



Tema 3 Estructures Lineals

Sessió Teo 3

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)

3.3 Concepte de TAD (Introducció)

Sessió Teoria 4 (Teo 4)

3.3 Concepte de TAD (Continuació i exemple)

3.4 TAD Pila i TAD Cua (representació dinàmica)

Sessió Teoria 5 (Teo 5)

3.5 TAD Llista

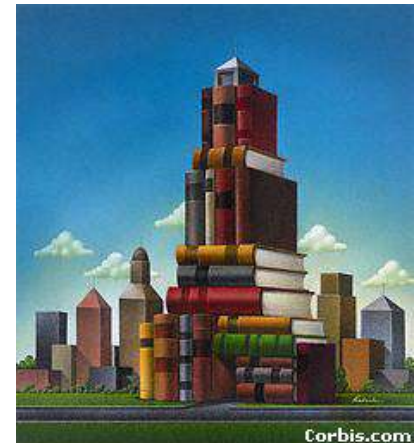
3.6 TAD Cua prioritària (es veurà al Tema 5)

Què són les estructures lineals?

- Parlem d'estructures lineals quan les dades s'organitzen de forma seqüencial
 - Tots els elements, excepte el primer i l'últim, tenen un element abans (predecessor) i un element després (successor)

$$a_1 \rightarrow \dots \rightarrow a_{i-1} \rightarrow a_i \rightarrow a_{i+1} \rightarrow \dots \rightarrow a_n$$

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



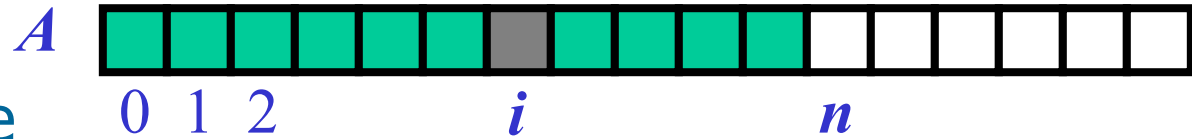


3.1 Introducció a les estructures lineals

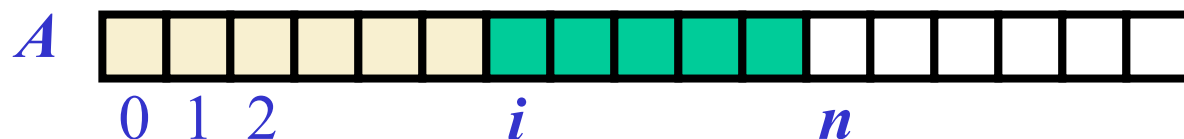
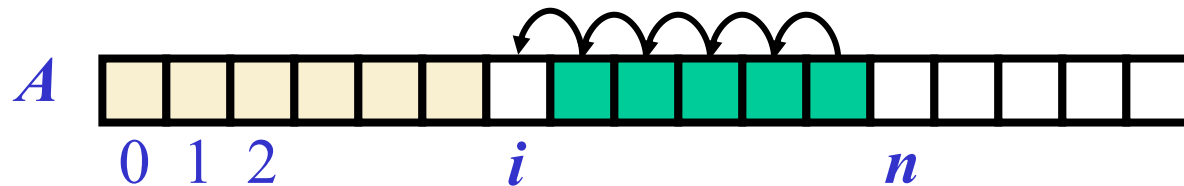
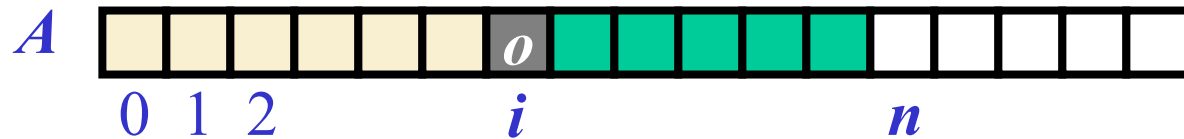
Implementacions bàsiques

