



Tema 3 **Estructures Lineals**

Maria Salamó Llorente
Estructura de Dades

Grau en Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona



Contingut

- 3.1 Introducció concepte de TAD
- 3.2 Introducció a les estructures lineals
- 3.3. TAD Cua
- 3.4. TAD Pila
- 3.5 TAD Llista
- 3.6. TAD Cua prioritària



3.1 Introducció 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
- L'ús de TADs permet desenvolupar aplicacions reduint:
 - Errors de programació
 - El nombre de programadors
 - El temps de desenvolupament
 - Els costos
- Les aplicacions que fan servir TADs són més robustes i més fàcils de mantenir



Revisió de tipus de dades

- Tipus de dades que coneixem fins ara:
 - Tipus de dades **predefinits** (són proporcionats pel llenguatge):
 - enter, real, caràcter, boolean
 - Tipus de dades **definits per l'usuari** (usant taules o tuples).



Tipus predefinits

- Un tipus de dades predefinit determina:
 - Quins valors pot representar (**domini**)
 - enters → 0, 1, -1, -2, ...
 - booleans → cert, fals
 - Quines **operacions** es poden aplicar als valors del tipus
 - Els enters admeten operacions aritmètiques, relacionals, de conversió, ...
 - Els booleans admeten operacions lògiques, relacionals, ...

Tipus definits per l'usuari

- Un tipus de dades definit per l'usuari determina únicament una cosa:
 - Quins elements el formen (és a dir, la seva estructura)
- A diferència dels tipus predefinits, no disposem d'un conjunt suficient d'operacions predefinides sobre aquests tipus.
 - La única operació predefinida per aquests tipus és l'accés a un element



Estructures de dades

- Les taules i les tuples permeten estructurar les dades amb que ha de treballar una aplicació
- Una **estructura de dades** és una combinació arbitrària de taules i tuples, amb la finalitat de representar les dades d'un problema

Tipus concrets i abstractes

- Un tipus de dades és **abstracte** quan proporciona un conjunt d'operacions prou complet com per poder-lo utilitzar sense saber com es representen internament els valors del tipus ni com estan implementades les seves operacions
- Tots els tipus predefinits (enter, real, caràcter, booleà) **són abstractes**. Observeu que:
 - Hi ha un conjunt d'operacions predefinides per a aquest tipus
 - No ens cal conèixer com es representen internament els seus valors (ex. coma fixa, coma flotant, ...) per a utilitzar-los
 - Tampoc ens cal saber com estan implementades internament les operacions (+, -, *, ...)

Tipus concrets i abstractes

- Un tipus de dades és **concret** quan només proporciona una representació (en forma de taula o tupla) pels valors del tipus. Per tant, per a utilitzar-lo caldrà fer servir codi que serà dependent de l'estructura de dades que representa el tipus.
- Tots els tipus definits per l'usuari són, inicialment, tipus concrets. Observeu que:
 - No hi ha cap operació predefinida per a aquests tipus
 - El codi que utilitza el tipus depèn de la seva estructura



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

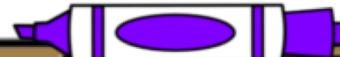
Polinomi = **taula** [1 .. M] **de** Parella

Problemàtica (cont.)

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

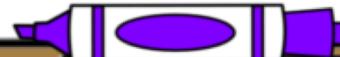


Problemàtica (cont.)





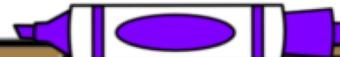
Solució





Solució (cont.)

Quines operacions afegirieu al tipus Polinomi?





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 TAD

- Definirem TAD (Tipus Abstracte de Dades) com un mòdul 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
- En programació modular, els mòduls de dades s'han de dissenyar com a TADs

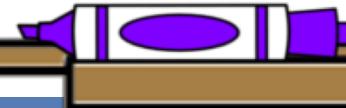
Components i notació

- 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. També és privada



Exercici

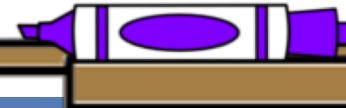
Especifiqueu el TAD conjunt d'enters (Cjt)





Exercici

Especifiqueu el TAD Punt3D

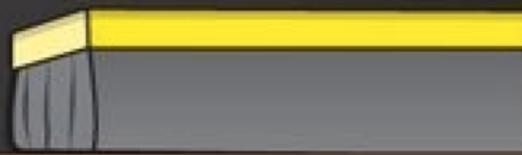


Exercicis

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 NxN (per exemple, per una aplicació per resoldre sistemes d'equacions lineals)
5. Implementeu una operació del TAD matriu que esbrini si és la identitat





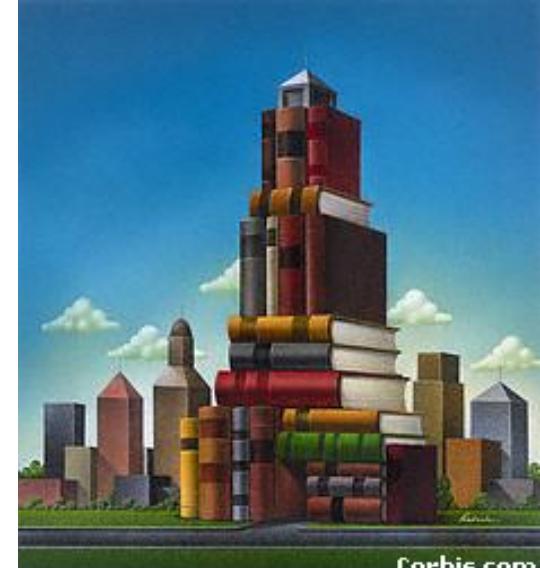
3.2 Introducció a les estructures lineals

Qué són les estructures lineals?

