ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS LINEALES

# Introducción a los iteradores

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

¿Para qué sirve el TAD iterador?

Para poder recorrer la lista desde el primer elemento hasta el último. los iteradores lo hacen de manera homogénea y puede ser más eficiente.

# Motivación

# Problema

- Tenemos una lista de enteros, y queremos calcular la suma de todos los elementos de la lista.

```
int suma_elems(const ListLinkedDouble<int> &l) {
  int suma = 0;
  for (int i = 0; i < l.size(); i++) {
    suma += l[i];
  }
  return suma;
}
```

este tiene coste constante almacenamos el tamaño de la lista en una tributo de la clase.

esto es como el at, por tanto el coste de hacer esta asignación sería LINEAL.

¿Qué coste tiene esta función?

Está función tiene coste cuadrático.

DEMASIADO CARO.

Cuadrático porque hacemos un for de coste lineal el cual en cada iteración del bucle se hace la operación l que es de coste lineal, luego n^2.

# Posible solución 1

- Utilizar otra implementación de listas, de modo que la operación `at()` tenga coste constante.

Implementación con arrays en vez de con nodos.

```
int suma_elems(const ListArray<int> &l) {
  int suma = 0;
  for (int i = 0; i < l.size(); i++) {
    suma += l[i];
  }
  return suma;
}
```

Esto ahora es constante, y en total lineal.

¿Y si necesito una lista enlazada?

# Posible solución 2

- Hacer copia de la lista de entrada, e ir eliminando los elementos de la copia uno a uno.

hacer una copia de la lista.

```
int suma_elems(const ListLinkedDouble<int> &l) {
  ListLinkedDouble<int> copia = l;
  int suma = 0;
  while (!copia.empty()) {
    suma += copia.front();
    copia.pop_front();
  }
  return suma;
}
```

elemento del principio y lo eliminamos. Modificamos una copia. Por tanto coste lineal.

Para así no tener que hacer ninguna modificación de la lista doble

¿Cuál es el coste en espacio de esta solución?

# Posible solución 3

- Integrar la operación dentro de la clase `ListLinkedDouble`.

```
template <typename T>
class ListLinkedDouble {
   ...

  int suma_elems() const {
    int suma = 0;
    Node *current = head→next;
    while (current ≠ head) {
      suma += current→value;
      current = current→next;
    }
    return suma;
  }
};
```

¿Y si la lista no es de enteros?

¿Tengo que prever de antemano todas las cosas que puedo hacer en el recorrido de una lista?

tendría que definir el operador + para el tipo de datos T con el que instanciamos la lista.

# La solución que presentamos

- Proporcionar una abstracción al programador/a para que pueda navegar por los elementos de una lista, de modo independiente de la implementación.
- La navegación se realiza de manera secuencial.
- Esta abstracción recibe el nombre de **iterador**.

No hace falta saber cómo está implementada.

De un elemento al siguiente. De un elemento al siguiente.

# Iteradores

Vamos a presentar a los iteradores.

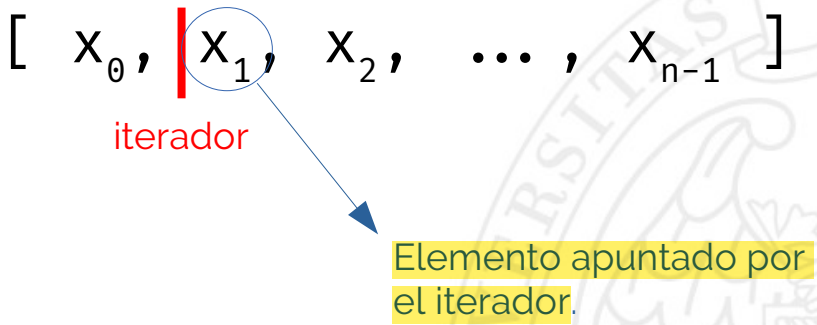
# Iteradores

- Un iterador es un cursor que se mueve por los elementos de la lista de manera secuencial.
- Es posible realizar operaciones de acceso y modificación en la posición actual de un iterador.

# Iteradores

Es un cursor ligado a la lista.

- En el momento de su creación, un iterador está ligado a una lista.
- Representamos los iteradores de la siguiente forma:



# Operaciones sobre un iterador

- Obtener el elemento apuntado por el iterador (***elem***)
- Avanzar el iterador a la siguiente posición de la lista (***advance***)
- Igualdad entre dos iteradores (==)
  - Dos iteradores son iguales si recorren la misma lista y apuntan a la misma posición dentro de esta.

$\{ [\ x_0, \ \ldots, \ x_i, \ \ldots, \ x_{n-1}\ ],\ 0 \le i < n \}$ (it)

***elem***(*it: Iterator*) → *(x: Elem)*

$\{ x = x_i \}$

$\{ [\ x_0, \ \ldots, \ x_i, \ \ldots, \ x_{n-1}\ ],\ 0 \le i < n \}$ (it)

***advance***(*it: Iterator*) advance(it)

$\{ [\ x_0, \ \ldots, \ x_i, \ x_{i+1} \ldots, \ x_{n-1}\ ] \}$ (it)

# Creación de iteradores

- Añadimos dos operaciones al TAD Lista:
  - Obtener un iterador al principio de la lista (***begin***)
  - Obtener un iterador al final de la lista (***end***)

Lo usamos para que el iterador empiece apuntando al principio de la lista por ejemplo.

Lo podemos usar para que itere hasta el final de la lista.

