ESTRUCTURAS DE DATOS

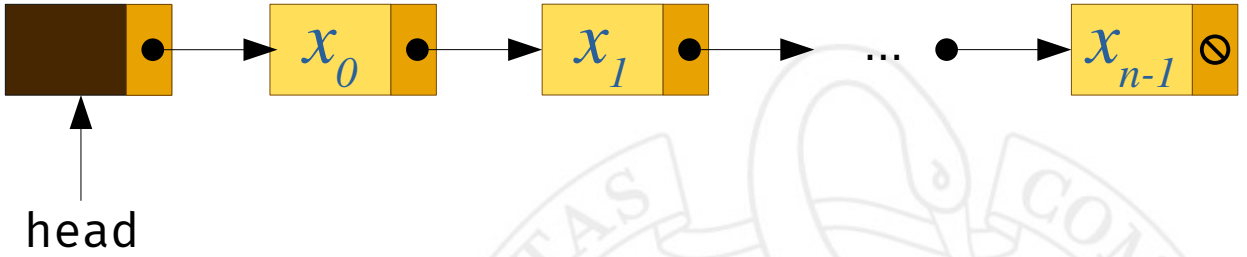
TIPOS ABSTRACTOS DE DATOS LINEALES

Listas doblemente enlazadas (1)

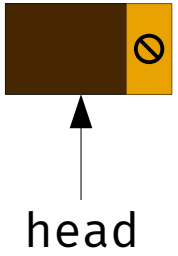
Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Recordatorio: listas enlazadas simples

$[\ x_0, x_1, \ldots, x_{n-1}\ ]$



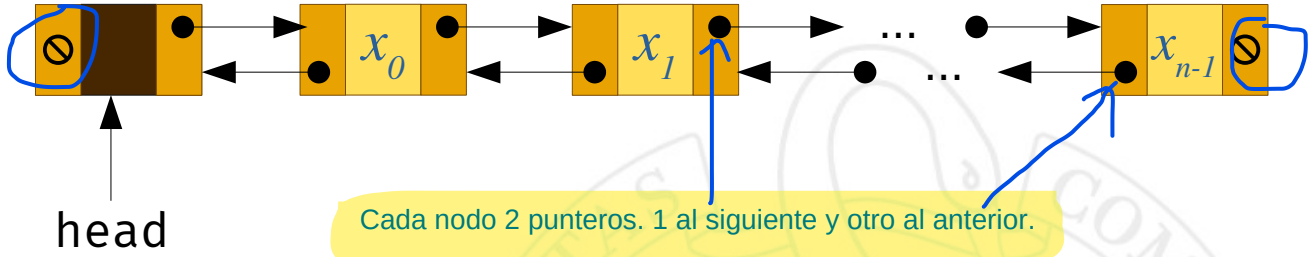
$[\ ]$



Listas doblemente enlazadas

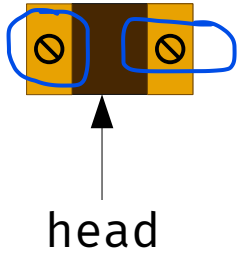
Apunta al siguiente y al anterior nodo, pero el primero, el head, no tiene anterior, y el último no tiene siguiente

$[\ x_0, x_1, \ldots, x_{n-1}\ ]$



Cada nodo 2 punteros. 1 al siguiente y otro al anterior.

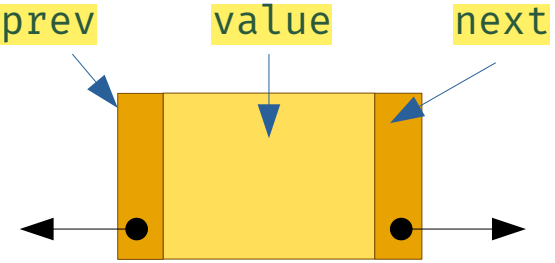
$[\ ]$



Ni sucesor ni predecesor.

Listas enlazadas dobles

```
struct Node {
  std::string value;
  Node *next;
  Node *prev;
};
```



- Cada nodo tiene dos punteros:
  - `next`: Nodo siguiente en la lista enlazada.
  - `prev`: Nodo anterior en la lista enlazada.

