Memoria Estática. Ejercicio 3

```
void gris(pixel* matriz, short n, uint8_t* resultado)
```

Dada una matriz de n x n píxeles RGB (1 byte por componente), hacer una función que convierta los pixeles a escala de grises usando la fórmula:

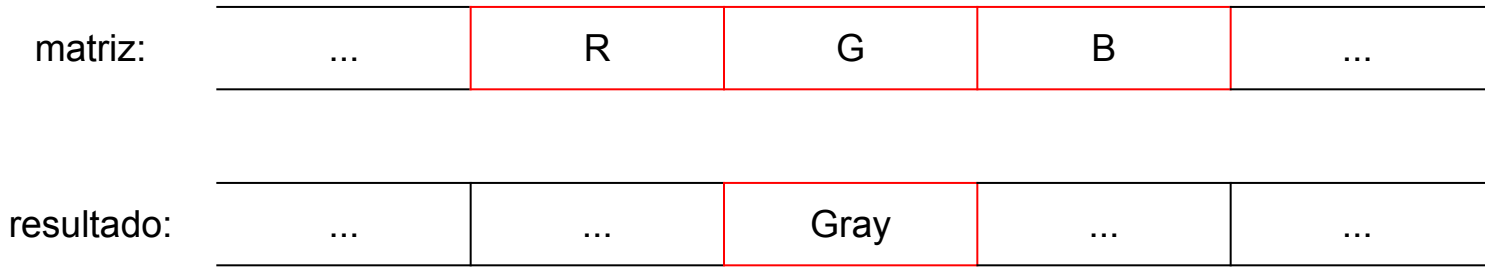
$$Gray = (R + 2 \times G + B) / 4.$$

Memoria Estática. Ejercicio 3

```
void gris(pixel* matriz, short n, uint8_t* resultado)
```

Dada una matriz de n x n píxeles RGB (1 byte por componente), hacer una función que convierta los pixeles a escala de grises usando la fórmula:

$$Gray = (R + 2 \times G + B) / 4.$$

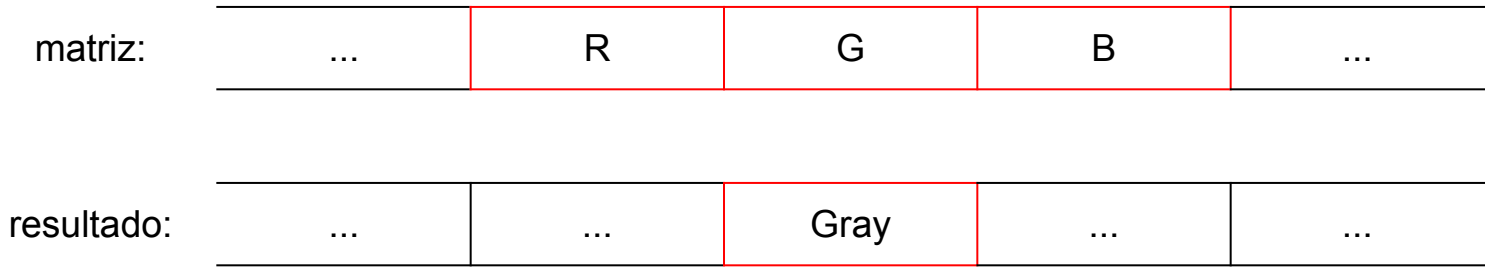


Memoria Estática. Ejercicio 3

```
void gris(uint8_t* matriz, short n, uint8_t* resultado)
```

Dada una matriz de n x n píxeles RGB (1 byte por componente), hacer una función que convierta los pixeles a escala de grises usando la fórmula:

$$Gray = (R + 2 \times G + B) / 4.$$



Memoria Estática. Ejercicio 3 - Explicación larga

```
void gris(pixel* matriz, short n, uint8_t* resultado)
```

Dada una matriz de n x n píxeles RGB (1 byte por componente), hacer una función que convierta los pixeles a escala de grises usando la fórmula:

$$Gray = (R + 2 \times G + B) / 4.$$

A continuación vamos a explicar dos partes del problema:

1. La representación de los datos.
2. La estrategia para resolver el enunciado.

Representación del problema

```
void gris(pixel* matriz, short n, uint8_t* resultado)
```

Dada una matriz de n x n píxeles RGB (1 byte por componente), hacer una función que convierta los pixeles a escala de grises usando la fórmula: ...

