Writing the reduced image data:

A for loop is used to iterate over each pixel of an image represented as a two-dimensional array.

In the loop, the pixel index (k) in the original input image is calculated using the formula k=i*2*rowSize+j*2*3 where “i” is the row index, “j” is the column index, and rowSize is the number of bytes in a row of the original image. On the other hand, the pixel index(k_reduced) in the reduced image is calculated using the formula i*(r_rowSize)+j*3, where r_rowSize is the number of bytes in a row of the reduced image. Lastly the blue, green, and red levels from the original input image is copied to the reduced image with the following lines of code:

```
rimageData[k_reduced]=imageData[k];// Copy blue level.
rimageData[k_reduced+1]=imageData[k+1];// Copy green level.
rimageData[k_reduced+2]=imageData[k+2];// Copy red level.
```

Writing the merged image:

k2=(i-frame_size*2-rHeight+1)*r_rowSize+(j-frame_size+1)*3;

> **(i-frame_size*2-rHeight+1)** adjusts the row index (i) to account for the frame size and the height of the reduced image. It shifts the row index to the appropriate position within the first quadrant.
>
> ***r_rowSize** calculates the offset in bytes needed to reach the correct row within the reduced image data array.
>
> **(j-frame_size+1)*3** calculates the offset in bytes needed to reach the correct column (in pixel) within the reduced image data array. Since each pixel has three color components (RGB), multiplying by 3 ensures the correct positioning within a row.

k1=(i-frame_size*2-rHeight+1)*r_rowSize+((rWidth-1)-(j-2*frame_size-rWidth))*3;

> **(j-2*frame_size-rWidth):** This part subtracts the frame size and the width of the reduced image from the column index j. It effectively mirrors the column indices from the first quadrant to the corresponding indices in the second quadrant.

k3=((rHeight-1)-(i-frame_size))*r_rowSize+(j-frame_size+1)*3;

((rHeight-1)-(i-frame_size)) adjusts the row index “i” to reflect its position within the third quadrant.

i-frame_size subtracts the frame size from the row index “i”, effectively shifting the rows upward to accommodate the frame.

(rHeight-1) represents the maximum row index in the reduced image. By subtracting the adjusted row index from **(rHeight-1)**, we mirror the row indices to fit within the third quadrant.

k4=((rHeight-1)-(i-frame_size))*r_rowSize+((rWidth-1)-(j-2*frame_size-rWidth))*3;

Flips the reduced image vertically and horizontally.

if(i>rHeight+frame_size*2-1&&i<rHeight*2+frame_size*2-1&&j>rWidth+frame_size*2-1&&j<rWidth*2+frame_size*2-1)

else if(i>rHeight+frame_size*2-1&&i<rHeight*2+frame_size*2-1&&j>frame_size-1 &&j<rWidth+frame_size-1)

else if(i>frame_size-1&&i<rHeight+frame_size-1&&j>frame_size-1 &&j<rWidth+frame_size-1)

else if(i>frame_size-1&&i<rHeight+frame_size-1&&j>rWidth+frame_size*2-1 &&j<rWidth*2+frame_size*2-1)

These four if else statements decide which quadrant of the merged image the program is currently editing.

```
else{
        merged_imageData[k]=B;
        merged_imageData[k+1]=G;
        merged_imageData[k+2]=R;
}
```

This block of code prints the RGB value of the frame outside the areas of the four quadrants