

Estructuras de Datos y Algoritmos 1

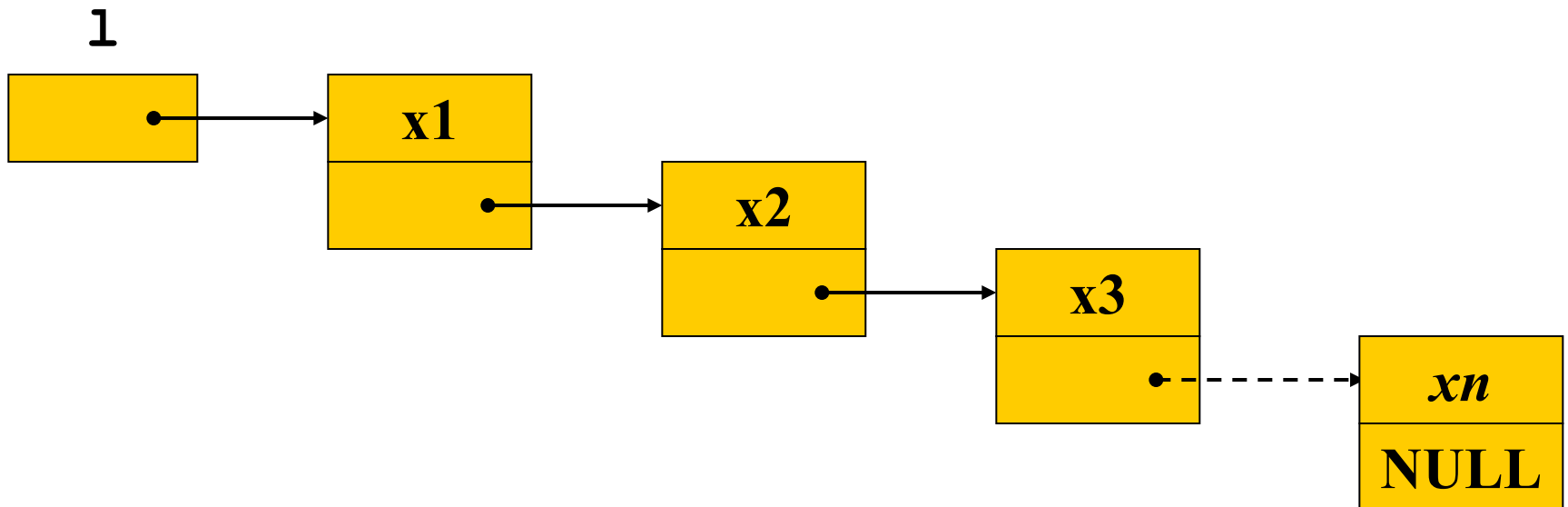
Ejercicios: Listas

Matías Crizul

Insertar al principio en una lista

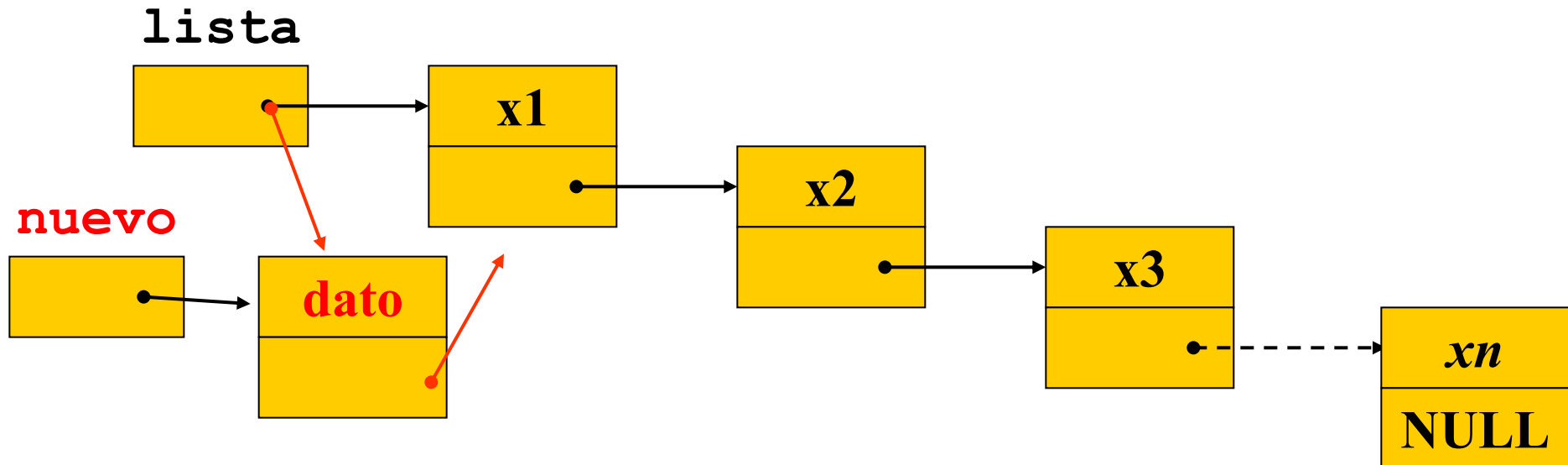
- Dada una lista, deseamos agregar al comienzo de la misma un nuevo elemento.

void agregarPrincipio(NodoLista* &lista, int dato);



