1.-EL PRIMER PROGRAMA: *Hola Mundo*

```
#include <stdio.h>

int main() {
   printf("Hola mundo\n");
   return 0;
}
```

Este programa lo único que hace es sacar por pantalla el mensaje:

```
Hola mundo
```

Vamos ahora a comentar el programa línea por línea (Esto no va a ser más que una primera aproximación).

```
#include <stdio.h>
#include <math.h> caso de la librería matemática
```

#include es lo que se llama una directiva. Sirve para indicar al compilador que incluya otro archivo. Cuando en compilador se encuentra con esta directiva la sustituye por el archivo indicado. En este caso es el archivo stdio.h que es donde está definida la función printf, que veremos luego.

```
int main()
```

Es la función principal del programa. Todos los programas de C deben tener una función llamada main. Es la que primero se ejecuta. El **int** (viene de Integer=Entero) que tiene al principio significa que cuando la función main acabe devolverá un número entero. Este valor se suele usar para saber cómo ha terminado el prorama. Normalmente este valor será 0 si todo ha ido bien, o un valor distinto si se ha producido algún error (pero esto lo decidimos nosotros, ya lo veremos). De esta forma si nuestro programa se ejecuta desde otro el programa 'padre' sabe como ha finalizado, si ha habido errores o no.

Se puede usar la definición 'void main()', que no necesita devolver ningún valor, pero se recomienda la forma con 'int' que es más correcta. A lo largo de este curso verás muchos ejemplos que uso 'void main' y falta el return 0; del final, el código funciona correctamente pero puede dar un 'warning' al compilar. En estos momentos estoy intentando corregir esto, pido perdón por las molestias.

```
{
```

Son las llaves que indican el comienzo de una función, en este caso la función main.

```
/* Aquí va el cuerpo del programa */
```

Esto es un comentario, no se ejecuta. Sirve para describir el programa. Conviene acostumbrarse a comentar los programas. Un comentario puede ocupar más de una línea. Por ejemplo el comentario:

```
/* Este es un comentario
   que ocupa dos filas */
```

es perfectamente válido.

```
printf( "Hola mundo\n" );
```

Aquí es donde por fin el programa hace algo que podemos ver al ejecutarlo. La

función printf muestra un mensaje por la pantalla. Al final del mensaje "Hola mundo" aparece el símbolo '\n'; este hace que después de imprimir el mensaje se pase a la línea siguiente.

Fíjate en el ";" del final. Es la forma que se usa en C para separar una instrucción de otra. Se pueden poner varias en la misma línea siempre que se separen por el punto y coma.

```
return 0;
```

Como he indicado antes el programa al finalizar develve un valor entero. Como en este programa no se pueden producir errores (nunca digas nunca jamás) la salida siempre será 0. La forma de hacer que el programa devuelva un 0 es usando return. Esta línea significa 'finaliza la función main haz que devuelva un 0.

```
}
```

...y cerramos llaves con lo que termina el programa. Todos los programas finalizan cuando se llega al final de la función main.

2.-VARIABLES: *Tipos , declaraciones, e impresión(printf)*

El tipo Int

En una variable de este tipo se almacenan números enteros (sin decimales). El rango de valores que admite es -32767 a 32767. Cuando definimos una variable lo que estamos haciendo es decirle al compilador que nos reserve una zona de la memoria para almacenar datos de tipo int. Para guardarla necesitaremos 16 bits de la memoria del ordenador (2^{16}=32767). Para poder usar una variable primero hay que declararla (definirla). Hay que decirle al compilador que queremos crear una variable y hay que indicarle de qué tipo. Por ejemplo:

```
int numero;
```

Esto hace que declaremos una variable llamada *número* que va a contener un número entero.

¿Pero dónde se declaran las variables?

Tenemos dos posibilidades, una es declararla como global y otra como local. Por ahora vamos a decir que global es aquella variable que se declara fuera de la función main y local la que se declara dentro:

Variable Global

```
#include <stdio.h>

int x;

int main()
        {
        }
```

Variable Local

```
#include <stdio.h>

int main()
        {
        int x;
        }
```

La diferencia práctica es que las variables globales se pueden usar en cualquier procedimiento. Las variables locales sólo pueden usarse en el procedimiento en el que se declaran. Como por ahora sólo tenemos el procedimiento (o función, o rutina, o subrutina, como prefieras) *main* esto no debe preocuparnos mucho por ahora. Cuando estudiemos cómo hacer un programa con más funciones aparte de main volveremos sobre el tema. Sin embargo debes saber que es buena costumbre usar variables locales que globales. Ya veremos por qué.

Podemos declarar más de una variable en una sola línea:

```
int x, y;
```

Mostrar variables por pantalla

Vamos a ir u poco más allá con la función printf. Supongamos que queremos mostrar el contenido de la variable

x
por pantalla:

```
printf( "%i", x );
```

Suponiendo que x valga 10 (x=10) en la pantalla tendríamos:

```
10
```

Empieza a complicarse un poco ¿no? Vamos poco a poco. ¿Recuerdas el símbolo "\" que usábamos para sacar ciertos caracteres? Bueno, pues el uso del "%" es parecido. "%i" no se muestra por pantalla, se sustituye por el valor de la variable que va detrás de las comillas. (%i, de integer=entero en inglés).

Para ver el contenido de dos variables, por ejemplo x e y, podemos hacer:

```
printf( "%i ", x );
printf( "%i", y );
```

resultado (suponiendo x=10, y=20):

```
10 20
```

Pero hay otra forma mejor:

```
printf( "%i %i", x, y );
```

... y así podemos poner el número de variables que queramos. Obtenemos el mismo resultado con menos trabajo. No olvidemos que por cada variable hay que poner un %i dentro de las comillas.

También podemos mezclar texto con enteros:

```
printf( "El valor de x es %i, ¡que bien!\n", x );
```

que quedará como:

```
El valor de x es 10, ¡que bien!
```

Como vemos %i al imprimir se sustituye por el valor de la variable.

Asignar valores a variables de tipo int

La asignación de valores es tan sencilla como:

```
x = 10;
```

También se puede dar un valor inicial a la variable cuando se define:

```
int x = 15;
```

También se pueden inicializar varias variables en una sola línea:

```
int x = 15, y = 20;
```

Hay que tener cuidado con lo siguiente:

```
int x, y = 20;
```

Podríamos pensar que x e y son igual a 20, pero no es así. La variable x está sin valor inicial y la variable 'y' tiene el valor 20.

Veamos un ejemplo para resumir todo:

```
#include <stdio.h>

int main()
        {
        int x = 10;

        prinft( "El valor inicial de x es %i\n", x );
        x = 50;
        printf( "Ahora el valor es %i\n", x );
        }
```

Cuya salida será:

```
El valor inicial de x es 10
Ahora el valor es 50
```

Importante! Si imprimimos una variable a la que no hemos dado ningún valor no obtendremos ningún error al compilar pero la variable tendrá un valor cualquiera. Prueba el ejemplo anterior quitando

```
int x = 10;
```

Puede que te imprima el valor 10 o puede que no.

El tipo Float

En este tipo de variable podemos almacenar números decimales, no sólo enteros como en los anteriores. El rango de posibles valores es del 3,4E-38 al 3,4E38.

Declaración de una variable de tipo float:

```
float numero;
```

Para imprimir valores tipo float Usamos %f.

```
float num=4060.80;
printf( "El valor de num es : %f", num );
```

Resultado:

```
El valor de num es: 4060.80
```

Si queremos escribirlo en notación exponencial usamos %e:

```
float num = 4060.80;
printf( "El valor de num es: %e", num );
```

Que da como resultado:

```
El valor de num es: 4.06080e003
```

El tipo Char

Las variables de tipo char sirven para almacenar caracteres. Los caracteres se almacenan en realidad como números del 0 al 255. Los 128 primeros (0 a 127) son el ASCII estándar. El resto es el ASCII extendido y depende del idioma y del ordenador. Consulta la tabla ASCII en el anexo.

Para declarar una variable de tipo char hacemos:

```
char letra;
```

En una variable char **sólo podemos almacenar solo una letra**, no podemos almacenar ni frases ni palabras. Eso lo veremos más adelante (strings, cadenas). Para almacenar un dato en una variable char tenemos dos posibilidades:

```
letra = 'A';
```

o

```
letra = 65;
```

En ambos casos se almacena la letra 'A' en la variable. Esto es así porque el código ASCII de la letra 'A' es el 65.

Para imprimir un char usamos el símbolo %c (c de character=caracter en inglés):

```
letra = 'A';
printf( "La letra es: %c.", letra );
```

resultado:

```
La letra es A.
```

También podemos imprimir el valor ASCII de la variable usando %i en vez de %c:

```
letra = 'A';
printf( "El número ASCII de la letra %c es: %i.", letra, letra );
```

resultado:

```
El código ASCII de la letra A es 65.
```

Como vemos la única diferencia para obtener uno u otro es el modificador (%c ó %i) que usemos.

Las variables tipo char se pueden usar (y de hecho se usan mucho) para almacenar enteros. Si necesitamos un número pequeño (entre -127 y 127) podemos usar una variable char (8bits) en vez de una int (16bits), con el consiguiente ahorro de memoria.

Todo lo demás dicho para los datos de tipo

int

se aplica también a los de tipo
char

.

Una curiosidad:

```
letra = 'A';
printf( "La letra es: %c y su valor ASCII es: %i\n", letra, letra );
letra = letra + 1;
printf( "Ahora es: %c y su valor ASCII es: %i\n", letra, letra );
```

En este ejemplo *letra* comienza con el valor 'A', que es el código ASCII 65. Al sumarle 1 pasa a tener el valor 66, que equivale a la letra 'B' (código ASCII 66). La salida de este ejemplo sería:

```
La letra es A y su valor ASCII es 65
Ahora es B y su valor ASCII es 66
```

3.-OPERADORES I.*De asignación y matemáticos.*

¿Qué es un operador?

Un operador sirve para manipular datos. Los hay de varios tipos: de asignación, de relación, lógicos, aritméticos y de manipulación de bits. En realidad los nombres tampoco importan mucho; aquí lo que queremos es aprender a programar, no aprender un montón de nombres.

[Arriba]

Operador de asignación

Este es un operador que ya hemos visto en el capítulo de Tipos de Datos. Sirve para dar un valor a una variable. Este valor puede ser un número que tecleamos directamente u otra variable:

```
a = 3;          /* Metemos un valor directamente */
```

o

```
a = b;          /* Le damos el valor de una variable */
```

Podemos dar valores a varias variables a la vez:

```
a = b = c = 10;          /* Damos a las variables a,b,c el valor 10 */
```

También podemos asignar a varias variables el valor de otra de un sólo golpe:

```
a = b = c = d;          /* a,b,c toman el valor de d */
```

Operadores aritméticos

Los operadores aritméticos son aquellos que sirven para realizar operaciones tales como suma, resta, división y multiplicación y otros más complejos.

Operador (+) : Suma

Este operador permite sumar variables:

```
#include <stdio.h>

int main()
        {
        int a = 2;
        int b = 3;
        int c;

        c = a + b;
        printf ( "Resultado = %i\n", c );
        }
```

El resultado será 5 obviamente.

Por supuesto se pueden sumar varias variables o variables más constantes:

```
#include <stdio.h>

int main()
        {
        int a = 2;
        int b = 3;
        int c = 1;
        int d;

        d = a + b + c + 4;
        printf ( "Resultado = %i\n", c );
        }
```

El resultado es 10.

Podemos utilizar este operador para incrementar el valor de una variable:

```
x = x + 5;
```

Pero existe una forma abreviada:

```
x += 5;
```

Esto suma el valor 5 al valor que tenía la variable x. Veamos un ejemplo:

```
#include <stdio.h>

int main()
        {
        int x, y;

        x = 3;
        y = 5;

        x += 2;
        printf( "x = %i\n", x );
        x += y;                 /* esto equivale a x = x + y */
        printf( "x = %i\n", x );
        }
```

Resultado:

```
x = 5
x = 10
```

Operador (++) : Incremento

Este operador equivale a sumar uno a la variable:

```
#include <stdio.h>

int main()
        {
        int x = 5;

        printf ( "Valor de x = %i\n", x );
        x++;
        printf ( "Valor de x = %i\n", x );
        }
```

Resultado:

```
Valor de x = 5
Valor de x = 6
```

Se puede poner antes o después de la variable.

Operador (-) : Resta/Negativo

Este operador tiene dos usos, uno es la resta que funciona como el operador suma y el otro es cambiar de signo.

Resta:

```
x = x - 5;
```

Para la operación resta se aplica todo lo dicho para la suma. Se puede usar también como: **x -= 5;**.

Pero también tiene el uso de cambiar de signo. Poniendolo delante de una variable o constante equivale a multiplicarla por -1.

```
#include <stdio.h>

int main()
        {
        int a, b;

        a = 1;

        b = -a;
        printf( "a = %i, b = %i\n", a, b );
        }
```

Resultado: a = 1, b = -1. No tiene mucho misterio.

Operador (--) : Decremento

Es equivalente a ++ pero en vez de incrementar disminuye el valor de la variable. Equivale a restar uno a la variable.

Operador (*) : Multiplicación

Este operador sirve para multiplicar .

Operador (/) : División

Este funciona también como los anteriores pero hay que tener cuidado. Si dividimos dos número en coma flotante (tipo float) tenemos las división con sus correspondientes decimales. Pero si dividimos dos enteros obtenemos un número entero. Es decir que si dividimos 4/3 tenemos como resultado 1. El redondeo se hace por truncamiento, simplemente se eliminan los decimales y se deja el entero.

Si dividimos dos enteros el resultado es un número entero, aunque luego lo saquemos por pantalla usando %f o %d no obtendremos la parte decimal.

Cuando dividimos dos enteros, si queremos saber cual es el resto (o módulo) usamos el operador %, que vemos más abajo.

Operador (%) : Resto

Si con el anterior operador obteníamos el módulo o cociente de una división entera con éste podemos tener el resto. No funciona más que con enteros, no vale para números float o double.

Cómo se usa:

```
#include <stdio.h>

int main()
        {
        int a, b;

        a = 18;
        b = 5;
        printf( "Resto de la división: %d \n", a % b );
        }
```

4.-INTRODUCCIÓN DATOS POR TECLADO.*(Scanf)*

El uso de scanf es muy similar al de printf con una diferencia, nos da la posibilidad de que el usuario introduzca datos en vez de mostrarlos. No nos permite mostrar texto en la pantalla, por eso si queremos mostrar un mensaje usamos un printf delante. Un ejemplo:

```
#include <stdio.h>

int main()
        {
        int num;
        printf( "Introduce un número " );
        scanf( "%i", &num );
        printf( "Has tecleado el número %i\n", num );
        return 0;
        }
```

Primero vamos a ver unas nota de estética, para hacer los programas un poco más elegantes. Parece una tontería, pero los pequeños detalles hacen que un programa gane mucho. El scanf no mueve el cursor de su posición actual, así que en nuestro ejemplo queda:

```
Introduce un número _          /* La barra horizontal indica dónde esta el cursor */
```

Esto es porque en el printf no hemos puesto al final el símbolo de salto de línea '\n'. Además hemos dejado un espacio al final de *Introduce un número:* para que así cuando tecleemos el número no salga pegado al mensaje. Si no hubiésemos dejado el espacio quedaría así al introducir el número 120 (es un ejemplo):

```
Introduce un número120
```

Bueno, esto es muy interesante pero vamos a dejarlo y vamos al grano. Veamos cómo funciona el scanf. Lo primero nos fijamos que hay una cadena entre comillas. Esta es similar a la de printf, nos sirve para indicarle al compilador qué tipo de datos estamos pidiendo. Como en este caso es un integer usamos %i. Después de la coma tenemos la variable donde almacenamos el dato, en este caso 'num'.

Fíjate que en el scanf la variable 'num' lleva delante el símbolo &, este es muy importante, sirve para indicar al compilador cual es la dirección (o posición en la memoria) de la variable. Por ahora no te preocupes por eso, ya volveremos más adelante sobre el tema.

Podemos preguntar por más de una variable a la vez en un sólo scanf, hay que poner un %i por cada variable:

```
#include <stdio.h>

int main()
        {
        int a, b, c;

        printf( "Introduce tres números: " );
        scanf( "%i %i %i", &a, &b, &c );
        printf( "Has tecleado los números %i %i %i\n", a, b, c );
        return 0;
        }
```

De esta forma cuando el usuario ejecuta el programa debe introducir los tres datos separados por un espacio.

También podemos pedir en un mismo scanf variables de distinto tipo:

```
#include <stdio.h>

int main()
        {
        int a;
        float b;

        printf( "Introduce dos números: " );
        scanf( "%i %f", &a, &b );

        printf( "Has tecleado los números %i %f\n", a, b );
        return 0;
        }
```

A cada modificador (%i, %f) le debe corresponder una variable de su mismo tipo. Es decir, al poner un %i el compilador espera que su variable correspondiente sea de tipo int. Si ponemos %f espera una variable tipo float.

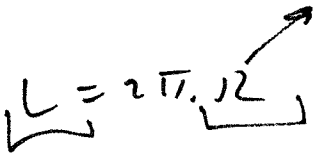
PRÁCTICA 1

Ejercicios de programación en c.

INTRODUCCIÓN ,INTRODUCCIÓN DE DATOS ,OBTENCIÓN DE DATOS ,TIPOS DE DATOS,OPERADORES MATEMÁTICOS

1.-Escribir en pantalla la palabra "HOLA".

2.- Programa que calcule la suma de dos números.



3.-Programa que calcula longitudes de circunferencia.

4. Programa que calcula la media aritmética de tres números cualesquiera.

5. Programa que calcula áreas de trapecios.

6. Programa que calcule el resto de cualquier división entera.

7. Programa que calcula el precio de un objeto sin el impuesto de valor añadido (IVA).

8. Programa que calcule el área de un triángulo.

9. Programa que calcule raíces cuadradas enteras. (Hay que cargar la librería math.h)

10.- Programa que obtenga la media geométrica de tres números.(Hay que cargar la librería math.h)

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 hubiesemos 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".

a

b

c

100

B

a*b*c

1/3

Pow

Pow (a*b*c, 1/3)

~~salida~~

cubica

float ~~*~~

float cubica;

~~printf dame precio~~ cubica

scanf (%f, &cubica);

pri