Planos y ASCII

2

Generado por Doxygen 1.8.11

Índice general

Capítulo 1

Índice de clases

1.1. Lista de clases

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

Capítulo 2

Indice de archivos

2.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

Capítulo 3

Documentación de las clases

3.1. Referencia de la Estructura Celda

Atributos públicos

- string **datos**
- Celda * siguiente

  valor de la celda actual

3.1.1. Descripción detallada

Definición en la línea 17 del archivo lista.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- include/lista.h

3.2. Referencia de la Clase Imagen

Una imagen en blanco y negro. Cada píxel es un byte.

```
#include <imagen.h>
```

Métodos públicos

- Imagen ()

  Construye una imagen vacía (0 filas, 0 columnas)
- Imagen (const int filas, const int columnas)

  Construye una imagen negra de tamaño filas x columnas.
- void crear (int filas, int columnas)

  Crea una imagen negra de tamaño filas x columnas.
- const int filas ()

  Devuelve el número de filas de las imagen.
- const int columnas ()

  Devuelve el número de columnas de las imagen.
- void set (int y, int x, byte v)

  Asigna el valor v a la posición (x,y) de la imagen.
- const byte get (int y, int x) const

  Devuelve el valor de la posición (x,y) de la imagen.
- bool leerImagen (const char nombreFichero[ ])

  Carga una imagen desde un fichero.
- bool escribirImagen (const char nombreFichero[ ], bool esBinario)

  Guarda una imagen desde un fichero.
- bool aArteASCII (const char grises[ ], char arteASCII[ ], int maxlong)

  Guarda un plano de imagen desde un fichero.
- void **destruir** ()
- void **setPos** (int i, byte v)
- bool **listaAArteASCII** (const Lista &celdas)
- **Imagen** (const Imagen &otra)
- void **copiarImagen** (const Imagen &otra)
- Imagen & **operator=** (const Imagen &otra)
- Imagen **operator+** (const Imagen &otra) const

3.2.1. Descripción detallada

Una imagen en blanco y negro. Cada píxel es un byte.

Definición en la línea 23 del archivo imagen.h.

3.2.2. Documentación del constructor y destructor

3.2.2.1. Imagen::Imagen (const int *filas*, const int *columnas*)

Construye una imagen negra de tamaño *filas* x *columnas*.

Parámetros

filas	número de filas de la imagen
columnas	número de columnas de la imagen

Construye una imagen de tamaño *filas* x *columnas* y pone todos sus elementos a 0.

Definición en la línea 17 del archivo imagen.cpp.

```
17                                      {
18      this->datos = 0;
19      crear(filas, columnas);
20    }
```

3.2.3. Documentación de las funciones miembro

3.2.3.1. bool Imagen::aArteASCII (const char *grises[ ],* char *arteASCII[ ],* int *maxlong*)

Guarda un plano de imagen desde un fichero.

Ver también

plano

Parámetros

plano	plano que queremos explorar

Devuelve

plano imagen obtenida en el plano k

Ver también

aArteASCII Guarda una imagen formada por caracteres desde un fichero

Parámetros

grises	caracteres para formar la imagen
arteAS↩ CII	nombre del fichero que contiene la imagen transformada en caracteres
maxlong	maximo de caracteres que puede almacenar la imagen ASCII

Valores devueltos

true	si ha tenido éxito en la escritura
false	si se ha producido algún error en la escritura

Definición en la línea 193 del archivo imagen.cpp.

```
193                                                                                 {
194      bool correcto = true;
195      int tamG = 0;
196      int contador = 0;
197
198      while (grises[tamG] != '\0'){
199        tamG++;
```

```
200     }
201
202     if (maxlong >= nfilas*(ncolumnas+1)+1){
203       for (int i = 0; i < filas(); i++){
204         for (int j = 0; j < columnas(); j++){
205           arteASCII[contador]=grises[get(i,j)*tamG/256];
206           contador++;
207         }
208         arteASCII[contador] = '\n';
209         contador++;
210       }
211     }
212     else
213       correcto = false;
214
215     arteASCII[contador]='\0';
216     return correcto;
217   }
```

3.2.3.2. const int Imagen::columnas ( )

Devuelve el número de columnas de las imagen.

Devuelve

el número de columnas de la imagen

Definición en la línea 125 del archivo imagen.cpp.

```
125                             {
126     return ncolumnas;
127   }
```

3.2.3.3. void Imagen::crear (int *filas*, int *columnas*)

Crea una imagen negra de tamaño *filas* x *columnas*.