Insertar al principio en una lista

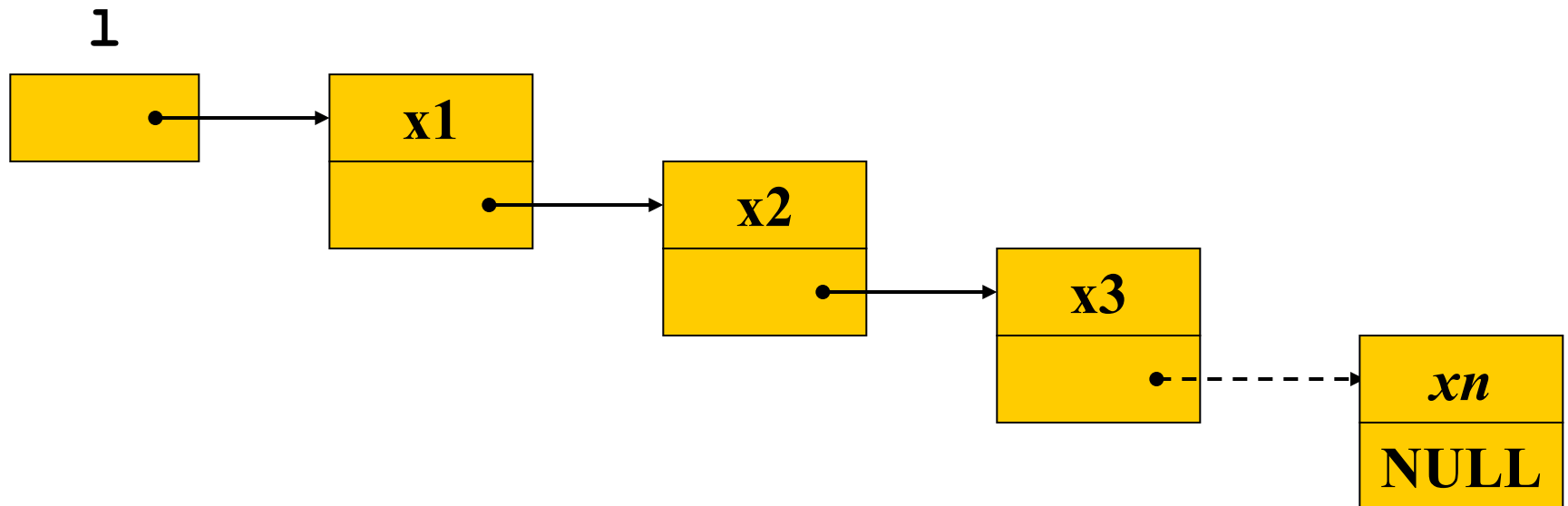
```
void agregarPrincipio(NodoLista* &lista, int dato) {  
    NodoLista* nuevo = new NodoLista;  
    nuevo->dato = dato;  
    nuevo->sig = lista;  
    lista = nuevo;  
}
```



Insertar al final en una lista

- Dada una lista, deseamos agregar al final de la misma un nuevo elemento.

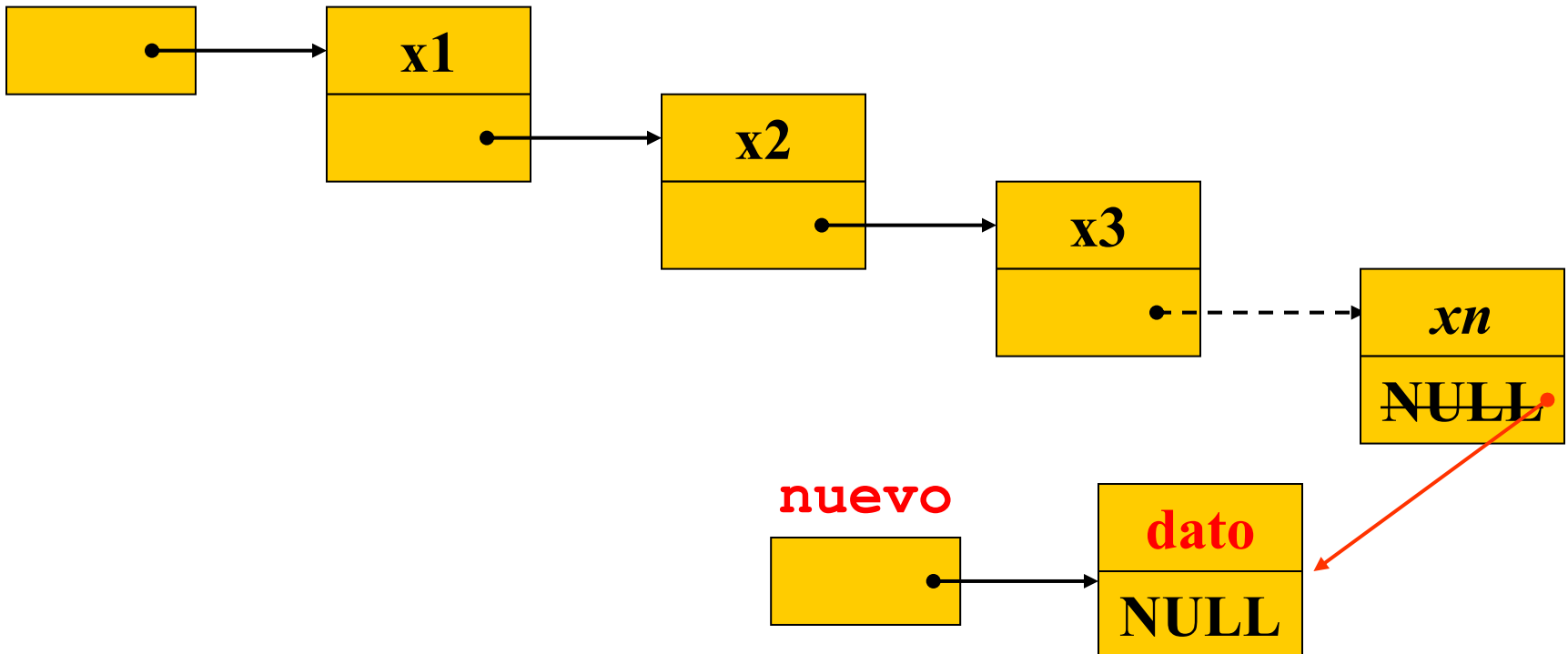
void agregarFin(NodoLista* &lista, int dato);



Insertar al final en una lista (Iterativo)

- Lista == NULL (lista vacia): No debo iterar.
- Lista != NULL (tiene al menos un elemento): Debo iterar hasta el ultimo elemento.

lista



Insertar al final en una lista (Iterativo)

