



Tema 3 Estructures Lineals

Sessió Teo 4

Maria Salamó Llorente

Estructura de Dades

Grau en Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona



Contingut

- 3.1 Introducció a les estructures lineals
- 3.2 TAD Pila i TAD Cua (representació estàtica)
- 3.3 Concepte de TAD
- 3.4 TAD Pila i TAD Cua (representació dinàmica)
- 3.5 TAD Llista
- 3.6 TAD Cua prioritària



Contingut

Sessió Teoria 3 (Teo 3)

3.1 Introducció a les estructures lineals

3.2 TAD Pila i TAD Cua (representació estàtica)

3.2 Concepta de TAD (introducció)

Sessió Teoria 4 (Teo 4)

3.3 Concepte de TAD (repàs i exemple)

3.4 TAD Pila i TAD Cua (representació dinàmica)

Sessió Teoria 5 (Teo 5)

3.5 TAD Llista

3.6 TAD Cua prioritària (es veurà al Tema 5)



3.3 Concepte de TAD



Introducció

TAD = Tipus Abstracte de Dades

- Un TAD permet usar un tipus de dades de forma complementament independent a la seva implementació
- **Tipus de dades**
 - Predefinit
 - Definit per l'usuari
 - Concret
 - Abstracte



Problemàtica dels tipus concrets

- Per exemple, definim un tipus **Polinomi** capaç de representar polinomis de grau N amb coeficients reals: $5x^3+4x^2+3x-6$
- Es proposa el tipus:

Coeficients = **taula** [0 .. N] **de real**

Polinomi = **tupla**

grau: enter

coef: Coeficients

fitupla

- L'ús dels programadors serà:
 - `pol.grau`
 - `pol.coef[i]`
 - `pol.coef[pol.grau]`



Problemàtica dels tipus concrets

- A la meitat del projecte, el cap del projecte se n'adona que:
 - L'aplicació haurà de treballar amb milions de polinomis (cal una representació més eficient de l'espai)
 - La majoria de polinomis seran de grau superior a 100, si bé la majoria dels coeficients seran zero
 - Després d'examinar el tipus Polinomi, decideix canviar la seva representació per emmagatzemar només els coeficients que són diferents de zero.

Parella= **tupla**

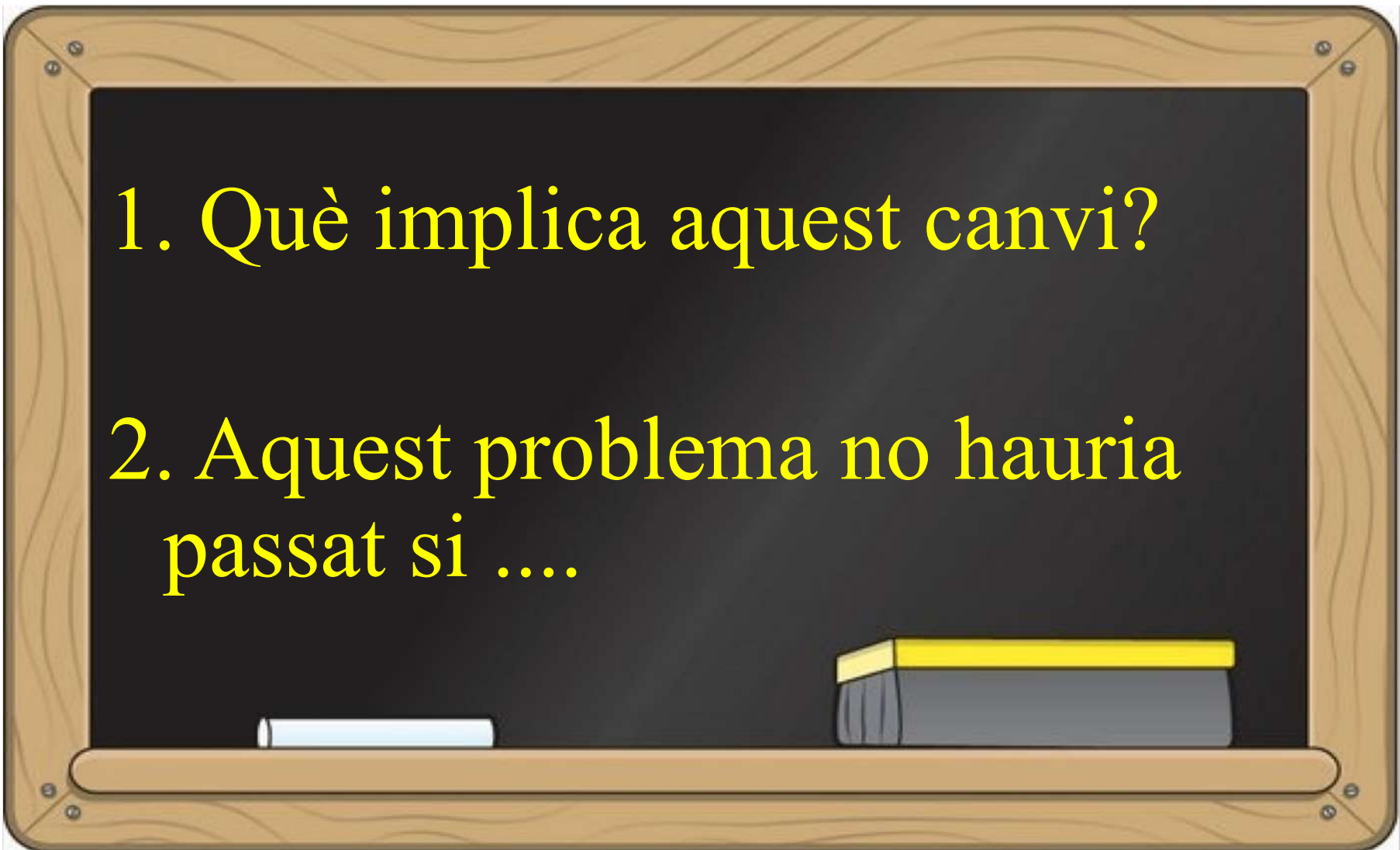
coef: real

exponent: enter

fitupla

Polinomi = **taula** [1 ..M] **de** Parella

Problemàtica (cont.)

- 
1. Què implica aquest canvi?
 2. Aquest problema no hauria passat si



Problemàtica (cont.)

