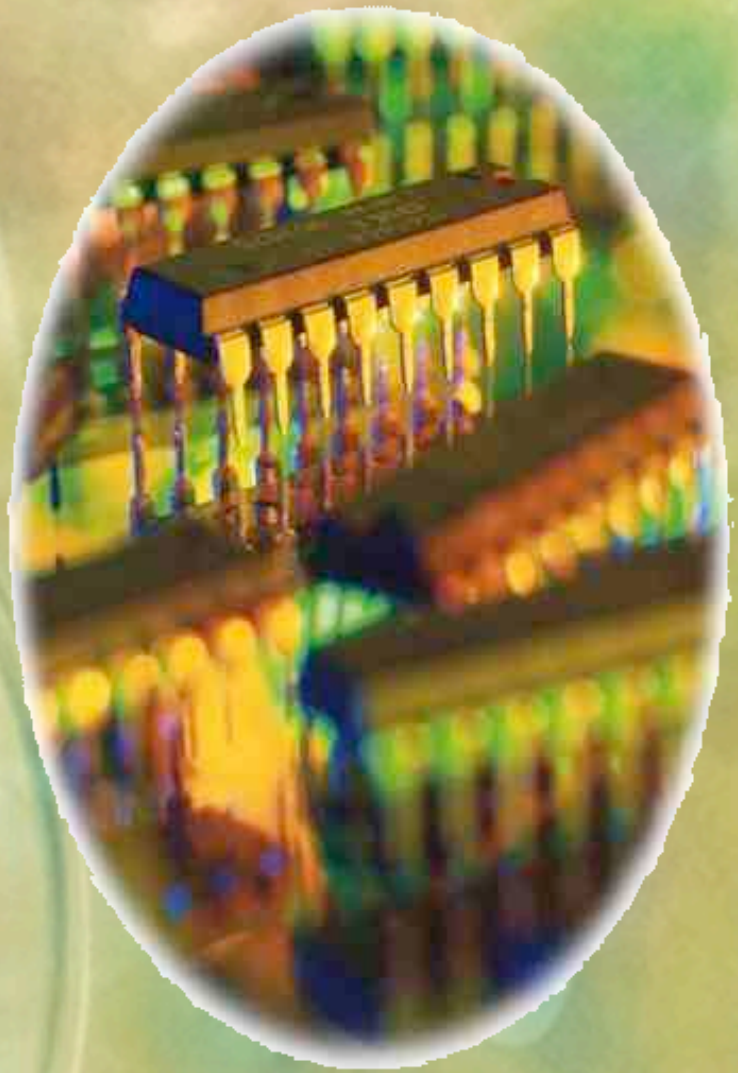
TEMA 2

El lenguaje de progamación C



DECSAI
Department of Computer Sciences and Artificial Intelligence
University of Granada

Dept. Ciencias de la Computación e I.A.

Universidad de Granada

Primeros Program

Directiva del precompilador para incluir funciones de E/S

Comentarios. Los ignora el compilador.
/* Comentario en varias líneas */

```
/* Primer    grama en C*/
/* Escr   e en pantalla un mensaje*/

#include <stdio.h>

int main(){
   printf ("Bienvenido a C\n");
   return 0;
}
```

Función **main** incluye entre llaves las instrucciones del programa

Todas las instrucciones terminan con ;

printf envia a pantalla mensajes y datos

Produce un salto de línea

```
Bienvenido a C
```

Salida del programa por **pantalla**

Primeros Programas en C

```
/*Programa que lee dos números enteros por teclado
  y muestra por pantalla su suma*/
#include <stdio.h>

int main(){
  int variable1;
  int variable2;
  int suma;

  printf("Introducir primer dato: ");
  scanf("%d", &variable1);
  printf("\nIntroducir segundo dato: ");
  scanf("%d", &variable2);
  suma = variable1 + variable2;
  printf("\nLa suma es: %d \n", suma);

  return 0;
}
```

Declaración de variables. Hay que especificar el nombre y tipo de todas las variables usadas

scanf capta valores para los datos desde teclado

El valor de variable1 se suma al valor de variable2 y el resultado se asigna a suma

Introducir primer dato: 45
Introducir segundo dato: 72
La suma es: 117

Tipos de Datos

Entendemos por "datos" a los diferentes objetos de información con los que un programa trabaja. Más formalmente, es la representación formal de hechos, conceptos o instrucciones adecuada para su comunicación , interpretación y procesamiento por seres humanos o medios automáticos.