- Parlem d'estructures lineals quan les dades s'organitzen de forma seqüencial (un element abans i un element després).

$$a_{i-1} \rightarrow a_i \rightarrow a_{i+1}$$

- En aquest curs veurem:
 - Pila
 - Cua
 - Llista
 - Cua prioritària

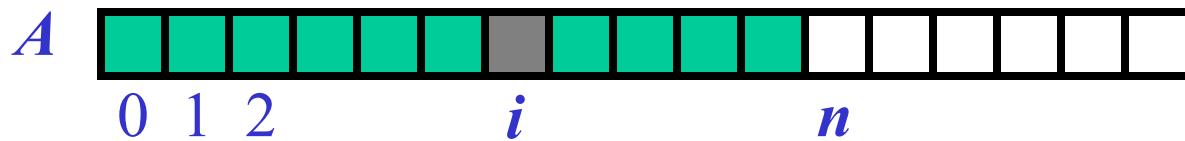


Corbis.com

Implementacions bàsiques

- **Arrays**

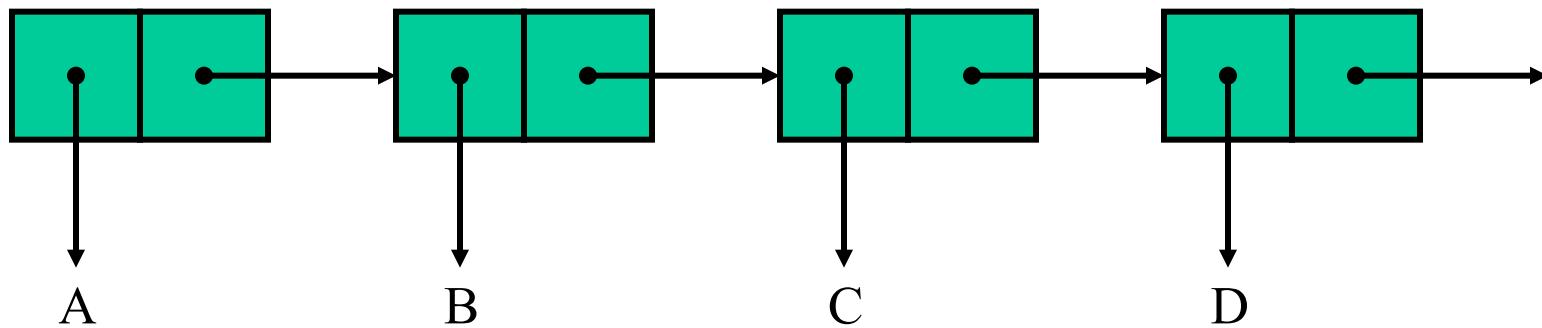
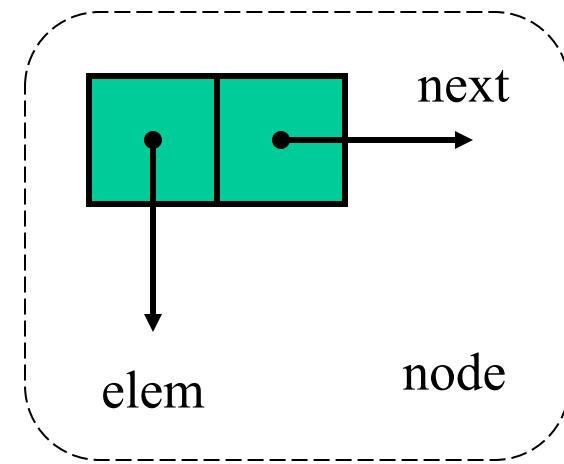
- Accés directe
- Inserir i Esborrar té un cost $O(n)$
- Errors quan s'omple del tot!!!



Implementacions bàsiques

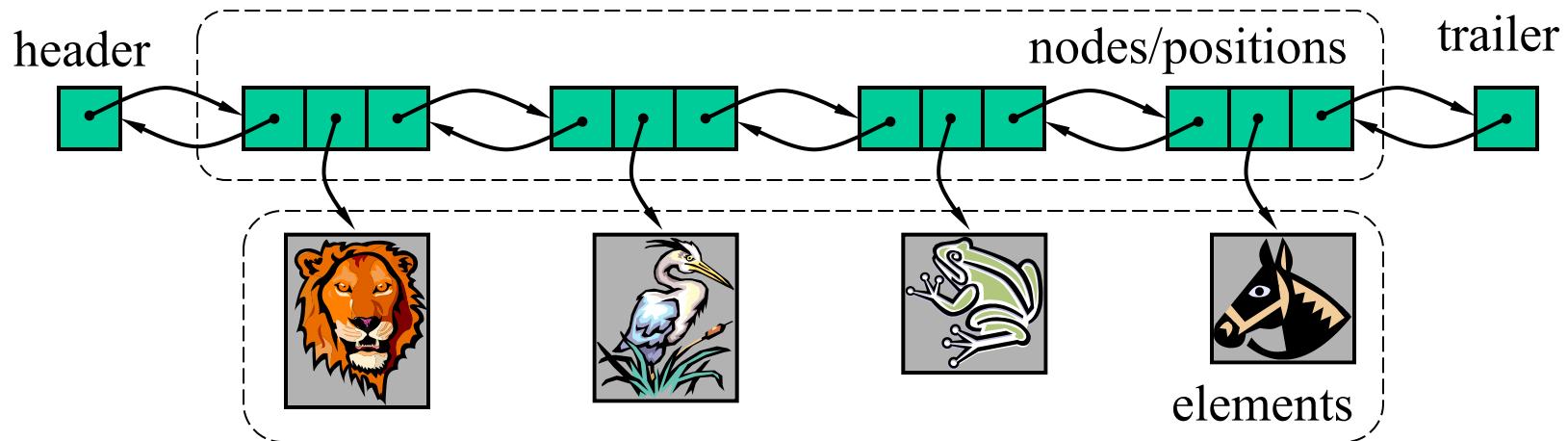
• Estructures encadenades

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



Implementacions bàsiques

- **Estructures doblement encadenades**
 - Estructura que no necessita conèixer el màxim a priori
 - Collecció de nodes encadenats
 - Entrada per head i per tail
 - No té accés directe
 - Inserir i Esborrar té un cost $O(n)$



TAD seqüència

- Les seqüències són una subfamília dels TADS que es correspon al concepte matemàtic de seqüència
 - Conjunt d'elements que segueixen una relació lineal
 - Tots els elements són del mateix tipus
- Especificació bàsica:
 - crea()
 - esBuida()
 - tamany()
 - consulta()
 - afegeix (E element)
 - treu()
 - esborra()
- Excepcions:
 - Treure un element de la seq. buida
 - Afegir elements a la seq. plena





Contenidor i Iterador

- Un contenidor és una estructura de dades que guarda una col·lecció d'elements i que suporta el seu accés a través d'iteradors
 - Contenidors de C++ són: vector, deque, list, stack, queue, priority_queue, set (i multiset) i map (i multimap)
- Un iterador fa una abstracció per recórrer tots els elements del contenidor
 - Especifica una posició dins del contenidor i té l'habilitat de navegar a altres posicions

Utilització de l'iterador

- Sigui C un contenidor i p un iterador per C

```
for (p = C.begin(); p != C.end(); ++p)  
    loop_body
```

- Exemple: (amb un STL vector)

```
typedef vector<int>::iterator Iterator;  
  
int sum = 0;  
  
for (Iterator p = v.begin(); p != v.end(); ++p)  
    sum += *p;  
  
return sum;
```