- Observeu que aquests canvis impliquen:
 - Que tot el codi que utilitzava l'antic tipus Polinomi ja no és vàlid. Per exemple, el coeficient de grau superior ja no s'obté com: `pol.coef[pol.grau]`
 - Que TOTS els programadors han de revisar TOTES les línies del seu codi per adaptar-lo a la nova representació!
- Aquest problema no hauria passat si:
 - Hagués proporcionat un conjunt complet d'operacions per manipular polinomis
 - Hagués ocultat la representació del tipus Polinomi per tal que els programadors només tinguessin accés a les operacions públiques
 - **És a dir, Polinomi fos un tipus abstracte!**

Solució

- **Definir inicialment:**
 - L'**especificació** de les operacions
 - La **representació** del tipus
 - La **implementació** de les operacions
- Però només es fa públic l'especificació de les operacions
- Els programadors no necessiten conèixer com està representat, doncs fan servir les operacions del tipus



Solució (cont.)

Quines operacions afegirieu al tipus Polinomi?





SOLUCIO Exemples d'operacions

Operacions:

funció grau(Polinomi) retorna enter

funció coef (Polinomi, enter) retorna real

acció escriurePolinomi(Polinomi)

funció avaluaPolinomi (Polinomi) retorna real

Ús:

```
i = grau(pol)
```

```
res = avaluaPolinomi(pol)
```

```
i = coef (pol, j)
```

```
escriurePolinomi(pol)
```



Solució

- En aquest segon cas, direm que Polinomi és un **tipus abstracte de dades**, perquè la resta d'usuaris el poden usar només a partir de l'especificació de les operacions, sense conèixer els detalls de la seva implementació
- Si en un futur canviem la representació d'un TAD, només haurem de canviar la implementació de les seves operacions; la resta de codi d'altres mòduls continuarà essent vàlida



Concepte de Tipus Abstracte de Dades

- Definirem TAD (**Tipus Abstracte de Dades**) com un codi que proporciona:
 - Un tipus de dades, i
 - Un conjunt d'operacions per a treballar sobre valors d'aquest tipus
- En un TAD, la paraula **abstracte** fa referència al fet de poder treballar amb el tipus amb independència de com està representat, gràcies al fet de disposar d'operacions que actuen sobre ell
- Definirem els TADs com classes:
 - Atributs són les dades
 - Mètodes són les operacions per treballar sobre les dades

Components i notació

Un TAD consta de tres parts:

- **Especificació:** conté la llista d'operacions que proporciona el TAD, amb les seves especificacions
 - Aquesta és la única part pública del TAD
- **Representació:** conté la definició de l'estructura de dades amb la qual es representa el tipus internament
 - Aquesta representació és privada, els usuaris del TAD no necessiten conèixer els detalls
- **Implementació:** conté la implementació de les operacions que manipulen el tipus

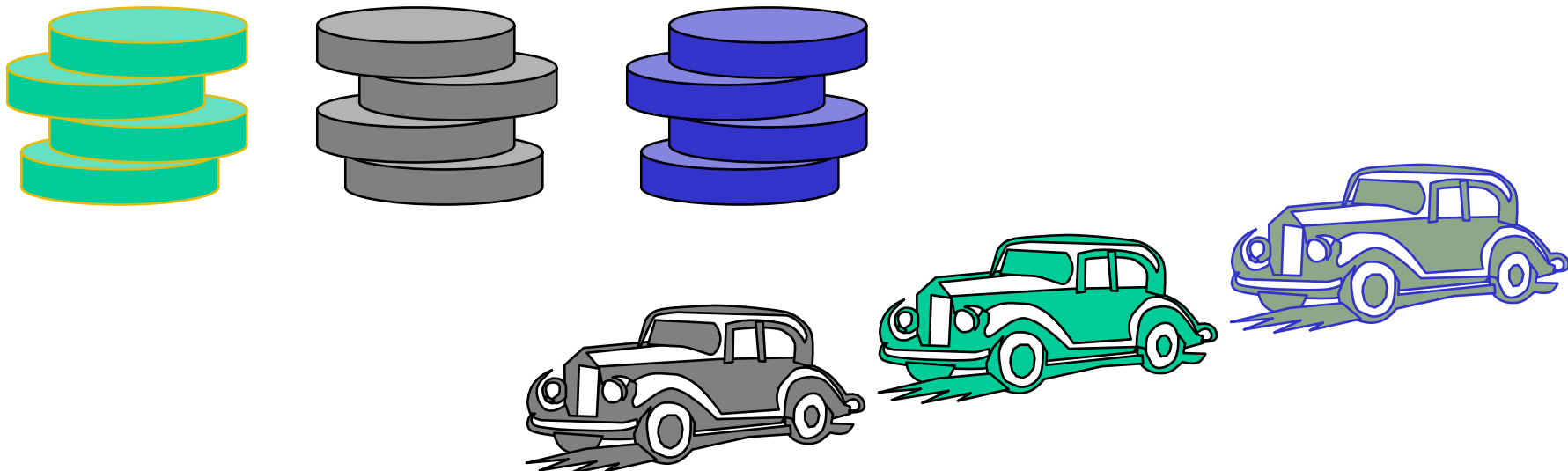
Exercicis

1. Trobeu l'especificació d'un TAD que permeti operar amb vectors de l'espai (ha de permetre fer productes escalars, vectorials, consulta del mòdul, normalització d'un vector, etc.)
2. Trobeu una representació pel TAD Vector3D i implementeu les operacions normalitza (donat un vector, el normalitza) i mòdul (retorna el mòdul d'un vector)

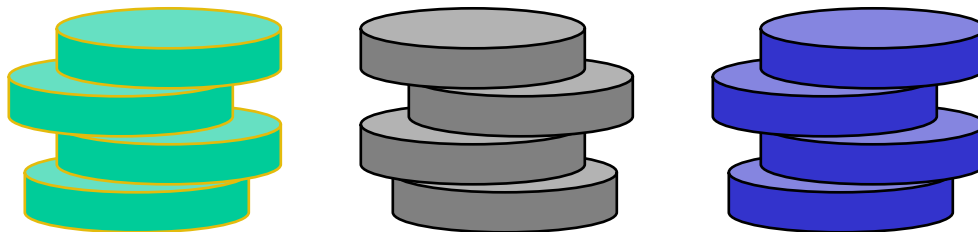
Exercicis

3. Dissenyeu un algorisme que esbrini si dos vectors de l'espai són paral·lels, usant les operacions bàsiques del TAD Vector3D
4. Trobeu l'especificació d'un TAD que proporcioni operacions sobre matrius $N \times N$ (per exemple, per una aplicació per resoldre sistemes d'equacions lineals)
5. Implementeu una operació del TAD matriu que esbrini si és la identitat

