TEMA 02
TIPOS, OPERADORES Y EXPRESIONES

2.1.- Introducción.

A modo de introducción tendremos en cuenta las siguientes consideraciones:

Las variables y constantes son objetos básicos que se manipulan en un programa.

Las declaraciones indican las variables que se van a utilizar y establecen su tipo y, a veces, su valor inicial.

Los operadores especifican lo que se va a hacer con ella.

Las expresiones combinan variables y constantes para producir nuevos valores.

2.2.- Datos

Los programas funcionan con datos que contienen la información que manipularán. Estos datos se pueden dividir en constantes (valores fijos no alterables) y variables (valores que pueden cambiar).

Para utilizar un dato este debe ser antes declarado dándole un nombre que utilizaremos después para referenciarlo. En la declaración avisamos al compilador de la existencia de una variable y del nombre de la misma.

Ejemplo de declaración: `int t ;` /* t es una variable de tipo entero */

Ejemplo de declaración e inicialización: `int t = 3 ;` /* t comienza valiendo 3 */

Las variables se pueden conjugar con operadores mediante expresiones para dar lugar a nuevos valores.

Ejemplo: `b = a + 3 ;`

Existen cinco tipos de datos básicos en C:

Tipo	Descripción	Código en printf	Longitud en bytes	Rango
char	Carácter	%c	1	0 a 255
int	Entero	%d	2	-32768 a 32767
float	punto flotante	%f	4	aprox. 6 dígitos de precisión
double	punto flotante	%lf	8	aprox. 12 dígitos de precisión.
void	sin valor		0	sin valor

A la función printf se le indica el lugar y el tipo en que se escribirá una variable mediante los siguientes códigos:

char	%c	int	%d	float	% f	double	%lf

Hay veces que interesa convertir un char en un int. Se puede hacer de varias formas. Una de ellas es escribiendo -‘Φ’ a continuación del elemento char. Ejemplo

Ejemplo 1: 6 -‘Φ’
Ejemplo 2: char frase[a]=3;
frase[a] -‘Φ’ frase[a] es un entero que vale 3

Programa 004

```
#include<stdio.h>

void main ( )

{

      int i=1;

      char c ='c';

      float f = 1.0;

      double d = 1e-1;

      printf(" i=%d\n c=%c\n f=%f\n d=%lf\n",i,c,f,d);

}
```

Los datos, según la tabla anterior, pueden ser:

2.2.1- Char.

Los CARACTERES se definen con apóstrofes.

Ejemplo válido:	‘r’ , ‘3' , ‘U’.
Ejemplo inválidos:	‘Rrr’ porque son varios caracteres. e porque no está delimitado por apóstrofes y es tomado como variable. 3 porque no está delimitado por apóstrofes y es tomado como un entero.

2.2.2.- Int.

Los ENTEROS indican un número con signo sin parte decimal. Se pueden escribir:

En decimal, escribiendo el número sin empezar por 0 (a excepción del 0).
En hexadecimal, empezando el número por 0x. Ejemplo: 0xE, 0x1d, 0x8.
En octal, empezando el número por 0. Ejemplo: 02, 010.

2.2.3.- Float y Double.

Los tipos FLOAT y DOUBLE tienen parte real y parte decimal. El tipo double tiene el doble de precisión que el tipo float. Por lo demás son iguales. El número no puede empezar por e o E ya que el compilador lo interpretaría como un identificador y no como un número. Ejemplos:

1.0e9	1*(10**9)	1 000 000 000
-3E-8	-3*(10**-8)	-0.000 000 03
-10.1	-10.1	
-3.082e-2	-0.03082	

2.2.4.- Void.

El tipo VOID significa sin valor, sin tipo. Tiene varios usos:

- Uno es al declarar un argumento de función de este tipo. Lo que estamos haciendo es indicar que este parámetro no se utilizará. Ej:main (void) indica que no utilizaremos ningún parámetro.

