

TDA Imagen

Generado por Doxygen 1.8.17



## Chapter 1

# Índice de clases

### 1.1 Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

[Imagen](#)

Clase que representa una imagen digital en escala de grises . . . . . ??



## Chapter 2

# Indice de archivos

### 2.1 Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

inc/ <a href="#">funciones_imagen.h</a>	Archivo con la suite de funciones sobre el <i>TDA_Imagen</i> . . . . .	??
inc/ <a href="#">imagenES.h</a>	Fichero cabecera para la E/S de imágenes . . . . .	??
src/ <a href="#">funciones_imagen.cpp</a>	Archivo con las definiciones de la suite de funciones sobre el <i>TDA_Imagen</i> . . . . .	??
src/ <a href="#">imagenES.cpp</a>	Fichero con definiciones para la E/S de imágenes Permite la E/S de archivos de tipo PGM,PPM	??
src/ <a href="#">main.cpp</a>	Archivo que se encarga de transformar imagenes a color a escala de grises . . . . .	??



## Chapter 3

# Documentación de las clases

### 3.1 Referencia de la Clase Imagen

clase que representa una imagen digital en escala de grises

```
#include <imagenES.h>
```

#### Métodos públicos

- `Imagen` (int filas, int columnas)  
*constructor de la clase `Imagen`.*
- `Imagen` (const `Imagen` &otra)  
*constructor de copia de la clase `Imagen`*
- `Imagen` (int filas, int columnas, `byte` \*vector)  
*constructor de la clase `imagen`*
- `~Imagen` ()  
*destructor de la clase `Imagen`.*
- `Imagen` & `operator=` (const `Imagen` &otra)  
*operador de asignación entre imágenes.*
- int `num_filas` () const  
*Calcula el número de filas de la imagen.*
- int `num_columnas` () const  
*Calcula el número de columnas de la imagen.*
- void `asigna_pixel` (int fila, int columna, `byte` valor)  
*Asigna el valor `valor` al pixel indicado por fila y columna de la imagen.*
- `byte valor_pixel` (int fila, int columna) const  
*Consulta el valor de un pixel de la imagen.*

#### 3.1.1 Descripción detallada

clase que representa una imagen digital en escala de grises

### 3.1.2 operaciones

Se definen una serie de operaciones:

1. Creación de una imagen
2. Destrucción de una imagen
3. Consultar el número de filas
4. Consultar el número de columnas
5. Asignar un valor a un punto de la imagen
6. Consultar el valor de un punto de la imagen

### 3.1.3 Documentación del constructor y destructor

#### 3.1.3.1 `Imagen()` [1/3]

```
Imagen::Imagen (
    int  filas,
    int  columnas )
```

constructor de la clase [Imagen](#).

##### Parámetros

<i>filas</i>	el número de filas que tendrá la imagen
<i>columnas</i>	el número de columnas que tendrá la imagen

##### Devuelve

el objeto nuevo de imagen esta inicializado a una imagen en negro

#### MÉTODOS DE LA CLASE IMAGEN

#### 3.1.3.2 `Imagen()` [2/3]

```
Imagen::Imagen (
    const Imagen & otra )
```

constructor de copia de la clase [Imagen](#)

##### Parámetros

<i>otra</i>	la imagen que copiamos
-------------	------------------------



**Devuelve**

el nuevo objeto, copia de *otra*

**3.1.3.3 Imagen() [3/3]**

```
Imagen::Imagen (
    int filas,
    int columnas,
    byte * vector )
```

constructor de la clase imagen

**Parámetros**

<i>filas</i>	el numero de filas que va a tener
<i>columnas</i>	el numero de columnas que va a tener la imagen
<i>vector</i>	vector unidimensional donde se almacenan todos los pixeles

**Devuelve**

la imagen creada a partir del vector

**3.1.3.4 ~Imagen()**

```
Imagen::~~Imagen ( )
```

destructor de la clase [Imagen](#).

**Postcondición**

destruye la imagen, libera la memoria, volverá a usarse con una llamada al constructor

**3.1.4 Documentación de las funciones miembro****3.1.4.1 asigna\_pixel()**

```
void Imagen::asigna_pixel (
    int fila,
    int columna,
    byte valor )
```

Asigna el valor *valor* al pixel indicado por *fila* y *columna* de la imagen.

**Parámetros**

<i>fila</i>	la fila del pixel a modificar
<i>columna</i>	la columna del pixel a modificar
<i>valor</i>	el valor que se asigna a la posicion ( <i>fila</i> , <i>columna</i> )

**Precondición**

*fila* y *columna* deben ser valores válidos [0, [num\\_filas\(\)](#)] para *fila* y [0,[num\\_columnas\(\)](#)] para *columnas* *valor* debe ser [0,255]

**Postcondición**

[Imagen](#)( *fila*, *columna* ) = *valor* . El resto de pixeles no se modifica

**3.1.4.2 num\_columnas()**

```
int Imagen::num_columnas ( ) const
```

Calcula el número de columnas de la imagen.

