**\*\*\* 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){

// 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);

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 \*\*\*/

// 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);

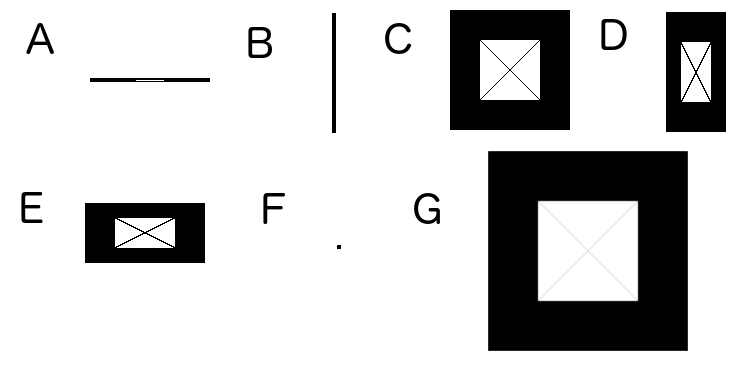
free(pgm\_image\_2);

free(pgm\_image\_3);

}

**\*\* FLOWCHARTS \*\***

**\*\* 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

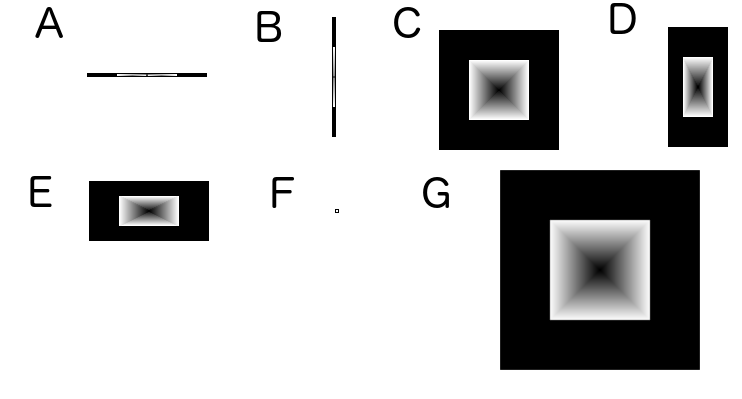
Specific Parameters: width=120, height=60, image\_name=120x60\_testCase.pbm

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



**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

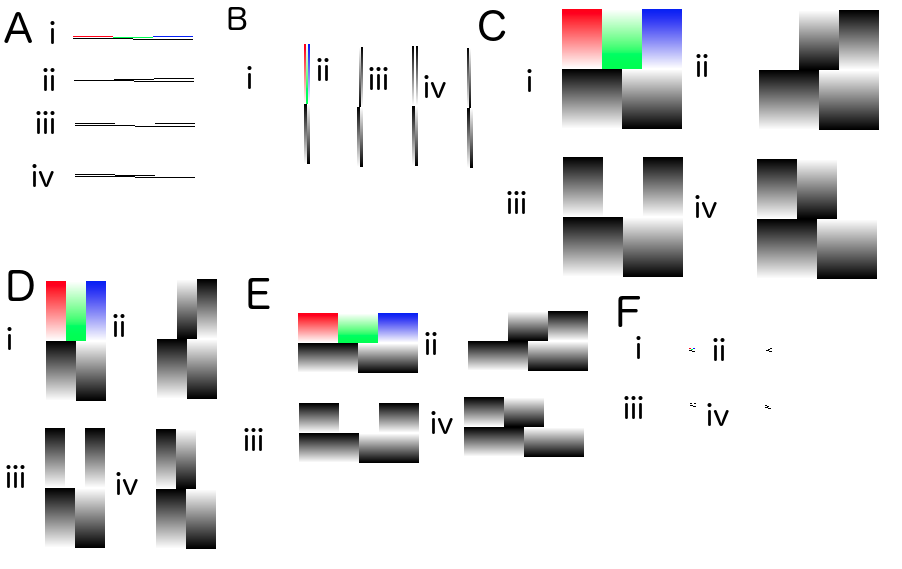
Specific Parameters: width=120, height=60, image\_name=120x60\_testCase.pgm

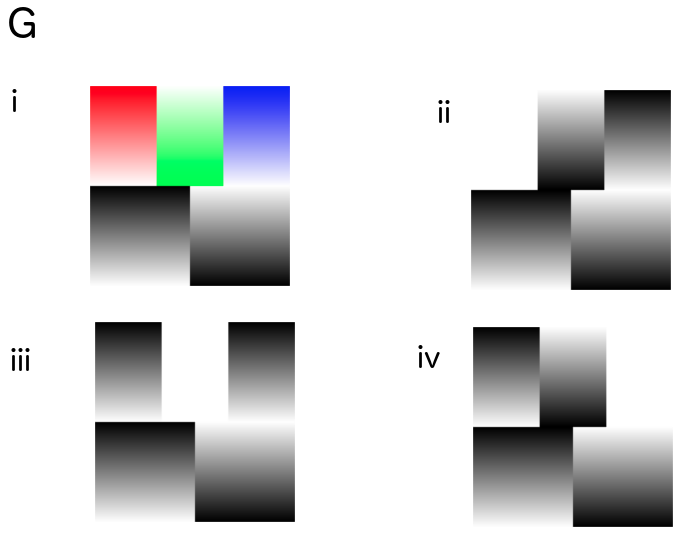
f) 4x4\_testCase.pgm

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 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

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

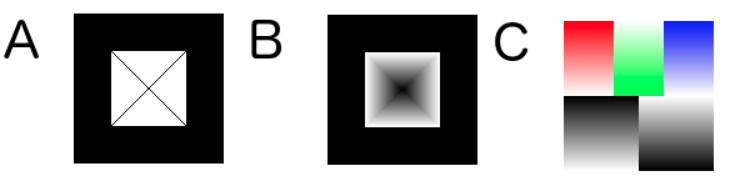
Specific Parameters: width=1200, height=1200, image\_name=1200x1200\_testCase.ppm

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



**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

**\*\* COMMENTS ON PPM TO PGM IMAGES \*\***