

Laboratorio de Arquitectura de Redes

Operadores y expresiones en
lenguaje C

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

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

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	Sólo enteros
-	Cambio de signo	Sólo un operando (entero o real) a la derecha

Operadores lógicos y de relación (I)

- ❑ 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
&&	AND. El resultado es 1 si ambos operandos son distintos de 0. Si uno es 0, el resultado también lo es
	OR. El resultado es 1 si cualquiera de los operando vale 1. Sólo si todos son 0 el resultado también lo es
!	NOT. El resultado es 1 si el operando es 0 y a la inversa

Operadores lógicos y de relación (I)

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

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
<code>&</code>	AND entre los bits de los operandos a ambos lados del operador
<code> </code>	OR entre los bits de los operandos a ambos lados del operador
<code>^</code>	OR-EXCLUSIVA (XOR) entre los bits de los operandos a ambos lados del operador
<code>~</code>	Complemento a 1 de los bits del operando a la derecha del operador
<code><<</code>	Desplazamiento a la izquierda 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.
<code>>></code>	Desplazamiento a la derecha 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 unsigned.

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.

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;`

Operadores de asignación (III)

OP.	OPERACIÓN Y RESULTADO
$* =$	Multiplicación y asignación. Multiplica la variable de la izquierda por el valor de la derecha y asigna el nuevo valor a la variable
$/ =$	División y asignación. Divide la variable de la izquierda entre el valor de la derecha y asigna el nuevo valor a la variable
$\% =$	Resto y asignación. 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
$+ =$	Suma y asignación. Suma la variable de la izquierda con el valor de la derecha y asigna el nuevo valor a la variable
$- =$	Suma y asignación. Suma la variable de la izquierda con el valor de la derecha y asigna el nuevo valor a la variable

Operadores de asignación (IV)

OP.	OPERACIÓN (A nivel de bits) Y RESULTADO
$\ll =$	Desplazamiento a izquierdas y asignación. Desplaza los bits de la variable de la izquierda tantas posiciones a la <i>izquierda</i> como indica el operando de la derecha y el resultado queda en la variable de la izquierda (por la derecha entran ceros).
$\gg =$	Desplazamiento a derechas y asignación. Desplaza los bits de la variable de la izquierda tantas posiciones a la <i>derecha</i> 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).
$\& =$	AND entre bits y asignación. 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.
$ =$	OR entre bits y asignación. 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.
$\wedge =$	OR-Exclusiva entre bits y asignación. 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.

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

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` hacer 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

Prioridad y orden de evaluación (I)

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

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 */
```

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

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

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`