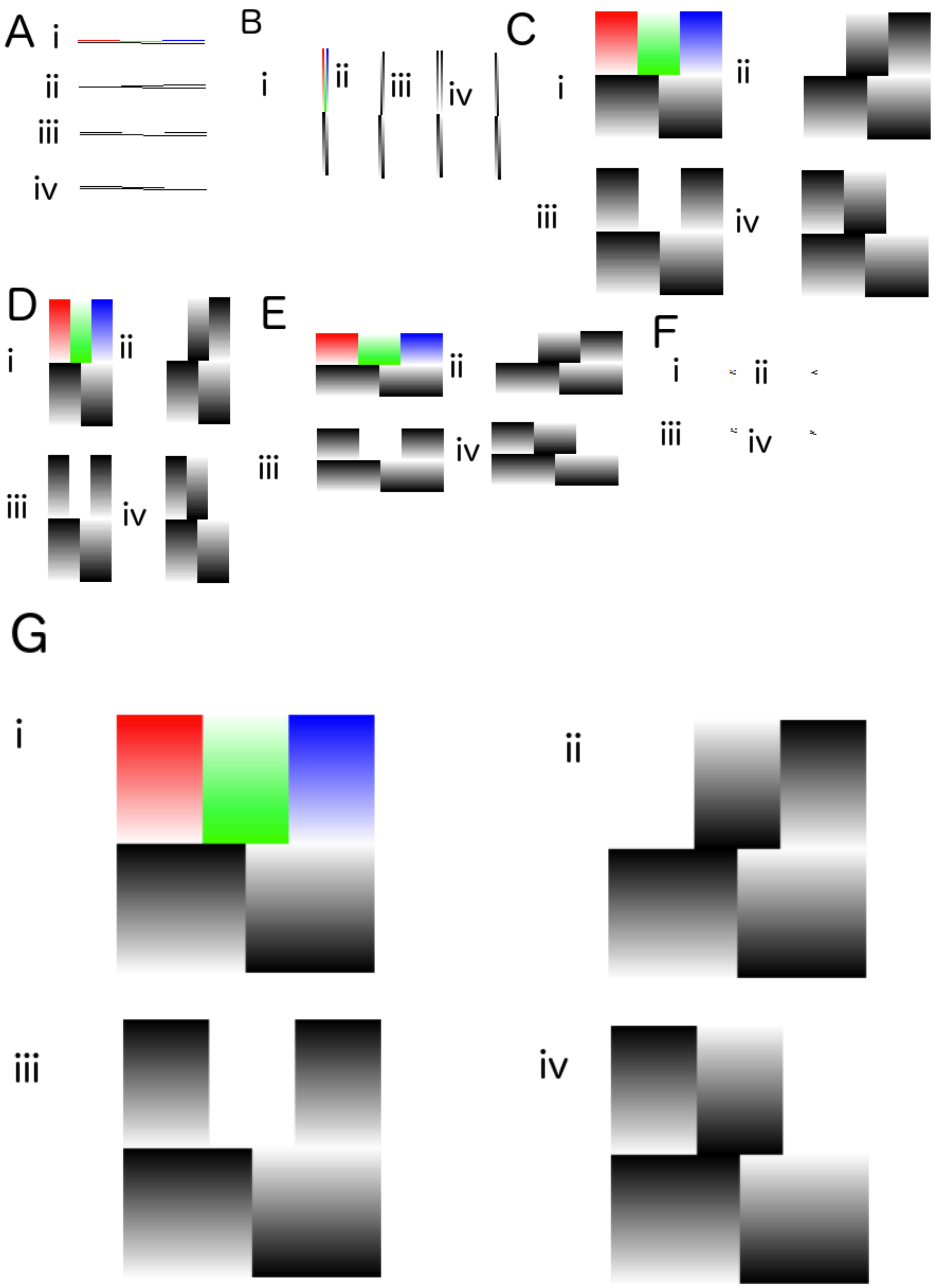


Figure 3. Test cases for ppm images
A test case for program-3, which creates a rectangular image with the following 5 gradient sections: red to white (top-left), white to green (top-middle), blue to white (top-left), black to white (bottom-left), and white to black (bottom-right). Also

4481 Assignment #1
Paul Salvatore 250668447

included are three versions of the image when converted to pgm images filter on a specific colour.

