

Laboratori 4. Pràctica 2

Estructura de Dades

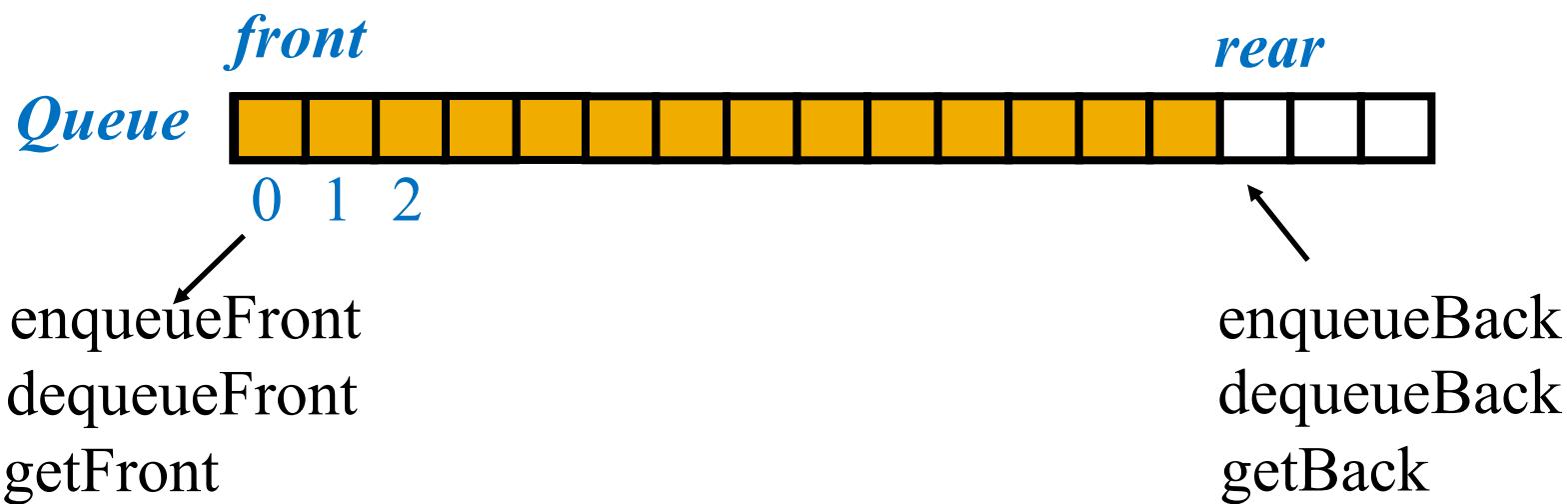
Grau en Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona

Exercici 1

- **Objectiu:**
 - Definir i implementar el **TAD ArrayDeque**
- Definir i implementar sense usar templates
 - La representació del TAD es farà en arrays, usant un *STL vector*
 - Les operacions són les definides a l'enunciat
- Fer un **main.cpp** que usi el TAD ArrayDeque i que contingui els dos casos de prova

DeQue

- Un DeQue és una cua que permet inserir i eliminar pels dos extrems de la cua



ArrayDeQue

- La implementació es farà usant STL vector
- Sobre el codi ...
 - **No useu les operacions del TAD vector, accediu als elements com un array**
 - Optimitzeu les operacions del TAD amb el menor cost computacional que sigui possible
 - Comenteu el codi (les operacions del TAD i el main)
 - Poseu noms entenedors a les variables i als mètodes
 - Identeu el codi adequadament

Exercici 2

- **Objectiu**
 - Definir i implementar el TAD **LinkedDeque**
- **Definir i implementar** (opcional usar templates)
 - La representació del TAD es farà en una estructura amb encadenaments dobles
 - Les operacions són les definides a l'enunciat
- Fer un **main.cpp** que usi el TAD LinkedDeque i que contingui els dos casos de prova de l'exercici 1
 - S'han de controlar les excepcions

Exercici 3

- **Objectiu:**
 - Usar el **TAD LinkedDeque**
- Es demana resoldre un problema d'una cua de pacients en un hospital utilitzant aquest TAD
 - Fer el menú del programa
 - Implementar la càrrega dels fitxers de les entrades a la cua de pacients
 - Implementar les altres opcions del menú
- S'han de controlar les excepcions



Descripció del DeQue amb encadenaments dobles

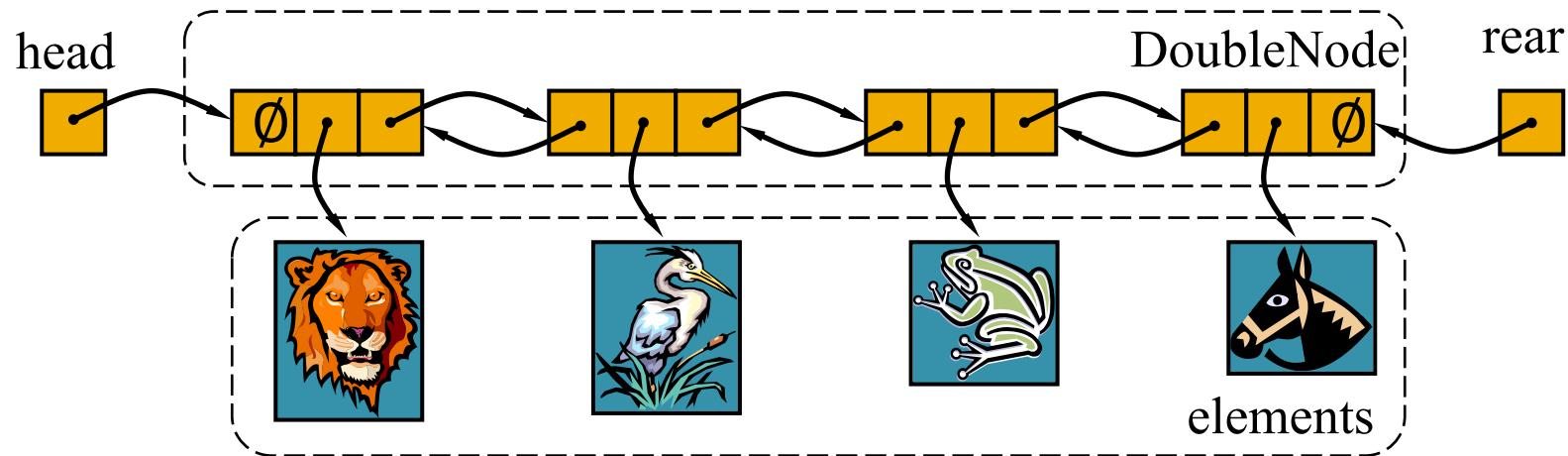
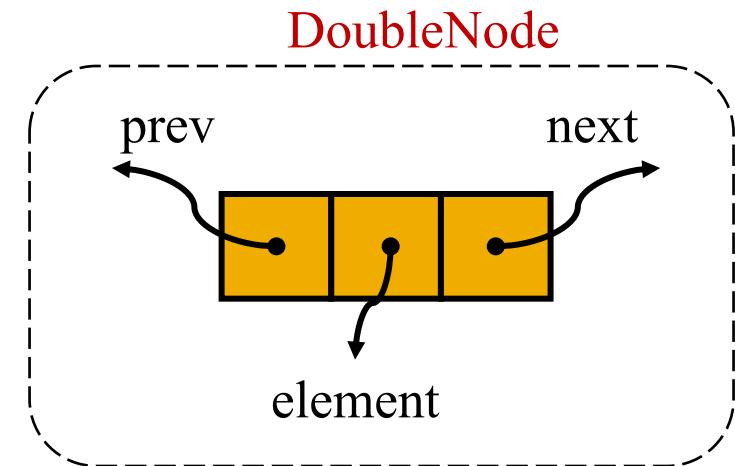
Implementació de la LinkedDeque

La LinkedDeque es podria implementar amb encadenaments simples o amb encadenaments dobles.

En aquesta pràctica hem decidit implementar el TAD LinkedDeque amb encadenaments dobles perquè és molt eficient, en termes de cost computacional, la implementació d'alguns mètodes, com per exemple el mètode dequeueBack

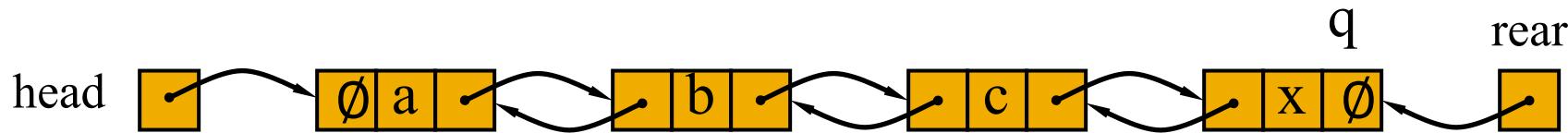
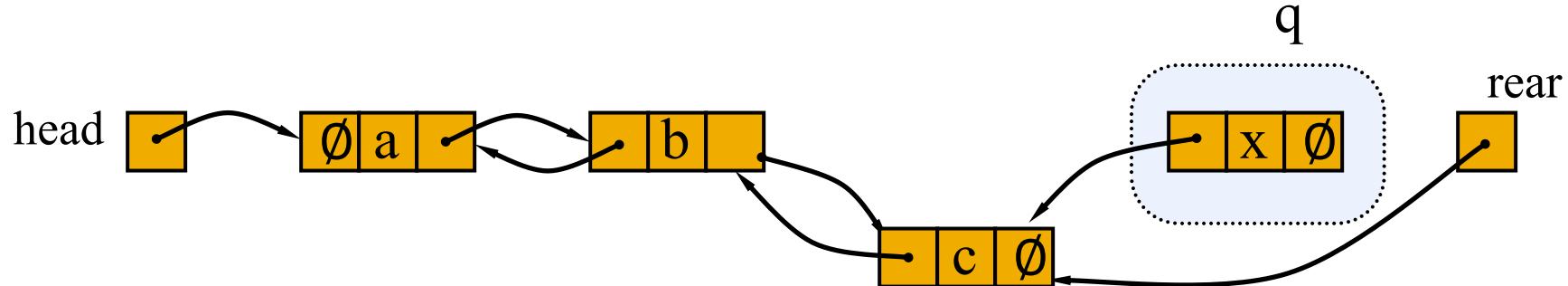
LinkedDeque amb encadenaments dobles

- Els nodes **DoubleNode** guarden:
 - element
 - link al DoubleNode anterior (previous)
 - link al següent DoubleNode (next)



Exemple EnqueueBack

- Visualització de l'operació **enqueueBack(x)**, la qual inserta l'element x al final de la LinkedDeque
- Noteu que primer s'enllacen els punters **previous** i **next** del nou node q i al final el punter **next** del node anterior i el rear

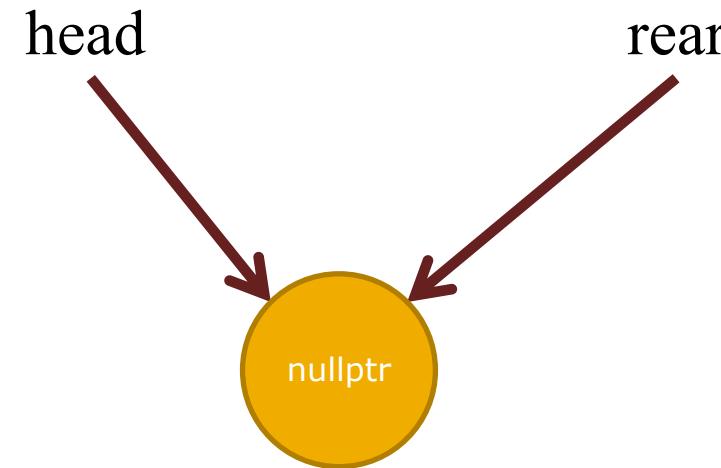




LinkedDeque amb encadenaments dobles sense sentinelles

LinkedDeque doble sense sentinelles

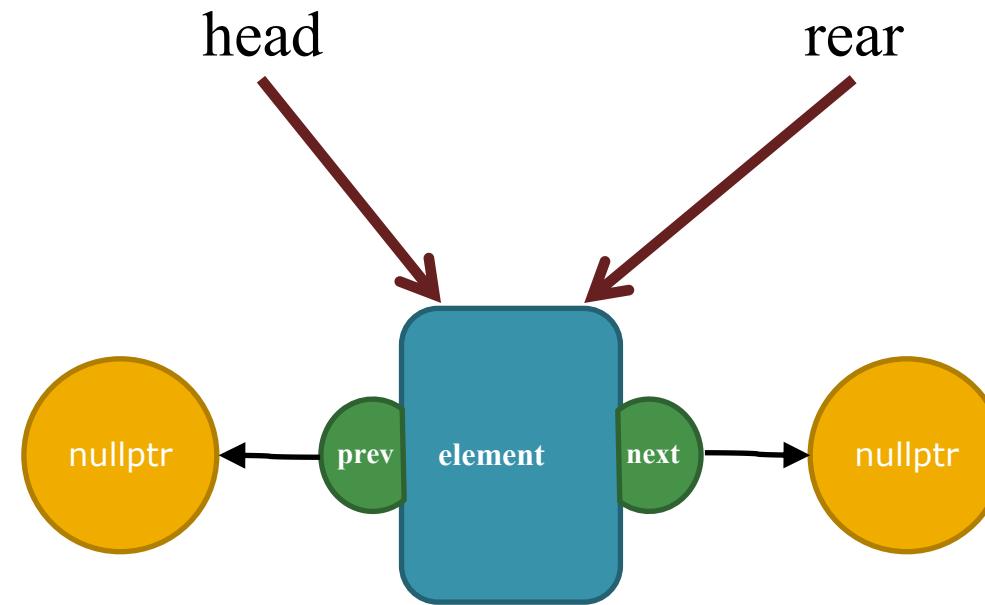
LinkedDeque buida



Noteu que quan la LinkedDeque està buida, els punters head i rear apunten a nullptr

LinkedDeque doble sense sentinelles

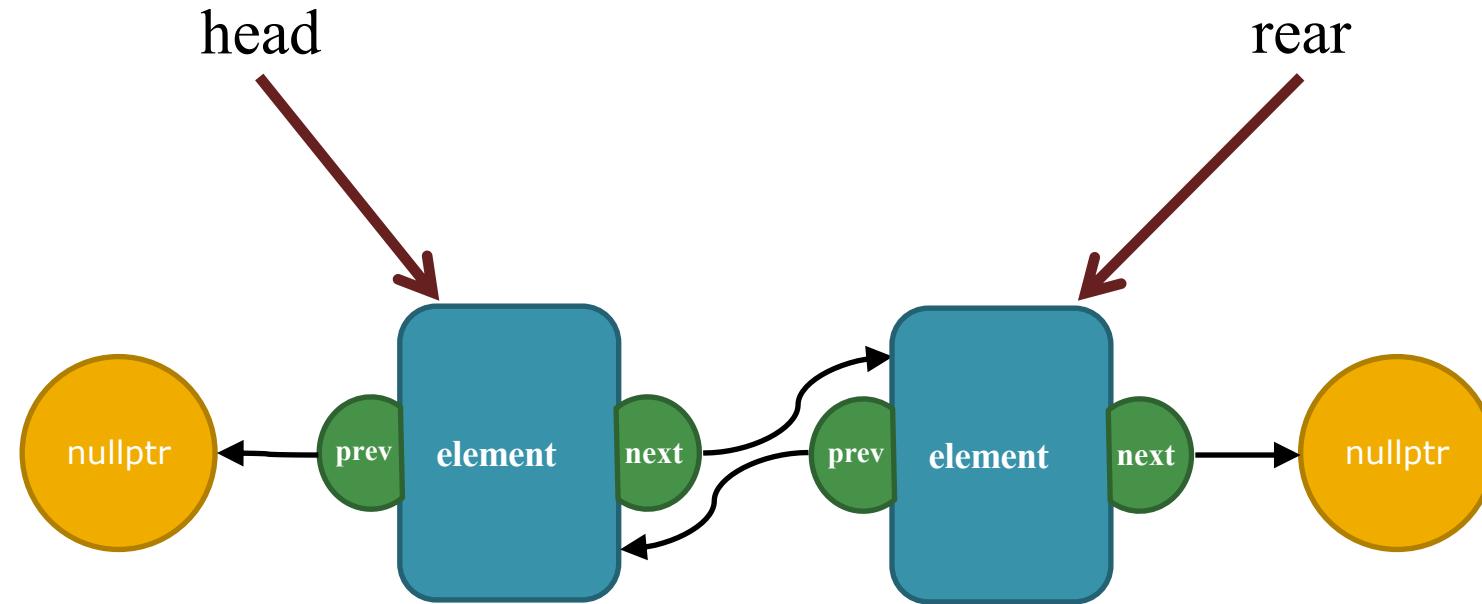
Fem un enqueue d'un element



Quan s'encua un primer element a la LinkedDeque, és necessari que tant head com rear apuntin al nou element. Aquest nou element no té anterior i no té successor, per això els dos punters estan a nullptr.

LinkedDeque doble sense sentinelles

Encuem un segon element



Quan s'insereix un segon element a la LinkedDeque, és necessari **que head o rear** apunti al nou element. Serà un o un altre dependent de quin tipus d'enqueue s'ha fet (Front o Back del que ja hi havia com primer element). Ara sí que farà falta assignar el **next o el previous** dels dos elements que hi ha a la LinkedDeque.



Descripció de Templates en C++

Templates

- Els templates són útils per definir un únic codi en un conjunt de funcions relacionades (**function template**) o un conjunt de classes relacionades (**class template**)

```
template <class Type>
Type larger(Type x, Type y)
{
    if (x >= y)
        return x;
    else
        return y;
}
cout << larger(5, 6) << endl
cout << larger ('A', 'B')<<endl;
```

Exemple funció template

```
#include <iostream>
using namespace std;

template <class T>
T sum (T a, T b){
    T result;
    result = a + b;
    return result;
}
```

```
int main () {
    int i=5, j=6, k;
    double f=2.0, g=0.5, h;
    k=sum<int>(i,j);
    h=sum<double>(f,g);
    cout << k << '\n';
    cout << h << '\n';
    return 0;
}
```

Exemple Class Template

```
template<class ItemType>
class GList
{
public:
    bool IsEmpty() const;
    bool IsFull() const;
    int Length() const;
    void Insert( /* in */ ItemType item );
    void Delete( /* in */ ItemType item );
    bool IsPresent( /* in */ ItemType item ) const;
    void SelSort();
    void Print() const;
    GList();                                // Constructor
private:
    int      length;
    ItemType data[MAX_LENGTH];
};
```

Definició del tipus genèric

Template parameter

Instanciant un Class Template

Es poden crear llistes de diferents tipus

// Client code

```
GList<int> list1;  
GList<float> list2;  
GList<string> list3;  
  
list1.Insert(356);  
list2.Insert(84.375);  
list3.Insert("Muffler bolt");
```

template argument

El compilador genera tres llistes diferents

```
GList_int list1;  
GList_float list2;  
GList_string list3;
```

Amb *Templates* cal linkar el codi de manera diferent. Hi ha vàries maneres:

La més simple és posar el .h i la seva implementació juntes (no feu el .cpp, cal incloure el codi del .cpp al .h)

Exemple de Template

```
#ifndef COMPLEXE_H #define COMPLEXE_H
template <class T>
class Complexe{
private:
    T v_real;
    T v_imaginari;
public:
    Complexe<T>(T r=0, T i=0);
    T real() const;
    T imaginari() const;
};
template <class T> Complexe<T>::Complexe(T r , T i) : v_real(r), v_imaginari(i){ }
template <class T> T Complexe<T>::real() const { return v_real;}
template <class T> T Complexe<T>::imaginari() const { return v_imaginari; }
#endif
```

Ús del Template

```
#include "Complexe.h" using namespace std;
int main() {
    Complexe<int> c(2.5, 3.2);
    cout << "Part real " << c.real() << endl;
    cout << "Part imaginaria " << c.imaginari() << endl;

    Complexe<int> c2(2.5, 3.5);
    cout << "Part real " << c2.real() << endl;
    cout << "Part imaginaria " << c2.imaginari() << endl;

    Complexe<int> c3(1, 2);
    cout << "Part real " << c3.real() << endl;
    cout << "Part imaginaria " << c3.imaginari() << endl;

    Complexe<double> c4;
    cout << "Part real " << c4.real() << endl;
    cout << "Part imaginaria " << c4.imaginari() << endl;
    return 0;
}
```

Links interessants

- Al campus virtual trobareu:
 - un document anomenat ***Exemple Templates Complex*** que implementa l'exemple de Complex fet amb *class template*
 - Obriu el projecte a NetBeans i testejeu el codi
- Links externs:
 - <http://www.cplusplus.com/doc/oldtutorial/templates/>
 - <https://youtu.be/scUUp4-7Cn4>