Tema 4
Tipus i estructures de dades

Informàtica
Grau en Física
Universitat de València

Ariadna.Fuertes@uv.es
Francisco.Grimaldo@uv.es

Guió del tema

- Introducció:
  - Concepte de dada estructurada.
  - Classificació de les agrupacions de dades.
  - Tipus d'agrupacions.
- Estructures de dades contigües:
  - vectors,
  - matrius,
  - cadenes de caràcters i,
  - estructures (o registres).
- Punters i estructures de dades dinàmiques.
  - Introducció a les estructures dinàmiques:

    Llistes enllaçades.

Introducció (1/3)

- Els tipus simples de dades: booleans, caràcters, enters i reals.
- En general, la informació tractada pels ordinadors s'agruparà d'una manera més o menys coherent en estructures especials, compostes per dades simples. Aquests tipus d'agrupacions els anomenem tipus de dades estructurats o tipus compostos.
- Els tipus de dades estructurats o tipus compostos es distingeixen pels elements que els componen i per les operacions que hom pot realitzar amb elles o entre els elements d'aquestes.
- De vegades, interessarà treballar amb les estructures com a elements únics i unes altres, interessarà accedir als elements que formen part dels tipus de dades estructurats com a elements separats.

Introducció (2/3)

- Classificacions de les agrupacions de dades:
  - Segons com s'allotgen a la memòria de l'ordinador: Agrupacions contigües/agrupacions enllaçades
    - Les agrupacions contigües són aquelles que ocupen posicions successives o contigües de bits en memòria. La posició d'un element dins de l'agrupació es pot calcular si sumem al començament de l'agrupació una quantitat determinada.
    - Les agrupacions enllaçades són aquelles que tenen la informació dispersa per la memòria. Per tant, la manera d'accedir des d'un element a un altre és mitjançant l'adreça de memòria on està l'element següent.
  - Segons els tipus de dades que continguen: Agrupacions homogènies/agrupacions heterogènies
    - Les agrupacions homogènies són aquelles que contenen grups d'elements iguals (p. ex. tots els elements que emmagatzemen són enters, reals,...).
    - Les agrupacions heterogènies són les que agrupen elements (camps) de diferents tipus.

Introducció (3/3)

- Segons si tenen una grandària d'element fixa o variable: Agrupacions estàtiques/agrupacions dinàmiques
  - Les agrupacions estàtiques són les que allotgen un nombre predeterminat d'elements, el qual no podrà ser modificat mentre s'execute el programa. Cal triar aquest nombre quan escrivim el programa, abans de compilar-lo.
  - Les agrupacions dinàmiques són aquelles que poden modificar el nombre d'elements que contenen si necessiten més espai per a emmagatzemar més informació mentre s'executa el programa.

- Tipus d'agrupacions de dades:
  - arrays
    - vectors o arrays unidimensionals,
    - matrius o arrays multidimensionals i
    - cadenes.
  - registres o estructures i
  - llistes dinàmiques.

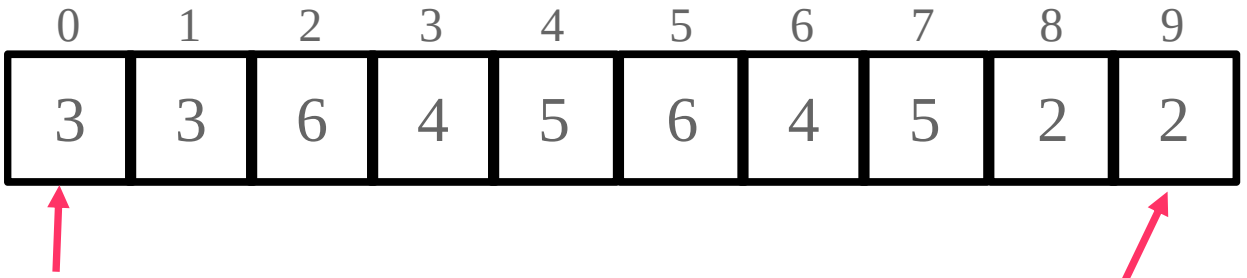
Les agrupacions, en general, no s'empren de manera aïllada. Açò és, podem trobar-ne combinacions quan siga necessari.

Arrays (1/27)

- Els arrays són *agrupacions homogènies, contigües i estàtiques*. Els elements que allotgen són tots del mateix tipus i el nombre d'elements que podran tindre es defineix quan escrivim el programa.
- Podem distingir entre els següents tipus d'array: vectors (arrays unidimensionals), matrius (arrays bidimensionals) i arrays multidimensionals.
- Vectors (o arrays unidimensionals).
- Són agrupacions en què cada element té associat un índex (un enter), de manera que es pot accedir a cadascun dels elements mitjançant l'ús d'aquest índex (operació d'**indexació**). L'índex indica la posició de l'element dins del vector.
- Aquest tipus de dades estructurat serveix per a agrupar variables d'un mateix tipus sota un nom únic.
- Suposem que volem declarar 10 variables de tipus enter. L'única manera de fer-ho fins ara seria declarar-les variables individuals. D'ara endavant, també podrem fer-ho si emprem un vector.

Arrays: Vectors (2/27)

- La declaració d'un vector es farà de la manera següent:
  Nom[Grandària] : Tipus
- On Tipus és el tipus de les dades emmagatzemades dins del vector, Nom és el nom que rep el vector i Grandària és el nombre d'elements que pot allotjar el vector.
- Els índexs comencen en 0 i acaben en Grandària-1.



Índexs
vec[10]

vec[0] Primer element del vector.
vec[1] Segon element del vector.

...

vec[9] Darrer element del vector (Grandària-1).

Arrays: Vectors - operacions (3/27)

Operacions:

- **Assignació**
- Per donar un valor determinat a algun element del vector, emprarem el nom del vector i l'index que fa referència a l'element que volem assignar:

Nom[índex] ← Valor

- **Exemple**: Per assignar el valor 4 a l'índex (o posició del vector) vuit:
  vect [8] ← 4

- NO es poden fer assignacions entre vectors. MAI s'ha de realitzar l'assignació següent:
  a[10], b[10]: enters
  a ← b ¡¡¡ERROR!!!

Arrays: Vectors - operacions (3/27)

- **Recorregut**
- Cal passar per tots els elements del vector per fer una tasca concreta sobre cada element.

- Recorregut complet

```
Des de i ← 0 fins a Grandària-1 fer
        Processar (Nom[i])
        i ← i+1
Final_des_de
```

- Recorregut parcial

```
i ← 0
Mentre (Nom[i] complisca una certa condició) fer
    Processar (Nom[i])
     i ← i+1
Final_mentre
```

- Per fer més senzilla la programació de bucles i evitar possibles errades en el recorregut dels vectors, és recomanable que el nombre d'elements que poseeix un vector (TAM) es definisca breviament com una constant.

Arrays: Vectors - operacions (4/27)

- **Exemple**: Feu una funció que calcule la mitjana dels elements contingut dins d'un vector.

```
Funció MitjanaVector ( vect[TAM]: enter) : real
Variables
    i : enter
    mit : real
Inici
    mit ← 0
    Des de i ← 0 fins a TAM-1 fer
        mit ← mit + vect[i]
        i ← i + 1
    Final_Des_de

    mit ← mit/TAM
    MitjanaVector ← mit
Final_Funció
```

Arrays: Vectors - operacions (5/27)

- **Iniciació**
- La declaració del vector només reserva l'espai però NO posa cap valor dins dels elements del vector.
- Per iniciar tots els elements del vector, caldrà recórrer-lo sencer i assignar-hi un valor (o demanar a l'usuari que proporcione cadascun dels valors que s'hi volen assignar).

- **Exemple**: Suposem que volem demanar a l'usuari que indique tots els valors del vector:

```
Des de i ← 0 fins a TAM-1 fer
    llegir( vect[i])
    i ← i + 1
Final_Des_de
```

- **Exemple**: Suposem que volem posar tots els elements del vector a zero:

```
Des de i ← 0 fins a TAM-1 fer
    vect[i] ← 0
    i ← i + 1
Final_Des_de
```

Arrays: Vectors - operacions (6/27)

- **Exemple**: Realitzeu un procediment que emplene els elements d'un vector amb dades introduïdes per l'usuari.

```
Procediment EmplenaVector (ref vect[TAM]: enter)
Variables
    i : enter
Inici
    Des de i ← 0 fins a TAM-1 fer
        llegir( vect[i] )
        i ← i + 1
    Final_Des_de
Final_Procediment
```

- PREGUNTA: Podria existir la Funció EmplenaVector?

Arrays: Vectors - operacions (7/27)

- **Cerca en vectors**
- Per a cercar un valor dins d'un vector, en principi, cal recórrer tots els elements i comprovar si hi és. Aquesta aproximació rep el nom de *cerca seqüencial*.

Exemple: *Feu una funció que diga si un cert valor 'x' es troba o no dins d'un vector anomenat 'vect'*:

```
Funció Cercar1 ( vect[TAM]: enter, x : enter) : booleà
Variables
    i : enter
    trobat : booleà
Inici
    trobat ← fals
    Des de i ← 0 fins a TAM-1 fer
        Si vect[i] = x aleshores
            trobat ← vertader
        Final_Si
        i ← i + 1
    Final_Des_de
    Cercar1 ← trobat
Final_Funció
```

Arrays: Vectors - operacions (8/27)

- Millora l'algorisme anterior: "Eixir del bucle quan es trobe l'element" a més a més de si s'arriba al final...

```
Funció Cercar2 ( vect[TAM]: enter, x : enter) : booleà
Variables
    i : enter
    trobat : booleà
Inici
    i ← 0
    Mentre i < TAM AND vect[i] ≠ x fer
        i ← i + 1
    Final_Mentre
    Si i = TAM aleshores
        trobat ← fals
    Si no
        trobat ← vertader
    Final_Si
    Cercar2 ← trobat
Final Funció
```

Arrays: Vectors - operacions (9/27)

- Podem estalviar-nos la comparació ‘i < TAM’ si estem segurs que l’element ‘x’ es troba dins del vector. I podem estar-ne si el posem al final del vector. Aquesta manera de buscar s’anomena *cerca seqüencial amb sentinella*.

```
Funció Cercar3 ( vect[TAM+1]: enter, x : enter) : booleà
Variables
    i : enter
    trobat : booleà
Inici
    vect[TAM] ← x
    i ← 0
    Mentre vect[i] ≠ x fer
        i ← i + 1
    Final_Mentre
    Si i = TAM aleshores
        trobat ← fals
    Si no
        trobat ← vertader
    Cercar3 ← trobat
Final_Funció
```

Arrays: Vectors - operacions (10/27)

- Si el vector es troba ordenat, podrem aplicar una tècnica especial de cerca que s'anomena *cerca binària o dicotòmica*:

```
Funció Cercar4 ( vect[TAM]: enter, x : enter) : booleà
Variables
        primer, ultim, central : enters
        trobat : booleà
Inici
        primer ← 0
        ultim ← TAM-1
        central ← (primer+ultim)/2
        Mentre primer < ultim AND vect[central] ≠ x fer
                Si x < vect[central] aleshores
                        ultim ← central -1
                Si no
                        primer ← central +1
                Final_Si
                central ← (primer+ultim)/2
        Final_Mentre
        Si x = vec[central] aleshores
                trobat ← vertader
        Si no
                trobat ← fals
        Cercar4 ← trobat
Final_Funció
```

Cerca binària o dicotòmica:
1. Es comprova l'element central.
2. Si és l'element cercat, aleshores acabar.
3. Si no ho és:
3.a. Determinar en quina 'zona' del vector estarà (per damunt o per davall del central).
3.b. Repetir el procés en la zona nova.

Arrays: Vectors - operacions (11/27)

- *El programa principal que cridarà les diverses funcions cercar (que podrien ser la 1 o la 2) seria:*

```
Variables
    v[TAM] , x : enters
    hi_es : booleà
Inici
    Escriure("Escriu les dades del vector")
    EmplenaVector( v )
    Escriure("Escriu el nombre que cal cercar")
    Llegir( x )

    hi_es ← Cercar2( v, x )

    Si hi_es = vertader aleshores
        Escriure("S'ha trobat l'element")
    Si no
        Escriure("L'element no hi és")
    Final_Si
Final
```

Arrays: Vectors - operacions (12/27)

- *Modificació de la funció Cercar2 perquè torne en quina posició ha trobat l'element:*

```
Funció CercarAmbPos ( vect[TAM]: enter, x : enter) : enter
Variables
      i , trobat: enter
Inici
      i ← 0
      trobat ← -1

      Mentre i < TAM AND vect[i] ≠ x fer
            i ← i + 1
      Final_Mentre

      Si i ≠ TAM aleshores
            trobat ← i
      Final_Si
      CercarAmbPos ← trobat
Final_Funció
```

```
PROGRAMA PRINCIPAL
Variables
   v[TAM] , x , hi_es : enters
Inici
   Escriure("Dona'm les dades del vector")
   EmplenaVector( v )
   Escriure("Dona'm el nombre que cal
      cercar")
   Llegir( x )

   hi_es ← CercarAmbPos( v, x )

   Si hi_es >=0 aleshores
      Escriure("Es troba a la posició", hi_es)
   Si no
      Escriure("L'element no hi és, ho sent")
   Final_Si
Final
```

Arrays: Matrius (13/27)

- Les *matrius o arrays bidimensionals* són agrupacions similars als vectors, però en les quals cadascun dels elements té associats dos índexs enters, mitjançant els quals s'hi pot accedir.

- La declaració d'una matriu segueix l'estructura següent: Nom[Grandària1][Grandària2] : Tipus.

On Tipus és el tipus de les dades emmagatzemades a la matriu, Nom és el nom que li donarem i Grandària1/Grandària2 és el nombre de files/columnes que té la matriu.

Arrays: Matrius - operacions (14/27)

Operacions:

- **Assignació**
- Dóna un valor determinat a algun element de la matriu. La manera de fer-ho és mitjançant el nom de la matriu i els índexs (ara en són dos) que volem assignar:

Nom[index1][index2] ← Valor

- **Exemple**: Per a assignar el valor 4 als índexs (o posició de la matriu) vuit i cinc:
  mat [8][5] ← 4

- Com amb els vectors, NO es poden realitzar assignacions entre matrius directament.
- Per tant, MAI una funció podrà tornar un vector, una matriu o, en general, un array.

Arrays: Matrius - operacions (15/27)

- **Recorregut**
- Cal passar pels elements de la matriu per a poder fer una operació concreta sobre cadascun dels elements.

- Recorregut complet

```
Des_de1 i ← 0 fins a TAMX-1 fer
        Des_de2 j ← 0 fins a TAMY-1 fer
                Processar ( Nom[i][j] )
                j ← j+1
        Final_Des_de2
        i ← i+1
Final_Des_de1
```

- Recorregut parcial

```
i ← 0
Mentre1 (Nom[i][j] complisca una certa condició) fer
        j ← 0
        Mentre2 (Nom[i][j] complisca una certa condició) fer
       Processar (Nom [i][j])
                j ← j+1
        Final_Mentre2
        i ← i+1
Final_Mentre1
```

Arrays: Matrius - operacions (16/27)

- **Exemple:** Feu una funció que recórrega tota una matriu i calcule la mitjana de tots els elements continguts en aquesta.

```
Funció MitjanaMatriu ( mat[TAM1][TAM2]: enter) : real
Variables
    i , j : enter
    mit : real
Inici
    mit ← 0
    Des_de1 i ← 0 fins a TAM1-1 fer
        Des_de2 j ← 0 fins a TAM2-1 fer
            mit ← mit + mat[i][j]
            j ← j + 1
        Final_Des_de2
        i ← i + 1
    Final_Des_de1
    mit ← mit / (TAM1*TAM2)
    MitjanaMatriu ← mit

Final_Funció
```

Arrays: Matrius - operacions (17/27)

- **Iniciació**
- Per a donar un valor inicial a tots els elements de la matriu, cal recórrer-la i assignar o demanar a l'usuari que introduisca cadascun dels valors dels elements. La resta d'operacions també són similars a les dels vectors, però amb dos índexs per a l'accés.

- **Exemple**: Suposem que volem demanar a l'usuari que done tots els valors de la matriu:

```
Des_de1 i ← 0 fins a TAM1-1 fer
        Des_de2 j ← 0 fins a TAM2-1 fer
                Llegir( mat[i][j] )
                j ← j+1
        Final_Des_de2
        i ← i+1
Final_Des_de1
```

- **Exemple**: Suposem que volem posar tots els elements de la matriu a zero:

```
Des_de1 i ← 0 fins a TAM1-1 fer
        Des_de2 j ← 0 fins a TAM2-1 fer
                mat[i][j] ← 0
                j ← j+1
        Final_Des_de2
        i ← i+1
Final_Des_de1
```

Arrays Multidimensionals (18/27)

- Els *arrays multidimensionals* són agrupacions similars als vectors, però en les quals cadascun dels elements té associats més de dos índexs enters, de manera que hom pot accedir als elements mitjançant aquests índexs.

- La declaració d'un array multidimensional es farà de la manera següent:
  Nom[Grandària1][Grandària2]...[GrandàriaN] : Tipus

On Tipus és el tipus de les dades emmagatzemades en la matriu, Nom és el nom que donem a l'array i Grandària1, Grandària2,..., GrandàriaN són les grandàries de les dimensions de l'array.

- Les operacions que es poden definir són similars a les que ja hem vist per als vectors o les matrius, tot i ampliant el nombre d'índexs segons calga. Cal recordar que per accedir a un element concret és necessari determinar el valor de tots els índexs.

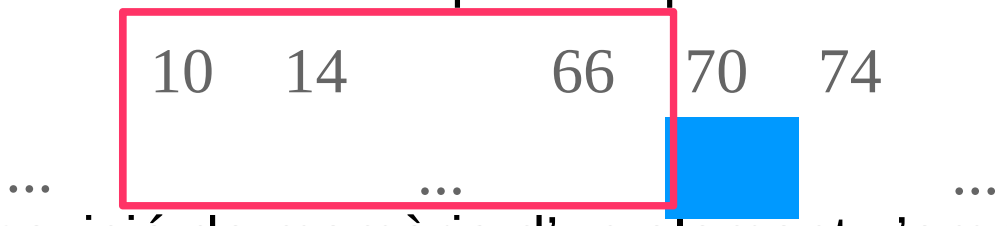
Arrays: Consideracions (19/27)

- Pas d'arrays com a paràmetre
- Els arrays, així com qualsevol altre tipus de dades, es poden passar com a paràmetre d'una funció, per valor o per referència.
- Com que una funció no pot retornar un array (ja que no es pot assignar), l'única manera serà si el paràmetre es passa per referència.

- Representació en memòria d'un array
- Els elements d'un vector s'emmagatzemen en memòria de manera contigua, açò és, l'un a continuació de l'altre.

Exemple: Vect[15] : enters

Si comença a la posició de memòria 10 i suposem que un enter ocupa 4 bytes, aleshores:



Per tant, per a calcular la posició de memòria d'un element s'empra la fórmula següent: *d = do + ind * grandàriaDe(element)*.

On *do* és la posició de memòria on comença el vector, *ind* és l'índex de l'element del vector del qual volem conéixer la posició i *element* es correspon amb el tipus d'elements continguts dins del vector.

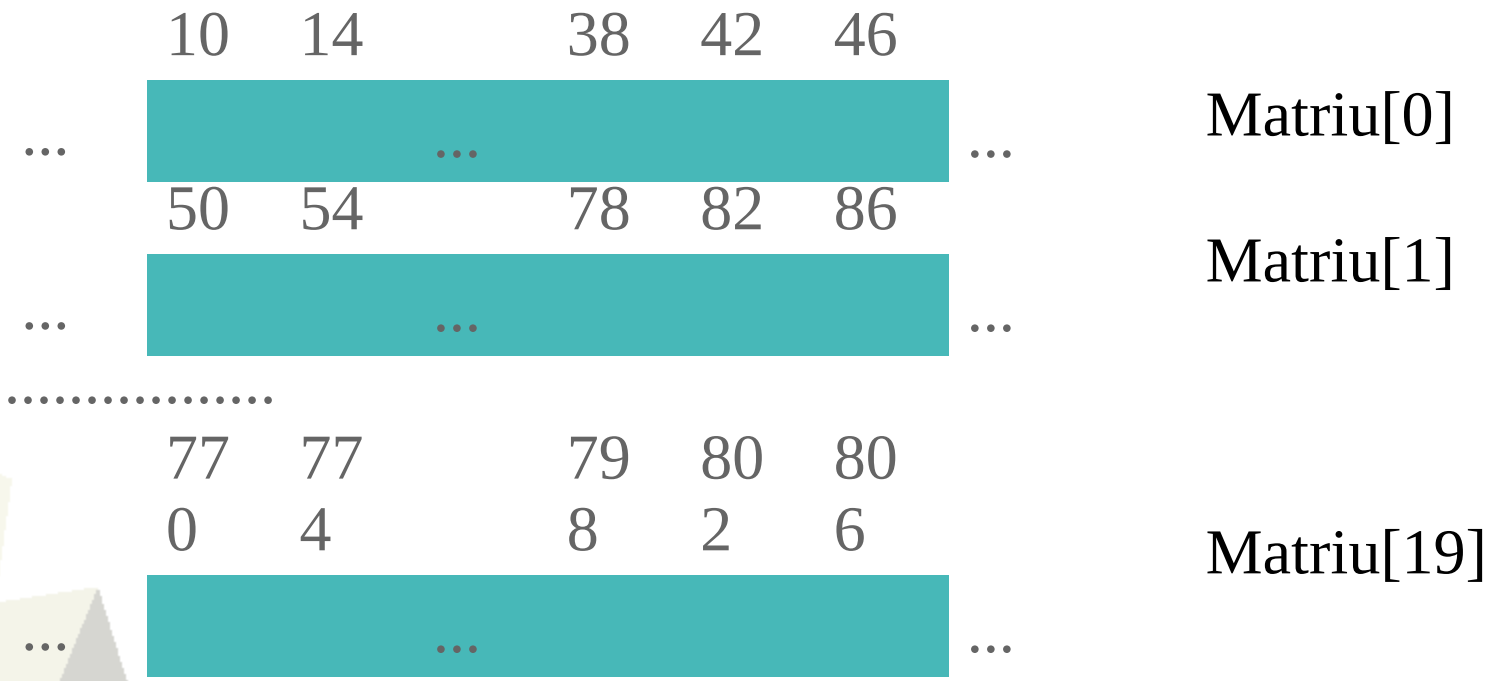
Nota: En C/C++, *grandàriaDe* és la funció sizeof(x). Ex.: sizeof(int) és 4.

Arrays: Consideracions (20/27)

- Donat que una matriu no és més que un vector de vectors, l'emmagatzematge en memòria és també contigu:

- Exemple: Matriu[20][10] : enters

Si comença a la posició de memòria 10 i *grandàriaDe(enter)* és 4 bytes:



Per a calcular la posició de l'element Matriu[19][1], primer cal resoldre la posició del primer índex (Matriu[19]), que té com a elements vectors enters de 10 elements, i després el segon es resol si prenem com a posició inicial la que resulta del càlcul anterior. Així, si el vector comença per exemple en la posició 10, la fórmula serà:

d = 10 + 19 * grandàriaDe(vect [10] d'enters) + 1 * grandàriaDe(enter) = 10 + 760 + 4 = 774

Arrays: Consideracions (21/27)

- Ús d'arrays amb una quantitat d'elements variable.
- La grandària d'un array és constant, per tant, no és possible modificar-ne la capacitat després d'haver compilat el programa.
- Per a solucionar açò, hi ha dues possibilitats:
  - Utilitzar memòria dinàmica (ho veurem al final del tema).
  - Declarar un array amb prou capacitat i fer-ne servir només una part.
- **Per exemple:** si estem fent un programa per a sumar vectors matemàtics i sabem que en cap cas tindrem vectors d'una grandària major que 10, podem declarar tots els vectors de grandària 10 i després utilitzar només una part del vector.
- Per tal de fer-ho, necessitarem també una variable entera addicional que indique els elements del vector que estem usant, ja que totes les operacions s'executaran sobre aquesta grandària i no sobre el total.
- Els desavantatges d'aquest mètode són que estem malbaratant memòria i, a més a més, no sempre hi ha un límit conegut a la grandària d'un array.
- Avantatges: és molt fàcil treballar amb l'array.

Arrays: Consideracions (22/27)

Ex.: Suma de dos vectors d'un espai vectorial real de dimensió n

```
Proc. Suma (v1[TAM]: real, v2[TAM]: real, ref res[TAM]: real, n :enter)
Variables
      i : enter
Inici
      Des_de i ← 0 fins a n-1 fer
            res[i] ← v1[i] + v2[i]
            i ← i + 1
      Final_Des_de
Final_Procediment
```

```
Proc. Emplena( ref vect[TAM]: real, n: enter)
Variables
      i : enter
Inici
      Des_de i ← 0 fins a n-1 fer
            Llegir (vect[i])
            i ← i + 1
      Final_Des_de
Final_Procediment
```

```
Proc. Mostrar ( vect[TAM]: real, n: enter)
Variables   i : enter
Inici
      Des_de i ← 0 fins a n-1 fer
            Escriure (vect[i])
            i ← i + 1
      Final_Des_de
Final_Procediment
```

```
PROGRAMA PRINCIPAL

Variables
   x[TAM] , y[TAM], z[TAM] : real
   dim : enter
Inici
   Fer
      Escriure("Dona'm la dimensió")
      Llegir(dim)
   Mentre( dim < 1)

   Escriure("Coord. del vector 1r vector")
   Emplena ( x , dim )
   Escriure("Coord. del vector 2n vector")
   Emplena ( y , dim )

   Suma ( x , y , z , dim )

   Mostrar ( z , dim )
Final
```

Arrays: Algorismes (23/27)

- **Esborrar un element d'una posició determinada**
- Per a esborrar un element, l'única cosa que cal fer és moure tots els elements posteriors a una posició anterior i dir que hi ha un element menys.

```
Proc. Esborra (ref v[TAM]: real, ref  n : enter, pos : enter)
Variables
      i : enter
Inici
      Des_de i ← pos fins a n-2 fer
            v[i] ← v[i+1]
            i ← i + 1
      Final_Des_de
      n ←  n -1
Final_Procediment
```

```
PROGRAMA PRINCIPAL

Variables
   vect[TAM] , dada : real
   dim,  i: enter
Inici
   // Llegir dim
   Emplena (vect , dim )
   Escriure("Dona'm la dada que cal esborrar")
   Llegir (dada)
   i ← Cercar ( vect , dim , dada )
   Si i >= 0 aleshores
      Esborrar ( vect , dim , i )
   Si no
      Escriure("La dada no hi és")
   Final_Si
   Mostrar ( vect , dim )
Final
```

- **Inserir un element en una posició determinada**
- Per a inserir un element en una posició, l'única cosa que cal fer és moure una posició tots els elements del vector des del final fins al lloc on volem inserir la dada nova i dir que hi ha un element més.

```
Proc. Inserir (ref v[TAM]: real, ref  n : enter, pos : enter, d : real)
Variables
      i : enter
Inici
      Si n = 0 fer
          v[n] ← d
         n←n+1
      Si no
        Des_de i ← n-1 fins a pos fer
             v[i+1] ← v[i]
             i ← i - 1
         Final_Des_de
         v[ pos ] ← d
         n ←  n +1
      Final_Si
Final_Procediment
```

```
PROGRAMA PRINCIPAL

Variables
   vect[TAM] , dada : real
   dim,  i: enter
Inici
   Emplena (vect , dim )
   Si dim < TAM aleshores
      Fer
         Escriure("Dona'm la dada i la posició")
         Llegir (dada , i )
      Mentre ( i >= dim )
      Inserir ( vect , dim , i , dada)
   Si no
      Escriure("Vector ple, no en caben més...")
   Final_Si
   Mostrar ( vect , dim )
Final
```

Arrays: Algorismes (25/27)

- **Emplena un vector amb elements inserits de manera ordenada**
- Suposem que volem emplenar un vector amb nombres reals de manera ordenada. Per exemple, de menor a major.

```
Proc. InserirOrdenat (ref v[TAM]: real, ref  n : enter, d : real)
Variables
        pos : enter
Inici
        pos ← CercarPosició (v , n, d)

        Si pos >= 0 AND pos < TAM aleshores
                Inserir ( v , n , pos , d)
        Final_Si

Final_Funció
```

```
Func. CercarPosició (v[TAM]: real, n: enter, d: real) : enter
Variables
    i  : enter
Inici
    i ←  0
    Mentre i < n AND d > v[i] fer
        i ←  i + 1
    Final_Mentre
    CercarPosició ←  i
Final_Funció
```

```
PROGRAMA PRINCIPAL

Variables
    vect[TAM] , dada : real
    dim: enter
    e : booleà
Inici
    dim ← 0

    Mentre dim <= TAM fer
        Escriure("Dona'm la dada")
        Llegir (dada )
        InserirOrdenat ( vect , dim , dada)
    Final_Mentre

    Mostrar ( vect , dim )
Final
```

Arrays: Cadenes i strings (26/27)

- **Cadenes**
- Anomenarem cadenes aquells vector en els quals s'emmagatzemen caràcters.
- Una característica especial és que només s'utilitza una part del vector. Per delimitar-la, es col·loca un marcador al final dels caràcters utilitzats (per exemple, en C/C++ és el caràcter amb valor 0, sovint representat '\0').
- Per exemple: Suposem una cadena de 10 caràcters en què volem desar la paraula 'hola'. Dels 10 caràcters possibles, només n'emprarem 4. Així doncs, el contingut del vector serà:

0	1	2	3	4	5	6	7	8	9
'h'	'o'	'l'	'a'	'\0'	?	?	?	?	?

- La declaració d'una cadena es fa com la d'un vector en què guardarem caràcters:
  Nom [Grandària] : caràcters
- Com que una cadena solament és un cas especial de vector, les operacions sobre les cadenes es realitzaran element a element, de manera similar als vectors, però tenint en compte el marcador de final de cadena ('\0').

Arrays: Cadenes i strings (27/27)

- **Strings**

- Malgrat que els llenguatges de programació tinguen funcions específiques per a realitzar operacions sobre les cadenes, aquestes són força incòmodes.
- Per aquesta raó, quasi tots els llenguatges ofereixen un nou tipus per a treballar amb cadenes de caràcters. Aquest tipus és, fonamentalment, un vector de caràcters amb un enter associat i un conjunt de funcions que permeten fer operacions sobre les cadenes. L'enter representa la llargària actual de la cadena (açò és, el nombre de caràcters que emmagatzema). Anomenarem aquest tipus de dades *string*.
- El concepte de string és equivalent al de cadena, però les funcions que permeten realitzar operacions sobre aquests són més senzilles i, a més a més, la llargària de la cadena no es determina amb cap caràcter especial. Per tant, sempre que treballem amb variables de tipus string ho farem a través de les operacions definides pel llenguatge de programació per a treballar sobre els string.

- Aquest tema el veurem en pràctiques.

Arrays: Cadenes i strings (27/27)

- Funcions per a treballar amb els string
- // núm. de caràcters d'una paraula
  i = paraula.length();
- // inserir una paraula a la posició 3 de la frase
  frase.insert(3, paraula);
- // unir paraula i "hola" i emmagatzemar el resultat en frase
  frase = paraula + "hola";
- // afegir paraula al final de frase
  frase += paraula;
- // esborra 7 caràcters de frase des de la posició 3
  frase.erase (3,7);
- // reemplaça 6 caràcters de frase, des de la posició 1, per la cadena paraula
  frase.replace (1, 6, paraula);
- // cerca paraula com una subcadena dins de frase des de la posició inici, retorna la posició on la troba (o -1 si no la troba)
  i = frase.find(paraula, inici);
  i = frase.find(paraula); //comença des de la posició 0
- // retorna la subcadena de 3 caràcters des de la posició 5 de frase
  paraula = frase.substr(5,3);
- // conversió de string a cadena de caràcters de C
  cadena = frase.c_str();

Estructures o registres (1/14)

- Les estructures o registres són agrupacions heterogènies, contigues i estàtiques.

- Els elements continguts en l'agrupació poden ser de diferent tipus, s'anomenen 'camps d'informació', i sovint fan referència a la informació sobre un objecte concret.

- Exemples:
  - Una persona (Nom, edat, DNI,...)
  - Un llibre (Títol, autor, preu, disponible/exhaurit,...)

Estructures o registres (2/14)

- La declaració d'una estructura o registre es fa de la manera següent:

```
Registre Nom
   Camp1 : Tipus1
   Camp2 : Tipus2
    ...
FINAL_Registre
```

- Nom és el nom que rebrà la nova estructura.
- Tipus1 és el tipus de dades del primer camp d'informació i Camp1 és el nom del primer camp d'informació.
- Tipus2 és el tipus de dades emmagatzemat en el segon camp d'informació i Camp2 el nom del segon camp d'informació.
- I així per a tots els camps d'informació que volem tindre.

Estructures o registres (3/14)

- La manera de declarar una estructura en C/C++ és:

```
struct Nom
{
    Tipus1 Camp1;
    Tipus2 Camp2;
    ...
};
```

- Els registres o estructures defineixen nous tipus de dades que podem utilitzar dins del nostre programa.

- Per tant, després de declarar l'estructura, podem fer-la servir i declarar variables d'aquest nou tipus (de la mateixa manera que declarem variables de tipus enter o real):
  - NomVariable: Nom
  - En C/C++: struct Nom NomVariable o Nom NomVariable

Estructures o registres (4/14)

- **Assignació**

- La manera d'accedir a cadascun dels camps és mitjançant l'operador d'accés '.'
  - NomVariable.Camp1

- El tractament de cadascun d'aquests camps depén del seu tipus (si és un enter, l'assignarem i tractarem com un enter; si és un vector, com un vector; si és un string, com un string,...).
  - NomVariable.Camp1 ← Valor

- Exemples: Si p és una variable que defineix una persona,
  - p.Nom ← "Pep"
  - p.Edat ← 45
  - p.DNI ← 45678987
  - p.Sexe ← 'H'

Estructures o registres (5/14)

- Exemple: Creació i assignació de valors a persones:

```
Registre persona
    Nom: string
    Edat: enter
    DNI: enter
    Sexe: caràcter
FINAL_Registre
```

```
per : persona
…
Escriure (“Dona’m el nom: \n”)
Llegir (per.Nom)
Escriure (“Dona’m l’edat: \n”)
Llegir (per.Edat)
Escriure (“Dona’m el DNI: \n”)
Llegir (per.DNI)
Escriure (“Dona’m el sexe: \n”)
Llegir (per.Sexe)
…
```

Estructures o registres (6/14)

- En C/C++ podem donar un valor inicial als camps d'un registre quan creem la variable:

```
struct Data
{
    int dia;
    string mes;
    int any;
};

Data f = {1,"Novembre",2010};
```

Estructures o registres (7/14)

- **Estructures imbricades:**
- Les estructures es poden imbricar, és a dir, un dels camps d'una estructura pot ser una altra estructura:

```
Registre alumne
    InfoPersonal : persona
    DataNaixement : data
    Curs : enter
FINAL_Registre
```

- Així mateix, també podem definir arrays de registres:
  - NomVector[Grandària] : NomEstructura

Estructures o registres (8/14)

- Les estructures sí que es poden assignar, com els strings o qualsevol tipus de dades simple.

- Una assignació d'estructures és equivalent a una assignació de cadascun dels components.
  - hui, ahir : data;
  - ahir ← hui;

- **Nota**: l'assignació és vàlida a condició que l'estructura no contingui un vector com un dels camps.

- Les estructures es passen com a paràmetres exactament igual que qualsevol altre tipus simple, açò és, per valor o per referència.

- Com que podem fer assignacions entre estructures, les funcions també poden retornar estructures.

Estructures o registres (9/14)

- Exemple: De dues persones, dir quina és la de major edat.

```
FUNCIÓ QuiÉsMajor(p1 : persona, p2 : persona) : persona
Variables:
    major : persona
Inici
   SI (p1.edat >= p2.edat) ALESHORES
      major ← p1
   SI NO
      major ← p2
   FINAL_SI
   QuiÉsMajor ← major
FINAL_Funció
FUNCIÓ LlegirPer() : persona
Variables:
    p : persona
Inici
   Escriure(“Dona’m el nom, edat, sexe”)
   Llegir ( p.Nom, p.edat, p.sexe)
   LlegirPer ← p
FINAL_Funció
PROCEDIMENT MostrarPer(p : persona)
Inici
   Escriure(“Les dades són:”)
   Escriure ( p.nom, p.edat, p.sexe)
FINAL_Procediment
```

```
Registre persona
    nom: string
    edat: enter
    sexe: caràcter
FINAL_Registre


PROGRAMA PRINCIPAL

Variables
   per1, per2, m : persona
Inici
   per1 ← LlegirPer()
   per2 ← LlegirPer()

   m ← QuiÉsMajor(per1, per2)

   MostrarPer (m )
Final
```

Estructures o registres (10/14)

- Exemple(v1): Dir quin és “l’alumne” de major edat entre dos

```
FUNCIÓ AlumneMajor(p1 : alumne, p2 : alumne) : alumne
Variables:
major : alumne
Inici
    SI (p1.dades.edat >= p2.dades.edat) aleshores
        major ← p1
    SI NO
        major ← p2
    FINAL_SI
    AumneMajor ← major
FINAL_Funció
PROCEDIMENT LlegirAlum( ref  a : alumne)
Inici
    Escriure(“Dona’m el nom, edat, sexe i curs”)
    Llegir ( a.dades.nom, a.dades.edat, a.dades.sexe)
    Llegir( a.curs)
FINAL_Procediment

PROCEDIMENT MostrarAlum (a : alumne)
Inici
    Escriure(“Les dades són:”,  a.dades.nom)
    Escriure( a.dades.edat, a.dades.sexe, a.curs)
FINAL_Procediment
```

```
Registre persona
    nom: string
    edat: enter
    sexe: caràcter
FINAL_Registre

Registre alumne
    dades: persona
    curs: enter
FINAL_Registre

PROGRAMA PRINCIPAL

Variables
   a1, a2, m : alumne
Inici
   LlegirAlum( a1 )
   LlegirAlum( a2 )

   m ← AlumneMajor(a1, a2)

   MostrarAlum (m )
Final
```

Estructures o registres (11/14)

- Exemple(v2): Fent servir les funcions de “persona”

```
FUNCIÓ AlumneMajor(p1 : alumne, p2 : alumne) : alumne
Variables:
major : alumne
Inici
    major.dades ← QuiÉsMajor( p1.dades, p2.dades )
    AumneMajor ← major
FINAL_Funció

PROCEDIMENT LlegirAlum( ref  a : alumn)
Inici
    a.dades ← LlegirPer()
    Escriure (“Dona’m el curs”)
    Llegir( a.curs)
FINAL_Procediment

PROCEDIMENT MostrarAlum (a : alumne)
Inici
    MostrarPer( a.dades )
    Escriure ( “I el curs és”, a.curs)
FINAL_Procediment
```

```
Registre persona
    nom: string
    edat: enter
    sexe: caràcter
FINAL_Registre

Registre alumne
    dades: persona
    curs: enter
FINAL_Registre

PROGRAMA PRINCIPAL

Variables
   a1, a2, m : alumne
Inici
   LlegirAlum( a1 )
   LlegirAlum( a2 )

   m ← AlumneMajor(a1, a2)

   MostrarAlum (m )
Final
```

Estructures o registres (12/14)

- Exemple: Programa que determina qui és el major de 10 alumnes (mitjançant les funcions/procediments anteriors).

```
PROGRAMA PRINCIPAL

Variables
        a[10], m : alumne
        i : enter
Inici
        Des_de1 i ← 0 fins a 9 fer
                LlegirAlum( a[i] )
                i ← i + 1
        Final_Des_de1

        m ← a[0]

        Des_de2  i ← 1 fins a 9 fer
                m ← AlumneMajor(m, a[i] )
                i ← i + 1
        Final_Des_de2

        MostrarAlum (m )
Final
```

```
Registre persona
    nom: string
    edat: enter
    sexe: caràcter
FINAL_Registre

Registre alumne
    dades: persona
    curs: enter
FINAL_Registre
```

Estructures o registres (13/14)

- Exemple: Programa que determina qui és el major de 10 alumnes (fent servir una funció que retorne la posició en què es troba).

```
FUNCIÓ AlumneMajor( v[10] : alumne) : enter
Variables:
major, pos, i : enter
Inici
      pos ← 0
      major ← v[0].dades.edat

      Des_de  i ← 1 fins a 9 fer
      SI (v[i].dades.edat >= major) aleshores
            major ← v[i].dades.edat
            pos ← i
      FINAL_SI
          i ← i + 1
    Final_Des_de

      AlumneMajor ← pos
FINAL_Funció
```

```
Registre persona
    nom: string
    edat: enter
    sexe: caràcter
FINAL_Registre

Registre alumne
    dades: persona
    curs: enter
FINAL_Registre
PROGRAMA PRINCIPAL
Variables
   a[10] : alumne
   i , pos: enter
Inici
   Des_de i ← 0 fins a 9 fer
      LlegirAlum( a[i] )
      i ← i + 1
   Final_Des_de

   pos ← AlumneMajor( a )
   MostrarAlum ( a[pos] )
Final
```

Estructures o registres (14/14)

- **Exercici**: Modifica el programa anterior perquè torne qui és el major de N alumnes, on N és un valor introduït a l'inici pel teclat i que no podrà ser major que 100.

- **Versió 1: emprant només les estructures anteriors de persona i alumne**

FUNCIÓ AlumMajor1(v[10] : alumne , t : enter) : enter

Registre persona
    nom: string
    edat: enter
    sexe: caràcter
FINAL_Registre

Registre alumne
    dades: persona
    curs: enter
FINAL_Registre

- **Versió 2: emprant l'estructura llista_alum**

FUNCIÓ AlumMajor2(llis: llista_alum) : enter

Registre llista_alum
    v[TAM]: alumne
    num : enter
FINAL_Registre

Introducció a les estructures dinàmiques (1/11)

- **Punters**
- En quasi tots els llenguatges de programació hi ha un tipus de dades simples anomenat 'punter'.
- Un punter és un tipus de dades que és capaç de guardar una adreça de memòria.
- Les adreces de memòria serveixen a l'ordinador per a saber en quin lloc de la memòria es troba exactament una determinada informació. En general, qualsevol variable duu associada una adreça de memòria.

- Els operadors bàsics amb els punters són:

& Obté l'adreça de memòria on es troba una variable a partir del nom de la variable.

***** Obté el contingut (valor) emmagatzemat en una determinada posició de la memòria.

`malloc` Reserva espai en memòria per a una determinada informació i retorna l'adreça de memòria on comença aquest espai.

`free` Allibera l'espai reservat per a una informació (diu a l'ordinador que aquest espai pot ser usat de nou per altres usuaris o programes).

Introducció a les estructures dinàmiques (2/11)

- Amb el tipus de dades punter i els registres podem construir agrupacions d'elements que poden crèixer en memòria a mesura que augmente la informació que volem emmagatzemar a l'ordinador. Llavors, no cal fixar la capacitat màxima de l'agrupació quan escrivim el programa.

- En general, podrem construir agrupacions dinàmiques mitjançant la inclusió d'un punter com el camp d'un registre. Així doncs, a través d'aquest punter podrem accedir als altres elements de l'agrupació.

- Una definició d'un registre d'aquest tipus en C tindrà la forma següent:

```
struct Node
{
   Valor Info;
   struct Node* Seg;
};
```

Introducció a les estructures dinàmiques (3/11)

- Aquest tipus de registres, que contenen un punter, s'anomenen sovint NODES i són el fonament de les agrupacions dinàmiques, ja que gràcies als punters podem enllaçar tots els elements d'informació que volem (limitats solament per la capacitat física de la memòria de l'ordinador). A més a més, mitjançant la instrucció 'malloc' podem reservar memòria per a elements nous a mesura que els necessitem.

- La manera d'accedir als camps d'un registre a partir d'un punter és fent servir l'operador 'contingut' (*) i l'operador d'accés ('.')

```
struct Node * p_aux;

(*p_aux).Info / (*p_aux).Seg;
```

- Aquesta combinació és equivalent a l'operador d'accés en punters '->' (un signe menys seguit d'un major):

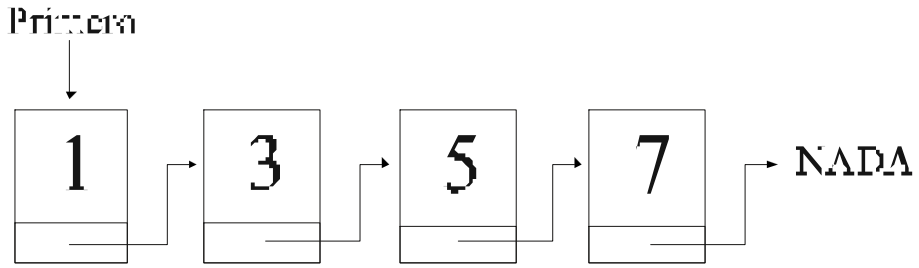
```
p_aux->Info / p_aux->Seg
```

Intro. estr. dinàmiques: Llistes enllaçades (4/11)

- Una llista és una agrupació d'elements entre els que n'hi ha un que podem anomenar 'primer' i a partir del qual hom pot accedir a la resta d'elements, un rere l'altre.
- Les llistes enllaçades es defineixen mitjançant:

1. Una estructura de tipus node.
2. Un punter que marca el primer element, a partir del qual hom pot accedir a qualsevol altre element de l'agrupació.

- La idea a la qual volem arribar és la següent: La llista de nombres 1, 3, 5, 7 en forma enllaçada tindria l'aspecte següent:



- Primer tindrem el punter que assenyala el primer element de la llista. Si fem servir el punter que es troba en cadascun dels nodes, és possible accedir al següent element des d'un element qualsevol.
- A 'RES' en C/C++ se l'anomena 'NULL'.

Intro. estr. dinàmiques: Llistes enllaçades (5/11)

- Operacions
- *Creació*
- Quan creem una llista, inicialment no tindrà cap element. La creació d'una llista durà associada: la declaració del tipus Node, d'una variable que siga capaç de guardar on es troba el primer dels elements i l'assignació que determina que no hi ha cap element en la llista.

```
struct Node
{
Valor Info;
struct Node * Seg;
};
...
struct Node * Primer;
...
Primer = NULL;
```

- La declaració del tipus Node s'ha de fer al començament del programa, després dels "includes" i els "defines".
- La variable de tipus punter es declararà juntament amb la resta de variables declarades. L'assignació a 'NULL' es localitzarà en aquella posició del programa on volguem iniciar la variable.

Intro. estr. dinàmiques: Llistes enllaçades (6/11)

- *Cerca d'un element*
- En les llistes dinàmiques solament podem realitzar cerques sequencials, açò és, que partisquen del primer element i miren successivament si l'element és el que trobem o hem de passar al següent.

```
bool Cercar (struct Node * primer, int x)
{
   struct Node * p_aux;
     bool trobat;

   p_aux = primer;
   while ( (p_aux != NULL) && (p_aux->Info != x) )
          p_aux = p_aux->Seg

   if (p_aux == NULL)
        trobat = false;
     else
        trobat = true;

   return trobat;
}
```

Intro. estr. dinàmiques: Llistes enllaçades (7/11)

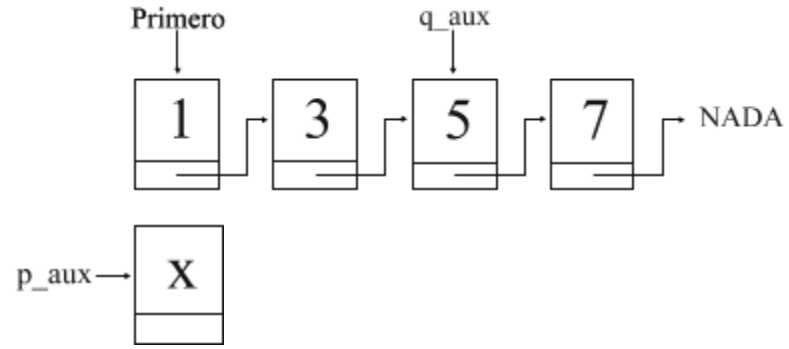
- *Inserció d'un element davant del primer*
- Suposem que volem inserir l'element 'x' davant del primer:

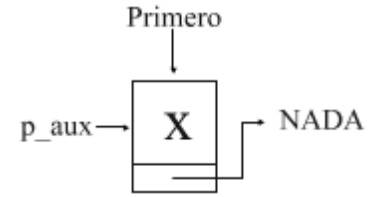
```
struct Node * p_aux;
...
/* Reservem memòria per al nou element */
p_aux = malloc (sizeof (struct Node) );

/* N'actualitzem la informació */
p_aux->Info = x;

/* Enllacem l'element en la llista */
p_aux->Seg = primer;

/* El primer és ara el que hem creat nou */
primer = p_aux;
```

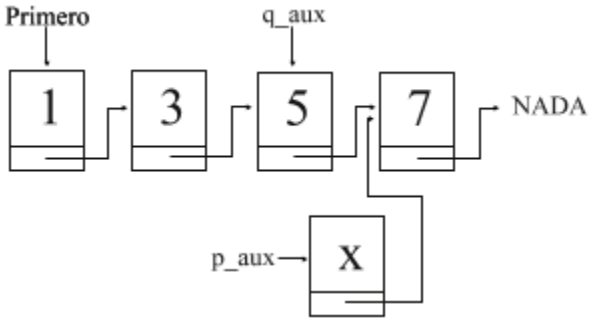
Intro. estr. dinàmiques: Llistes enllaçades (8/11)

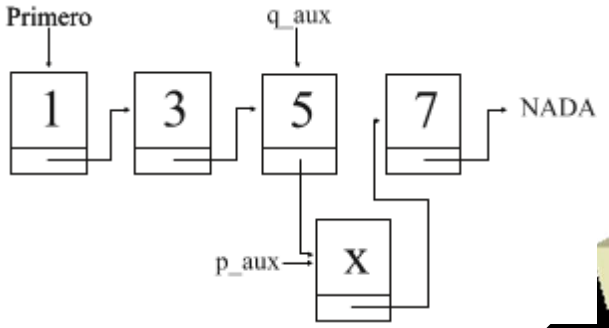
- *Inserció d'un element darrere d'un altre*
- Suposem que volem inserir l'element 'x' darrere d'un altre el punter del qual anomenem 'q_aux':

```
void InserirDarrere (struct Node * primer, struct Node * q_aux, Valor x)
{
   struct Node * p_aux;


   p_aux = malloc (sizeof (struct Node) );
   p_aux->Info = x;

   /* Si no hi havia elements, solament tindrem el que acabem de crear */
   if (primer == NULL)
   {
      p_aux->Seg = NULL;
      primer = p_aux;
   }
   else
   {
      p_aux->Seg = q_aux->Seg;

      q_aux->Seg = p_aux;

   }
```





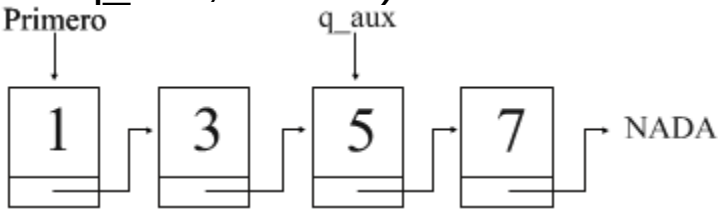


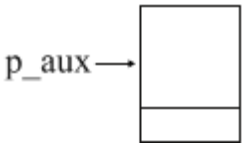


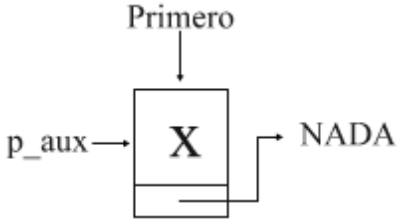
Intro. estr. dinàmiques: Llistes enllaçades (9/11)

- *Inserció d'un element davant d'un altre*
- Suposem que volem inserir l'element 'x' davant d'un altre el punter del qual anomenem 'q_aux':

```
void InserirDavant (struct Node * primere, struct Node * q_aux, Valor x)
{
   struct Nod * p_aux;


   p_aux = malloc (sizeof (struct Node) );


   /* Si no hi havia elements, solament tindrem el que acabem de crear */
   if (primer == NULL)
   {
       p_aux->Info = x
      p_aux->Seg = NULL;
      primer = p_aux;
   }
   else
   {

            (vegeu diapositiva següent...)

   }
}
```

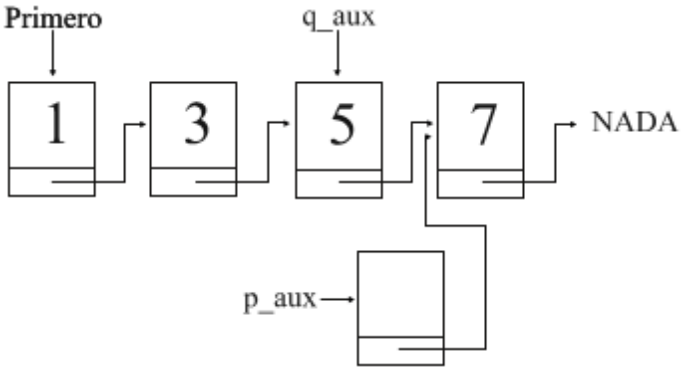




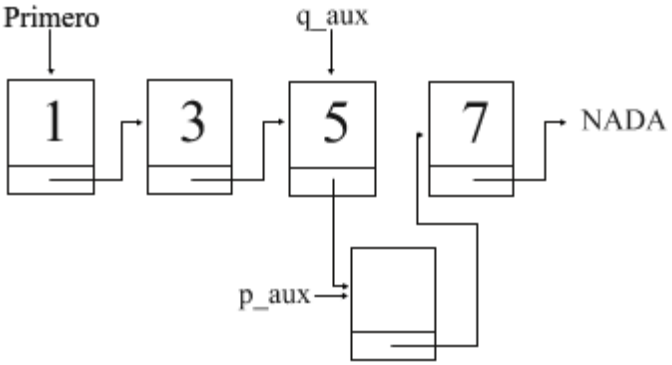


Intro. estr. dinàmiques: Llistes enllaçades (10/11)

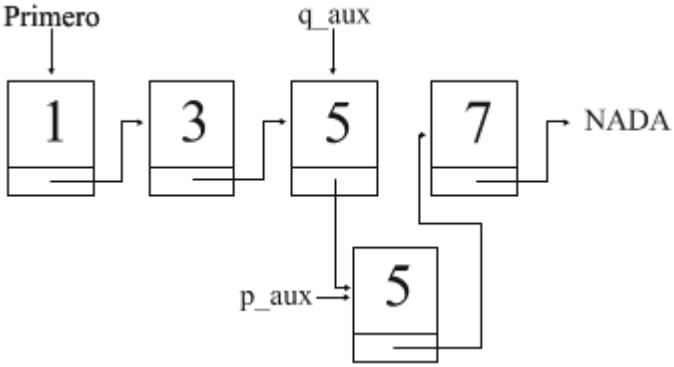
```
else
{
        p_aux->Seg = q_aux->Seg;
```



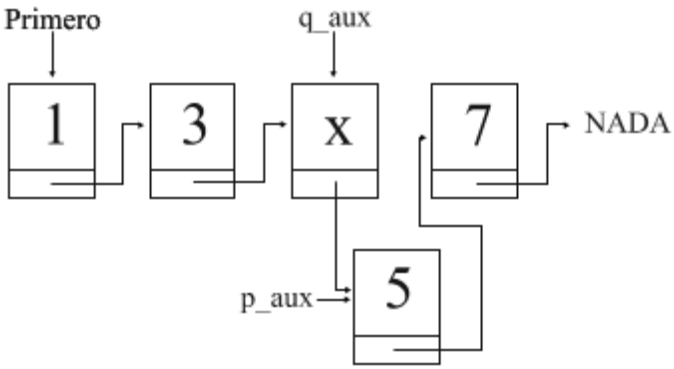

```
        q_aux->Seg = q_aux;
```




```
        p_aux->Info = q_aux->Info;
```




```
        q_aux->Info = x;
```



Intro. estr. dinàmiques: Llistes enllaçades (11/11)

- *Avantatges i desavantatges de les agrupacions estàtiques enfront de les agrupacions dinàmiques:*
  - Més facilitat per a inserir i eliminar elements en les agrupacions dinàmiques.
  - Més fàcil i ràpid fer cerques en vectors.
  - Si el nombre d'elements és conegut o no variarà molt → les agrupacions estàtiques seran millor.
  - Si el nombre d'elements pot ser qualsevol → les agrupacions dinàmiques seran millor.
  - En general, per a un mateix nombre d'element les agrupacions dinàmiques ocuparan més memòria que les estàtiques (per cadascun dels elements cal emmagatzemar també el punter al node següent).