Todos los datos tienen un ***tipo asociado*** que es la especificación de un dominio (rango de valores) y de un conjunto válido de operaciones.

Ejemplos:

- El DNI de una persona es un número entero
- El Ph de una solución es un número con decimales
- El título de un libro es una lista de letras

Tipos de datos usuales: numéricos, caracteres, lógicos

Variables

- Es equivalente a una "variable" en el contexto matemático.
- Una variable no es más que un nombre simbólico que identifica una dirección de memoria
- Los identificadores son los nombres que utilizamos para referirnos a las variables.
- Al declararla, hay que definir su tipo: la variable sólo admitirá valores del tipo especificado.
- Las variables tienen que declararse **antes** de su uso siguiendo el esquema:

```
tipo identificador
tipo lista_de_identificadores
```

- **Ejemplos:**

```
int velocidad, tiempo;
double Ph;
char inicial;
```

Identificadores en C

Todos los objetos de un programa (variables, constantes, funciones, etc) deben tener un nombre o ***identificador.*** Para asignar un ***identificador*** deben respetarse las siguientes reglas:

- El primer símbolo del identificador será un carácter alfabético o un guión de subrayado. (a,..z,A..z,'_').
- Las mayúsculas y las minúsculas se consideran diferentes.
- El guión de subrayado ´_´se interpreta como una letra.
- Los identificadores no pueden coincidir con las palabras reservadas de C:

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

Tipos de Datos

Tipos de Datos Básicos

`char`	Caracteres (‘u’,’%’,’_’,’A’…..)
`int`	Números enteros (234, -4, 0456, 0x23D4, ….)
`float`	Números en coma flotante (0.07,123e45,23E-5,
`double`	Números en coma flotante de doble precisión
`void`	Tipo nulo
Punteros	Direcciones de memoria (*char, *int, *float, *double, *void)

Modificadores

- Tamaño del dato

  `short` (`int` por defecto)

  `long` (`int` por defecto)

- Signo

  `signed` (con signo)

  `unsigned` (sin signo)

- Modo de Almacenamiento: `register`, `auto`, `static`, `extern`

Codificación de los datos en el ordenador

- En el interior del ordenador, los datos se representan en binario. El sistema binario sólo emplea dos símbolos: 0 y 1
  - Un bit nos permite representar dos símbolos distintos:0 y 1
  - Dos bits permiten codificar 4 símbolos: 00, 01, 10 y 11
  - Tres bits permiten codificar 8 símbolos: 000, 001, 010, 011, 100, 101, 110 y 111
- En general, con N bits podemos codificar 2^N valores diferentes

Tipo de dato	Espacio en memoria	Mínimo (valor absoluto)	Máximo (valor absoluto)	Dígitos significativos
`float`	32 bits	1.2×10^{-38}	3.4×10^{38}	6
`double`	64 bits	2.2×10^{-308}	1.8×10^{308}	15
`long double`	80 bits	3.4×10^{-4932}	1.2×10^{4932}	18

Tipo de dato	Espacio en memoria	Valor Mínimo	Valor Máximo
`char`	8 bits	-128	127
`unsigned char`		0	255
`short`	16 bits	-32768	32767
`unsigned short`		0	65535
`long`	32 bits	-2147483648	2147483647
`unsigned long`		0	4294967295

Tipo de dato	Espacio en memoria	Codificación
`char`	8 bits	ASCII

Variables: dar valores

Antes de usar una variable, es necesario darles algún valor. Es decir ***"inicializarla".***

La forma más directa es mediante una sentencia de asignación. Supongamos que hemos declarado:

int nro;

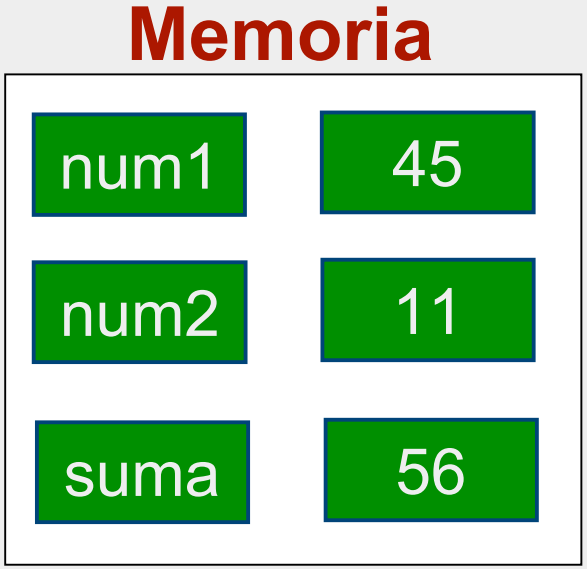
Ahora podemos hacer una asignación:

Variables: dar valores

Cuando se ejecuta una operación de asignación, primero se evalúa la expresión del lado derecho y luego se almacena el valor resultante en la variable indicada en el lado izquierdo.

```
int num1, num2, suma;
num1 = 45;
num2 = 11;
suma = num1 + num2;
```

```
suma =  45  +  11;
suma = 56;
```



Regla de Asignación

- Una variable en el lado derecho de una sentencia de asignación debe tener un valor antes de que la sentencia de asignación se ejecute. Hasta que un programa le da un valor a una variable, esa variable no tiene valor.

Ejemplo:

```
int x,y;
y = x + 1;
```

ERROR LÓGICO: *la variable x no tiene ningún valor. El valor que toma y es impredecible!!!*

- En la izquierda de una sentencia de asignación solo pueden existir variables. La siguiente expresión no es valida:

```
int Valor_Neto, Tasas;

Valor_Neto - Tasas =  34015;
```

Regla de Asignación

La operación de asignación es una operación destructiva: el valor almacenado en una variable se pierde o se destruye y se sustituye por el nuevo valor en la sentencia de asignación.

Literales

Un literal es una especificación de un valor concreto de un tipo de dato. Se utilizan en el lado derecho de una asignación.

Ejemplos

Valor	Tipo Literal
2	entero
45.2	real
'c'	caracter: se escribe entre ' '
"Hola"	cadena de caracteres: se escribe entre " "
3.49e4	equivale a 3.49×10^{4} (34900.0)
5.89e-6	equivale a 5.89×10^{-6} (0.00000589)

Constantes

Una constante hace referencia a un valor que no puede modificarse.

En C se definen usando la directiva `#define`

```
#define <IDENTIFICADOR> <valor>;
```

Por ejemplo:

```
#define PI 3.1415;
#define SALARIO_BASE 1000;
```

Suelen usarse identificadores (nombres) sólo con mayúsculas para diferenciarlos de las variables. Si se usan varias palabras se separan con el carácter de subrayado.

Ventajas de la declaración de constantes:

- *Proporcionan información*
- *Imposibilidad de cambiarlo por error (PI)*

Expresiones y Sentencias

- Expresión: Construcción que se evalúa para devolver un valor.

```
x=x+1;
x=x*2+37;
x=sumatoria(serie)/10;
```

- Sentencia: En C, todas las sentecias terminan con un punto y coma.

Ejemplo

```
/*Calculo del area de una
  circunferencia*/
#include <stdio.h>
#define PI = 3.141516;

int main(){
double radio,rta;
radio = 8.73;
rta = PI * radio * radio;

printf("El valor del area es:%lf \n",
  rta);
return(0);
}
```

Declaración de Constantes

Declaración de Variables

Un literal en una asignación

Una expresión en una asignación

Compatibilidad de Tipos

Hay que asignar a las variables valores de su mismo tipo.

Generalmente, los valores de tipo *int* pueden almacenarse en variables de tipo *double.*

```
int unEntero;
double unReal;
unEntero = 18;
unReal = unEntero;
unReal = 98;
```

unReal vale 18.0

unReal vale 98.0

La asignacion de valores de tipo *double* a variables de tipo *int* provoca la pérdida de la parte decimal.

```
int unEntero;
unEntero = 2.456;
```

Conversión de tipos (Castings)

- En determinadas ocasiones nos interesa convertir el tipo de dato en otro para poder operar con él
- C permite la conversión entre datos de tipo numérico, así como usar trabajar con caracteres como si fueran números enteros.
- La conversión de un tipo a otro con menos bits es automática (implícita)

  Long double>double>float>unsigned long>long>unsigned

  int> int> unsigned short>short>char
- La conversión de un tipo a otro con más bits debe hacerse de forma explícita con castings.

  (tipo) expresion

  Ejemplo:

  ```
  Double x=3.7;
  int entero=(int)(x+0.5);
  ```

Operadores aritméticos

Se pueden utilizar con valores de tipo entero, de tipo real o uno de cada tipo.