Parámetros

filas	número de filas de la imagen
columnas	número de columnas de la imagen

Dimensiona la imagen a tamaño *filas* x *columnas* y pone todos sus elementos a 0.

Definición en la línea 24 del archivo imagen.cpp.

```
24                                          {
25
26     destruir();
27
28     nfilas=filas;
29     ncolumnas=columnas;
30
31     datos = new byte*[filas];
32     datos[0] = new byte[filas*columnas];
33
34
35     for(int i = 1; i < filas; i++) //Recorremos la imagen
36             datos[i] = &datos[0][i*columnas];
37
```

```
38      for(int i=0; i < nfilas; i++)
39          for(int j = 0; j < ncolumnas; j++)
40                datos[i][j] = 0;
41
42    }
```

3.2.3.4. bool Imagen::escribirImagen (const char *nombreFichero[ ],* bool *esBinario*)

Guarda una imagen desde un fichero.

Parámetros

nombreFichero	nombre del fichero que contiene la imagen
esBinario	toma el valor `true` si el fichero se escribe en formato binario o `false` en caso contrario.

Valores devueltos

true	si ha tenido éxito en la escritura
false	si se ha producido algún error en la escritura

Definición en la línea 180 del archivo imagen.cpp.

```
180                                                                    {
181
182     bool result;
183     if(esBinario)
184       result = escribirPGMBinario (nombreFichero, datos[0], nfilas, ncolumnas);
185     else
186       result = escribirPGMTexto (nombreFichero, datos[0], nfilas, ncolumnas);
187
188     return result;
189   }
```

3.2.3.5. const int Imagen::filas ( )

Devuelve el número de filas de las imagen.

Devuelve

el número de filas de la imagen

Definición en la línea 119 del archivo imagen.cpp.

```
119                          {
120     return nfilas;
121   }
```

3.2.3.6. const byte Imagen::get (int *y,* int *x*) const

Devuelve el valor de la posición (*x*,*y*) de la imagen.

Parámetros

y	fila de la imagen
x	columna de la imagen

Devuelve

el valor de la posición (*x*,*y*) de la imagen

Devuelve el valor de la posición (*x*,*y*) de la imagen. Dado que la imagen se guarda como un vector, la posición (*x*,*y*) corresponde a la posición *y* ∗ `ncolumnas` + *x* del vector.

Definición en la línea 142 del archivo imagen.cpp.

```
142                                         {
143     return datos[x][y];
144   }
```

3.2.3.7. bool Imagen::leerImagen (const char *nombreFichero[ ]*)

Carga una imagen desde un fichero.

Parámetros

nombreFichero	nombre del fichero que contiene la imagen

Valores devueltos

true	si ha tenido éxito en la lectura
false	si se ha producido algún error en la lectura

Lee desde disco los datos de la imagen llamada *nombreFichero* y la guarda en la memoria. La función debe asegurarse de que la imagen es de un tipo de imagen conocido y de que su tamaño es menor del tamaño máximo permitido (`MAXDATOS`).

Definición en la línea 160 del archivo imagen.cpp.

```
160                                                 {
161     bool resultado = false;
162     TipoImagen mi_tipo;
163
164     mi_tipo = infoPGM(nombreFichero, nfilas, ncolumnas);
165
166     if(mi_tipo == IMG_PGM_BINARIO){
167       crear(nfilas, ncolumnas);
168       resultado = leerPGMBinario(nombreFichero, datos[0], nfilas, ncolumnas);
169     }
170     else if(mi_tipo == IMG_PGM_TEXTO){
171       crear(nfilas, ncolumnas);
172       resultado = leerPGMBinario(nombreFichero, datos[0], nfilas, ncolumnas);
173     }
174
175     return resultado;
176   }
```

3.2.3.8. void Imagen::set (int *y*, int *x*, byte *v*)

Asigna el valor *v* a la posición (*x*,*y*) de la imagen.

Parámetros

y	fila de la imagen
x	columna de la imagen
v	valor a asignar

Asigna el valor *v* a la posición (*x*,*y*) de la imagen. Dado que la imagen se guarda como un vector, la posición (*x*,*y*) corresponde a la posición *y* * `ncolumnas` + *x* del vector.

