



Práctica final: clases y E/S. Correlación de imágenes. Buscar a Wally.

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada



Metodología de la Programación Grado en Ingeniería Informática

AVISO IMPORTANTE: Se recuerda la prohibición de copiar total o parcialmente la práctica de otra persona. Las prácticas de todos los alumnos de todos los grupos serán comparadas con un software automático de detección de plagios. El alumno al que se le compruebe que ha plagiado la práctica, o que se ha dejado plagiar, suspenderá la asignatura con nota igual a 0.

Índice de contenido

1.Introducción.....	3
1.1.Encapsulamiento.....	3
1.2.Funciones de E/S de imágenes.....	4
2.Problemas a resolver.....	4
2.1.Ocultar/Revelar un mensaje.....	4
2.1.1.Ocultar.....	5
2.1.2.Revelar.....	6
2.2.Operación de correlación.....	6
2.2.1.Ejemplos de Filtros.....	7
2.2.2.Rango y dominio de la operación de correlación	8
2.2.3.Algoritmo de correlación	8
2.2.4.Filtrar.....	9
2.2.5.Construir un filtro.....	9
2.2.6.¿Dónde está Wally?.....	10
3.Diseño Propuesto.....	11
3.1.La interfaz de la clase Imagen.....	12
3.2.La interfaz de la clase Signal.....	12
3.2.1.Formato de archivos.....	13
3.3.Archivos y selección de la representación.....	13
4.Práctica a entregar.....	14
5.Referencias.....	15

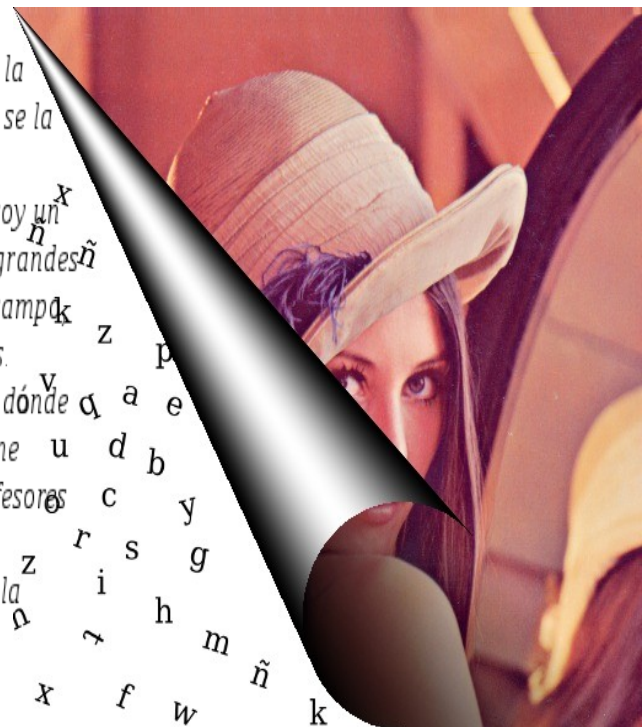


¡Hola amigos de MP!
Me llamo Wally, y la señora a la izquierda Lenna (creo que ya se la han presentado).

Bueno, hablaré ahora de mí: soy un trotamundos y me gustan las grandes aglomeraciones, ya sea en el campo, ciudad, festejos, playa y demás.

Como a veces no me acuerdo dónde estoy, les pido por favor que me busquen e informen a sus profesores de MP, dónde estoy. Si me encuentran, yo les ayudaré en la asignatura de MP.

! Espero que me encuentren!



1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

1. Practicar con un problema en el que es necesaria la modularización. Para desarrollar los programas de esta práctica, el alumno debe crear distintos archivos, compilarlos, y enlazarlos para obtener los ejecutables.
2. Practicar con el uso de la memoria dinámica. El alumno deberá usar estructuras de datos que se alojan en memoria dinámica.
3. Profundizar en los conceptos relacionados con la abstracción de tipos de datos.
4. Practicar con el uso de clases como herramienta para implementar los tipos de datos donde se requiera encapsular la representación.
5. Usar los tipos que ofrece C++ para el uso de ficheros.

Los requisitos para poder realizar esta práctica son:

1. Saber manejar punteros, memoria dinámica, clases y ficheros.
2. Conocer el diseño de programas en módulos independientes, así como la compilación separada, incluyendo la creación de bibliotecas y de archivos *makefile*.
3. Conocer en qué consisten los formatos de imágenes *PGM* y *PPM* que se han dado en el primer ejercicio práctico de la asignatura ([MP2012a]).

La práctica está diseñada para que se lleve a cabo mientras se asimila la última parte de la asignatura (clases y ficheros). Así, el alumno puede empezar a realizarla después de haber asimilado los conceptos básicos sobre clases, incluyendo constructores, destructores y sobrecarga del operador de asignación.

Podrá observar que parte del contenido de esta práctica coincide con prácticas anteriores. Esta información se ha repetido para que este documento sea autocontenido, y en particular para aquellos alumnos que no hayan realizado los ejercicios anteriores.

1.1. Encapsulamiento

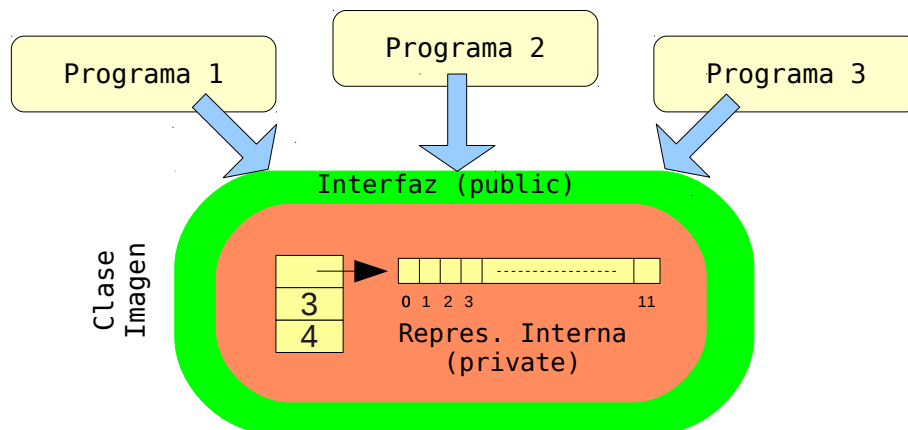
Uno de los objetivos de esta práctica es que el alumno entienda las ventajas del encapsulamiento y cómo las clases en C++ facilitan en gran medida su implementación.