Shared parameters:
Parameters: type_code=3, format_code=0

a) 120x4_testCase.ppm
Specific Parameters: width=120, height=4, image_name=120x4_testCase.ppm

i) ppm image
ii) ppm to pgm image filtered for red
iii) ppm to pgm image filtered for green
iv) ppm to pgm image filtered for blue

b) 6x120_testCase.ppm
Specific Parameters: width=6, height=120, image_name=6x120_testCase.ppm

i) ppm image
ii) ppm to pgm image filtered for red
iii) ppm to pgm image filtered for green
iv) ppm to pgm image filtered for blue

c) 120x120_testCase.ppm
Specific Parameters: width=120, height=120, image_name=120x120_testCase.ppm

i) ppm image
ii) ppm to pgm image filtered for red
iii) ppm to pgm image filtered for green
iv) ppm to pgm image filtered for blue

d) 60x120_testCase.ppm
Specific Parameters: width=60, height=120, image_name=60x120_testCase.ppm

i) ppm image
ii) ppm to pgm image filtered for red
iii) ppm to pgm image filtered for green
iv) ppm to pgm image filtered for blue

e) 120x60_testCase.ppm
Specific Parameters: width=120, height=60, image_name=120x60_testCase.ppm

i) ppm image
ii) ppm to pgm image filtered for red
iii) ppm to pgm image filtered for green
iv) ppm to pgm image filtered for blue

4481 Assignment #1
Paul Salvatore 250668447

f) 6x4_testCase.ppm
Specific Parameters: width=6, height=4, image_name=6x4_testCase.ppm

i) ppm image
ii) ppm to pgm image filtered for red
iii) ppm to pgm image filtered for green
iv) ppm to pgm image filtered for blue

g) 1200x1200_testCase.ppm
Specific Parameters: width=1200, height=1200, image_name=1200x1200_testCase.ppm

i) ppm image
ii) ppm to pgm image filtered for red
iii) ppm to pgm image filtered for green
iv) ppm to pgm image filtered for blue

4481 Assignment #1
Paul Salvatore 250668447

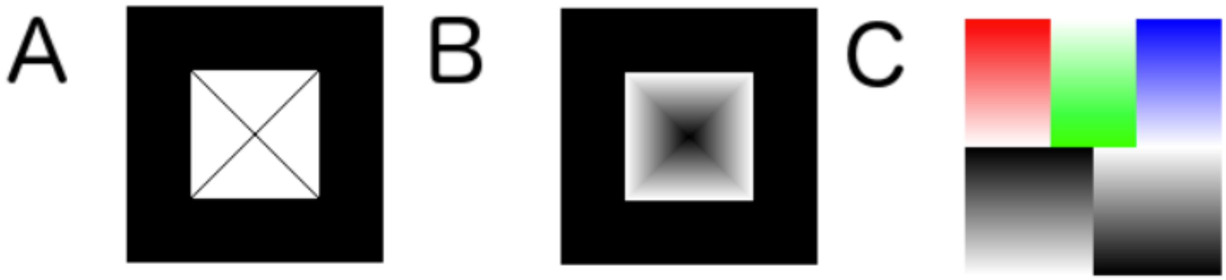


Figure 4. Test cases for binary images
A test case for program-1, program-2, and program-3. Please refer to figures 1-3 for the specifications of these programs.

