



# MODULO 2

## Expresiones y Sentencias

### Breve descripción

En este documento se explica la diferencia entre expresiones y sentencias, así como también los distintos tipos entre ellas.

**Versión 1.1**

Martin Jerman

Martin.jerman@inspt.utn.edu.ar



## Introducción

Programar es “tirar líneas de código”. ¿Pero, qué líneas de código? ¿Es escribir líneas de código simplemente? Es mucho mas que eso. Programar es idear un algoritmo que resuelve un problema y plasmarlo en un programa. Para ello, necesitaremos saber que posibilidades me ofrece el lenguaje en cuestión, en este caso C.

## Expresiones

Las expresiones son combinaciones de operandos, operadores y paréntesis, que al evaluarse dan como resultado un valor. Cada operando puede ser:

- Variables
- Constantes (valores fijos)
- Funciones
- Otra expresión (definida entre **()** como un subconjunto de la expresión completa)

En el Módulo 1 ya vimos las variables, constantes y operadores. Si hay alguna duda repasarlos en dicho modulo. Respecto a las funciones, es un tema que se desarrollara mas adelante. De momento quedémonos con la idea de que puede ser un operando en una expresión.

Dicho esto, veamos un ejemplo de expresión:

`2+3;`

Esta es una expresión dado que tras ser evaluada devuelve un único resultado. Por consiguiente, también son expresiones los siguientes ejemplos:

(sabiendo que `int a=1, b=3;`)

`a+1;`

`a+b;`

`a>b;`

`(b+a) ;`

En función del resultado obtenido las podemos clasificar en:

- Expresiones numéricas: aquellas cuyo resultado es un número.
- Expresiones alfanuméricas: aquellas que devuelven un carácter.
- Expresiones lógicas: aquellas que devuelven un verdadero o falso por resultado.

## Instrucciones

Como definición de instrucciones entendemos que es un hecho o suceso de duración limitada que genera unos cambios en la ejecución del programa. Con esta premisa, podemos distinguir 3 tipos de instrucciones:

- Sentencias
- Condicionales (o bifurcaciones).
- Repeticiones (o ciclos)

*Los bloques son conjuntos de instrucciones encerrados entre llaves {}*

Veamos los tipos de instrucciones en detalle.

### Sentencias

Las sentencias son todas aquellas instrucciones que caben en una línea. Ejemplos de ello son las declaraciones de variables (`int a, b;`), asignaciones (`a=1;`) o las llamadas a `printf`. Toda sentencia debe terminar con un punto y coma “;”.



Las sentencias son los elementos básicos en los que se divide el código en un lenguaje de programación. Al fin y al cabo, un programa no es más que un conjunto de sentencias que se ejecutan una tras otra para realizar una cierta tarea. En nuestro caso, el signo que las separa una sentencia de otra es el punto y coma (;).

Las sentencias se dividen en simples y compuestas (o estructuradas):

- Simples: son simples las sentencias de asignación o llamadas a funciones.
- Compuestas: son aquellas que implican un bloque de código. Son de este tipo todas las estructuras de control (condicionales, repetitivas, etc.). Ya veremos mas adelante cada una en detalle.

## Condicionales

### Sentencia if

Esta instrucción evalúa una expresión y en función de su resultado (verdadero o falso) decide qué sección de código ejecutar. También llamada como sentencia `if`, su forma es la siguiente:

```
if ( condición )
{
    printf("Acción por verdadero\n");
}
else
{
    printf("Acción por falso\n");
}
```

En el lenguaje C, si la acción por verdadero (o falso) es una única instrucción, podemos omitir los caracteres de bloque (las llaves), quedando el código mas compacto:

```
if (condición)
    printf("Acción por verdadero\n");
else
    printf("Acción por falso\n");
```

Ahora bien, ¿qué colocamos en `condición`? Pues bien, cualquier expresión que se nos ocurra. Podría ser `a>b` o `b<5` o `a==4` o cualquiera otra.

La sentencia `if` admite la omisión por el caso falso. De este modo, es válido y correcto colocar únicamente las sentencias a ejecutar por el caso verdadero:

```
if (condición)
{
    printf("Acciones solo por\n");
    printf("el caso verdadero\n");
}
```

*Puedo omitir el caso por falso,  
pero nunca el caso verdadero*

**Importante:** en C, toda instrucción devuelve un resultado. Y dado que en C el valor cero es falso y cualquier otra cosa es verdadero, puedo colocar sentencias en la condición y utilizar el valor devuelto como "verdadero" o "falso". Al empezar a programar es fácil cometer el error de olvidar un signo = en la condición por igualdad. Por ejemplo:

```
if (a = 5)
    printf("A vale 5!\n");
else
    printf("A es distinto de 5!\n");
```

En este código, el programa compila sin problemas y el valor que evalúa el `if` es el resultado de la asignación, que es un 1 si se pudo asignar la variable o un cero si no se pudo. La real intención del programador era evaluar si la variable `a` era igual a 5.



