MP 20-21 Tema 2
Estructures dinàmiques enllaçades
Sessió 13: Estructures de dades dinàmiques

Exercici

- Tornem a l'exemple inicial dels estudiants i les titulacions...

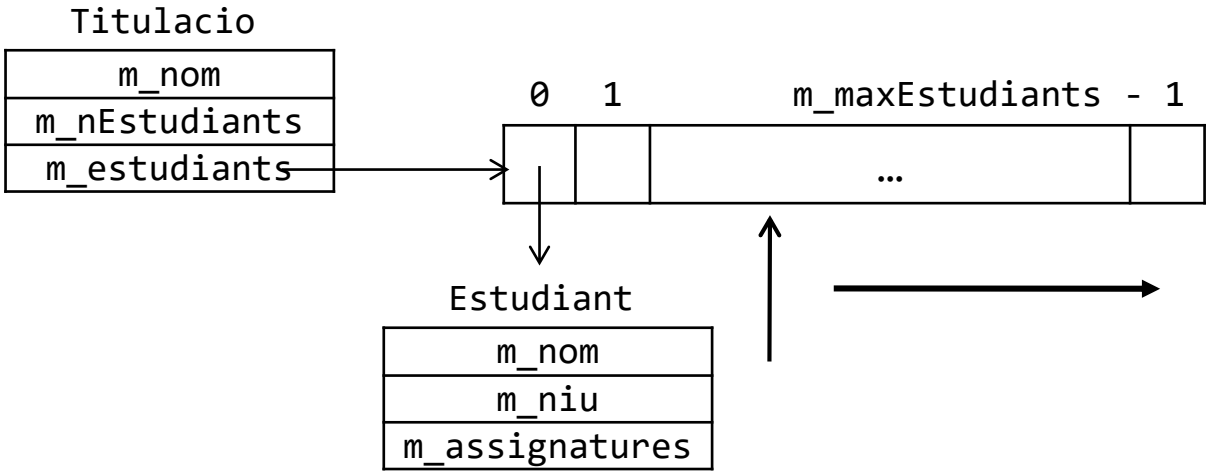
Volem crear un conjunt de classes que ens permetin guardar els alumnes que s'han matriculat a una determinada titulació de la UAB. Per cada estudiant hem de poder guardar les seves dades bàsiques (NIU i nom) i el nom de totes les assignatures a les que s'ha matriculat. Suposem que hem creat aquestes classes per guardar les dades d'un estudiant i de tots els estudiants matriculats a una titulació.

```
class Titulacio
{
private:
   string m_nom;
   Estudiant* m_estudiants;
   int m_nEstudiants;
};
```

```
class Estudiant
{
private:
   string m_nom;
   string m_NIU;
   vector<string> m_assignatures;
};
```

Exercici

- Suposem que utilitzem un array dinàmic per guardar tots els estudiants d'una titulació



- Quins problemes té la utilització d'un array dinàmic en aquest cas?
  - Quin tamany fixem per l'array dinàmic?
    - Podríem utilitzar un sistema de gestió de memòria similar al que utilitza la classe `vector` per adaptar el tamany al nº d'estudiants real en cada moment.
    - Igualment, sobrecost de redimensionar i copiar el contingut de l'array.
  - Sobrecost de desplaçar els elements de l'array quan inserim/eliminem

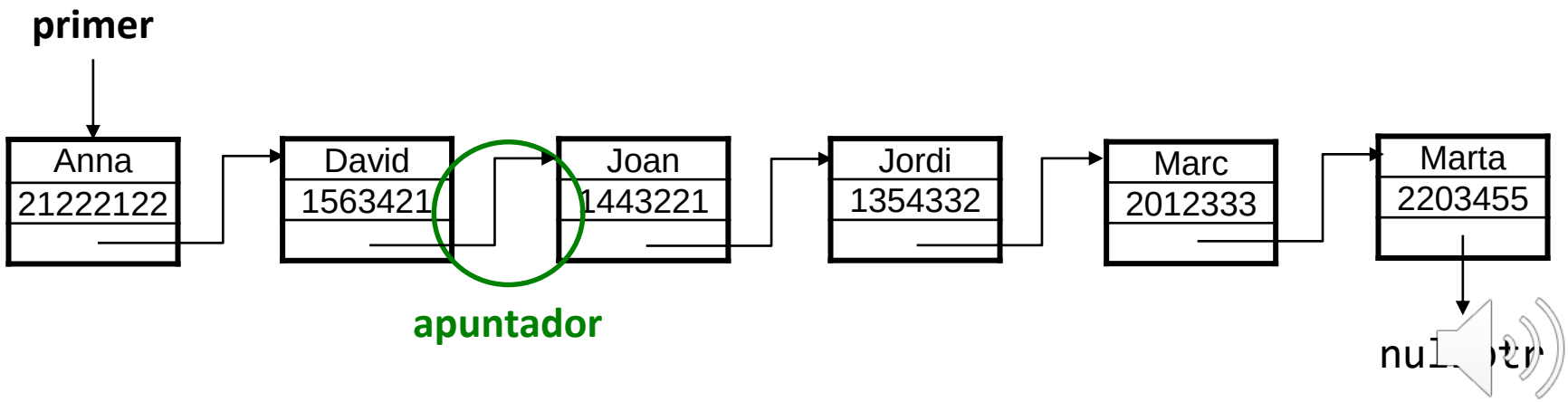
Estructures dinàmiques enllaçades

Alternativa: estructures dinàmiques enllaçades

- Només s'ocupa la memòria necessària per guardar els elements actuals. S'assignarà o alliberarà memòria cada cop que s'insereixi o elimini un element.
- Cada element és independent de la resta. No cal moure la resta dels elements quan s'insereix o s'elimina.
- Cada element té un enllaç (apuntador) al següent element de l'estructura.

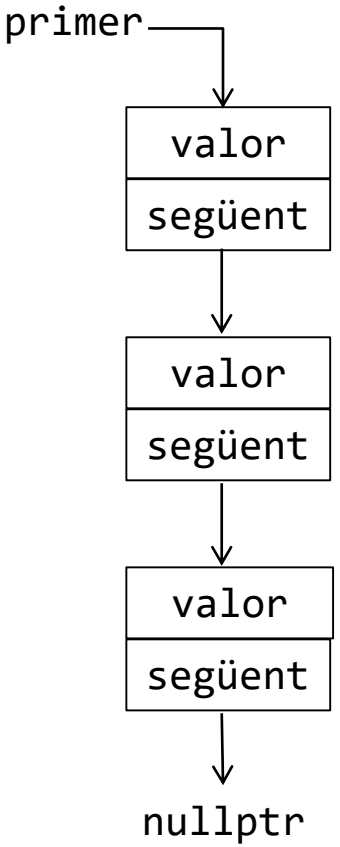
Inconvenients:

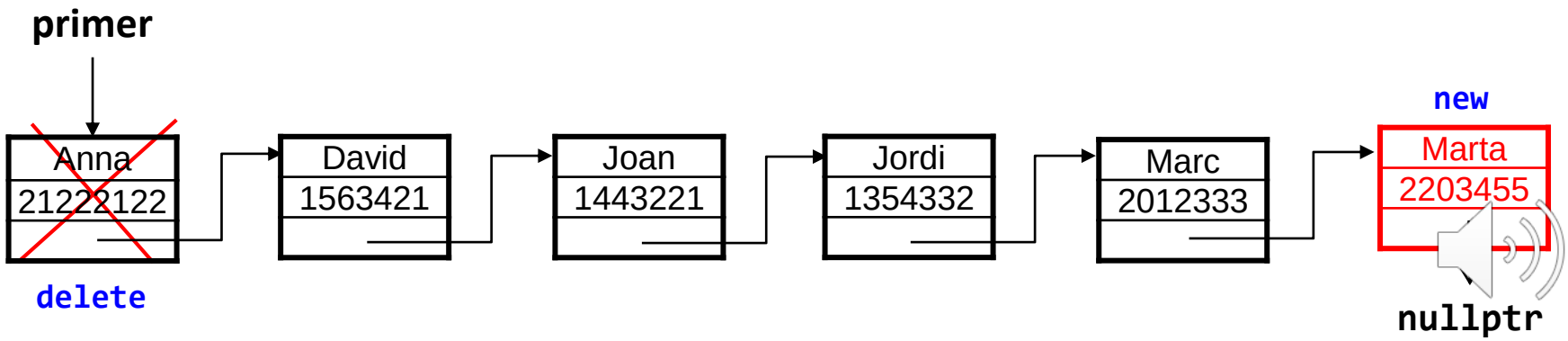
- Perdem la possibilitat d'accedir de forma directa a qualsevol element
- Per accedir a un element haurem de recórrer tots els anteriors



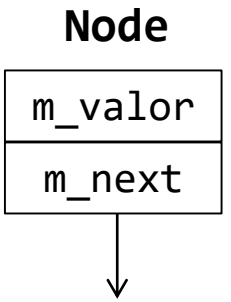
Estructures dinàmiques enllaçades

- Cada element serà dinàmic:
  - Es crearan amb l'operador new quan s'afegeixin a l'estructura
  - Es destruiran amb l'operador delete quan s'eliminin de l'estructura
- Cada element enllaçarà amb el següent:
  - Necessitem afegir un apuntador al següent element
  - Haurem de mantenir la coherència dels enllaços entre elements quan afegim i eliminem
- Necessitarem un apuntador al primer element de l'estructura
- L'últim element de l'estructura apuntarà a `nullptr`





Classe Node:



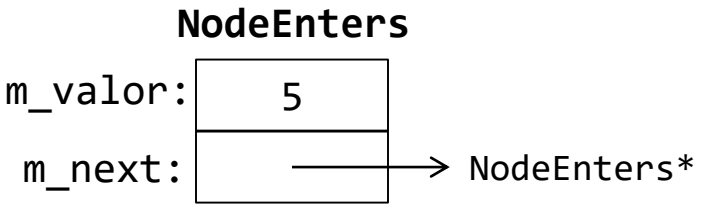
- Serveix per gestionar l'apuntador que permet mantenir els enllaços entre els elements de l'estructura. Ha de contenir:
  - El valor que volem guardar a l'estructura, del tipus que correspongui.
  - Un apuntador al següent element, de tipus apuntador a node (Node*).
  - Getters i setters tant pel valor que hi guardem com per l'apuntador al següent element.

```
class Node
{
public:
   <tipus>& getValor() { return m_valor; }
   Node* getNext() { return m_next; }
   void setValor(const <tipus>& valor) { m_valor = valor; }
   void setNext(Node* next) { m_next = next; }
private:
   <tipus> m_valor;
   Node* m_next;
};
```

Serà diferent en cada cas, depenent del tipus de valor que volem guardar a l'estructura

Classe Node:

```
class NodeEnters
{
public:
   int& getValor();
   NodeEnters* getNext();
   void setValor(const int& valor);
   void setNext(NodeEnters* next);
private:
   int m_valor;
   NodeEnters* m_next;
};
```




```
class Estudiant
{
   ...
}

class NodeEstudiant
{
public:
   Estudiant& getValor();
   NodeEstudiant* getNext();
   void setValor(const Estudiant& valor);
   void setNext(NodeEstudiant* next);
private:
   Estudiant m_valor;
   NodeEstudiant* m_next;
};
```