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

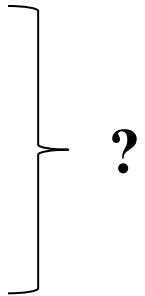
Insertar al final en una lista (Recursivo)

agregarFin: **NLista** x **N** \rightarrow **NLista**

agregarFin (**[]**, n) = n.**[]**

agregarFin (**x.S**, n) = x.agregarFin(**S**, n)

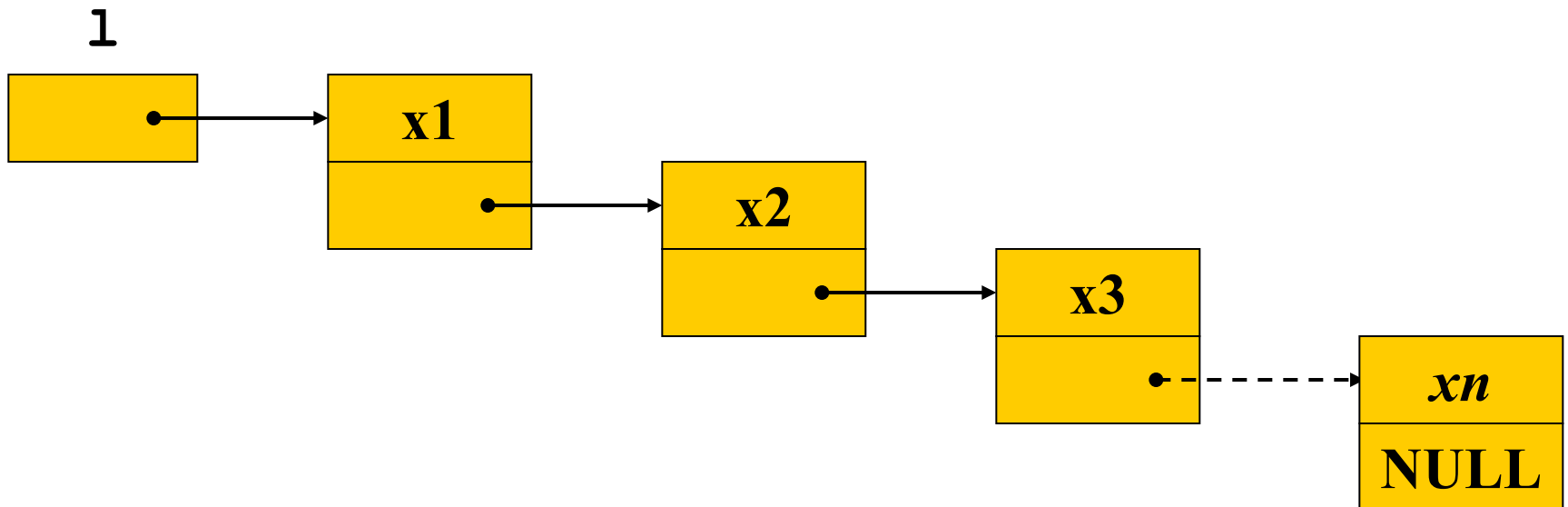
```
void agregarFin(NodoLista* &lista, int dato) {  
    if(lista == NULL) {  
        NodoLista* nuevo = new NodoLista;  
        nuevo->dato = dato;  
        nuevo->sig = NULL;  
        lista = nuevo;  
    }else {  
        agregarFin(lista->sig, dato);  
    }  
}
```



Insertar en posicion en una lista

- Dada una lista, deseamos agregar en cierta posicion de la misma un nuevo elemento.

void agregarPos(NodoLista* &lista, int dato, int pos);



Insertar en posicion en una lista

agregarPos: **NLista** x N x N \rightarrow NLista

agregarPos ([], n, pos) = si pos==0: n.[]

agregarPos (x.**S**, n, pos) = si pos==0: n.x.S

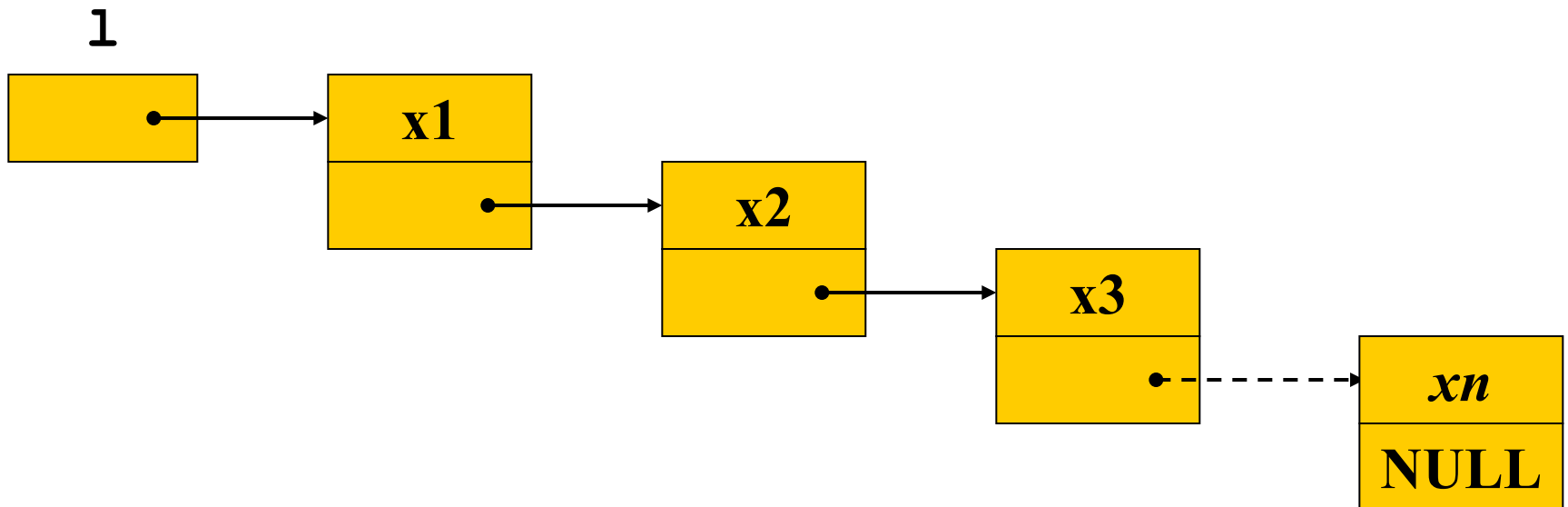
agregarPos (x.**S**, n, pos) = si pos!=0: x.agregarPos (**S**, n, pos-1)

```
void agregarPos(NodoLista* &lista, int dato, int pos) {  
    if((lista == NULL && pos == 0) ||  
        (lista != NULL && pos == 0)){  
        NodoLista* nuevo = new NodoLista;  
        nuevo->dato = dato;  
        nuevo->sig = lista;  
        lista = nuevo;  
    }else {  
        agregarPos(lista->sig, dato, pos-1);  
    }  
}
```

