Clase Llista

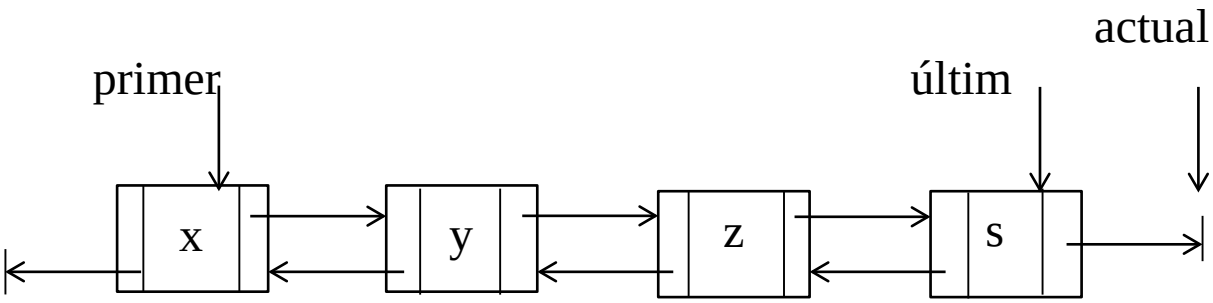
- nodo con doble enlace: anterior y siguiente
- solo un acceso a los elementos: **punto de interés**
- si queremos más de un punto de interés hay que usar herencia desde la clase **`iterator`**

Clase Llista

```
template <class T> class Llista{
private:
  struct node_llista {
    T info;
    node_llista* seguent; // enllaç doble
    node_llista* anterior;
  };
 int longitud;
 node_llista* primer_node;
 node_llista* ultim_node;
 node_llista* act;
```

Clase Llista

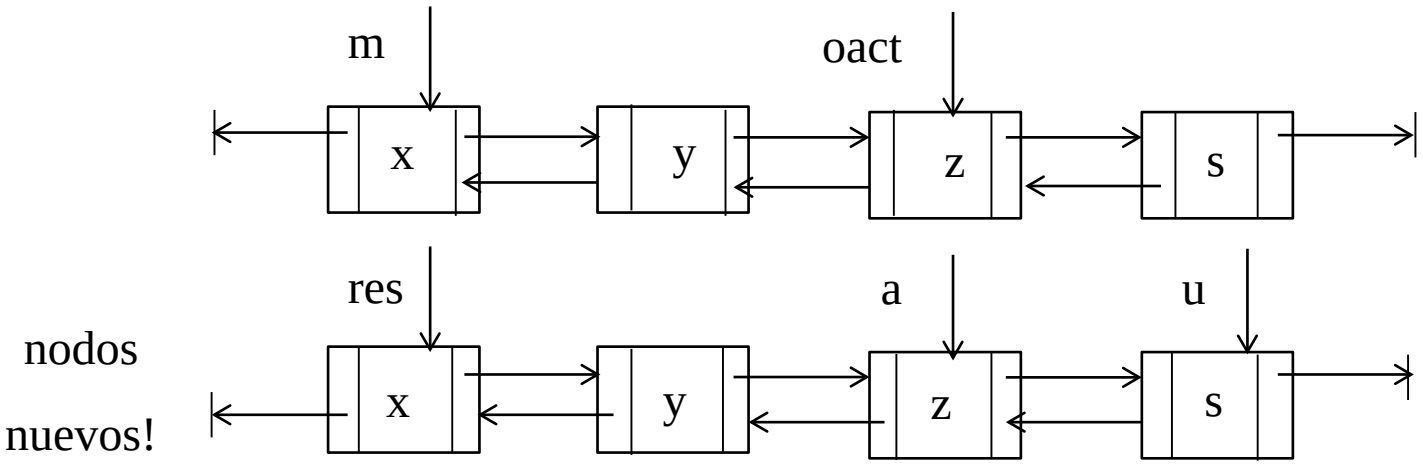
Convenio: el punto de interés puede estar después del último elemento, *pero no antes del primero*



longitud = 4

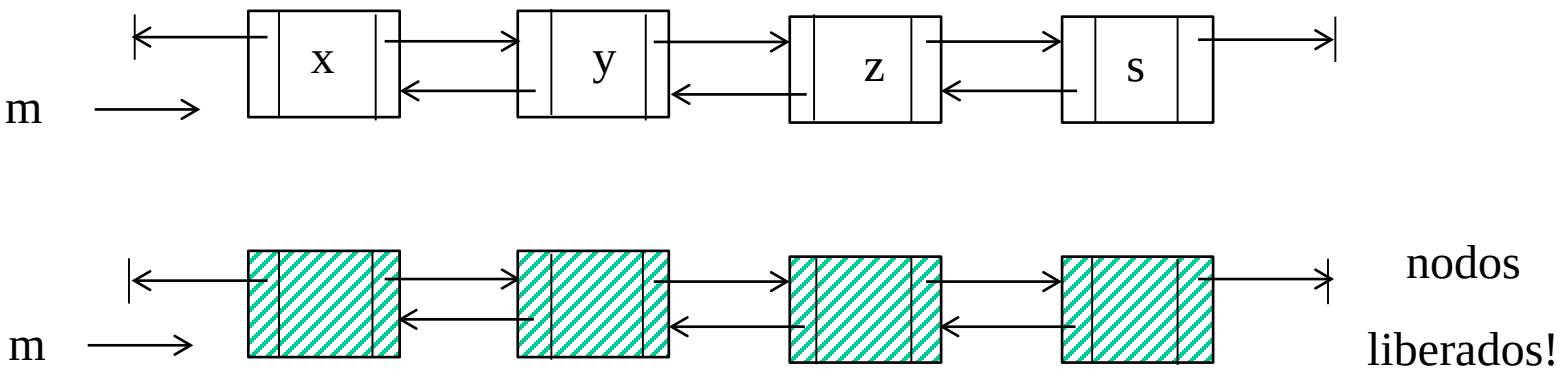
Clase Llista

```
static node_llista* copia_node_llista(node_llista* m,
node_llista* oact, node_llista* &u, node_llista* &a)
/* Pre: cert */
/* Post: si m és NULL, el resultat és NULL; en cas
contrari, el resultat apunta al primer node d'una cadena de
nodes que són còpia de la cadena que té el node apuntat per
m com a primer; u apunta a l'últim node; a és o bé NULL si
oact no apunta a cap node de la cadena que comença amb m o
bé apunta al node còpia del node apuntat per oact  */
```



Clase Llista

```
static void esborra_node_llista(node_llista* m)
/* Pre: cert */
/* Post: no fa res si m és NULL, en cas contrari, allibera
espai dels nodes de la cadena que té el node apuntat per m
com a primer */
```



Clase Llista

Ver el resto en Llista.hh

- **Constructora vacía y copiadora; destructora (se han de programar!)**
- **Redefinición (!) de la asignación**
- **Modificadoras: adaptadas a un único punto de interés**
- **Consultoras: idem**

Correspondencias respecto a `list`

`List()`	`Llista()`
`l.clear()`	`l.buida()`
`l.insert(it, x)`	`l.afegir(x) (*)`
`it1=l.erase(it2)`	`l.eliminar() (*)`
`l1.splice(it,l2)`	`l1.concat(l2) (**)`
`l.empty()`	`l.es_buida()`
`l.size()`	`l.mida()`

(*) solo respecto al punto de interés

(**) solo al final de la primera lista

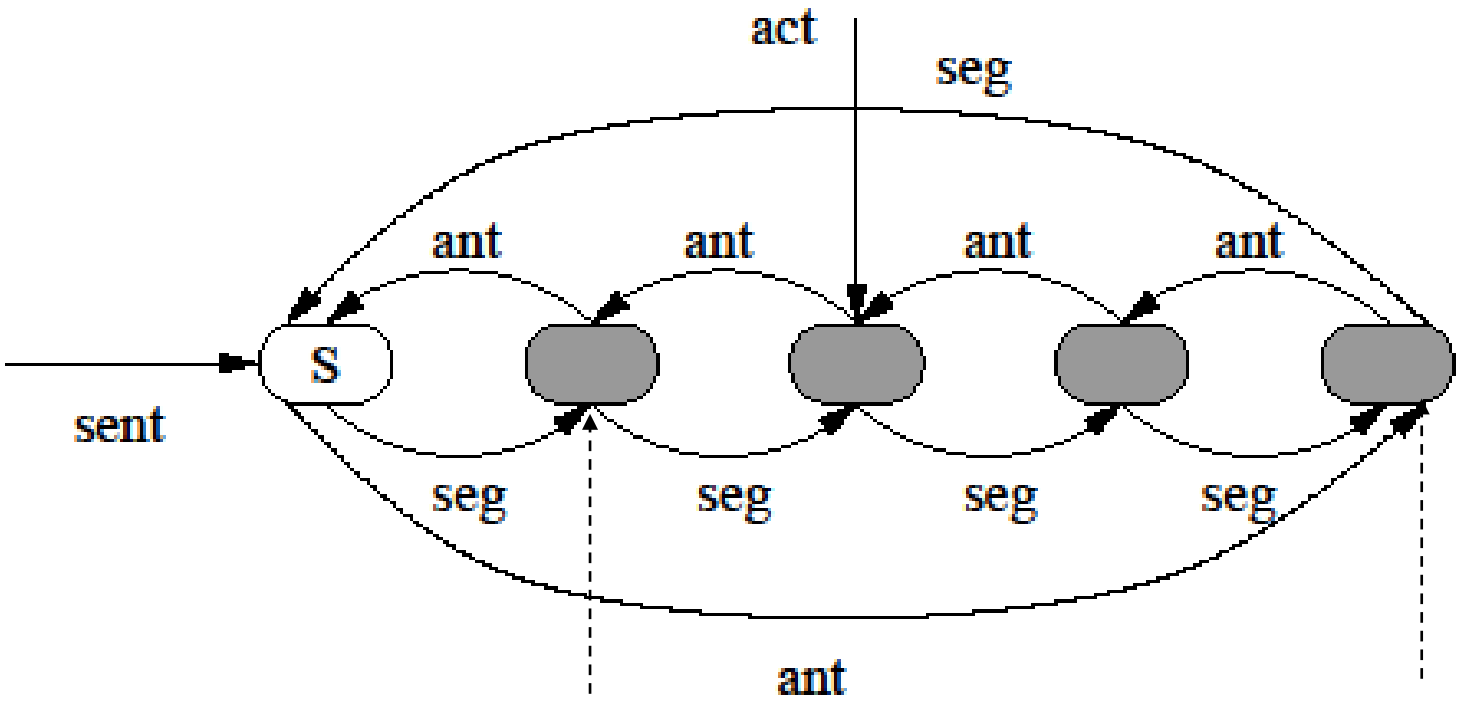
Correspondencias respecto a `list`

`List()`	`Llista()`
`x=*it (consulta)`	`x=l.actual()`
`*it=x (modificación)`	`l.modificar_actual(x)`
`it=l.begin()`	`l.inici()`
`it=l.end()`	`l.fi()`
`++it`	`l.avanca()`
`--it`	`l.retrocedeix()`
`it==l.begin()`	`l.sobre_el_primer()`
`it==l.end()`	`l.dreta_de_tot()`

Implementación de listas con centinela

- toda estructura tiene al menos un nodo, el centinela, que no se puede consultar, borrar ni modificar y no cuenta para la longitud
- el centinela es el anterior al primer elemento de la lista y el siguiente del último: ya no hacen falta `primer` y `ultim`
- `NULL` no es siguiente ni anterior de ningún elemento: afegir y eliminar se reducen de 4 casos a 1; `concat` se complica un poco

Implementación de listas con centinela



Implementación de listas con centinela

Convenio:

- **act puede ser sent, aunque la lista no sea vacía (indica que estamos al final de la lista)**
- **la especificación de la clase implica que**
  - **no se puede avanzar desde sent; sí que podemos usarlo para saber si estamos al final**
  - **no se puede retroceder desde el siguiente de sent; sí que podemos usarlo para saber si estamos al principio**
  - **no se puede consultar ni modificar el nodo apuntado por sent**
  - **no se puede borrar el nodo apuntado por sent (salvo cuando interviene la destructora)**