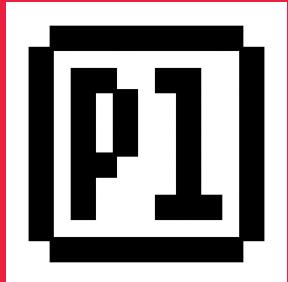


# Matrices y archivos

**UNRN**

Universidad Nacional  
de Río Negro

r20



**Primero, un  
repaso de**

# **Arreglos Unidimensionales**



# Arrays

---



# Pointers

**tipo nombre[tamaño];**

# Declaración e Inicialización

```
int arreglo[] = {0, 1, 2, 3, 4};
```

0	1	2	3	4
---	---	---	---	---

# Declaración y acceso

```
int arreglo[MAX];
for (int i = 0; i < MAX; i++)
{
    arreglo[i] = i;
}
```

0	1	2	3	4
---	---	---	---	---



**¿Preguntas?**



---

# Matrices



$n \times m$   
`arreglo[3][4]`

$m = \text{columnas}$

$n = \text{filas}$


0

1

2

0

1

2

3

Posiciones filas

Posiciones columnas

---

# Definición estática

```
tipo nombre[tamano_fila][tamano_columna];
```

```
int matrix[10][20];
```

---

# Definición e inicialización estática

```
int matriz[3][3] = {{1, 2, 3},  
                     {4, 5, 6},  
                     {7, 8, 9}};
```

# Carga de una matriz

```
int numeros[3][3];  
  
for (i = 0; i < 3; i++)  
{  
    for (j = 0; j < 3; j++)  
    {  
        numeros[i][j] = sum++;  
    }  
}
```

0	1	2
3	4	5
6	7	8

# Impresión de una matriz

```
for (i = 0; i < 3; i++)
{
    printf("matriz=\n");
    for (j = 0; j < 3; j++)
    {
        printf("%d", numeros[i][j]);
    }
    printf("\n");
}
```

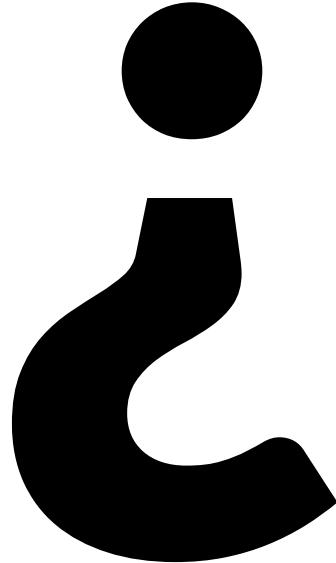
```
matriz=
[0][1][2]
[3][4][5]
[6][7][8]
```

0	1	2
3	4	5
6	7	8

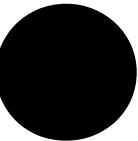


**¿Preguntas?**





# Y dinámica



# Primer enfoque

```
int** matriz = (int**)malloc(filas * sizeof(int));  
  
for(int i = 0; i < filas; i++){  
    matriz[i] = (int *) malloc(columnas * sizeof(int));  
}
```

**¿Cuáles son los pasos acá?**

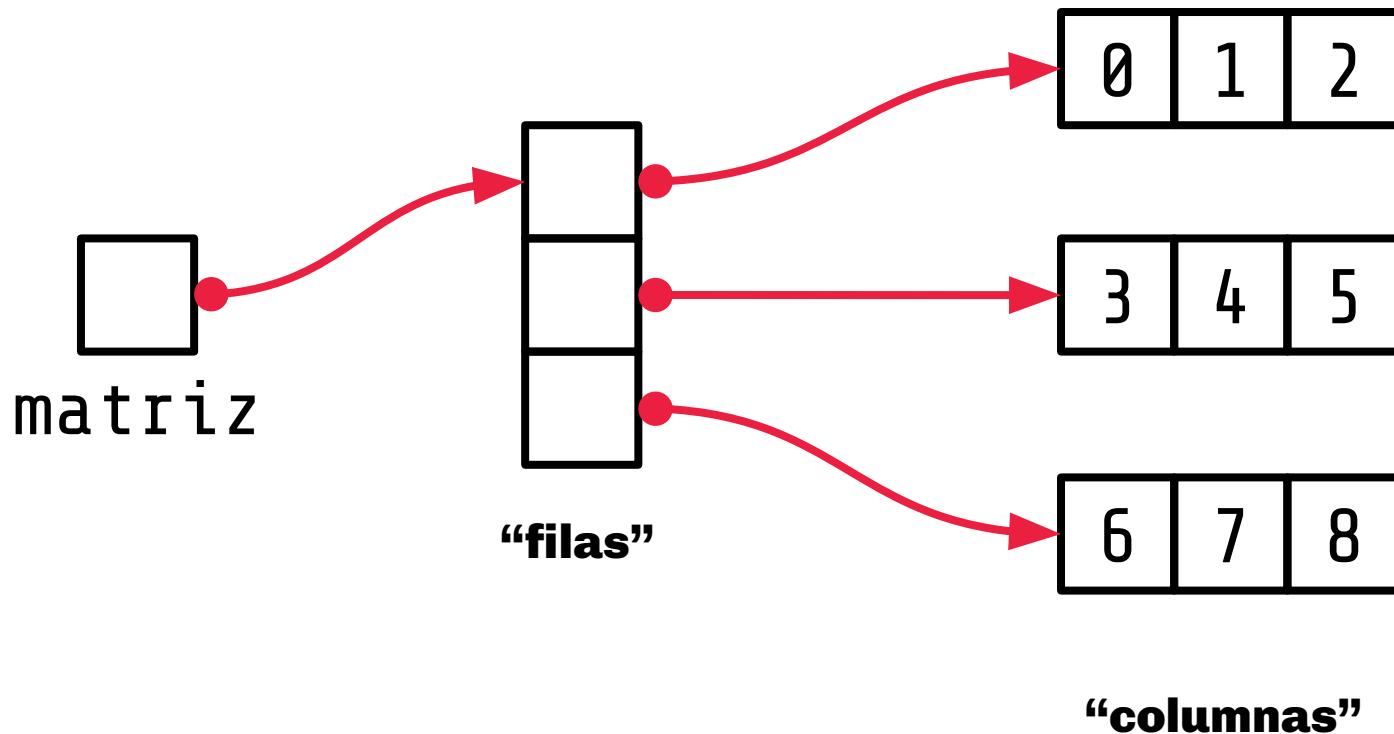
# Pero, ¿qué son?

```
int** matriz = (int**)malloc(filas * sizeof(int*));  
  
for(int i = 0; i < filas; i++){  
    matriz[i] = (int *) malloc(columnas * sizeof(int));  
}
```

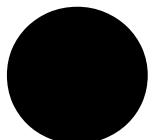
¿Son lo mismo?

El uso de la  
matriz es igual  
despues

# A efectos prácticos, este es el resultado



Y técnicamente ocupa más espacio



**Cuantos free  
son  
necesarios**

# **La liberación de memoria es paso a paso**

```
for(int i = 0; i < filas; i++){
    free(matriz[i]);
}
free(matriz);
```

**Muy importante que sea de ‘dentro a fuera’**

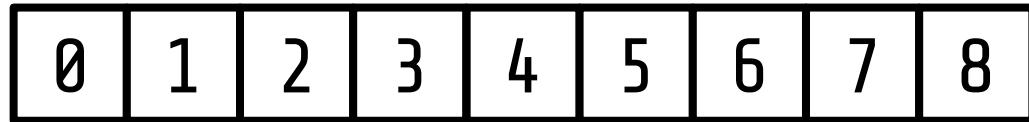
A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



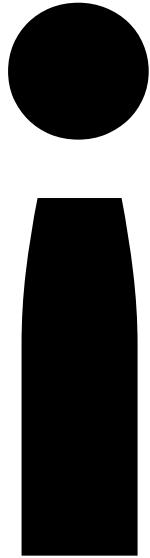
# Dinámica como un único bloque

```
int* matriz = (int*) malloc(filas * columnas * sizeof(int));
```

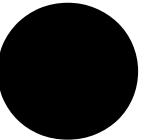
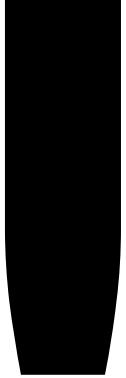


Y los ?

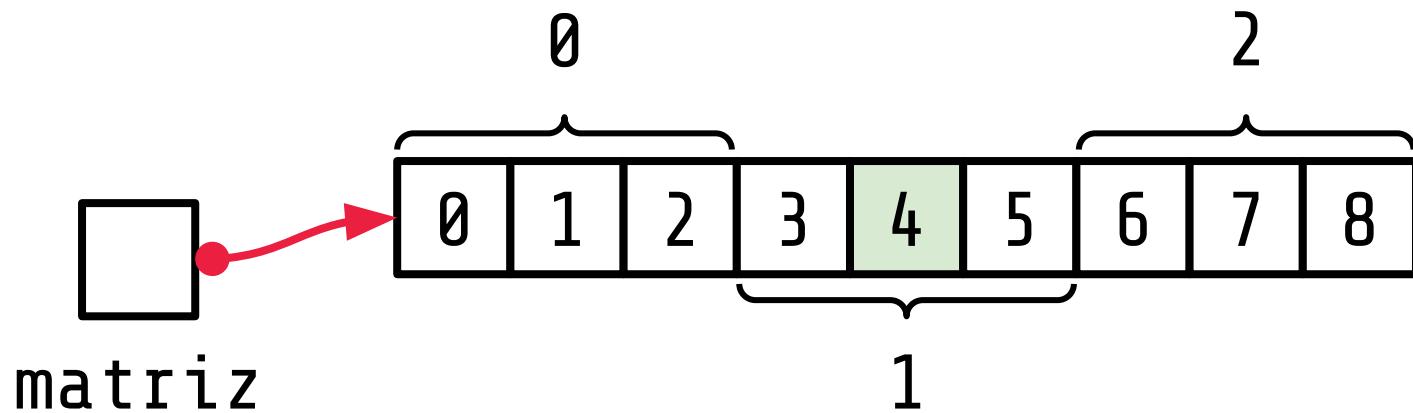
[ ] [ ]



No están



# En memoria, queda algo como

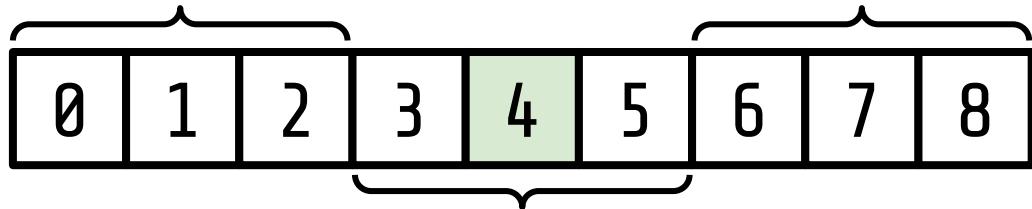


Pero no tenemos []

# Es necesario simularlo

```
int valor = matriz[(i * columnas) + j];
```

0	1	2
3	4	5
6	7	8



A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



# EJ: Matriz traspuesta

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \rightarrow \quad A^T = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 11 & 12 & 13 \end{pmatrix} \quad \rightarrow \quad A^T = \begin{pmatrix} 1 & 11 \\ 2 & 12 \\ 3 & 13 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \quad \rightarrow \quad A^T = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \rightarrow \quad A^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

# EJ: Matriz traspuesta

MATRIZ

0	1	2
3	4	5
6	7	8

TRASPUESTA

0	3	6
1	4	7
2	5	8

# EJ: Matriz traspuesta

```
for (i=0;i<3;i++){  
    printf("\n");  
    for (j=0;j<3;j++)  
        printf("%d",numeros[i][j]);};}
```

```
for (i=0;i<3;i++){  
    printf("\n");  
    for (j=0;j<3;j++)  
        printf("%d",numeros[j][i]);};}
```

0	1	2
3	4	5
6	7	8

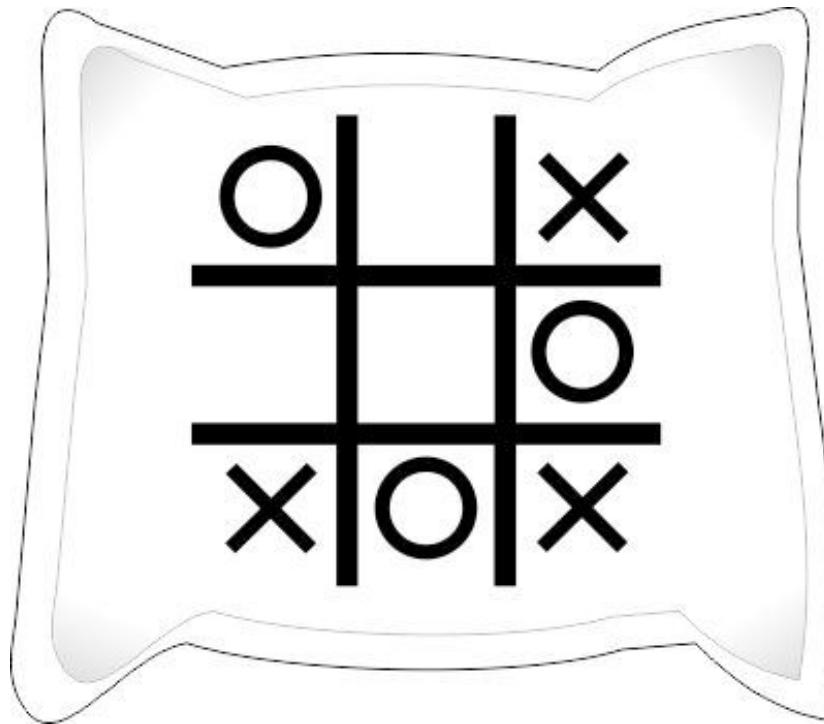
0	3	6
1	4	7
2	5	8

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



# Ejercicio: TATETI



# CASOS:

# Ejercicio: TATETI

[x][o][o]

[x][o][ ]

[x][ ][ ]

[ ][x][o]

[ ][x][ ]

[o][x][o]

[ ][o][x]

[o][o][x]

[x][ ][x]

[x][x][x]

[ ][o][ ]

[x][ ][ ]

[ ][x][o]

[x][x][x]

[o][ ][o]

[ ][o][x]

[o][o][ ]

[x][x][x]

# CASOS:

# Ejercicio: TATETI

[x][o][o]  
[ ][x][ ]  
[x][ ][x]

[o][ ][x]  
[ ][x][o]  
[x][x][o]

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



# Archivos stdio.h (parte 2)

---

# apertura

```
FILE *fopen(char nombre_archivo[], char modo[]);
```

**Si hay problemas  
para abrir el  
archivo**

# **El puntero apuntará a NULL**

# modos

---

" I "

para leer  
el archivo debe existir

---

" " W "

para escribir  
si el archivo existe, **se reemplaza por uno vacío**

---

" a "

Lo que escribamos va al final  
Si el archivo no existe es creado

---

"r+"

Para leer y escribir  
El archivo tiene que existir

---

"w+"

crea un archivo vacío para leer y escribir  
**reemplazandolo si existia por uno vacío**

---

"a+"

Para leer y escribir al final  
si no existe, se crea

---

" b "

modo binario, y va junto a si es RBA+.  
\*no le daremos uso\*

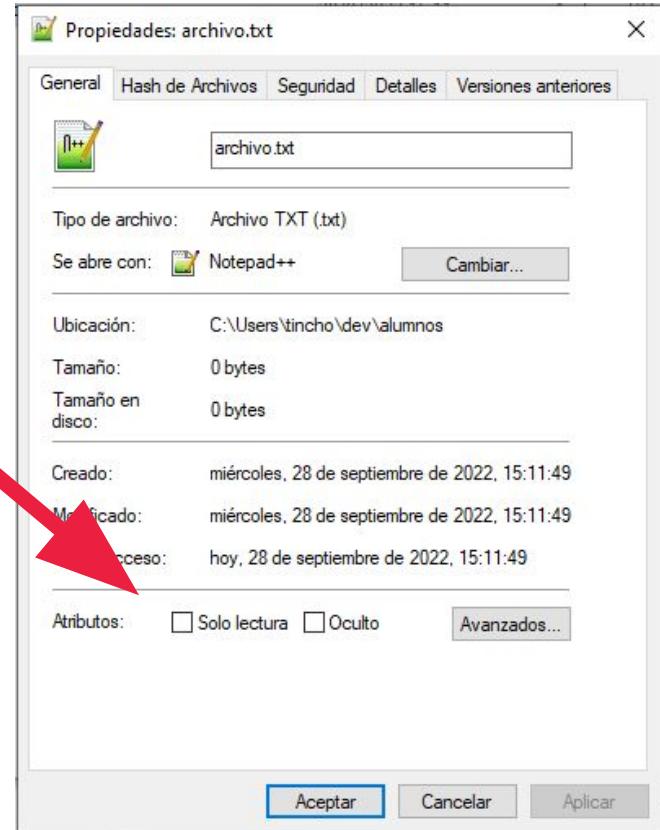
# — ¿Qué puede fallar?

**Si hay problemas  
para abrir el  
archivo**

**El puntero apuntará  
a NULL**

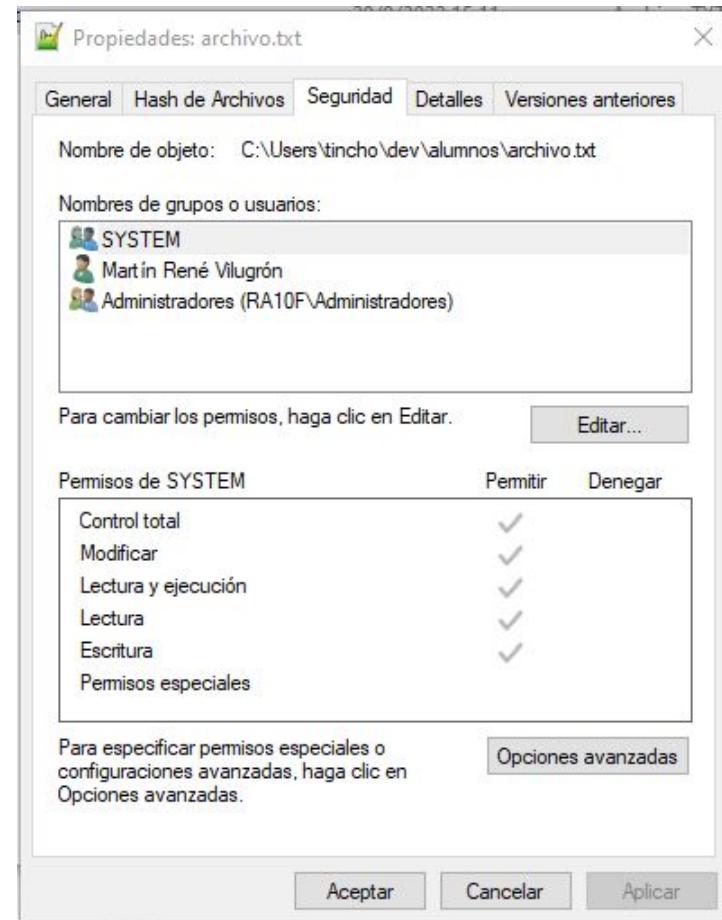
# Permisos de archivo

**El archivo a  
modificar estaba  
como 'solo lectura'**



(botón derecho, propiedades)

# El usuario no puede acceder\* archivos en la ubicación



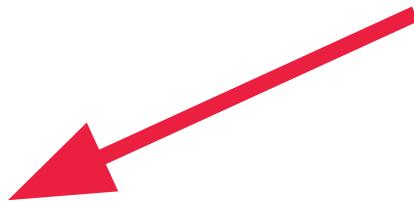
# También bloqueos por otros programas

# **Para liberar el archivo (y ser prolijo)**

---

# Cierre

EOF sí hay problemas



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

# El archivo ya estaba abierto\*

```
a1 = fopen("archivo.txt", "r");  
a2 = fopen("archivo.txt", "w");
```

\*en especial si olvidan el **fclose** y necesitan del archivo  
después

---

# EOF

End of File

Se usa para indicar que nos quedamos sin archivo para leer

# **Escritura a archivo**

# salida a archivo

caracteres impresos  o  
valores negativos 



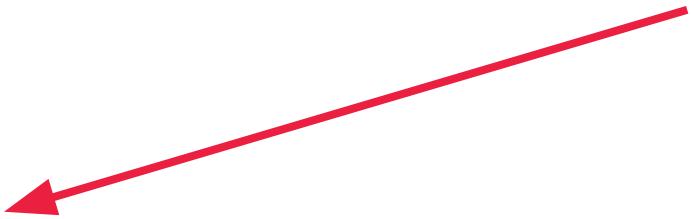
```
int fprintf(FILE *stream, ...);
```

# Lectura a carácter a carácter

# Para leer de a carácter de archivo

un carácter individual 

o  
EOF 



```
int fgetc(FILE *stream);
```

---

# ¿El archivo está en el final?

1 dale que va

0

0 no hay más archivo

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

---

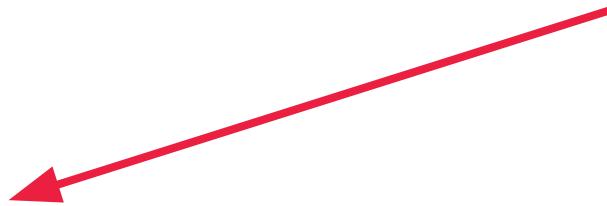
# Imprime carácter a carácter

```
do
{
    c = fgetc(fp);
    printf("%c", c);
}
while (!feof(fp));
```

# Leer de a bloques

# Para leer de a bloque desde archivo

str  o  
NULL 



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

---

# Un bloque de 100 caracteres

```
#define MAX_CADENA 100  
  
char string[MAX_CADENA];  
fgets(string, MAX_CADENA, fp);  
printf("%s\n", string);
```

**¿Si hay más de  
MAX\_CADENA?**

**A repetir la lectura  
hasta feof**

**Para archivos de tamaño  
desconocido es mucho  
más simple carácter a  
carácter**

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



# Gestión de errores

**Si falla algo, ¿no  
podemos saber  
más?**

# perror en stdio.h

```
void perror(char* mensaje);
```

*Imprime por consola (stderr) el mensaje sobre el último error junto a un mensaje propio.*

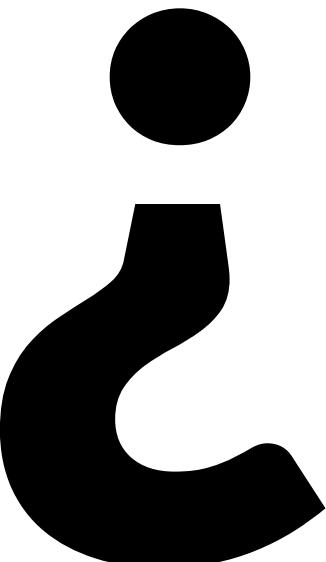
# Archivo inexistente

```
#include <stdio.h>

int main()
{
    FILE *archivo = fopen("no_existi.txt", "r");
    if (archivo == NULL)
    {
        perror("fopen() fallo");
    }
    else
    {
        fclose(archivo);
    }
}
```

# Y veremos por consola

```
fopen() fallo: No such file or directory
```



**Podemos  
decidir cómo  
seguir**



# errno en errno.h

Una *variable global* con el código de error producido por la **última llamada a función**.

**1**

funciones como  
**printf** puede  
cambiar su valor

**Copien el valor de  
errno antes de  
llamar a otra  
función**

**2**

el valor **solo** tiene sentido si la función retorna valores como **NULL** o **-1**

# Por lo que se utiliza

```
FILE *archivo = fopen("no_existi.txt", "r");
if (archivo == NULL)
{
    printf("%s\n\n", strerror(errno));
    perror("fopen() fallo");
}
else
{
    // todo OK
}
```



string.h

# Algunos valores relevantes de errno.h

ENOENT - El archivo no existe

EEXIST - El archivo ya existe

ENAMETOOLONG - Nombre muy largo

EACCES - Error de permisos de acceso

Todos  
comienzan con E

**Y nos dan lugar a  
decidir cómo seguir**

# Por ejemplo

Si el archivo ya existía, pueden agregar un “-copia”

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



**Pero, si  
necesitamos el  
mensaje**

**También lo podemos  
obtener igual que el  
de perro!**

# **strerror en string.h**

```
char* strerror( int errnum );
```

Retorna un puntero a la cadena que describe el error indicado por errnum.  
Idéntico al que saldría por perror.



Que retorna  
`fprintf`

# fprintf (y printf)

La cantidad de caracteres escritos  
Con valores negativos si hay problemas

Y aunque todas  
las operaciones  
de archivos  
*pueden fallar*

**atajarlos en fopen es lo  
más importante**

---

# Otras situaciones de error como para ver

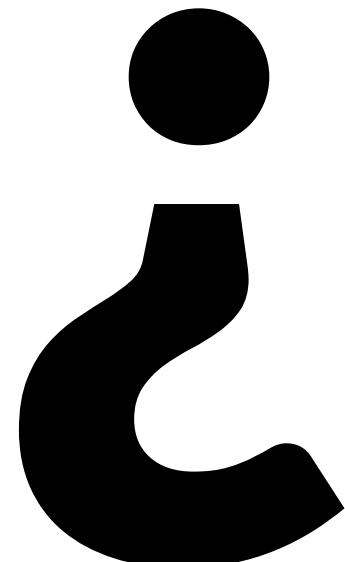
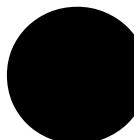
- Abrir o cerrar dos veces un archivo
- Operar con un FILE en NULL
- No cerrar el archivo luego enviar información
- El archivo deja de estar disponible

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

**¿Preguntas?**



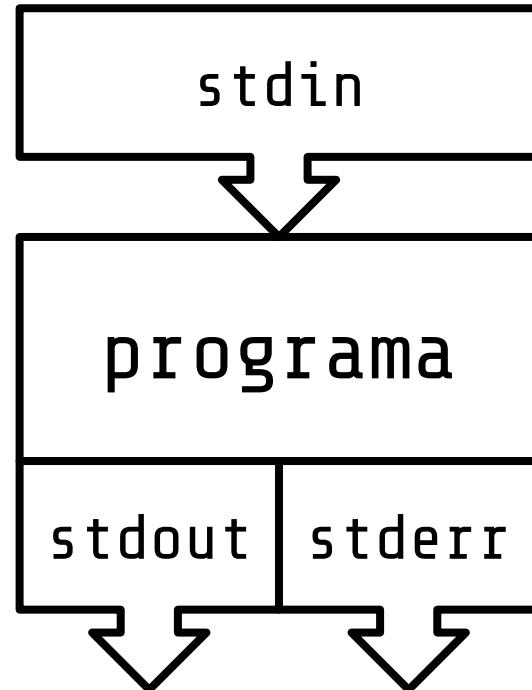
**PLUS  
ULTRA**



**Qué es eso de  
stderr**

# Otro canal de salida como el usado por printf (stdout)

scanf  
gets



printf\*



fprintf\*



**Lo normal es que  
ambos esten  
conectados a la  
consola**

**Y permite separar  
los mensajes de  
error de la salida  
normal del  
programa**

**Se usa para los  
programas  
conectados por la  
terminal**

# Para escribir a ‘error estándar’

```
fprintf(stderr, "Esto es un error!\n");
```

**Aunque este  
‘superpuesto’ con  
los printf  
tradicionales**

```
$> ./miprograma 2> errores.txt
```

**Envía solo los errores al archivo  
errores.txt**

**Hay más combinaciones  
y posibilidades, pero  
escapan a este curso**

**Le darán uso en la  
materia  
Sistemas Operativos**

# Lo mismo que printf y scanf

```
fprintf(stdout, "Esto es un printf!\n");  
fscanf(stdin, "%d", &n umero);
```

**Lo interesante es que los  
tres**

**son archivos**

y aplican todos  
los códigos de  
error

**Interceptando  
STDIN / STDOUT es  
possible interceptar  
printf / scanf y  
armar tests**

**unrn.edu.ar**

**UNRN**

Universidad Nacional  
de Río Negro



| unrnionegro