TEMA 1.- FUNDAMENTOS DEL LENGUAJE C

1.1.- El lenguaje C

Historia

1967 – Martin Richards inventa el lenguaje BCPL
1969 – Aparece el S.O. UNIX (Ritchie y Thompson)
1970 - Aparece el lenguaje B escrito por Ken Thompson para el primer sistema UNIX de la DEC PDP-7, basado en BCPL.
1972 – Aparece el C y se reescribe el S.O. UNIX en C. Inicialmente muy ligado a UNIX, posteriormente se reescriben compiladores para todas las plataformas.
1984 – Bjarne Stroustrup inventa el C++.
1989 – El organismo americano de estándares, ANSI normaliza el lenguaje.
1990 – El organismo internacional de estandarización ISO, redacta otro documento de estandarización.

Características principales del C:

- Es un lenguaje de **propósito general**, es decir no está específicamente diseñado para ninguna aplicación concreta, aunque se le llama "lenguaje de programación de sistemas" debido a su utilidad para escribir compiladores y sistemas operativos, se utiliza con igual eficacia para escribir importantes programas en diversas disciplinas.

- Es un lenguaje de **nivel medio**, trata con el mismo tipo de objetos que la mayoría de las computadoras, caracteres, números y direcciones. Es capaz de descender al estrato de lenguaje de bajo nivel, así como subir al de los lenguajes de alto nivel, por eso se sitúa en una posición intermedia.

- Es un **lenguaje estructurado**, ofrece un control de flujo franco y lineal, condiciones, ciclos, anidamiento y subprogramas. En C todo se hace mediante la definición de funciones. Permite un diseño modular

- Es un lenguaje **pequeño**: tiene muy pocas instrucciones propias: operaciones aritméticas, asignación de variables, sentencias de flujo de control, etc. El resto de mecanismos de alto nivel lo proporcionan funciones especificas que se encuentran en bibliotecas del C.

- No está ligado a ningún hardware ni arquitectura concretos. Esto lo hace un lenguaje muy **portable**, capaz de escribir programas que funcionen sin cambios en una gran variedad de máquinas.

- Es un lenguaje **eficiente**. Cualquier compilador de C es capaz de generar un eficiente programa ejecutable desde un fichero fuente .c . Un programa ejecutable en C tiende a ser eficiente en espacio y tiempo; el ejecutable de un programa generado por un compilador de C tiende a requerir poco espacio de almacenamiento y a ser rápido en la velocidad de proceso.
- C mantiene la filosofía de que los programadores saben lo que están haciendo, por tanto es muy **ineficiente detectando errores**. Que un programa compile no quiere decir que sea correcto.

1.2.- Estructura general de un programa en C

El fichero que contendrá un programa fuente (o módulo de un programa) escrito en C, tendrá como extensión **.c**. Normalmente un fichero .c se estructura del siguiente modo:

- Primero debemos introducir **comentarios** iniciales sobre el programa o módulo.

- A continuación la inclusión de **bibliotecas del sistema**, los ficheros .h. En la cabecera de todo programa en C debemos incluir los 'includes' del sistema que vayamos a necesitar: estos son ficheros .h que incluyen determinadas funciones de biblioteca del C.

  Ejemplos de includes del sistema:

  alloc.h – funciones de manejo de memoria dinámica.
  ctype.h – funciones de manejo de caracteres.
  math.h – funciones matemáticas
  stdlib.h – funciones varias
  string.h – funciones de manejo de cadenas.

- **Bibliotecas de la aplicación**. En grandes aplicaciones compuestas de varios módulos se suelen crear librerías de funciones propias de la aplicación que se incluyen en aquellos módulos que las necesiten.

- **Variables globales, usadas en el módulo y declaradas en otro módulo distinto**, con la palabra reservada extern.

- **Constantes simbólicas y definiciones de macros**.

- **Definiciones de tipos**

- **Declaración de funciones del módulo**. Sólo nombre, tipo y lista de argumentos de la función.

- **Declaración de variables globales del módulo**

- **Implementación de funciones**. Se programan las funciones incluida la función principal main()

Exceptuando los dos primeros puntos, el orden del resto puede variar, pero debemos programar coherentemente y conservar el mismo orden en todos los módulos de nuestro programa. Los conceptos nombrados se irán viendo a lo largo del curso, de momento veamos un ejemplo de un programa sencillo en C, analizando cada sentencia paso a paso:

```
#include <stdio.h>

main()/* Funcion principal*/
{
int num;
num= 1;

printf("¡Hola a todos!\n");
printf("Este es el número %d \n", num);
}
```

En nuestro ejemplo la línea #include <stdio.h> indica al compilador que debe incluir información sobre la biblioteca estándar de E/S (standard input/ output)

Todo programa en C se compone de una o más funciones, que componen los módulos básicos del programa, en este caso sólo se ha definido una función, la función main(); los paréntesis que están después del nombre de la función encierran a la lista de argumentos que esa función va a recibir(información que va a necesitar), en este caso no recibe ningún argumento. Un programa en C siempre comienza su ejecución a partir de la función main, podemos definir tantas funciones como queramos con los nombres que elijamos, pero siempre debe haber una y solo una función llamada main.

Los comentarios en C se pueden incluir en cualquier parte del programa entre los símbolos /* y */. Cualquier secuencia encerrada entre esos símbolos será ignorada por el compilador. Es recomendable incluir comentarios que faciliten la comprensión del código del programa.

Las llaves {} indican el comienzo y final de una secuencia de instrucciones, una función o un conjunto de sentencias dentro de una función.

Tras la cabecera de la función, debemos incluir la declaración de variables que vayamos a usar en la función. En este caso se ha definido la variable de nombre num y de tipo entero. El punto y coma al final de cada sentencia identifica ésta como una sentencia C. La declaración de variables es necesaria para indicar al compilador cuánta memoria debe reservar.

La sentencia de asignación num = 1 da a la variable num el valor 1, almacenando en la zona de memoria reservada anteriormente el valor 1.

Las funciones se invocan al nombrarlas, seguidas de una lista de argumentos entre paréntesis; en la sentencia printf(“¡Hola a todos! \n”); se está llamando a la función printf perteneciente a la librería estandar de E/S de C, con el argumento “¡Hola a todos! \n”. Printf escribe en la salida estandar la cadena de caracteres que aparece entre comillas. La secuencia \n representa el carácter nueva línea, que hace avanzar la impresión al margen izquierdo de la siguiente línea.

La sentencia printf(“Este es el número %d \n”, num); es otra llamada a la función printf, el símbolo % indica que se va a imprimir una variable del sistema en esa posición en la cadena entre comillas, d significa que se va a introducir un número entero decimal. A continuación debemos incluir como argumento el nombre de la variable que queremos imprimir, num en este caso.

El resultado del programa sería:

¡Hola a todos!
Este es el numero 1

Ejercicios:

1.- Ejecuta el programa del ejemplo propuesto y observa su resultado. Realiza modificaciones sobre el mismo, eliminando distintas sentencias o instrucciones y observa los errores que resultan. Por ejemplo. ¿qué sucede si cambiamos num en la declaración por NUM? ¿Qué sucede si omitimos primera línea del programa? ¿Qué resultado daría el programa si omitimos el primer \n?

2- ¿Qué resultado daría el siguiente programa?

```
#include <stdio.h>
main()
{
printf(“hola”);
printf(“a todos”);
printf(“\n”);
}
```

Elementos del lenguaje

Para escribir un programa en C se usa un alfabeto o conjunto de caracteres. Este conjunto incluye las letras mayúsculas y minúsculas del alfabeto Inglés (en C se distingue entre ambas), los diez dígitos decimales del sistema de numeración arábigo y el carácter de subrayado. Los espacios en blanco se usan para separar los diferentes elementos del programa. Los espacios en blanco incluyen también el tabulador, el salto de línea y otros caracteres de control. El alfabeto también incluye los signos de puntuación.

Con este alfabeto escribiremos todos aquellos elementos de los que se compone un programa en C. Como palabras reservadas, identificadores, comentarios, constantes, etc.

Las **palabras reservadas** son aquellas que tienen un significado especial para el compilador y que no se pueden redefinir ni usar de otro modo. El C al ser un lenguaje pequeño tiene tan sólo 32 palabras reservadas que son las siguientes:

auto	extern	sizeof	const
long	unsigned	struct	signed
char	goto	enum	do
short	while	union	else
default	int	for	typedef
static	case	volatile	float
double	return	break	void
switch	continue	register	if

1.3.- Datos y tipos de datos

Todos los programas, para realizar el proceso para el que fueron realizados, necesitan trabajar con datos. Estos datos básicos que manipulan los programas son variables y constantes. Aquellos datos cuyo valor está predefinido antes de la ejecución de un programa y mantienen sus valores inalterados durante ésta, son llamados constantes. Aquellos que pueden cambiar su valor durante la ejecución de un programa, son llamados variables. El tipo de un dato determina el conjunto de valores que puede tener, la cantidad de memoria que se va a reservar para almacenar ese dato y las operaciones se pueden realizar sobre él.

Tipos de datos básicos

Los tipos más elementales del C son:

- **char** : tipo carácter. Se usan normalmente para almacenar cadenas de caracteres. Su tamaño normalmente es de un byte. Realmente lo que almacena es el número entero que corresponde a ese carácter en el código de caracteres por tanto se pueden realizar operaciones con ellos.
- **int** : tipo entero, su tamaño normalmente es de dos bytes (puede ser más grande, depende de la máquina, normalmente se corresponde con el tamaño del bus de datos del ordenador.)
- **float** : tipo punto flotante de precisión normal. Su tamaño generalmente es de cuatro bytes
- **double** : tipo punto flotante de doble precisión. Tiene el doble del tamaño de float. Es el más usado para números reales.

Existen modificadores de los tipos básicos que se anteponen a la definición del tipo en la declaración, por el momento sólo veremos los modificadores de tamaño y signo.

Modificadores de tamaño: Su correcto funcionamiento depende mucho del hardware en el que estemos ejecutando la aplicación. (No se usan para el tipo char, para enteros podemos omitir la palabra int cuando los usemos)

- **long**: obtiene una variable el doble de grande de la del tipo básico.
- **short** : obtiene una variable con la mitad del tamaño de la que se define. (no suele usarse)

Modificadores de signo: Indican explícitamente si la variable es con signo o sin signo. Aplicables a los tipos char o int (Para los caracteres la opción por defecto depende del hardware concreto, los enteros son con signo por defecto).

- **signed**: con signo
- **unsigned**: sin signo

Modificadores de tipo de almacenamiento : Indican al compilador cómo debe almacenar la variable que le sigue. El especificador de almacenamiento precede al resto de la declaración de variable.

- extern
- static
- register
- auto

El uso de estos modificadores se verá en el tema 4 en el apartado de ámbito y persistencia de las variables

Veamos como ejemplo los tamaños de los tipos básicos para un compilador típico de un IBM PC.

Identificador de tipo	**Significado**	**Rango de valores**
char	carácter	-128 a 127
int	entero	-32.768 a 32.767
short	entero corto	-32.768 a 32.767
long	entero largo	-2.147.483.648 a 2.147.483.647
unsigned char	carácter sin signo	0 a 255
unsigned	entero sin signo	0 a 65.535
unsigned short	entero corto sin signo	0 a 65.535
unsigned long	entero largo sin signo	0 a 4.294.967.295
float	real (coma flotante)	3,4E +/-38 (7 digitos)
double	real doble precisión	1,7E +/- 308 (15 dígitos)
long double	real doble prec. largo	1,7E +/- 4932 (15 dígitos)

Identificadores

Un identificador es un nombre asociado a un objeto de programa, que puede ser una variable, constante, tipo de dato, etc

Un identificador se puede formar con los caracteres del alfabeto ingles, los dígitos del 0 al 9 y el carácter de subrayado, no puede contener ningún carácter más y debe comenzar siempre por una letra. La longitud máxima que reconoce el lenguaje es de 31 caracteres. Los identificadores deben estar relacionados con el objeto que identifican, es decir deben ser descriptivos, sin ser excesivamente largos.

Variables

Todas las variables deben ser declaradas antes de su uso. Una **declaración** informa al compilador del nombre identificador asociado a cada variable así como de su tipo. Se puede pensar en una variable como en una posición específica de memoria, reservada para un tipo específico de datos y con un nombre para referenciarla fácilmente. Esencialmente se usan variables para permitir que la misma posición de memoria pueda tener diferentes valores del mismo tipo en instantes de tiempo distintos.

La plantilla general de declaración de una variables es:

[Modificador_ almacenam.] [Modificador_ signo] [Modificador_ tamaño] tipo nombre;

Ejemplos:

```
int num;
```

Se puede combinar una declaración de variable con el operador de asignación, dando un valor inicial a la variable en el momento de su declaración:

```
int num = 2;
char c = ‘x’;
```

Se pueden declarar varias variables del mismo tipo en la misma sentencia, separando los diferentes nombres por comas:

```
int n1,n2,n3;
```

Las variables se clasifican según su ámbito de visibilidad en:

- Variables globales. Son declaradas fuera de cualquier función de un programa, al mismo nivel que éstas. Esto quiere decir que podrá ser utilizada por cualquier función definida en nuestro programa. Debemos usar estas variables para valores grandes que necesiten mucha memoria.

- Variables locales. Se declaran dentro del cuerpo de la función en la que vayamos a usar esta variable(justo tras la llave de apertura del bloque de sentencias). De este modo sólo puede ser usada por esa función. La variable desaparece cuando termina de ejecutarse la función en la que está definida. (Veremos estos conceptos más ampliamente el tema 4 en el apartado *ámbito y persistencia de las variables*)

Constantes

Una constante es un dato que escribimos en el propio programa. Normalmente no debemos especificar el tipo de dato de la constante si éste se deduce de su propia definición, pero en algunos casos es necesario. Si queremos usar una constante *long* pondremos al final del valor la letra L. Constantes flotantes como 2.5 serán *double* a menos que al final del valor se añada la letra F que indica que es un *float* o l que indicará que es un *long double*. Podemos asociar identificadores a constantes simbólicas, de modo que podamos usar este valor en cualquier parte del programa mediante el uso de su identificador. Se definen utilizando la directiva define, las sentencias de definición de constantes (y macros) no llevan punto y coma al final de la instrucción.

Ejemplo:

```
#define MAX 1000
```

Otra forma de definir constantes en C ISO es añadir a la declaración de una variable la directiva const delante, que indica que la variable no será modificada.

Ejemplo:

```
const double e= 2.71828182845905;
```

1.4.- Operaciones básicas y expresiones

Un **operador** es un símbolo que indica al compilador que lleve a cabo ciertas manipulaciones matemáticas o lógicas.

Una **expresión** es un conjunto de símbolos (operadores y operandos) que produce un valor.

Operador de asignación.

El signo igual es el operador de asignación. El operando de la izquierda debe ser una variable mientras que el de la derecha puede ser una constante, otra variable o bien una expresión. Devuelve el valor que ha asignado.

Ejemplo:

```
numero = 7;
```

Operadores aritméticos

Los operadores aritméticos son suma, resta, multiplicación, división y módulo. El módulo calcula el resto de una división y sólo es aplicable a enteros. La división es la división entera.

Operador	**Significado**	**Ejemplo**
+	Suma	res = 3 + 5 (res ->8)
-	Resta	res = 5 – 3 (res ->2)
*	Multiplicación	res = 5 * 3 (res->15)
/	División entera	res = 10/ 3 (res ->3)
%	Módulo	res = 3 % 2 (res ->1)

Operadores de asignación compuestos

Combinan el operador de asignación simple con otro operador. Por ejemplo:
i = i + 1; en este caso los identificadores i tienen significados distintos. El izquierdo se refiere a la posición de memoria correspondiente a i. Con el de la derecha al valor almacenado en esa posición. Por tanto la instrucción significa que a la posición de memoria denominada i se le asignará un nuevo valor que será el resultado de sumarle 1 al valor actual de la variable i.

Ejemplo: Supongamos que i vale 10 y hacemos

i = i + 5;

hará que el nuevo valor de i sea 15

Esto puede acortarse como:
i += 5;
Válido para todos los operadores aritméticos vistos en el apartado anterior.

Operadores de incremento y decremento

Existen en C dos operadores poco comunes para incrementar y decrementar variables. El operador de aumento ++ añade 1 a su operando, mientras que el operador - - resta 1 a su operando. Por tanto:

x= x+ 1; es equivalente a ++x;
y = y-1; es equivalente a - -y;

Estos operadores pueden usarse como prefijos(antes de la variable ++n) o cómo postfijos (después de la variable n++). En ambos casos el efecto es incrementar el valor de

la variable n. Pero la expresión ++n incrementa n antes de usarla , mientras que n ++ la incrementa después de usarla.

Ejemplos: Supongamos i= 3 y j= 15

Expresión	**Resultado**
i= ++j	j =16 y i=16
i = j++	i= 15 y j=16
j=++i + 5	i= 4 y j =9
j= i++ + 5	j= 8 y i= 4

Operadores relacionales y lógicos

En el término "operador relacional" la palabra relación entre unos valores y otros. En el término "operador lógico" la palabra lógico se refiere a las formas en que esas relaciones pueden conectarse entre sí siguiendo las reglas de la lógica formal.

La clave de los conceptos de operadores relacionales y lógicos es la idea de *verdadero* y *falso*. En C, *verdadero* es cualquier valor distinto de 0, *falso* es 0. Las expresiones que utilizan operadores relacionales y lógicos devuelven el valor 1 en caso de *verdadero* y 0 en caso de *falso*.

Operador	**Acción**
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
= =	Igual que (*)
!=	Distinto que

Operadores relacionales

Operador	**Acción**
&&	Y (and)
||	O (or)
!	NO (not)

Operadores lógicos

(*) El operador de asignación y el operador de igualdad son distintos y tienen distinto funcionamiento. El operador de igualdad compara los valores de los operandos a ambos lados del operador, mientras que el operador de asignación, visto anteriormente, asigna a la posición de memoria de la variable de la parte izquierda de la expresión el valor de la parte derecha de la misma.

Tanto los operadores de relación como los lógicos tienen un nivel de precedencia menor que los operadores aritméticos. Esto significa que una expresión como 10 > 1+ 12 se evalúa como si fuera 10 > (1+ 12), el resultado es falso.

Es posible combinar varios operadores en una expresión.

Ejemplo: 10 > 5 && ! (10 < 9) || 4 < = 4 que es verdadero

El operador sizeof

El operador **sizeof** es un operador monario que devuelve la longitud en bytes de la variable o del especificador de tipo, entre paréntesis, al que precede.

Ejemplo:

float f;

printf (“%f, sizeof f);

printf(“%d, sizeof(int));

Precedencia de operadores y orden de evaluación

La tabla siguiente resume las reglas de precedencia y evaluación de todos los operadores. Los operadores que están en la misma línea tienen la misma precedencia:
Los renglones están en orden de precedencia decreciente. Los operadores – ‘>’ y ‘.’ son utilizados para tener acceso a miembros de estructuras, los veremos más tarde así como el operador sizeof (tamaño de un objeto) y & (dirección de un objeto).

Operadores	**Asociatividad**
( ) [ ] ->	De izquierda a derecha
! ++ -- + - * & (tipo) sizeof	De derecha a izquierda
* / %	De izquierda a derecha
+ - (binarios)	De izquierda a derecha
< <= > >=	De izquierda a derecha
= = !=	De izquierda a derecha
&&	De izquierda a derecha
||	De izquierda a derecha
?:	De derecha a izquierda
= += -= *= /= %=	De derecha a izquierda
,	De izquierda a derecha

Como muchos lenguajes, C no especifica el orden en que se evalúan los operandos de una expresión, ni el orden en que se evalúan los argumentos de una función. Éste depende del compilador que se está usando. Una misma expresión puede proporcionar diferentes resultados con compiladores distintos**.** Las llamadas a funciones, proposiciones de asignación anidadas y los operadores de incremento y decremento provocan instrucciones con **efectos laterales**. Alguna variable es modificada como producto de la evaluación de una expresión. En una expresión que involucra efectos laterales, el resultado puede depender del orden en que se evalúen las variables involucradas en la expresión. Debemos evitar el uso de este tipo de instrucciones.

Ejemplo: La siguientes instrucción daría resultados diferentes con distintos compiladores dependiendo de si i es incrementada antes o después de llamar a la función potencia.

printf (“%d %d\n”, ++i, potencia(2, i))

Conversiones de tipos

Cuando un operador tiene operandos de tipos diferentes, estos se convierten a un tipo único. El compilador de C convierte todos los operandos al tipo del operando de mayor rango, esto se hace operación a operación. El rango de los tipos es:

Rango inferior <= char, int,long,float,double => Rango Superior

Ejemplo:

```
char ch;
int i;
float f;
double d;

result = (ch / i) +  ( f * d) – (f + i);
```

Primero, el carácter ch se convierte a entero, float se convierte a double. Después el resultado de (ch/i) se convierte a double debido a que el otro operando f*d es double. El resultado final es double porque ambos operandos lo son.

En las asignaciones el tipo del valor del lado derecho de la asignación se convierte al tipo de la variable del lado izquierdo de la asignación.

Moldes

Es posible forzar a que una expresión sea de un tipo determinado utilizando una construcción denominada molde. La forma general de un molde es:

```
(tipo) expresión
```

donde tipo es uno de los tipos estándar de C o uno definido por el usuario. Por ejemplo:

Dada la variable

```
int i;
```

Suponemos que queremos realizar la división por 2 de esa variable y queremos conocer la parte fraccionaria, la única manera de hacerlo sería:

```
(float) ½
```

El molde es un operador monario y tiene la misma precedencia que cualquier otro operador monario. Debemos tener cuidado al realizar conversiones de tipos con un rango mayor a tipos con un rango menor por la pérdida de información que puede producirse.

1.5 .- Entrada y salida básicas

Todas las funciones de entrada y salida pertenecen a una biblioteca estándar de C que es *stdio.h*, por tanto debe añadirse la instrucción *#include<stdio.h>* en la cabecera de cualquier módulo o programa que vaya a hacer uso de alguna de las funciones que veremos a continuación

Entrada y salida de caracteres

La biblioteca estándar proporciona varias funciones para leer y escribir caracteres, las más simples son getchar y putchar.

int getchar (void): No recibe ningún argumento, devuelve un entero. Lee un carácter del teclado dando un eco por pantalla.

Ejemplo:

c= getchar(); Almacenará en c el carácter que introduzcamos por teclado.

int putchar (int c) : Recibe un entero, devuelve también un entero. Escribe un carácter por pantalla.

Ejemplo:

putchar (c); Escribe el contenido de la variable entera c, cómo un carácter.

Entrada y salida formateada

La función printf (), salida formateada

Se define como: **int printf (cadena de caracteres, variables o valores)**: Devuelve un entero(nº de caracteres que escribe o EOF en caso de error), recibe un número de argumentos variable: cadenas de caracteres con especificadores de formato y variables o valores concretos. Se usa para escribir información por pantalla con un formato determinado.

Su primer argumento es una cadena de caracteres entre comilla, donde se especifican los caracteres directamente imprimibles así como los especificadores de formato que informan a la función sobre la forma de convertir, dar forma e imprimir los argumentos. Están precedidos del símbolo % y se enumeran en la siguiente tabla:

Carácter	Argumento	Salida
d	entero (*)	Entero con signo en base decimal
i	entero	Entero con signo en base decimal
o	entero (*)	Entero con signo en base octal
u	entero (*)	Entero con signo en base decimal
x	entero (*)	Entero con signo en base hexadecimal - minúsculas
x	real	Entero con signo en base hexadecimal – mayúsculas
f	real (*)	Número real con signo
e	real (*)	Número real con signo usando notación e
e	real	Número real con signo usando notación E
g	real	Número real con signo formato e o f, tamaño corto
g	real	Número real con signo formato E o f, tamaño corto
c	carácter (*)	Un carácter individual
s	cadena de caracteres	Cadenas de caracteres
%	ninguno	Imprime el símbolo %

- e,f,g precedidas de l indican que es un double
- d,i,o,u,x precedidas de l indican que es un long

Ejemplo :

```
#include <stdio.h>

main()
{
printf ("El valor 92 con el tipo de campo d  es %d. \n", 92);
printf ("El valor 92 con el tipo de campo i  es %i \n", 92);
printf ("El valor 92 con el tipo de campo u  es %u \n", 92);
printf ("El valor 92 con el tipo de campo o  es %o \n", 92);
printf ("El valor 92 con el tipo de campo x  es %x \n", 92);
printf ("El valor 92 con el tipo de campo X  es %X \n", 92);
printf ("El valor 92 con el tipo de campo f  es %f \n", 92.0);
printf ("El valor 92 con el tipo de campo e  es %e \n", 92.0);
printf ("El valor 92 con el tipo de campo E  es %E \n", 92);
printf ("El valor 92 con el tipo de campo g  es %g \n", 92);
printf ("El valor 92 con el tipo de campo G  es %G \n", 92);
printf ("El valor 9 con el tipo de campo c  es %c \n", 9);
printf ("El valor 92 con el tipo de campo s  es %s \n", "92");
}
```

La salida será:

El valor 92 con el tipo de campo d es 92
El valor 92 con el tipo de campo i es 92
El valor 92 con el tipo de campo u es 92
El valor 92 con el tipo de campo o es 134
El valor 92 con el tipo de campo x es 5c
El valor 92 con el tipo de campo X es 5c
El valor 92 con el tipo de campo f es 92.000000
El valor 92 con el tipo de campo e es 9.20000e+01
El valor 92 con el tipo de campo E es 9.20000E+01
El valor 92 con el tipo de campo g es 92
El valor 92 con el tipo de campo G es 92
El valor 92 con el tipo de campo c es 9
El valor 92 con el tipo de campo s es 92

Entre el símbolo % y la el carácter especificador de formato podemos incluir

- El símbolo – indicando que los valores se justificarán a la izquierda
- Un número indicando el tamaño en pantalla de la salida
- Dos números separados por un punto para tipos reales - El primer nº indica el tamaño en pantalla de la salida y el segundo el nº de decimales que se imprimirán.

Ejercicio: Realizar pruebas añadiendo estos caracteres a las sentencias del ejemplo anterior

Tenemos que tener en cuenta que el número de especificadores de formato debe corresponderse con el número y tipo de los argumentos que tiene la función a continuación.

El carácter \n escribe un retorno de línea. Veamos en la siguiente tabla algunos de estos caracteres especiales llamados secuencias de escape:

Código	**Significado**
\b	Retrocede un carácter
\f	Salto de página
\n	Salto de línea
\r	Retorno de carro
\t	Tabulación horizontal
\”	Comillas dobles
\’	Comillas simples
\\	Barra invertida
\v	Tabulación vertical

La función scanf(), entrada formateada

Permite a los programas leer la entrada del usuario con formato desde el teclado. Podemos pensar en esta función cómo la opuesta a printf. Devuelve el número de variables o campos leídos o EOF en caso de error. Tiene un formato parecido al de printf, sólo que en éste caso, entre comillas sólo se incluyen los especificadores de formato del carácter que se espera recibir. (Son los especificados en la tabla anterior con (*)).

Ejemplo:

```
#include <stdio.h>
main()
{
float valor;

printf (“Introduzca un numero =>”);
scanf (“ %f”, &valor);
printf(“El valor es => %f “, valor);
}
```

La salida, suponiendo que el usuario introduce cuando sea requerido el número 5 será:

```
Introduzca un número => 5
El valor es 5.000000
```

El argumento “%f” significa que espera recibir un valor tipo real, el siguiente argumento le indica al programa dónde debe almacenar el valor que se reciba, para ello se pone el símbolo & delante del identificador de la variable que lo almacenará, de este modo se le pasa la dirección de memoria de la variable.

La función *scanf()* puede aceptar más de una entrada con una sola sentencia:

```
scanf (“%f%d%c” , &numero1, &numero2, &caracter)
```

El usuario podría teclear: 52.7 18 t por ejemplo.