• Arrays

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



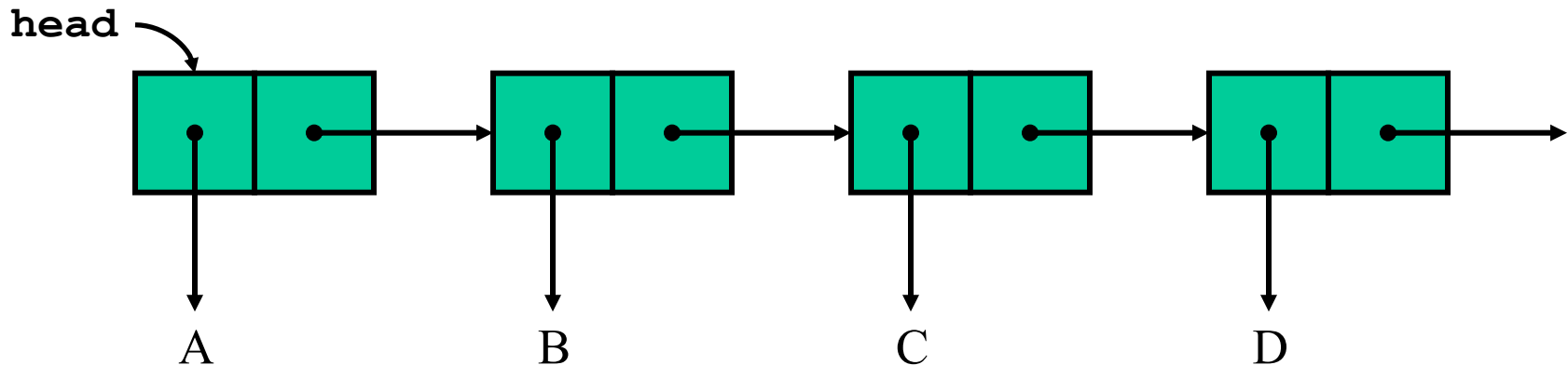
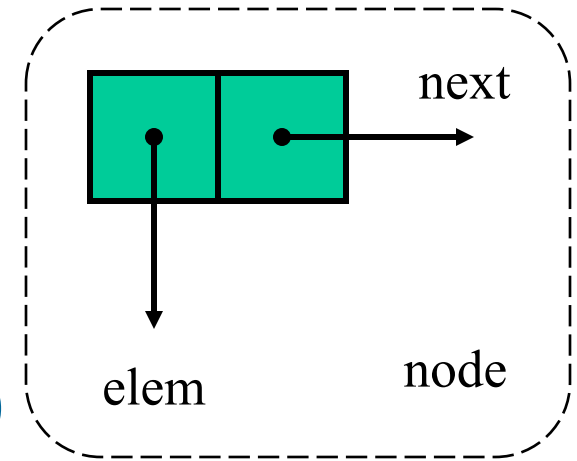
Per exemple, esborrar:



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 $O(n)$





Què és un Tipus Abstracte de Dades

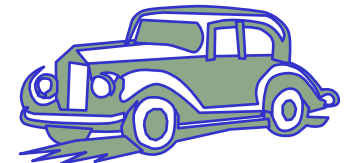
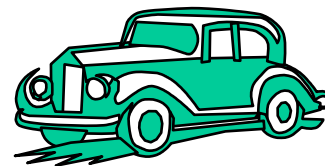
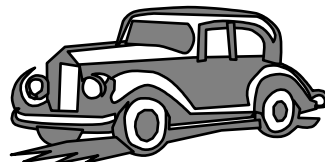
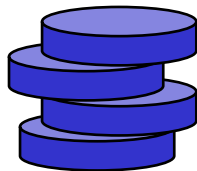
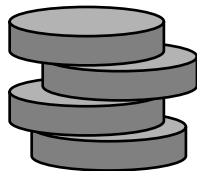
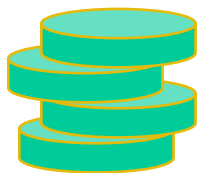
- Definirem TAD (**Tipus Abstracte de Dades**) com un codi que proporciona:
 - Un tipus de dades, i
 - Un conjunt d'operacions per a treballar sobre valors d'aquest tipus
- En un TAD, la paraula **abstracte** fa referència al fet de poder treballar amb el tipus amb independència de com està representat,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

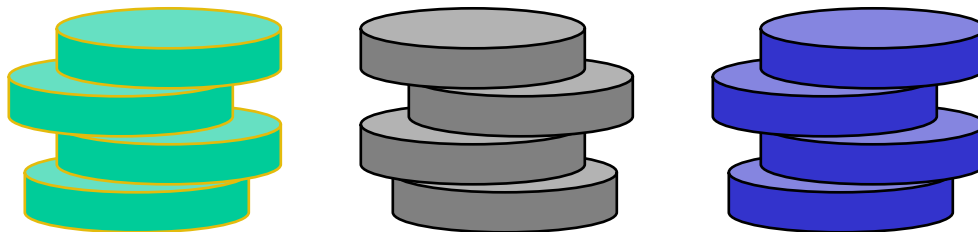
Un TAD consta de tres parts:

- **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

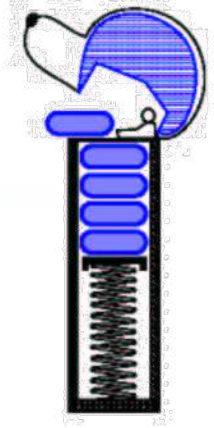
3.3. TAD Pila i TAD Cua (representació estàtica)



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

Applications de les piles

- Aplicacions directes
 - Historial de visites de les pàgines web en un navegador
 - Seqüències “Undo” en un editor de text
 - Cadena de crides als mètodes en un programa en execució
- Aplicacions indirectes
 - Estructura de dades auxiliar per altres algorismes
 - Component d’altres estructures de dades

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

Excepcions

- Intentar l'execució d'alguna operació del TAD, a vegades pot produir-se una condició d'error, anomenada EXCEPCIÓ
- Les excepcions són llençades ("throw") per una operació que no pot executar-se
- En el TAD Pila, les operacions **pop** i **top** no poden executar-se si la pila està buida
- En aquest cas, la pila llença l'excepció **StackEmpty** (**PilaPlena**)

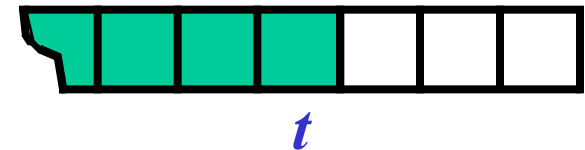
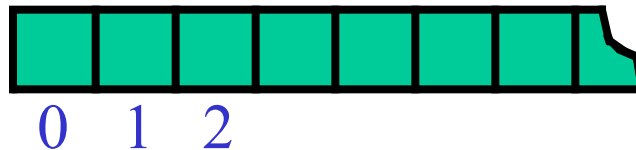
Pila en array

- El mode més simple de representar i 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  $S[t + 1]$ 
```

stack



Pila en array (cont.)

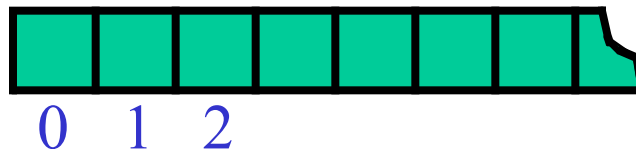
- ❑ L'array que guarda els elements es pot omplir
- ❑ L'operació de "push" en aquest cas llençarà l'excepció indicant

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  
     $t \leftarrow t + 1$   
     $S[t] \leftarrow o$ 
```

stack



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 definides al TAD
private:
    E* stack; //també es pot representar com un vector o un array estàtic
    int capacity;
    int t;
};
```

stack



0 1 2



t

SOLUCIO Exemple d'ús en C++

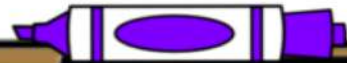
* indica el top

```
ArrayStack<int> A;  
A.push(7);  
A.push(13);  
cout << A.top() << endl; A.pop();  
A.push(9);  
cout << A.top() << endl;  
cout << A.top() << endl; A.pop();
```

```
// A = [ ], size = 0  
// A = [7*], size = 1  
// A = [7, 13*], size = 2  
// A = [7*], outputs: 13  
// A = [7, 9*], size = 2  
// A = [7, 9*], outputs: 9  
// A = [7*], outputs: 9
```

```
ArrayStack<string> B(10);  
B.push("Bob");  
B.push("Alice");  
cout << B.top() << endl; B.pop();  
B.push("Eve");
```

```
// B = [ ], size = 0  
// B = [Bob*], size = 1  
// B = [Bob, Alice*], size = 2  
// B = [Bob*], outputs: Alice  
// B = [Bob, Eve*], size = 2
```



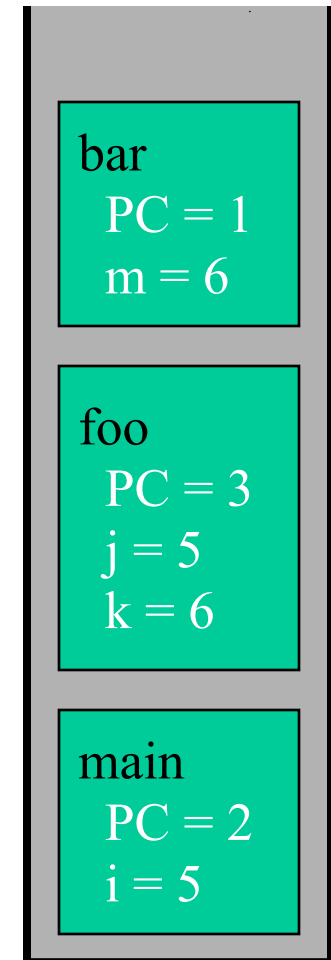
Aplicació: Pila de crides en C++

- 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 (PC) 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

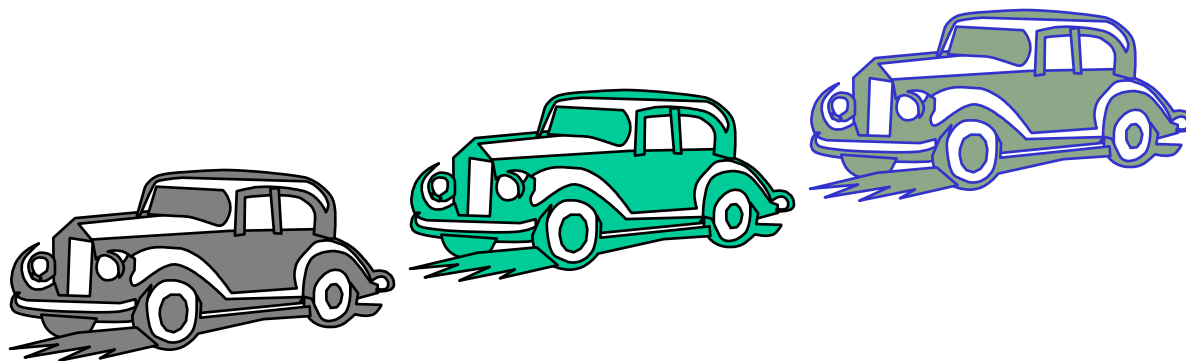
```
main() {  
    int i = 5;  
    foo(i);  
}
```

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

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



TAD Cua





TAD Cua

- El TAD Cua guarda arbitràriament 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 en una cua buida



Aplicacions de les cues

- Aplicacions directes
 - Cues d'espera
 - Accés a recursos compartits, per exemple una impressora
- Aplicacions indirectes
 - Estructura de dades auxiliar per altres algorismes
 - Component d'altres estructures de dades

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)



Consideracions de l'especificació de la Classe Queue

- Noteu que a la transparència anterior tenim definides les operacions bàsiques que es defineixen a una cua, independentment de la seva representació i implementació
- Estem definint **l'especificació** del TAD Cua, i per això no s'han definit atributs perquè es farà quan es defineixi la implementació
- Els mètodes **size()**, **empty()** i **front()** són funcions **consultores**
- Els mètodes **enqueue()** i **dequeue()** són funcions **modificadores**



Implementació cua en array

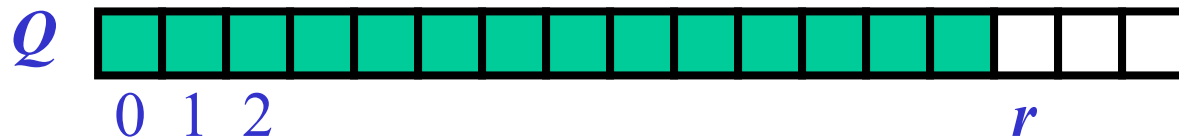
Hi ha diverses maneres d'implementar la cua en array. A continuació les definim.

- Volem la versió que genera un cost computacional menor a les funcions TAD del Queue, sobretot es busca el menor cost de les funcions d'enqueue i dequeue.
- Recordeu que el problema de les implementacions amb array es que definim un màxim d'espai i ens podem quedar fàcilment sense espai per guardar elements, si s'ha definit un màxim petit, o definim un màxim molt gran i ocupem més espai del que realment necessitem per l'estructura de dades.

Implementació cua en array

1a. Versió d'implementació

- Només cal una variable per definir el rear (variable r al dibuix) ja que el front sempre estarà a la posició 0 de l'array
- El rear creix quan s'insereixen elements
- Quan s'elimina un element del front, es desplacen tots els elements a l'esquerra

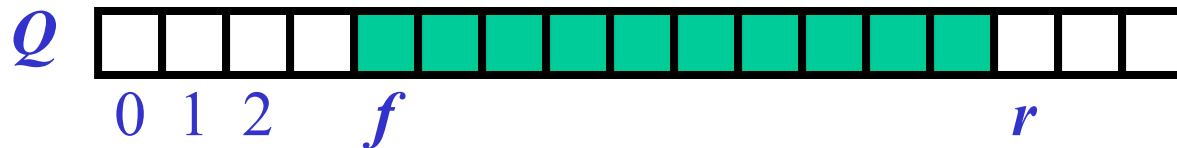


- **Enqueue** té un cost computacional de **$O(1)$**
- **Dequeue** té un cost computacional de **$O(n)$** , en el pitjor dels casos

Implementació cua en array

2a. Versió d'implementació

- Usarem dues variables per definir el front (f al dibuix) i el rear (r al dibuix), f i r són de tipus enter
- El front creix quan **s'eliminen** elements
- El rear creix quan **s'insereixen** elements



- D'aquesta manera, **dequeue** i **enqueue** tenen un cost computacional de **$O(1)$**
- **Problema !!!**
 - Front i rear avancen en cada operació, tot i tenir espais buits a l'array es poden trobar que arriben al màxim de l'array



Implementació cua en array

2a. Versió d'implementació (cont.)

- **Solució** al problema anterior
 - Fer que el vector usi l'array de forma circular. Quan f i r arriben al final de l'array, si la cua no està plena s'han d'enviar a la posició 0 de l'array per seguir avançant.
- A continuació teniu la descripció dels elements de la cua implementada amb array circular.

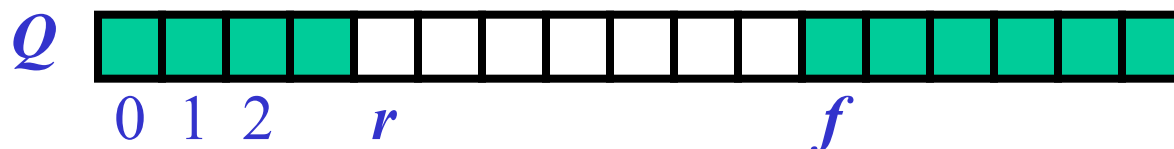
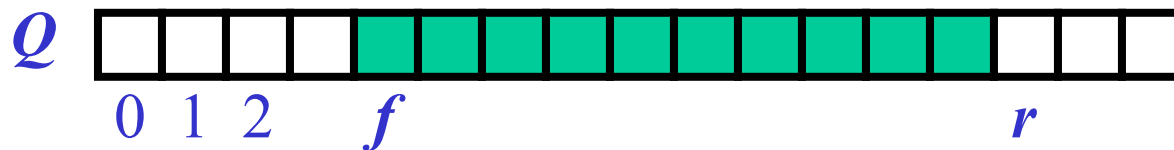
Cua en array circular

- Usa un array de mida N en mode circular
- Tres variables mantenen el control de la cua, el primer element (**front**) a la variable **f**, l'últim element (**rear**) a la variable **r** i una variable per comptar el nombre d'elements de la cua (**n**)

f índex de l'element al front

r índex immediat passat el darrer (rear) element

n nombre d'ítems a la cua

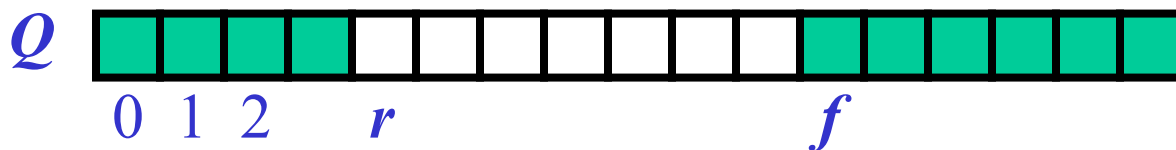
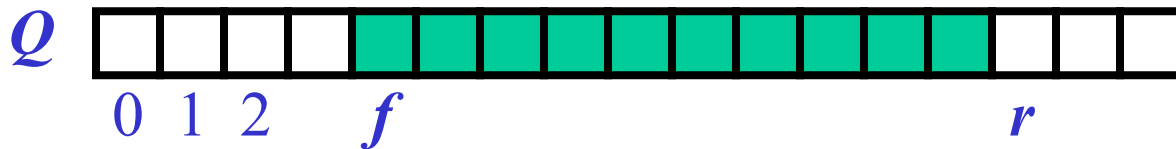


Cua en array circular

- Com que f i r poden estar a qualsevol posició, *mireu el dibuix*
 - Si f i r estan junts no sabem si la cua està plena o buida
- Usa n per determinar la mida de la cua i si està buida

Algorithm *size()*
return n

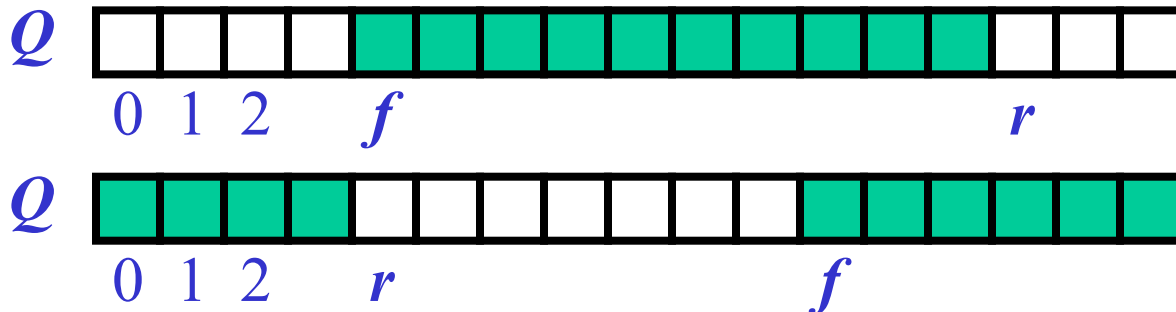
Algorithm *empty()*
return $(n = 0)$



Cua en array circular

- Per fer que f i r siguin circulars, només cal usar l'operació de mòdul per obtenir la nova posició. *Mireu el pseudocodi de la funció **enqueue***
- L'operació d'enqueue llença una excepció si l'array està ple

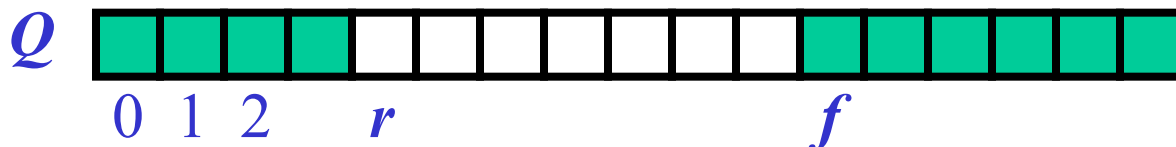
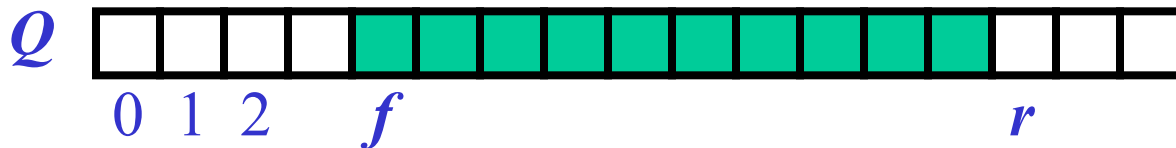
```
Algorithm enqueue(Element  $o$ )  
  if  $size() = N - 1$  then  
    throw QueueFull  
  else  
     $Q[r] \leftarrow o$   
     $r \leftarrow (r + 1) \bmod N$   
     $n \leftarrow n + 1$ 
```



Cua en array circular

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

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





Comentaris de la Cua en array

- **La cua en array circular té un cost computacional de $O(1)$ en totes les seves operacions**
- Tot i aquest bon cost computacional, encara té el **problema de l'espai limitat a la mida de l'array**
- **Solució:** Implementar la cua amb una estructura encadenada.



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
- 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 **predefinit**s (són proporcionats pel llenguatge):
 - enter, real, caràcter, boolean
 - Tipus de dades **definit per l'usuari** (usant taules o tuples)



Tipus predefinit

Un **tipus de dades predefinit** determina:

- Quins valors pot representar (**domini**)
 - enters $\rightarrow 0, 1, -1, -2, \dots$
 - booleans \rightarrow 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 predefinitos, 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 predefinitos (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 depenent 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**

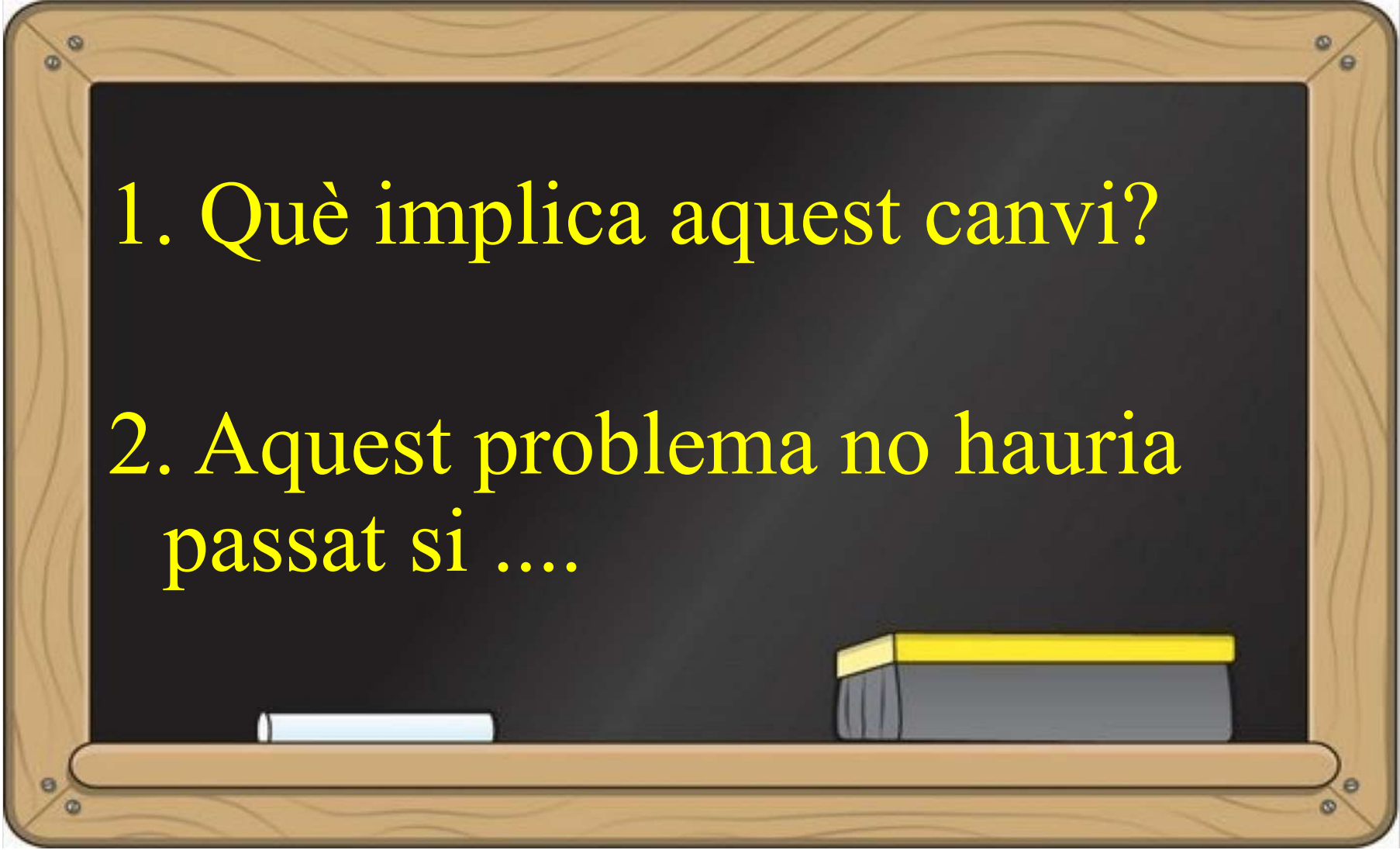
coef: real

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.)

- 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!**

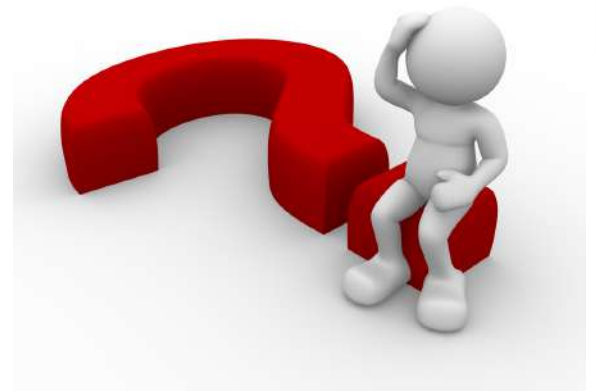
Solució

- **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



Tema 3 Estructures Lineals

Sessió Teo 3

Maria Salamó Llorente

Estructura de Dades

Grau en Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona