Palabras Reservadas del Lenguaje C

|  |  |
| --- | --- |
| Palabra | Para que sirve |
| Auto | Es un especificador de almacenamiento de clases para variables temporales. Estas variables se crean al introducirse en un bloque de sentencias y se destruyen al salir de él. Las variables locales de una función tienen clase de almacenamiento auto por defecto. |
| Break | break permite salir del bucle do, while o for más interno. También se puede utilizar para salir de una sentencia switch. |
| Case | Sirve para etiquetar los diferentes casos de la sentencia switch. |
| Char | Tipo de dato incorporado en C++. Se puede utilizar char para declarar variables carácter (tienen un rango de 0 a 255 en código ASCII). |
| Const | Es un calificador de tipo, utilizado para indicar que la variable que le sigue no puede ser modificada por el programa. Esto significa que no se puede asignar un valor a esa variable, incrementarla o decrementarla; sin embargo, se puede inicializar a un valor dado, cuando se declara |
| Continue | Se utiliza continue en la ejecución de un bucle. Es equivalente a ejecutar una sentencia goto al final del bucle. Esta sentencia afecta al bucle más interno en el que aparece. |
| Default | Se utiliza en la sentencia switch para marcar el código que se ejecutará cuando ninguna de las etiquetas de case se corresponde con la expresión switch. |
| Do | Se utiliza un while para construir bucles iterativos en los cuales las instrucciones del cuerpo del bucle se ejecutan hasta que la condición se evalúa a 0 (falso). |
| Double | Especificador de tipo de dato double que declara variables y arrays de coma flotante de doble precisión. |
| Else | Se utiliza con if para controlar el flujo de ejecución con sentencias if |
| Enum | Tipo de datos de valor entero que puede tomar sus valores de una lista de constantes enumerados. |
| Extern | Especificador de clase de almacenamiento utilizado para indicar al compilador que una variable se declara en otra parte del programa. Si una declaración de variable comienza con la palabra reservada extern no es una definición, es decir, especifica el tipo y nombre de la variable e implica que una definición de esta variable se encuentra en otra parte del programa. Sin esta palabra reservada, es una definición; cada definición de variable es al mismo tiempo declaración de variable.  Este especificador se suele utilizar (en archivos compilados) separadamente, que comparten los mismos datos globales y se enlazan juntos. Así, por ejemplo, si test se declara en otro archivo, como entero, la declaración siguiente se utilizará en otros archivos:  extern int test;  No utilice extern con frecuencia. Existen otros métodos para pasar información entre funciones. En general, utilice extern sólo para referirse a variables globales que se definen en otra parte. Debido a que esta característica crea espacio permanente para las variables, la mayoría de las variables locales no son extern. |
| Float | float es un especificador de tipos de datos utilizados para declarar variables de coma flotante. |
| For | El bucle for permite inicializar e incrementar variables contadoras.  Si el bloque de sentencias sólo contiene una, no son necesarias las llaves ({,}). Si la condición es falsa, al comenzar el bucle éste no se ejecutará ni una sola vez. Se puede omitir cualquiera de las tres expresiones de for, pero deben dejarse los puntos y comas (;). Si se omite condición, ésta se considerará como verdadera. El bucle infinito es for (;;)y equivale a while(1). Se puede utilizar el operador coma (,) para poner varias expresiones en el interior de las diferentes partes de for. |
| Goto | Produce un salto en la ejecución del programa a una etiqueta de la función actual. Su uso no está recomendado más que para situaciones excepcionales, tales como la salida directa del bucle más interior perteneciente a una serie de bucles anidados. |
| If | La palabra reservada if se utiliza para ejecutar código sólo bajo ciertas condiciones. Se puede utilizar if sola o con else. Si las sentencias sólo son una, no se necesitan las llaves. |
| Int | Especificador de tipo para variables y arrays de enteros. Los cualificadores short y long se pueden utilizar para declarar un entero del tamaño deseado. |
| Long | Especificador de tipo de datos para declarar variables enteras, que ocupa dos veces más bytes que los enteros de tipo short. |
| Register | register es un especificador de almacenamiento para tipos de datos enteros, utilizados para informar al compilador de que el acceso a los datos debe ser tan rápido como sea posible. El compilador almacenará los datos enteros en un registro de la CPU, en lugar de situarlos en memoria. |
| Return | La sentencia return se utiliza para detener la ejecución de la función actual y devolver el control al llamador. Si la función devuelve un valor utiliza una sentencia  return expresion;  para devolver el valor representado por la expresión. |
| Short | short es un calificador de tamaño para variables enteras con y sin signo; al menos ocupa dos bytes, short significa en realidad signed short int. |
| Signed | El modificador de tipo signed se utiliza para indicar que los datos almacenados en un tipo entero (int o char tienen signo). Los valores int, long y short, por defecto, tienen signo. |
| Sizeof | El operador sizeof determina el número de bytes que se utilizan para almacenar una variable particular o tipo de datos. Así, cuando se aplica a una variable, sizeof devuelve el tamaño del objeto referenciado, y cuando se aplica a una clase, sizeof devuelve el tamaño total de un objeto de esa clase. |
| Static | static es un modificador de tipo de datos que instruye al compilador para crear almacenamiento permanente para la variable local que le precede. Si se declara una variable de tipo static, tendrá almacenamiento permanente y retiene su valor a lo largo de toda la vida del programa (y, por consiguiente, entre llamadas a funciones). static se utiliza también para ocultar datos y funciones de otros módulos o archivos. |
| Struct | La palabra reservada struct se utiliza para representar el tipo de datos estructura que reagrupa variables y/o funciones. En C++ una estructura tiene sintaxis idéntica a una clase, con la única diferencia de que en el acceso por defecto los miembros de una estructura tienen acceso público; mientras que el acceso por defecto de una clase es privado. |
| Switch | La sentencia switch se utiliza para realizar una bifurcación múltiple, dependiendo del valor de una expresión.  Se debe utilizar break para separar el código de una etiqueta de case de otra. Una etiqueta default marca el código que se ejecutará si ninguna de las etiquetas de case se corresponde con el valor de la expresión. |
| Typedef | typedef se utiliza para dar un nuevo nombre a un tipo de dato existente. Esta característica mejora la legibilidad de un programa. |
| Unión | Tipo de dato que se utiliza para asignar almacenamiento de diferentes elementos de datos en la misma posición. La declaración de union es la misma que la de struct, con la diferencia de que en una unión todos los elementos de datos de la declaración comparten la misma posición de almacenamiento. |
| Unsigned | El calificador de tipos unsigned se utiliza para tipos de datos enteros (char, int, short, int y long int) que informa al compilador que la variable se utilizará para almacenar sólo valores no negativos. Este calificador doble el máximo valor que se pueda almacenar en esa variable. |
| Void | void es un tipo de dato que se utiliza para indicar la no existencia de un valor de retorno o argumentos en una declaración y definición de una función. Se puede utilizar también void\* para declarar un puntero a cualquier tipo de objeto dado. |
| Volatile | El calificador de tipo volatile se utiliza para indicar que la variable que sigue a volatile se puede modificar por factores externos al control del programa. Por ejemplo, las variables que se cambian por hardware, tales como reloj de tiempo real, interrupciones u otras entradas, se declararán volatile. |
| While | La sentencia while permite construir un bucle cuyas sentencias interiores se ejecutan hasta que una condición o expresión se hace falsa (cero). |

Operadores

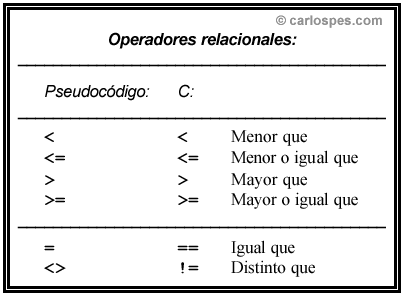
**Aritméticos:**

Se llaman operadores aritméticos a aquellos que permiten realizar cálculos con valores numéricos para obtener un resultado. Los operadores aritméticos más habituales son la suma, resta, multiplicación y división.

|  |  |
| --- | --- |
| **Operador** | **Significado** |
| Operador = | Asignación |
| Operador \* | Multiplicación |
| Operador / | División |
| Operador % | Resto de división entera (mod) |
| Operador + | Suma |
| Operador - | Resta |

**Relacionales:**

Un operador relacional se utiliza para comparar los valores de dos expresiones.



**Lógicos:**

Los operadores lógicos básicos en C son los siguientes:

|  |  |
| --- | --- |
| **Operador** | **Significado** |
| Operador && | Operador lógico and |
| Operador || | Operador lógico or |
| Operador ! | Operador lógico not |

Suponiendo que tenemos cuatro variables A, B, C y D cuyos valores se han establecido en A = 5, B = 7, C = 2, D = 5 podríamos evaluar estas expresiones:

|  |  |  |  |
| --- | --- | --- | --- |
| **Expresión** | **Pregunta equivalente** | **Resultado** | **Ejemplo código** |
| (A == B) && (A < B) | ¿Es A igual a B y A menor que C? | 0 (falso) | printf ("Pregunta (A == B) && (A < B) vale %d\n", (A == B) && (A < B)); |
| (A == 5) || (A > 7) | ¿Es A igual a 5 ó es A mayor que 7? | 1 (verdadero) | printf ("Pregunta (A == 5) || (A > 7) vale %d\n", (A == 5) || (A > 7)); |
| ! ( A == 5 ) | ¿A es NO igual a 5? | 0 (falso) | printf ("Pregunta ! ( A == 5 ) vale %d\n", ! ( A == 5 )); |

**Operador coma “,”**

El operador de comas tiene asociatividad de izquierda a derecha. Dos expresiones separadas por una coma se evalúan de izquierda a derecha. El operando izquierdo se evalúa siempre, y todos los efectos secundarios se completan antes de que se evalúe el operando derecho.

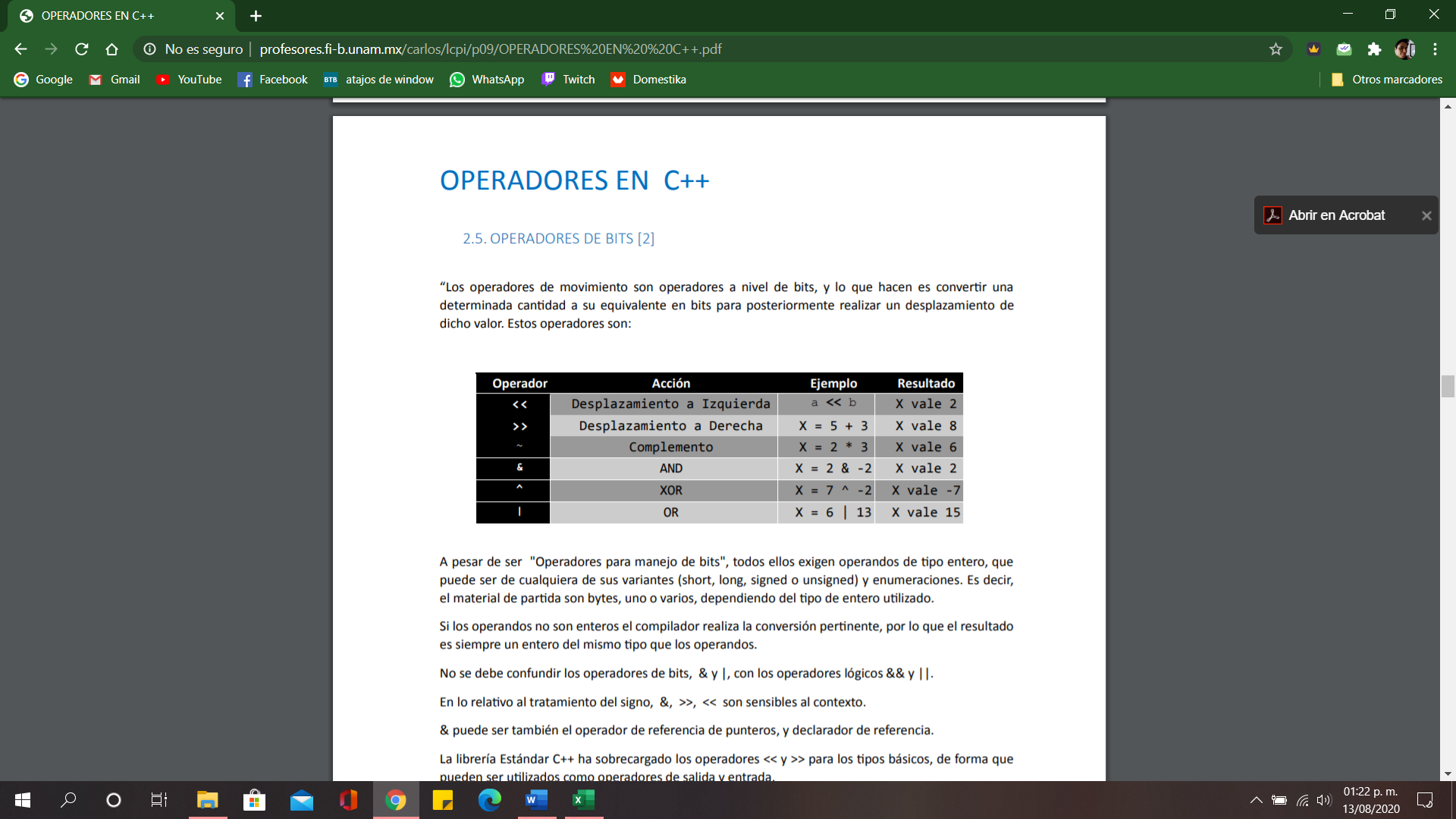
Las comas se pueden utilizar como separadores en algunos contextos, como las listas de argumentos de función. No confunda el uso de la coma como separador y como operador; los dos usos son completamente diferentes.

**Bit a Bit:**

En el lenguaje de programación C, operaciones se pueden realizar en un nivel de bits utilizando operadores bit a bit.

Operaciones a nivel de bit se contrastan por a nivel de byte operaciones que caracterizan homólogos de los operadores bit a bit lógicos, el AND, OR y NOT. En lugar de realizar en bits individuales, los operadores a nivel de byte se realizan en cadenas de ocho bits (conocidos como bytes) a la vez. La razón de esto es que un byte es normalmente la unidad más pequeña de memoria direccionable (es decir, los datos con una única dirección de memoria.)

|  |  |
| --- | --- |
| **Símbolo** | **Operador** |
| & | AND bit a bit |
| | | bit a bit incluido OR |
| ^ | bit a bit XOR (OR exclusivo) |
| << | Shift izquierdo |
| >> | Giro a la derecha |
| ~ | NO bit a bit (complemento a uno) (unario) |



A pesar de ser "Operadores para manejo de bits", todos ellos exigen operandos de tipo entero, que puede ser de cualquiera de sus variantes (short, long, signed o unsigned) y enumeraciones. Es decir, el material de partida son bytes, uno o varios, dependiendo del tipo de entero utilizado.

Si los operandos no son enteros el compilador realiza la conversión pertinente, por lo que el resultado es siempre un entero del mismo tipo que los operandos. No se debe confundir los operadores de bits, & y |, con los operadores lógicos && y ||.

En lo relativo al tratamiento del signo, &, >>, << son sensibles al contexto. & puede ser también el operador de referencia de punteros, y declarador de referencia.

**Operador punto y flecha**

En C y C++, para acceder a los [campos](https://es.wikipedia.org/wiki/Campo_(inform%C3%A1tica)) de un objeto apuntado por un puntero, se puede desreferenciar dicho puntero y usar el operador punto . o bien usar el operador flecha ->:

struct objeto

{

int o;

};

objeto o;

objeto \*po = &o;

(\*po).o = 1; // Desreferenciar y acceder al campo ‘o’.

po->o = 2; // Acceso al campo ‘o’ con el operador flecha.

El operador unario de indirección o el operador flecha tan sólo pueden usarse sobre punteros; estos operadores pueden encadenarse, por ejemplo en una [lista enlazada](https://es.wikipedia.org/wiki/Lista_enlazada) puede accederse al nodo posterior al siguiente nodo->siguiente->siguiente (siempre y cuando nodo->siguiente apunte a una dirección válida).

**Operador Ternario**

El operador condicional ?: tiene tres operandos, lo que significa que es un operador ternario.

La forma general de este operador es:

expresion1? expresion2: expresion3

Este pseudo-código lo representa: **condition? value\_if\_true : value\_if\_false** . Cada valor puede ser el resultado de una expresión evaluada.

Se evalúa expresion1, si el resultado de la evaluación es verdadero (diferente de cero), se ejecuta expresion2; si el resultado de la evaluación de expresion1 es falso (igual a cero), se ejecuta expresion3.

**Operador indirección**

En C la indirección se denota con el operador “\*” seguido del nombre de una variable de tipo puntero. Su significado es “accede al contenido al que apunta el puntero”. Desafortunadamente, este operador coincide con el utilizado para denotar los tipos de datos punteros y para declarar este tipo de variables.

Es muy importante tener una distinción muy clara entre estos dos usos del operador. Al traducir un programa, el compilador no comprueba ni que el puntero contiene una dirección de memoria correcta, ni que en esa dirección de memoria hay un dato correcto. Esta característica hace que el diseño de programas con punteros sea complejo.

Ejemplo del uso del operador indireccion:

