

# TPN°1: Listas, Uso de Variables Dinámicas, Punteros.

Algoritmos y Estructuras de Datos  
2024



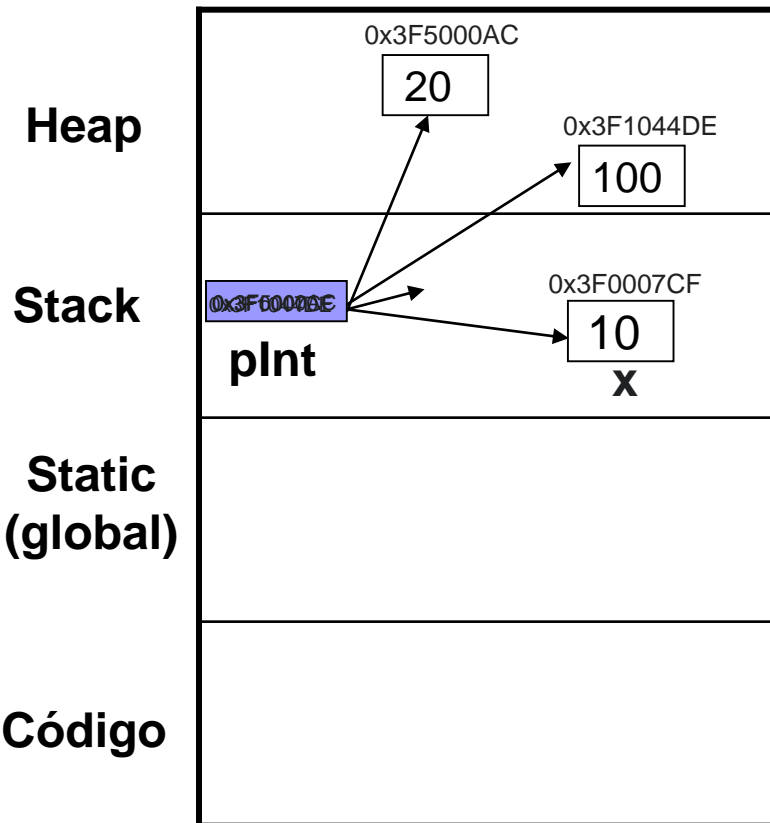
# Uso de Variables Dinámicas

## Punteros

## REPASO

# Variables Dinámicas

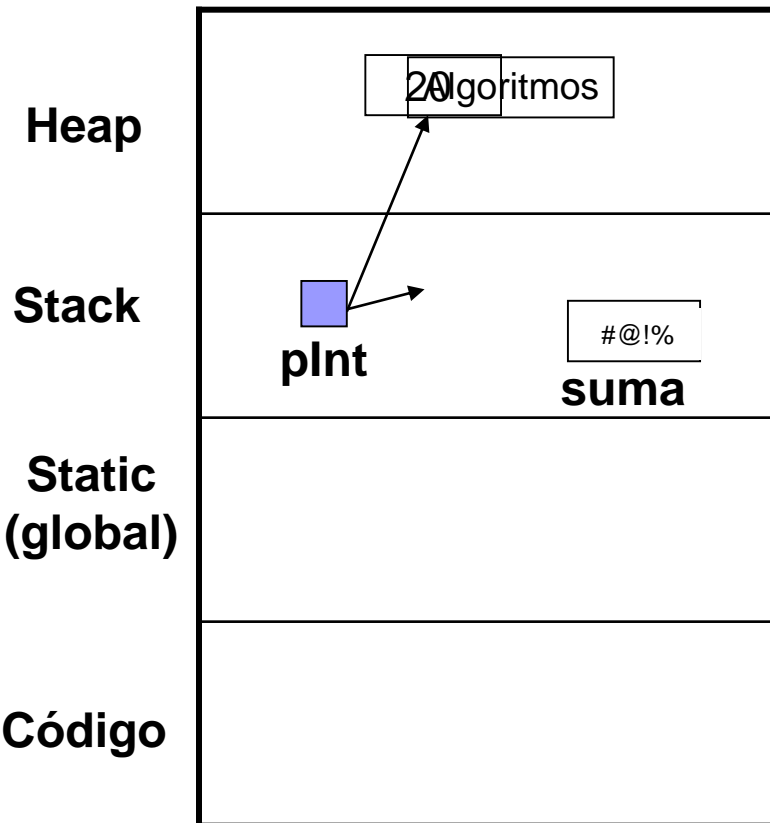
## MEMORIA RAM



```
main(){  
    . . .  
    int x = 10; //x vble creada estáticamente  
    int *pInt;  
    pInt = &x;  
    pInt = (int *)malloc(sizeof(int));  
    //contenido de pInt dir vble creada dinámicamente  
    *pInt = 20;  
    free pInt;  
    pInt = (int *)malloc(sizeof(int));  
    *pInt = 100;  
    free pInt;  
    . . .  
}
```

