

**Informática**  
*Ingeniería en Electrónica y Automática Industrial*  
**Operadores y expresiones en lenguaje C**

V1.3 © Autores

## Operadores y expresiones en lenguaje C

- Expresiones numéricas y operadores
- Operadores aritméticos
- Operadores lógicos y de relación
- Operadores de manejo de bits
- Operadores de asignación
- Otros operadores
  - Operador condicional
  - Operadores de concatenación
  - Operadores de dirección e indirección
  - Operador `sizeof`
- Prioridad y orden de evaluación
- Conversión de tipos
  - Conversión explícita

V1.3 © Autores

## Expresiones numéricas y operadores

- Una **expresión numérica** está formada por
  - Operadores
  - Operandos
- Un **operador** es un símbolo que indica cómo se procesan los operandos dentro de las expresiones
- Los **operandos** son los objetos del procesamiento: variables, constantes, etc.
- En lenguaje C cualquier expresión es *evaluable*
  - Se consideran *falsa* si se evalúa con cero
  - Se considera *cierta* si tiene un valor distinto de cero

V1.3 © Autores

## Operadores aritméticos

- Realizan operaciones aritméticas
- Si los operandos son de diferentes tipos, se transforma el menor precisión al tipo del mayor

OPERADOR	OPERACIÓN	OPERANDOS
+	Suma	Enteros o reales
-	Resta	Enteros o reales
*	Multiplicación	Enteros o reales
/	División	Enteros o reales.
%	Resto de la división entera ("módulo")	Sólo enteros
-	Cambio de signo	Sólo un operando (entero o real) a la derecha

V1.3 © Autores

## Operadores lógicos y de relación (I)

OPERADORES DE RELACIÓN	
OPERADOR	OPERACIÓN Y RESULTADO
<	El resultado es 1 si el operando a la izquierda es <b>menor</b> que el de la derecha; 0 en caso contrario
>	El resultado es 1 si el operando a la izquierda es <b>mayor</b> que el de la derecha; 0 en caso contrario
<=	El resultado es 1 si el operando a la izquierda es <b>menor o igual</b> que el de la derecha; 0 en caso contrario
>=	El resultado es 1 si el operando a la izquierda es <b>mayor o igual</b> que el de la derecha; 0 en caso contrario
!=	El resultado es 1 si los operandos son <b>distintos</b> y 0 en caso contrario
==	El resultado es 1 si los operandos son <b>iguales</b> y 0 en caso contrario

V1.3 © Autores

## Operadores lógicos y de relación (II)

- El resultado es siempre un valor de tipo `int` que sólo puede ser 1 (verdadero) o 0 (falso)
- Los operandos pueden ser de cualquier tipo, pero sólo son considerados como *verdadero* (1) o *falso* (0)

OPERADORES LÓGICOS	
OPERADOR	OPERACIÓN Y RESULTADO
&&	<b>AND.</b> El resultado es 1 si ambos operandos son distintos de 0. Si uno es 0, el resultado también lo es
	<b>OR.</b> El resultado es 1 si cualquiera de los operando vale 1. Solo si todos son 0 el resultado también lo es
!	<b>NOT.</b> El resultado es 1 si el operando es 0 y a la inversa

V1.3 © Autores

## Operadores de manejo de bits

- Operan con los bits de los operandos, que sólo pueden ser de tipo entero (`int` o `char`)

OP.	OPERACIÓN Y RESULTADO
&	<b>AND</b> entre los bits de los operandos a ambos lados del operador
	<b>OR</b> entre los bits de los operandos a ambos lados del operador
^	<b>OR-EXCLUSIVA (XOR)</b> entre los bits de los operandos a ambos lados del operador
~	<b>Complemento a 1</b> de los bits del operando a la derecha del operador
<<	<b>Desplazamiento a la izquierda</b> de los bits del operando de la izquierda tantas posiciones como indique el de la derecha (también de tipo entero). Por la derecha entran ceros.
>>	<b>Desplazamiento a la derecha</b> de los bits del operando de la izquierda tantas posiciones como indique el de la derecha (también de tipo entero). Por la izquierda entra el bit de signo o ceros si es <code>unsigned</code> .

V1.3

© Autores

7

## Operadores de asignación (I)

- En lenguaje C son varios los operadores de asignación
- Los operadores de asignación actualizan el valor de una única variable
  - Operador de asignación simple: `=`
    - Actualiza el valor de la variable de la izquierda con el valor de la expresión de la derecha
  - Operador de incremento `++` o decremento `--`
    - Incrementa/decrementa la variable sobre la que se aplica
    - Aplicados a una variable dentro de una expresión
      - `++variable`. Primero se incrementa y después se utiliza la variable incrementada
      - `variable++`. Primero se utiliza la variable y después se incrementa.

V1.3

© Autores

8

## Operadores de asignación (II)

- Hay operadores de *operación* y *asignación* cuya sintaxis es

```
variable (op) = expresion;
```

- `variable` es la variable a actualizar
- `(op) =` es el operador de asignación
- `expresion` es la expresión cuyo valor se operará con el valor de `variable` para obtener su nuevo valor

- La expresión equivalente es:

```
variable = variable (op) expresion;
```

V1.3

© Autores

9

## Operadores de asignación (III)

OP.	OPERACIÓN Y RESULTADO
* =	<b>Multiplicación y asignación.</b> Multiplica la variable de la izquierda por el valor de la derecha y asigna el nuevo valor a la variable
/ =	<b>División y asignación.</b> Divide la variable de la izquierda entre el valor de la derecha y asigna el nuevo valor a la variable
% =	<b>Resto y asignación.</b> Obtiene el resto de la división entera de la variable de la izquierda entre el valor de la derecha y asigna el nuevo valor a la variable
+ =	<b>Suma y asignación.</b> Suma la variable de la izquierda con el valor de la derecha y asigna el nuevo valor a la variable
- =	<b>Suma y asignación.</b> Resta la variable de la izquierda con el valor de la derecha y asigna el nuevo valor a la variable

*Nota:* La asignación es, en todos los casos, a la variable de la izquierda

V1.3

© Autores

10

## Operadores de asignación (IV)

OP.	OPERACIÓN (A nivel de bits) Y RESULTADO
<<=	<b>Desplazamiento a izquierdas y asignación.</b> Desplaza los bits de la variable de la izquierda tantas posiciones a la izquierda como indica el operando de la derecha y el resultado queda en la variable de la izquierda (por la derecha entran ceros).
>>=	<b>Desplazamiento a derechas y asignación.</b> Desplaza los bits de la variable de la izquierda tantas posiciones a la derecha como indica el operando de la derecha y el resultado queda en la variable de la izquierda (por la izquierda entra el bit de signo).
&=	<b>AND entre bits y asignación.</b> Realiza la operación AND entre los bits de la variable de la izquierda con los bits de la variable de la derecha, guardando el resultado en la primera.
=	<b>OR entre bits y asignación.</b> Realiza la operación OR entre los bits de la variable de la izquierda con los bits de la variable de la derecha, guardando el resultado en la primera.
^=	<b>OR-Exclusiva entre bits y asignación.</b> Realiza la operación OR Exclusiva entre los bits de la variable de la izquierda con los bits de la variable de la derecha, guardando el resultado en la primera.

V1.3

© Autores

11

## Otros operadores (I)

- Operador condicional «?:»**

```
expresion1 ? expresion2 : expresion3
```

  - Si `expresion1` es verdadera, se toma en consideración `expresion2`, si es falsa (0) se toma `expresion3`
  - Ejemplo:
 

```
(a >= b) ? puts("a>=b") : puts("b>a");
```
- Operador coma « , »**
  - Concatena expresiones, listas de variables, etc.
  - Actúa como separador en las listas de argumentos
  - Tiene el significado de la conjunción "y" en español

V1.3

© Autores

12

## Otros operadores (II)

- **Operador de dirección «&»**
  - Aplicado a un identificador (a su derecha) obtiene la dirección de memoria de la variable correspondiente
- **Operador de indirección «\*»**
  - Cuando *precede* a un identificador convierte al identificador en una dirección de memoria y el conjunto \*identificador hace referencia al dato contenido por la dirección `identificador`
- **Operador «sizeof»**
  - Aplicado a un operando nos devuelve el número de bytes que el operando ocupa en memoria

V1.3

© Autores

13

## Prioridad y orden de evaluación (I)

Orden	OPERADORES											ASOCIATIVIDAD	
1º	()	[]	.	->	sizeof								Izda. a derecha
2º	-	~	!	*	++	--	(tipo)						Derecha a Izda.
3º	*	/	%										Izda. a derecha
4º	+	-											Izda. a derecha
5º	<<	>>											Izda. a derecha
6º	<	<=	>	>=									Izda. a derecha
7º	==	!=											Izda. a derecha
8º	&												Izda. a derecha
9º	^												Izda. a derecha
10º													Izda. a derecha
11º	&&												Izda. a derecha
12º													Izda. a derecha
13º	?:												Derecha a Izda.
14º	=	*=	/=	%=	+=	-=	<<=	>>=	&=	=	^=		Derecha a Izda.
15º	,												Izda. a derecha

V1.3

© Autores

14

## Prioridad y orden de evaluación (II)

- **Prioridad y orden de evaluación (tabla)**
    - Los operadores de la misma línea tienen la misma prioridad
    - La prioridad es decreciente de arriba hacia abajo
    - Los paréntesis anidados se evalúan de dentro hacia fuera
    - No es posible conocer el orden de evaluación de las expresiones con ambigüedades
- ```
x = f() + g(); /* no sabemos qué función es llamada antes*/
a[i] = i++; /* Diferentes resultados según el compilador */
```

V1.3

© Autores

15

## Conversión de tipos (I)

- En las expresiones, los operandos cambian de tipo automáticamente
  - Si intervienen operandos reales, se unifican los tipos al de mayor precisión
  - Las constantes reales son tipo `double` por omisión
  - Los `char` y `short` se convierten a `int` si el `int` puede representar todos los valores del tipo original o a `unsigned int` en caso contrario
  - Si intervienen operando enteros, se unifican los tipos al de mayor longitud

V1.3

© Autores

16

## Conversión de tipos (II)

- **Ejemplo**

```
long a;
char b;
int c, f;
float d;
f = a + b * c / d;
```

  - `b` se convierte al tipo de `c` (`int`) y se realiza `b*c`. Se obtiene un `int`
  - El `int b*c` se convierte a `float` y se divide entre `d`. Se obtiene un `float`
  - `a` se convierte a `float` y se suma a `b*c/d`. Se obtiene un `float`
  - El `float` resultante de `a+b*c/d` se convierte a `int` (eliminando la parte fraccionaria) y se guarda en la variable entera `f`

V1.3

© Autores

17

## Conversión de tipos (III)

- **Conversión explícita: operador «(cast)»**
  - Consiste en convertir el tipo de dato de una variable o de una expresión
  - Sólo sirve para la evaluación de la expresión donde se realiza la conversión
  - Sintaxis
    - (`tiponuevo`) `expresion`;
    - `tiponuevo` es el tipo de dato al que se convertirá `expresion`
  - Ejemplo: La expresión `7/2` da como resultado 3, sin embargo la expresión `(float)7/2` convierte el 7 en real y el resultado será un número real: 3.5

V1.3

© Autores

18