En la práctica anterior ([MP2012b]) se ha presentado el encapsulamiento como una herramienta que facilita el desarrollo y mantenimiento de los programas ya que hacemos que los módulos que usan un tipo sean independientes de los detalles de su representación. En esa práctica, el alumno tenía que ser disciplinado para que sus programas no accedieran a dicha implementación. Para confirmar que todo se realizaba correctamente, proponíamos cambiar la representación para ver que todo el programa seguía siendo válido.

En esta práctica, vamos a encapsular la representación de una imagen, es decir, vamos a crear un módulo para manejar un tipo "*Imagen*". En este módulo encapsulamos la representación con una interfaz, de forma que los programas que lo usen sean independientes de los detalles internos de la representación, ya que sólo necesitarán conocer dicha interfaz.

Para crear este tipo, se usará una clase "*Imagen*" que garantiza que la representación queda encapsulada. Por tanto, el alumno deberá escoger una representación interna, la que considere más eficiente, para crear una solución en base a ella. Lógicamente, si alguna vez se deseara un cambio interno de esa representación, tendríamos garantizado que se puede realizar sin problemas, ya que el lenguaje es el que cuidará de que nuestros programas no accedan a esa parte "privada".

La siguiente figura muestra gráficamente esta idea, donde hemos enfatizado la existencia de una clase "*Imagen*" que contiene dos partes, una pública (la interfaz) y otra privada (la representación interna).



En esta práctica vamos a proponer el desarrollo de varios programas que usan la clase Imagen. Estos programas serán válidos independientemente de la representación interna que seleccionemos. Observe que en la figura anterior se ha mostrado una posible representación interna, aunque el alumno es libre de escoger la que vea conveniente.

1.2. Funciones de E/S de imágenes

El formato de imagen que vamos a manejar será *PGM (Portable Grey Map file format)*, que tiene un esquema de almacenamiento con cabecera seguida de la información. Por tanto, nuestros programas se usarán para procesar imágenes de niveles de gris.

Para simplificar la E/S de imágenes de disco, se facilita un módulo (archivo de cabecera y de definiciones), que contiene el código que se encarga de resolver la lectura y escritura del formato *PGM*. Por tanto, el alumno no necesitará estudiar los detalles de cómo es el formato interno de estos archivos. En lugar de eso, deberá usar las funciones proporcionadas para resolver ese problema. Para más detalles sobre estas funciones, puede consultar [MP2012a].

2. Problemas a resolver

En esta práctica vamos a desarrollar aplicaciones sobre imágenes para resolver dos problemas independientes:

1. Ocultar/Revelar un mensaje.
2. Realizar la operación de correlación con un filtro a una imagen. Como ejemplo práctico de esta operación se realizará la búsqueda de Wally en un escena.

Por tanto, la práctica del alumno debe permitir generar los programas ejecutables que se detallan en las secciones siguientes y que resuelven los problemas asociados.

2.1. Ocultar/Revelar un mensaje

Se van a realizar dos programas para la inserción y extracción de un mensaje "oculto" en una imagen. Para ello, modificaremos el valor de cada píxel para que contenga parte de la información a ocultar. Ahora bien, ¿cómo almacenamos un mensaje (*cadena-C*) dentro de una imagen?

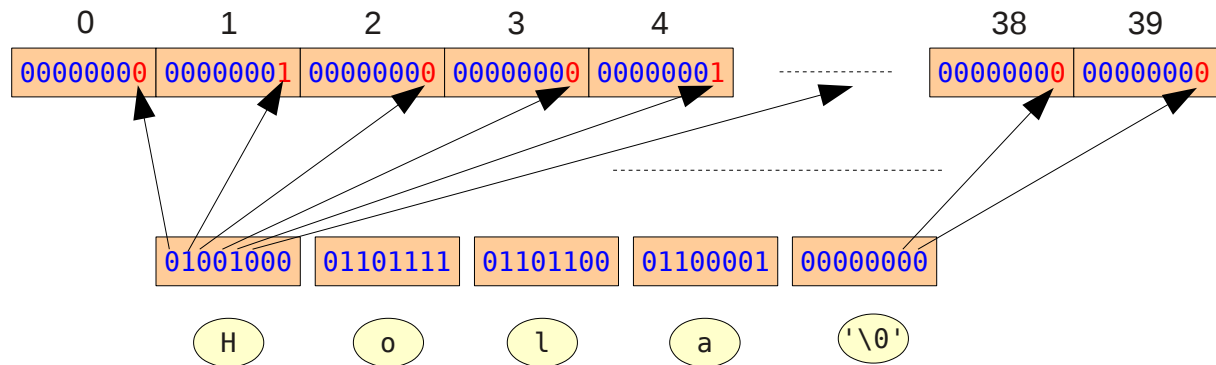
Tenga en cuenta que los valores que se almacenan en cada píxel corresponden a un valor en el rango $[0, 255]$ y que, por tanto, el contenido de una imagen no es más que una secuencia de valores consecutivos en este rango. Si consideramos que el ojo humano no es capaz de detectar cambios muy pequeños en dichos valores, podemos insertar el mensaje deseado modificando ligeramente cada uno de ellos. Concretamente, si cambiamos el valor del bit menos significativo¹, habremos afectado al valor del píxel, como mucho, en una unidad de entre las 255. La imagen la veremos, por tanto, prácticamente igual.

Ahora que disponemos del bit menos significativo para cambiarlo como deseemos, podemos

¹ El que representamos a la derecha, y que corresponde a las unidades del número binario.

usar todos los bits menos significativos de la imagen para codificar el mensaje.

