


ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS LINEALES

# Iteradores en ListArray

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



# Recordatorio: ListArray

```
template<typename T>
class ListArray {
public:
```

```
    ...
private:    contador con los elementos
            que tenemos en la lista.
```

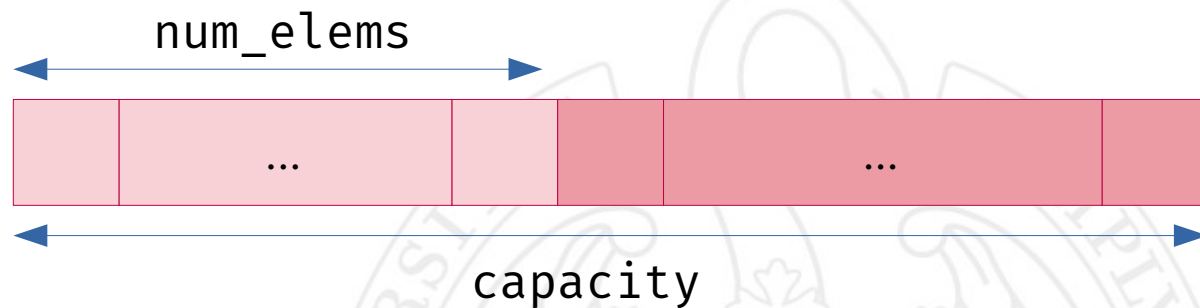
```
    int num_elems;
```

```
    int capacity;
```

```
    T *elems;
```

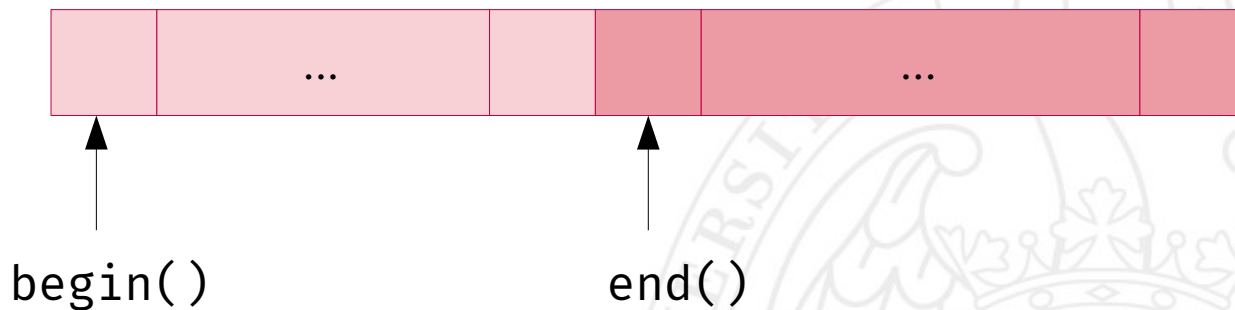
```
};    Array de elementos
```

Objetivo es ver el concepto de los iteradores como una generalización de los punteros.



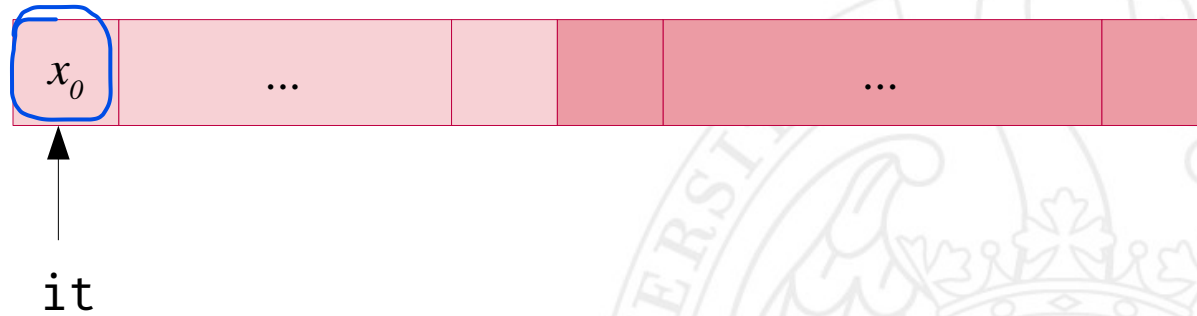
# Iteradores en ListArray

- Los iteradores van a ser **punteros** a los elementos del array.



# Iteradores en ListArray

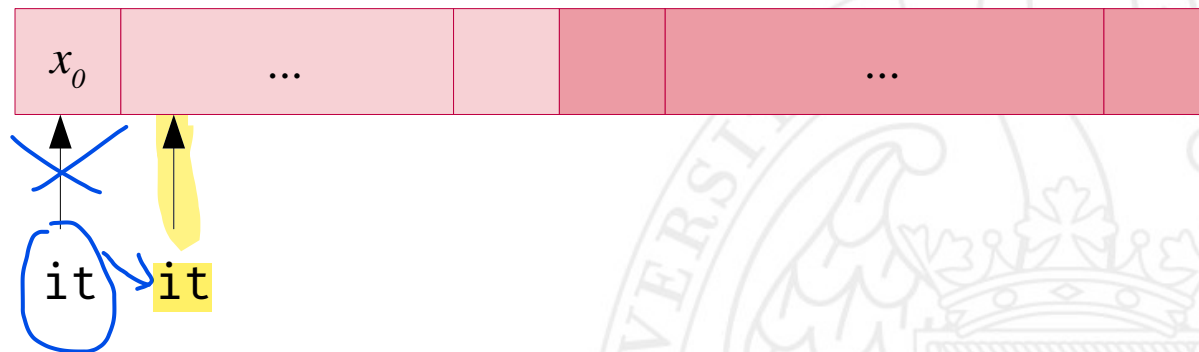
- Los iteradores van a ser **punteros** a los elementos del array.



$$\underline{*it} = x_0$$

# Iteradores en ListArray

- Los iteradores van a ser **punteros** a los elementos del array.



`++it;`

Con esto el iterador avanza a la siguiente posición

# Definición de iteradores

```
template<typename T>
class ListArray {
public:
    ...
    using iterator = T*;
    using const_iterator = const T*;

private:
    int num_elems;
    int capacity;
    T *elems;
};
```

No sería necesario definir una clase iterator.



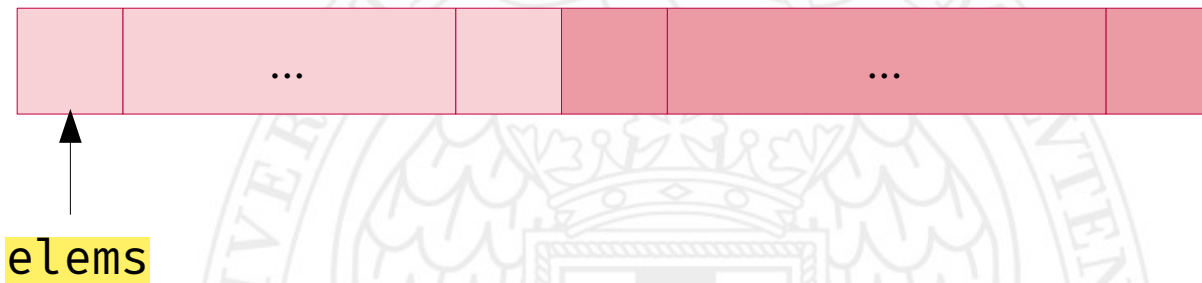
# Método begin()

```
template<typename T>
class ListArray {
public:
    ...
    using iterator = T *;
    using const_iterator = const T *;

    iterator begin() {
        return elems;
    }
    const_iterator begin() const {
        return elems;
    }

private:
    int num_elems;
    int capacity;
    T *elems;
};
```

Puntero a la posición inicial de la lista.



# Método end()

```
template<typename T>
class ListArray {
public:
```

```
    ...
    using iterator = T *;
    using const_iterator = const T *;
```

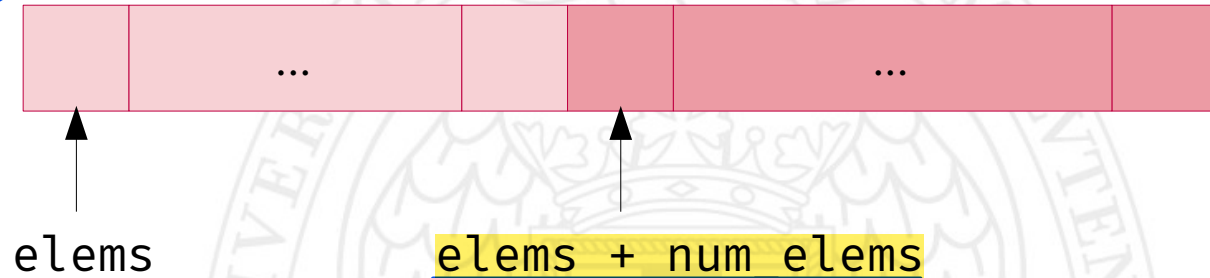
```
    iterator end() {
        return elems + num_elems;
    }
```

```
    const_iterator end() const {
        return elems + num_elems;
    }
```

```
private:
    int num_elems;
    int capacity;
    T *elems;
};
```

es equivalente a:

$\text{elems}[\text{num\_elems}]$





# Ejemplos

En vez de a las listas enlazadas

```
void mult_por_dos(ListArray<int> &l) {  
    for (auto it = l.begin(); it != l.end(); ++it) {  
        *it *= 2;  
    }  
}
```

```
int suma_elems(const ListArray<int> &l) {  
    int suma = 0;  
    for (auto it = l.begin(); it != l.end(); ++it) {  
        suma += *it;  
    }  
    return suma;  
}
```

# Ejemplos

```
void mult_por_dos(ListArray<int> &l) {  
    for (int &x : l) {  
        x *= 2;  
    }  
}
```

ya que modificábamos la lista.

Con la Sintaxis de la hoja 5 del tema.

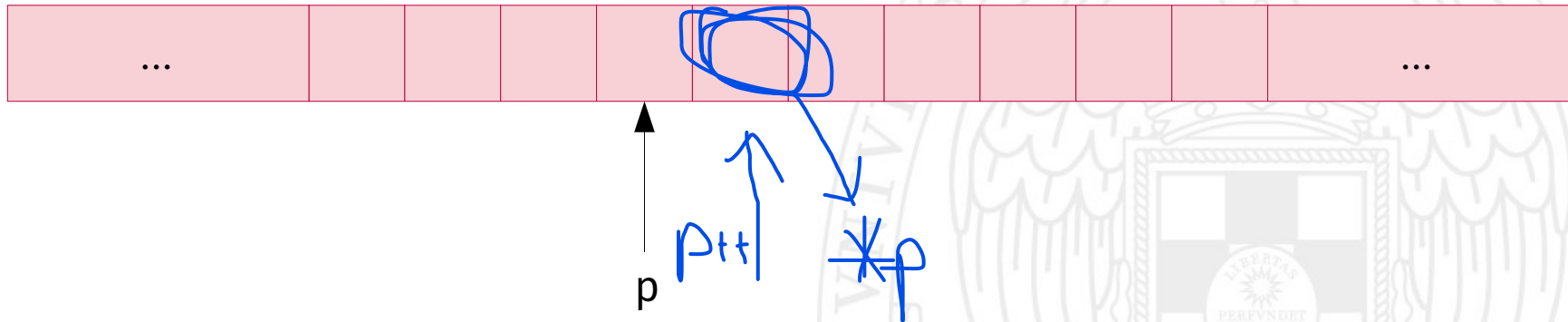
```
int suma_elems(const ListArray<int> &l) {  
    int suma = 0;  
    for (int x : l) {  
        suma += x;  
    }  
    return suma;  
}
```

# Moraleja

- En C++, **los iteradores son generalizaciones de punteros.**

```
char *p;
```

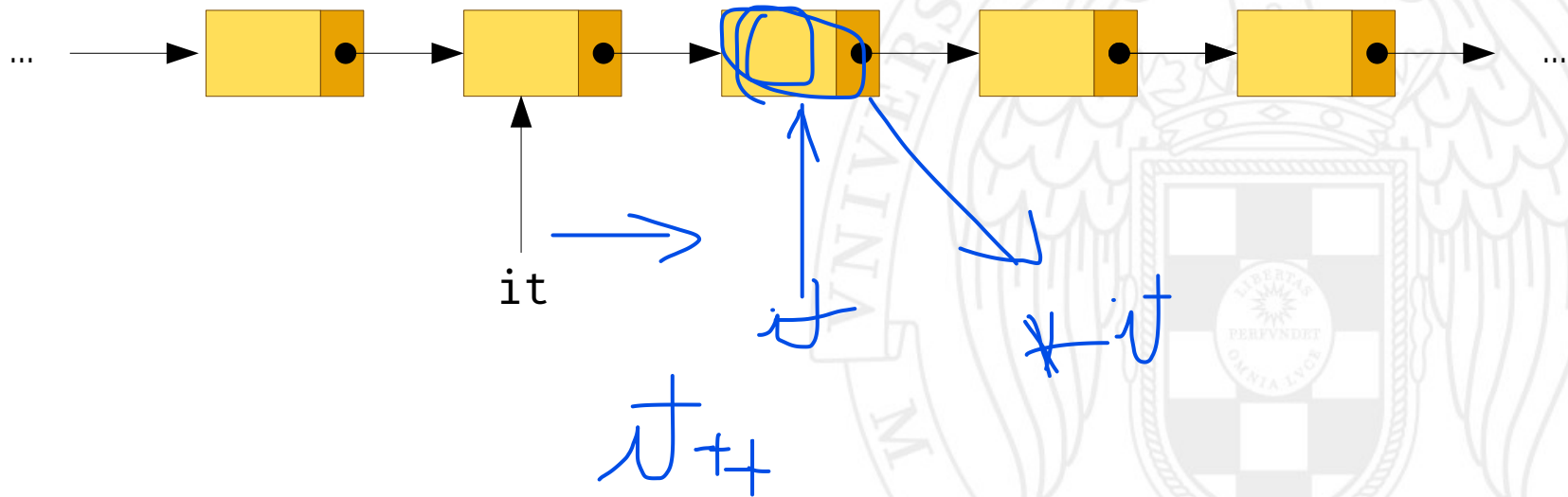
puntero te permite moverte entre las distintas posiciones físicas de memoria.  
un ITERADOR te permite moverte entre los elementos de una estructura de datos.



# Moraleja

- En C++, **los iteradores son generalizaciones de punteros.**

```
ListLinked<int>::iterator it;
```



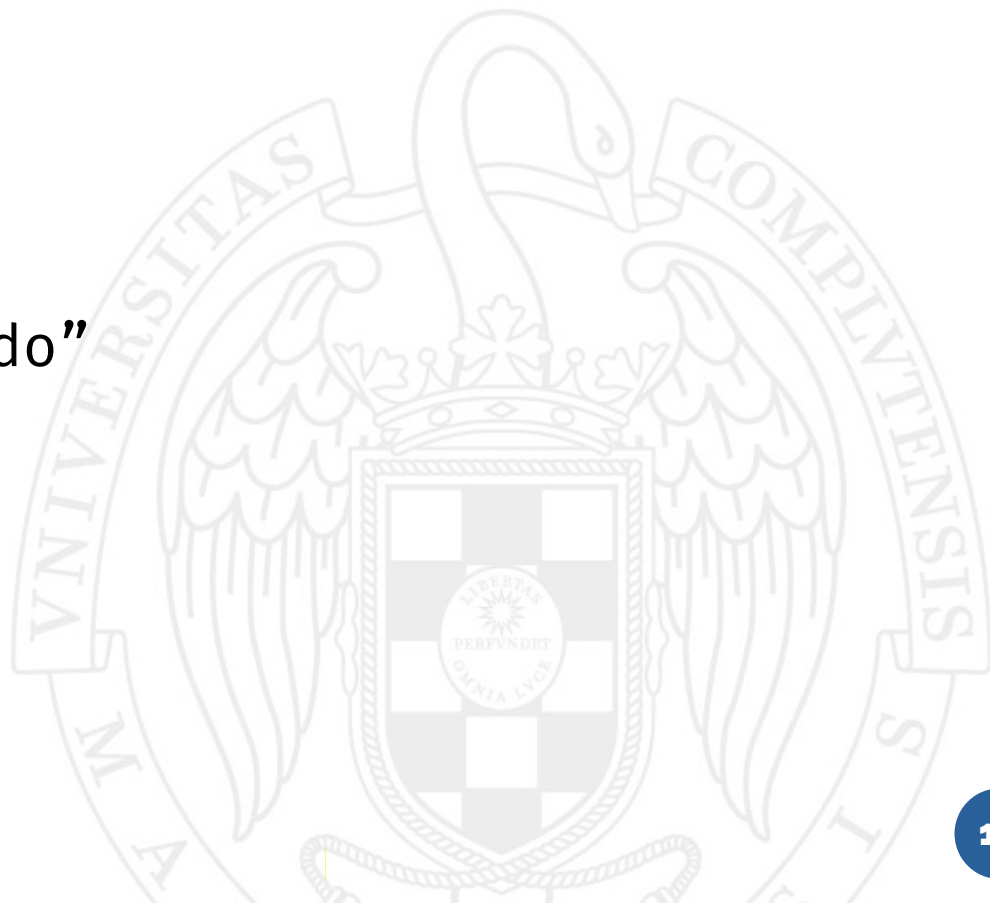
# Moraleja

- En C++, **los iteradores son generalizaciones de punteros.**

```
string::iterator it;
```

“Ho**l**a, mundo”

↑  
**it**



# Moraleja

- En C++, **los iteradores son generalizaciones de punteros.**

```
std::string cadena = "Hola, mundo";  
for (auto it = cadena.begin(); it != cadena.end(); ++it) {  
    std::cout << *it << std::endl;  
}
```