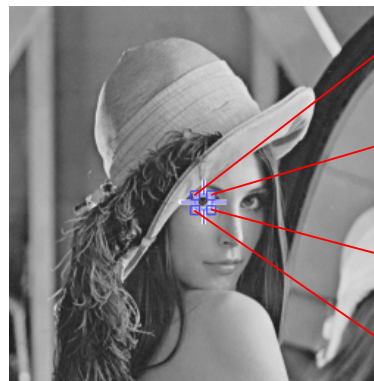




# Metodología de la Programación

DGIM

Curso 2021/2022



53	58	53	53	50	49	50	51	53	52	44
52	52	53	48	45	44	45	62	91	82	55
49	49	48	49	43	52	59	95	164	164	111
77	59	60	57	34	53	77	82	185	197	180
100	79	74	89	55	66	93	65	185	203	200
102	115	66	79	87	81	62	119	205	208	206
103	116	109	73	59	70	116	186	204	206	203
100	116	130	125	118	139	177	196	202	207	203
101	104	121	133	145	153	161	173	181	183	178
132	126	106	118	99	125	136	130	135	128	151

## Guion de prácticas

*Manejo del histograma y acceso a disco*

*Febrero de 2022*



# Índice

<b>1. Descripción</b>	<b>6</b>
<b>2. Objetivos</b>	<b>6</b>
<b>3. Formato PGM de imágenes digitales</b>	<b>6</b>
<b>4. La clase histograma</b>	<b>7</b>
4.1. Segmentación del histograma . . . . .	8
<b>5. Histogram</b>	<b>11</b>
<b>6. Image</b>	<b>12</b>
<b>7. Práctica a entregar</b>	<b>14</b>
7.1. Ejemplo de ejecución . . . . .	16
<b>8. TESTS DOCUMENTATION FOR PROJECT Imaging2</b>	<b>17</b>
8.1. _01_Basics . . . . .	17
8.1.1. UnitByte_Constructor . . . . .	17
8.1.2. UnitByte_getValue . . . . .	17
8.1.3. UnitByte_setValue . . . . .	17
8.1.4. UnitByte_onBit . . . . .	17
8.1.5. UnitByte_offBit . . . . .	17
8.1.6. UnitByte_getBit . . . . .	17
8.1.7. UnitByte_to_string . . . . .	17
8.1.8. UnitByte_shiftRByte . . . . .	18
8.1.9. UnitByte_shiftLByte . . . . .	18
8.1.10. Image_Constructor . . . . .	18
8.1.11. Image_Width . . . . .	18
8.1.12. Image_Height . . . . .	18
8.1.13. Image_setPixel . . . . .	18
8.1.14. Image_getPixel . . . . .	18
8.1.15. Image_getPos . . . . .	18
8.1.16. Histogram_Constructor . . . . .	18
8.1.17. Histogram_Size . . . . .	19
8.1.18. Histogram_Clear . . . . .	19
8.1.19. Histogram_getLevel . . . . .	19
8.1.20. Histogram_setLevel . . . . .	19
8.1.21. Histogram_getMaxLevel . . . . .	19
8.1.22. Histogram_getAverageLevel . . . . .	19
8.1.23. Histogram_getBalancedLevel . . . . .	19
8.2. _02_Intermediate . . . . .	19
8.2.1. UnitByte_onByte . . . . .	19
8.2.2. UnitByte_offByte . . . . .	19
8.2.3. Image_flatten . . . . .	19
8.2.4. Image_getHistogram . . . . .	20
8.2.5. Image_depictsHistogram . . . . .	20
8.2.6. INTEGRATION_ImageP2 . . . . .	20



8.3.    _03_Advanced . . . . .	20
8.3.1.    UnitByte_encodeByte . . . . .	20
8.3.2.    UnitByte_decodeByte . . . . .	20
8.3.3.    UnitByte_decomposeByte . . . . .	20
8.3.4.    Image_readFromFile . . . . .	21
8.3.5.    Image_saveToFile . . . . .	21
8.3.6.    Image_extractObjects . . . . .	21
8.4.    Tests run . . . . .	21



## 1. Descripción

Como ya se indicó con anterioridad, las prácticas tienen como objeto principal las imágenes en blanco y negro. Una imagen todos la percibimos como una matriz de (*alto x ancho*) de píxeles. Por el hecho de ser en blanco y negro, cada píxel se puede representar mediante un byte, de ahí que centráramos nuestra atención sobre este tipo en la primera práctica.

Las funciones de bajo nivel que implementamos en Imaging0 para manipular los bytes, se van a encapsular dentro una clase Byte, junto con su espacio de datos, de modo que podamos usarla como componente base de una Imagen.

Una vez definido Byte, vamos a abordar la estructura interna de una imagen y su implementación dentro de una clase. Vamos a dotar a la clase con los primeros métodos para poderla manipular a bajo nivel. Para poner todo esto en práctica vamos a crear nuestras primeras imágenes inicialmente sin preocuparnos de formato y procederemos a su visualización con la ayuda de utilidades.

