

Listas enlazadas

Programación en C

Vectores

Reservamos memoria para una cantidad determinada de datos.



Vectores

Reservamos memoria para una cantidad determinada de datos.

Estáticos:

→ especificamos su tamaño en tiempo de compilación.
`int vector[100];`

Dinámicos:

→ especificamos su tamaño en tiempo de ejecución.
`int * vector, n;`
`scanf("%d", &n);`
`vector = (int*)malloc (n* sizeof(int));`



Listas enlazadas

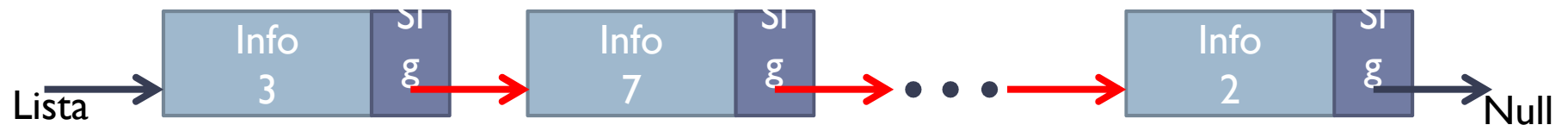
Reservamos espacio para un nuevo elemento a medida que se va necesitando.



Listas enlazadas

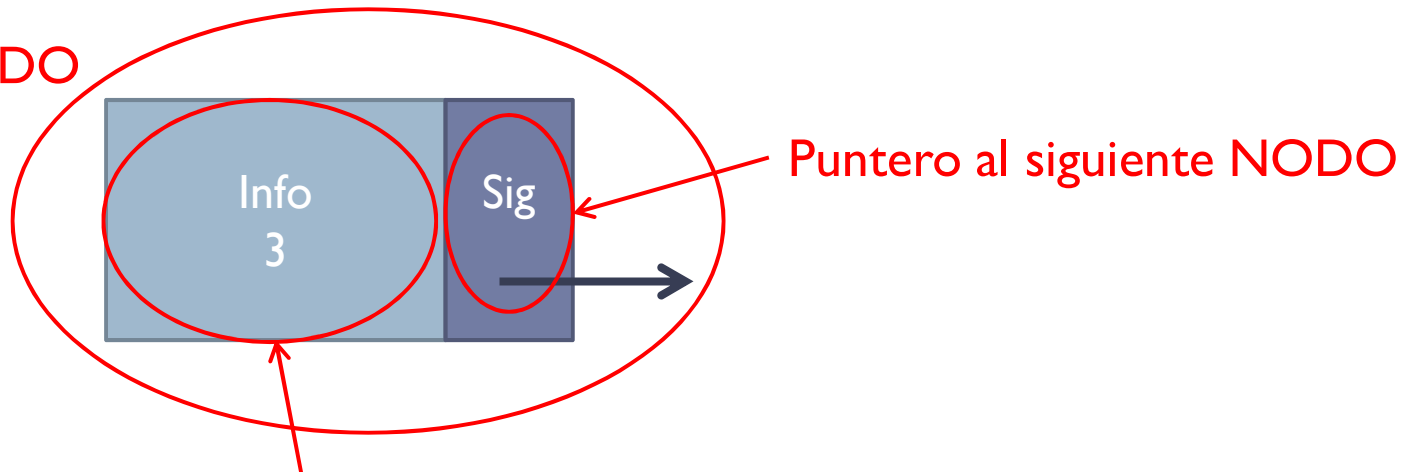
Reservamos espacio para un nuevo elemento a medida que se va necesitando.

Para insertar un elemento en la lista necesitamos unirlos mediante un puntero.



Listas enlazadas

NODO



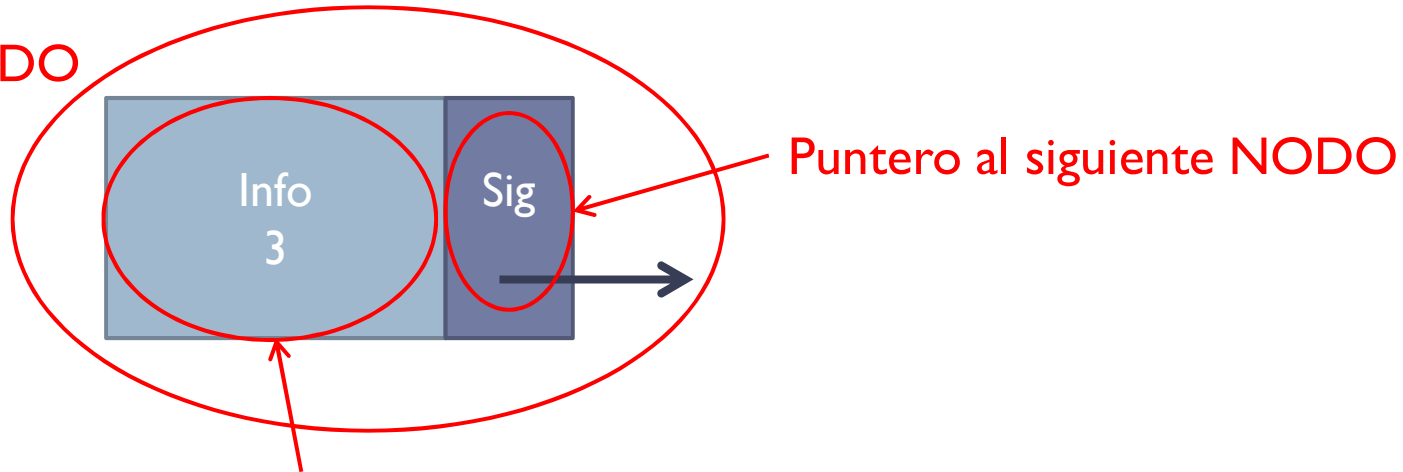
Elemento del tipo de datos de la lista

```
struct Nodo{  
    int Info;  
    struct Nodo * Sig;  
};
```