Borrar principio en una lista

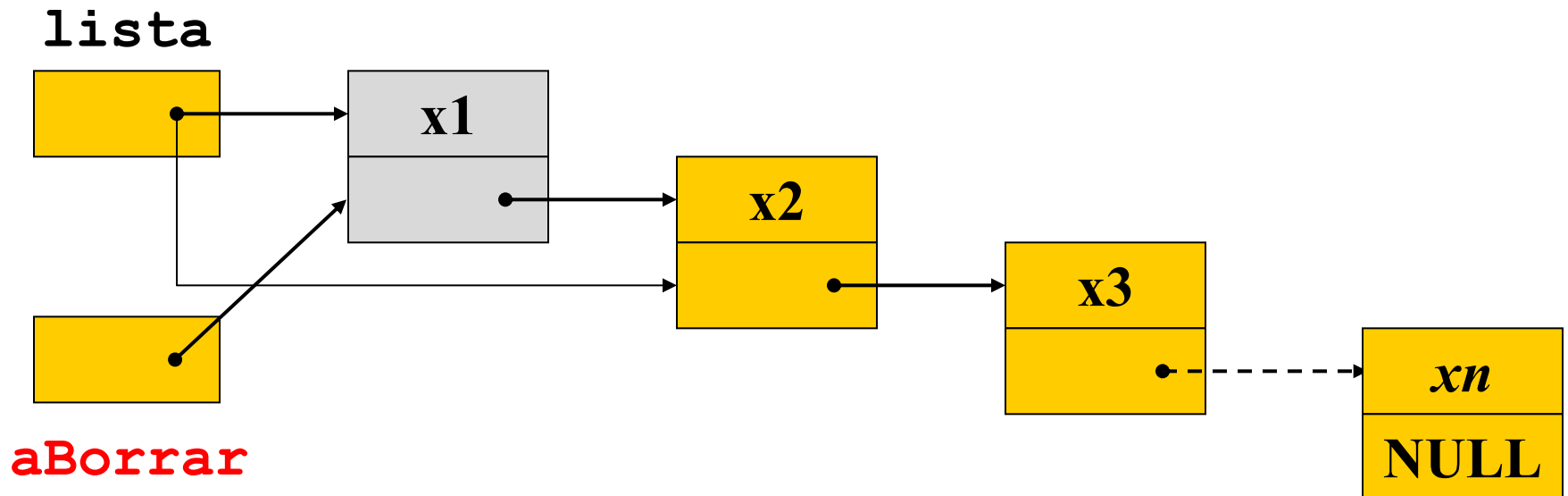
- Dada una lista, deseamos borrar el primer elemento.
- La funcion tiene alguna PRE condicion?

void borrarPrincipio(NodoLista* &lista);



Borrar principio en una lista

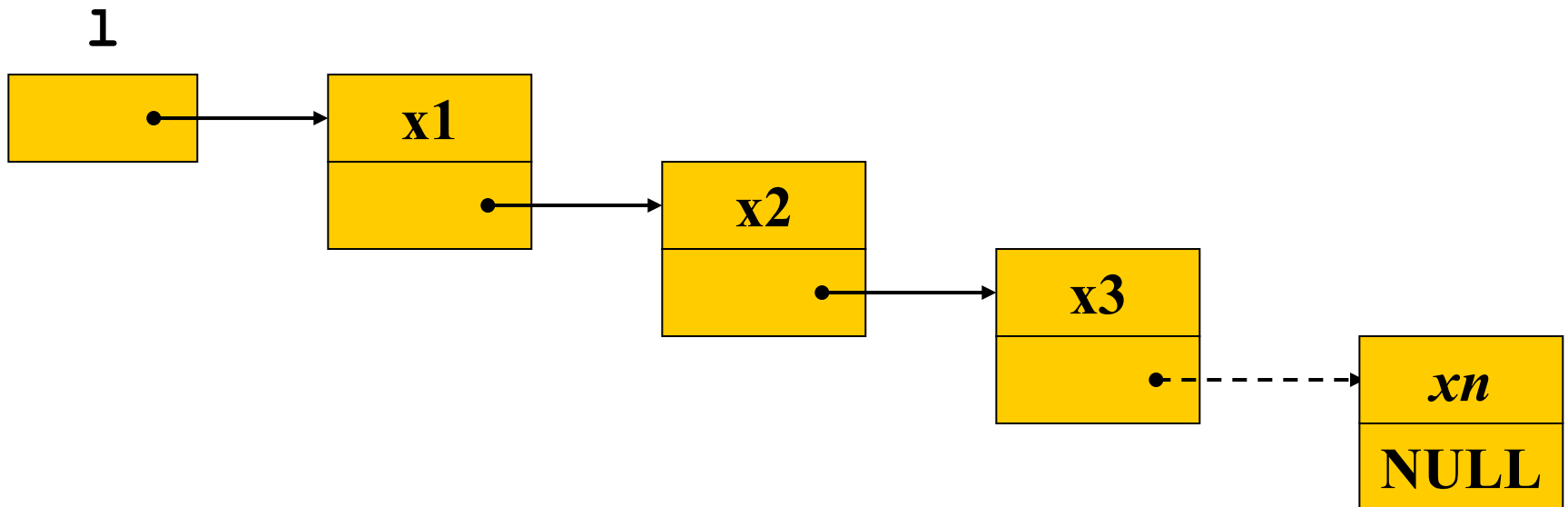
```
void borrarPrincipio(NodoLista* &lista) {  
    NodoLista* aBorrar = lista;  
    lista = lista->sig; // Borrado lógico  
    delete aBorrar; // Borrado físico  
}
```



