Estructura de Datos y Algoritmos 1

Teórico #8 :

Listas enlazadas (Ejercicios)

Recursión con Listas enlazadas

Ejercicio 1: Listas enlazadas ordenada: Insertar

Dada una lista L ordenada de menor a mayor, y un valor **X**, se desea insertar a **X** en L de manera que la lista resultante quede ordenada.

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

```
Adicionar(L,65)
```

```
L = (12,55,65,71)
```

Ejercicio 2: Eliminar N primeros elementos

Dada una lista **L** y un número natural **N**, se desea eliminar a lo sumo los **N** primeros elementos. Si **N** es mayor que la longitud de la lista, esta debe quedar vacía.

```
L = (12,55,65,71)
```

```
eliminarNPrimeros(L,2)
```

```
L = (65,71)
```

Ejercicio 3: Invertir eliminando repetidos

Dada una lista **L** ordenada de menor a mayor, se desea obtener una nueva lista con los mismos elementos de **L**, pero ordenada de mayor a menor y sin elementos repetidos.

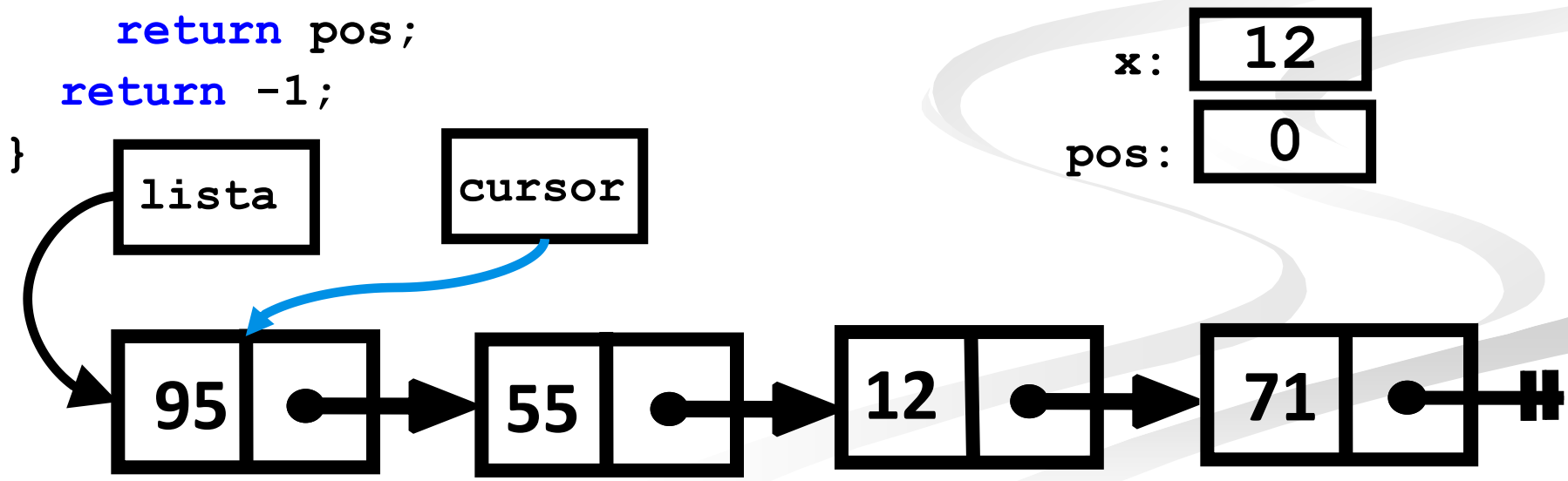
```
L = (1,3,3,4,5,5,5,9)
```

```
L2=InvSinRepetidos(L)
```

```
L2 = (9,5,4,3,1)
```

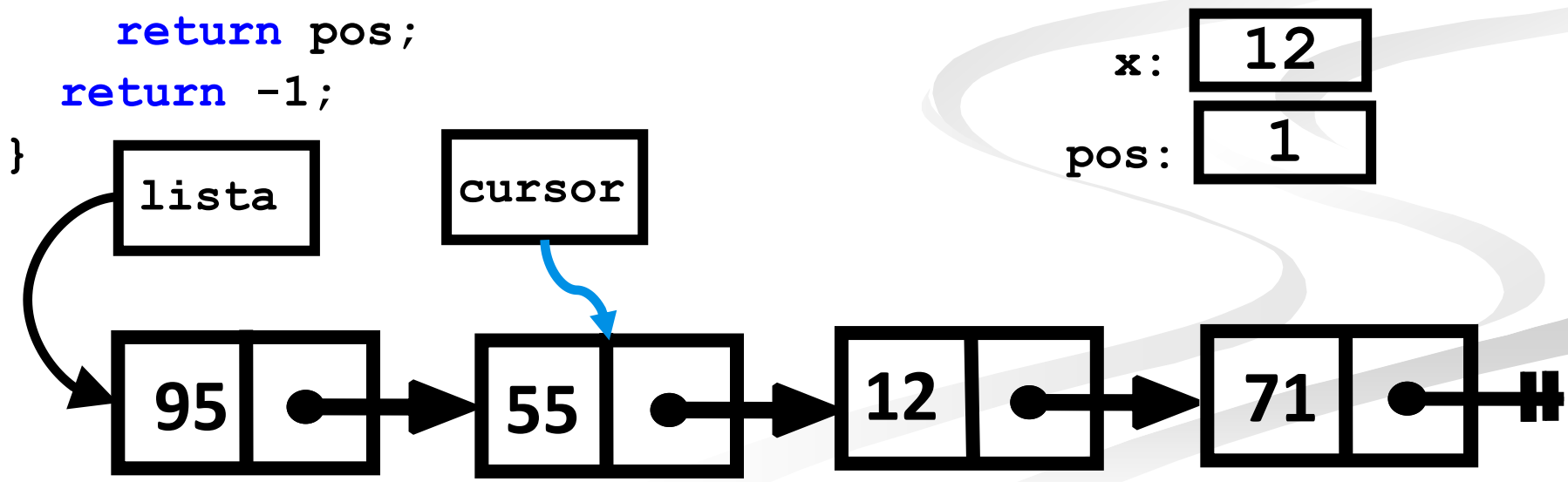
Listas enlazadas: Buscar un elemento

```
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;
}
```



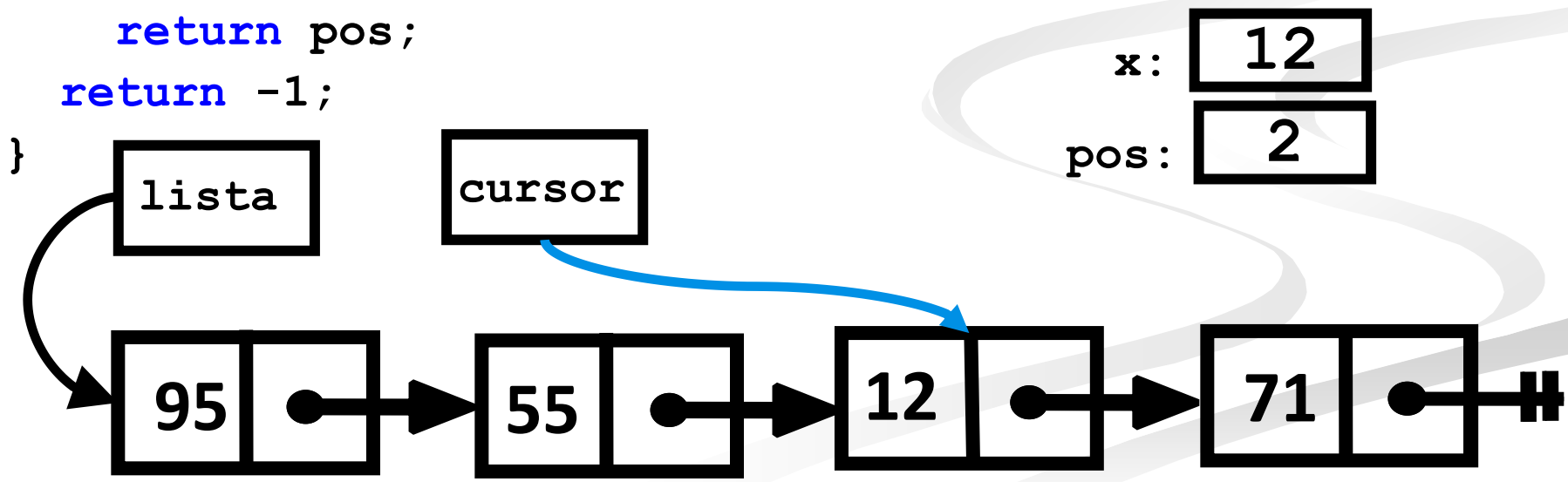
Listas enlazadas: Buscar un elemento

```
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;
}
```



