Estructura de Datos y Algoritmos 1

Teórico #7:

Listas enlazadas

Intro: Listas basadas en *arrays* (vectores)

Declaración y creación

```
<tipo>* <nombre> = new <tipo>[<longitud>];
```

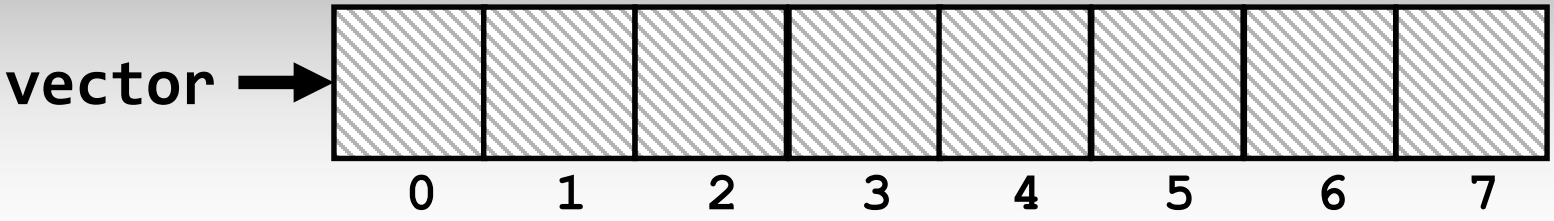
Ejemplo

```
int* puntuaciones = new int[30];
```

- Precisamos establecer un tamaño inicial.
- Se reservaría memoria que posiblemente no se utilice totalmente en ciertos momentos del programa.

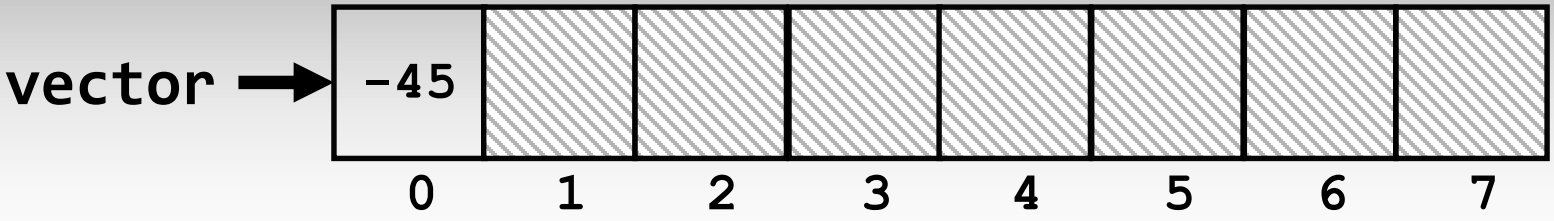
Intro: Listas basadas en *arrays* (vectores)

```
int* vector = new int[8];
```



Intro: Listas basadas en *arrays* (vectores)

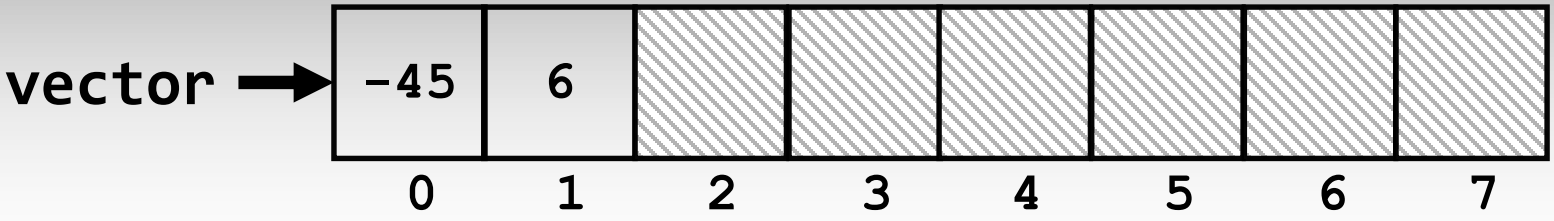
```
int* vector = new int[8];
```




```
vector[0] = -45;
```

Intro: Listas basadas en *arrays* (vectores)

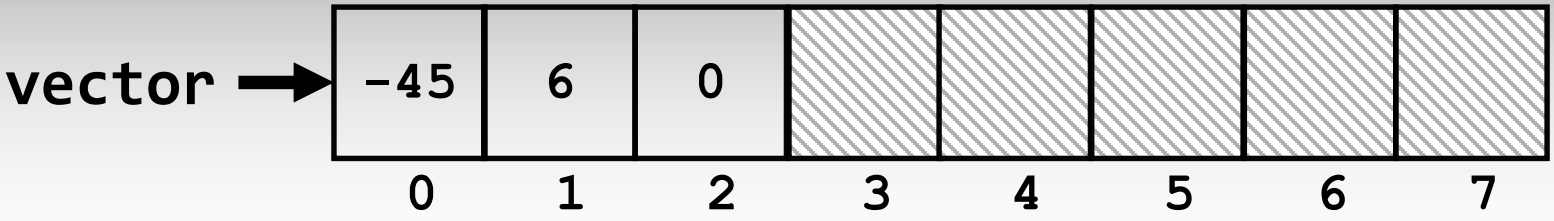
```
int* vector = new int[8];
```




```
vector[0] = -45;
vector[1] = 6;
```

Intro: Listas basadas en *arrays* (vectores)

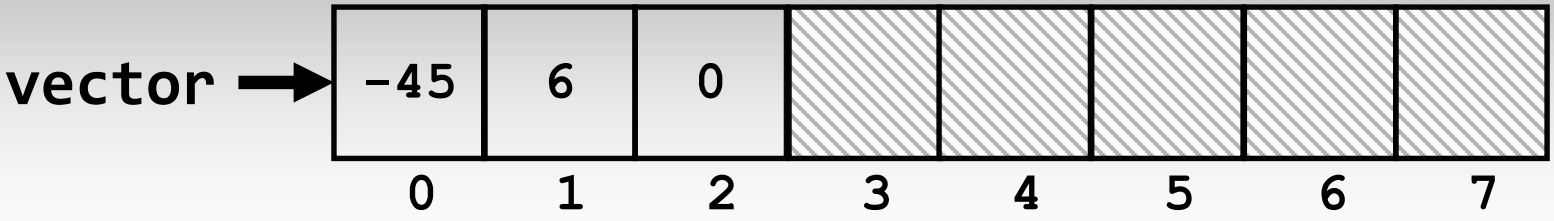
```
int* vector = new int[8];
```




```
vector[0] = -45;
vector[1] = 6;
vector[2] = 0;
```

Adicionar en una lista basadas en *arrays*

```
int* vector = new int[8];
```




```
vector[0] = -45;
vector[1] = 6;
vector[2] = 0;
```

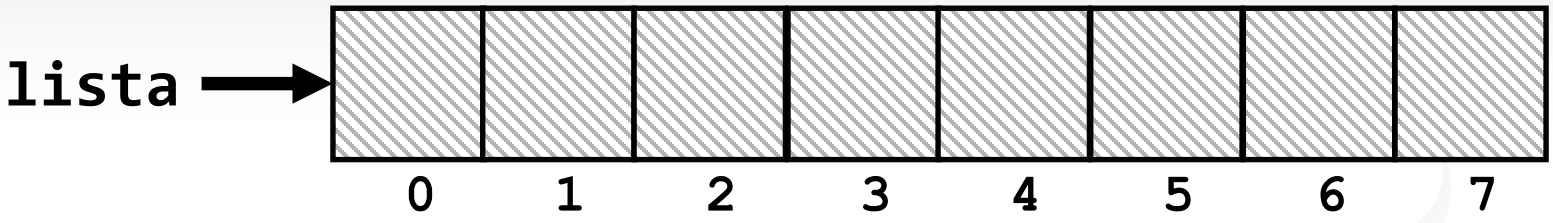
- Precisamos una variable para tener control de cuántos valores se han agregado a nuestra lista (**`longitud`**)
- Ya antes deberíamos tener también una variable para tener control del tamaño en memoria del vector (**`tamañoMax`**)

Adicionar en una lista basadas en *arrays*

```
void adicionar(int* lista, unsigned int &longitud,
               unsigned int tamañoMax, int nuevo) {
  if (longitud < tamañoMax)
    lista[longitud++] = nuevo;
}
```

Adicionar en una lista basadas en *arrays*

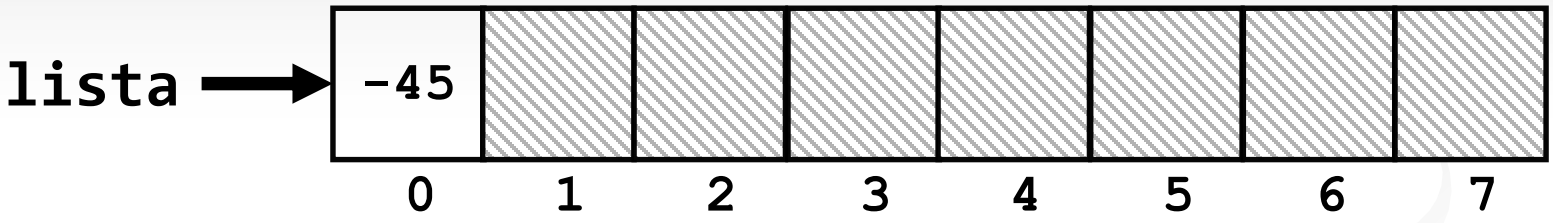
```
unsigned int tamañoMax = 8;
unsigned int longitud = 0;
int* lista = new int[8];
```



longitud: 0

Adicionar en una lista basadas en *arrays*

```
unsigned int tamañoMax = 8;
unsigned int longitud = 0;
int* lista = new int[8];
```

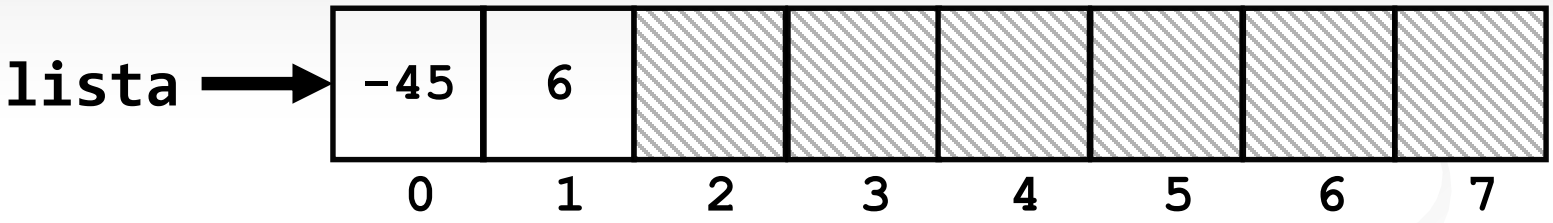


longitud: 1

```
adicionar(lista, longitud, tamañoMax, -45);
```

Adicionar en una lista basadas en *arrays*

```
unsigned int tamañoMax = 8;
unsigned int longitud = 0;
int* lista = new int[8];
```

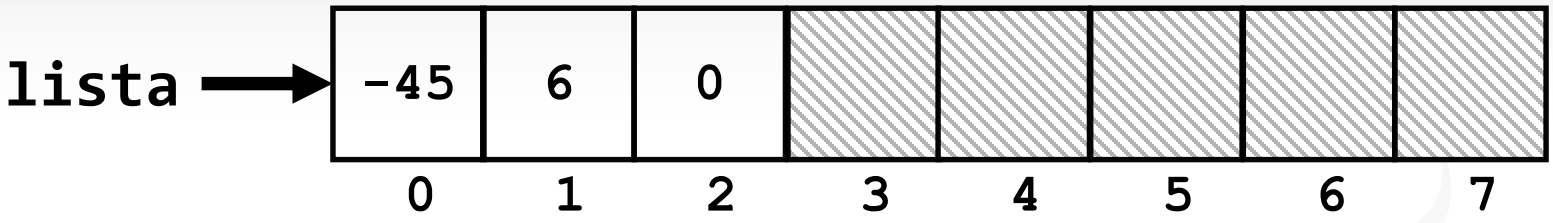


longitud: 2

```
adicionar(lista, longitud, tamañoMax, -45);
adicionar(lista, longitud, tamañoMax, 6);
```

Adicionar en una lista basadas en *arrays*

```
unsigned int tamañoMax = 8;
unsigned int longitud = 0;
int* lista = new int[8];
```

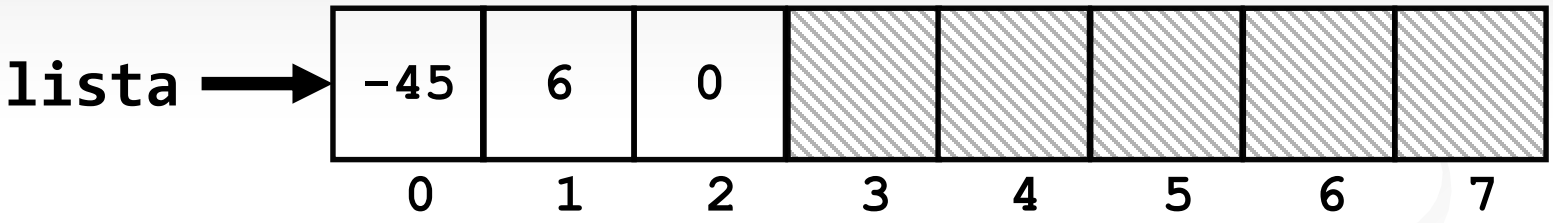


longitud: 3

```
adicionar(lista, longitud, tamañoMax, -45);
adicionar(lista, longitud, tamañoMax, 6);
adicionar(lista, longitud, tamañoMax, 0);
```

Adicionar en una lista basadas en *arrays*

```
unsigned int tamañoMax = 8;
unsigned int longitud = 0;
int* lista = new int[8];
```

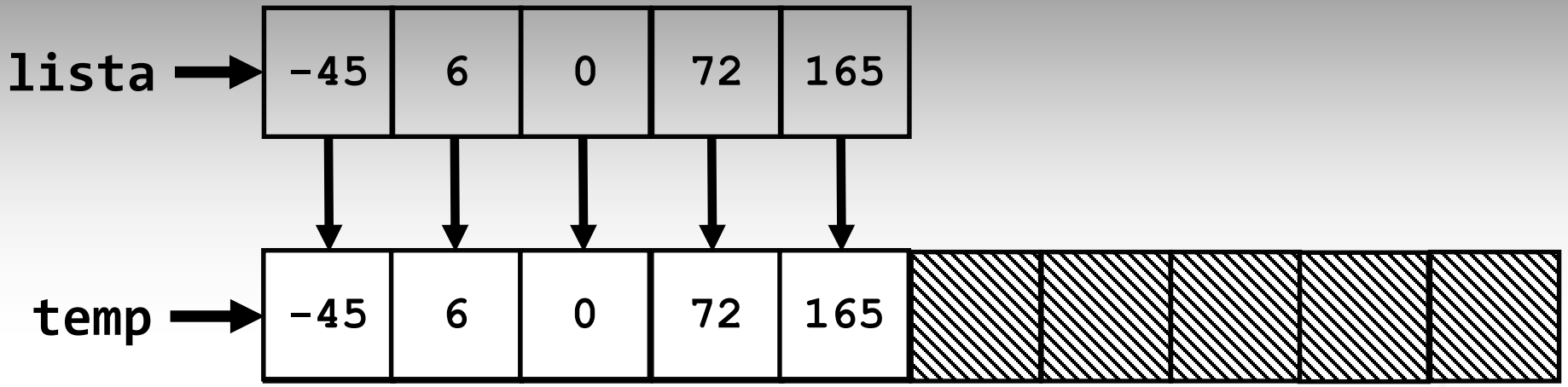


longitud: 3

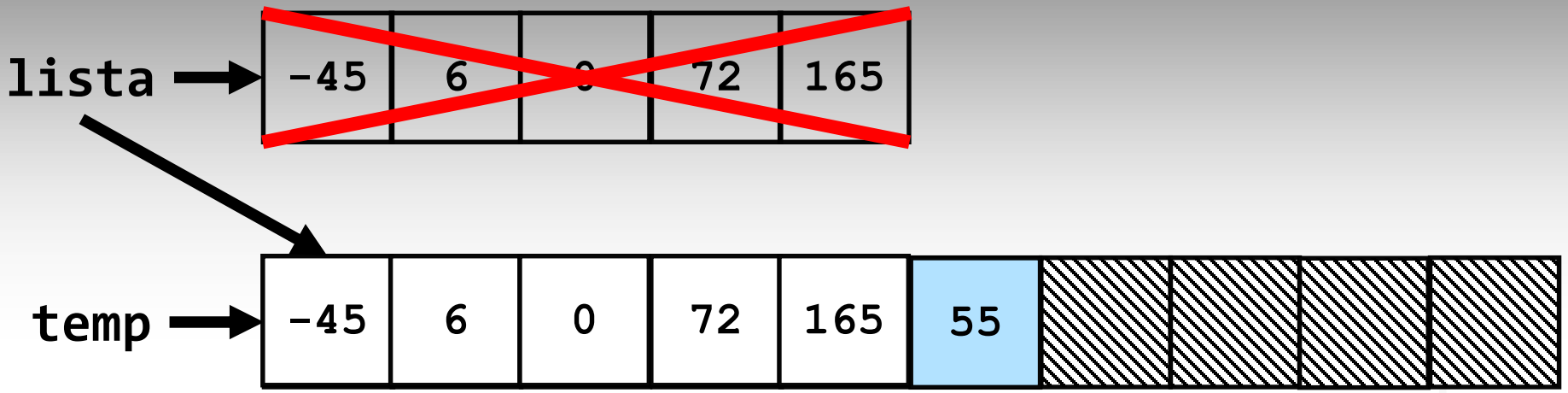
```
adicionar(lista, longitud, tamañoMax, -45);
adicionar(lista, longitud, tamañoMax, 6);
adicionar(lista, longitud, tamañoMax, 0);
```

¿Y si precisamos que la lista tenga más elementos que el establecido por el tamaño máximo?

Adicionar en una lista basadas en *arrays*



Adicionar en una lista basadas en *arrays*

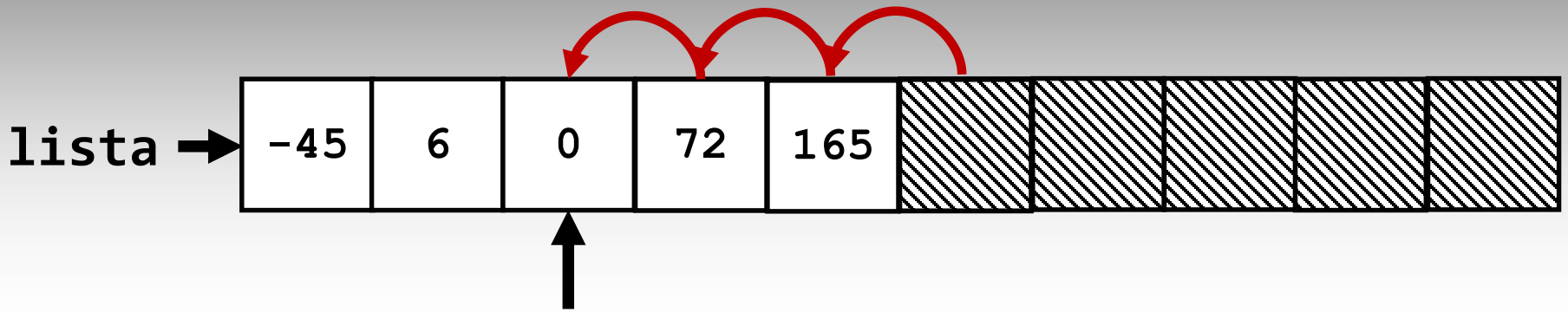


En una lista dinámica basada en arrays, la operación de adicionar un nuevo elemento toma un tiempo de ejecución O(*n*)

Adicionar en una lista basadas en *arrays*

```
void adicionar(int* &lista, unsigned int &longitud,
               unsigned int &tamañoMax, int nuevo) {
  if (longitud == tamañoMax){
    incrementar(tamañoMax);
    int temp = new int[tamañoMax];
    for(int i = 0; i < longitud; i++)
      temp[i] = lista[i];
    delete [] lista;
    lista = temp;
  }
  lista[longitud++] = nuevo;
}
```

Eliminar en una lista basadas en *arrays*




```
void eliminar(int* lista, unsigned int &longitud,
                                  unsigned int pos) {
  for(int i = pos+1; i < longitud; i++)
      lista[i-1] = lista[i];
  longitud--;
}
```

Desventajas de las listas basadas en arrays

- El array debe crearse con un tamaño fijo.
- Puede ser que solo se use una parte del array, y quedaría memoria reservada sin usar
- Para ampliar su tamaño hay que crear un nuevo array y copiar todos los elementos (alto consumo de tiempo y memoria)

¿Qué necesitamos?

Poder implementar las listas de modo que puedan crecer en la medida de nuestras necesidades, y que utilice la memoria de forma más eficiente.

Estructuras Dinámicas vs Estructuras Estáticas

Estructuras estáticas: Poseen como característica común el tener tamaño fijo, el cual es declarado en tiempo de compilación.

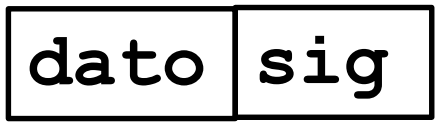
Estructuras dinámicas: No es necesario especificarle su tamaño. Pueden contraerse o expandirse durante la ejecución del programa. De forma explícita se va asignando(o liberando) memoria para los componentes individuales de la estructura de datos)

Listas enlazadas

Una ***lista enlazada*** es una lista en la cual cada elemento contiene dentro de sí la dirección del elemento siguiente.

Un ***nodo*** es una ***estructura*** que almacena dentro el dato de la lista y la ***dirección*** del nodo que contiene el elemento siguiente en la lista.

En C++ representaremos estos nodos mediante registros (***struct***)




```
struct NodoLista {
  int dato;
  NodoLista *sig;  //Autoreferencia
  NodoLista(): dato(0), sig(NULL) {}
  NodoLista(int d): dato(d), sig(NULL) {}
};
```

Estructuras en C++ (Structs)

Los ***structs*** agrupan un conjunto de miembros de diversos tipos en nuevo tipo compuesto.

```
struct Producto {
  char* nombre;
  float precio;
  int cantidad;
};
int main(){
  Producto p;
  p.nombre = "Mesa";
  p.precio = 3.0;
  p.cantidad = 5;
  cout << p.nombre << endl;
  return 0;
}
```

- Cantidad fija de campos.
- Se accede a los elementos por nombre, no por índice.
- Los campos pueden ser de diferentes tipos de datos

Punteros a Struct

Se usan los operadores **new** y **delete** para crear y destruir.

Se accede a los componentes mediante el operador **->**

Los structs pueden tener varios ***constructores***.

```
struct Point {
  int x;
  int y;
  Point():x(0), y(0){}
  Point(int x, int y)
       :x(x), y(y){}
};
```

```
int main(){
  Point *p1 = new Point(3,4);
  Point *p2 = new Point();
  p2->x = p1->x + 2;
  p2->y = p1->y - 2;
  cout << p2->x << endl;
  cout << p2->y << endl;
  delete p1;
  delete p2;
  return 0;
}
```

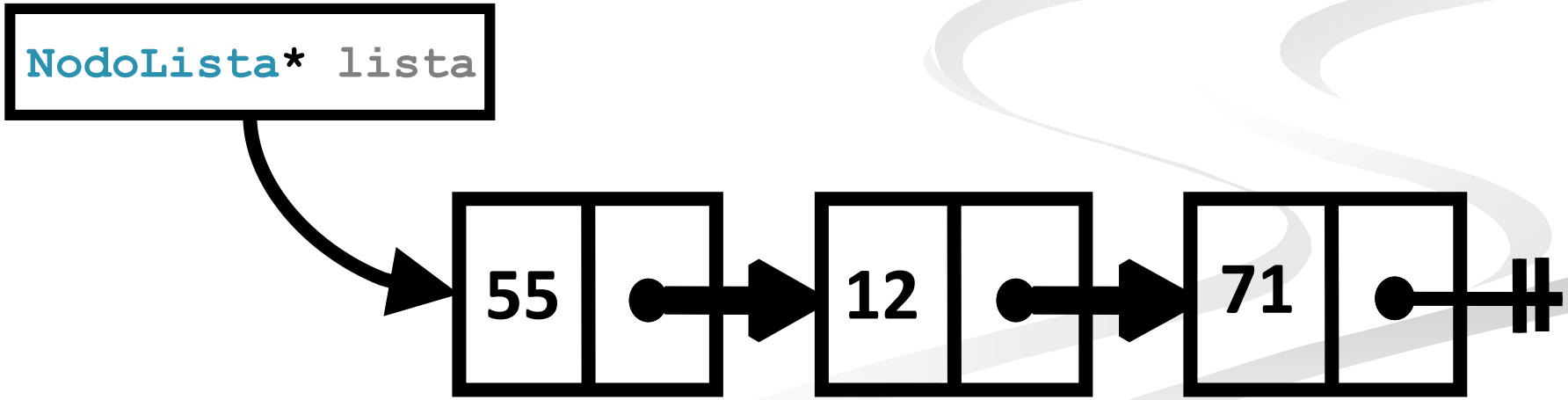
Listas enlazadas

En las listas enlazadas es suficiente con conocer la dirección del primer nodo.

El último nodo se identifica porque el valor de su campo **sig** es **NULL**

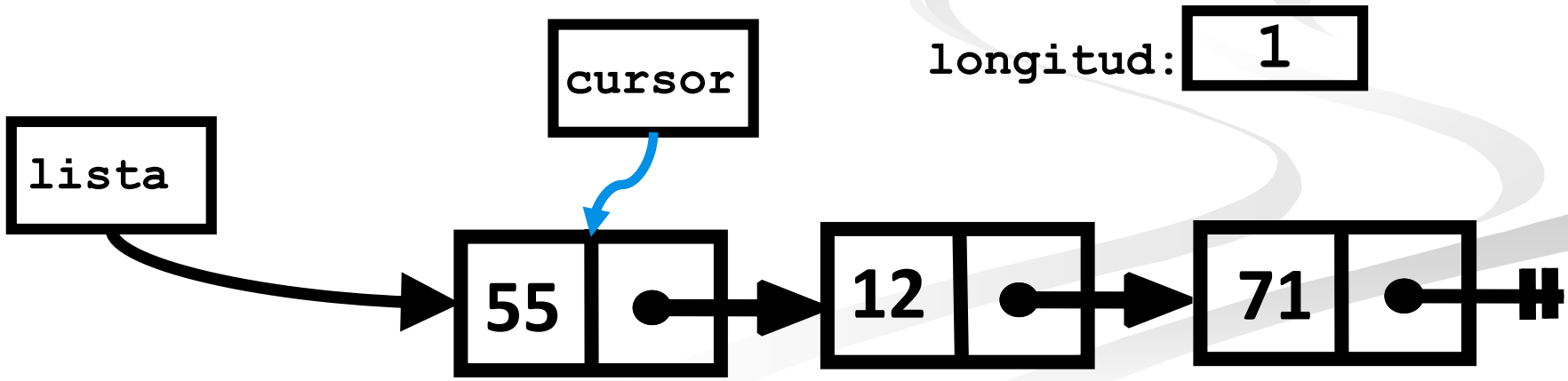
L = (55,12,71)

```
struct NodoLista {
  int dato;
  NodoLista *sig;
  NodoLista(): dato(0),
               sig(NULL) {}
  NodoLista(int d):
  dato(d), sig(NULL) {}
};
```



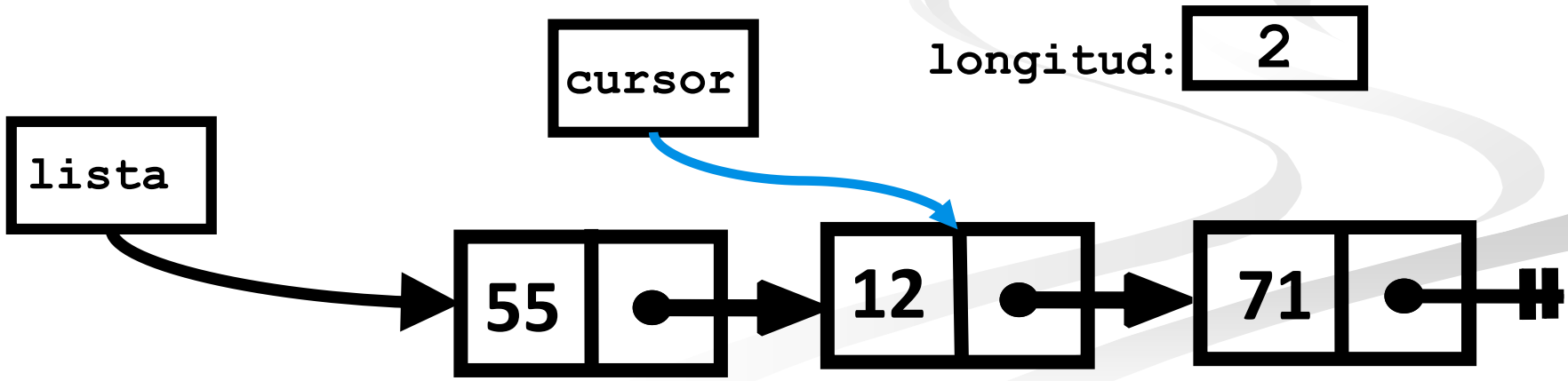
Listas enlazadas: Longitud

```
int longitudLista(NodoLista* lista)
{
  int longitud = 0;
  NodoLista* cursor = lista;
  while (cursor != NULL) {
    longitud++;
    cursor = cursor->sig;
  }
  return longitud;
}
```



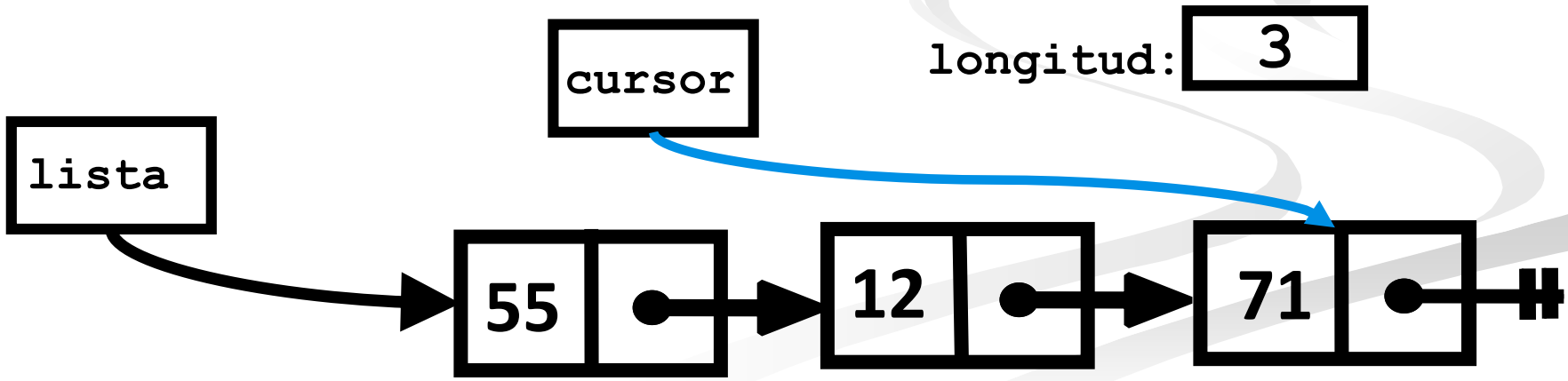
Listas enlazadas: Longitud

```
int longitudLista(NodoLista* lista)
{
  int longitud = 0;
  NodoLista* cursor = lista;
  while (cursor != NULL) {
    longitud++;
    cursor = cursor->sig;
  }
  return longitud;
}
```



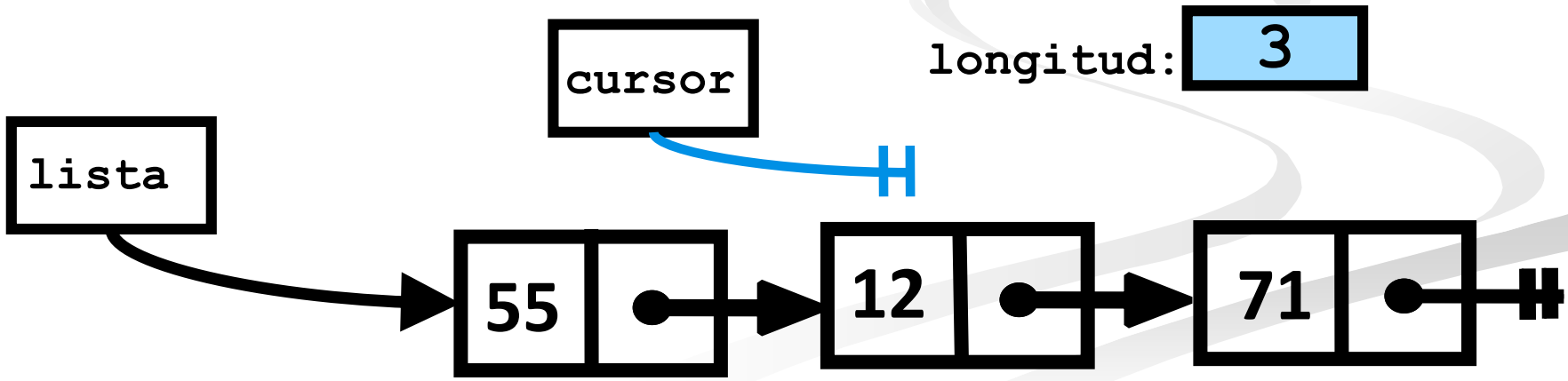
Listas enlazadas: Longitud

```
int longitudLista(NodoLista* lista)
{
  int longitud = 0;
  NodoLista* cursor = lista;
  while (cursor != NULL) {
    longitud++;
    cursor = cursor->sig;
  }
  return longitud;
}
```



Listas enlazadas: Longitud

```
int longitudLista(NodoLista* lista)
{
  int longitud = 0;
  NodoLista* cursor = lista;
  while (cursor != NULL) {
    longitud++;
    cursor = cursor->sig;
  }
  return longitud;
}
```



Listas enlazadas: Adicionar al final

Dada una lista y un valor **X**, se desea adicionar a **X** al final de la lista.

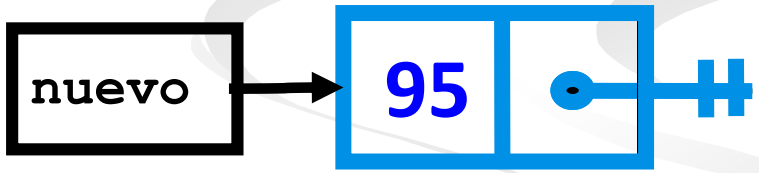
```
L = (55,12,71)
```

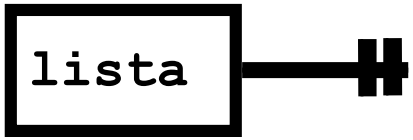
Adicionar(L,95)

```
L = (55,12,71,95)
```

Listas enlazadas: Adicionar al final

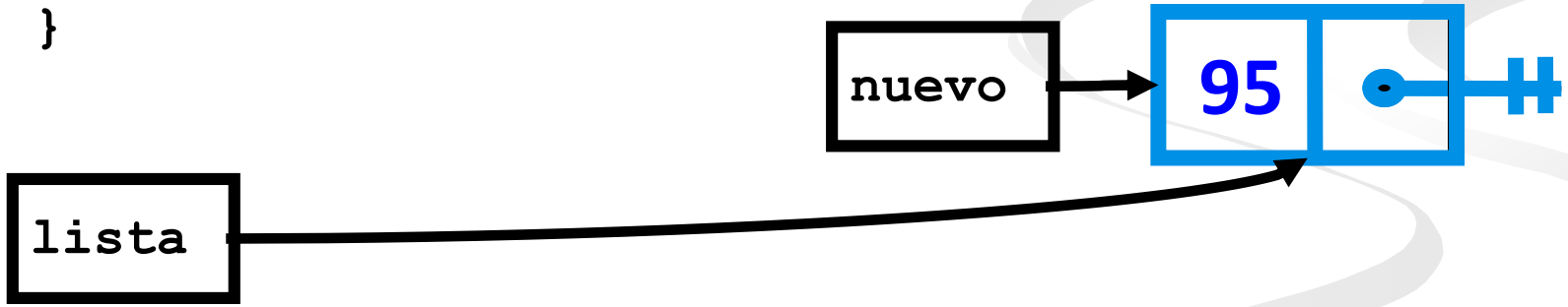
```
void adicionar(NodoLista*& lista, int x){
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    NodoLista* cursor = lista;
    while (cursor->sig != NULL)
      cursor = cursor->sig;
    cursor->sig = nuevo;
  }
}
```





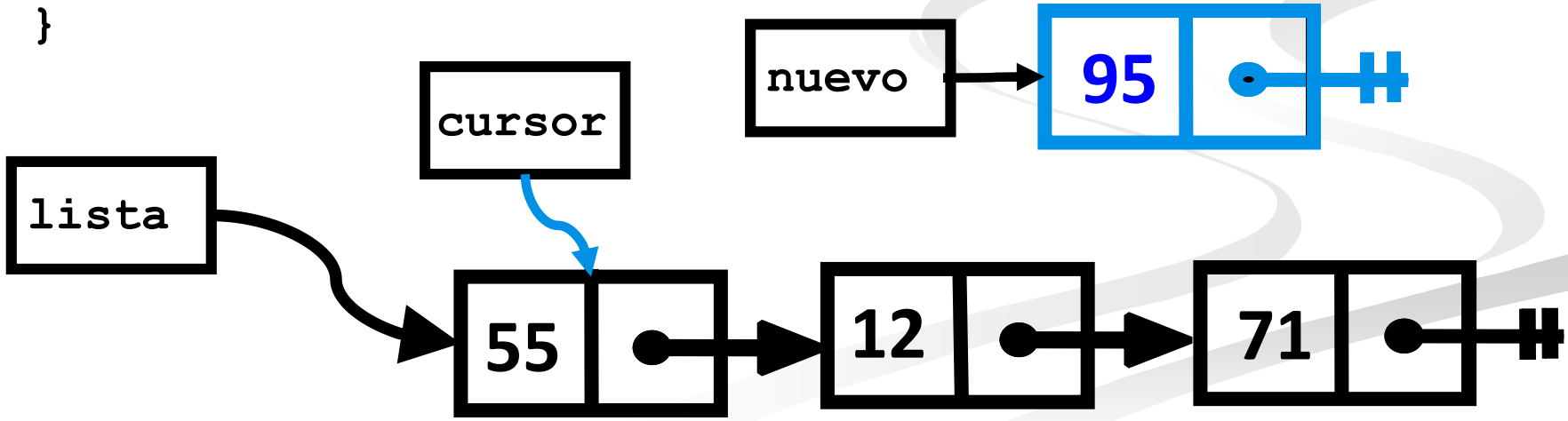
Listas enlazadas: Adicionar al final

```
void adicionar(NodoLista*& lista, int x){
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    NodoLista* cursor = lista;
    while (cursor->sig != NULL)
      cursor = cursor->sig;
    cursor->sig = nuevo;
  }
}
```



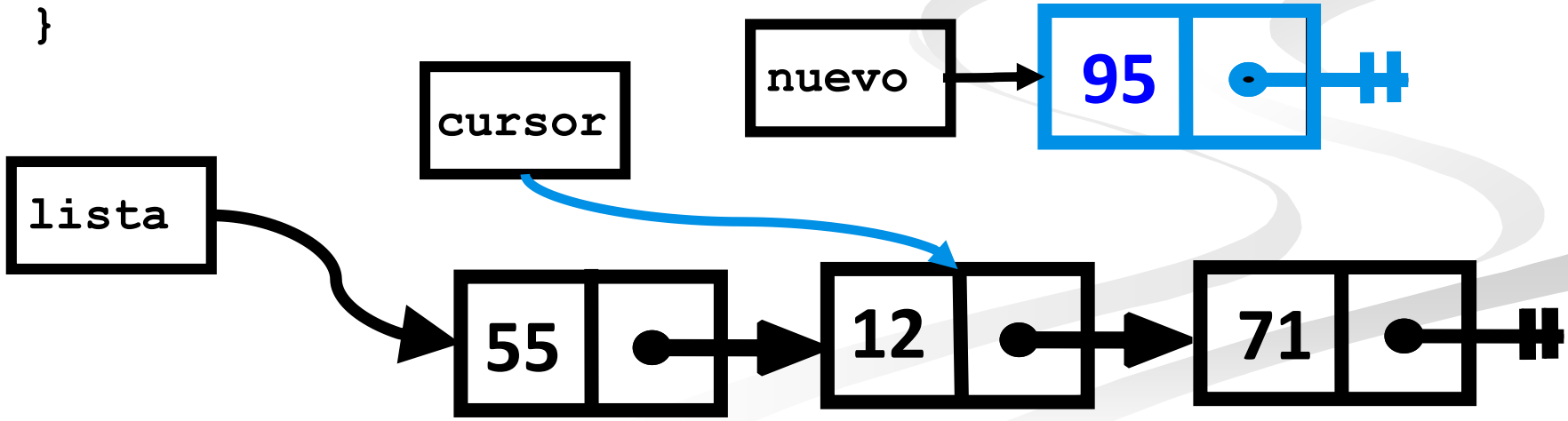
Listas enlazadas: Adicionar al final

```
void adicionar(NodoLista*& lista, int x){
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    NodoLista* cursor = lista;
    while (cursor->sig != NULL)
      cursor = cursor->sig;
    cursor->sig = nuevo;
  }
}
```



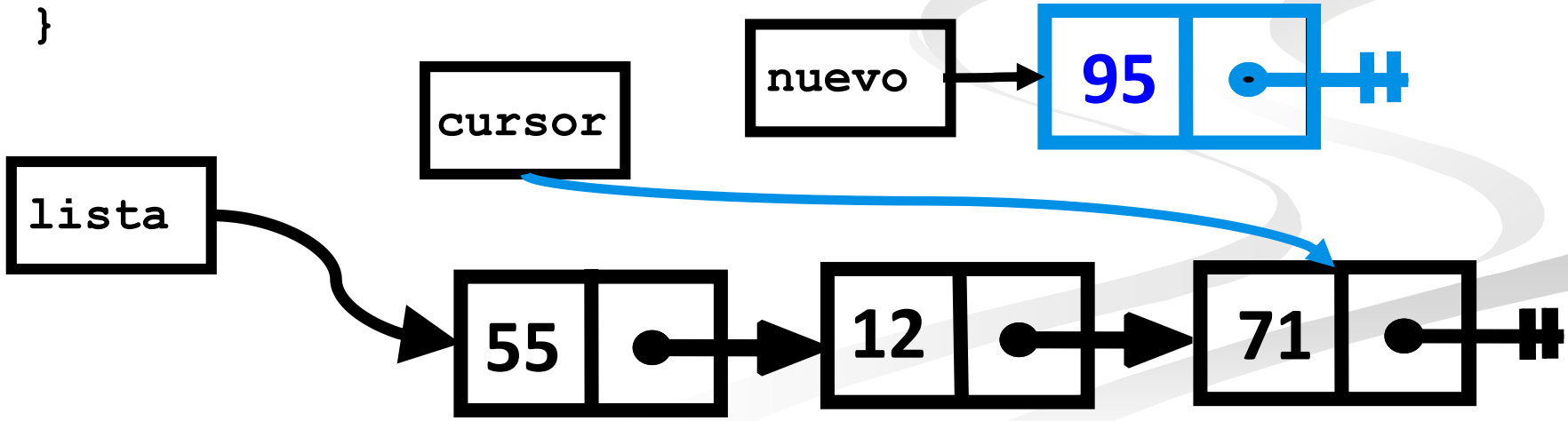
Listas enlazadas: Adicionar al final

```
void adicionar(NodoLista*& lista, int x){
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    NodoLista* cursor = lista;
    while (cursor->sig != NULL)
      cursor = cursor->sig;
    cursor->sig = nuevo;
  }
}
```



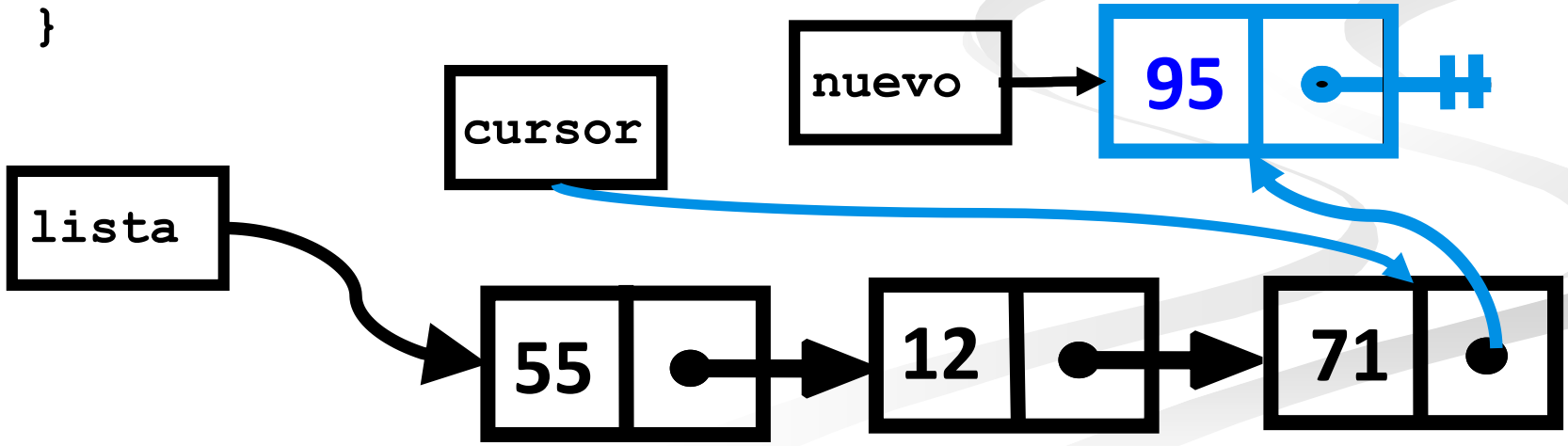
Listas enlazadas: Adicionar al final

```
void adicionar(NodoLista*& lista, int x){
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    NodoLista* cursor = lista;
    while (cursor->sig != NULL)
      cursor = cursor->sig;
    cursor->sig = nuevo;
  }
}
```



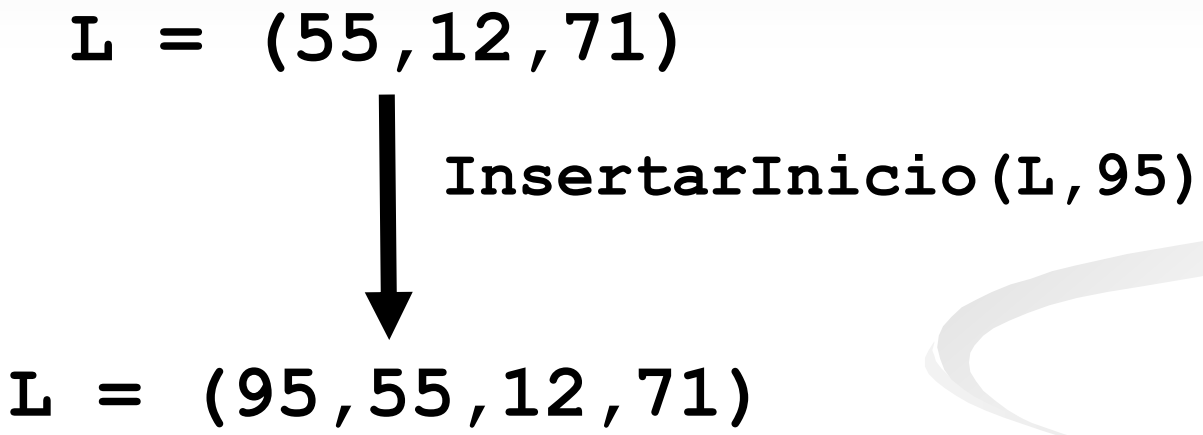
Listas enlazadas: Adicionar al final

```
void adicionar(NodoLista*& lista, int x){
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    NodoLista* cursor = lista;
    while (cursor->sig != NULL)
      cursor = cursor->sig;
    cursor->sig = nuevo;
  }
}
```



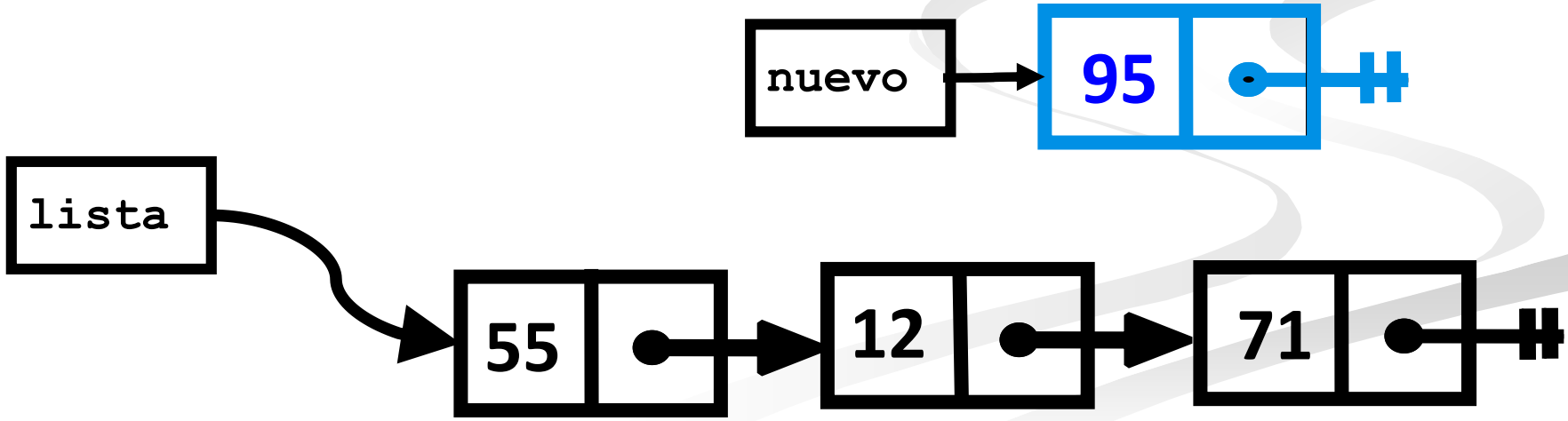
Listas enlazadas: Inserción al inicio

Dada una lista y un valor **X**, se desea insertar a **X** al inicio de la lista.



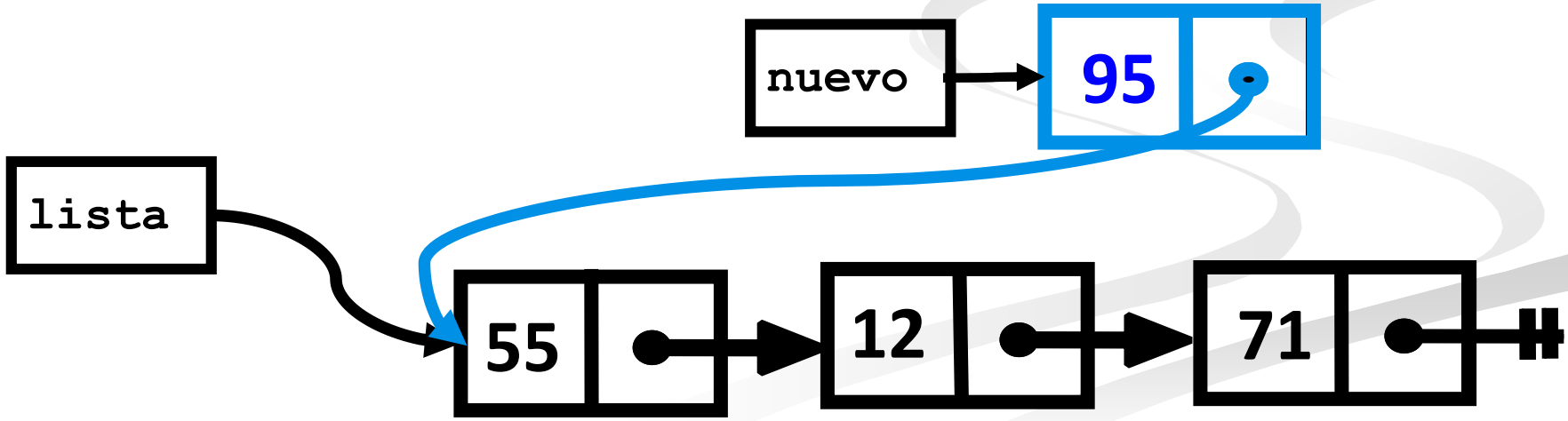
Listas enlazadas: Inserción primera posición

```
void insertarInicio(NodoLista*& lista, int x)
{
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    nuevo->sig = lista;
    lista = nuevo;
  }
}
```



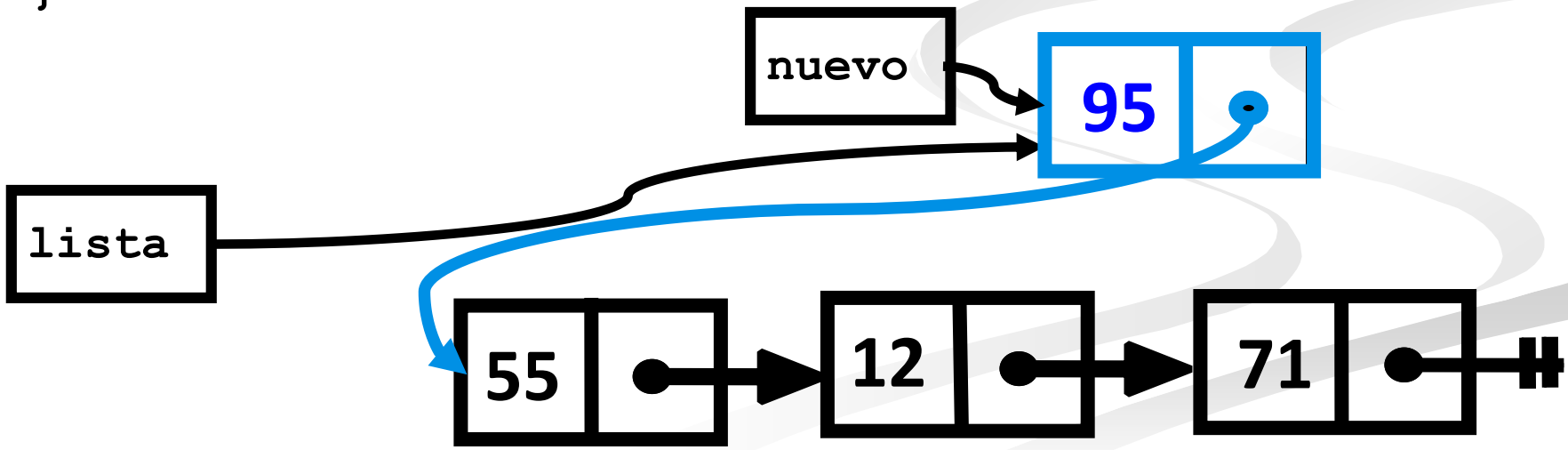
Listas enlazadas: Inserción primera posición

```
void insertarInicio(NodoLista*& lista, int x)
{
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    nuevo->sig = lista;
    lista = nuevo;
  }
}
```