Por otro lado, el mensaje será una *cadena-C*, es decir, una secuencia de valores de tipo *char* que terminan en `'\0'` (carácter con código cero). En este caso, igualmente, tenemos una secuencia de *bytes* (8 bits) que queremos insertar en la imagen. Dado que podemos modificar los bits menos significativos de la imagen, podemos “repartir” cada carácter del mensaje en 8 píxeles consecutivos. En la siguiente figura mostramos un esquema que refleja esta idea:



Como puede ver, la secuencia de 40 octetos (*bytes*) en la parte superior de la figura corresponde a los valores almacenados en el vector de “*unsigned char*” que corresponde a la imagen. Podemos suponer, por ejemplo, que la imagen es negra, y que por tanto todos los píxeles tienen un valor de cero.

En la fila inferior de la figura, podemos ver un mensaje con 4 caracteres (5 incluyendo el cero final) que corresponde a la secuencia a ocultar. Observe que se han repartido en la secuencia superior, de forma que la imagen ha quedado modificada, aunque visualmente no podremos distinguir la diferencia.

Para realizar la extracción del mensaje tendremos que resolverlo con la operación inversa, es decir, tendremos que consultar cada uno de esos bits menos significativos y colocarlos de forma consecutiva, creando una secuencia de octetos (*bytes*), hasta que extraigamos un carácter cero.

Por último, es interesante destacar que en el dibujo hemos representado una distribución de *bits* de izquierda a derecha. Es decir, el *bit* más significativo se ha insertado en el primer *byte*, el siguiente en el segundo, hasta el menos significativo que se ha insertado en el octavo. El alumno debe realizar la inserción en este orden y, obviamente, tenerlo en cuenta cuando esté revelando el mensaje codificado.

2.1.1. Ocultar

El programa de ocultación debe insertar un mensaje en una imagen. El programa recibe en la línea de órdenes el nombre de la imagen de entrada y el nombre de la imagen de salida. El mensaje de entrada se carga desde la entrada estándar hasta el fin de entrada. Un ejemplo de ejecución podría ser el siguiente:

```
prompt% ocultar lenna.pgm salida.pgm < mensaje.txt
Ocultando...
prompt%
```

El resultado de esta ejecución deberá ser una nueva imagen en disco, con nombre “*salida.pgm*”, que contendrá una imagen similar a “*lenna.pgm*”, ya que visualmente será igual, pero ocultará el mensaje que se encuentra en el fichero “*mensaje.txt*”.

Lógicamente, esta ejecución corresponde a un caso con éxito, ya que si ocurre algún tipo de error, deberá acabar con un mensaje adecuado. Por ejemplo, en caso de que la imagen indicada no exista, que tenga un formato desconocido, o que el mensaje sea demasiado grande.

2.1.2. Revelar

El programa para revelar un mensaje oculto realizará la operación inversa al anterior, es decir, deberá obtener el mensaje que previamente se haya ocultado con el programa "ocultar". Un ejemplo de ejecución podría ser el siguiente:

```
prompt% revelar salida.pgm > resultado.txt
Revelando...
prompt%
```

Observe que hemos usado la misma imagen que se ha obtenido en la ejecución anterior, y el resultado ha sido exitoso al no obtener ningún mensaje de error. Por tanto, en el fichero "resultado.txt" tendremos el mensaje original. De nuevo, tenga en cuenta que si la ejecución encuentra un error (por ejemplo la imagen no existe o el formato no es el adecuado o no tenga un mensaje) deberá terminar con el mensaje correspondiente.

2.2. Operación de correlación

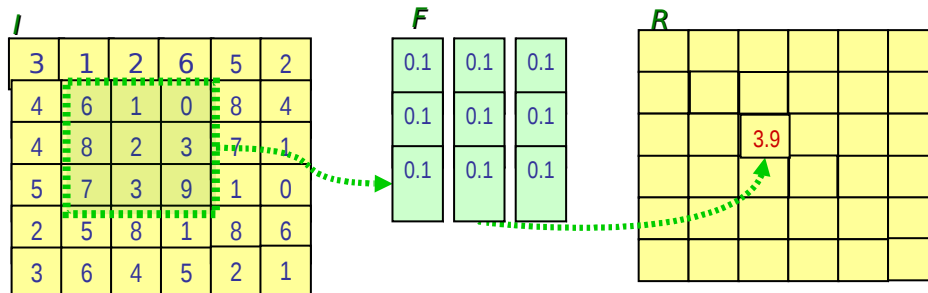
Cuando correlamos dos señales (audio, imágenes, vídeo, etc) obtenemos una medida de similitud entre ellas. La correlación entre dos señales da lugar a un valor alto cuando las señales son parecidas (un máximo cuando son iguales).

La operación de correlación aplicada sobre dos imágenes I y F da lugar a una nueva imagen R , cuyo valor en el punto (i,j) se calcula mediante la expresión:

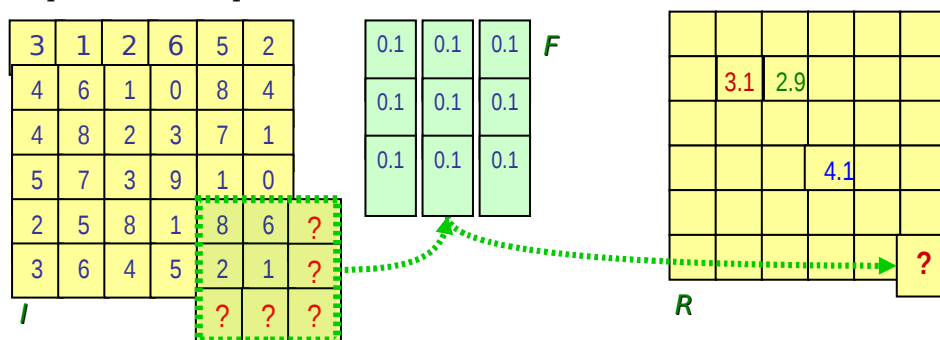
$$R(i, j) = \sum_{k=-N}^N \sum_{l=-N}^N I(i+k, j+l) * F(N+k, N+l)$$

donde $2N+1$ es el tamaño de la imagen F .

