ESTRUCTURAS
y
UNIONES

- **Estructura (struct)**
  - Es un tipo de dato definido por el usuario
  - Permiten la construción de tipos complejos a partir de otros tipos
  - Puede considerarse como una colección de una o más variables (miembros) eventualmente de tipos diferentes (vs arrays) reunidas bajo un nombre comun
  - Ayudan a organizar los datos y evitar el tratamiento independiente de variables que conforman características de una misma “instancia lógica”

Ejemplo

```
struct persona {
        unsigned long int dni;
        unsigned short int edad;
        char sexo;
};   //no olvidar el ;
```

- **Variables tipo struct**
  - Es posible definir variables tipo struct y, por supuesto, punteros a struct, arrays, usarse como tipo para atributos de otras estructuras, etc

```
struct persona {
        unsigned long int dni;
        unsigned short int edad;
        char sexo;
} variable, *puntero;

//Más legible
struct persona juan;
struct persona *ese_es_juan;
struct persona equipo_de_juan[11];

struct familia_tipo {
        struct persona papa;
        struct persona mama;
        struct persona hijos[2];
}
```

- **Variables tipo struct – Acceso a atributos**

```
struct persona {
        unsigned long int dni;
        unsigned short int edad;
        char sexo;
};

struct persona var;
struct persona* ptr;
```

Acceso a atributos con una variable y punto (.)

```
var.atributo            // var.dni
```

Acceso a atributos con un puntero y flecha (->)

```
ptr->atributo           // ptr->dni
```

- **Variables tipo struct – Acceso a atributos**

```
struct persona {
        unsigned long int dni;
        unsigned short int edad;
        char sexo;
};

struct persona var;
struct persona* ptr;
```

Acceso a atributos con una variable y punto (.)

```
var.atributo            // var.dni
```

Acceso a atributos con un puntero y flecha (->)

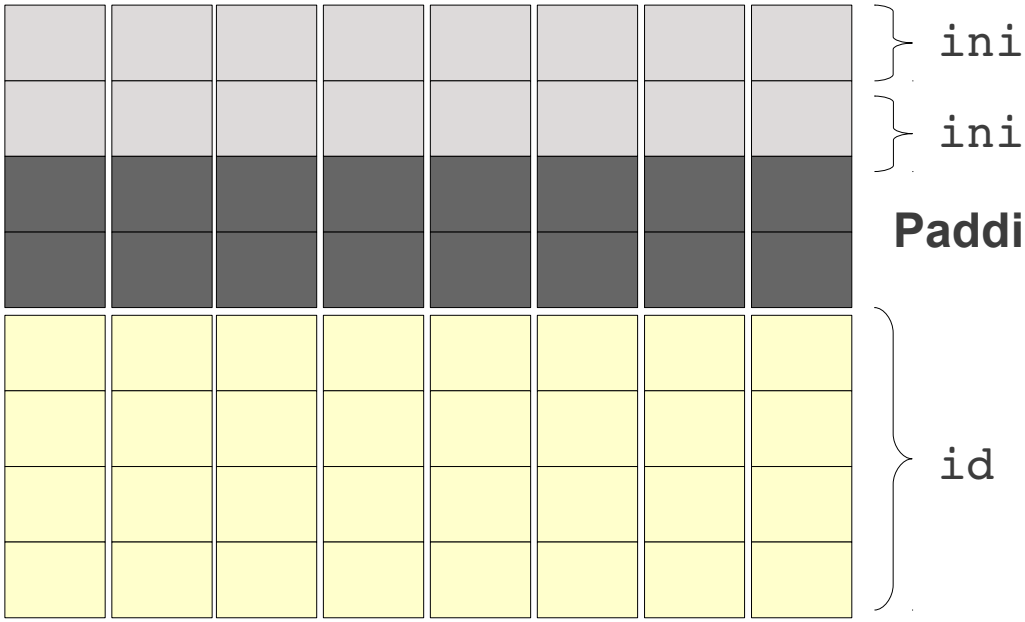
```
ptr->atributo           // ptr->dni
```

Instituto
Politécnico
Superior

- **struct – sizeof**

```
struct ejemplo1 {
    char iniciales[2];
    int id;
};

sizeof(struct ejemplo1)
    1 byte * 2(char) + 4 bytes * 1(int) =   6 bytes?
                                    ----> no, 8 bytes!!
```



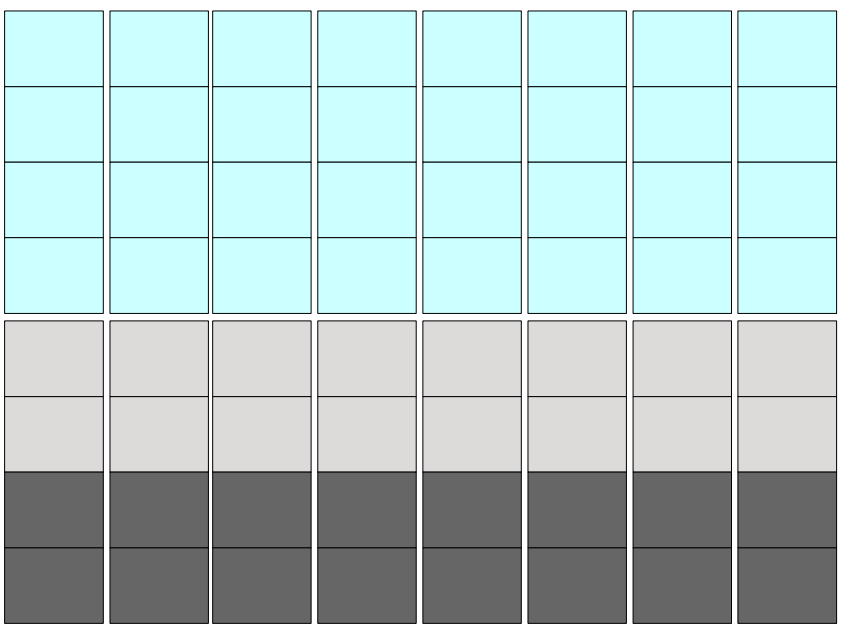
El compilador alinea las variables miembro a un múltiplo del min(pack size, type size)

Instituto
Politécnico
Superior

- **struct – sizeof**

```
struct ejemplo2 {
    int id;
    char iniciales[2];
};
```

Ahora si, sizeof(struct ejemplo2) == 6?
----> no, nuevamente 8 bytes!!

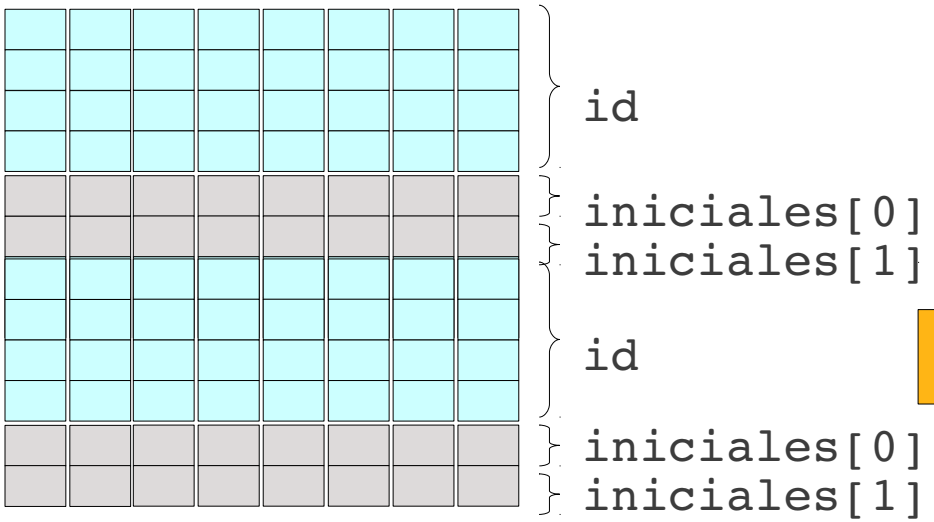


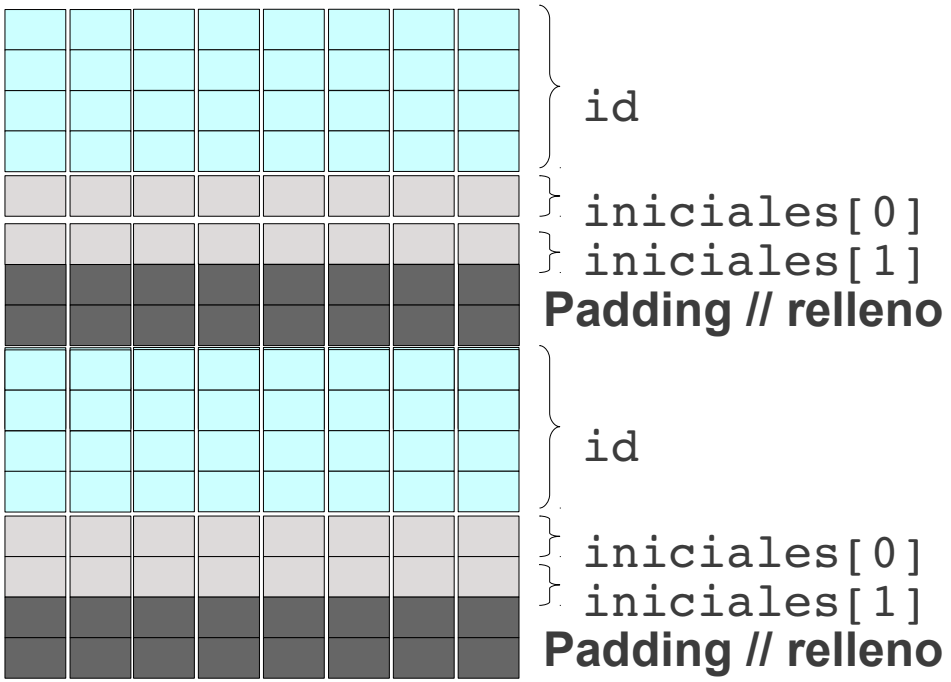
Esto se debe a que la variable miembro “id” precisa ser alineada como múltiplo de 4 bytes si se crea un array de estructuras ejemplo2

Padding // relleno --- dummy bytes

- **struct – sizeof**

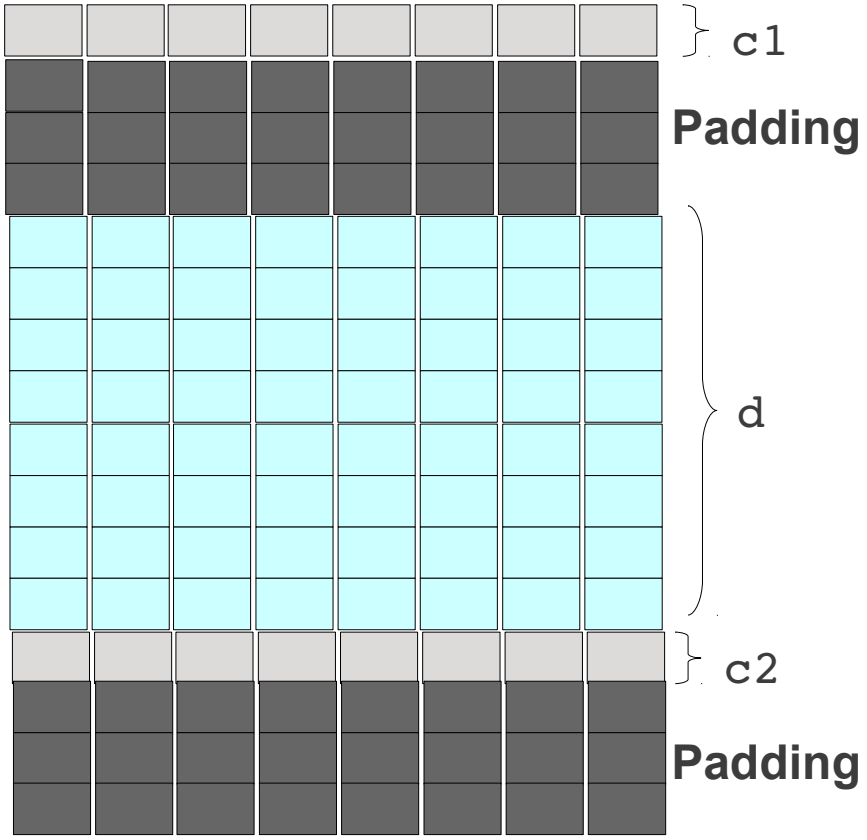
```
struct ejemplo2 {
    int id;
    char iniciales[2];
};
```





- **struct – sizeof**

```
struct ejemplo3 {
    char c1;            // sizeof 1 byte
    double d;           // sizeof 8 bytes
    char c2;            // sizeof 1 byte
};
sizeof(struct ejemplo3) == 16 bytes
```

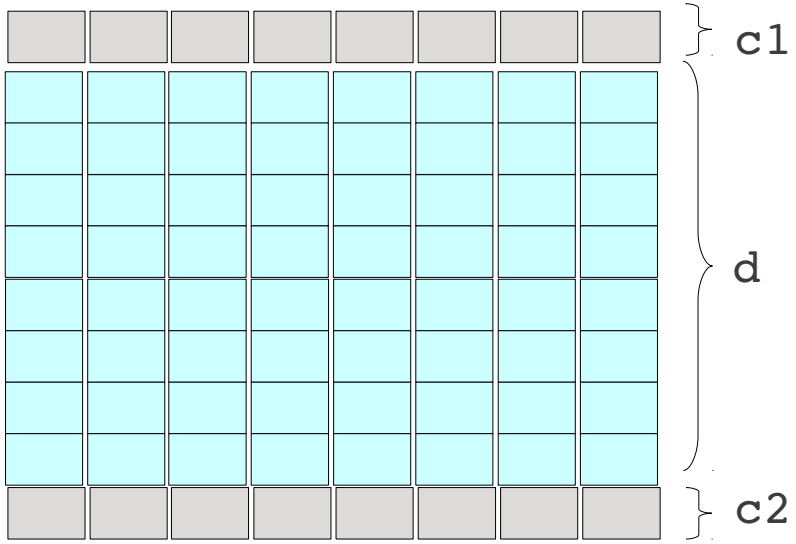


En este caso el pack size es 4 y el size type es 8 (double)
alineación a 4 bytes
37.5% de relleno

Si el pack size fuese 8
sizeof(struct ejemplo3) == 24 bytes
58.3 % de relleno

- **Control del pack size**
  - Es posible controlar el pack size en tiempo de compilación
  - Se emplea una directiva del pre-procesador llamada #pragma

```
#pragma pack(push, 1)  // setear el valor 1
struct ejemplo4 {
    char c1;           // sizeof 1 byte
    double d;          // sizeof 8 bytes
    char c2;           // sizeof 1 byte
};
#pragma pack(pop)      // retornar al valor por defecto
sizeof(struct ejemplo4) == 10 bytes
```



- **Control del pack size**
  - **Caveats**
    - Tener presente el padding
      El mero hecho de cambiar de orden los miembros o atributos puede colaborar en mejor uso de memoria

    - El reducir espacio de almacenamiento impacta en la velocidad de acceso a las variables miembro o atributos (lectura/escritura)

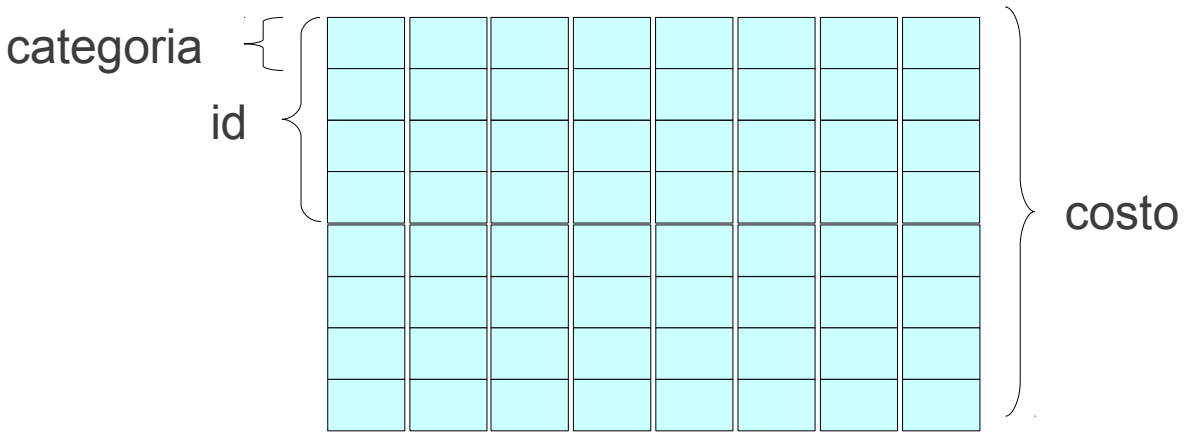
      Priorizar.
      Espacio vs performance

- **Uniones (unions)**
  - Tipo de dato definido por el usuario muy parecido a las estructuras
  - Permiten almacenar, en diferentes momentos, datos de tipo (y tamaño) distinto

Ejemplo

```
union vehiculo {
    int id;                 // sizeof 4 bytes
    double costo;           // sizeof 8 bytes
    char categoria;         // sizeof 1 byte
};  //no olvidar el ;

sizeof(union vehiculo)          8 bytes
```



- **Unions**
  - Cuando se altera el valor de un miembro de una union se alteran eventualmente varios de ellos – comparten la memoria -

  - Se permiten las mismas operaciones que con estructuras
    . asignación o copia como una unidad
    . obtener la dirección mediante &
    . acceder a un miembro

  - Tener presente que debe considerarse el valor de un miembro por vez
    Permiten **preservar memoria** y aplican si se está considerando el uso de una estructura donde no hace falta contar con todos los miembros a la vez

```
struct libro {
    int isbn;
    Char titulo[30];
    union nuevo_o_usado {
        int nuevo;
        int usado;
    };
};
```

- **Construcción enum**
  - Permite definir un rango de enteros y establecer un nombre a cada valor de dicho rango

```
enum bool {false, true};
// false=0, true=1
enum bool bandera;
if (bandera == true) { … }


enum dias_laborables {lunes, martes, miercoles, jueves, viernes};
// lunes vale 0, martes 1, miercoles=2...
enum dias_laborables {lunes=1, martes, miercoles, jueves, viernes};
// lunes vale 1, martes 2, miercoles=3…
enum dias_laborables {lunes, martes=0, miercoles, jueves, viernes};
// lunes vale 0, martes 0, miercoles=1…
```