XDR

(External Data Representation)

Tipos y codificación

Características

- Tipos implícitos
- La codificación (*big endian)* ocupa siempre un tamaño múltiplo de **4 bytes** (rellenando con ceros por la dcha)
- Se asume que la transmisión:
  - Preserva el contenido de cada byte
  - Preserva el orden de los bytes (el primero se numera con 0)

Sintaxis genérica

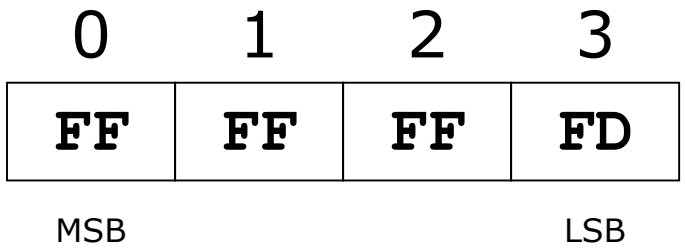
- Muchos de los *tipos simples* tienen el mismo nombre que en C
- El usuario puede definir *tipos complejos* agregando tipos simples
- **<>** denota un número variable de datos en una secuencia
- **[]** denota un número fijo de datos en una secuencia

Tipos simples: Entero

- Sintaxis:

  `int` *`identificador`*`;`

- Codificación
  - 32 bits (4 bytes), negativos en C-2
  - Rango: `[-2147483648, 2147483647]`
  - Ejemplo: **-3**

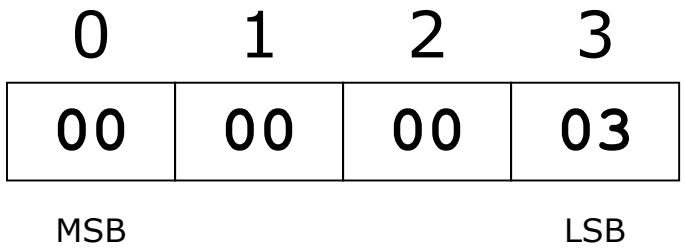


Tipos simples: Natural

- Sintaxis:

  `unsigned int` *`identificador`*`;`

- Codificación
  - 32 bits (4 bytes)
  - Rango: `[0, 4294967295]`
  - Ejemplo: **3**



Tipos simples: Enumerados

- Sintaxis:

```
enum identificador
  {nombre1=constante, ...};
```

- Ejemplo:

```
enum colores {
 ROJO=1, AMARILLO=3, AZUL=5 };
```

- Codificación
  - Como los enteros

Tipos simples: Booleanos

- Sintaxis:

  bool *identificador;*

- Codificación
  - Equivalente a:

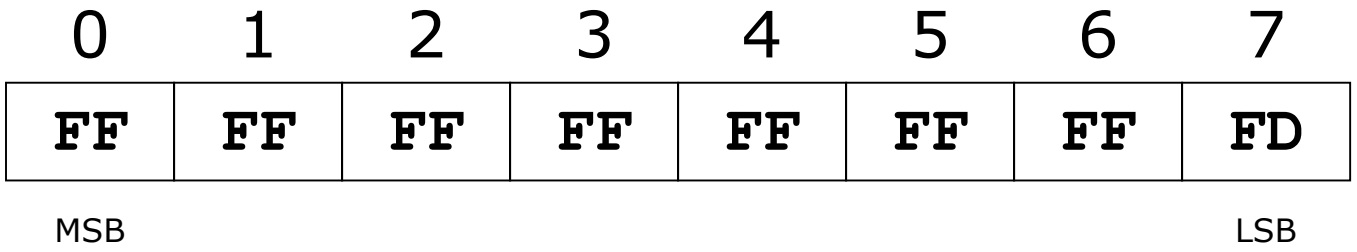
    enum bool {TRUE=1, FALSE=0};

Tipos simples: Hiperenteros

- Sintaxis:

  `hyper` *`identificador;`*

- Codificación
  - 64 bits (8 bytes)
  - Rango: $[-2^{63}, 2^{63}-1]$
  - Ejemplo: **-3**

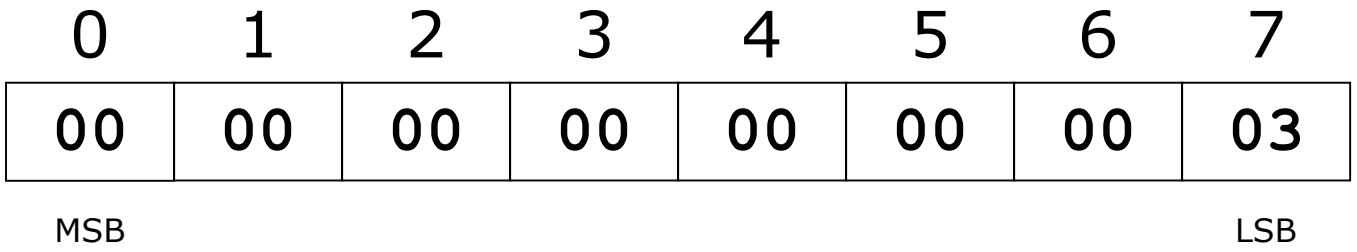


Tipos simples: Hiperenteros sin signo

- Sintaxis:

  `unsigned hyper` *`identificador;`*

- Codificación
  - 64 bits (8 bytes)
  - Rango: [0, 2^{64}-1]
  - Ejemplo: **3**

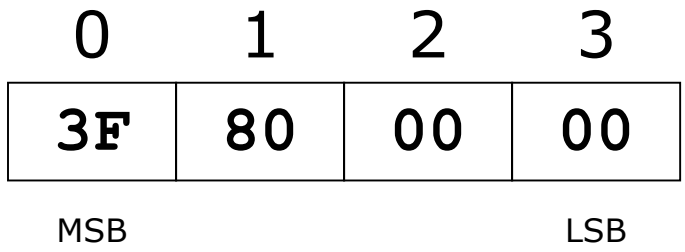


Tipos simples: Reales (precisión simple)

- Sintaxis:

  `float` *`identificador`*`;`

- Codificación
  - 32 bits (4 bytes)
  - Coma flotante, norma IEEE-754
  - Ejemplo: +**1**

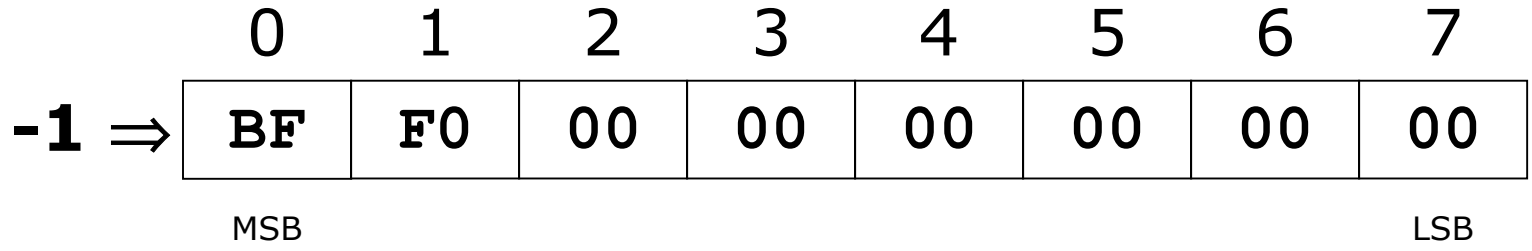


Tipos simples: Reales (precisión doble)

- Sintaxis:

  `double` *`identificador`*`;`

- Codificación
  - 64 bits (8 bytes)
  - Coma flotante, norma IEEE-754, precisión ampliada.



Tipos simples: Constantes

- Sintaxis:

  const *identificador*=valor;

- Codificación
  - La del tipo asociado al valor.
  - Ejemplo:

    const MAX_CLIENTES=5;

Tipo opaco

- Secuencia de *n* bytes sin significado especial para XDR.
- Opaco de longitud fija:
  - Sintaxis:

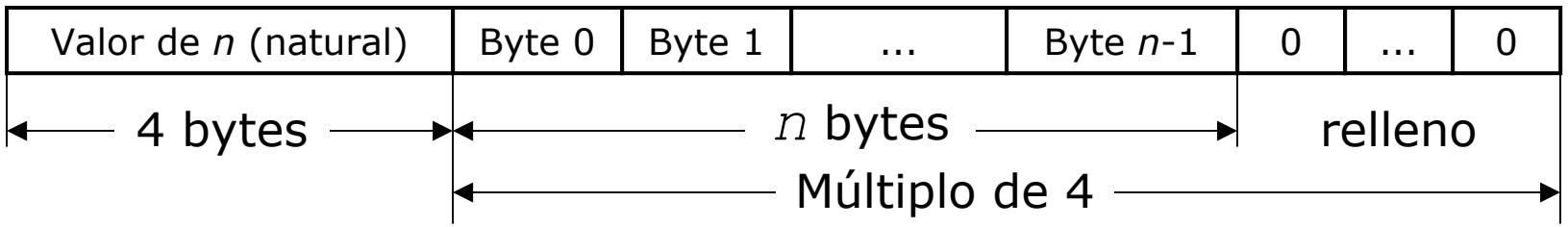
    opaque *identificador*[n];
  - Codificación

Tipo opaco

- Opaco de longitud variable:
  - Sintaxis:

    opaque *identificador*<m>;

    m especifica el tamaño *maximo* (n ≤ m)
    Puede omitirse (pero no las <>)
  - Codificación:
    - Primero se envia el número *n* de bytes, y despues sus valores



Datos estructurados

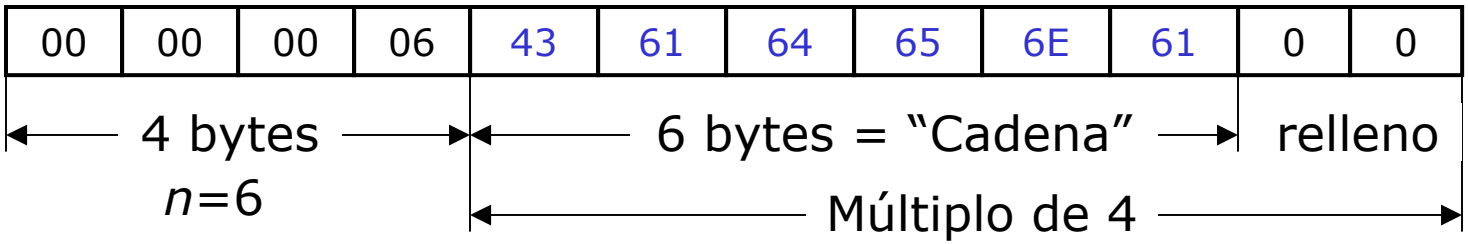
- Cadenas de texto
- Arrays
  - De longitud fija
  - De longitud variable
- Estructuras
- Uniones discriminadas
- Datos opcionales

Cadenas de texto

- Una cadena es una secuencia de *n* códigos ASCII
- Sintaxis:

  `string` *`identificador`*`<m>;`

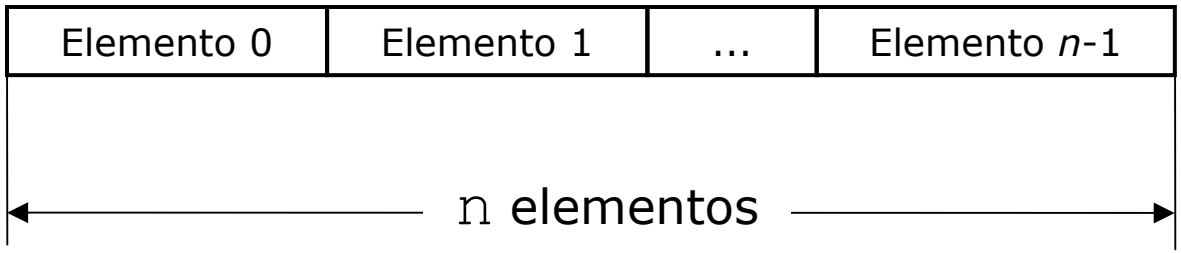
  `m` especifica el tamaño *maximo* de la cadena (`n`≤`m`).
  Puede omitirse (pero no las <>)
- Codificación: como los opacos
- Ejemplo: **`string txt<20>;`**
  inicializado con el texto "**`Cadena`**"



Arrays

- Un array es una secuencia de *n* datos del mismo tipo.
- Array de longitud fija.
  - Sintaxis

    nombre_tipo *identificador*`[n]`
  - Codificación:

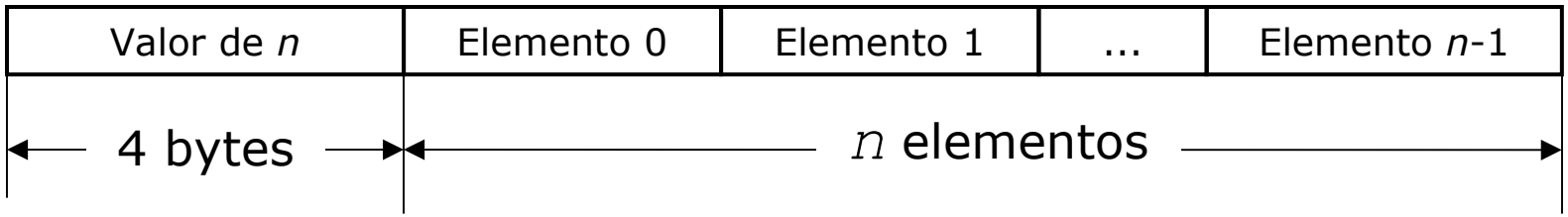


Arrays

- Arrays de longitud variable
  - Sintaxis

    nombre_tipo *identificador***<m>**

    m es el número máximo de elementos. Puede omitirse (pero no las <>)
  - Codificación:
    - Se envía primero el número *n* de elementos (dato entero), y despúes los elementos.



Estructuras

- Sintaxis

```
struct identificador {
    declaracion_componente_A;
    declaracion_componente_B;
      ...
    declaracion_componente_X;
}
```

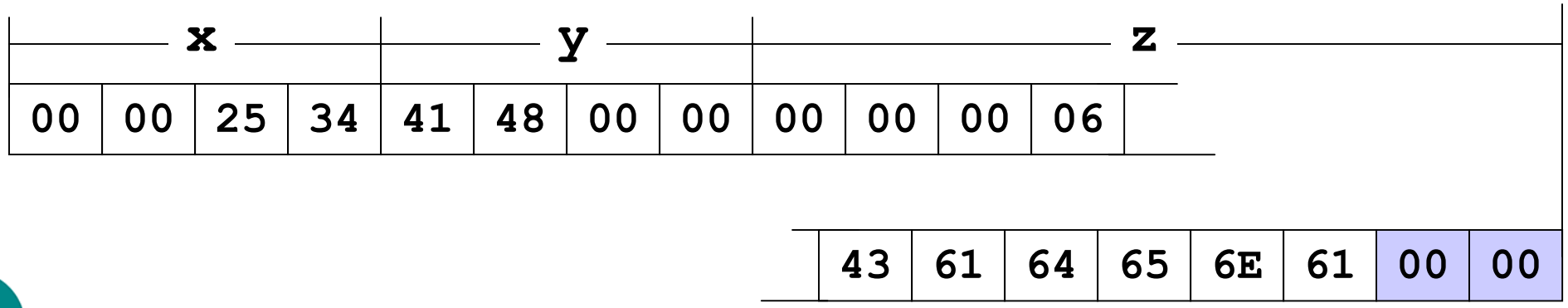
- Codificación:
  - Cada componente se codifica según su tipo y se envían en secuencia.

Componente A	Componente B	...	Componente X

Estructuras: ejemplo

```
struct prueba {
    int x;
    float y;
    string z<10>;
};
```

- Se carga campo **x** con **2534h**, campo **y** con **12.5**, campo **z** con "**Cadena**"
- Codificación:

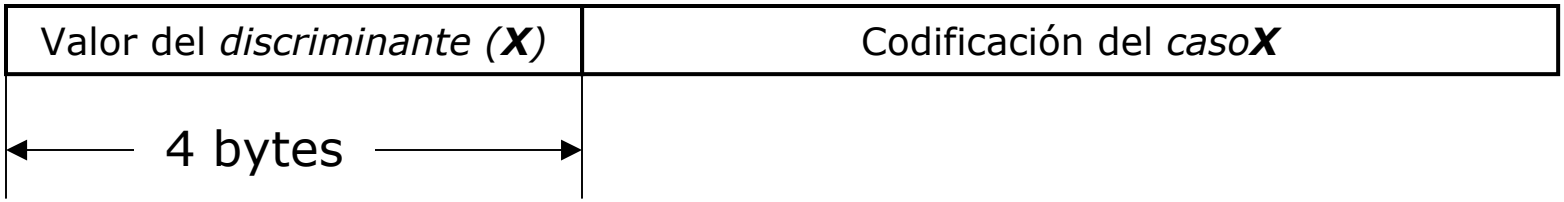


Uniones discriminadas

- Permiten "elegir" el tipo del dato, mediante un *discriminante*
- Sintaxis

```
union identificador
  switch(declaración discriminante){
    case valor1: declaración caso1;
    case valor2: declaración caso2;
       ...
    default: declaración caso defecto;
  }
```

- Codificación:

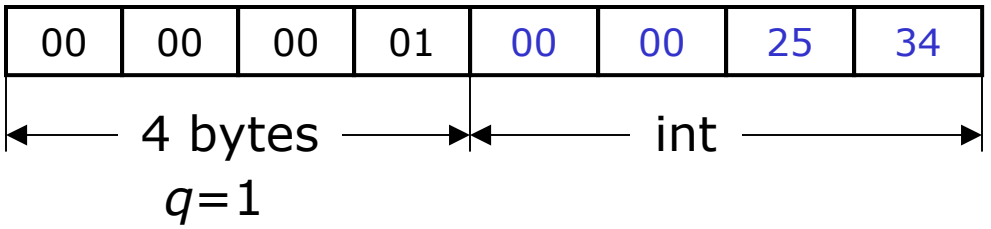


Uniones discriminadas: Ejemplos

```
union ejemplo
switch(int q){
  case 1: int x;
  case 2: float y;
  case 3: double z;
 default:
     string txt<20>;
};
```

Inicialización:

- **q** con **1**
- **x** con **0x2534**

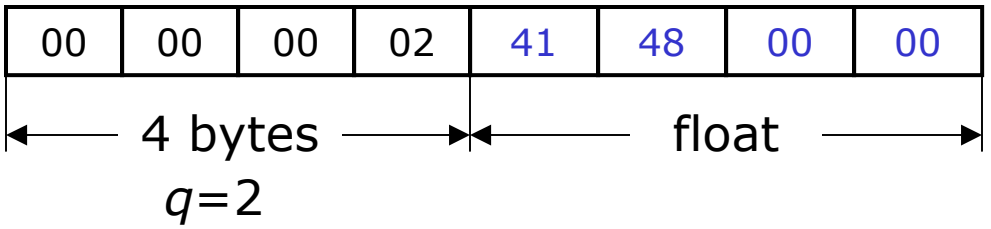


Uniones discriminadas: Ejemplos

```
union ejemplo
switch(int q){
  case 1: int x;
  case 2: float y;
  case 3: double z;
 default:
     string txt<20>;
};
```

Inicialización:

- **q** con **2**
- **y** con **12.5**

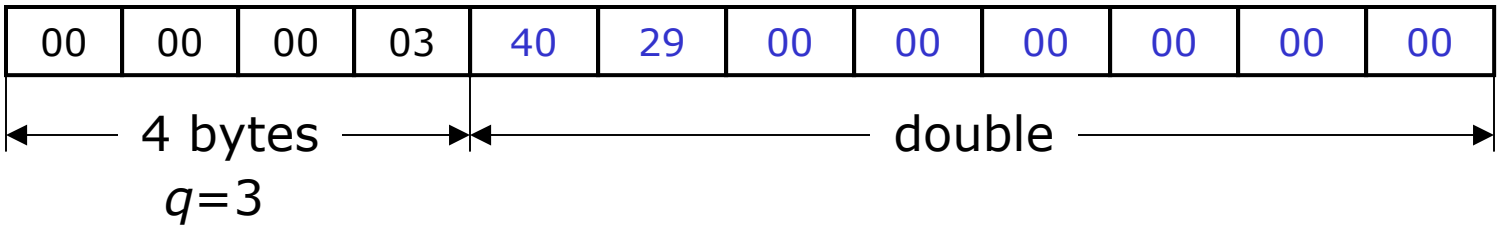


Uniones discriminadas: Ejemplos

```
union ejemplo
switch(int q){
  case 1: int x;
  case 2: float y;
  case 3: double z;
 default:
     string txt<20>;
};
```

Inicialización:

- **q** con 3
- **z** con **12.5**

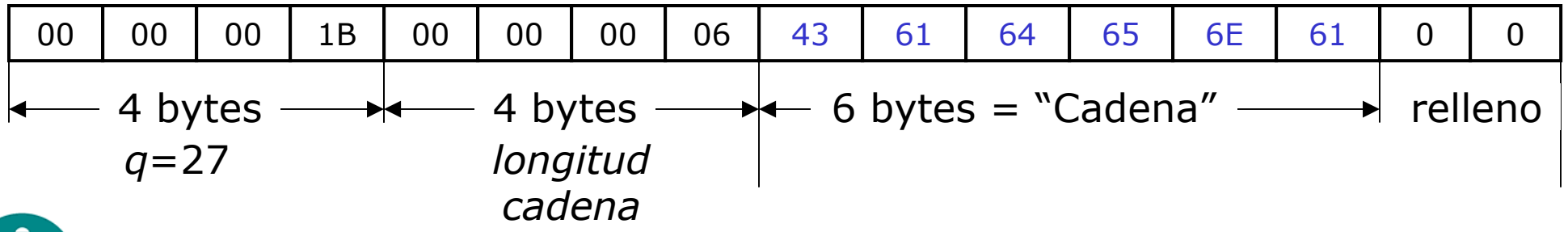


Uniones discriminadas: Ejemplos

```
union ejemplo
switch(int q){
   case 1: int x;
   case 2: float y;
   case 3: double z;
  default:
      string txt<20>;
};
```

Inicialización:

- **q** con **27**
- **txt** con "**Cadena**"

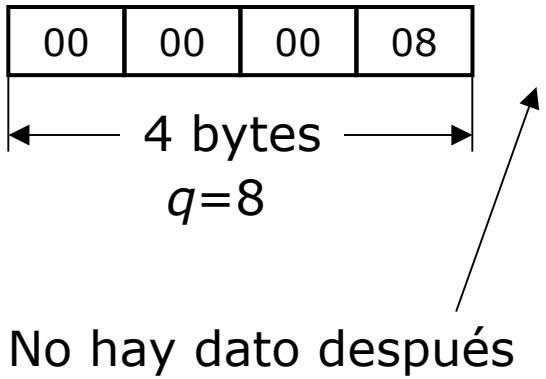


Casos vacíos: void

- El tipo **`void`** permite especificar un dato vacío (inexistente).
- Este tipo no tiene codificación.
- Ejemplo:

```
union ejemplo
switch(int q){
   case 1: int x;
   case 2: float y;
   case 3: double z;
 default: void;
};
```

Si **`q`** vale 8

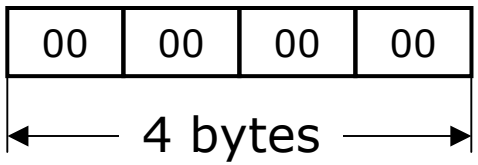


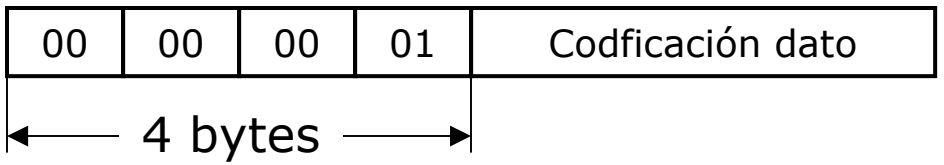
Datos opcionales

- Pueden llevar información o estar vacíos
- Sintaxis

  tipo ***identificador;***

- Codificación:
  - Si no hay dato se codifica el entero 0
  - Si hay dato, se codifica el entero 1 seguido de la codificación del dato





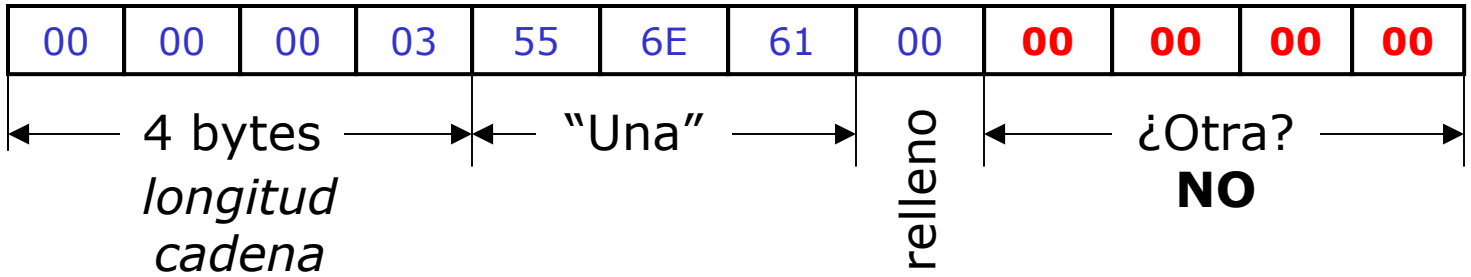
Datos opcionales: uso

- **Uso**: codificar punteros cuyo valor puede ser **`NULL`** (no hay dato) o bien apuntan a un dato.
- **Aplicación típica**: estructuras recursivas (listas, árboles...)
- **Ejemplo**:

```
struct lista {
  string cadena<>;
  lista *otra;
};
```

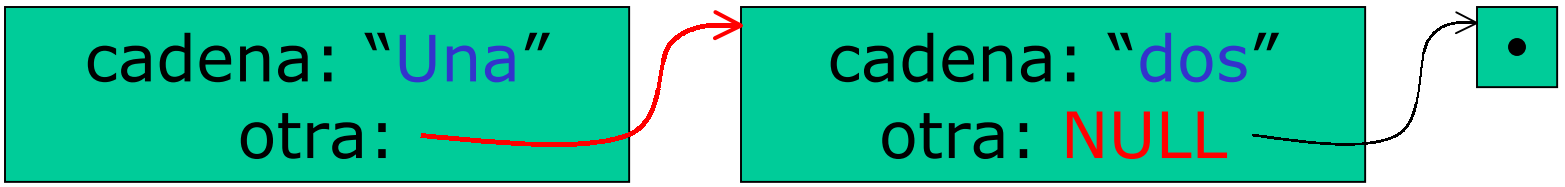
Datos opcionales: ejemplo

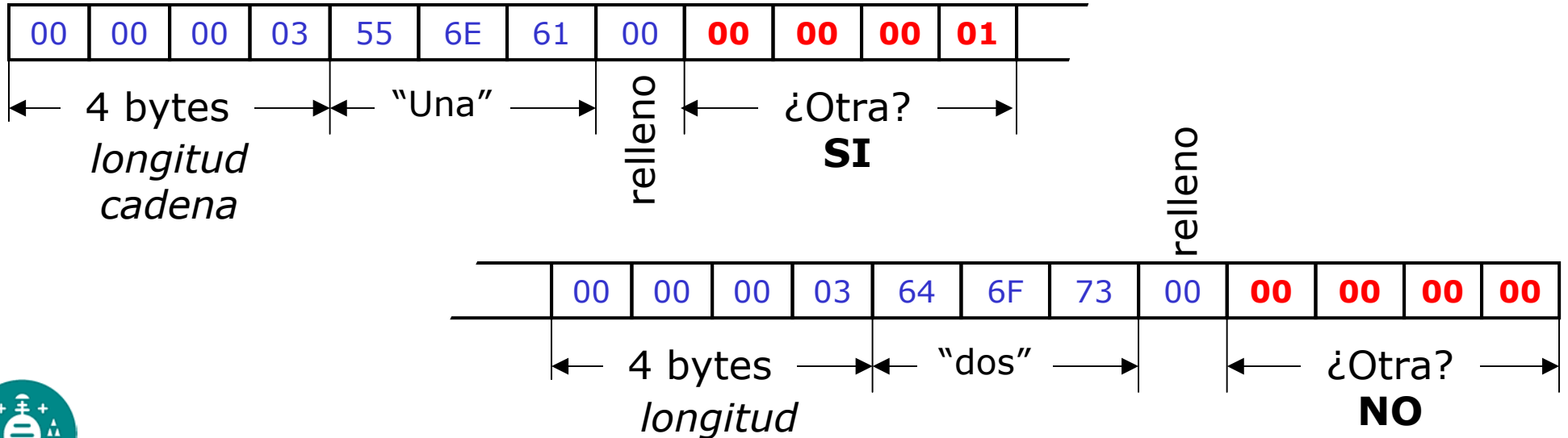
```
struct lista {
   string cadena<>;
   lista *otra;
};
```



Datos opcionales: ejemplo

```
struct lista {
   string cadena<>;
   lista *otra;
};
```





Tipos definidos por el usuario

- La palabra **`typedef`** delante de una *declaración* permite dar nombres a los tipos.
- Ejemplo:

```
typedef int huevo;
typedef huevo huevera[12];
```

  - En lo sucesivo, *huevera* equivale a un array de 12 enteros.