- *Si ambos son int, el resultado tendrá tipo int*
- *Si uno de ellos es double, el resultado tendrá tipo double*

Operadores

- + Suma
- - Resta
- * Multiplicación
- / Division
- % Modulo (sólo enteros)

Operadores aritméticos

División

- La división entre operandos enteros descarta la parte decimal del resultado:
  - 10 / 3 → 3 (no 3.33333)
  - 5 / 2 → 2 (no 2.5)
- La división usando al menos un operando de tipo double se comporta de la manera esperada.
  - 7.0 / 5 → 1.4
  - 7 / 5.0 → 1.4
  - 7.0 / 5.0 → 1.4

Módulo. Resto de la división entera

- 7 % 5 → 2

Ejercicios

1) Cuales de los siguientes son nombres válidos de variables?

x_1 x1 12342 _hh %valor prog.c

2) ¿ Que hace el siguiente ejemplo?

```
int valor;
valor = 0;
valor = valor + 1;
imprimir valor;
```

3) Convierta las siguientes fórmulas a expresiones en C

3x 3x+y $\frac{x+y}{7}$ $\frac{x+y}{z+2}$

4) ¿Cuál es la salida del siguiente programa?

```
char a,b,c;
a='b';
b='c';
c = a;
imprimir a,b,c,'c';
```

Datos de tipo booleano

- En C no existe explícitamente un tipo de dato booleano para representar algo que pueda ser verdadero (V) o falso (F). True(T)/False(F)
- Cualquier valor distinto de cero se considera verdadero.
- Por convección:
  - Se usa el valor 1 para representar algo verdadero.
  - Se usa el valor 0 para representar algo falso.

Operadores lógicos

- Son operadores binarios
  - AND (&&)
  - OR (||)
  - XOR (^)
  - NOT (!) (este es un operador unario)
- Devuelven true o false

AND	True	False
True	True	False
False	False	False

OR	True	False
True	True	True
False	True	False

XOR	True	False
True	False	True
False	True	False

NOT	True	False
	False	True

Operadores relacionales

Son los operadores habituales de comparación de números. El resultado es de tipo lógico (0 falso, en otro caso, verdadero).

(4 < 5) ⇒ true

(4 > 5) ⇒ false

((x >= 1) && (x <= 10)) ⇒ ¿x ∈ [1, 10]?

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	>	`x > y`	`x` is greater than `y`
<	<	`x < y`	`x` is less than `y`
≥	>=	`x >= y`	`x` is greater than or equal to `y`
≤	<=	`x <= y`	`x` is less than or equal to `y`
Equality operators			
=	==	`x == y`	`x` is equal to `y`
≠	!=	`x != y`	`x` is not equal to `y`

Ejercicio

Dadas las variables *count = 0, limit = 10*, *x=2, y=7,* calcule el valor de las siguientes expresiones booleanas

```
(count == 0) && (limit < 20)
(limit > 20) || (count < 5)
!(count == 12)
(count == 1) && (x < y)
!(((count < 10) || ( x < y )) && (count >= 0))
((count>5) && (y==7)) || ((count<=0) && (limit==5*x))
!((limit != 10) && (z > y))
```

Funciones matemáticas predefinidas

- Llevan a cabo cálculos matemáticos y devuelven un valor (hay que incluir: **#include <math.h>)**
- La forma para invocarlos es:

NombreFun(argumento1, argumento2, …)

Ejemplo

- *sqrt(900);*
- *cos(45)*

Los argumentos pueden ser:

- *Literales: sqrt(4)*
- *Variables: sqrt(x)*
- *Expresiones: sqrt(sqrt(x)) ó sqrt(3 - 6x)*

Algunas Funciones Matemáticas

Method	Description	Example
`ceil( x )`	rounds *x* to the smallest integer not less than *x*	`ceil( 9.2 )` is `10.0` `ceil( -9.8 )` is `-9.0`
`cos( x )`	trigonometric cosine of *x* (*x* in radians)	`cos( 0.0 )` is `1.0`
`exp( x )`	exponential function *ex*	`exp( 1.0 )` is `2.71828` `exp( 2.0 )` is `7.38906`

`fabs( x )`	absolute value of *x*	`fabs( 5.1 )` is `5.1`
		`fabs( 0.0 )` is `0.0`
		`fabs( -8.76 )` is `8.76`

`floor( x )`	rounds *x* to the largest integer not greater than *x*	`floor( 9.2 )` is `9.0` `floor( -9.8 )` is `-10.0`
`fmod( x, y )`	remainder of *x*/*y* as a floating-point number	`fmod( 13.657, 2.333 )` is `1.992`
`log( x )`	natural logarithm of *x* (base *e*)	`log( 2.718282 )` is `1.0` `log( 7.389056 )` is `2.0`
`log10( x )`	logarithm of *x* (base 10)	`log10( 10.0 )` is `1.0` `log10( 100.0 )` is `2.0`
`pow( x, y )`	*x* raised to power *y* (*xy*)	`pow( 2, 7 )` is `128` `pow( 9, .5 )` is `3`
`sin( x )`	trigonometric sine of *x* (*x* in radians)	`sin( 0.0 )` is `0`
`sqrt( x )`	square root of *x*	`sqrt( 900.0 )` is `30.0` `sqrt( 9.0 )` is `3.0`
`tan( x )`	trigonometric tangent of *x* (*x* in radians)	`tan( 0.0 )` is `0`

Fig. 3.2 Math library functions.