Implementación: ListLinkedDouble

```
class ListLinkedDouble {
public:
  ListLinkedDouble();
  ListLinkedDouble(const ListLinkedDouble &other);
  ~ListLinkedDouble();

  void push_front(const std::string &elem);
  void push_back(const std::string &elem);
  void pop_front();
  void pop_back();
  int size() const;
  bool empty() const;
  const std::string & front() const;
  std::string & front();
  const std::string & back() const;
  std::string & back();
  const std::string & at(int index) const;
  std::string & at(int index);
  void display() const;
private:
   ...
  Node *head;
};
```

cons

acceso

mod

Esto cambia

Nueva lista doblemente enlazada.

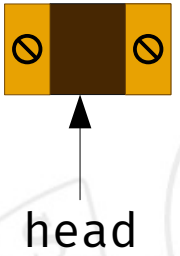
esto cambia

ESTOS SON IGUALES QUE EN LAS LISTAS SIMPLES.

Constructores y destructor

```
ListLinkedDouble() {
  head = new Node;
  head→next = nullptr;
  head→prev = nullptr;
}
```

incializar el prev al puntero nulo




```
~ListLinkedDouble() {
  delete_list(head);
}
```

libera memoria de todos los nodos de la lista

NO CAMBIA

```
ListLinkedDouble(const ListLinkedDouble &other)
    : head(copy_nodes(other.head)) { }
```

constructora de copia. La función copy nodes que copia los nodos + a quién apuntaba, ahora apunta a dos sitios distintos.

SI CAMBIA

Copia de una cadena de nodos

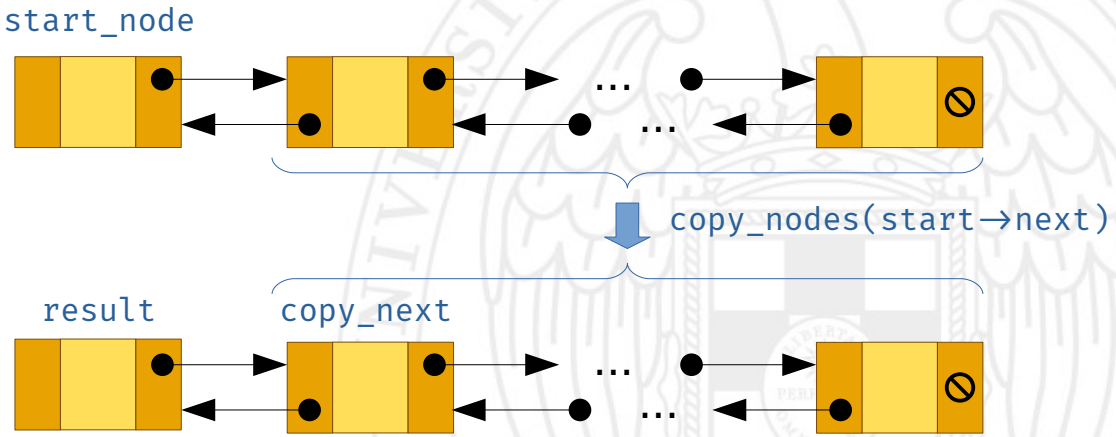
Implementación recursiva

Nodo a partir del cual empezamos la copia

```
Node * ListLinkedDouble::copy_nodes(Node *start_node) const {
  if (start_node ≠ nullptr) {
    Node *copy_next = copy_nodes(start_node→next);
    Node *result = new Node { start_node→value, copy_next, nullptr };
    if (copy_next ≠ nullptr) {
      copy_next→prev = result;
    }
    return result;
  } else {
    return nullptr;
  }
}
```

llama recursivamente sobre el next.