- Un píxel contiene colores.
  - Un pixel RGB contiene los colores rojo, verde y azul.
  - Un pixel en escala de gris contiene al color gris.

- Cada color ocupa 1 byte.
  - Un pixel RGB ocupa 3 bytes.

  - Un pixel en escala de gris ocupa 1 byte.

Representación del problema

Recordando un poco cómo se representa una matriz en memoria.

- Del concepto abstracto de una matriz de elementos

e(0, 0)	e(0, 1)	e(0, 2)
e(1, 0)	e(1, 1)	e(1, 2)
e(2, 0)	e(2, 1)	e(2, 2)

- Pasamos a un vector de vector. En donde los elementos se ordenan de forma lineal, pero representan datos en dos dimensiones.

...	e(0, 0)	e(0, 1)	e(0, 2)	e(1, 0)	e(1, 1)	e(1, 2)	e(2, 0)	e(2, 1)	e(2, 2)	...

Representación del problema

- Ahora bien, si tengo un vector de elementos, y cada elemento es un pixel puedo decir que tengo un vector de pixeles como el siguiente:

...	Px(0)	Px(1)	Px(2)	Px(3)	Px(4)	Px(5)	Px(6)	Px(7)	Px(8)	...

- Si el vector de pixeles cuemple con la condición de tener los mismos elementos que la siguiente matriz:

Px(0)	Px(1)	Px(2)
Px(3)	Px(4)	Px(5)
Px(6)	Px(7)	Px(8)

- Puede pensar que estoy trabajando verdaderamente con una matriz de píxeles.
- Cómo quedan ordenados los componentes de colores en esta matriz?

Representación del problema

- Un pixel RGB en memoria

Memoria:	...	R	G	B	...

- Vector de píxeles RGB

...	R(0)	G(0)	B(0)	R(1)	G(1)	B(1)	...

- Matriz de píxeles RGB

R(0)	G(0)	B(0)	R(1)	G(1)	B(1)	R(2)	G(2)	B(2)
R(3)	G(3)	B(3)	R(4)	G(4)	B(4)	R(5)	G(5)	B(5)
R(6)	G(6)	B(6)	R(7)	G(7)	B(7)	R(8)	G(8)	B(8)

- Podemos tener otra representación? SI, muchas más! esta representación es muy práctica para resolver este problema.
- Más adelante en la materia vamos a ver "estructuras" en C que nos ayudan a generar mejores representaciones.

Representación del problema

Con el color gris es más fácil.

- Un pixel Gris en memoria

Memoria:	...	...	G	...	...

- Vector de píxeles grises

...	G(0)	G(1)	G(2)	G(3)	G(4)	G(5)	...

- Matriz de píxeles grises

G(0)	G(1)	G(2)
G(3)	G(4)	G(5)
G(6)	G(7)	G(8)

Representación del problema

```
void gris(pixel* matriz, short n, uint8_t* resultado)
```

En base a la representación del problema presentada:

¿Qué cambios podemos hacer en el prototipo de la función?

Representación del problema

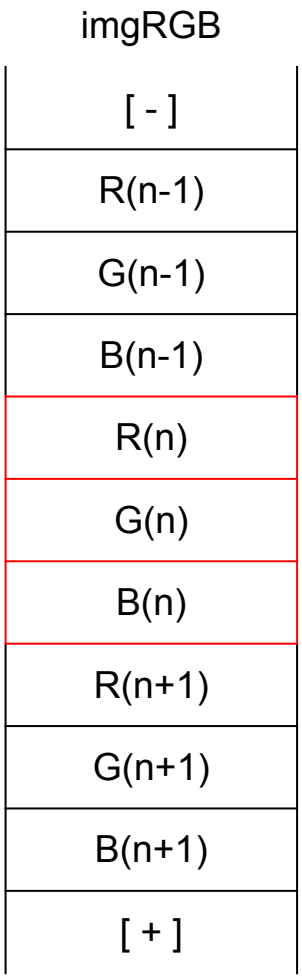
```
void gris(pixel* matriz, short n, uint8_t* resultado)
```

En base a la representación del problema presentada:

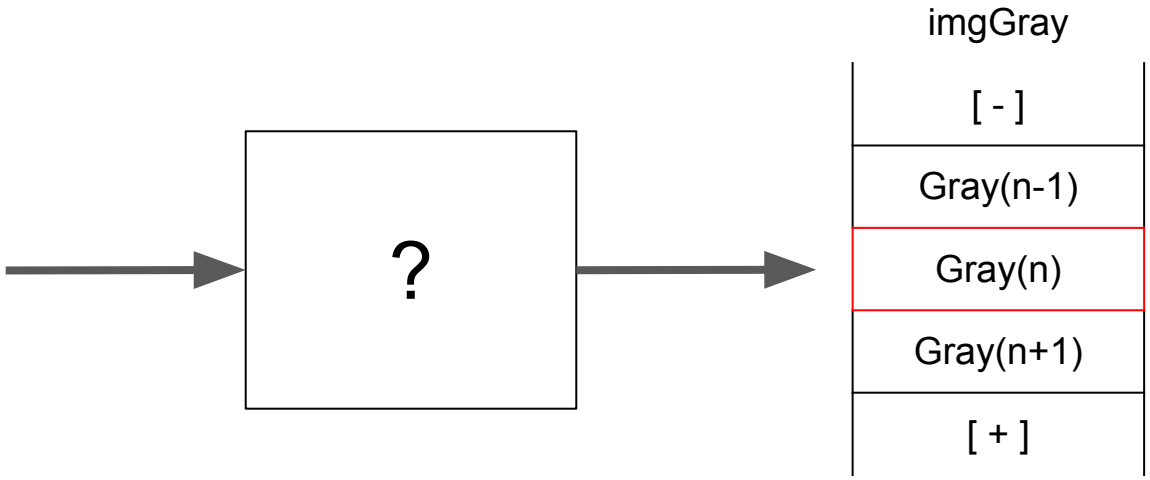
¿Qué cambios podemos hacer en el prototipo de la función?

```
void gris(uint8_t* matriz, short n, uint8_t* resultado)
```