3.3 TAD Cua



TAD Cua

- El TAD Cua guarda arbitràriaments objectes
- Insereix i elimina segons l'esquema FIFO (first-in first-out)
- S'insereix per darrera i s'elimina per davant de la cua
- Operacions principals:
 - enqueue(object): inserta un element al final de la cua
 - dequeue(): elimina un element de l'inici de la cua
- Operacions auxiliars:
 - object front(): retornar l'element de l'inici de la cua sense eliminar-lo
 - integer size(): retorna el nombre d'elements guardats
 - boolean empty(): indica si no hi ha elements a la cua
- Excepció
 - Intentar desencuar un element de la cua buida

Especificació de la Cua en C++

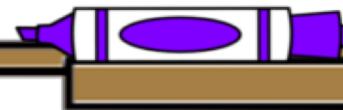
```
template <class E>
class Queue {
public:
    int size() const;
    bool empty() const;
    const E& front() const ;
    void enqueue (const E& e);
    void dequeue();
};
```

□ Possibles implementacions:

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

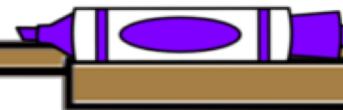


Cua en array



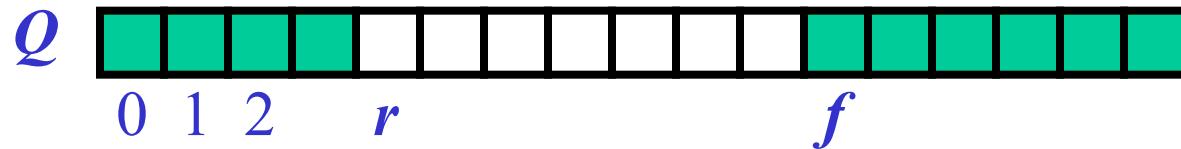
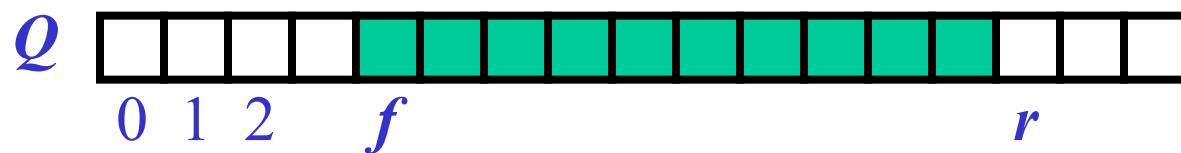


Cua en array



Cua en array

- Usa un array de mida N en mode circular
- Tres variables mantenen el control de primer element (front) i l'últim (rear) element
 - f índex de l'element al front
 - r índex immediat passat el darrer (rear) element
 - n nombre d'ítems a la cua

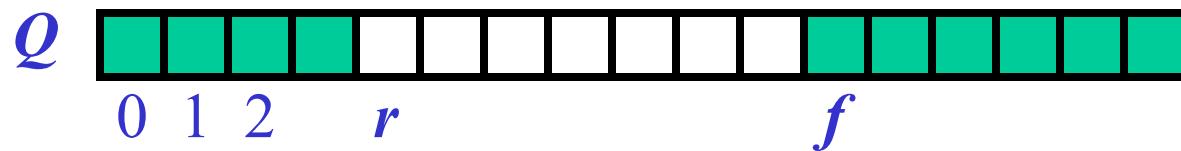


Cua en array

- Usa n per determinar la mida i si està buida

Algorithm *size()*
return *n*

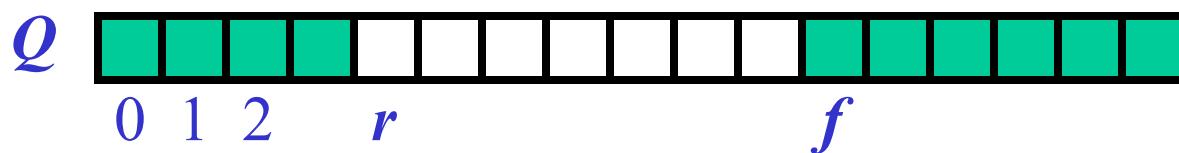
Algorithm *empty()*
 return (*n* = 0)



Cua en array

- L'operació d'enqueue
llença una excepció si
l'array està ple

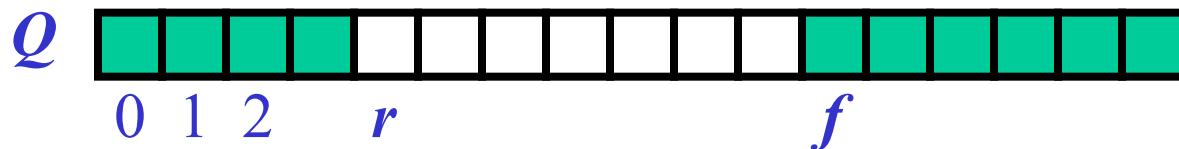
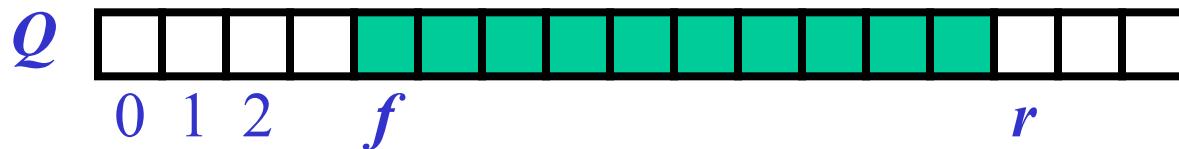
```
Algorithm enqueue(Element o)
  if size() = N – 1 then
    throw QueueFull
  else
    Q[r]  $\leftarrow$  o
    r  $\leftarrow$  (r + 1) mod N
    n  $\leftarrow$  n + 1
```



