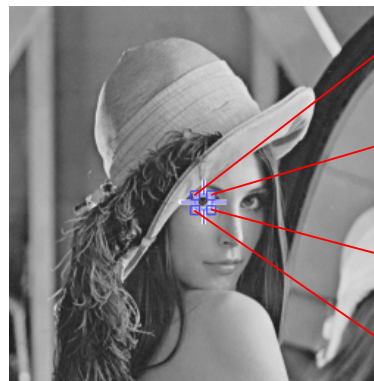




# Metodología de la Programación

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

*Imaging0*

*Acceso a memoria a nivel de bit*

*Febrero de 2022*



# Índice

<b>1. Las prácticas del curso, una visión general</b>	<b>5</b>
<b>2. Arquitectura de las prácticas</b>	<b>5</b>
<b>3. Objetivos</b>	<b>7</b>
<b>4. Imaging0</b>	<b>7</b>
<b>5. Operadores a nivel de bit</b>	<b>7</b>
<b>6. Módulo Byte</b>	<b>8</b>
<b>7. Imaging0, práctica a entregar</b>	<b>11</b>
7.1. Ejemplo de ejecución . . . . .	11
<b>8. TESTS DOCUMENTATION FOR PROJECT Imaging0</b>	<b>13</b>
8.1. _01_Basics . . . . .	13
8.1.1. UnitByte_onBit . . . . .	13
8.1.2. UnitByte_offBit . . . . .	13
8.1.3. UnitByte_getBit . . . . .	13
8.1.4. UnitByte_printByte . . . . .	13
8.1.5. UnitByte_shiftRByte . . . . .	13
8.1.6. UnitByte_shiftLByte . . . . .	13
8.1.7. UnitImage_Constructors . . . . .	13
8.1.8. UnitImage_Width . . . . .	14
8.1.9. UnitImage_Height . . . . .	14
8.1.10. INTEGRATION_Byte . . . . .	14
8.1.11. P1 . . . . .	14
8.2. _02_Intermediate . . . . .	14
8.2.1. UnitByte_onByte . . . . .	14
8.2.2. UnitByte_offByte . . . . .	14
8.3. _03_Advanced . . . . .	14
8.3.1. UnitByte_encodeByte . . . . .	14
8.3.2. UnitByte_decodeByte . . . . .	14
8.3.3. UnitByte_decomposeByte . . . . .	15
8.4. Tests run . . . . .	15
<b>Introducción a la metodología Test Driven Development (TDD)</b>	<b>15</b>
<b>9. Configuración de las prácticas</b>	<b>17</b>
9.1. El espacio de trabajo . . . . .	17
<b>10. Configurar el proyecto en NetBeans</b>	<b>19</b>



## 1. Las prácticas del curso, una visión general

Todas las prácticas que se van a desarrollar en este curso tienen como objeto principal imágenes en blanco y negro.

Una imagen digital en blanco y negro, como las que figuran en la portada (de la original [\(Abrir →\)](#) un conocido ejemplo de procesamiento digital de imágenes..), puede verse como una matriz de puntos, llamados píxeles, determinada por sus dimensiones (altura x anchura). Por el hecho de ser en blanco y negro, cada píxel se puede representar mediante un byte, el rango de valores que este puede tomar es el intervalo [0,255]). Una ilustración puede verse en la imagen derecha de la portada.

De forma general, una imagen se almacena en fichero en un determinado formato, reconocible por su extensión, que contiene además de la matriz, unos metadatos o información interna para su visualización, así los hay **.jpg**, **.png**, **.gif** etc. Para visualizar imágenes de fotos, dibujos, logos, creadas o transformadas es necesario un programa de visualización como *ImageMagick* ([\(Abrir →\)](#)) o *gimp* ([\(Abrir →\)](#)) entre otros muchos<sup>1</sup>, ambos disponibles para linux y windows. Se recomienda la instalación de *ImageMagick* para las prácticas. En el anexo I puede encontrar algunos detalles sobre la configuración recomendada.