valor siguiente previo

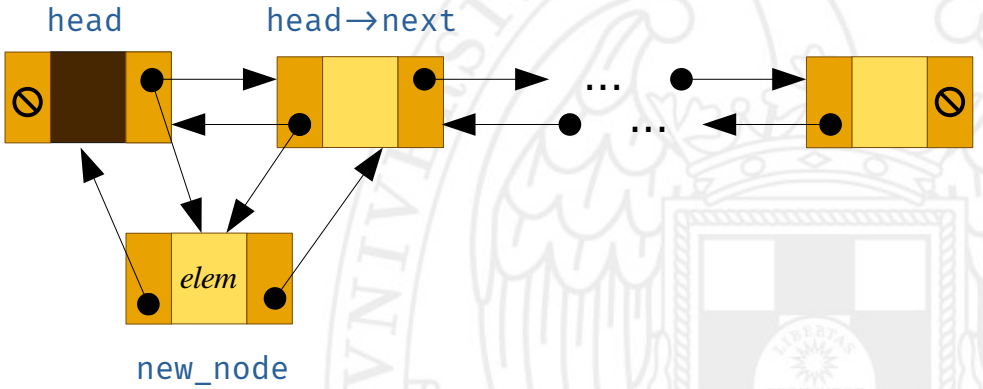


Insertar al principio de la cadena

```
void push_front(const std::string &elem) {
  Node *new_node = new Node { elem, head→next, head };
  if (head→next ≠ nullptr) {si no es el último
    head→next→prev = new_node;
  }
  head→next = new_node;
}
```

modifico prev de head->next y apunta al nuevo nodo creado, no al nodo fantasma (head).

el siguiente del nodo fantasma es el new node.



Eliminar al principio de la cadena

```
void pop_front() {
  assert (head→next ≠ nullptr);
  Node *target = head→next;
  head→next = target→next;
  if (target→next ≠ nullptr) {
    target→next→prev = head;
  }
  delete target;
}
```

Queremos eliminar head->next.

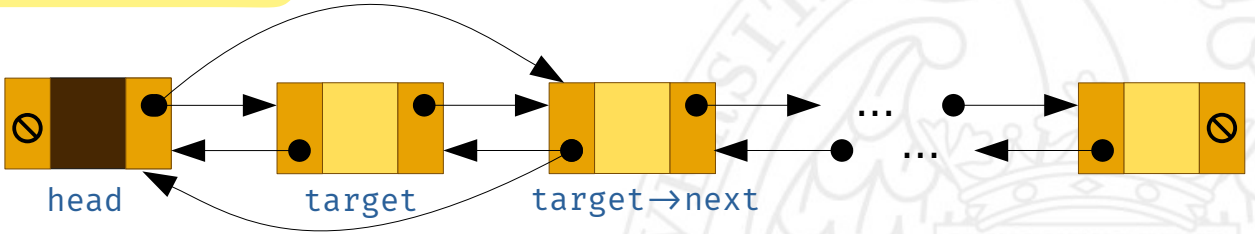
Este es el que queremos eliminar

Enlazamos head con el sucesor de target.

si el sucesor de target NO es nullptr.

el siguiente a target, su previo apunta a head.

eliminamos el siguiente



Insertar al final de la cadena

```
void push_back(const std::string &elem) {
  Node *last = last_node();
  Node *new_node = new Node { elem, nullptr, last };
  last→next = new_node;
}
```

accedo al último nodo de la cadena

hacemos que el nuevo nodo->next apunte a null (ultimo elemento) y el previous sea el que antes era el último

que el siguiente al último sea el nuevo

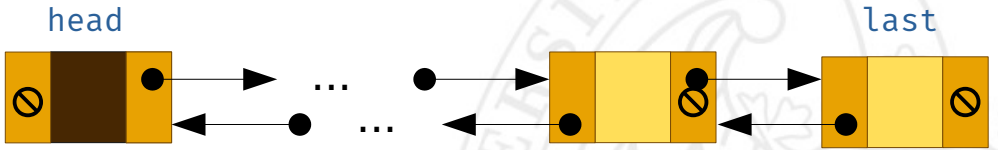
Eliminar al final de la cadena

```
void pop_back() {
  assert (head→next ≠ nullptr);
  Node *last = last_node();
  last→prev→next = nullptr;
  delete last;
}
```

el último

hacemos que el previo, su next apunte a null, es decir, que el penúltimo apunte a null

borramos el último



¿Mejoras en el coste?

DE MOMENTO NO HAY NIGUNA MEJORA EN EL COSTE

Operación	Listas enlazadas simples	Listas doblemente enlazadas
Creación	$O(1)$	$O(1)$
Copia	$O(n)$	$O(n)$
`push_back`	$O(n)$	$O(n)$
`push_front`	$O(1)$	$O(1)$
`pop_back`	$O(n)$	$O(n)$
`pop_front`	$O(1)$	$O(1)$
`back`	$O(n)$	$O(n)$
`front`	$O(1)$	$O(1)$
`display`	$O(n)$	$O(n)$
`at(index)`	$O(index)$	$O(index)$
`size`	$O(n)$	$O(n)$
`empty`	$O(1)$	$O(1)$

n = *número de elementos de la lista de entrada*