Definición en la línea 131 del archivo imagen.cpp.

```
131                                     {
132     datos[x][y]=v;
133   }
```

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

- include/imagen.h
- src/imagen.cpp

3.3. Referencia de la Clase Lista

Métodos públicos

- Lista ()

  Construye una lista vacia (0 elementos)
- ~Lista ()

  Libera la memoria reservada en la lista de cadenas.
- void destruir ()

  Libera la memoria reservada en la lista de cadenas.
- **Lista** (const Lista &otra)
- Lista & **copiarLista** (const Lista &otra)
- Lista & **operator=** (const Lista &otra)
- Lista & **operator+** (const string mi_string)
- Lista (string cadena)

  Construye una lista a partir de un elemento.
- void insertar (string cadena)

  inserta una nueva cadena al final de la lista
- string getCelda (int i) const

  devuelve la cadena de la posicion i-esima de la lista o una cadena vacia en caso de que el valor de i sea erroneo.
- int longitud () const

  devuelve el numero de celdas que contiene la lista
- bool leerLista (const char nombrefichero[ ])

  Construye una lista de celdas enlazadas a partir de la informacion contenida en un fichero.

3.3.1. Descripción detallada

Definición en la línea 22 del archivo lista.h.

3.3.2. Documentación del constructor y destructor

3.3.2.1. Lista::∼Lista ( )

Libera la memoria reservada en la lista de cadenas.

Libera la memoria reservada en el vetor de imagen y actualiza el numero de elementos de la misma a 0.

Definición en la línea 26 del archivo lista.cpp.

```
26              {
27   destruir();
28 }
```

3.3.2.2. Lista::Lista (string *cadena*)

Construye una lista a partir de un elemento.

Parámetros

cadena	el elemento a insertar en la lista

Construye una lista de tamaño 1 e inserta la cadena *cadena*

Definición en la línea 20 del archivo lista.cpp.

```
20                          {
21   insertar(valor);
22 }
```

3.3.3. Documentación de las funciones miembro

3.3.3.1. void Lista::destruir ( )

Libera la memoria reservada en la lista de cadenas.

Libera la memoria reservada en el vetor de imagen y actualiza el numero de elementos de la misma a 0.

Definición en la línea 32 del archivo lista.cpp.

```
32                      {
33     Celda *ptr = cabecera;
34     while (cabecera != 0) {
35       ptr=cabecera;
36       cabecera=cabecera->siguiente;
37       delete ptr;
38     }
39     delete cabecera;
40 }
```

3.3.3.2. string Lista::getCelda (int *i*) const

devuelve la cadena de la posicion i-esima de la lista o una cadena vacia en caso de que el valor de i sea erroneo.

Parámetros

i	indice del elemento dentro de la lista

Devuelve

la cadena que se encuentra en la celda con indice *i* siempre que este valor se encuentre en los margenes de la lista, o una cadena vacia en caso contrario

Definición en la línea 110 del archivo lista.cpp.

```
110                                     {
111   string cadena = "";
112     int cnt=0;
113     Celda *puntero = cabecera;
114 
115     if(puntero != 0){
116         while(cnt != pos){
117             puntero = puntero->siguiente;
118             cnt++;
119         }
120         cadena = puntero->datos;
121     }
122 
123     return cadena;
124 }
```

3.3.3.3. void Lista::insertar (string *cadena*)

inserta una nueva cadena al final de la lista

Parámetros

cad	elemento a insertar en la lista

Añade un nuevo elemento (*cadena*) a la lista

Definición en la línea 89 del archivo lista.cpp.

```
89                                     {
90    Celda *celda = new Celda();
91   Celda *puntero;
92 
93     celda -> datos = valor;
94     celda -> siguiente = 0;
95 
96     if(cabecera == 0)
97         cabecera=celda;
98     else{
99         puntero=cabecera;
100 
101         while(puntero->siguiente!=0)
102             puntero = puntero->siguiente;
103 
104         puntero->siguiente = celda;
105     }
106 }
```

3.3.3.4. bool Lista::leerLista (const char *nombrefichero[ ]*)

Construye una lista de celdas enlazadas a partir de la informacion contenida en un fichero.

Parámetros

nombreFichero	ruta del fichero de texto con el contenido de las cadenas a insertar en la lista

Valores devueltos

true	si ha tenido éxito en la lectura y el formato es el correcto
false	si se ha producido algún error en la lectura

Lee desde disco los elementos almacenados en *nombreFichero* y los guarda en la lista. La función debe asegurarse de que la estructura sigue un patron determinado, y se ha de crear la lista con el numero de elementos que contenga.

