



UNRN

Universidad Nacional
de Río Negro



| unrionegro

Estructuras II

UNRN

Universidad Nacional
de Río Negro

13

2023



punteros a estructuras

Dada esta estructura, como implementamos 'crear'

```
struct persona
{
    char nombre*;
    int edad;
}

struct persona* crear(char nombre[], int edad)
```

¡A la terminal!

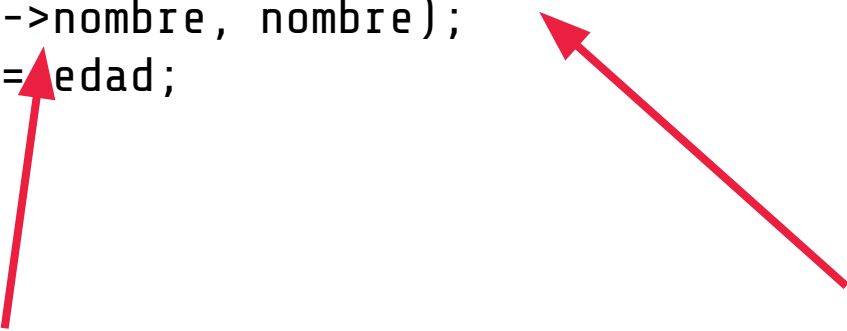


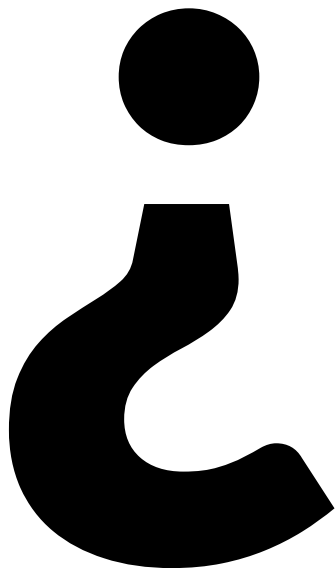
Resuelto

```
persona_t* crear(char nombre[], int edad)
{
    persona_t* nuevo = malloc(sizeof(persona_t));
    if (nuevo != NULL)
    {
        nuevo->nombre = malloc(sizeof(char)*strlen(nombre)+1);
        strcpy(nuevo->nombre, nombre);
        nuevo->edad = edad;
    }
    return nuevo;
}
```

¿Que son estas dos cosas?

```
persona_t* crear(char nombre[], int edad)
{
    persona_t* nuevo = malloc(sizeof(persona_t));
    if (nuevo != NULL)
    {
        nuevo->nombre = malloc(sizeof(char)*strlen(nombre)+1);
        strcpy(nuevo->nombre, nombre);
        nuevo->edad = edad;
    }
    return nuevo;
}
```

Two red arrows are present. One arrow points from the bottom left towards the variable 'edad' in the line 'nuevo->edad = edad;'. The other arrow points from the bottom right towards the 'strcpy' function call in the line 'strcpy(nuevo->nombre, nombre);'.



Que es la
->



**t_persona* usr;
usr->nombre
es igual a
(*usr).nombre**

La flecha desreferencia el puntero



Que falta ahora



Para liberar al mismo nivel

```
void destruir(struct persona* persona)
```

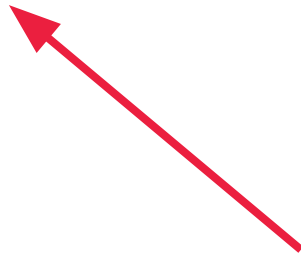
¡A la terminal!



```
void destruire (persona_t* persona)
{
    free(persona->nombre);
    free(personal);
}
```

¿Por que en este orden?

```
void destruir (persona_t* persona)
{
    free(persona->nombre);
    free(personal);
}
```





**Y lo que está en el
medio**

Operaciones interesantes

```
int modificar_edad(persona_t* persona, int nueva_edad)
int comparar_por_edad(persona_t* p1, persona_t* p2)
```

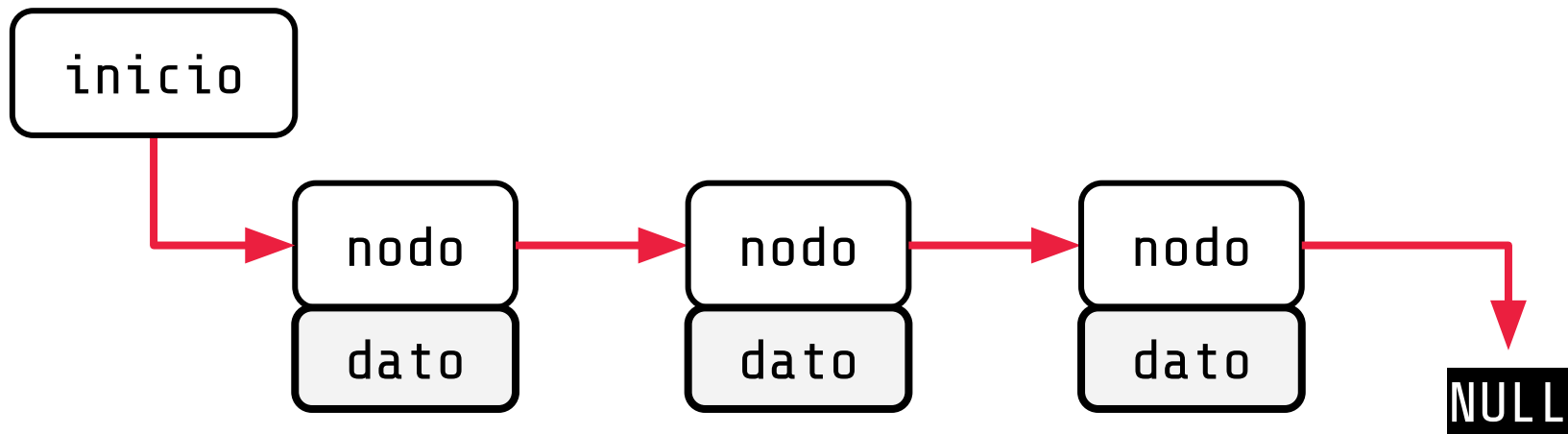
y más



¿Preguntas?

Introducción a estructuras de datos

Lista enlazada

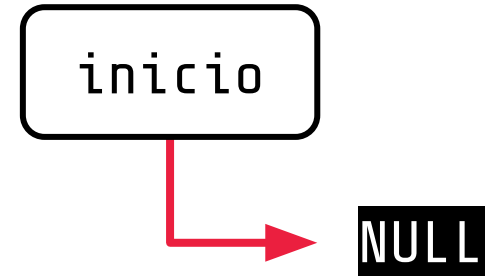


Dada esta estructura

```
typedef struct nodo
{
    struct nodo* siguiente;
    int valor;
}nodo_t;
```

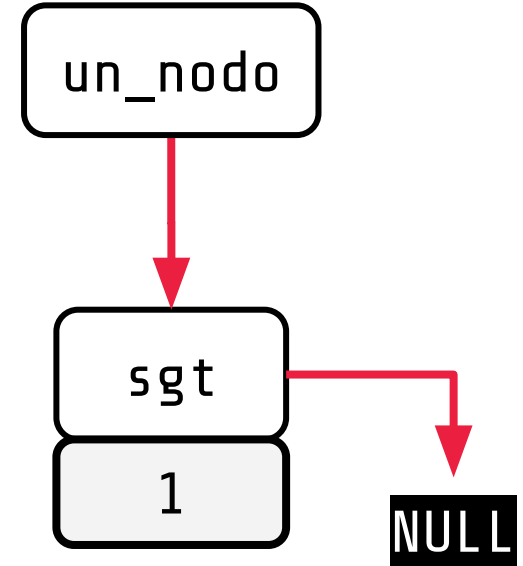
La variable inicio

```
nodo_t inicio = NULL;
```



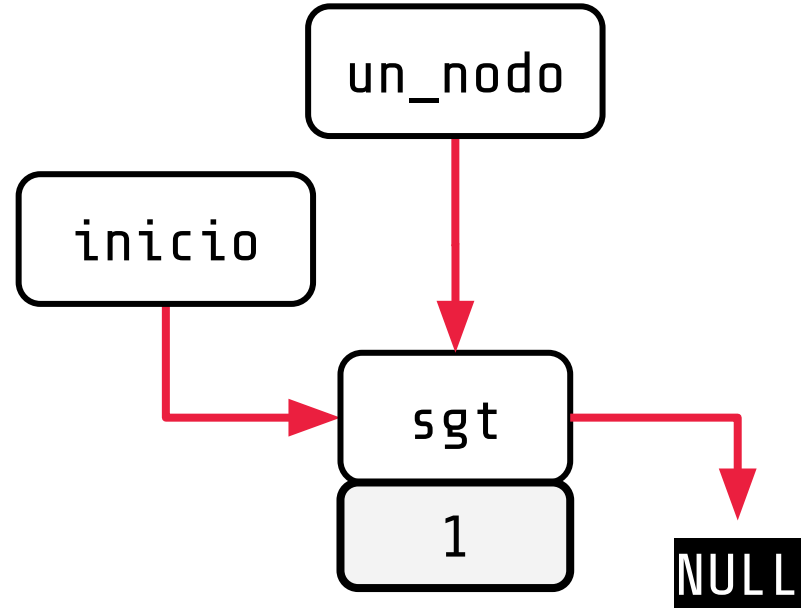
Creamos un nodo_t

```
nodo_t* un_nodo = malloc(sizeof(nodo_t));  
un_nodo->siguiente = NULL;  
otro_nodo->dato = 1;
```



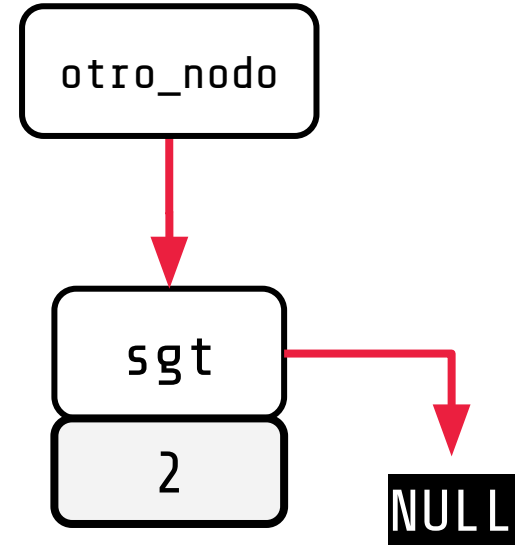
Y lo conectamos a la lista

```
inicio = un_nodo;
```