# Variables Dinámicas

MEMORIA RAM

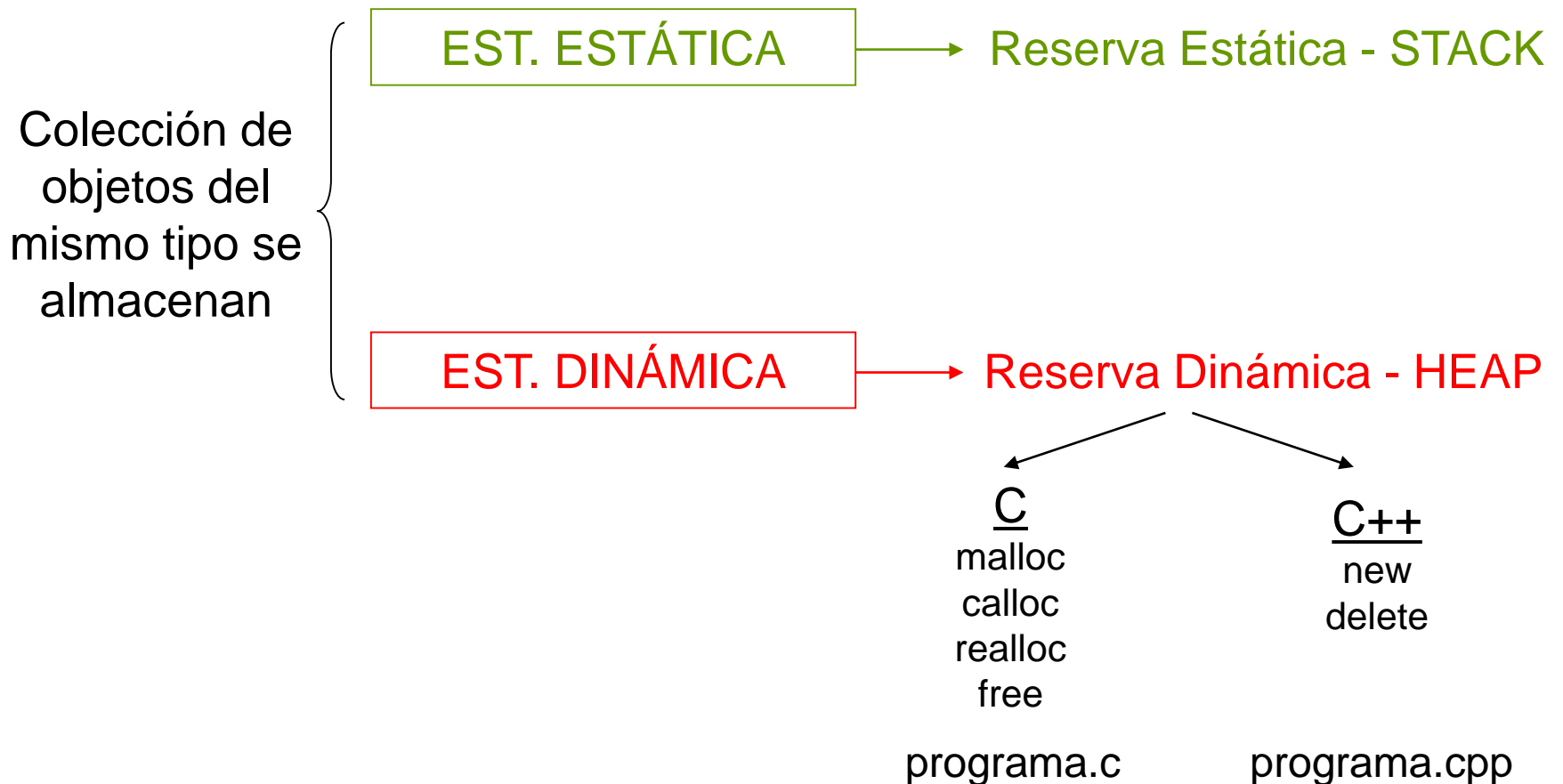


```
main(){
    . . .
    int *pInt;
    pInt = (int *)malloc(sizeof(int));
    //contenido de pInt dir vble creada dinámicamente
    *pInt = 20;
    int suma = *pInt + 10;
    free pInt;
    //al liberar la memoria reservada, el sistema
    puede reutilizar el espacio del heap y asignarlo
    a otros procesos para su uso
    suma = *pInt + 100;
    //Error
    . . .
}
```