Listas enlazadas

NODO



Elemento del tipo de datos de la lista

¡Recursividad!

```
struct Nodo{  
    int Info;  
    struct Nodo * Sig;  
};
```



Listas enlazadas

Lista vacía:

```
int main()
{
    struct Nodo * lista = NULL;
    ...
}
```

lista → Null

lista es una variable de tipo "Puntero a struct"



Listas enlazadas

Lista vacía:

```
int main()
{
    struct Nodo * lista = NULL;
    ...
}
```

lista → Null

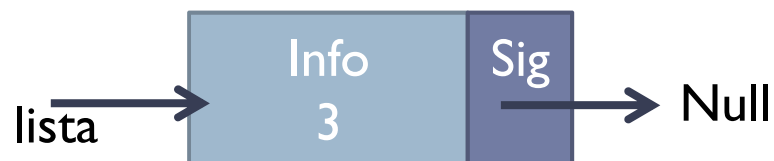
Inicialmente es un
puntero a NULL



Listas enlazadas

Añadir 1º nodo a la lista:

```
int main()
{
    struct Nodo * lista = NULL;
    lista = (struct Nodo*)malloc (sizeof(struct Nodo));
    lista->Info = 3;
    lista->Sig = NULL;
    ...
}
```

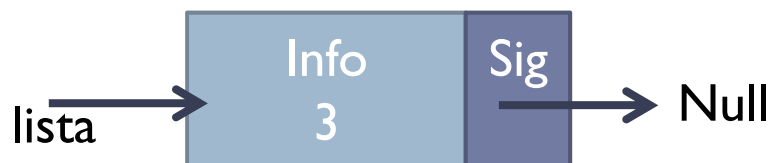


Reservamos espacio para el nuevo elemento

Listas enlazadas

Añadir 1º nodo a la lista:

```
int main()
{
    struct Nodo * lista = NULL;
    lista = (struct Nodo*)malloc (sizeof(struct Nodo));
    lista->Info = 3;
    lista->Sig = NULL;
    ...
}
```

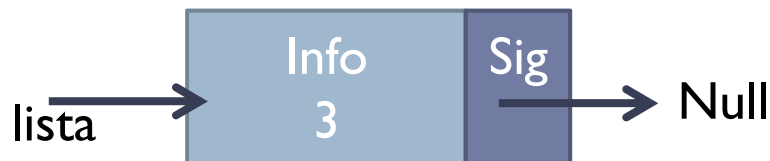


Rellenamos la información
del elemento

Listas enlazadas

Añadir 1º nodo a la lista:

```
int main()
{
    struct Nodo * lista = NULL;
    lista = (struct Nodo*)malloc (sizeof(struct Nodo));
    lista->Info = 3;
    lista->Sig = NULL;
    ...
}
```

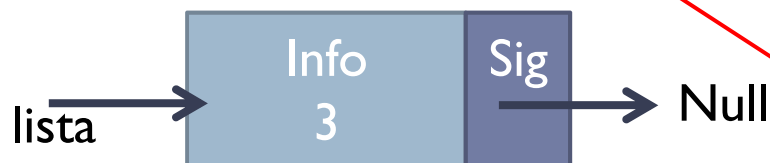


Ahora es “el último elemento”
el que apunta a NULL

Listas enlazadas

Añadir 1º nodo a la lista:

```
int main()
{
    struct Nodo * lista = NULL;
    lista = (struct Nodo*)malloc (sizeof(struct Nodo));
    lista->Info = 3;
    lista->Sig = NULL;
    ...
}
```

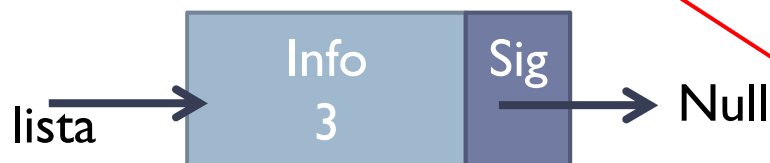


“lista” es un puntero, así que para acceder a la información:
 $*(lista).Info = 3;$ \leftrightarrow $lista->Info = 3;$

Listas enlazadas

Añadir 1º nodo a la lista:

```
int main()
{
    struct Nodo * lista = NULL;
    lista = (struct Nodo*)malloc (sizeof(struct Nodo));
    lista->Info = 3;
    lista->Sig = NULL;
    ...
}
```

