```c
#include "helpers.h"
#include <math.h>

// Protótipo das funções.
void verification (int *a);
int se(int line, int column, int height, int width);

// Converte a imagem para escala de cinza(grayscale).
void grayscale(int height, int width, RGBTRIPLE image[height][width])
{
    // Controla a linha que estamos;
    for(int i = 0; i  < height; i++)
    {
        // Passa por todas as colunas da imagem;
        for(int j = 0; j < width; j++)
        {
            // Faz a média das 3 cores e arredonda o valor;
            int number = round((float)(image[i][j].rgbtBlue +
image[i][j].rgbtGreen + image[i][j].rgbtRed) / 3);

            // Iguala todas as cores no valor da média;
            image[i][j].rgbtBlue = number;
            image[i][j].rgbtGreen = number;
            image[i][j].rgbtRed = number;
        }
    }
    return;
}

// Reflete a imagem horizontalmente.
void reflect(int height, int width, RGBTRIPLE image[height][width])
{
    // Controla a linha que estamos;
    for(int i = 0; i < height; i++)
    {
        // Passa por todas as colunas da imagem;
        for(int j = 0; j < width; j++)
        {
            // Altera o pixel que estamos com o seu oposto;
            RGBTRIPLE temp = image[i][j];
            image[i][j] = image[i][width - j - 1];
            image[i][width - j - 1] = temp;
        }
    }
    return;
}

// Borra a imagem (Blur).
void blur(int height, int width, RGBTRIPLE image[height][width])
```

```
{
    // Variável para armazenar a imagem;
    RGBTRIPLE copia[height][width];

    // Controla a linha que estamos;
    for(int i = 0; i < height; i++)
    {
        // Passa por todas colunas da imagem;
        for(int j = 0; j < width; j++)
        {
            // Copia o pixel da imagem e salva na nova variável.
            copy[i][j] = image[i][j];
        }
    }

    // Controla a linha que estamos;
    for(int i = 0; i< height; i++)
    {
        // Controla a coluna que estamos;
        for(int j = 0; j < width; j++)
        {
            // Variáveis de controle;
            int sum[3] = {0, 0, 0}, count = 0;

            // Pasa pela linha anterior, atual e posterior ao pixel que
estamos;
            for(int line = i - 1; line < i + 2; line++)
            {
                // Passa pela coluna anterior, atual e posterior ao pixel
que estamos;
                for(int column = j -1; column <  j + 2; column++)
                {
                    // Verifica se o pixel deve ou não ser somado;
                    if(se(line, column, height, width) == 0)
                    {
                        // Soma cada cor em uma parte do array;
                        sum[0] += copy[line][column].rgbtRed;
                        sum[1] += copy[line][column].rgbtGreen;
                        sum[2] += copy[line][column].rgbtBlue;
                        count++;
                    }
                }
            }

            // Tira a média e arredonda a mesma;
            for(int k = 0; k < 3; k++)
            {
                sum[k] = round((float)sum[k] / count);
                verification(&sum[k]);
```

```
            }

            // Salva os novos valores na imagem.
            image[i][j].rgbtRed = sum[0];
            image[i][j].rgbtGreen = sum[1];
            image[i][j].rgbtBlue = sum[2];
        }
    }
    return;
}

// Detecta linhas, contornos (Edge).
void edges(int height, int width, RGBTRIPLE image[height][width])
{
    // Variável para armazenar a imagem;
    RGBTRIPLE copia[height][width];

    // Controla a linha que estamos;
    for(int i = 0; i < height; i++)
    {
        // Passa por todas colunas da imagem;
        for(int j = 0; j < width; j++)
        {
            // Copia o pixel da imagem e salva na nova variável.
            copy[i][j] = image[i][j];
        }
    }

    // Matriz de valores para multiplicação.
    int gx[3][3] = {{-1, 0, 1}, {-2, 0, 2},{-1, 0, 1}}, gy[3][3] = {{-1,
-2, -1}, {0, 0, 0}, {1, 2, 1}};

    // Controla a linha em que estamos;
    for(int i = 0; i < height; i++)
    {
        // Controla a coluna em que estamos;
        for(int j = 0; j < width; j++)
        {
            // Arrays para soma dos valores das cores; ([0] = Red, [1] =
Green, [2] = Blue)
            // Obs.: A Variável countx, controla o valor da linha que
estamos verificando da matriz 3x3 para gx e gy;
            int sumgx[3] = {0, 0, 0}, sumgy[3] = {0, 0, 0},  countx = 0;

            // Controla a linha da matriz 3x3 para os pixels;
            for(int line = i - 1; line < i + 2; line++)
            {
                // Variável de controle da coluna em que estamos na
matriz 3x3 para gx e gy;
```

```
                int county = 0;

                // Controla a coluna da matriz 3x3, para os pixels;
                for(int column = j - 1; column < j + 2; column++)
                {
                    // Chama a função "se" se o resultado dela for 0;
soma os valores solicitados em seus arrays;
                    if(se(line, column, height, width) == 0)
                    {
                        // Soma = soma + (quantidade de cor * valor de gx
ou gy correspondente na matriz 3x3);
                        sumgx[0] += copia[line][column].rgbtRed *
                        gx[countx][county];

                        sumgx[1] += copia[line][column].rgbtGreen *
                        gx[countx][county];

                        sumgx[2] += copia[line][column].rgbtBlue *
                        gx[countx][county];

                        sumgy[0] += copia[line][column].rgbtRed *
                        gy[countx][county];

                        sumgy[1] += copia[line][column].rgbtGreen *
                        gy[countx][county];

                        sumgy[2] += copia[line][column].rgbtBlue *
                        gy[countx][county];
                    }
                    county++;
                }
                countx++;
            }

            // Passa pelos 3 valores do array juntando gx e gy e
arredondando o valor.
            // g = raiz quadrada  (gx^2 + gy^2);
            // Chama a função verificação;
            for(int k = 0; k < 3; k++)
            {
                sumgx[k] = round(sqrt(pow((float)sumgx[k], 2) +
pow(sumgy[k], 2)));
                verification(&sumgx[k]);
            }

            // Após os valores estarem corretos, ele atribui ao pixel
seus novos valores.
            image[i][j].rgbtRed = sumgx[0];
            image[i][j].rgbtGreen = sumgx[1];
```

```
            image[i][j].rgbtBlue = sumgx[2];
        }
    }
    return;
}


// Verifica se o valor da soma é menor que 0x00(0) ou maior que
0xff(255), atribuindo seus limites caso ultrapassem os mesmos.
void verification (int *a)
{
    if(*a < 0x00)
    {
        *a = 0x00;
    }
    else if (*a > 0xff)
    {
        *a = 0xff;
    }
    else
    {
        *a = *a;
    }
}

// Verifica se o pixel analisado está dentro da imagem.
int se(int line, int column, int height, int width)
{
    if(line < 0 || line >= height)
    {
        return 1;
    }
    else if(column < 0 || column >= width)
    {
        return 1;
    }

    return 0;
}
```