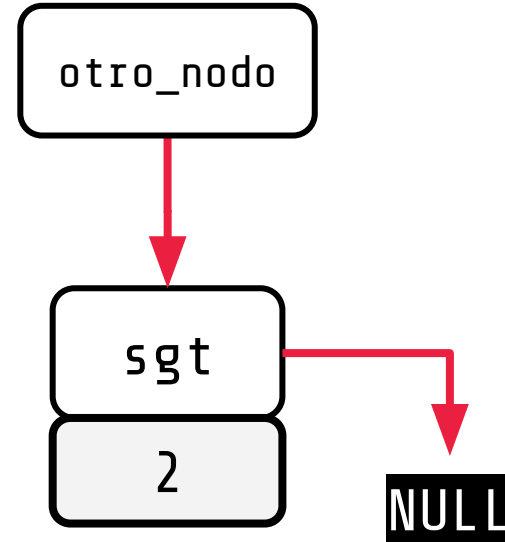
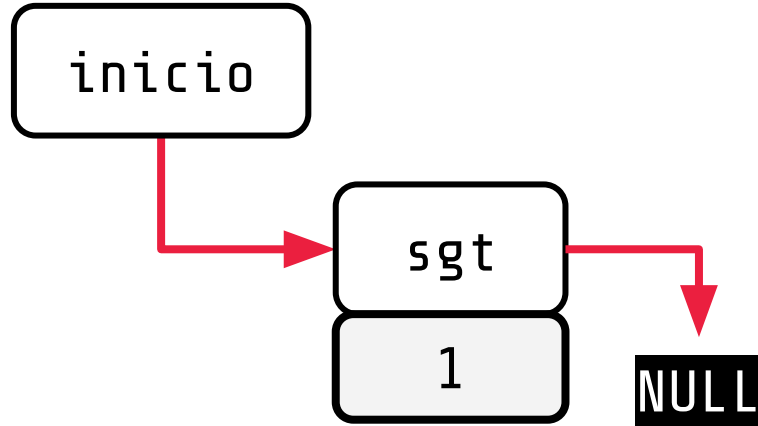


Creamos *otro* nodo_t

```
nodo_t* otro_nodo = malloc(sizeof(nodo_t));  
otro_nodo->siguiente = NULL;  
otro_nodo->dato = 2;
```



¿Y como lo conectamos al final lista?

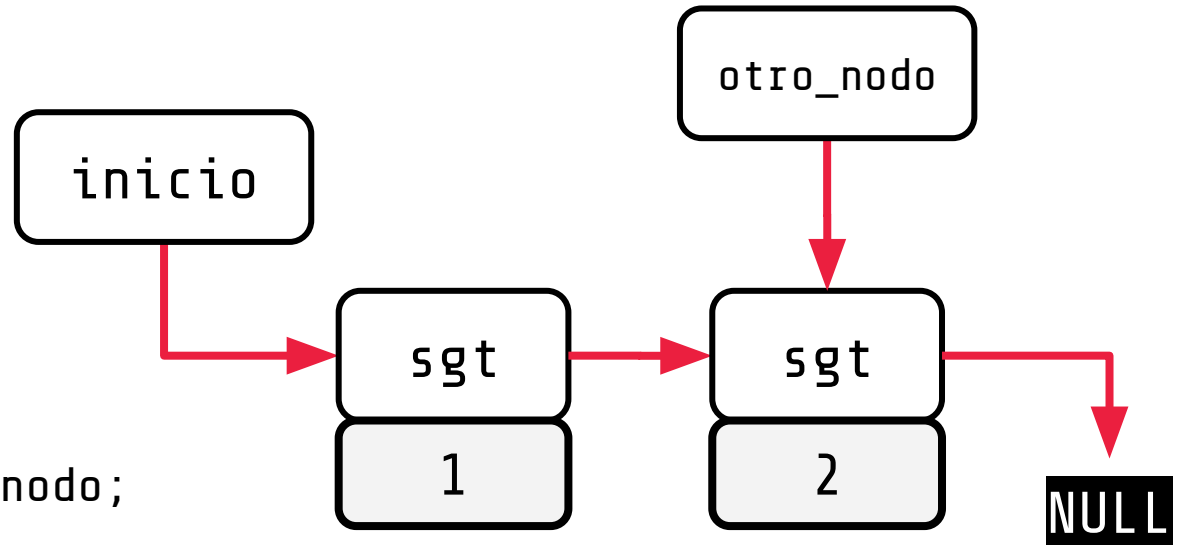


Para avanzar en la lista

```
ptr = inicio;  
mientras (ptr->siguiente != NULL)  
{  
    ptr = ptr->siguiente;  
}
```

Al final

```
ptr->siguiente = otro_nodo;
```

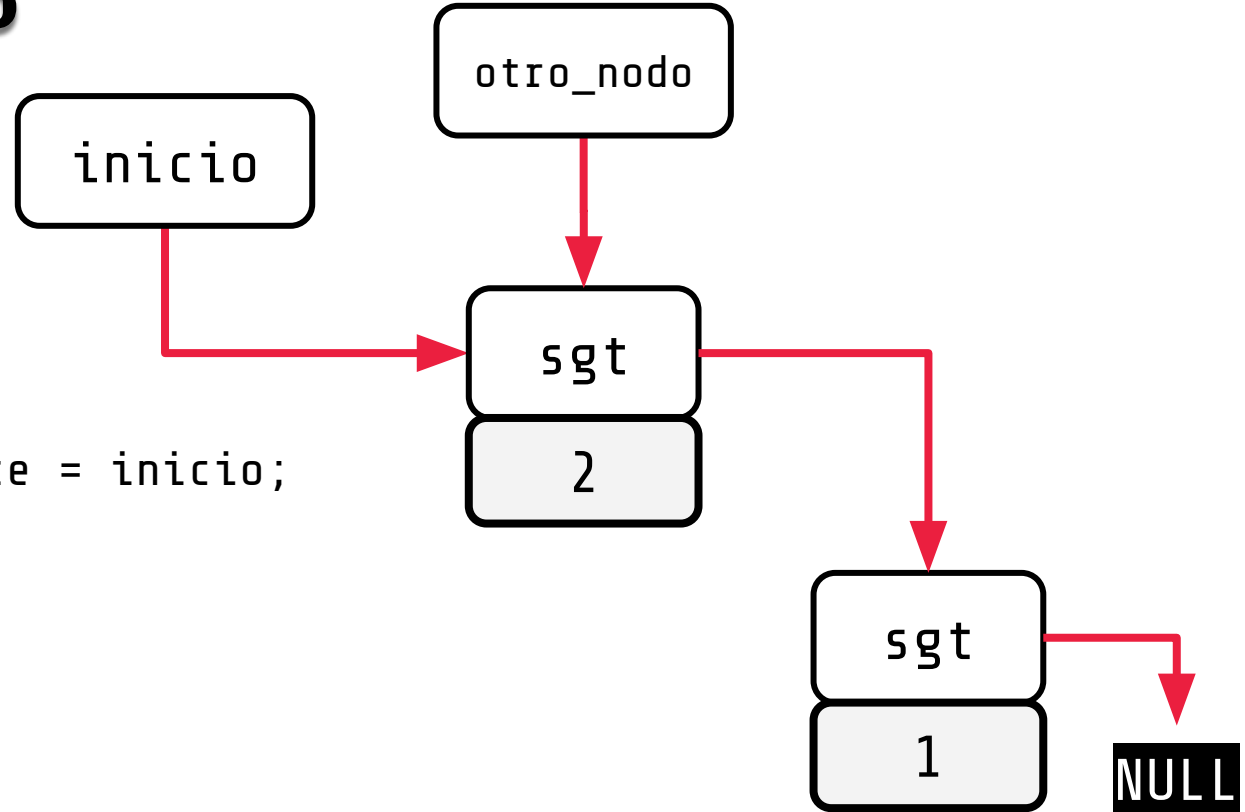




¿Preguntas?

O al principio

```
otro_nodo->siguiente = inicio;  
inicio = otro_nodo;
```




¿Para que estan?

Comparado con un arreglo

0

1	2	3	4	5	6	7	8		
---	---	---	---	---	---	---	---	--	--



0	1	2	3	4	5	6	7	8	
---	---	---	---	---	---	---	---	---	--

Qué es necesario 'hacer lugar'

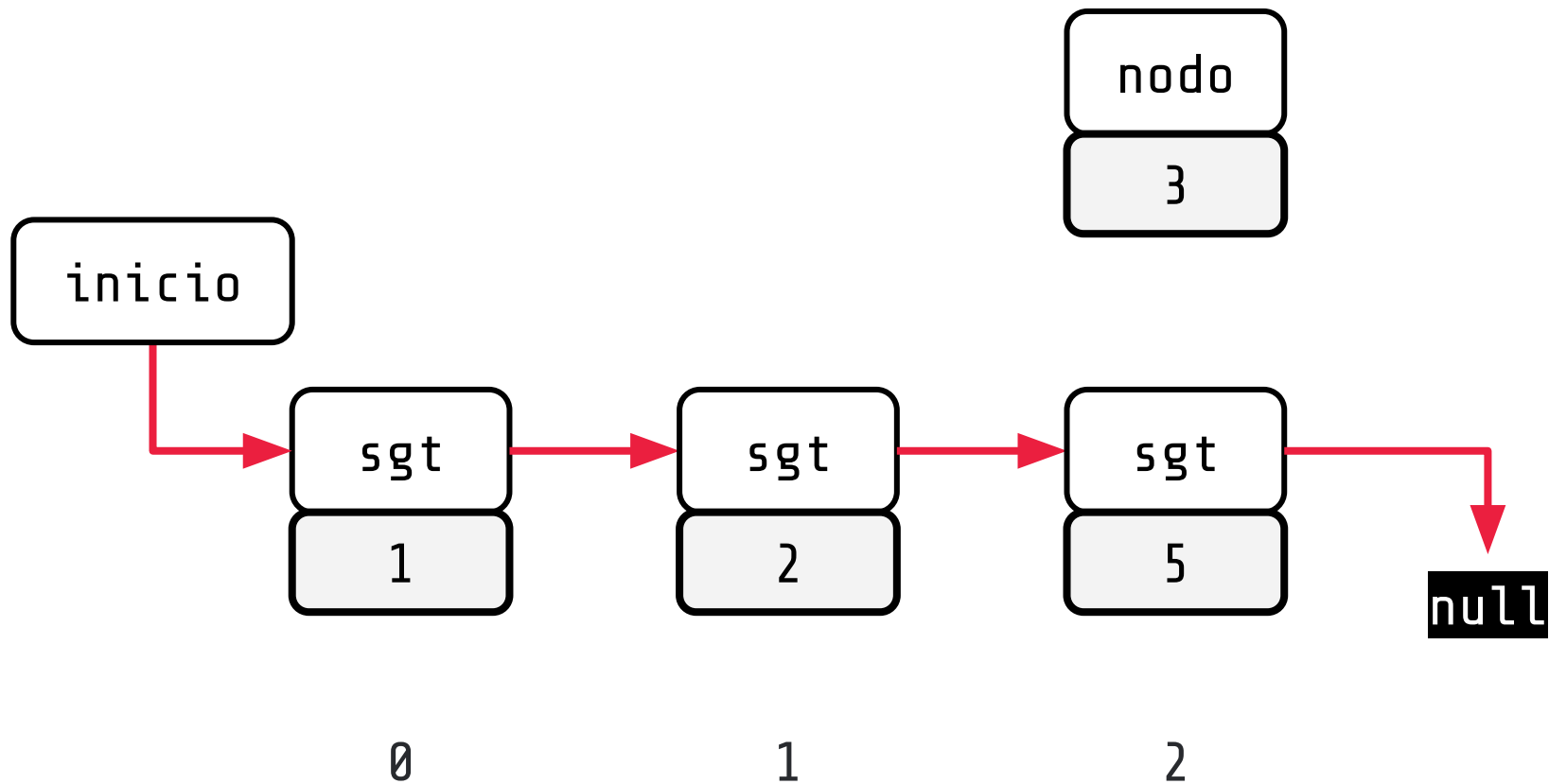


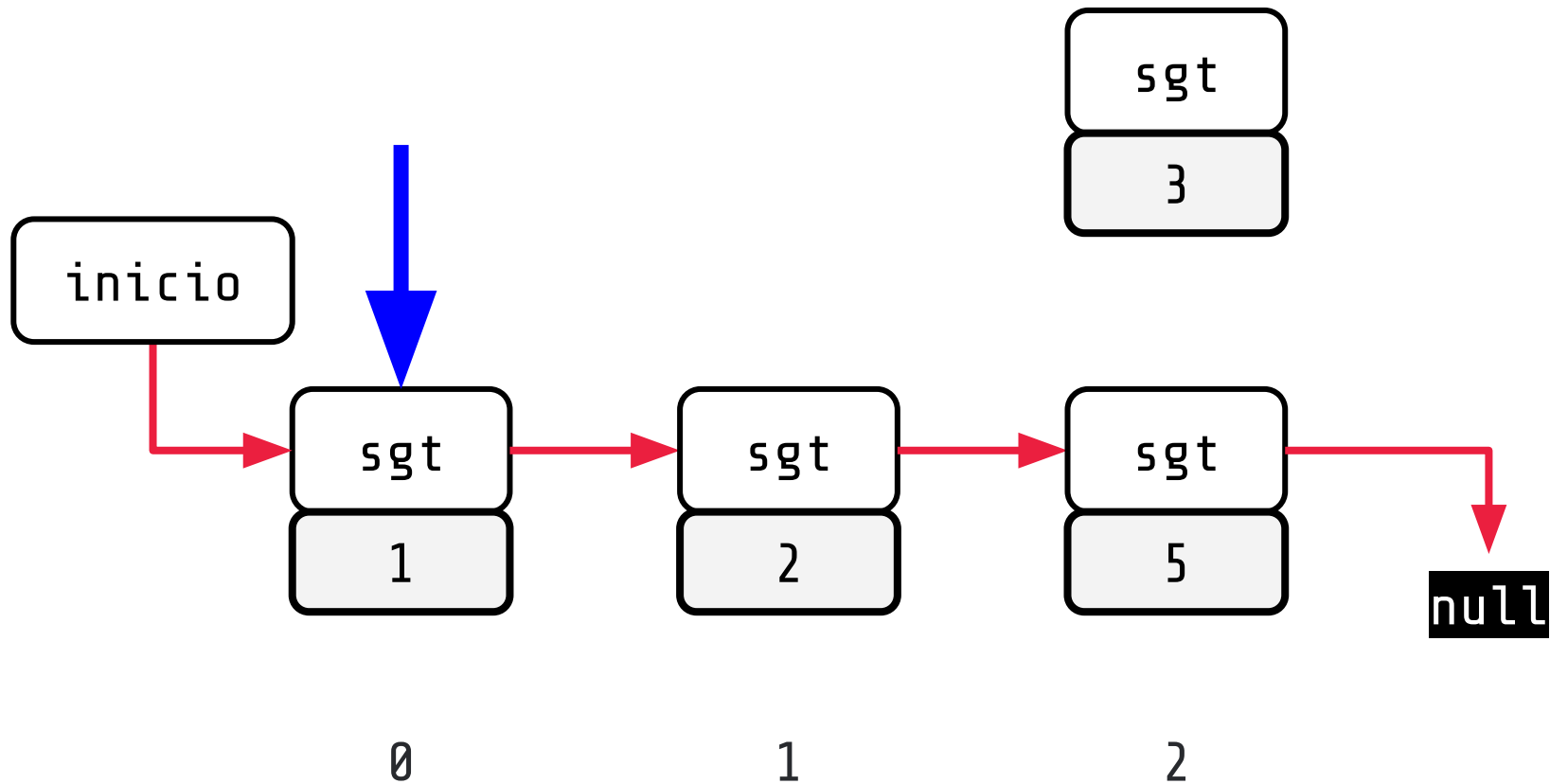
**El detalle lo
veremos un poco
mas adelante**

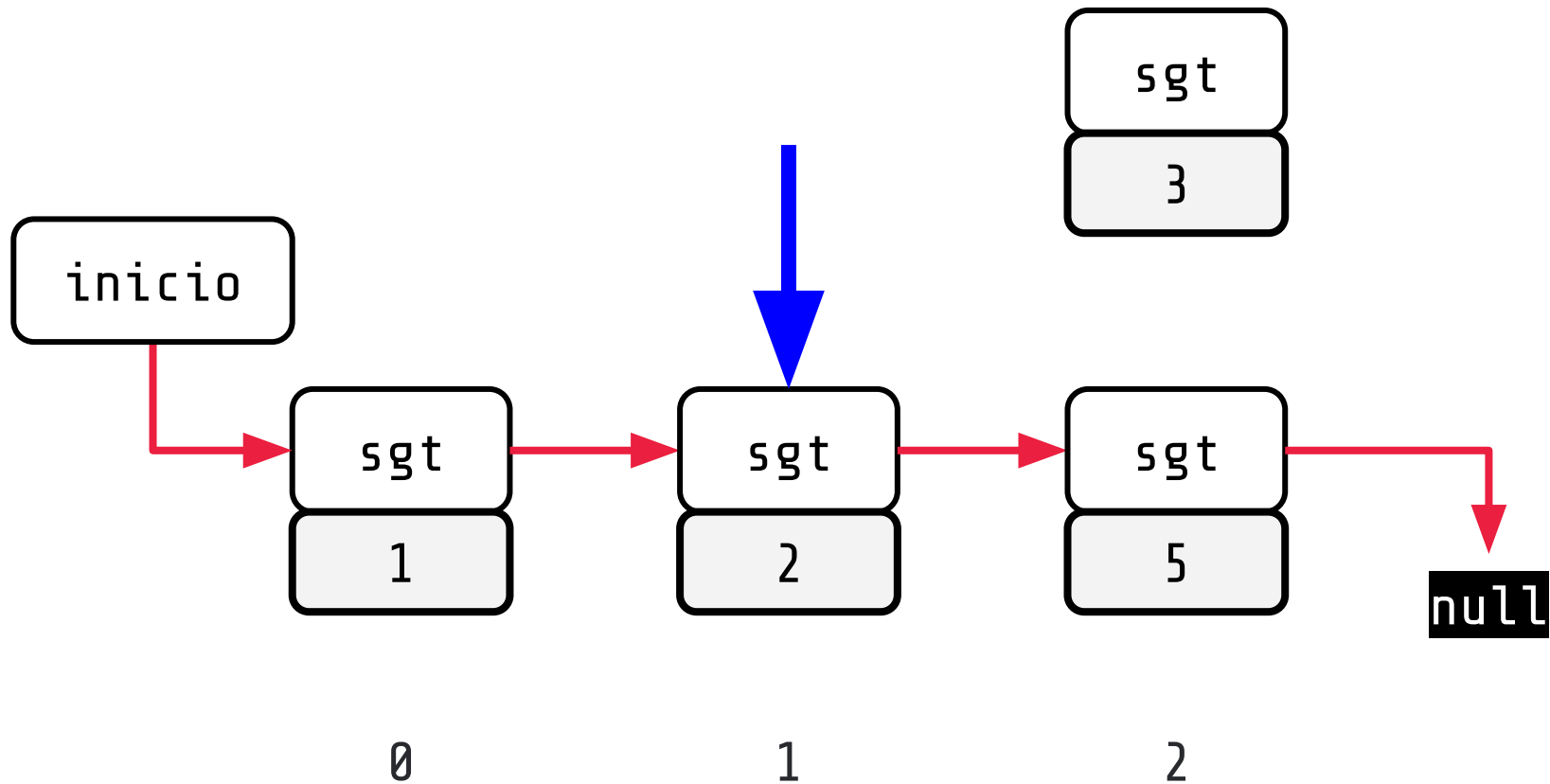


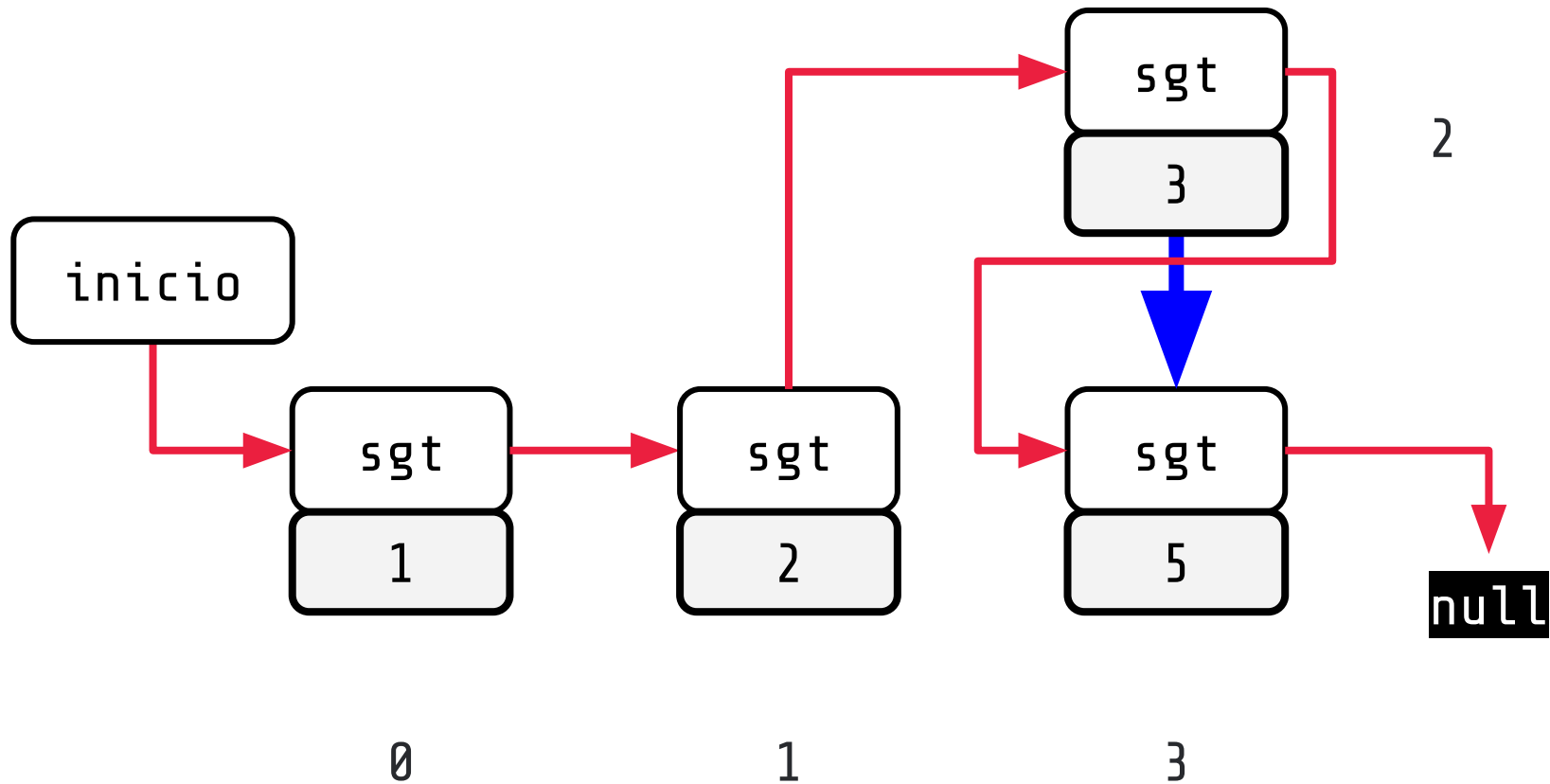
¿Preguntas?

Una inserción en una lista mas grande









Al igual que los arreglos

insercion
eliminación
modificacion
obtencion

Además, por qué no meter algo como

```
typedef struct nodo
{
    struct nodo* siguiente;
    int valor;
}nodo_t;
```

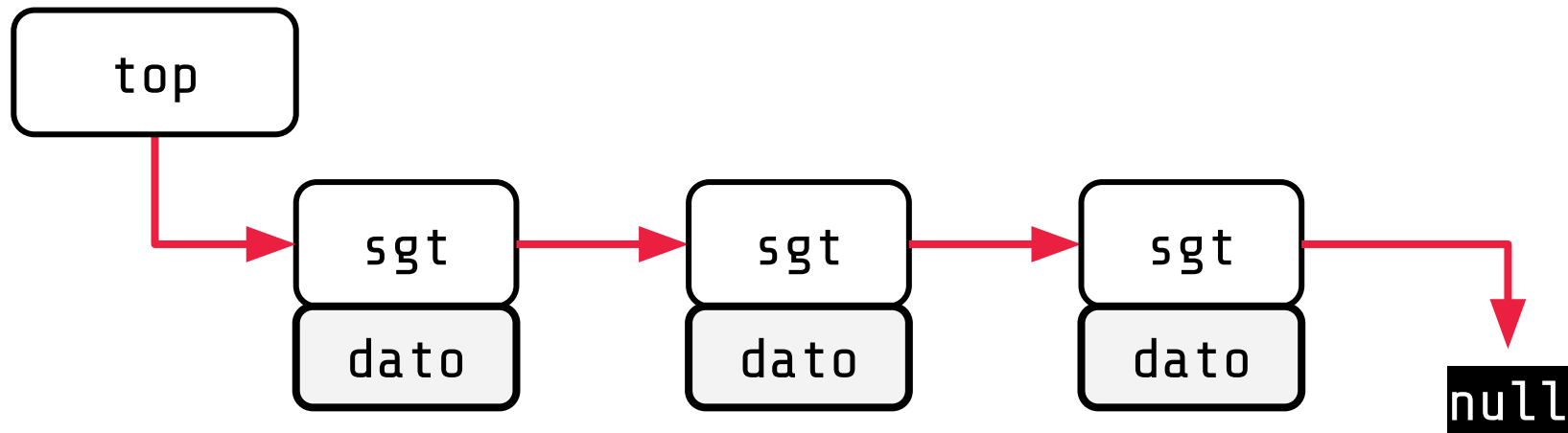
Algo mas genérico

```
typedef struct nodo
{
    struct nodo* siguiente;
    void* datos;
    int largo;
}nodo_t;
```

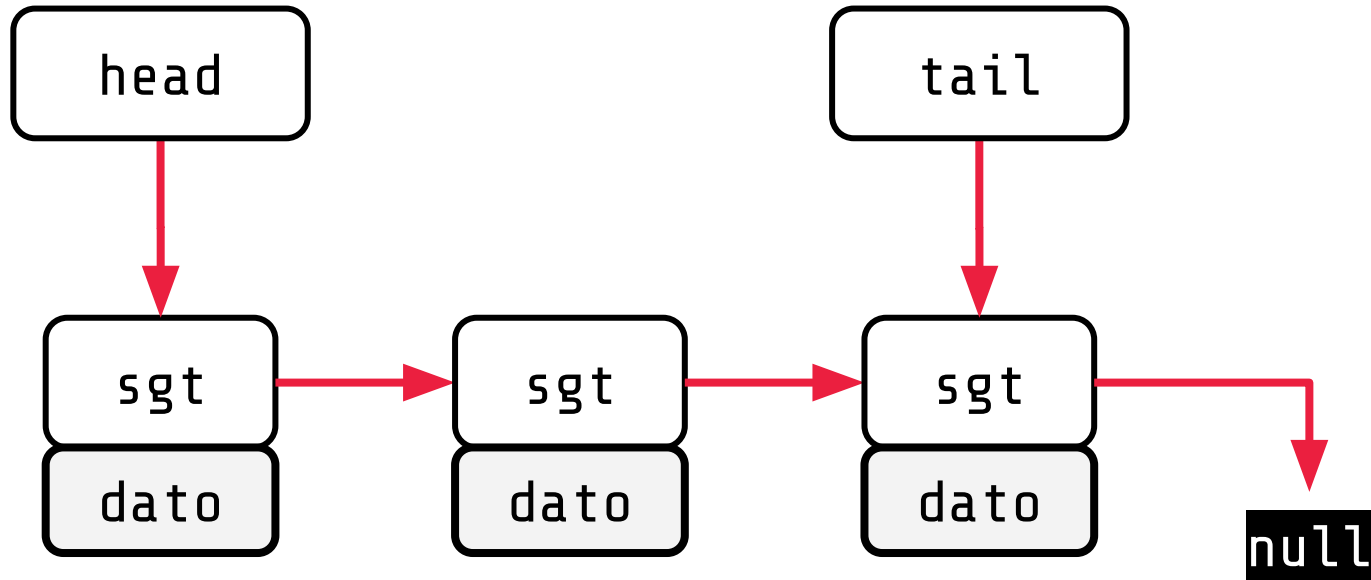


Son la base de otras estructuras

Pila (stack)



Cola (queue)

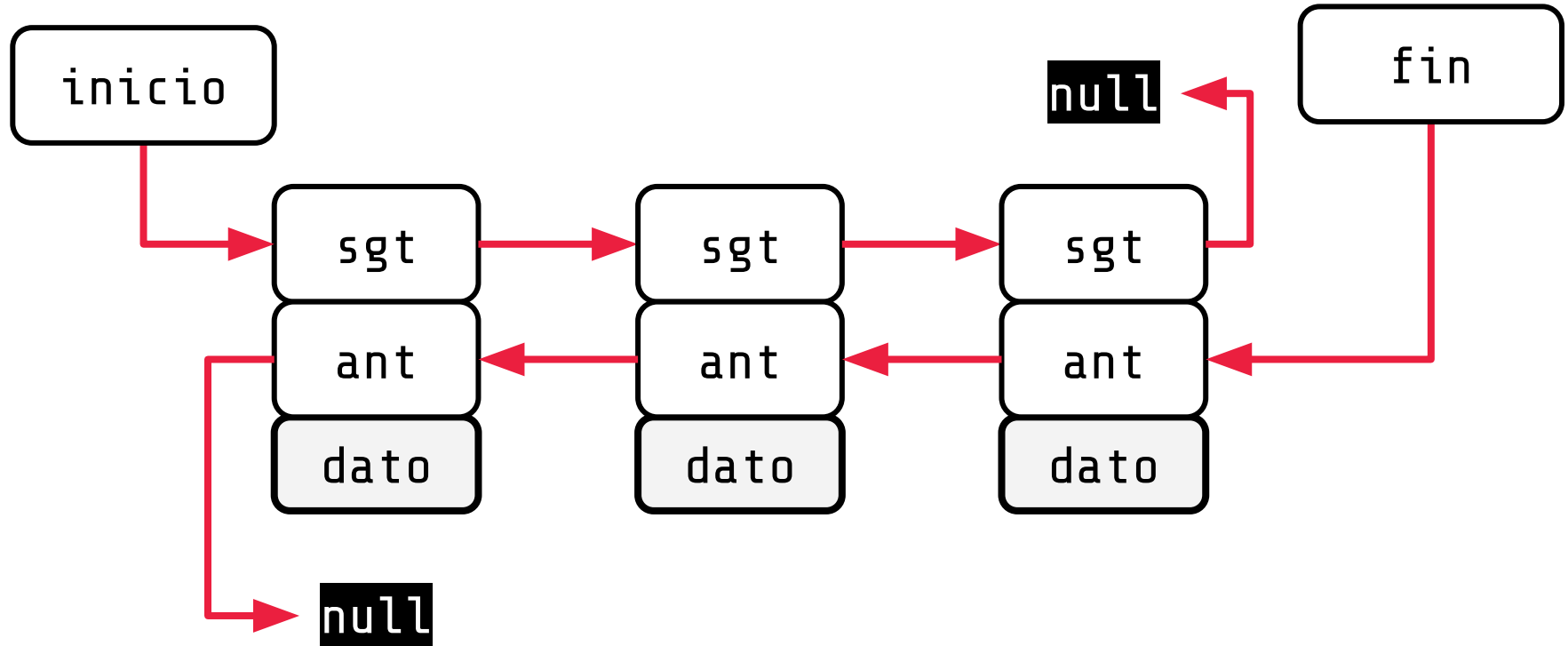


**Pero tambien otras
mas avanzadas**

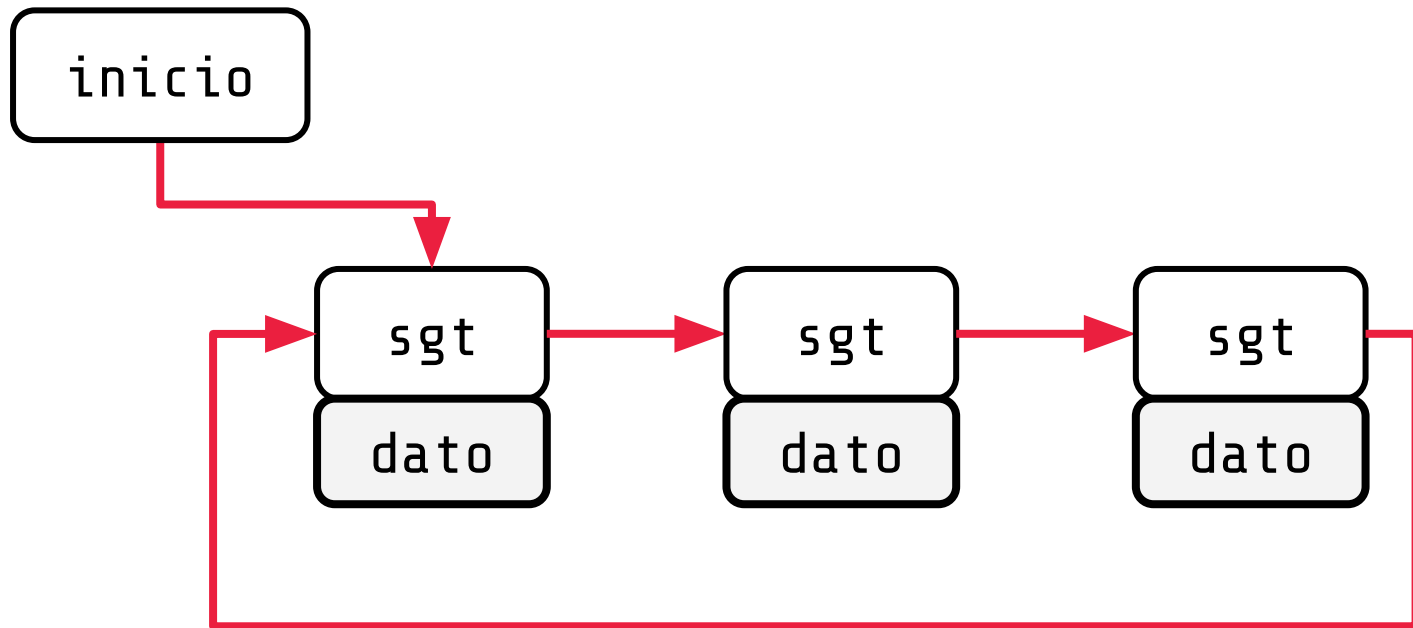
Dada esta estructura

```
typedef struct nodo
{
    struct nodo* siguiente;
    struct nodo* anterior;
    int valor;
}nodo_t;
```

Listas doblemente enlazadas



Lista enlazada circular



Lista enlazada doble circular

