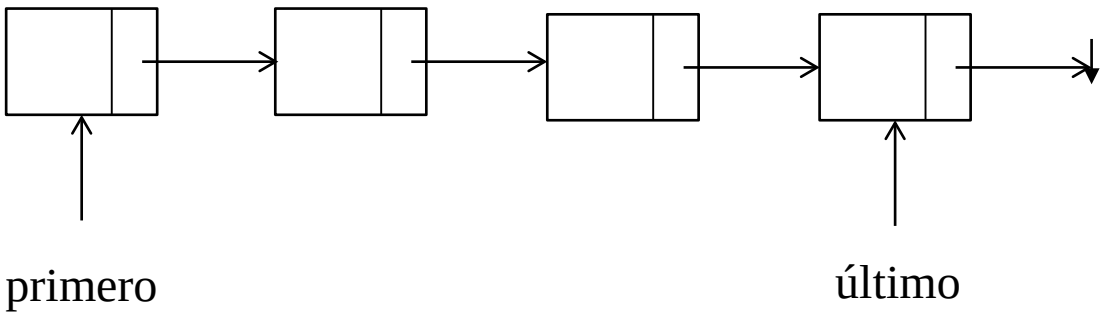
Estructuras enlazadas

Son implementaciones de contenedores tales que cada elemento contenido lleva asociada información sobre su siguiente, anterior, …



Punteros

Tipo de datos básico que representa una dirección de memoria, asociada a un tipo concreto.

```
T* p; // p es un puntero a objetos tipo T
```

Ejemplo: si los objetos de tipo `T` ocupan n posiciones, un `array` puede representarse con un puntero p a `T`:

```
v[0] = objeto que comienza en la dirección p
v[1] = objeto “    “       “  “ dirección p+n
v[k] = objeto “    “       “  “ dirección p+kn
```

Operaciones de punteros

- **objeto apuntado por p: *p**
- **dirección del objeto x: &x**
- **si el objeto es un struct**

```
struct T1{
  int camp1;
  bool camp2;
};
```

podemos acceder a los campos de *p:

(*p).camp1 equivale a p->camp1,

(*p).camp2 equivale a p->camp2

Operaciones de punteros

- **igualdad, desigualdad: p==q, p!=q**
- **asignación: p = q (son direcciones!)**
- **valor nulo (NULL, nullptr): no apunta a nada**

OJO! si p es nulo, *p da error

http://www.cplusplus.com/doc/tutorial/pointers/

Operaciones de punteros

Si un puntero p está recién declarado, solo se puede hacer lo siguiente:

- **asignarle uno ya existente: p = q**
- **asignarle el valor NULL: p = NULL**
- **reservar espacio nuevo y hacer que apunte a él: p = new T1;**

ahora se pueden dar valores a los campos: p->camp1=20; p->camp2=true;

Operaciones de punteros

Aliasing:

p=q; p apunta al mismo objeto que q; p->camp1=x modifica *q y viceversa

Paso por valor:

```
void f(T1* p)
```

Si se le asigna algo a p, no es permanente, pero si es a *p, sí.

Operaciones de punteros

Consecuencia: se ha de distinguir entre si dos objetos son iguales o son el mismo objeto

- **Iguales: diferente dirección, mismo contenido**
- **Mismo objeto: misma dirección**

Operaciones de punteros

Liberación de espacio:

Si no nos hace falta el objeto apuntado por p, debemos liberar su espacio con

```
delete p;
```

Típico de las operaciones de borrado en contenedores: pop, erase, fills

Plantilla para una estructura enlazada

```
template <class T> class estructura_rec {
private:
  struct node {
    tipus_info T;
    node* següent; // <-- recursivitat
  };
  tipus_que_sigui info_general;
  node* element_distingit_1; // tants como calgui
  ...
public:
  ...
};
```

Plantilla para una estructura enlazada

- **Todo en un .hh (sin .cc); template**
- **Las operaciones se implementan directamente en el .hh**
- **Ejemplos**
  - **Pila (versión de stack)**
  - **Cua (versión de queue)**
  - **Llista (versión de list, sin iterators)**
  - **Arbol binario y otras versiones de arboles**

Plantilla para una estructura enlazada

Restricciones (forman parte implícita de toda precondición y toda postcondición):

- **Cada objeto contenido en la estructura tendrá su propio nodo; un mismo nodo no puede representar a dos o más objetos**
- **Dos estructuras distintas no pueden compartir nodos**

