**CONCEPTOS BÁSICOS DE LENGUAJE C**

**BIENVENIDO, hoy aprenderás:**

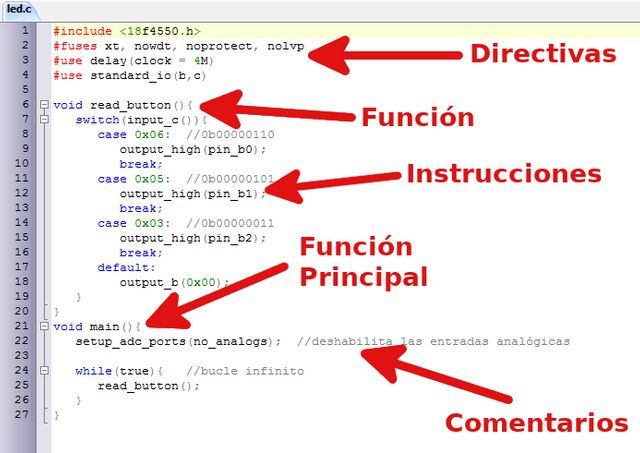
* Que es un Compilador de C y para que sirve.
* Partes de un programa escrito en lenguaje C.
* Conceptos básicos de lenguaje C como:
  + Tipos de datos.
  + Variables y Constantes.
  + Empleo de Operadores control conocidos.
  + Declaraciones de control.
  + Creación y empleo de Funciones de usuario.

Para programar microcontroladores PIC en lenguaje C, es necesario utilizar un compilador de C. Este compilador cumple la función de "traducir" el código lenguaje C del archivo fuente (.c) a lenguaje máquina, generando un archivo en formato hexadecimal (.hex) necesario para programar microcontroladores.  
El compilador de C que vamos a utilizar es el PCW PIC C Compiler de CCS (Custom Computer Services Incorporated) que ofrece una amplia librería de funciones predefinidas, comandos de pre-procesado y ejemplos. Además, suministra drivers para controlar dispositivos LCD, convertidores A/D, memoria EEPROM, relojes en tiempo real, etc.

Si quieres tener más información respecto a este compilador, ve a la web oficial de CCS haciendo clic [aquí](http://www.ccsinfo.com/).

**Estructura de un programa**

Para escribir un programa en C con CCS se deben tener en cuenta una serie de conceptos básicos con respecto a su estructura, estas son:



**DIRECTIVAS DE PRE-PROCESADO:** Indispensables para controlan la conversión del programa a código maquina por parte del compilador.

**FUNCIONES:** Son un conjunto de instrucciones. Puede haber una o varias funciones dentro del programa; en cualquier caso siempre se debe definir una como principal mediante la inclusión de una función llamada *main()*.

**INSTRUCCIONES:** Indican como se debe comportar el PIC en todo momento.

**COMENTARIOS:** Permiten describir el significado de cada línea del programa.

A continuación, haremos un repaso de conceptos necesarios para programar PIC's con lenguaje C.

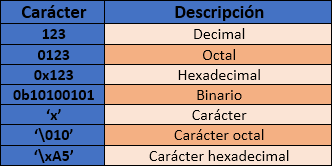
**Tipos de datos**

CCS C acepta los siguientes tipos de datos:



**Constantes**

Las constantes se pueden clasificar en decimal, octal hexadecimal o en binario:



También se definen caracteres especiales, tales como:

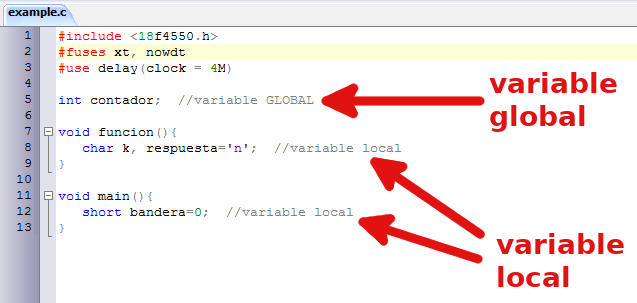


**Variables**

Una variable es un espacio de memoria identificado por un nombre, donde se almacenan valores de un tipo determinado y que pueden variar a lo largo del programa. Para hacer uso de ellas es necesario declararlas antes, para ello se debe indicar el tipo de dato que se manejara y el nombre de la variable, ejemplo:

int alto;  
alto = 5;  
alto = 70;

Las variables pueden declararse como tipo ***LOCAL*** o ***GLOBAL*** dependiendo del uso que se le quiera dar.  
Las ***variables locales*** son declaradas en el interior de una función y solo tienen validez dentro de esta, es decir, solo existen dentro de la función donde fueron declaradas.  
Las ***variables globales*** deben ser declaradas antes de cualquier función y fuera de ellas, pero, a diferencia de las locales, se pueden utilizar y/o modificar su valor dentro de una función del programa.



**Operadores de asignación**

Una expresión de asignación tiene la siguiente forma:  
***expr1 = expr1 operador expr2;*** por ejemplo: ***i = i + 7;***  
pero se puede representar de otra forma más corta:  
***expr1 operador= expr2;*** por ejemplo: ***i += 7;***

A continuación se muestra una tabla que describe los operadores de asignación representados en su forma corta:

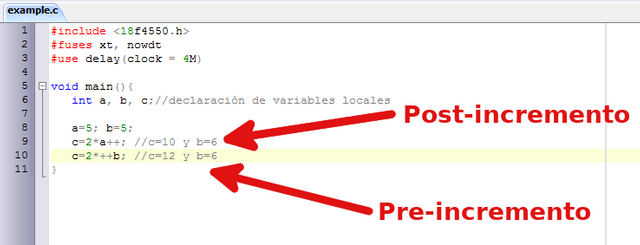


**Operadores aritméticos**

Los operadores aritméticos se usan para realizar operaciones matemáticas:

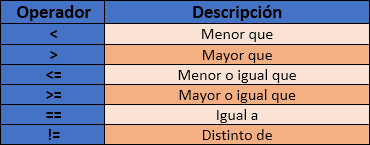


Las operaciones de ***incremento*** y ***decremento*** dependen de la posición del operador, es decir, (++a) representa un ***pre-incremento***, mientras que (a++) representa un ***post-incremento***.



**Operadores relacionales**

Su misión es comparar dos valores y dar un resultado de tipo short: 0 o *false* (falso), 1 o *true* (verdadero).



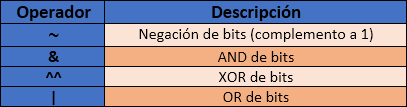
**Operadores lógicos**

Al igual que los operadores relacionales, estos devuelven un resultado de tipo *short*: 0 o *false* (falso), 1 o *true* (verdadero) tras la evaluación de sus operandos.



**Operadores de manejo de bits**

Estos operadores permiten actuar sobre los operandos a nivel de bits y solo pueden ser enteros (tipo *int*).



**Operadores de desplazamiento de bits**

Los operadores de desplazamiento otorgan al C capacidad de control a bajo nivel similar al lenguaje ensamblador. Estos operadores utilizan dos operandos enteros (tipo *int*): el primero es el elemento a desplazar y el segundo, el número de posiciones de bits que se desplaza.



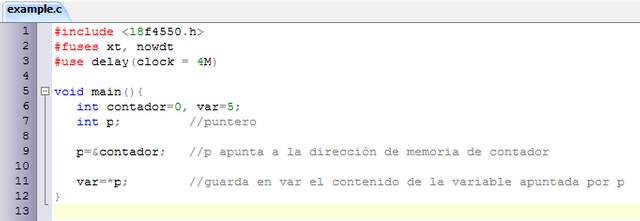
**Operadores de dirección (&) e indirección (\*)**

Los operadores & y \* se utilizan para trabajar con punteros, pero *¿qué es un puntero?* Bueno un puntero es una variable que contiene la dirección de una variable o de una función, es decir, es una variable que apunta a otra variable. El uso de punteros es muy beneficioso debido a que permiten la manipulación indirecta de datos y códigos, pero su uso es muy delicado debido a que se trabaja directamente con direcciones de memoria, por lo tanto para su uso se recomienda práctica.  
Disponemos de dos operadores:



*El operador de dirección & :* Nos da la dirección de memoria de su operando.  
*El operador de indirección \* :* Nos da el contenido de la variable cuya dirección está apuntada por el puntero.

Por ejemplo, si queremos guardar en el puntero *p* la dirección de memoria de la variable entera *contador*, debemos hacer lo siguiente:



**Ejemplos de operaciones**



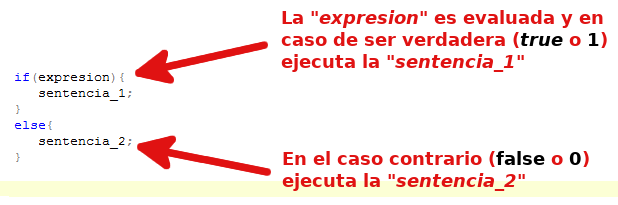
**Declaraciones de control**

Son empleadas para controlar el proceso de ejecución del programa.  
Las declaraciones de control admitidas por CCS son:

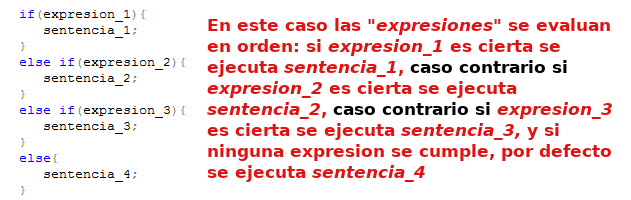
* If-else
* While
* Do-while
* For
* Switch-case
* Return
* Break-continue
* Goto

**IF-ELSE**

Empleado en situaciones donde sea necesario realizar una toma de decisiones.

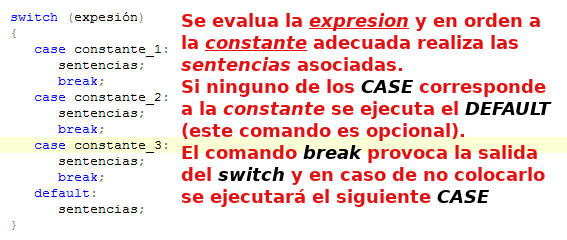


Los IF-ELSE pueden anidarse, es decir, usarse unos dentro de otros permitiendo tomar decisiones múltiples.



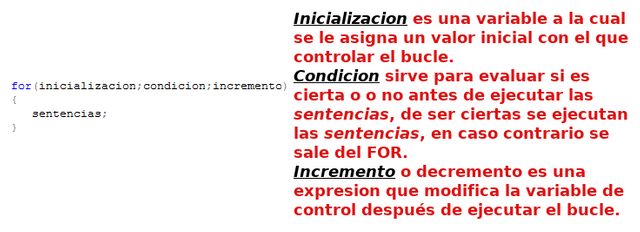
**SWITCH**

Es una herramienta de control de decisión múltiple empleada muy seguido debido a su simplicidad y alta fiabilidad:



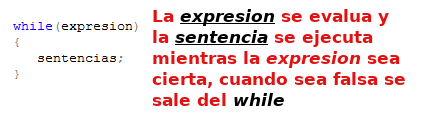
**BUCLE FOR**

Se trata de un ciclo repetitivo empleado para repetir sentencias un número conocido de veces.



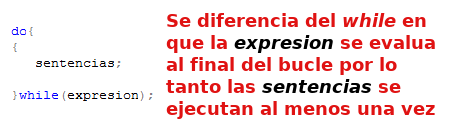
**BUCLE WHILE**

Es un tipo de ciclo empleado para repetir sentencias un número indefinido de veces siempre que se cumpla una condición especifica (expresión).

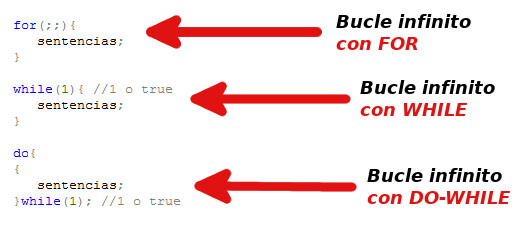


**BUCLE DO-WHILE**

Parecido al ciclo *while*, con la diferencia de que este se ejecuta al menos una sola vez.



Si se ejecuta alguno de los siguientes segmentos de código se consigue ciclo infinito



**RETURN**

Se emplea para devolver datos en las funciones.

**BREAK**

Permite salir de un bucle. Se utiliza para *while*, *do-while*, *for* y *switch*.

**GOTO**

Provoca un salto incondicional a cualquier lugar dentro del programa.

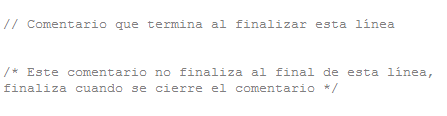
**COMENTARIOS**

Los comentarios cumplen un papel importante dentro del programa y es que facilitan la comprensión de las distintas expresiones tanto para el programador como para quien tiene que interpretar dicho programa, además, no afecta al código ni al funcionamiento normal del PIC por lo que pueden ser tan extensos como se quiera.

Existen dos formatos de comentarios según su uso:

**Comentario de una línea:** Al colocar estos signos // se comienza el comentario y finaliza automáticamente al final de la línea.

**Comentario multi-lí­nea:** Se logran utilizando los signos /\* y \*/ al inicio y al final del comentario, pero no pueden repetirse dentro del mismo.



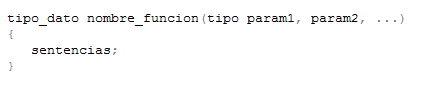
**FUNCIONES DE USUARIO**

Una función se describe en pocas palabras como un bloque de sentencias agrupadas en un solo paquete que son llamadas a ejecutarse en el momento que se requiera y con solo escribir el nombre de la función. Son empleadas con frecuencia en la ejecución de tareas repetitivas como configuración de puertos, control de giro de un motor, encendido secuencial de leds, control de relés, en fin, el límite es la imaginación, y como son *funciones de usuario* somos nosotros quienes elegimos en que emplearlas.

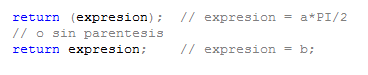
**Características de una función**

* Al igual que las variables, las funciones deben definirse antes de utilizarse.
* Una función puede ser invocada desde una sentencia de otra función, a esto se le llama *Función anidada*.
* Una función puede devolver un valor a la sentencia que la ha llamado a través del comando *return*.
* El tipo de dato del valor que retorna al concluir su ejecución se indica en la definición de la función; en el caso de no indicarse nada se entiende que es un int8 y en el caso de no devolver un valor se debe especificar el valor *void*.
* La función además de devolver un valor, puede recibir *parámetros* y *argumentos*.
* Las funciones pueden agruparse en ficheros de librerías como por ejemplo *ficheros.h*, que se puede utilizar mediante la directiva [*#include*](https://steemit.com/trending/include)*<fichero.h>* incluyéndola al inicio del programa.

La estructura de una función es:

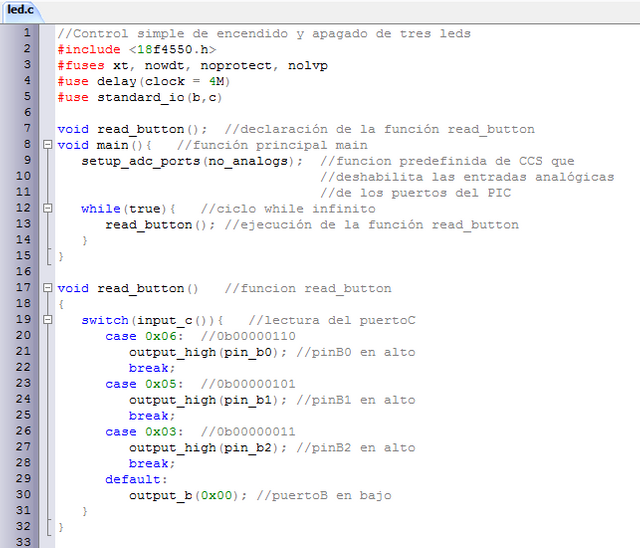


La forma de devolver un valor es mediante la sentencia *return*:



Donde *expresión* debe manejar el mismo tipo de dato que el especificado en la definición de la función. En el caso de no devolver nada se finaliza con *return*, al encontrar esta sentencia el compilador vuelve a la ejecución de la sentencia de llamada. También se puede finalizar la función sin el comando *return*, tan solo con la llave de cierre.

Por ejemplo analicen el siguiente programa:



El programa anterior es un ejemplo simple del uso de una función. Pueden ver que en la línea\_#7 se ha declarado una función llamada *read\_button*, y como no retorna valor alguno se declara tipo *void*. Más adelante, en la lí­nea\_#17 se puede apreciar la definición de la función indicando cada una de las sentencias que debe ejecutar cuando sea invocada. Para ejecutar la función vean la línea\_#13 dentro de la función principal *main* donde podrán notar que solo se requiere escribir su nombre seguido de sus parámetros encerrados entre paréntesis, pero en este caso como no presenta parámetros dichos paréntesis están vacíos. Además, se muestra una aplicación simple de la declaración de control *switch*.