Definición en la línea 144 del archivo lista.cpp.

```
144                                             {
145     ifstream fin;
146     fin.open(nombrefichero);
147     if(!fin){
148         return false;
149     }else{
150         string grises;
151         int lineas;
152         getline(fin,grises); //la primera linea se ignora
153         fin » lineas; //leo el numero de datos de grises
154         getline(fin,grises); //leer salto de linea
155         if (!fin){
156             return false;
157         }else {
158             int i = 0;
159             getline(fin,grises); //leer cadena de caracteres
160             while ((i < lineas)&&(fin)){
161                 if (grises != ""){
162                     insertar(grises);
163                     i++;
164                 }
165                 getline(fin,grises); //leer cadena de caracteres
166             }
167         }
168         fin.close();
169     }
170     return true;
171 }
```

3.3.3.5. int Lista::longitud ( ) const

devuelve el numero de celdas que contiene la lista

Devuelve

el tamaño de la lista

Definición en la línea 128 del archivo lista.cpp.

```
128                         {
129   int cnt = 0;
130     if(cabecera != 0){
131         Celda *puntero = cabecera;
132         cnt++;
133 
134         while(puntero->siguiente != 0){
135             puntero = puntero->siguiente;
136             cnt++;
137         }
138     }
139     return cnt;
140 }
```

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

- include/lista.h
- src/lista.cpp

Capítulo 4

Documentación de archivos

4.1. Referencia del Archivo include/byte.h

Funciones de manejo de bytes.

```
#include <iostream>
#include <string>
```

'typedefs'

- typedef unsigned char byte

  Un `byte` contiene el estado de 8 bytes.

Funciones

- void on (byte &b, int pos)

  enciende el byte `pos` del `byte b`
- void off (byte &b, int pos)

  apaga el byte `pos` del `byte b`
- bool getbit (byte b, int pos)

  devuelve el estado del byte (encendido = true, apagado = false) en la posición `pos`
- void print (byte b)

  Muestra por la salida estándar una secuencia de 0s y 1s correspondiente al estado de cada byte.
- void encender (byte &b)

  enciende todos los bytes
- void apagar (byte &b)

  apaga todos los bytes
- void asignar (byte &b, const bool v[ ])

  enciende los bytes según la configuración de `v`
- void volcar (byte b, bool v[ ])

  recupera la configuración de todos los byte
- void encendidos (byte b, int posic[ ], int &cuantos)

  devuelve las posiciones de los bytes encendidos
- string byteToString (byte b)

  tranforma el valor de los bytes a string

4.1.1. Descripción detallada

Funciones de manejo de bytes.

Autor

Antonio Miguel Morillo Chica

Fecha

12 Abril de 2016, 20:45

4.1.2. Documentación de las funciones

4.1.2.1. void apagar (byte & *b*)

apaga todos los bytes

Parámetros

b	el `byte` que se quiere apagar completamente.

Definición en la línea 51 del archivo byte.cpp.

```
51                      {
52     unsigned char mask=0;
53     b = (b & mask);
54 }
```

4.1.2.2. void asignar (byte & *b*, const bool *v[]*)

enciende los bytes según la configuración de `v`

Parámetros

b	el `byte` sobre el que se quiere actuar
v	vector de 8 elementos que contiene la secuencia de byteS que se quieren asignar

Asigna a `b` la configuración de bytes contenida en `v`. `v` es un vector de 8 booleanos donde `true` significa encendido y `false` significa apagado.

Definición en la línea 56 del archivo byte.cpp.

```
56                                      {
57     apagar(b);
58     bool valor;
59
60     //b = (b | v[])
61     for(int i=0; i <= 7; i++){
62         valor = v[i];
```

```
63      if(valor == 1)
64          on(b,i);
65      }
66  }
```

4.1.2.3. string byteToString (byte *b*)

tranforma el valor de los bytes a string

Parámetros

byte	b bloque de bytes que queremos convertir a string

Devuelve

devuelve el valor convertido en un string

Definición en la línea 88 del archivo byte.cpp.

```
88                              {
89
90      string salida;
91      for (int i = 7; i <= 0; ++i)
92          if(getbit(b, i))
93              salida += '1';
94          else
95              salida += '0';
96
97      return salida;
98  }
```

4.1.2.4. void encender (byte & *b*)

enciende todos los bytes

Parámetros

b	el `byte` que se quiere encender completamente.

Definición en la línea 45 del archivo byte.cpp.

```
45                          {
46      unsigned char mask=0;
47      mask = mask ^ mask;
48      b = (b | mask);
49  }
```

4.1.2.5. void encendidos (byte *b*, int *posic[ ]*, int & *cuantos*)

devuelve las posiciones de los bytes encendidos

Parámetros

b	el `byte` que se quiere consultar
posic	vector de enteros (valores entre 0 a 7) que indican la posición de los bytes de `b` que están encendidos
cuantos	número de bytes encendidos en `b` (número de elementos usados en el vector `posic`)

Definición en la línea 74 del archivo byte.cpp.

```
74                                                  {
75
76      for(int i=0; i<=7; i++){
77          if(getbit(b,i)){
78              posic[i]=1;
79              cuantos++;
80          }
81          else{
82              posic[i]=0;
83              cuantos++;
84          }
85      }
86  }
```

4.1.2.6. bool getbit (byte *b*, int *pos*)

devuelve el estado del byte (encendido = true, apagado = false) en la posición `pos`

Parámetros

b	el `byte` que se quiere consultar
pos	el byte dentro de @ b que se quiere consultar (0 más a la derecha)

Valores devueltos

true	si el byte en la posición `pos` está encendido
false	si el byte en la posición `pos` está apagado

Definición en la línea 21 del archivo byte.cpp.

```
21                              {
22
23      unsigned char mask=1;
24      mask = (mask<<pos);
25      bool salida;
26
27      if (mask & b)
28      salida = 1;
29      else
30      salida = 0;
31
32      return salida;
33
34  }
```

4.1.2.7. void off (byte & *b*, int *pos*)

apaga el byte `pos` del `byte b`

Parámetros

b	el `byte` cuyo byte se quiere desactivar
pos	el byte dentro de `b` que se quiere desactivar (0 más a la derecha)

Definición en la línea 13 del archivo byte.cpp.

```
13                          {
14
15     unsigned char mask=1;
16     mask = mask « pos;
17     b = (mask & b);
18
19 }
```

4.1.2.8. void on (byte & *b*, int *pos*)

enciende el byte `pos` del `byte b`

Parámetros

b	el `byte` cuyo byte se quiere activar
pos	el byte dentro de `b` que se quiere activar (0 más a la derecha)

Definición en la línea 5 del archivo byte.cpp.

```
5                           {
6
7     unsigned char mask=1;
8     mask = mask « pos;
9     b = (mask | b);
10
11 }
```

4.1.2.9. void print (byte *b*)

Muestra por la salida estándar una secuencia de 0s y 1s correspondiente al estado de cada byte.

Parámetros

b	el `byte` que se quiere imprimir

Muestra por la salida estándar una secuencia de 0s y 1s correspondiente al estado de cada byte del byte donde un cero representa que un byte está apagado y un uno que el byte está encendido. Se implementa utilizando la función "get".

Por ejemplo, si en el `byte b` están encendidos los bytes en posiciones 1 y 3 (0 más a la derecha), `print(b);` mostrará `00001010`

Definición en la línea 36 del archivo byte.cpp.

```
36                  {
37     for (int i = 0; i < 8; ++i)
38             if(getbit(b, i))
39                     cout « "1" « " ";
40             else
41                     cout « "0" « " ";
42     cout « endl;
43 }
```

4.1.2.10. void volcar (byte *b*, bool *v[ ]*)

recupera la configuración de todos los byte

Parámetros

b	el `byte` que se quiere consultar
v	vector de 8 elementos que contendrá el estado de cada uno de los bytes de @ b

Vuelca en `v` la configuración de bytes contenida en `b`. `true` significa encendido y `false` significa apagado. El tamaño de `v` debe ser 8.

Definición en la línea 68 del archivo byte.cpp.

```
68                              {
69     for(int i=0; i<=7;i++){
70         v[i] = getbit(b,i);
71     }
72 }
```

4.2. Referencia del Archivo include/imagen.h

Clase imagen blanco y negro.

```
#include <fstream>
#include <cstring>
#include <string>
#include "lista.h"
#include "byte.h"
#include "pgm.h"
```

Clases

- class Imagen

  Una imagen en blanco y negro. Cada píxel es un byte.

'typedefs'

- typedef unsigned char byte

  byte = 8bits almacenado en un unsigned char

4.2.1. Descripción detallada

Clase imagen blanco y negro.

Autor

Antonio Miguel Morillo Chica

Fecha

12 Abril de 2016, 20:45

4.3. Referencia del Archivo include/lista.h

Clase para la estructura de datos de lista de strings.

```
#include <string>
#include <fstream>
```

Clases

- struct Celda
- class Lista

4.3.1. Descripción detallada

Clase para la estructura de datos de lista de strings.

Permite el manejo de cadenas (strings) en una lista enlazada

4.4. Referencia del Archivo include/pgm.h

Fichero cabecera para la E/S de imágenes PGM.

```
#include <iostream>
#include "byte.h"
#include <fstream>
#include <string>
```

Enumeraciones

- enum TipoImagen { IMG_DESCONOCIDO, IMG_PGM_BINARIO, IMG_PGM_TEXTO }

  Tipo de imagen.

Funciones

- TipoImagen infoPGM (const char nombre[ ], int &filas, int &columnas)

  Consulta el tipo de imagen del archivo y sus dimensiones.
- bool leerPGMBinario (const char nombre[ ], unsigned char ∗datos, int &filas, int &columnas)

  Lee una imagen de tipo PGM binario.
- bool escribirPGMBinario (const char nombre[ ], const unsigned char ∗datos, int filas, int columnas)

  Escribe una imagen de tipo PGM binario.
- bool leerPGMTexto (const char nombre[ ], unsigned char ∗datos, int &filas, int &columnas)

  Lee una imagen de tipo PGM Texto.
- bool escribirPGMTexto (const char nombre[ ], const unsigned char ∗datos, int filas, int columnas)

  Escribe una imagen de tipo PGM Texto.

4.4.1. Descripción detallada

Fichero cabecera para la E/S de imágenes PGM.

Permite la E/S de archivos de tipos PGM

4.4.2. Documentación de las enumeraciones

4.4.2.1. enum TipoImagen

Tipo de imagen.

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

Ver también

LeerTipoImagen

Valores de enumeraciones

IMG_DESCONOCIDO Tipo de imagen desconocido.

IMG_PGM_BINARIO Imagen tipo PGM Binario.

IMG_PGM_TEXTO Imagen tipo PGM Texto.

Definición en la línea 25 del archivo pgm.h.

```
25                {
26   IMG_DESCONOCIDO,
27   IMG_PGM_BINARIO,
28   IMG_PGM_TEXTO
29 };
```

4.4.3. Documentación de las funciones

4.4.3.1. bool escribirPGMBinario (const char *nombre[ ],* const unsigned char ∗ *datos,* int *filas,* int *columnas*)

Escribe una imagen de tipo PGM binario.

Parámetros

nombre	nombre del archivo a escribir
datos	vector con *filas* x *columnas* bytes que corresponden a los valores de los píxeles de la imagen de grises.
filas	número de filas de la imagen
columnas	número de columnas de la imagen

Valores devueltos

true	si ha tenido éxito en la escritura.
false	si se ha producido algún error en la escritura.

Definición en la línea 98 del archivo pgm.cpp.

```
99                          {
100
101   ofstream f(nombre);
102   bool res= true;
103
104   if (f) {
105     f « "P5" « endl;
106     f « columnas « ' ' « filas « endl;
107     f « 255 « endl;
108     f.write(reinterpret_cast<const char *>(datos),filas*columnas);
109     if (!f) res=false;
110   }
111   return res;
112 }
```

4.4.3.2. bool escribirPGMTexto (const char *nombre[ ],* const unsigned char * *datos,* int *filas,* int *columnas*)

Escribe una imagen de tipo PGM Texto.

Parámetros

nombre	nombre del archivo a escribir
datos	vector con *filas* x *columnas* bytes que corresponden a los valores de los píxeles de la imagen de grises.
filas	número de filas de la imagen
columnas	número de columnas de la imagen

Valores devueltos

true	si ha tenido éxito en la escritura.
false	si se ha producido algún error en la escritura.

Definición en la línea 138 del archivo pgm.cpp.

```
139                          {
140
141   ofstream f(nombre);
142   bool res= true ;
```

```
143
144   int numero;
145   if (f) {
146     f « "P2" « endl;
147     f « columnas « ' ' « filas « endl;
148     f « 255 « endl;
149     for (int i = 0; i < filas*columnas; i++){
150       numero = int(datos[i]);
151       f « numero;
152       f « ' ';
153     }
154
155     if (!f)
156       res=false;
157   }
158
159   return res;
160 }
```

4.4.3.3. TipoImagen infoPGM (const char *nombre[ ],* int & *filas,* int & *columnas*)

Consulta el tipo de imagen del archivo y sus dimensiones.

Parámetros

nombre	indica el nombre del archivo de disco a consultar
filas	Parámetro de salida con las filas de la imagen.
columnas	Parámetro de salida con las columnas de la imagen.

Devuelve

Devuelve el tipo de la imagen en el archivo

Ver también

TipoImagen

Definición en la línea 63 del archivo pgm.cpp.

```
63                                                                  {
64
65     TipoImagen tipo;
66     filas=columnas=0;
67     ifstream f(nombre);
68
69     tipo=LeerTipo(f);
70     if (tipo!=IMG_DESCONOCIDO)
71       if (!LeerCabecera(f,filas,columnas)) {
72         tipo=IMG_DESCONOCIDO;
73       }
74
75     return tipo;
76 }
```

4.4.3.4. bool leerPGMBinario (const char *nombre[ ],* unsigned char * *datos,* int & *filas,* int & *columnas*)

Lee una imagen de tipo PGM binario.

