LENGUAJE C: IDENTIFICADOR

- Es una secuencia de caracteres que pueden ser de cualquier longitud. Representa objetos de un programa (constantes, variables, funciones, etc.).
- Debe comenzar con una letra y no puede contener espacios en blanco. Letras, dígitos y caracteres de subrayado están permitidos después del primer caracter.
- No se puede utilizar una **palabra reservada** como identificador.

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**
case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unisgned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
do	**if**	**static**	**while**

LENGUAJE C: TIPOS DE DATOS

- **Datos**: información que procesa el programa. Pueden ser diferentes objetos.
  - **Constantes**: objetos cuyo **valor no cambia** durante la ejecución (PI - COLOR).
  - **Variables**: objetos cuyo **valor cambia** durante la ejecución (edad, nombre).
- Todos los datos tienen un **tipo** asociado, que determina los valores que una variable puede tomar. La asignación de tipos a los datos tiene dos objetivos:
  - Detectar errores de operaciones en programas.
  - Determinar como ejecutar las operaciones.
- Tipos de datos simples: **int** – **float** – **double** – **char**.
- **int** y **char** admiten modificadores: **short** – **signed** – **unsigned** – **long** (**double**).
- Otros tipos de datos: **void** – punteros.
- Los datos pueden ser simples (un único elemento) o **estructurados** (varios elementos): **vectores**, **matrices**, **strings** (cadena de caracteres) y **estructuras**.

TIPOS DE DATOS SIMPLES

Tipo	Descripción	Tamaño	Valor Mínimo	Valor Máximo
char (signed)	Caracteres alfanuméricos	1 byte	-128	127
unsigned char	Caracteres	1 byte	0	255
int (signed short)	Número entero	2 bytes	-32.768	32.767
int (unsigned)	Número entero	2/4 bytes	0	65535
long (signed)	Número entero	4 bytes	-2.147.483.648	2.147.483.647
unsigned long	Número entero	4 bytes	0	4.294.967.295
float	Número real, simple precisión (7 decimales)	4 bytes	3.4e-38	3.4e38
double	Número real , doble precisión (15 decimales)	8 bytes	1.7e-308	1.7e308
long double	Número real (19 decimales)	10/12 bytes	3.4e-4932	3.4e4932

- Para saber cuantos bytes ocupa un objeto se utiliza la función **sizeof**(*objeto*)

TIPOS DE DATOS SIMPLES

- Tipo caracter (ASCII): siempre se escriben entre comillas simples.

```
char letra = 'b';

char salto_linea = '\n'        /* secuencias de escape = caracteres especiales */
```

- Tipos enteros: C admite tres formas de representarlos.

```
int  valor1 = 17;              /* decimal , base = 10 */

int  valor2 = 021;             /* octal , base = 8 */

int  valor3 = 0x11;            /* hexadecimal , base = 16 */
```

- Tipos reales:

```
float  valor1 = 3.14e-2;          /* notación exponencial  = 3.14x10-2 */

float  valor2 = .34;              /* correcto pero conviene escribir 0.34 */

double valor3 = -3.27e58;         /* permite calcular con mayor precisión */
```

FLOAT vs. DOUBLE – FORMATO IEEE-754

	float (4 bytes) 32 bits	double (8bytes) 64 bits	
1.45e-3	3ABE0DED	3F57C1BDA5119CE0	1.45e-3
1.45000009797513 48e-3	3ABE0DEE	3F57C1BDA5119CE1	1.4500000000000001e-3
1.4499998651444912e-3	3ABE0DEC	3F57C1BDA5119CDF	1.4499999999999997e-3
1.45000001e-3	3ABE0DED	3F57C1BDA7D14CDD	1.4500000100000000e-3

- Existen páginas que permiten realizar las conversiones:

  Decimal (punto flotante) → Hexadecimal (en 32 y 64 bits)

  Hexadecimal → Punto Flotante

TIPO DE DATOS VOID

- Sólo se utiliza para:
  - Indicar que una función no tiene argumentos: **int** mi_funcion(**void**);
  - Indicar que una función no devuelve ningún valor: **void** mi_funcion(**int**);
  - Crear punteros genéricos: **void** *mi_puntero;

TIPO DE DATOS ESTRUCTURADOS: STRINGs

- Secuencia de caracteres: siempre se escriben entre comillas dobles. “valor = 4.56”
- Se almacenan en memoria como una secuencia de códigos ASCII. Siempre terminan con un carácter nulo ‘\0’ = 0x0 = 0 que se inserta automáticamente.

  “HOLA MUNDO” → ‘H’,’O’,’L’,’A’,’ ‘,’M’,’U’,’N’,’D’,’O’,’\0’ (Vector de **char**)

  valores en memoria: 72 – 79 – 76 – 65 – 32 – 77 – 85 – 78 – 68 – 79 – 0
- Existe un librería estándar para trabajar con strings: **string.h**.

LENGUAJE C: VARIABLES

- Objetos del programa cuyo valor puede cambiar en tiempo de ejecución.
- Deben ser declaradas antes de usarlas (se reserva espacio en memoria).
- Formato declaración: <*tipo de dato*> <*identificador*> ; (tamaño nombre_variable)
- Asignación: otorga valor a la variable (guarda su valor en la memoria reservada).
  - En C: signo = Ej: num = 3.1728; /* asigna el valor 3.1728 a num */
  - En pseudocódigo: ← (asigna a la variable de la izquierda el valor de la derecha).
- Declaración simple:

```
char car;
unsigned int valor;
```

- Declaración múltiple:

```
char c, d, tecla;                    /* resta legibilidad */
unsigned int valor, cantidad;
```

- Declaración y asignación:

```
char tecla = 'A';
unsigned int i = 133, j = 1229;      /* resta legibilidad */
```

LENGUAJE C: CONSTANTES

- Identificadores del programa cuyo valor es fijo y no puede ser alterado por el usuario (no cambia en tiempo de ejecución).
- Pueden ser: Numéricas (enteros y reales) – Caracteres – Cadena de caracteres.
- Los nombres se suelen escribir en **mayúsculas** (para distinguir de la variables).
- Formato declaración: C: `#define PI 3.1415927` /* **directiva** para compilador */

  C: **const float** PI = 3.1415927**;** /* instrucción */

  Pseudocódigo: constante PI ← 3.1415927
- Directiva `#define`: se escribe en la cabecera del fichero luego de los `#include`.
- Modificador **const**: se escribe dentro de una función y finaliza con '**;**'.
- El compilador genera código más eficiente utilizando **const** y además puede comprobar si el tipo declarado coincide con el valor asignado (facilita la detección de errores). No se puede usar para dimensionar un array.

LENGUAJE C: MACROS

- Definen funciones que son expandidas en fase de pre-procesamiento:

  #define *evaluar*(*args*) función_a_evaluar

  #define AREA(radio) 3.1415927*radio*radio

LENGUAJE C: CASTING

- El lenguaje C permite declarar variables de un tipo y asignarles directamente valores de otro tipo. En algunos casos se generan **warnings** (avisos) para indicar que se puede producir un error, como por ejemplo asignarle a un **int** un **float** (se pierden los dígitos decimales). El casting es el mecanismo usado para hacer conversiones de tipo de variables (en algunos casos es automático).

```
int a;
float b=65.32;
char c;

a = (int) b;        /* a = 65 */
c = (char) a;       /* c = 'A' , corresponde al código ASCII 65 */
```

LENGUAJE C: EXPRESIONES (OPERACIONES)

- Son combinaciones de datos mediante uno o mas operadores (uso de paréntesis).
- Los datos (operandos) pueden ser variables, constantes y nombres de funciones.
- Para formar las expresiones los operandos deben ser del mismo tipo (conversión automática) y respetar la precedencia de los operadores (producto, suma, etc.).
- Cada expresión retorna un valor que se determina tomando los valores de variables y constantes implicadas y la ejecución de las operaciones indicadas.
- Los operadores pueden ser de distinto tipo:
  - Aritméticos.
  - Relacionales.
  - Lógicos.
- De acuerdo a la cantidad de operandos los operadores pueden ser:
  - Unarios: afectan a un solo operando.
  - Binarios: requieren de dos operandos.
  - Ternario: produce dos valores distintos dependiendo de una condición.

LENGUAJE C: OPERADORES ARITMETICOS

- Requieren de 2 operandos (binarios).
- Suma (+): `c = a + b;`
  `c += b;` /* equivale a c = c + b */
- Resta (-): `c = a - b;`
  `c -= b;` /* equivale a c = c - b */
- Multiplicación (*): `c = a * b;`
  `c *= b;` /* equivale a c = c * b */
- División (/): `c = a / b;`
  `c /= b;` /* equivale a c = c / b */

  Si ambos operandos son enteros, el cociente también lo es → 4/3 = 1

  Si uno de los operandos es real, el resultado es real → 4.0/3 = 4/3.0 = 1.3333
- Módulo (%): resto de la división. Los operandos deben ser enteros.
- No existen los operadores potencia y raíz cuadrada. Se deben utilizar las funciones pow() y sqrt() de la librería **math.h**.

LENGUAJE C: OPERADORES RELACIONALES

- Se utilizan para expresar condiciones y describen una relación entre **2** valores.
- El resultado de una expresión relacional es **Verdadero** (**1**) ó **Falso** (**0**).
- Relaciones:
  - < Menor que , a < b será V si a es menor que b.
  - <= Menor o igual que , a <= b será V si a es menor o igual que a.
  - > Mayor que , a > b será V si a es mayor que b.
  - >= Mayor que , a >= b será V si a es mayor o igual que b.
- Operadores de igualdad:
  - == Igual que, a == b será V si a es igual a b.
  - != Distinto de, a != b será V si a NO es igual a b.

OBSERVACION: no debe confundirse el operador de igualdad (**==**) que produce un valor **1** o **0**; con el de asignación (**=**) que establece el valor de un objeto.

LENGUAJE C: OPERADORES LOGICOS

- Actúan sobre operandos de tipo lógico o **booleano**. Suponiendo un valor de num cualquiera distinto de 0, entonces (num == 0) → FALSE y (num != 0) → TRUE (V).
- Una expresión lógica combina variables, literales, operadores aritméticos, relacionales y lógicos. El resultado es 1 (V) ó 0 (F).
- Operadores: conjunción (**AND**, **&&**), disyunción (**OR**, **||**) y negación (**NOT**, **!**).
- Tablas de verdad:

AND

A	B	A&&B
F	F	F
F	V	F
V	F	F
V	V	V

OR

A	B	A||B
F	F	F
F	V	V
V	F	V
V	V	V

NOT

A	!A
F	V
V	F

- Ejemplos:

```
valorlog  =  numero > 0 ;            /* valorlog = 1 si numero es positivo */
a = ( (3>2) || (5==4) ) &&  (!1);          /* a=(3>2||5==4)&&!1; → a = 0 */
valor = ( (num<=100) && (num>=25) ) ;     /* = 1 si 25 <= num <= 100 */
valor = ( (num<25) || (num>100) );       /* =1 si  num < 25 ó num > 100 */
```

LENGUAJE C: OPERADORES UNARIOS

- **Incremento** (**++**): equivale a sumar uno a la variable. x++; → x = x + 1;
- **Decremento** (**--**): equivale a restar uno a la variable. x--; → x = x - 1;
- **Pre**/**post**-incrementos (decrementos): dependiendo de la posición de los símbolos ++ (--) la variable incrementa (decrementa) su valor antes o después de utilizarse:

  x++: utiliza el valor de x y luego lo incrementa en 1.

  ++ x: primero incrementa en 1 el valor de x y luego lo utiliza.

- Ejemplos:

```
a = 3;   b = a++;   /* post-inc. Valores  finales:  a = ?,  b = ? */
a = 3;   b = ++a;   /* pre-inc. Valores  finales: a = ?,   b = ? */
a = 3;   b = a--;   /* post-dec. Valores  finales:  a = ?,   b = ? */
a = 3;   b = --a;   /* pre-dec. Valores  finales:  a = ?,  b = ? */
```

- **Resta**/**Negativo** (**-**): con dos operandos efectúa la resta; delante de una variable o constante equivale a multiplicarla por (-1).

```
a = 3.2467;   b = -a; /* Valores  finales:  a = 3.2467,  b = -3.2467 */
```

LENGUAJE C: OPERADORES BITWISE (BIT-BIT)

- Actúan sobre los operandos a nivel bit.
- **AND**, **&**: pone un bit = 1 si en ambos operandos hay un 1 en esa posición.
- **OR**, **|**: pone un bit = 1 si en alguno de los operandos hay un 1 en esa posición.
- **OR EXCLUSIVA** (**XOR**), **^**: pone un bit = 1 si en esa posición existe un 1 en uno de los operandos pero no en otro (elimina la combinación 1 1).
- Complemento (**NOT**), **~**: cambia los 0 por 1 y los 1 por 0.
- **Left Shift**, **<<**: desplaza los bits del op1 op2 bits hacia la izquierda (multip. x 2^{op2}).
- **Right Shift**, **>>**: desplaza los bits del op1 op2 bits hacia la derecha (divide x 2^{op2}).
- Tablas de verdad:

b1	b2	b1&b2
0	0	0
0	1	0
1	0	0
1	1	1

b1	b2	b1|b2
0	0	0
0	1	1
1	0	1
1	1	1

b1	b2	b1^b2
0	0	0
0	1	1
1	0	1
1	1	0

LENGUAJE C: OPERADORES BITWISE (BIT-BIT)

- Ejemplo1:

```
char  a = 48;
```

0	0	1	1	0	0	0	0

a

```
char  b = 19;
```

0	0	0	1	0	0	1	1

b

```
char  and, or, xor, not, rs, ls;
```

```
and = a & b;
```

x	x	x	x	x	x	x	x

and = ?

```
or = a | b;
```

x	x	x	x	x	x	x	x

or = ?

```
xor = a ^ b;
```

x	x	x	x	x	x	x	x

xor = ?

```
not = ~ a;
```

x	x	x	x	x	x	x	x

not = ?

```
rs = a >> 2;
```

x	x	x	x	x	x	x	x

rs = ?

```
ls = b << 3;
```

x	x	x	x	x	x	x	x

ls = ?

- Ejemplo2:

```
const char LECTURA  = 1;          /* set b0 = 1 (LSB) */
const char ESCRITURA = 2;         /* set b1 = 1 */

char permisos = LECTURA | ESCRITURA;     /* permite leer y escribir */
permisos& = ~ESCRITURA;           /* elimina el permiso de escritura */
```

LENGUAJE C: PRECEDENCIA DE OPERADORES

- a + b > c | | c < 0 qué operación se realiza primero? Existe un orden dado por:

Operadores	Orden	Descripción
( ) [ ] -> .	→	funciones, matrices, punteros, estructuras
! ~ ++ -- (tipo) * & sizeof	←	unarios, contenido y dirección (punteros)
* / %	→	multiplicativos
+ -	→	aditivos
<< >>	→	desplazamiento bits
< <= > >=	→	relacionales
== !=	→	igualdad
&	→	and bitwise
^	→	xor (or exclusive) bitwise
|	→	or bitwise
&&	→	and lógico
||	→	or lógico
condición ? valor1 : valor2	←	ternario (equivale a if / else)
= += -= *= /= &= >>= . . .	←	asignación

- Conviene utilizar paréntesis para evitar confusiones. Siempre se evalúa primero la expresiones entre paréntesis. Si están anidados, los más internos primero.