

FITXERS I GESTIÓ DINÀMICA DE LA MEMÒRIA.

FITXERS I

GESTIÓ DINÀMICA DE LA MEMÒRIA.

TEORIA.

TEORIA.

FONAMENTS DE PROGRAMACIÓ II

CURS: 2024-25

GRAUS: GEI, GEI-Biotec

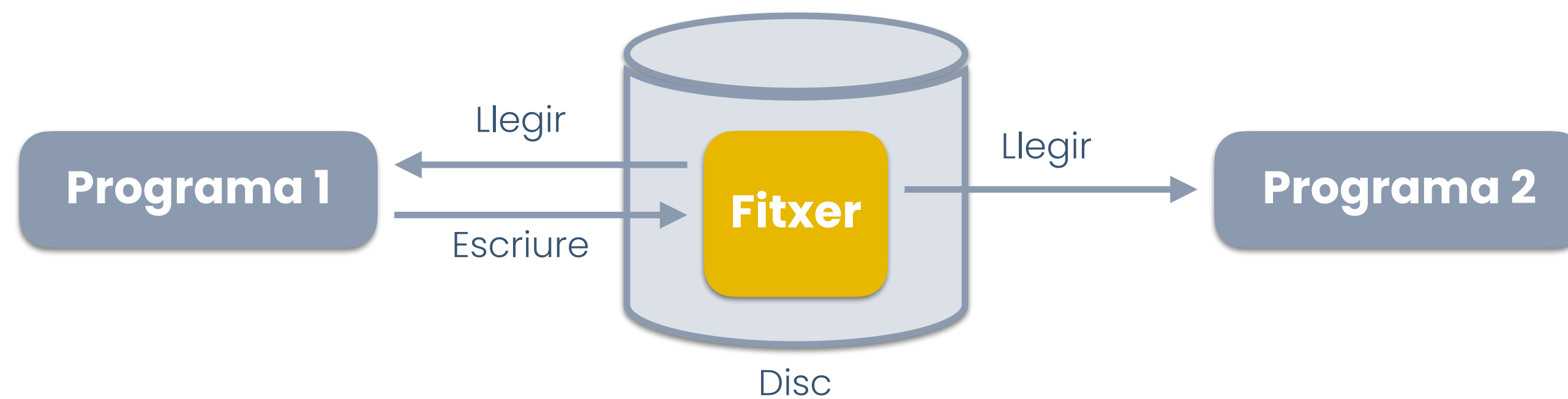
UNIVERSITAT ROVIRA I VIRGILI

FITXERS

Persistència de dades

Necessitat

- Quan s'acaba l'execució del programa, les dades que contenen les variables es perden. Sovint convé que les dades siguin persistents, és a dir, que es puguin desar per tornar-les a utilitzar més endavant. Per tant, necessitarem saber com **escriure dades a fitxer**.
- A més, moltes vegades necessitarem treballar amb dades que ja existeixen (que no generem a través del nostre programa: resultats de simulacions anteriors, dades públiques, etc). Per tant, necessitarem poder **llegir dades de fitxer**.



Persistència de dades

Treballar amb fitxers

- **Fitxers físics:**

- Els fitxers que contenen la informació desada al dispositiu d'emmagatzematge (disc) els anomenem fitxers físics.
- Són una estructura que conté les dades, gestionada pel sistema operatiu.
- Els fitxers físics estan identificats per un **nom** [i una **extensió**]



Fitxer físic

Persistència de dades

Treballar amb fitxers

- **Fitxers físics:**

- Els fitxers que contenen la informació desada al dispositiu d'emmagatzematge (disc) els anomenem fitxers físics.
- Són una estructura que conté les dades, gestionada pel sistema operatiu.
- Els fitxers físics estan identificats per un **nom** [i una **extensió**]



Fitxer físic

- **Fitxers lògics:**

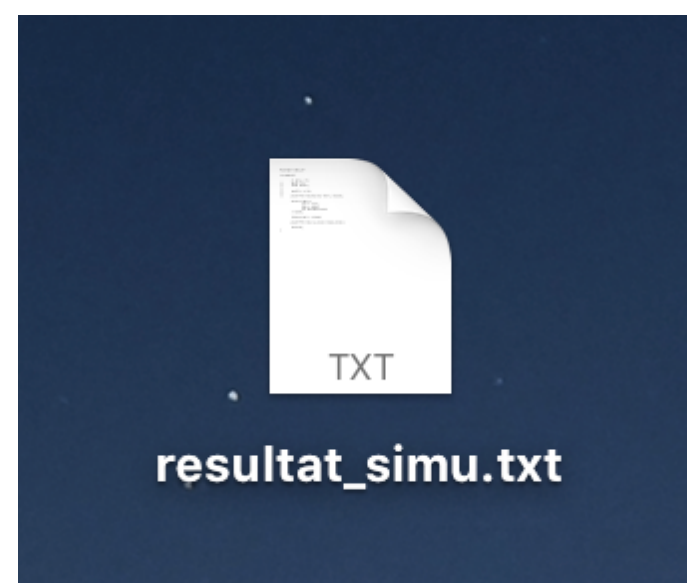
- El nostre objectiu és poder llegir fitxers físics i escriure a fitxers físics des del nostre programa.
- Per fer-ho, necessitem, des del nostre programa, associar el fitxer físic a un fitxer lògic.
- El fitxer lògic és com una **variable** de tipus "fitxer".

Persistència de dades

Treballar amb fitxers

- **Fitxers físics:**

- Els fitxers que contenen la informació desada al dispositiu d'emmagatzematge (disc) els anomenem fitxers físics.
- Són una estructura que conté les dades, gestionada pel sistema operatiu.
- Els fitxers físics estan identificats per un **nom** [i una **extensió**]



Fitxer físic

```
FILE *f;  
f = fopen("resultat_simu.txt","r");
```

Fitxer lògic



- **Fitxers lògics:**

- El nostre objectiu és poder llegir fitxers físics i escriure a fitxers físics des del nostre programa.
- Per fer-ho, necessitem, des del nostre programa, associar el fitxer físic a un fitxer lògic.
- El fitxer lògic és com una **variable** de tipus "fitxer".

Parèntesi tècnic: Estructura de `FILE`



Què conté exactament aquesta estructura?

- **Un buffer de lectura/escriptura:** És a dir, una taula on es guarden temporalment les dades llegides del fitxer o les que s'escriuran a fitxer.
- **Un punter a la posició actual dins del fitxer**, que indica on estem llegint o escrivint dins del fitxer.
 - Així és com sabem quina serà la següent dada llegida o escrita sense haver de llegir tot el fitxer des del principi. Aquesta posició es pot modificar amb funcions com `fseek()` o `rewind()`.
- **Un descriptor de fitxer:** És un número enter assignat pel sistema operatiu quan s'obre un fitxer.
- **Flags d'estat intern:** Són indicadors que el sistema utilitza per saber si hi ha hagut algun problema amb el fitxer.
 - **Flag EOF (End-Of-File):** S'activa quan una funció de lectura intenta llegir més enllà del final del fitxer.
 - Es pot comprovar amb la funció `feof()`.
 - **Flag d'error:** S'activa si hi ha hagut un problema de lectura o escriptura.
 - Es pot comprovar amb la funció `ferror()`.
- El **mode d'obertura**, que indica com es va obrir el fitxer (lectura, escriptura, mode binari, etc.)

Persistència de dades

Procediment

Persistència de dades

Procediment

Cal definir un fitxer lògic i associar-lo a un fitxer físic fent:

```
...  
var      f: fitxer;  
fvar  
inici  
    f := obrir_f(nom_fitxer, mode);  
...
```

Per **obrir**, cal especificar el **nom del fitxer físic** i un **mode d'obertura**.

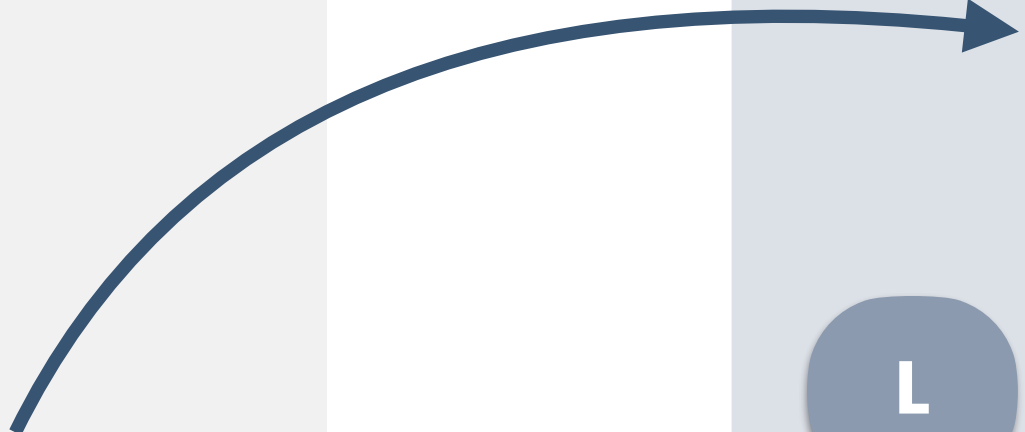
Retorna un punter al fitxer lògic. Si aquest no existeix no es podrà usar el fitxer!

Persistència de dades

Procediment

Cal definir un fitxer lògic i associar-lo a un fitxer físic fent:

```
...  
var      f: fitxer;  
fvar  
inici  
    f := obrir_f(nom_fitxer, mode);  
...
```



Per **obrir**, cal especificar el **nom del fitxer físic** i un **mode d'obertura**.

Retorna un punter al fitxer lògic. Si aquest no existeix no es podrà usar el fitxer!

Modes d'obertura:

L

r

Mode **lectura**

Només podem llegir

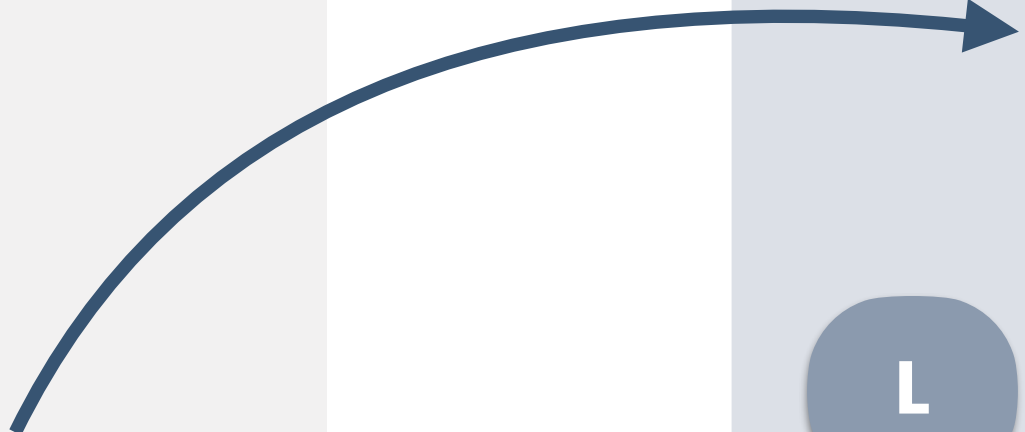
Si fitxer no existeix, retorna un punter nul.

Persistència de dades

Procediment

Cal definir un fitxer lògic i associar-lo a un fitxer físic fent:

```
...  
var      f: fitxer;  
fvar  
inici  
    f := obrir_f(nom_fitxer, mode);  
...
```



Per **obrir**, cal especificar el **nom del fitxer físic** i un **mode d'obertura**.

Retorna un punter al fitxer lògic. Si aquest no existeix no es podrà usar el fitxer!

Modes d'obertura:

L

r

Mode **lectura**

Només podem llegir

Si fitxer no existeix, retorna un punter nul.

E

w

Mode **escriptura**

Només escriptura

Des de la primera posició

Si fitxer no existeix, el crea

Si fitxer existeix, se sobreescriu

Persistència de dades

Procediment

Cal definir un fitxer lògic i associar-lo a un fitxer físic fent:

```
...  
var      f: fitxer;  
fvar  
inici  
    f := obrir_f(nom_fitxer, mode);  
...
```

Per **obrir**, cal especificar el **nom del fitxer físic** i un **mode d'obertura**.

Retorna un punter al fitxer lògic. Si aquest no existeix no es podrà usar el fitxer!

Modes d'obertura:

L

r

Mode **lectura**

Només podem llegir

Si fitxer no existeix, retorna un punter nul.

E

w

Mode **escriptura**

Només escriptura

Des de la primera posició

Si fitxer no existeix, el crea

Si fitxer existeix, se sobreescriu

A

a

Mode **afegir** (append)

Només escriptura

Des de l'última posició

Si fitxer no existeix, el crea

Persistència de dades

Procediment

Cal definir un fitxer lògic i associar-lo a un fitxer físic fent:

```
...  
var      f: fitxer;  
fvar  
inici  
    f := obrir_f(nom_fitxer, mode);  
...
```

Per **obrir**, cal especificar el **nom del fitxer físic** i un **mode d'obertura**.

Retorna un punter al fitxer lògic. Si aquest no existeix no es podrà usar el fitxer!



Per **llegir**, especifiquem el fitxer lògic i la variable on volem desar allò que llegim de fitxer.

```
llegir_f(f, variable);
```

Cal haver obert el fitxer en mode lectura

Modes d'obertura:

L

r

Mode **lectura**

Només podem llegir

Si fitxer no existeix, retorna un punter nul.

E

w

Mode **escriptura**

Només escriptura

Des de la primera posició

Si fitxer no existeix, el crea

Si fitxer existeix, se sobreescriu

A

a

Mode **afegir** (append)

Només escriptura

Des de l'última posició

Si fitxer no existeix, el crea

Persistència de dades

Procediment

Cal definir un fitxer lògic i associar-lo a un fitxer físic fent:

```
...  
var  
    f: fitxer;  
fvar  
inici  
    f := obrir_f(nom_fitxer, mode);  
...
```

Per **obrir**, cal especificar el **nom del fitxer físic** i un **mode d'obertura**.
Retorna un punter al fitxer lògic. Si aquest no existeix no es podrà usar el fitxer!



Per **llegir**, especifiquem el fitxer lògic i la variable on volem desar allò que llegim de fitxer.

```
llegir_f(f, variable);
```

Cal haver obert el fitxer en mode lectura



Per **escriure**, especifiquem el fitxer lògic i la variable que volem escriure a fitxer

```
escriure_f(f, variable);
```

Cal haver obert el fitxer en un mode compatible amb l'escriptura

Modes d'obertura:

L

r

Mode **lectura**

Només podem llegir
Si fitxer no existeix, retorna un punter nul.

E

w

Mode **escriptura**

Només escriptura
Des de la primera posició
Si fitxer no existeix, el crea
Si fitxer existeix, se sobreescriu

A

a

Mode **afegir** (append)

Només escriptura
Des de l'última posició
Si fitxer no existeix, el crea

Persistència de dades

Procediment

Cal definir un fitxer lògic i associar-lo a un fitxer físic fent:

```
...  
var  
    f: fitxer;  
fvar  
inici  
    f := obrir_f(nom_fitxer, mode);  
...
```

Per **obrir**, cal especificar el **nom del fitxer físic** i un **mode d'obertura**.
Retorna un punter al fitxer lògic. Si aquest no existeix no es podrà usar el fitxer!

Per **llegir**, especifiquem el fitxer lògic i la variable on volem desar allò que llegim de fitxer.

llegir_f(f, variable);

Cal haver obert el fitxer en mode lectura

Quan ja no l'hem de fer servir més, el fitxer s'ha de **tancar**, amb el procediment:

tancar_f(f);

Per **escriure**, especifiquem el fitxer lògic i la variable que volem escriure a fitxer

escriure_f(f, variable);

Cal haver obert el fitxer en un mode compatible amb l'escriptura

Modes d'obertura:

L
r

Mode **lectura**
Només podem llegir
Si fitxer no existeix, retorna un punter nul.

E
w

Mode **escriptura**
Només escriptura
Des de la primera posició
Si fitxer no existeix, el crea
Si fitxer existeix, se sobreescriu

A
a

Mode **afegir** (append)
Només escriptura
Des de l'última posició
Si fitxer no existeix, el crea

Persistència de dades

Exemple

- Què fa aquest codi?

```
algorisme exemple és
var
    fit1, fit2: fitxer;
    temp: real;
fvar
inici
    fit1 := obrir_f("temperaturesC.txt", "L");
    fit2 := obrir_f("temperaturesF.txt", "E");

    llegir_f (fit1, temp);
    escriure_f(fit2, temp * 1.8 + 32);

    tancar_f(fit1);
    tancar_f(fit2);

falgorisme
```

Persistència de dades

Exemple

- Què fa aquest codi?

```
algorisme exemple és
var
    fit1, fit2: fitxer;
    temp: real;
fvar
inici
    fit1 := obrir_f("temperaturesC.txt", "L");
    fit2 := obrir_f("temperaturesF.txt", "E");

    llegir_f (fit1, temp);
    escriure_f(fit2, temp * 1.8 + 32);

    tancar_f(fit1);
    tancar_f(fit2);

falgorisme
```

Atenció: si el fitxer no existeix o no es pot obrir, hi haurà errors.

Sempre hem de comprovar-ho!

```
...
    fit1 := obrir_f("temperaturesC.txt", "L");
    si (fit1 = NULL) llavors
        error("No es pot obrir el fitxer");
    fsi
...
```

Persistència de dades

Exemple

- Què fa aquest codi?

```
algorisme exemple és
var
    fit1, fit2: fitxer;
    temp: real;
fvar
inici
    fit1 := obrir_f("temperaturesC.txt", "L");
    fit2 := obrir_f("temperaturesF.txt", "E");

    llegir_f (fit1, temp);
    escriure_f(fit2, temp * 1.8 + 32);

    tancar_f(fit1);
    tancar_f(fit2);


falgorisme
```

Atenció: si el fitxer no existeix o no es pot obrir, hi haurà errors.

Sempre hem de comprovar-ho!

```
...
    fit1 := obrir_f("temperaturesC.txt", "L");
    si (fit1 = NULL) llavors
        error("No es pot obrir el fitxer");
    fsi
...
```

Per comoditat, definim dues noves directives en pseudocodi:

```
acció error (message: taula[] de caràcter) és
inici
    escriure(message);
    sortir();  Equivalent a exit()
facció
```

Persistència de dades

Exemple II

- Sovint voldrem llegir més d'una dada de fitxer (generalment, totes)
- Necessitarem recórrer tot el fitxer per poder llegir totes les dades
- Funció que ens diu si estem al final de fitxer:
final_f (fitxer);
- Aquesta funció, exclusiva de pseudocodi (veurem als laboratoris que en C no es fa servir igual), serveix per saber, abans de llegir, si estem al final de fitxer.

Persistència de dades

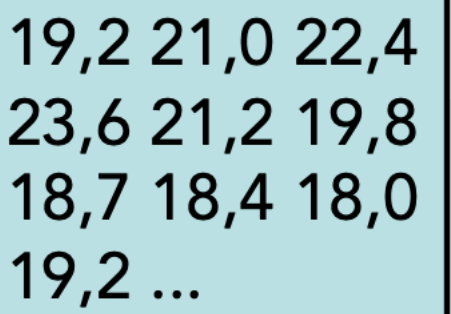
Exemple II

- Sovint voldrem llegir més d'una dada de fitxer (generalment, totes)
- Necessitarem recórrer tot el fitxer per poder llegir totes les dades
- Funció que ens diu si estem al final de fitxer: **final_f (fitxer);**
- Aquesta funció, exclusiva de pseudocodi (veurem als labotaroris que en C no es fa servir igual), serveix per saber, abans de llegir, si estem al final de fitxer.

```
algorisme mostrar_dades és
var
    f_in: fitxer;
    temp: real;
fvar
inici
    f_in := obrir_f("temperatures.txt", "L");
    si (f_in = NULL) llavors
        error("No es pot obrir el fitxer");
    fsi
    mentre (no final_f(f_in)) fer
        llegir_f (f_in, temp);
        escriure("Temperatura: ", temp);
    fmentre

    tancar_f(f_in);

falgorisme
```



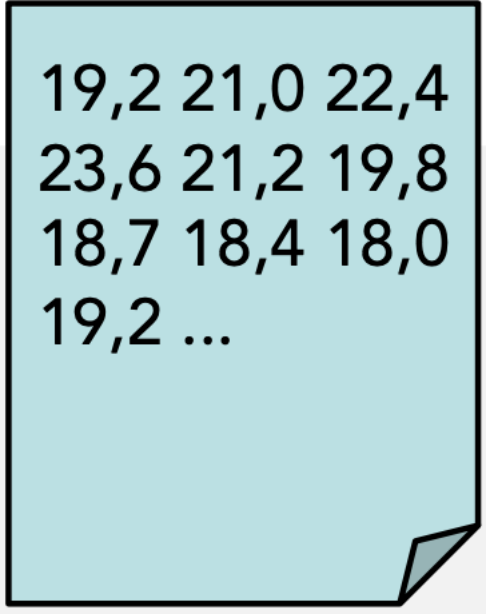
```
19,2 21,0 22,4
23,6 21,2 19,8
18,7 18,4 18,0
19,2 ...
```

temperatures.txt

Persistència de dades

Exemple III

- Calcular la mitjana de les temperatures que llegim de fitxer



temperatures.txt

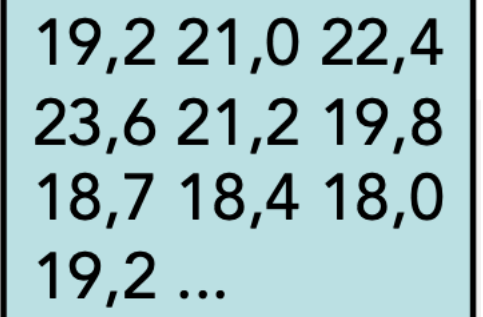
Persistència de dades

Exemple III

- Calcular la mitjana de les temperatures que llegim de fitxer

```
algorisme calcular_mitjana és  
var  
    f_in: fitxer;  
    n_elems: enter;  
    temp, mitjana: real;  
fvar  
  
inici
```

```
falgorisme
```



19,2 21,0 22,4
23,6 21,2 19,8
18,7 18,4 18,0
19,2 ...

temperatures.txt

Persistència de dades

Exemple III

- Calcular la mitjana de les temperatures que llegim de fitxer

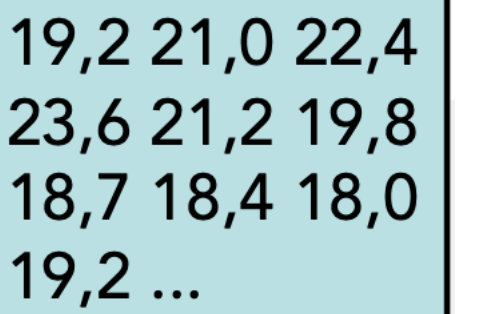
```
algorisme calcular_mitjana és
var
    f_in: fitxer;
    n_elems: enter;
    temp, mitjana: real;
fvar

inici
    f_in := obrir_f("temperatures.txt", "L");
    si (f_in = NULL) llavors
        error("No es pot obrir el fitxer");
    fsi
    n_elems := 0;
    mitjana := 0;

    mentre (no final_f(f_in)) fer
        llegir_f (f_in, temp);
        n_elems := n_elems + 1;
        mitjana := mitjana + temp;
    fmentre

    tancar_f(f_in);

    si (n_elems ≠ 0) llavors
        mitjana := mitjana / n_elems;
    fsi
falgorisme
```



19,2 21,0 22,4
23,6 21,2 19,8
18,7 18,4 18,0
19,2 ...

temperatures.txt

Persistència de dades

Real-world problems

- Molt sovint necessitarem guardar les dades que llegim a una taula, per fer-les servir més d'una vegada
- **Problema:** a l'hora de definir la taula, necessitem dir la **mida** abans de compilar i executar el programa. Com sabem el nombre de dades que llegirem?



```
temperatures: taula[NUM_TEMPS] de real;
```

Persistència de dades

Real-world problems

- Molt sovint necessitarem guardar les dades que llegim a una taula, per fer-les servir més d'una vegada
- **Problema:** a l'hora de definir la taula, necessitem dir la **mida** abans de compilar i executar el programa. Com sabem el nombre de dades que llegirem?

↓

```
temperatures: taula[NUM_TEMPS] de real;
```

- **Opcions** (amb **memòria estàtica**):
 - Reservar un espai màxim i fer acabar la taula amb un sentinella.



22.5
18.9
19.1
21.3
25.6
21.3

temperatures.txt

Persistència de dades

Exemple IV

- Llegir un fitxer de temperatures i desar-los en una taula (amb alocatació estàtica de memòria)

```
algorisme desar_temperatures és
const
    MAX := 200;
    SENTINELLA := -999;
fconst
var
    f_in: fitxer;
    n_elems: enter;
    temp: real;
    vector_temp : taula[MAX] de real;
fvar

inici
    f_in := obrir_f("temperatures.txt", "L");
    si (f_in = NULL) llavors
        error("No es pot obrir el fitxer");
    fsi
    n_elems := 0;
    mentre (no final_f(f_in)) i (n_elems < MAX-1) fer
        llegir_f(f_in, temp);
        vector_temp[n_elems] := temp;
        n_elems := n_elems + 1;
    fmentre
    vector_temp[n_elems] := SENTINELLA;
    tancar_f(f_in);
falgorisme
```

22.5
18.9
19.1
21.3
25.6
21.3

temperatures.txt

Persistència de dades

Exemple IV

- Llegir un fitxer de temperatures i desar-los en una taula (amb alocatació estàtica de memòria)

```
algorisme desar_temperatures és
const
    MAX := 200;
    SENTINELLA := -999;
fconst
var
    f_in: fitxer;
    n_elems: enter;
    temp: real;
    vector_temp : taula[MAX] de real;
fvar

inici
    f_in := obrir_f("temperatures.txt", "L");
    si (f_in = NULL) llavors
        error("No es pot obrir el fitxer");
    fsi
    n_elems := 0;
    mentre (no final_f(f_in)) i (n_elems < MAX-1) fer
        llegir_f(f_in, temp);
        vector_temp[n_elems] := temp;
        n_elems := n_elems + 1;
    fmentre
    vector_temp[n_elems] := SENTINELLA;
    tancar_f(f_in);
falgorisme
```

Per què posem
MAX-1?

22.5
18.9
19.1
21.3
25.6
21.3

temperatures.txt

Persistència de dades

Real-world problems

- Molt sovint necessitarem guardar les dades que llegim a una taula, per fer-les servir més d'una vegada
- **Problema:** a l'hora de definir la taula, necessitem dir la **mida** abans de compilar i executar el programa. Com sabem el nombre de dades que llegirem?

↓

```
temperatures: taula[NUM_TEMPS] de real;
```

- **Opcions** (amb **memòria dinàmica**):
 - Reservar un espai de memòria i redimensionar-lo si en cal més
 - Llegir un cop el fitxer per saber quantes línies té, llegir-lo un segon cop per poder reservar la memòria justa
 - Que la primera línia del fitxer contingui la dada del nombre d'elements

Veurem totes aquestes opcions als Laboratoris

```
# 6
19,2
21,0
22,4
23,6
18,4
19,7
```

temperatures.txt



Dubtes?

GESTIÓ DINÀMICA DE LA MEMÒRIA

Gestió dinàmica de la memòria

Alocatació dinàmica de memòria

- Com hem vist, haver de definir en temps de compilació la mida de les taules és molt **limitant** i **poc eficient**
- És com haver de saber de quina mida seran les prestatgeries abans de construir la casa
- Hi ha una altra manera d'alocar la memòria, i és fer-ho en temps d'execució

Gestió dinàmica de la memòria

Alocatació dinàmica de memòria

- Com hem vist, haver de definir en temps de compilació la mida de les taules és molt **limitant** i **poc eficient**
 - És com haver de saber de quina mida seran les prestatgeries abans de construir la casa
 - Hi ha una altra manera d'alocar la memòria, i és fer-ho en temps d'execució
-
- Pseudocodi:
 - Variable "punter": **punter_a_T**: emmagatzema l'adreça de memòria on està guardat un objecte de tipus "**T**"
 - Operacions:
 - Per demanar un espai de memòria: **reservar_espai(num_elems, mida_elems)**
 - Per alliberar un espai de memòria que ja no fem servir: **alliberar_espai(punter_a_T)**

Gestió dinàmica de la memòria

Alocatació dinàmica de memòria

reservar_espai(num_elems, mida_elems)

- Demana un espai de memòria per desar **num_elems** elements de mida **mida_elems** i et retorna l'adreça d'aquest espai.
- I si no hi ha prou espai? la funció retorna NULL
 - Sempre hem de comprovar-ho!

```
...  
var  
    temperatures: punter_a_real;  
fvar  
    ...  
    temperatures := reservar_espai(N, mida(real));  
    si (temperatures = NULL)  
        error("No hi ha espai");  
    fsi  
  
    ...  
...
```

Gestió dinàmica de la memòria

Alocatació dinàmica de memòria


reservar_espai(num_elems, mida_elems)

- Demana un espai de memòria per desar **num_elems** elements de mida **mida_elems** i et retorna l'adreça d'aquest espai.
- I si no hi ha prou espai? la funció retorna NULL
 - Sempre hem de comprovar-ho!

A pseudocodi farem servir la funció `mida` com a analogia de `sizeof()`.

`mida` retorna el nombre de bytes d'un tipus

```
...  
var  
    temperatures: punter_a_real;  
fvar  
    ...  
    temperatures := reservar_espai(N, mida(real));  
    si (temperatures = NULL)  
        error("No hi ha espai");  
    fsi  
  
    ...  
...
```



Gestió dinàmica de la memòria

Alocatació dinàmica de memòria

`reservar_espai(num_elems, mida_elems)`

- Demana un espai de memòria per desar **num_elems** elements de mida **mida_elems** i et retorna l'adreça d'aquest espai.
- I si no hi ha prou espai? la funció retorna NULL
 - Sempre hem de comprovar-ho!


`alliberar_espai(punter_a_T)`

- Destruïx l'objecte apuntat per P i P valdrà NULL.
- Precaució! Si hi ha més d'un punter que apunta a aquella zona de memòria, **tots** quedaran inservibles

A pseudocodi farem servir la funció `mida` com a analogia de `sizeof()`.

`mida` retorna el nombre de bytes d'un tipus

```
...  
var  
    temperatures: punter_a_real;  
fvar  
    ...  
    temperatures := reservar_espai(N, mida(real));  
    si (temperatures = NULL)  
        error("No hi ha espai");  
    fsi  
    alliberar_espai(temperatures);  
    ...  
...
```



Gestió dinàmica de la memòria

Com declarar una taula amb alocatació dinàmica de la memòria

Gestió dinàmica de la memòria

Com declarar una taula amb alocatció dinàmica de la memòria

Fins ara: alocatció **estàtica** de la memòria

```
algorisme desar_temperatures és  
const  
    NUM_TEMPS = 200;  
fconst  
var  
    temperatures: taula[NUM_TEMPS] de real;  
fvar  
inici  
    temperatures[0] := 17.5;  
    temperatures[1] := 19;  
    temperatures[2] := 21.4;  
    ...  
falgorisme
```

Gestió dinàmica de la memòria

Com declarar una taula amb alocatació dinàmica de la memòria

Fins ara: alocatació **estàtica** de la memòria

```
algorisme desar_temperatures és
const
    NUM_TEMPS = 200;
fconst
var
    temperatures: taula[NUM_TEMPS] de real;
fvar
inici
    temperatures[0] := 17.5;
    temperatures[1] := 19;
    temperatures[2] := 21.4;
    ...
falgorisme
```

Què volem fer? Alocatació **dinàmica** de la memòria

```
algorisme desar_temperatures_v2 és
var
    temperatures: punter_a_real;
    n: enter;
fvar
inici
    ... $ Aquí esbrinar quant espai necessitem
        i emmagatzemar en variable "n"
    temperatures := reservar_espai(n, mida(real));

    si (temperatures = NULL)
        error("No hi ha espai");
    fsi

    temperatures[0] := 17.5;
    temperatures[1] := 19;
    temperatures[2] := 21.4;
    ...
    alliberar_espai(temperatures);
falgorisme
```

Parèntesi tècnic: hi ha dos tipus de memòria



Stack

"A pile of things arranged one on top of another"



Fins ara, totes les variables que hem fet servir (incloent taules), estaven a la pila.

Heap

"An untidy pile or mass of things"



Amb `malloc()`, ara tenim accés al heap. La variable `int *taula` seguirà estant a la pila, però els continguts de la taula estaran al heap.

Diferències entre Stack i Heap



	Stack	Heap
Allocation	Automatic	Manual (malloc, free)
Memory Management	System managed	Programmer managed
Size	Fixed, limited	Dynamic, potentially very large
Allocation Speed	Fast	Slower than stack
Lifetime	Lasts for the scope of the function	Lasts until manually deallocated
Scope	Local to the function	Global to the program
Overhead	Minimal	Greater due to metadata
Reallocation	Not possible	Possible with realloc
Error Handling	No need (automatic)	Required (manual checking) (comparació amb NULL)
Deallocation	Automatic	Manual
Portability	Fully portable	Fully portable
Performance	Very fast access	Slower access
Risk	Stack overflow	Memory leaks, fragmentation

	Memòria estàtica	Memòria dinàmica	VLA (Variable-Length Arrays) 🚫
	<pre>#define N 100 ... int taula[N];</pre>	<pre>int n = 100; int *taula; taula = malloc(n * sizeof(int));</pre>	<pre>int n = 100; ... int taula[n];</pre>
La mida es determina	En temps de compilació	En temps d'execució	En temps d'execució
Mida	Fixa. Un cop decidida no es pot canviar	Variable. Es pot canviar amb realloc() .	Fixa. Un cop decidida no es pot canviar
La taula es guarda	A la pila	Al heap	A la pila
Alliberar memòria	S'allibera automàticament al sortir de la funció que l'ha creat	S'ha d'alliberar manualment amb un free() .	S'allibera automàticament al sortir de la funció que l'ha creat
Portabilitat	Funciona en totes les versions de C	Funciona en totes les versions de C	No ho admeten totes les versions de C (Es va implementar a la c99 però a la següent ja es va deixar de fer servir). Per això no ho fem servir. 🚫

Gestió dinàmica de la memòria

Exercici

- Escriure un programa que demani quants enters volem llegir, els introdueixi per teclat, i els emmagatzemi en un vector, usant alocatació dinàmica de la memòria.

Gestió dinàmica de la memòria

Exercici

- Escriure un programa que demani quants enters volem llegir, els introdueixi per teclat, i els emmagatzemi en un vector, usant allocatació dinàmica de la memòria.

```
algorisme desar_nombres és  
var  
    vector: punter_a_enter;  
    n, numero: enter;  
fvar  
inici  
    escriure("Quants enters vols guardar?");  
    llegir(n);
```

```
falgorisme
```

Gestió dinàmica de la memòria

Exercici

- Escriure un programa que demani quants enters volem llegir, els introdueixi per teclat, i els emmagatzemi en un vector, usant allocatació dinàmica de la memòria.

```
algorisme desar_nombres és  
var  
    vector: punter_a_enter;  
    n, numero: enter;  
fvar  
inici  
    escriure("Quants enters vols guardar?");  
    llegir(n);  
  
    vector = reservar_espai(n, mida(enter));  
    si (vector=NULL)  
        error("No hi ha prou memòria");  
    fsi  
    per(i=0; i<n; i:=i+1)  
        llegir(numero);  
        vector[i] := numero;  
    fper  
    $ Aquí se suposa que fem alguna cosa amb  
    el vector...  
    alliberar_espai(vector);  
falgorisme
```

Gestió dinàmica de la memòria i fitxers **tot junt**

Exercici: Escriure un programa que llegeixi n nombres de fitxer i els guardi en una taula (dinàmica)

```
10
500
203
533
453
134
467
245
357
999
666
```

dades.txt

- Format del fitxer: primera línia: un nombre especificant quantes files de dades hi ha
- La resta de files: les dades a desar

Gestió dinàmica de la memòria i fitxers **tot junt**

Exercici: Escriure un programa que llegeixi n nombres de fitxer i els guardi en una taula (dinàmica)

```
10
500
203
533
453
134
467
245
357
999
666
```

dades.txt

- Format del fitxer: primera línia: un nombre especificant quantes files de dades hi ha
- La resta de files: les dades a desar

```
algorisme llegir_nombres és
var
    vector: punter_a_enter;
    n, numero: enter;
    f_in: fitxer;
fvar
inici
```

falgorisme

Gestió dinàmica de la memòria i fitxers **tot junt**

Exercici: Escriure un programa que llegeixi n nombres de fitxer i els guardi en una taula (dinàmica)

```
10
500
203
533
453
134
467
245
357
999
666
```

dades.txt

- Format del fitxer: primera línia: un nombre especificant quantes files de dades hi ha
- La resta de files: les dades a desar

```
algorisme llegir_nombres és
var
    vector: punter_a_enter;
    n, numero: enter;
    f_in: fitxer;
fvar
inici
    $ Obrir fitxer en mode lectura
    f_in := obrir_f("dades.txt", "L");
    $ Comprovar que s'ha obert bé
    si (f_in = NULL) llavors
        error("No he pogut obrir el fitxer.");
    fsi
```

falgorisme

Gestió dinàmica de la memòria i fitxers **tot junt**

Exercici: Escriure un programa que llegeixi n nombres de fitxer i els guardi en una taula (dinàmica)

```
10
500
203
533
453
134
467
245
357
999
666
```

dades.txt

- Format del fitxer: primera línia: un nombre especificant quantes files de dades hi ha
- La resta de files: les dades a desar

```
algorisme llegir_nombres és
var
    vector: punter_a_enter;
    n, numero: enter;
    f_in: fitxer;
fvar
inici
    $ Obrir fitxer en mode lectura
    f_in := obrir_f("dades.txt", "L");
    $ Comprovar que s'ha obert bé
    si (f_in = NULL) llavors
        error("No he pogut obrir el fitxer.");
    fsi
    $ llegir la primera línia, que és el nombre d'elements
    llegir(f_in, n);
```

falgorisme

Gestió dinàmica de la memòria i fitxers **tot junt**

Exercici: Escriure un programa que llegeixi n nombres de fitxer i els guardi en una taula (dinàmica)

```
10
500
203
533
453
134
467
245
357
999
666
```

dades.txt

- Format del fitxer: primera línia: un nombre especificant quantes files de dades hi ha
- La resta de files: les dades a desar

```
algorisme llegir_nombres és
var
    vector: punter_a_enter;
    n, numero: enter;
    f_in: fitxer;
fvar
inici
    $ Obrir fitxer en mode lectura
    f_in := obrir_f("dades.txt", "L");
    $ Comprovar que s'ha obert bé
    si (f_in = NULL) llavors
        error("No he pogut obrir el fitxer.");
    fsi
    $ llegir la primera línia, que és el nombre d'elements
    llegir(f_in, n);
    $ Reservar l'espai
    vector = reservar_espai(n, mida(enter));
    si (vector = NULL) llavors
        error("No hi ha prou memòria");
    fsi
```

falgorisme

Gestió dinàmica de la memòria i fitxers **tot junt**

Exercici: Escriure un programa que llegeixi n nombres de fitxer i els guardi en una taula (dinàmica)

```
10
500
203
533
453
134
467
245
357
999
666
```

dades.txt

- Format del fitxer: primera línia: un nombre especificant quantes files de dades hi ha
- La resta de files: les dades a desar

```
algorisme llegir_nombres és
var
    vector: punter_a_enter;
    n, numero: enter;
    f_in: fitxer;
fvar
inici
    $ Obrir fitxer en mode lectura
    f_in := obrir_f("dades.txt", "L");
    $ Comprovar que s'ha obert bé
    si (f_in = NULL) llavors
        error("No he pogut obrir el fitxer.");
    fsi
    $ llegir la primera línia, que és el nombre d'elements
    llegir(f_in, n);
    $ Reservar l'espai
    vector = reservar_espai(n, mida(enter));
    si (vector = NULL) llavors
        error("No hi ha prou memòria");
    fsi
    $ Llegir la resta de dades i guardar
    per(i=0; i<n; i:=i+1)
        llegir(f_in, numero);
        vector[i] := numero;
    fper
```

falgorisme

Gestió dinàmica de la memòria i fitxers **tot junt**

Exercici: Escriure un programa que llegeixi n nombres de fitxer i els guardi en una taula (dinàmica)

```
10
500
203
533
453
134
467
245
357
999
666
```

dades.txt

- Format del fitxer: primera línia: un nombre especificant quantes files de dades hi ha
- La resta de files: les dades a desar

```
algorisme llegir_nombres és
var
    vector: punter_a_enter;
    n, numero: enter;
    f_in: fitxer;
fvar
inici
    $ Obrir fitxer en mode lectura
    f_in := obrir_f("dades.txt", "L");
    $ Comprovar que s'ha obert bé
    si (f_in = NULL) llavors
        error("No he pogut obrir el fitxer.");
    fsi
    $ llegir la primera línia, que és el nombre d'elements
    llegir(f_in, n);
    $ Reservar l'espai
    vector = reservar_espai(n, mida(enter));
    si (vector = NULL) llavors
        error("No hi ha prou memòria");
    fsi
    $ Llegir la resta de dades i guardar
    per(i=0; i<n; i:=i+1)
        llegir(f_in, numero);
        vector[i] := numero;
    fper
    $ Tancar fitxer
    tancar_f(f_in);
```

falgorisme

Gestió dinàmica de la memòria i fitxers **tot junt**

Exercici: Escriure un programa que llegeixi n nombres de fitxer i els guardi en una taula (dinàmica)

```
10
500
203
533
453
134
467
245
357
999
666
```

dades.txt

- Format del fitxer: primera línia: un nombre especificant quantes files de dades hi ha
- La resta de files: les dades a desar

```
algorisme llegir_nombres és
var
    vector: punter_a_enter;
    n, numero: enter;
    f_in: fitxer;
fvar
inici
    $ Obrir fitxer en mode lectura
    f_in := obrir_f("dades.txt", "L");
    $ Comprovar que s'ha obert bé
    si (f_in = NULL) llavors
        error("No he pogut obrir el fitxer.");
    fsi
    $ llegir la primera línia, que és el nombre d'elements
    llegir(f_in, n);
    $ Reservar l'espai
    vector = reservar_espai(n, mida(enter));
    si (vector = NULL) llavors
        error("No hi ha prou memòria");
    fsi
    $ Llegir la resta de dades i guardar
    per(i=0; i<n; i:=i+1)
        llegir(f_in, numero);
        vector[i] := numero;
    fper
    $ Tancar fitxer
    tancar_f(f_in);
    $ Quan no l'haguem de fer servir més, alliberar memòria
    alliberar_espai(vector);
falgorisme
```



Dubtes?