**SOLUTION**

C program to perform histogram equalisation to adjust contrast levels .

// library functions

#include <fcntl.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Function to perform histogram equalisation on an image

// Function takes total rows, columns, input file name and output

// file name as parameters

void histogramEqualisation(int cols, int rows, char\* input\_file\_name, char\* output\_file\_name)

{

// creating image pointer

unsigned char\* image;

// Declaring 2 arrays for storing histogram values (frequencies) and

// new gray level values (newly mapped pixel values as per algorithm)

inthist[256] = { 0 };

intnew\_gray\_level[256] = { 0 };

// Declaring other important variables

Int input\_file, output\_file, col, row, total, curr, i;

// allocating image array the size equivalent to number of columns

// of the image to read one row of an image at a time

image = (unsigned char\*)calloc(cols, sizeof(unsigned char));

// opening input file in Read Only Mode

input\_file = open(input\_file\_name, O\_RDONLY);

if (input\_file< 0) {

printf("Error opening input file\n");

exit(1);

}

// creating output file that has write and read access

output\_file = creat(output\_file\_name, 0666);

if (output\_file< 0) {

printf("Error creating output file\n");

exit(1);

}

// Calculating frequency of occurrence for all pixel values

for (row = 0; row < rows; row++) {

// reading a row of image

read(input\_file, &image[0], cols \* sizeof(unsigned char));

// logic for calculating histogram

for (col = 0; col < cols; col++)

hist[(int)image[col]]++;

}

// calculating total number of pixels

total = cols \* rows;

curr = 0;

// calculating cumulative frequency and new gray levels

for (i = 0; i< 256; i++) {

// cumulative frequency

curr += hist[i];

// calculating new gray level after multiplying by

// maximum gray count which is 255 and dividing by

// total number of pixels

new\_gray\_level[i] = round((((float)curr) \* 255) / total);

}

// closing file

close(input\_file);

// reopening file in Read Only Mode

input\_file = open(input\_file\_name, O\_RDONLY);

// performing histogram equalisation by mapping new gray levels

for (row = 0; row < rows; row++) {

// reading a row of image

read(input\_file, &image[0], cols \* sizeof(unsigned char));

// mapping to new gray level values

for (col = 0; col < cols; col++)

image[col] = (unsigned char)new\_gray\_level[image[col]];

// reading new gray level mapped row of image

write(output\_file, &image[0], cols \* sizeof(unsigned char));

}

// freeing dynamically allocated memory

free(image);

// closing input and output files

close(input\_file);

close(output\_file);

}

// driver code

int main()

{

// declaring variables

char\* input\_file\_name;

char\* output\_file\_name;

int cols, rows;

// defining number of rows and columns in an image

// here, image size is 512\*512

cols = 512;

rows = 512;

// defining input file name (input image name)

// this boat\_512\_512 is a raw grayscale image

input\_file\_name = "boat\_512\_512";

// defining output file name (output image name)

output\_file\_name = "boat\_512\_512\_histogram\_equalised";

// calling function to do histogram equalisation

histogramEqualisation(cols, rows, input\_file\_name, output\_file\_name);

return 0;

}