Parámetros

nombre	nombre del archivo a leer
filas	Parámetro de salida con las filas de la imagen.
columnas	Parámetro de salida con las columnas de la imagen.
datos	vector para obtener el valor de cada uno de los píxeles desde la esquina superior izqda a la inferior dcha.

Valores devueltos

true	si ha tenido éxito en la lectura.
false	si se ha producido algún error en la lectura.

Precondición

datos debe tener tamaño suficiente para almacenar *filas* x *columnas* bytes de datos de la imagen.

Definición en la línea 80 del archivo pgm.cpp.

```
81              {
82
83   bool exito= false;
84   filas=0;
85   columnas=0;
86   ifstream f(nombre);
87
88   if (LeerTipo(f)==IMG_PGM_BINARIO)
89     if (LeerCabecera (f, filas, columnas))
90         if (f.read(reinterpret_cast<char *>(datos),filas*columnas))
91         exito= true;
92
93   return exito;
94 }
```

4.4.3.5. bool leerPGMTexto (const char *nombre[ ],* unsigned char * *datos,* int & *filas,* int & *columnas*)

Lee una imagen de tipo PGM Texto.

Parámetros

nombre	nombre del archivo a leer
filas	Parámetro de salida con las filas de la imagen.
columnas	Parámetro de salida con las columnas de la imagen.
datos	vector para obtener el valor de cada uno de los píxeles desde la esquina superior izqda a la inferior dcha.

Valores devueltos

true	si ha tenido éxito en la lectura.
false	si se ha producido algún error en la lectura.

Precondición

datos debe tener tamaño suficiente para almacenar *filas* x *columnas* bytes de datos de la imagen.

Definición en la línea 116 del archivo pgm.cpp.

```
117              {
118
119   bool exito = false;
120   ifstream f(nombre);
121   int i, numero;
122
123   if (LeerTipo(f)==IMG_PGM_TEXTO)
124     if (LeerCabecera (f, filas, columnas))
125       for (i = 0; i < filas*columnas; i++){
126         f » numero;
127         datos[i] = numero;
128       }
129
130   if(f)
131     exito = true;
132
133   return exito;
134 }
```

4.5. Referencia del Archivo src/pgm.cpp

Fichero con las definiciones para la E/S de imágenes PGM.

```
#include "pgm.h"
```

Funciones

- TipoImagen **LeerTipo** (ifstream &f)
- char **SaltarSeparadores** (ifstream &f)
- bool **LeerCabecera** (ifstream &f, int &filas, int &columnas)
- TipoImagen infoPGM (const char nombre[ ], int &filas, int &columnas)

  Consulta el tipo de imagen del archivo y sus dimensiones.
- bool leerPGMBinario (const char nombre[ ], unsigned char ∗datos, int &filas, int &columnas)

  Lee una imagen de tipo PGM binario.
- bool escribirPGMBinario (const char nombre[ ], const unsigned char ∗datos, int filas, int columnas)

  Escribe una imagen de tipo PGM binario.
- bool leerPGMTexto (const char nombre[ ], unsigned char ∗datos, int &filas, int &columnas)

  Lee una imagen de tipo PGM Texto.
- bool escribirPGMTexto (const char nombre[ ], const unsigned char ∗datos, int filas, int columnas)

  Escribe una imagen de tipo PGM Texto.

4.5.1. Descripción detallada

Fichero con las definiciones para la E/S de imágenes PGM.

Permite la E/S de archivos de tipos PGM

4.5.2. Documentación de las funciones

4.5.2.1. bool escribirPGMBinario (const char *nombre[ ],* const unsigned char ∗ *datos,* int *filas,* int *columnas*)

Escribe una imagen de tipo PGM binario.

Parámetros

nombre	nombre del archivo a escribir
datos	vector con *filas* x *columnas* bytes que corresponden a los valores de los píxeles de la imagen de grises.
filas	número de filas de la imagen
columnas	número de columnas de la imagen

Valores devueltos

true	si ha tenido éxito en la escritura.
false	si se ha producido algún error en la escritura.