**Devuelve**

el número de columnas que tienen la imagen

**Postcondición**

no se modifica la imagen

**3.1.4.3 num\_filas()**

```
int Imagen::num_filas ( ) const
```

Calcula el número de filas de la imagen.

**Devuelve**

el número de filas de la imagen

**Postcondición**

la imagen no se modifica

**3.1.4.4 operator=()**

```
Imagen & Imagen::operator= (
    const Imagen & otra )
```

operador de asignación entre imagenes.

**Parámetros**

<i>otra</i>	la imagen que asignamos a la que llama al operador
-------------	--

**Devuelve**

una referencia al objeto que llama el operador

**Postcondición**

el objeto es una copia del pasado por referencia

**3.1.4.5 valor\_pixel()**

```
byte Imagen::valor_pixel (
    int fila,
    int columna ) const
```

Consulta el valor de un pixel de la imagen.

**Parámetros**

<i>fila</i>	la fila del pixel a consultar
<i>columna</i>	la columna del pixel a consultar

**Precondición**

*fila* y *columna* deben ser valores válidos. [0, [num\\_filas\(\)](#)] para *fila* y [0, [num\\_columnas\(\)](#)] para *columnas*.

**Postcondición**

no se modifica la imagen

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- [inc/imagenES.h](#)
- [src/imagenES.cpp](#)



## Chapter 4

# Documentación de archivos

### 4.1 Referencia del Archivo inc/funciones\_imagen.h

Archivo con la suite de funciones sobre el *TDA\_Imagen*.

```
#include <iostream>
#include <imagenES.h>
#include <cmath>
```

Dependencia gráfica adjunta para funciones\_imagen.h:

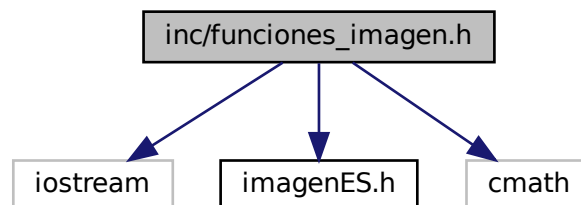
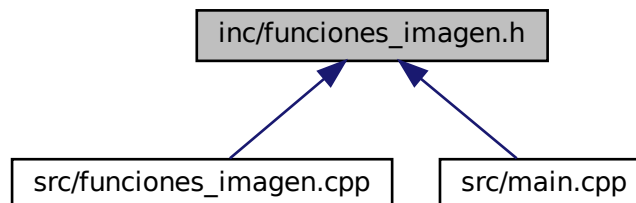


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



## Funciones

- void `error` (std::string mensaje)  
*funcion para mostrar los mensajes de error*
- `Imagen leerVectorPGM` (byte \*vector, int filas, int columnas)  
*funcion para obtener una imagen a partir de un vector 1d de una imagen PGM*
- `Imagen leerVectorPPM` (byte \*vector, int filas, int columnas)  
*funcion para obtener una imagen a partir de un vector 1d de una imagen PPM*
- void `escribirVectorPGM` (const `Imagen` &img, byte \*vector, int filas, int columnas)  
*funcion para transformar un objeto imagen en un vector unidimensional*
- void `colorAGris` (const char \*nombre\_ppm, const char \*nombre\_pgm)  
*función que permite convertir una imagen PPM a una imagen PGM*
- bool `escribirImagen` (const `Imagen` &img, const char \*nombre\_archivo)  
*función para escribir en el disco un objeto de la clase imagen*
- void `umbralizar_escala_grises` (const char \*original, const char \*salida, int umbral\_min, int umbral\_max)  
*funcion para generar una imagen nueva a partir de una original, basado en un umbral de escala de grises. Si el pixel se encuentra en el intervalo, conserva su color, en caso contrario se vuelve negro*
- void `zoom` (const char \*entrada, const char \*salida, int x1, int y1, int x2, int y2)  
*funcion para realizar zoom en una porción cuadrada de la imagen*
- void `contrastar` (const char \*original, const char \*salida, int minimo, int maximo)  
*funcion para contrastar una imagen basado en unos valores min y max*
- void `morphing` (const char \*fuente, const char \*destino, const char \*basename, int pasos)  
*funcion para realizar una transicion entre dos imagenes (morphing)*
- double `transformacion_morph` (byte s, byte d, double a\_i)  
*funcion que realiza la transformacion de un pixel*

### 4.1.1 Descripción detallada

Archivo con la suite de funciones sobre el *TDA\_Imagen*.