$\{ l = [ x_0, \ldots, x_i, \ldots, x_{n-1} ] \}$

***begin***(*l: List*) → *(it: Iterator)*

it

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

$\{ l = [ x_0, \ldots, x_i, \ldots, x_{n-1} ] \}$

***end***(*l: List*) → *(it: Iterator)*

it

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

No apunta al último elemento

# Creación de iteradores

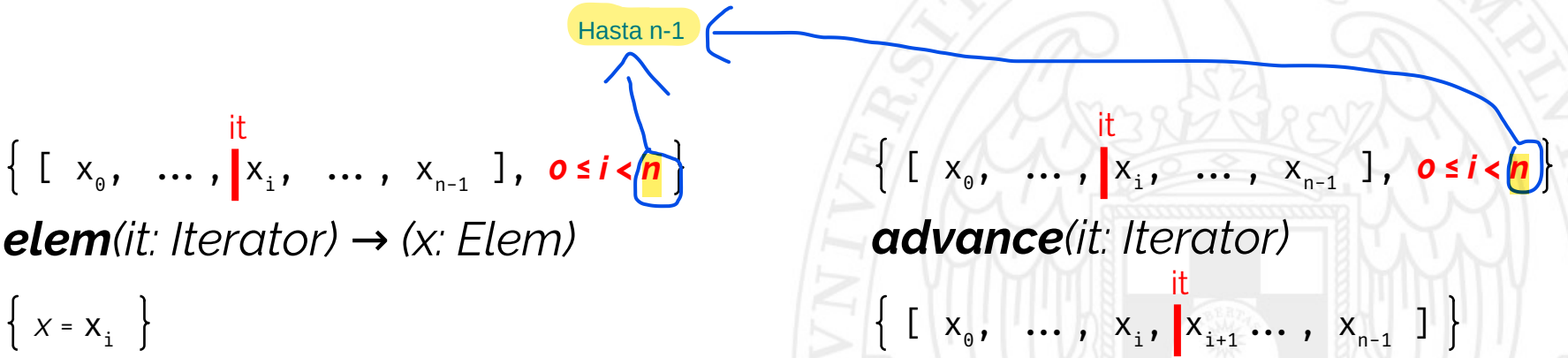
- Añadimos dos operaciones al TAD Lista:
  - Obtener un iterador al principio de la lista (***begin***)
  - Obtener un iterador al final de la lista (***end***)

end solo nos sirve para realizar comprobaciones. Para saber si hemos llegado al final de la lista-

Las operaciones **elem**() y **advance**() no están definidas para el iterador devuelto por **end**()

# Creación de iteradores

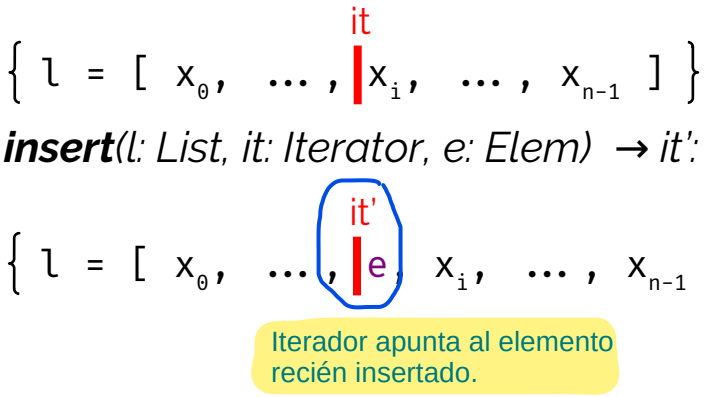
- Añadimos dos operaciones al TAD Lista:
  - Obtener un iterador al principio de la lista (***begin***)
  - Obtener un iterador al final de la lista (***end***)

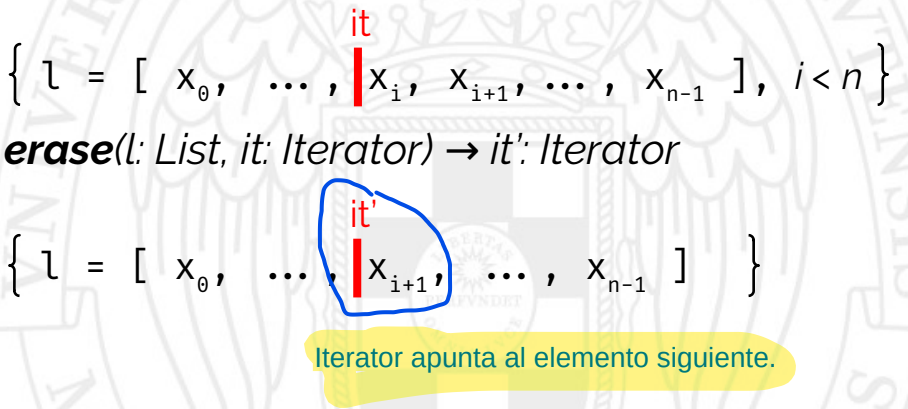


# Operaciones adicionales sobre listas

- Insertar un elemento en la posición apuntada por un iterador (***insert***)
- Eliminar el elemento apuntado por el iterador (***erase***)

Siempre en relación a la posición del iterador.



# Ejemplo: suma de enteros

```
int suma_elems(ListLinkedDouble<int> &l) {
  int suma = 0;
  ListLinkedDouble<int>::iterator it = l.begin();
  while (it != l.end()) {
    suma += it.elem();
    it.advance();
  }
  return suma;
}
```

Mientras que el iterador no esté al final de la lista.

# Ejemplo: suma de enteros

```
int suma_elems_iterator(ListLinkedDouble<int> &l) {
  int suma = 0;
  for (ListLinkedDouble<int>::iterator it = l.begin(); it =/= l.end(); it.advance()) {
    suma += it.elem();
  }
  return suma;
}
```

Se usa bastante.

Todo eso los sustituimos por: "auto"