**PROGRAMACIÓN EN C (CONTINUACIÓN), OPERADORES ,BUCLES Y CONDICIONALES**

**1.-OPERADORES**

**Operadores de comparación**

Los operadores de condición se utilizan para comprobar las condiciones de las sentencias de control de flujo (las estudiaremos en el capítulo sentencias).

Cuando se evalúa una condición el resultado que se obtiene es 0 si no se cumple y un número distinto de 0 si se cumple. Normalmente cuando se cumplen devuelven un 1.

Los operadores de comparación son:

|  |  |  |
| --- | --- | --- |
| == | igual que | se cumple si son iguales |
| != | distinto que | se cumple 1 si son diferentes |
| > | mayor que | se cumple si el primero es mayor que el segundo |
| < | menor que | se cumple si el primero es menor que el segundo |
| >= | mayor o igual que | se cumple si el primero es mayor o igual que el segundo |
| <= | menor o igual que | se cumple si el primero es menor o igual que el segundo |

Veremos la aplicación de estos operadores en el capítulo **Sentencias**. Pero ahora vamos a ver unos ejemplos:

#include <stdio.h>

int main()

{

printf( "10 > 5 da como resultado %i\n", 10>5 );

printf( "10 > 5 da como resultado %i\n", 10>5 );

printf( "5== 5 da como resultado %i\n", 5==5 );

printf( "10==5 da como resultado %i\n", 10==5 );

}

No sólo se pueden comparar constantes, también se pueden comparar variables.

**Operadores lógicos**

Estos son los que nos permiten unir varias comparaciones: 10>5 **y** 6==6. Los operadores lógicos son: AND (&&), OR (||), NOT(!).

Operador && (AND, en castellano Y): Devuelve un 1 si se cumplen dos condiciones.

printf( "Resultado: %i", (10==10 && 5>2 );

Operador || (OR, en castellano O): Devuelve un 1 si se cumple una de las dos condiciones.

Operador ! (NOT, negación): Si la condición se cumple NOT hace que no se cumpla y viceversa.

**2.-BUCLES**

Los bucles nos ofrecen la solución cuando queremos repetir una tarea un número determinado de veces. Supongamos que queremos escribir 100 veces la palabra *hola*. Con lo que sabemos hasta ahora haríamos:

#include <stdio.h>

int main()

{

printf( "Hola\n");

printf( "Hola\n");

printf( "Hola\n");

printf( "Hola\n");

printf( "Hola\n");

... (y así hasta 100 veces)

}

¡Menuda locura! Y si queremos repetirlo más veces nos quedaría un programa de lo más largo.

Sin embargo usando un bucle *for* el programa quedaría:

#include <stdio.h>

int main()

{

int i;

for ( i=0 ; i<100 ; i++ )

{

printf( "Hola\n" );

}

}

Con lo que tenemos un programa más corto.

**El bucle For**

El formato del bucle for es el siguiente:

**for(** dar valores iniciales **;** condiciones **;** incrementos **)**

**{**

conjunto de intrucciones a ejecutar en el bucle

**}**

Vamos a verlo con el ejemplo anterior:

...

for ( i=0 ; i<100 ; i++ )

...

En este caso asignamos un valor inicial a la variable *i*. Ese valor es *cero*. Esa es la parte de *dar valores iniciales*. Luego tenemos *i<100*. Esa es la parte *condiciones*. En ella ponemos la condición es que i sea menor que 100, de modo que el bucle se ejecutará mientras i sea menor que 100. Es decir, mientras se cumpla la condición. Luego tenemos la parte de *incrementos*, donde indicamos cuánto se incrementa la variable.

Como vemos, el for va delante del grupo de instrucciones a ejecutar, de manera que si la condición es falsa, esas instrucciones no se ejecutan ni una sola vez.

Cuidado: No se debe poner un ";" justo después de la sentencia for, pues entonces sería un bucle vacío y las instrucciones siguientes sólo se ejecutarían una vez. Veamoslo con un ejemplo:

#include <stdio.h>

int main()

{

int i;

for ( i=0 ; i<100 ; i++ ); /\* Cuidado con este punto y coma \*/

{

printf( "Hola\n" );

}

}

Este programa sólo escribirá en pantalla

Hola

una sola vez.

Tambíen puede suceder que quieras ejecutar un cierto número de veces una sóla instrucción (como sucede en nuestro ejemplo). Entonces no necesitas las llaves "{}":

#include <stdio.h>

int main()

{

int i;

for ( i=0 ; i<100 ; i++ ) printf( "Hola\n" );

}

o también:

for ( i=0 ; i<100 ; i++ )

printf( "Hola\n" );

Sin embargo, yo me he encontrado muchas veces que es mejor poner las llaves aunque sólo haya una instrucción; a veces al añadir una segunda instrucción más tarde se te olvidan las comillas y no te das cuenta. Parece una tontería, pero muchas veces, cuando programas, son estos los pequeños fallos los que te vuelven loco.

