

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

# Motivación



# Problema

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

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

¿Qué coste tiene esta función?

# Posible solución 1

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

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

¿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.

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

¿Cuál es el coste en espacio de esta solución?

# Posible solución 3

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

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

    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?

# 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**.



# 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.

[ 1, 4, 15, 7, 10, 23 ]

# Iteradores

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

$[ x_0, | x_1, x_2, \dots, x_{n-1} ]$

iterador

Elemento apuntado por  
el iterador.

# 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.

$\{ [ x_0, \dots, \overset{\text{it}}{\mid} x_i, \dots, x_{n-1} ], 0 \leq i < n \}$

***elem***(it: Iterator)  $\rightarrow$  (x: Elem)

$\{ x = x_i \}$

$\{ [ x_0, \dots, \overset{\text{it}}{\mid} x_i, \dots, x_{n-1} ], 0 \leq i < n \}$

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

$\{ [ x_0, \dots, x_i, \overset{\text{it}}{\mid} x_{i+1}, \dots, x_{n-1} ] \}$

# 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***)

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

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

$$\{ [ \overset{\text{it}}{\color{red}|} x_0, \dots, x_i, \dots, x_{n-1} ] \}$$
$$\{ l = [ x_0, \dots, x_i, \dots, x_{n-1} ] \}$$

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

$$\{ [ x_0, \dots, x_i, \dots, x_{n-1} \overset{\text{it}}{\color{red}|} ] \}$$

# 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***)



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***)

$\{ [x_0, \dots, \overset{\text{it}}{|}x_i, \dots, x_{n-1}], \textcolor{red}{0 \leq i < n} \}$

***elem***(*it: Iterator*)  $\rightarrow$  (*x: Elem*)

$\{ x = x_i \}$

$\{ [x_0, \dots, \overset{\text{it}}{|}x_i, \dots, x_{n-1}], \textcolor{red}{0 \leq i < n} \}$

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

$\{ [x_0, \dots, x_i, \overset{\text{it}}{|}x_{i+1} \dots, x_{n-1}] \}$

# Operaciones adicionales sobre listas

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

$$\{ l = [ x_0, \dots, \overset{\text{it}}{\mid} x_i, \dots, x_{n-1} ] \}$$

***insert***(*l*: List, *it*: Iterator, *e*: Elem)  $\rightarrow$  *it'*: Iterator

$$\{ l = [ x_0, \dots, \overset{\text{it}'}{\mid} e, x_i, \dots, x_{n-1} ] \}$$

$$\{ l = [ x_0, \dots, \overset{\text{it}}{\mid} x_i, x_{i+1}, \dots, x_{n-1} ], i < n \}$$

***erase***(*l*: List, *it*: Iterator)  $\rightarrow$  *it'*: Iterator

$$\{ l = [ x_0, \dots, \overset{\text{it}'}{\mid} x_{i+1}, \dots, x_{n-1} ] \}$$

# Ejemplo: suma de enteros

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





# 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;  
}
```