Definición en la línea 98 del archivo pgm.cpp.

```
99                          {
100
101   ofstream f(nombre);
102   bool res= true;
103
104   if (f) {
105     f « "P5" « endl;
106     f « columnas « ' ' « filas « endl;
107     f « 255 « endl;
108     f.write(reinterpret_cast<const char *>(datos),filas*columnas);
109     if (!f) res=false;
110   }
111   return res;
112 }
```

4.5.2.2. bool escribirPGMTexto (const char *nombre[ ],* const unsigned char * *datos,* int *filas,* int *columnas*)

Escribe una imagen de tipo PGM Texto.

Parámetros

nombre	nombre del archivo a escribir
datos	vector con *filas* x *columnas* bytes que corresponden a los valores de los píxeles de la imagen de grises.
filas	número de filas de la imagen
columnas	número de columnas de la imagen

Valores devueltos

true	si ha tenido éxito en la escritura.
false	si se ha producido algún error en la escritura.

Definición en la línea 138 del archivo pgm.cpp.

```
139                          {
140
141   ofstream f(nombre);
142   bool res= true ;
```

```
143
144   int numero;
145   if (f) {
146     f « "P2" « endl;
147     f « columnas « ' ' « filas « endl;
148     f « 255 « endl;
149     for (int i = 0; i < filas*columnas; i++){
150       numero = int(datos[i]);
151       f « numero;
152       f « ' ';
153     }
154
155     if (!f)
156       res=false;
157   }
158
159   return res;
160 }
```

4.5.2.3. TipoImagen infoPGM (const char *nombre[ ],* int & *filas,* int & *columnas*)

Consulta el tipo de imagen del archivo y sus dimensiones.

Parámetros

nombre	indica el nombre del archivo de disco a consultar
filas	Parámetro de salida con las filas de la imagen.
columnas	Parámetro de salida con las columnas de la imagen.

Devuelve

Devuelve el tipo de la imagen en el archivo

Ver también

TipoImagen

Definición en la línea 63 del archivo pgm.cpp.

```
63                                                                  {
64
65     TipoImagen tipo;
66     filas=columnas=0;
67     ifstream f(nombre);
68
69     tipo=LeerTipo(f);
70     if (tipo!=IMG_DESCONOCIDO)
71       if (!LeerCabecera(f,filas,columnas)) {
72         tipo=IMG_DESCONOCIDO;
73       }
74
75     return tipo;
76 }
```

4.5.2.4. bool leerPGMBinario (const char *nombre[ ],* unsigned char * *datos,* int & *filas,* int & *columnas*)

Lee una imagen de tipo PGM binario.

Parámetros

nombre	nombre del archivo a leer
filas	Parámetro de salida con las filas de la imagen.
columnas	Parámetro de salida con las columnas de la imagen.
datos	vector para obtener el valor de cada uno de los píxeles desde la esquina superior izqda a la inferior dcha.

Valores devueltos

true	si ha tenido éxito en la lectura.
false	si se ha producido algún error en la lectura.

Precondición

datos debe tener tamaño suficiente para almacenar *filas* x *columnas* bytes de datos de la imagen.

Definición en la línea 80 del archivo pgm.cpp.

```
81              {
82
83   bool exito= false;
84   filas=0;
85   columnas=0;
86   ifstream f(nombre);
87
88   if (LeerTipo(f)==IMG_PGM_BINARIO)
89     if (LeerCabecera (f, filas, columnas))
90         if (f.read(reinterpret_cast<char *>(datos),filas*columnas))
91         exito= true;
92
93   return exito;
94 }
```

4.5.2.5. bool leerPGMTexto (const char *nombre[ ],* unsigned char * *datos,* int & *filas,* int & *columnas*)

Lee una imagen de tipo PGM Texto.

Parámetros

nombre	nombre del archivo a leer
filas	Parámetro de salida con las filas de la imagen.
columnas	Parámetro de salida con las columnas de la imagen.
datos	vector para obtener el valor de cada uno de los píxeles desde la esquina superior izqda a la inferior dcha.

Valores devueltos

true	si ha tenido éxito en la lectura.
false	si se ha producido algún error en la lectura.

Precondición

datos debe tener tamaño suficiente para almacenar *filas* x *columnas* bytes de datos de la imagen.

Definición en la línea 116 del archivo pgm.cpp.

```
117              {
118
119   bool exito = false;
120   ifstream f(nombre);
121   int i, numero;
122
123   if (LeerTipo(f)==IMG_PGM_TEXTO)
124     if (LeerCabecera (f, filas, columnas))
125       for (i = 0; i < filas*columnas; i++){
126         f » numero;
127         datos[i] = numero;
128       }
129
130   if(f)
131     exito = true;
132
133   return exito;
134 }
```

Índice alfabético