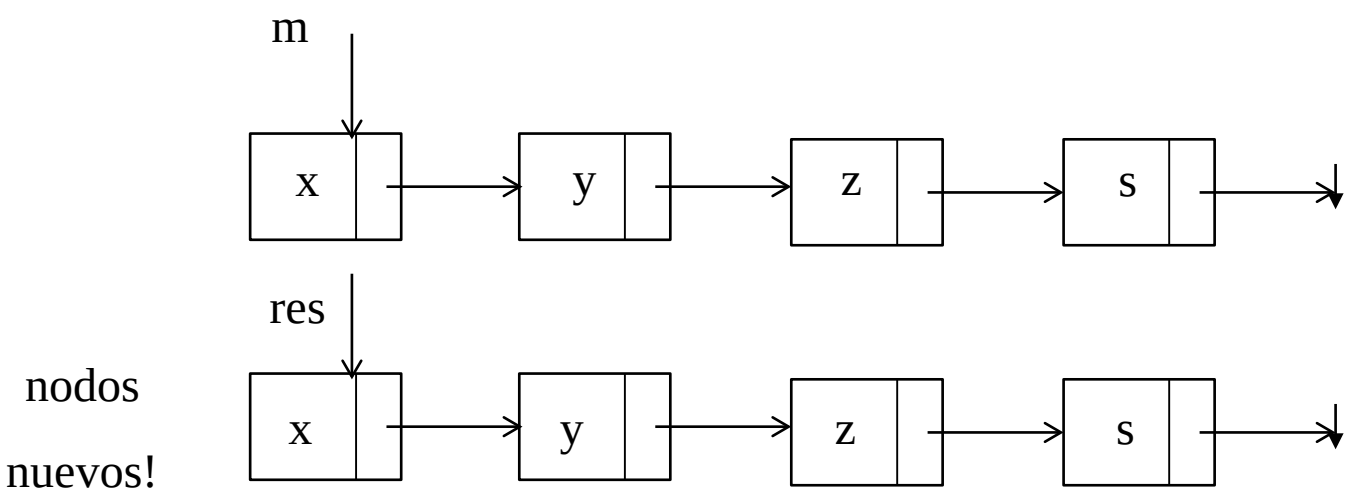
Clase Pila

```
template <class T> class Pila {
private:
  struct node_pila {
    T info;
    node_pila* seguent; // enllac simple
  };

  int altura; // nombre d'elements
  node_pila* primer_node; // cim/top
```

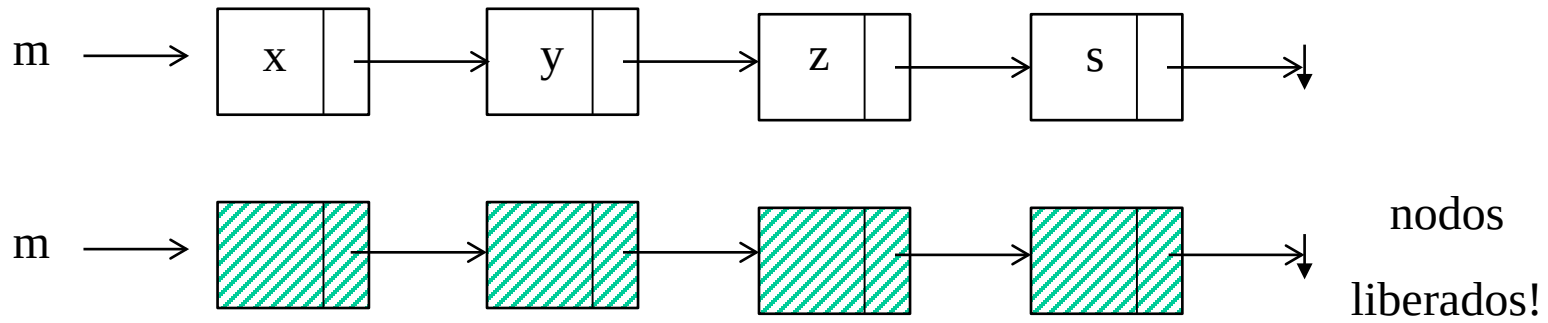
Clase Pila

```
static node_pila* copia_node_pila(node_pila* m) // privada
/* 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 de la cadena que té el node
apuntat per m com a primer */
```



Clase Pila

```
static void esborra_node_pila(node_pila* m) // privada
/* 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 Pila

Ver el resto en Pila.hh

- **Constructora vacía y copiadora; destructora (se ha de programar!)**
- **Redefinición (!) de la asignación**
- **Modificadoras: empilar (push), desempilar (pop), p_buida (clear)**
- **Consultoras: cim (top), mida (size), es_buida (empty)**

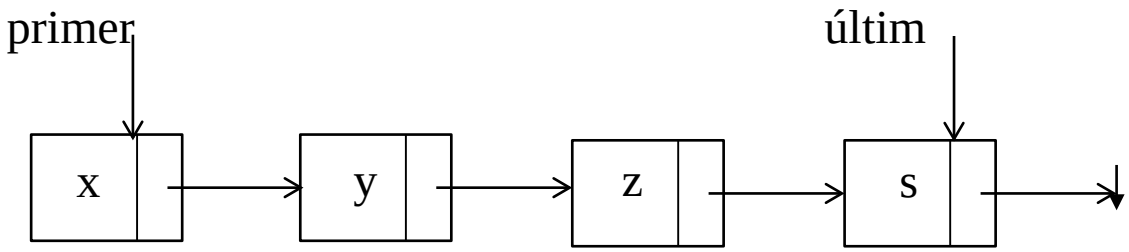
Clase Cua

```
template <class T> class Cua {
private:
  struct node_cua {
    T info;
    node_cua* seguent; // enllaç simple
  };

  int mida; // nombre d'elements
  node_cua* primer_node; // primer/front
  node_cua* ultim_node; // on s'afegeix
```

Clase Cua

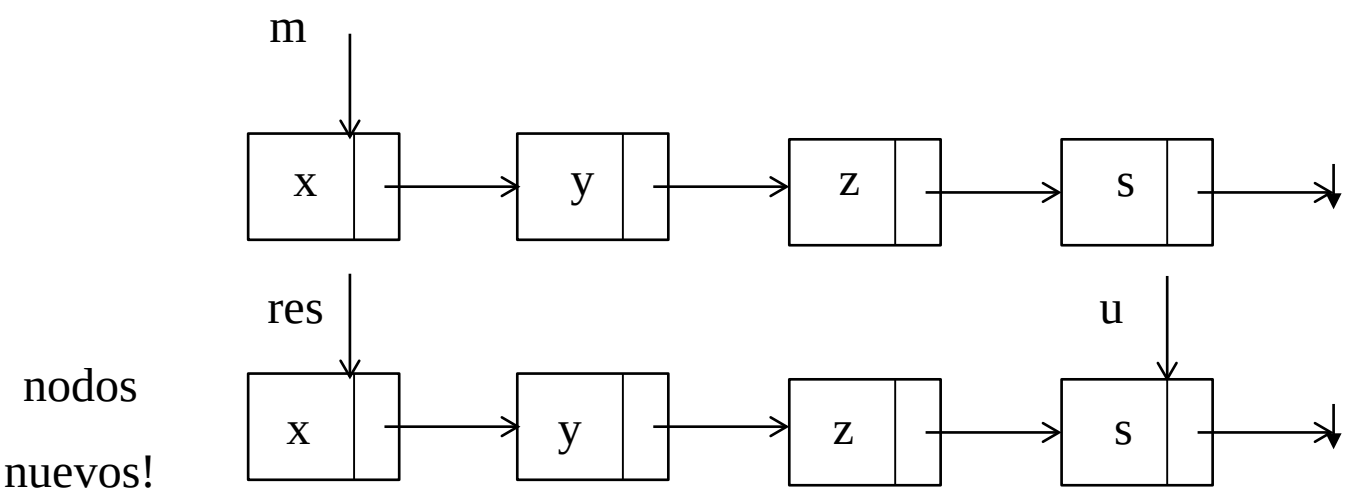
Convenio: el siguiente del último elemento es siempre NULL; usamos ultim_node para que el acceso sea en tiempo constante



mida = 4

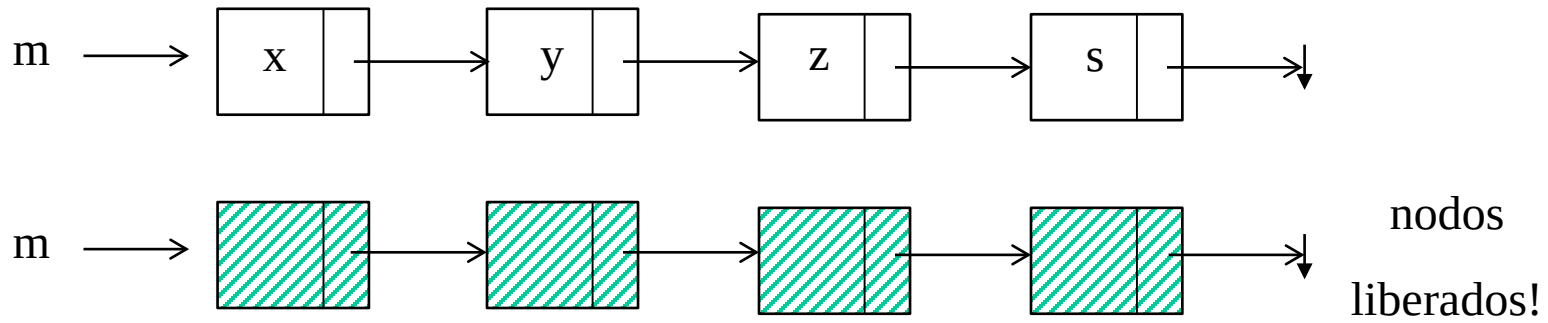
Clase Cua

```
static node_cua* copia_node_cua(node_cua* m, node_cua* &u)
/* 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 de la cadena que té el node
apuntat per m com a primer, i u apunta a l'últim node  */
```



Clase Cua

```
static void esborra_node_cua(node_cua* 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 Cua

Ver el resto en Cua.hh

- **Constructora vacía y copiadora; destructora (se ha de programar!)**
- **Redefinición (!) de la asignación**
- **Modificadoras: demanar_torn (push), avançar (pop), c_buida (clear)**
- **Consultoras: primer (front), mida (size), es_buida (empty)**