Cua en array

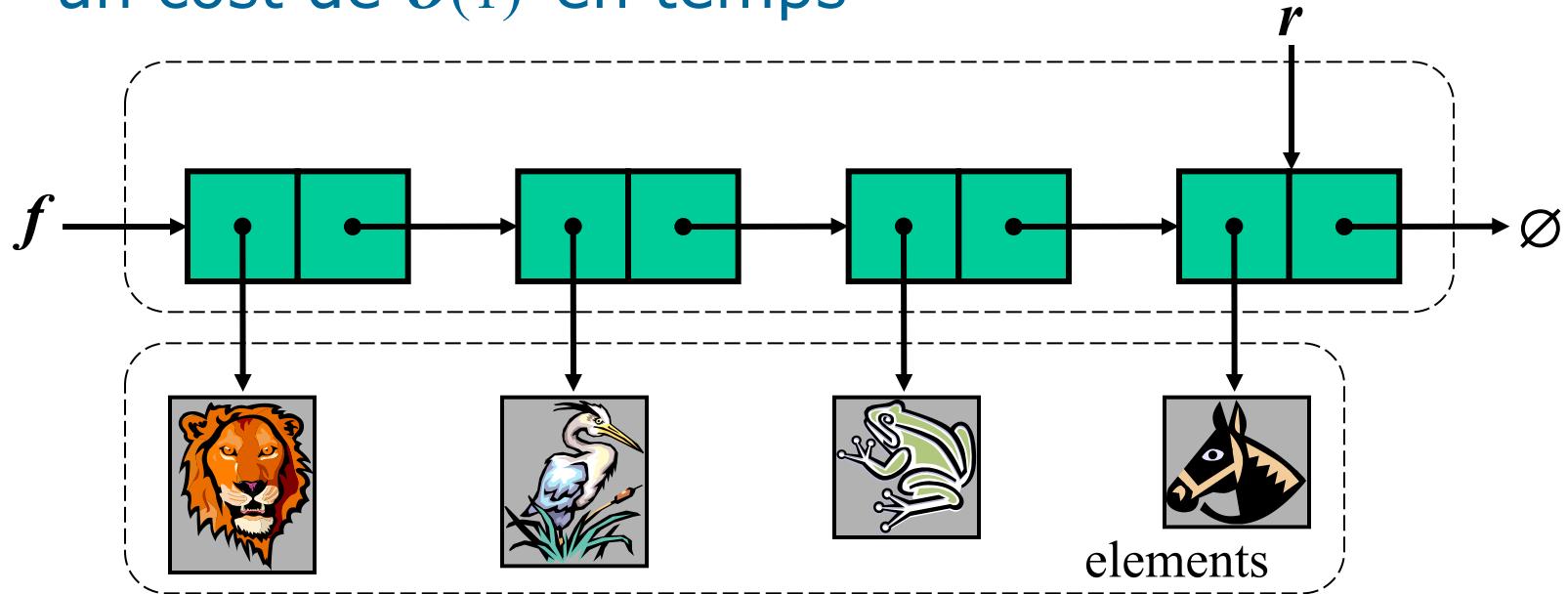
- L'operació `dequeue` llença una excepció si la cua està buida

```
Algorithm dequeue()  
if empty() then  
    throw QueueEmpty  
else  
     $f \leftarrow (f + 1) \bmod N$   
     $n \leftarrow n - 1$ 
```



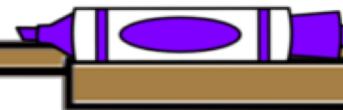
Cua en estructura encadenada

- Es pot implementar una cua amb una estructura encadenada simple
 - L'element front es guarda en el primer node
 - L'element rear es guarda en l'últim node
- L'espai usat és $O(n)$ i cada operació de la cua té un cost de $O(1)$ en temps





Cua en estructura encadenada





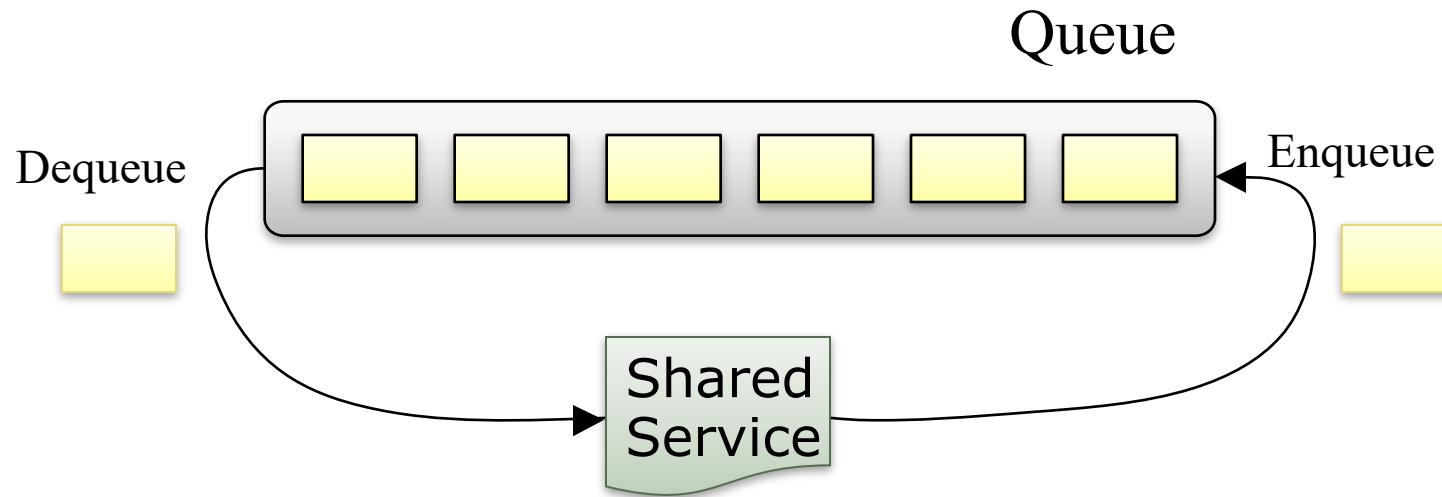
Exemple d'ús

<i>Operation</i>	<i>Output</i>	<i>Q</i>
enqueue(5)		
enqueue(3)		
dequeue()		
enqueue(7)		
dequeue()		
front()		
dequeue()		
dequeue()		
empty()		
enqueue(9)		
enqueue(7)		
size()		
enqueue(3)		
enqueue(5)		
dequeue()		

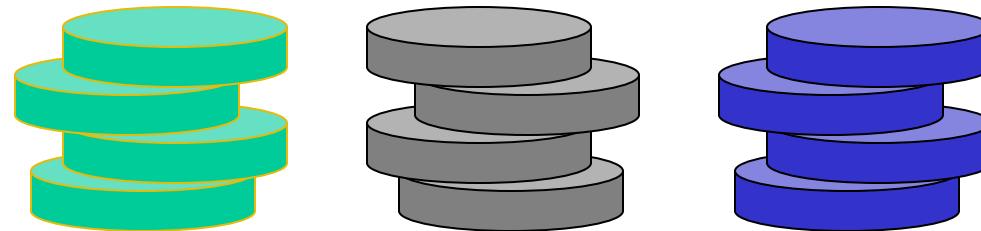
Aplicació: Round Robin Schedulers

Es pot implementar una filosofia round-robin usant una cua, Q , fent els següents pasos:

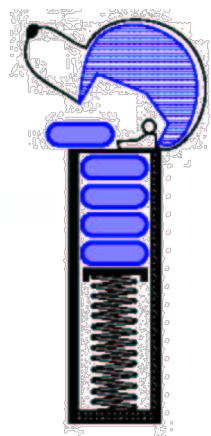
1. $e = Q.\text{front}(); Q.\text{dequeue}()$
2. Dona servei a l'element e
3. $Q.\text{enqueue}(e)$



3.4 TAD Pila



TAD Pila



- El TAD Pila guarda objectes arbitraris
- Inserta i elimina segons l'esquema LIFO (last-in first-out)
- Metàfora de la “pila de plats”
- Operacions modificadores:
 - `push(object)`: inserta un element
 - `object pop()`: treu l’últim element insertat
- Operació creadora
 - `pila()`: crea una pila buida
- Operacions consultores:
 - `object top()`: retorna l’últim element
 - `integer size()`: retorna el nombre d’elements de la pila
 - `boolean empty()`: retorna CERT si no hi ha elements

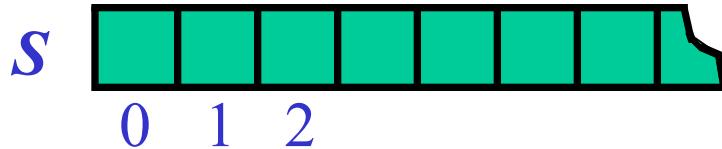
Especificació de la Pila en C++

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

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

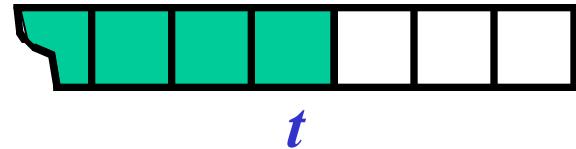
Pila en array

- El mode més simple d'implementar una pila és amb un array
- S'afegeixen els elements d'esquerra a dreta
- Una variable manté el seguiment de l'índex que indica el "top" element



```
Algorithm size()
    return  $t + 1$ 

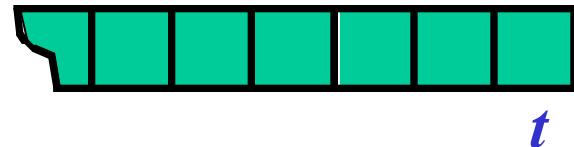
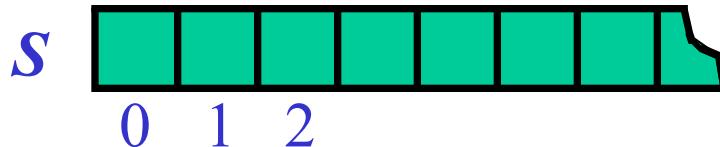
Algorithm pop()
    if empty() then
        throw StackEmpty
    else
         $t \leftarrow t - 1$ 
    return  $S[t + 1]$ 
```



Pila en array (cont.)

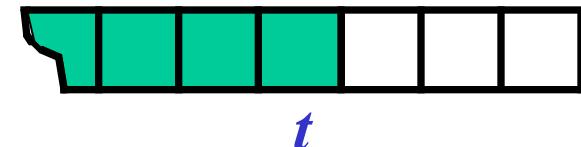
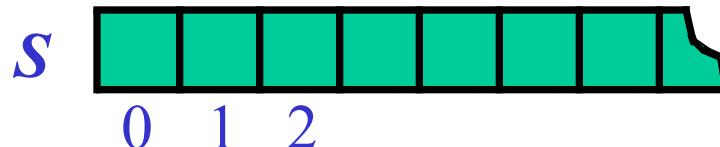
- L'array que guarda els elements es pot omplir
- L'operació de "push" en aquest cas llençarà l'excepció indicant PilaPlena
 - Això és una limitació de la implementació amb un array
 - No és intrínsec al TAD Pila

```
Algorithm push(o)
  if t = S.size() – 1 then
    throw StackFull
  else
    t  $\leftarrow$  t + 1
    S[t]  $\leftarrow$  o
```



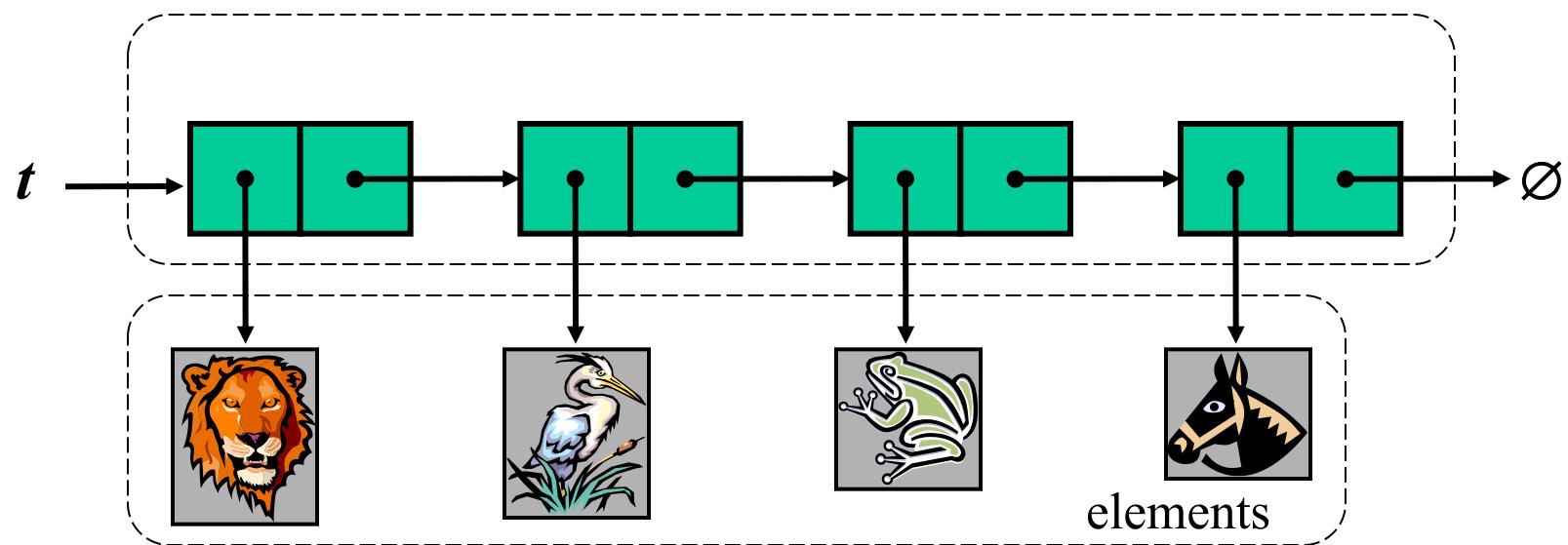
Pila implementada en un vector

```
template <class E>
class ArrayStack {
    enum{ DEF_CAPACITY = 100 } ;
public:
    ArrayStack(int cap = DEF_CAPACITY);
    int size() const;
    const E& top() const;
    void push(const E& e);
    void pop();
    // pot haver-hi més funcions
private:
    E* S;
    int capacity;
    int t;
};
```



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 temps



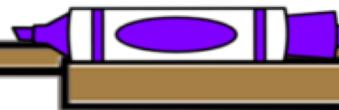
Pila implementada en estructura encadenada

```
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;
};
```

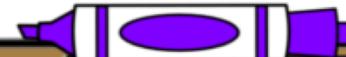


Pila en estructura encadenada



Exemple d'ús en C++

```
ArrayStack<int> A;           // A =  
A.push(7);                   // A =  
A.push(13);                  // A =  
cout << A.top() << endl; A.pop(); // A =  
A.push(9);                   // A =  
cout << A.top() << endl;      // A =  
cout << A.top() << endl; A.pop(); // A =  
ArrayStack<string> B(10);    // B =  
B.push("Bob");               // B =  
B.push("Alice");              // B =  
cout << B.top() << endl; B.pop(); // B =  
B.push("Eve");               // B =
```



Aplicació: Pila de crides en C++

- El sistema d'execució de C++ guarda la cadena de mètodes actius en una pila
- Quan es crida un mètode, el sistema fa un push a la pila d'un frame que conté:
 - Variables locals i el valor de retorn
 - El comptador de programa per saber on retornar
- Quan un mètode s'acaba es fa un pop del frame i es passa el control al mètode que està en el top de la pila

```
main() {  
    int i = 5;  
    foo(i);  
}
```

```
foo(int j) {  
    int k;  
    k = j+1;  
    bar(k);  
}
```

```
bar(int m) {  
    ...  
}
```

bar
PC = 1
m = 6

foo
PC = 3
j = 5
k = 6

main
PC = 2
i = 5



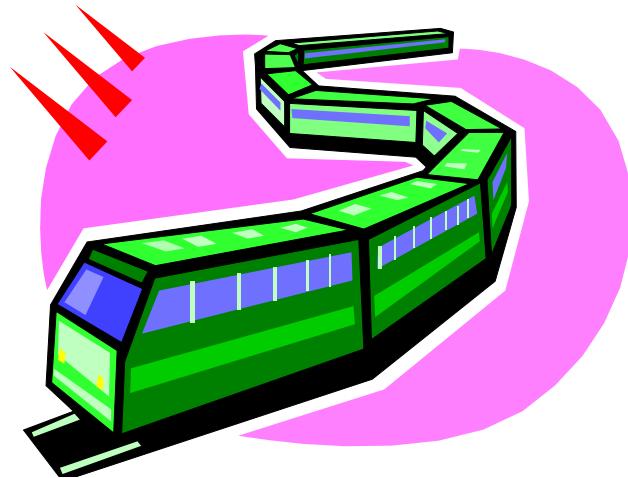
Aplicació: Problema de parentitzacions correctes

Cada “(”, “{”, “[” ha de tenir el seu corresponent “)”, “}”, “]”

- correcte: ()(()){([()])}
- correcte: ((()(())){([()])})
- incorrecte:)(()){([()])}
- incorrecte: ({[]})}
- incorrecte: (

3.5 TAD Llista

- Per posició
- Per punt d'interès





TAD Posició

- El TAD Posició modela el concepte de lloc en una estructura de dades on es guarda un conjunt d'elements
- És una vista unificada de les diferents formes de guardar les dades, com és:
 - una cel.la d'un vector
 - un node d'una representació encadenada
- Un únic mètode:
 - objecte `p.element()`: retorna l'element (objecte) de la posició
 - En C++ és convenient implementar això com `*p`



TAD posició

- Un iterador és una extensió del TAD posició
- Mètodes:
 - **p.element()**: consultar l'element
 - sobrecarregar operador (*)
 - **p.next()**: avançar a la posició següent
 - sobrecarregar operador (++)
 - retrocedir (sobrecarregar operador (--))
- Assumim que els contenidors implementen:
 - **begin()**: per retornar la primera posició del contenidor
 - **end()**: per retornar la darrera posició del contenidor

TAD Llista

- El TAD Llista (NodeList) modela una seqüència de posicions que guarden objectes arbitraris
 - Estableix una relació d'anterior/posterior entre posicions
 - Mètodes genèrics:
 - `size()`, `empty()` (no són necessaris)
- Iteradors:
 - `begin()`, `end()`
 - Mètodes modificadors:
 - `insertFront(e)`, `insertBack(e)`
 - `removeFront()`, `removeBack()`
 - Modificadors basats en iteradors:
 - `insert(p, e)`
 - `remove(p)`