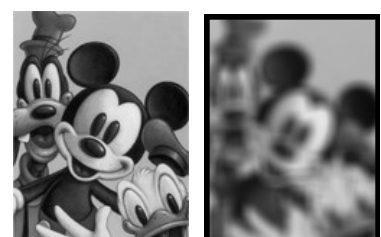
F es conocida como filtro o núcleo de correlación y suele tener tamaño cuadrado e impar.



Existen algunos puntos para los que, en principio, no está definida esta operación: *los bordes de la imagen*. Por ejemplo, en la figura inferior vemos una imagen I y un filtro F . Al aplicar el filtro sobre el píxel de la esquina inferior derecha, nos faltan valores en I .



Al no estar definida la correlación en esos puntos, el resultado sería una imagen del mismo tamaño de I pero con un valor arbitrario en esos puntos (por ejemplo cero). A la derecha vemos una imagen y el resultado de aplicar la correlación con un filtro de tamaño 9×9 . Vemos como alrededor hay un borde de 4 píxeles en el que no ha sido posible hacer ningún cálculo.



2.2.1. Ejemplos de Filtros

Los filtros pueden clasificarse según la información que resalten de la imagen de partida. Así podemos tener filtros que “emborronan” la imagen original (filtros paso bajo) o filtros que nos muestren las fronteras de los objetos en la imagen (filtros paso alto). En la Tabla 1 podemos ver diferentes filtros así como los resultados obtenidos cuando se correlacionan con la imagen “Disney”.


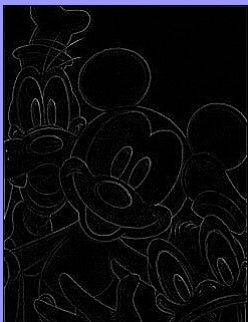

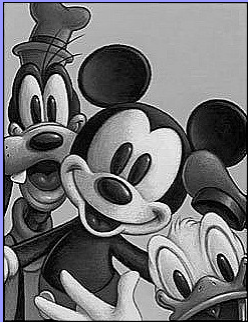
Filtro	Resultado																														
<table><tr><th colspan="5">Gaussiana (5x5)</th></tr><tr><td>0.035</td><td>0.039</td><td>0.04</td><td>0.039</td><td>0.035</td></tr><tr><td>0.039</td><td>0.043</td><td>0.044</td><td>0.043</td><td>0.039</td></tr><tr><td>0.04</td><td>0.044</td><td>0.045</td><td>0.044</td><td>0.04</td></tr><tr><td>0.039</td><td>0.043</td><td>0.044</td><td>0.043</td><td>0.039</td></tr><tr><td>0.035</td><td>0.039</td><td>0.04</td><td>0.039</td><td>0.035</td></tr></table>	Gaussiana (5x5)					0.035	0.039	0.04	0.039	0.035	0.039	0.043	0.044	0.043	0.039	0.04	0.044	0.045	0.044	0.04	0.039	0.043	0.044	0.043	0.039	0.035	0.039	0.04	0.039	0.035	
Gaussiana (5x5)																															
0.035	0.039	0.04	0.039	0.035																											
0.039	0.043	0.044	0.043	0.039																											
0.04	0.044	0.045	0.044	0.04																											
0.039	0.043	0.044	0.043	0.039																											
0.035	0.039	0.04	0.039	0.035																											
<table><tr><th colspan="3">Laplaciana</th></tr><tr><td>0.167</td><td>0.667</td><td>0.167</td></tr><tr><td>0.667</td><td>-3.333</td><td>0.667</td></tr><tr><td>0.167</td><td>0.667</td><td>0.167</td></tr></table>	Laplaciana			0.167	0.667	0.167	0.667	-3.333	0.667	0.167	0.667	0.167																			
Laplaciana																															
0.167	0.667	0.167																													
0.667	-3.333	0.667																													
0.167	0.667	0.167																													
<table><tr><th colspan="3">Media</th></tr><tr><td>0.11</td><td>0.11</td><td>0.11</td></tr><tr><td>0.11</td><td>0.11</td><td>0.11</td></tr><tr><td>0.11</td><td>0.11</td><td>0.11</td></tr></table>	Media			0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11																			
Media																															
0.11	0.11	0.11																													
0.11	0.11	0.11																													
0.11	0.11	0.11																													
<table><tr><th colspan="3">Mejora de Contraste</th></tr><tr><td>-0.17</td><td>-0.67</td><td>-0.17</td></tr><tr><td>-0.67</td><td>4.33</td><td>-0.67</td></tr><tr><td>-0.17</td><td>-0.67</td><td>-0.17</td></tr></table>	Mejora de Contraste			-0.17	-0.67	-0.17	-0.67	4.33	-0.67	-0.17	-0.67	-0.17																			
Mejora de Contraste																															
-0.17	-0.67	-0.17																													
-0.67	4.33	-0.67																													
-0.17	-0.67	-0.17																													

Tabla 1: Ejemplos de filtros y resultados obtenidos

2.2.2. Rango y dominio de la operación de correlación

La operación de correlación, pueden dar lugar a valores en un rango diferente al rango de la imagen de entrada. Para evitar perder precisión cuando aplicamos esta operación vamos a definir un nuevo tipo de dato "**Signal**" que representan a las matrices 2-D de double. Por lo tanto, la operación de correlación la aplicaremos en el rango de los reales. Para poder pasar un objeto Imagen a un objeto Signal y viceversa debemos aprovisionarnos de funciones que nos permitan hacerlo. Con tal fin definiremos dos funciones "*SignalToImagen*" e "*ImagenToSignal*". La primera pasa un objeto de tipo Signal, con valores double a un objeto de tipo Imagen, con valores unsigned char. Y la segunda pasa una Imagen con valores en el rango [0,255] a un objeto de tipo Signal con valores double en el rango [0,1]. La implementación de estas dos funciones se realizará aplicando la transformación:

$$Destino(i, j) = \begin{cases} 0 & \text{si } Origen(i, j) < 0 \\ escala * Origen(i, j) & \text{si } 0 \leq Origen(i, j) \leq max \\ escala * max & \text{si } Origen(i, j) > max \end{cases}$$

donde *max* vale 255 en el caso de que Origen sea un objeto de tipo *Imagen* y 1 en el caso de que sea de tipo *Signal* y *escala* vale 1/255 en el caso de que Origen sea un objeto de tipo *Imagen* y 255 en el caso de que sea de tipo *Signal*.

