# Estructures lineals I: Piles i cues

Ricard Gavaldà

Programació 2

Facultat d'Informàtica de Barcelona, UPC

Primavera 2019

Aquesta presentació no substitueix els apunts

# Contingut

# Estructures lineals

# Estructures lineals

- $e_1, e_2, \ldots, e_n$
- $n = 0$: estructura buida
- "primer" i "darrer"
- "anterior" i "següent"

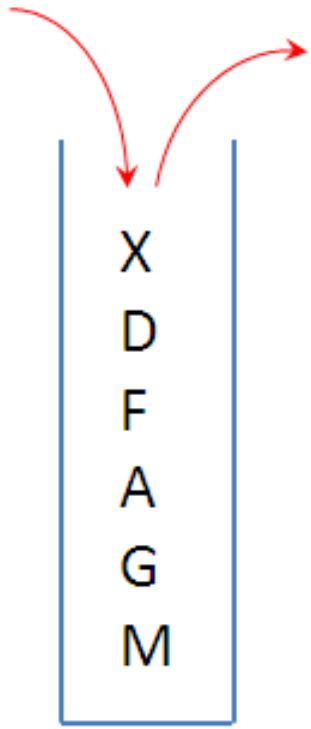
# Estructures lineals

- Vectors, piles, cues, llistes
- Piles i cues són versions restringides de les llistes, que permeten algunes implementacions una mica més eficients
- En la STL, `vector`, `stack`, `queue`, `list`.
- Són *templates*: necessiten un tipus com a paràmetre
- Són casos particulars de *contenidors*

# El tipus pila (`stack`)

# El tipus pila (stack)

Només podem accedir a un extrem de l'estructura



# El tipus pila (`stack`)

- LIFO - *Last In, First Out*: el darrer que ha entrat serà el primer en sortir, i és l'únic accessible
- Operacions bàsiques:
  - `empty`: diu si és buida (booleà)
  - `push`: afegir un element
  - `pop`: treure un element, *el darrer afegit*
  - `top`: consultar un element, *el darrer afegit*

# Especificació de stack

Vegeu `Pila.doc`

Observeu: paraula reservada `template` i paràmetre tipus `T`

Instanciacions:

```
stack<int> p;
stack<Estudiant> q;
stack<vector<int> > sv;
vector<stack<Estudiant> > vs;
```

# Advertiment

- En molts d'aquests exercicis (i els de cues), tenim problemes perquè intentem accedir a la pila (o la cua) d'una manera que no és la que la pila o la cua espera
- Per exemple, necessitem accedir a tots els elements i no només al que és accessible
- Això genera problemes, per exemple, que la pila és destruïda i cal reconstruir-la, o copiar-la abans
- Probablement significa que *si necessitem fer això, hauríem d'usar una llista i no una pila o cua*
- Preneu-vos-ho com a exercicis amb valor pedagògic més que d'aplicació real

# Alçària d'una pila

l'alçària de la pila $p = e_1 \dots e_n$ és $n$, el seu nombre d'elements

Especificació:

```
/* Pre: cert */
/* Post: resultat = nombre d'elements de p */
int alcaria(const stack<int>& p);
```

Nota. És un exercici. Qualsevol implementació decent de les piles vindrà amb aquesta funció "de sèrie"

# Alçària d'una pila, versió iterativa

```
/* Pre: cert */
/* Post: resultat = nombre d'elements de p */
int alcaria(const stack<int>& p) {
    int n = 0;
    while (not p.empty()) {
        ++n;
        p.pop();
    }
    return n;
}
```

Quin problema hi veieu?

# Alçària d'una pila

Problema: modifiquem `p`, que és `const &`
Solucions:

- passar-la sense `&`, còpia automàtica en fer la crida
- `const &` i fer còpia dins, en variable local

```
int alcaria(const stack<int>& p) {
    stack<int> aux(p);
    comptar elements d'aux;
}
```

- modificar l'especificació: eliminar el `const`
  - L'usuari és responsable de fer una còpia prèvia, si vol
  - Risc: que se n'obliddi
  - Avantatge: no hi ha cap còpia, si no fa falta

# Alçària d'una pila, versió recursiva

```
/* Pre: p = P */
/* Post: resultat = nombre d'elements de P i p es buida */
int alcaria(stack<int>& p) {
    if (p.empty()) return 0;
    else {
        p.pop();
        return 1 + alcaria(p);
    }
}
```

Si passem p sense &, una còpia a cada crida
Temps i memòria quadràtics en la mida de p

# Suma dels elements d'una pila

Si $P = e_1, \ldots, e_n$, $\text{suma}(P) = e_1 + \cdots + e_n$

```
/* Pre: p = P */
/* Post: resultat = suma dels elements de P i p es buida */
int suma(stack<int>& p);
```

No posem `const` però avisem que `p` queda buida

# Suma dels elements d'una pila

```
/* Pre: p = P */
/* Post: resultat = suma dels elements de P i p es buida */
int suma(stack<int>& p) {
    if (p.empty()) return 0;
    else {
        int x = p.top();
        p.pop();
        return x + suma(p);
    }
}
```

# Suma dels elements d'una pila

En aquest cas, podem restaurar p al seu valor original, però encara hem d'eliminar el const

```
/* Pre: p = P */
/* Post: resultat = suma dels elements de P, i p = P */
int suma(stack<int>& p) {
    if (p.empty()) return 0;
    else {
        int x = p.top();
        p.pop();
        int res = x + suma(p);
        p.push(x);
        return res;
    }
}
```

# Cerca en una pila

Donada una pila p i un element x, dir si x apareix en p

```
/* Pre: p = P */
/* Post: El resultat ens diu si x és un element de P o no,
         i no es segur que p = P */
bool cerca(stack<int>& p, int x);
```

Versió iterativa: exercici
Ull: que sigui una *cerca*, no un *recorregut*

# Cerca en una pila

```
/* Pre: p = P */
/* Post: El resultat ens diu si x és un element de P o no,
         i no es segur que p = P */
bool cerca(stack<int>& p, int x) {
    if (p.empty()) return false;
    else if (p.top() == x) return true;
    else {
        p.pop();
        return cerca(p, x);
    }
}
```

# Cerca en una pila d'Estudiants

Solució iterativa. Cal usar consultores d'Estudiant:

```
/* Pre: x > 0, p = P */
/* Post: El resultat ens diu si hi ha algun estudiant
    amb dni x a P, i p pot no ser igual que P */
bool cerca(stack<Estudiant>& p, int x) {
    bool ret = false;
    while (not p.empty() and not ret){
        if (p.top().consultar_DNI() == x) ret = true;
        else p.pop();
    }
    return ret;
}
```

# Sumar un nombre a una pila

```
/* Pre: p = P */
/* Post: cada element de p es la suma de l'element de P
         que ocupa la mateixa posicio mes el valor k */
void suma_k(stack<int>& p, int k);
```

p.ex, p = (3,6,2,1), k = 2 $\rightarrow$ p = (5,8,4,3)

# Sumar un nombre a una pila, recursiu

```
/* Pre: p = P */
/* Post: cada element de p es la suma de l'element de P
         que ocupa la mateixa posicio mes el valor k */
void suma_k(stack<int>& p, int k) {
    if (not p.empty()) {
        int x = p.top();
        p.pop();
        suma_k(p,k);
        p.push(x+k);
    }
}
```

# Sumar un nombre a una pila: Versió iterativa

Sembla que necessitem pila auxiliar ...

```
void suma_k(stack<int>& p, int k) {
    stack<int> aux;
    while (not p.empty()) {
        aux.push(p.top()+k);
        p.pop();
    }
    p = aux;
}
```

Problema?

Solucions: 1) un altre bucle; 2) funció revessar pila; 3) sumar_k(p,0)

# El tipus cua (queue)

# El tipus cua (queue)

Només podem afegir en un extrem, i consultar *l'altre* extrem

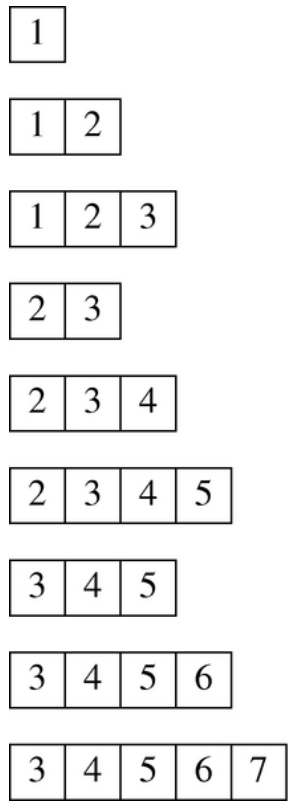
# Cues: Definició

- FIFO - *First In, First Out* el que ha arribat primer serà el primer en sortir; i és l'únic accessible
- Operacions bàsiques:
  - `empty()`: diu si és buida
  - `push(x)`: demanar tanda/torn com a últim element
  - `pop()`: avançar; elimina el primer element
  - `front()`: retorna el primer element

# Exemple d'evolució d'una cua

Cua d'enters:

1. valors 1, 2 i 3 demanen tanda
2. avança
3. valors 4 i 5 demanen tanda
4. avança
5. valors 6 i 7 demanen tanda



# Especificació de la classe Cua

- Vegeu Cua.doc
- Instanciació: `queue<tipus> nom_cua;`
- Exemple:

```
queue<int> c;
queue<Estudiant> q;
```

# Exercici: llargada d'una cua

```
/* Pre: cert */
/* Post: resultat = nombre d'elements de c */
int longitud(const queue<int>& c);
```

Però serà més convenient fer:

```
/* Pre: c = C */
/* Post: resultat = nombre d'elements de C, i c es buida */
int longitud(queue<int>& c);
```

# Solució: llargada d'una cua, versió iterativa

```
/* Pre: c = C */
/* Post: resultat = nombre d'elements de C, i c es buida */
int longitud(queue<int>& c) {
    int ret = 0;
    while (not c.empty()) {
        ++ret;
        c.pop();
    }
    return ret;
}
```

# Solució: llargada d'una cua, versió recursiva

```
/* Pre: c = C */
/* Post: resultat = nombre d'elements de C, i c es buida */
int longitud(queue<int>& c) {
    if (c.empty()) return 0;
    else {
       c.pop();
       return 1 + longitud(c);
    }
}
```

c és buida a la Post? Per inducció!

# Suma dels elements d'una cua, versió iterativa

```
/* Pre: c = C */
/* Post: resultat = suma dels elements de C; c es buida */
int suma(queue<int>& c) {
    int ret = 0;
    while (not c.empty()) {
        ret += c.front();
        c.pop();
    }
    return ret;
}
```

Exercici: feu-n una versió que deixi c com al principi, usant `c.size()`

# Suma dels elements d'una cua, versió recursiva

```
/* Pre: c = C */
/* Post: resultat = suma dels elements de C; c es buida */
int suma(queue<int>& c) {
    if (c.empty()) return 0;
    else {
        int x = c.front();
        c.pop();
        return x + suma(c);
    }
}
```

Exercici: comproveu que no és tan fàcil fer-ne una versió que deixi c com al principi

# Cerca en una cua

```
/* Pre: c = C */
/* Post: El resultat diu si x apareix a C o no;
         c pot no ser C */
bool cerca(queue<int>& c, int x);
```

# Cerca en una cua, versió iterativa

```
/* Pre: c = C */
/* Post: El resultat diu si x apareix a C o no;
         c pot no ser C */
bool cerca(queue<int>& c, int x) {
    while (not c.empty() and c.front() != x) c.pop();
    return not c.empty();
}
```

Exercici: escriure la cerca per a cues d'`Estudiant`

# Cerca en una cua, versió recursiva

```
/* Pre: c = C */
/* Post: El resultat diu si x apareix a C o no;
         c pot no ser C */
bool cerca(queue<int>& c, int x) {
    if (c.empty()) return false;
    else if (x == c.front()) return true;
    else {
        c.pop();
        return cerca(c,x);
    }
}
```

# Sumar $k$ als elements d'una cua

```
/* Pre: c = C */
/* Post: cada element de c es la suma de l'element de C
         que ocupa la mateixa posició més el valor k */
void suma_k(queue<int>& c, int k);
```

# Sumar $k$ als elements d'una cua: versió iterativa

Al menys 3 maneres:

- Si tenim un mètode `size` ràpid, ...
- Si sabem que tots els elements de c són positius, ...
- O bé, usant una variable temporal de tipus cua

Fem la tercera. Les dues primeres són exercicis

# Sumar $k$ als elements d'una cua: versió iterativa

```
/* Pre: c = C */
/* Post: cada element de c es la suma de l'element de C
         que ocupa la mateixa posició més el valor k */
void suma_k(queue<int>& c, int k) {
    queue<int> c_aux;
    while (not c.empty()) {
        c_aux.push(c.front() + k);
        c.pop();
    }
    c = c_aux;
}
```

Noteu la còpia d'una cua al final

# Sumar $k$ als elements d'una cua: versió recursiva

Solució amb un problema:

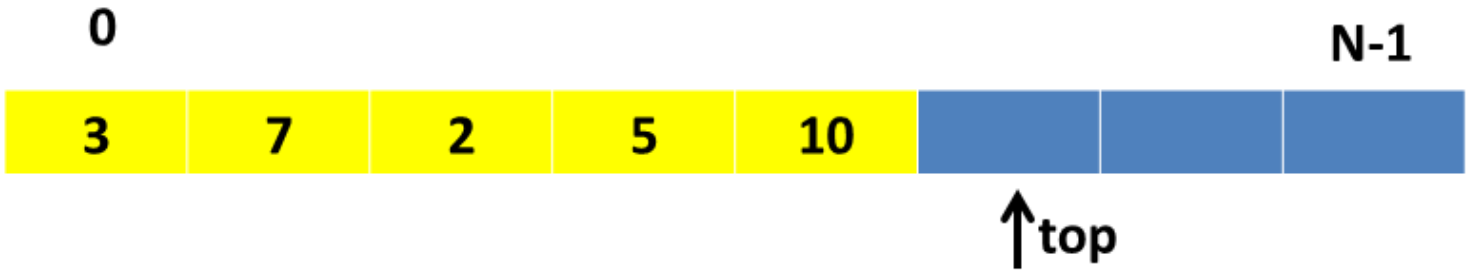
```
/* Pre: c = C */
/* Post: cada element de c es la suma de l'element de C
         que ocupa la mateixa posició més el valor k */
void suma_k(queue<int>& c, int k) {
   if (not c.empty()) {
       int x = c.front();
       c.pop();
       suma_k(c,k);
       c.push(x+k);
   }
}
```

Exercicis relacionats.
Jugueu amb cues/piles auxiliars i/o amb paràmetres &

- Revessar una cua, iteratiu
- Revessar una pila, recursiu

# Implementacions amb vectors

# Implementació de piles amb vectors




- Vegeu `PilaInt.hpp` i `PilaInt.cpp`
- Invariant de la representació:

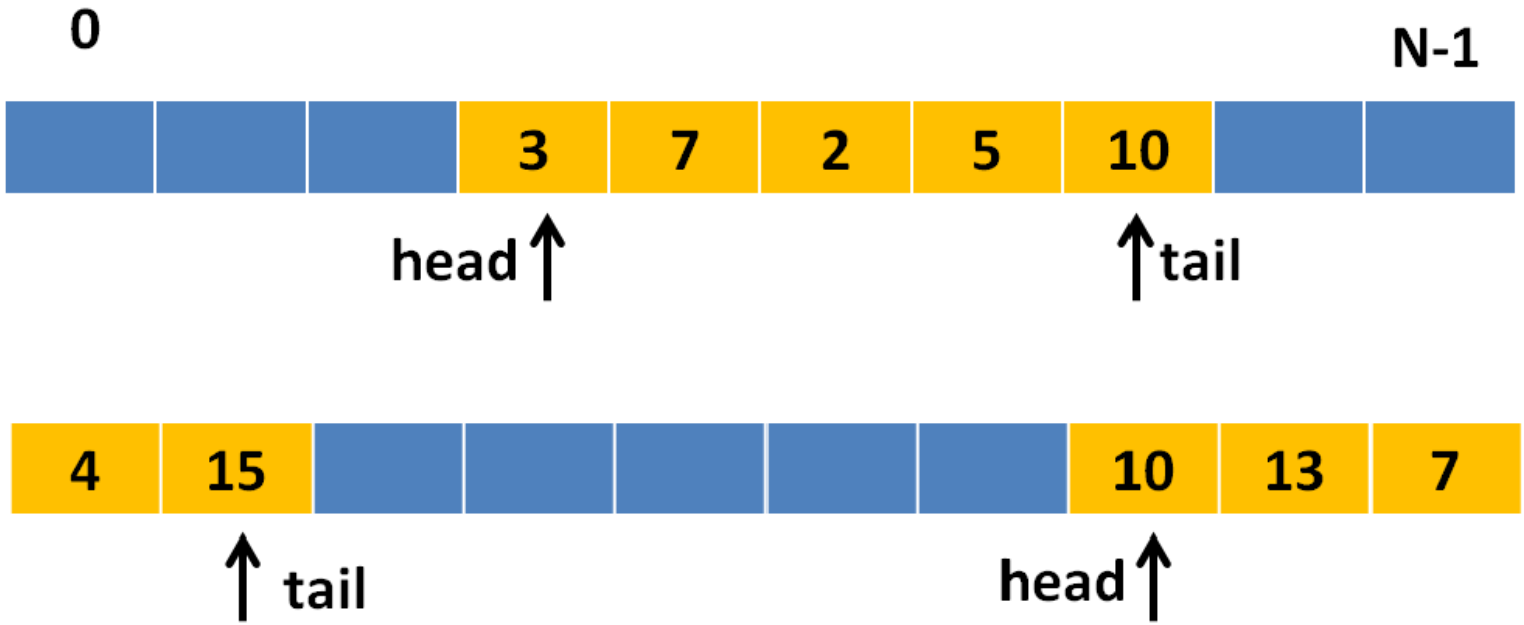
$$0 \leq \texttt{top} \leq \texttt{elems.size()} = \texttt{MAX\_SIZE}$$

- Precondició d'implementació a push:

  "el paràmetre implícit té menys de `MAX_SIZE` elements"

- Alternativament, sense mida màxima fent servir `push_back()`

# Implementació de cues amb vectors: Cua circular




- push: tail = (tail+1)%N; pop: head = (head+1)%N
- longitud de la cua = (tail-head+1)%N
- ambigüetat quan tail = head-1. Buida o plena?