



# Teoría 5

## Lenguaje “C”: Introducción

# THE C PROGRAMMING LANGUAGE

Int. a la Computación- Int. a la Programación – Fund. De la Informática

Segundo Cuatrimestre – 2024

## Etapas en el proceso de resolver un problema:

### 1- Comprender el problema: lograr una abstracción

- ▶ Descomposición del problema.



### 2- Bosquejar una solución:

- ▶ Determinar un Algoritmo.



- ▶ Codificar ese algoritmo.

### 4- Volver hacia atrás: perfeccionar la solución

### 3- Ejecutar el plan:

- ▶ Comprobar cada uno de los pasos.
- ▶ Énfasis en la habilidad de ejecutar el plan trazado y no en realizar los cálculos en sí.

- ▶ Ejecutar el algoritmo bosquejado.



Algoritmo



Lenguaje del Problema



Codificación

*Lenguaje de  
Programación*



Implica:

“C”

- ▶ determinar las variables con los que se trabaja.
- ▶ determinar cómo se operan las variables.
- ▶ las estructuras de control que se pueden expresar.
- ▶ la simbología asociada a ellos.





### El Lenguaje C

Pertenece al **Paradigma Imperativo**

**Control del Flujo:** implementa las 3 estructuras de control. En algunos casos, ofrece varias alternativas de sentencias para implementarlas.

**Manipulación de Memoria.** C dispone de sentencias que permite manipular directamente direcciones de memoria, lo cual es una característica destacada del paradigma imperativo.

Es un lenguaje de **alto nivel**.

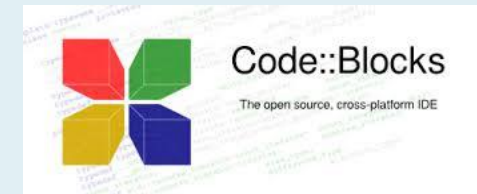
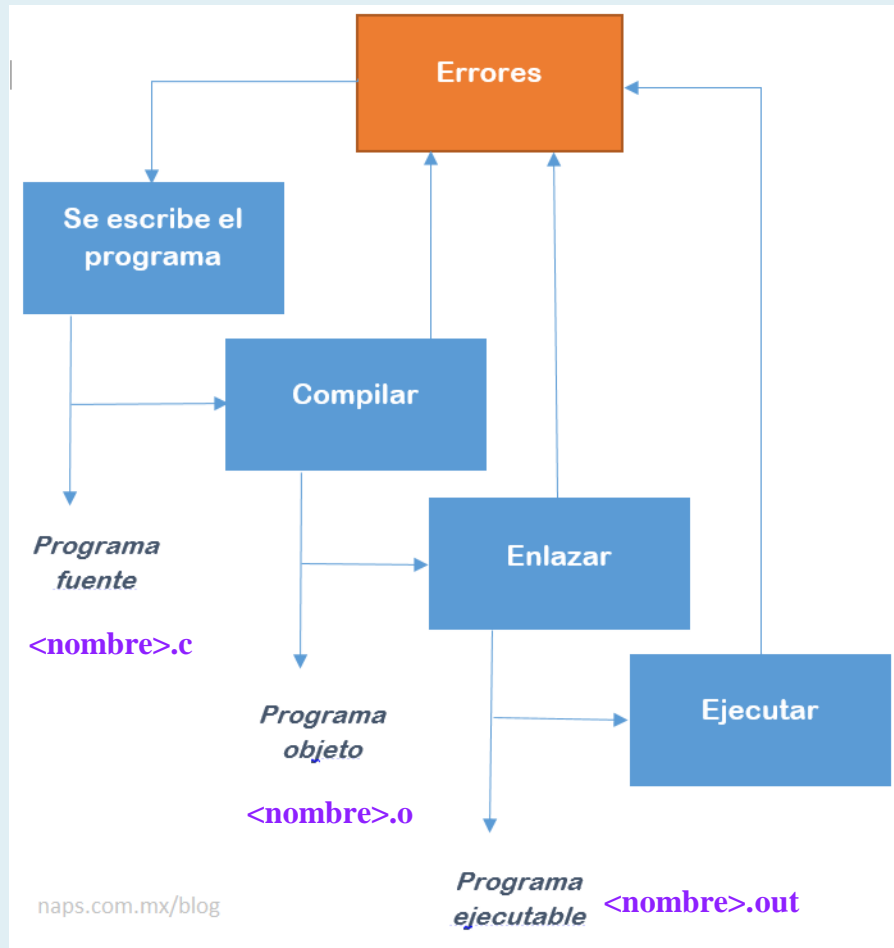
- Abstracción
- Estructuras de Control
- Portabilidad
- Sintaxis Legible

**Soporta** el Paradigma de Modularización

**No soporta** el Paradigma Orientado a Objetos ni Funcional

C es diseñado para ser eficiente y de bajo nivel, lo que lo hace ideal para el desarrollo de sistemas operativos, software embebido, y otras aplicaciones donde el control de hardware y el rendimiento son críticos. se debe a su capacidad para abstraer el hardware y facilitar la escritura de código portátil y legible, manteniendo al mismo tiempo un alto rendimiento y eficiencia.

El Lenguaje C es un lenguaje de alto nivel



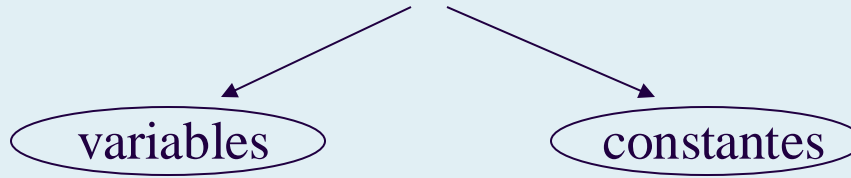
## Conceptos vistos

- Variables (Tipos de datos primitivos)
- Estructuras de Control (secuencia, repetición y selección)
  - Secuencia: está garantizada por el orden en que se escriben las sentencias
  - Selección: sentencia SI...SINO...
  - Repetición:
    - Sentencia MIENTRAS ...
    - Sentencia REPITA ...
- Entrada de datos y Salida de información
- Diagrama de Flujos



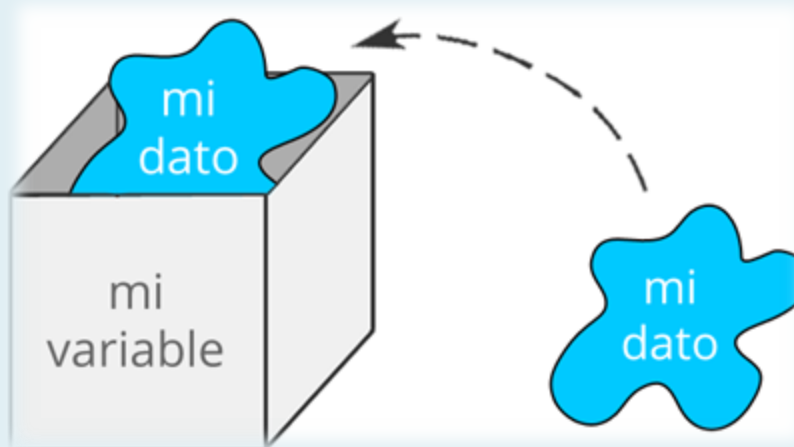
# OBJETOS

## Lenguaje “C”



Variable: recurso del ambiente cuyo contenido puede cambiar.

Constante: recurso cuyo contenido no puede cambiar.



Variable: recurso del ambiente cuyo contenido puede cambiar.



En C siempre se **declaran** las variables y, según sea necesario, se **inician**.

- Favorecen la generación de buenos programas
- Se evitan errores y confusiones





**Variable**: recurso del ambiente cuyo contenido puede cambiar.

**Constante**: recurso cuyo contenido no puede cambiar.

### Variables

- ▶ un *nombre* que la identifica.  
Deben comenzar con una letra o el signo “subrayado” (\_) y luego seguir con cualquier combinación de letras (mayúsculas, minúsculas), dígitos o signo de subrayado.
- ▶ un *tipo* que describe los valores que puede tomar la variable y las operaciones que se pueden realizar con la misma.

### Lenguaje de Problemas

- ▶ entero →
- ▶ real →
- ▶ caracter →
- ▶ lógico →

### Lenguaje “C”

- ▶ entero
- ▶ real (flotante)
- ▶ caracter
- ▶ ?

int  
float  
char



El *tipo entero*, consiste de un conjunto finito de valores de los números enteros. La cardinalidad del conjunto depende de las capacidades del procesador. En general el rango de valores es **-32768 a 32767 (2bytes)**.

El *tipo real (flotante)*, consiste de un conjunto finito de valores de los números reales. La cardinalidad del conjunto depende de las capacidades del procesador. En general el rango de valores es **-3.4E<sup>+38</sup> a 3.4E<sup>+38</sup> (4 bytes)**.

El *tipo character*, permite representar caracteres individuales. El conjunto de caracteres posibles se encuentra detallado en una tabla denominada **ASCII**. En general el rango de valores ASCII es **0 a 127 ( 1 byte)**.

## Declaración de variables en Lenguaje C



Reserva de  
espacio de  
memoria

Toda variable DEBE ser declarada antes  
de ser usada

**<tipo de datos> <Nombre de la variable>**



## Declaración de variables en Lenguaje C



Reserva de  
espacio de  
memoria

<tipo de datos> <Nombre de la variable>



## Ejemplos

```
int factorial  
float factorial  
char letra
```

```
int i
```

```
int I
```



Son dos variables diferentes



## Constantes

- ▶ un *nombre* que la identifica.  
Deben comenzar con una letra o el signo "subrayado" (\_) y luego seguir con cualquier combinación de letras (mayúsculas, minúsculas), dígitos o signo de subrayado. Por convención los nombres de constantes se escriben en **MAYUSCULAS**.
- ▶ un *tipo* que describe los valores que puede tomar la constante y las operaciones que se pueden realizar con la misma.

## Ejemplo

### Lenguaje de Problema

valores expresados  
directamente en el  
código

Conta



### Lenguaje "C"

Modificador de acceso

```
const int V_ENTERO = 15
const float V_REAL = 15.0
const char LETRA = 'a'
```

## Operaciones entre objetos

### Tipos:

- ▶ **Aritméticas.**
- ▶ **Lógicas**
- ▶ Relacionales

### Operadores Relacionales

operador	descripción
<	menor
>	mayor
<=	menor o igual
>=	mayor o igual
==	igual
!=	distinto

## Operadores Aritméticos

operador	descripción
+	suma
-	resta
*	producto
/	división
%	módulo

### Operadores Lógicos

operador	descripción
&&	and
	or
!	not



### Precedencia y asociatividad

Nivel	Operadores	Descripción	Asoci.
1	() [] -> .	Acceso a un elemento de un vector y paréntesis	Izquierdas
2	+ - ! ~ * & ++ -- (cast) sizeof	Signo (unario), negación lógica, negación bit a bit Acceso a un elemento (unarios): puntero y dirección Incremento y decremento (pre y post) Conversión de tipo (casting) y tamaño de un elemento	Derechas
3	* / %	Producto, división, módulo (resto)	Izquierdas
4	+ -	Suma y resta	Izquierdas
5	>> <<	Desplazamientos	Izquierdas
6	< <= >= >	Comparaciones de superioridad e inferioridad	Izquierdas
7	== !=	Comparaciones de igualdad	Izquierdas
8	&	Y (And) bit a bit (binario)	Izquierdas
9	^	O-exclusivo (Exclusive-Or) (binario)	Izquierdas
10		O (Or) bit a bit (binario)	Izquierdas
11	&&	Y (And) lógico	Izquierdas
12		O (Or) lógico	Izquierdas
13	?:	Condicional	Derechas
14	= *= /= %= += -= >>= <<= &= ^=  =	Asignaciones	Derechas
15	,	Coma	Izquierdas





### Precedencia y asociatividad

Nivel	Operadores	Descripción	Asoci.
1	() [] -> .	Acceso a un elemento de un vector y paréntesis	Izquierdas
2	+ - ! ~ * & ++ -- (cast) sizeof	Signo (unario), negación lógica, negación bit a bit Acceso a un elemento (unarios): puntero y dirección Incremento y decremento (pre y post) Conversión de tipo (casting) y tamaño de un elemento	Derechas
3	* / %	Producto, división, módulo (resto)	Izquierdas
4	+ -	Suma y resta	Izquierdas
5	>> <<	Desplazamientos	Izquierdas
6	< <= >= >	Comparaciones de superioridad e inferioridad	Izquierdas
7	== !=	Comparaciones de igualdad	Izquierdas
8	&	Y (And) bit a bit (binario)	Izquierdas
9	^	O-exclusivo (Exclusive-Or) (binario)	Izquierdas

La tabla resume las reglas de precedencia y asociatividad de todos los operadores, incluyendo aquellos que no usaremos en este curso.

- Los operadores en un mismo nivel tienen igual precedencia. Así, la suma y la resta tienen igual precedencia.
- Las filas están en orden **decreciente** de precedencia. Así, la \*, / y % (fila 3) tienen más precedencia que la + y - (fila 4)



## Operador de Asignación

Las variables reciben un valor a través de una operación de asignación

## Ejemplos

### Lenguaje de Problemas

T<sub>i</sub> Asignar a v\_entero el valor 10

c ← (a + 1) + b

a ← b

d ← el resto de c/4



### Lenguaje “C”

v\_entero = 10

c = (a + 1) + b

a = b

d = c%4

El operador de asignación en C es el símbolo =





Se debe tener cuidado cuando se mezclan enteros con flotantes pues se puede obtener resultados inesperados

Resultado a almacenar

```
int v_entero
float v_real
int v_resul
v_entero = 2
```

`v_real = 7 / v_entero` → 3.0

`v_real = 7 / 2.0` → 3.5

`v_resul = v_entero + 3` → 5

`v_real = (v_entero * 1.0) + 3.2` → 5.2

`v_entero = v_real + 3` → 8

**Conversión de Tipos:** el tipo “**menor**” es promovido al tipo “**mayor**” antes de que la operación se resuelva.



# Tabla ASCII

## Lenguaje "C"

0	NUL	32	space	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	O	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93	]	125	}
30	S	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Internamente, no existe una diferencia real entre los caracteres y los enteros. El tipo **char** es un caso particular de un entero, enteros en el rango **0** al **127** (128 en total).

Cada caracter de la tabla ASCII tiene asociada una posición numérica en ella. Se podría utilizar entonces su número de posición para poder referenciar un caracter en particular.

```
int v_entero  
char letra
```

```
letra = 'A'  
Letra= 65
```

Resultado a almacenar



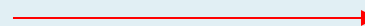
```
v_entero = letra * 2
```



**130**



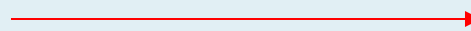
```
v_entero = letra + '$'
```



**101**

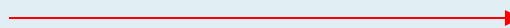


```
letra = 'A' + '$'
```



**e**

```
letra = 'A' + 5
```



**F**

# Tabla ASCII

## Lenguaje "C"

0	NUL	32	space	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	O	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3			83	S	115	s
20	DC4			84	T	116	t
21	NAK			85	U	117	u
22	SYN			86	V	118	v
23	ETB			87	W	119	w
24	CAN			88	X	120	x
25	EM			89	Y	121	y
26	SUB			90	Z	122	z
27	ESC			91	[	123	{
28	FS			92	\	124	
29	GS			93	]	125	}
30	S			94	^	126	~
31	US			95	_	127	DEL

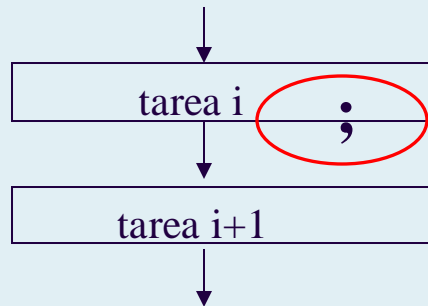
char Letra = 'a'  
 Letra = Letra - 32  
 ¿Contenido de Letra?

## Estructuras de Control

- Secuencial
- Condicional
- Repetición o Iteración



## Estructuras de Control: Secuencial



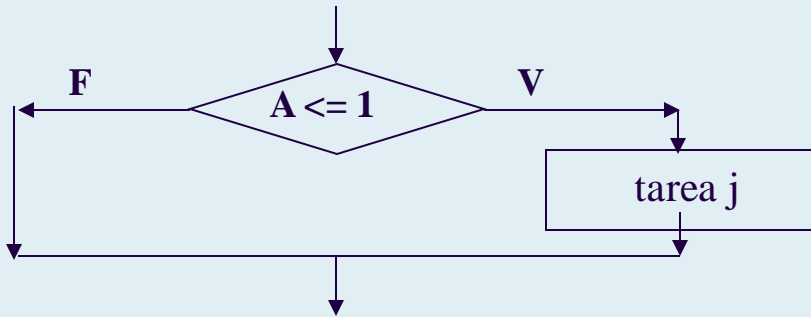
En "C" la secuencia de dos acciones (sentencias) se expresa por medio del símbolo ; (punto y coma)

```

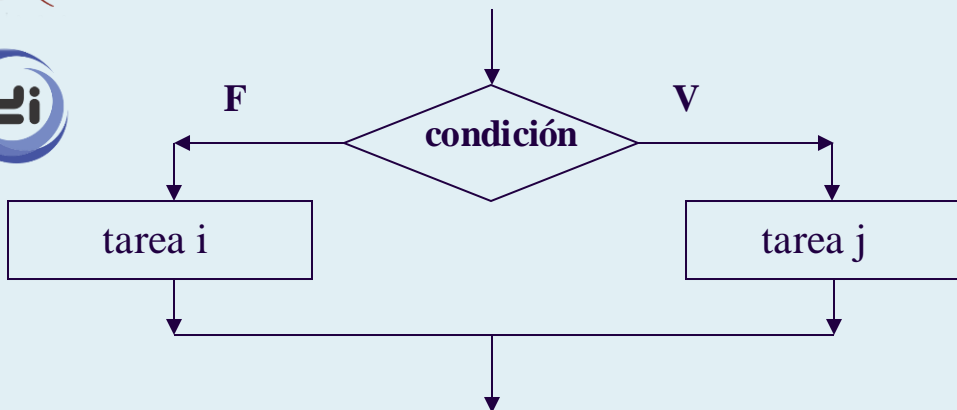
int v_entero;
char letra;

letra = 'A';
v_entero = letra * 2;
v_entero = letra + '$';
letra = 'A' + '$';
letra = 'A' + 5;
  
```





```
if (<condición>) {  
    <sentencias>  
}
```



```
if ( A== 10)  
{  
    <sentencias>  
}  
else  
{  
    <sentencias>  
}
```

## Ejemplo:

a- Dados dos valores enteros si el primero es mayor que el segundo, intercambiarlos.

```
int v_ent1;  
int v_ent2;  
int aux;  
  
v_ent1 = 30;  
v_ent2 = 20;  
aux = 0;  
if (v_ent1 > v_ent2) {  
    aux = v_ent1;  
    v_ent1 = v_ent2;  
    v_ent2 = aux;  
}
```



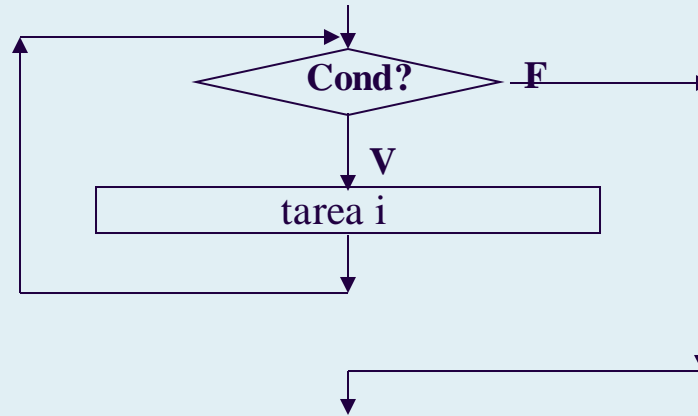


## Ejemplo:

b - Dados dos valores enteros si el primero es mayor que el segundo, intercambiarlos, caso contrario dejar indicado en la variable auxiliar que no se hizo ningún cambio.

```
int v_ent1;  
int v_ent2;  
int aux;  
  
v_ent1 = 30;  
v_ent2 = 20;  
aux = 0;  
if (v_ent1 > v_ent2) {  
    aux = v_ent1;  
    v_ent1 = v_ent2;  
    v_ent2 = aux;  
}  
else {  
    aux = -1;  
}
```





```
while (<condición>) {  
    <sentencias>  
}
```

La/s sentencia/s se ejecutará/n mientras la condición sea verdadera. Puede suceder que si la condición es falsa en la primer evaluación la/s sentencia/s no se ejecuten nunca.

**Ejemplo:** a- Dados un valor entero, calcular su factorial.

(en Pseudocódigo)

Definir las variables NUMERO, FACTORIAL, AGREGO del tipo entero

Asinar a NUMERO el valor 4

Asignar a FACTORIAL el valor 1

Asignar a AGREGO el valor 1

**MIENTRAS** AGREGO <= NUMERO **HACER**

    FACTORIAL ← FACTORIAL \* AGREGO

    AGREGO ← AGREGO + 1

**FINMIENTRAS**



(en Lenguaje “C”)

```
int numero;
int factorial;
int agregado;

numero = 4;
factorial = 1;
agregado = 1;
while (agregado <= numero) {
    factorial = factorial * agregado;
    agregado = agregado + 1;
}
```

## Repetición: While versus For

```
for (<sentencia inicial>; <condición>;<sentencia iteración>) {
    <sentencias cuerpo>
}
```

```
while (<condición>) {

    <sentencias cuerpo>
    <sentencia iteración>
}
```



```

i = 1;
while (i <= 10) {
    <sentencias cuerpo>;
    i = i+1;
}

for (i=1; i <=10; i=i+1) {
    <sentencias cuerpo>;
}
```

Diagram illustrating the equivalence between a while loop and a for loop. The while loop code is shown above the for loop code. Colored ovals and arrows highlight the mapping: a red oval around 'i = 1;' in the while loop maps to 'i=1' in the for loop; a blue oval around '(i <= 10)' in the while loop maps to 'i <=10' in the for loop; and a green oval around 'i = i+1;' in the while loop maps to 'i=i+1' in the for loop. Arrows also show the flow of execution from the initialization to the condition and then to the body of each loop.

## Ejemplo:(código del cálculo de factorial con “while”)

## Lenguaje “C”

```
int numero;  
int factorial;  
int agregó;  
numero = 4;  
factorial = 1;  
agregó = 1;  
while (agregó <= numero) {  
    factorial = factorial * agregó;  
    agregó = agregó + 1;  
}
```

(código del cálculo  
de factorial con “for”)

```
int numero;  
int factorial;  
int agregó;  
  
numero = 4;  
factorial = 1;  
for (agregó = 1; agregó <= numero;  
    agregó = agregó + 1)  
{  
    factorial = factorial * agregó;  
}
```



## Estructura del programa

La/s sentencia/s que se ejecutará/n deben estar reunidas u organizadas de manera que el procesador sepa donde comienza y donde termina un programa en lenguaje “C”.

(en Lenguaje “C”)

```
int main() {  
    ...Sentencias ejecutables...  
    return (0);  
}
```



La instrucción **return(0)** debe ser la última instrucción del programa. Esta instrucción sirve como ayuda para conocer si la ejecución del algoritmo terminó exitosamente.

## Ejemplo:

(en Lenguaje “C”)

```
int main() {  
    int numero;  
    int factorial;  
    int agregado;  
  
    numero = 4;  
    factorial = 1;  
    for (agregado = 1; agregado <= numero;  
        agregado = agregado + 1) {  
        factorial = factorial * agregado;  
    }  
    return (0);  
}
```

En diferentes renglones,  
tres sentencias que  
declaran a 3 variables  
enteras. Cada  
declaración se separa con  
el símbolo ;



## Ejemplo:

(en Lenguaje “C” – Otra alternativa)

## Lenguaje “C”

En un mismo renglón,  
tres sentencias que  
declaran a 3 variables  
enteras. Cada  
declaración se separa  
con el símbolo ;

```
int main() {  
    int numero=4;int factorial; int agrego;  
  
    factorial = 1;  
    for (agrego = 1; agrego <= numero;  
        agrego = agrego +1) {  
        factorial = factorial * agrego;  
    }  
    return (0) ;  
}
```





## Ejemplo:

## Lenguaje “C”

(en Lenguaje “C” – Otra alternativa)

Sentencia que  
declara una **lista**  
de variables  
enteras

```
int main() {  
    int numero=4, factorial, agrego;  
  
    factorial = 1;  
    for (agrego = 1; agrego <= numero;  
        agrego = agrego +1) {  
        factorial = factorial * agrego;  
    }  
    return (0) ;  
}
```





## Lenguaje “C”

# Entrada de Datos y Salida de Información

## El preprocesador

En un programa codificado en Lenguaje C, es posible incluir dentro del código fuente del programa, diversas instrucciones (o directivas) para el compilador y que permiten aumentar el ámbito del entorno de programación de C

El preprocesador contiene algunas directivas como por ejemplo `#if`, `#ifdef`, `#line`, **`#include`** ... entre otras

**`#include` nombre del archivo**

Obliga al compilador a **incluir** otro archivo fuente en el archivo que contiene la directiva y, luego, **compilarlo**.

**`#include <stdio.h>`**   ó   **`#include "mi_biblioteca.h"`**



`#include <stdio.h>`   ó   `#include "mi_biblioteca.h"`

## Lenguaje "C"

`< >`: Se usa para bibliotecas estándar del sistema. El compilador buscará en las rutas predefinidas donde se almacenan las bibliotecas de C.

`" "`: Se usa para bibliotecas definidas por el usuario o locales. El compilador busca primero en el directorio donde está el archivo fuente, y luego en las rutas estándar si no lo encuentra.

**Bibliotecas Estándar de C:** `stdio.h` - `math.h` – `stdlib.h` - entre otras

`#include <stdio.h>`

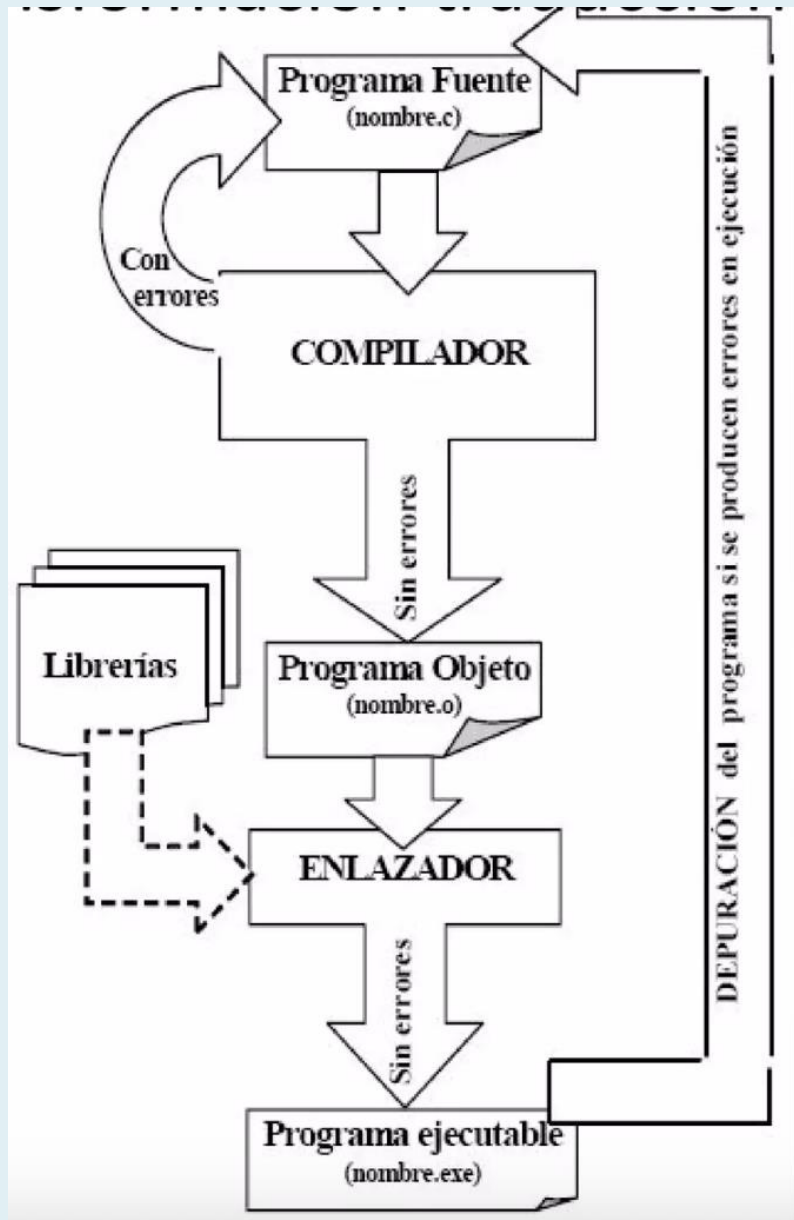
El archivo cabecera **`stdio.h`** (librería estándar de I/O) contiene la declaración de cómo se hace la entrada de datos y la salida de información, y todo lo necesario para que se lleven a cabo estas acciones con éxito



## Lenguaje “C”

Es obligación incluir entre las primeras sentencias de un programa en C la sentencia

**#include <stdio.h>**



## Salida de Información

La salida de información en “C” no se realiza a través de una acción primitiva, sino a través de una función denominada **printf**. Las funciones son instrucciones que realizan una tarea en particular (el mismo concepto al de subalgoritmo visto en el lenguaje de diseño)

### Sintáxis

```
printf(<formato>, expresion1, expresion2, ...)
```



Texto asociado a los valores a ser impresos. Puede ocurrir que no se desee asociar ningún valor al texto.

Expresiones cuyo resultado se desea imprimir luego de ser calculadas. Pueden ser simplemente el contenido de una variable.

### Restricción

Dado que existen diversos tipos de datos y la función es de uso general, se debe especificar el tipo (formato) de la información que se va a mostrar. En general el formato coincide con el tipo resultante de la expresión.

## Ejemplos:

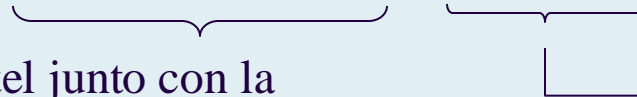
a) `printf("Esto es un cartel")`



Formato: solo es un cartel, sin expresiones asociadas.

b) `int numero;`

```
numero = 4;
printf("El valor es %d", numero);
```



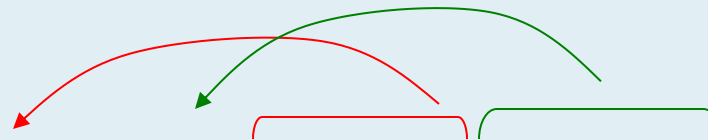
Formato: cartel junto con la especificación de cómo debe mostrarse el valor almacenado en "numero".

Expresión: es simplemente el nombre de una variable.

c)

```
int numero;
```

```
numero = 4;
printf("El doble de %d es %d", numero, numero*2);
```



## Especificaciones de Formato

caracter	significado
%d	entero decimal
%f	flotante
%c	caracter

### Ejemplo:

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

```
int main() {
    int numero;
    int factorial;
    int agrego;

    numero = 4;
    factorial = 1;
    for (agrego = 1; agrego <= numero;
        agrego = agrego +1) {
        factorial = factorial * agrego;
    }
    printf("El Factorial de %d es %d", numero,
        factorial);

    return(0);
}
```

**stdio.h**, que significa "standard input-output header" (cabecera estandar E/S), es la biblioteca estándar del lenguaje C. El archivo de cabecera contiene las definiciones de macros, las constantes, las declaraciones de funciones y la definición de tipos usados por varias operaciones estándar de entrada y salida

Lugar donde se encuentra el código de la función





```
printf("El Factorial de %d es %d", numero, factorial);
```

## Salida en pantalla:

```
El Factorial de 4 es 24
```

## Variantes:

```
El número ingresado es: 4  
Su Factorial es: 24
```

```
printf("El número ingresado es: %d \n", numero);  
printf("Su Factorial es: %d", factorial);
```



## Ingreso de Datos

El ingreso de información en “C” se realiza de una manera muy simple: caracter a caracter, a través de una función denominada **getchar**.

### Sintáxis

`getchar()`

Retorna el valor ASCII del carácter (tecla) que se ingresó desde teclado.

### Uso

```
int c;
c = getchar();    /* en "c " se encuentra el
                  valor ASCII de la tecla apretada */
```

Tecla apretada

Contenido de “c”

▶ “a”	97
▶ “F”	70
▶ “3”	51
▶ “)”	41



# Lenguaje “C” – Ingreso de Datos

## Ejemplo

```
#include <stdio.h>

int main() {

    int c, i;

    i = 1;
    while (i != 0) {
        c = getchar();
        if (c >= 'a' && c <= 'z') {
            printf("Es letra minúscula \n");
        }
        if (c >= 48 && c <= 57) {
            printf("Es dígito numérico \n");
        }
    }
    return(0);
}
```

**stdio.h**, que significa "standard input-output header" (cabecera estandar E/S), es la biblioteca estándar del lenguaje C. El archivo de cabecera contiene las definiciones de macros, las constantes, las declaraciones de funciones y la definición de tipos usados por varias operaciones estándar de entrada y salida

### Inconveniente:

- Como interpretar información consistente de más de un carácter (strings, números).

### Ejemplos:

- La estrella es brillante
- 324
- 127,98



# Lenguaje “C” – Ingreso de Datos

## Primer Solución:

$t_1$  - Ingresar los caracteres en un arreglo.

$t_2$  - Procesar el arreglo identificando lo que se desee.



## Hasta cuando se leen caracteres?

- Se necesita una señal de finalización (delimitador) de ingreso que le indique al programa cuando finalizar  $t_1$  y comenzar con  $t_2$ .



Tecla  
**Return ó Enter**

Carácter asociado  
**'\n'**

## Ejemplo

## Lenguaje “C” – Ingreso de Datos

```
#include <stdio.h>

int main() {

    char letra1, char letra2;
    int j;

    /* Ingreso los caracteres */
    j = 1;
    letra1 = getchar();
    while (letra1 != '\n') {
        j = j+1;
        if (letra1 >='A' && letra1 <='Z' {
            letra2= letra1+32;
            printf ("Ingreso %c y se transformo en %c", letra1, letra2);
        }
        letra1 = getchar();
    }

    return(0);
}
```



# Lenguaje “C” – Ingreso de Datos

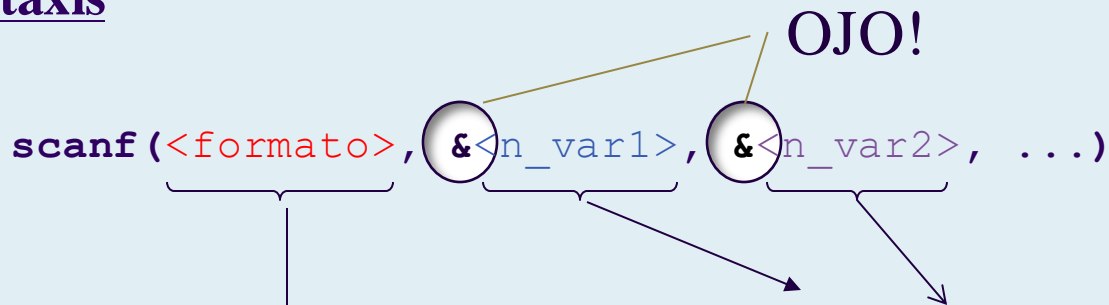
## Segunda Solución

El lenguaje “C” provee una función que realiza esas dos tareas por nosotros denominada **scanf**.

## Sintáxis

**OJO!**

```
scanf (<formato>, &<n_var1>, &<n_var2>, ...)
```



Formato de interpretación a dar a los caracteres ingresados. Se respeta la modalidad usada por **printf**.

Nombres de variables **donde** se almacenarán los valores interpretados desde teclado.

## Especificaciones de Formato

%d → entero decimal  
%f → flotante  
%c → caracter



# Lenguaje “C” – Ingreso de Datos

## Ejemplos:

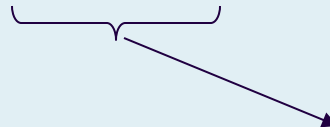
a)

```
int numero;
```

```
scanf ("%d", &numero);
```



Formato **entero** a dar a los caracteres ingresados.



Variable donde quedará almacenado el valor interpretado.



# Lenguaje “C” – Ingreso de Datos

## Ejemplos:

```
#include <stdio.h>
int main()
{
    int a,b;
    printf("\nIntroduce el valor de a: ");
    scanf("%d",&a);
    getchar();
    printf("\nIntroduce el valor de b: ");
    scanf("%d",&b);
    getchar();
    if (b!=0)
        printf("\nEl valor de %d dividido %d es: %f\n", a,b,a/b);
    else
        printf("\nError, b vale 0\n");
    return 0;
}
```





## **Referencias:**

- Tutorial del Lenguaje “C”, Dpto. de Informática.
- Practical “C” Programming, Steve Oualline.
- El Lenguaje de Programación “C”, C. Brian, W. Kerninghan and D. Ritchie.