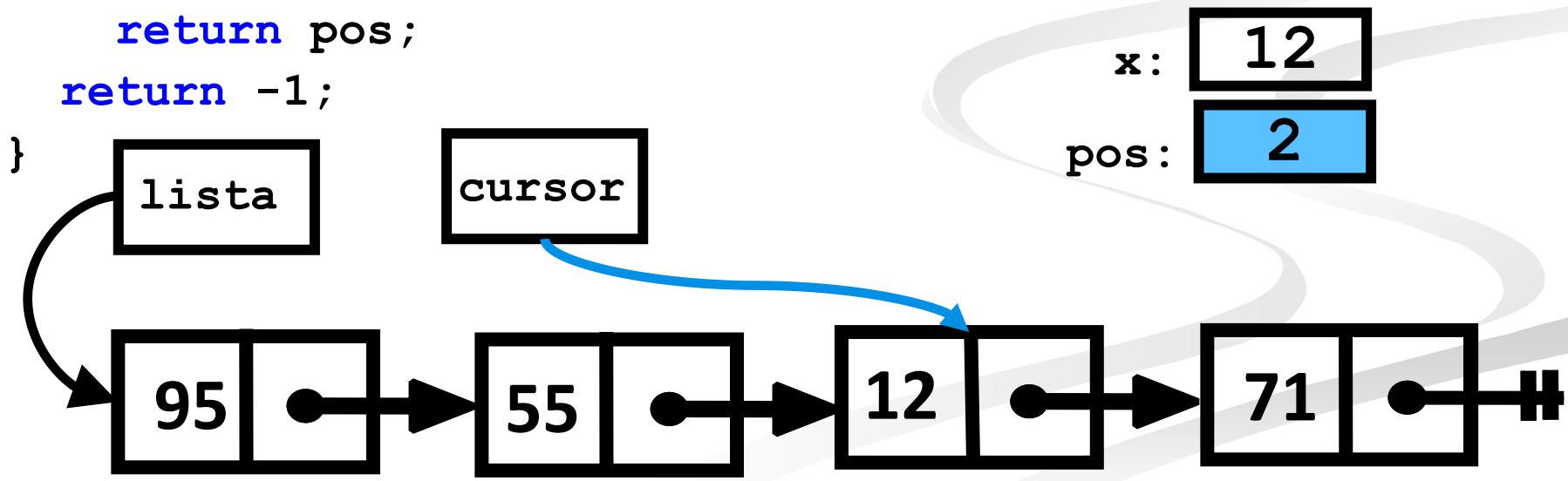
Listas enlazadas: Buscar un elemento

```
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: 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;
}
```



¿Cómo hacerlo recursivamente?

Caso base 1: La lista está vacía (`L = []`)

→ El elemento **x** no está en la lista (`posResult = -1`)

Caso base 2: El 1er elemento de la lista es igual a **x** (`L[0] == x`)

→ `posResult = 0`

Caso inductivo:

- Obtener la posición que ocupa **x** en `L[1:n-1]` (`posTemp`)
- Si `posTemp != -1`

  → `posResult = postTemp + 1`
- Sino `posResult = -1`

Listas enlazadas: Buscar un elemento (recursivo)

```
int buscar(NodoLista* lista, int x){
  if (lista == NULL) //Caso base 1
      return -1;
  if (lista->dato == x) //Caso base 2
      return 0;

  /* Caso inductivo */
  int posTemp = buscar(lista->sig, x);
  if (posTemp != -1)
      return posTemp + 1;
  return -1;
}
```

NOTA: Con **`lista->sig`** obtenemos la dirección donde comienza el resto de la lista enlazada, lo cual es a su vez otra lista.

Listas enlazadas: Longitud (Recursivo)

$$largo: \text{Lista} \rightarrow \mathbb{N}$$

$$largo([\,]) = 0$$

$$largo(x.L) = 1 + largo(L)$$

Caso base 1: La lista está vacía (`L = []`)

→ (`longitud = 0`)

Caso inductivo:

- Obtener la longitud de la lista `L[1:n-1]` (`longitudTemp`)
- `longitud = longitudTemp + 1`

Listas enlazadas: Longitud (Recursivo)

```
int longitudLista(NodoLista* lista)
{
  if (lista == NULL)
    return 0;
  return 1 + longitudLista(lista->sig)
}
```

Listas enlazadas: Adicionar al final (Recursivo)

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)
```

$$adicionar: \mathbb{N} \times \text{Lista} \rightarrow \text{Lista}$$

$$adicionar(e, [\,]) = e.[\,]$$

$$adicionar(e, x.L) = x.adicionar(e, L)$$

Listas enlazadas: Adicionar al final (Recursivo)

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

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.

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

```
Insertar(L,95,2)
```

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

$$\boldsymbol{insertar}: \mathbb{N} \times \mathbb{N} \times \text{Lista} \rightarrow \text{Lista}$$

$$\boldsymbol{insertar}(e, \boldsymbol{pos}, x.L) =$$

$$\begin{cases} e.x.L & \text{si } \boldsymbol{pos} = 0 \\ x.\boldsymbol{insertar}(e, \boldsymbol{pos} - 1, L) & \text{si } \boldsymbol{pos} < \boldsymbol{longitud}(L) \end{cases}$$

Listas enlazadas: Inserción en cualquier posición

```
void insertar(NodoLista*& lista, int e, int pos){
  if (pos == 0){
    NodoLista* nuevo = new NodoLista(e);
    nuevo->sig = lista;
    lista = nuevo;
  }
  else if (((pos > 0)&& (pos < longitudLista(lista)))
    insertar(lista->sig, e, pos-1);
}
```

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.

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

Eliminar(L,2)

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

$$eliminar: \mathbb{N} \times \mathbb{N} \times \text{Lista} \rightarrow \text{Lista}$$

$$eliminar(pos, x.L) = \begin{cases} L & \text{si } pos = 0 \\ x.eliminar(pos - 1, L) & \text{si } pos < longitud(L) \end{cases}$$

Listas enlazadas: Eliminación en cualquier posición

```
void eliminar(NodoLista*& lista, int pos){
  if (pos == 0){
    NodoLista* cursor = lista;
    lista = lista->sig;
    delete cursor;
  }
  else if ((pos > 0)&&(pos < longitudLista(lista)))
    eliminar(lista->sig, pos - 1);
}
```

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)
```

$$invertir: \text{Lista} \rightarrow \text{Lista}$$

$$invertir([\,]) = [\,]$$

$$invertir(x.L) = invertir(L).x$$

Listas enlazadas: Invertir una lista

```
NodoLista* invertir(NodoLista* lista){
  if (lista == NULL)
    return NULL;
  NodoLista* inv = invertir(lista->sig);
  adicionar(inv, lista->dato);
  return inv;
}
```

Listas enlazadas: Eliminar todas las ocurrencias

3. Eliminar todas las ocurrencias de un elemento de una lista.

$$\boldsymbol{elimT}(4, 3.4.4.6.4.[]) = 3.6.[]$$

$$\boldsymbol{elimT}: \mathbb{N} \times \text{Lista} \rightarrow \text{Lista}$$

$$\boldsymbol{elimT}(e, [\,]) = [\,]$$

$$\boldsymbol{elimT}(e, x.L) = \begin{cases} \boldsymbol{elimT}(e, L) & \text{si } e == x \\ x.\boldsymbol{elimT}(e, L) & \text{en otro caso} \end{cases}$$

$$\begin{aligned} \boldsymbol{elimT}(4, 3.4.4.6.4.[]) &= 3.\boldsymbol{elimT}(4, 4.4.6.4.[]) \\ &= 3.\boldsymbol{elimT}(4, 4.6.4.[]) \\ &= 3.\boldsymbol{elimT}(4, 6.4.[]) \end{aligned}$$

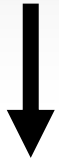
Listas enlazadas: Eliminar todas las ocurrencias

```
void eliminarOcurrencias(NodoLista* &lista, int e){
  if (lista == NULL) return;
  if (e == lista->dato){
    NodoLista* cursor = lista;
    lista = lista->sig;
    delete cursor;
    eliminarOcurrencias(lista, e);
  }
  else
    eliminarOcurrencias(lista->sig, e);
}
```

Listas enlazadas: Eliminar duplicados

3. Eliminar duplicados de una lista, dejando solo una ocurrencia (la primera) de cada elemento diferente.

```
L = (3,1,3,9,5,4,5,4)
```



```
eliminarDuplicados(L)
```

```
L = (3,1,9,5,4)
```

$$elimDup: \text{Lista} \rightarrow \text{Lista}$$

$$elimDup([\,]) = [\,]$$

$$elimDup(x.L) = x.elimDup(elimT(x, L))$$

Listas enlazadas: Eliminar duplicados

```
void eliminarDuplicados(NodoLista* &lista){
  if (lista == NULL) return;
  eliminarOcurrencias(lista->sig, lista->dato);
  eliminarDuplicados(lista->sig);
}
```