```
/** @author Miguel Cantón Cortés
  * @file funciones.cpp
  * @brief Fichero con definiciones de funciones extra para la modificación de imagenes
  *
  * Estas funciones sirven de ejemplo de la clase Imagen
  *
  */
#include "Imagen.h"
#include "imagenES.h"
#include <cstdio>
#include <cmath>
#include <cassert>

using namespace std;

/**
  * @brief Función que convierte una imagen RGB (PPM) a escala de grises (PGM)
  *
  * Cada 3 bytes de la imagen PPM se corresponden a uno en la PGM, para pasar de uno a otro
  * usamos una sencilla función: f(r,g,b) = 0.2989*r + 0.587*g + 0.114*b
  *
  * @param fich_E dirección de la imagen origen
  * @param fich_S dirección de la imagen destino
  */
void RGB2Gris(const char* fich_E, const char* fich_S) {
        Imagen imagen_S;
        int nf, nc;
        unsigned char *imagen_E = 0;

        if(IMG_PPM == LeerTipoImagen(fich_E)) {
                imagen_E = LeerImagenPPM(fich_E, nf, nc);
                imagen_S.Reserva(nf, nc);
                for(int i = 0; i < nf; i++)
                        for(int j = 0; j < nc*3; j += 3)
                                imagen_S.asigna_pixel(i, j/3, 0.2989*imagen_E[i*nc*3+j]+0.587*imagen_E
[i*nc*3+j+1]+0.114*imagen_E[i*nc*3+j+2]);
                imagen_S.guardarPGM(fich_S);
        }
        delete [] imagen_E;
}

/**
  * @brief Función que cambia el contraste de la imagen en función del parámetro gamma
  *
  * Obtenemos el nuevo valor de cada píxel mediante la transformación f(x) = 255 * (x/255)^gamma
  *
  * @param fich_E dirección de la imagen origen
  * @param fich_S dirección de la imagen destino
  * @param gamma
  */
void mejorarContraste(const char* fich_E, const char* fich_S, float gamma) {
        Imagen imagen;

        imagen.cargarPGM(fich_E);
        for(int i = 0; i < imagen.num_filas(); i++)
                for(int j = 0; j < imagen.num_columnas(); j++)
                        imagen.asigna_pixel(i, j, 255.0*pow(imagen.valor_pixel(i,j)/255.0, gamma));
        imagen.guardarPGM(fich_S);
}

/**
  * @brief Función que realiza un ecualizado automático de la imagen
  *
  *  Calculamos el numero de apariciones de cada nivel de gris y lo almacenamo en el vector
  *  histograma. Calculamos las probabilidades de cada nivel de gris, dividiendo su número de
  *  apariciones entre el total de píxeles, y lo almacenamos en el vector probabilidades.
  *  Finalmente usando el vector de probabilidades, y aplicando una fórmula, obtenemos un
  *  nuevo vector con los nuevos valores de grises, transformacion
  *
  * @param fich_E dirección de la imagen origen
  * @param fich_S dirección de la imagen destino
  */
```

```
void ecualizar(const char* fich_E, const char* fich_S) {
	Imagen imagen, im;
	int histograma[256];
	double probabilidades[256];
	double transformacion[256];
	double suma;

	for(int i = 0; i < 256; i++)
		histograma[i] = 0;
	imagen.cargarPGM(fich_E);

	for(int i = 0; i < imagen.num_filas(); i++)
		for(int j = 0; j < imagen.num_columnas(); j++)
			histograma[imagen.valor_pixel(i,j)]++;

	for(int i = 0; i < 256; i++)
		probabilidades[i] = 1.0*histograma[i]/(imagen.num_filas()*imagen.num_columnas());

	for(int i = 0; i < 256; i++) {
		suma = 0;
		for(int j = 0; j <= i; j++)
			suma += probabilidades[j];
		transformacion[i] = 255.0*suma;
	}
	for(int i = 0; i < imagen.num_filas(); i++)
		for(int j = 0; j < imagen.num_columnas(); j++)
			imagen.asigna_pixel(i, j, transformacion[imagen.valor_pixel(i,j)]);
	imagen.guardarPGM(fich_S);
}

/**
  * @brief Función que calcula el umbral T de una imagen
  *
  *  Se trata de una función recursiva que calcula la media de los niveles de gris de la imagen
  *  a partir de las medias parciales de los píxeles por encima de un umbral dado y los que están por
debajo
  *  Repite este proceso de manera recursiva hasta que las medias se estabilizan
  *
  * @param imagen imagen original
  * @param T umbral
  */
int calcularUmbral(const Imagen &imagen, int T) {
	int P1 = 0, P2 = 0, V1 = 0, V2 = 0, media;

	for(int i = 0; i < imagen.num_filas(); i++) {
		for(int j = 0; j < imagen.num_columnas(); j++) {
			if(T <= imagen.valor_pixel(i,j)) {
				V1 += imagen.valor_pixel(i,j);
				P1++;
			} else {
				V2 += imagen.valor_pixel(i,j);
				P2++;
			}
		}
	}
	media = (V1/P1 + V2/P2)/2;
	if(media == T)
		return media;
	else
		return calcularUmbral(imagen, media);
}

/**
  * @brief Función que realiza un umbralizado automático de la imagen
  *
  * Una vez calculado el umbral T, recorre la imagen, convirtiendo en
  * 255 aquellos píxeles con un gris >= que T, y a 0 aquellos con un
  * gris < T
  *
  * @param fich_E dirección de la imagen origen
  * @param fich_S dirección de la imagen destino
  */
```

```
void umbralizar(const char* fich_E, const char* fich_S, int &T) {
	Imagen imagen;
	imagen.cargarPGM(fich_E);

	T = 0;
	for(int i = 0; i < imagen.num_filas(); i++)
		for(int j = 0; j < imagen.num_columnas(); j++)
			T += imagen.valor_pixel(i,j);
	T = T/(imagen.num_filas()*imagen.num_columnas());
	T = calcularUmbral(imagen, T); //Iniciamos la búsqueda recursiva con la media de valor umbral
inicial

	for(int i = 0; i < imagen.num_filas(); i++)
		for(int j = 0; j < imagen.num_columnas(); j++)
			if(imagen.valor_pixel(i,j) >= T)
				imagen.asigna_pixel(i, j, 255);
			else
				imagen.asigna_pixel(i, j, 0);
	imagen.guardarPGM(fich_S);
}

/**
  * @brief Función que reduce el tamaño de una imagen en un factor "factor"
  *
  * Agrupa la imagen en bloques de factor*factor píxeles y calcula su media.
  * Esta media se corresponde con cada píxel de la imagen reducida.
  *
  * @param fich_E dirección de la imagen origen
  * @param fich_S dirección de la imagen destino
  * @param factor factor de reducción de la imagen
  */
void crearIcono(const char* fich_E, const char* fich_S, int factor) {
	Imagen imagen_E, imagen_S;
	int n;
	double media;

	imagen_E.cargarPGM(fich_E);
	imagen_S.Reserva(ceil(imagen_E.num_filas()/factor), ceil(imagen_E.num_columnas()/factor));

	for(int i = 0; i < imagen_E.num_filas(); i += factor) {
		for(int j = 0; j < imagen_E.num_columnas(); j += factor) {
			media = 0;
			n = 0;
			for(int k = 0; k < factor && i+k < imagen_E.num_filas(); k++)
				for(int s = 0; s < factor && j+s < imagen_E.num_columnas(); s++) {
					media += imagen_E.valor_pixel(i+k,j+s);
					n++;
				}
			media = ceil(1.0*media/n);
			imagen_S.asigna_pixel(i/factor, j/factor, media);
		}
	}

	imagen_S.guardarPGM(fich_S);
}

/**
  * @brief Función que genera una lista de imagenes interpoladas entre dos de referencia
  *
  * Va estabilizando una imagen en relación a otra sumándole o restándole en cada ciclo
  * 1 a cada píxel hasta llegar a los mimos niveles de gris de la segunda imagen.
  * En cada ciclo guarda cada imagen intermedia
  *
  * @param fich_E dirección de la imagen A
  * @param fich_S dirección de la imagen B
  * @param prefijo dirección de las imagenes destino
  */
void morphing(const char* fich_E, const char* fich_S, const char* prefijo) {
	int num_iter = 0;
	bool continuar = true;
	Imagen imagen_A, imagen_B;
	char nombre[100];
```

```
        imagen_A.cargarPGM(fich_E);
        imagen_B.cargarPGM(fich_S);
        assert(imagen_A.num_columnas() == imagen_B.num_columnas() && imagen_A.num_filas() ==
imagen_B.num_filas());

        while(continuar) {
                continuar = false;
                for(int i = 0; i < imagen_A.num_filas(); i++) {
                        for(int j = 0; j < imagen_A.num_columnas(); j++) {
                                if(imagen_A.valor_pixel(i,j) < imagen_B.valor_pixel(i,j)) {
                                        imagen_A.asigna_pixel(i, j, 1+imagen_A.valor_pixel(i,j));
                                        continuar = true;
                                } else if(imagen_A.valor_pixel(i,j) > imagen_B.valor_pixel(i,j)) {
                                        imagen_A.asigna_pixel(i, j, imagen_A.valor_pixel(i,j)-1);
                                        continuar = true;
                                }
                        }
                }
                if(continuar) {
                        sprintf(nombre, "%s_%i.PGM", prefijo, num_iter);
                        imagen_A.guardarPGM(nombre);
                }
                num_iter++;
        }

}
```