## 2. Objetivos

El desarrollo de esta práctica pretende servir a los siguientes objetivos:

- Leer y grabar imágenes en disco, en un formato compatible con cualquier visualizador de fotos digitales
- Evolucionar el tipo de dato Histograma, que hasta ahora era un simple vector y transformarlo en una clase, con funciones propias, incluyendo la visualización del histograma. Para ello habrá que reestructurar algunas de las funciones de la práctica anterior.
- Realizar operaciones basadas en el histograma, como por ejemplo la segmentación de objetos de una imagen basada en sus colores.

## 3. Formato PGM de imágenes digitales

(Extraído de Wikipedia  [http://](#) **(Abrir →)**)

Netpbm (antes Pbmplus) es un paquete de programas gráficos de código abierto y una biblioteca de programación. Se utiliza principalmente en el mundo Unix, donde se puede encontrar incluido en las principales distribuciones de sistemas operativos de código abierto, pero también funciona en Microsoft Windows, macOS y otros sistemas operativos[2].

### Formatos de archivo

El proyecto Netpbm utiliza y define varios formatos gráficos. El formato de mapa de píxeles portátil (PPM), el formato de mapa de grises portátil (PGM) y el formato de mapa de bits portátil (PBM) son formatos de archivo de imagen diseñados para ser intercambiados fácilmente entre plataformas. A veces también se les denomina colectivamente formato

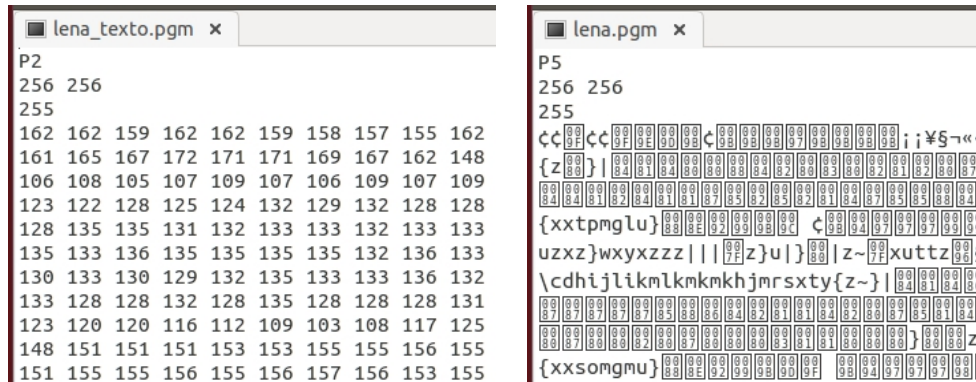


Figura 1: Contenido de la imagen 'lena.pgm' en formato de texto (izquierda) y binario (derecha).

de mapa de bits portátil (PNM) que no debe confundirse con el formato de mapa arbitrario portátil (PAM). El "número mágico" (Px) al principio de un archivo determina el tipo, no la extensión del archivo, aunque es una buena práctica utilizar la extensión correcta si es posible.


El formato PBM fue inventado por Jef Poskanzer en la década de 1980 como un formato que permitía transmitir mapas de bits monocromáticos dentro de un mensaje de correo electrónico como texto ASCII plano, lo que le permitía sobrevivir a cualquier cambio en el formato del texto. Poskanzer desarrolló la primera biblioteca de herramientas para manejar el formato PBM, Pbmplus, publicada en 1988. Contenía principalmente herramientas para convertir entre PBM y otros formatos gráficos. A finales de 1988, Poskanzer había desarrollado los formatos PGM y PPM junto con sus herramientas asociadas y los añadió a Pbmplus. La versión final de Pbmplus fue el 10 de diciembre de 1991.