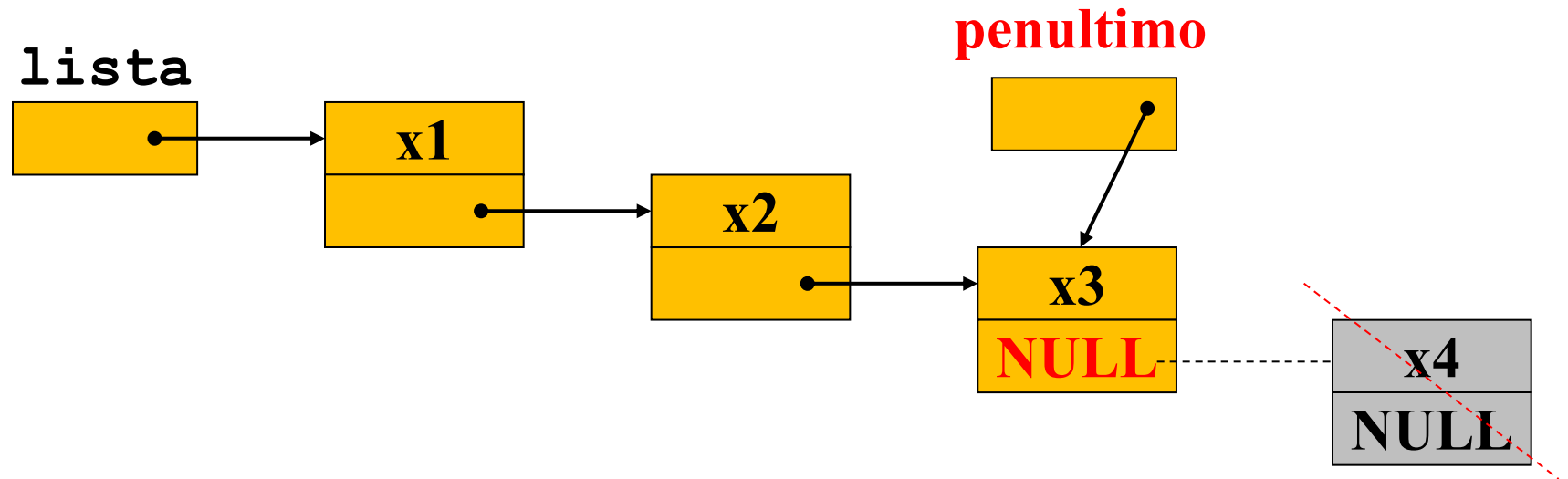
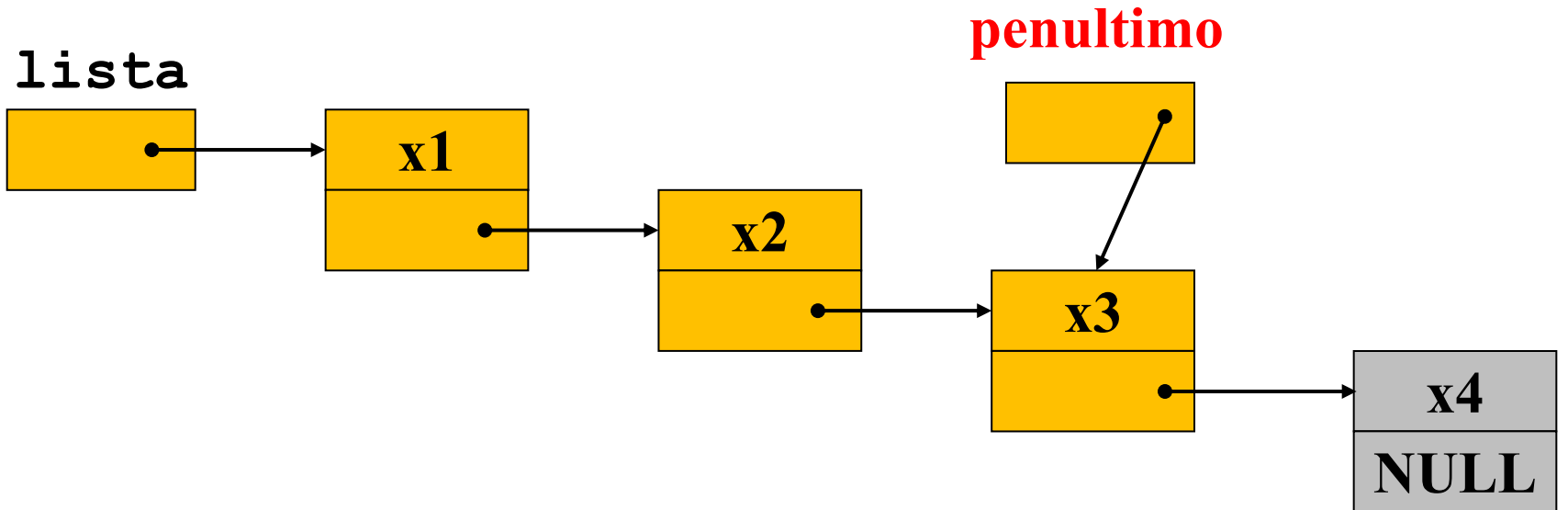
Borrar fin en una lista

- Dada una lista, deseamos borrar el ultimo elemento.
- La funcion tiene alguna PRE condicion?

void borrarFin(NodoLista* &lista);



Borrar fin en una lista (Iterativo)



