

# Sesión 08: *Structs* y cadenas enlazadas

Programación para Sistemas

---

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

# Recordatorio

- ¡Dame más memoria!

```
int *enteros = (int *) malloc(N * sizeof(int));  
char *s = (char *) malloc(N * sizeof(char));  
double *reales = (double *) malloc(N * sizeof(double));
```

- ¡Ya no la necesito más!

```
free(enteros);  
free(s);  
free(reales);
```

- malloc en C es como new en Java
- free en C no existe en Java porque en Java es automático

# En el capítulo de hoy...

- *Structs*
- Cadenas enlazadas

# *Structs*

---

*A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. (Structures are called “records” in some languages, notably Pascal.)  
[...]*

*Capítulo 6, K&R*


## struct ii

- Empezamos creando una variable para representar un punto en coordenadas cartesianas enteras

```
struct {  
    int x;  
    int y;  
} a;
```

- El código anterior declara la variable *a*,
- como un registro (*struct*),
- con dos atributos (*members*) *x* e *y* de tipo entero,
- accesibles con la sintaxis *a.x* y *a.y*

# Sintaxis *popular*

 Escribe un programa con dos structs a y b

```
struct {  
    int x;  
    int y;  
} a, b;
```

y explora la sintaxis de **struct**

- Ideas:

```
a.x = 1;  
printf("x == %i\n", a.x);  
sizeof(a)  
b = a;
```

## struct iii

- Si observas con detalle las declaraciones anteriores, la frase

```
struct {int x; int y;}
```

se puede considerar como **un nuevo tipo** que se puede declarar con una **etiqueta (tag)** de esta forma

```
struct punto {  
    int x;  
    int y;  
};
```

- Ahora **la etiqueta punto** nos permite declarar variables así:  
**struct punto** a, b;



- Por supuesto, es posible declarar **structs de structs** y **arrays de structs**


```
struct rectangulo {  
    struct punto so;  
    struct punto ne;  
};
```

```
struct rectangulo r;  
struct punto h[6];
```

# Cadenas enlazadas

---

# Punteros a *structs*

 ¿Cómo lo hacemos?

# Punteros a *structs*

💬 ¿Cómo lo hacemos?

💬 Dibujar la ejecución:

```
struct rectangulo *rectp;
```

# Punteros a *structs*

💬 ¿Cómo lo hacemos?

💬 Dibujar la ejecución:

```
struct rectangulo *rectp;
```

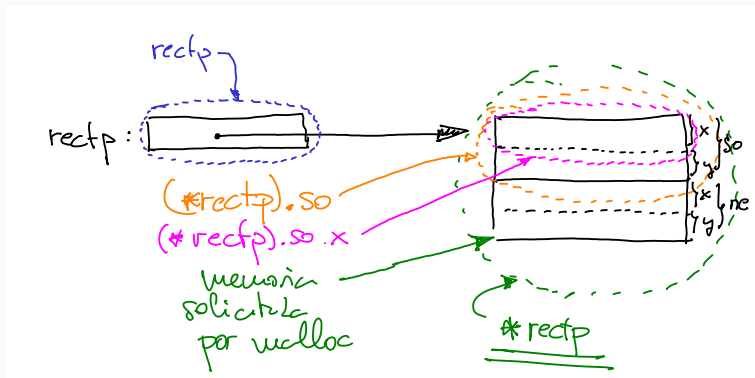
```
rectp =
```

```
    (struct rectangulo *)
```

```
    malloc(sizeof(struct rectangulo));
```

# Solución

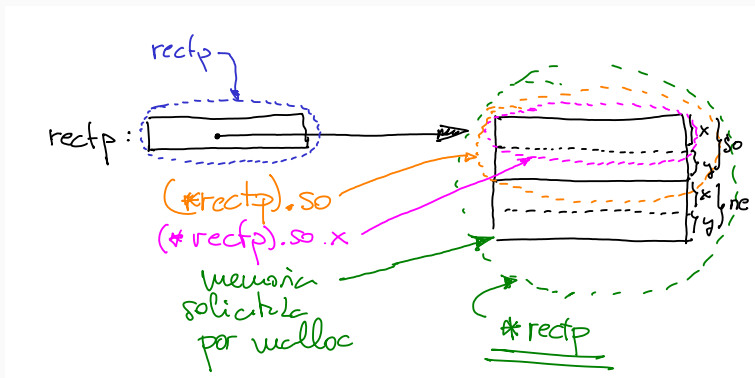
- Deberías haber dibujado algo parecido a esto:



¿Qué significa `*rectp.so`? ¿por qué es incorrecto?

# Solución

- Deberías haber dibujado algo parecido a esto:



¿Qué significa `*rectp.so`? ¿por qué es incorrecto?

`(*rectp).so = rectp->so`

# Uso masivo en las bibliotecas estándares

```
$ man fopen
```

```
FOPEN(3)          Linux Programmer's Manual          FOPEN(3)
```

```
NAME
```

```
    fopen, fdopen, freopen - stream open functions
```

```
SYNOPSIS
```

```
    #include <stdio.h>
```

```
    FILE *fopen(const char *pathname, const char *mode);
```

```
    ...
```

```
$ |
```



Interpreta esas líneas de la página del manual:

FILE es internamente un tipo *struct*



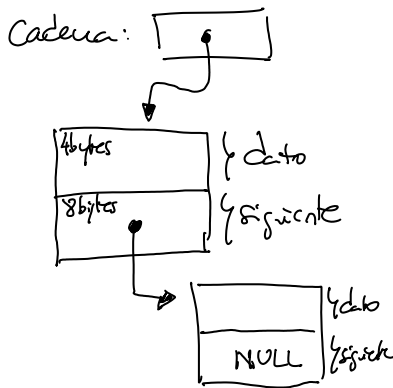
# Cadenas enlazadas: el tipo

```
struct nodo {  
    int dato;  
    struct nodo *siguiente;  
};
```

# Cadenas enlazadas: el tipo

```
struct nodo {  
    int dato;  
    struct nodo *siguiente;  
};
```

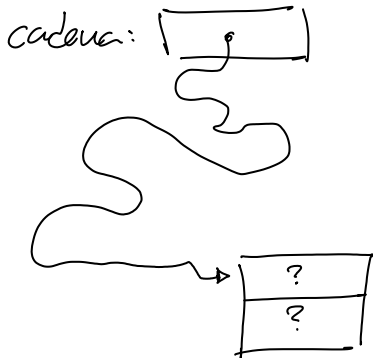
```
struct nodo *cadena;
```



# Cadenas enlazadas: vacía

```
#include <stdlib.h>
```

```
struct nodo *cadena;
```

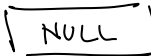


# Cadenas enlazadas: vacía

```
#include <stdlib.h>
```

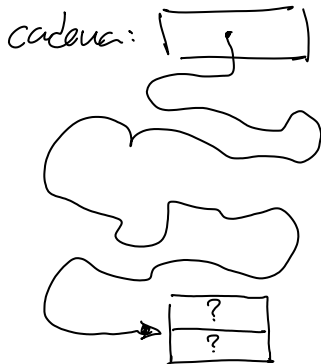
```
struct nodo *cadena;
```

```
cadena = NULL;
```

cadena: A hand-drawn diagram showing a pointer variable 'cadena' pointing to a rectangular box. Inside the box, the word 'NULL' is written. The box has a double-line border, with the top and bottom lines having arrowheads pointing to the right, indicating a pointer to the next node in a linked list.

# Cadenas enlazadas: un elemento

```
struct nodo *cadena;
```



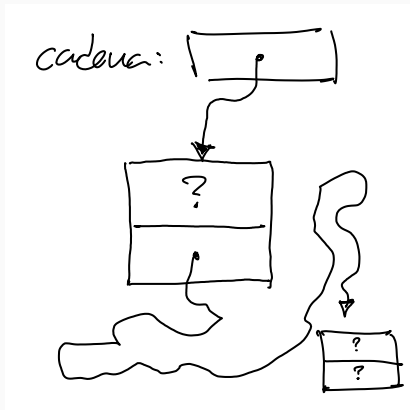
# Cadenas enlazadas: un elemento

```
struct nodo *cadena;
```

```
cadena =
```

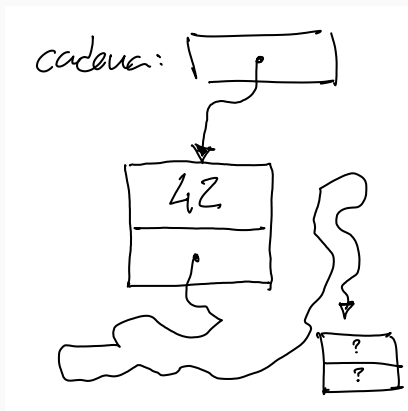
```
(struct nodo *)
```

```
malloc(sizeof(struct nodo));
```



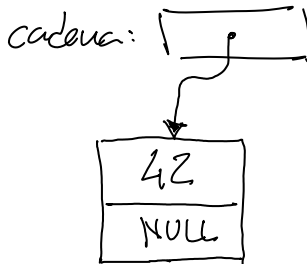
# Cadenas enlazadas: un elemento

```
struct nodo *cadena;  
cadena =  
    (struct nodo *)  
    malloc(sizeof(struct nodo));  
cadena->dato = 42;
```



# Cadenas enlazadas: un elemento

```
struct nodo *cadena;  
cadena =  
    (struct nodo *)  
    malloc(sizeof(struct nodo));  
cadena->dato = 42;  
cadena->siguiente = NULL;
```





# Cadenas enlazadas: primero y último

- Expresión que representa el primero:

cadena->dato

- Recorrido hasta el último:

```
struct nodo *ultimo;  
ultimo = cadena;  
while (ultimo->siguiente != NULL) {  
    ultimo = ultimo->siguiente;  
}
```



Dibujar

# Cadenas enlazadas: añadir al principio

```
struct nodo *primero;  
primero = (struct nodo*)malloc(sizeof(struct nodo));  
primero->dato = nuevo;  
primero->siguiente = cadena;  
cadena = primero;
```



Dibujar

# Cadenas enlazadas: añadir al final

```
ultimo = cadena;  
while (ultimo->siguiente != NULL) {  
    ultimo = ultimo->siguiente;  
}  
ultimo->siguiente =  
    (struct nodo*)malloc(sizeof(struct nodo));  
ultimo = ultimo->siguiente;  
ultimo->dato = nuevo;  
ultimo->siguiente = NULL;
```



Dibujar

# Cadenas enlazadas: borrar el primero

```
cadena = cadena->siguiente;
```



Dibujar ¿Algún problema?

# Cadenas enlazadas: borrar el primero

```
cadena = cadena->siguiente;
```



Dibujar ¿Algún problema?

*¡Memory leak!*  
*¿Solución?*

# Cadenas enlazadas: borrar el primero

```
primero = cadena;  
cadena = cadena->siguiente;  
free(primero);
```

 Dibujar ¿Algún problema?

*¡Memory leak!*  
*¿Solución?*

# Cadenas enlazadas: borrar el último

```
penultimo = cadena;  
while (penultimo->siguiente->siguiente != NULL) {  
    penultimo = penultimo->siguiente;  
}  
ultimo = penultimo->siguiente;  
penultimo->siguiente = NULL;  
free(ultimo);
```



Dibujar