En otros lenguajes, como Basic, la sentencia for es muy rígida. En cambio en **C** es muy flexible. Se puede omitir cualquiera de las secciones (inicialización, condiciones o incrementos). También se pueden poner más de una variable a inicializar, más de una condición y más de un incremento. Por ejemplo, el siguiente programa sería perfectamente correcto:

#include <stdio.h>

int main()

{

int i, j;

for( i=0, j=5 ; i<10 ; i++, j=j+5 )

{

printf( "Hola " );

printf( "Esta es la línea %i", i );

printf( "j vale = %i\n", j );

}

}

Como vemos en el ejemplo tenemos más de una variable en la sección de inicialización y en la de incrementos. También podíamos haber puesto más de una condición. Los elementos de cada sección se separan por comas. Cada sección se separa por punto y coma.

Más tarde veremos cómo usar el for con cadenas.

**While**

El formato del bucle while es es siguiente:

**while (** condición **)**

**{**

bloque de instrucciones a ejecutar

**}**

While quiere decir *mientras*. Aquí se ejecuta el bloque de intrucciones mientras se cumpla la condición impuesta en while.

Vamos a ver un ejemplo:

#include <stdio.h>

int main()

{

int contador = 0;

while ( contador<100 )

{

contador++;

printf( "Ya voy por el %i, pararé enseguida.\n", contador );

}

}

Este programa imprime en pantalla los valores del 1 al 100. Cuando i=100 ya no se cumple la condición. Una cosa importante, si hubíesemos cambiado el orden de las instrucciones a ejecutar:

...

printf( "Ya voy por el %i, pararé enseguida.\n", contador );

contador++;

...

En esta ocasión se imprimen los valores del 0 al 99. Cuidado con esto, que a veces produce errores difíciles de encontrar.

**Do While**

El formato del bucle do-while es:

**do**

**{**

instrucciones a ejecutar

**} while (** *condición* **);**

La diferencia entre *while* y *do-while* es que en este último, la condición va despues del conjunto de instrucciones a ejecutar. De esta forma, esas instrucciones se ejecutan al menos una vez.

Su uso es similar al de while.

**3.-CONDICIONALES**

Hasta aquí hemos visto cómo podemos repetir un conjunto de instrucciones las veces que deseemos. Pero ahora vamos a ver cómo podemos controlar totalmente el flujo de un programa. Dependiendo de los valores de alguna variable se tomarán unas acciones u otras. Empecemos con la sentencia *if*.

**If**

La palabra *if* significa *si* (condicional), pero supongo que esto ya lo sabías. Su formato es el siguiente:

**if (** *condición* **)**

**{**

*instrucciones a ejecutar*

**}**

Cuando se cumple la condición entre paréntesis se ejecuta el bloque inmediatamente siguiente al if (bloque *instrucciones a ejecutar*).

En el siguiente ejemplo tenemos un programa que nos pide un número, si ese número es 10 se muestra un mensaje. Si no es 10 no se muestra ningún mensaje:

#include <stdio.h>

int main()

{

int num;

printf( "Introduce un número " );

scanf( "%i", &num );

if (num==10)

{

printf( "El número es correcto\n" );

}

}

Como siempre, la condición es falsa si es igual a cero. Si es distinta de cero será verdadera.

**If - Else**

El formato es el siguiente:

**if (** condición **)**

**{**

bloque que se ejecuta si se cumple la condición

**}**

**else**

**{**

bloque que se ejecuta si no se cumple la condición

**}**

En el if si no se cumplía la condición no se ejecutaba el bloque siguiente y el programa seguía su curso normal. Con el if else tenemos un bloque adicional que sólo se ejecuta si no se cumple la condición. Veamos un ejemplo:

#include <stdio.h>

int main()

{

int a;

printf( "Introduce un número " );

scanf( "%i", &a );

if ( a==8 )

{

printf ( "El número introducido era un ocho.\n" );

}

else

{

printf ( "Pero si no has escrito un ocho!!!"\n" );

}

}

Al ejecutar el programa si introducimos un 8 se ejecuta el bloque siguiente al if y se muestra el mensaje:

El número introducido era un ocho.

Si escribimos cualquier otro número se ejecuta el bloque siguiente al else mostrándose el mensaje:

Pero si no has escrito un ocho!!!

**If else if**

Se pueden poner if else anidados si se desea:

#include <stdio.h>

int main()

{

int a;

printf( "Introduce un número " );

scanf( "%i", &a );

if ( a<10 )

{

printf ( "El número introducido era menor de 10.\n" );

}

else if ( a>10 && a<100 )

{

printf ( "El número está entre 10 y 100"\n" );

}

else if ( a>100 )

{

printf( "El número es mayor que 100\n" );

}

printf( "Fin del programa\n" );

}

El símbolo **&&** de la condición del segundo if es un AND (Y). De esta forma la condición queda: *Si a es mayor que 10 Y a es menor que 100*.

Y así todos los if else que queramos. Si la condición del primer if es verdadera se muestra el mensaje *El número introducido era menor de 10* y se saltan todos los if-else siguientes (se muestra *Fin del programa*). Si la condición es falsa se ejecuta el siguiente else-if y se comprueba si a está entre 10 y 100. Si es cierto se muestra *El número está entre 10 y 100*. Si no es cierto se evalúa el último else-if.

**Switch**

El formato de la sentencia switch es:

**switch (** variable **)**

**{**

**case** *opción 1***:**

código a ejecutar si la variable tiene el

valor de la opción 1

**break;**

**case** *opción 2***:**

código a ejecutar si la variable tiene el

valor de la opción 2

**break;**

**default:**

código que se ejecuta si la variable tiene

un valor distinto a los anteriores

**break;**

**}**

Vamos a ver cómo funciona. La sentencia switch sirve par elegir una opción entre varias disponibles. Aquí no tenemos una condición que se debe cumplir sino el valor de una variable. Dependiendo del valor se cumplirá un caso u otro.

Vamos a ver un ejemplo de múltiples casos con if-else y luego con switch:

#include <stdio.h>

int main()

{

int num;

printf( "Introduce un número " );

scanf( "%i", &num );

if ( num==1 )

printf ( "Es un 1\n" );

else if ( num==2 )

printf ( "Es un 2\n" );

else if ( num==3 )

printf ( "Es un 3\n" );

else

printf ( "No era ni 1, ni 2, ni 3\n" );

}

Ahora con switch:

#include <stdio.h>

int main()

{

int num;

printf( "Introduce un número " );

scanf( "%i", &num );

switch( num )

{

case 1:

printf( "Es un 1\n" );

break;

case 2:

printf( "Es un 2\n" );

break;

case 3:

printf( "Es un 3\n" );

break;

default:

printf( "No es ni 1, ni 2, ni 3\n" );

}

}

Como vemos el código con switch es más cómodo de leer.

Vamos a ver qué pasa si nos olvidamos algún break:

#include <stdio.h>

int main()

{

int num;

printf( "Introduce un número " );

scanf( "%i", &num );

switch( num )

{

case 1:

printf( "Es un 1\n" );

/\* Nos olvidamos el break que debería haber aquí \*/

case 2:

printf( "Es un 2\n" );

break;

default:

printf( "No es ni 1, ni 2, ni 3\n" );

}

}

Si al ejecutar el programa escribimos un dos tenemos el mensaje *Es un dos*. Todo correcto. Pero si escribimos un uno lo que nos sale en pantalla es:

Es un 1

Es un 2

¿Por qué? Pues porque cada caso empieza con un case y acaba donde hay un break. Si no ponemos break aunque haya un case el programa sigue hacia adelante. Por eso se ejecuta el código del case 1 y del case 2.

Puede parecer una desventaja pero a veces es conveniente. Por ejemplo cuando dos case deben tener el mismo código. Si no tuviéramos esta posibilidad tendríamos que escribir dos veces el mismo código. (Vale, vale, también podríamos usar funciones, pero si el código es corto puede ser más conveniente no usar funciones. Ya hablaremos de eso más tarde.).

Sin embargo switch tiene algunas limitaciones, por ejemplo no podemos usar condiciones en los case. El ejemplo que hemos visto en el apartado if-else-if no podríamos hacerlo con switch.

**EJERCICIOS**

**CONDICIONALES**

1.-Programa que indica qué numero de los dos introducidos por el usuario es mayor.

2.-Programa que escriba en pantalla un comentario con respecto a la temperatura del día.Temp menor de 15 ,frio.Temp menor de 25 ,templado.Temp mayor de 25 ,calor

3.-Programa que resuelve ecuaciones de segundo grado.Se tendrá que comprobar que la raíz no es negativa , si eso pasa aparecerá “no tiene solución”.

4.- Programa que muestre una cuenta atrás desde diez hasta cero.

5.-Programa que indica la correspondencia de un número introducido por el usuario con un mes del año (aconsejable realizarlo con SWITCH).

6.-Programa que simula un cajero automático con un saldo inicial de 1000 Euros.El cajero tendrá las opciones 1.Ingreso en cuenta.2.Reintegro 3.Ver el saldo disponible. 0.Salir

**BUCLES**

7.-Programa que muestra los veinte primeros números naturales.

8.-Programa que muestra los números pares hasta 30.

9.-Programa que muestre los múltiplos de siete (hasta 123).

10.-Programa que diga si un número es par o no. Finaliza al pulsar “0”.