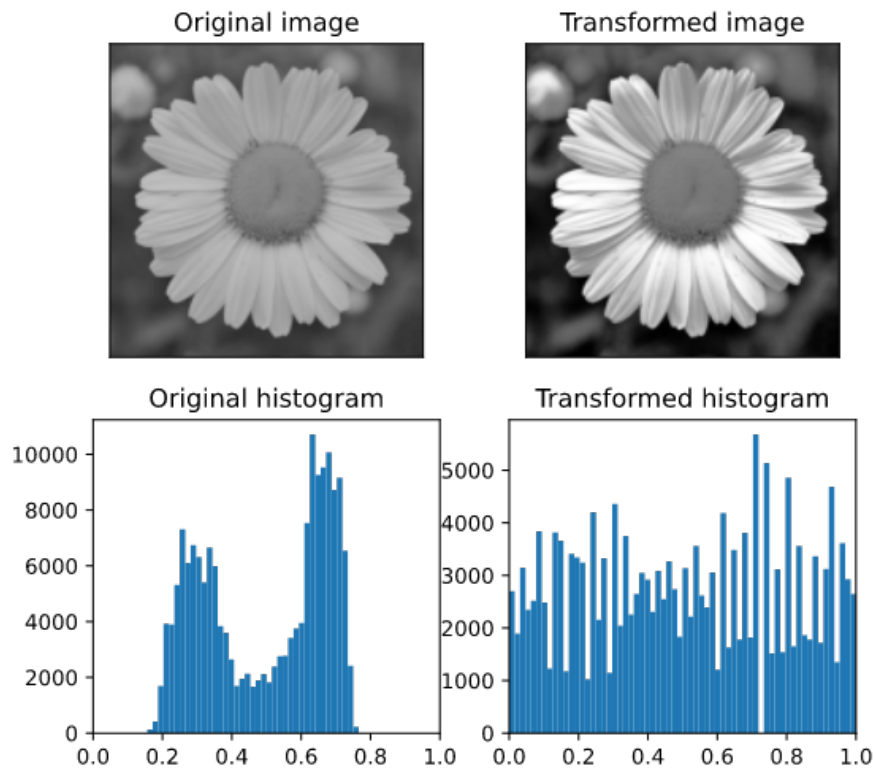
#### El formato PGM ASCII

- La primera línea del fichero es diferente (ver Figura 1), siendo P2 para las de texto y P5 para las binarias. A continuación vienen el número de columnas (256) y de filas (256), y el valor máximo de todos de grises que contiene (siempre 255, independientemente del valor máximo real).
- El contenido de la imagen de texto es perfectamente legible con un editor de texto mientras que la imagen binaria no lo es.

## 4. La clase histograma

La información del histograma que se ha visto en la práctica anterior es muy útil para realizar operaciones sobre la propia imagen, como realzar

los tonos ecualizando el histograma  ([Abrir →](#)) o la segmentación de la imagen a partir del histograma, que es la que se va a abordar en esta práctica.



#### 4.1. Segmentación del histograma

Para ello se supone que la imagen contiene objetos bien delimitados y cada uno de un color/tono diferente. De hecho si se observan estas imágenes, se puede ver en los histogramas como aparecen grupos de píxeles agrupados por tonos parecidos.



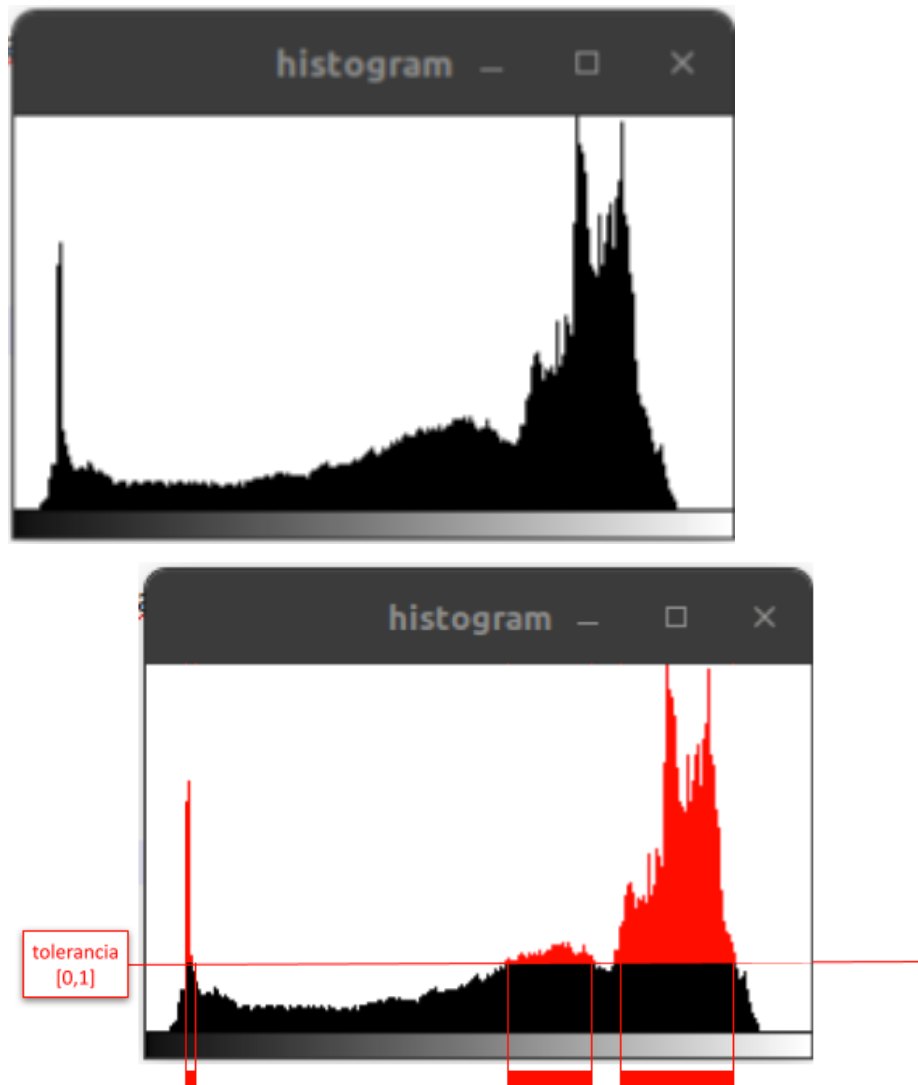


La segmentación permite identificar cada una de estas agrupaciones de píxeles de tonos similares como un objeto de la imagen. Para ello, el procedimiento será el siguiente.