2.2.3. Algoritmo de correlación

Este algoritmo usará objetos de tipo Signal (matrices 2-D de double) tanto para la imagen a ser correlada como para el filtro usado. Los pasos del algoritmo de correlación se puede ver a continuación:

```
1. Sea I la imagen de entrada con dimensiones (num_filasI,num_colsI) de tipo Signal
2. Sea F el filtro de entrada con dimensiones (num_filasF,num_colsF) de tipo Signal
3. Sea Iout la imagen resultante con dimensiones (num_filasI,num_colsI) de tipo Signal.
4. Iniciar Iout a 0
5. mitadF=num_filasF/2, mitadC=num_colsF/2
6. Definir el entorno en que se aplicará la correlación
    7. fila_inicio = mitadF, fila_fin=num_filasI-1-mitadF
    8. col_inicio = mitadC, col_fin=num_colsI-1-mitadC
9. Para i:fila_inicio hasta fila_fin
    10. Para j:col_inicio hasta col_fin
        11. suma=0.0
        12. Para k:-mitadF hasta mitadF
            13. Para l:-mitadC hasta mitadC
                14. suma+=I(i+k,j+l)*F(k+mitadF,l+mitadC)
            15. end
        16. end
    17. Iout(i,j)=suma
    18. end
19. end
```

Algoritmo 1: Correlación entra una Imagen I y un Filtro F

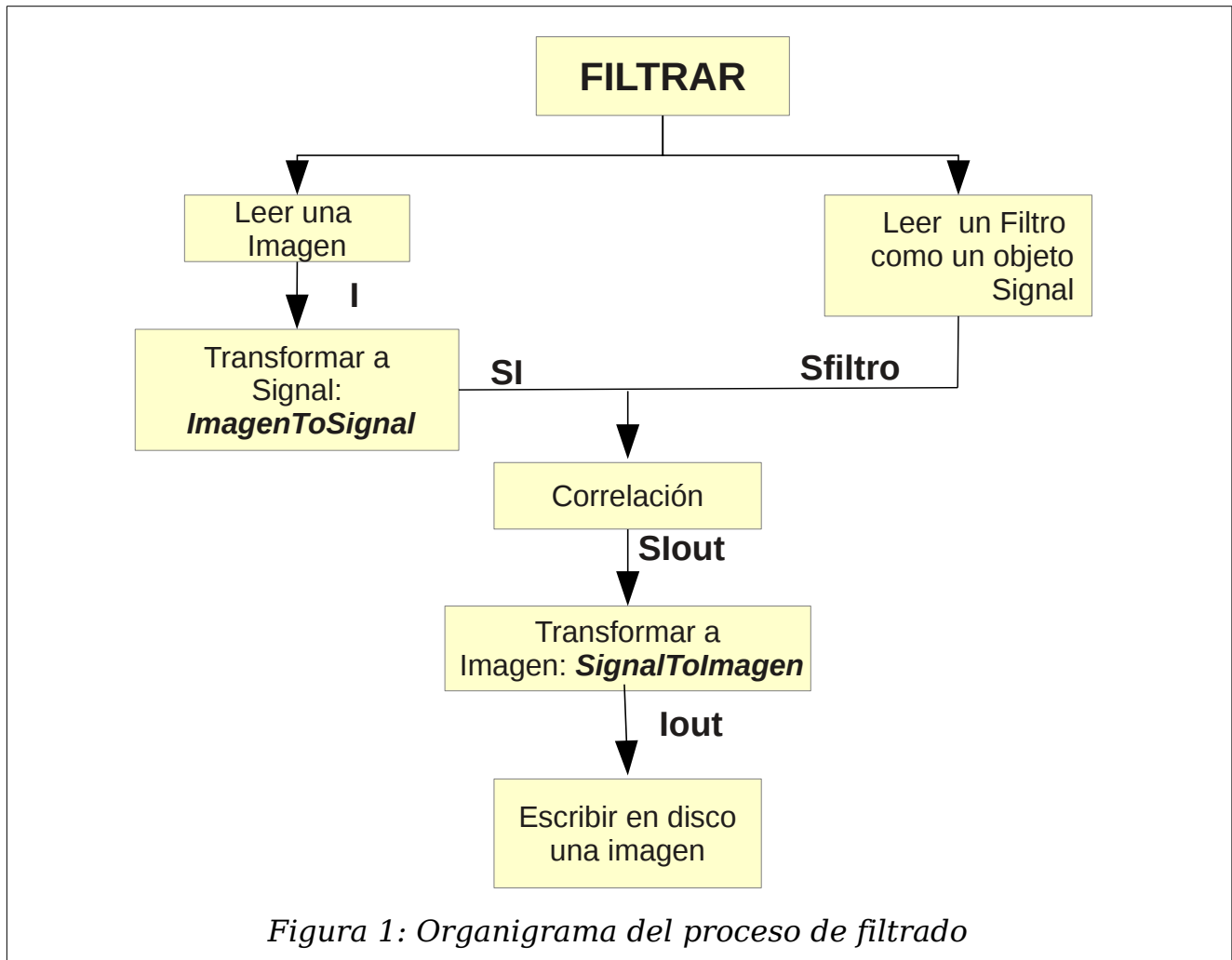
2.2.4. Filtrar

Este programa se encargará de obtener la correlación entre una imagen y un filtro. Para ello el programa leerá una imagen y un filtro (matriz 2-D de reales), y obtendrá como resultado una nueva imagen que resulta de aplicar la correlación con la imagen de entrada y el filtro. Un ejemplo de su ejecución es:

```
prompt% filtrar lenna.pgm laplaciana_txt.fil lenna_laplaciana.pgm
```

Después de esta ejecución, y si no hay errores, deberá aparecer un nuevo archivo "*lenna_laplaciana.pgm*" en el disco. Este archivo es el resultado de aplicar la correlación entre la imagen de entrada "*lenna.pgm*", con el filtro Laplaciana.

El proceso de filtrar seguirá el organigrama que se muestra en la Figura 1.



2.2.5. Construir un filtro

Este programa permitirá al usuario generar un filtro de unas dimensiones dadas y con unos valores determinados. Un ejemplo de ejecución será el siguiente:

```
prompt% construir_filtro -b 3 3 laplaciana_bin.fil
```

en esta llamada se indica al programa el numero de filas 3 y columnas 3. A continuación se le indica el nombre del fichero en disco en donde se almacenará los valores del filtro. El filtro se almacenará en disco en formato texto o en formato binario dependiendo del primer parámetro de entrada. Así si queremos almacenarlo en formato binario este parámetro será "-b" o si queremos guardarlo en formato texto este parámetro será "-t". Para mayor detalle de los

formatos de ficheros ver la sección 3.2.1. Los datos del filtro se introducirán desde la entrada estándar, bien desde teclado o redireccionando la entrada estándar a un fichero texto que contenga los valores concretos del filtro separados por espacios. Por ejemplo, redireccionando la entrada estándar al fichero laplaciana.txt sería:

```
prompt% construir_filtro -b 3 3 laplaciana_bin.fil < laplaciana.txt
```

En el fichero laplaciana.txt tenemos los siguientes datos:

```
0.17 0.67 0.17 0.67 -3.33 0.67 0.17 0.67 0.17
```

Los tres primeros valores corresponden a la primera fila, los tres segundos a las segunda fila y los últimos tres a la última fila del filtro Laplaciana.

2.2.6. ¿Dónde está Wally?

Es un juego creado por el británico Martin Handford en 1987 consistente en encontrar a Wally en una imagen con decenas de detalles que despistan al lector (ver Figura 2). Para facilitar su labor, Wally siempre va vestido del mismo modo: jersey de rayas horizontales rojo y blanco, gafas, pantalón vaquero y un gorro de lana, también de rayas. Además, suele llevar complementos como cámaras de fotos, enseres de camping o libros, su bastón, etc.



Figura 2: Imagen grande. Wally en la playa con multitud de personajes.

El objetivo en esta parte de la práctica será construir el programa **buscar_Wally**. Este programa, a partir de la imagen con todos los detalles (llamémosla imagen grande) y una imagen aislada de Wally (llamémosla imagen pequeña), obtendrá una imagen resultante con todos los detalles de la primera imagen en la que se resalta la posición de Wally. Para facilitar todo el proceso ambas imágenes estarán en formato PGM (niveles de gris). Para lograr su objetivo el alumno seguirá los siguientes pasos:

1. Obtener las fronteras de la imagen grande usando el filtro Laplaciana (*Sgrande*)
2. Obtener las fronteras de la imagen pequeña usando el filtro Laplaciana (*Speque*)

3. Obtener la *Sout* que resulta de aplicar la correlación entre *Sgrande* por *Speque*
4. Obtener la posición del pixel de mayor valor, p_{\max} , de *Sout* (este pixel será aquel donde se da mayor similitud, mayor valor de correlación, entre *Sgrande* y *Speque*)
5. Resaltar en la imagen grande el recuadro que tiene como centro p_{\max} y longitud de los lados el número de filas y columnas de *Speque*.

Una alternativa para resaltar un recuadro en una imagen (representada en un objeto de tipo *Signal*) es multiplicar toda la imagen por un factor menor que 1 (por ejemplo, 0.5) y, después, colocar sobre el recuadro a resaltar el trozo correspondiente de la imagen original.

Un ejemplo de ejecución del programa es el siguiente:

```
prompt% buscar_Wally 03.pgm 03_wally.pgm laplaciana_bin.fil res_03.pgm
```

siendo 03.pgm la imagen con todos los detalles, 03_wally.pgm la imagen aislada de Wally, laplaciana_bin.fil el fichero con el filtro Laplaciana y res_03.pgm la imagen resultante. En la Figura 3 se puede ver las imágenes de entrada y salida al realizar este ejemplo de ejecución.