Autor

Salvador Romero Cortés

### 4.1.2 Documentación de las funciones

#### 4.1.2.1 colorAGris()

```
void colorAGris (
    const char * nombre_ppm,
    const char * nombre_pgm )
```

función que permite convertir una imagen PPM a una imagen PGM

Parámetros

<code>nombre_ppm</code>	la imagen a color
<code>nombre_pgm</code>	la imagen en escala de grises

**Precondición**

debe existir el archivo con el nombre *nombre\_ppm*

**Postcondición**

se escribe directamente el archivo desde esta funcion

**4.1.2.2 contrastar()**

```
void contrastar (
    const char * original,
    const char * salida,
    int minimo,
    int maximo )
```

funcion para contrastar una imagen basado en unos valores *min* y *max*

**Precondición**

*min* y *max* deben ser  $>0$  y  $<255$

*original* debe ser una imagen valida

**Parámetros**

<i>original</i>	nombre de la imagen original a partir de la cual se genera la imagen contrastada
<i>salida</i>	nombre de la imagen contrastada
<i>minimo</i>	inicio del intervalo
<i>maximo</i>	final del intervalo

**Postcondición**

se escribe en disco la imagen *salida*

**4.1.2.3 error()**

```
void error (
    std::string mensaje )
```

funcion para mostrar los mensajes de error

**FUNCIONES AUXILIARES****Parámetros**

<i>mensaje</i>	el mensaje de error
----------------	---------------------

#### 4.1.2.4 escribirImagen()

```
bool escribirImagen (
    const Imagen & img,
    const char * nombre_archivo )
```

función para escribir en el disco un objeto de la clase imagen

##### Parámetros

<i>img</i>	imagen a escribir en disco
<i>nombre_archivo</i>	el nombre del archivo que se guarda

##### Precondición

*img* deber se una imagen inicializada y valida

*nombre\_archivo* debe ser un nombre valido

##### Devuelve

true si no ha habido fallos, false en caso contrario

#### 4.1.2.5 escribirVectorPGM()

```
void escribirVectorPGM (
    const Imagen & img,
    byte * vector,
    int filas,
    int columnas )
```

funcion para transformar un objeto imagen en un vector unidimensional

##### Parámetros

<i>img</i>	la imagen que se va a convertir
<i>vector</i>	el vector donde se escribe la imagen
<i>filas</i>	el número de filas de la imagen
<i>columnas</i>	el número de columnas de la imagen

##### Precondición



**Postcondición**

todos los parámetros se modifican menos *img*

**4.1.2.6 leerVectorPGM()**

```
Imagen leerVectorPGM (
    byte * vector,
    int filas,
    int columnas )
```

funcion para obtener una imagen a partir de un vector 1d de una imagen PGM

**Parámetros**

<i>vector</i>	el vector unidimensional que contiene los pixeles de la imagen
<i>filas</i>	el número de filas que tendrá la imagen
<i>columnas</i>	el número de columnas que tendrá la imagen

**Devuelve**

objeto imagen con los datos del vector

**4.1.2.7 leerVectorPPM()**

```
Imagen leerVectorPPM (
    byte * vector,
    int filas,
    int columnas )
```

funcion para obtener una imagen a partir de un vector 1d de una imagen PPM

**Parámetros**

<i>vector</i>	el vector unidimensional que contiene los pixeles de la imagen
<i>filas</i>	el número de filas que tendrá la imagen
<i>columnas</i>	el número de columnas que tendrá la imagen

**Devuelve**

objeto imagen con los datos del vector (en escala de grises)

#### 4.1.2.8 morphing()

```
void morphing (
    const char * fuelle,
    const char * destino,
    const char * basename,
    int pasos )
```

funcion para realizar una transicion entre dos imagenes (morphing)

##### Parámetros

<i>fuelle</i>	imagen de la que se parte (nombre)
<i>destino</i>	imagen a la que se va transicionando (nombre)
<i>basename</i>	nombre basico de los archivos intermedios que se generan.
<i>pasos</i>	numero de pasos intermedios (e imagenes) que se van a hacer para la transicion

##### Precondición

las imagenes deben ser imagenes validas, asi como que *basename* debe ser un nombre valido para nombre de archivo del so. *paso* > 0 las imagenes deben tener tambien el mismo tamaño

#### 4.1.2.9 tranformacion\_morph()

```
double tranformacion_morph (
    byte s,
    byte d,
    double a_i )
```

funcion que realiza la transformacion de un pixel

##### Parámetros

<i>s</i>	pixel inicial
<i>d</i>	pixel final
$a \leftarrow$ $\_ \leftarrow$ <i>i</i>	cociente de transformacion

##### Devuelve

la transformacion

#### 4.1.2.10 umbralizar\_escala\_grises()

```
void umbralizar_escala_grises (
    const char * original,
```

```
const char * salida,  
int umbral_min,  
int umbral_max )
```

funcion para generar una imagen nueva a partir de una original, basado en un umbral de escala de grises. Si el pixel se encuentra en el intervalo, conserva su color, en caso contrario se vuelve negro

#### Parámetros

<i>original</i>	el nombre de la imagen de la que se parte
<i>salida</i>	el nombre de la imagen nueva
<i>umbral_min</i>	el minimo del intervalo umbral
<i>umbral_max</i>	el maximo del intervalo umbral

#### Precondición

*original* debe ser una imagen valida

*umbral\_min* < *umbral\_max* además de ser ambos entre 0 y 255

#### Postcondición

*salida* se guarda como imagen valida en disco

#### 4.1.2.11 zoom()

```
void zoom (  
    const char * entrada,  
    const char * salida,  
    int x1,  
    int y1,  
    int x2,  
    int y2 )
```

funcion para realizar zoom en una porción cuadrada de la imagen

#### Parámetros

<i>entrada</i>	nombre del archivo de la imagen de entrada
<i>salida</i>	nombre del archivo de la imagen de salida
<i>x1</i>	posicion x de la esquina superior izquierda
<i>y1</i>	posicion y de la esquina superior izquierda
<i>x2</i>	posicion x de la esquina inferior derecha
<i>y2</i>	posicion y de la esquina inferior derecha

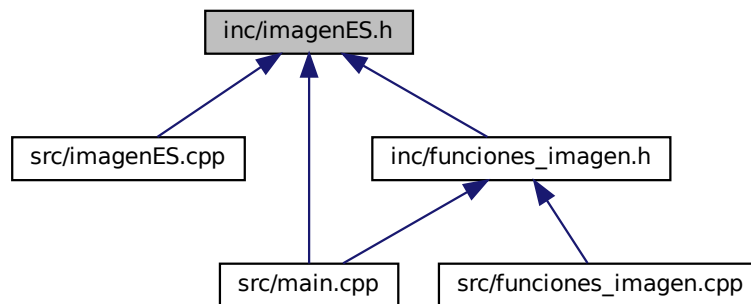
#### Precondición

$x1 < x2$  y  $y1 < y2$ . Además debe ser cuadrado,  $|x1-x2| == |y1-y2|$

## 4.2 Referencia del Archivo inc/imagenES.h

Fichero cabecera para la E/S de imágenes.

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



### Clases

- class `Imagen`  
*clase que representa una imagen digital en escala de grises*

### defines

- `#define BLANCO 255`  
*Macro para definir el color blanco.*
- `#define NEGRO 0`  
*Macro para definir el color negro.*

### typedefs

- `typedef unsigned char byte`

### Enumeraciones

- enum `TipolImagen` { `IMG_DESCONOCIDO`, `IMG_PGM`, `IMG_PPM` }  
*Tipo de imagen.*

## Funciones

- [TipImagen LeerTipImagen](#) (const char \*nombre)  
*Devuelve el tipo de imagen del archivo.*
- unsigned char \* [LeerImagenPPM](#) (const char \*nombre, int &filas, int &columnas)  
*Lee una imagen de tipo PPM.*
- bool [EscribirImagenPPM](#) (const char \*nombre, const unsigned char \*datos, const int filas, const int columnas)  
*Escribe una imagen de tipo PPM.*
- unsigned char \* [LeerImagenPGM](#) (const char \*nombre, int &filas, int &columnas)  
*Lee una imagen de tipo PGM.*
- bool [EscribirImagenPGM](#) (const char \*nombre, const unsigned char \*datos, const int filas, const int columnas)  
*Escribe una imagen de tipo PGM.*

### 4.2.1 Descripción detallada

Fichero cabecera para la E/S de imágenes.

#### Autor

Salvador Romero Cortés Permite la E/S de archivos de tipo PGM,PPM

### 4.2.2 Documentación de los 'typedefs'

#### 4.2.2.1 byte

```
typedef unsigned char byte
```

COMIENZA TDA IMAGEN

### 4.2.3 Documentación de las enumeraciones

#### 4.2.3.1 TipImagen

```
enum TipoImagen
```

Tipo de imagen.

Declara una serie de constantes para representar los distintos tipos de imágenes que se pueden manejar.

#### Ver también

[LeerTipImagen](#)

## 4.2.4 Documentación de las funciones

### 4.2.4.1 EscribirImagenPGM()

```
bool EscribirImagenPGM (
    const char * nombre,
    const unsigned char * datos,
    const int filas,
    const int columnas )
```

Escribe una imagen de tipo PGM.

#### Parámetros

<i>nombre</i>	archivo a escribir
<i>datos</i>	punteros a los $f \times c$ bytes que corresponden a los valores de los píxeles de la imagen de grises.
<i>filas</i>	filas de la imagen
<i>columnas</i>	columnas de la imagen

#### Devuelve

si ha tenido éxito en la escritura.

### 4.2.4.2 EscribirImagenPPM()

```
bool EscribirImagenPPM (
    const char * nombre,
    const unsigned char * datos,
    const int filas,
    const int columnas )
```

Escribe una imagen de tipo PPM.

#### Parámetros

<i>nombre</i>	archivo a escribir
<i>datos</i>	punteros a los $f \times c \times 3$ bytes que corresponden a los valores de los píxeles de la imagen en formato RGB.
<i>filas</i>	filas de la imagen
<i>columnas</i>	columnas de la imagen

#### Devuelve

si ha tenido éxito en la escritura.

#### 4.2.4.3 LeerImagenPGM()

```
unsigned char* LeerImagenPGM (
    const char * nombre,
    int & filas,
    int & columnas )
```

Lee una imagen de tipo PGM.

##### Parámetros

<i>nombre</i>	archivo a leer
<i>filas</i>	Parámetro de salida con las filas de la imagen.
<i>columnas</i>	Parámetro de salida con las columnas de la imagen.

##### Devuelve

puntero a una nueva zona de memoria que contiene *filas* x *columnas* bytes que corresponden a los grises de todos los píxeles (desde la esquina superior izqda a la inferior drcha). En caso de que no se pueda leer, se devuelve cero. (0).

##### Postcondición

En caso de éxito, el puntero apunta a una zona de memoria reservada en memoria dinámica. Será el usuario el responsable de liberarla.

#### 4.2.4.4 LeerImagenPPM()

```
unsigned char* LeerImagenPPM (
    const char * nombre,
    int & filas,
    int & columnas )
```

Lee una imagen de tipo PPM.

##### Parámetros

<i>nombre</i>	archivo a leer
<i>filas</i>	Parámetro de salida con las filas de la imagen.
<i>columnas</i>	Parámetro de salida con las columnas de la imagen.

##### Devuelve

puntero a una nueva zona de memoria que contiene *filas* x *columnas* x 3 bytes que corresponden a los colores de todos los píxeles en formato RGB (desde la esquina superior izqda a la inferior drcha). En caso de que no se pueda leer, se devuelve cero. (0).

### Postcondición

En caso de éxito, el puntero apunta a una zona de memoria reservada en memoria dinámica. Será el usuario el responsable de liberarla.

#### 4.2.4.5 LeerTipoImagen()

```
TipoImagen LeerTipoImagen (
    const char * nombre )
```

Devuelve el tipo de imagen del archivo.

#### Parámetros

<i>nombre</i>	indica el archivo de disco que consultar
---------------	--

#### Devuelve

Devuelve el tipo de la imagen en el archivo

#### Ver también

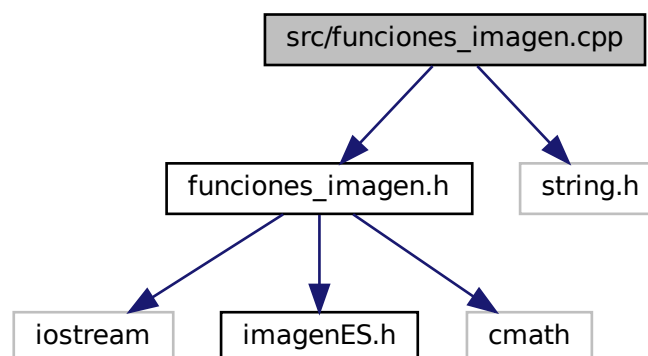
[TipoImagen](#)

## 4.3 Referencia del Archivo src/funciones\_imagen.cpp

Archivo con las definiciones de la suite de funciones sobre el *TDA\_Imagen*.

```
#include <funciones_imagen.h>
#include <string.h>
```

Dependencia gráfica adjunta para funciones\_imagen.cpp:





## Funciones

- void `error` (std::string mensaje)  
*funcion para mostrar los mensajes de error*
- `Imagen leerVectorPPM` (byte \*vector, int filas, int columnas)  
*funcion para obtener una imagen a partir de un vector 1d de una imagen PPM*
- void `escribirVectorPGM` (const `Imagen` &img, byte \*vector, int filas, int columnas)  
*funcion para transformar un objeto imagen en un vector unidimensional*
- void `colorAGris` (const char \*nombre\_ppm, const char \*nombre\_pgm)  
*función que permite convertir una imagen PPM a una imagen PGM*
- bool `escribirImagen` (const `Imagen` &img, const char \*nombre\_archivo)  
*función para escribir en el disco un objeto de la clase imagen*
- void `umbralizar_escalas_grises` (const char \*original, const char \*salida, int umbral\_min, int umbral\_max)  
*funcion para generar una imagen nueva a partir de una original, basado en un umbral de escala de grises. Si el pixel se encuentra en el intervalo, conserva su color, en caso contrario se vuelve negro*
- void `zoom` (const char \*entrada, const char \*salida, int x1, int y1, int x2, int y2)  
*funcion para realizar zoom en una porción cuadrada de la imagen*
- void `contrastar` (const char \*original, const char \*salida, int minimo, int maximo)  
*funcion para contrastar una imagen basado en unos valores min y max*
- void `morphing` (const char \*fuente, const char \*destino, const char \*basename, int pasos)  
*funcion para realizar una transicion entre dos imagenes (morphing)*
- double `transformacion_morph` (byte s, byte d, double a\_i)  
*funcion que realiza la transformacion de un pixel*

## Variables

- const double `ROJO_GRIS` = 0.2989  
*Macro para definir la constante para transformar de rojo a escala de grises.*
- const double `VERDE_GRIS` = 0.587  
*Macro para definir la constante para transformar de verde a escala de grises.*
- const double `AZUL_GRIS` = 0.114  
*Macro para definir la constante para transformar de azul a escala de grises.*

### 4.3.1 Descripción detallada

Archivo con las definiciones de la suite de funciones sobre el *TDA\_Imagen*.

Autor

Salvador Romero Cortés

### 4.3.2 Documentación de las funciones

#### 4.3.2.1 colorAGris()

```
void colorAGris (
    const char * nombre_ppm,
    const char * nombre_pgm )
```

función que permite convertir una imagen PPM a una imagen PGM

**Parámetros**

<i>nombre_ppm</i>	la imagen a color
<i>nombre_pgm</i>	la imagen en escala de grises

**Precondición**

debe existir el archivo con el nombre *nombre\_ppm*

**Postcondición**

se escribe directamente el archivo desde esta funcion

**4.3.2.2 contrastar()**

```
void contrastar (
    const char * original,
    const char * salida,
    int minimo,
    int maximo )
```

funcion para contrastar una imagen basado en unos valores *min* y *max*

**Precondición**

*min* y *max* deben ser  $>0$  y  $<255$   
*original* debe ser una imagen valida

**Parámetros**

<i>original</i>	nombre de la imagen original a partir de la cual se genera la imagen contrastada
<i>salida</i>	nombre de la imagen contrastada
<i>minimo</i>	inicio del intervalo
<i>maximo</i>	final del intervalo

**Postcondición**

se escribe en disco la imagen *salida*

**4.3.2.3 error()**

```
void error (
    std::string mensaje )
```

funcion para mostrar los mensajes de error

**FUNCIONES AUXILIARES**

**Parámetros**

<i>mensaje</i>	el mensaje de error
----------------	---------------------

**4.3.2.4 escribirImagen()**

```
bool escribirImagen (
    const Imagen & img,
    const char * nombre_archivo )
```

función para escribir en el disco un objeto de la clase imagen

**Parámetros**

<i>img</i>	imagen a escribir en disco
<i>nombre_archivo</i>	el nombre del archivo que se guarda

**Precondición**

*img* deber se una imagen inicializada y valida

*nombre\_archivo* debe ser un nombre valido

**Devuelve**

true si no ha habido fallos, false en caso contrario

**4.3.2.5 escribirVectorPGM()**

```
void escribirVectorPGM (
    const Imagen & img,
    byte * vector,
    int filas,
    int columnas )
```

funcion para transformar un objeto imagen en un vector unidimensional

**Parámetros**

<i>img</i>	la imagen que se va a convertir
<i>vector</i>	el vector donde se escribe la imagen
<i>filas</i>	el número de filas de la imagen
<i>columnas</i>	el número de columnas de la imagen

**Precondición****Postcondición**

todos los parámetros se modifican menos *img*

**4.3.2.6 leerVectorPPM()**

```
Imagen leerVectorPPM (
    byte * vector,
    int filas,
    int columnas )
```

funcion para obtener una imagen a partir de un vector 1d de una imagen PPM

**Parámetros**

<i>vector</i>	el vector unidimensional que contiene los pixeles de la imagen
<i>filas</i>	el número de filas que tendrá la imagen
<i>columnas</i>	el número de columnas que tendrá la imagen

**Devuelve**

objeto imagen con los datos del vector (en escala de grises)

**4.3.2.7 morphing()**

```
void morphing (
    const char * fuente,
    const char * destino,
    const char * basename,
    int pasos )
```

funcion para realizar una transicion entre dos imagenes (morphing)

**Parámetros**

<i>fuentes</i>	imagen de la que se parte (nombre)
<i>destino</i>	imagen a la que se va transicionando (nombre)
<i>basename</i>	nombre basico de los archivos intermedios que se generan.
<i>pasos</i>	numero de pasos intermedios (e imagenes) que se van a hacer para la transicion

**Precondición**

las imagenes deben ser imagenes validas, asi como que basename debe ser un nombre valido para nombre de archivo del so. paso > 0 las imagenes deben tener tambien el mismo tamaño

**4.3.2.8 tranformacion\_morph()**

```
double tranformacion_morph (
    byte s,
    byte d,
    double a_i )
```

funcion que realiza la transformacion de un pixel

**Parámetros**

<i>s</i>	pixel inicial
<i>d</i>	pixel final
$a \leftrightarrow i$	cociente de transformacion

**Devuelve**

la transformacion

**4.3.2.9 umbralizar\_escala\_grises()**

```
void umbralizar_escala_grises (
    const char * original,
    const char * salida,
    int umbral_min,
    int umbral_max )
```

funcion para generar una imagen nueva a partir de una original, basado en un umbral de escala de grises. Si el pixel se encuentra en el intervalo, conserva su color, en caso contrario se vuelve negro

**Parámetros**

<i>original</i>	el nombre de la imagen de la que se parte
<i>salida</i>	el nombre de la imagen nueva
<i>umbral_min</i>	el minimo del intervalo umbral
<i>umbral_max</i>	el maximo del intervalo umbral

**Precondición**

*original* debe ser una imagen valida

*umbral\_min* < *umbral\_max* además de ser ambos entre 0 y 255

**Postcondición**

*salida* se guarda como imagen valida en disco

**4.3.2.10 zoom()**

```
void zoom (
    const char * entrada,
    const char * salida,
    int x1,
    int y1,
    int x2,
    int y2 )
```

funcion para realizar zoom en una porción cuadrada de la imagen

**Parámetros**

<i>entrada</i>	nombre del archivo de la imagen de entrada
<i>salida</i>	nombre del archivo de la imagen de salida
<i>x1</i>	posicion x de la esquina superior izquierda
<i>y1</i>	posicion y de la esquina superior izquierda
<i>x2</i>	posicion x de la esquina inferior derecha
<i>y2</i>	posicion y de la esquina inferior derecha

**Precondición**

$x1 < x2$  y  $y1 < y2$ . Además debe ser cuadrado,  $|x1-x2| == |y1-y2|$

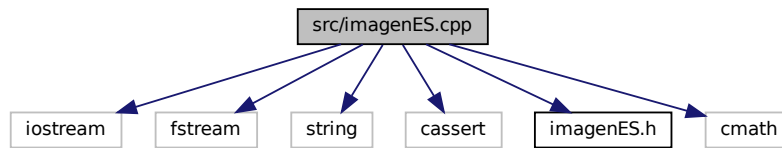
**4.4 Referencia del Archivo src/imagenES.cpp**

Fichero con definiciones para la E/S de imágenes Permite la E/S de archivos de tipo PGM,PPM.

```
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>
#include <imagenES.h>
```

```
#include <cmath>
```

Dependencia gráfica adjunta para imagenES.cpp:



## Funciones

- [TipImagen LeerTipo](#) (ifstream &f)
- [TipImagen LeerTipImagen](#) (const char \*nombre)  
*Devuelve el tipo de imagen del archivo.*
- char [SaltarSeparadores](#) (ifstream &f)
- bool [LeerCabecera](#) (ifstream &f, int &filas, int &cols)
- unsigned char \* [LeerImagenPPM](#) (const char \*nombre, int &filas, int &cols)  
*Lee una imagen de tipo PPM.*
- unsigned char \* [LeerImagenPGM](#) (const char \*nombre, int &filas, int &cols)  
*Lee una imagen de tipo PGM.*
- bool [EscribirImagenPPM](#) (const char \*nombre, const unsigned char \*datos, const int filas, const int cols)  
*Escribe una imagen de tipo PPM.*
- bool [EscribirImagenPGM](#) (const char \*nombre, const unsigned char \*datos, const int filas, const int cols)  
*Escribe una imagen de tipo PGM.*

### 4.4.1 Descripción detallada

Fichero con definiciones para la E/S de imágenes Permite la E/S de archivos de tipo PGM,PPM.

Autor

Salvador Romero Cortés

### 4.4.2 Documentación de las funciones

#### 4.4.2.1 EscribirImagenPGM()

```
bool EscribirImagenPGM (
    const char * nombre,
    const unsigned char * datos,
    const int filas,
    const int columnas )
```

Escribe una imagen de tipo PGM.

**Parámetros**

<i>nombre</i>	archivo a escribir
<i>datos</i>	punteros a los $f \times c$ bytes que corresponden a los valores de los píxeles de la imagen de grises.
<i>filas</i>	filas de la imagen
<i>columnas</i>	columnas de la imagen

**Devuelve**

si ha tenido éxito en la escritura.

**4.4.2.2 EscribirImagenPPM()**

```
bool EscribirImagenPPM (
    const char * nombre,
    const unsigned char * datos,
    const int filas,
    const int columnas )
```

Escribe una imagen de tipo PPM.

**Parámetros**

<i>nombre</i>	archivo a escribir
<i>datos</i>	punteros a los $f \times c \times 3$ bytes que corresponden a los valores de los píxeles de la imagen en formato RGB.
<i>filas</i>	filas de la imagen
<i>columnas</i>	columnas de la imagen

**Devuelve**

si ha tenido éxito en la escritura.

**4.4.2.3 LeerImagenPGM()**

```
unsigned char* LeerImagenPGM (
    const char * nombre,
    int & filas,
    int & columnas )
```

Lee una imagen de tipo PGM.

**Parámetros**

<i>nombre</i>	archivo a leer
<i>filas</i>	Parámetro de salida con las filas de la imagen.
<i>columnas</i>	Parámetro de salida con las columnas de la imagen.



**Devuelve**

puntero a una nueva zona de memoria que contiene *filas* x *columnas* bytes que corresponden a los grises de todos los píxeles (desde la esquina superior izqda a la inferior drcha). En caso de que no se pueda leer, se devuelve cero. (0).

**Postcondición**

En caso de éxito, el puntero apunta a una zona de memoria reservada en memoria dinámica. Será el usuario el responsable de liberarla.

**4.4.2.4 LeerImagenPPM()**

```
unsigned char* LeerImagenPPM (
    const char * nombre,
    int & filas,
    int & columnas )
```

Lee una imagen de tipo PPM.

**Parámetros**

<i>nombre</i>	archivo a leer
<i>filas</i>	Parámetro de salida con las filas de la imagen.
<i>columnas</i>	Parámetro de salida con las columnas de la imagen.

**Devuelve**

puntero a una nueva zona de memoria que contiene *filas* x *columnas* x 3 bytes que corresponden a los colores de todos los píxeles en formato RGB (desde la esquina superior izqda a la inferior drcha). En caso de que no se pueda leer, se devuelve cero. (0).

**Postcondición**

En caso de éxito, el puntero apunta a una zona de memoria reservada en memoria dinámica. Será el usuario el responsable de liberarla.

**4.4.2.5 LeerTipoImagen()**

```
TipoImagen LeerTipoImagen (
    const char * nombre )
```

Devuelve el tipo de imagen del archivo.

**Parámetros**

<i>nombre</i>	indica el archivo de disco que consultar
---------------	--

**Devuelve**

Devuelve el tipo de la imagen en el archivo

**Ver también**

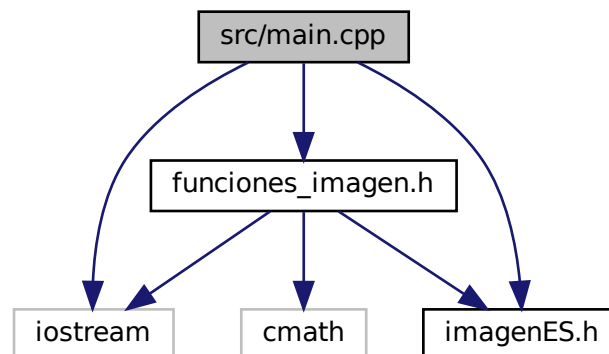
[TipoImagen](#)

## 4.5 Referencia del Archivo src/main.cpp

archivo que se encarga de transformar imagenes a color a escala de grises

```
#include <iostream>
#include <imagenES.h>
#include <funciones_imagen.h>
```

Dependencia gráfica adjunta para main.cpp:



### Funciones

- `int main (int argc, char *args[])`

*funcion principal del programa. Se encarga de la ejecución del programa*

#### 4.5.1 Descripción detallada

archivo que se encarga de transformar imagenes a color a escala de grises

archivo que ejecuta el programa principal

**Autor**

Salvador Romero Cortés

## 4.5.2 Documentación de las funciones

### 4.5.2.1 main()

```
int main (
    int argc,
    char * args[ ] )
```

funcion principal del programa. Se encarga de la ejecución del programa

#### Parámetros

<i>argc</i>	el número de parametros pasados en la ejecucion
<i>args</i>	el vector de los parametros pasados en la ejecucion, esto es imagen_entrada,imagen_salida,min,max