# LISTA ENLAZADA

# Variables Dinámicas



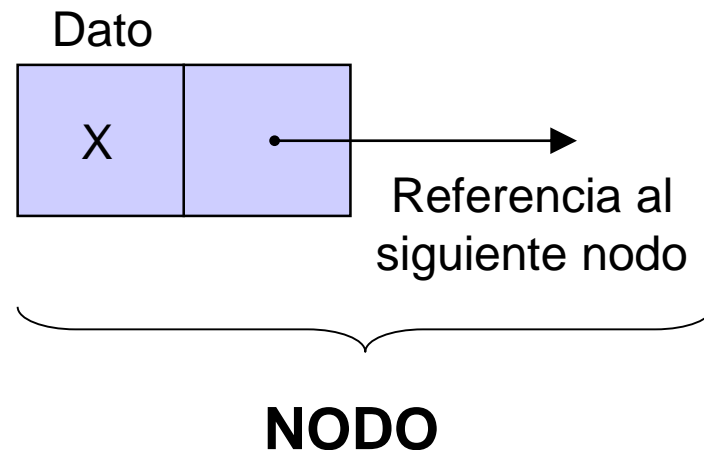
# Listas Enlazadas

*Una lista enlazada es una colección de elementos llamados nodos*

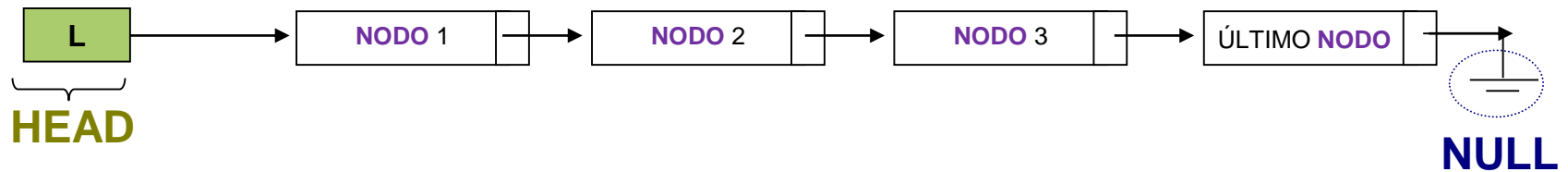
- Cada nodo se representa por medio de dos campos:

***Campo dato:*** contiene el valor del nodo

***Campo siguiente:*** indica cuál es el nodo con el que se enlaza



# Listas Enlazadas

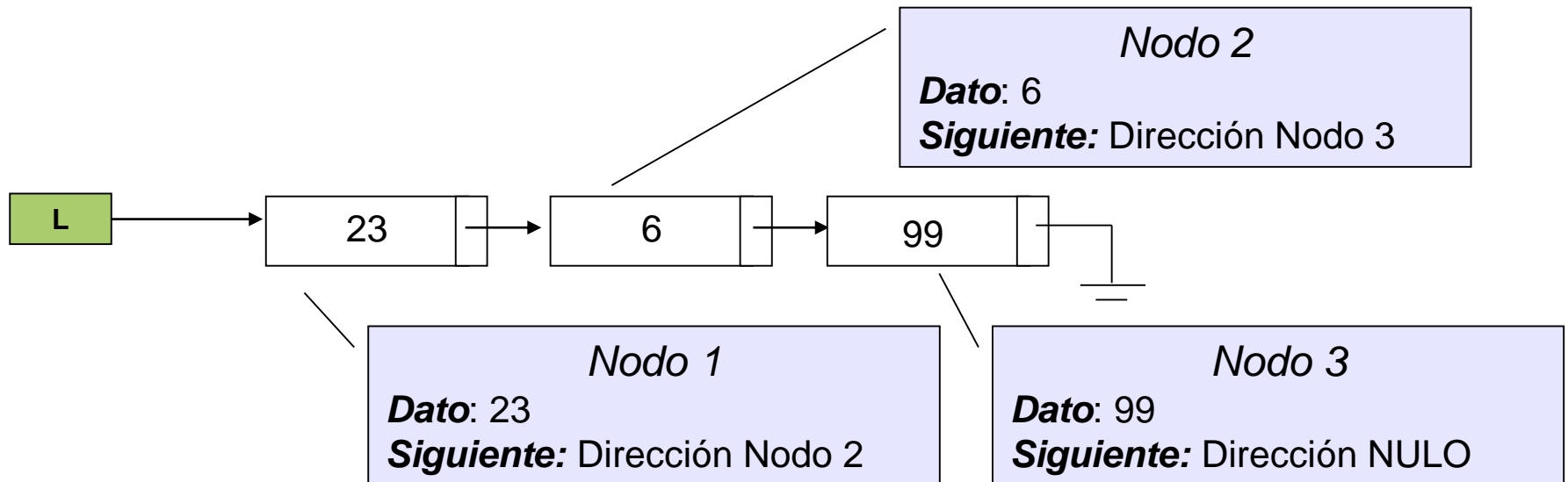


Las listas enlazadas son estructuras de datos dinámicas cuyo tamaño puede crecer o disminuir a medida que se ejecuta el programa