1. Ir recorriendo todo el histograma y buscar qué conjunto de tonos sucesivos tiene una frecuencia igual o superior a

$$tolerancia * \max_{i \in [0, 255]} histograma(i)$$

$$tolerancia \in [0, 1]$$



2. Por cada grupo de tonos similares, crear una nueva imagen, de las mismas dimensiones, totalmente negra excepto aquellos píxeles que contienen el rango de tonos seleccionado. Las imágenes anteriores se han segmentado con una tolerancia del histograma de 0,1.



## 5. Histogram

```
1  /**
2   * @file Histogram.h
3   * @author MP
4   */
5  #include <istream>
6  #include <fstream>
7  #include "Byte.h"
8  #include "Image.h"
9
10 #ifndef _HISTOGRAM.H_
11 #define _HISTOGRAM.H_
12
13
14
15
16 /**
17  * @brief A black and white histogram
18  */
19 class Histogram {
20 public:
21     static const int HISTOGRAM_LEVELS=256; ///< Max number of bytes allowed for
22     static const double HISTOGRAM_TOLERANCE; ///< Default tolerance
23
24
25     /**
26      * @brief It builds an empty
27      */
28     Histogram();
29     /**
30      * @brief It returns the number of levels in the histogram
31      * @return The number of levels
32      */
33     int size() const;
34     /**
35      * @brief Sets the whole histogram to 0
36      */
37     void clear();
38     /**
39      * @brief It returns the value associated to the level indicated
40      * @param level The level indicated
41      * @return The value associated to the level
42      */
43     int getLevel(Byte level) const;
44     /**
45      * @brief It sets the value associated to the level
46      * @param level The level
47      * @param npixeles The new value
48      */
49     void setLevel(Byte level, int npixeles);
50     /**
51      * @brief It returns the maximum value stored
52      * @return The max of the levels
53      */
54     int getMaxLevel() const;
55     /**
56      * @brief it returns the average value stored
57      * @return The average level
58      */
59     int getAverageLevel() const;
60     /**
61      * It returns a balance level, that is, the level that leaves half of the points
62      * underneath or equal to it.
63      * @return The point of balance of the histogram
64      */
65     int getBalancedLevel() const;
66     /**
67      * @brief It returns a unique hash code for every object so that they might be compared
68      * @return The hash code as an string
69      */
70     std::string inspect() const;
71
72 private:
73     int _data[HISTOGRAM_LEVELS]; ///< datos de la imagen
74
75 };
76 #endif
```



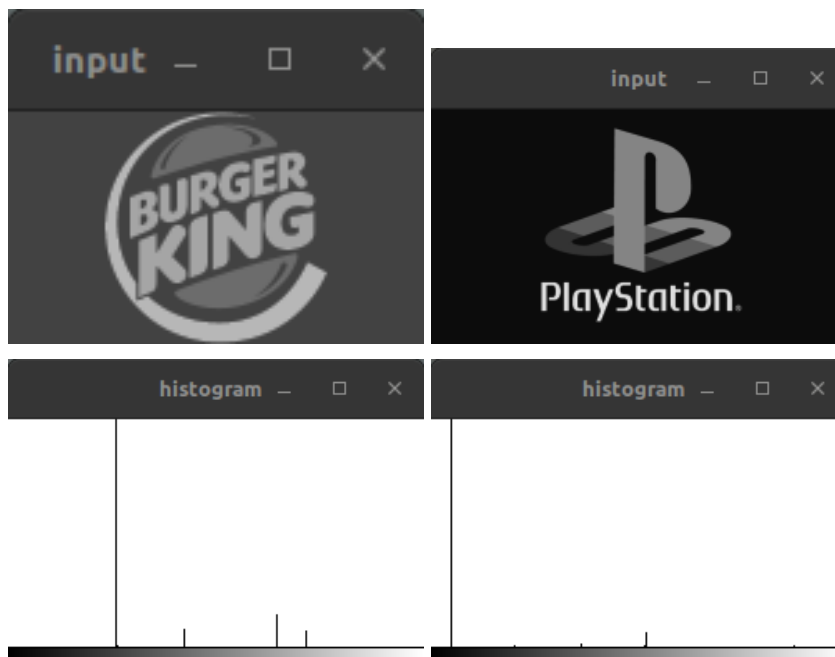
## 6. Image

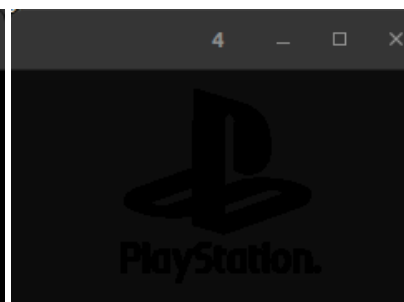
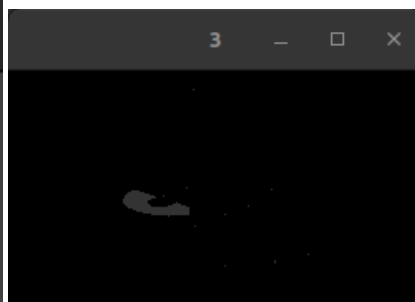
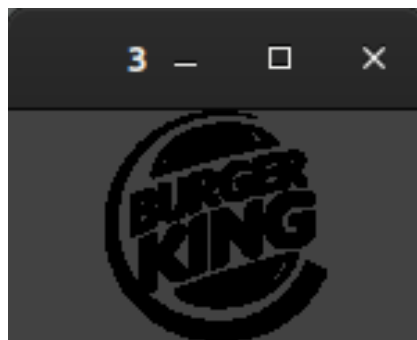
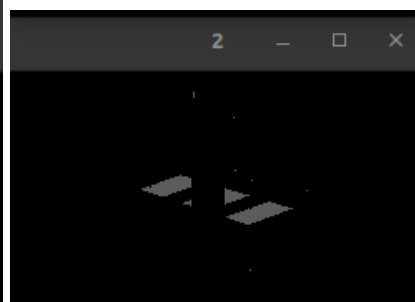
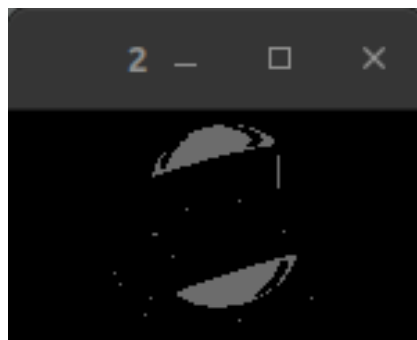
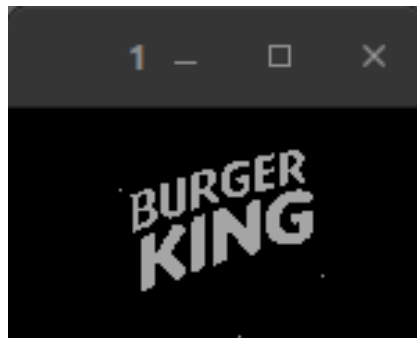
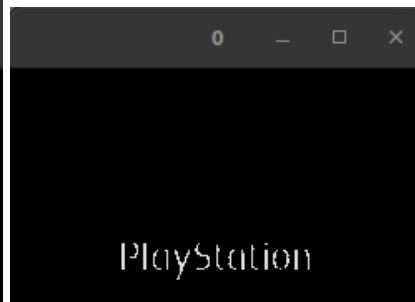
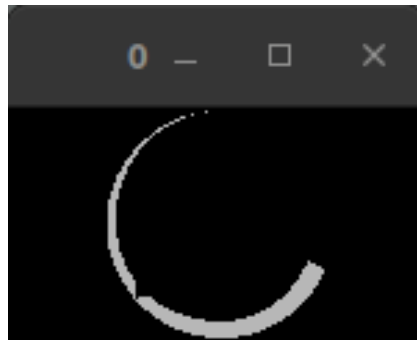
```
1  /**
2  @file Image.h
3  @brief Manejo de imágenes digitales en formato PGM blanco y negro
4  @author MP-DGIM - Grupo A
5  */
6
7  #ifndef _IMAGE_H_
8  #define _IMAGE_H_
9
10 #include <istream>
11 #include <fstream>
12 #include "Byte.h"
13 #include "Histogram.h"
14
15 /**
16 @brief A black and white image
17 */
18 class Image {
19 public:
20     static const int IMAGE_MAX_SIZE=200000; ///< Max number of bytes allowed for
21     static const int IMAGE_DISK_OK=0; ///< Image read/write successful
22     static const int IMAGE_ERROR_OPEN=1; ///< Error opening the file
23     static const int IMAGE_ERROR_DATA=2; ///< Missing data in the file
24     static const int IMAGE_ERROR_FORMAT=3; ///< Unknown image format
25     static const int IMAGE_TOO_LARGE=4; ///< The image is too large and does not fit into memory
26
27
28     /**
29      * @brief It builds an empty image
30      */
31     Image();
32     /**
33      * @brief It builds a fully black image with @a width columns and @a height rows
34      * @param height number of rows
35      * @param width number of columns
36      */
37     Image(int width, int height);
38     /**
39      * @brief It gives the number of rows of the image
40      * @return number of rows
41      */
42     int height() const;
43     /**
44      * @brief It gives the number of columns of the image
45      * @return The number of rows
46      */
47     int width() const;
48     /**
49      * @brief It assigns the value @a v to the position(x,y) of the image. It must check that
50      * the values x and y are valid, otherwise, it does not do anything.
51      * @param x The column
52      * @param y the row
53      * @param v The new value
54      */
55     void setPixel(int x, int y, Byte v);
56     /**
57      * @brief It returns the value of the requested (x,y) position. It must check that
58      * the values x and y are valid, otherwise, it returns a negative value. Please note that
59      * the value returned is a int
60      * @param x The column
61      * @param y the row
62      * @return The value of the pixel in [0-256] or -1 if there is an access error
63      */
64     int getPixel(int x, int y) const;
65     /**
66      * @brief It assigns the value @a v to the linear position i of the image. It must check that
67      * the values i is valid, otherwise, it does not do anything.
68      * @param i The linear position
69      * @param v The new value
70      */
71     void setPos(int i, Byte v);
72     /**
73      * @brief It returns the value of the requested linear position. It must check that
74      * the value i is valid, otherwise, it returns a negative value. Please note that
75      * the value returned is a int
76      * @param i The linear position
77      * @return The value of the pixel in [0-256] or -1 if there is an access error
78      */
79     int getPos(int i) const;
80     /**
81      * @brief It sets all pixels of the image to the value given
82      * @param b The value
83      */
84     void flatten(Byte b);
85
86     /**
87      * @brief It shows an image in an external window, ready for inspection. It uses
88      * the program display (ImageMagick) to display every image. For an easier identification
89      * process of all images shown are labeled with a title
90      * @param title The title on top of the window
91      */
92     void showInWindow(std::string title) const;
93     /**
94      * @brief It calculates the hash value of the image and returns it inside a string,
95      * together with its dimension.
96      * @param binary Its default value is true and then it shows the hash code of the image
```

[illegible]

## 7. Práctica a entregar

- Se deben implementar las funciones incluidas en el fichero `Image.h` y en `Histogram.h`
- Lectura de datos en disco. Las funciones deben abrir un fichero PGM ASCII con el formato comentado anteriormente y deben cargar los datos leídos en una imagen en memoria. La función de lectura debe comprobar que todo ha funcionado correctamente e informar de ello (ver `Imagen.h`). En caso contrario, deberá indicar un código de error en las siguientes circunstancias
  1. El fichero no ha podido abrirse correctamente.
  2. El fichero no sigue el formato de encabezado de un PGM ASCII.
  3. La imagen des disco es demasiado grande y no cabría en la memoria asignada (ver constantes definidas en `Imagen.h`)
  4. El fichero contiene menos datos de los esperados.
- Escritura de datos a disco. Las funciones deben de poder escribir los datos contenidos en una instancia de la clase `Imagen` en un fichero PGM ASCII con el formato comentado anteriormente. Igualmente, se deben comprobar los posibles errores en esta operación.
- Se debe implementar la segmentación de objetos por histograma tal y como se ha comentado anteriormente, comenzando a recorrer el histograma desde los puntos más brillantes (255) hasta los más oscuros, y devolverlos en un array de imágenes, de forma que la posición `[0]` sea siempre la agrupación de tonos más brillantes.







## 7.1. Ejemplo de ejecución

Este ejemplo se realiza sobre la imagen `burgerking.pgm` que se encuentra en la carpeta `data` con *tolerancia* = 0,1 y todas las imágenes resultantes se guardan en la carpeta `data`.

```
lcv@numenor:Imaging2: dist/Debug/GNU-Linux/imaging2

...Reading image from ./data/burgerking.pgm
150x84
./data/burgerking.pgm

[im_input] 150x84 459731281

[im_histogram] 256x160 2124314153
Found object 0 in [182,183]
Found object 1 in [164,165]
Found object 2 in [107,108]
Found object 3 in [65,67]
Found 4objects

[im_collection[i]] 150x84 1525489000

...Saving image into ./data/0.pgm

[im_collection[i]] 150x84 345575867

...Saving image into ./data/1.pgm

[im_collection[i]] 150x84 3644445128

...Saving image into ./data/2.pgm

[im_collection[i]] 150x84 2975546243

...Saving image into ./data/3.pgm
```





## 8. TESTS DOCUMENTATION FOR PROJECT Imaging2

### 8.1. \_01\_Basics

#### 8.1.1. UnitByte\_Constructor

1. Declaring a Byte gives 0 by default
2. Declaring a Byte(1) gives 1
3. Declaring a Byte(128) gives 128

#### 8.1.2. UnitByte\_getValue

1. Declaring a Byte gives 0 by default
2. Declaring a Byte(1) gives 1
3. Declaring a Byte(128) gives 128

#### 8.1.3. UnitByte\_setValue

1. Declaring a Byte and setting its value to 0 gives 0 by default
2. Declaring a Byte and setting its value to 1 gives 1
3. Declaring a Byte and setting its value to 128 gives 128

#### 8.1.4. UnitByte\_onBit

1. Given a byte 00000000, activating the 0-bit gives 1
2. Given a byte 00000000, activating the 1-bit gives 2
3. Given a byte 00000000, activating the 7-bit gives 128

#### 8.1.5. UnitByte\_offBit

1. Given a byte 11111111, deactivating the 0-bit gives 254
2. Given a byte 11111111, deactivating the 1-bit gives 253
3. Given a byte 11111111, deactivating the 7-bit gives 127

#### 8.1.6. UnitByte\_getBit

1. Given a byte 11111111, querying any bit always give true
2. Given a byte 00000000, querying any bit gives false

#### 8.1.7. UnitByte\_to\_string

1. A byte 11111111 prints as it is
2. A byte 00000000 prints as it is



#### 8.1.8. **UnitByte\_shiftRByte**

1. A byte 11111111 shifted to the right gives 127
2. A byte 11111111 shifted twice to the right gives 63
3. A byte 00000001 shifted to the right gives 0

#### 8.1.9. **UnitByte\_shiftLByte**

1. A byte 11111111 shifted to the left gives 254
2. A byte 11111111 shifted twice to the right gives 252
3. A byte 00000001 shifted to the right gives 2

#### 8.1.10. **Image\_Constructor**

1. and empty data
2. and empty data
3. and empty data

#### 8.1.11. **Image\_Width**

1. gives width
2. gives width
3. gives width

#### 8.1.12. **Image\_Height**

1. gives height
2. gives height
3. gives height

#### 8.1.13. **Image\_setPixel**

1. but should have been
2. but should have been

#### 8.1.14. **Image\_getPixel**

1. but should have been
2. but should have been

#### 8.1.15. **Image\_getPos**

1. but should have been
2. but should have been

#### 8.1.16. **Histogram\_Constructor**

1. A newly created instance of an histogram must be empty
2. A newly created instance of an histogram must be empty hash



#### **8.1.17. Histogram\_Size**

1. Any histogram must have a capacity for 256 values

#### **8.1.18. Histogram\_Clear**

1. Any modified histogram must not be empty
2. A crescent triangular histogram is wrong
3. Once filled up, and cleared, an histogram must be empty again

#### **8.1.19. Histogram\_getLevel**

1. A crescent triangular histogram has wrong values
2. A crescent triangular histogram has wrong values

#### **8.1.20. Histogram\_setLevel**

1. A crescent triangular histogram is wrong

#### **8.1.21. Histogram\_getMaxLevel**

1. A crescent triangular histogram has wrong values
2. A crescent triangular histogram has wrong values

#### **8.1.22. Histogram\_getAverageLevel**

1. A crescent triangular histogram has wrong values
2. A crescent triangular histogram has wrong values

#### **8.1.23. Histogram\_getBalancedLevel**

1. A crescent triangular histogram has wrong values
2. A crescent triangular histogram has wrong values

### **8.2. \_02\_Intermediate**

#### **8.2.1. UnitByte\_onByte**

1. Activating a Byte gives 255

#### **8.2.2. UnitByte\_offByte**

1. Deactivating a Byte gives 0

#### **8.2.3. Image\_flatten**

1. is wrong
2. is wrong



#### 8.2.4. **Image\_getHistogram**

1. The single pixel image must have one pixel per each 256 gray level
2. The single pixel image must have a maximum histogram of 1
3. The single pixel image must have a balanced level of 128
4. The checkers image must have only 4 levels
5. The checkers image must have a maximum histogram of 64
6. The checkers image must have a balanced level of 86

#### 8.2.5. **Image\_depictsHistogram**

1. The histogram of singlepix Image is wrong
2. The histogram of a flat-128 Image is wrong

#### 8.2.6. **INTEGRATION\_ImageP2**

1. The output of the main program must match that of the assignment
2. of the objects found in ./data/burgerking.pgm has not correctly been saved on disk
3. of the objects found in ./data/burgerking.pgm has not correctly been saved on disk
4. of the objects found in ./data/burgerking.pgm has not correctly been saved on disk
5. of the objects found in ./data/burgerking.pgm has not correctly been saved on disk

### 8.3. **\_03\_Advanced**

#### 8.3.1. **UnitByte\_encodeByte**

1. Activating bits 0,1 and 7 gives 131

#### 8.3.2. **UnitByte\_decodeByte**

1. A byte 131 gives true only in bits 0,1 and 7
2. A byte 131 gives true only in bits 0,1 and 7
3. A byte 131 gives true only in bits 0,1 and 7
4. A byte 131 gives true only in bits 0,1 and 7
5. A byte 131 gives true only in bits 0,1 and 7

#### 8.3.3. **UnitByte\_decomposeByte**

1. Decomposing byte 131 gives 3 active bits
2. Decomposing byte 131 gives 3 active bits
3. Decomposing byte 131 gives 3 active bits
4. Decomposing byte 131 gives 3 active bits



### 8.3.4. Image\_readFromFile

1. Method readFromFile must warn if a file could not be open
2. Method readFromFile must warn if a file has a data error
3. Method readFromFile must warn if a file does not follow the ASCII PGM format
4. Method readFromFile must warn if a file is too large
5. Method readFromFile must read valid files with ASCII PGM format
6. Method readFromFile does not read well valid files with ASCII PGM format

### 8.3.5. Image\_saveToFile

1. Method saveToFile must warn if a file could not be open
2. Method saveToFile must save to disk valid ASCII PGM images
3. Method saveToFile must save to disk valid ASCII PGM images

### 8.3.6. Image\_extractObjects

1. The checkers image should decompose into 4 objects
2. of the objects found in checkers image is wrong
3. of the objects found in checkers image is wrong
4. of the objects found in checkers image is wrong
5. of the objects found in checkers image is wrong
6. The flat image should decompose into 1 object

## 8.4. Tests run

```
[=====] Running 35 tests from 3 test suites.  
[-----] Global test environment set-up.  
[-----] 23 tests from _01_Basics  
[ RUN      ] _01_Basics.UnitByte_Constructor  
[ OK       ] _01_Basics.UnitByte_Constructor (1 ms)  
[ RUN      ] _01_Basics.UnitByte_getValue  
[ OK       ] _01_Basics.UnitByte_getValue (0 ms)  
[ RUN      ] _01_Basics.UnitByte_setValue  
[ OK       ] _01_Basics.UnitByte_setValue (1 ms)  
[ RUN      ] _01_Basics.UnitByte_onBit  
[ OK       ] _01_Basics.UnitByte_onBit (1 ms)  
[ RUN      ] _01_Basics.UnitByte_offBit  
[ OK       ] _01_Basics.UnitByte_offBit (1 ms)  
[ RUN      ] _01_Basics.UnitByte_getBit  
[ OK       ] _01_Basics.UnitByte_getBit (3 ms)  
[ RUN      ] _01_Basics.UnitByte_to_string  
[ OK       ] _01_Basics.UnitByte_to_string (0 ms)  
[ RUN      ] _01_Basics.UnitByte_shiftRByte  
[ OK       ] _01_Basics.UnitByte_shiftRByte (1 ms)  
[ RUN      ] _01_Basics.UnitByte_shiftLByte  
[ OK       ] _01_Basics.UnitByte_shiftLByte (1 ms)  
[ RUN      ] _01_Basics.Image_Constructor  
[ OK       ] _01_Basics.Image_Constructor (3 ms)  
[ RUN      ] _01_Basics.Image_Width  
[ OK       ] _01_Basics.Image_Width (3 ms)  
[ RUN      ] _01_Basics.Image_Height  
[ OK       ] _01_Basics.Image_Height (3 ms)  
[ RUN      ] _01_Basics.Image_setPixel  
[ OK       ] _01_Basics.Image_setPixel (1 ms)  
[ RUN      ] _01_Basics.Image_getPixel  
[ OK       ] _01_Basics.Image_getPixel (1 ms)  
[ RUN      ] _01_Basics.Image_getPos
```



```
[ OK ] _01_Basics.Image_getPos (1 ms)
[ RUN ] _01_Basics.Histogram_Constructor
[ OK ] _01_Basics.Histogram_Constructor (1 ms)
[ RUN ] _01_Basics.Histogram_Size
[ OK ] _01_Basics.Histogram_Size (0 ms)
[ RUN ] _01_Basics.Histogram_Clear
[ OK ] _01_Basics.Histogram_Clear (1 ms)
[ RUN ] _01_Basics.Histogram_getLevel
[ OK ] _01_Basics.Histogram_getLevel (1 ms)
[ RUN ] _01_Basics.Histogram_setLevel
[ OK ] _01_Basics.Histogram_setLevel (0 ms)
[ RUN ] _01_Basics.Histogram_getMaxLevel
[ OK ] _01_Basics.Histogram_getMaxLevel (1 ms)
[ RUN ] _01_Basics.Histogram_getAverageLevel
[ OK ] _01_Basics.Histogram_getAverageLevel (1 ms)
[ RUN ] _01_Basics.Histogram_getBalancedLevel
[ OK ] _01_Basics.Histogram_getBalancedLevel (0 ms)
[-----] 23 tests from _01_Basics (27 ms total)

[-----] 6 tests from _02_Intermediate
[ RUN ] _02_Intermediate.UnitByte_onByte
[ OK ] _02_Intermediate.UnitByte_onByte (1 ms)
[ RUN ] _02_Intermediate.UnitByte_offByte
[ OK ] _02_Intermediate.UnitByte_offByte (0 ms)
[ RUN ] _02_Intermediate.Image_flatten
[ OK ] _02_Intermediate.Image_flatten (1 ms)
[ RUN ] _02_Intermediate.Image_getHistogram
[ OK ] _02_Intermediate.Image_getHistogram (2 ms)
[ RUN ] _02_Intermediate.Image_depictsHistogram
[ OK ] _02_Intermediate.Image_depictsHistogram (3 ms)
[ RUN ] _02_Intermediate.INTEGRATION_ImageP2
[ OK ] _02_Intermediate.INTEGRATION_ImageP2 (49 ms)
[-----] 6 tests from _02_Intermediate (56 ms total)

[-----] 6 tests from _03_Advanced
[ RUN ] _03_Advanced.UnitByte_encodeByte
[ OK ] _03_Advanced.UnitByte_encodeByte (1 ms)
[ RUN ] _03_Advanced.UnitByte_decodeByte
[ OK ] _03_Advanced.UnitByte_decodeByte (1 ms)
[ RUN ] _03_Advanced.UnitByte_decomposeByte
[ OK ] _03_Advanced.UnitByte_decomposeByte (1 ms)
[ RUN ] _03_Advanced.Image_readFromFile
[ OK ] _03_Advanced.Image_readFromFile (7 ms)
[ RUN ] _03_Advanced.Image_saveToFile
[ OK ] _03_Advanced.Image_saveToFile (2 ms)
[ RUN ] _03_Advanced.Image_extractObjects
[ OK ] _03_Advanced.Image_extractObjects (7 ms)
[-----] 6 tests from _03_Advanced (19 ms total)

[-----] Global test environment tear-down
[=====] 35 tests from 3 test suites ran. (102 ms total)
[ PASSED ] 35 tests.
```