

LABORATORIS

PROGRAMACIÓ CIENTÍFICA


Sessió 8: Fitxers i
memòria dinàmica

GESTIÓ DINÀMICA

DE LA MEMÒRIA

Gestió dinàmica de la memòria


Memòria estàtica (en temps de **compilació**)



```
1  #include <stdio.h>
2  #define MAX 100
3  int main(){
4
5      int taula[MAX];
6      int n;
7
8      printf("Quants nombres vols introduir?\n");
9      scanf("%d",&n);
10
11     // llegir nombres i omplir taula
12
13     return 0;
14 }
```

Gestió dinàmica de la memòria

Memòria estàtica (en temps de **compilació**)



```
1  #include <stdio.h>
2  #define MAX 100
3  int main(){
4
5      int taula[MAX];
6      int n;
7
8      printf("Quants nombres vols introduir?\n");
9      scanf("%d",&n);
10
11     // llegir nombres i omplir taula
12
13     return 0;
14 }
```

Quan s'executa el programa...

Gestió dinàmica de la memòria

Memòria estàtica (en temps de **compilació**)

```
1  #include <stdio.h>
2  #define MAX 100
3  int main(){
4
5      int taula[MAX];
6      int n;
7
8      printf("Quants nombres vols introduir?\n");
9      scanf("%d",&n);
10
11     // llegir nombres i omplir taula
12
13     return 0;
14 }
```

Quan s'executa el programa...

Es reserva memòria suficient perquè hi càpiguen 100 enters

Gestió dinàmica de la memòria

Memòria estàtica (en temps de **compilació**)

```
1  #include <stdio.h>
2  #define MAX 100
3  int main(){
4
5     int taula[MAX];
6     int n;
7
8     printf("Quants nombres vols introduir?\n");
9     scanf("%d",&n);
10
11     // llegir nombres i omplir taula
12
13     return 0;
14 }
```

Quan s'executa el programa...

Es reserva memòria suficient perquè hi càpiguen 100 enters

L'usuari introdueix quants nombres vol desar
(O procés similar que esbrini quants nombres desarem)

Gestió dinàmica de la memòria

Memòria estàtica (en temps de **compilació**)

```
1  #include <stdio.h>
2  #define MAX 100
3  int main(){
4
5     int taula[MAX];
6     int n;
7
8     printf("Quants nombres vols introduir?\n");
9     scanf("%d",&n);
10
11     // llegir nombres i omplir taula
12
13     return 0;
14 }
```

Quan s'executa el programa...

Es reserva memòria suficient perquè hi càpiguen 100 enters

L'usuari introdueix quants nombres vol desar
(O procés similar que esbrini quants nombres desarem)

n = 10

Desaprofitem
memòria

Gestió dinàmica de la memòria

Memòria estàtica (en temps de **compilació**)

```
1  #include <stdio.h>
2  #define MAX 100
3  int main(){
4
5     int taula[MAX];
6     int n;
7
8     printf("Quants nombres vols introduir?\n");
9     scanf("%d",&n);
10
11    // llegir nombres i omplir taula
12
13    return 0;
14 }
```

Quan s'executa el programa...

Es reserva memòria suficient perquè hi càpiguen 100 enters

L'usuari introdueix quants nombres vol desar
(O procés similar que esbrini quants nombres desarem)

n = 10

Desaprofitem
memòria

n = 120

No hi caben!

Gestió dinàmica de la memòria

Memòria estàtica (en temps de **compilació**)

```
1  #include <stdio.h>
2  #define MAX 100
3  int main(){
4
5     int taula[MAX];
6     int n;
7
8     printf("Quants nombres vols introduir?\n");
9     scanf("%d",&n);
10
11    // llegir nombres i omplir taula
12
13    return 0;
14 }
```

Quan s'executa el programa...

Es reserva memòria suficient perquè hi càpiguen 100 enters

L'usuari introdueix quants nombres vol desar
(O procés similar que esbrini quants nombres desarem)

n = 10

Desaprofitem
memòria

n = 120

No hi caben!

Una possible solució...

- Reservar la memòria **després** d'esbrinar quanta en necessitem exactament.

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

A teoria:

- Punter: emmagatzema l'adreça de memòria on està guardat un objecte de tipus "**T**"

```
ip: punter_a_T;
```


Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

A teoria:

- Punter: emmagatzema l'adreça de memòria on està guardat un objecte de tipus "**T**"

```
ip: punter_a_T;
```

- Reservar espai: funció que demana un espai de memòria i en retorna l'adreça

```
reservar_espai(num_elems, mida_elems)
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

A teoria:

- Punter: emmagatzema l'adreça de memòria on està guardat un objecte de tipus "**T**"

```
ip: punter_a_T;
```

- Reservar espai: funció que demana un espai de memòria i en retorna l'adreça

```
reservar_espai(num_elems, mida_elems)
```

- Alliberar espai: "des-reserva" l'espai de memòria apuntat pel punter.

```
alliberar_espai(punter_a_T)
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

```
int *taula;
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

```
int *taula;
```


Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

OJO! Com que retorna un punter a void haurem de fer un casting al nostre tipus

```
int *taula;
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

OJO! Com que retorna un punter a void haurem de fer un casting al nostre tipus

OJO2! "size" fa referència al nombre de bytes, no al nombre d'elements!

```
int *taula;
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

OJO! Com que retorna un punter a void haurem de fer un casting al nostre tipus

OJO2! "size" fa referència al nombre de bytes, no al nombre d'elements!

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));
```


Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

OJO! Com que retorna un punter a void haurem de fer un casting al nostre tipus

OJO2! "size" fa referència al nombre de bytes, no al nombre d'elements!

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));

if (taula == NULL){
    printf("ERROR: No hi ha prou espai\n");
    return -1;
}
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'execució)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

OJO! Com que retorna un punter a void haurem de fer un casting al nostre tipus

OJO2! "size" fa referència al nombre de bytes, no al nombre d'elements!

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));

if (taula == NULL){
    printf("ERROR: No hi ha prou espai\n");
    return -1;
}
```

- L'accés a la taula es fa amb normalitat, com si fos una taula estàtica.

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

OJO! Com que retorna un punter a void haurem de fer un casting al nostre tipus

OJO2! "size" fa referència al nombre de bytes, no al nombre d'elements!

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));

if (taula == NULL){
    printf("ERROR: No hi ha prou espai\n");
    return -1;
}
```

```
taula[0] = 100;
taula[1] = 200;
```

- L'accés a la taula es fa amb normalitat, com si fos una taula estàtica.

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'execució)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

OJO! Com que retorna un punter a void haurem de fer un casting al nostre tipus

OJO2! "size" fa referència al nombre de bytes, no al nombre d'elements!

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));

if (taula == NULL){
    printf("ERROR: No hi ha prou espai\n");
    return -1;
}
```

```
taula[0] = 100;
taula[1] = 200;
```

- L'accés a la taula es fa amb normalitat, com si fos una taula estàtica.
- **free**: funció que rep un punter i allibera l'espai de memòria on apunta aquell punter.

```
void free(void *ptr)
```


Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'execució)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

OJO! Com que retorna un punter a void haurem de fer un casting al nostre tipus

OJO2! "size" fa referència al nombre de bytes, no al nombre d'elements!

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));
```

```
if (taula == NULL){  
    printf("ERROR: No hi ha prou espai\n");  
    return -1;  
}
```

```
taula[0] = 100;  
taula[1] = 200;
```

```
free(taula);
```

- L'accés a la taula es fa amb normalitat, com si fos una taula estàtica.
- **free**: funció que rep un punter i allibera l'espai de memòria on apunta aquell punter.

```
void free(void *ptr)
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'execució)

- Objectiu: poder decidir quanta memòria reservem durant l'execució del programa. Com ho fem?

En C (llibreria **stdlib.h**)

- Declarar un punter on guardarem l'adreça de memòria:

```
type *var_name;
```

- **malloc**: funció que reserva un espai de memòria de mida "size" i retorna un punter a aquell espai, o bé NULL si no s'ha pogut reservar

```
void *malloc(size_t size)
```

OJO! Com que retorna un punter a void haurem de fer un casting al nostre tipus

OJO2! "size" fa referència al nombre de bytes, no al nombre d'elements!

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));
```

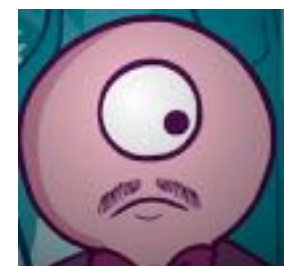
```
if (taula == NULL){  
    printf("ERROR: No hi ha prou espai\n");  
    return -1;  
}
```

```
taula[0] = 100;  
taula[1] = 200;
```

```
free(taula);
```

- L'accés a la taula es fa amb normalitat, com si fos una taula estàtica.
- **free**: funció que rep un punter i allibera l'espai de memòria on apunta aquell punter.

```
void free(void *ptr)
```



OJO CUIDAO!

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Altres funcions relacionades:

```
int *taula;
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Altres funcions relacionades:
- **calloc**: igual que malloc però inicialitza el contingut de la memòria a 0 i rep per paràmetre el nombre d'elements i la mida.

```
void *calloc(size_t nitems, size_t size)
```

```
int *taula;
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'**execució**)

- Altres funcions relacionades:
- **calloc**: igual que malloc però inicialitza el contingut de la memòria a 0 i rep per paràmetre el nombre d'elements i la mida.

```
void *calloc(size_t nitems, size_t size)
```

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));  
taula = (int *) calloc( n, sizeof(int));
```


Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'execució)

- Altres funcions relacionades:
- **calloc**: igual que malloc però inicialitza el contingut de la memòria a 0 i rep per paràmetre el nombre d'elements i la mida.

```
void *calloc(size_t nitems, size_t size)
```

- **realloc**: funció que, donat un punter que apunta a un espai de memòria ja reservat, n'incrementa la mida fins "size"

```
void *realloc(void *ptr, size_t size)
```

size = nova mida que volem, en bytes

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));  
taula = (int *) calloc( n, sizeof(int));
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'execució)

- Altres funcions relacionades:
- **calloc**: igual que malloc però inicialitza el contingut de la memòria a 0 i rep per paràmetre el nombre d'elements i la mida.

```
void *calloc(size_t nitems, size_t size)
```

- **realloc**: funció que, donat un punter que apunta a un espai de memòria ja reservat, n'incrementa la mida fins "size"

```
void *realloc(void *ptr, size_t size)
```

size = nova mida que volem, en bytes

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));  
taula = (int *) calloc( n, sizeof(int));
```

```
taula = (int *) realloc(taula, 2 * n * sizeof(int));
```

Gestió dinàmica de la memòria

Memòria dinàmica (en temps d'execució)

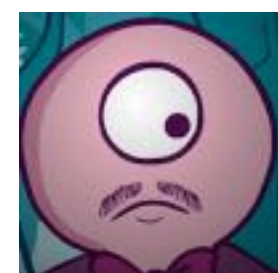
- Altres funcions relacionades:
- **calloc**: igual que malloc però inicialitza el contingut de la memòria a 0 i rep per paràmetre el nombre d'elements i la mida.

```
void *calloc(size_t nitems, size_t size)
```

- **realloc**: funció que, donat un punter que apunta a un espai de memòria ja reservat, n'incrementa la mida fins "size"

```
void *realloc(void *ptr, size_t size)
```

size = nova mida que volem, en bytes



OJO CUIDAO!

Hem de comprovar que la reallocatació s'hagi fet bé:
comprovar que el punter no sigui NULL

```
int *taula;
```

```
taula = (int *) malloc( n * sizeof(int));  
taula = (int *) calloc( n, sizeof(int));
```

```
taula = (int *) realloc(taula, 2 * n * sizeof(int));
```

Recapitulem

Memòria dinàmica en temps d'execució)

VS.

Memòria estàtica (en temps de compilació)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5
6     int *taula;
7     int n;
8     printf("Quants nombres vols introduir?\n");
9     scanf("%d",&n);
10
11     taula = (int *) malloc( n * sizeof(int));
12
13     if (taula == NULL){
14         printf("ERROR: No hi ha prou espai\n");
15         return -1;
16     }
17     free(taula);
18     return 0;
19 }
```

- S'ha de declarar la taula com un punter al tipus
- S'ha de reservar memòria amb malloc o calloc
- S'ha de comprovar sempre que no sigui NULL
- Accessos: igual que sempre, amb []
- S'ha d'alliberar quan haguem acabat
- Es guarda al "heap": podem declarar vectors grans, en funció de la memòria disponible

```
1 #include <stdio.h>
2 #define MAX 100
3 int main(){
4
5     int taula[MAX];
6     int n;
7
8     printf("Quants nombres vols introduir?\n");
9     scanf("%d",&n);
10
11     // llegir nombres i omplir taula
12
13     return 0;
14 }
```


- S'ha de decidir en temps de compilació quina mida tindrà i després no es pot canviar
- Si hi ha menys dades, s'ha de controlar amb un sentinella fins on fem servir
- Es guarda a la pila: Només útil per vectors petits

Exercicis

Exemple de teoria: Tradueix aquest pseudocodi a C usant gestió 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++) fer
        llegir(numero);
        vector[i] := numero;
    fper
    alliberar_memoria(vector);
falgorisme
```



```
#include <stdio.h>
#include <stdlib.h>
```

```
/* Exemple que hem fet a teoria quan parlàvem de gestió
dinàmica de la memòria. Demanar a l'usuari quants nombres vol
guardar*/
```

```
int main(){
    int *vector;
    int n, numero;
```

```
    /* Preguntar quants enters vol guardar (n) */
```

```
    /* Declarar vector de mida n */
```

```
    /* Comprovar que tenim prou memòria */
```

```
    /* Llegir de teclat n valors i els guardem a la taula */
```

```
    /* Imprimir els valors per pantalla */
```

```
    /* Alliberar memòria quan he acabat */
```


```
    return 0;
}
```


Exercicis

Exemple de teoria: Tradueix aquest pseudocodi a C usant gestió 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++) fer
        llegir(numero);
        vector[i] := numero;
    fper
    alliberar_memoria(vector);
falgorisme
```



```
#include <stdio.h>
#include <stdlib.h>
```

```
/* Exemple que hem fet a teoria quan parlàvem de gestió
dinàmica de la memòria. Demanar a l'usuari quants nombres vol
guardar*/
```

```
int main(){
    int *vector;
    int n, numero;

    printf("Quants enters vols guardar?\n");
    scanf("%d",&n);

    /* Declarar vector de mida n */

    /* Comprovar que tenim prou memòria */

    /* Llegir de teclat n valors i els guardem a la taula */

    /* Imprimir els valors per pantalla */


    /* Alliberar memòria quan he acabat */
    return 0;
}
```

Exercicis

Exemple de teoria: Tradueix aquest pseudocodi a C usant gestió 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++) fer
        llegir(numero);
        vector[i] := numero;
    fper
    alliberar_memoria(vector);
falgorisme
```



```
#include <stdio.h>
#include <stdlib.h>
```

```
/* Exemple que hem fet a teoria quan parlàvem de gestió
dinàmica de la memòria. Demanar a l'usuari quants nombres vol
guardar*/
```

```
int main(){
    int *vector;
    int n, numero;

    printf("Quants enters vols guardar?\n");
    scanf("%d",&n);

    vector = (int*) malloc(n * sizeof(int));

    /* Comprovar que tenim prou memòria */

    /* Llegir de teclat n valors i els guardem a la taula */

    /* Imprimir els valors per pantalla */

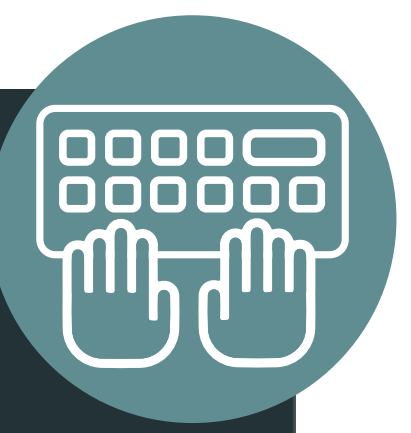
    /* Alliberar memòria quan he acabat */
    return 0;
}
```

Exercicis

Exemple de teoria: Tradueix aquest pseudocodi a C usant gestió 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++) fer
    llegir(numero);
    vector[i] := numero;
  fper
  alliberar_memoria(vector);
falgorisme
```



```
#include <stdio.h>
#include <stdlib.h>
```

```
/* Exemple que hem fet a teoria quan parlàvem de gestió
dinàmica de la memòria. Demanar a l'usuari quants nombres vol
guardar*/
```

```
int main(){
  int *vector;
  int n, numero;

  printf("Quants enters vols guardar?\n");
  scanf("%d",&n);

  vector = (int*) malloc(n * sizeof(int));
  if (vector == NULL){
    printf("ERROR. No hi ha prou memoria");
    return -1;
  }
```

```
/* Llegir de teclat n valors i els guardem a la taula */
```

```
/* Imprimir els valors per pantalla */
```


```
/* Alliberar memòria quan he acabat */
return 0;
}
```

Exercicis

Exemple de teoria: Tradueix aquest pseudocodi a C usant gestió 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++) fer
        llegir(numero);
        vector[i] := numero;
    fper
    alliberar_memoria(vector);
falgorisme
```



```
#include <stdio.h>
#include <stdlib.h>
```

```
/* Exemple que hem fet a teoria quan parlàvem de gestió
dinàmica de la memòria. Demanar a l'usuari quants nombres vol
guardar*/
```

```
int main(){
    int *vector;
    int n, numero;

    printf("Quants enters vols guardar?\n");
    scanf("%d",&n);

    vector = (int*) malloc(n * sizeof(int));
    if (vector == NULL){
        printf("ERROR. No hi ha prou memoria");
        return -1;
    }

    for (int i=0; i<n; i++){
        scanf("%d", &numero);
        vector[i] = numero;
    }

    /* Imprimir els valors per pantalla */

    /* Alliberar memòria quan he acabat */
    return 0;
}
```


Exercicis

Exemple de teoria: Tradueix aquest pseudocodi a C usant gestió 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++) fer
        llegir(numero);
        vector[i] := numero;
    fper
    alliberar_memoria(vector);
falgorisme
```



```
#include <stdio.h>
#include <stdlib.h>

/* Exemple que hem fet a teoria quan parlàvem de gestió
dinàmica de la memòria. Demanar a l'usuari quants nombres vol
guardar*/

int main(){
    int *vector;
    int n, numero;

    printf("Quants enters vols guardar?\n");
    scanf("%d",&n);

    vector = (int*) malloc(n * sizeof(int));
    if (vector == NULL){
        printf("ERROR. No hi ha prou memoria");
        return -1;
    }

    for (int i=0; i<n; i++){
        scanf("%d", &numero);
        vector[i] = numero;
    }
    // Imprimim els continguts del vector
    printf("El teu vector es:\n");
    for (int i=0; i<n; i++){
        printf("%d, ",vector[i]);
    }

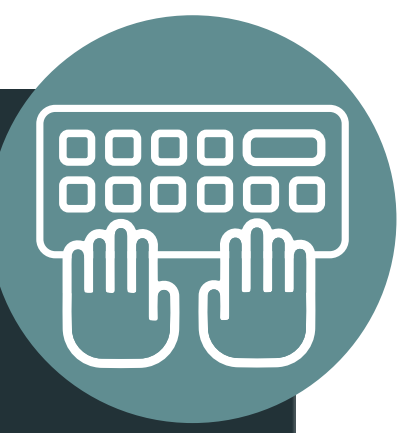
    /* Alliberar memòria quan he acabat */
    return 0;
}
```

Exercicis

Exemple de teoria: Tradueix aquest pseudocodi a C usant gestió 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++) fer
    llegir(numero);
    vector[i] := numero;
  fper
  alliberar_memoria(vector);
falgorisme
```



```
#include <stdio.h>
#include <stdlib.h>
```

```
/* Exemple que hem fet a teoria quan parlàvem de gestió
dinàmica de la memòria. Demanar a l'usuari quants nombres vol
guardar*/
```

```
int main(){
  int *vector;
  int n, numero;

  printf("Quants enters vols guardar?\n");
  scanf("%d",&n);

  vector = (int*) malloc(n * sizeof(int));
  if (vector == NULL){
    printf("ERROR. No hi ha prou memoria");
    return -1;
  }

  for (int i=0; i<n; i++){
    scanf("%d", &numero);
    vector[i] = numero;
  }

  // Imprimim els continguts del vector
  printf("El teu vector es:\n");
  for (int i=0; i<n; i++){
    printf("%d, ",vector[i]);
  }

  // Alliberar memòria quan he acabat
  free(vector);
  return 0;
}
```


FITXERS

I/O de Fitxer

A teoria:

Persistència de dades

Procediment

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

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

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



I/O de Fitxer

Procediment en C:

- Declarar un fitxer lògic.
- Obrir-lo en el mode corresponent
- Escriure-hi o llegir-ne els continguts
- Tancar el fitxer

I/O de Fitxer

Procediment en C:

- Declarar un fitxer lògic.
- Obrir-lo en el mode corresponent
- Escriure-hi o llegir-ne els continguts
- Tancar el fitxer

Tipus "FILE *"

```
FILE *fp;
```

- És un punter a un fitxer. És el nostre fitxer lògic que després associarem al fitxer físic.

I/O de Fitxer

Obrir un fitxer: **fopen**

```
FILE *fopen( const char * filename, const char * mode );
```

```
FILE *fp;
```

I/O de Fitxer

Obrir un fitxer: **fopen**

```
FILE *fopen( const char * filename, const char * mode );
```

```
FILE *fp;
```

```
fp = fopen("test.txt", "w");
```


I/O de Fitxer

Obrir un fitxer: **fopen**

```
FILE *fopen( const char * filename, const char * mode );
```

```
FILE *fp;
```

```
fp = fopen("test.txt", "w");
```

- Modes d'obertura:

r	Opens an existing text file for reading purpose.
w	Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.
r+	Opens a text file for both reading and writing.
w+	Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
a+	Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

I/O de Fitxer

Obrir un fitxer: **fopen**

```
FILE *fopen( const char * filename, const char * mode );
```

```
FILE *fp;
```

```
fp = fopen("test.txt", "w");
```

- Modes d'obertura:

r

Opens an existing text file for reading purpose.

w

Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.

a

Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.

r+

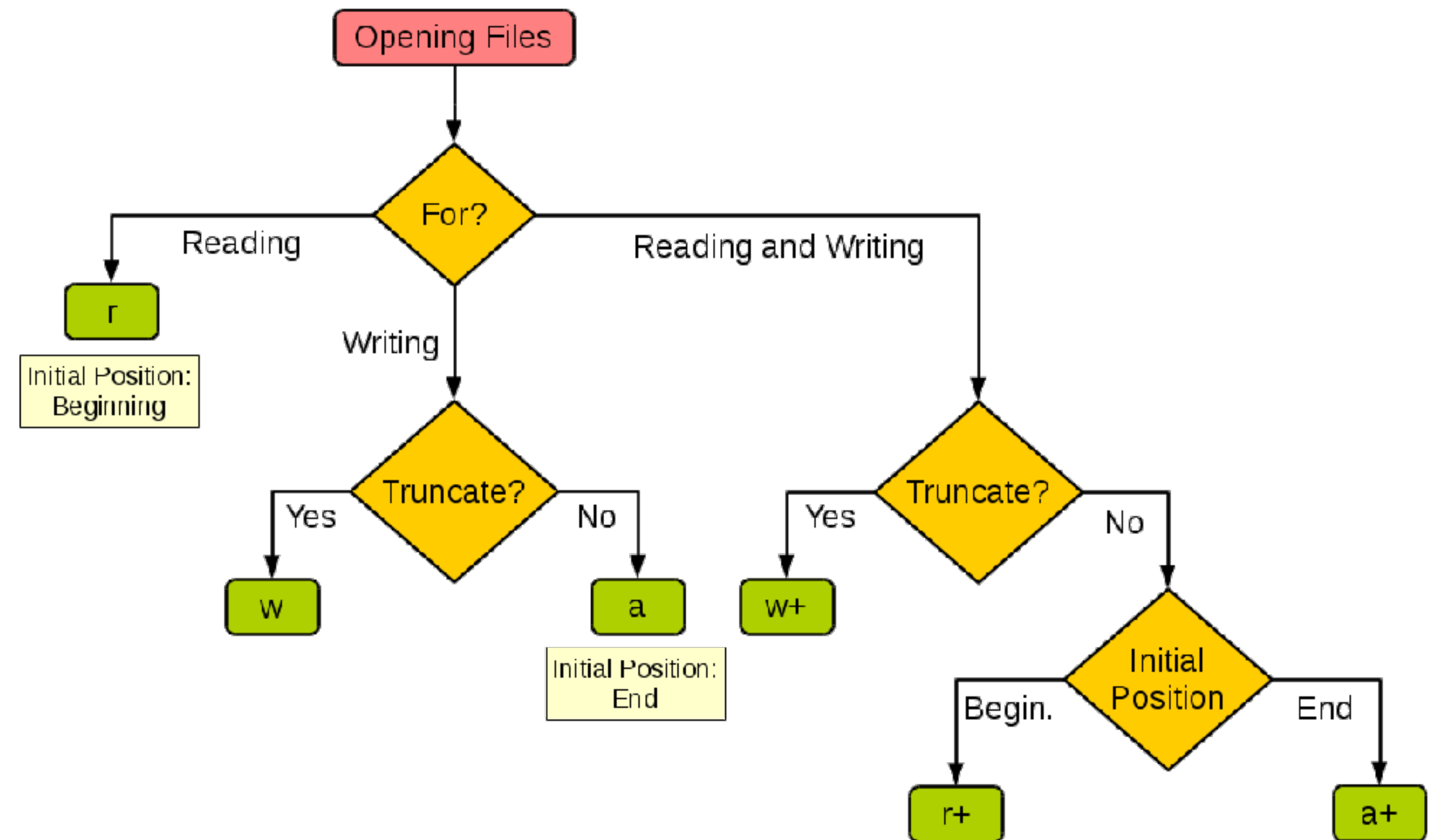
Opens a text file for both reading and writing.

w+

Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.

a+

Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.



I/O de Fitxer

Escriptura a fitxer

- Hem d'haver obert el fitxer en un mode compatible amb escriptura.
- Escrivim com amb printf però amb la funció **fprintf**, passant per paràmetre el punter al fitxer

```
int fprintf(FILE *fp, const char *format, ...)
```

```
FILE *fp;
```

```
fp = fopen("test.txt", "w");
```

I/O de Fitxer

Esriptura a fitxer

- Hem d'haver obert el fitxer en un mode compatible amb escriptura.
- Escrivim com amb printf però amb la funció **fprintf**, passant per paràmetre el punter al fitxer

```
int fprintf(FILE *fp, const char *format, ...)
```

Punter al fitxer lògic
que hem obert

```
FILE *fp;
```

```
fp = fopen("test.txt", "w");
```

I/O de Fitxer

Esriptura a fitxer

- Hem d'haver obert el fitxer en un mode compatible amb escriptura.
- Escrivim com amb printf però amb la funció **fprintf**, passant per paràmetre el punter al fitxer

```
int fprintf(FILE *fp, const char *format, ...)
```

Punter al fitxer lògic
que hem obert

```
FILE *fp;
```

```
fp = fopen("test.txt", "w");
```

```
fprintf(fp, "Hola, què tal?\n");  
fprintf(fp, "La variable n val %d\n", n);
```

I/O de Fitxer

Tancament de fitxer

- Quan hem acabat d'escriure o llegir, necessitem tancar la connexió entre el fitxer lògic i el fitxer físic.

```
int fclose( FILE *fp );
```

↓
Punter al fitxer lògic
que hem obert

- Fins que no tanquem la connexió no s'escriuran els continguts del fitxer

```
FILE *fp;
```

```
fp = fopen("test.txt", "w");
```

```
fprintf(fp, "Hola, què tal?\n");  
fprintf(fp, "La variable n val %d\n", n);
```

```
fclose(fp);
```


I/O de Fitxer

Exemple complet d'escriptura



```
#include <stdio.h>
#define NOM_FITXER "exemple_escrigure.txt"
#define N 10
/* Programa que demana 10 nombres a l'usuari i els guarda a fitxer */
int main() {

    int taula[N];
    FILE *f_out;

    printf("Introdueix %d enters.\n",N);
    for(int i=0; i<N; i++){
        scanf("%d", &taula[i]);
    }

    /* Obrir el fitxer en mode escriptura i comprovar que
    ha anat bé */

    /* Imprimir la taula a fitxer */

    /* Tancar el fitxer */
    return 0;
}
```

I/O de Fitxer

Exemple complet d'escriptura



```
#include <stdio.h>
#define NOM_FITXER "exemple_escrigure.txt"
#define N 10
/* Programa que demana 10 nombres a l'usuari i els guarda a fitxer */
int main() {

    int taula[N];
    FILE *f_out;

    printf("Introdueix %d enters.\n",N);
    for(int i=0; i<N; i++){
        scanf("%d", &taula[i]);
    }

    // Escriure taula a fitxer
    f_out = fopen(NOM_FITXER, "w");
    if (f_out == NULL){
        printf("ERROR! No s'ha pogut obrir el fitxer\n");
        return -1;
    }

    /* Imprimir la taula a fitxer */

    /* Tancar el fitxer */
    return 0;
}
```

I/O de Fitxer

Exemple complet d'escriptura



```
#include <stdio.h>
#define NOM_FITXER "exemple_escruiure.txt"
#define N 10
/* Programa que demana 10 nombres a l'usuari i els guarda a fitxer */
int main() {

    int taula[N];
    FILE *f_out;

    printf("Introdueix %d enters.\n",N);
    for(int i=0; i<N; i++){
        scanf("%d", &taula[i]);
    }

    // Escriure taula a fitxer
    f_out = fopen(NOM_FITXER, "w");
    if (f_out == NULL){
        printf("ERROR! No s'ha pogut obrir el fitxer\n");
        return -1;
    }

    for(int i=0; i<N; i++){
        fprintf(f_out, "taula[%d] = %d\n", i, taula[i]);
    }

    /* Tancar el fitxer */
    return 0;
}
```

I/O de Fitxer

Exemple complet d'escriptura



```
#include <stdio.h>
#define NOM_FITXER "exemple_escruiure.txt"
#define N 10
/* Programa que demana 10 nombres a l'usuari i els guarda a fitxer */
int main() {

    int taula[N];
    FILE *f_out;

    printf("Introdueix %d enters.\n",N);
    for(int i=0; i<N; i++){
        scanf("%d", &taula[i]);
    }

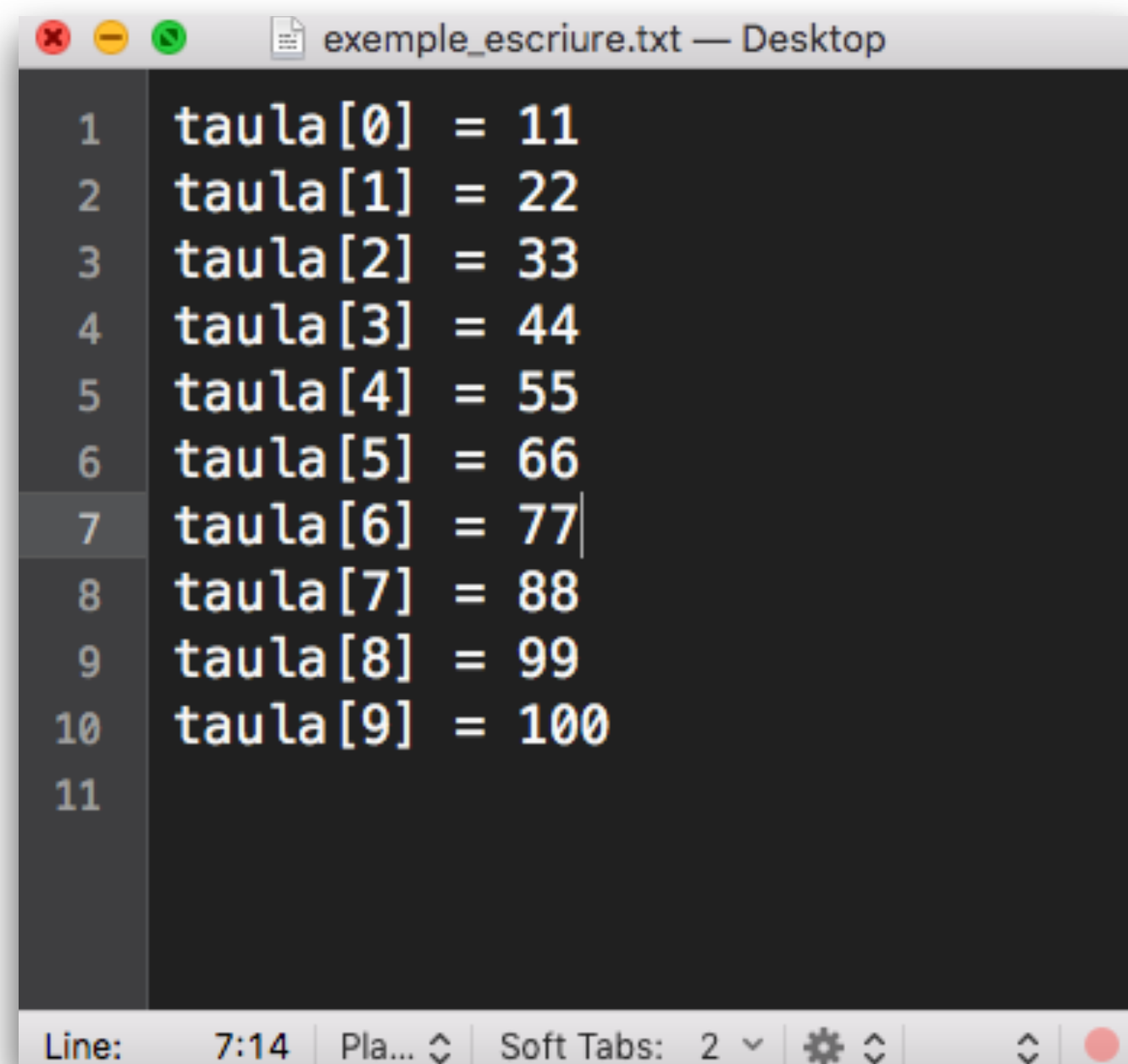
    // Escriure taula a fitxer
    f_out = fopen(NOM_FITXER, "w");
    if (f_out == NULL){
        printf("ERROR! No s'ha pogut obrir el fitxer\n");
        return -1;
    }

    for(int i=0; i<N; i++){
        fprintf(f_out, "taula[%d] = %d\n", i, taula[i]);
    }

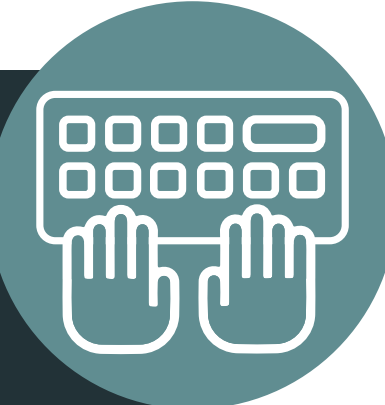
    fclose(f_out);
    return 0;
}
```

I/O de Fitxer

Exemple complet d'escriptura



```
1 taula[0] = 11
2 taula[1] = 22
3 taula[2] = 33
4 taula[3] = 44
5 taula[4] = 55
6 taula[5] = 66
7 taula[6] = 77
8 taula[7] = 88
9 taula[8] = 99
10 taula[9] = 100
11
```



```
#include <stdio.h>
#define NOM_FITXER "exemple_escruiure.txt"
#define N 10
/* Programa que demana 10 nombres a l'usuari i els guarda a fitxer */
int main() {

    int taula[N];
    FILE *f_out;

    printf("Introdueix %d enters.\n",N);
    for(int i=0; i<N; i++){
        scanf("%d", &taula[i]);
    }

    // Escriure taula a fitxer
    f_out = fopen(NOM_FITXER, "w");
    if (f_out == NULL){
        printf("ERROR! No s'ha pogut obrir el fitxer\n");
        return -1;
    }

    for(int i=0; i<N; i++){
        fprintf(f_out, "taula[%d] = %d\n", i, taula[i]);
    }

    fclose(f_out);
    return 0;
}
```

I/O de Fitxer

Lectura de fitxer: **línia a línia**

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- **fgets** llegeix N caràcters fins que troba un final de línia.
Compte! Es guarda el caràcter de final de línia '**\n**'

```
char *fgets(char *str, int n, FILE *stream)
```


I/O de Fitxer

Lectura de fitxer: línia a línia

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- **fgets** llegeix N caràcters fins que troba un final de línia.
Compte! Es guarda el caràcter de final de línia '**\n**'

```
char *fgets(char *str, int n, FILE *stream)
```



String on guardarem
el contingut llegit

I/O de Fitxer

Lectura de fitxer: línia a línia

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- **fgets** llegeix N caràcters fins que troba un final de línia.
Compte! Es guarda el caràcter de final de línia '**\n**'

```
char *fgets(char *str, int n, FILE *stream)
```

String on guardarem
el contingut llegit

Màxim nombre de
caràcters a llegir

I/O de Fitxer

Lectura de fitxer: línia a línia

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- fgets** llegeix N caràcters fins que troba un final de línia.
Compte! Es guarda el caràcter de final de línia '**\n**'

```
char *fgets(char *str, int n, FILE *stream)
```

String on guardarem
el contingut llegit

Màxim nombre de
caràcters a llegir

Fitxer d'on llegim

I/O de Fitxer

Lectura de fitxer: línia a línia

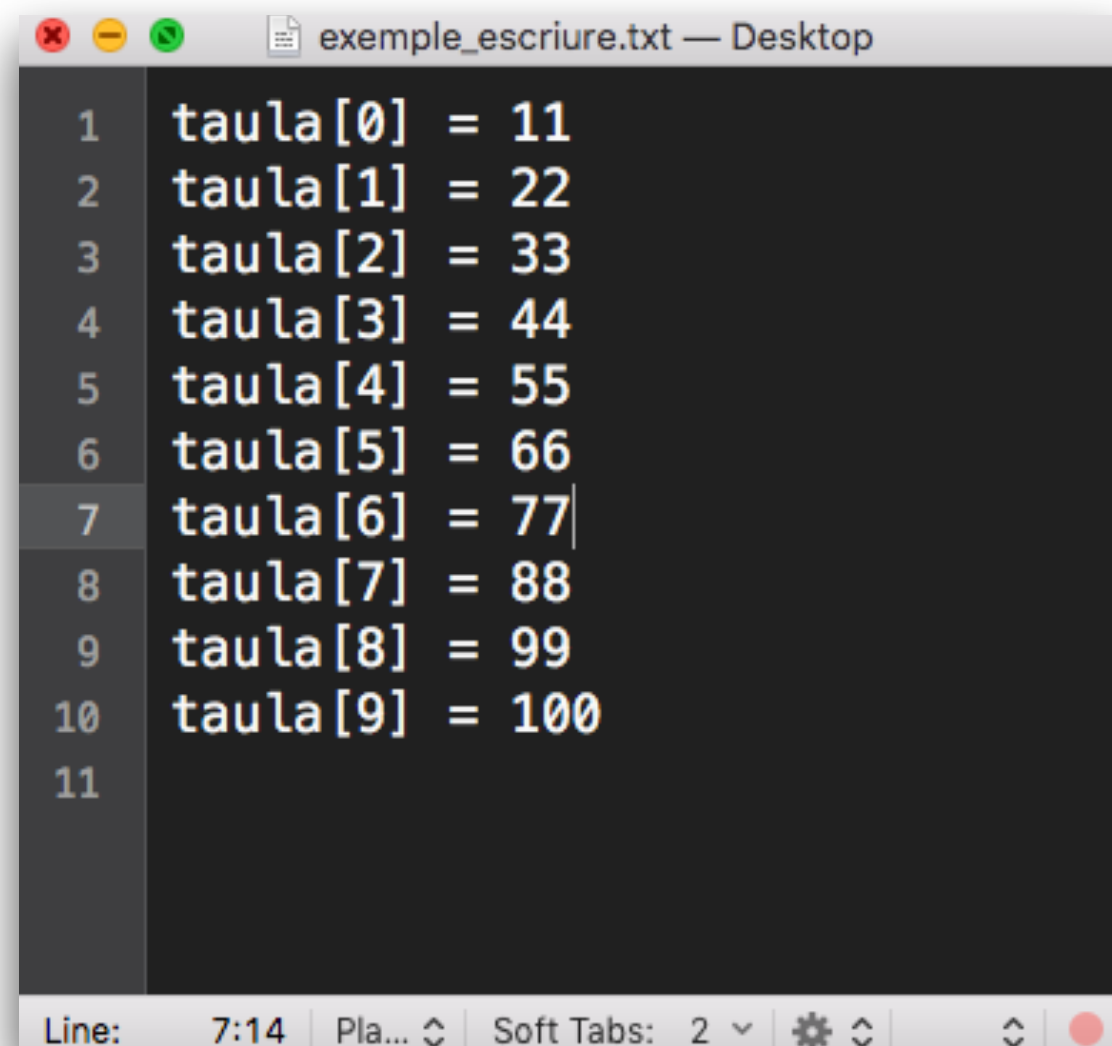
- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- fgets** llegeix N caràcters fins que troba un final de línia. Compte! Es guarda el caràcter de final de línia '**\n**'

```
char *fgets(char *str, int n, FILE *stream)
```

String on guardarem
el contingut llegit

Màxim nombre de
caràcters a llegir

Fitxer d'on llegim



```
#include <stdio.h>

#define MAX_LINIA 30
#define NOM_FITXER "exemple_escruiure.txt"

int main(){

    char s[MAX_LINIA];
    FILE *f_in;

    // Obrir en mode lectura
    f_in = fopen(NOM_FITXER, "r");
    if (f_in == NULL){
        printf("ERROR, no s'ha pogut obrir el fitxer\n");
        return -1;
    }

    return 0;
}
```

I/O de Fitxer

Lectura de fitxer: línia a línia

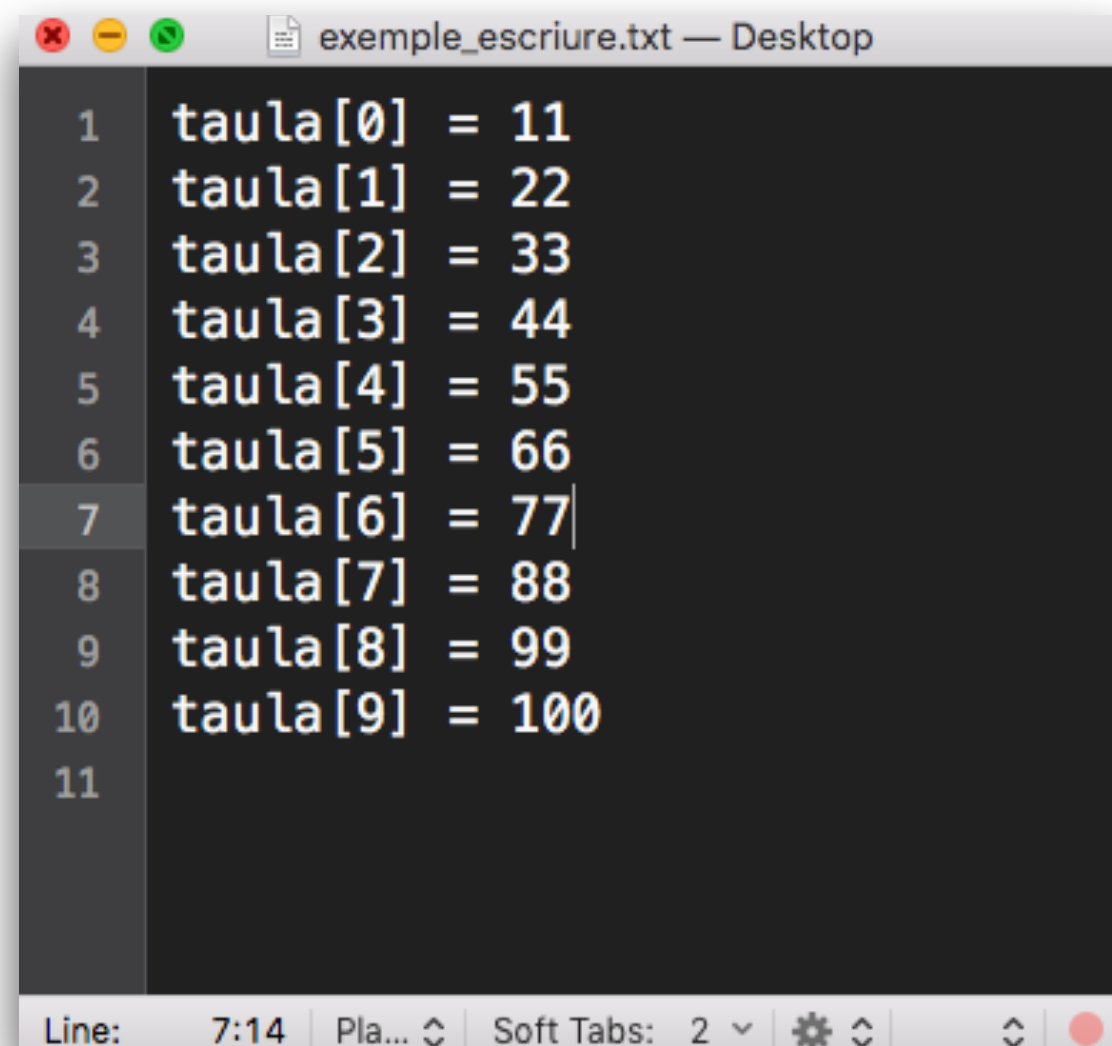
- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- fgets** llegeix N caràcters fins que troba un final de línia. Compte! Es guarda el caràcter de final de línia '**\n**'

```
char *fgets(char *str, int n, FILE *stream)
```

String on guardarem
el contingut llegit

Màxim nombre de
caràcters a llegir

Fitxer d'on llegim



```
#include <stdio.h>

#define MAX_LINIA 30
#define NOM_FITXER "exemple_escruiure.txt"

int main(){

    char s[MAX_LINIA];
    FILE *f_in;

    // Obrir en mode lectura
    f_in = fopen(NOM_FITXER, "r");
    if (f_in == NULL){
        printf("ERROR, no s'ha pogut obrir el fitxer\n");
        return -1;
    }

    // Llegir una línia
    fgets(s, MAX_LINIA, f_in);
    printf("Línia llegida: %s\n", s);

    return 0;
}
```


I/O de Fitxer

Lectura de fitxer: línia a línia

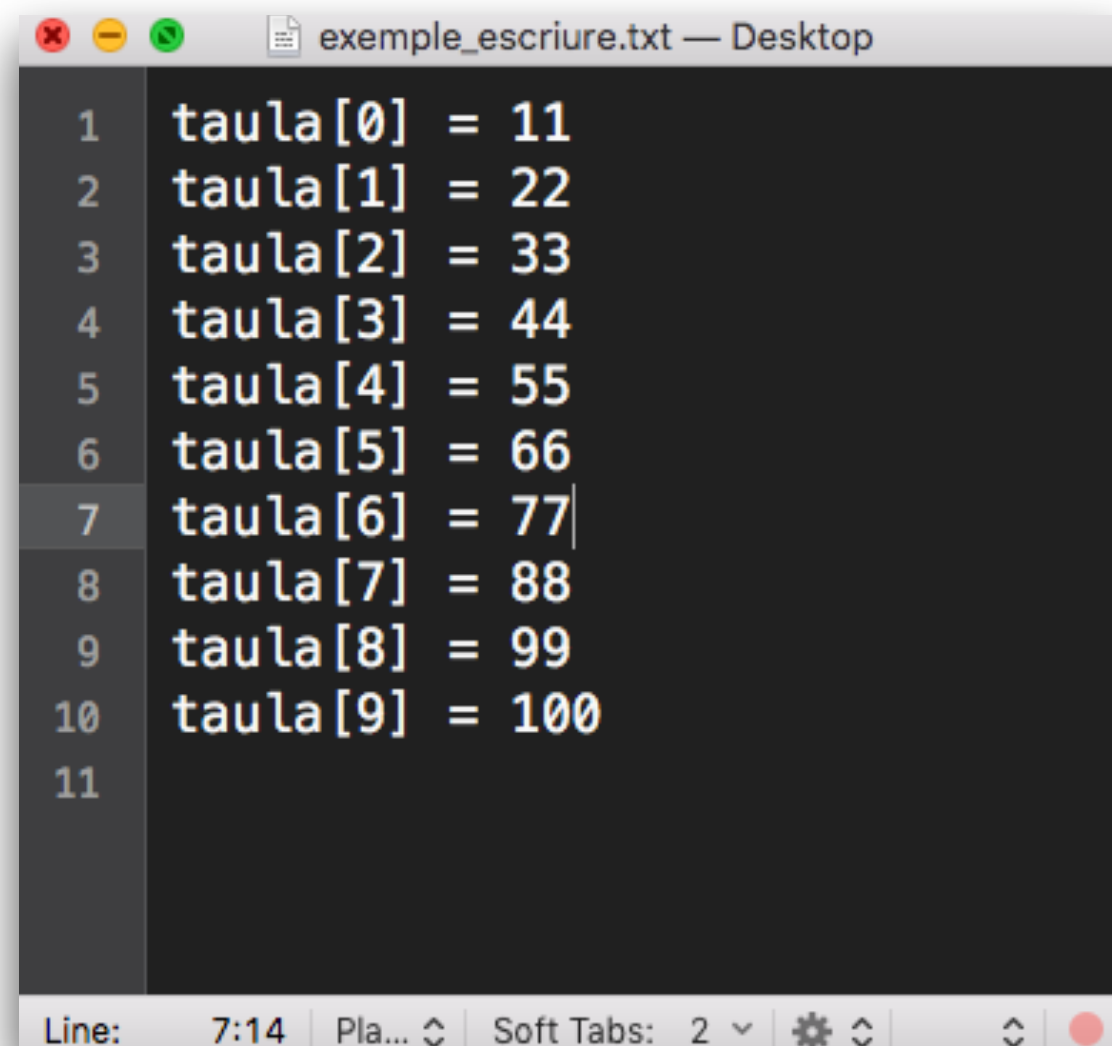
- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- fgets** llegeix N caràcters fins que troba un final de línia. Compte! Es guarda el caràcter de final de línia '**\n**'

```
char *fgets(char *str, int n, FILE *stream)
```

String on guardarem
el contingut llegit

Màxim nombre de
caràcters a llegir

Fitxer d'on llegim



Línia llegida: taula[0] = 11

```
#include <stdio.h>

#define MAX_LINIA 30
#define NOM_FITXER "exemple_escruiure.txt"

int main(){

    char s[MAX_LINIA];
    FILE *f_in;

    // Obrir en mode lectura
    f_in = fopen(NOM_FITXER, "r");
    if (f_in == NULL){
        printf("ERROR, no s'ha pogut obrir el fitxer\n");
        return -1;
    }

    // Llegir una línia
    fgets(s, MAX_LINIA, f_in);
    printf("Línia llegida: %s\n", s);

    return 0;
}
```

I/O de Fitxer

Lectura de fitxer: línia a línia

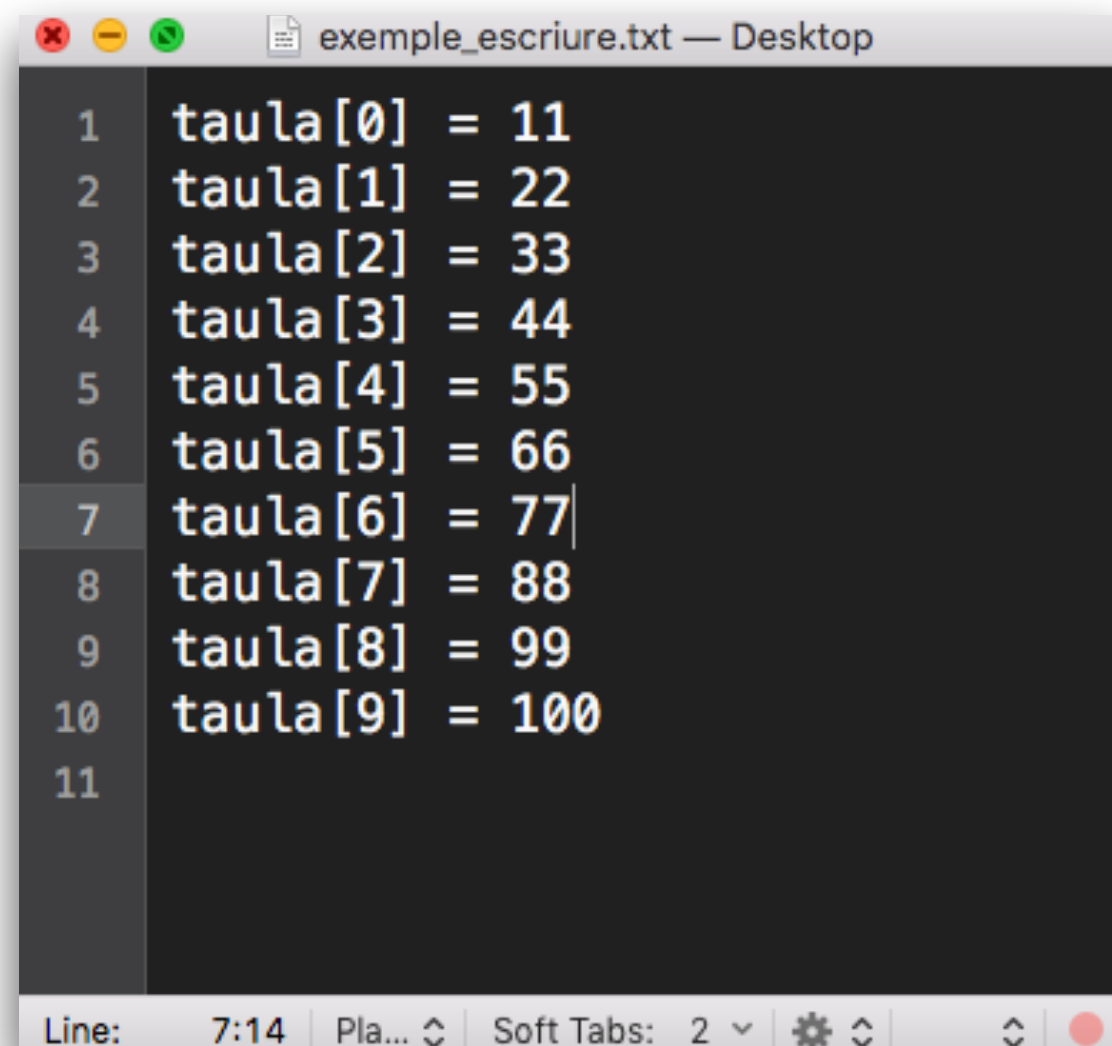
- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- fgets** llegeix N caràcters fins que troba un final de línia. Compte! Es guarda el caràcter de final de línia '\n'

```
char *fgets(char *str, int n, FILE *stream)
```

String on guardarem
el contingut llegit

Màxim nombre de
caràcters a llegir

Fitxer d'on llegim



Linia llegida: taula[0] = 11

```
#include <stdio.h>

#define MAX_LINIA 30
#define NOM_FITXER "exemple_escruiure.txt"

int main(){

    char s[MAX_LINIA];
    FILE *f_in;

    // Obrir en mode lectura
    f_in = fopen(NOM_FITXER, "r");
    if (f_in == NULL){
        printf("ERROR, no s'ha pogut obrir el fitxer\n");
        return -1;
    }

    // Llegir una linia
    fgets(s, MAX_LINIA, f_in);
    printf("Linia llegida: %s\n", s);

    // Què passa si n petita?
    fgets(s, 5, f_in);
    printf("Linia llegida: %s\n", s);

    return 0;
}
```

I/O de Fitxer

Lectura de fitxer: línia a línia

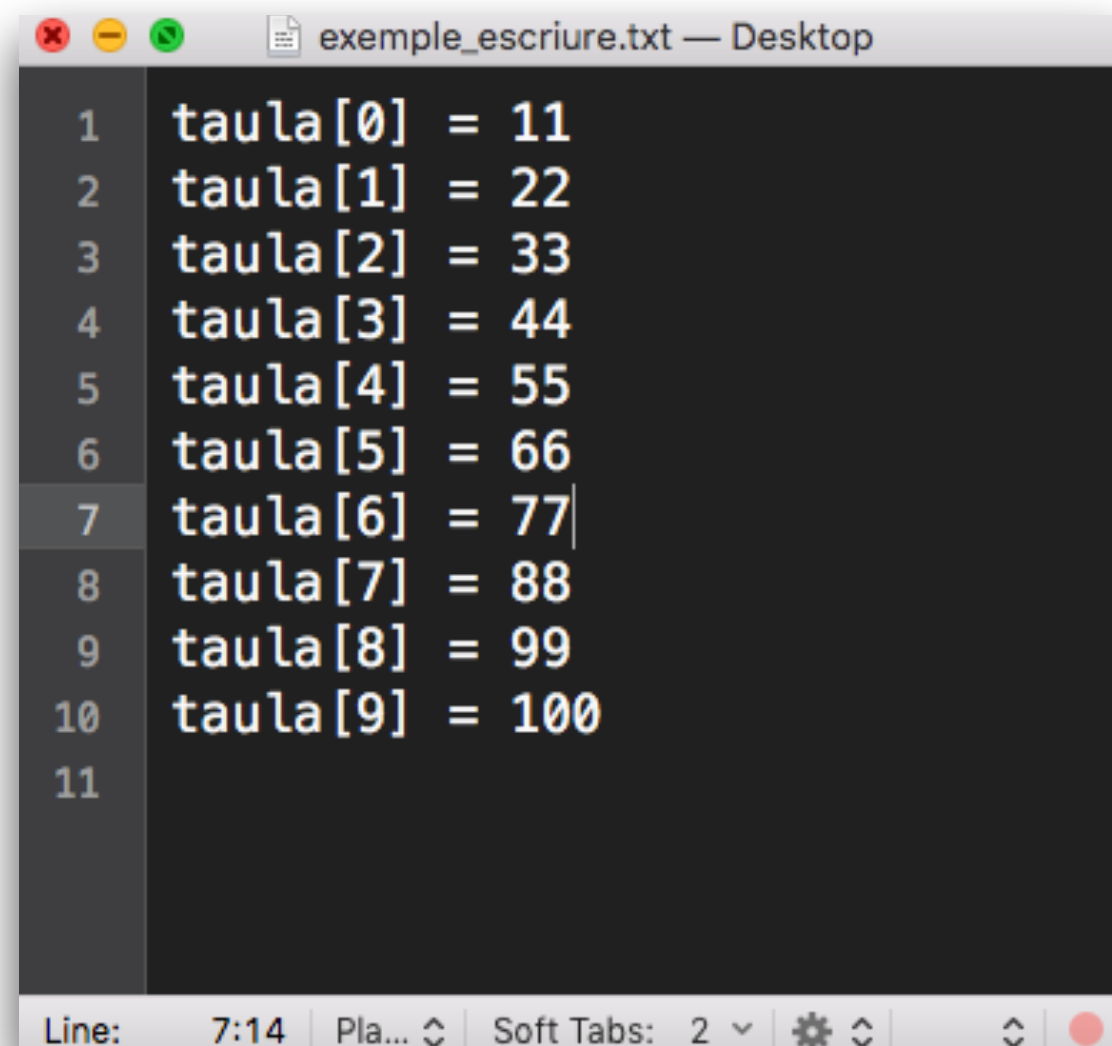
- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- fgets** llegeix N caràcters fins que troba un final de línia. Compte! Es guarda el caràcter de final de línia '\n'

```
char *fgets(char *str, int n, FILE *stream)
```

String on guardarem
el contingut llegit

Màxim nombre de
caràcters a llegir

Fitxer d'on llegim



```
1 taula[0] = 11
2 taula[1] = 22
3 taula[2] = 33
4 taula[3] = 44
5 taula[4] = 55
6 taula[5] = 66
7 taula[6] = 77
8 taula[7] = 88
9 taula[8] = 99
10 taula[9] = 100
11
```

Linia llegida: taula[0] = 11

Linia llegida: taul

```
#include <stdio.h>

#define MAX_LINIA 30
#define NOM_FITXER "exemple_escruiure.txt"

int main(){

    char s[MAX_LINIA];
    FILE *f_in;

    // Obrir en mode lectura
    f_in = fopen(NOM_FITXER, "r");
    if (f_in == NULL){
        printf("ERROR, no s'ha pogut obrir el fitxer\n");
        return -1;
    }

    // Llegir una linia
    fgets(s, MAX_LINIA, f_in);
    printf("Linia llegida: %s\n", s);

    // Què passa si n petita?
    fgets(s, 5, f_in);
    printf("Linia llegida: %s\n", s);

    return 0;
}
```

I/O de Fitxer

Lectura de fitxer amb **format**

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- Llegir amb format vol dir que podem especificar quins tipus volem llegir (igual que féiem amb scanf)
- **fscanf** llegeix grups de caràcters separats per espais

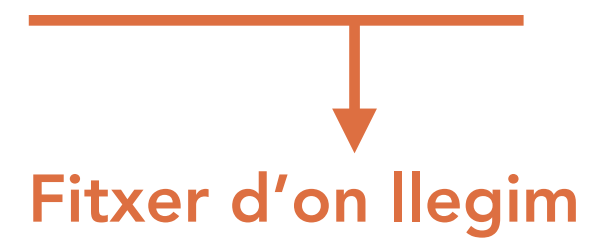
```
int fscanf(FILE *stream, const char *format, ...)
```

I/O de Fitxer

Lectura de fitxer amb **format**

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- Llegir amb format vol dir que podem especificar quins tipus volem llegir (igual que féiem amb scanf)
- **fscanf** llegeix grups de caràcters separats per espais

```
int fscanf(FILE *stream, const char *format, ...)
```



Fitxer d'on llegim

I/O de Fitxer

Lectura de fitxer amb **format**

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- Llegir amb format vol dir que podem especificar quins tipus volem llegir (igual que féiem amb scanf)
- **fscanf** llegeix grups de caràcters separats per espais

```
int fscanf(FILE *stream, const char *format, ...)
```

Fitxer d'on llegim

Format

I/O de Fitxer

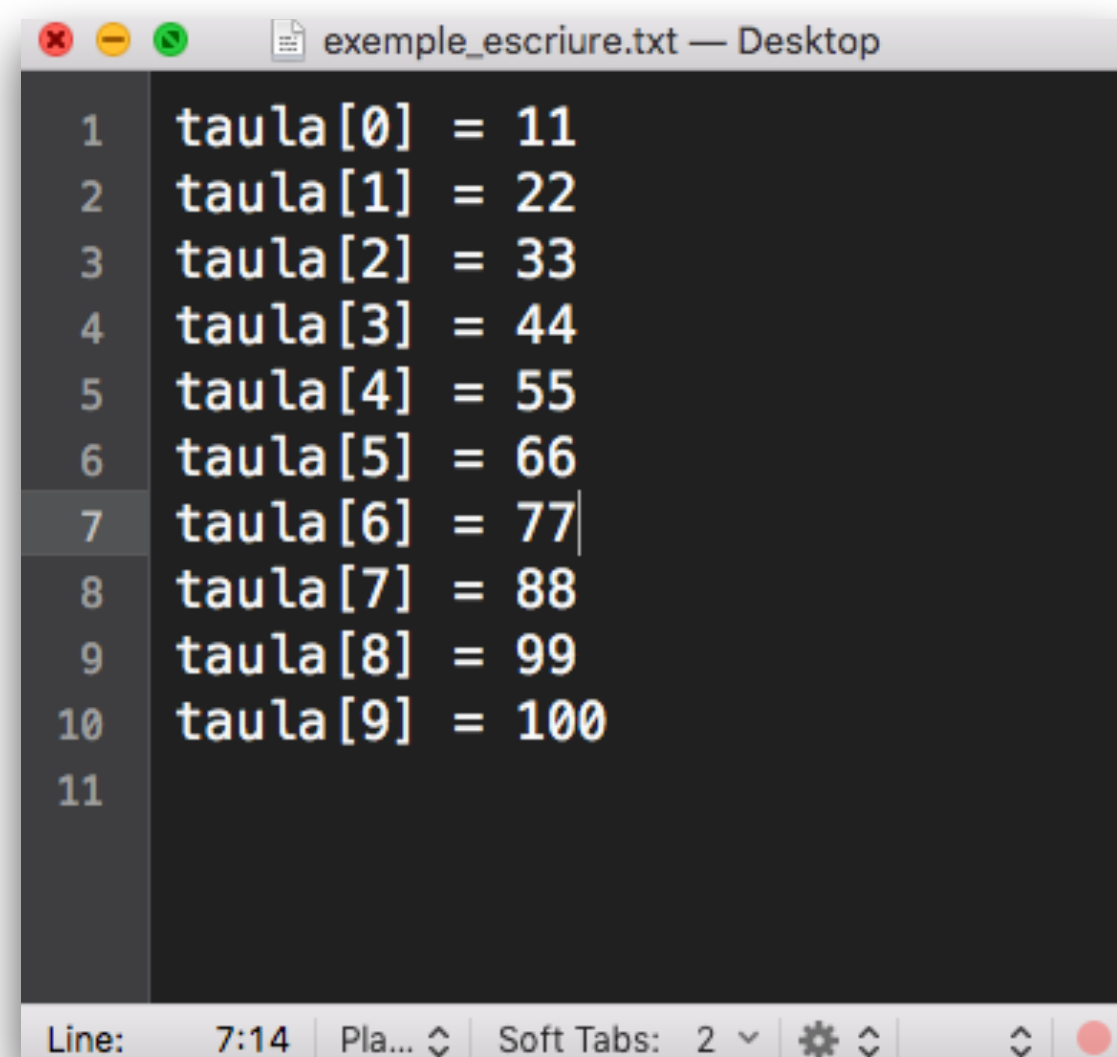
Lectura de fitxer amb **format**

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- Llegir amb format vol dir que podem especificar quins tipus volem llegir (igual que féiem amb scanf)
- **fscanf** llegeix grups de caràcters separats per espais

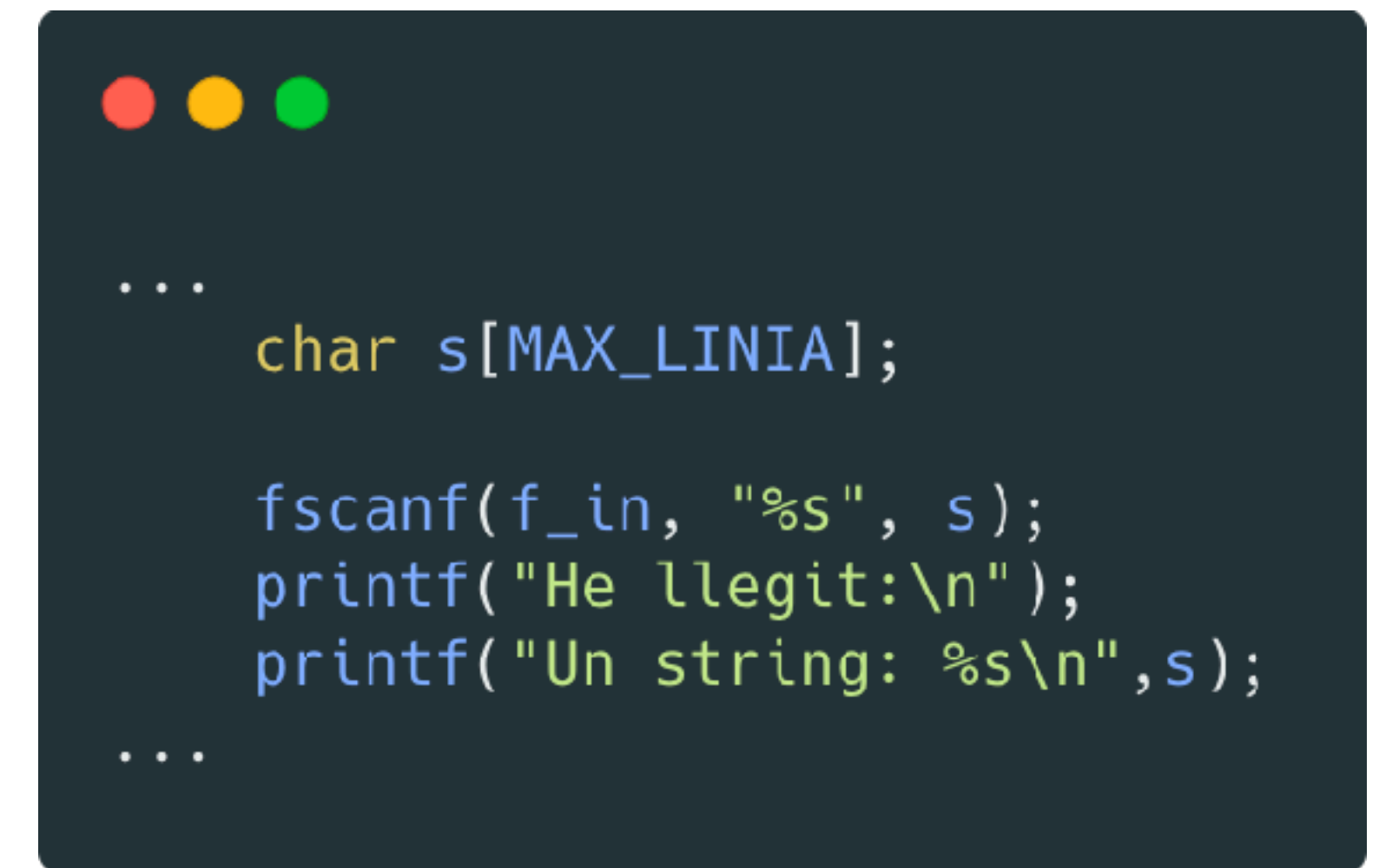
```
int fscanf(FILE *stream, const char *format, ...)
```

Fitxer d'on llegim

Format



```
1 taula[0] = 11
2 taula[1] = 22
3 taula[2] = 33
4 taula[3] = 44
5 taula[4] = 55
6 taula[5] = 66
7 taula[6] = 77
8 taula[7] = 88
9 taula[8] = 99
10 taula[9] = 100
11
```



```
...
char s[MAX_LINIA];

fscanf(f_in, "%s", s);
printf("He llegit:\\n");
printf("Un string: %s\\n",s);
...
```

I/O de Fitxer

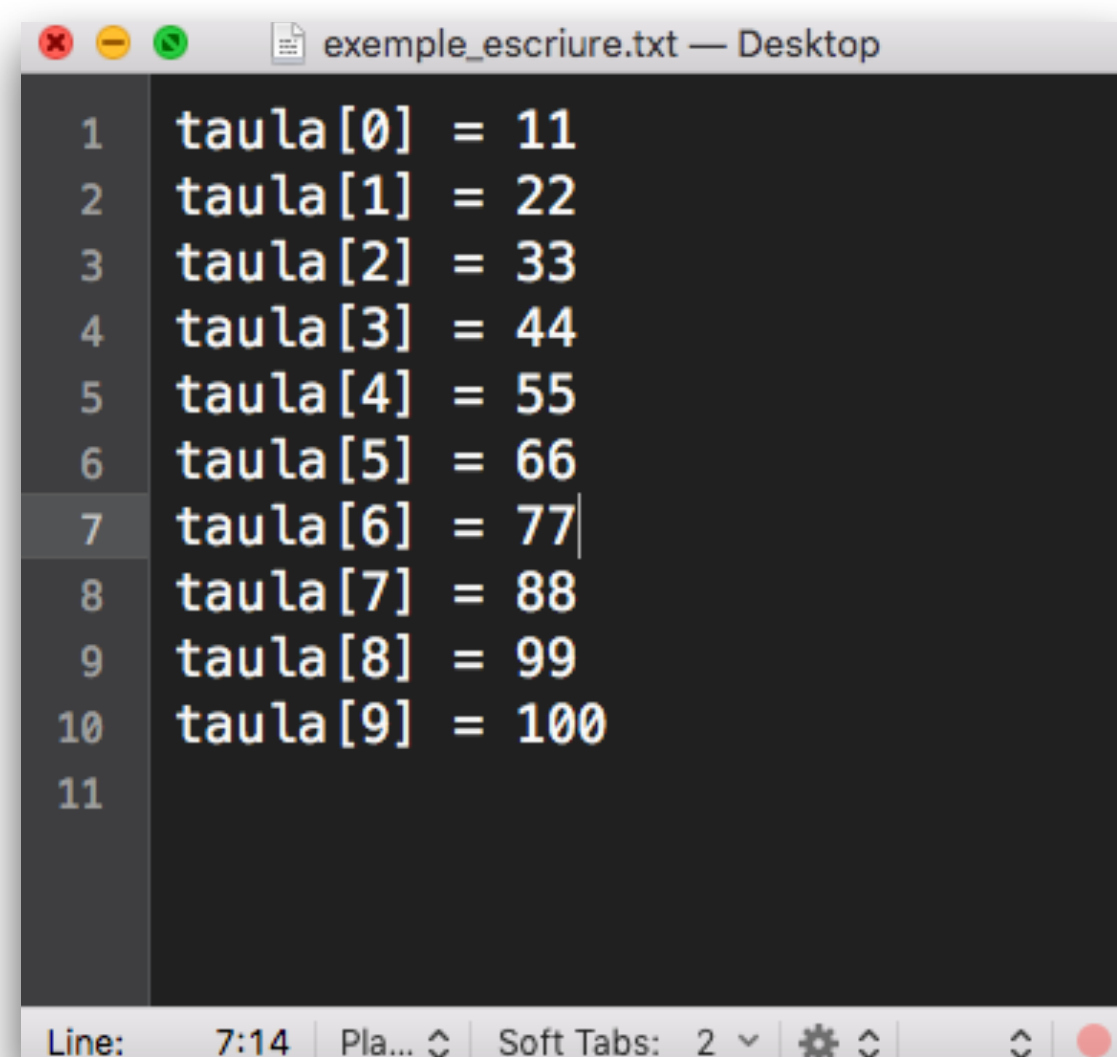
Lectura de fitxer amb **format**

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- Llegir amb format vol dir que podem especificar quins tipus volem llegir (igual que féiem amb scanf)
- **fscanf** llegeix grups de caràcters separats per espais

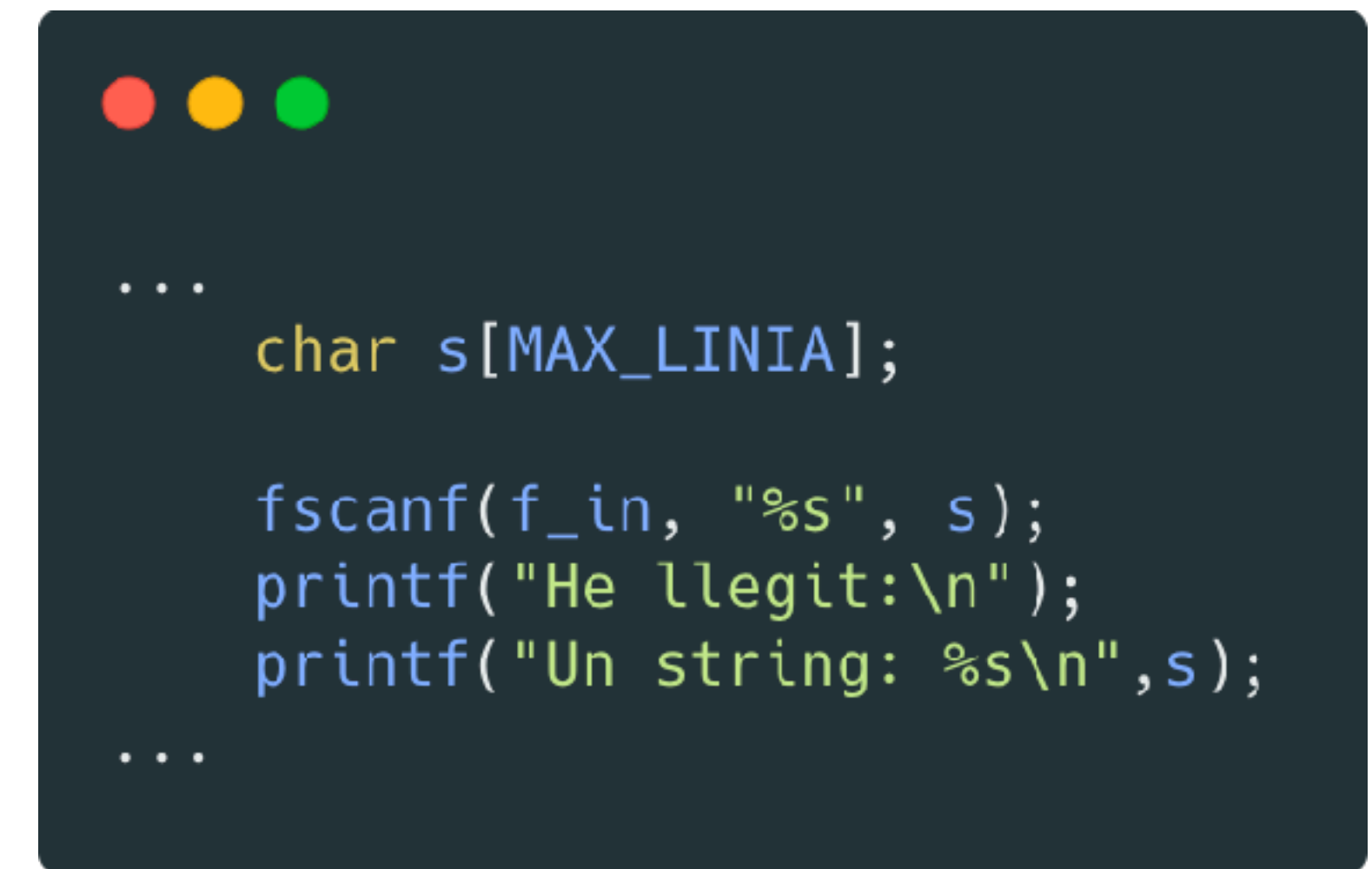
```
int fscanf(FILE *stream, const char *format, ...)
```

Fitxer d'on llegim

Format



```
1 taula[0] = 11
2 taula[1] = 22
3 taula[2] = 33
4 taula[3] = 44
5 taula[4] = 55
6 taula[5] = 66
7 taula[6] = 77
8 taula[7] = 88
9 taula[8] = 99
10 taula[9] = 100
11
```



```
...
char s[MAX_LINIA];

fscanf(f_in, "%s", s);
printf("He llegit:\n");
printf("Un string: %s\n",s);
...
```

He llegit:
Un string: taula[0]

I/O de Fitxer

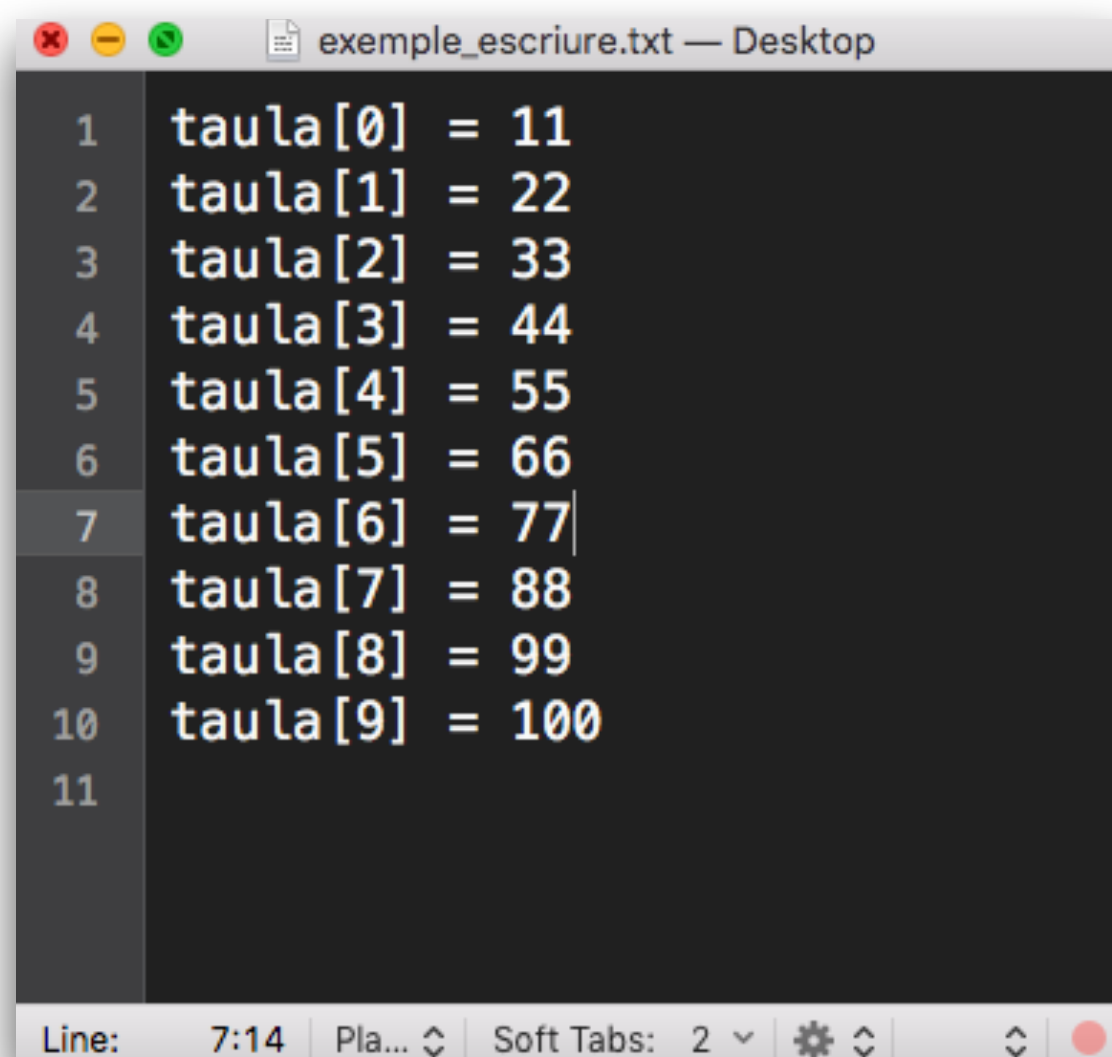
Lectura de fitxer amb **format**

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- Llegir amb format vol dir que podem especificar quins tipus volem llegir (igual que féiem amb scanf)
- **fscanf** llegeix grups de caràcters separats per espais

```
int fscanf(FILE *stream, const char *format, ...)
```

Fitxer d'on llegim

Format



```
...  
char s[MAX_LINIA];  
char s2[MAX_LINIA];  
int num;  
  
// Llegir especificant format  
fscanf(f_in, "%s%s%d", s, s2, &num);  
printf("He llegit:\n");  
printf("Un string: %s\n", s);  
printf("Un string: %s\n", s2);  
printf("Un enter: %d\n", num);  
  
...
```

I/O de Fitxer

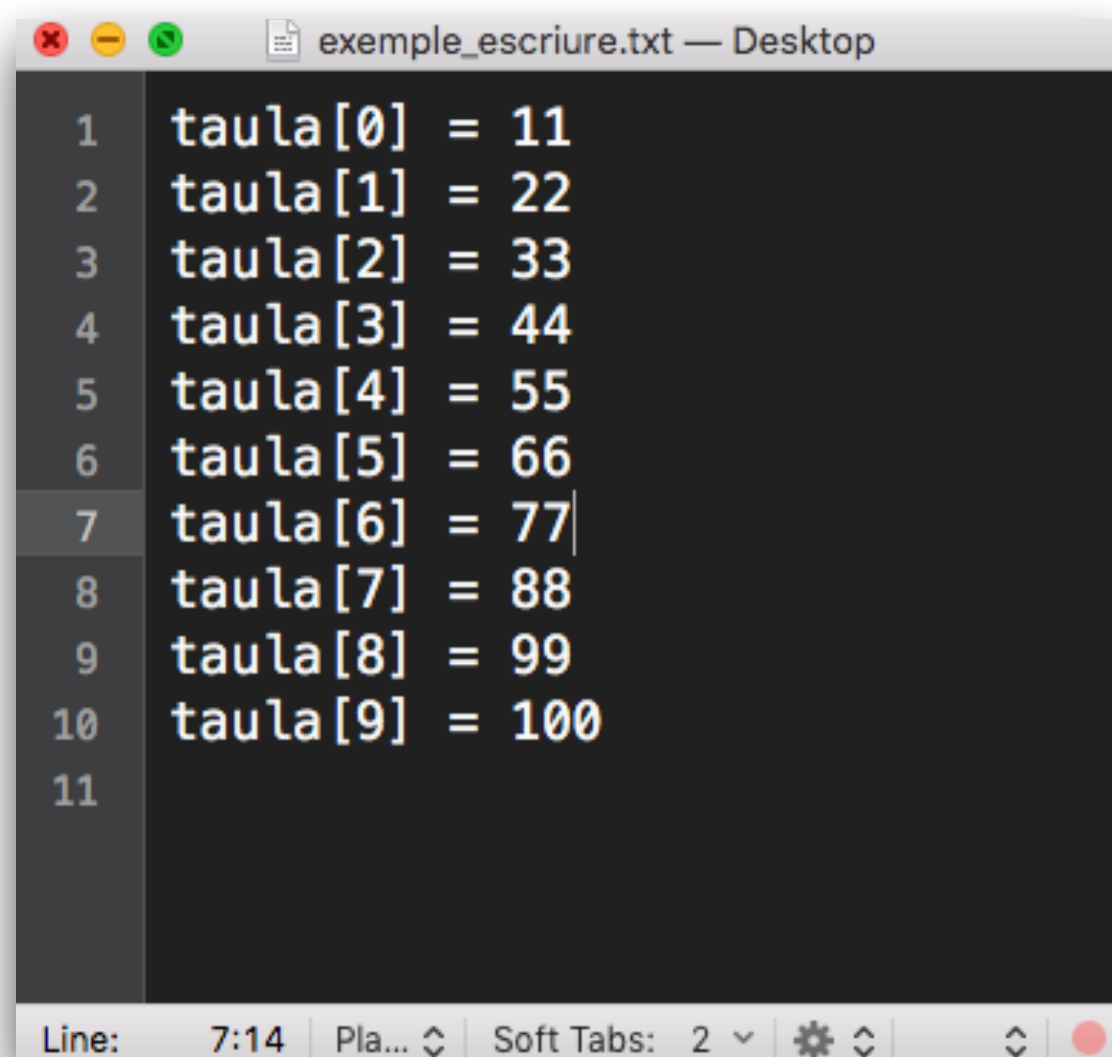
Lectura de fitxer amb **format**

- Hem d'haver obert el fitxer en un mode compatible amb lectura.
- Llegir amb format vol dir que podem especificar quins tipus volem llegir (igual que féiem amb scanf)
- **fscanf** llegeix grups de caràcters separats per espais

```
int fscanf(FILE *stream, const char *format, ...)
```

Fitxer d'on llegim

Format



```
...  
char s[MAX_LINIA];  
char s2[MAX_LINIA];  
int num;  
  
// Llegir especificant format  
fscanf(f_in, "%s%s%d", s, s2, &num);  
printf("He llegit:\n");  
printf("Un string: %s\n", s);  
printf("Un string: %s\n", s2);  
printf("Un enter: %d\n", num);  
  
...
```

```
He llegit:  
Un string: taula[0]  
Un string: =  
Un enter: 11
```

I/O de Fitxer

Lectura de fitxer amb **format**

Els espais no importen, però milloren la legibilitat



```
fscanf(f_in, "%s%s%d", s, s2, &num);
```



```
fscanf(f_in, "%s %s %d", s, s2, &num);
```

Aquests dos codis fan exactament el mateix

I/O de Fitxer

Lectura de fitxer amb **format**

Com saltar-nos coses que no ens interessin?

```
#NUM_PARAULES 6
```

Per exemple, aquí no m'interessa NUM_PARAULES, però sí el número 6

I/O de Fitxer

Lectura de fitxer amb **format**

Com saltar-nos coses que no ens interessen?

```
#NUM_PARAULES 6
```

Per exemple, aquí no m'interessa NUM_PARAULES, però sí el número 6

- Se'ns podria acudir de guardar-ho en algun lloc i després no fer-ho servir:

```
char junk[15];  
int num_paraules;  
...  
fscanf(f_in, "%s%d", junk, &num_paraules);
```

I/O de Fitxer

Lectura de fitxer amb **format**

Com saltar-nos coses que no ens interessin?

```
#NUM_PARAULES 6
```

Per exemple, aquí no m'interessa NUM_PARAULES, però sí el número 6

- Se'ns podria acudir de guardar-ho en algun lloc i després no fer-ho servir:

```
char junk[15];  
int num_paraules;  
...  
fscanf(f_in, "%s%d", junk, &num_paraules);
```

- **Una manera millor:** podem posar un asterisc davant d'allò que no ens interessa (en aquest cas, la paraula)

```
int num_paraules;  
...  
fscanf(f_in, "%*s%d", &num_paraules);
```



I/O de Fitxer

Lectura de fitxer **sencer**

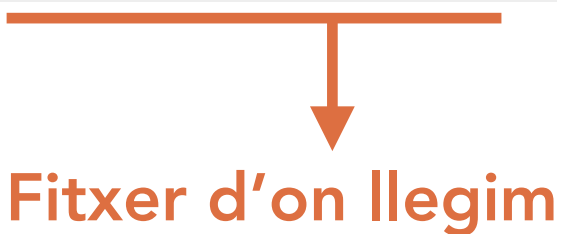
- Com podem llegir tots els continguts d'un fitxer?
- Per saber si hem arribat al final consultem **feof()**
- Què és l'**EOF**?
 - End-of-file. Codi al final de l'últim byte de dades en un arxiu. El valor del EOF depèn de la implementació i ha de ser negatiu.
 - Realment no ens importa quin valor tingui, nosaltres l'evaluem amb feof().

I/O de Fitxer

Lectura de fitxer **sencer**

- Com podem llegir tots els continguts d'un fitxer?
- Per saber si hem arribat al final consultem **feof()**
- Què és l'**EOF**?
 - End-of-file. Codi al final de l'últim byte de dades en un arxiu. El valor del EOF depèn de la implementació i ha de ser negatiu.
 - Realment no ens importa quin valor tingui, nosaltres l'evaluem amb feof().

```
int feof(FILE *stream)
```



Fitxer d'on llegim

- Retorna:
 - **0** si no hem arribat al final
 - **1** si hem arribat al final.

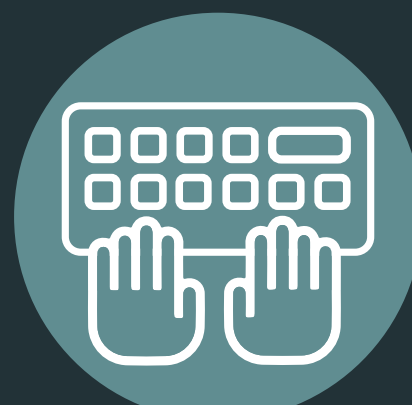
I/O de Fitxer

Lectura de fitxer **sencer**

```
int feof(FILE *stream)
```

↓
Fitxer d'on llegim

- Retorna:
 - **0** si no hem arribat al final
 - **1** si hem arribat al final.



```
#include <stdio.h>
#define MAX_LINIA 30
#define NOM_FITXER "exemple_escriture.txt"
```

```
int main(){
    char s[MAX_LINIA];
    int i;
    FILE *f_in;
```

```
/* Obrir el fitxer en mode lectura i comprovar que ha anat bé */
```

```
/* Llegir línia a línia fins Final de fitxer i anar
imprimint per pantalla*/
```

```
/* Tancar el fitxer */
```

```
return 0;
```

```
}
```

I/O de Fitxer

Lectura de fitxer **sencer**

```
int feof(FILE *stream)
```

↓
Fitxer d'on llegim

- Retorna:
 - **0** si no hem arribat al final
 - **1** si hem arribat al final.



```
#include <stdio.h>
#define MAX_LINIA 30
#define NOM_FITXER "exemple_escriture.txt"

int main(){
    char s[MAX_LINIA];
    int i;
    FILE *f_in;

    // Obrir en mode lectura
    f_in = fopen(NOM_FITXER, "r");
    if (f_in == NULL){
        printf("ERROR, no s'ha pogut obrir el fitxer\n");
        return -1;
    }

    /* Llegir línia a línia fins Final de fitxer i anar
    imprimint per pantalla*/

    /* Tancar el fitxer */

    return 0;
}
```


I/O de Fitxer

Lectura de fitxer **sencer**

```
int feof(FILE *stream)
```

↓
Fitxer d'on llegim

- Retorna:
 - **0** si no hem arribat al final
 - **1** si hem arribat al final.



```
#include <stdio.h>
#define MAX_LINIA 30
#define NOM_FITXER "exemple_escriture.txt"

int main(){
    char s[MAX_LINIA];
    int i;
    FILE *f_in;

    // Obrir en mode lectura
    f_in = fopen(NOM_FITXER, "r");
    if (f_in == NULL){
        printf("ERROR, no s'ha pogut obrir el fitxer\n");
        return -1;
    }
    // Llegir fins EOF
    printf("Continguts del document:\n");
    i = 1;
    while(feof(f_in)==0){
        fgets(s, MAX_LINIA, f_in);
        printf("Linia %d: %s", i, s);
        i++;
    }

    /* Tancar el fitxer */

    return 0;
}
```

I/O de Fitxer

Lectura de fitxer **sencer**

```
int feof(FILE *stream)
```

Fitxer d'on llegim

- Retorna:
 - **0** si no hem arribat al final
 - **1** si hem arribat al final.

Continguts del document:

```
Linia 1: taula[0] = 11
Linia 2: taula[1] = 22
Linia 3: taula[2] = 33
Linia 4: taula[3] = 44
Linia 5: taula[4] = 55
Linia 6: taula[5] = 66
Linia 7: taula[6] = 77
Linia 8: taula[7] = 88
Linia 9: taula[8] = 99
Linia 10: taula[9] = 100
```



```
#include <stdio.h>
#define MAX_LINIA 30
#define NOM_FITXER "exemple_escriture.txt"

int main(){
    char s[MAX_LINIA];
    int i;
    FILE *f_in;

    // Obrir en mode lectura
    f_in = fopen(NOM_FITXER, "r");
    if (f_in == NULL){
        printf("ERROR, no s'ha pogut obrir el fitxer\n");
        return -1;
    }
    // Llegir fins EOF
    printf("Continguts del document:\n");
    i = 1;
    while(feof(f_in)==0){
        fgets(s, MAX_LINIA, f_in);
        printf("Linia %d: %s", i, s);
        i++;
    }

    /* Tancar el fitxer */

    return 0;
}
```

I/O de Fitxer

Lectura de fitxer **sencer**

```
int feof(FILE *stream)
```

Fitxer d'on llegim

- Retorna:
 - **0** si no hem arribat al final
 - **1** si hem arribat al final.

Continguts del document:

```
Linia 1: taula[0] = 11
Linia 2: taula[1] = 22
Linia 3: taula[2] = 33
Linia 4: taula[3] = 44
Linia 5: taula[4] = 55
Linia 6: taula[5] = 66
Linia 7: taula[6] = 77
Linia 8: taula[7] = 88
Linia 9: taula[8] = 99
Linia 10: taula[9] = 100
```



```
#include <stdio.h>
#define MAX_LINIA 30
#define NOM_FITXER "exemple_escriture.txt"

int main(){
    char s[MAX_LINIA];
    int i;
    FILE *f_in;

    // Obrir en mode lectura
    f_in = fopen(NOM_FITXER, "r");
    if (f_in == NULL){
        printf("ERROR, no s'ha pogut obrir el fitxer\n");
        return -1;
    }
    // Llegir fins EOF
    printf("Continguts del document:\n");
    i = 1;
    while(feof(f_in)==0){
        fgets(s, MAX_LINIA, f_in);
        printf("Linia %d: %s", i, s);
        i++;
    }
    // Tancar fitxer
    fclose(f_in);
    return 0;
}
```

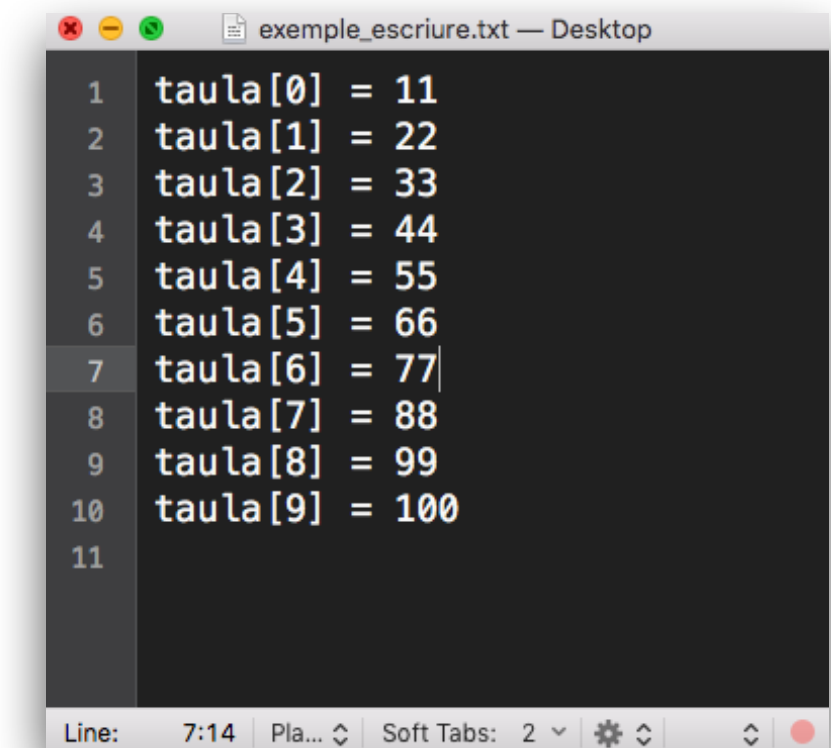
I/O de Fitxer

Cas real: lectura del fitxer anterior guardant només els nombres en una taula

- Volem aconseguir una taula amb continguts:

11	22	33	44	55	66	77	88	99	100
----	----	----	----	----	----	----	----	----	-----

- Procés
 - Obrir el fitxer en el mode corresponent
 - Decidir quina funció ens va millor per extreure el nombre
 - Construir el bucle feof()
 - Guardar les dades a una taula **de mida ???**



```
1 taula[0] = 11
2 taula[1] = 22
3 taula[2] = 33
4 taula[3] = 44
5 taula[4] = 55
6 taula[5] = 66
7 taula[6] = 77
8 taula[7] = 88
9 taula[8] = 99
10 taula[9] = 100
11
```

exemple_escruiure.txt

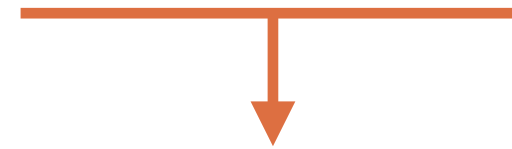
I/O de Fitxer

Cas real: lectura del fitxer anterior guardant només els nombres en una taula

- Volem aconseguir una taula amb continguts:

11	22	33	44	55	66	77	88	99	100
----	----	----	----	----	----	----	----	----	-----

- Procés
 - Obrir el fitxer en el mode corresponent
 - Decidir quina funció ens va millor per extreure el nombre
 - Construir el bucle feof()
 - Guardar les dades a una taula **de mida ???**



No sé la mida de la taula!

```
1 taula[0] = 11
2 taula[1] = 22
3 taula[2] = 33
4 taula[3] = 44
5 taula[4] = 55
6 taula[5] = 66
7 taula[6] = 77
8 taula[7] = 88
9 taula[8] = 99
10 taula[9] = 100
11
```

exemple_escriure.txt

I/O de Fitxer

Com guardar dades de fitxer a taula

- Ja sabem llegir de fitxer i ja sabem declarar taules
- Com sé la longitud de la taula a declarar?

I/O de Fitxer

Com guardar dades de fitxer a taula

- Ja sabem llegir de fitxer i ja sabem declarar taules
- Com sé la longitud de la taula a declarar?

Opció 1

- Declarar una taula de mida arbitrària i fer un resize si veig que necessito més espai.

I/O de Fitxer

Com guardar dades de fitxer a taula

- Ja sabem llegir de fitxer i ja sabem declarar taules
- Com sé la longitud de la taula a declarar?

Opció 1

- Declarar una taula de mida arbitrària i fer un resize si veig que necessito més espai.

Opció 2

- *(Només aplicable si l'arxiu no és molt gran)*

Llegir l'arxiu 2 vegades: una per comptar quantes línies té i així saber de quina mida declarar la taula, i una segona vegada per llegir-ne els continguts i guardar-los

I/O de Fitxer

Com guardar dades de fitxer a taula

- Ja sabem llegir de fitxer i ja sabem declarar taules
- Com sé la longitud de la taula a declarar?

Opció 1

- Declarar una taula de mida arbitrària i fer un resize si veig que necessito més espai.

Opció 2

- *(Només aplicable si l'arxiu no és molt gran)*

Llegir l'arxiu 2 vegades: una per comptar quantes línies té i així saber de quina mida declarar la taula, i una segona vegada per llegir-ne els continguts i guardar-los

Opció 3

- *(Només aplicable si l'arxiu l'hem generat nosaltres o ja té aquest format).*

Tenir per costum guardar al propi arxiu el nombre d'elements que hem de llegir. Llegir primer el nombre d'elements, declarar taula d'aquella mida, i després llegir la resta.

```
NumElems = 6
235
168
153
8976
123
21
```

I/O de Fitxer

Com guardar dades de fitxer a taula

- Ja sabem llegir de fitxer i ja sabem declarar taules
- Com sé la longitud de la taula a declarar?

Opció 1

- Declarar una taula de mida arbitrària i fer un resize si veig que necessito més espai.

Opció 2

- *(Només aplicable si l'arxiu no és molt gran)*

Llegir l'arxiu 2 vegades: una per comptar quantes línies té i així saber de quina mida declarar la taula, i una segona vegada per llegir-ne els continguts i guardar-los

Opció 3

- *(Només aplicable si l'arxiu l'hem generat nosaltres o ja té aquest format).*

Tenir per costum guardar al propi arxiu el nombre d'elements que hem de llegir. Llegir primer el nombre d'elements, declarar taula d'aquella mida, i després llegir la resta.

```
NumElems = 6
235
168
153
8976
123
21
```

Treballarem aquestes tres opcions als exercicis del moodle

A CASA...

EXERCICIS L8 del Moodle

EL PROPER DIA...

REGISTRES I TIPUS D'USUARI