Borrar fin en una lista (Iterativo)

```
void borrarFin(NodoLista* &lista) {  
    if(lista != NULL) {  
        if(lista->sig == NULL) {  
            delete lista;  
            lista = NULL;  
        }else{  
            NodoLista* penultimo = lista;  
            while(penultimo->sig->sig != NULL) {  
                penultimo = penultimo->sig;  
            }  
            delete penultimo->sig;  
            penultimo->sig = NULL;  
        }  
    }  
}
```

Borrar fin en una lista (Recursivo)

borrarFin: NLista \rightarrow **NLista**

borrarFin (**[]**) = **[]**

borrarFin(**x.[]**) = **[]**

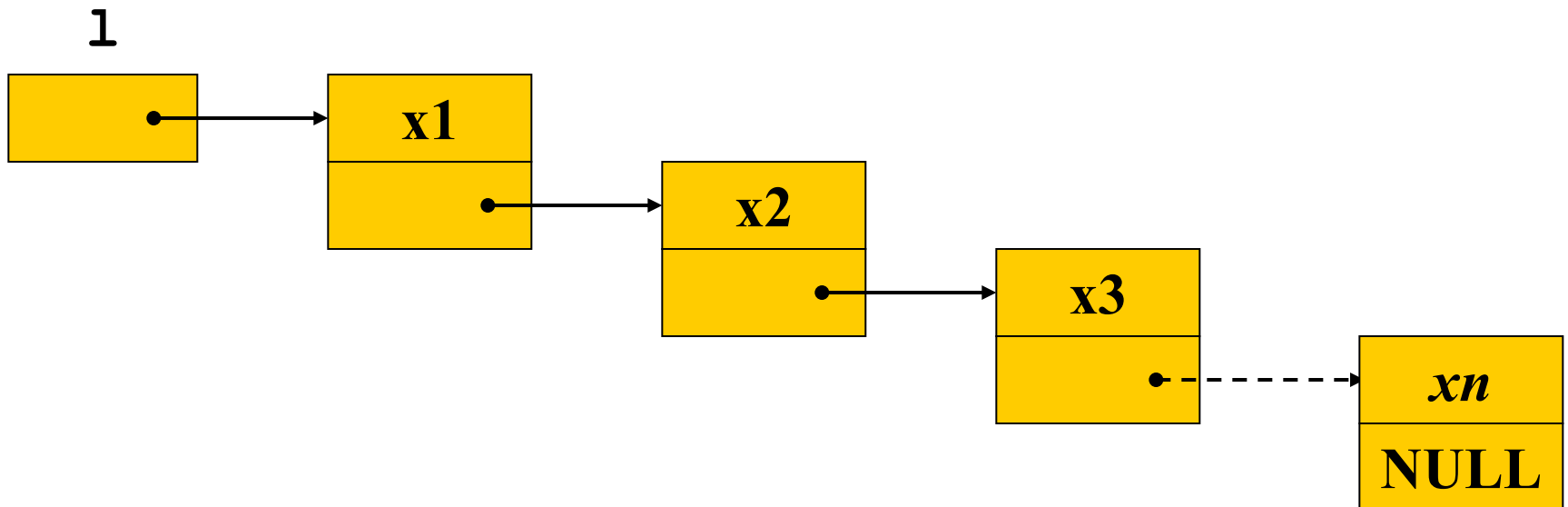
borrarFin (**x.S**) = **x.borrarFin**(**S**)

```
void borrarFin(NodoLista* &lista) {  
    if(lista != NULL){  
        if(lista->sig == NULL){  
            delete lista;  
            lista = NULL;  
        }else{  
            borrarFin(lista->sig) ;  
        }  
    }  
}
```

Borrar toda la lista

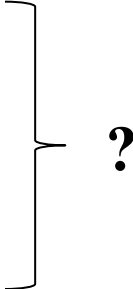
- Dada una lista, deseamos borrar todos los elementos.
- La función tiene alguna PRE condición?

void borrarTodo(NodoLista* &lista);



Borrar toda la lista (Iterativo)

```
void borrarTodo(NodoLista* &lista) {  
    while(lista != NULL) {  
        NodoLista* aBorrar = lista;  
        lista = lista->sig;  
        delete aBorrar;  
    }  
}
```



Borrar toda la lista (Recursivo)

```
void borrarTodo(NodoLista* &lista) {  
    if(lista != NULL){  
        borrarTodo(lista->sig);  
        delete lista;  
        lista = NULL;  
    }  
}
```

Imprimir una lista

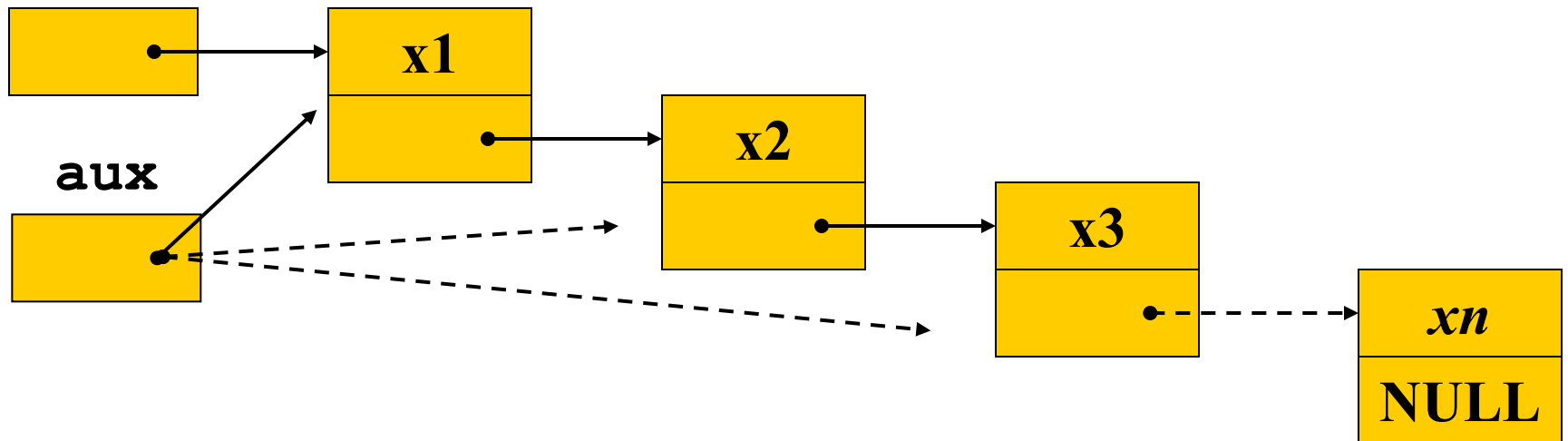
- Dada una lista, deseamos imprimir todos los elementos.

```
void impLista(NodoLista* lista);
```