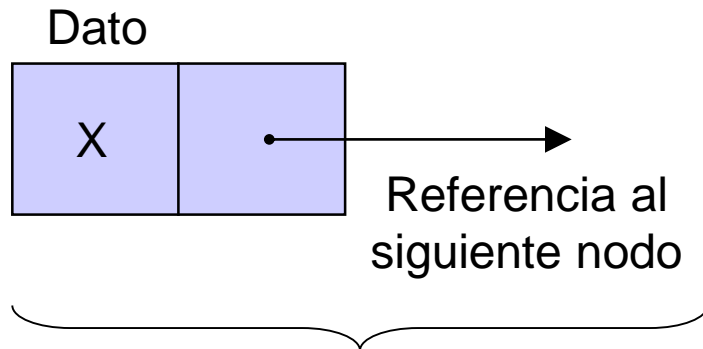
# Listas Enlazadas

- Ejemplo de lista enlazada con 3 nodos
- Los nodos tienen datos de tipo entero



# Listas Enlazadas

## IMPLEMENTACIÓN EN C++

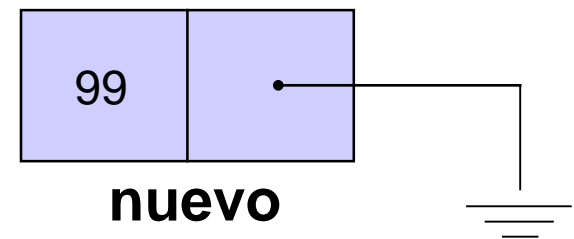


**NODO**

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

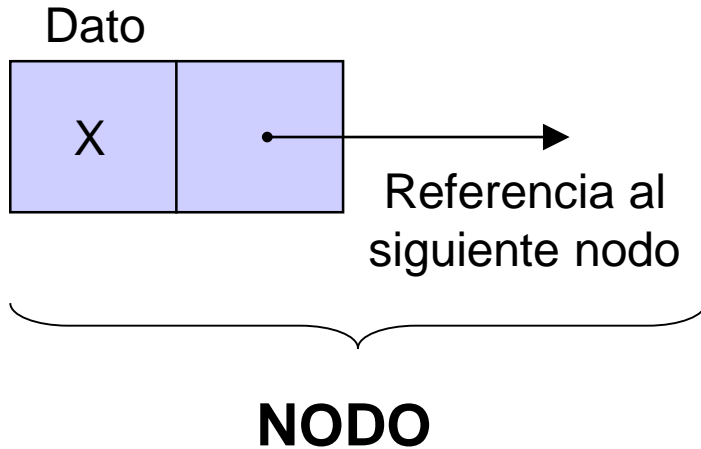
## EJEMPLO CREACIÓN ESTÁTICA DE UN NODO

```
main(){  
    ...  
    Nodo nuevo;  
    nuevo.dato=99;  
    nuevo.siguiente = NULL;  
    ...  
}
```



# Listas Enlazadas

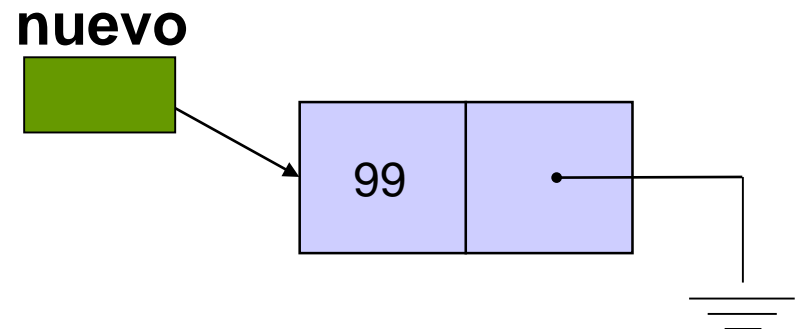
IMPLEMENTACIÓN EN C++



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

## EJEMPLO CREACIÓN DINÁMICA DE UN NODO

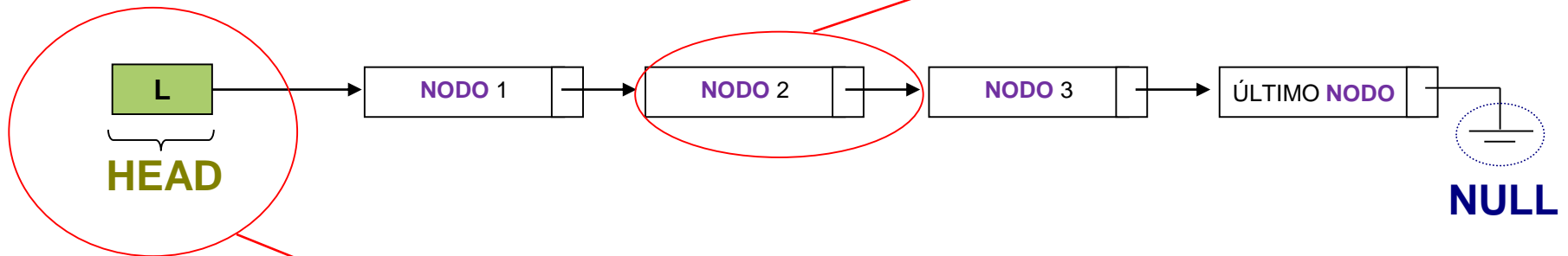
```
main(){  
    ...  
    Nodo *nuevo;  
    nuevo = new Nodo;  
    nuevo->dato=99;  
    nuevo->siguiente = NULL;  
    ...  
}
```



# Listas Enlazadas

IMPLEMENTACIÓN EN C++

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



```
typedef struct Nodo* Lista;
```

**Lista  $\equiv$  HEAD**

# Listas Enlazadas

LISTA.H

## TIPIFICACIÓN DE LA LISTA ENLAZADA EN C++

```
struct Nodo {  
    int dato;  
    Nodo* siguiente;  
};  
typedef struct Nodo* Lista;
```

DE FORMA GENÉRICA...

```
typedef int item;  
struct Nodo {  
    item dato;  
    Nodo* siguiente;  
};  
typedef struct Nodo* Lista;
```



# Listas Enlazadas

Posibles operaciones sobre una lista enlazada:

- Crear una lista vacía
- Insertar un nodo al comienzo de la lista
- Insertar un nodo al final de la lista
- Eliminar el primer nodo de la lista
- Eliminar el último nodo de la lista
- Buscar el nodo que contiene un dato determinado
- Escribir los datos de la lista
- Test para determinar si una lista es vacía
- Invertir la lista
- Etc.

# Listas Enlazadas

LISTA.H

OPERACIONES DE LA LISTA:

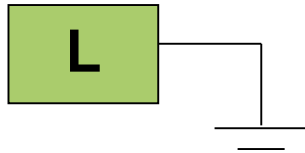
TRABAJO PRÁCTICO N° 1

- a) **crearLista**: crea una lista vacía.
- b) **esListaVacía**: booleana, determina si la lista está vacía.
- c) **mostrar**: muestra por pantalla el contenido de los nodos de la lista.
- d) **primerElemento**: retorna el valor del primer nodo de la lista.
- e) **insertar**: inserta un valor dado al comienzo de la lista.
- f) **borrar**: borra el nodo del comienzo de la lista.
- g) **longitud**: cuenta la cantidad de nodos que tiene una lista.
- h) **pertenece**: booleana, determina si un valor dado pertenece a la lista.
- i) **insertarK**: inserta un valor dado en la posición K-ésima de la lista si es que existe, caso contrario se inserta al final.

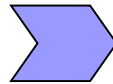
# Listas Enlazadas

- **crearLista**: función que crea una lista vacía

```
typedef int item;
struct Nodo {
    item dato;
    Nodo* siguiente;
};
typedef struct Nodo* Lista;
```



```
Lista crearLista(){
    Lista L;
    L = NULL;
    return L;
}
```



```
Lista crearLista(){
    return NULL;
}
```

```
main(){
    ...
    Lista H;
    H = crearLista();
    ...
}
```

La **cabecera** de la lista debe estar inicializada con `crearLista` antes de aplicar cualquier otra operación





# Listas Enlazadas

• **esListaVacia**: función booleana que determina si la lista está vacía.

```
bool esListaVacia(Lista L){...}
```

```
typedef int item;  
struct Nodo {  
    item dato;  
    Nodo* siguiente;  
};  
typedef struct Nodo* Lista;
```

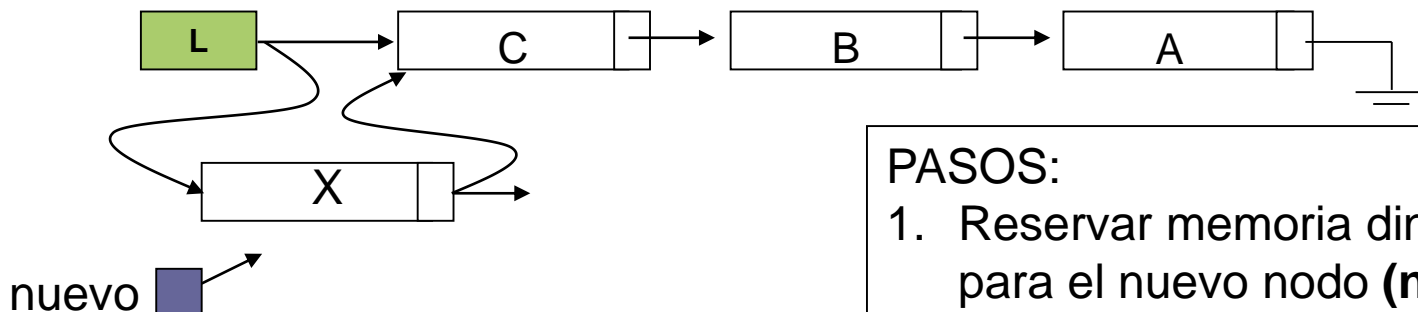
```
FUNCION esListaVacia(L): Lista → BOOL  
    SI L = NULA ENTONCES  
        RETORNA true;  
    ELSE  
        RETORNA false;
```

# Listas Enlazadas

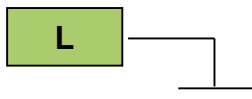
•**insertar**: función que inserta un nodo al comienzo de la lista.

```
typedef int item;
struct Nodo {
    item dato;
    Nodo* siguiente;
};
typedef struct Nodo* Lista;
```

**FUNCION insertar(L, x): Lista x item → Lista**



¿Funcionan estos pasos si la lista está vacía?



## PASOS:

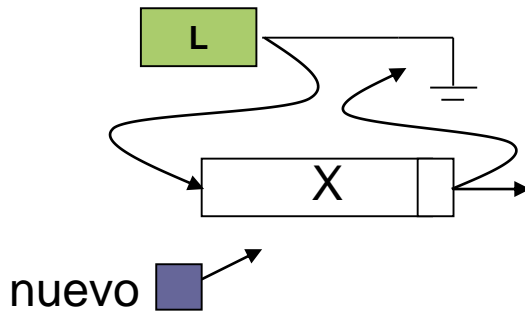
1. Reservar memoria dinámicamente para el nuevo nodo (**new**)
2. Agregar el dato X al nodo
3. Enlazar el nuevo nodo a la lista
  - a. El puntero siguiente del nuevo nodo apunta al primer nodo de la lista
  - b. La cabecera de la lista apunta al nuevo nodo
4. Retornar la cabecera de la lista

# Listas Enlazadas

•**insertar**: función que inserta un nodo al comienzo de la lista.

```
typedef int item;
struct Nodo {
    item dato;
    Nodo* siguiente;
};
typedef struct Nodo* Lista;
```

**FUNCION insertar(L, x): Lista x item → Lista**



## PASOS:

1. Reservar memoria dinámicamente para el nuevo nodo (**new**)
2. Agregar el dato X al nodo
3. Enlazar el nuevo nodo a la lista
  - a. El puntero siguiente del nuevo nodo apunta al primer nodo de la lista
  - b. La cabecera de la lista apunta al nuevo nodo
4. Retornar la cabecera de la lista

# Listas Enlazadas

• **insertar**: función que inserta un nodo al comienzo de la lista.

```
typedef int item;
struct Nodo {
    item dato;
    Nodo* siguiente;
};
typedef struct Nodo* Lista;
```

**FUNCION insertar(L, x): Lista x item → Lista**

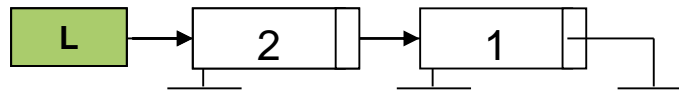
Aux: nuevo: puntero a nodo

```
nuevo ← reserva de memoria con new
nuevo_dato ← x
nuevo_siguiente ← L
L ← nuevo
RETORNA L
```

```
main(){
    Lista L = crearLista();
    L = insertar(L, 1);
    L = insertar(L, 2);
    ...
}
```

**PASOS:**

1. Reservar memoria dinámicamente para nuevo nodo (**new**)
2. Agregar el dato X al nodo
3. Enlazar el nuevo nodo a la lista
  - a. El puntero siguiente del nuevo nodo apunta al primer nodo de la lista
  - b. La cabecera de la lista apunta al nuevo nodo
4. Retornar la cabecera de la lista



**Lista insertar(Lista L, item X){...}**

# Listas Enlazadas

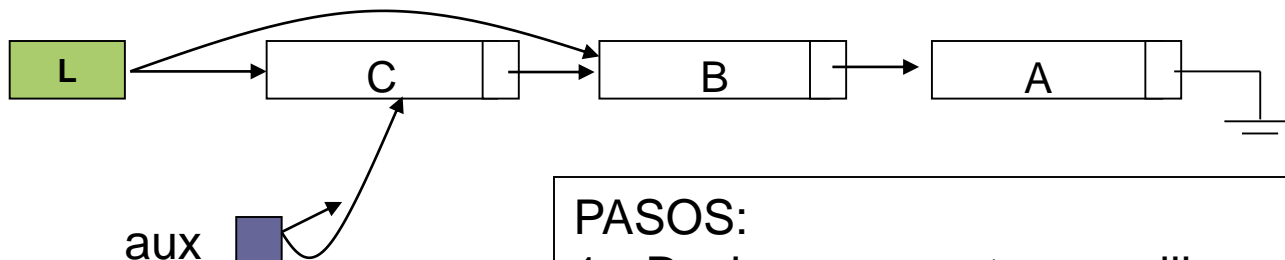
•**borrar**: función que borra el nodo del comienzo de la lista.

```
Lista borrar(Lista L){...}
```

```
typedef int item;  
struct Nodo {  
    item dato;  
    Nodo* siguiente;  
};  
typedef struct Nodo* Lista;
```

*CASO 1: La lista está vacía => no tengo nodo para borrar*

*CASO 2: La lista NO está vacía*



PASOS:

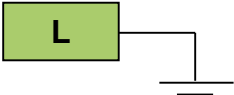
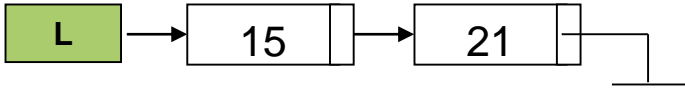
1. Declarar un puntero auxiliar a nodo
2. Apuntar el puntero auxiliar al 1er nodo de la lista.
3. Modificar la cabecera para que apunte al siguiente nodo de la lista
4. Borrar el nodo » **delete (aux);**
5. Retornar la cabecera de la lista

# Listas Enlazadas

- **primerElemento**: función que retorna el primer elemento de la lista.

```
typedef int item;  
struct Nodo {  
    item dato;  
    Nodo* siguiente;  
};  
typedef struct Nodo* Lista;
```

```
item primerElemento(Lista L){...}
```

CASOS	RESULTADO DE APLICAR LA FUNCIÓN
	Indefinido
	15

# Listas Enlazadas

• **mostrar**: función que muestra por pantalla el contenido de los nodos de la lista.

```
void mostrar(Lista L){...}
```

```
typedef int item;  
struct Nodo {  
    item dato;  
    Nodo* siguiente;  
};  
typedef struct Nodo* Lista;
```

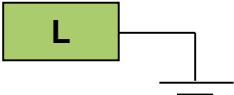
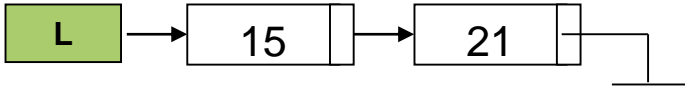
```
PROCEDIMIENTO mostrar(L): Lista  
    MIENTRAS NO esListaVacia(L)  
        ESCRIBIR(Ldato)  
        L ← Lsiguiente
```

# Listas Enlazadas

• **longitud**: función que cuenta la cantidad de nodos que tiene una lista.

```
int longitud(Lista L){...}
```

```
typedef int item;  
struct Nodo {  
    item dato;  
    Nodo* siguiente;  
};  
typedef struct Nodo* Lista;
```

CASOS	RESULTADO DE APLICAR LA FUNCIÓN
	0
	2

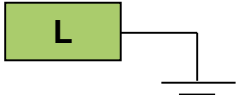
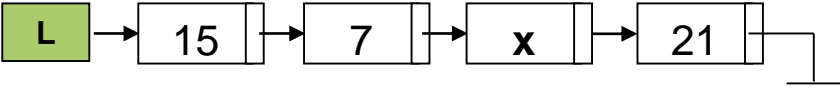
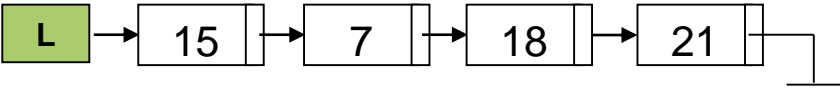


# Listas Enlazadas

•**pertenece**: función booleana que determina si un dato pertenece a la lista.

```
typedef int item;
struct Nodo {
    item dato;
    Nodo* siguiente;
};
typedef struct Nodo* Lista;
```

```
bool pertenece(Lista L, item x){...}
```

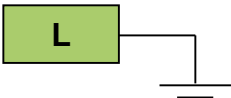
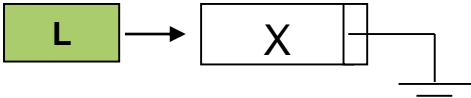
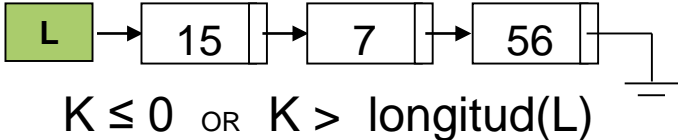
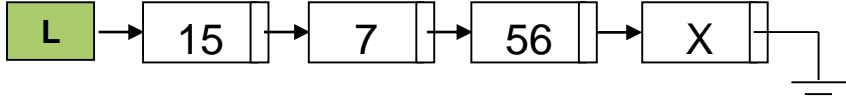
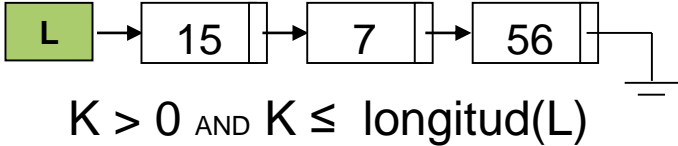
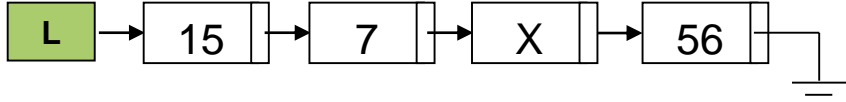
CASOS	RESULTADO DE APLICAR LA FUNCIÓN
	false
	true
	false

# Listas Enlazadas

•**insertarK**: inserta un valor en la posición K de la lista si es que existe, caso contrario se inserta al final

```
typedef int item;
struct Nodo {
    item dato;
    Nodo* siguiente;
};
typedef struct Nodo* Lista;
```

```
Lista insertarK(Lista L, int K, item X){...}
```

CASOS	RESULTADO DE APLICAR LA FUNCIÓN
	
	
	

# Listas Enlazadas

## IMPORTANTE

Si manipulamos una lista **COMO USUARIO**  
de la misma **NO** podemos acceder a su  
estructura interna  
(dato y puntero a siguiente nodo)

Solo podemos manipular la misma a través de  
las operaciones que ésta provee  
(crearLista, esListaVacía, mostrar, primerElemento,  
insertar, borrar, longitud, pertenece, insertarK).

# Listas Enlazadas

## COMO OPERACIÓN DE LA LISTA

```
PROCEDIMIENTO mostrar(L): Lista
  MIENTRAS NO esListaVacia(L)
    ESCRIBIR(Ldato)
    L ← Lsiguiente
```

¿Cómo debería ser el pseudocódigo del procedimiento ***mostrar*** si lo tuviéramos que hacer **COMO USUARIO** de la lista?

# Listas Enlazadas

crearLista  
primerElemento  
longitud

esListaVacia  
insertar  
pertenece

mostrar  
borrar  
insertarK.

## COMO USUARIO DE LA LISTA

```
PROCEDIMIENTO mostrar(L): Lista
  MIENTRAS NO esListaVacia(L)
    ESCRIBIR( primerElemento(L) )
    L ← borrar(L)
```

Borrar libera la memoria del  
nodo reservada dinámicamente

Lo correcto es retornar la  
cabecera modificada

## COMO OPERACIÓN DE LA LISTA

```
PROCEDIMIENTO mostrar(L): Lista
  MIENTRAS NO esListaVacia(L)
    ESCRIBIR(Ldato)
    L ← Lsiguiente
```

```
FUNCION mostrar(L): Lista → Lista
  MIENTRAS NO esListaVacia(L)
    ESCRIBIR( primerElemento(L) )
    L ← borrar(L)
  RETORNA L
```

¿Qué sucederá con  
nuestra lista?

# Listas Enlazadas

## ESTILOS DE PROGRAMACIÓN

```
typedef int item;
struct Nodo {
    item dato;
    Nodo* siguiente;
};
typedef struct Nodo* Lista;
```

ESTILO 1	ESTILO 2
Lista crearLista()	void crearLista(Lista* L)
Lista insertar(Lista L, item x)	void insertar(Lista *L, item x)
Lista borrar(Lista L)	void borrar(Lista *L)
Lista insertarK(Lista L, item x, int k)	void insertarK(Lista *L, item x, int k)



Prueba1.cpp

IMPLEMENTACIÓN  
EN C++



Prueba2.cpp

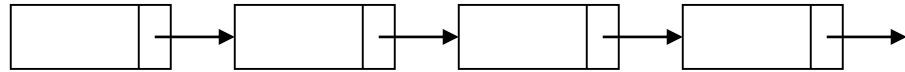
# Listas Enlazadas

**LISTAS**

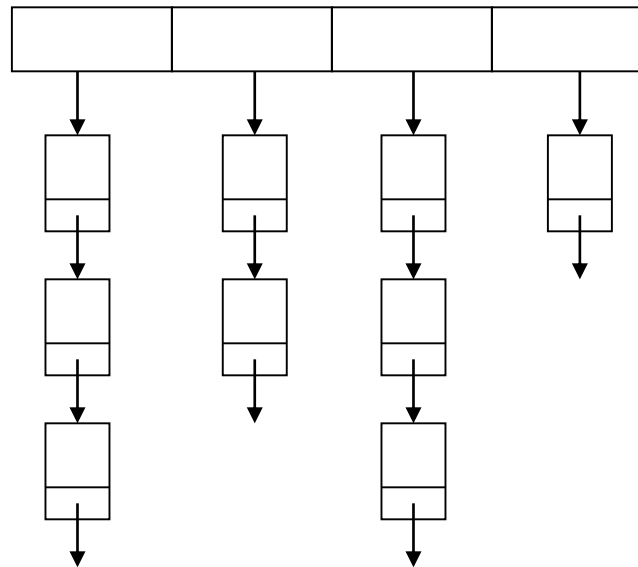
**PILAS**

**FILAS**

**LISTA CIRCULAR**

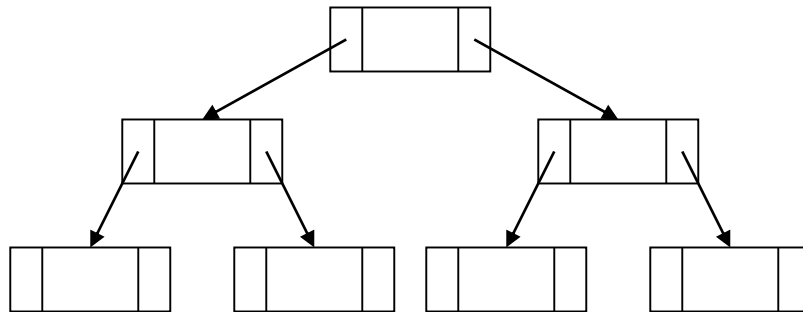


**GRAFOS**

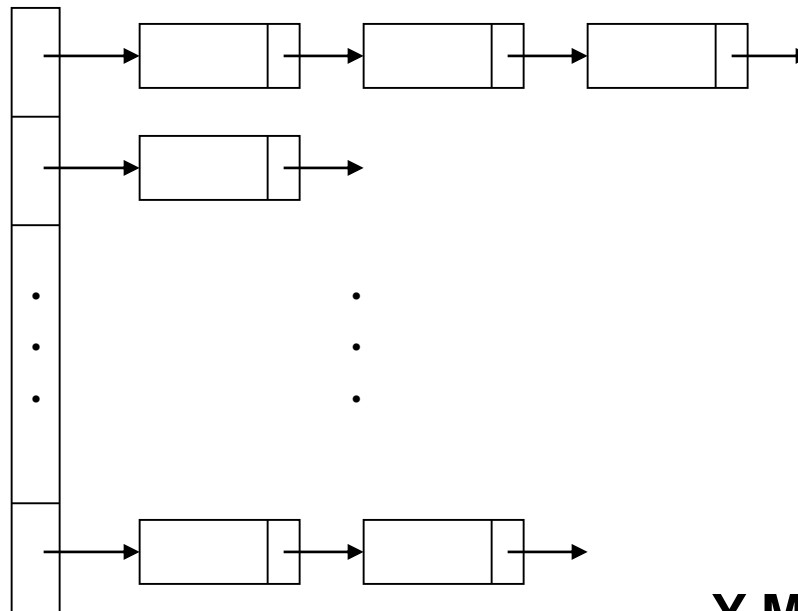


# Listas Enlazadas

ÁRBOLES



TABLAS DE HASH



Y MUCHOS MÁS...



# Compilador G++

## Comando

```
g++ -Wall nombreArchivo.cpp -o nombreEjecutable
```

### IMPLEMENTACIÓN EN C++

#### OBS:

- Al compilar con g++ automáticamente se compilan los archivos .h que se hayan incluido con la directiva #include
- Los archivos .h deben estar en la misma ubicación que el archivo .cpp que se está compilando

Preguntas...  
...y a practicar...