3.4 TAD Pila i Cua (implementació dinàmica)



TAD Pila



Especificació de la Pila en C++

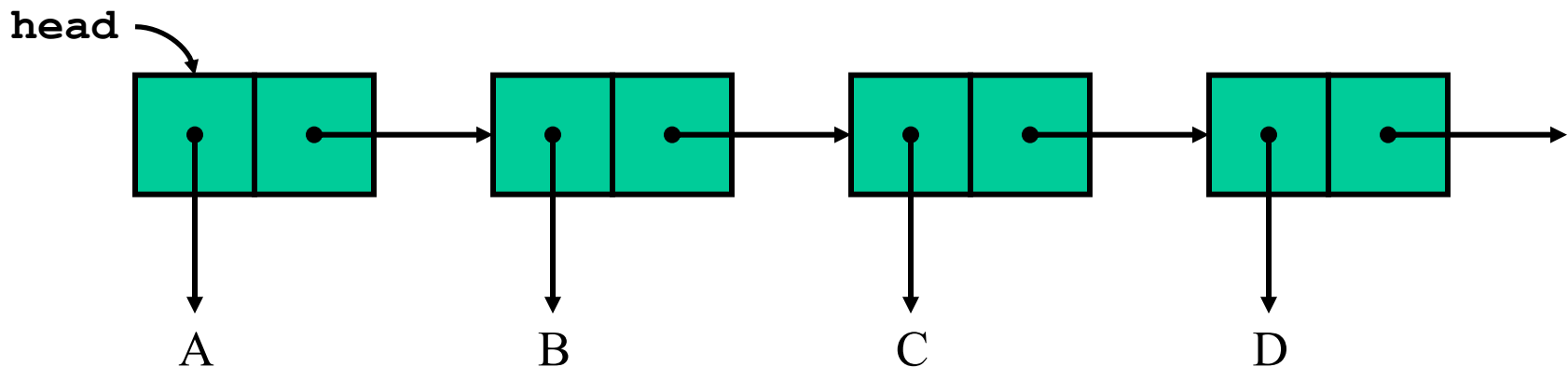
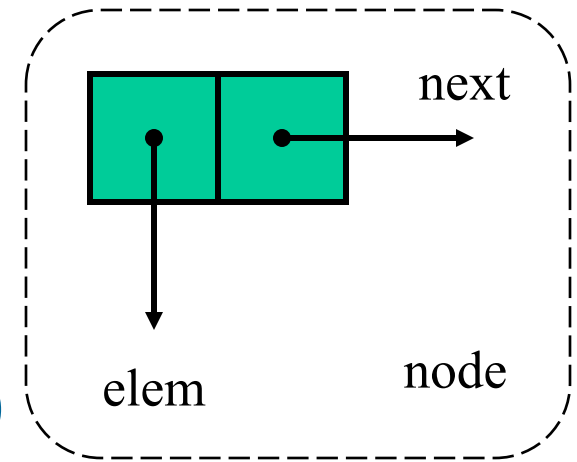
```
template <class E>
class Stack {
public:
    int size() const;
    bool empty() const;
    const E& top() const;
    void push(const E& e);
    void pop();
};
```

- ❑ Possibles representacions i implementacions:
 - ❑ Array
 - ❑ Estructura encadenada (apuntador encadenat)
- ❑ Diferent de la pila en C++ STL class stack

Implementacions bàsiques

- **Estructures encadenades**

- Estructures que no necessiten conèixer el màxim a priori
- Colecció de nodes encadenats
 - Entrada per head
- No té accés directe
- Inserir i Esborrar té un cost $O(n)$