- Otro es al declarar una función como de este tipo. La función no devolverá valor alguno. Ej: void main (int argc) no devolverá nada al sistema.

- Otro es declarar un puntero void que podrá apuntar a cualquier tipo de dato.

2.3.- Entrada y salida básica.

Analizaremos ahora las funciones de entrada y salida básica que nos proporciona el lenguaje C para poseer, al menos, unos elementos con los que podamos dar nuestros primeros pasos. Para ello observamos el siguiente programa:

Programa 005

```
#include<stdio.h>

void main ()

{

      int num1;

      printf("Introduzca un numero:");

      scanf("%d",&num1);

      printf("El n£mero %d en decimal\n",num1);

      printf("es %x en hexadecimal",num1);

}
```

La segunda sentencia declara una variable de tipo int llamada num1.

La cuarta sentencia, scanf, es una llamada a una función de entrada. Esta función recoge información del dispositivo de entrada estándar, en este caso el teclado. La función recibe dos parámetros: el primero es una cadena de formato (va entre comillas) y el segundo es un puntero a una variable donde se va almacenar ese dato. El primero indica el tipo de dato que se va almacenar en el segundo.

%d	especifica un número entero
%u	especifica un número entero, sin signo
%f	especifica un número de coma flotante
%e	especifica un número de coma flotante en formato exponencial
%g	selecciona %f o %e según el tamaño del número
%c	especifica un carácter
%s	especifica una cadena (string)
%o	especifica un número OCTAL, sin signo
% x	especifica un número HEXADECIMAL, sin signo.

2.4.- Modificadores.

A excepción del tipo void, los tipos de datos básicos pueden tener varios modificadores precediéndoles. Pueden ser modificadores de tipo o modificadores de acceso.

2.4.1.- Modificadores de tipo.

Un modificador de tipo se usa para alterar el significado del tipo base para que se ajuste más precisamente a las necesidades de cada momento

Modificador	**Descripción**	**Tipos a los que se les puede aplicar**
signed	con signo	int, char
unsigned	sin signo	int, char
long	Largo	int, char, doublé
short	Corto	int, char

2.4.2.- Modificadores de acceso.

Indican el tipo de acceso al tipo de datos que modifican.

Los de tipo constante (const) se inicializan al declarar la variable y no se pueden modificar en el programa. El siguiente ejemplo daría error:

```
const int i = 8 ;

int i = 7 ;
```

Los del tipo volatil (volatile) indica al compilador que esa variable puede ser modificada por algo externo al programa (por el sistema operativo, por un driver de dispositivo).

2.5.- Cadenas de caracteres.

Las cadenas de caracteres (string) es un tipo derivado. Esto significa que se forman a partir de los tipos básicos. Se forman delimitando los caracteres entre comillas. Las comillas no forman parte de la secuencia. Ejemplos:

```
“ Esto es una cadena “

”a “           // Tiene solo un carácter pero sigue siendo una cadena.

“\n    Esto es otra cadena \a con alarma “
```

Una cadena es sinónimo de un array unidimensional, un vector. Es una secuencia de datos que se encuentran almacenados en memoria de una forma consecutiva limitados por un carácter nulo (null). Este carácter se utiliza para señalar el final de la cadena. Por ello hemos de tener en cuenta que al definir una cadena el tamaño de la variable es 1 carácter mayor que la cadena escrita. Por ejemplo:

“abc” se almacena en memoria

a	b	C	\0

Vemos que tiene 4 caracteres. ¡Atención! El carácter nulo ‘\0' no es la cifra 0 (cuyo código ASCII es 48), sino un carácter no imprimible, cuyo código ASCII es 0.

Para dividir una cadena de caracteres en varias líneas usamos el código %s que indica a la función printf que escriba una cadena en su lugar. Ejemplo:

La sentencia printf (“Hola pepito %s”, “grillo”);

saldría Hola pepito grillo

2.6.- Operadores.

Un operador es un signo que realiza una determinada acción sobre unos operandos. El operando es el valor manipulado por el operador, puede ser un dato o el valor devuelto por la función. Se dice que el operador es binario si opera con dos operandos y es monario cuando opera con uno solo.

Hay varios tipos de operadores: aritméticos, relacionales y lógicos, a nivel de bits y especiales.

2.6.1.- Operador aritmético.

Operador	Signo	Tipo
Suma	+	Binario o unario
Resta	-	Binario o unario
Multiplicación	*	Binario
División	/	Binario
Resto	%	Binario
Incremento	+ +	Unario
Decremento	- -	Unario

Los operadores de suma y resta unarios devuelven los valores positivo y negativo del operando. El operador resto hace la misma acción que la división pero devuelve el resto de la operación en vez del cociente (Ejemplo a = 11 % 3; a = 2;). Los operadores incremento (++) y decremento (- -) modifican en 1 el valor del operando (Ejemplo a ++; a = a + 1; y ejemplo a - - ; a = a -1;).

¡ Atención ! Hay dos modalidades para cada operador incremento y decremento. Son el prefijo (antes) y el sufijo (después). Los operadores prefijo incrementan o decrementan el valor del dato antes de devolver el valor del dato que modifican. Los operadores sufijo modifican el dato después de devolver el dato.

Ejemplo

```
int x,y;                          int x,y;
x = 2;                            x = 2;
y = ++ x;                         y = x ++;
printf("x= %d y= %d",x,y);        printf("x= %d y= %d",x,y);
```

Salida x = 3 y = 3 — Modifica y devuelve

Salida x = 3 y = 2 — Devuelve y modifica

2.6.2.- Operadores relacionales y lógicos.

Se usan para generar resultados del tipo Verdadero / Falso.

Símbolo	**Nombre**	**Ejemplos**		
>	Mayor	3 > 1 4 > 4 5 > 12	Verdadero Falso Falso	1 0 0
>=	Mayor o igual	16 >= 10 2 >= 2 11 >= 12	Verdadero Verdadero Falso	1 1 0
<	Menor	5 < 4 3 < 3 2 < 3	Falso Falso Verdadero	0 0 1
<=	Menor o igual	2 <= 3 4 <= 4 5 < 11	Verdadero Verdadero Falso	1 1 0
= =	Igual	60 = = 60 12 = = 32	Verdadero Falso	1 0
! =	No igual	6 ! = 7 12 ! = 12	Verdadero Falso	1 0

Los operadores relacionales realizan una comparación entre los dos operandos situados a sus lados. Esta comparación da lugar a un valor verdadero o falso. En C cualquier valor distinto de 0 es verdadero, cualquier valor igual a 0 es falso (Cuando no se puede definir un valor determinado, para indicar verdadero se utiliza el 1).

Los operadores lógicos conectan dos expresiones entre si. Son:

Símbolo	**Nombre**	**Ejemplos**
& &	And	Verdadero si los dos operandos verdadero 5 & & 6 Verdadero 1 19 & & 0 Falso 0
| |	Or	Verdad, si es uno de ellos. 3 | | 0 Verdadero 1 2 | | 6 Verdadero 1 0 | | 0 Falso 0
!	Not	Si el operando verdadero da falso y al revés ! 5 Falso 0 ! -15 Falso 0 ! 0 Verdadero 1

2.6.3.- Operadores de bits.

Estos operadores modifican o comparan a nivel de bits.

* operador OR |

Si x = 1

0	0	0	0	0	0	0	1

z = 3

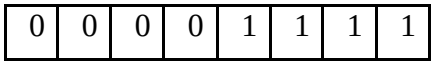
0	0	0	0	0	0	1	1

entonces x | z

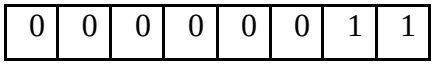
0	0	0	0	0	0	1	1

* operador XOR ^

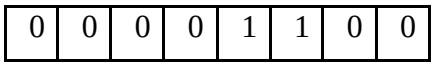
Si y = 15



z = 3

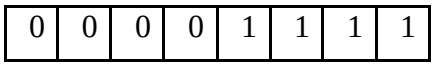


entonces y ^ z = 12

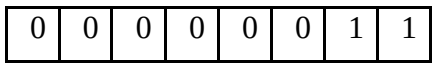


* operador AND &

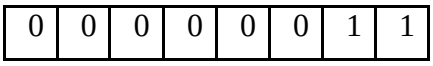
Si y = 15



z = 3



entonces y & z = 3



* operador desplazamiento hacia la derecha >>

Si	x = 1	x >> 1	x = 0
	y = 15	y >> 2	y = 3
	z = 3	z >> 1	z = 1

en general indica dividir por la n-esima potencia de dos.

* operador desplazamiento hacia la izquierda <<

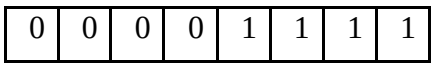
Si	x = 1	x << 1	x = 2
	y = 15	y << 2	y = 60
	z = 3	z << 1	z = 6

en general indica multiplicar por la n-esima potencia de dos.

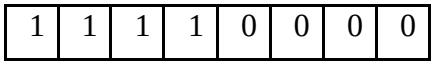
* operador complemento “

Invierte los ceros por unos y los unos por ceros

Si y = 15



Entonces “ y = 240



2.6.4.- Operadores especiales

2.6.4.1.- Operador condicional ?

Su expresión es: variable = expresión1 ? expresión 2 : expresión 3 ;
Si la expresión 1 es verdad la variable toma el valor de la expresión 2
Si la expresión 1 es falsa la variable toma el valor de la expresión 3

Ejemplo	mayor = (a > b) ? 1 : 2 ;
Si a > b	mayor = 1
Si a <= b	mayor = 2

2.6.4.2.- Operadores de dirección & y contenido *.

Operan con punteros. Un puntero es una variable que contiene dirección de memoria. Son monarios.

2.6.4.3.- Operador sizeof

Es monario y devuelve el tamaño en bytes del operando

2.6.4.4.- Operador coma,

Tiene dos usos, uno para representar una lista de elementos y otro para encadenar varias expresiones (en este último uso se evalúa de izquierda a derecha y el valor final es el de la expresión más a la derecha).

2.6.4.5.- Operadores punto . y flecha →

Se utilizan en los tipos compuestos de datos.

2.6.4.6.- Operadores paréntesis y corchetes

Los paréntesis aumentan la precedencia de la expresión que encierran, osea fuerzan la evaluación de la expresión antes que del resto de la sentencia. Los corchetes se usan para indexar arrays.

2.6.4.7.- Operadores de asignación.

Se utiliza para asignar un valor a las variables. Una sentencia de asignación es una expresión. El valor de esta expresión es el valor que se le asigna a la variable. Se evalúa de derecha a izquierda.

=	x = 5	
* =	x * = 10	x = x * 10
/ =	x / = 10	x = x / 10
% =	x % = 2	x = x % 2
+ =	x + = 5	x = x + 5
- =	x - = 5	x = x - 5
<< =	x << = 4	x = x << 4
>> =	x >> = 12	x = x >> 12
& =	x & = 1	x = x & 1
| =	x | = 4	x = x | 4
^ =	x ^ = 6	x = x ^ 6

2.7.- Precedencia de operadores.

Al evaluar una expresión se sigue un orden de izquierda a derecha evaluándose antes los operadores de mayor preferencia. Esta preferencia es:

Categoría	**Operador**
Más alta	( ) [ ] → .
Unarios	! “ + - ++ - - & * sizeof
Multiplicadores	* / %
Aditivos	+ - (binarios)
Desplazamiento	<< >>
Relacionales	< < = > > =
Igualdad	= = ! =
AND de bits	&
XOR de bits	^
OR de bits	|
AND lógico	& &
OR lógico	| |
Condicional	?
Asignamiénto	= *= /= += -= &= ^= |= << = >> =
Coma	,