Imprimir una lista (Iterativo)

```
void impLista(NodoLista* &lista) {  
    NodoLista* aux = lista;  
    while(aux != NULL) {  
        cout << aux->dato;  
        aux = aux->sig;  
    }  
}
```

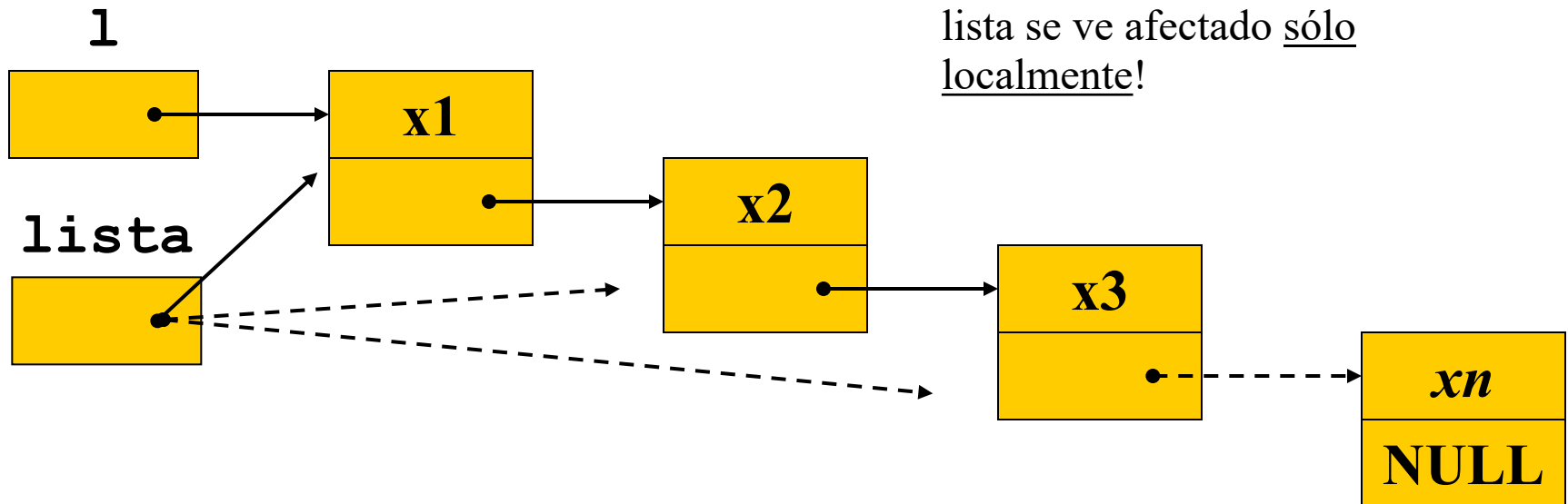
`l=lista`



Imprimir una lista (Iterativo)

```
void impLista(NodoLista* lista) {  
    while(lista != NULL){  
        cout << lista->dato;  
        aux = lista->sig;  
    }  
}
```

Al ser **lista** un parámetro por copia, el puntero al comienzo de la lista se ve afectado sólo localmente!



Imprimir una lista (Recursivo)

```
void impLista(NodoLista* lista) {  
    if(lista != NULL){  
        cout << lista->dato;  
        impLista(lista->sig);  
    }  
}
```

```
void impLista(NodoLista* lista) {  
    if(lista != NULL){  
        impLista(lista->sig);  
        cout << lista->dato;  
    }  
}
```

En orden inverso, basta con solo cambiar el orden de la llamada recursiva.

Insertar ordenado en una lista

- Realizar una función insertar ordenado que dado una lista **ordenada**, inserte un elemento en ella de manera ordenada.

void insertarOrdenado(NodoLista* &lista, int dato);

Implementar de forma recursiva y de forma iterativa

Insertar ordenado en una lista

insertarOrdenado: **NLista** \rightarrow N \rightarrow NLista

insertarOrdenado (**[]**, n) = n.[]

insertarOrdenado(**x.S**, n) = si $n \leq x$: n.x.S

insertarOrdenado(**x.[]**, n) = sino: x.insertarOrdenado(**S**, n)

Insertar ordenado en una lista (Recursivo)

```
void insertarOrdenado(NodoLista* &lista, int n) {  
    if(lista == NULL){  
        agregarPpio(lista, n);  
    } else {  
        if(lista->dato <= n){  
            agregarPpio(lista, n);  
        }else{  
            insertarOrdenado(lista->sig, n);  
        }  
    }  
}
```

Copiar listas

- Realizar una función copiar que dado una lista realice y retorne una copia de la misma.

NodoLista* copiar(NodoLista* lista);

Implementar de forma recursiva y de forma iterativa

Copiar lista (Recursivo)

copiar: **NLista** → NLista

copiar ([]) = []

copiar(**x**.[]) = **x**'.**copiar**(**S**)

```
NodoLista* copiar(NodoLista* lista) {  
    if(lista == NULL){  
        return NULL;  
    } else {  
        NodoLista* copia = new NodoLista;  
        copia->dato = lista->dato;  
        copia->sig = copiar(lista->sig);  
        return copia;  
    }  
}
```

Copiar lista (Iterativo)

```
NodoLista* copiar (NodoLista* lista){
    NodoLista* copia = NULL;
    while(lista != NULL){
        insertarFin(copia, lista->dato);
        lista = lista->sig;
    }
    return copia;
}
```

Concatenar dos listas

- Realizar una función concatenar que dado dos listas concatene estas, es decir junte el final de la primera con el comienzo de la segunda.
- La lista retornada no debe compartir memoria con las dos listas recibidas.

NodoLista* concatenar(NodoLista* l1, NodoLista* l2);

Implementar de forma recursiva y de forma iterativa

Concatenar dos listas

concatenar: NLista → NLista → NLista

concatenar ([], l2) = copiarLista(l2)

concatenar(l1, []) = copiarLista(l1)

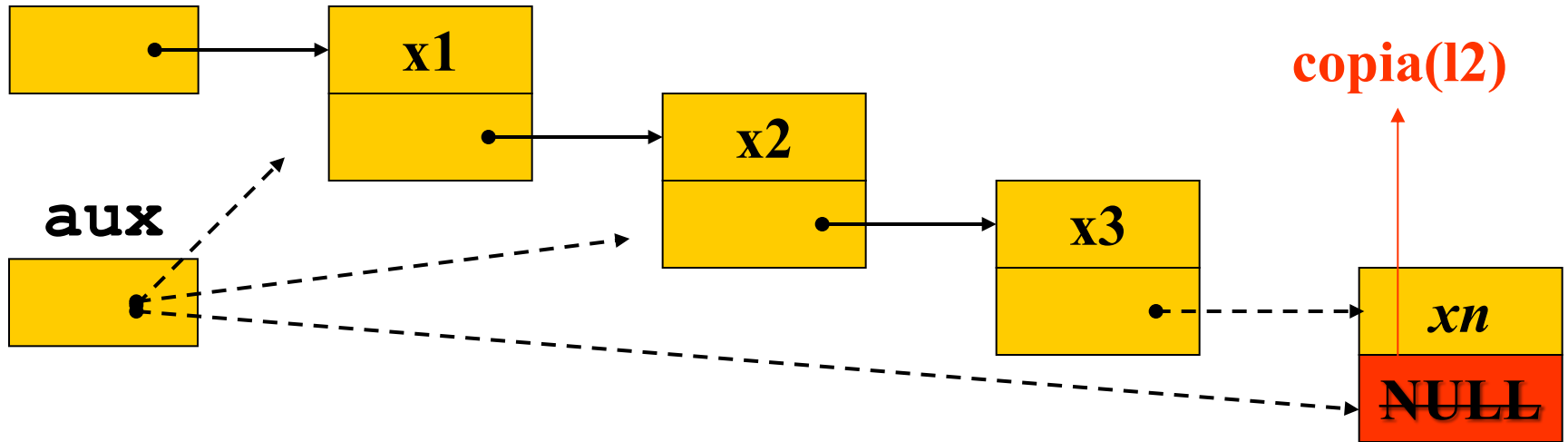
concatenar(x.S, l2) = x'.concatenar(S, l2)

Concatenar dos listas (Recursivo)

```
void concatenar(NodoLista* l1, NodoLista* l2) {  
    if(l1 == NULL){  
        return copiarLista(l2);  
    } else if(l2 == NULL){  
        return copiarLista(l1);  
    } else {  
        NodoLista* nodo = new NodoLista;  
        nodo->dato = l1->dato;  
        nodo->sig = concatenar(l1->sig, l2);  
        return nodo;  
    }  
}
```

Concatenar dos listas (Iterativo)

copiaL1



Concatenar dos listas (Iterativo)

```
void concatenar(NodoLista* l1, NodoLista* l2) {  
    if(l1 == NULL){  
        return copiarLista(l2);  
    } else if(l2 == NULL){  
        return copiarLista(l1);  
    } else {  
        NodoLista* copiaL1 = copiarLista(l1);  
        NodoLista* aux = copiaL1;  
        while(aux->sig != NULL){  
            aux = aux->sig;  
        }  
        aux->sig = copiarLista(l2);  
        return copiaL1;  
    }  
}
```

Interseccion de dos listas

- Realizar una función interseccion que dada dos listas ordenadas de menor a mayor sin elementos repetidos, retorna una nueva lista que no comparte memoria y contiene las interseccion de estas.

NodoLista* interseccion(NodoLista* l1, NodoLista* l2);

La funcion no debe recorrer l1 y l2 mas de una vez.

Implementar de forma recursiva y de forma iterativa

Interseccion de dos listas

interseccion: NLista \rightarrow NLista \rightarrow NLista

interseccion ([], l2) = []

interseccion (l1, []) = []

interseccion(x.S2, y.S2) = si $x < y$: intersección(S2, y.S2)

interseccion(x.S2, y.S2) = si $x > y$: intersección(x.S2, S2)

interseccion(x.S2, y.S2) = si $x == y$: x'.intersección(S2, S2)

Interseccion de dos listas (Recursivo)

```
NodoLista* interseccion(NodoLista* l1, NodoLista* l2) {
    if(l1 == NULL || l2 == NULL){
        return NULL;
    } else {
        if(l1->dato < l2->dato){
            return interseccion(l1->sig, l2);
        }else if(l1->dato > l2->dato){
            return interseccion(l1, l2->sig);
        }else {
            NodoLista* nodo = new NodoLista;
            nodo->dato = l1->dato;
            nodo->sig = interseccion(l1->sig, l2->sig);
            return nodo;
        }
    }
}
```

Interseccion de dos listas (Iterativo)

```
NodoLista* interseccion(NodoLista* l1, NodoLista* l2) {  
    NodoLista* lista = NULL;  
    while(l1 != NULL && l2 != NULL) {  
        if(l1->dato < l2->dato) {  
            l1 = l1->sig;  
        }else if(l1->dato > l2->dato) {  
            l2 = l2->sig;  
        }else {  
            agregarFin(lista, l1->dato);  
            l1 = l1->sig;  
            l2 = l2->sig;  
        }  
    }  
    return lista;  
}
```