

ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS LINEALES

Sobrecargando operadores en el TAD Lista

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

Sobrecarga del operador <<

Implementamos la sobrecarga de nuestros operadores sobre el TAD Lista.

Generalizando el método display()

```
class ListArray {
public:
    void display() const;
    ...
};

void ListArray::display() const {
    std::cout << "[";
    if (num_elems > 0) {
        std::cout << elems[0];
        for (int i = 1; i < num_elems; i++) {
            std::cout << ", " << elems[i];
        }
    }
    std::cout << "];"
}
```

Método que nosotros teníamos definido antes.

Generalizando el método display()

```
class ListArray {  
public:  
    void display(std::ostream &out) const;  
    ...  
};
```

PARÁMETRO QUE REPRESENTA EL FLUJO DE SALIDA EN EL CUAL QUIERO ESCRIBIR EL FLUJO DE LA LISTA.

```
void ListArray::display(std::ostream &out) const {  
    out << "[";  
    if (num_elems > 0) {  
        out << elems[0];  
        for (int i = 1; i < num_elems; i++) {  
            out << ", " << elems[i];  
        }  
        out << "];"  
    }  
}
```



Sobrecargando el operador <<

```
class ListArray {  
public:  
    void display(std::ostream &out) const;  
    ...  
};
```

```
std::ostream & operator<<(std::ostream &out, const ListArray &l) {  
    l.display(out);  
    return out;  
}
```

Llama al método display de esa lista.

Ejemplo

```
ListArray l1;  
l1.push_back("David");  
l1.push_back("Maria");  
l1.push_back("Elvira");
```

```
ListArray l2 = l1;  
l2.at(1) = "Manuel";
```

```
std::cout << l1 << " " << l2 << std::endl;
```

[David, Maria, Elvira] [David, Manuel, Elvira]

Sobrecarga del operador []



Acceso a elementos de una lista

- El método `at(i)` nos permitía acceder a la posición i -ésima de una lista.

```
std::cout << l.at(1);  
l.at(2) = "Francisco";
```

- Resultaría más intuitiva una notación similar a la de los arrays.

```
std::cout << l[1];  
l[2] = "Francisco";
```

Esta es la notación de los arrays.

- Es posible habilitar esta notación sobrecargando el operador `[]`.
 - Para ello hay que definir un método llamado `operator[]` en la clase lista.
 - La expresión `l[i]` equivale a `l.operator[](i)`

Sobrecarga del operador []

```
class ListArray {
public:
    ...
    const std::string & at(int index) const {
        assert (0 ≤ index && index < num_elems);
        return elems[index];
    }

    std::string & at(int index) {
        assert (0 ≤ index && index < num_elems);
        return elems[index];
    }

    ...
};
```

Implementación anterior de los métodos at.

Sobrecarga del operador []

```
class ListArray {  
public:
```

```
...  
const std::string & operator[](int index) const {  
    assert (0 ≤ index && index < num_elems);  
    return elems[index];  
}
```

Cambiamos el nombre de at, por la sobrecarga del operador[]

```
std::string & operator[](int index) {  
    assert (0 ≤ index && index < num_elems);  
    return elems[index];  
}
```

```
};  
...
```

Ejemplo

```
ListArray l;  
l.push_back("David");  
l.push_back("Maria");  
l.push_back("Elvira");  
l[2] = "Enriqueta";  
std::cout << l << std::endl;
```

[David, Maria, Enriqueta]

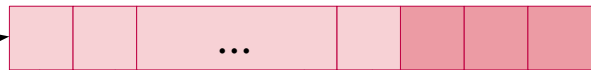
Sobrecarga del operador de asignación



Listas mediante arrays

`l1` de ListArray

num_elems: `n1`
capacity: `c1`
elems: ●



`l2`

num_elems: `n2`
capacity: `c2`
elems: ●



Al hacer la asignación `l2 = l1`.

1) Si los elementos de `l1` caben en `l2`:

- 1) Copiarlos de `l1` a `l2`.
- 2) Copiar el atributo `num_elems`.

Listas mediante arrays

l_1

num_elems: n_1
capacity: c_1
elems: ●



l_2

num_elems: n_1
capacity: c_2
elems: ●



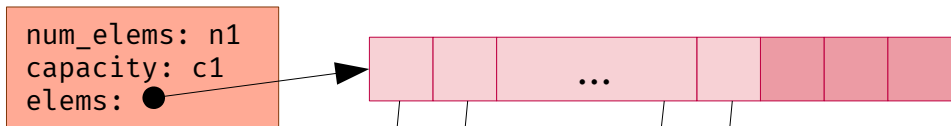
Al hacer la asignación $l_2 = l_1$.

1) Si los elementos de l_1 caben en l_2 :

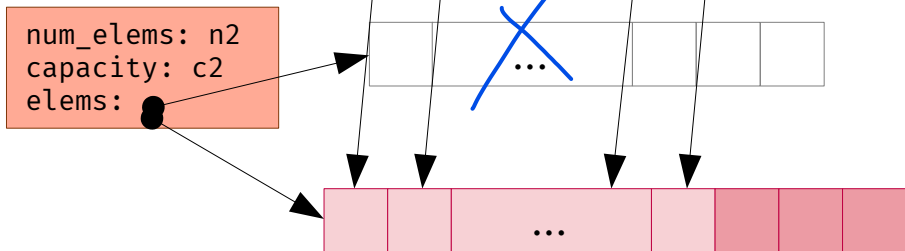
- 1) Copiarlos de l_1 a l_2 .
- 2) Copiar el atributo num_elems.

Listas mediante arrays

l_1



l_2



Al hacer la asignación $l_2 = l_1$.

1) Si los elementos de l_1 caben en l_2 :

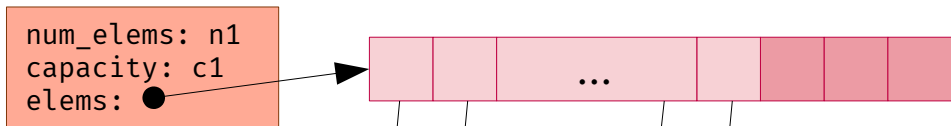
- 1) Copiarlos de l_1 a l_2 .
- 2) Copiar el atributo `num_elems`.

2) En caso contrario:

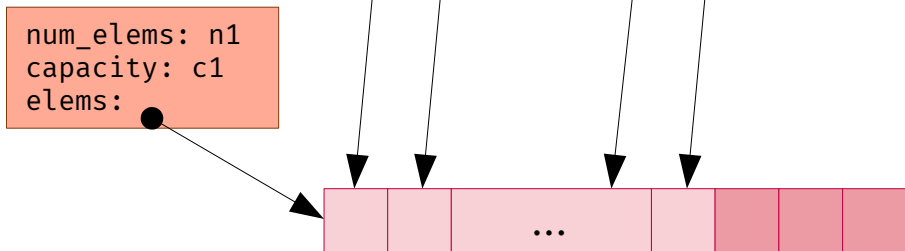
- 1) Desechar `l2.elems` y reemplazarlo por otro array con la misma capacidad que el de l_1 .
- 2) Copiar el atributo `capacity`.
- 3) Copiar los elementos del array `elems` de l_1 a l_2 .
- 4) Copiar el atributo `num_elems`.

Listas mediante arrays

`l1`



`l2`



Al hacer la asignación `l2 = l1`.

- 1) Si los elementos de `l1` caben en `l2`:
 - 1) Copiarlos de `l1` a `l2`.
 - 2) Copiar el atributo `num_elems`.
- 2) En caso contrario:
 - 1) Desechar `l2.elems` y reemplazarlo por otro array con la misma capacidad que el de `l2`.
 - 2) Copiar el atributo `capacity`.
 - 3) Copiar los elementos del array `elems` de `l1` a `l2`.
 - 4) Copiar el atributo `num_elems`.

Listas mediante arrays

```
ListArray & operator=(const ListArray &other) {
```

```
    if (this != &other) {  
        if (capacity < other.num_elems) {  
            delete[] elems;  
            elems = new std::string[other.capacity];  
            capacity = other.capacity;  
        }  
        num_elems = other.num_elems;  
        for (int i = 0; i < num_elems; i++) {  
            elems[i] = other.elems[i];  
        }  
    }  
    return *this;  
}
```

si no
caben.

Al hacer la asignación `l2 = l1`.

1) Si los elementos de `l1` caben en `l2`:

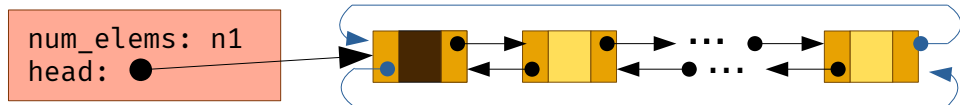
- 1) Copiarlos de `l1` a `l2`.
- 2) Copiar el atributo `num_elems`.

2) En caso contrario:

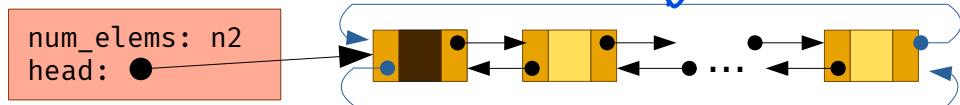
- 1) Desechar `l2.elems` y reemplazarlo por otro array con la misma capacidad que el de `l2`.
- 2) Copiar el atributo `capacity`.
- 3) Copiar los elementos del array `elems` de `l1` a `l2`.
- 4) Copiar el atributo `num_elems`.

Listas doblemente enlazadas circulares

$l1$



$l2$

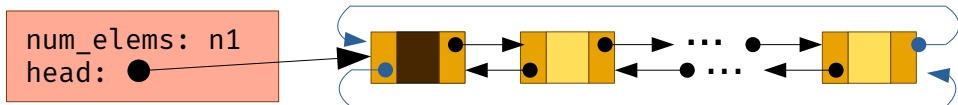


Al hacer la asignación $l2 = l1$.

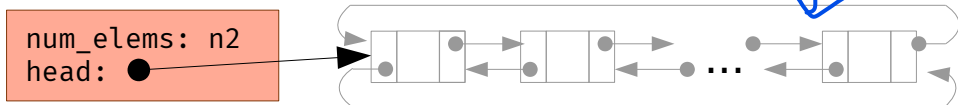
- 1) Borrar la cadena de nodos de $l2$.
- 2) Crear nodo fantasma en $l2$ y hacer que $l2.head$ apunte a él.
- 3) Hacer copias de los nodos de $l1$ y encadenarlos en $l2$ (similar al constructor de copia).
- 4) Copiar atributo num_elems de $l1$ a $l2$.

Listas doblemente enlazadas circulares

$l1$



$l2$

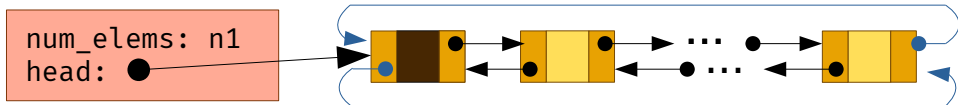


Al hacer la asignación $l2 = l1$.

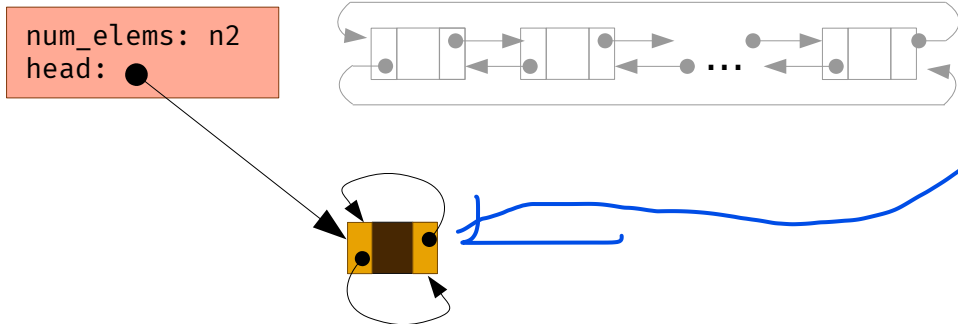
- 1) Borrar la cadena de nodos de $l2$.
- 2) Crear nodo fantasma en $l2$ y hacer que $l2.head$ apunte a él.
- 3) Hacer copias de los nodos de $l1$ y encadenarlos en $l2$ (similar al constructor de copia).
- 4) Copiar atributo `num_elems` de $l1$ a $l2$.

Listas doblemente enlazadas circulares

$l1$



$l2$

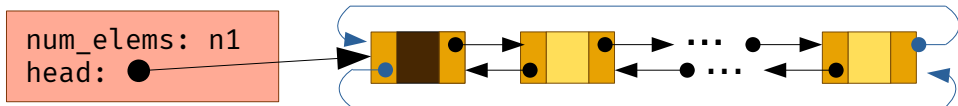


Al hacer la asignación $l2 = l1$.

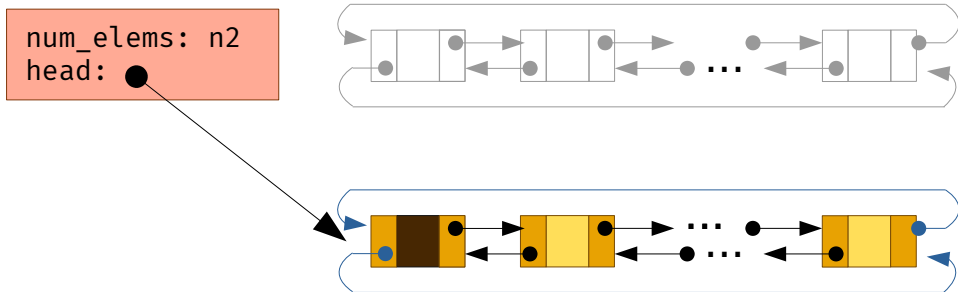
- 1) Borrar la cadena de nodos de $l2$.
- 2) Crear nodo fantasma en $l2$ y hacer que $l2.head$ apunte a él.
- 3) Hacer copias de los nodos de $l1$ y encadenarlos en $l2$ (similar al constructor de copia).
- 4) Copiar atributo `num_elems` de $l1$ a $l2$.

Listas doblemente enlazadas circulares

$l1$



$l2$

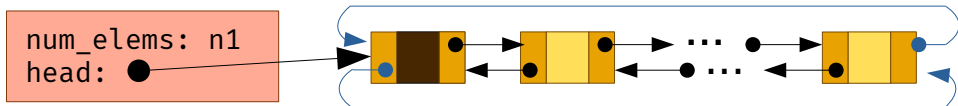


Al hacer la asignación $l2 = l1$.

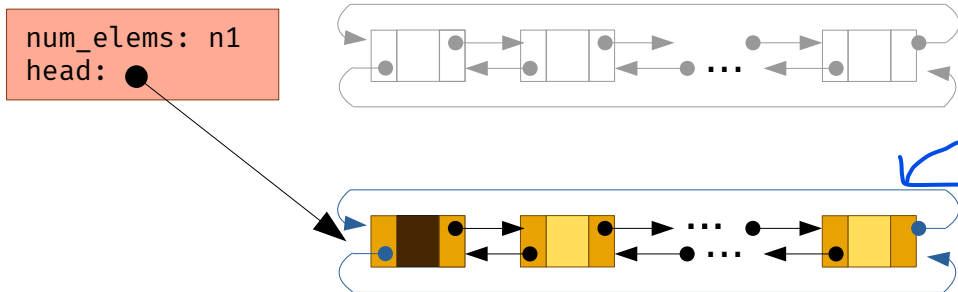
- 1) Borrar la cadena de nodos de $l2$.
- 2) Crear nodo fantasma en $l2$ y hacer que $l2.head$ apunte a él.
- 3) Hacer copias de los nodos de $l1$ y encadenarlos en $l2$ (similar al constructor de copia).
- 4) Copiar atributo `num_elems` de $l1$ a $l2$.

Listas doblemente enlazadas circulares

$l1$



$l2$



Al hacer la asignación $l2 = l1$.

- 1) Borrar la cadena de nodos de $l2$.
- 2) Crear nodo fantasma en $l2$ y hacer que $l2.head$ apunte a él.
- 3) Hacer copias de los nodos de $l1$ y encadenarlos en $l2$ (similar al constructor de copia).
- 4) Copiar atributo `num_elems` de $l1$ a $l2$.

Listas doblemente enlazadas circulares

```
ListLinkedDouble &
operator=(const ListLinkedDouble &other) {

    if (this != &other) {
        delete_nodes(); borrar la cadena de nodos.
        head = new Node;
        head->next = head->prev = head;
        copy_nodes_from(other);
        num_elems = other.num_elems;
    }
    return *this;
}
```

Al hacer la asignación $l2 = l1$.

- 1) Borrar la cadena de nodos de $l2$.
- 2) Crear nodo fantasma en $l2$ y hacer que $l2.head$ apunte a él.
- 3) Hacer copias de los nodos de $l1$ y encadenarlos en $l2$ (similar al constructor de copia).
- 4) Copiar atributo `num_elems` de $l1$ a $l2$.