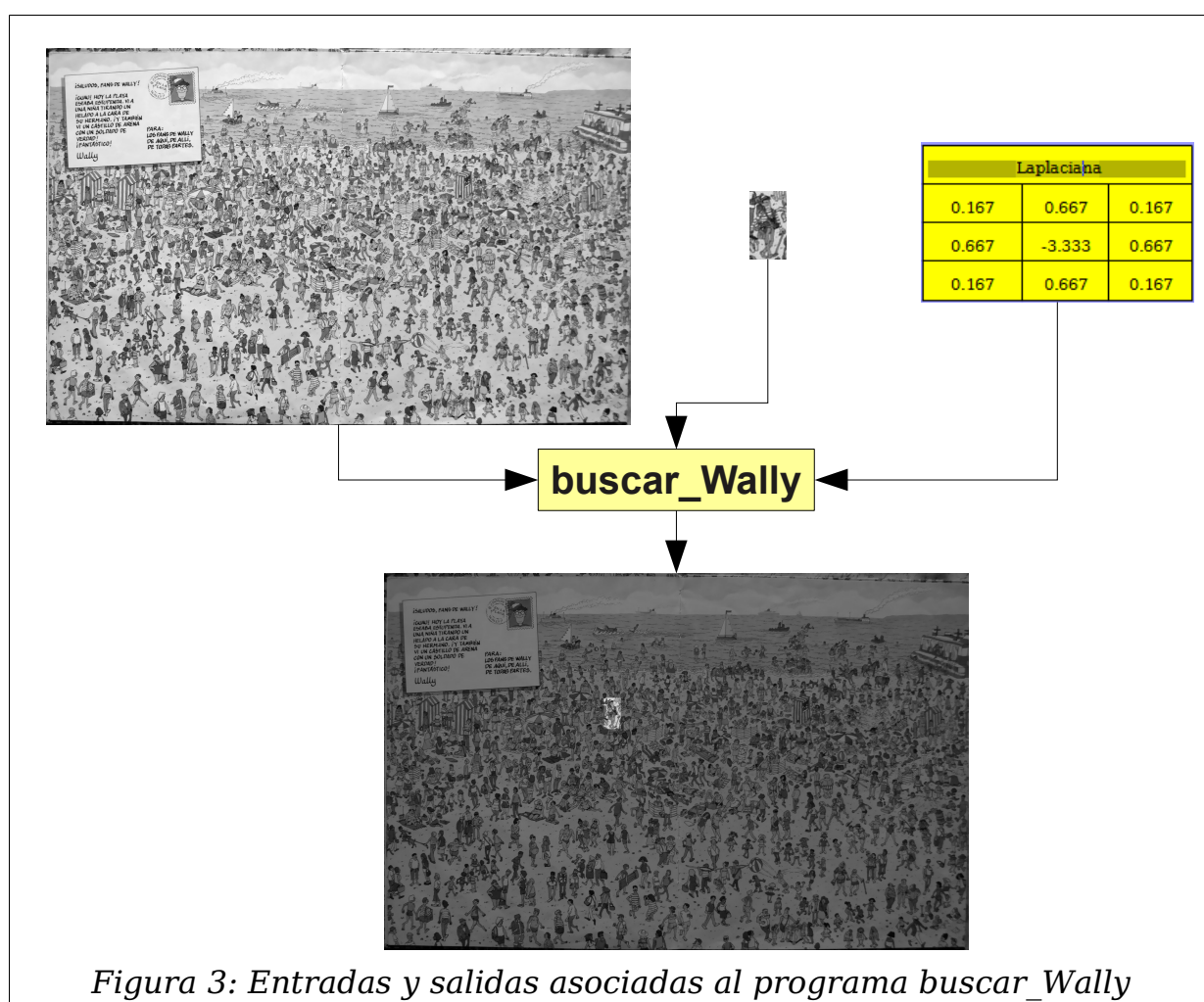


Figura 3: Entradas y salidas asociadas al programa *buscar_Wally*

3. Diseño Propuesto

Aunque los problemas se pueden resolver de forma independiente, se desea obtener una buena solución modular, de forma que favorezca la reutilización y la abstracción. Dado que las aplicaciones están relacionadas con imágenes, se propone la creación de un módulo para trabajar con este tipo de dato. Para ello, se creará la clase *Imagen*, junto con una serie de operaciones para trabajar con ella. En segundo lugar, para poder cargar en memoria un filtro y disponer de imágenes en rangos más amplios a unsigned char se creará también la clase

Signal.

3.1. La interfaz de la clase *Imagen*

Este tipo de dato se creará en memoria dinámica, para permitir procesar imágenes de cualquier tamaño. Proponemos la siguiente interfaz:

```
class Imagen {
private:
    // Implementación....

public:
    int Filas () const; // Devuelve el número de filas de m
    int Columnas () const; // Devuelve el número de columnas de m
    void Set (int f, int c, unsigned char v); // Hace img(i,j)=v
    unsigned char Get (int f, int c) const; // Devuelve img(i,j)
    bool LeerImagen(const char file[]); // Carga imagen file en img
    bool EscribirImagen(const char file[]) const; //Salva img en file
    // otros métodos...
};
```

Observe que:

- La parte interna de la clase *Imagen* no se especifica. Los campos que la componen dependen de la representación que queramos usar para la imagen.
- En esta clase será necesario incluir los constructores, destructor y operador de asignación.

Cuando decimos que nuestros programas no van a acceder a la representación, queremos decir que no deben acceder a ningún campo privado que haya en la clase *Imagen*. En lugar de eso, deberán usar la lista de métodos públicos que permiten manejarla.

Para realizar nuevas operaciones relacionadas con imágenes, puede evaluar si realizarlas dentro o fuera de la clase, es decir, como funciones miembro o como funciones externas. Tenga en cuenta que si la función se puede realizar sin acceder a la parte privada, podrá implementarse como función externa, independiente de la representación de la imagen.

3.2. La interfaz de la clase *Signal*

Este tipo de dato también se creará en memoria dinámica para permitir procesar matrices 2-D de valores reales. Servirá para mantener en memoria filtros e incluso imágenes que necesitan un rango dado por el conjunto de los números reales para su representación.

```
class Signal {
private:
    // Implementación. Tener en cuenta que el tipo de cada elemento es double ....

public:

    int Filas () const; // Devuelve el número de filas de m
    int Columnas () const; // Devuelve el número de columnas de m
    void Set (int f, int c, double v); // Hace m[i][j]=v
    double Get (int f, int c) const; // Devuelve m[i][j]
    bool LeerSignal(const char file[]); // Carga una señal de disco
    bool EscribirSignal(const char file[]) const; //Salva una señal en disco
    // otros métodos...
};
```

Observe que en esta clase será necesario incluir los constructores, destructor, el operador de

asignación y el de multiplicación por un escalar.

3.2.1. Formato de archivos

En esta práctica necesitamos leer/escribir filtros desde disco. Para poder realizarlo de una forma más segura, vamos a determinar un formato de almacenamiento de forma que cuando usamos un archivo, tengamos prácticamente garantizado que corresponde a un filtro.

Nuestros programas tendrán que ser capaces de leer dos tipos de ficheros:

1. Fichero de texto. Este formato estará compuesto por:

- Una cadena “mágica”. La cadena se usa para distinguir este archivo de otros tipos. En este caso la cadena está compuesta por los siguientes 11 caracteres: *MP-FILTRO-T*.
- Un separador. Es decir, un carácter “blanco”², normalmente un espacio.
- Un comentario que se inicia con el carácter #. Puede aparecer o no. Si está, ocupa una línea.
- Dos enteros, en texto, que indican el número de filas y columnas de nuestro filtro, separados por un carácter “blanco”.
- Un separador: un carácter “blanco”, normalmente un salto de línea.
- Tantas filas como indica el número de filas y, por cada fila, tantos valores reales como indica el número de columnas separados por un separador (normalmente un tabulador).

2. Fichero binario. Este formato estará compuesto por:

- Una cadena “mágica”. La cadena se usa para distinguir este archivo de otros tipos. En este caso la cadena está compuesta por los siguientes 11 caracteres: *MP-FILTRO-B*.
- Un separador. Es decir, un carácter “blanco”, normalmente un espacio.
- Un comentario que se inicia con el carácter #. Puede aparecer o no. Si está, ocupa una línea.
- Dos enteros, en texto, que indican el número de filas y columnas de nuestro filtro separados por un carácter “blanco”.
- Un separador: un carácter “blanco”, normalmente un salto de línea.
- Tantos datos de tipo double como sean necesarios para representar el filtro. Estarán almacenados en formato binario, sin separación entre ellos. En total leeremos **número filas × número columnas** valores double.

En el material asociado a la práctica, y que puede bajar desde la página web de la asignatura, podrá encontrar archivos con extensión “*fil*” (en el directorio datos) con ejemplos de ambos formatos. Para que pueda localizarlos más fácilmente, se ha añadido “*bin*” o “*txt*” al nombre para distinguir a qué formato corresponden. Lógicamente, el nombre del archivo puede ser cualquiera, sin estas letras, o incluso con otra extensión, ya que los programas reconocerán el filtro por la cadena mágica inicial.

3.3. Archivos y selección de la representación

Para desarrollar los distintos programas podríamos generar los siguientes archivos:

- E/S de imágenes: “*imagenES.h*”, “*imagenES.cpp*”. Contienen las funciones de E/S para leer y escribir imágenes en disco. Estos dos archivos se pueden descargar de la página web de la asignatura.

² Un carácter “blanco” puede ser cualquiera de los siguientes caracteres: espacio, tabulador o salto de línea.

- Módulo *Imagen*: “*imagen.h*”, “*imagen.cpp*”. Este módulo implementa el nuevo tipo *Imagen* y las operaciones asociadas.
- Módulo *Signal*: “*signal.h*”, “*signal.cpp*”. Este módulo implementa el nuevo tipo *Signal* que se usará para mantener en memoria tanto un filtro como una *Imagen* que necesita un rango de valores reales para su representación.
- Módulo *Procesar*: “*procesar.h*”, “*procesar.cpp*”. Este módulo implementa los algoritmos necesarios para ocultar/revelar un mensaje y resaltar un trozo de la señal.
- Módulo *Conversiones*: “*conversiones.h*”, “*conversiones.cpp*”. En este módulo se implementan las funciones para convertir un objeto de tipo *Signal* a uno de tipo *Imagen* (*SignalToImagen*) o viceversa (*ImagenToSignal*).
- Módulo *Correlacion*: “*correlacion.h*”, “*correlacion.cpp*”. Este módulo implementa el algoritmo de correlación.
- Programas: “*ocultar.cpp*”, “*revelar.cpp*”, “*filtrar.cpp*”, “*construir filtro.cpp*” y “*buscar_Wally.cpp*”. Estos archivos contendrán las funciones *main* que implementan los programas, con posibles funciones auxiliares si se consideran necesarias.

Crearemos una biblioteca con todos los módulos que manejan imágenes excepto los que incluyen la función *main*. Es decir, crearemos una biblioteca “*libimagen.a*”, que contendría los archivos “*imagenES.o*”, “*imagen.o*” y “*procesar.o*”, con la que enlazar cualquiera de los ejecutables.

De igual manera crearemos la biblioteca “*libsignal.a*” con los módulos que manejan señales, “*signal.o*”, “*conversiones.o*” y “*correlacion.o*”.

4. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre “*imagen.tgz*” y entregarlo antes de la fecha que se publicará en la página web de la asignatura. Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta *datos*. Es recomendable que haga una “limpieza” para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

Para simplificarlo, el alumno debe ampliar el archivo *Makefile* para que también se incluyan las reglas necesarias que generen los ejecutables correspondientes. Tenga en cuenta que los archivos deben estar distribuidos en directorios:

imagen	— include	Ficheros de cabecera (.h)
	— src	Código fuente (.cpp)
	— obj	Código objeto (.o)
	— lib	Bibliotecas
	— doc	Documentación
	— bin	Ficheros ejecutables
	— datos	Imágenes y filtros

Por consiguiente, lo más sencillo es que comience con la estructura de directorios y archivos que ha descargado desde la página y añada lo necesario para completar el proyecto.

Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta “*imagen*”) para ejecutar:

```
prompt% tar zcvf imagen.tgz imagen
```

tras lo cual, dispondrá de un nuevo archivo *imagen.tgz* que contiene la carpeta *imagen*, así como todas las carpetas y archivos que cuelgan de ella. Se debe eliminar la carpeta *datos*

para hacer la entrega.

5. Referencias

- [GAR06a] Garrido, A. *“Fundamentos de programación en C++”*. Delta publicaciones, 2006.
- [GAR06b] Garrido, A. Fdez-Valdivia, J. *“Abstracción y estructuras de datos en C++”*. Delta publicaciones, 2006.
- [MP2012a] Garrido, A., Martínez-Baena, J. *“Mensajes e imágenes”*. Guión de ejercicio de la asignatura “Metodología de la Programación”, curso 2012/2013.
- [MP2012b] Garrido, A., Martínez-Baena, J. *“Tipos de datos abstractos: imágenes”*. Guión de ejercicio de la asignatura “Metodología de la Programación”, curso 2012/2013.