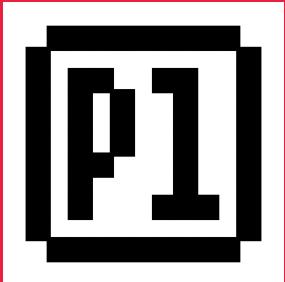


Compilación y pruebas

UNRN

Universidad Nacional
de Río Negro

r20



Arreglos

parte 2

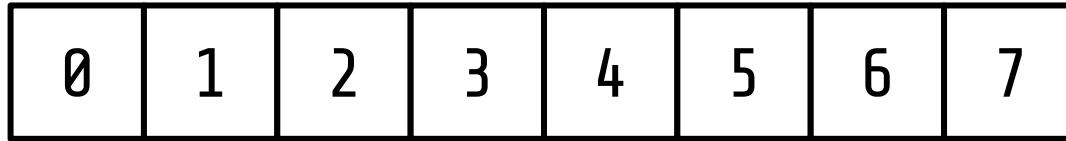


¿qué es?

Para
qué

Memoria contigua

```
int arreglo[8];
```

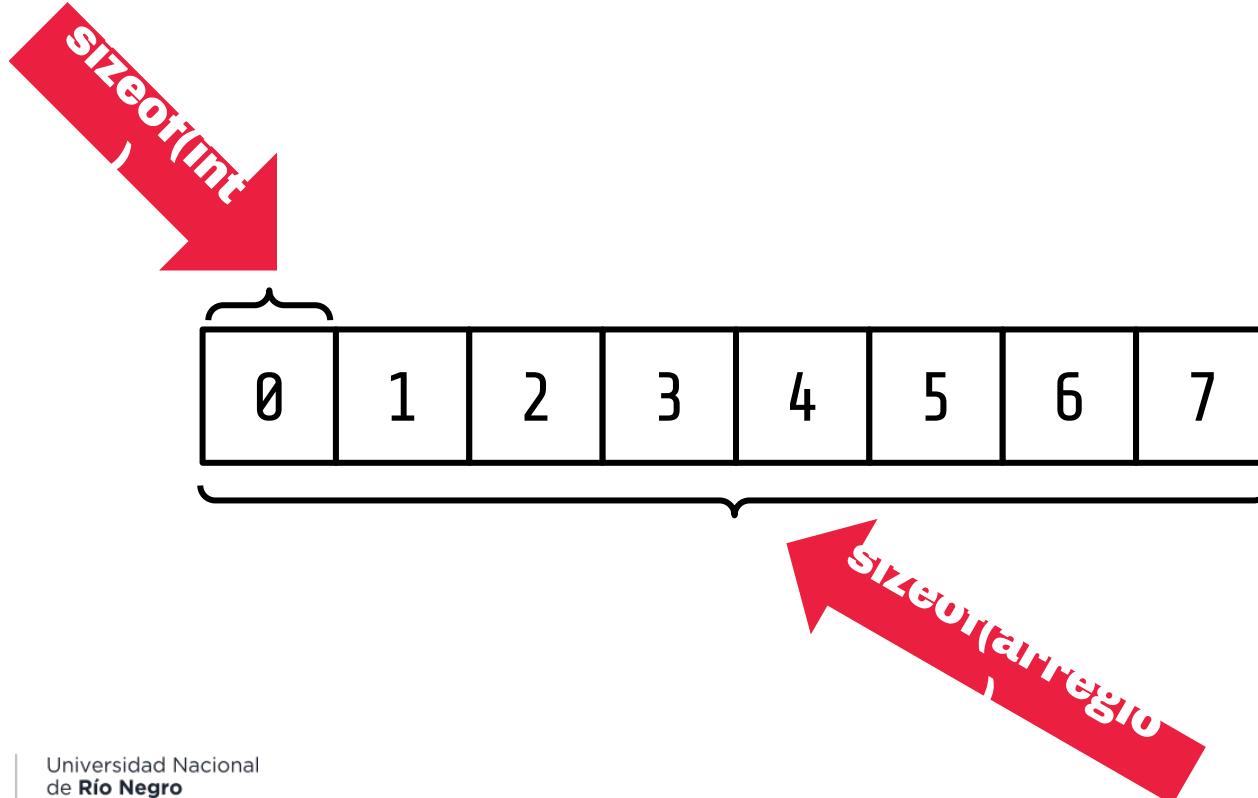


Tamaños de los arreglos

```
int arreglo[8];
sizeof(arreglo[0]); 1
sizeof(arreglo); 2
```

Memoria contigua

```
int arreglo[8];
```



Formas de inicialización

```
int completa[5] = {1, 2, 3, 4, 5};  
int parcial[5] = {1, 2};  
int implicita[] = {1, 2, 3};
```

Y en funciones

```
void imprimirArreglo(int arreglo[], int size) {  
    sizeof(arreglo[0]); 1  
    sizeof(arreglo); 2  
    ...  
}
```

Algunas preguntas sobre arreglos (y por extensión, cadenas)

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?



La compilación

Para ejecutar



./a.exe



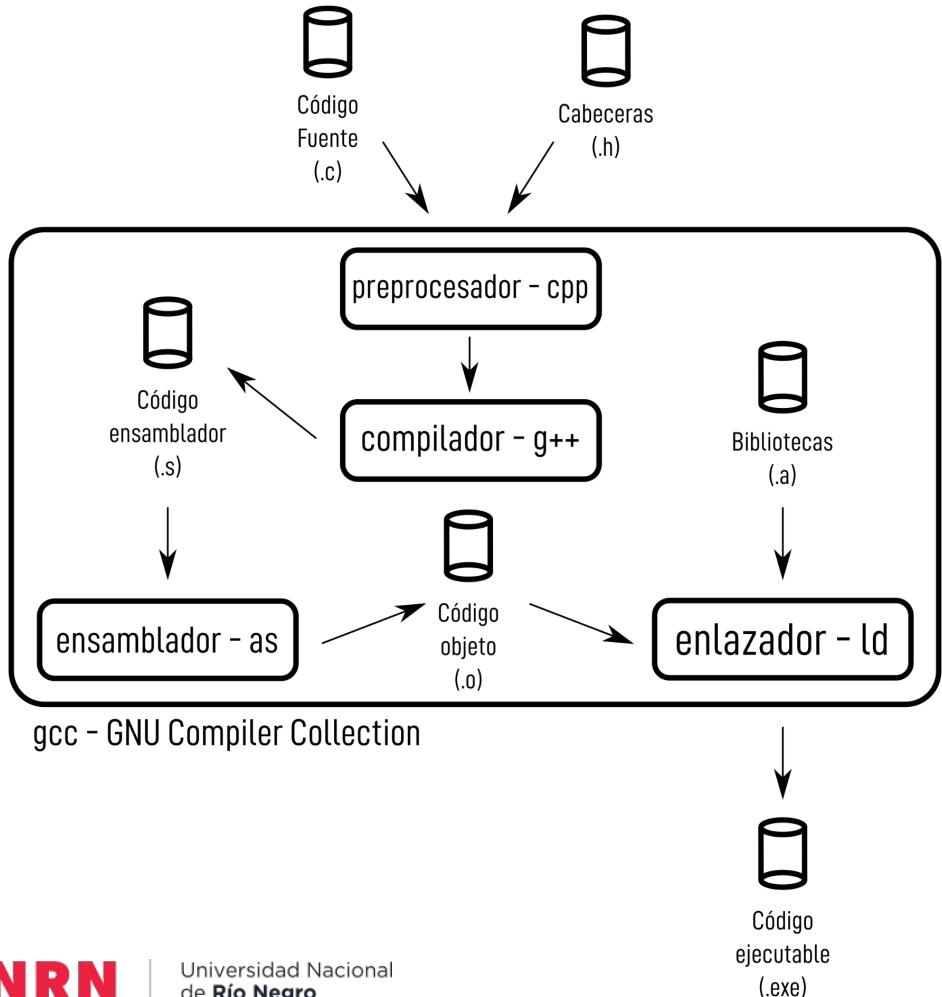
./a.out

**¿Pero qué pasa
cuando compilamos?**

**¿¡expecto
binarium?!**



**¡Solo un conjunto de
programas!**



**Es posible
llamar a cada
programa
individualmente**

Pero si hacemos

```
$> gcc --verbose programa.c
```

vemos por qué no es viable

**Usualmente uno
quiere indicar el
nombre del binario
generado.**

```
$> gcc -o binarium programa.c
```

Opciones de compilador recomendadas

-wall

Activa más mensajes de advertencia.

-Wextra

Activa aún más mensajes de advertencia

-Werror

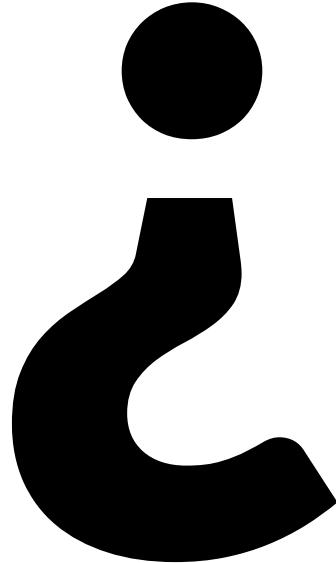
Las advertencias ahora son errores que frenan la compilación



¿Preguntas?



headers



Qué son



1

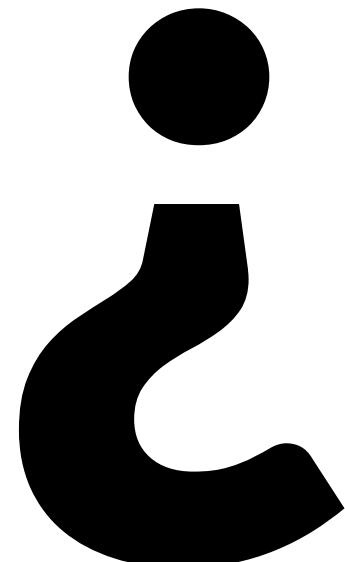
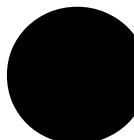
Organización del código

2

Facilitar la reutilización

3

Separación de la interfaz y la implementación



Qué puede contener

Prototipos de funciones

```
int suma(int a, int b);  
int resta(int a, int b);
```

(normalmente) la documentación va acá también

Constantes de preprocesador

```
#define PI 3.14159  
#define CAPACIDAD 300
```

La estructura de un header

La ‘guarda’ clásica

```
#ifndef MATH_OPERATIONS_H
#define MATH_OPERATIONS_H

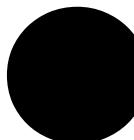
// El contenido del header va acá

#endif
```

Evita que el preprocesador lo inserte múltiples veces

**¡A la
terminal!**





**Y como es la
compilación**

```
$> gcc -o binarium main.c funciones.c
```

Headers importantes

stdio.h

Define printf, scanf, fgets y más.

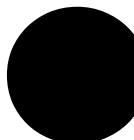
string.h

Contiene las funciones referidas a la manipulación de cadenas

math.h

Contiene las funciones para hacer cálculos como valor absoluto, exponencial, raíz cuadrada, trigonométricas entre otras.

Volviendo a lo anterior

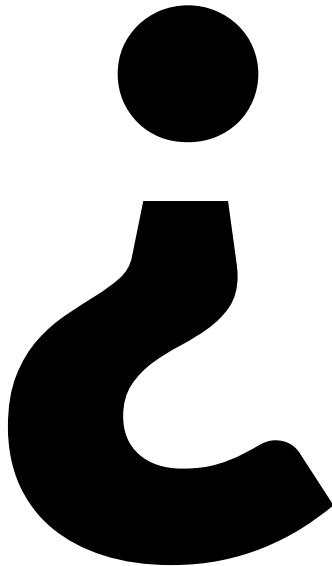


**Cada
compilación
tiene que ser
así**

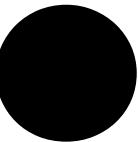
i **Nope!**

makefiles





Qué son



Para programas
más grandes
que un archivo

Regla básica

salida: archivo_entrada
instrucciones



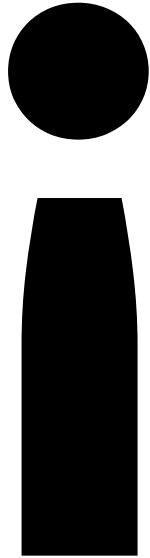
Admite variables

Necesita
indentación con
tabulador

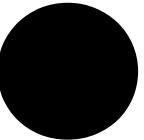
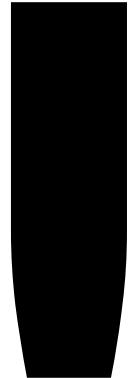
**¡A la
terminal!**



**No es *crítico*
escribir
Makefiles por el
momento**



Pero hay que
entender que
pasa



Las reglas en los próximos TP

make

Ejecuta el primer objetivo del Makefile,
habitualmente usada para solo compilar
el programa.

make clean

Usada para limpiar todo lo que se genera
por **make**

make test

Regla comúnmente usada para ejecutar pruebas del código en el repositorio.

make run

Regla comúnmente usada para compilar y ejecutar el código en el repositorio.

**¡A la
terminal!**



La estructura de las próximas prácticas

Las piezas base

└── Makefile	el Makefile general
└── plantilla	un directorio base
├── ejercicio.c	el archivo con el código
├── ejercicio.h	el encabezado de las funciones
├── main.c	donde va el main
└── Makefile	el Makefile del ejercicio
└── prueba.c	el main de pruebas

1

**No es necesario
cambiar los
Makefiles**

2

**Completen la
plantilla antes de
copiarla para los
ejercicios**

3

**Si quieren cambiar
los nombres de los
archivos es
necesario ajustar el
Makefile**

**¡A la
terminal!**





¿Preguntas?



Y esto
para qué ?

Testing

Objetivos

- Validar Funcionalidad Aislada
- Detectar Errores Tempranamente
- Facilitar Refactorización
- Documentación del Código
- Mejora de la Calidad del Código

y aprovechando
la compilación
separada

Suma lenta

Reinventando la rueda de la suma, de la forma más ineficiente posible

¿Que podemos probar?

**Algunos números
importantes,**

{ $\pm 0, \pm 1, \pm 2, \pm 3, \pm 10$ }

La estructura de una prueba

Puesta en marcha / Preparativos (setup)

```
int a = 5;  
int b = 3;
```

Ejecución

```
int resultado = suma_lenta(a, b);
```

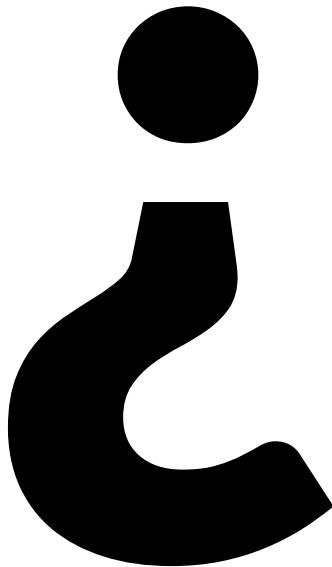
Verificación (aserciones)

```
assert(resultado == 8);
```

Limpieza (teardown)

Acá no es estrictamente necesario, pero más adelante vamos a ver cuestiones que si necesitan de este paso.

```
void test_suma_positivo_positivo() {  
    // Preparativos  
    int a = 5;  
    int b = 3;  
  
    // Ejecución  
    int resultado = suma(a, b);  
  
    // Verificación  
    assert(resultado == 8);  
  
    // Teardown (no es necesario en este caso)  
}
```



assert



assert.h

```
assert(condicion);
```

Si condición es falsa, el programa se abortará mostrando un mensaje de error.

Pero cuidado

```
#define NDEBUG  
#include <assert.h>
```

No está pensado para su uso en “Producción”

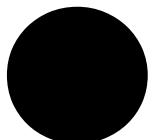
1

Un caso por función de prueba

5

**¡A la
terminal!**





**Como sabemos
contra qué
probar**

Contratos





Qué es un contrato

Historia y Origen

Atributos

Precondición

Son condiciones que deben ser verdaderas antes de que una función sea ejecutada.

Ejemplo - división

```
int division_entera(int dividendo, int divisor)
```

Postcondición

Son condiciones que deben ser verdaderas después de que la función ha sido ejecutada.

Ejemplo - división

```
int division_entera(int dividendo, int divisor)
```

Invariante

Son condiciones que deben ser verdaderas antes y después de la ejecución de una función y a lo largo de toda la vida del programa.

Ejemplo - Lazo de suma en un arreglo

En un bucle que suma los elementos de un arreglo, la suma acumulada siempre sea igual a la suma de los elementos procesados hasta ese punto.

**Lo revisamos para ver
que entendieron de la
consigna**

e j e m p l o s

Descripción de la función - ejemplo 1

```
/*
 * Toda función debe tener un comentario que
 * describa que es lo que hace.
 * Indicando que rol cumple y hace cada argumento.
 * Esta función se encarga de sumar dos numeros enteros
 * @param termino1 es el primer termino de la suma
 * @param termino2 es el segundo termino a sumar
 * @returns la suma de ambos términos
 * PRE-CONDICION: numero enteros en el rango de +/-2^8.
 * POST-CONDICION: numero entero que es suma de los
 *                  términos de entrada. La salida tendrá
 *                  un rango entre +/- 2^9
 */
int suma(int termino1, int termino2);
```

Descripción de la función - ejemplo 2

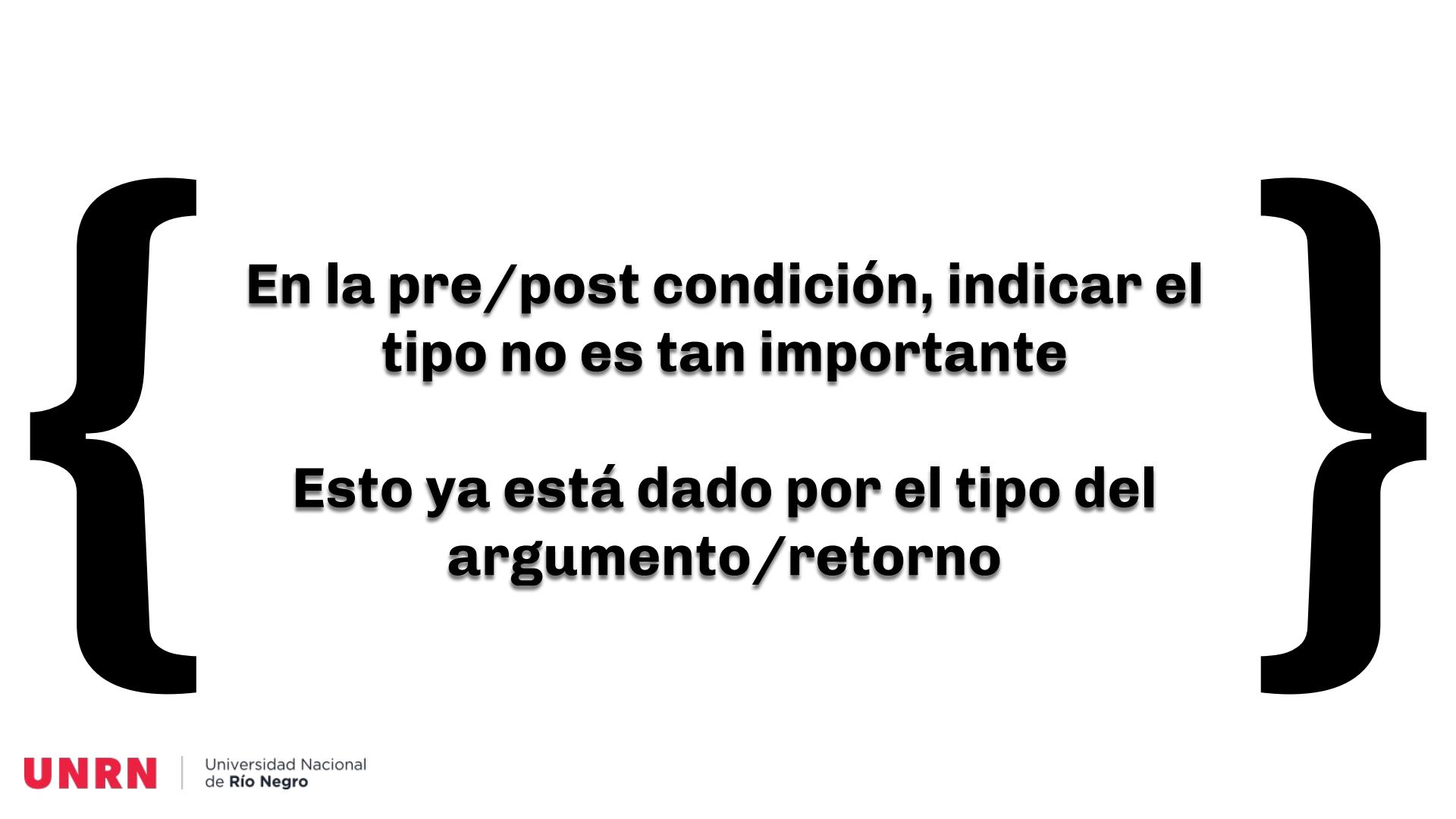
```
/*
 * La descripción de la función.
 * ¿Cuál es su objetivo?
 * @param termino1 es el primer término a operar
 *           debe ser positivo y menor a 3000 (PRE)
 * @param termino2 es el segundo término a operar
 *           debe ser negativo y mayor a 3000 (PRE)
 * @returns la operación de ambos términos
 *          el signo del resultado será el del número mayor
 *          el resultado no puede ser mayor a +/-6000 (POS)
 */
int operar_terminos(int termino1, int termino2);
```

**No hay un estilo
correcto/incorrecto**

**Tiene que
estar**

—

Por otro lado



En la pre/post condición, indicar el tipo no es tan importante

Esto ya está dado por el tipo del argumento/retorno

**Céntrense en las
características que tiene
el valor para probar la
función**

Rango de valores

¿Que valores son válidos para la función?
de entrada y de salida

El divisor no puede ser cero

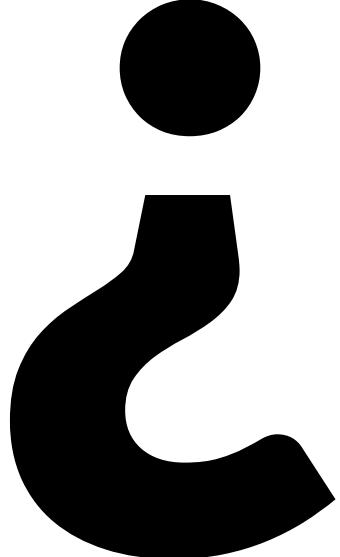
El factorial no puede ser negativo

Valores especiales

Si usan algún valor en particular con un significado diferente al que pueda ser

'c' para celcius

Retorno -1 cuando no puedo calcular



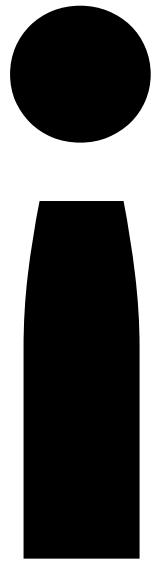
Y las invariantes



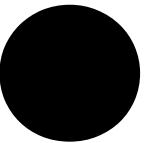
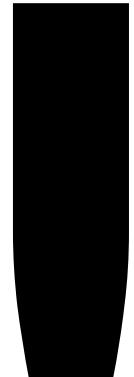
Por ahora no
son importantes

Y todo esto...

¿Por qué?



**Son una invitación
a pasarles esos
valores**



**Hace más fácil
asegurarnos de que hace
lo que supuestamente
debe**

Documentació n

i pruebas!!

TP3

Refuerzo de tests y documentación

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?



unrn.edu.ar

UNRN

Universidad Nacional
de Río Negro



| unrnionegro