TAD Llista per posició

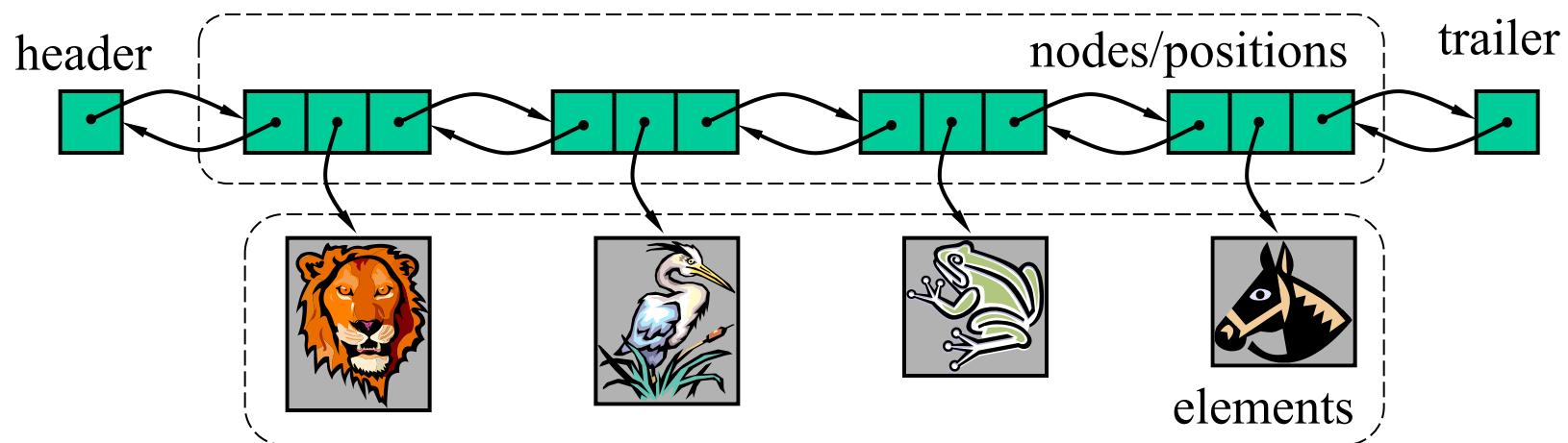
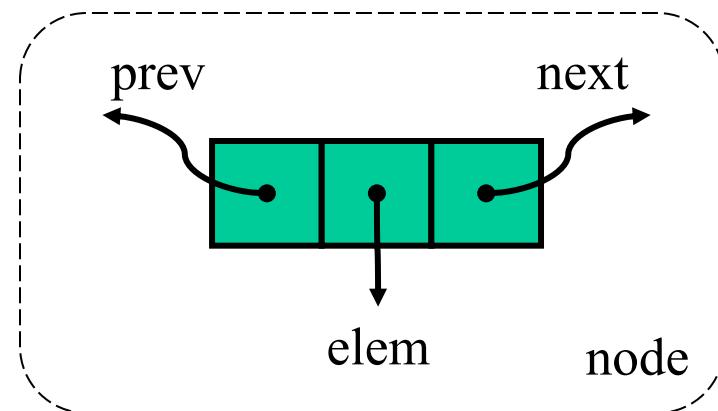
- Mètodes genèrics:
 X `size()`, `empty()`
- Mètodes consultors:
 X `begin()`, `end()`
 X `back(p)`, `next(p)`
- Mètodes modificadors:
 X `insert(p, e)`
 X `insertBefore(p, e)`, `insertAfter(p, e)`,
 X `insertFront(e)`, `insertBack(e)`
 X `remove(p)`

TAD Llista per punt d'interès

- Mètodes genèrics:
 - ✗ `size()`, `empty()`
- Mètodes consultors:
 - ✗ `get()` retorna Object del punt d'interès
- Mètodes modificadors:
 - ✗ `modify(e)` : modifica l'element del punt d'interès
 - ✗ `insert(e)` : inserta element al punt d'interès
 - ✗ `remove()` retorna Object
- **Mètodes del punt d'interès:**
 - ✗ `begin()`: es sitúa a l'inici
 - ✗ `next()`: avança la posició
 - ✗ `end()` retorna boolea si ha arribat al final

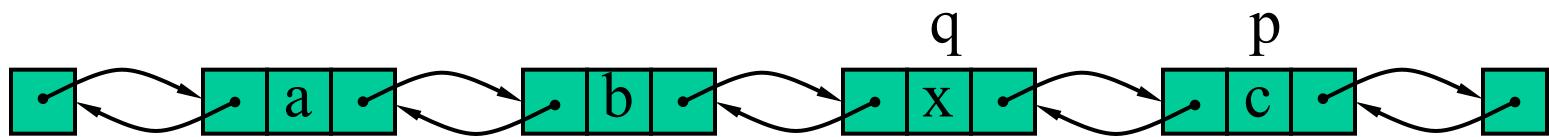
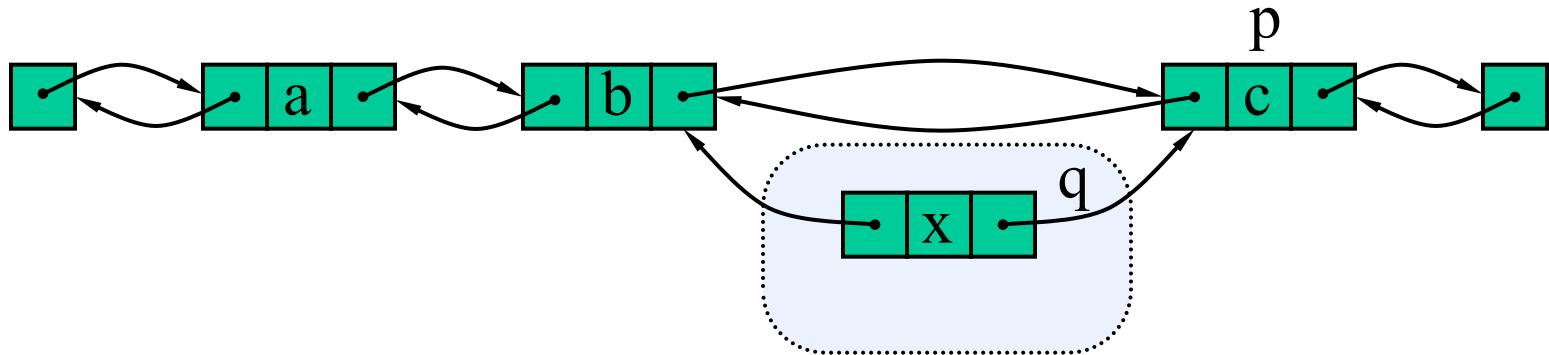
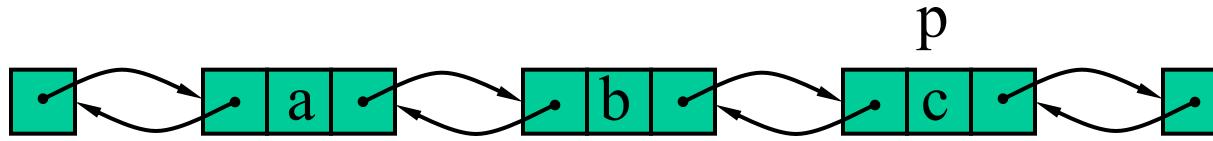
Llista amb encadenaments dobles

- Llista amb encadenaments dobles és la implementació natural del TAD Llista (NodeList)
- Els nodes implementen la Posició i guarden:
 - element
 - link al node anterior
 - link al següent node



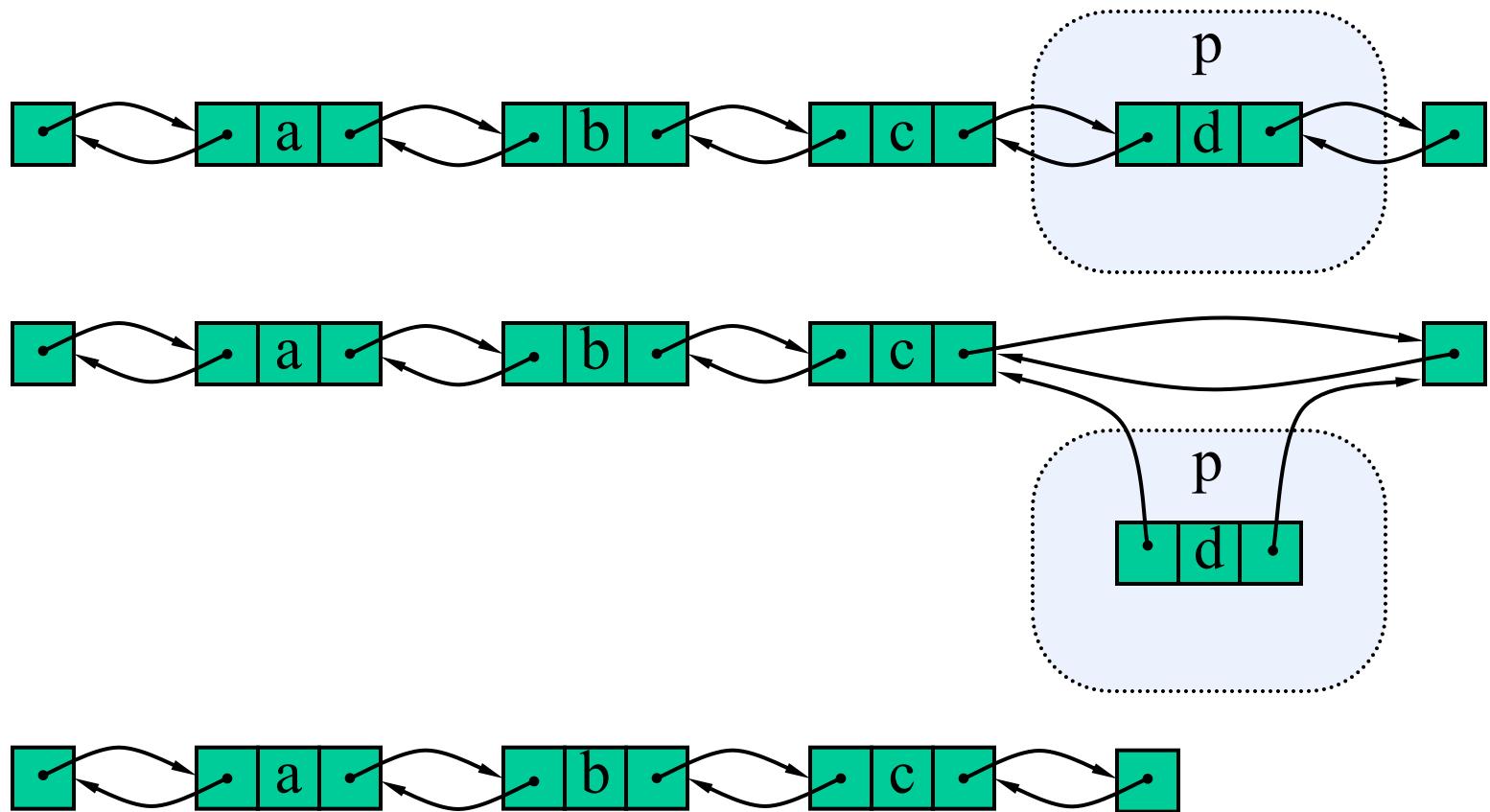
Inserció

- Visualització de l'operació $\text{insert}(p, x)$, la qual inserta x abans de p



Eliminar

- Visualització de l'operació `remove(p)`



Rendiment TAD NodeList

- En la implementació del TAD Llista amb una estructura doblement encadenada
 - L'espai usat per la llista amb n elements és $O(n)$
 - L'espai usat per cada posició de la llista és $O(1)$
 - Totes les operacions del TAD NodeList tenen un cost $O(1)$ en temps
 - L'operació `element()` del TAD Posició té un cost $O(1)$ en temps

3.7 TAD cua prioritària





TAD cua prioritària

- Una cua prioritària guarda una col·lecció d'entrades
 - Cada entrada (entry) és un parell (clau, valor), on clau indica la prioritat
 - Dos entrades diferents en una cua prioritària poden tenir la mateixa clau
- Mètodes
 - `insert(e)` insereix una entry e
 - `removeMin()` elimina la entry amb la clau mínima
- Mètodes addicionals:
 - `min()` retorna però no elimina la entry amb clau mínima
 - `size()`, `empty()`

TAD Comparador

- Implementa la funció booleana `isLess(p,q)`, la qual testeja si $p < q$
- Pot derivar altres relacions des d'aquí:
 - $(p == q)$ és equivalent a
 - $(\text{!isLess}(p, q) \& \text{!isLess}(q, p))$
- Pot implementar-se en C++ sobrecarregant `"()"`

Dues maneres de comparar 2D points:

```
class LeftRight { // left-right comparator
public:
    bool operator()(const Point2D& p,
                     const Point2D& q) const
    { return p.getX() < q.getX(); }

class BottomTop { // bottom-top
public:
    bool operator()(const Point2D& p,
                     const Point2D& q) const
    { return p.getY() < q.getY(); }
```

Ordenació amb cues prioritàries

- Es pot usar una cua prioritària per ordenar un conjunt d'elements
 1. Insereix els elements una a una amb una sèrie d'operacions de `insert`
 2. Extreu els elements en ordre amb una sèrie d'operacions de `removeMin`
- El temps de l'ordenació depèn de la implementació de la cua prioritària

Algorithm $PQ\text{-}Sort}(S, C)$

Input seqüència S , comparador C pels elements de S

Output seqüència S ordenada en ordre creixent segons C

$P \leftarrow$ cua prioritària amb el comparador C

while $\neg S.\text{empty}()$

$e \leftarrow S.\text{front}(); S.\text{eraseFront}()$

$P.\text{insert}(e, \emptyset)$

while $\neg P.\text{empty}()$

$e \leftarrow P.\text{removeMin}()$

$S.\text{insertBack}(e)$

Cues prioritàries basades en encadenaments (ordenació)

- Llista desordenada



- Eficiència:

- insert tarda $O(1)$ en temps ja que quan s'insereix un item, s'insereix al començament de la seqüència
 - removeMin i min tarden $O(n)$ en temps ja que s'ha de recórrer tota la seqüència per trobar la clau mínima

- Llista ordenada



- Eficiència:

- insert tarda $O(n)$ en temps ja que s'ha de trobar el lloc on correspon inserir l'ítem
 - removeMin i min tarden $O(1)$ en temps, ja que la clau mínima està sempre al començament



Tema 3 Estructures Lineals

Maria Salamó Llorente
Estructura de Dades

Grau en Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona