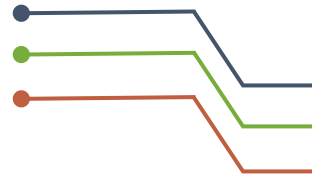
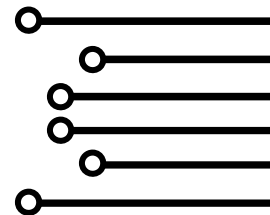




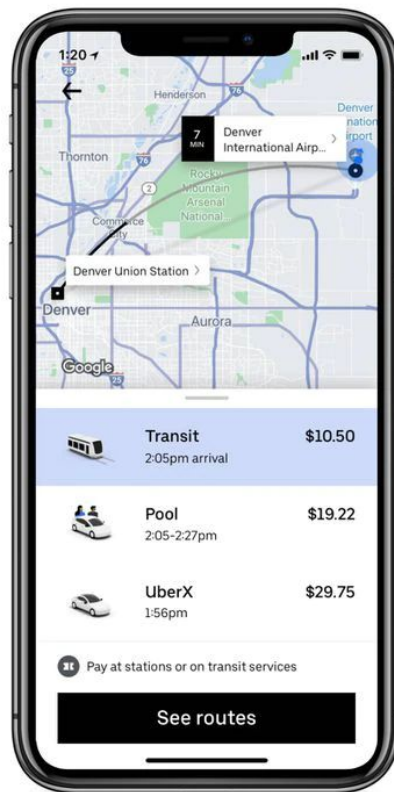
C Embebido





UBER

Wels

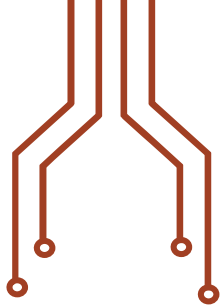


UBER



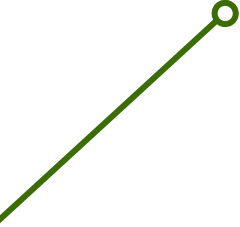
- Identificación del carro.
- Color del carro.
- Nombre del conductor.

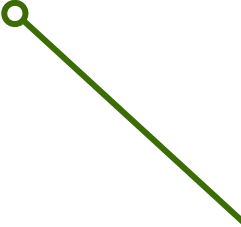
UBER PROPIEDADES



```
struct uber_t
{
    int32_t id_carro;
    char color[1];
    char conductor[10];
};

struct uber_t uberX;
```



- Permite combinar elementos de datos de diferentes tipos.
- 

Struct

Estructuras

```
struct uber_t
{
    int32_t id_carro;
    char color[1];
    char conductor[10];
};

struct uber_t uberX;
```

- La estructura es una estructura de datos utilizada para crear tipos de datos definidos por el usuario.
- Puede contener cualquier número de miembros.
- Los miembros pueden ser de cualquier tipo de datos.
- La estructura no toma ningún almacenamiento de memoria.
- Facilita la organización de datos complicados.

Acceder a miembros

- Se utiliza el punto “.”
- Para acceder a los miembros

```
uberX.color[0] = 'R';
```

```
struct uber_t
{
    int32_t id_carro;
    char color[1];
    char conductor[10];
};

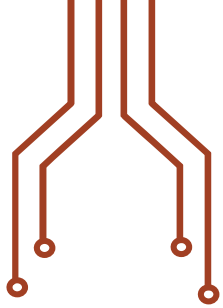
struct uber_t uberX;
```

Acceder a miembros

- Se utiliza el punto “.”
- Para acceder a los miembros

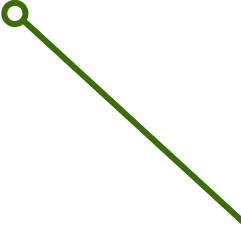
```
uberX = {  
    .id_carro = 152,  
    .color[0] = 'E',  
    .conductor = "pepe juan"  
};
```

```
struct uber_t  
{  
    int32_t id_carro;  
    char color[1];  
    char conductor[10];  
};  
  
struct uber_t uberX;
```

```
typedef struct
{
    int32_t id_carro;
    char color[1];
    char conductor[10];
}uber_t;

uber_t uberX;
```



```
typedef struct{
    ....
}nombre_struct_t;
```

Struct

SizeOf Struct

La estructura no toma ningún almacenamiento de memoria.

```
sizeof(uber_t)
```

```
typedef struct  
{  
    uint16_t id_carro;  
    uint32_t gasolina;  
    char color[1];  
    float decimal;  
} uber_t;
```

Padding

```
typedef struct  
{  
    uint16_t id_carro;  
    uint32_t gasolina;  
    char color;  
    float decimal;  
} uber_t;
```

1 Byte	2 Byte	3 Byte	4 Byte
id_carro			
gasolina			
color			
decimal			

Relleno de estructura

¿Por qué el Padding?

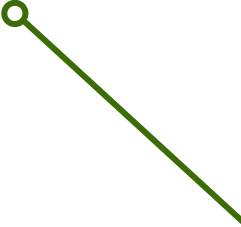
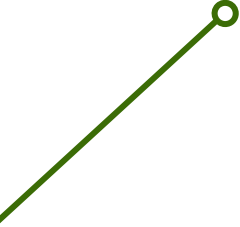
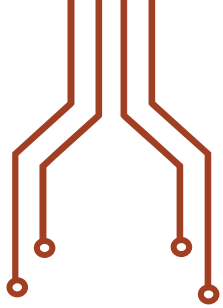
1 Byte	2 Byte	3 Byte	4 Byte
id_carro			
gasolina			
color			
decimal			

- Se hace para minimizar los ciclos de lectura del CPU.
- Ejemplo:
 - Tenemos un procesador de 32 bits.
 - El procesador para acceder a la variable gasolina en dos ciclos de la CPU.

¿Por qué el Padding?

1 Byte	2 Byte	3 Byte	4 Byte
id_carro		gasolina	
gasolina			

- Lo realiza el compilador
- Se hace para minimizar los ciclos de lectura del CPU.
- Ejemplo:
 - Tenemos un procesador de 32 bits.
 - El procesador para acceder a la variable gasolina en dos ciclos de la CPU.



1 Byte	2 Byte	3 Byte	4 Byte
id_carro			
gasolina			
color			
decimal			

Pero podemos obviar el Padding?

Packed

```
typedef struct
{
    uint16_t id_carro;
    uint32_t gasolina;
    char color[1];
    float decimal;
}__attribute__((packed)) uber_t;
```

Usando el atributo se evita el relleno

Packed

```
#pragma pack(1)
```

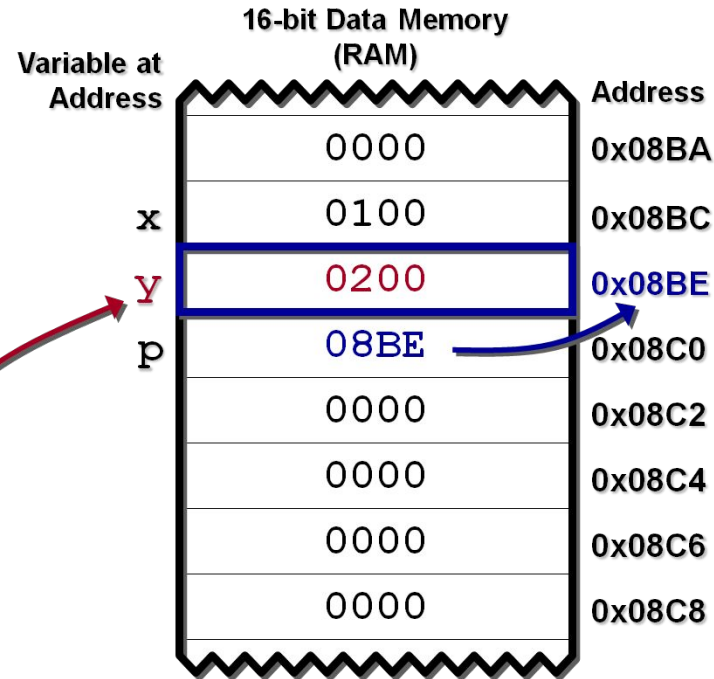
- La directiva **#pragma pack(1)** se utiliza para evitar el relleno de la estructura

Veamos el tamaño de la estructura


```
{  
  int x, y;  
  int *p;
```

```
  x = 0xDEAD;  
  y = 0xBEEF;  
  p = &x;
```

```
  *p = 0x0100;  
  p = &y;  
  *p = 0x0200;  
}
```



Recordando Puntero

Puntero a estructuras

- Se utiliza la flecha “->”
- **-> == (*data).valor**

```
void calculo(valor_x *data)
{
    data->resultado = data->valor*2;
    printf("Funcion: %d \r\n", data->resultado);
}
```



Gracias

@welstheory

hola@welstheory.com

+51 918 899 684