La sentencia condicional también admite una versión compacta. Esta versión compacta se suele utilizar cuando hay una sentencia muy corta a ejecutar por cada caso o por un único caso. Su forma es:

```
condicion ? sentenciaPorVerdadero : sentenciaPorFalso ;
```

Nuevamente, en condición puede ir cualquier expresión seguida por el signo de cierre de interrogación y finalmente las dos instrucciones (por verdadero y por falso) separadas por un carácter dos puntos ":". Por ejemplo

```
2 + 3 == 5 ? printf("Verdadero!\n") : printf("****falso****");
```

Ahora bien, ¿cómo desarrollaríamos un condicional para saber si un alumno de X años está en la escuela primaria, secundaria o universitaria? Podemos armar una sentencia if por cada condición:

```
if (edad <= 12)
    printf("Escuela primaria\n");
if (edad > 12 && edad <=18)
    printf("Escuela secundaria\n");
if (edad >18)
    printf("Escuela universitaria\n");
```

En el ejemplo, las expresiones hacen que solamente una sentencia if sea la que muestre el mensaje. Esto es así porque las expresiones que definimos se descartan una a la otra. Pero todas las sentencias if se ejecutan por igual. Podríamos re-escribir las sentencias if anteriores anidandolas, quedando:

```
if (edad <= 12)
    printf("Escuela primaria\n");
else
    if (edad > 12 && edad <=18)
        printf("Escuela secundaria\n");
    else
        if (edad >18)
            printf("Escuela universitaria\n");
```

De esta manera, si bien el resultado es el mismo, el comportamiento es distinto: aquí las sentencias if **se descartan** una a la otra. Si la primera sentencia if es verdadera, las demás no ejecutan. Aplicando la idea del descarte, podríamos reescribirlo quedando:

```
if (edad <= 12)
    printf("Escuela primaria\n");
else
    if (edad <=18)
        printf("Escuela secundaria\n");
    else
        printf("Escuela universitaria\n");
```

Esto es importante tenerlo presente al momento de desarrollar; dado que a veces es mejor anidar para descartar posibilidades (como en el último ejemplo) en lugar de listar condiciones independientemente.

## Sentencia switch

Supongamos que dadas 5 opciones de un menú, tenemos que saber qué opción ingreso el usuario y en función de lo ingresado hacer algo diferente. Hasta este punto, tenemos 2 opciones: 1) hacer if's anidados evaluando cada opción posible o 2) hacer distintos if's no anidados evaluando cada opción. Ambos casos son con if's. Pero para estos casos en los que la variable a evaluar puede tomar valores discretos, existe la sentencia switch. Veamos su estructura:

```
switch ( miVariable ) {
case 1:
```

*El switch solamente puede usarse cuando los posibles valores de la variable son discretos (char o int)*



```
a++;      // sentencia o sentencias por si miVariable es igual a 1
break;    // sentencia que corta la ejecución del bloque switch
case 2:
    a+=10;
    break;
case 3:
    a+=99;
    break;
default:   // si miVariable no es igual a ningún caso, ejecuta aquí
    printf("El valor ingresado no es válido\n");
}
```

Como se ve en la estructura, `miVariable` debe tomar un valor discreto, es decir que debe valer 1 o 2 o 50 (si fuera un `int`) o 'a' (si fuera `char`) y los evalúa según cada caso que se especifica con la sentencia `case`. Si la variable tiene un valor que no se corresponde con ningún caso especificado, se ejecutará el caso `default`. Es importante saber que el caso `default` es opcional, por lo que si no se coloca el `switch` no ejecuta nada para los casos no contemplados.

Es importante entender cómo funciona el `switch`. Al iniciar la evaluación de la variable, se compara el valor de la variable con cada caso especificado. Al encontrar una coincidencia, se ejecutan todas las instrucciones que siguen. En el ejemplo, si `miVariable` vale 2, se ejecutarían todas las instrucciones que se listan abajo, de todos los casos, hasta terminar el `switch`. Es por este motivo que se coloca la instrucción `break`; al terminar las instrucciones de cada caso, para evitar que las demás se ejecuten.

Pero el `switch` tiene restricciones:

- 1- Como la variable a evaluar debe tener un valor discreto, ésta **no puede ser una variable float**.
- 2- La restricción anterior nos lleva a otra: el `switch` no puede evaluar rangos en sus casos. Es decir que no podemos poner `case 1 to 10:` o `case 1..10:`. Estas instrucciones no son válidas. Para dicho caso, lo que se puede hacer es aprovechar la forma que tiene en `switch` de ejecutar, colocando todos los casos que especifican el rango uno por uno y dejando el espacio de instrucciones libre. De este modo, el ultimo caso es el que debe llevar las instrucciones a ejecutar. Por ejemplo:

```
switch (a) {
case 1:
case 2:
case 3:
    a+=99;
    break;
case 4:
    a-=20;
    break;
}
```

Un uso común del `switch` en asociándolo a un menú de opciones: se muestran las opciones, se le pide al usuario ingresar la opción deseada y usar un `switch` para evaluar la opción ingresada.

## Repeticiones

Las repeticiones o también llamadas bucles o ciclos son estructuras de control reiterativas; es decir que repiten las mismas instrucciones hasta que cierta condición **deje de ser verdadera**.

Dentro de estas instrucciones tenemos 3 casos:

- Ciclo while
- Ciclo do-while
- Ciclo for



## Ciclo while

Este ciclo evalúa una condición dada antes de ejecutar el bloque de código que encierra y si es verdadero, ejecuta el código una vez. Al finalizar el bloque de código, se evalúa la condición nuevamente para ver si cambia. Si sigue siendo verdadera, se vuelve a ejecutar el bloque; caso contrario se sale del ciclo. La estructura de la sentencia es:

```
while ( condicion )
{
    Sentencia1;
    Sentencia2;
}
```

Veamos un ejemplo:

```
#include <stdio.h>
int main()
{
    int a=0,maximo=5;           // declare las variables a utilizar
    while (a<=maximo)          // inicio ciclo while
    {                           // inicio del bloque de instrucciones
        printf("variable a=%d\n",a);
        a++;
    }                           // fin del bloque del ciclo while
    return 0;
}
```

En el ejemplo, el programa muestra el valor de la variable "a" 6 veces.

Como se ve, las variables a utilizar en el ciclo se deben declarar debajo de `main`, como cualquier otra variable. Además, las variables que se utilizan en la condición del `while` deben estar inicializadas o cargadas con algún valor para asegurar que la condición funcione como se espera.

Por ultimo y muy importante, la variable que se evalúa en la condición del ciclo `while` **debe ser modificada** dentro del bloque de instrucciones del ciclo (ver el resaltado en rojo). Si esto no ocurre, el ciclo no tendrá condición de corte y por ende será infinito.

Tras ver el ejemplo y como es que ejecuta, ¿qué pasaría si la condición fuese falsa desde el primer momento? Pues bien, todo el ciclo `while` se ignoraría. Por esta razón se dice que el ciclo `while` **puede ejecutar o más veces**.

## Ciclo do-while

El ciclo `do while` tiene la siguiente forma:

```
do {
    sentencia 1;
    sentencia 2;
    ...
} while ( condicion );
```

Como se puede observar, primero se ejecutan las sentencias del ciclo antes de evaluar la condición expresada después del `while`. Por analogía, su funcionamiento es igual al ciclo `while` con la diferencia del momento al que se evalúa la condición. Es por este motivo que se dice que el `do-while` **ejecuta 1 o más veces**.

Para este ciclo son válidas todas las demás afirmaciones respecto de la forma que debe tener la condición a evaluar y el uso de las variables en dicha condición.

## Ciclo for

Este ciclo requiere de una estructura entre sus paréntesis que no es únicamente una condición, sino también instrucciones.

Veámoslo en un ejemplo:



```
int acum=0;
for (int i=0 ; i<5 ; i++) {
    acum+=i;
    printf("ejecución de un ciclo\n");
}
```

En este ejemplo vemos como se declara y funciona el ciclo. Primero los parámetros (int i=0 ; i<5 ; i++) . Los parámetros son:

- El primer parámetro define una variable que se usa como contador del ciclo. El alcance de esta variable es local al ciclo únicamente y debe ser inicializada en algún valor (por lo general es cero).
- El segundo parámetro define la condicion a evaluar utilizando la variable contador, definida anteriormente.
- El tercer parámetro define el salto o paso que dará el contador en cada ciclo. Por lo general es 1, pero puede ser lo que se necesite.

Veamos como funciona. En el primer ciclo, se ejecuta la sentencia del primer parámetro. Luego se evalúa la expresión del segundo parámetro. Si es verdadero se sigue con la ejecución del las instrucciones del ciclo; caso contrario se sale del ciclo. Al terminar de ejecutar cada vez la lista de instrucciones del ciclo, se ejecuta la instrucción del tercer parámetro o incremento y finalmente se vuelve a evaluar el segundo parámetro para ver si se sigue con el ciclo o no.

Como se puede ver, este ciclo **ejecuta una N cantidad de veces**, determinada por los tres parámetros: cuando inicia, cuando termina y cuanto avanza.

A diferencia del ciclo while, en el ciclo `for` no es necesario modificar la variable contador (también llamada **variable de control** del ciclo) en las instrucciones del bloque del ciclo dado que el propio ciclo lo explícitamente en el tercer término.



## Notas extra sobre ciclos...

En este modulo vimos los distintos tipos de sentencias. Dada la naturaleza de C, hay instrucciones que pueden tener usos especiales que no son recomendados en la materia Programación I, pero pueden ser útiles.

Dado que en C el valor cero se entiende como falso y lo distinto a cero como verdadero, es correcto y valido lo siguiente:

```
if (5)
    printf("5 siempre valdra 5!\n");
else
    printf("esto jamas ejecutara!\n");
```

En este caso el if está bien escrito y siempre entrará por el caso verdadero y la condicion por falso será inalcanzable. Si el Codeblocks está bien configurado, el compilador lo advertirá.

¿Pero qué ocurre si aplicamos la misma idea a un ciclo while?

```
while (3){
    printf("ejecuto ciclo\n");
}
```

Si, es un **ciclo infinito**. Esto no es lo que se suele utilizar, pero en algún caso puede ser necesario. ¿Como se puede hacer para cortar el ciclo infinito en estos casos? Utilizando la instrucción `break`; dentro de `if` que evalué algo. Por ejemplo:

```
char opcion;
while (1)
{
    printf("ingrese una 'f' para finalizar\n");
    scanf("%c", &opcion);
    if (opcion=='f')
        break;
    else
        printf("Se ingreso %c\n",opcion);
}
```

Este mismo ejemplo aplica para el ciclo `for`. De hecho, el ciclo `for` es aún más interesante.

Por definición los parámetros del ciclo `for` no son únicamente parámetros, sino instrucciones. Por ello, podemos tener mas de un contador, una única pero compleja expresión a evaluar en el segundo parámetro y ninguna o una o varias instrucciones de salto. Dicho esto, es totalmente valido el siguiente ejemplo:

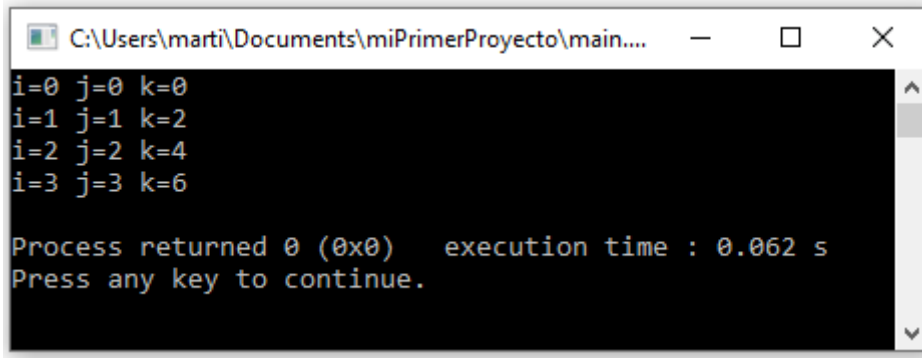
```
int i,j,k=0;
for (i=0, j=0; i<5 && j < 4 && k < 10 ; i++)
{
    printf("i=%d j=%d k=%d\n",i,j++,k);
    k+=2;
}
```





En este caso, las variable  $i$  y  $j$  se inicializan en el ciclo, pero no así  $k$ . En estos casos confiamos en que  $k$  tiene un valor valido ya fuera del ciclo. Por otra parte, la expresión puede evaluar cualquier variable, sea un contador del ciclo o no (en este caso  $k$  no parecería ser contador del ciclo). Finalmente, en el tercer parámetro solo incrementamos el paso de la variable  $i$ ; pero en el mismo bloque del ciclo incrementamos las otras variables contador, lo que hace que el ciclo tenga sentido; de hecho, ¡la variable  $k$  tiene un salto de dos unidades!

La salida por pantalla seria la siguiente:



```
C:\Users\marti\Documents\miPrimerProyecto\main....
i=0 j=0 k=0
i=1 j=1 k=2
i=2 j=2 k=4
i=3 j=3 k=6

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

Estos usos mas avanzados no son aconsejables para los que recién se inician en la programación. Pero es bueno saberlos para casos futuros o cuando se lee código ajeno.