UNIVERSITAT DE BARCELONA

Tema 3 **Estructures Lineals**
Sessió Teo 5

Maria Salamó Llorente

Estructura de Dades

Grau en Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

UNIVERSITAT DE BARCELONA

Contingut

UNIVERSITAT DE BARCELONA

Contingut

Sessió Teoria 3 (Teo 3)

3.1 Introducció a les estructures lineals

3.2 TAD Pila i TAD Cua (representació estàtica)

Sessió Teoria 4 (Teo 4)

3.3 Concepte de TAD

3.4 TAD Pila i TAD Cua (representació dinàmica)

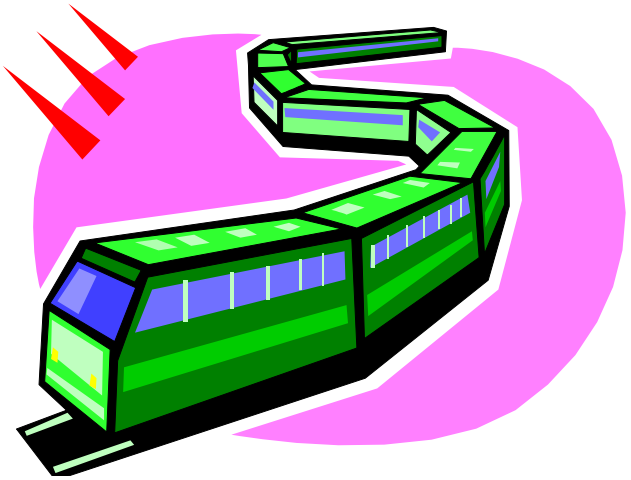
Sessió Teoria 5 (Teo 5)

3.5 TAD Llista

3.6 TAD Cua prioritària (es veurà al Tema 5)

3.5 TAD Llista

- Per posició
- Per punt d'interès



Definició de Llista

- Les **piles** permeten manipular l'element que hi ha a dalt de tot de la pila. És a dir, només el darrer element que s'ha inserit a la seqüència
- Les **cues** permeten manipular el primer i el darrer element de la seqüència, les veurem amb calma al següent punt del tema

- Les **llistes** són una generalització de les piles i les cues
  - Permeten manipular qualsevol element de la seqüència.
  - Quan diem manipular, estem referint-nos a inserir, modificar o consultar.

Definició de Llista (cont.)

- Per abstraure i unificar les diferents maneres de guardar elements in les diferents implementacions de la llista, introduïm un tipus de dades que abstrau el concepte de posició relativa o lloc d'un element dins de la llista. Aquest tipus de dades és el TAD anomenat **POSICIO**.
- Quan parlem de posició relativa és que se sap quina posició ocupa un element a la seqüència respecte els elements que té com a veïns, element al darrera i pel davant de la seqüència, però no es coneix la seva posició global.

UNIVERSITAT DE BARCELONA

TAD Posició

- El TAD **Posició** modela el concepte de lloc en una estructura de dades on es guarda un conjunt d'elements
- És una vista unificada de les diferents formes de guardar les dades, com és:
  - una cel.la d'un vector
  - un node d'una representació encadenada
- Un únic mètode:
  - objecte p.element(): retorna l'element (objecte) de la posició
  - En C++ és convenient implementar això com *p

TAD Posició explicació

- Una POSICIO es defineix com un tipus abstracte de dades que està associat a un contenidor particular i el qual suporta una única funció, que es diu element().
- C++ pot sobrecarregar operadors i ens permet una forma elegant d'expresar l'operació element().
  - Concretament, es sobrecarrega l'operador *deferencing operator (*)*, de tal manera que, donada una posició variable p, l'element pot ser accedit amb *p, enlloc de p.element(). Així podem usar-lo per dues coses: accedir i modificar el valor de l'element.

TAD Posició explicació

Tot i que la posició d'un objecte és molt útil, ens interessa també navegar per tot el contenidor, avançant a la posició següent.

Aquest tipus d'objectes s'anomenen **iteradors**.

Un iterador és una extensió del TAD posició.

Té habilitat d'accedir al node, permet següent i anterior. Està explicat a la transparència següent.

UNIVERSITAT DE BARCELONA

TAD Posició

- Un iterador és una extensió del TAD posició
- Mètodes:
  - **p.element()**: consultar l'element
    - sobrecarregar operador (*)
  - **p.next()**: avançar a la posició següent
    - sobrecarregar operador (++)
  - retrocedir (sobrecarregar operador (--)
- Assumim que els contenidors implementen:
  - **begin()**: per retornar la primera posició del contenidor
  - **end()**: per retornar la darrera posició del contenidor

UNIVERSITAT DE BARCELONA

TAD LLista

- El TAD LLista (LinkedList) modela una seqüència de posicions que guarden objectes arbitraris
- Estableix una relació d'anterior/posterior entre posicions
- Mètodes genèrics:
  - size(), empty() (no són necessaris)

- Iteradors:
  - begin(), end()
- Mètodes modificadors:
  - insertFront(e), insertBack(e)
  - removeFront(), removeBack()
- Modificadors basats en iteradors:
  - insert(p, e)
  - remove(p)

UNIVERSITAT DE BARCELONA

TAD Llista (explicació)

- A la transparència anterior teniu definides les operacions bàsiques del TAD Llista.
- Noteu que hi ha dues operacions per iterar begin i end.
- Dues operacions per manipular el principi i final de la seqüència i són les que permeten fer les mateixes operacions que fan les piles i les llistes.
- Dues operacions que estan basades en iteradors, permeten inserir un element e a la posició p, o esborrar l'element que hi ha a la posició p de la seqüència.

UNIVERSITAT DE BARCELONA

TAD Llista

Hi ha dos tipus de llistes:

- **Llistes per posició** → permeten accedir directament a una posició donada externament de la seqüència i fer operacions en aquesta posició (ja sigui inserir, esborrar o consultar).
- **Llistes per punt d'interès**→ tenen una posició interna que és la posició de la seqüència on es fan les operacions. La posició mai ve donada externament, sempre serà la interna.

A continuació teniu definides les operacions més importants que són necessàries de la definició del TAD Llista vist anteriorment per cada tipus de llista.

UNIVERSITAT DE BARCELONA

TAD Llista per posició

- Mètodes genèrics:
  - size(), empty()
- Mètodes consultors:
  - begin(), end()
  - back(p), next(p)
- Mètodes modificadors:
  - insert(p, e)
  - insertBefore(p, e), insertAfter(p, e),
  - insertFront(e), insertBack(e)
  - remove(p)

TAD Llista per punt d'interès

- Mètodes genèrics:
  - size(), empty()
- Mètodes consultors:
  - get () retorna Object del punt d'interès
- Mètodes modificadors:
  - modify(e) : modifica l'element del punt d'interès
  - insert(e) : inserta element al punt d'interès
  - remove() retorna Object
- **Mètodes del punt d'interès:**
  - begin(): es sitúa a l'inici
  - next(): avança la posició
  - end() retorna booleà indicant si ha arribat al final

Implementació TAD Llista

Per implementar el TAD llista hi ha vàries possibilitats:

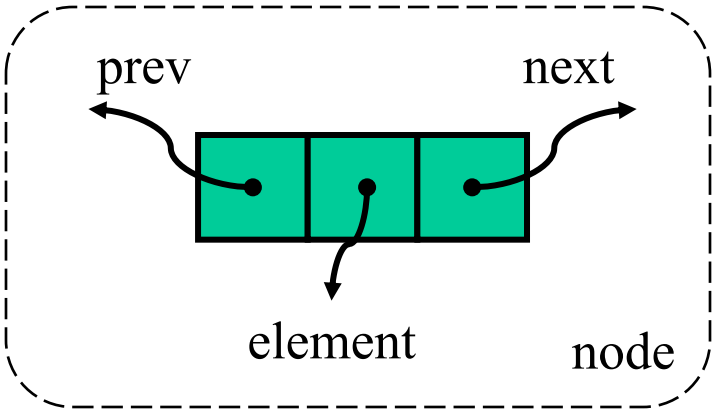
1. Implementació amb enllaços simples
2. Implementació amb enllaços dobles
3. Implementació amb enllaços dobles i circular

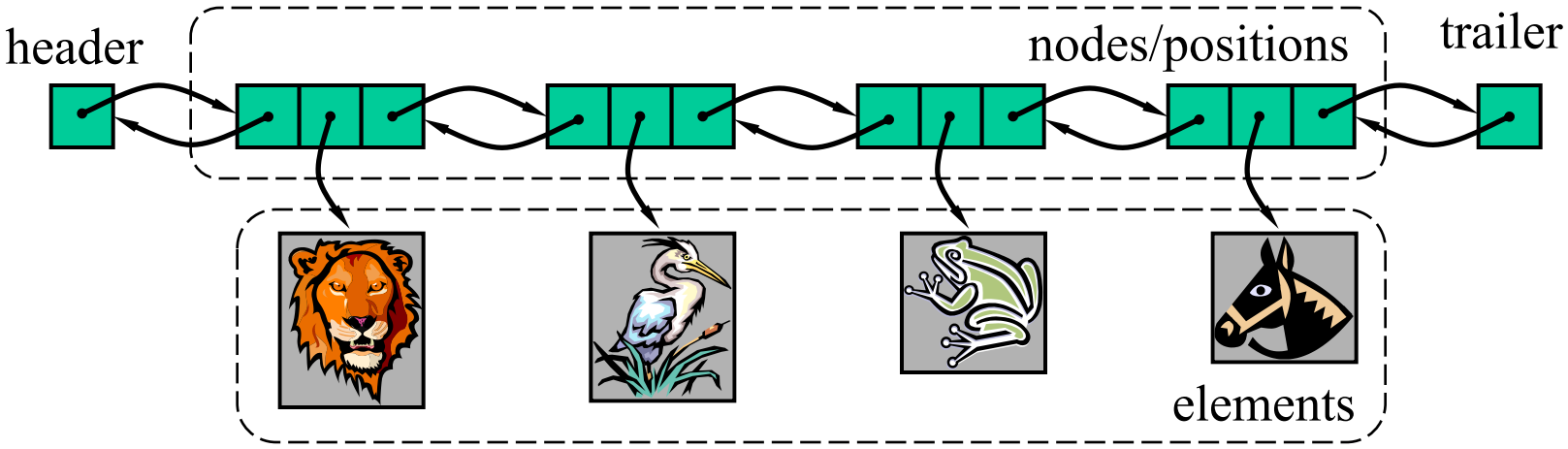
No considerem la implementació amb array perquè la gestió de memòria no és òptima.

En aquest curs treballarem sobre la implementació amb enllaços dobles. A continuació la teniu definida.

Llista amb encadenaments dobles

- Llista amb encadenaments dobles és la implementació natural del TAD llista (l'anomenarem LinkedList)
- Els nodes implementen la Posició i guarden:
  - element
  - link al node anterior (prev)
  - link al següent node (next)



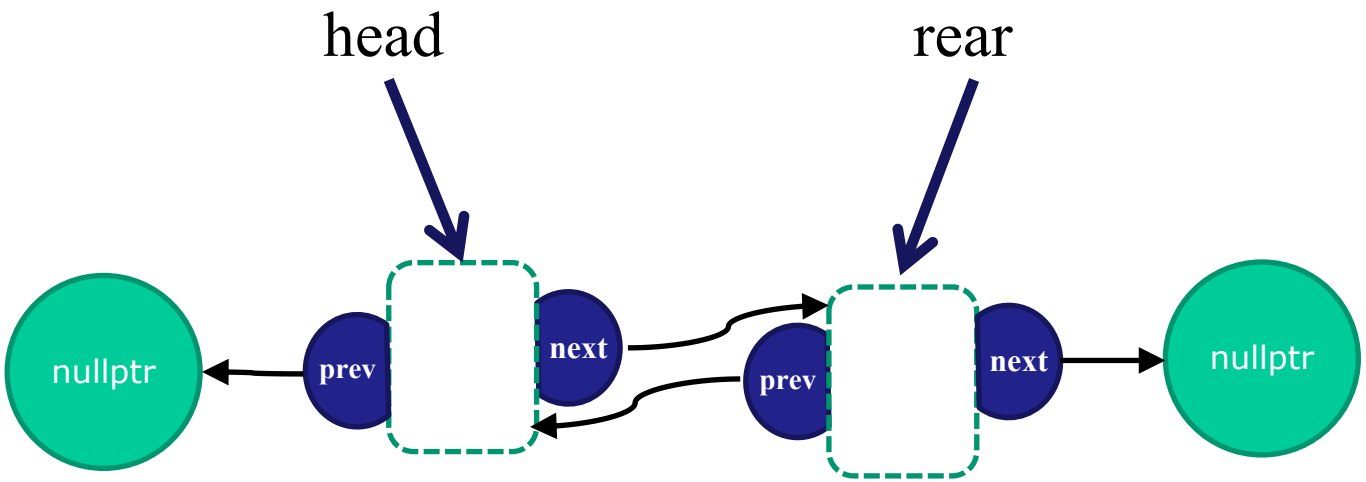


Problemàtica i solució

- **Problema**: A la llista amb encadenaments dobles sense sentinelles, s'han de considerar diversos casos a l'hora d'inserir elements, si és el primer element, si és el segon element o si és el tercer en endavant.
- **Solució**: Es pot fer una implementació que incorpori dos elements sense contingut, aquests elements s'anomenen sentinelles.
- **Per a que serveixen els sentinelles?**
  - Per definir un punt fixe d'inici i final de seqüència.
  - Es defineixen al constructor dos elements (sense contingut) que s'enllacen com a primer i darrer element de la seqüència. El head apunta al primer sentinella, el rear al darrer sentinella. I mai es mouran aquests dos punters d'aquests dos elements.
  - A partir d'aquí, tots els nous elements que s'incorporaran a la llista hauran d'estar entre el primer sentinella i el darrer sentinella.

LinkedList doble amb sentinelles

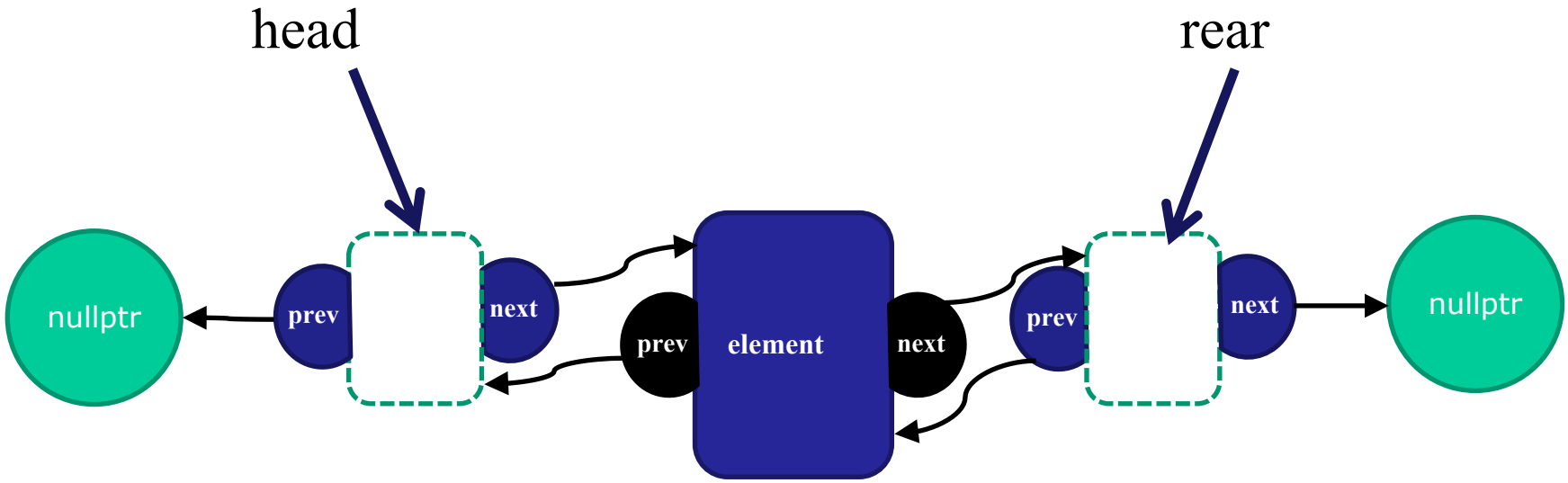
Llista buida



Al construir la llista, es demanarà espai pels dos sentinelles. El punter head apuntarà al primer i el punter rear al segon sentinella. Els next del primer sentinella apunta al segon i el prev del segon sentinella apunta al primer sentinella. En qualsevol operació del TAD LinkedList, els puntes head i rear estaran sempre en aquestes posicions (sentinelles). **Sabem que la llista està buida quan els sentinelles s'apunten entre ells.**

LinkedList doble amb sentinelles

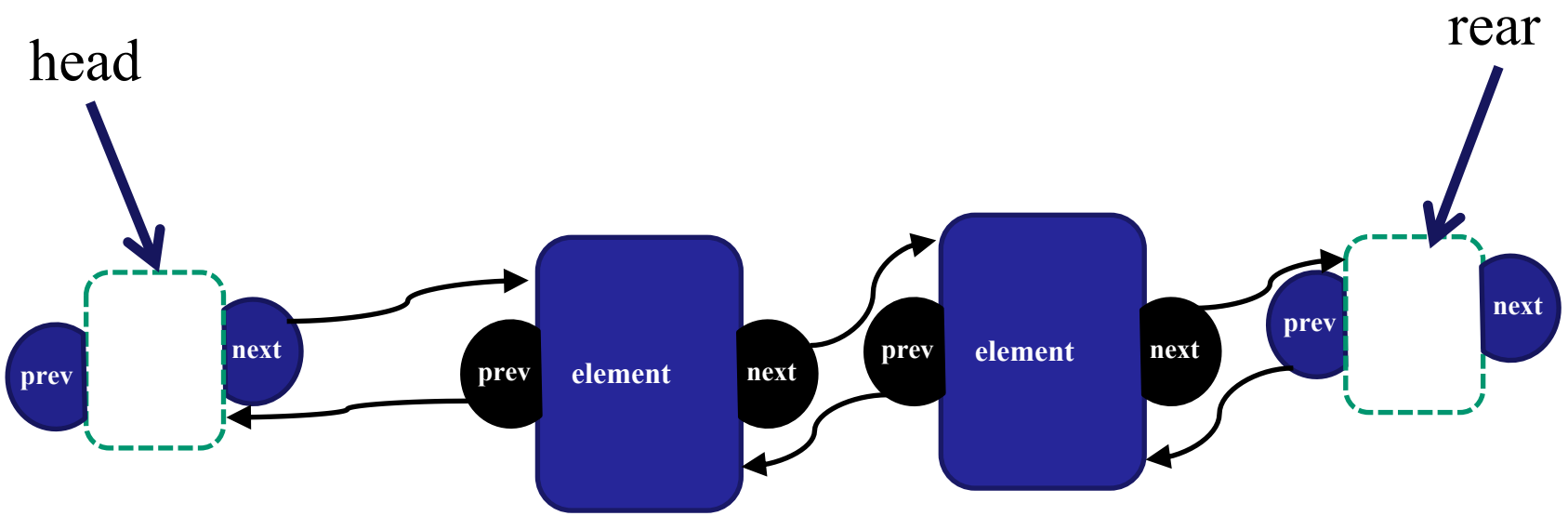
Inserim un element



Per incorporar un primer element a la llista, s'ha de fer abans del rear o darrera del head. Noteu que els punters head i rear no es mouran de la posició que tenen establerta des de la creació de la llista. Ja no cal considerar el cas d'inserir el primer o el segon element a la llista.
insertFront insereix darrera del primer sentinella
insertBack insereix davant del darrer sentinella

LinkedList doble amb sentinelles

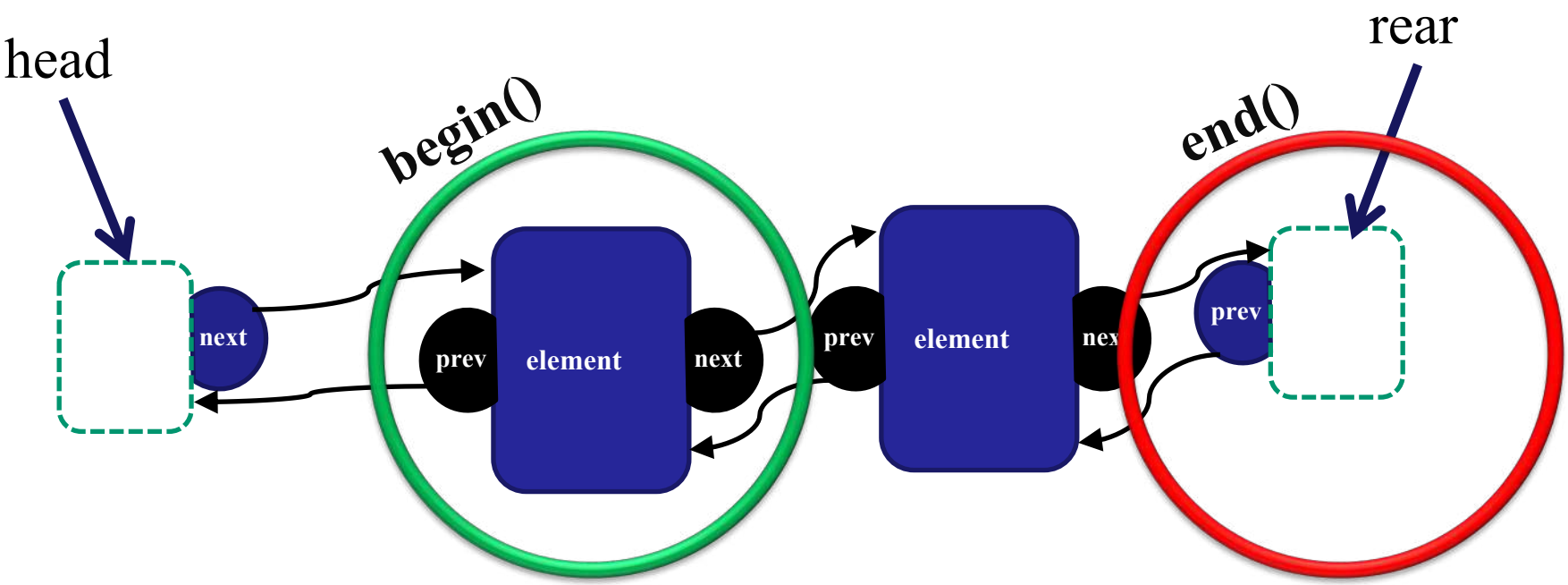
Inserim un segon element



Per incorporar un segon element a la llista, es pot fer al davant de l'element inserit anteriorment o al darrera d'ell. En aquest cas, noteu que els punters head i rear no es mouran de la posició que tenen establerta des de la creació de la llista. Només cal enllaçar els 4 punters involucrats en l'operació.

Iterant llistes dobles amb sentinelles

Iterar endavant - I for (Position itr = begin(); itr != end(); itr = ++itr)



Iterant llistes dobles amb sentinelles

- A la transparència anterior podeu veure que:
  - El mètode **begin**() us retorna la primera posició després del primer sentinella.
  - El mètode **end**() us retorna el darrer sentinella.
  - En el cas que la llista estigui buida, el **begin**() us donarà la mateixa posició que el **end**(), i per tant, no cal fer recorregut ja que no hi ha elements.
  - Per iterar volem anar des del primer element (no el sentinella, sinó el primer element real) i aturar la iteració quan ja s'hagin visitat tots els elements (aquest és el cas quan arribem al darrer sentinella).

Iterant llistes dobles amb sentinelles

- Per iterar sobre la llista fem:

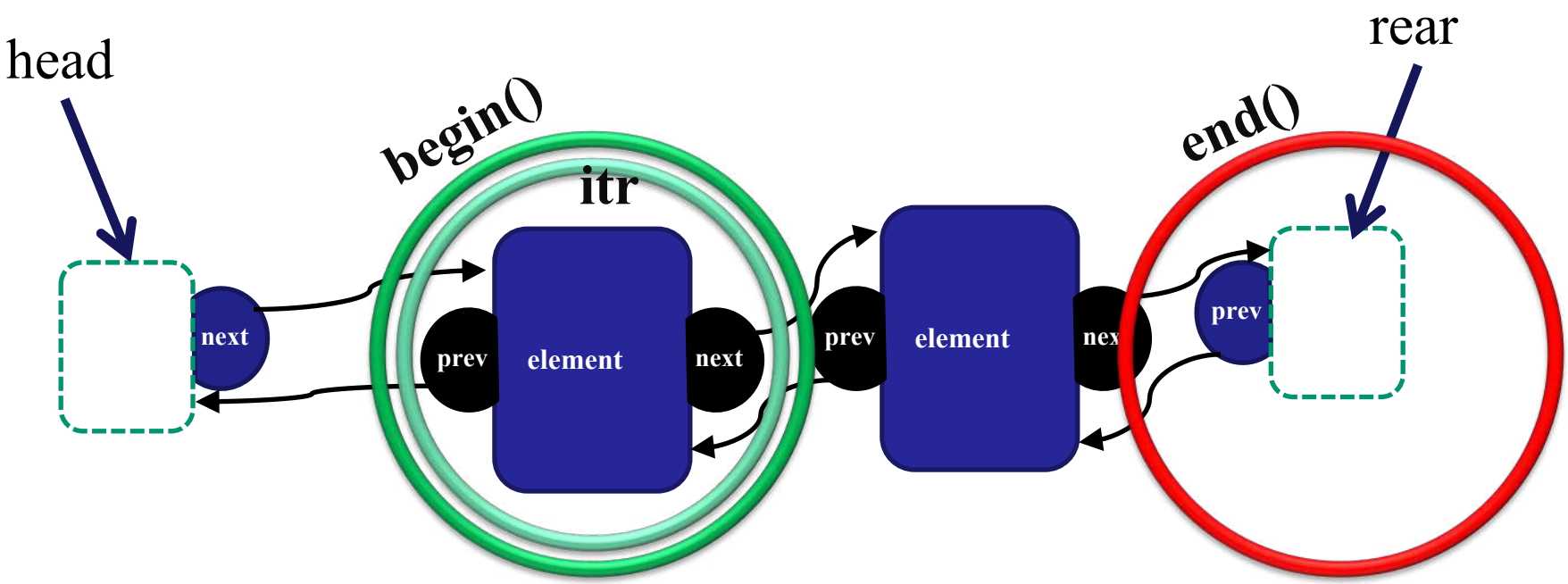
```
for (Position itr = begin(); itr != end(); itr = ++itr)
```

  - El begin() ens posa l'iterador al primer element
  - El ++itr ens avança una posició a cada volta del bucle
  - El itr!=end() fa que el bucle finalitzi quan trobem la posició del darrer sentinella

- A continuació teniu gràficament com es fa aquest bucle pas a pas.

Iterant llistes dobles amb sentinelles

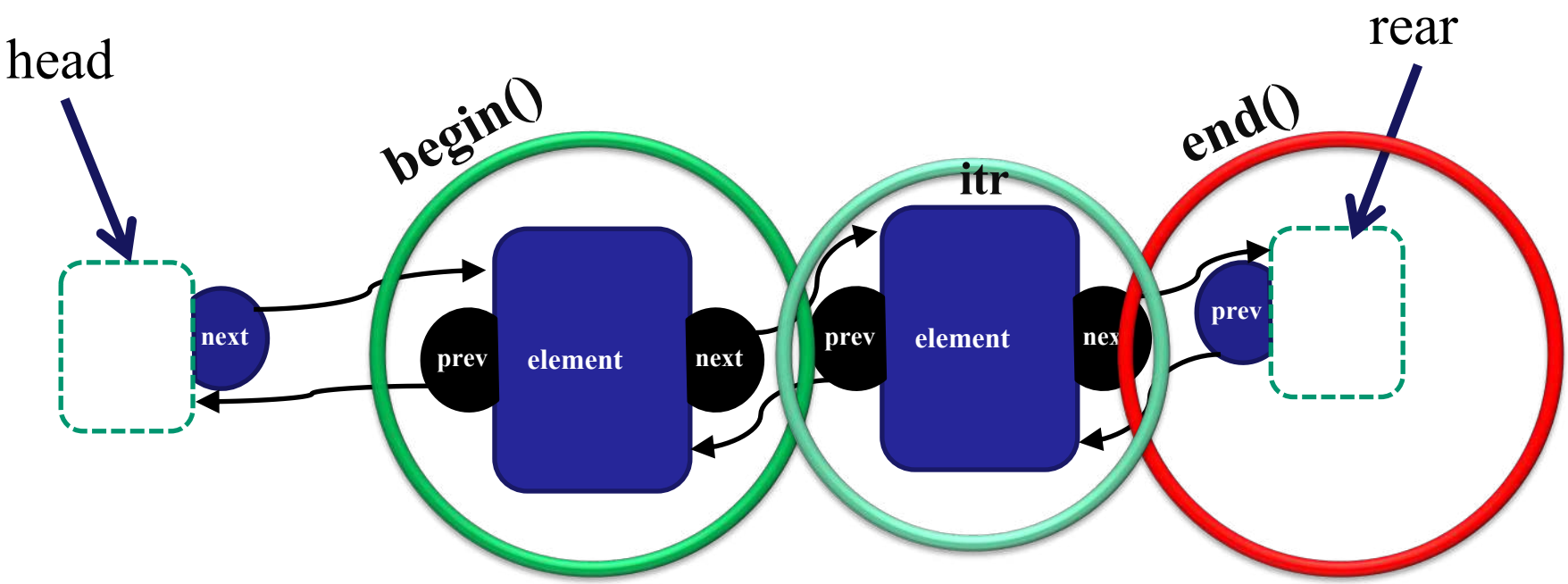
Iterar endavant - II for (Position itr = begin(); itr != end(); itr = ++itr)



Entrada al for amb itr = begin()

Iterant llistes dobles amb sentinelles

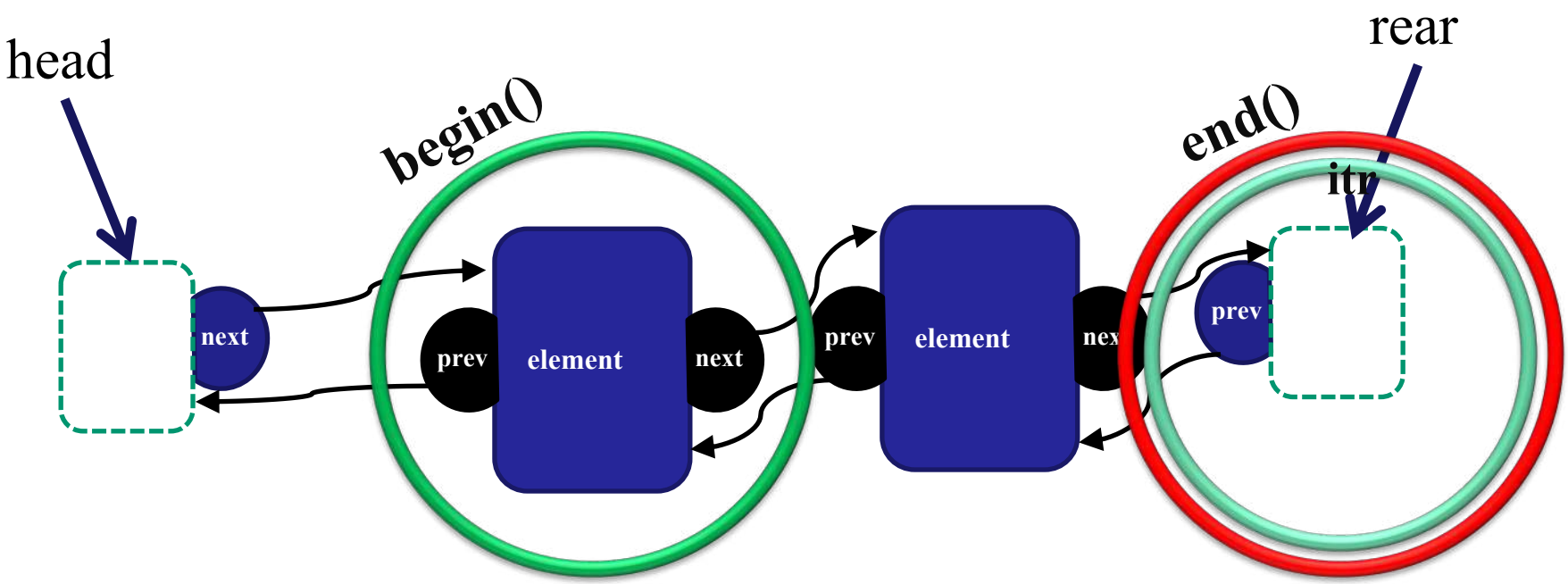
Iterar endavant - III for (Position itr = begin(); itr != end(); itr = ++itr)



Segona volta del bucle, s'ha avançat la posició de itr amb el ++itr.

Iterant llistes dobles amb sentinelles

Iterar endavant - IV for (Position itr = begin(); itr != end(); itr = ++itr)



Final del bucle, s'ha avançat la posició de itr amb el ++itr i el bucle s'acaba perquè ha trobat que la condició de finalització ja no és certa (itr == end()).

LinkedList.h

```
#include "node.hpp"
#include "position.hpp"
#include <initializer_list>
#include <iostream>
#include <stdexcept>

template <class Element>
class LinkedList
{
public:
    LinkedList();
    LinkedList(std::initializer_list<Element> elements);
    LinkedList(const LinkedList<Element>& list);
    virtual ~LinkedList();

    int size() const;
    bool empty() const;
    bool begin(Position<Element>& position) const; // return a bool indicating if it is first sentinel
    bool end(Position<Element>& position) const; // return a bool if it is last sentinel
    Position<Element> first() const; // return first position
    Position<Element> last() const; // return last position

    void insertAfter(Position<Element>& position, const Element& element);
    void insertBefore(Position<Element>& position, const Element& element);
    void insertFront(const Element& element);
    void insertBack(const Element& element);

    void print();

private:
    NodeList<Element>* _head;
    NodeList<Element>* _rear;
    int _size;
};
```

Node.h

```
template <class Element>
class NodeList
{
public:
    NodeList();
    NodeList(Element element);
    ~NodeList();

    const Element& getElement() const;

    NodeList<Element>* getNext() const;
    void setNext(NodeList<Element>* node);

    NodeList<Element>* getPrevious() const;
    void setPrevious(NodeList<Element>* node);

private:
    Element _element;
    NodeList<Element>* _next;
    NodeList<Element>* _previous;
};
```

Position.h

```
#include "node.hpp"

template <class Element>
class Position
{
public:
    Position(NodeList<Element>* node);

    Position<Element> next() const;
    Position<Element> previous() const;
    const Element& element() const;

    void setPrevious(NodeList<Element>* node);
    void setNext(NodeList<Element>* node);

    Position<Element> operator++() const;
    Position<Element> operator--() const;
    bool operator!=(const Position& other) const;
    const Element& operator*() const;

private:
    NodeList<Element>* _node;
};
```

Constructor

```
template <class Element>
 LinkedList<Element>::LinkedList()
{
   _head = new NodeList<Element>();  // sentinella front
   _rear = new NodeList<Element>();// sentinella rear
   _head->setNext(_rear);
   _rear->setPrevious(_head);
}
```

Begin

```
template <class Element>
bool LinkedList<Element>::begin(Position<Element>& position) const
{
    if (empty())
    {
        throw std::out_of_range("LinkedList<>::begin(): empty list");
    }

    return (Position<Element>(_head) != position);

}
```

Exercicis

Tenint en compte l'especificació de les classes LinkedList i Node anteriors, implementeu els mètodes per:

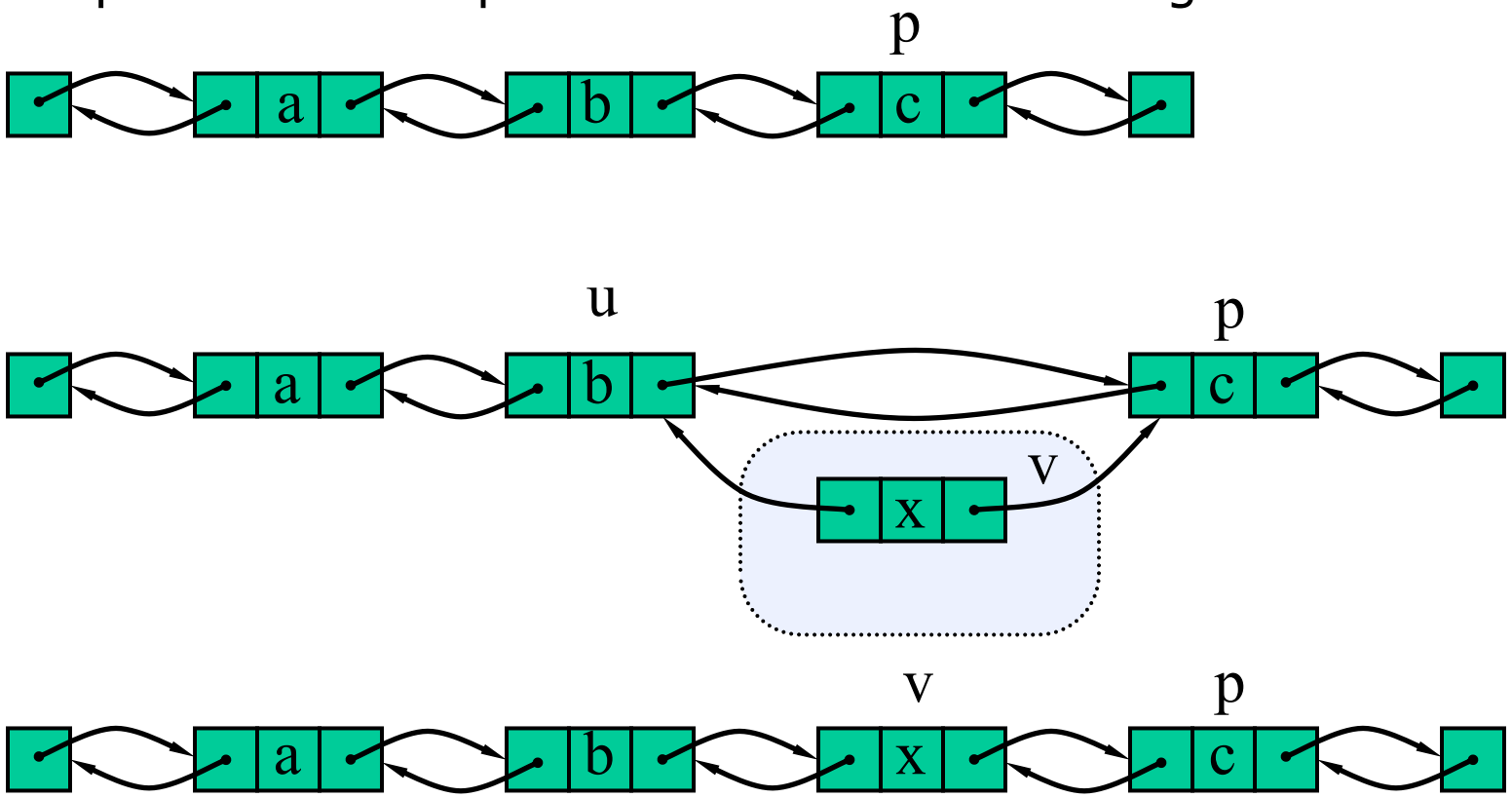
- Inserir a la posició p de la llista
- Esborrar a la posició p de la llista

Intenteu fer els exercicis sense mirar la solució, si us plau.

A continuació teniu un dibuix per ajudar-vos i posteriorment teniu la solució en pseudocodi.

Inserció a la posició p de la llista

- Visualització de l'operació insert(p, x), la qual inserta l'element x abans de la posició p
- Noteu que primer s'enllacen els punters prev i next del node i al final els punters next i prev del node anterior i següent.



UNIVERSITAT DE BARCELONA

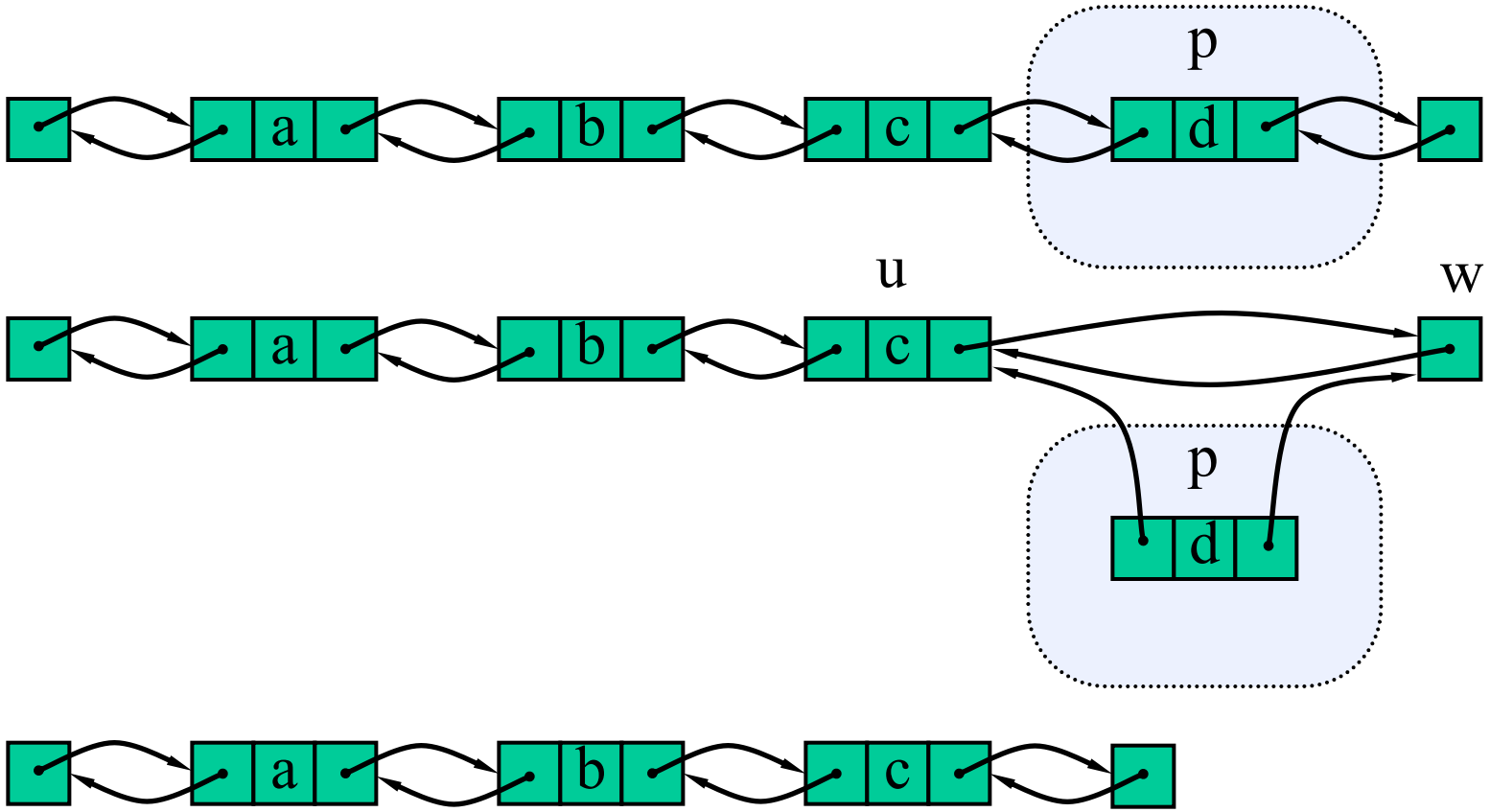
Algorisme d'inserció

```
Algorithm insert(p, e): //inserta e abans de la posició p
{
   // AQUÍ LA VOSTRA SOLUCIÓ
}
```

ESCRIVIU EL PSEUDOCODI

Eliminar a la posició p de la llista

- Visualització de l'operació remove(p)
- Noteu que s'han de reenllaçar el punter next de l'anterior node i el punter prev del node següent



Algorismes d'eliminació

```
Algorithm remove(p)
{
   //AQUÍ EL VOSTRE CODI
}
```

ESCRIVIU EL PSEUDOCODI

Rendiment TAD NodeList

- En la implementació del TAD Llista amb una estructura doblement encadenada
  - L'espai usat per la llista amb n elements és $O(n)$
  - L'espai usat per cada posició de la llista és $O(1)$
  - Totes les operacions del TAD NodeList tenen un cost $O(1)$ en temps computacional teòric
  - L'operació element() del TAD Posició té un cost $O(1)$ en temps computacional teòric

3.6 TAD cua prioritària

La cua prioritària la veurem conjuntament amb el Tema 5 de HEAPS

UNIVERSITAT DE BARCELONA

TAD cua prioritària

- Una cua prioritària guarda una col·lecció d'entrades
  - Cada entrada (entry) és un parell (clau, valor), on clau indica la prioritat
  - Dos entrades diferents en una cua prioritària poden tenir la mateixa clau
- Mètodes
  - insert(e) insereix una entry e
  - removeMin() elimina la entry amb la clau mínima
- Mètodes addicionals:
  - min() retorna però no elimina la entry amb clau mínima
  - size(), empty()

TAD Comparador

- Implementa la funció booleana isLess(p,q), la qual testeja si p < q
- Pot derivar altres relacions des d'aquí:
  - (p == q) és equivalent a
  - (!isLess(p, q) && !isLess(q, p))
- Pot implementar-se en C++ sobrecarregant "()"

Dues maneres de comparar 2D points:

```
class LeftRight { // left-right comparator
public:
    bool operator()(const Point2D& p,
     const Point2D& q) const
    { return p.getX() < q.getX(); }
};
class BottomTop { // bottom-top
public:
    bool operator()(const Point2D& p,
    const Point2D& q) const
    { return p.getY() < q.getY(); }
};
```

Ordenació amb cues prioritàries

- Es pot usar una cua prioritària per ordenar un conjunt d'elements
  1. Insereix els elements un a un amb una sèrie d'operacions de insert
  2. Extreu els elements en ordre amb una sèrie d'operacions de removeMin
- El temps de l'ordenació depèn de la implementació de la cua prioritària

Algorithm *PQ-Sort*(*S*, *C*)
Input seqüència *S*, comparador *C* pels elements de *S*
Output seqüència *S* ordenada en ordre creixent segons *C*

P ← cua prioritària amb el comparador *C*
while ¬*S.empty* ()
e ← *S.front*(); *S.eraseFront*()
P.insert (*e*, ∅)
while ¬*P.empty*()
e ← *P.removeMin*()
S.insertBack(*e*)

Cues prioritàries basades en encadenaments (ordenació)

- Llista desordenada

- Eficiència:
  - insert tarda $O(1)$ en temps ja que quan s'insereix un item, s'insereix al començament de la seqüència
  - removeMin i min tarden $O(n)$ en temps ja que s'ha de recórrer tota la seqüència per trobar la clau mínima

- Llista ordenada

- Eficiència:
  - insert tarda $O(n)$ en temps ja que s'ha de trobar el lloc on correspon inserir l'ítem
  - removeMin i min tarden $O(1)$ en temps, ja que la clau mínima està sempre al començament