A lo largo de las prácticas vamos a reconocer, leer y escribir imágenes en formato PGM (**.pgm**), sobre las que vamos a realizar algunas transformaciones comunes en cualquier editor de imágenes, avanzando hacia otras operaciones más complejas que trabajan con varias imágenes. Durante el desarrollo de las prácticas se van a elaborar estructuras y métodos necesarios para la realización de una atractiva práctica final que consiste en la implementación de un proceso de *esteganografía* que nos permitirá ocultar una imagen y/o texto, dentro de una imagen portadora sin que la alteración sea perceptible (hay preparado un reto final, véase



[\(Abrir →\)](#) Recuerde que hay que ser usuario de la ugr, para acceder al vídeo).

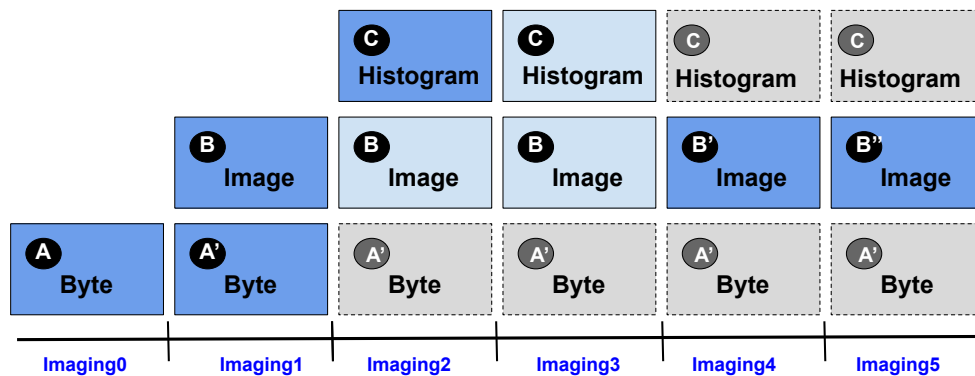
## 2. Arquitectura de las prácticas

La práctica *Imaging* se ha diseñado por etapas, las primeras contienen estructuras más sencillas, sobre las cuales se asientan otras estructuras más complejas y se van completando nuevas funcionalidades. La Figura 1 muestra el diseño de la arquitectura que se va a emplear y que se va a ir desarrollando progresivamente en esta y las siguientes sesiones de prácticas. Indicar aquí, que las estructuras de datos que se van emplear van a sufrir varias modificaciones y van evolucionar conforme se van adquiriendo nuevos conocimientos (los impartidos en teoría que se van a aplicar) o bien conforme llegan nuevas especificaciones (nuevos problemas para resolver), esta situación no es excepcional sino que, la

---

<sup>1</sup>La mayoría son además editores de imágenes, pero sólo nos interesa su utilidad como visualizadores.

modificación y evolución de clases es un proceso habitual en cualquier desarrollo de software.



*Figura 1: Arquitectura de las prácticas de MP 2022. Los cambios esenciales en las clases (cambio en estructura) se muestran en azul intenso, los que incorporan nuevas funcionalidades en azul tenue. En gris se muestran las clases que no sufren cambios en la evolución de las prácticas.*

#### **A** Byte.cpp

Tipo de dato, definido sin clase por razones pedagógicas, para el que se van a desarrollar una serie de funciones de bajo nivel.

#### **A'** Byte.cpp

Implementa la clase Byte, incorporando las funciones anteriores como métodos.

#### **B** Image.cpp

Implementa la clase Image que contiene la información necesaria para su visualización. En una primera aproximación se usará un vector estático.

#### **C** Histogram.cpp

Implementa una clase Histograma, una estructura para almacenar frecuencias de valores, en nuestro caso se usará para el conteo de frecuencias de los tonos de una imagen.

#### **B'** Image.cpp

Manteniendo la interfaz previa, se cambia la estructura de la clase para alojar el vector de bytes de forma más eficiente en memoria dinámica.



## B" Image.cpp

Manteniendo la interfaz previa, se cambia la estructura de la clase para alojar una matriz en memoria dinámica .

Este trabajo progresivo, con la incorporación de nuevos métodos, se ha planificado en hitos sucesivos con entregas en Prado.