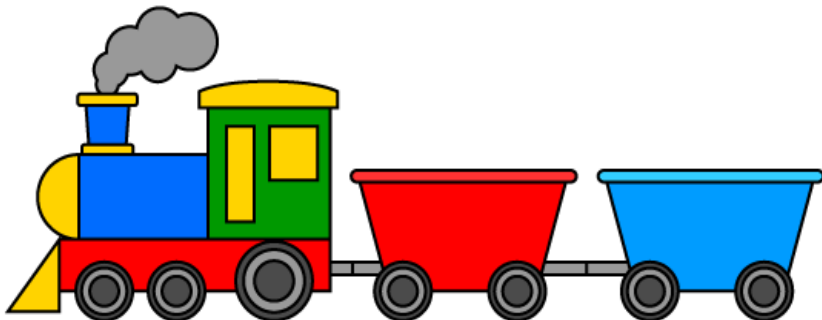




Estructura encadenada

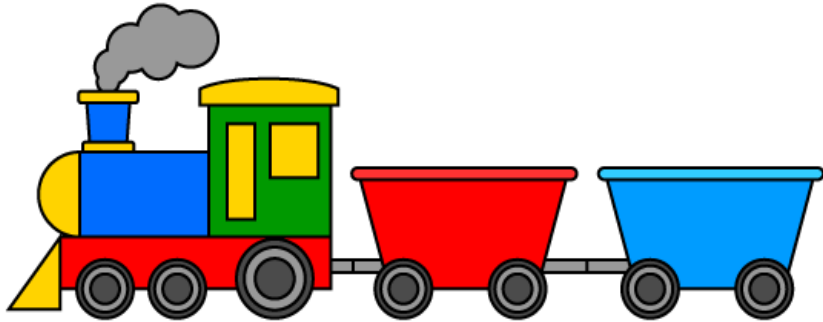


©DESIGNALIKIE



©DESIGNALIKIE

Estructura encadenada

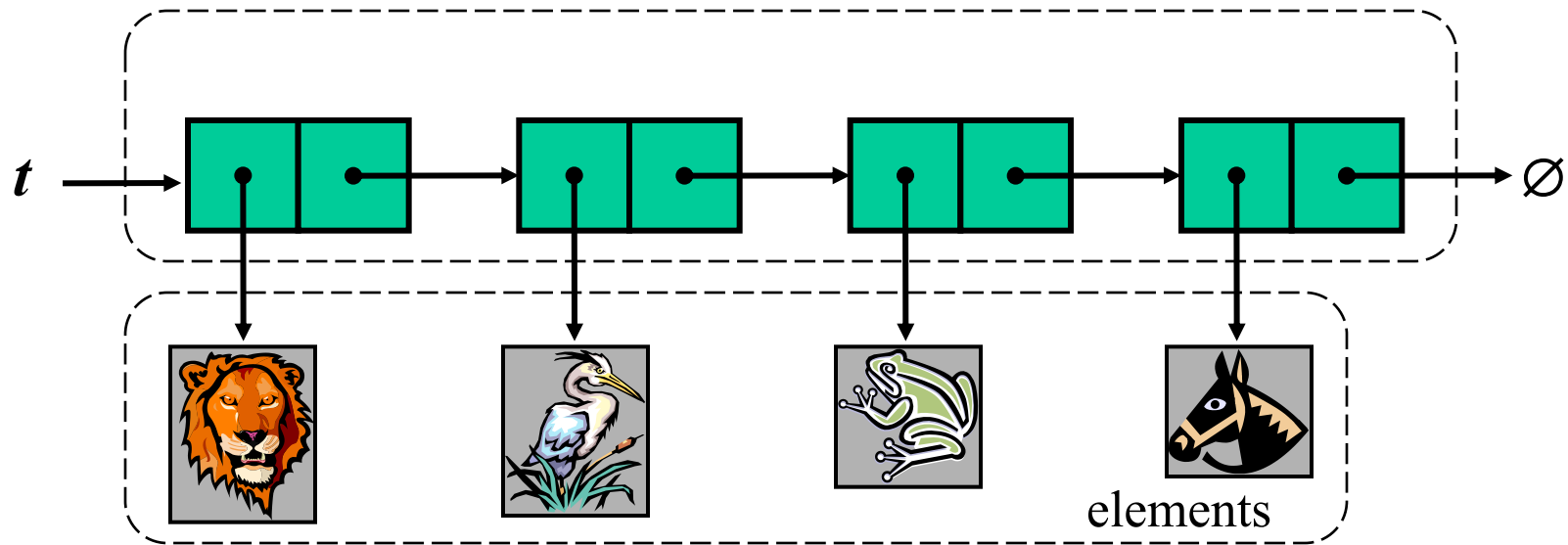


©DESIGNALIKIE



Pila en estructura encadenada

- Es pot implementar una pila amb una estructura encadenada amb apuntadors simples
- L'element del top es guarda al primer node de l'estructura encadenada
- L'espai usat és $O(n)$ i cada operació de la pila té un cost $O(1)$ en cost computacional





Pila implementada en estructura encadenada

```
template <class E>
class Node{
public:
    Node(E e);
    const E& getElement() const;
    Node<E> *getNext() const;
    void setNext(Node<E> *e);
    // ...
private:
    Node<E> *next;
    E element;
};
```

```
template <class E>
class LinkedStack{
public:
    LinkedStack();
    ~LinkedStack();
    int size() const;
    const E& top() const;
    void push(const E& e);
    void pop();
    // ...
private:
    Node<E> *front;
    int num_elements;
};
```



Pila implementada en estructura encadenada

Explicació transparència anterior

- La pila només necessita un punter al primer element de la seqüència
 - A aquest punter li direm **front** (mireu transparència anterior), també hi ha qui l'anomena top o head de la pila
 - El punter **front** apunta a un node de la seqüència d'elements que guarda la pila
 - Guardem el **num_elements** per saber quants nodes hi ha a la pila i no haver de fer un recorregut que faria que la funció size tingués un cost $O(n)$, amb num_elements tindrà un cost $O(1)$
 - Noteu que a la transparència anterior, les **operacions** de la pila són les mateixes que vam definir a la pila amb array
 - L'únic que hem canviat ara és la representació del TAD, és a dir, els seus atributs per guardar la seqüència.
 - Ara la seqüència està representada amb **Nodes**
 - Els nodes guarden un element (del tipus que sigui) i un punter (l'adreça) al següent element de la seqüència



Pila implementada en estructura encadenada (explicació continuació)

- Com que els atributs són privats a la Classe Node, es necessiten operacions per accedir i modificar els dos camps del node
 - **getElement()** permet retornar el contingut del node
 - **getNext()** ens permet retornar l'adreça on està el següent element de la seqüència
 - **setNext(Node *)** permet guardar l'adreça del node que serà el següent de la seqüència
- A l'hora d'implementar la pila encadenada hi ha dues opcions:
 1. Guardar al front el primer element que s'ha inserit i afegir pel final el nou element
 - Amb aquesta opció el mètode inserir i eliminar el darrer element de la pila tindran un cost $O(n)$
 2. Guardar al front el darrer element i al final de la seqüència tindrem el primer element
 - Amb aquesta opció el mètode inserir i eliminar tindran un cost $O(1)$
 - PER TANT, escollim aquesta opció per implementar la pila, ja que volem minimitzem el cost computacional

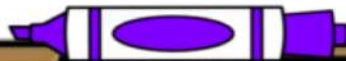


Pila en estructura encadenada

Exercicis

Tenint en compte l'especificació de les classes `LinkedStack` i `Node` anteriors, implementeu els mètodes `push` i `pop`.

Intenteu fer els exercicis sense mirar la solució, si us plau.



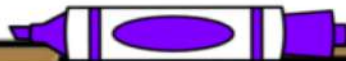
Pila en estructura encadenada

Exercicis

Recordeu: Feu els exercicis sense mirar la solució i després comproveu com ho heu fet. Hi ha més d'una solució, us proposo unes instruccions bàsiques per a que sigui més fàcil.

Per fer el push:

- Demaneu espai (a guardar en el heap) pel nou node
- Com que el volem posar a l'inici el nou node, aquest tindrà com a següent el front actual
- Com que ja estarà enllaçat el nou node amb el seu següent, només falta que el front tingui el nou element com a top de la pila
- No us oblideu que cal incrementar el nombre d'elements que hi ha a la pila





Solució push

```
template <class E>
void LinkedStack<E>::push(const E& ele)
{
    Node<E> *node = new Node<E>(ele);
    node->setNext(front);
    this->front = node;
    this->num_elements++;
}
```

Noteu que **NO s'ha d'alliberar el node** al final del mètode, sinó el node enllaçat s'elimina i la seqüència es queda mal enllaçada.

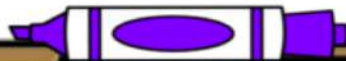
Pila en estructura encadenada

Exercicis

Recordau: Feu els exercicis sense mirar la solució i després comproveu com ho heu fet. Hi ha més d'una solució, us proposo unes instruccions bàsiques per a que sigui més fàcil.

Per fer el pop:

- Enllaceu en un node temporal l'element que voleu esborrar
- Com que el volem esborrar, primer hem de deixar els enllaços de la seqüència ben lligats. Bàsicament que el front quedi al següent element que ara serà el nou front.
- No us oblideu de decrementar el nombre d'elements que hi ha a la pila





Solució pop

```
template <class E>
void LinkedStack<E>::pop()
{ if (this->empty()) throw out_of_range("LinkedStack<>::pop(): empty stack");
  else
  { Node<Element> * aux_node = front;
    front = aux_node->getNext();
    delete aux_node;
  }
  this->num_elements--;
}
```

Noteu que **NO s'ha de demanar espai**, sino deixareu un espai a memòria



Links per llegir

- Aquest vídeo per entendre com és la pila amb enllaços.
 - Vigileu, la implementació és diferent a la que s'ha proposat en aquest curs. Nosaltres no usem structs sinó **classes**

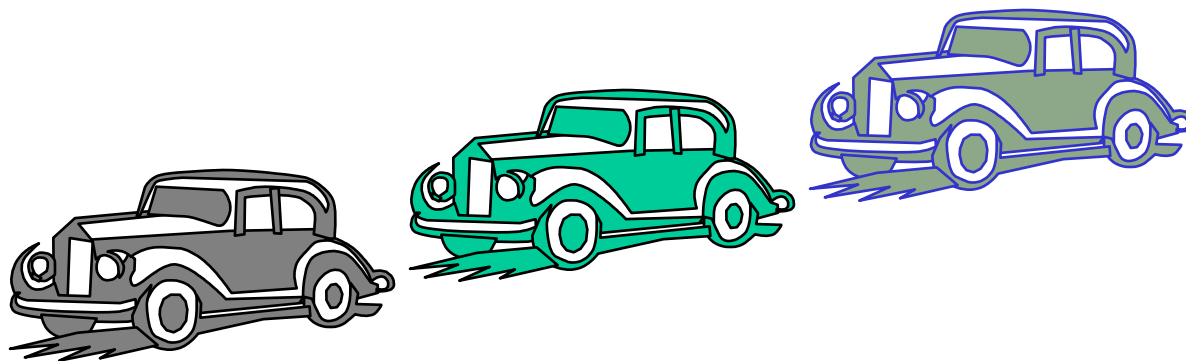
<https://www.youtube.com/watch?v=Anq11tezBSM>

- Per veure com funciona la pila amb array i enllaçada

<https://www.cs.usfca.edu/~galles/visualization/StackArray.html>

<https://www.cs.usfca.edu/~galles/visualization/StackLL.html>

TAD Cua



Especificació de la Cua en C++

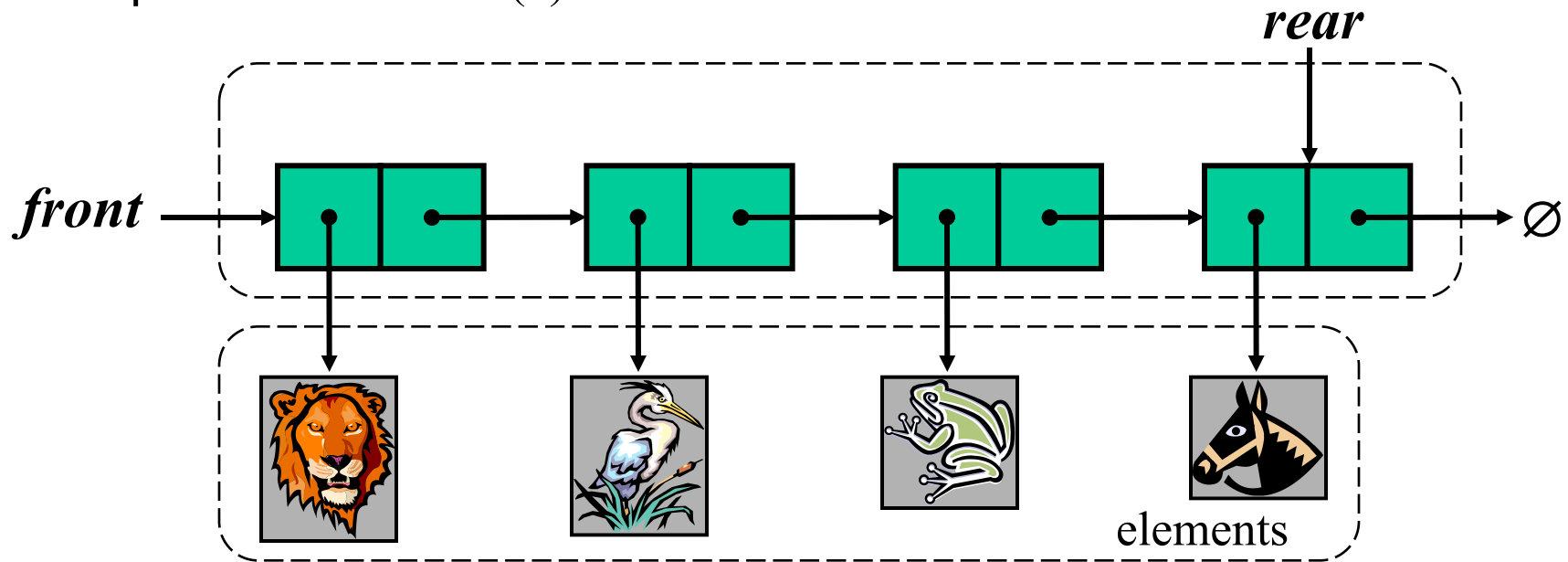
```
template <class E>
class Queue {
public:
    int size() const;
    bool empty() const;
    const E& front() const ;
    void enqueue (const E& e) ;
    void dequeue() ;
};
```

❑ Possibles implementacions:

- ❑ Array, ArrayCircular
- ❑ Estructura encadenada (apuntador encadenat simple o doble)

Cua en estructura encadenada

- La cua es guarda dos punters **front** i **rear**
 - El primer element de la seqüència es guarda en el primer node i és el node que apuntarà el **front**
 - El darrer element de la seqüència es guarda en el darrer node i és el node on apuntarà **rear**.
- L'espai usat és $O(n)$ i cada operació de la cua té un cost computacional de $O(1)$





Exemple de Cua en estructura encadenada

```
#ifndef LINKEDQUEUE_H
#define LINKEDQUEUE_H

#include "Node.h"

class LinkedQueue
{
public:
    LinkedQueue();
    virtual ~LinkedQueue();
    LinkedQueue(const LinkedQueue& q);
    void enqueue(const int key);
    void dequeue();
    bool isEmpty();
    void print();
    const int getFront();

private:
    int _size;
    Node* _front;
    Node* _rear;
};

#endif // LINKEDQUEUE_H
```

```
#ifndef NODE_H
#define NODE_H

class Node {
public:
    Node(const int e);
    virtual ~Node();
    const int getElement();
    Node* getNext();
    void setNext(Node& n);

private:
    int _element;
    Node* _next;
};

#endif /* NODE_H */
```

Implementació cua encadenada

Per implementar el TAD LinkedQueue hi ha vàries possibilitats:

1. Implementació amb enllaços simples sense sentinelles

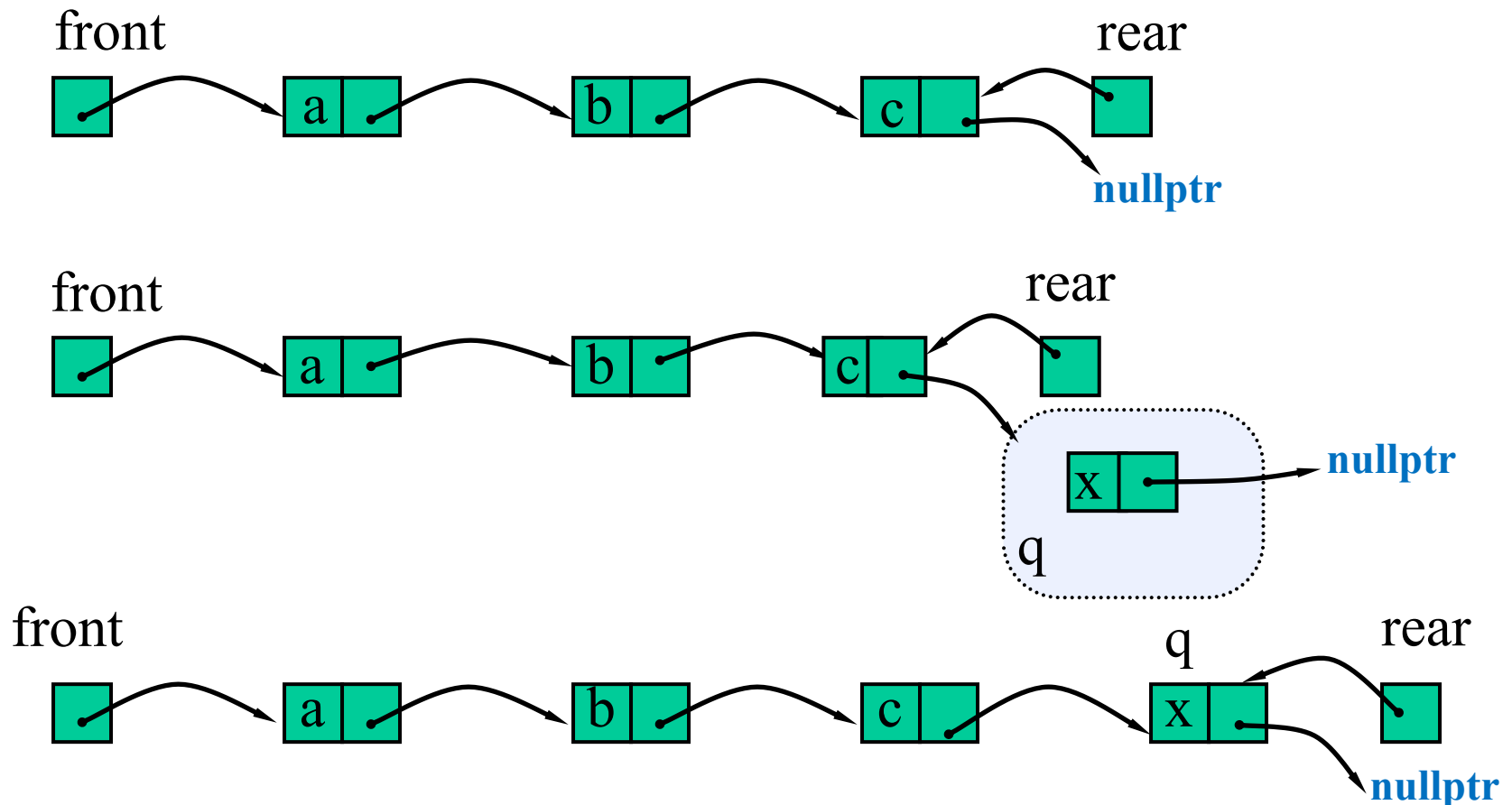
- S'han de tenir en compte diferents possibilitats:
 - S'ha de considerar que es crea la cua buida i, front i rear han d'estar a nullptr
 - Quan s'insereix el primer element o s'ha eliminat tots els elements i només queda un a la cua, front i rear han d'apuntar a aquest primer element
 - Quan hi ha més d'un element i es fa un enqueue, només rear ha de modificar l'adreça a la que està apuntant

2. Implementació amb enllaços simples amb sentinelles

- Per facilitar els passos de la implementació, es pot definir un sentinella a l'inici de la cua. D'aquesta manera el front sempre apunta al sentinella i el rear serà l'únic element que s'ha de moure a la cua
- Quan es crea la cua, front i rear apunten al sentinella que s'ha creat prèviament al constructor.
- Cua buida és quan front i rear estan els dos apuntant al sentinella.

Inserció al final de la Queue sense sentinella

- Visualització de l'operació enqueue(x), la qual inserta x al final de la cua.





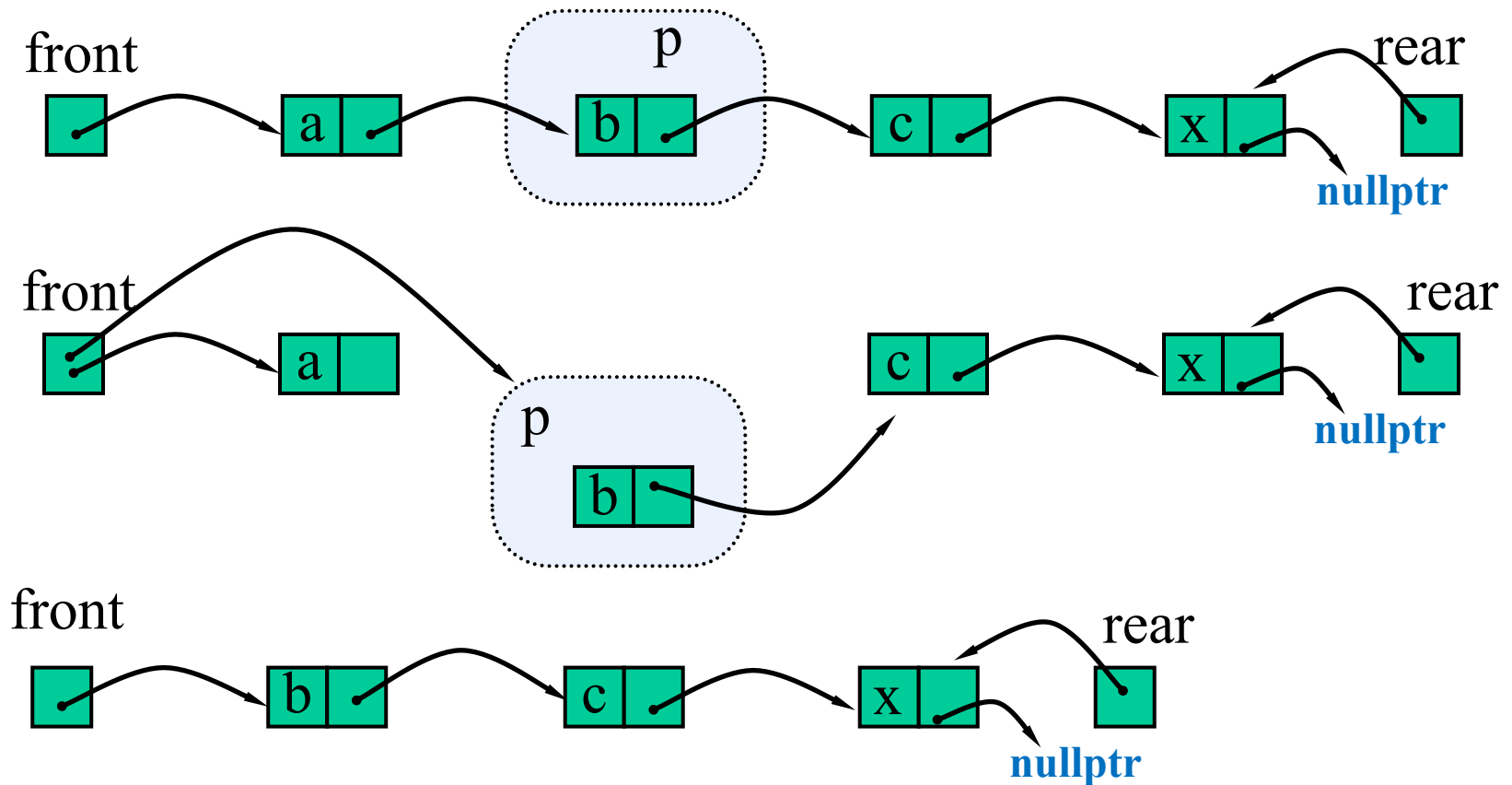
Algorisme d'enqueue sense sentinella

```
void LinkedList::enqueue(const int key) {  
    Node* node_q = new Node(key); // demana espai  
  
    if (isEmpty()) { // enllaça si buida  
        _front = node_q;  
        _rear = node_q;  
    } else { // enllaça si hi ha més elements  
        _rear->setNext(node_q);  
        _rear = node_q;  
    }  
    _size++;  
}
```

ATENCIÓ !!! Hi ha diverses maneres d'implementar aquest mètode, aquí només teniu una d'elles, seguint el dibuix anterior.

Eliminar a l'inici de la Queue sense sentinella

- Visualització de l'operació dequeue()





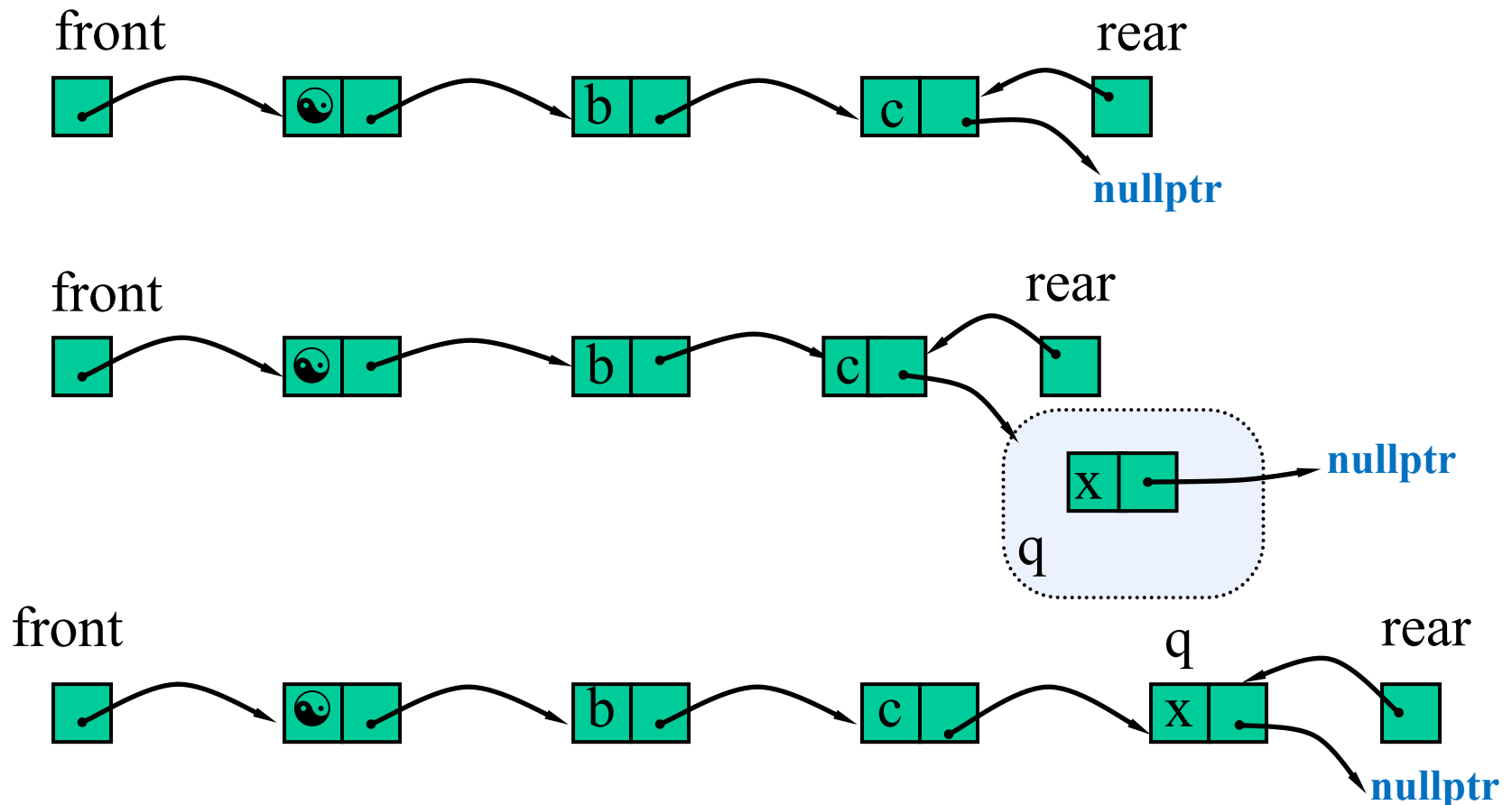
Algorisme dequeue sense sentinella

```
void LinkedList::dequeue() {  
    if (!isEmpty()) {  
        Node* node_p= _front->getNext();  
        delete _front;  
        _front = node_p;  
        _size--;  
    } else {  
        throw std::string("Queue is empty!");  
    }  
}
```

ATENCIÓ !!! Hi ha diverses maneres d'implementar aquest mètode, aquí només teniu una d'elles.

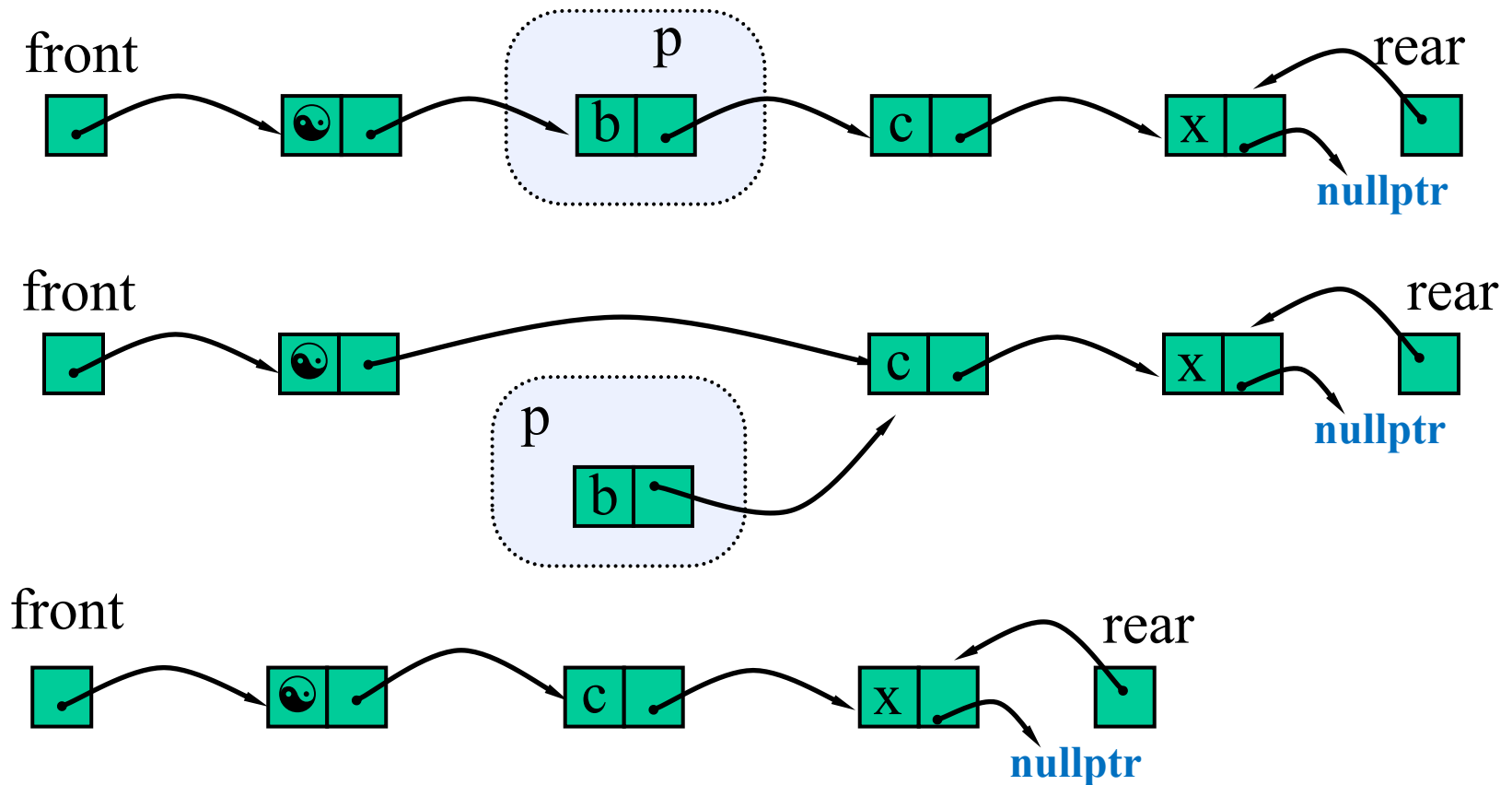
Inserció al final de la Queue amb sentinella

- Visualització de l'operació enqueue(x), la qual inserta x al final de la cua. (el sentinella és el node pintat com ☯)



Eliminar a l'inici de la Queue amb sentinella

- Visualització de l'operació dequeue()

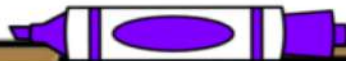




Cua en estructura encadenada

Exercicis

1. Implementar el mètode enqueue de la LinkedQueue amb sentinella
2. Implementar el mètode dequeue de la LinkedQueue amb sentinella





SOLUCIÓ Exemple d'ús

<i>Operation</i>	<i>Output</i>	<i>Q</i>
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
dequeue()	–	(3)
enqueue(7)	–	(3, 7)
dequeue()	–	(7)
front()	7	(7)
dequeue()	–	()
dequeue()	"error"	()
empty()	true	()
enqueue(9)	–	(9)
enqueue(7)	–	(9, 7)
size()	2	(9, 7)
enqueue(3)	–	(9, 7, 3)
enqueue(5)	–	(9, 7, 3, 5)
dequeue()	–	(7, 3, 5)



Tema 3 Estructures Lineals

Sessió Teo 4

Maria Salamó Llorente

Estructura de Dades

Grau en Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona