Fundamentos de la programación

5 Tipos de datos estructurados

Grados en Ingeniería Informática, Ingeniería del Software e Ingeniería de Computadores

Ana Gil Luezas
(adaptadas del original de Luis Hernández Yáñez)

Facultad de Informática
Universidad Complutense

Índice

Tipos de datos

Tipos de datos

Clasificación de tipos

- ✓ Simples
  - Primitivos: `int`, `float`, `double`, `char`, `bool`
    Conjunto de valores predeterminado ✓
  - Definidos por el usuario: *enumerados*
    Conjunto de valores definido por el programador ✓
- ✓ Estructurados
  - Colecciones homogéneas: *arrays*
    Todos los elementos del mismo tipo ✓
  - Colecciones heterogéneas: *estructuras*
    Los elementos pueden ser de tipos distintos

Tipos estructurados

Colecciones o tipos aglomerados

Agrupaciones de datos:

- ✓ Todos del mismo tipo: *array* o *tabla*
- ✓ De tipos distintos: *estructura* o *registro*

Arrays (tablas)

- ➢ Elementos organizados por posición: 0, 1, 2, 3, ...
- ➢ Acceso por índice: 0, 1, 2, 3, ...
- ➢ Una o varias dimensiones

Estructuras (registros)

- ➢ Elementos (campos) sin orden establecido
- ➢ Acceso por nombre

Estructuras

Estructuras

Colecciones heterogéneas (registros)

Elementos de (posiblemente) distintos tipos: *campos*

Campos identificados por su nombre

Información relacionada que se puede manejar como una unidad

Acceso a cada elemento por su nombre de campo (operador `.`)

```
typedef struct {
    tipo1 nombre_de_campo1;
    tipo2 nombre_de_campo2;
    ...
} tTipo; // nombre de tipo al final
```

Tipos de estructuras

```
typedef struct {
    tipo1 nombre_de_campo1;
    tipo2 nombre_de_campo2;
    ... // declaraciones de campos (como variables)
} tTipo; // nombre de tipo al final
```

```
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
```

Campos:

Tipos estándar o previamente declarado

Variables de estructuras

Usamos el tipo para declarar variables:

```
tPersona persona;
```

Las variables de tipo `tPersona` contienen cuatro datos (campos):

`nombre` `apellidos` `edad` `nif`

Acceso a los campos con el operador punto (`.`):

```
persona.nombre // una cadena (string)
persona.apellidos // una cadena (string)
persona.edad   // un entero (int)
persona.nif // una cadena (string)
```

Podemos copiar dos estructuras directamente:

```
tPersona persona1, persona2;
...
persona2 = persona1;
```

Se copian como bloque o campo a campo?

Agrupación de datos heterogéneos

```
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
tPersona persona;
```

persona

nombre	Luis Antonio
apellidos	Hernández Yáñez
edad	22
nif	00223344F

Memoria

persona.nombre	Luis Antonio
persona.apellidos	Hernández Yáñez
persona.edad	22
persona.nif	00223344F

Acceso por nombre

```
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
tPersona persona;
```

Acceso directo por nombre de campo (operador `.`)

Con cada campo se puede hacer lo que permita su tipo

Las estructuras se pasan a los subprogramas por valor (sin &) o por referencia (con &)

Estructuras dentro de estructuras

```
typedef struct {
    int dni;
    char letra;
} tNif;
```

```
typedef struct {
    ...
    tNif nif;
} tPersona;
```

```
tPersona persona;
```

Acceso al NIF :

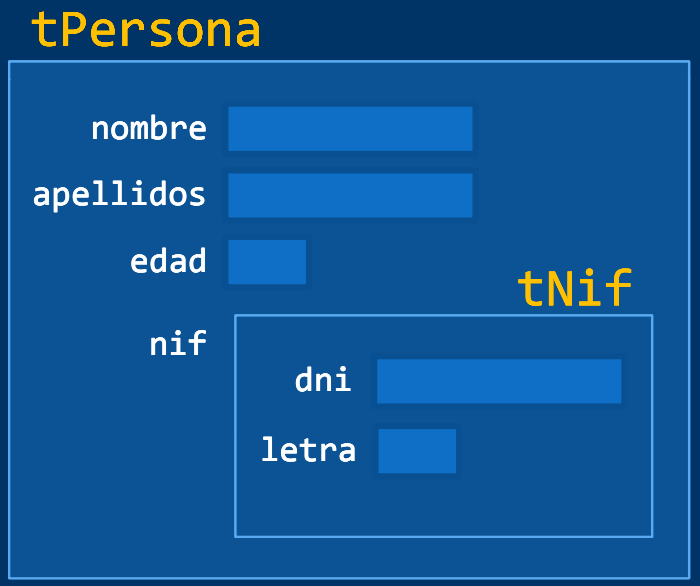
```
persona.nif // Otra estructura
```

Acceso a la letra del NIF:

```
persona.nif.letra
```

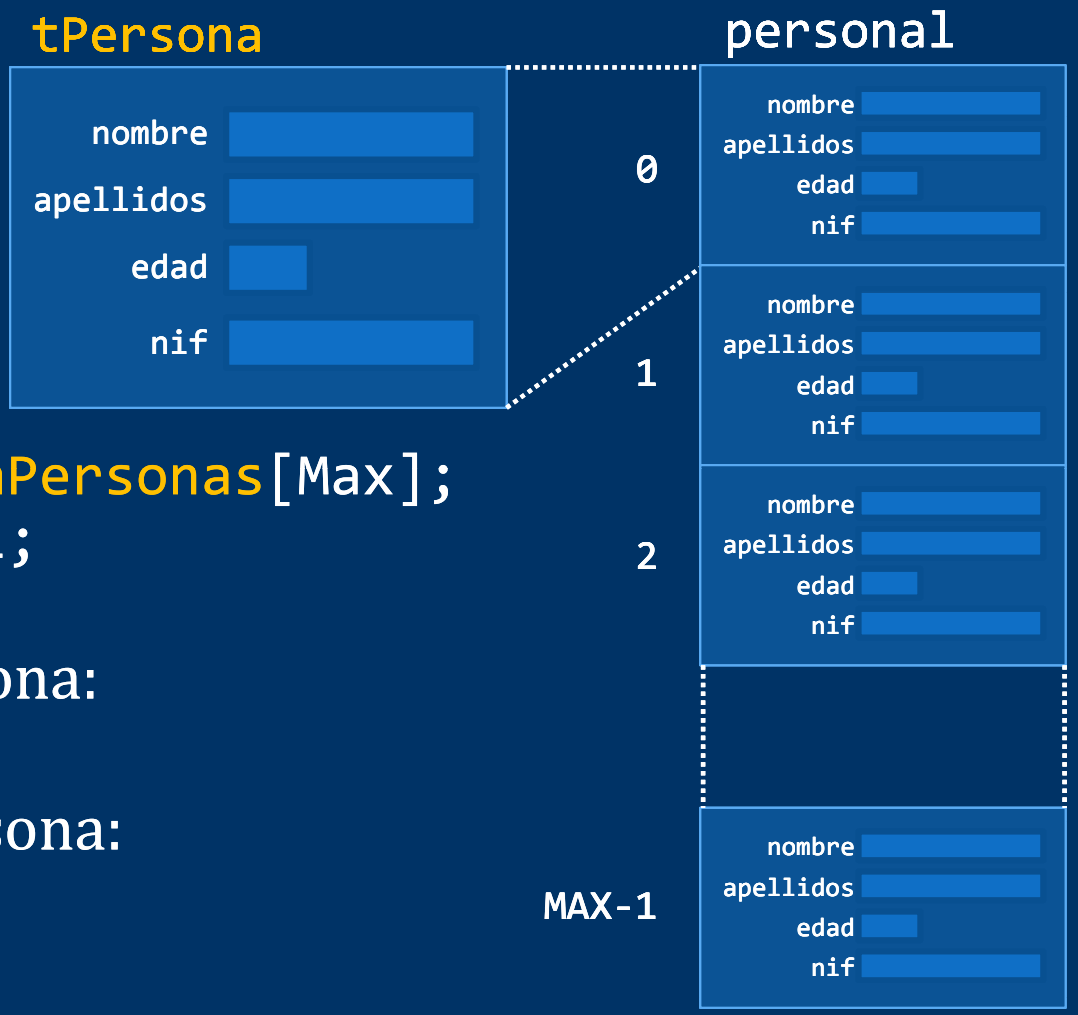
Acceso al DNI:

```
persona.nif.dni
```



Arrays de estructuras

```
const int Max = 100;
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
typedef tPersona tTablaPersonas[Max];
tTablaPersonas personal;
```



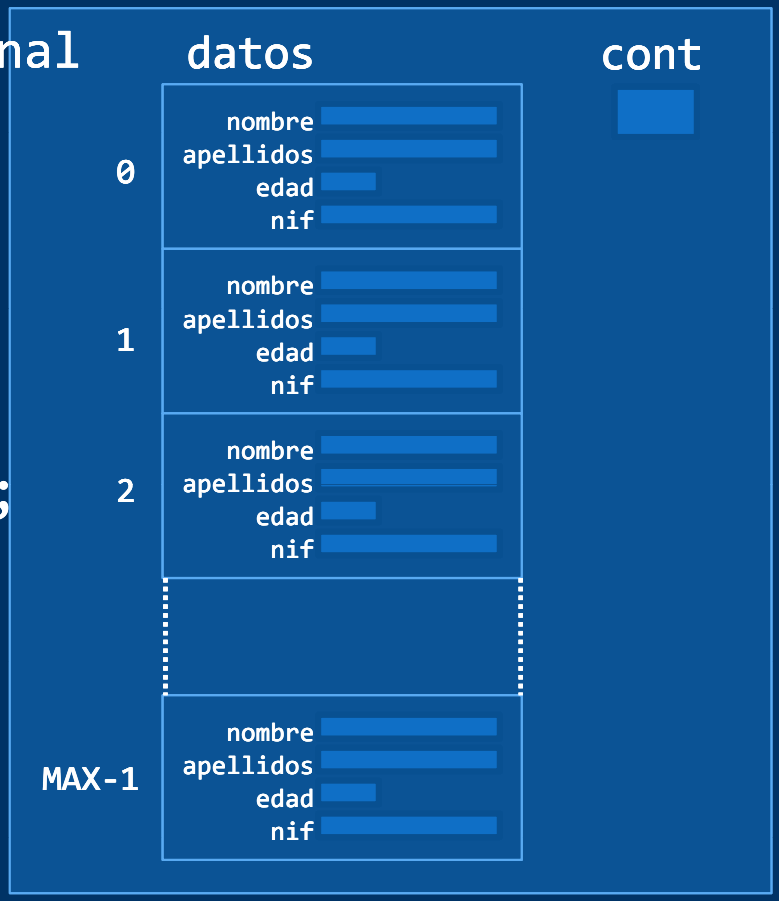
Nombre de la tercera persona:
`personal[2].nombre`

Edad de la duodécima persona:
`personal[11].edad`

NIF de la primera persona:
`personal[0].nif`

Arrays dentro de estructuras

```
const int MAX = 100;
typedef struct {
    string nombre;
    string apellidos;
    int edad;
    string nif;
} tPersona;
typedef tPersona tTablaPersonas[MAX];
typedef struct {
    tTablaPersonas datos;
    int contador;
} tListaPersonas;
tListaPersonas personal;
```



Nombre de la tercera persona: `personal.datos[2].nombre`

Edad de la duodécima persona: `personal.datos[11].edad`

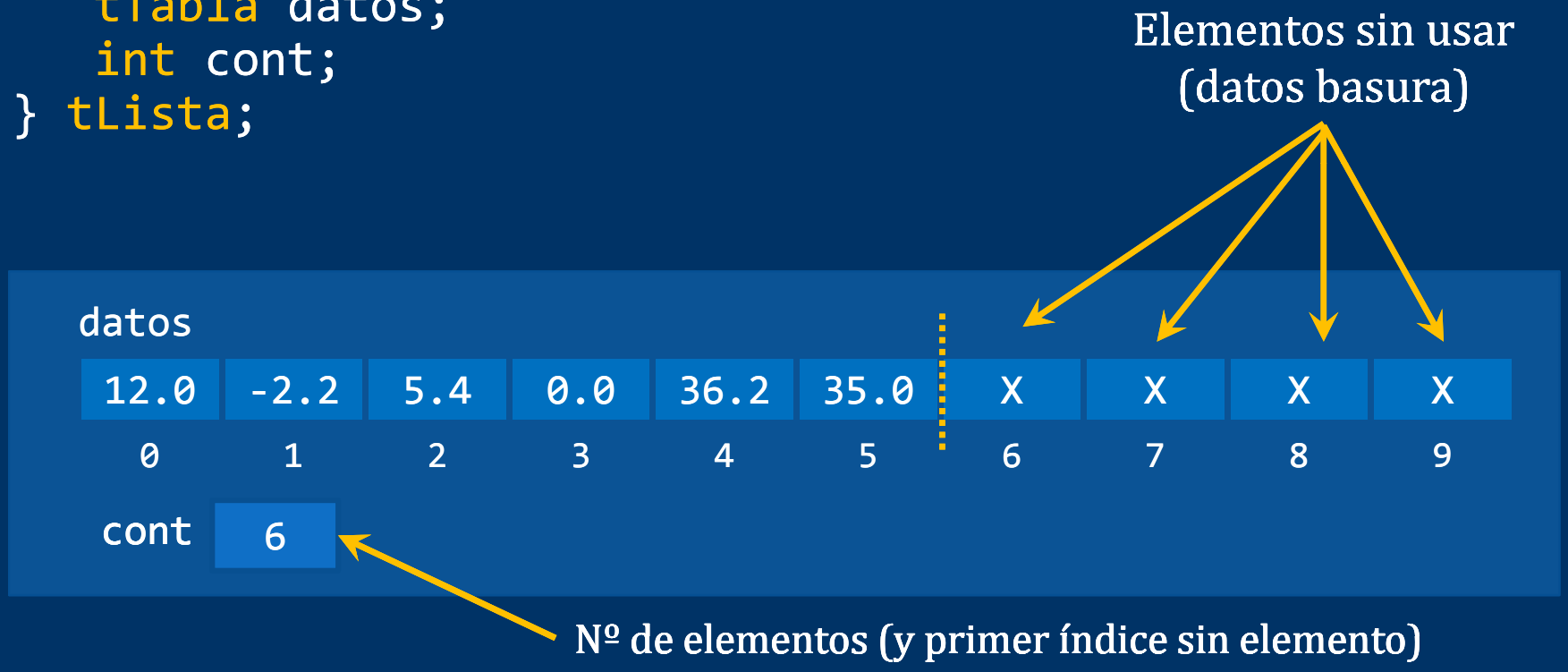
NIF de la primera persona: `personal.datos[0].nif`

Listas de longitud variable

Listas de longitud variable

Estructura que agrupe el array y el contador:

```
const int MAX = N; // Tamaño máximo estimado N>0
typedef tDatos tTabla[MAX]; // supongamos tDato->double
typedef struct {
    tTabla datos;
    int cont;
} tLista;
```

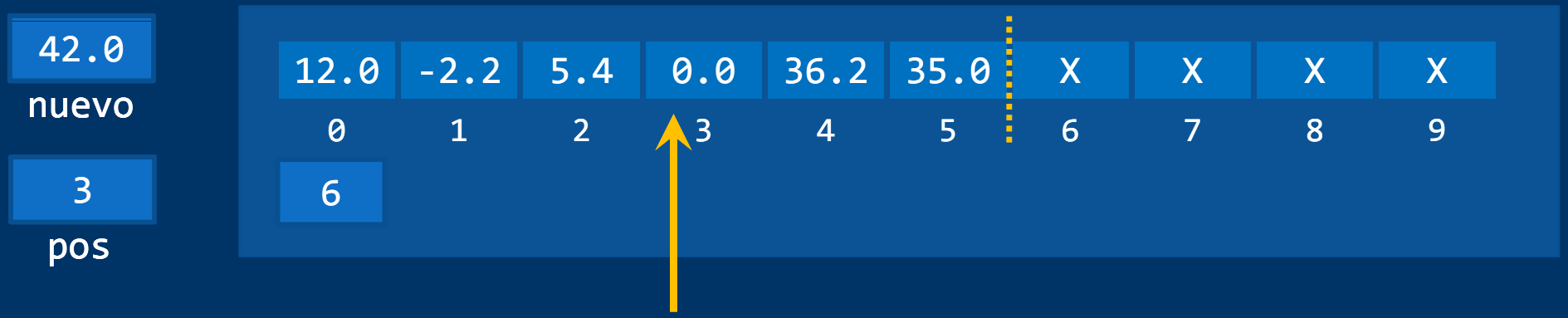


Operaciones principales: búsqueda, inserción y eliminación de elementos

Inserción de elementos

Insertar un nuevo elemento en una posición

Posiciones válidas: 0 a `contador`



Hay que asegurarse de que haya sitio (contador < máximo)

Operación en 3 pasos:

1.- Abrir hueco para el nuevo elemento (en la posición)

2.- Colocar el elemento nuevo en la posición

3.- Incrementar el contador

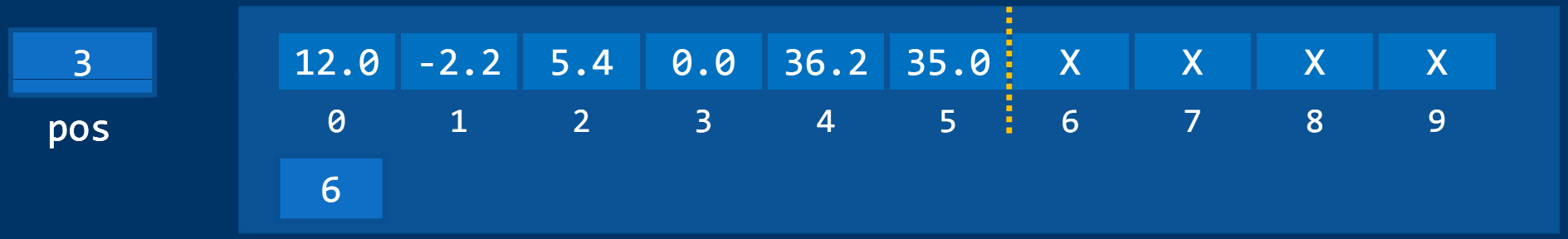
Inserción de elementos

```
bool insertarPos(tLista & list, tDato dato, int pos)
   if (list.cont == MAX || pos>list.cont || pos<0) return false;
   else {
      for (int i = list.cont; i > pos; i--) // Abrir hueco
         list.datos[i] = list.datos[i - 1];
      list.datos[pos] = dato;          // Nuevo dato en pos
      list.cont++;                     // Incrementar contador
      return true;
   }
```

Eliminación de elementos

Eliminar el elemento en una posición

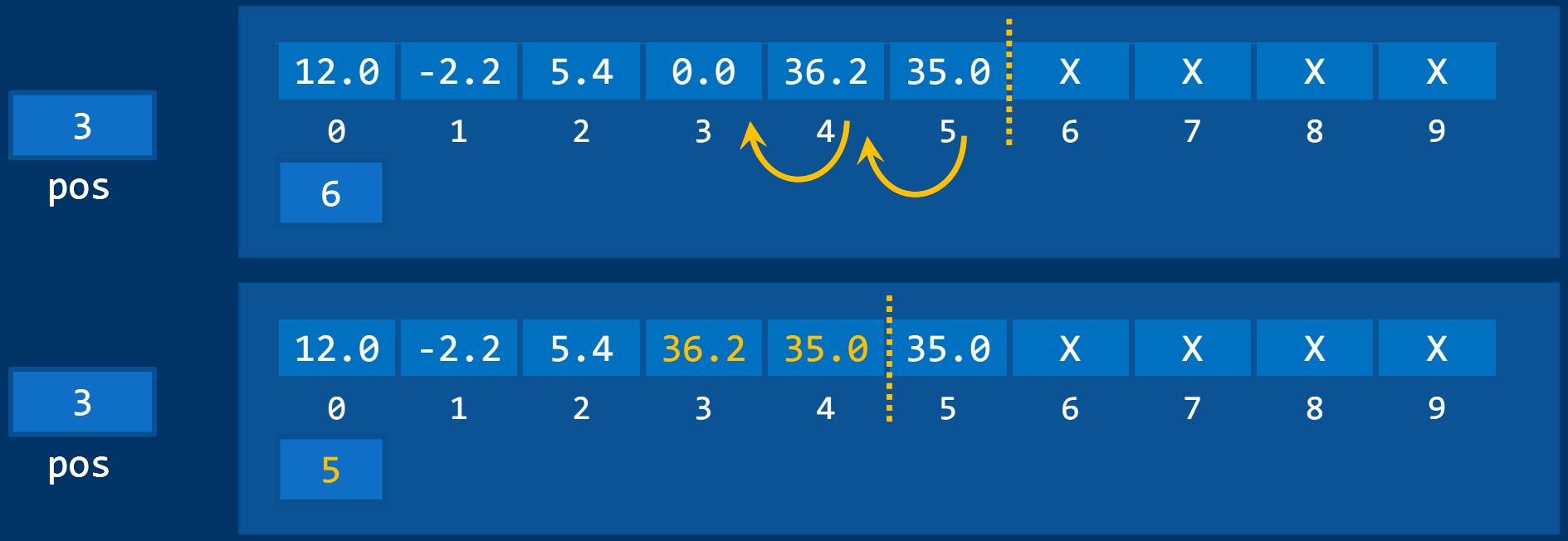
Posiciones válidas: 0 a contador-1




```
bool eliminarPos(tLista & list, int pos){
  if (pos<0 || pos >= list.cont) return false;
  else {
   for (int i = pos + 1; i < list.cont ; i++)
      list.datos[i - 1] = list.datos[i];
   list.cont--;
  return true;
  }
}
```


Eliminación de elementos

```
for (int i = pos + 1; i < lista.cont; i++) {
    lista.datos[i - 1] = lista.datos[i];
}
lista.cont--;
```



Búsqueda

Posición del primer elemento que cumple una propiedad

```
bool buscar(const tLista & list, …, int & pos) {
  bool encontrado = false;
  pos = 0;   // primer elemento
  while ((pos < list.cont) && !encontrado)
  // Mientras no se llegue al final de la lista y no encontrado
    if (propiedad(list, pos, …)) encontrado = true;
    else pos++;
  return encontrado; // ... En al posición pos
}
```

Búsqueda

Posición del primer elemento a partir de una posición dada

```
bool buscarDesde(const tLista & list, …, int & pos) {
  // pos = 0; // primer elemento a partir de pos !!
  bool encontrado = false;
  while ((pos < list.cont) && ! encontrado)
  // Mientras no se llegue al final de la lista y no encontrado
    if (propiedad(list, pos, …)) encontrado = true;
    else pos++;
  return encontrado; // ... En al posición pos
}

bool buscarDesde(const tLista & list, …, int & pos) {
  // primer elemento a partir de pos !!
  while ((pos < list.cont) && ! propiedad(list, pos, …))
  // Mientras no se llegue al final de la lista y no encontrado
    pos++;
  return (pos < list.cont); // ... En al posición pos
}
```

Búsqueda

Posición del primer elemento a partir de una posición dada

Ejemplo: propiedad ser igual a un dato

```
bool buscarDesde(const tLista &list, tDato buscado, int & pos)
{  // primer elemento a partir de pos !!
  bool encontrado = false;
  while ((pos < list.cont) && ! encontrado)
    if (list.datos[pos] == buscado) encontrado = true;
    else pos++;
  return encontrado; // ... En al posición pos
}

bool buscarDesde(const tLista &list, tDato buscado, int & pos)
{  // primer elemento o a partir de pos !!
  while ((pos < list.cont) && list.datos[pos] != buscado)
    pos++;
  return (pos < list.cont); // ... En al posición pos
}
```

Cadenas de caracteres de tipo `string`

Cadenas de tipo `string`

El tipo `string`

- ✓ El tipo asume la responsabilidad de la gestión de memoria
- ✓ Define operadores sobrecargados (p.e., + para concatenar)
- ✓ Cadenas más eficientes y seguras de usar

Biblioteca `string`
Requiere establecer el espacio de nombres a `std`

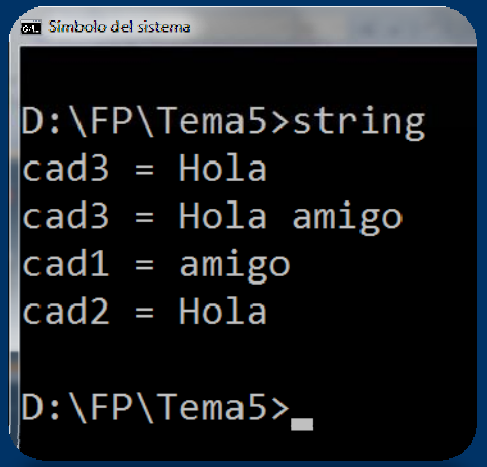
- ✓ Se pueden inicializar en la declaración
- ✓ Se pueden copiar (asignación, paso de parámetros por valor)
- ✓ Se pueden comparar con los operadores: ==, <, <=, >, >=
- ✓ Se pueden concatenar con el operador +
- ✓ Multitud de funciones de utilidad

Cadenas de tipo string

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string cad1("Hola");   // inicialización
    string cad2 = "amigo"; // inicialización
    string cad3;
    cad3 = cad1; // copia
    cout << "cad3 = " << cad3 << endl;
    cad3 = cad1 + " "; // concatenación
    cad3 += cad2;      // concatenación
    cout << "cad3 = " << cad3 << endl;
    cad1.swap(cad2);  // intercambio
    cout << "cad1 = " << cad1 << endl;
    cout << "cad2 = " << cad2 << endl;

    return 0;
}
```



E/S con cadenas de tipo `string`

- ✓ Se muestran en la pantalla con `cout <<`
- ✓ Lectura con `cin >>`: termina con separador
- ✓ Descartar el resto de los caracteres del búfer:

```
cin.sync();   cin.ignore(INT_MAX, '\n');
```

- ✓ Lectura incluyendo espacios en blanco:

```
getline(cin, cadena);
```

Guarda en la *cadena* los caracteres leídos hasta el fin de línea

`cin.getline(`*cad, max*`)`	Cadenas al estilo de C
`getline(cin, cad)`	Cadenas de tipo `string`

- ✓ Lectura de archivos de texto: Igual que de consola

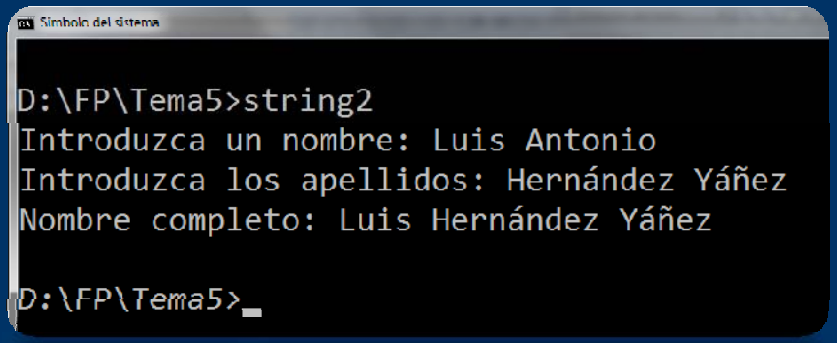
`archivo >> cadena` `getline(archivo, cadena)`

E/S con cadenas de tipo string

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string nombre, apellidos;
    cout << "Introduzca un nombre: ";
    cin >> nombre;
    cout << "Introduzca los apellidos: ";
    cin.sync();
    getline(cin, apellidos);
    cout << "Nombre completo: " << nombre << " "
         << apellidos << endl;

    return 0;
}
```



Operaciones con `string`

Notación punto (.)

- ✓ Longitud de la cadena: `cadena.length()` o `cadena.size()`
- ✓ Acceso a los caracteres de una cadena

  Recuerda que los índice comienzan en 0.
  - ❖ Como array de caracteres: *`cadena`*`[`*`i`*`]`
    Sin control de acceso a posiciones inexistentes del array

    Sólo debe usarse si se está seguro de que el índice es válido
  - ❖ Función `at(`*`índice`*`)`: *`cadena`*`.at(`*`i`*`)`
    Error de ejecución si se accede a una posición inexistente
- ✓ Añadir y eliminar caracteres al final de la cadena
  - ❖ *`push_back(char)`*
  - ❖ *`pop_back();`*

  Modifican la longitud de la cadena

Operaciones con `string`

- ✓ `substr(`*`posición, longitud`*`)` Notación punto (.)

  Subcadena de *longitud* caracteres desde *posición*

```
string cad = "abcdefg";
cout << cad.substr(2, 3); // Muestra cde
```

- ✓ `compare(`*`cadena2`*`)`: 0 si las cadenas son iguales, 1 si *cadena2* es menor que la cadena receptora y -1 si *cadena2* es mayor

```
string cad1 = "Hola", cad2 = "Adiós";
cout << cad1.compare(cad2); // Muestra 1
```

  Operadores relaciones: `==`, `<`, `<=`, `>`, y `>=`

- ✓ `find(`*`subcadena`*`)`: posición en la que empieza la primera ocurrencia de la *subcadena* en la cadena receptora

```
string cad = "Hola";
cout << cad.find("la"); // Muestra 2
```

Operaciones con `string`

- ✓ `rfind(`*`subcadena`*`)`: posición en la que empieza la última ocurrencia de la *subcadena* en la cadena receptora

```
string cad = "Olala";
cout << cad.rfind("la"); // Muestra 3
```

- ✓ `find_first_of(cadena2)`: posición en la que aparece por primera vez cualquier carácter de *cadena2* en la cadena receptora

```
cout << cad.find_first_of("aeiou"); // Muestra 2
```

- ✓ `c_str()`: devuelve la cadena de caracteres al estilo C

```
cout << cad.c_str(); // Muestra "Olala"
```

- ✓ `stoi(cadena)`: si la cadena es un literal entero, lo devuelve como valor `int`
- ✓ `tostring(entero)`: devuelve una cadena con la representación del entero.

Operaciones con `string`

Más sobre cadenas de tipo `string`

- ✓ Modificación de cadenas:

```
cadena.erase(0, 7); // Elimina 7 caracteres desde el 1º
cadena.replace(9, 5, cad2); // Reemplaza 5 caracteres a
                            // partir del 9º por cad2
cadena.insert(0, cad3); // Inserta en la posición 0 cad3
cadena.append(3, '!'); // Añade al final 3 caracteres
                       // de signo de exclamación (!)
```

http://www.cplusplus.com/reference/string/string/

- ✓ Parámetros de tipo `string` *:*

De entrada: por valor o por referencia constante (`const &`*)*

De salida o entrada/salida: por referencia (poniendo &)

Un ejemplo completo

Ejemplo de lista de longitud variable

Descripción

Programa que mantenga una lista de los estudiantes de una clase

De cada estudiante: nombre, apellidos, edad, NIF y nota

- ✓ Se desconoce el número total de estudiantes (máximo 100)
- ✓ La información de la lista se mantiene en un archivo `clase.txt`

  Se carga al empezar y se guarda al finalizar
- ✓ El programa debe ofrecer estas opciones:
  - Añadir un nuevo alumno
  - Eliminar un alumno existente
  - Calificar a los estudiantes
  - Listado de notas, identificando la mayor y la media

Ejemplo de lista de longitud variable

```
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
using namespace std;

const int MAX = 100;
typedef struct {
   string nombre;
   string apellidos;
   int edad;
   string nif;
   double nota;
} tEstudiante;
typedef tEstudiante tTablaEstu[MAX];
typedef struct {
   tTablaEstu datos;
   int cont;
} tListaEstu;
```

Declaraciones de constantes y tipos globales

Ejemplo de lista de longitud variable

```
int menu(); // Menú del programa - devuelve la opción elegida
bool cargar(tListaestu &lista); // Carga del archivo
bool leer(istream & flujo, tEstudiante &estudiante);
void escribir(ostream & flujo, const tEstudiante &estudiante);
// Lee (escribir) los datos del siguiente estudiante del flujo
void guardar(const tListaEstu &lista); // La guarda en el archivo
void leer(tEstudiante &estudiante); // Lee los datos de consola
bool insertar(tListaEstu &lista, const tEstudiante & estudiante);
// Inserta un nuevo estudiante al final de la lista
bool eliminar(tListaEstu &lista, int pos);
// Elimina el estudiante en esa posición
bool buscar(const tListaEstu &lista, string nif int & pos);
// Busca el estudiante con ese nif y devuelve la posición
void calificar(tListaEstu &lista); // Notas de los estudiantes
double mediaClase(const tListaEStu &lista); // Nota media
int mayorNota(const tListaEstu &lista);
// Índice del estudiante con mayor nota
void mostrar(const tEstudiante &estudiante);
void listado(const tListaEstu &lista, double media, int mayor);
```

Ejemplo de lista de longitud variable

El archivo `clase.txt`

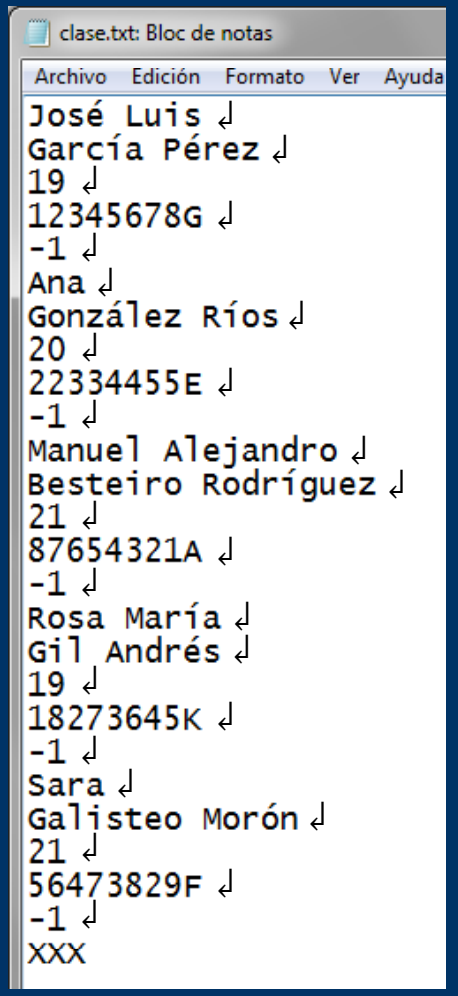
Un dato en cada línea

Por cada estudiante:

- ✓ Nombre (cadena)
- ✓ Apellidos (cadena)
- ✓ Edad (entero)
- ✓ NIF (cadena)
- ✓ Nota (real; -1 si no calificado)

Termina con XXX como nombre

El archivo se supone correcto



Ejemplo de lista de longitud variable

Lectura de la información de un estudiante

Nombre y apellidos:

Puede haber varias palabras → `getline()`

Edad → extractor (>>)

NIF: Una sola palabra → extractor (>>)

Nota → extractor (>>)

Queda pendiente de leer el Intro

Hay que saltar (leer) ese carácter con `get()`, `ignore`, `getline`

Si no, en el siguiente nombre se leería una cadena vacía (Intro)

Carga del archivo clase.txt

```
bool cargar(tListaEstu & lista) {  // versión 1
   ifstream archivo;
   lista.cont = 0; // Inicializamos la lista
   archivo.open("clase.txt");
   if (!archivo.is_open()) return false;
   else {
     tEstudiante estudiante;
     getline(archivo, estudiante.nombre);
     while ((lista.cont < MAX) && estudiante.nombre != "XXX"){
        getline(archivo, estudiante.apellidos);
        archivo >> estudiante.edad;
        archivo >> estudiante.nif;
        archivo >> estudiante.nota;
        archivo.ignore(INT_MAX, '\n');
        lista.datos[lista.cont] = estudiante;
        lista.cont++;
        getline(archivo, estudiante.nombre);
     }
     // Si hay más de MAX estudiantes, ignoramos el resto
     archivo.close();
     return true;
   }
}
```


Lectura de los datos de un estudiante

```
bool leer(istream & flujo, tEstudiante & estudiante) {
   getline(flujo, estudiante.nombre);
   getline(flujo, estudiante.apellidos);
   flujo >> estudiante.edad;
   flujo >> estudiante.nif;
   flujo >> estudiante.nota;
   if (flujo.fail()) return false;
   else {    // Descartamos cualquier entrada pendiente
     flujo.ignore(INT_MAX, '\n');
     return true;
   }
}
void escribir(ostream & flujo, const tEstudiante &estudiante){
   flujo << estudiante.nombre << endl;
   flujo << estudiante.apellidos << endl;
   flujo << estudiante.edad << endl;
   flujo << estudiante.nif << endl;
   flujo << estudiante.nota << endl;
}
```

Carga del archivo clase.txt

```
bool cargar(tListaEstu & lista) {  // versión 2

    ifstream archivo;
    char aux;
    lista.cont = 0; // Inicializamos la lista
    archivo.open("clase.txt");

    if (!archivo.is_open())
       return false;
    else {
       bool fin = false;
       while ((lista.cont < MAX) && ! fin)
          if (leer(archivo, lista.datos[lista.cont]))
                lista.cont++;
          else fin = true;
       // Si hay más de MAX estudiantes, ignoramos el resto
       archivo.close();
       return true;
    }
}
```

Volcado en el archivo clase.txt

```
void guardar(const tListaEstu & lista) {
   ofstream archivo;
   archivo.open("clase.txt");
   for (int i = 0; i < lista.cont; i++)
      escribir(archivo, lista.datos[i]);
   // Centinela final ??
   archivo << "XXX" << endl;
   archivo.close();
}
```

`const tListaEStu &lista` → Referencia constante
Paso por referencia pero como constante ≡ Parámetro de entrada
Evita la copia del argumento en el parámetro (estructuras grandes)

Lectura de los datos de un estudiante

```
void leer(tEstudiante &estudiante) {
    cin.sync(); // Descartamos cualquier entrada pendiente
    cout << "Nombre: ";
    getline(cin, estudiante.nombre);
    cout << "Apellidos: ";
    getline(cin, estudiante.apellidos);
    cout << "Edad: ";
    cin >> estudiante.edad;
    cout << "NIF: ";
    cin >> estudiante.nif;
    estudiante.nota = -1; // Sin calificar de momento
    cin.sync(); // Descartamos cualquier entrada pendiente
}
```

Inserción de un nuevo estudiante

```
bool insertar(tLista &lista, const tEstudiante & estudiante) {
   if (lista.cont == MAX)
      return false;
   else {   // Insertamos al final
      lista.datos[lista.cont] = estudiante;
      lista.cont++;
      return true;
   }
}

bool buscar(const tListaEstu &lista, string nif, int & pos) {
  pos = 0;   // primer elemento
  while ((pos < list.cont) && list.datos[pos].nif != nif)
     pos++;
  return pos < list.cont; // ... En al posición pos
}
```

Eliminación de un estudiante

```
bool eliminar(tListaEstu &lista, int pos) {

   if ((pos < 0) || (pos >= lista.cont)) {
      return false; // Elemento inexistente
   }
   else {
      for (int i = pos + 1; i < lista.cont; i++) {
         lista.datos[i - 1] = lista.datos[i];
      }
      lista.cont--;
      return true;
   }
}
```

Calificación de los estudiantes

```
void calificar(tListaEstu &lista) {

   for (int i = 0; i < lista.cont; i++) {
      cout << "Nota del estudiante "
         << lista.datos[i].nombre << " "
         << lista.datos[i].apellidos << ": ";
      cin >> lista.datos[i].nota;
   }
}
```


Más subprogramas

```
double mediaClase(const tListaEstu &lista) {
   double total = 0.0;
   for (int i = 0; i < lista.cont; i++) {
      total = total + lista.datos[i].nota;
   }
   return total / lista.cont;
}

int mayorNota(const tListaEstu &lista) {
   double max = 0;
   int pos = 0;
   for (int i = 0; i < lista.cont; i++) {
      if (lista.datos[i].nota > max) {
         max = lista.datos[i].nota;
         pos = i;
      }
   }
   return pos;
}
```

El listado

```
void mostrar(const tEstudiante & estudiante) {
   cout << setw(35) << left
        << estudiante.nombre + " " + estudiante.apellidos;
   cout << estudiante.nif << " ";
   cout << setw(2) << estudiante.edad << " años ";
   cout << fixed << setprecision(1) << estudiante.nota;
}

void listado(const tLista &lista, double media, int mayor) {
   for (int i = 0; i < lista.contador; i++) {
      cout << setw(3) << i << ": ";
      mostrar(lista.datos[i]);
      if (i == mayor)
         cout << " <<< Mayor nota!";
      cout << endl;
   }
   cout << "Media de la clase: " << fixed << setprecision(1)
      << media << endl << endl;
}
```

El programa principal

```
int main() {
   tListaEstu lista;
   tEstudiante estudiante;
   bool exito;
   int op, pos; string nif;
    if (! cargar(lista))
       cout << "No se ha podido cargar la lista!" << endl;
    else {
       op = menu();
       while (op != 0) {
          if (op == 1) { // Añadir estudiante
             leer(estudiante);
             if (! insertar(lista, estudiante))
                cout << "Lista llena: imposible insertar" << endl;
          }
          else if(op == 2) { // Eliminar estudiante
             cout << "Nif: ";
             cin >> nif;
             if (buscar(lista, nif, pos)) eliminar(lista, pos);
             else  cout << "Elemento inexistente!" << endl;
          }
```

El programa principal

```
            else if(op == 3)
               calificar(lista);
            else if(op == 4)
               listado(lista, mediaClase(lista), mayorNota(lista));

            op = menu();
         } // fin while

         guardar(lista);
      }
      return 0;
   }
```


Acerca de *Creative Commons*

Licencia CC (Creative Commons)

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

- Reconocimiento (*Attribution*):
  En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
- No comercial (*Non commercial*):
  La explotación de la obra queda limitada a usos no comerciales.
- Compartir igual (*Share alike*):
  La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.