

---

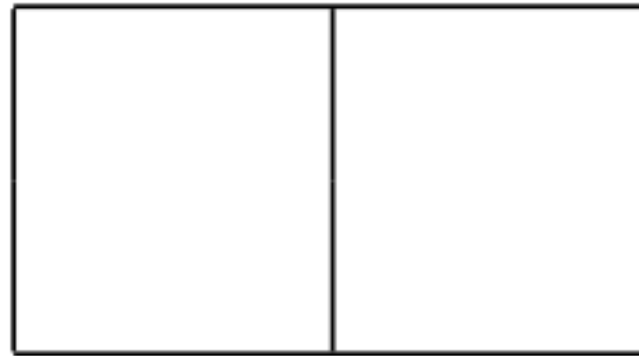
**Doble cua (deque) implementada amb una llista doblement enca-**  
**denada i circular**

---

**X96884\_ca**

La classe doble cua (en anglès **double-ended queue** o abreviadament **deque**) és una classe que permet fer insercions, supressions i consultes en els dos extrems de la cua. És a dir, ha de disposar de les següents operacions (mira el PDF de l'enunciat):

pop ←  
push →  
front



Donada la classe *deque* que permet guardar elements en una doble cua implementada amb una llista doblement encadenada, **sense** fantasma i **circular**, cal implementar els mètodes:

```
void push(T e);  
// Pre: True  
// Post: Insereix un element al davant de la deque.
```

```
void inject (T e);  
// Pre: True  
// Post: Insereix un element al darrera de la deque.
```

```
void pop();  
// Pre: La deque no és buida.  
// Post: Elimina el primer element de la deque.
```

```
void eject ();  
// Pre: La deque no és buida.  
// Post: Elimina l'últim element de la deque.
```

Pots veure exemples de cada mètode en els jocs de prova públics. Cal enviar a jutge.org la següent especificació de la classe *deque* i la implementació dels quatre mètodes anteriors dins del mateix fitxer (la resta de mètodes públics ja estan implementats en el fitxer *main.cc*). Indica dins d'un comentari a la capçalera de cada mètode el seu cost en funció del nombre d'elements *n* de la deque.

```
#include <cstdint>
```

```

using namespace std;
typedef unsigned int nat;

template <typename T>
class deque {
public:
    deque();
    // Pre: True
    // Post: El p.i. és una deque buida.

    deque(const deque &dq);
    // Pre: True
    // Post: El p.i. conté una còpia de dq.

    ~deque();
    // Post: Destruïx els elements del p.i.

    nat size() const;
    // Pre: True
    // Post: Retorna el nombre d'elements de la deque.

    bool empty() const;
    // Pre: True
    // Post: Retorna true si la deque és buida; false en cas contrari.

    T front() const;
    // Pre: La deque no és buida.
    // Post: Retorna el primer element de la deque.

    T rear() const;
    // Pre: La deque no és buida.
    // Post: Retorna l'últim element de la deque.

    void push(T e);
    // Pre: True
    // Post: Insereix un element al davant de la deque.

    void inject(T e);
    // Pre: True
    // Post: Insereix un element al darrera de la deque.

    void pop();
    // Pre: La deque no és buida.
    // Post: Elimina el primer element de la deque.

    void eject();
    // Pre: La deque no és buida.
    // Post: Elimina l'últim element de la deque.

```

```

private:
/* Double-ended queue implementada amb una llista doblement encadenada,
sense fantasma i circular . */
struct node {
    T info;    // Informació del node
    node *seg; // Punter al següent element
    node *ant; // Punter a l'anterior element
};
node *_prim; // Punter al primer element
nat _long;   // Nombre d'elements

    // Aquí va l'especificació dels mètodes privats addicionals
};

// Aquí va la implementació dels mètodes públics i privats addicionals

```

Degut a que jutge.org només permet l'enviament d'un fitxer amb la solució del problema, en el mateix fitxer hi ha d'haver l'especificació de la classe i la implementació dels mètodes que falten (el que normalment estarien separats en els fitxers *.hpp* i *.cpp*).

Per testejar la solució, jutge.org ja té implementats la resta de mètodes de la classe *deque* i un programa principal que crea una deque d'enters i processa comandes que executen els diferents mètodes de la classe.

## Entrada

L'entrada conté varies comandes, una per línia, amb el següent format (e és un enter):

- size
- empty
- front
- rear
- push e
- inject e
- pop
- eject
- mostra
- mostra\_invertida

## Sortida

Per a cada línia d'entrada, escriu una línia amb la comanda d'entrada, el separador ": " i el resultat de la comanda.

El resultat de les comandes *push* i *inject* és el mateix element inserit, el resultat de les comandes *pop* i *eject* és l'element que s'eliminarà. La comanda *mostra* envia tots els elements al canal de sortida entre claudàtors i separats per espais. La comanda *mostra\_invertida* és similar a *mostra* però els envia al revés, començant amb el darrer i acabant amb el primer.

## Observació

Només cal enviar la classe requerida i la implementació dels mètodes que falten. Pots ampliar la classe amb mètodes privats. Segueix estrictament la definició de la classe de l'enunciat. Les comandes *mostra* i *mostra\_invertida* criden als mètodes *pop* i *eject* respectivament. Fins que aquests mètodes no estiguin ben implementats, no es mostraran correctament els elements de la deque per pantalla. Indica dins d'un comentari a la capçalera de cada mètode el seu cost en funció del nombre d'elements  $n$  de la deque.

### Exemple d'entrada 1

```
mostra
mostra_invertida
empty
size
push 7
push 5
push 9
empty
size
front
rear
mostra
mostra_invertida
```

### Exemple d'entrada 2

```
inject 6
inject 4
inject 8
empty
size
front
rear
mostra
mostra_invertida
```

### Exemple d'entrada 3

```
push 7
push 5
push 9
pop
pop
empty
size
front
rear
mostra
mostra_invertida
```

### Exemple d'entrada 4

```
inject 6
inject 4
inject 8
eject
eject
```

### Exemple de sortida 1

```
mostra: []
mostra_invertida: []
empty: true
size: 0
push 7: 7
push 5: 5
push 9: 9
empty: false
size: 3
front: 9
rear: 7
mostra: [9 5 7]
mostra_invertida: [7 5 9]
```

### Exemple de sortida 2

```
inject 6: 6
inject 4: 4
inject 8: 8
empty: false
size: 3
front: 6
rear: 8
mostra: [6 4 8]
mostra_invertida: [8 4 6]
```

### Exemple de sortida 3

```
push 7: 7
push 5: 5
push 9: 9
pop: 9
pop: 5
empty: false
size: 1
front: 7
rear: 7
mostra: [7]
mostra_invertida: [7]
```

```
empty
size
front
rear
mostra
mostra_invertida
```

### Exemple de sortida 4

```
inject 6: 6
inject 4: 4
inject 8: 8
eject: 8
eject: 4
```

```
empty: false
size: 1
front: 6
rear: 6
mostra: [6]
mostra_invertida: [6]
```

### Exemple d'entrada 5

```
push 7
push 5
push 9
pop
eject
inject 6
inject 4
inject 8
eject
pop
empty
size
front
rear
mostra
mostra_invertida
```

### Exemple de sortida 5

```
push 7: 7
push 5: 5
push 9: 9
pop: 9
eject: 7
inject 6: 6
inject 4: 4
inject 8: 8
eject: 8
pop: 5
empty: false
size: 2
front: 6
rear: 4
mostra: [6 4]
mostra_invertida: [4 6]
```

### Exemple d'entrada 6

```
push 7
push 5
push 9
eject
pop
inject 6
inject 4
inject 8
pop
eject
pop
eject
empty
size
mostra
mostra_invertida
```

### Exemple de sortida 6

```
push 7: 7
push 5: 5
push 9: 9
eject: 7
pop: 9
inject 6: 6
inject 4: 4
inject 8: 8
pop: 5
eject: 8
pop: 6
eject: 4
empty: true
size: 0
mostra: []
mostra_invertida: []
```

## Informació del problema

Autoria: Jordi Esteve

Generació: 2026-01-25T17:33:48.928Z

© *Jutge.org*, 2006–2026.  
<https://jutge.org>