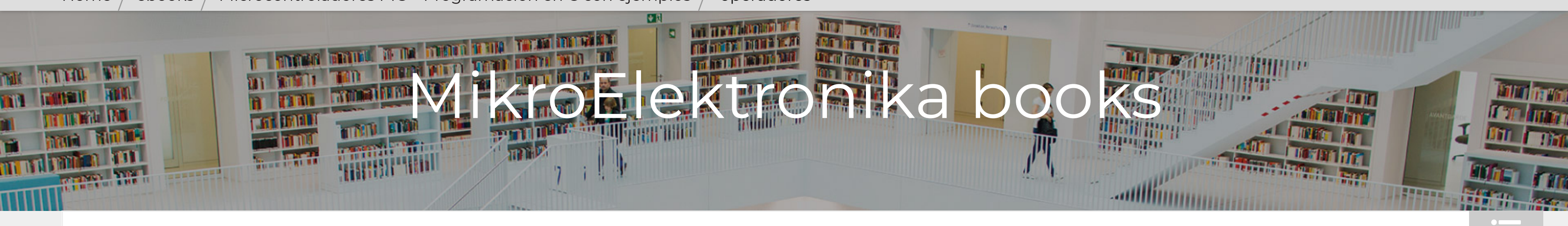


Book: Microcontroladores PIC – Programación en C con ejemplos

Table of Contents

2.5 Operadores

Un operador es un símbolo que denota una operación aritmética, lógica u otra operación particular. Dicho de manera sencilla, varias operaciones aritméticas y lógicas se realizan por medio de los operadores. Hay más de 40 operaciones disponibles en el lenguaje C, pero se utiliza un máximo de 10-15 de ellas en práctica. Cada operación se realiza sobre uno o más operandos que pueden ser variables o constantes. Además, cada operación se caracteriza por la prioridad de ejecución y por la asociatividad.

OPERADORES ARITMÉTICOS

Los operadores aritméticos se utilizan en las operaciones aritméticas y siempre devuelven resultados numéricos. Hay dos tipos de operadores, los unitarios y los binarios. A diferencia de las operaciones unitarias que se realizan sobre un operando, las operaciones binarias se realizan sobre dos operandos. En otras palabras, se requieren dos números para ejecutar una operación binaria. Por ejemplo:

OPERADOR	OPERACIÓN
+	Adición
-	Resta
*	Multiplicación
/	División
%	Resto de la división

a+b o a/b.

```
int a,b,c; // Declarar 3 enteros a, b, c
a = 5;     // Inicializar a
b = 4;     // Inicializar b
c = a + b; // c = 9
c = c%2;   // c = 1. Esta operación se utiliza con frecuencia
           // para comprobar la paridad. En este caso, el
           // resultado es 1 lo que significa que la variable
           // es un número imparo
```

OPERADORES DE ASIGNACIÓN

Hay dos tipos de asignación en el lenguaje C:

- Los operadores simples asignan los valores a las variables utilizando el carácter común '='. Por ejemplo: a =8
- Las asignaciones compuestas son específicas para el lenguaje C. Consisten en dos caracteres como se muestra en la tabla a la derecha. Se utilizan para simplificar la sintaxis y habilitar la ejecución más rápida.

OPERADOR	EJEMPLO	
	Expresión	Equivalente
+=	a += 8	a = a + 8
-=	a -= 8	a = a - 8
*=	a *= 8	a = a * 8
/=	a /= 8	a = a / 8
%=	a %= 8	a = a % 8

```
int a = 5; // Declarar e inicializar la variable a
a += 10;   // a = a + 10 = 15
```

OPERADORES DE INCREMENTO Y DECREMENTO

Las operaciones de incremento y decremento por 1 se denotan con "++" y "--". Estos caracteres pueden preceder o seguir a una variable. En primer caso (++x), la variable x será incrementada por 1 antes de ser utilizada en la expresión. De lo contrario, la variable se utilizará en la expresión antes de ser aumentada por 1. Lo mismo se aplica a la operación de decremento.

OPERADOR	EJEMPLO	DESCRIPCIÓN
++	++a	Variable "a" es incrementada por 1
	a++	
--	--b	Variable "a" es decrementada por 1
	b--	

```
int a, b, c;
a = b = 5;
c = 1 + a++; // c = 6
b = ++c + a  // b = 7 + 6 = 13
```

OPERADORES RELACIONALES

Los operadores relacionales se utilizan en comparaciones con el propósito de comparar dos valores. En mikroC, si una expresión es evaluada como falsa (false), el operador devuelve 0, mientras que si una oración es evaluada como verdadera (true), devuelve 1. Esto se utiliza en expresiones tales como 'si la expresión es evaluada como verdadera, entonces...'

OPERADOR	DESCRIPCIÓN	EJEMPLO	CONDICIÓN DE VERACIDAD
>	mayor que	b > a	si **b** es mayor que **a**
>=	mayor o igual que	a >= 5	si **a** es mayor o igual que **5**
<	menor que	a < b	si **a** es menor que **b**
<=	menor o igual que	a <= b	si **a** es menor o igual que **b**
==	igual que	a == 6	si **a** es igual que **6**
!=	desigual que	a != b	si **a** es desigual que **b**

```
int prop;
int var = 5;
prop = var < 10; // Expresión es evaluada como verdadera, prop = 1
```

OPERADORES LÓGICOS

Hay tres tipos de operaciones lógicas en el lenguaje C: Y (AND) lógico, O (OR) lógico y negación - NO (NOT) lógico. Los operadores lógicos devuelven verdadero (1 lógico) si la expresión evaluada es distinta de cero. En caso contrario, devuelve falso (0 lógico) si la expresión evaluada equivale a cero. Esto es muy importante porque las operaciones lógicas se realizan generalmente sobre las expresiones, y no sobre las variables (números) particulares en el programa. Por lo tanto, las operaciones lógicas se refieren a la veracidad de toda la expresión. Por ejemplo: `1 && 0` es igual a `(expresión verdadera) && (expresión falsa)` El resultado 0, o sea - Falso en ambos casos.

OPERADOR	FUNCIÓN
&&	Y
||	O
!	NO

OPERADORES DE MANEJO DE BITS

A diferencia de las operaciones lógicas que se realizan sobre los valores o expresiones, las operaciones de manejo de bits se realizan sobre los bits de un operando. Se enumeran en la siguiente tabla:

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO	
`~`	Complemento a uno	a = ~b	b = 5	a = -5
`<<`	Desplazamiento a la izquierda	a = b << 2	b = 11110011	a = 11001100
`>>`	Desplazamiento a la derecha	a = b >> 2	b = 11110011	a = 00011110
`&`	Y lógico para manejo de bits	c = a & b	a = 11100011 b = 11001100	c = 11000000
`|`	O lógico para manejo de bits	c = a | b	a = 11100011 b = 11001100	c = 11101111
`^`	EXOR lógico para manejo de bits	c = a ^ b	a = 11100011 b = 11001100	c = 00101111

Note que el resultado de la operación de desplazamiento a la derecha depende del signo de la variable. En caso de que el operando se aplique a una variable sin signo o positiva, se introducirán los ceros en el espacio vacío creado por desplazamiento. Si se aplica a un entero con signo negativo, se introducirá un 1 para mantener el signo correcto de la variable.

¿CÓMO UTILIZAR LOS OPERADORES?

- Aparte de los operadores de asignación, dos operadores no deben estar escritos uno junto al otro.

  ```
  x*%12; // esta expresión generará un error
  ```

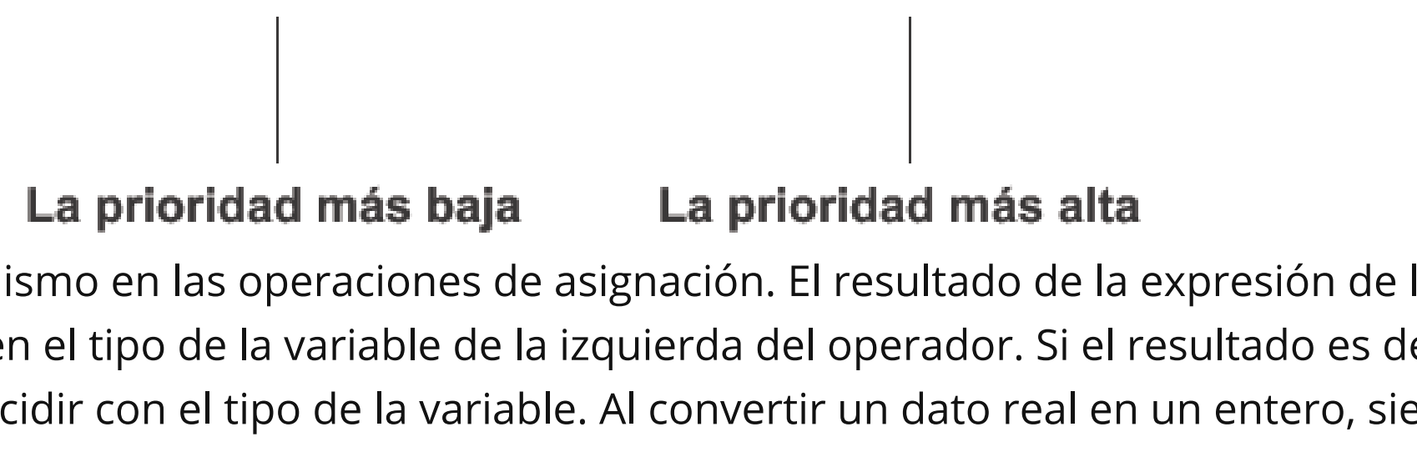
- Cada operador tiene su prioridad y asociatividad como se muestra en la tabla:
- Similar a las expresiones aritméticas, los operadores se agrupan juntos por medio de paréntesis. Primero se calculan las expresiones encerradas entre paréntesis. Si es necesario, se pueden utilizar los paréntesis múltiples (anidados).

PRIORIDAD	OPERADORES	ASOCIATIVIDAD
Alta	`() [] -> .`	de izquierda a derecha
	`! ~ ++ -- +(unitario) -(unitario) *Puntero &Puntero`	de derecha a izquierda
	`* / %`	de izquierda a derecha
	`+ -`	de izquierda a derecha
	`< >`	de izquierda a derecha
	`< <= > >=`	de izquierda a derecha
	`== !=`	de izquierda a derecha
	`&`	de izquierda a derecha
	`^`	de izquierda a derecha
	`|`	de izquierda a derecha
	`&&`	de izquierda a derecha
	`||`	de derecha a izquierda
	`?:`	de derecha a izquierda
Baja	`= += -= *= /= /= &= ^= |= <= >=`	de izquierda a derecha

```
int a, b, res;
a = 10;
b = 100;
res = a*(a + b); // resultado = 1100
res = a*a + b;   // resultado = 200
```

CONVERSIÓN DE TIPOS DE DATOS

Algunas operaciones implican conversión de datos. Por ejemplo, si divide dos valores enteros, hay una alta posibilidad de que el resultado no sea un entero. El mikroC realiza una conversión automática cuando se requiera. Si dos operandos de tipo diferente se utilizan en una operación aritmética, el tipo de operando de la prioridad más baja se convierte automáticamente en el tipo de operando de la prioridad más alta. Los tipos de datos principales se colocan según el siguiente orden jerárquico:



La autoconversión se realiza asimismo en las operaciones de asignación. El resultado de la expresión de la derecha del operador de la asignación siempre se convierte en el tipo de la variable de la izquierda del operador. Si el resultado es de tipo de la prioridad más alta, se descarta o se redondea para coincidir con el tipo de la variable. Al convertir un dato real en un entero, siempre se descartan los números que siguen al punto decimal.

```
int x;     // A la variable x se le asigna el tipo integer (un entero)
x = 3;     // A la variable x se le asigna el valor 3
x+ = 3.14; // El valor 3.14 se agrega a la variable x al
           // realizar la operación de asignación

/* El resultado de la adición es 6 en vez de 6.14, como era de esperar.
Para obtener el resultado esperado sin descartar los números que siguen al
punto decimal, se debe declarar x como un punto flotante. */
```

Para realizar una conversión explícita, antes de escribir una expresión o una variable hay que especificar el tipo de resultado de operación entre paréntesis.

```
double distancia, tiempo, velocidad;
distancia = 0.89;
tiempo = 0.1;
velocidad = (int)(a/b); // c = (int)8.9 = 8.0
velocidad = ((int)a)/b; // c = 0/0.1 = 0.0
```