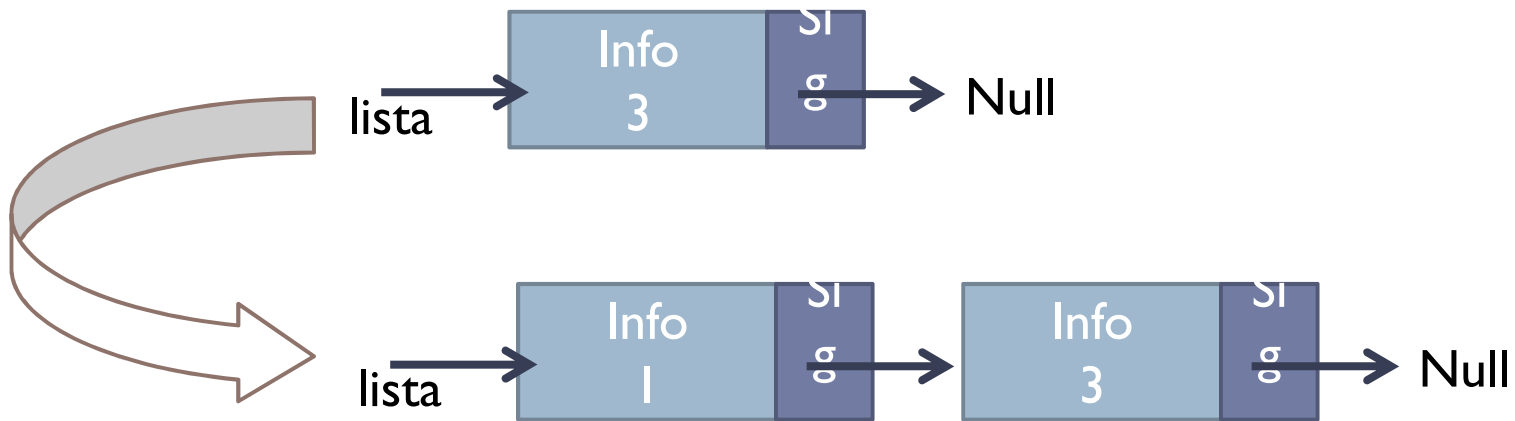


“lista” es un puntero, así que para acceder a la información:
 $*(lista).Info = 3;$ \leftrightarrow $lista->Info = 3;$

Operador directo “.”
Operador indirecto “->”

Listas enlazadas

Añadir nodo a la lista (POR DELANTE):



Listas enlazadas

Añadir nodo a la lista (POR DELANTE):

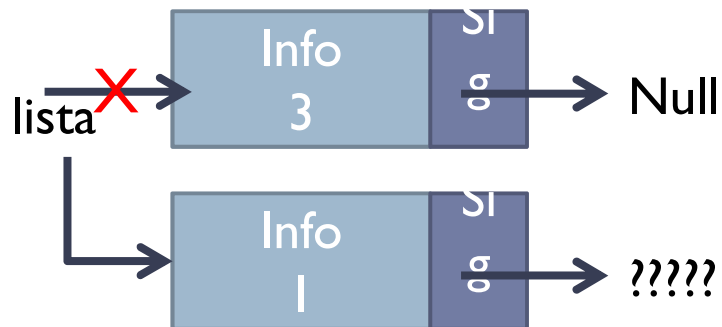
```
int main()
{
    struct Nodo * lista = NULL;
    . . .
    lista = (struct Nodo*) malloc (sizeof(struct Nodo));
    lista->Info = 1;
    lista->Sig = ???;
    . . .
}
```



Listas enlazadas

Añadir nodo a la lista (POR DELANTE):

```
int main()
{
    struct Nodo * lista = NULL;
    . . .
    lista = (struct Nodo*) malloc (sizeof(struct Nodo));
    lista->Info = 1;
    lista->Sig = ???;
    . . .
}
```

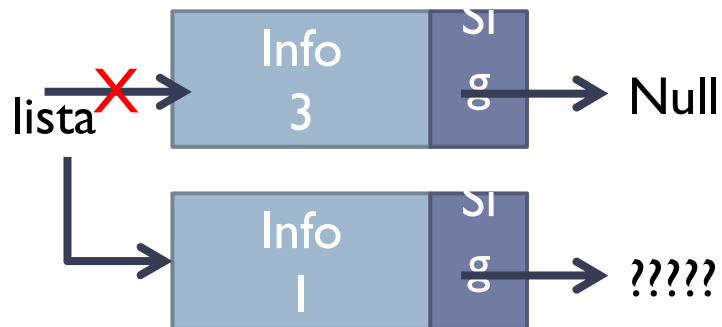


- Creamos un nuevo nodo.
- Rellenamos la información.
- Pero...

Listas enlazadas

Añadir nodo a la lista (POR DELANTE):

```
int main()
{
    struct Nodo * lista = NULL;
    . . .
    lista = (struct Nodo*) malloc (sizeof(struct Nodo));
    lista->Info = 1;
    lista->Sig = ????;
    . . .
}
```

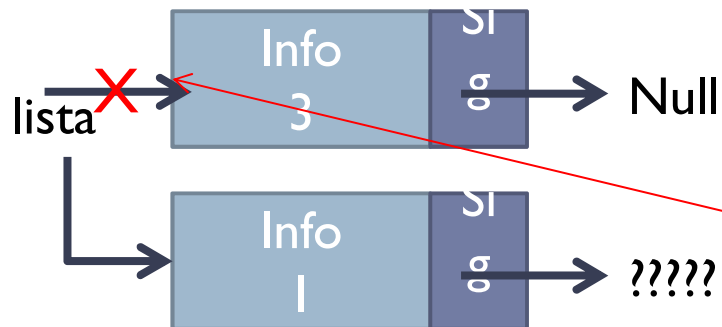


Dónde asignamos
esta referencia??

Listas enlazadas

Añadir nodo a la lista (POR DELANTE):

```
int main()
{
    struct Nodo * lista = NULL;
    . . .
    lista = (struct Nodo*) malloc (sizeof(struct Nodo));
    lista->Info = 1;
    lista->Sig = ???;
    . . .
}
```



Hemos perdido
esta referencia!!

Listas enlazadas

Añadir nodo a la lista (POR DELANTE):

```
int main()
{
    struct Nodo * lista = NULL;
    struct Nodo * aux;
    . . .
    aux = lista;
    lista = (struct Nodo*)malloc (sizeof(struct Nodo));
    lista->Info = 1;
    lista->Sig = aux;
    . . .
}
```

Usamos una
variable auxiliar

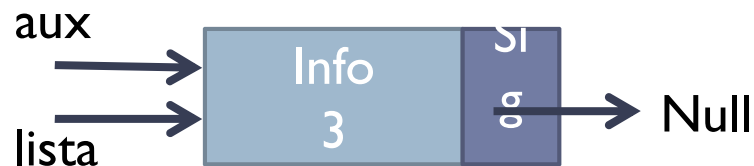


Listas enlazadas

Añadir nodo a la lista (POR DELANTE):

```
int main()
{
    struct Nodo * lista = NULL;
    struct Nodo * aux;
    . . .
    aux = lista;
    lista = (struct Nodo*)malloc (sizeof(struct Nodo));
    lista->Info = 1;
    lista->Sig = aux;
    . . .
}
```

Usamos una
variable auxiliar



aux = lista

Listas enlazadas

Añadir nodo a la lista (POR DELANTE):

```
int main()
{
    struct Nodo * lista = NULL;
    struct Nodo * aux;
    . . .
    aux = lista;
    lista = (struct Nodo*)malloc (sizeof(struct Nodo));
    lista->Info = 1;
    lista->Sig = aux;
    . . .
}
```

Usamos una
variable auxiliar



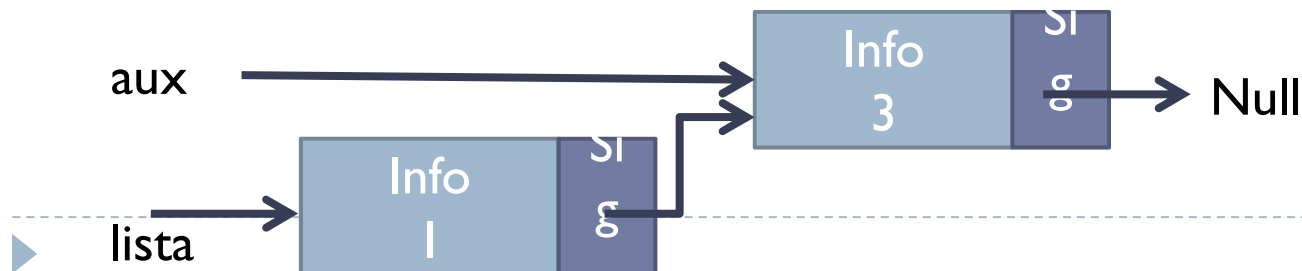
`lista = (struct Nodo*)malloc (sizeof(struct Nodo))`

Listas enlazadas

Añadir nodo a la lista (POR DELANTE):

```
int main()
{
    struct Nodo * lista = NULL;
    struct Nodo * aux;
    . . .
    aux = lista;
    lista = (struct Nodo*)malloc (sizeof(struct Nodo));
    lista->Info = 1;
    lista->Sig = aux;
    . . .
}
```

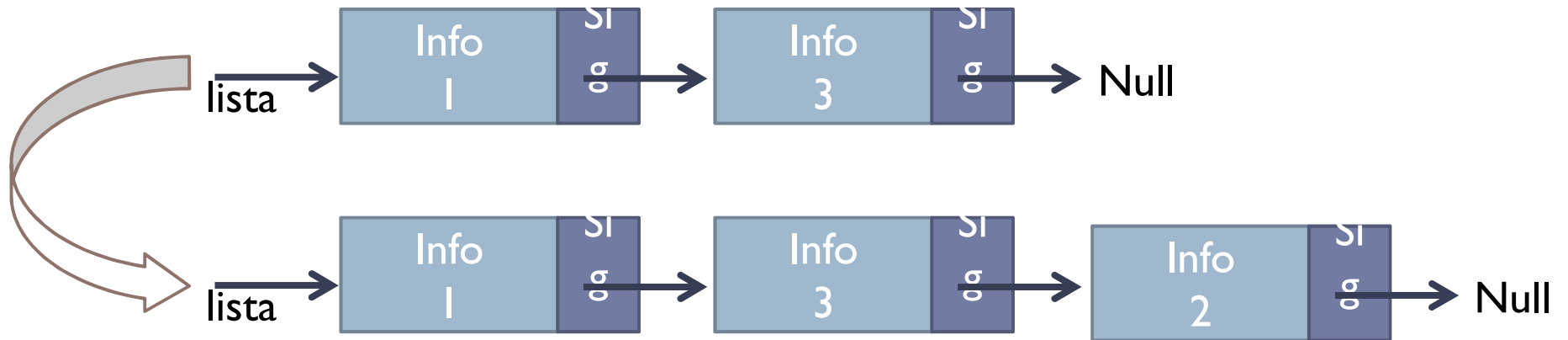
Usamos una
variable auxiliar



lista->Info = 1;
lista->Sig = aux;

Listas enlazadas

Añadir nodo a la lista (POR DETRÁS):




Listas enlazadas

Añadir nodo a la lista (POR DETRÁS):

```
int main()
{
    struct Nodo * lista = NULL;
    struct Nodo * aux;
    struct Nodo * nuevo;
    . . .
    for(aux=lista; aux->Sig!=Null; aux=aux->Sig);
    nuevo= (struct Nodo*)malloc (sizeof(struct Nodo));
    nuevo->Info = 2;
    nuevo->Sig = Null;
    aux->Sig = nuevo;
    . . .
}
```

Necesitamos 2
variables auxiliares

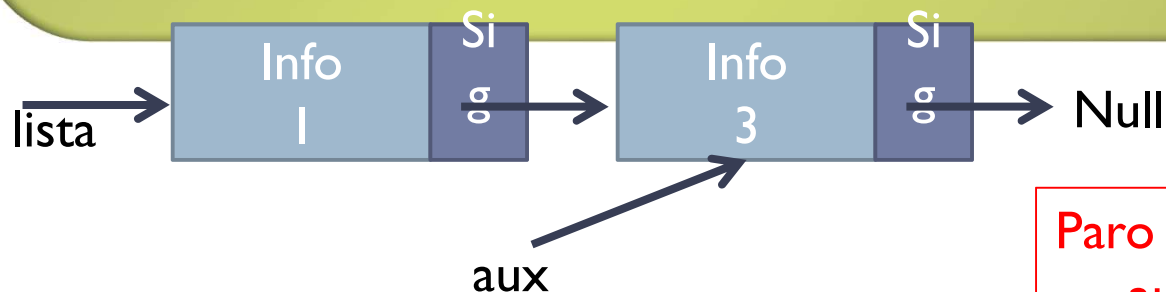


Listas enlazadas

Añadir nodo a la lista (POR DETRÁS):

```
int main()
{
    struct Nodo * lista = NULL;
    struct Nodo * aux;
    struct Nodo * nuevo;
    . . .
    for(aux=lista; aux->Sig!=Null; aux=aux->Sig);
    nuevo= (struct Nodo*)malloc (sizeof(struct Nodo));
    nuevo->Info = 2;
    nuevo->Sig = Null;
    aux->Sig = nuevo;
    . . .
}
```

“aux” recorre
la lista



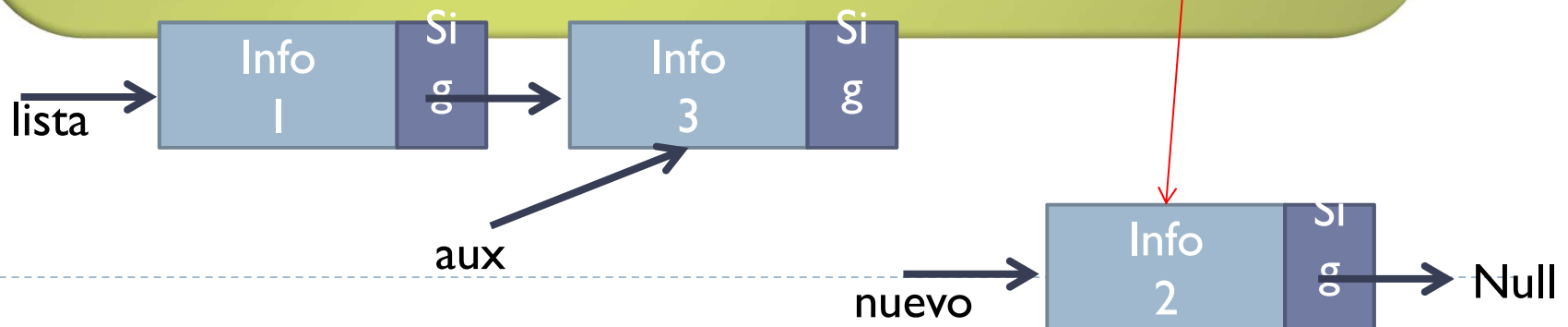
Paro cuando
`aux->Sig == Null`

Listas enlazadas

Añadir nodo a la lista (POR DETRÁS):

```
int main()
{
    struct Nodo * lista = NULL;
    struct Nodo * aux;
    struct Nodo * nuevo;
    . . .
    for(aux=lista; aux->Sig!=Null; aux=aux->Sig);
    nuevo= (struct Nodo*)malloc (sizeof(struct Nodo));
    nuevo->Info = 2;
    nuevo->Sig = Null;
    aux->Sig = nuevo;
    . . .
}
```

“nuevo” contendrá la información del nuevo nodo.

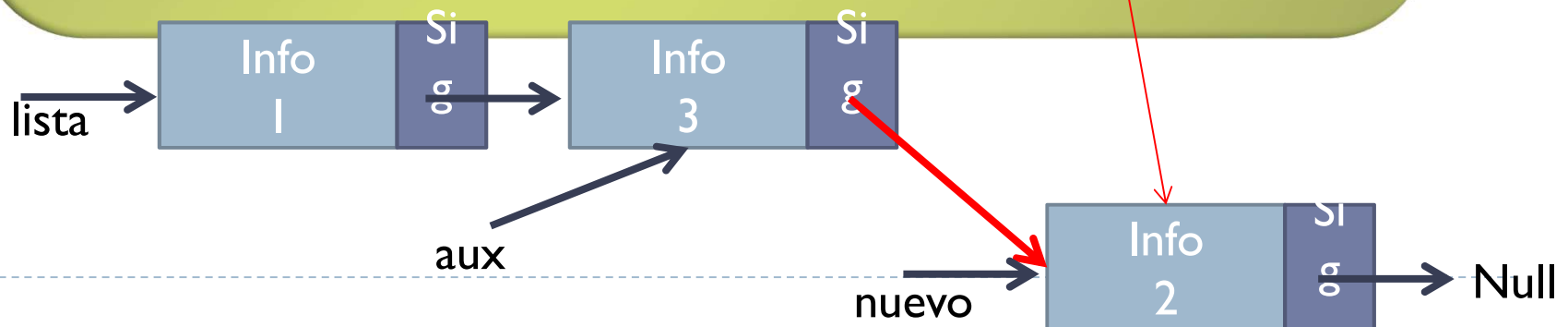


Listas enlazadas

Añadir nodo a la lista (POR DETRÁS):

```
int main()
{
    struct Nodo * lista = NULL;
    struct Nodo * aux;
    struct Nodo * nuevo;
    . . .
    for(aux=lista; aux->Sig!=Null; aux=aux->Sig);
    nuevo= (struct Nodo*)malloc (sizeof(struct Nodo));
    nuevo->Info = 2;
    nuevo->Sig = Null;
    aux->Sig = nuevo;
    . . .
}
```

Uno el nuevo
nodo a la lista.



Funciones para el manejo de una lista enlazada...

```
struct Nodo{  
    int Info;  
    struct Nodo * Sig;  
};
```

Definimos la
estructura de datos

```
int main()  
{  
    struct Nodo * lista = NULL;  
    . . .  
}
```



Funciones para el manejo de una lista enlazada...

```
typedef struct _Nodo{  
    int Info;  
    struct _Nodo * Sig;  
}Nodo;
```

```
int main()  
{  
    Nodo * lista = NULL;  
    Nodo * lista = (Nodo*)malloc (sizeof(Nodo));  
    . . .  
}
```

De este modo no
necesito usar la
palabra “struct” cada
vez que declaro una
variable Nodo



Funciones para el manejo de una lista enlazada...

```
typedef struct _Nodo{
    int Info;
    struct _Nodo * Sig;
}Nodo;

void recorrer(Nodo * lista);
int contarNodos(Nodo *lista);
void insertarInicio(Nodo **lista, int dato);
void insertarFinal(Nodo **lista, int dato);
void insertarEnPos(Nodo **lista, int dato, int pos);
Nodo *buscar(Nodo *lista, int dato);
void eliminar(Nodo **lista, Nodo *n);
void eliminarPos(Nodo **lista, int pos);
void eliminarlista(Nodo **lista, Nodo *n);

int main()
{
    Nodo * lista = NULL;
    . . .
}
```



Funciones para el manejo de una lista enlazada...

Recorrer la lista

```
//Función para recorrer una lista y mostrar su contenido.
void recorrer(Nodo *lista){
    //verifica si la lista está vacía
    if(lista == NULL){
        //si está vacía muestra este mensaje
        printf("La lista esta vacía.");
    }
    else{
        //si no esta vacía declara un puntero p al primer elemento de la lista
        Nodo *aux = lista;
        do{
            //muestra "Info" para el nodo apuntado por "aux"
            printf("Elemento: %d ", aux->Info);
            //movemos "aux" al elemento siguiente (o NULL, si no hubiera más)
            aux = aux->Sig;
            //repite el proceso mientras p sea diferente de NULL
        }while(aux != NULL);
    }
}
```


Funciones para el manejo de una lista enlazada...

Contar los elementos de una lista

```
//Función que devuelve el número de nodos de una lista.  
int contarNodos(Nodo *lista){  
    int cont = 0;  
    while(lista!=NULL){  
        lista = lista->Sig;  
        cont++;  
    }  
    return cont;  
}
```

Funciones para el manejo de una lista enlazada...

Insertar al principio de la lista

```
//Función que inserta un nodo nuevo al final de la lista
void insertarInicio(Nodo **lista, int dato){

    //crea un nuevo nodo e inicializa sus campos
    Nodo *nuevo = (Nodo*)malloc (sizeof(Nodo));
    nuevo->Info = dato;

    //nuevo->Sig apuntará al primer elemento de la lista
    //si la lista está vacía Sig sera NULL
    //si la lista no esta vacia Sig apuntara al 1er elemento
    nuevo->Sig= *lista;

    //lista apuntará al nuevo nodo
    //el nuevo nodo será ahora el primero

    *lista = nuevo;

}
```

Funciones para el manejo de una lista enlazada...

Insertar al principio de la lista

```
//Función que inserta un nodo nuevo al final de la lista
void insertarInicio(Nodo **lista, int dato){

    //crea un nuevo nodo e inicializa sus campos
    Nodo *nuevo = (Nodo*)malloc (sizeof(Nodo));
    nuevo->Info = dato;

    //nuevo->Sig apuntará al primer elemento de la lista
    //si la lista esta vacia Sig sera NULL
    //si la lista no esta vacia Sig apuntara al primer
    elemento
    nuevo->Sig= *lista;

    //lista apuntará al nuevo nodo
    //el nuevo nodo será ahora el primer elemento
    *lista = nuevo;

}
```

Para modificar el valor al que apunta “lista” debemos pasar “lista” por referencia. Pero “lista” es un puntero, para recibir la dirección de memoria de un puntero usaremos en los parámetros de la función un puntero a puntero.

Funciones para el manejo de una lista enlazada...

Insertar al final de la lista

```
//Función que inserta un nodo nuevo al final de la lista
void insertarFinal(Nodo **lista, int dato){
    //crea un nuevo nodo e inicializa sus campos
    Nodo * nuevo = (Nodo*)malloc (sizeof(Nodo));
    nuevo->Info = dato;
    nuevo->Sig= NULL;

    //si la lista está vacía
    if(*lista == NULL){
        //modifica la cabecera para que apunte al nuevo nodo
        *lista = nuevo;
    }else{
        //declara un puntero "aux" que apunta al primer nodo
        Nodo *aux = *lista;
        //avanza aux hasta el ultimo nodo
        while(aux->Sig != NULL){
            aux = aux->Sig;
        }
        //Añade "nuevo" al final de la lista
        aux->Sig = nuevo;
    }
}
```

Para modificar el valor al que apunta "lista" debemos pasar "lista" por referencia. Pero "lista" es un puntero, para recibir la dirección de memoria de un puntero usaremos en los parámetros de la función un puntero a puntero.

Funciones para el manejo de una lista enlazada...

Insertar en la posición enésima de la lista

```
//Función que inserta un nodo nuevo en la posición enésima de la lista
void insertarEnPos(Nodo **lista, int dato, int pos){
    //obtiene el numero de elementos de la lista y comprueba si 'pos' es correcto
    int n = contarNodos(*lista);
    if( pos < 1 || pos > n ) {
        printf("Posicion %d no valida. Hay %d elementos en la lista.",pos,n);
    } else if(pos == 1){//si pos es 1 (primera posición) llamamos a insertarInicio()
        insertarInicio(lista, dato);
    } else if(pos == n){//si n es igual al numero de nodos (ultima posición)
        //llamamos a insertarFinal()
        insertarFinal(lista, dato);
    }else{
        //declara un puntero "aux" al primer nodo y lo avanza a la posición pos-1
        Nodo *aux = *lista;
        int i;
        for(i=1; i<pos-1; i++) {
            aux = aux->Sig;
        }
        //crea un nuevo nodo e inicializa sus campos
        Nodo * nuevo = (Nodo*)malloc (sizeof(Nodo));
        nuevo->Info = dato;
        //el campo Sig del nuevo elemento apuntara al nodo siguiente a "aux"
        nuevo->Sig = aux->Sig;
        //el campo Sig de aux apuntara al nuevo nodo
        aux->Sig = nuevo;
    }
}
```

Funciones para el manejo de una lista enlazada...

Buscar un elemento en la lista

```
//Función que busca un nodo y devuelve una referencia al nodo.  
//Si no lo encuentra devuelve NULL.  
Nodo *buscar(Nodo *lista, int dato){  
    //si la lista esta vacia  
    if(lista == NULL){  
        printf("La lista esta vacia.");  
    }else{  
        //declara un puntero aux al primer elemento de la lista  
        Nodo *aux = lista;  
        do{  
            //si es el elemento buscado  
            if(aux->Info == dato){  
                //devuelve la referencia al nodo y termina la función  
                return aux;  
            }else{  
                //muevo aux al siguiente nodo  
                aux = aux->Sig;  
            }  
            //se repite mientras no se llegue al final de la lista  
        }while(aux!=NULL);  
    }  
    //si termina la lista y no encontró el nodo buscado devuelve NULL  
    return NULL;  
}
```

Funciones para el manejo de una lista enlazada...

Eliminar un elemento de la lista

```
//Función que elimina un nodo usando como criterio de búsqueda la info del nodo.
void eliminar(Nodo **lista, Nodo *n){
    if(*lista == NULL){//si la lista esta vacia
        printf("La lista está vacía.");
    }else{
        if(n != NULL){//si el nodo a eliminar existe
            //si el nodo a eliminar es el primero, que lista apunte al segundo
            if(n == *lista){
                *lista = (*lista)->Sig;
            }else{
                //declara un puntero 'aux' y hace que apunte al nodo anterior
                //al que se va a eliminar
                Nodo *aux = *lista;
                while(aux->Sig!=NULL && aux->Sig!=n){
                    aux = aux->Sig;
                }
                //enlaza al nodo anterior con el siguiente
                aux->Sig= n->Sig;
            }
            //libera el espacio de memoria del nodo eliminado
            free (n);
        }else{
            printf("No se encontró el elemento.");
        }
    }
}
```

Funciones para el manejo de una lista enlazada...

Eliminar el enésimo elemento de la lista

```
void eliminarPos(Nodo **lista, int pos){
    //obtiene el numero de elementos de la lista y comprueba si 'pos' es
    correcto
    int n = contarNodos(*lista);
    if( pos < 1 || pos > n ) {
        printf("Posicion %d no valida. Hay %d elementos en la lista.",pos,n);
    }else{
        //declara un puntero "aux" que apunte a la posición indicada y
        "ant_aux"
        //que apunte al elemento anterior
        Nodo *aux = *lista, *ant_aux;
        int i;
        for(i=1; i<pos; i++) {
            ant_aux = aux;
            aux = aux->Sig;
        }
        //si el nodo a eliminar es el primero
        if(aux == *lista) {
            //que lista apunte al segundo elemento
            *lista= (*lista)->Sig;
        } else {
            //enlaza el nodo anterior con el nodo siguiente
            ant_aux->Sig = aux->Sig;
        }
        //libera el espacio de memoria del nodo eliminado
        free (aux);
    }
}
```


Funciones para el manejo de una lista enlazada...

Eliminar una lista

```
//Función que borra una lista
void eliminarlista(Nodo **lista){
    int n,i;
    if(*lista == NULL){//si la lista esta vacia
        printf("La lista está vacía.");
    }else{
        n=contarNodos(*lista);
        for(i=0;i<n;i++)
            eliminarPos(lista,1);
    }
    return;
}
```

El TAD Lista

Se trata de definir e implementar un TAD (Tipo Abstracto de Datos), en este caso el “tipo de datos Lista”, usando como estructura de datos una “Lista enlazada”.



El TAD Lista

Se trata de definir e implementar un TAD (Tipo Abstracto de Datos), en este caso el “tipo de datos Lista”, usando como estructura de datos una “Lista enlazada”.

El TAD debe disponer de una **interfaz**: la definición del tipo de datos y las funciones para trabajar con dichos datos.

```
typedef struct _Nodo{
    int Info;
    struct _Nodo * Sig;
}Nodo;

void recorrer(Nodo * lista);
int contarNodos(Nodo *lista);
void insertarInicio(Nodo **lista, int dato);
void insertarFinal(Nodo **lista, int dato);
void insertarEnPos(Nodo **lista, int dato, int pos);
Nodo *buscar(Nodo *lista, int dato);
void eliminar(Nodo **lista, Nodo *n);
void eliminarPos(Nodo **lista, int pos);
```



El TAD Lista

Se trata de definir e implementar un TAD (Tipo Abstracto de Datos), en este caso el “tipo de datos Lista”, usando como estructura de datos una “Lista enlazada”.

El TAD debe disponer de una **interfaz**: la definición del tipo de datos y las funciones para trabajar con dichos datos.

Definimos dicha
interfaz en un archivo
de cabecera .h

lista.h

```
typedef struct _Nodo{
    int Info;
    struct _Nodo * Sig;
}Nodo;

void recorrer(Nodo * lista);
int contarNodos(Nodo *lista);
void insertarInicio(Nodo **lista, int dato);
void insertarFinal(Nodo **lista, int dato);
void insertarEnPos(Nodo **lista, int dato, int pos);
Nodo *buscar(Nodo *lista, int dato);
void eliminar(Nodo **lista, Nodo *n);
void eliminarPos(Nodo **lista, int pos);
```



El TAD Lista

Se trata de definir e implementar un TAD (Tipo Abstracto de Datos), en este caso el “tipo de datos Lista”, usando como estructura de datos una “Lista enlazada”.

El TAD debe disponer de una **interfaz**: la definición del tipo de datos y las funciones para trabajar con dichos datos.

Definimos dicha interfaz en un archivo de cabecera .h

Así, para usar este tipo de datos sólo será necesario incluir este archivo en nuestro programa.
`#include “lista.h”`

```
#ifndef __LISTA_H
#define __LISTA_H

typedef struct _Nodo{
    int Info;
    struct _Nodo * Sig;
}Nodo;

void recorrer(Nodo * lista);
int contarNodos(Nodo *lista);
void insertarInicio(Nodo **lista, int dato);
void insertarFinal(Nodo **lista, int dato);
void insertarEnPos(Nodo **lista, int dato, int pos);
Nodo *buscar(Nodo *lista, int dato);
void eliminar(Nodo **lista, Nodo *n);
void eliminarPos(Nodo **lista, int pos);

#endif // __LISTA_H
```

lista.h

El TAD Lista

La implementación de dichas funciones estará en otro archivo .c

Nuestro archivo lista.c deberá incluir nuestro archivo de cabecera.
`#include "lista.h"`

lista.c

```
#include "lista.h"
```

```
//Función para recorrer una lista y mostrar su contenido.  
void recorrer(Nodo *lista){
```

```
    ...  
}
```

```
//Función que devuelve el numero de nodos de una lista.  
int contarNodos(Nodo *lista){
```

```
    ...  
}
```

```
//Función que inserta un nodo nuevo al final de la lista  
void insertarInicio(Nodo **lista, int dato){
```

```
    ...  
}
```

```
//Función que inserta un nodo nuevo al final de la lista  
void insertarFinal(Nodo **lista, int dato){
```

```
    ...  
}
```

El TAD Lista

Ya podemos crear nuestro programa y usar nuestro nuevo tipo de datos y las funciones deseadas. Nuestro archivo programa.c deberá incluir nuestro archivo de cabecera. `#include "lista.h"`

programa.c

```
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

int main()
{
    Nodo * lista = NULL;
    int n;

    printf("Introduce números...(-1 para
terminar)\n");
    scanf("%d",&n);
    while (n!=-1) {
        insertarFinal(&lista,n);
        scanf("%d",&n);
    }

    recorrer(lista);
    printf("total: %d nodos\n",contarNodos(lista));
}
```

¿Cómo compilamos?

Como usamos una nueva librería (lista.h) debemos “linkarla”

```
$gcc -c lista.c -o lista.o  
$gcc -c programa.c -o programa.o  
$gcc -o programa programa.o lista.o
```

Ya podemos ejecutar nuestro programa

```
$ ./programa
```



Antes de comenzar...

Crea el TAD Lista con las funciones proporcionadas. Para ello:

- Crea el archivo de cabecera lista.h
- Crea el archivo lista.c con las funciones que se te proporcionan.
- Crea un archivo de prueba (programa.c) y prueba las funciones básicas.
- Compila y ejecuta para comprobar que todo funciona correctamente.

Para realizar los ejercicios propuestos, haz uso de este TAD:

- Si se te pide hacer una nueva función, añádela a los archivos lista.h y lista.c y prueba que funciona en programa.c
- Si se te pide hacer un TAD, sigue la misma estructura.



Ejercicios propuestos...

1. Realiza una modificación de la función insertar al final de la lista de modo que no se permita insertar datos repetidos: si un dato ya está almacenado entonces la lista no varía. Llama a la función *InsertarFinalSinRep.*
2. Implementa la función *InsertarOrdenado*. Esta función insertará los datos en la lista en la posición que le corresponda siguiendo un orden ascendente.



Más ejercicios propuestos...

1. Escribe una función que devuelva los Números que se encuentran en posiciones Pares de una Lista.
2. Escribe una función que muestre por pantalla la Posición de un Nodo Especifico (Pedido por Teclado).
3. Escribe una función que muestre por pantalla todos los Nodos de una lista que superen un valor determinado.
4. Escribe una función que calcule el mayor de los datos e indique la posición en que se encuentra.
5. Escribe una función que intercambie dos elementos de una Lista Enlazada (según sus valores).



Un último ejercicio...

EL juzgado de policía local recibe diariamente 10 multas de tráfico, cada una con la siguiente información:

- a) Tipo de Multa:
1.- Velocidad 2.- Mal estacionado
- b) Edad del conductor
- c) Sexo del conductor
M= masculino F= femenino
- d) Estado civil
C= casado S= soltero

Se pide confeccionar un programa que pida la información de las 10 multas, la guarde en una lista y luego muestre por pantalla:

- Cuántas multas por velocidad se cursaron
- Cuántas multas por velocidad cometieron las mujeres
- Cuántos hombres casados tienen multas por mal estacionado
- Cuántos hombres y mujeres solteros tiene multa por velocidad
- Cuántas mujeres mayores de 35 años tienen multas



Listas doblemente enlazadas

Programación en C

Listas doblemente enlazadas

Reservamos espacio para un nuevo elemento a medida que se va necesitando.

Para insertar un elemento en la lista necesitamos unirlo mediante un puntero.

Cada nodo guarda información del nodo anterior y del siguiente