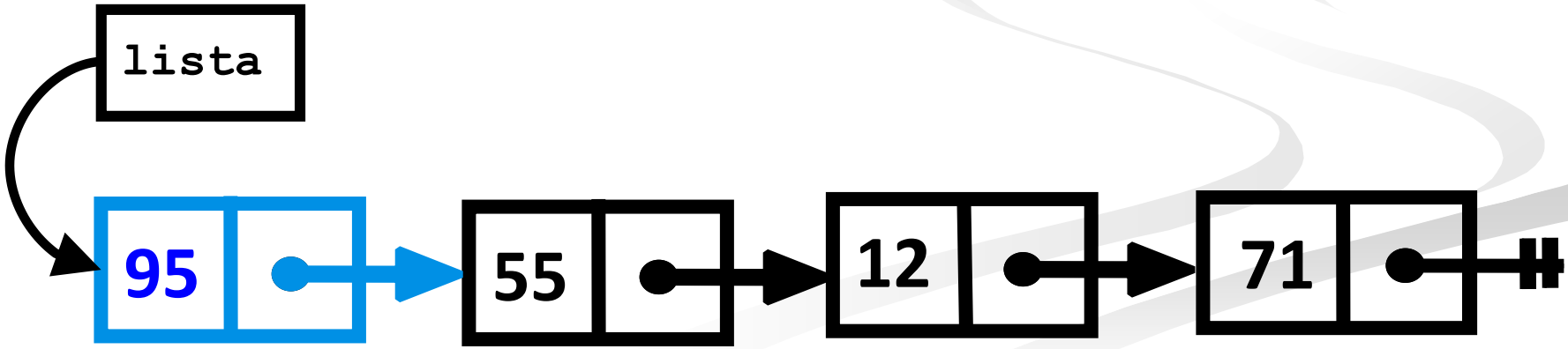
Listas enlazadas: Inserción primera posición

```
void insertarInicio(NodoLista*& lista, int x)
{
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    nuevo->sig = lista;
    lista = nuevo;
  }
}
```



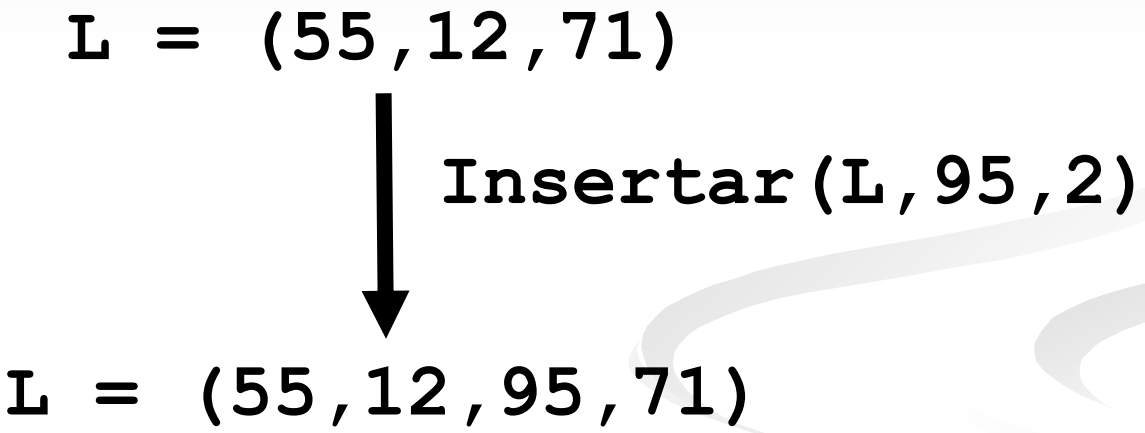
Listas enlazadas: Inserción primera posición

```
void insertarInicio(NodoLista*& lista, int x)
{
  NodoLista* nuevo = new NodoLista(x);
  if (lista == NULL)
    lista = nuevo;
  else {
    nuevo->sig = lista;
    lista = nuevo;
  }
}
```



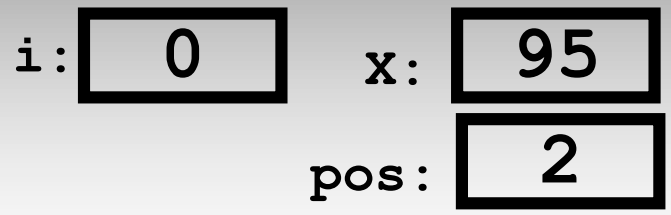
Listas enlazadas: Inserción en cualquier posición

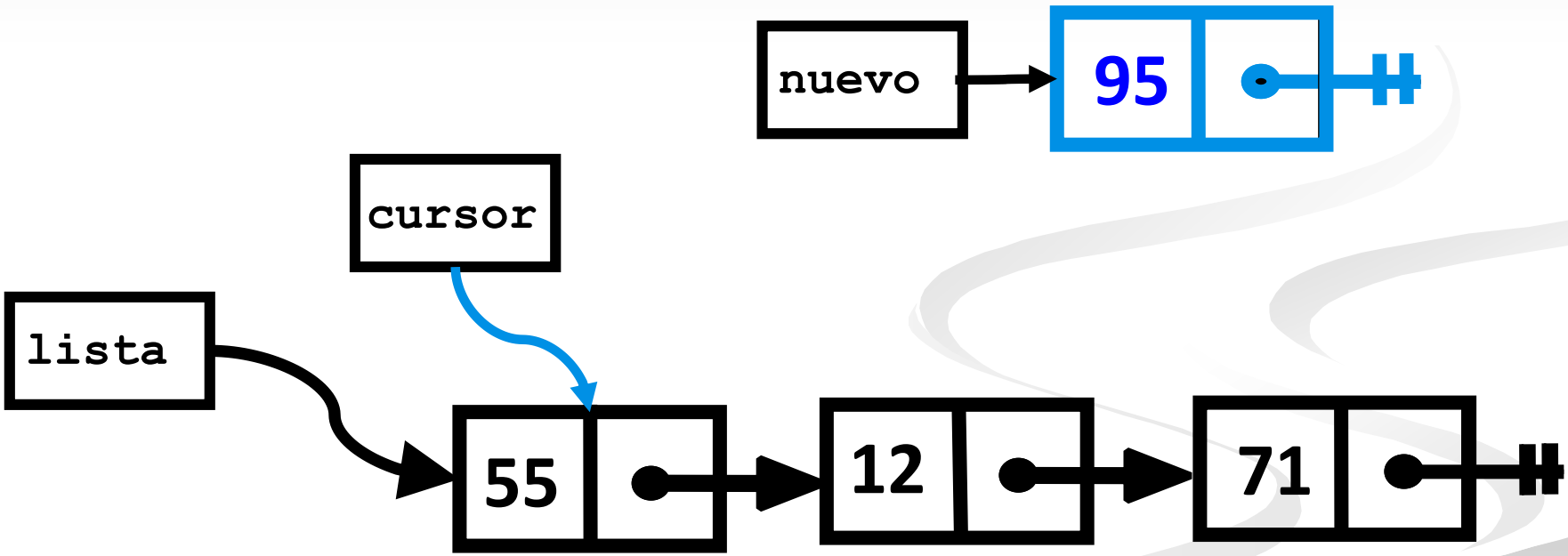
Dada una lista, un valor **X** y un valor entero **pos** que indique una posición relativa a la lista, se desea insertar a **X** en la posición **pos** de la lista.



Listas enlazadas: Inserción en cualquier posición

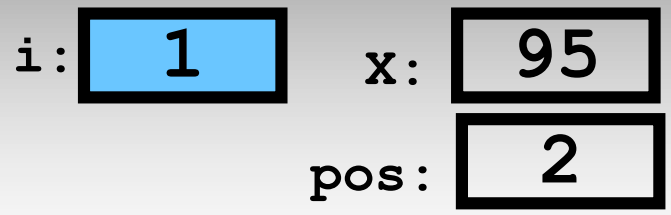
Avanzar el cursor hasta ubicarlo en la posición anterior a **`pos`** (mientras **`i+1 < pos`**)

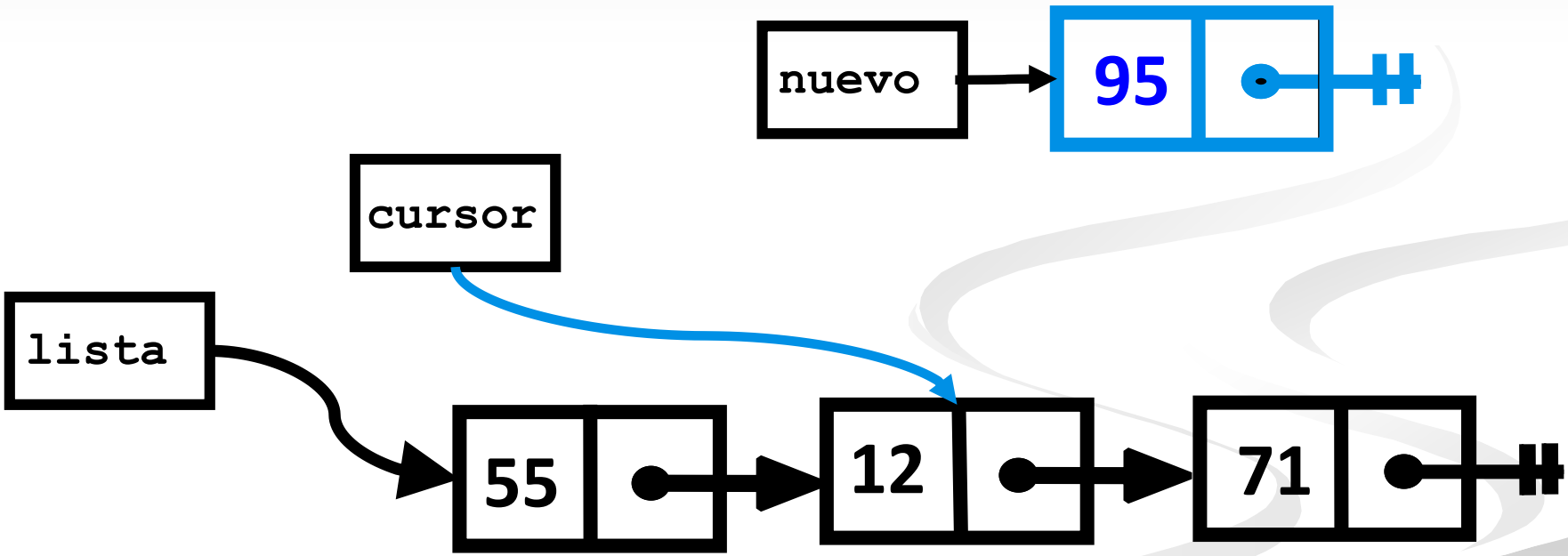




Listas enlazadas: Inserción en cualquier posición

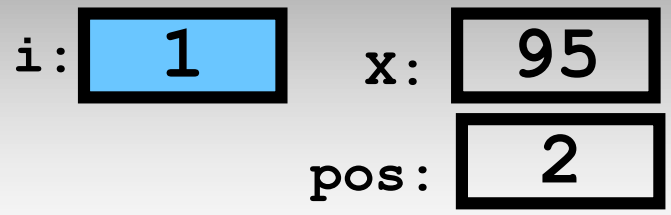
Avanzar el cursor hasta ubicarlo en la posición anterior a **`pos`** (mientras **`i+1 < pos`**)

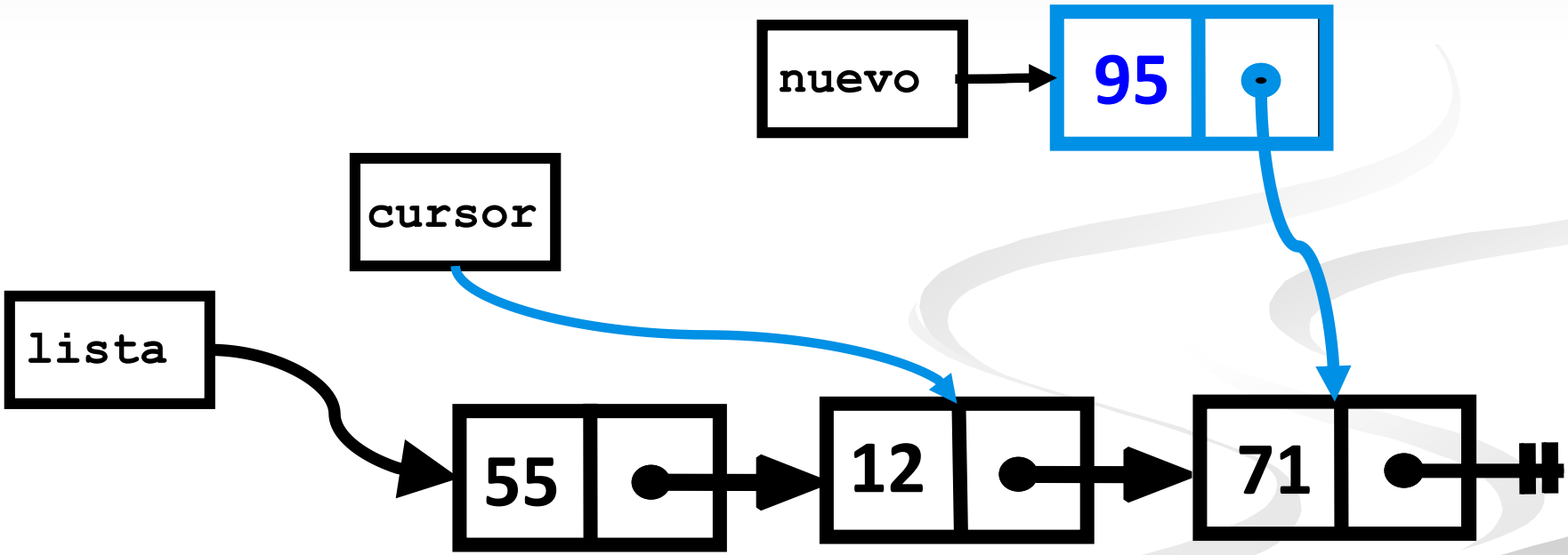




Listas enlazadas: Inserción en cualquier posición

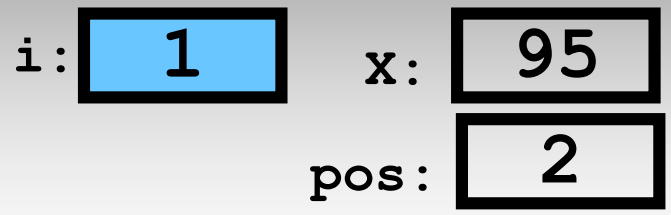
Avanzar el cursor hasta ubicarlo en la posición anterior a **`pos`** (mientras **`i+1 < pos`**)

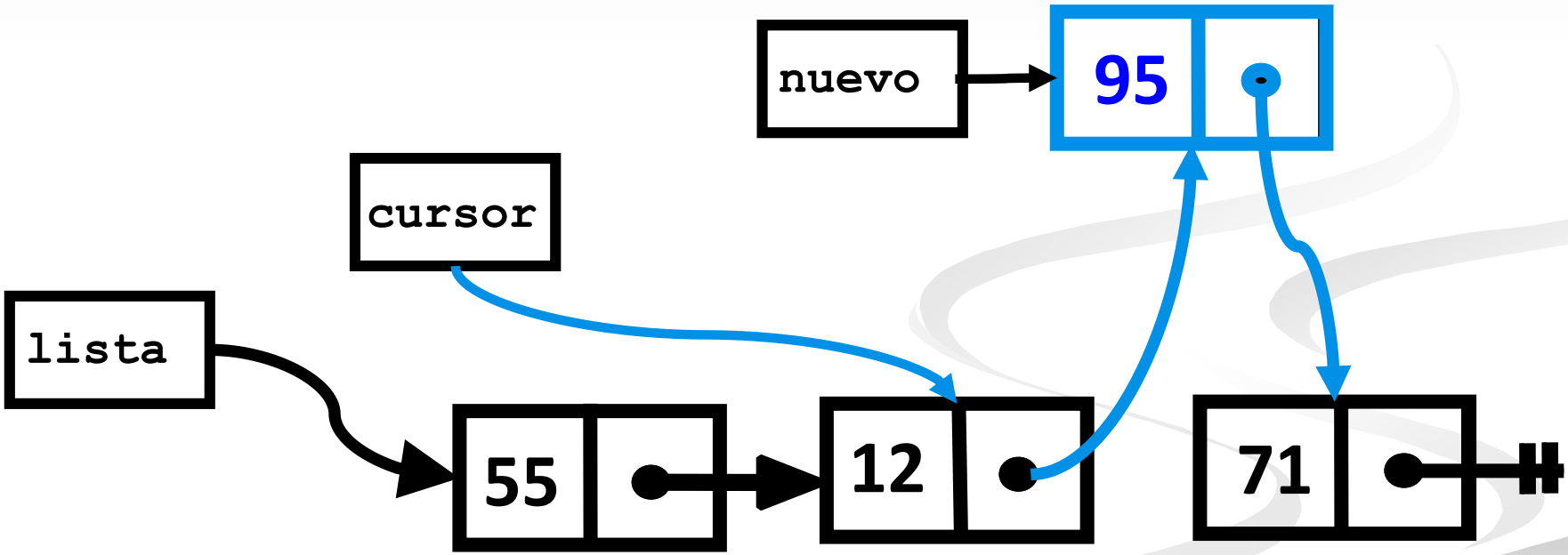




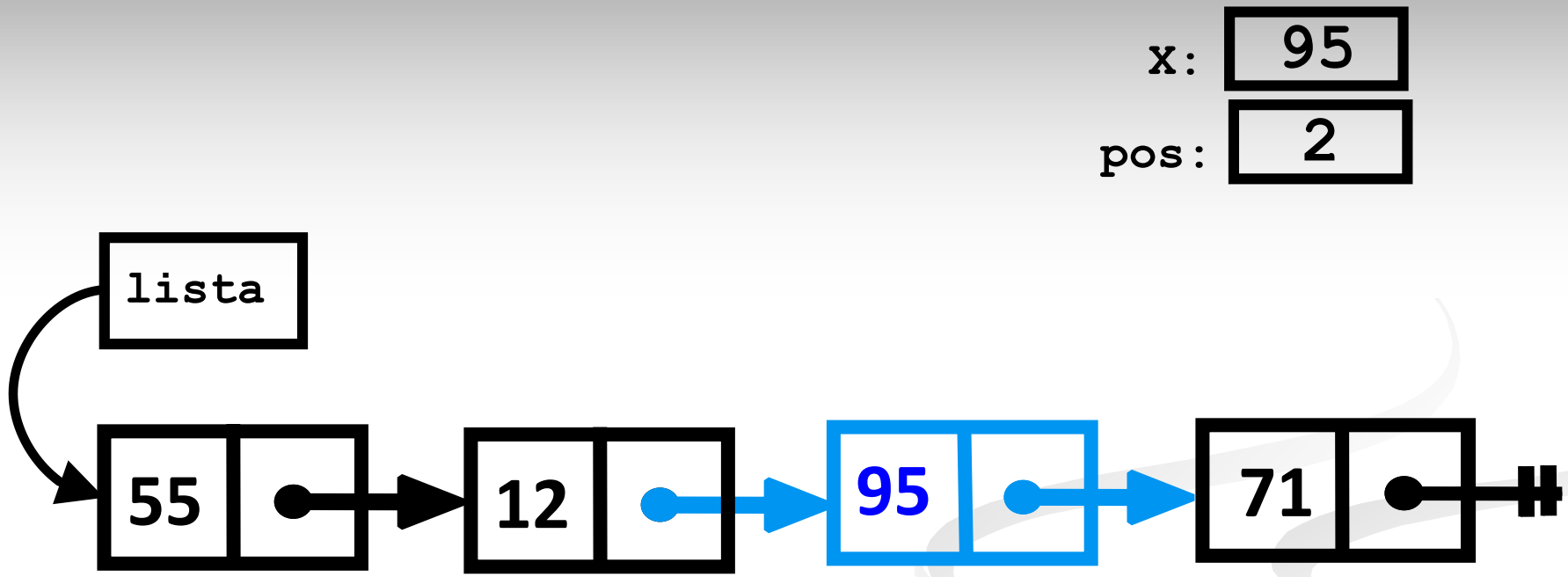
Listas enlazadas: Inserción en cualquier posición

Avanzar el cursor hasta ubicarlo en la posición anterior a **`pos`** (mientras **`i+1 < pos`**)





Listas enlazadas: Inserción en cualquier posición

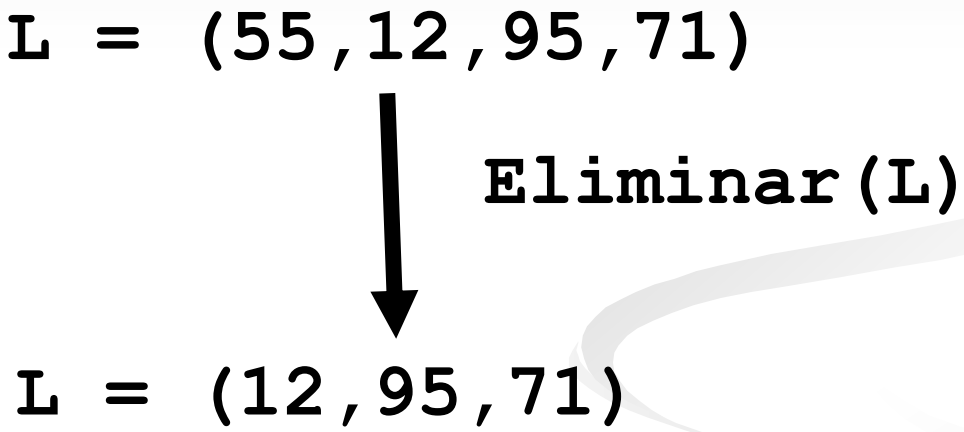


Listas enlazadas: Inserción en cualquier posición

```
void insertar(NodoLista*& lista, int X, int pos){
  NodoLista* nuevo = new NodoLista(X);
  if (pos == 0){
    nuevo->sig = lista;
    lista = nuevo;
  }
  else {
    int i = 0;
    NodoLista* cursor = lista;
    while ((cursor != NULL)&&(i+1 < pos)){
      cursor = cursor->sig;
      i++;
    }
    nuevo->sig = cursor->sig;
    cursor->sig = nuevo;
  }
}
```

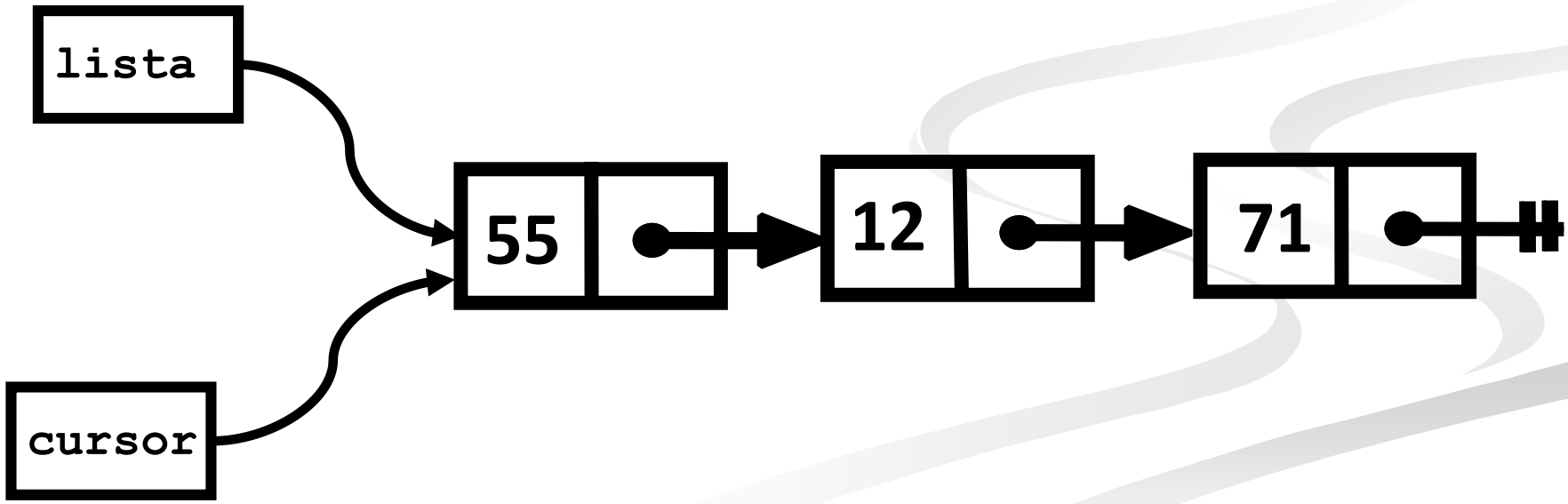
Listas enlazadas: Eliminar primera posición

Dada una lista, se desea eliminar el elemento que ocupa la primera posición de la lista.



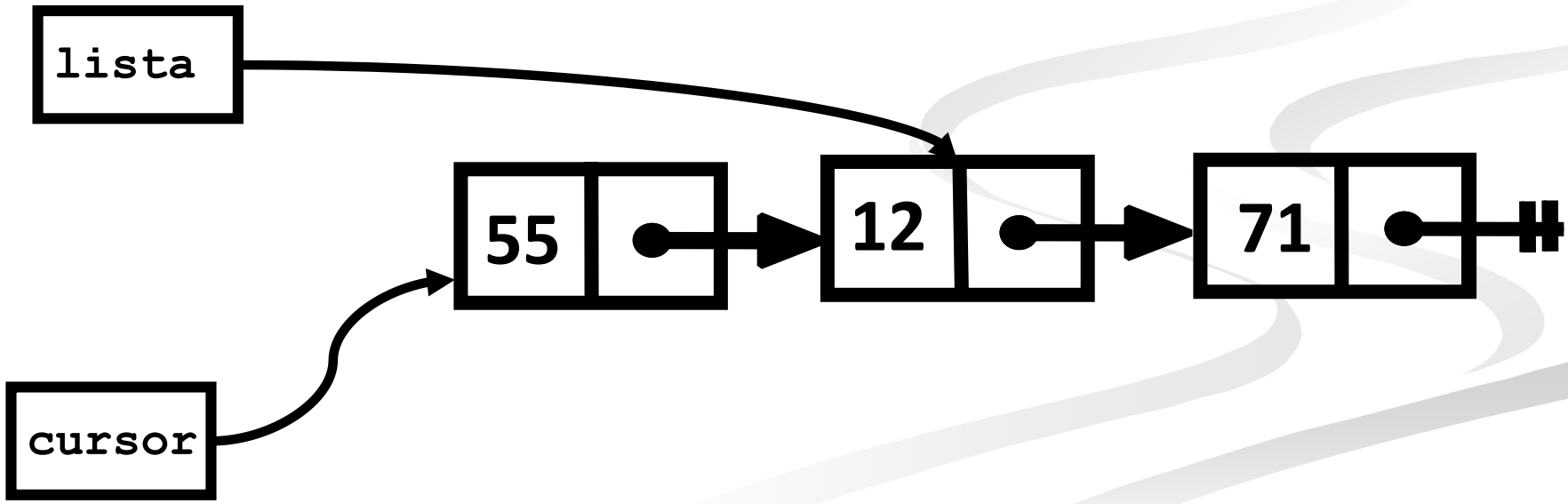
Listas enlazadas: Eliminar primera posición

```
void eliminarPrimeraPos(NodoLista*& lista)
{
  if (lista == NULL) return;
  NodoLista* cursor = lista;
  lista = lista->sig;
  delete cursor;
}
```



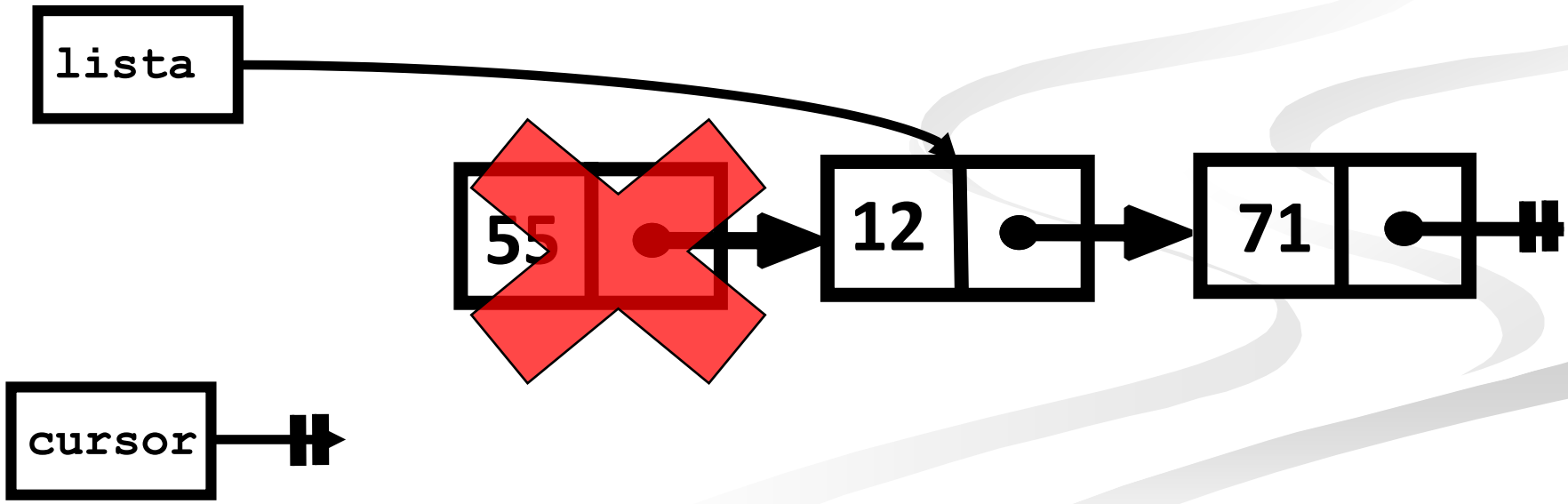
Listas enlazadas: Eliminar primera posición

```
void eliminarPrimeraPos(NodoLista*& lista)
{
  if (lista == NULL) return;
  NodoLista* cursor = lista;
  lista = lista->sig;
  delete cursor;
}
```



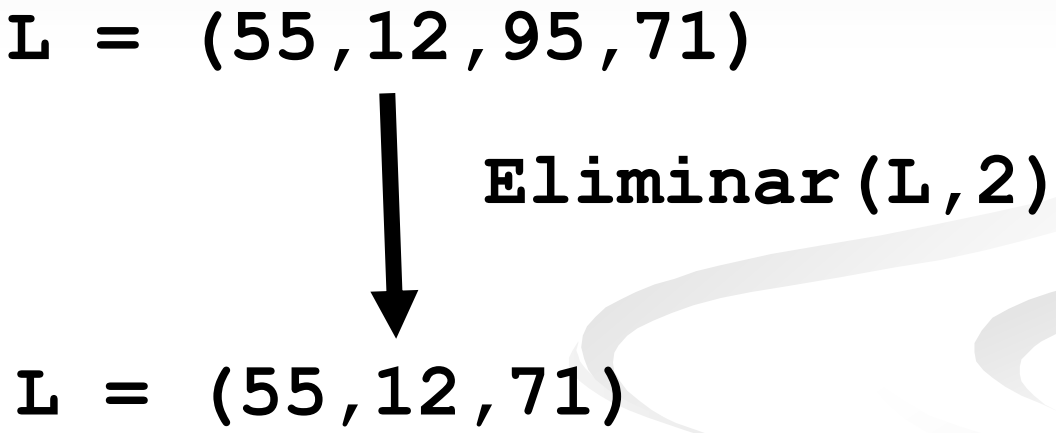
Listas enlazadas: Eliminar primera posición

```
void eliminarPrimeraPos(NodoLista*& lista)
{
  if (lista == NULL) return;
  NodoLista* cursor = lista;
  lista = lista->sig;
  delete cursor;
}
```



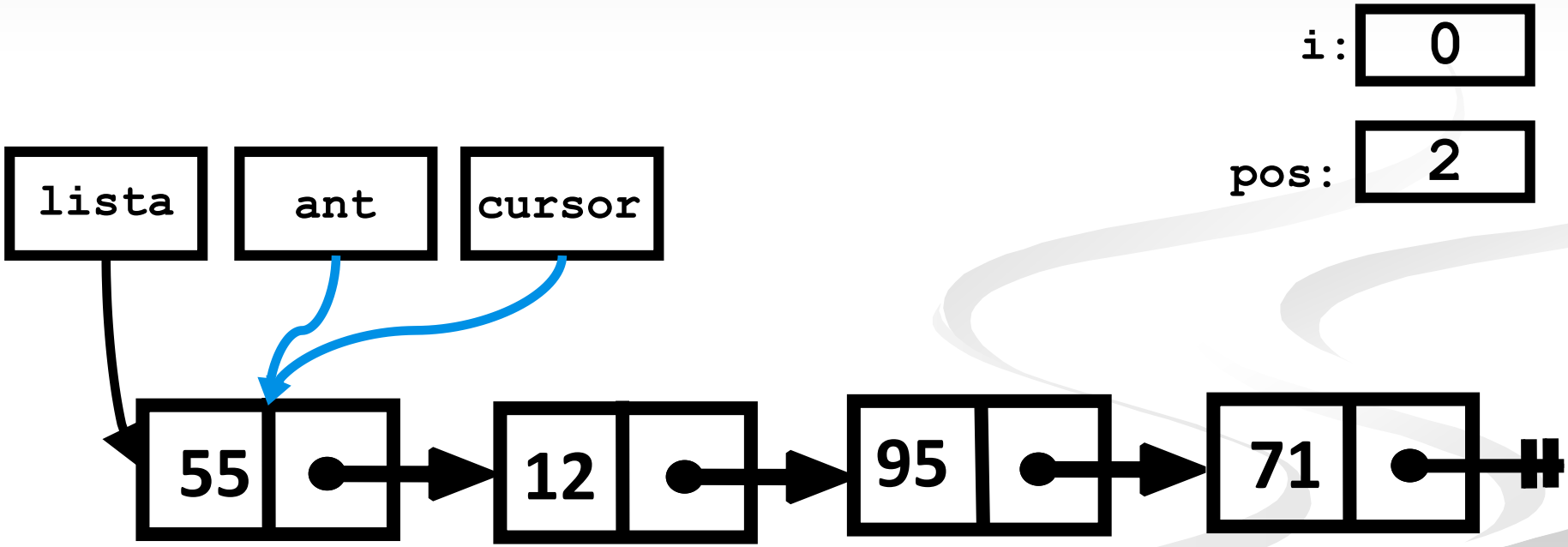
Listas enlazadas: Eliminación en cualquier posición

Dada una lista y un valor entero **pos** que indique una posición relativa a la lista, se desea eliminar el elemento que ocupa la posición **pos** de la lista.



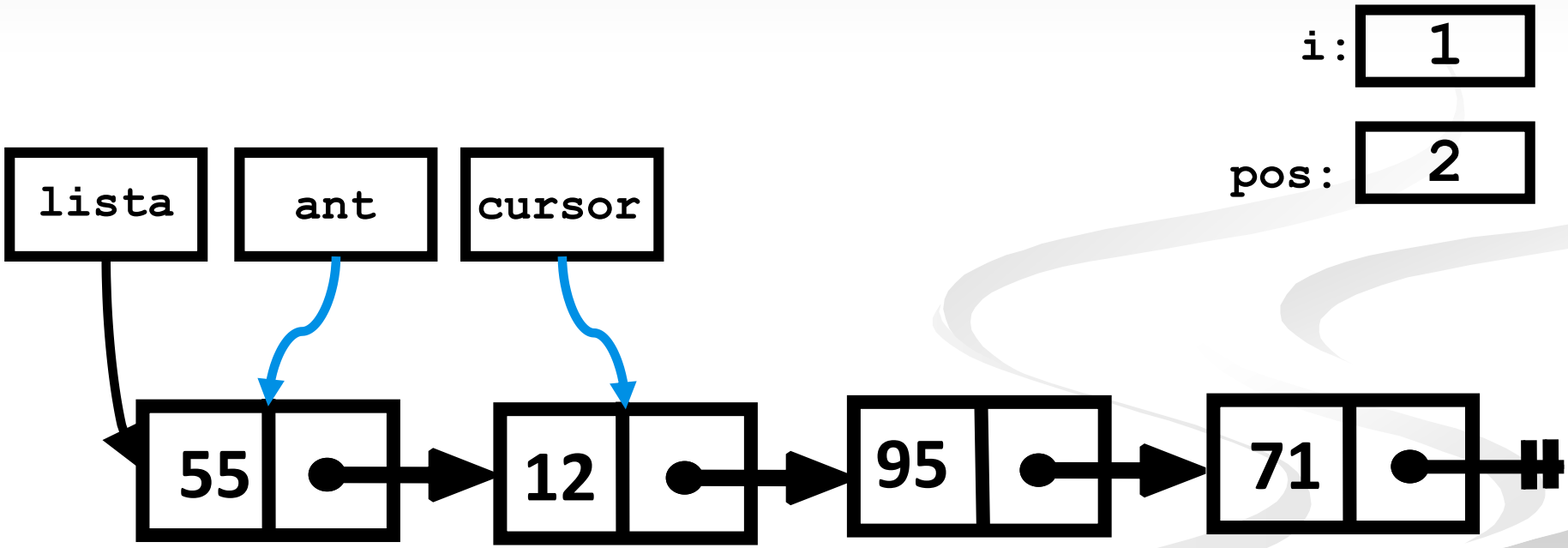
Listas enlazadas: Eliminación en cualquier posición

Es necesario tener dos cursores: uno que avance hasta la posición **`pos`** y otro que llegue hasta la posición **`pos-1`**



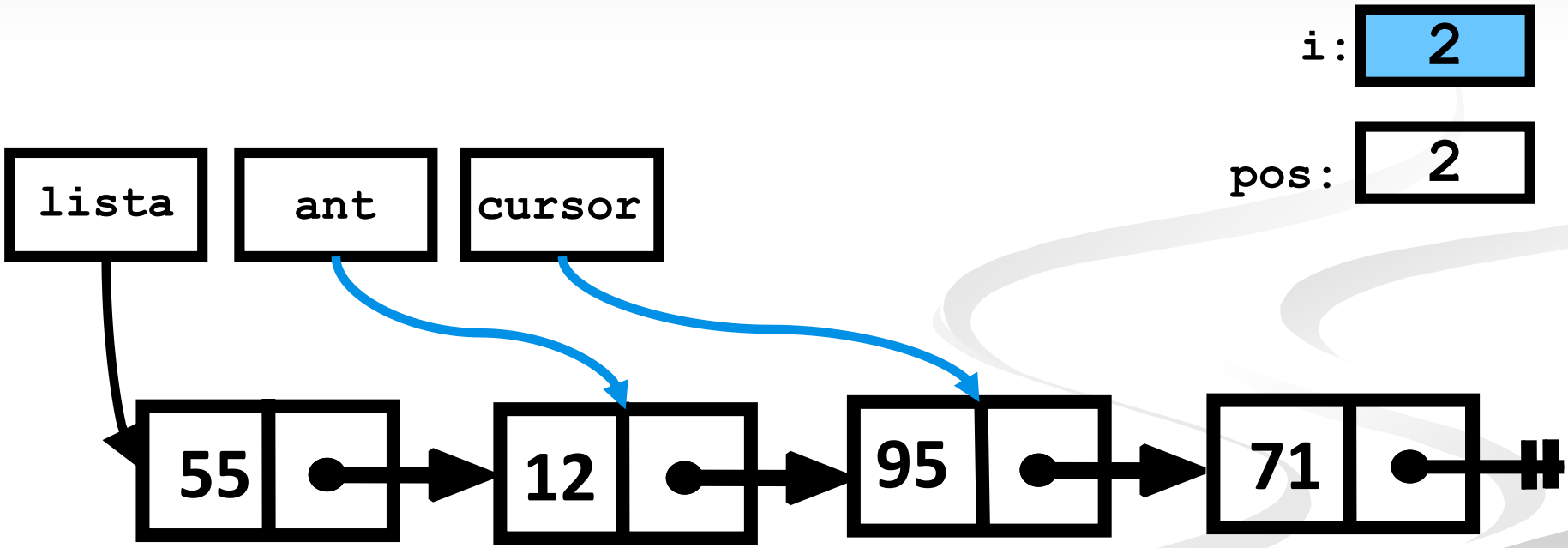
Listas enlazadas: Eliminación en cualquier posición

Es necesario tener dos cursores: uno que avance hasta la posición **`pos`** y otro que llegue hasta la posición **`pos-1`**



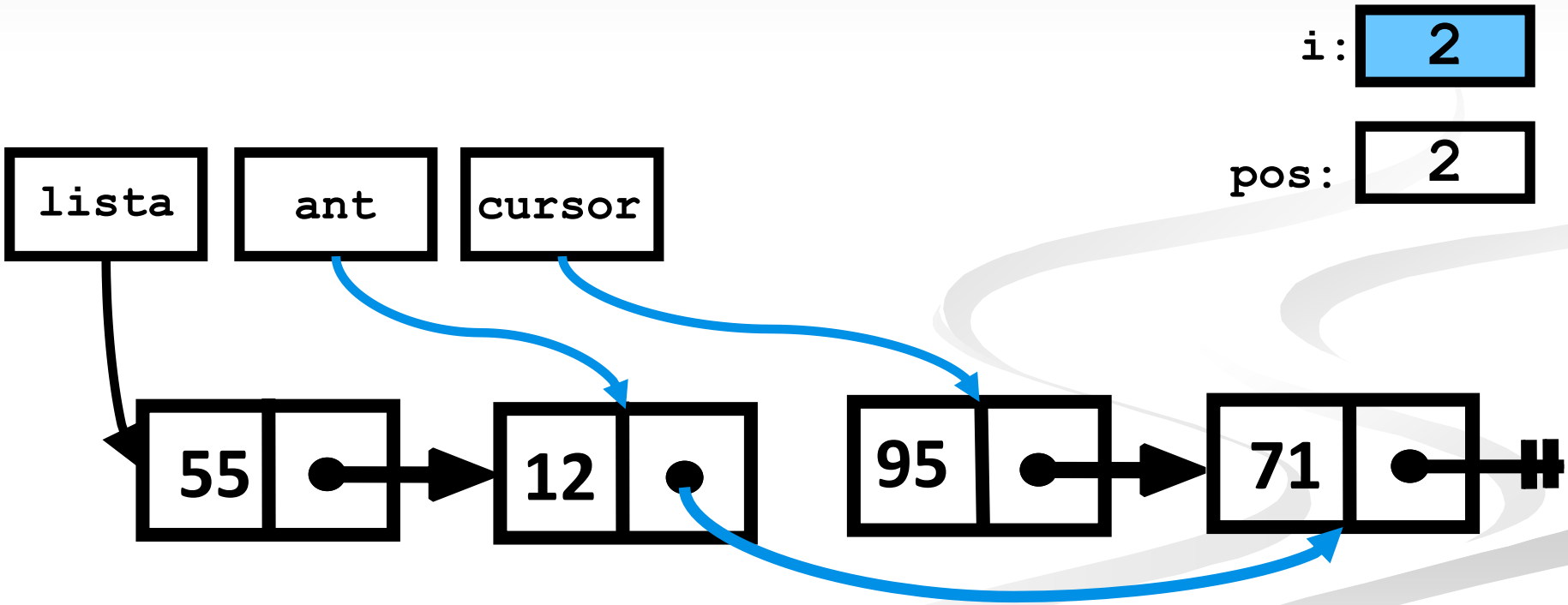
Listas enlazadas: Eliminación en cualquier posición

Es necesario tener dos cursores: uno que avance hasta la posición **`pos`** y otro que llegue hasta la posición **`pos-1`**



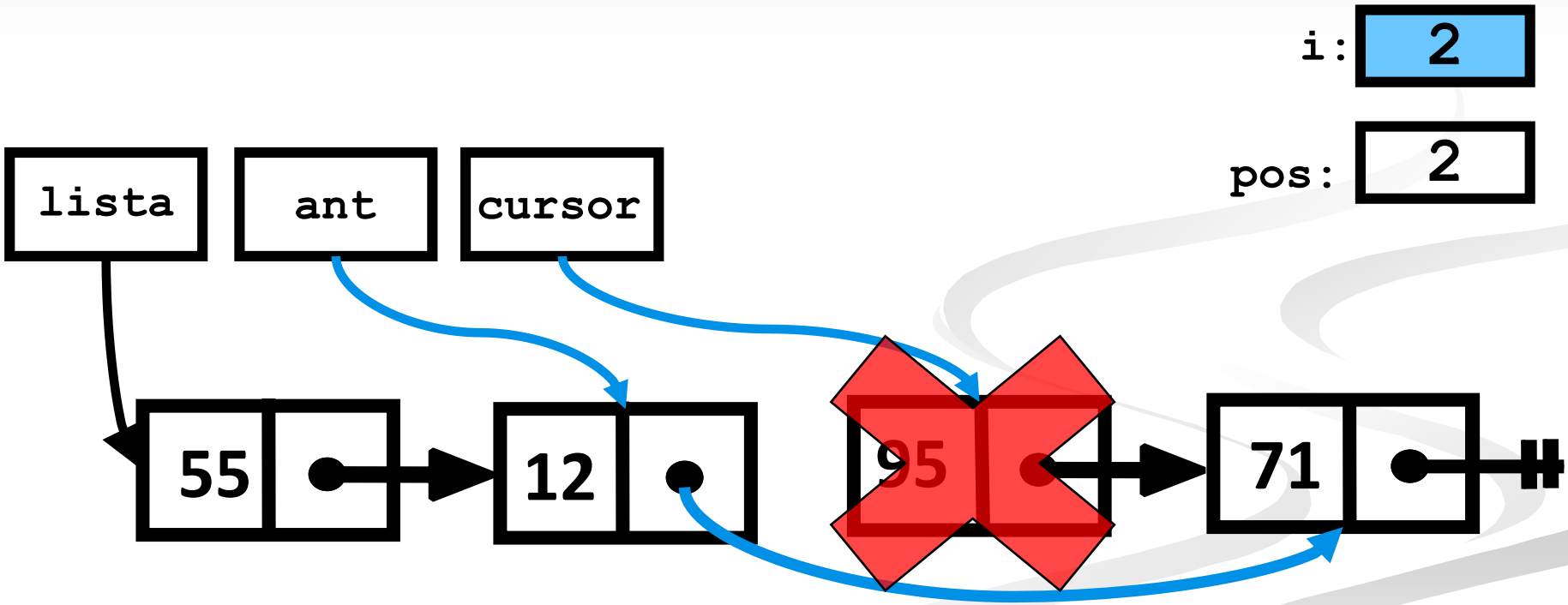
Listas enlazadas: Eliminación en cualquier posición

Es necesario tener dos cursores: uno que avance hasta la posición **`pos`** y otro que llegue hasta la posición **`pos-1`**



Listas enlazadas: Eliminación en cualquier posición

Es necesario tener dos cursores: uno que avance hasta la posición **`pos`** y otro que llegue hasta la posición **`pos-1`**

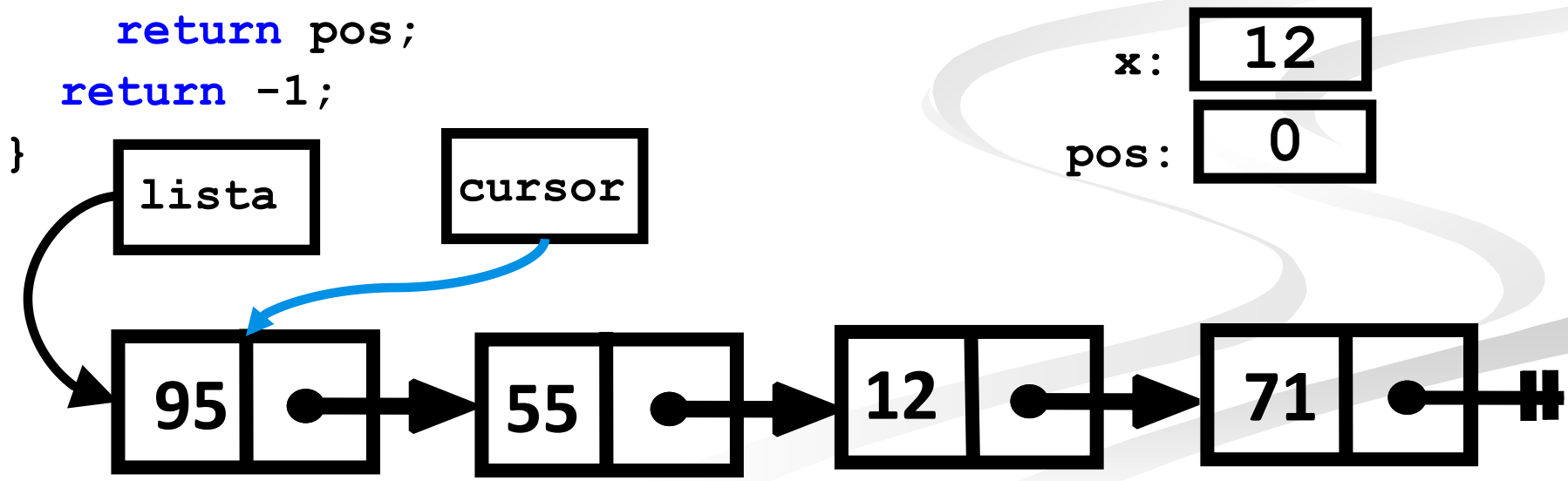


Listas enlazadas: Eliminación en cualquier posición

```
void eliminar(NodoLista*& lista, int pos){
  if (lista == NULL) return;
  NodoLista* cursor = lista;
  if (pos == 0){
    lista = lista->sig;
    delete cursor;
  }
  else if ((pos > 0)&&(pos < longitudLista(lista))) {
    NodoLista* anterior = lista;
    int i = 0;
    while ((cursor != NULL)&&(i < pos)){
      anterior = cursor;
      cursor = cursor->sig;
      i++;
    }
    anterior->sig = cursor->sig;
    delete cursor;
  }
}
```

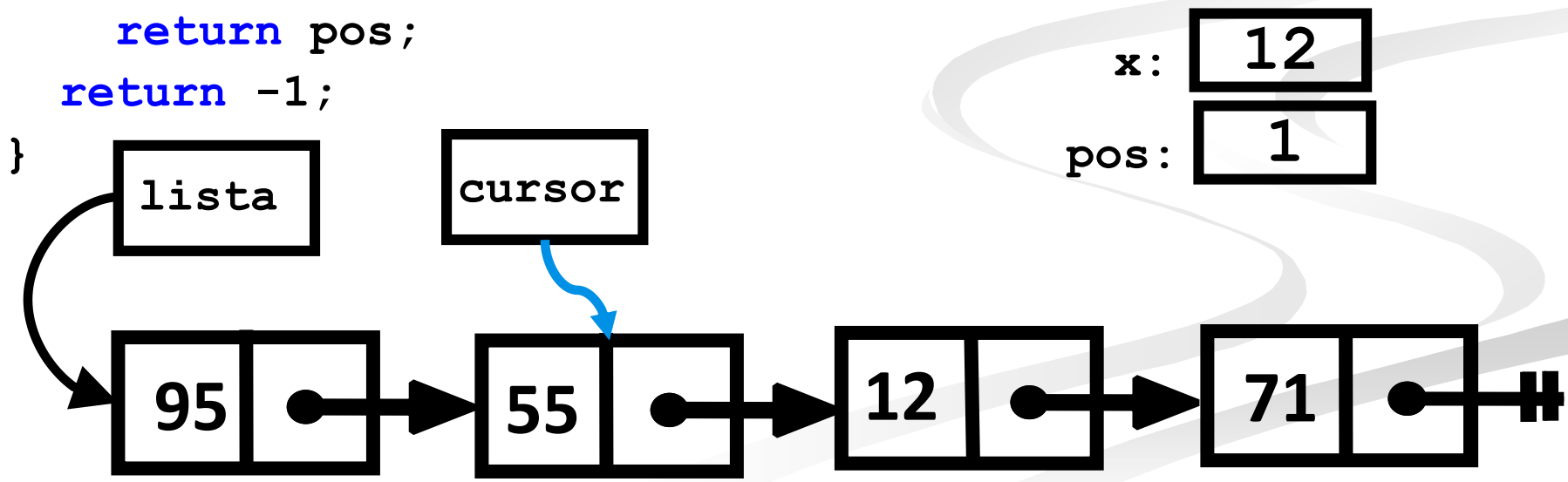
Listas enlazadas: Buscar un elemento

```
int buscar(NodoLista* lista, int x){
  int pos = 0;
  NodoLista* cursor = lista;
  while ((cursor != NULL)&&(cursor->dato != x)){
      cursor = cursor->sig;
      pos++;
  }
  if (cursor != NULL)
    return pos;
  return -1;
}
```



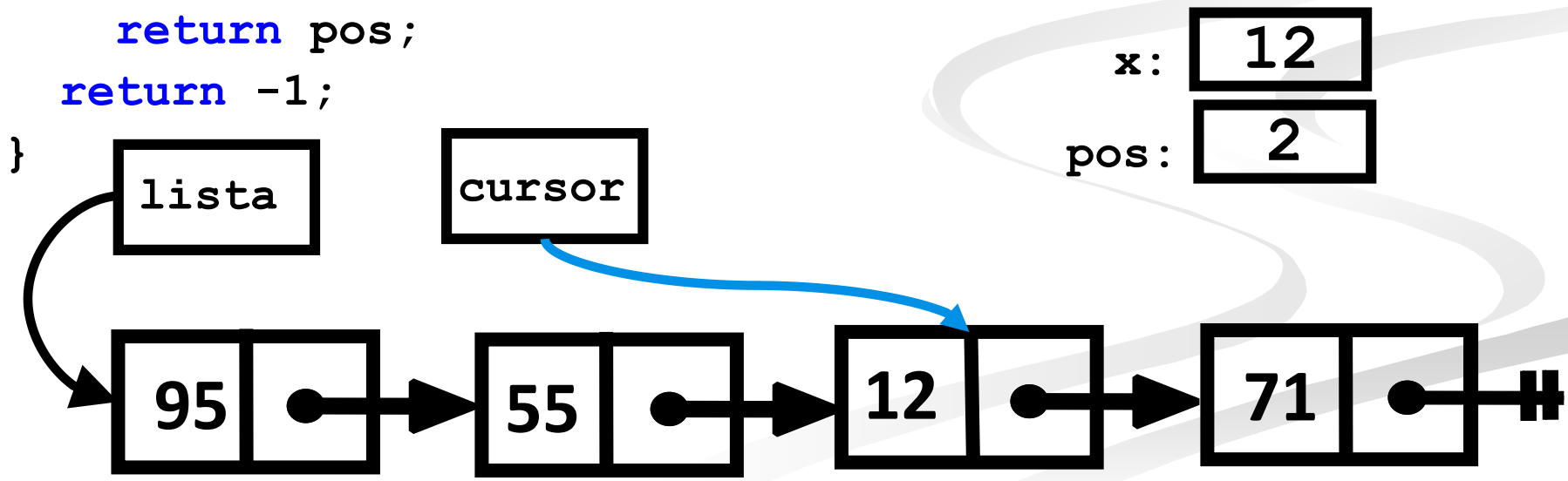
Listas enlazadas: Buscar un elemento

```
int buscar(NodoLista* lista, int x){
  int pos = 0;
  NodoLista* cursor = lista;
  while ((cursor != NULL)&&(cursor->dato != x)){
      cursor = cursor->sig;
      pos++;
  }
  if (cursor != NULL)
    return pos;
  return -1;
}
```



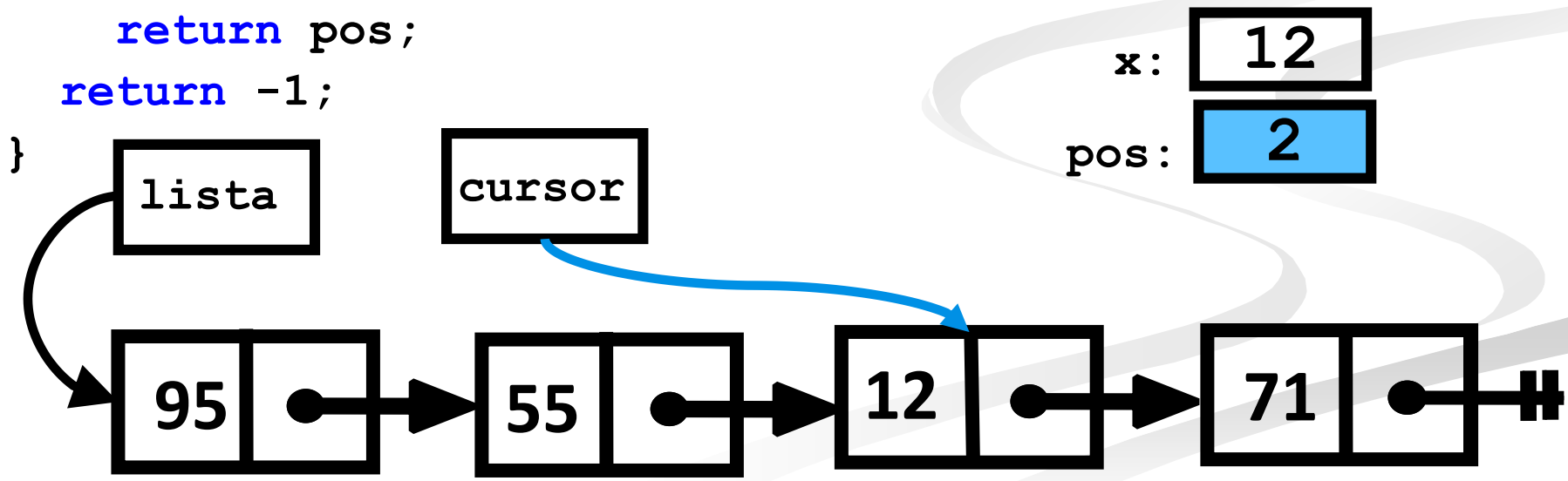
Listas enlazadas: Buscar un elemento

```
int buscar(NodoLista* lista, int x){
  int pos = 0;
  NodoLista* cursor = lista;
  while ((cursor != NULL)&&(cursor->dato != x)){
      cursor = cursor->sig;
      pos++;
  }
  if (cursor != NULL)
    return pos;
  return -1;
}
```



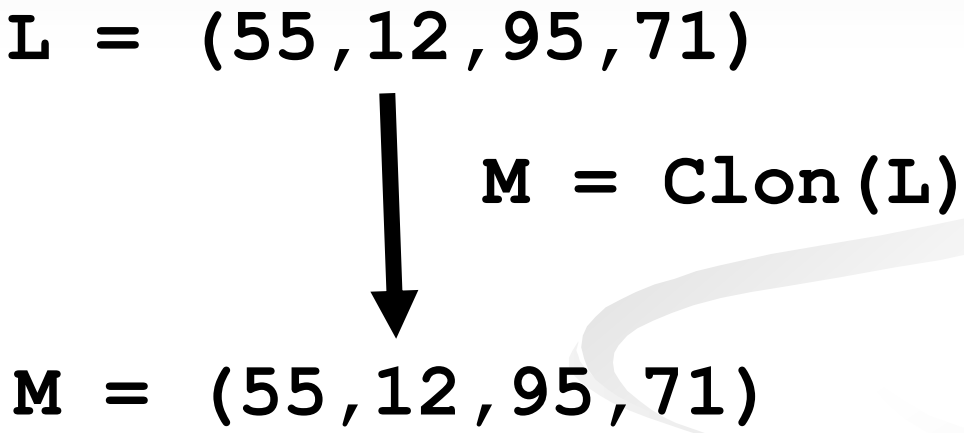
Listas enlazadas: Buscar un elemento

```
int buscar(NodoLista* lista, int x){
  int pos = 0;
  NodoLista* cursor = lista;
  while ((cursor != NULL)&&(cursor->dato != x)){
      cursor = cursor->sig;
      pos++;
  }
  if (cursor != NULL)
    return pos;
  return -1;
}
```



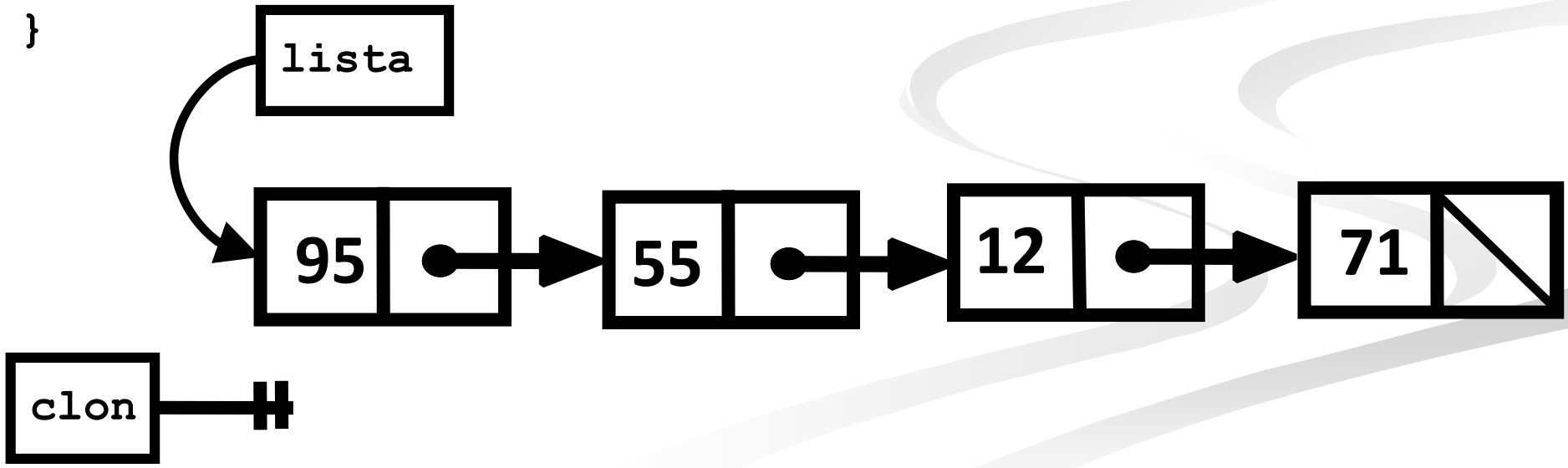
Listas enlazadas: Copia (Clon) de una lista

Dada una lista, se desea retornar una copia de ella que no comparta memoria



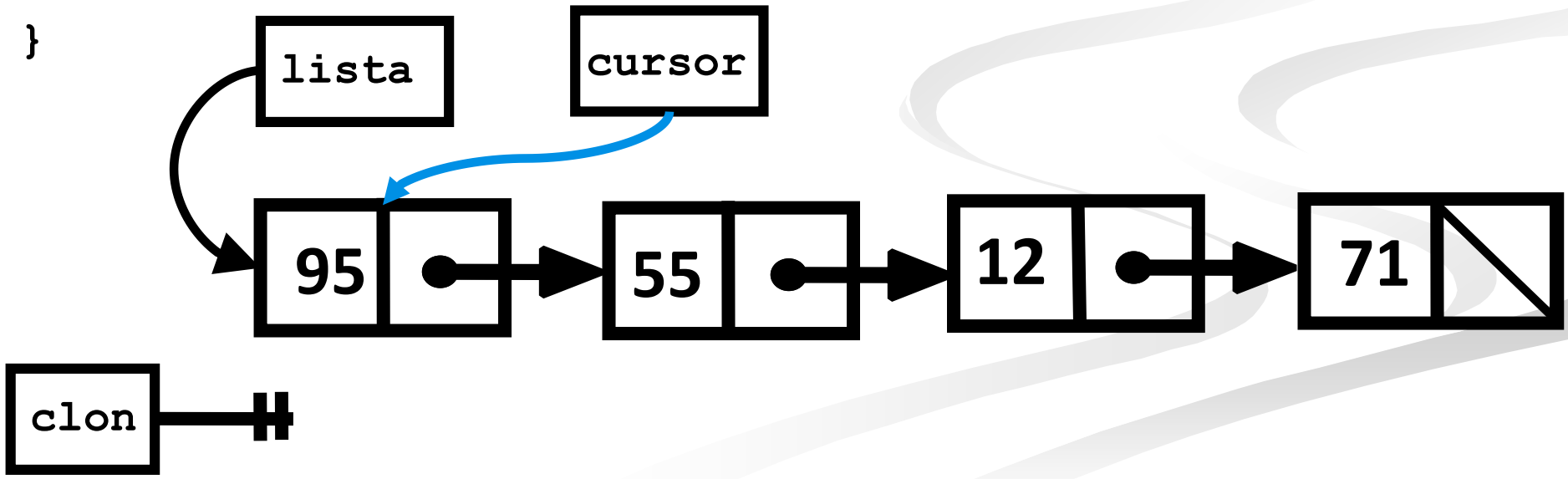
Listas enlazadas: Copia (Clon) de una lista

```
NodoLista* obtenerCopia(NodoLista* lista){
  NodoLista* clon = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      adicionar(clon, cursor->dato);
      cursor = cursor->sig;
  }
  return clon;
}
```



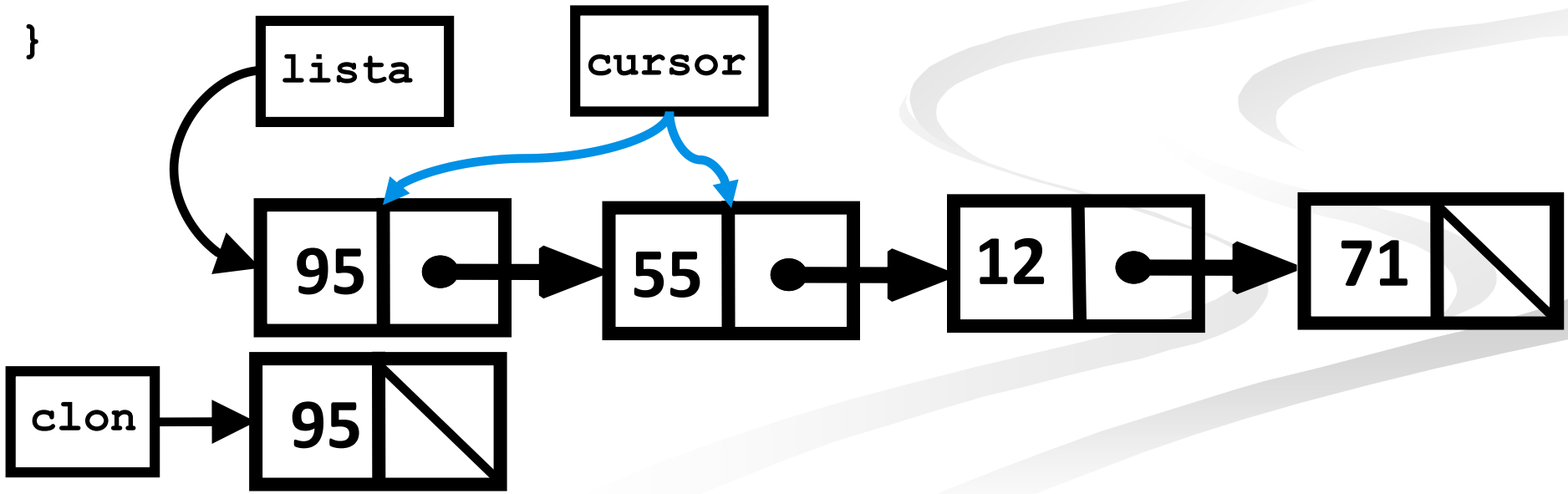
Listas enlazadas: Copia (Clon) de una lista

```
NodoLista* obtenerCopia(NodoLista* lista){
  NodoLista* clon = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      adicionar(clon, cursor->dato);
      cursor = cursor->sig;
  }
  return clon;
}
```



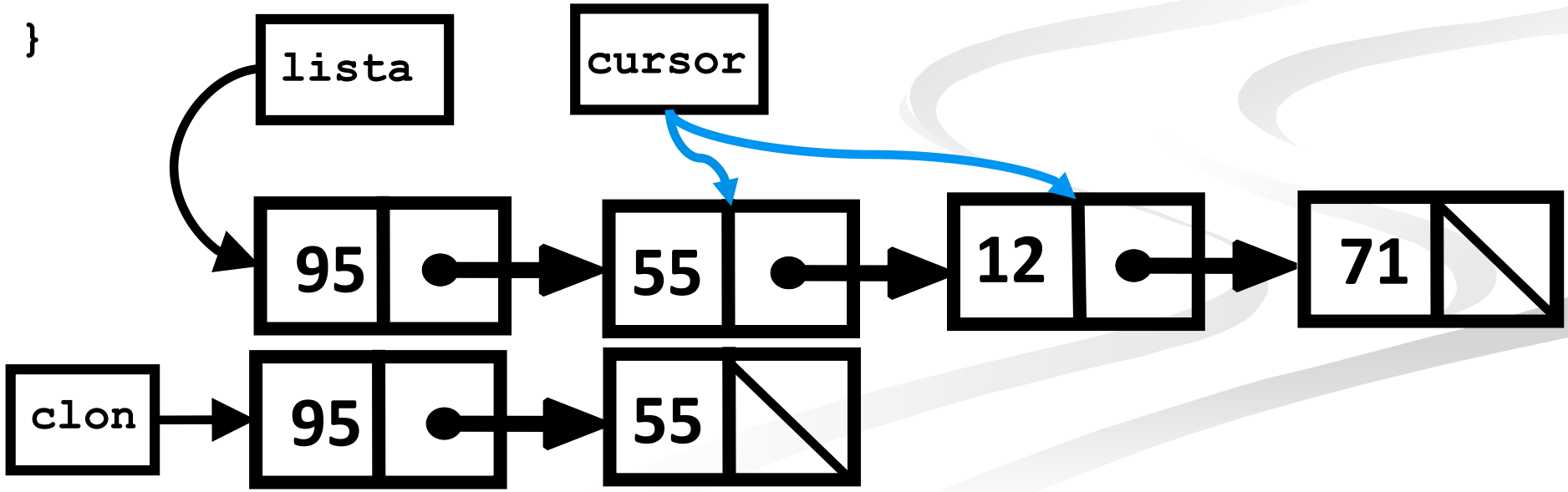
Listas enlazadas: Copia (Clon) de una lista

```
NodoLista* obtenerCopia(NodoLista* lista){
  NodoLista* clon = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      adicionar(clon, cursor->dato);
      cursor = cursor->sig;
  }
  return clon;
}
```



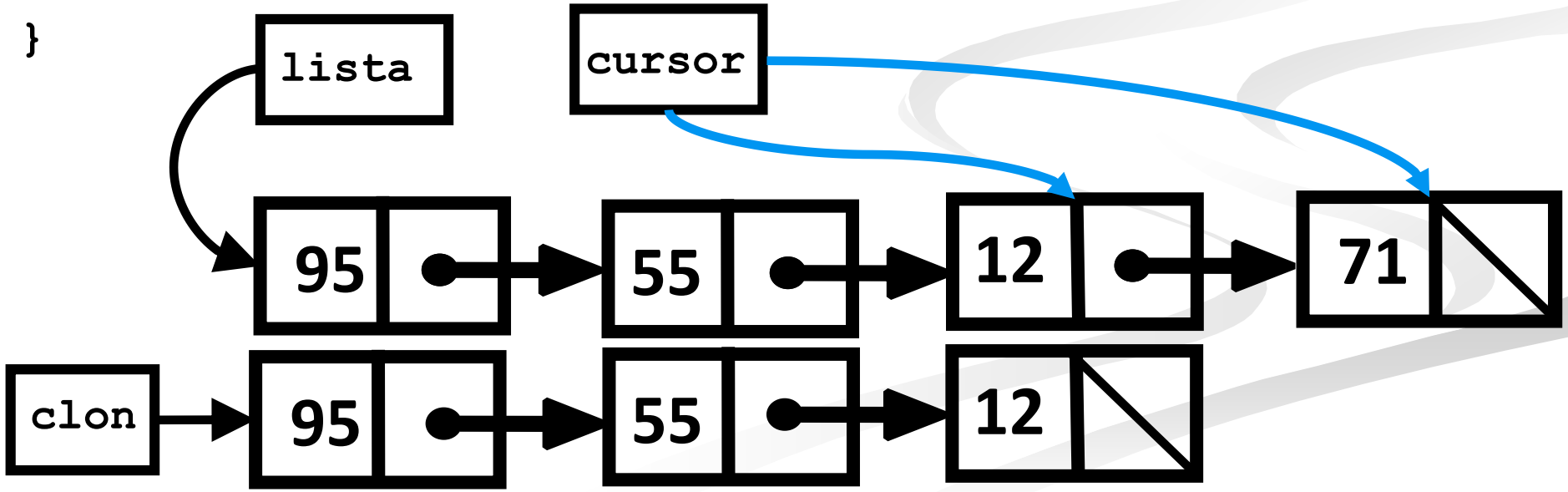
Listas enlazadas: Copia (Clon) de una lista

```
NodoLista* obtenerCopia(NodoLista* lista){
  NodoLista* clon = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      adicionar(clon, cursor->dato);
      cursor = cursor->sig;
  }
  return clon;
}
```



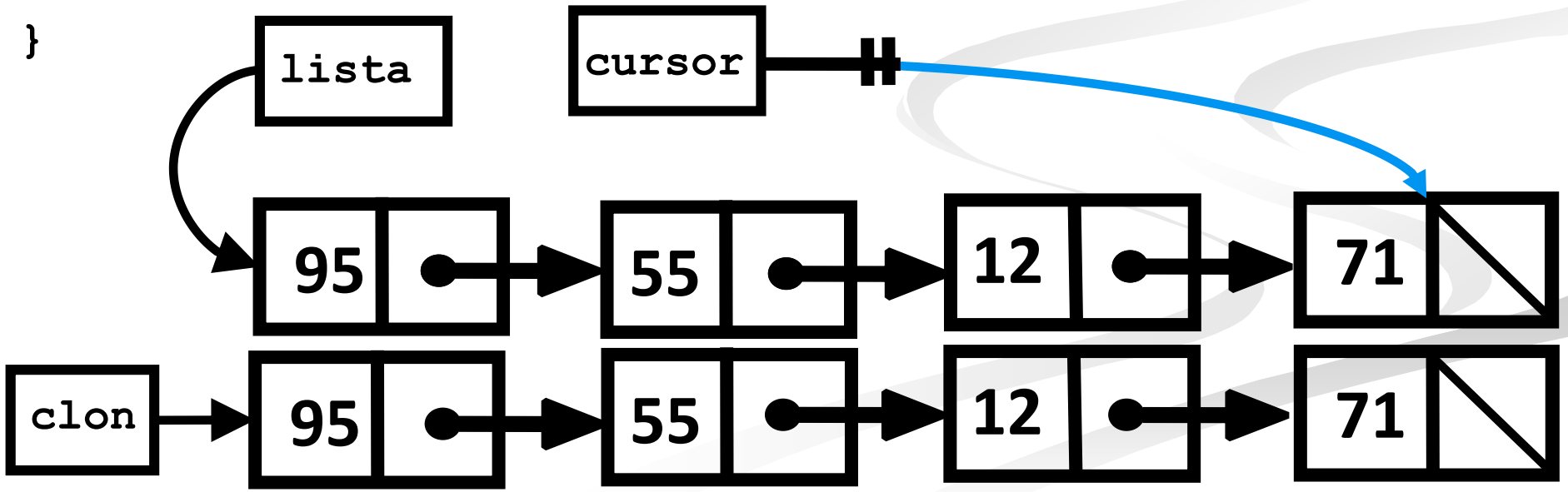
Listas enlazadas: Copia (Clon) de una lista

```
NodoLista* obtenerCopia(NodoLista* lista){
  NodoLista* clon = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      adicionar(clon, cursor->dato);
      cursor = cursor->sig;
  }
  return clon;
}
```



Listas enlazadas: Copia (Clon) de una lista

```
NodoLista* obtenerCopia(NodoLista* lista){
  NodoLista* clon = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      adicionar(clon, cursor->dato);
      cursor = cursor->sig;
  }
  return clon;
}
```



Listas enlazadas: Invertir una lista

Dada una lista, se desea retornar otra lista que tenga los mismos elementos pero en orden invertido al original.

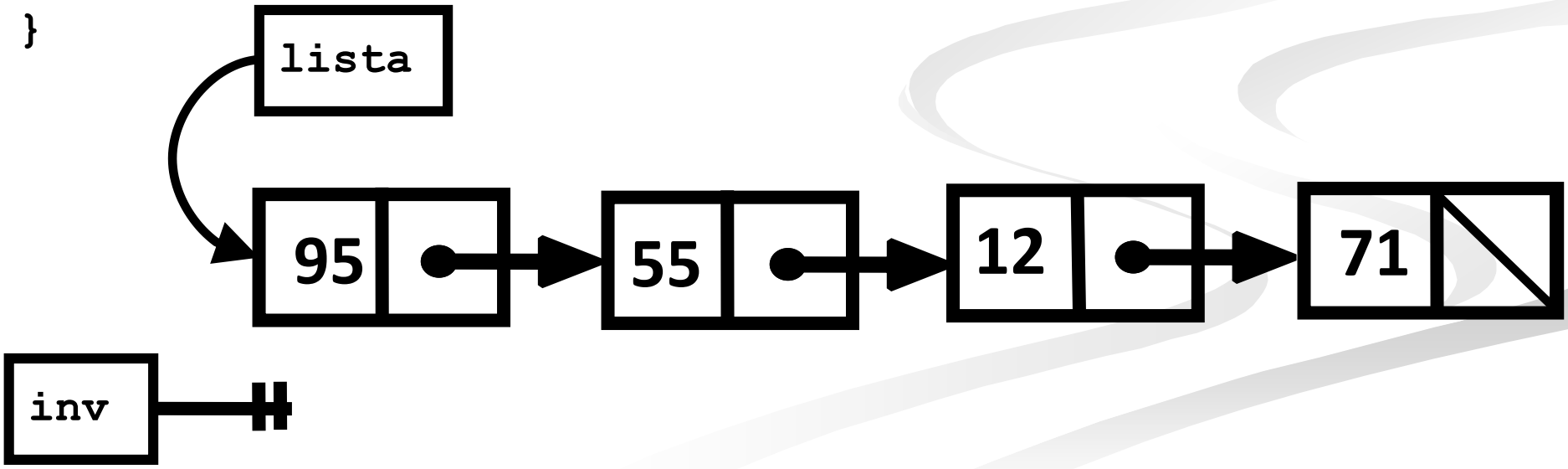
```
L = (55,12,95,71)
```

```
M = Invertir(L)
```

```
M = (71,95,12,55)
```

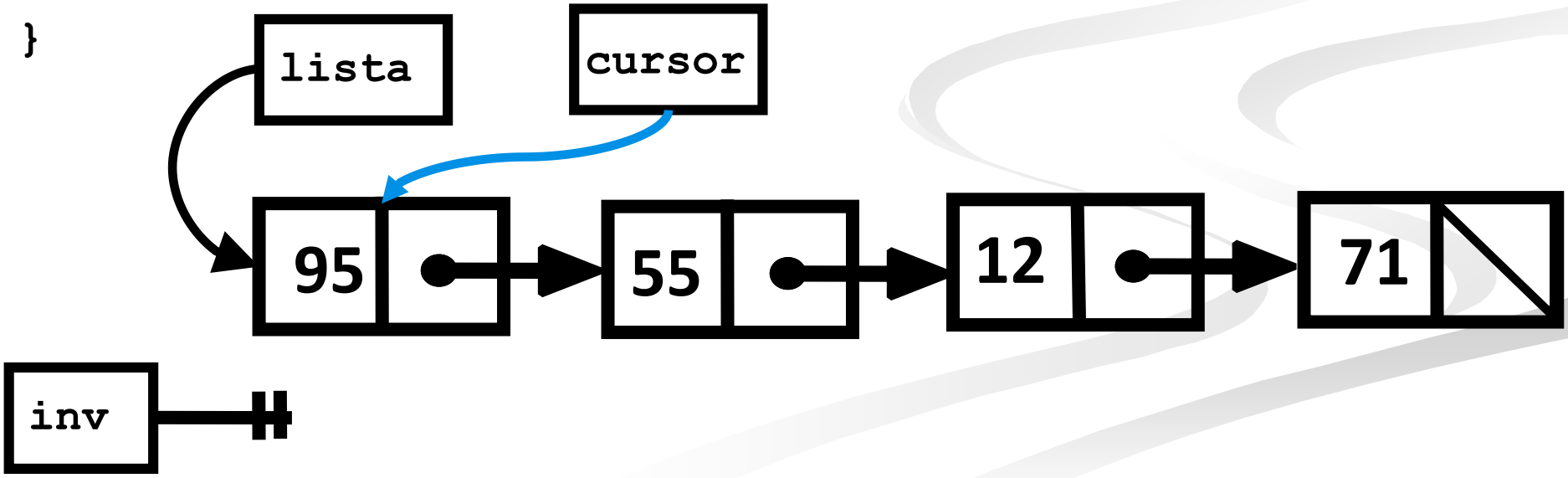
Listas enlazadas: Invertir una lista

```
NodoLista* invertir(NodoLista* lista){
  NodoLista* inv = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      insertarInicio(inv, cursor->dato);
      cursor = cursor->sig;
  }
  return inv;
}
```



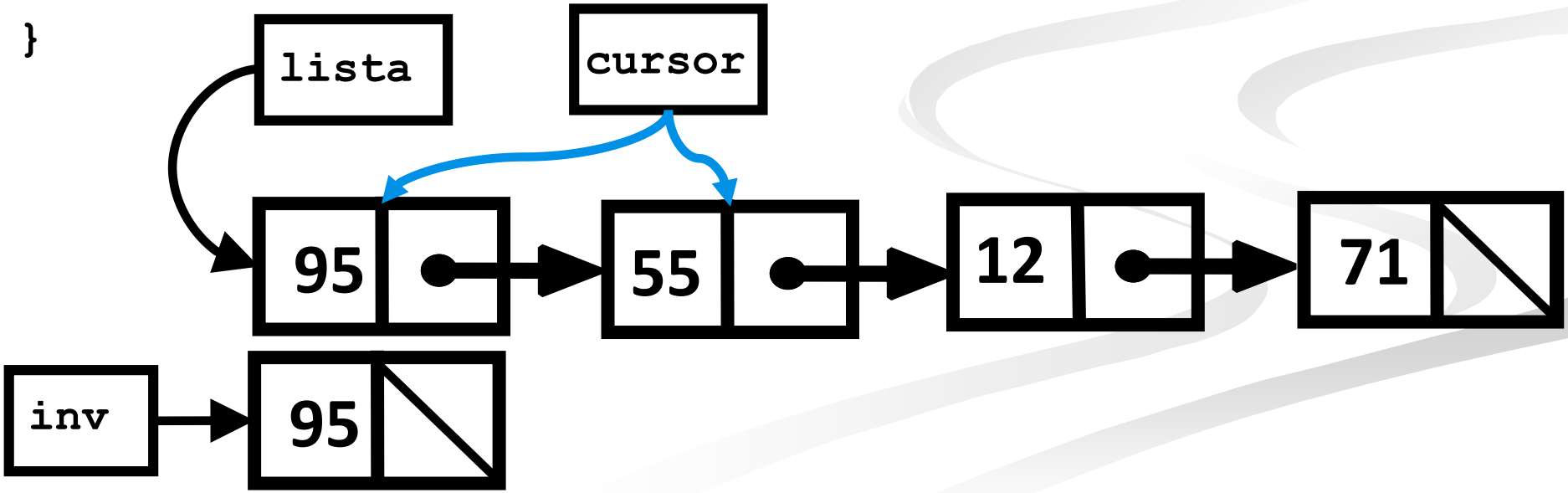
Listas enlazadas: Invertir una lista

```
NodoLista* invertir(NodoLista* lista){
  NodoLista* inv = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      insertarInicio(inv, cursor->dato);
      cursor = cursor->sig;
  }
  return inv;
}
```



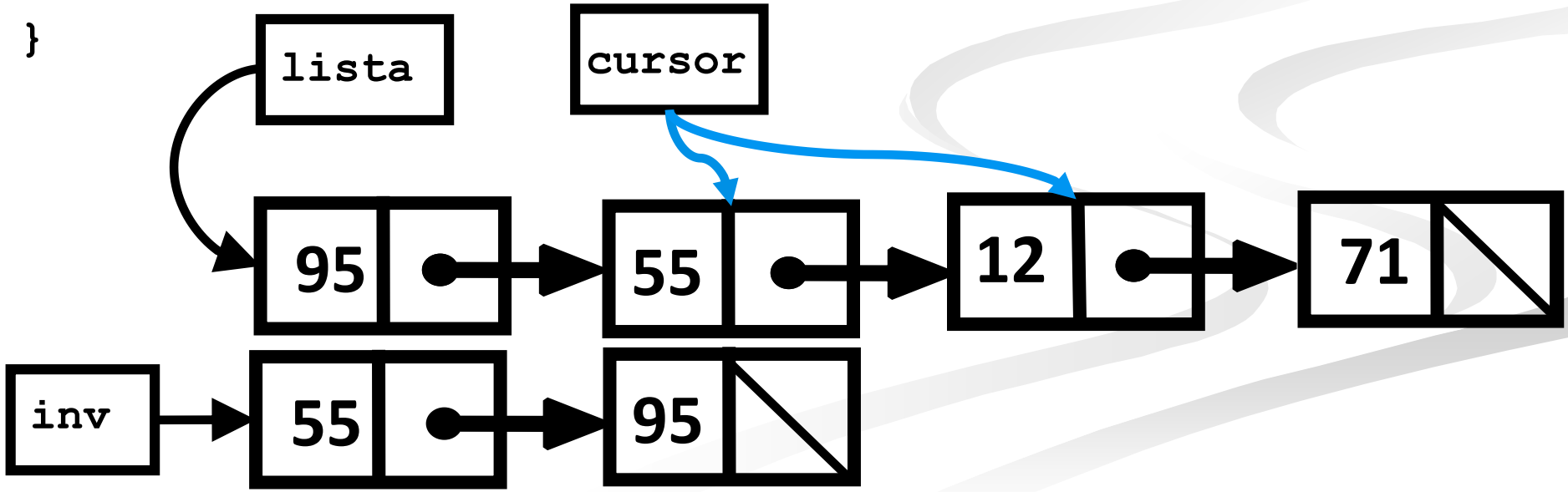
Listas enlazadas: Invertir una lista

```
NodoLista* invertir(NodoLista* lista){
  NodoLista* inv = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      insertarInicio(inv, cursor->dato);
      cursor = cursor->sig;
  }
  return inv;
}
```



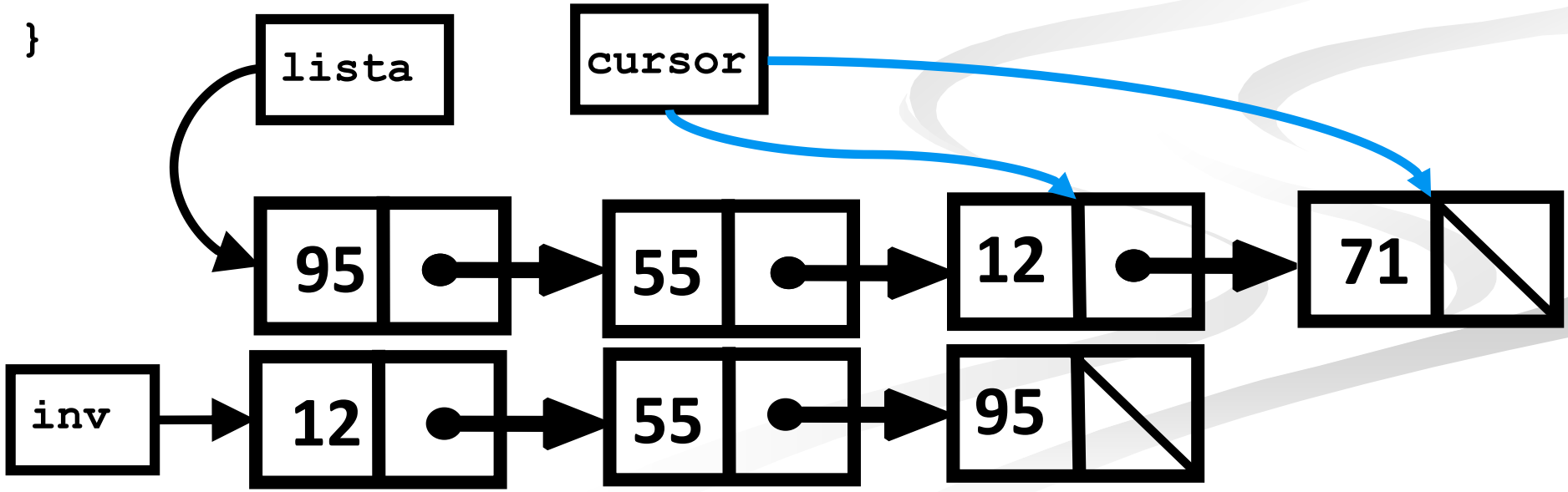
Listas enlazadas: Invertir una lista

```
NodoLista* invertir(NodoLista* lista){
  NodoLista* inv = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      insertarInicio(inv, cursor->dato);
      cursor = cursor->sig;
  }
  return inv;
}
```



Listas enlazadas: Invertir una lista

```
NodoLista* invertir(NodoLista* lista){
  NodoLista* inv = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      insertarInicio(inv, cursor->dato);
      cursor = cursor->sig;
  }
  return inv;
}
```



Listas enlazadas: Invertir una lista

```
NodoLista* invertir(NodoLista* lista){
  NodoLista* inv = NULL;
  NodoLista* cursor = lista;
  while (cursor != NULL){
      insertarInicio(inv, cursor->dato);
      cursor = cursor->sig;
  }
  return inv;
}
```