Estrategia

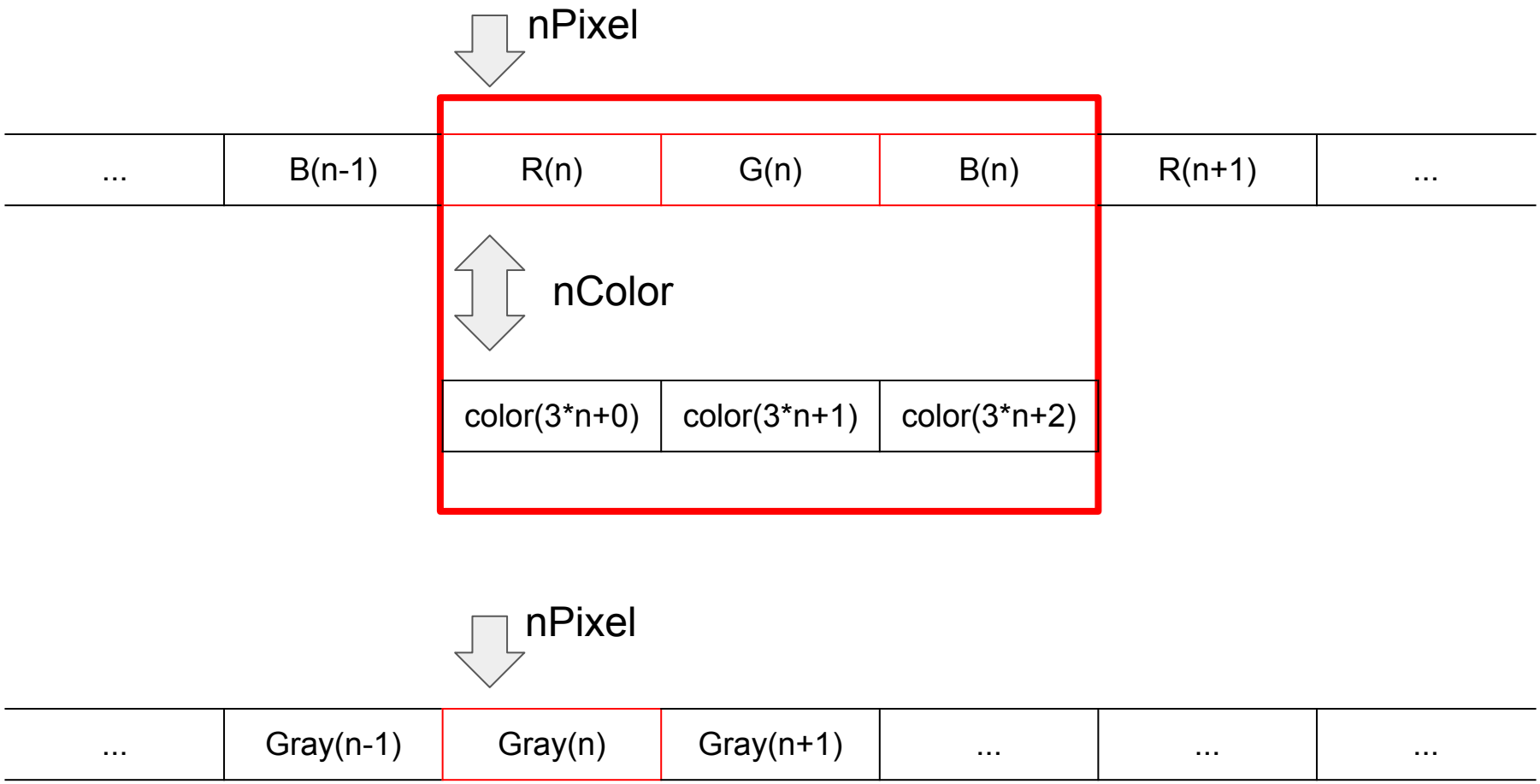


- Tenemos dos espacios en memoria.
- Tenemos que construir un algoritmo que procese los datos de un lado y los coloque en otro.c

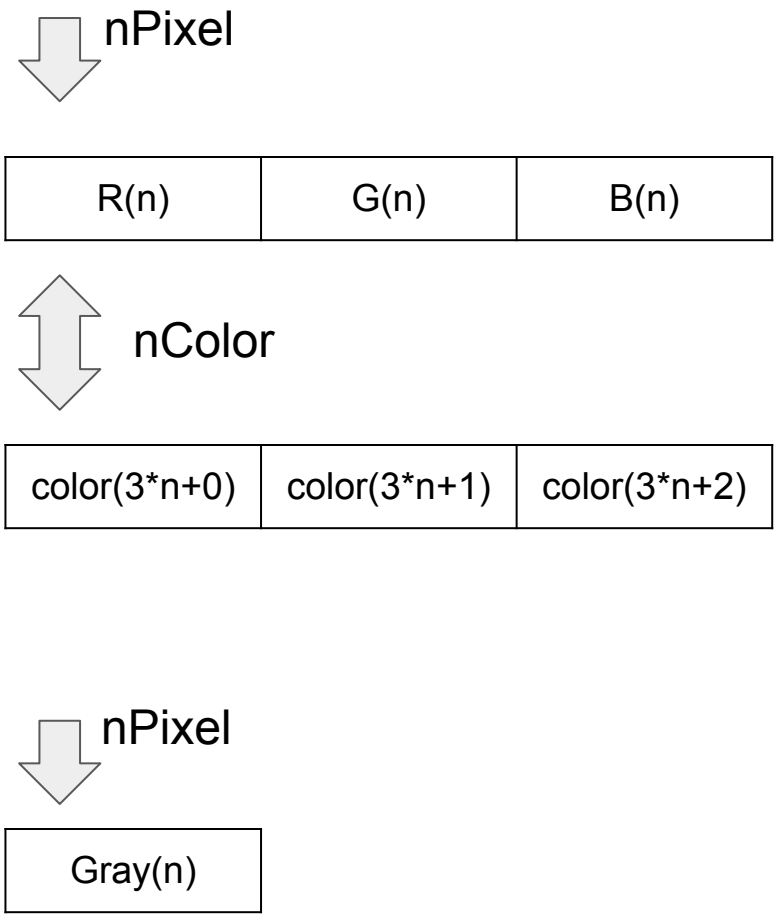


Estrategia

- La idea es tener dos índices.
- Un índice al pixel y otro al color.



Estrategia



El acceso a la memoria y la utilización de los punteros sería la siguiente:

```
R = color(3*n + 0) = R(n)
G = color(3*n + 1) = G(n)
B = color(3*n + 2) = B(n)

gray = (R + 2*G + B) / 4
```

Gray(n) = `gray`

Estrategia

No hay que olvidar que los índices siempre hacen referencia a la dirección de inicio de la matriz.