Shared parameters:
Parameters: format_code=1

a) 120x120_testCase.pbm
Specific Parameters: type_code=1, width=120, height=120, image_name=120x120_testCase.pbm

b) 120x120_testCase.pgm
Specific Parameters: type_code=2, width=120, height=120, image_name=120x120_testCase.pgm

b) 120x120_testCase.ppm
Specific Parameters: type_code=3, width=120, height=120, image_name=120x120_testCase.ppm

4481 Assignment #1
Paul Salvatore 250668447

**** COMMENTS ON PPM TO PGM IMAGES (PROGRAM-3) ****

1) PPM to PGM filter on red

Since the top-left region has red permanently set to 255 (the max gray value), when we filter on red this region appears white since it is unchanging.

In the top-middle region, we gradually decrease the amount of red and blue from 255 to 0, in order to go from white to fully saturated green. Therefore, we can see the gradient of red going from white to black this region.

In the top-right region, we gradually increase the amount of red and green from 0 to 255, in order to go from fully saturated blue to white. Therefore, we can see the gradient of red going from black to white in this region.

The bottom half of the image contains equal parts red, green, and blue to create the grayscale gradient, therefore when filtered on red we see no change to this gradient.

2) PPM to PGM filter on green

In the top-left region, we gradually increase the amount of blue and green from 0 to 255, in order to go from fully saturated red to white. Therefore, we can see the gradient of green going from black to white in this region.

Since the top-middle region has green permanently set to 255 (the max gray value), when we filter on green this region appears white since it is unchanging.

In the top-right region, we gradually increase the amount of red and green from 0 to 255, in order to go from fully saturated blue to white. Therefore, we can see the gradient of green going from black to white in this region.

The bottom half of the image contains equal parts red, green, and blue to create the grayscale gradient, therefore when filtered on red we see no change to this gradient.

4481 Assignment #1
Paul Salvatore 250668447

3) PPM to PGM filter on blue

In the top-left region, we gradually increase the amount of blue and green from 0 to 255, in order to go from fully saturated red to white. Therefore, we can see the gradient of blue going from black to white in this region.

In the top-middle region, we gradually decrease the amount of red and blue from 255 to 0, in order to go from white to fully saturated green. Therefore, we can see the gradient of blue going from white to black this region.

Since the top-right region has blue permanently set to 255 (the max gray value), when we filter on blue this region appears white since it is unchanging.

The bottom half of the image contains equal parts red, green, and blue to create the grayscale gradient, therefore when filtered on red we see no change to this gradient.
when filtered on red we see no change to this gradient.

4481 Assignment #1
Paul Salvatore 250668447

**** WHAT IS HAPPENING IN THE 4X120 AND 120X4 CASES ****

4x120 or 6x120

a) pbm image

This image is entirely black because the two lines that intersect take up the entire 2x60 inner rectangle.

The first line moves from the top-left to the bottom-right corner of the inner rectangle, taking up the top-left and bottom right quadrants of the inner rectangle whereas the second line moves in the opposite direction taking up the top-right and bottom-left quadrants, therefore making the image appear black.

b) pgm image

Since there is no space for the triangles, the 2x60 inner rectangle is displaying the crossing lines explained in the pbm image example.

The difference is that the lines are not solid since as we progress through the image the colour of the lines changes since it is the average of the triangles current colour, which cannot be displayed due to space constraints, but would normally make up a gradient from white to black.

c) ppm image

This image appears as expected, all required gradients can be seen in full.

120x4

a) pbm image

Similar to the 4x120 example, the lines criss-cross in the 60x2 space, making the image appear entirely black.

b) pgm image

Similar to the 4x120 image, the lines criss-cross with a changing colour gradient as it relates to the expected colours of the triangles if the image was large enough for them to be drawn

4481 Assignment #1
Paul Salvatore 250668447

c) ppm image

This image shows the five distinct sections outlined in the instructions, however the gradient takes place in 1 step. Therefore, for example, in the top-left quadrant we move from red to white, but since there is only 2 available pixels the gradient goes directly from red to white. This can be seen in all other segments of the image.