

# Sessió 9. Mètodes en estructures dinàmiques

**Estructura de Dades  
Curs 2020-2021**

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

# Contingut

Problema 1 – Linked Deque (enqueueIfNotContained)

Problema 2 – Linked Deque (enqueueAt)

Problema 3 – Linked Deque (enqueueDecimalNumber)

Problema 4 – OrderedList (remove)

Problema 5 – BSTree (sum)

# Problema 1 LinkedDeque

```
void enqueueIfNotContained (int & key);
```

Aquesta funció comprova que el valor no està en el deque, i si és així, l'afegeix a continuació del front del deque. Si no hi ha elements al deque, llença l'excepció que no es pot afegir. Si es troba la key al deque, aquest mètode deixa el deque intacte.

NOTA: No podeu usar les operacions del TAD LinkedDeque

```
LinkedDeque q;
int elements[5] = {1,2,3,4,5};

for (int i = 0; i < 5; i++) { q.enqueueFront(elements[i]);
q.print();}

int elem = 5; q.enqueueIfNotContained(elem); q.print();
elem = 6; q.enqueueIfNotContained(elem); q.print();
elem = 7; q.enqueueIfNotContained(elem); q.print();
elem = 10; q.enqueueBack(elem); q.print();
q.dequeueBack(); q.print();
q.dequeueFront(); q.print();
```

SORTIDA PER PANTALLA

```
[1]
[2,1]
[3,2,1]
[4,3,2,1]
[5,4,3,2,1]
[5,4,3,2,1]
[5,6,4,3,2,1]
[5,7,6,4,3,2,1]
[5,7,6,4,3,2,1,10]
[5,7,6,4,3,2,1]
[7,6,4,3,2,1]
```

```
class LinkedDeque {
public:
    LinkedDeque();
    virtual ~LinkedDeque();
    LinkedDeque(const LinkedDeque& q);
    void enqueueFront(int& f);
    void enqueueBack(int& f);
    void dequeueFront();
    void dequeueBack();
    bool isEmpty();
    void print();
    const int& getFront();
    const int& getBack();
    int size();

private:
    int _size;
    DoubleNode* _front;
    DoubleNode* _rear;
};
```

# Problema 1 (Solució)

```
void LinkedDeque::enqueueIfNotContained(int &key)
{
    DoubleNode * node = _front;

    bool trobat = false;

    while(node!=nullptr && !trobat )
    {
        if (node->getElement() == key) {
            trobat = true;
        }
        else node = node->getNext();
    }

    if (!trobat)
    {
        if (_front != nullptr)
        {
            DoubleNode* temp = new DoubleNode(key);
            temp->setPrevious(_front);
            temp->setNext(_front->getNext());
            _front->getNext()->setPrevious(temp);
            _front->setNext(temp);
            _size++;
        }
        else throw string("Queue without enough elements");
    }
}
```

# Problema 2 LinkedDeque

```
void enqueueAt(int& key, int pos);
```

Aquesta funció afegeix un element (*key*) a la posició (*pos*) indicada del deque, tenint en compte que el front del deque és la posició 0 i els elements següents tenen posicions successives. Si *pos* és més gran que el nombre d'elements a la cua, s'afegirà l'element al final de tot.

**NOTA:** No podeu usar les operacions del TAD LinkedDeque

```
LinkedDeque q;
int elements[5] = {1,2,3,4,5};

for (int i = 0; i < 5; i++) { q.enqueueBack(elements[i]); q.print();}

int elem = 5; q.enqueueAt(elem, 2); q.print();
elem = 6; q.enqueueAt(elem, 7); q.print();
elem = 7; q.enqueueAt(elem, 0); q.print();
```

## SORTIDA PER PANTALLA

```
[1]
[1,2]
[1,2,3]
[1,2,3,4]
[1,2,3,4,5]
enqueueAt 5 in position 2
[1,2,5,3,4,5]
enqueueAt 6 in position 7
[1,2,5,3,4,5,6]
enqueueAt 7 in position 0
[7,1,2,5,3,4,5,6]
```

```
class LinkedDeque {
public:
    LinkedDeque();
    virtual ~LinkedDeque();
    LinkedDeque(const LinkedDeque& q);
    void enqueueFront(int& f);
    void enqueueBack(int& f);
    void dequeueFront();
    void dequeueBack();
    bool isEmpty();
    void print();
    const int& getFront();
    const int& getBack();
    int size();

private:
    int _size;
    DoubleNode* _front;
    DoubleNode* _rear;
};
```

```

void LinkedDeque::enqueueAt(int & key, int pos)
{
    DoubleNode * node = _front;
    while(node!=nullptr && pos >=1 )      {
        node = node->getNext();
        pos--;
    }
    DoubleNode * temp = new DoubleNode(key);
    if (node == _front ) // case of first element
    {   temp->setNext(_front);
        if (_front != nullptr )temp->getNext()->setPrevious(temp);
        else _rear = temp;
        _front = temp;
    }
    else // case of next elements
    {   temp->setNext(node);
        if (node != nullptr)// we are at the end
        {
            temp->setPrevious(node->getPrevious());
            node->setPrevious(temp);
        }
        else {
            temp->setPrevious(_rear);
            _rear->setNext(temp);
            _rear = temp;
        }
        temp->getPrevious()->setNext(temp);
    }
    _size++;
}

```

## Problema 2 (SOLUCIÓN)

# Problema 3 LinkedDeque

```
void enqueueDecimalNumber();
```

Aquesta funció calcula el nombre decimal que formen els elements de la cua i afegeix un nou node al final de la cua amb aquest nombre decimal. Llença una excepció si algun element no està entre 1 i 9.

NOTA: No podeu usar les operacions del TAD LinkedDeque

```
LinkedDeque q;  
int elements[5] = {1,2,3,2,4};  
  
for (int i = 0; i < 5; i++) q.enqueueBack(elements[i]);  
q.print();  
q.enqueueDecimalNumber(); q.print();
```

SORTIDA PER PANTALLA

```
[1,2,3,2,4]  
[1,2,3,2,4,12324]
```

```
class LinkedDeque {  
public:  
    LinkedDeque();  
    virtual ~LinkedDeque();  
    LinkedDeque(const LinkedDeque& q);  
    void enqueueFront(int& f);  
    void enqueueBack(int& f);  
    void dequeueFront();  
    void dequeueBack();  
    bool isEmpty();  
    void print();  
    const int& getFront();  
    const int& getBack();  
    int size();  
  
private:  
    int _size;  
    DoubleNode* _front;  
    DoubleNode* _rear;  
};
```

```
void LinkedDeque::enqueueDecimalNumber()
{
    int number = 0;
    int exp = this->size()-1;
    DoubleNode* node = this->_front;

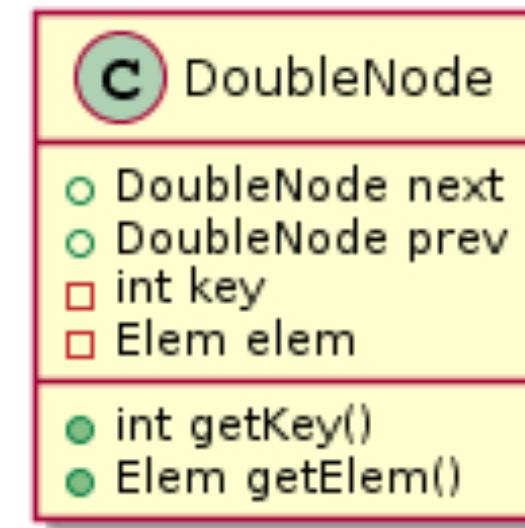
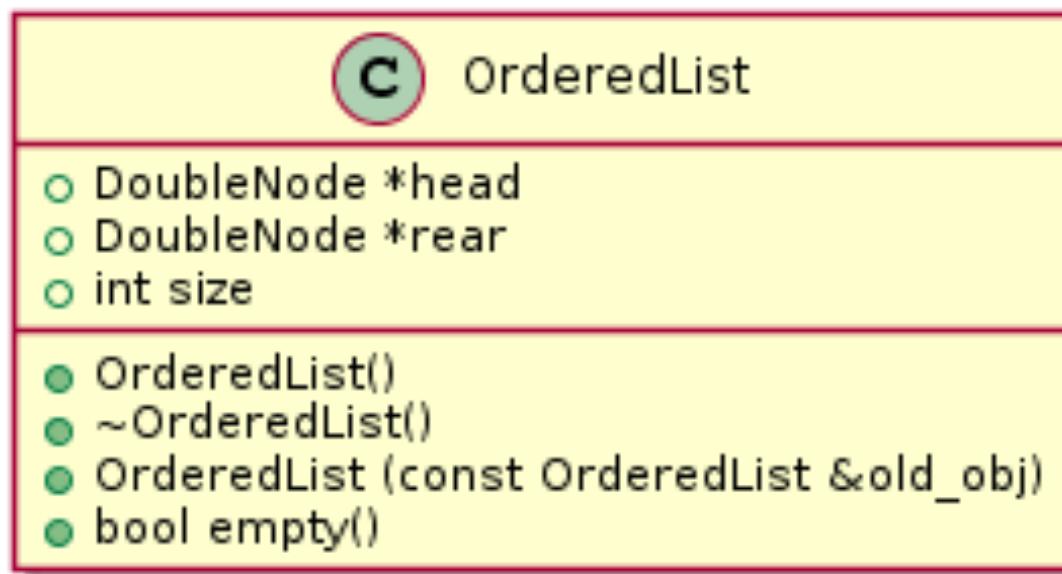
    while(node != 0)
    {   // A l'enunciat dels alumnes assumim que sempre estan entre 1 i 9
        if (node->getElement() > 9){
            throw std::string("Digit més gran que 9!");
        }
        else{
            number = number + node->getElement() * pow (10, exp);
            node = node->getNext();
            exp--;
        }
    }
    DoubleNode * nou_node = new DoubleNode(number);
    _rear->setNext(nou_node);
    nou_node->setPrevious(_rear);
    _rear = nou_node;
    _size++;
}
```

## Problema 3 (SOLUCIÓ)

# Problema 4 Ordered List

Donada una llista ordenada amb la següent especificació, implementa el mètode `remove (int key, Elem elem)` que elimina un element a la llista ordenada si té la mateixa key i el mateix element.

Penseu que un l'element `(key, value) =(2,5)` estarà per davant de l'element `(3,1)` a la llista.



```
28 void OrderedList::delete(int key, Elem ele) {
29     if(this->size==0) {
30         throw "Empty List!";
31     }
32     DoubleNode * it = head;
33     bool found = false;
34     while(!found && it != nullptr) {
35         if(it->getElem()==elem && it->getKey()==key) {
36             found = true;
37             if(it->prev != nullptr) it->prev->next=it->next;
38             if(it->next != nullptr) it->next->prev=it->prev;
39             delete it;
40         }
41     else {
42         it = it->next;
43     }
44 }
45 if(!found) {
46     throw "It does not exist the element with the given key!";
47 }
48 }
```

## Problema 4 (SOLUCIO 1)

Penseu que ocorre si  
esborro el head o rear?

# Problema 4 (SOLUCIO 2)

```
27 void OrderedList::delete(int key, Elem ele) {
28     if(this->size==0) { throw "Empty List!";      }
29     else{
30         DoubleNode * it = head;
31         bool found = false, continuar = true;
32         while(!found && continuar) {
33             if(it->getElem()==elem && it->getKey()==key) {
34                 found = true;
35                 if (it->prev != nullptr) it->prev->next=it->next;
36                 else {
37                     head = it->next;
38                     it->next->prev = nullptr;
39                 }
40                 if (it->next != nullptr) it->next->prev=it->prev;
41                 else{
42                     rear = it->prev;
43                     it->prev->next = nullptr;
44                 }
45             }else if(it->getElem()<=elem && it->getKey()<=key){
46                 continuar = false
47             } else{
48                 it = it->next;
49                 continuar = it != nullptr;
50             }
51         }
52     }
53 }
```

# Problema 5

Implementeu recursivament un mètode anomenat `sum()` de la classe `BSTree` (un arbre de cerca binària). Aquest mètode retorna la suma de tots els elements de l'arbre binari. Suposeu que l'arbre és d'enters i que el `BSTree` té definit una funció `root()` que retorna el `root_node` que és de tipus `NodeTree *`.

Els `NodeTree` tenen un atribut `_right`, un atribut `_left` i un atribut `_element` privats que s'accedeixen mitjançant les funcions `right()`, `left()`, `getElement()`, i també teniu una funció `isExternal()` per retornar si un node és fulla o no.

```
template <class Element>
int BSTree<Element>::sum() {
    Aquí el vostre codi
}
```

Definiu aquí les funcions auxiliars que necessiteu

# Problema 5 (Solució 1)

```
template <class Element>
int BSTree<Element>::sum() {
    return this->sumElements(this->root());
}
```

Solució no  
optimitzada !!

```
template <class Element>
int BSTree<Element>::sumElements(NodeTree<Element> * node)
{
    if (node->isExternal()) return node->element();
    else
    {
        int d = 0, r = 0;
        if (node->left() != nullptr) r = this->sumElements(node->left());
        if (node->right() != nullptr) d = this->sumElements(node->right());
        return (d + r + node->element());
    }
}
```

# Problema 5 (Solució 2)

```
template <class Element>

int BSTree<Element>::sum() {
    return this->sumElements(this->root());
}

template <class Element>
int BSTree<Element>::sumElements(Position<Element> * node)
{
    if (node == nullptr) return 0;
    else{
        int l = this->sumElements(node->left());
        int r = this->sumElements(node->right());
        return l + r + node->getElement();
    }
}
```

Solució no  
optimitzada !!