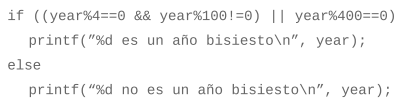
**Capítulos:** 2.5 a 2.12

### 2.5 Operadores Aritméticos

Los operadores aritméticos son **+ - \* / %**.

La división entera trunca cualquier parte fraccionaria.

La expresión **x % y** produce el resto cuando **x** es dividido por **y**. Por ejemplo, cálculo de años bisiestos:



Un año es bisiesto si es divisible entre 4 pero no entre 100, excepto aquellos años que son divisibles entre 400, que sí son bisiestos.

El operador **%** no puede aplicarse a operandos float o double.

Los operadores aritméticos se asocian de izquierda a derecha.

### 2.6 Operadores de Relación y Lógicos

Los operadores de **relación** son **> >= < <=**. Todos tienen la misma precedencia.

Los operadores de **igualdad** son **== !=**.

Los operadores de relación tienen precedencia inferior que los operadores aritméticos.

Los operadores **lógicos** son **&&** y **||**. Son evaluados de izquierda a derecha, y la evaluación se detiene tan pronto como se conoce el resultado verdadero o falso.

La precedencia de **&&** es más alta que la de **||**, y ambas son menores que los operadores de relación y de asignación. Expresiones como la siguiente no requieren paréntesis adicionales



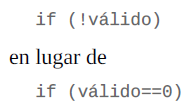
Pero, puesto que la precedencia de **!=** es superior que la asignación, los paréntesis se necesitan en



para así, primero asignar el valor obtenido de getchar a la variable y luego compararla con ‘\n’.

Por definición, el valor numérico de una expresión de relación o lógica es **1** si la relación es **verdadera** o **0** si la relación es **falsa**.

El operador unario de negación **!** convierte a un operando que no es cero en 0, y a un operando cero en 1.

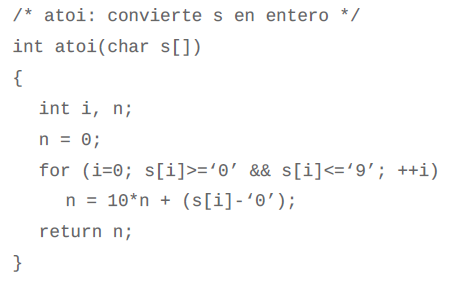


### 2.7 Conversiones de Tipo

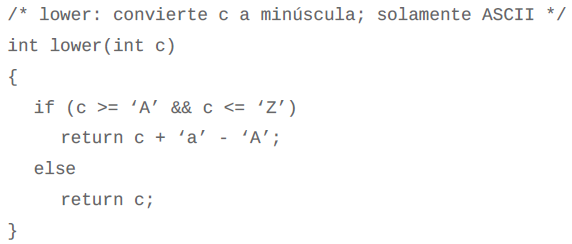
Las operaciones que involucran tipos de datos diferentes se convierten a un tipo en común de acuerdo a ciertas reglas. Las únicas conversiones automáticas son las que convierten un operando de un tipo “menor” a uno “mayor” sin pérdida de información.

Las expresiones que podrían provocar pérdida de información, como pasar de un tipo de dato “mayor” a uno “menor”, (float a int), pueden producir advertencias, pero no son ilegales.

Función **atoi**, que convierte una cadena de dígitos en su equivalente decimal



Función **lower**, que convierte un carácter a minúscula para el conjunto de caracteres

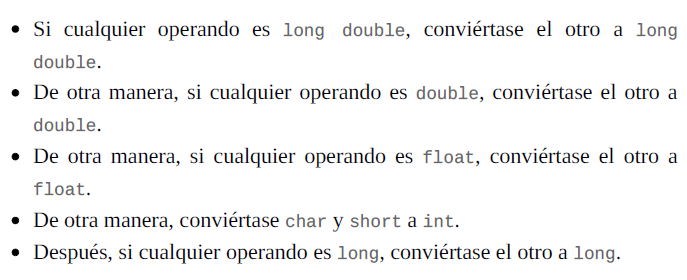


Esto funciona para ASCII debido a que las correspondientes letras mayúsculas y minúsculas están a una distancia fija como valores numéricos y cada alfabeto es contiguo.

Este tipo de funciones se encuentran en **<ctype.h>**.

El lenguaje no especifica si las variables tipo char son con signo o sin signo. La definición de C garantiza que ningún carácter esté en el conjunto estándar de caracteres de impresión de la máquina será negativo. Por transportabilidad se debe especificar signed o unsigned si se van a almacenar datos que no sean caracteres en variables tipo char.

**Reglas de conversión de tipos:**



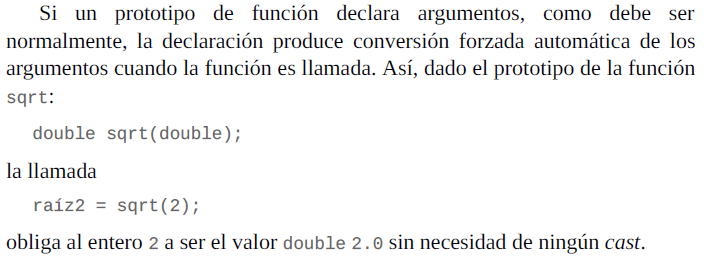
Las conversiones también se realizan en las asignaciones; el valor del lado derecho es convertido al tipo de la izquierda, el cual es el tipo del resultado.

Puesto que un argumento de la llamada a una función es una expresión, también suceden conversiones de tipo cuando se pasan argumentos a funciones.

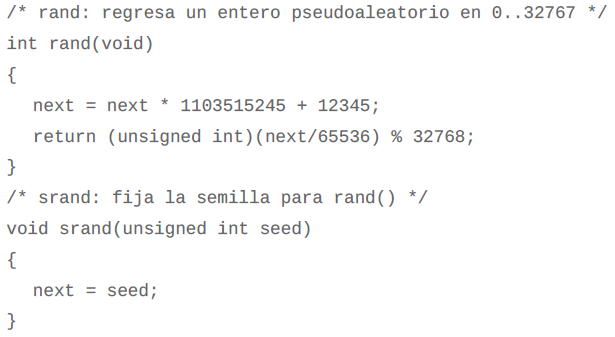
La conversión explícita de tipo puede ser forzada en cualquier expresión, con un operador unario llamado **cast**.



La expresión es convertida al tipo nombrado, según las reglas anteriores. Cast funciona como si la expresión fuera asignada a una variable del tipo especificado, que se utiliza entonces en lugar de la construcción completa.







### 2.8 Operadores de Incremento y Decremento

**++** agrega 1 a su operando.

**--** resta 1 a su operando.

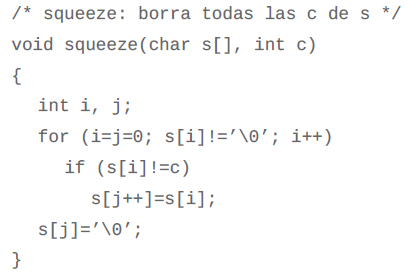
Pueden ser utilizados como prefijo o como postfijo. En ambos casos, el efecto es incrementar n.

La expresión **++n** incrementa a n antes de que su valor se utilice.

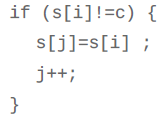
Mientras que, **n++** incrementa a n después de que su valor se ha empleado.

Los operandos de incremento y decremento sólo pueden aplicarse a variables. Una expresión como **(i+j)++** es **ilegal**.

Considérese la siguiente función **squeeze(s, c)**, que elimina todas las ocurrencias del carácter c de una cadena s

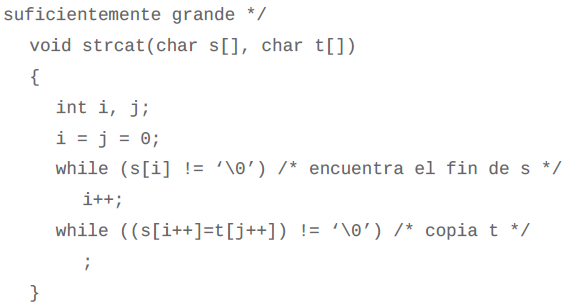


Cada vez que se encuentra un valor diferente de c, éste se copia en la posición actual de j, y sólo entonces j es incrementada para prepararla para el siguiente carácter. Esto es equivalente a



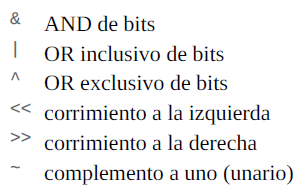
Otro ejemplo, es la función **strcat(s, t)**, que concatena la cadena t al final de la cadena s.





### 2.9 Operadores para manejo de bits

Los operadores lógicos sólo pueden ser aplicados a operandos de tipo **char, short, int** y **long,** con o sin signo.



El operador AND de bits **&** a menudo es usado para enmascarar algún conjunto de bits.

El operador OR de bits **|** es empleado para encender bits.

El operador OR exclusivo **^** pone un uno en cada posición en donde sus operandos tienen bits diferentes, y cero en donde son iguales.

Los operadores de corrimiento (o decalaje) **<<** y **>>** realizan corrimientos a la izquierda y a la derecha de su operando que está a la izquierda, el número de posiciones de bits dado por el operando de la derecha, el cual debe ser positivo. El corrimiento se realiza de acuerdo al sistema en el que se trabaja, por ende, no produce cambios de signo erróneos.

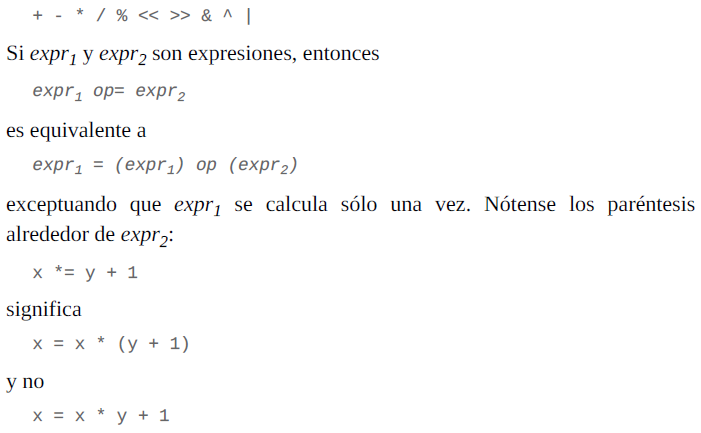
Corrimiento a izquierda **<<** equivale a multiplicar el número por potencias de 2. (2^n siendo n el operando derecho).

Corrimiento a derecha **>>** equivale a dividir el número por potencias de 2. (2^n siendo n el operando derecho).

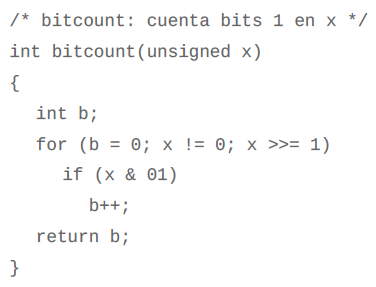
El operador unario **~** da el complemento a uno de un entero; esto es, convierte cada bit 1 en un bit 0 y viceversa.

### 2.10 Operadores de asignación y expresiones

**Operador de asignación**



Como ejemplo, la función **bitcount** cuenta el número de bits en 1 en su argumento entero.



### 2.11 Expresiones condicionales

La expresión condicional, escrita con el operador ternario **?**, proporciona una forma alternativa para escribir construcciones de decisión.

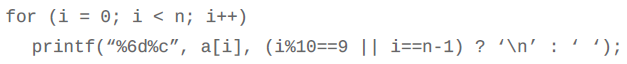


La expresión *expr1* es evaluada primero. Si es diferente de cero (verdadero), entonces la expresión *expr2*es evaluada, y ése es el valor de la expresión condicional. De otra forma, *expr3* es evaluada, y ése es el valor. Sólo uno de entre *expr2* y *expr3* se evalúa.



Es recomendable colocar paréntesis, para facilitar la lectura.

El siguiente ciclo imprime n elementos de un arreglo, 10 por línea, con cada columna separada por un blanco, y con cada línea terminada con una nueva línea.



----



### 2.12 Precedencia y orden de evaluación