## 3. Objetivos

El desarrollo de la práctica Imaging0 persigue los siguientes objetivos:

- repasar conceptos básicos de funciones,
- practicar el paso de parámetros por referencia,
- practicar el paso de parámetros de tipo array,
- entender el uso de operaciones a nivel de bit,
- reforzar la comprensión de los conceptos de compilación separada.

## 4. Imaging0

De momento, antes de abordar ninguna imagen, vamos a trabajar a más bajo nivel, así en esta práctica aprenderemos a acceder a cada byte de memoria, bit a bit. Para ello, debemos implementar ciertas operaciones básicas con los bits como activar/desactivar bits determinados, o activar/desactivar todos los bits simultáneamente.

Supondremos que cada bit puede estar sólo en dos estados, encendido (1) y apagado (0), y los 8 bits se representan como un byte. En C++, las variables de tipo `unsigned char` ocupan exactamente un byte, por lo que, el bloque de bits se representará como una variable de dicho tipo. Para facilitar su uso y abstraernos de la representación interna, podemos utilizar un “alias” para `unsigned char` y hacer:

```
typedef unsigned char Byte;
```

y de esta manera definir variables de tipo `Byte`.

## 5. Operadores a nivel de bit

Las funciones de manejo de bits concretos (encendido, apagado, consulta de estado, etc.) necesitarán acceder y modificar posiciones específicas de la variable de tipo `Byte`. Es decir, tendremos que manipular los bits de la variable de manera independiente.

El lenguaje C++ ofrece un conjunto de operadores lógicos **a nivel de bit** para operar con diversos tipos de datos, en particular con **caracteres** y **enteros**. Los operadores son: `&`, `|`, `^`, `>>`, `<<` y `~` y se clasifican como:

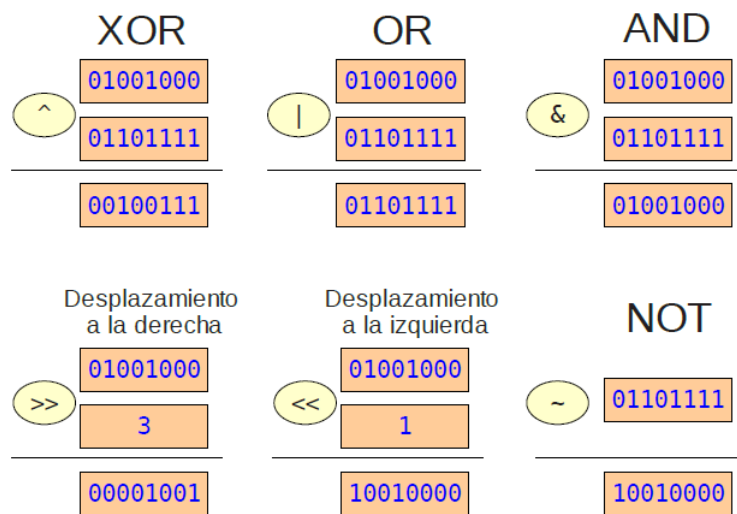


Figura 2: Ejemplos de operaciones a nivel de bit

- operadores binarios, donde la operación afecta a los bits de dos operandos (de tipo `unsigned char`, en nuestro caso):
  - `and (op1 & op2)`: devuelve 1 si los dos bits valen 1
  - `or (op1 | op2)`: devuelve 1 cuando al menos 1 de los bits vale 1
  - `or-exclusivo (op1 ^ op2)`: produce un valor 1 en aquellos bits en que sólo 1 de los bits de los operandos es 1
  - `desplazamiento a la derecha un determinado número de bits (posiciones) (op1 >> desp)`: Los `desp` bits más a la derecha se perderán, al tiempo que se insertan bits con valor 0 por la izquierda
  - `desplazamiento a la izquierda (op1 << desp)`: ahora los bits que se pierden son los de la izquierda y se introducen bits a 0 por la derecha
- operador unario de negación (`~op1`): cambia los bits de `op1` de forma que aparecerá un 0 donde había un 1 y viceversa.

La Fig. 2 muestra un ejemplo de dichas operaciones.

## 6. Módulo Byte

El módulo a implementar contendrá las siguientes funciones (archivo `Byte.h` incluido en la carpeta `include`).

```

1  /**
2  @file Byte.h
3  @brief Operators for bit level
4  @note To be implemented by students
5  @author MP-DGIM, MP-IADE, MP-II (grupo B)
6  */
7
8  #ifndef _BYTE_H_
9  #define _BYTE_H_
10

```





```
11 typedef unsigned char Byte; ///< A @c Byte contains the state of 8 Bits
12
13 #define MAX.BYTE 256 ///< Max number of values allowed
14 #define MAX.BYTE.VALUE 255 ///< Highest value allowed
15 #define MIN.BYTE.VALUE 0 ///< Lowest value allowed
16 #define NUM.BITS 8 ///< Number of bits
17 #define MIN.BIT 0 ///< Rightmost bit
18 #define MAX.BIT 8 ///< Leftmost bit
19 /**
20  * @brief Turns on the @p position bit of @c Byte @p b, whenever the value of pos is in [0..7] otherwise
21  * Byte remains unchanged.
22  * @param b the @c Byte, output parameter
23  * @param pos the position @p b of the bit within the Byte (0 means the rightmost position)
24  */
25 void onBit(Byte &b, int pos);
26
27 /**
28  * @brief Turns off the @p position bit of @c Byte @p b, , whenever the value of pos is in [0..7] otherwise
29  * Byte remains unchanged.
30  * @param b the @c Byte, output parameter
31  * @param pos the position @p b of the bit within the Byte (0 means the rightmost position)
32  */
33 void offBit(Byte &b, int pos);
34
35 /**
36  * @brief It returns the state of a certain bit inside a Byte (on = true, off = false) at the position @p pos
37  * @param b the @c Byte
38  * @param pos the position @p b of the bit within the Byte (0 means the rightmost position)
39  * @retval true when the bit at the @p pos position is 1
40  * @retval false when the bit at the @p pos position is 0
41  */
42 bool getBit(Byte b, int pos);
43
44 /**
45  * @brief It returns a string with a sequence of 0 and 1 corresponding to the state of each bit. For instance,
46  * the value 5 is represented as "00000101" since the bits 0 and 2 are set to 1
47  */
48 std::string to_string(Byte b);
49
50 /**
51  * @brief Turns all the bytes on
52  * @param b the @c Byte
53  */
54 void onByte(Byte &b);
55
56 /**
57  * @brief Turns all the bytes off
58  * @param b the @c Byte
59  */
60 void offByte(Byte &b);
61
62 /**
63  * @brief Turns on just the bits contained in the vector. That is, if vector[i] is true,
64  * then the bit at the position @a i is turned on.
65  * @param b the @c Byte
66  * @param v A vector of 8 elements representing which bits are to be turned on. b output parameter
67  */
68 void encodeByte(Byte &b, const bool v[]);
69
70 /**
71  * @brief It dumps the bits in the Byte into the vector, such as, if bit @a i is 1, then vector[i] is true
72  * @param b the @c Byte
73  * @param v A vector of 8 elements representing which bits are turned on. v output parameter
74  */
75 void decodeByte(Byte b, bool v[]);
76
77 /**
78  * @brief It returns a vector of int, such that if vector[i]=k, then the bit at the position @a k is 1
79  * @param b the @c Byte
80  * @param posits An array of positions that contains a bit set to 1, output parameter.
81  * @param n Number of bits 1 in the byte. output parameter.
82  */
83 void decomposeByte(Byte b, int posits[], int &n);
84
85 /**
86  * @brief Shifts the byte @a n bits to the right, overflowing the rightmost bits
87  * @param b The byte
88  * @param n The number of shifts
89  */
90 void shiftRByte(Byte &b, int n);
91
92 /**
93  * @brief Shifts the byte @a n bits to the left, overflowing the leftmost bits
94  * @param b The byte
95  * @param n The number of shifts
96  */
97 void shiftLByte(Byte &b, int n);
98
99 /**
100  * @brief It merges the byte @a merge into the byte @a carrier following
101  * a linear combination carrier = (carrier *(100-percentage)+merge*percentage)/100
102  * @param carrier The byte to absorb the merge
103  * @param merge The byte to merge
104  * @param percentage A value [0,100]
105  */
106 void mergeByte(Byte &carrier, Byte merge, int percentage);
107
108 #endif
109
```

## Cómo consultar y modificar el estado de un bit

Las operaciones de consulta, apagado y encendido de bits se traducen a operaciones a nivel de bits. La consulta se corresponde con una lectura y el encendido/apagado con una operación de asignación.

Las operaciones de asignación y lectura de bits se pueden dividir en dos pasos: a) generar una “máscara” (un byte con una secuencia determinada de 0’s y 1’s) y b) aplicar un operador lógico.

Por convención, asumiremos que el bit más a la derecha representa el primer bit y tiene asignada la posición 0, mientras que el bit de más a la izquierda ocupa la posición 7.

### Consulta del estado de un bit

Supongamos que queremos averiguar si el bit en la posición 5 se encuentra encendido. Esto es equivalente a averiguar si el bit en la posición 5 del bloque de bits `b` está en 1. Para ello, debemos seguir los siguientes pasos.

1. Crear una máscara (un byte específico) que contenga sólo un 1 en la posición de interés. En este caso: **0010 0000**. Para ello:
  - a) Partimos del valor decimal **1** (su codificación en binario es **0000 0001**)<sup>2</sup>
  - b) lo desplazamos a la izquierda el número de posiciones deseadas (en este caso 5).
2. hacer una operación **AND** entre `b` y `mask`.
3. Si el resultado es distinto de cero, entonces el bit 5 es un 1 y por tanto, el bit está encendido. Caso contrario es un cero y el bit está apagado.

### Apagar y Encender un bit

**Apagar un bit** implica poner a 0 el bit correspondiente. Supongamos que deseamos apagar el bit de la posición 2. Para poner el bit `k=2` del bloque de bits `b` a cero, se deben seguir los siguientes pasos.

1. generar la máscara `mask` con valor `1111 1011` Para ello:
  - a) generar primero una máscara `0000 0100` como hemos explicado antes.
  - b) aplicarle el operador de negación **NOT**.
2. hacer un **AND** entre `mask` y `b` y guardar el resultado en `b`.

---

<sup>2</sup>En C++ se pueden escribir literales en hexadecimal precediéndolos por `0x`, p. ej. 32 decimal es `0x20`, o en octal precediéndolos por `0`, p. ej. `040`. Desde C++14, los literales binarios pueden representarse precediéndolos con `0b`, p. ej. `0b00100000`



**Encender un bit** implica poner a 1 el bit correspondiente. Por ejemplo, para poner el bit  $k=2$  a 1 del bloque de bits  $b$  haremos:

1. generar la máscara `mask` con valor 0000 0100.
2. aplicar el operador OR entre `mask` y la variable `b`. El resultado se guarda en `b`.

## Secuencias de Animación

El bloque de bits puede mostrar una “animación” si se encienden y apagan los bits en un orden determinado y se muestran sus valores. A continuación se muestran dos ejemplos.

Ejemplo 1	Ejemplo 2
11111111	11111111
01111111	01111110
10111111	00111100
11011111	00011000
11101111	00000000
11110111	00011000
11111011	00111100
11111101	01111110
11111110	11111111

## 7. Imaging0, práctica a entregar

- Implementar las funciones indicadas en el fichero **Byte.h**
- El módulo **main.cpp** ya incluye instrucciones para probar la funcionalidad básica del Byte. Extienda el módulo para generar las secuencias de animación indicadas en la sección anterior.

### 7.1. Ejemplo de ejecución

```
byte apagado bits:
00000000

Inicializo el byte a partir de un vector de bool
00000101

Ahora enciendo los bits 0, 1 y 2 con la funcion on
10000101
11000101
11100101

Los bits encendidos estan en las posiciones:
0,1,2,5,7,

Todos encendidos:
11111111
Todos apagados:
00000000
```



Ejemplo 1

```
11111111
11111110
11111101
11111011
11110111
11101111
11011111
10111111
01111111
```

Ahora la animacion

Ejemplo 2

```
11111111
01111110
00111100
00011000
00000000
00011000
00111100
01111110
11111111
```

RUN FINISHED; exit value 0; real time: 0ms; user: 0ms; system: 0ms



## 8. TESTS DOCUMENTATION FOR PROJECT Imaging0

### 8.1. \_01\_Basics

#### 8.1.1. 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 2

#### 8.1.2. 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

#### 8.1.3. UnitByte\_getBit

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

#### 8.1.4. UnitByte\_printByte

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

#### 8.1.5. 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

#### 8.1.6. 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

#### 8.1.7. UnitImage\_Constructors

1. A newly created instance of Image should have 0 columns, and 0 rows.
2. A newly created instance of a 10x10 Image should have 10 columns, and 10 rows.



#### 8.1.8. UnitImage\_Width

1. A newly created instance of Image should have 0 columns, and 0 rows.
2. A newly created instance of a 10x5 Image should have 10 columns, and 5 rows.

#### 8.1.9. UnitImage\_Height

1. A newly created instance of Image should have 0 columns, and 0 rows.
2. A newly created instance of a 10x5 Image should have 10 columns, and 5 rows.

#### 8.1.10. INTEGRATION\_Byte

1. The output of the main program must match that of the assignment

#### 8.1.11. P1

1. The output of the main program must match that of the assignment

### 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

### 8.3. \_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

## 8.4. Tests run

```
[=====] Running 11 tests from 3 test suites.
[-----] Global test environment set-up.
[-----] 6 tests from _01_Basics
[ 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_printByte
[       OK ] _01_Basics.UnitByte_printByte (1 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)
[-----] 6 tests from _01_Basics (8 ms total)

[-----] 2 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)
[-----] 2 tests from _02_Intermediate (1 ms total)

[-----] 3 tests from _03_Advanced
[ RUN      ] _03_Advanced.UnitByte_encodeByte
[       OK ] _03_Advanced.UnitByte_encodeByte (0 ms)
[ RUN      ] _03_Advanced.UnitByte_decodeByte
[       OK ] _03_Advanced.UnitByte_decodeByte (2 ms)
[ RUN      ] _03_Advanced.UnitByte_decomposeByte
[       OK ] _03_Advanced.UnitByte_decomposeByte (1 ms)
[-----] 3 tests from _03_Advanced (3 ms total)

[-----] Global test environment tear-down
[=====] 11 tests from 3 test suites ran. (12 ms total)
[ PASSED ] 11 tests.
```

## Introducción a la metodología Test Driven Development (TDD)

TDD es una metodología de desarrollo de aplicaciones que se basa en escribir, en primer lugar, los tests que debe de pasar correctamente el software que se va a crear y, posteriormente, crear ese software con el objetivo fundamental de que pase todos los tests para, posteriormente, adaptarlo a un diseño y/o arquitectura específica. Eso sí, sin dejar de pasar nunca todos los tests. Quizás los dos objetivos fundamentales de TDD sean buscar el diseño más sencillo posible, evitando toda complejidad innecesaria (KISS principle<sup>3</sup>) y, por otro lado, proporcionar confianza en el desarrollo de software.

<sup>3</sup>KISS = Keep It Simple Stupid, ([Abrir →](#))

Hay varios tipos de tests, que varían según la aplicación que se esté desarrollando, entre los que podemos distinguir los más básicos:

- Tests Unitarios<sup>4</sup>. Cuyo objetivo es comprobar que cada unidad funcional mínima que se construya, ya sea una función o un método, cumple una serie de buenas propiedades. Así, podemos considerar que la implementación de una clase es correcta si todos sus métodos han sido probados individualmente, lo cual ofrece una base de trabajo garantizada desde la que desarrollar nuevas clases o construir el programa completo.
- Tests de Integración<sup>5</sup>. Cuyo objetivo es comprobar que la integración de todas las clases construidas, la entrada y salida de datos, sigue siendo correcta, partiendo de la base de que las clases se han probado correctas con tests unitarios.

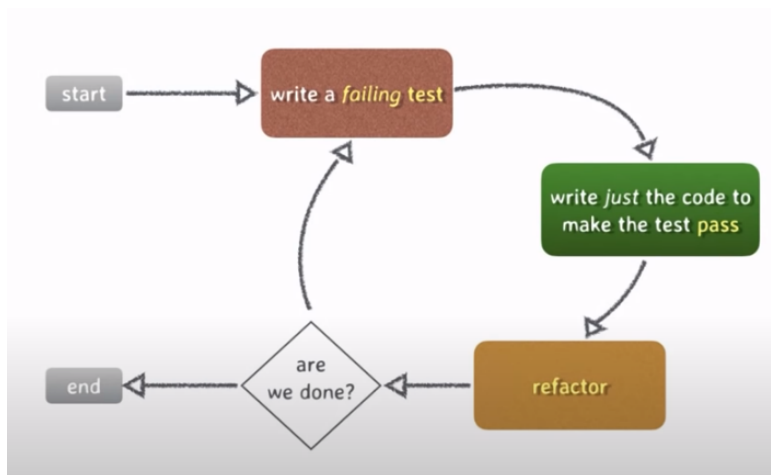


Figura 3: Ciclo de vida de TDD. Phil Nash, CppCon The C++ Conference, 2020. ([Abrir →](#))

<sup>4</sup>Unit Testing ([Abrir →](#))

<sup>5</sup>Integration Testing ([Abrir →](#))





Los pasos a seguir en la metodología TDD, que no son muy diferentes del proceso tradicional, solo que están más ordenados (ver Figura 3), son los siguientes (en negrilla los que debe de hacer el estudiante, en tipografía normal los pasos que aporta el profesor).

1. Escribir el diseño de la API de la clase, solo los métodos.
2. Escribir los tests unitarios y de integración.
3. **Escribir el código de los métodos hasta que pasen todos los tests unitarios, sea como sea.**
4. **Cuando pasen todos los tests, intentar refactorizar el código para mejorarlo, en cuyo caso habría que volver al paso 3 para testear los cambios hechos.**
5. **Escribir el main() para que pase los tests de integración, sea como sea.**
6. **Es posible que haya que refactorizar alguna clase previa. Si es así, volver al paso 3.**
7. **Si ha pasado todos los tests de integración, intentar refactorizar el código para mejorarlo y volver al paso 5.**
8. **Compilar la versión definitiva.**

## 9. Configuración de las prácticas

### 9.1. El espacio de trabajo

Partimos de **MPTools.zip** , material inicialmente entregado a los estudiantes en Prado, organizado en carpetas. Contamos con que lo descomprimas en **ProyectosNetBeans** : Carpeta raíz en la que se van a crear todos los proyectos de la asignatura. A continuación se muestra la estructura de directorios después de descomprimir el archivo en su lugar correcto.

```
ProyectosNetbeans
├── MPGeometry*
├── MPTools
│   ├── build
│   ├── dist
│   ├── include
│   ├── src
│   └── *
├── Imaging0
├── Imaging1
├── *
└── Scripts
```

**MPTools:** Librería donde se encuentran ya programadas ciertas funciones que van a facilitar el desarrollo de las prácticas.

**Scripts:** Una serie de scripts Bash de apoyo a las funciones de NetBeans.

```

Imaging
├── build
├── dist
│   ├── Debug
│   └── Release
├── data
├── doc
│   └── doc.doxy
├── include
├── Makefile
├── nbproject
├── scripts
├── src
├── tests
└── zip

```

Para tener bien ordenado el contenido de cada proyecto Imaging, se va a organizar en las siguientes carpetas:

**build** Es una carpeta temporal que contiene código precompilado y pendiente de enlazar

**dist** Contiene los binarios ejecutables. Vamos a generar dos versiones por cada programa:

**Release** Es la versión más eficiente y que menos espacio ocupa. Es la que debería entregarse al cliente.

**Debug** Es la versión sobre la que se depuran errores y se corrigen defectos. Es más grande por-

que el binario contiene información extra para poder usar el depurador. Puede llegar a ser el doble de grande que la anterior.

**data** Ficheros de datos, bases de datos, ficheros de configuración que pueda necesitar el programa durante su ejecución, que no sean para testearla.

**doc** Aquí se almacena la documentación del proyecto, tanto del código, como la generada por los tests.

**include** Contiene los ficheros de cabeceras **.h**.

**nbproject** Es la carpeta que utiliza NetBeans para almacenar la configuración del proyecto

**scripts** En esta carpeta colocaremos los scripts de apoyo a Netbeans que se han desarrollado en esta asignatura y se puede ampliar con otros más.

**src** Contiene los principales ficheros fuente del proyecto **.cpp**, sin incluir los ficheros de test, que también son **.cpp**, que van en la siguiente carpeta






Figura 4: Estructura interna de cada proyecto Imaging

**tests** Carpeta dedicada a los tests unitarios y de integración. Dentro de ella se encuentran, en el primer nivel, los ficheros fuente de los tests **.cpp**, la carpeta **output** que se utiliza como carpeta temporal durante los tests y la carpeta **validation** que se utiliza para **ficheros de datos únicamente usados en la validación de la aplicación** completamente integrada.

**zip** Carpeta para copias de seguridad del proyecto.

## 10. Configurar el proyecto en NetBeans

Se recomienda repasar primero los videotutoriales completos sobre este tema en Google Drive

- Introducción a TDD  (Abrir →)
- Creando el proyecto  (Abrir →)
- Primeros tests  (Abrir →)
- Resultado de los tests  (Abrir →)
- Tests de integración  (Abrir →)

### 1. Crear el proyecto en NetBeans como C++ Application.

- a) Eliminar el entorno de trabajo Release y dejar sólo Debug. Aunque en adelante se crearán otros entornos de trabajo, en lo siguiente se trabajará en Debug.

### 2. Copiar la carpeta `./scripts`, copiar el script `runUpdate.sh` dentro y ejecutarlo<sup>6</sup> Esto crea la estructura de carpetas descrita anteriormente y actualiza los scripts que pudiese haber nuevos. Estos cambios se han llevado a cabo através del SO, compruébalo con la instrucción unix: `ls`, sin embargo no aparecen en Netbeans... Para que se refleje en el IDE, desplegar opción de menú `Source -> Scan for external changes` y ya aparecen las nuevas carpetas...

### 3. Colocar cada fichero en su sitio, según indica la imagen anterior.

### 4. Propiedades de proyecto. Compilador

- a) Poner el estándar a C++14

---

<sup>6</sup>Comprueba que tenga permiso de ejecución, se cambia con la instrucción unix: `chmod +x`, en la terminal.



- b) Añadir los Include Directories ... siguientes:  
del propio proyecto **./include**  
y el de la librería MPTools **../MPTools/include**  
las de testeo **../MPTools/googletest-master/googletest/include,**  
**../googletest-master/googletest**  
.
- 5. Desde la vista lógica del proyecto.
  - a) En Header Files, Add existing item el fichero `Byte.h`.
  - b) En Source Files, Add existing item añadir los ficheros `main.cpp` y `Byte.cpp`.
- 6. Sobre el nombre del proyecto, botón derecho Set As Main Project o desde el menú principal Run – Set Main Project y seleccionar este proyecto.
- 7. Con esto se puede ejecutar el proyecto normalmente. Obviamente, no hace nada porque está vacío y no se llama a ninguna función. A partir de aquí empezaría el desarrollo del proyecto, el cual cada programador seguirá un orden determinado y, probablemente, diferente a todos los demás, hasta que la aplicación esté terminada. En ese momento habrá que probarla, para ver que todo funciona como se espera, en un proceso que, de nuevo, dependerá de cada programador en concreto.

La práctica deberá ser entregada en Prado, en la fecha que se indica en cada entrega, y consistirá en un fichero ZIP del proyecto (para ello se sugiere utilizar el script `runZipProject.sh`).

## Anexo I. Detalle sobre la instalación de ImageMagick

Para la visualización de imágenes es necesario instalar, en modo administrador, el software ImageMagick con la selección de *legacy utilities* como se muestra en el la siguiente figura.