Reglas de precedencia

Determinan el orden con el que se evalúan los operadores

- Paréntesis
- Funciones matemáticas
- * / %
- + -
- A igual precedencia, evaluar de izquierda a derecha
- Ejemplo:

```
y = 2 * 5 / 5 + 3 * 5 + 7
y = 10 / 5 + 3 * 5 + 7
y = 2 + 3 * 5 + 7
y = 2 + 15 + 7
y = 17 + 7
y = 24
```

Precedencia de Operadores

Operador	Significado	Asociatividad
()	Llamada a una función	De izquierda a derecha
[]	Acceso a los elementos de un array	
.	Miembro de una estructua o unión	De derecha a izquierda
->	Miembro de una estructura o unión (punteros)	
sizeof	Tamaño de una variable / tipo	
++	Postincremento (v++)	
--	Postdecremento (v--)	
++	Preincremento (++v)	
--	Predecremento (--v)	
~	Complemento bit a bit	
!	Negación lógica	
-	Menos unario	
+	Más unario	
&	Referencia (Dirección de…)	
*	Indirección (Contenido de…)	
(tipo)	Casting (conversión de tipo)	
*	Multiplicación	De izquierda a derecha
/	División	
%	Resto	
+	Suma	
-	Resta	
<<	Desplazamiento de bits a la izquierda	
>>	Desplazamiento de bits a la derecha	
<	Menor que…	
<=	Menor o igual que…	
>	Mayor que…	
>=	Mayor o igual que…	
==	Igual que…	
!=	Distinto de…	
&	Operador AND (bit a bit)	
^	Operador XOR (bit a bit)	
|	Operador OR (bit a bit)	
&&	Operador AND (lógico)	
||	Operador OR (lógico)	
? :	Operador condicional	De derecha a izquierda
=	Operador de asignación += -= *= /= %= &= ^= |= >>= <<=	

Caracteres

- Los literales de tipo carácter en C se escriben como valores entre comillas simples: ‘a’, ‘1’
- Códigos ASCII (en hexadecimal): \x??
  - ‘\x0a’ (avance de línea)
  - ‘\x0d’ (retorno de carro)
- También hay secuencias de escape para representar caracteres especiales.

Secuencia de escape	Descripción
\t	Tabulador (tab)
\n	Avance de línea (new line)
\r	Retorno de carro (carriage return)
\b	Retroceso (backspace)
\f	Salto de página (form feed)
\a	Sonido de alerta
\’	Comillas simples (apóstrofe)
\”	Comillas dobles
\\	Barra invertida
\?	Signo de interrogación
\0	Carácter nulo (NULL)

Tabla ASCII

Regular ASCII Chart (character codes 0 - 127)

000d	00h		(nul)	016d	10h	▶	(dle)	032d	20h	sp	048d	30h	0	064d	40h	@	080d	50h	P	096d	60h	`	112d	70h	p
001d	01h	☺	(soh)	017d	11h	◀	(dc1)	033d	21h	!	049d	31h	1	065d	41h	A	081d	51h	Q	097d	61h	a	113d	71h	q
002d	02h	☻	(stx)	018d	12h	↕	(dc2)	034d	22h	"	050d	32h	2	066d	42h	B	082d	52h	R	098d	62h	b	114d	72h	r
003d	03h	♥	(etx)	019d	13h	‼	(dc3)	035d	23h	#	051d	33h	3	067d	43h	C	083d	53h	S	099d	63h	c	115d	73h	s
004d	04h	♦	(eot)	020d	14h	¶	(dc4)	036d	24h	$	052d	34h	4	068d	44h	D	084d	54h	T	100d	64h	d	116d	74h	t
005d	05h	♣	(enq)	021d	15h	§	(nak)	037d	25h	%	053d	35h	5	069d	45h	E	085d	55h	U	101d	65h	e	117d	75h	u
006d	06h	♠	(ack)	022d	16h	▬	(syn)	038d	26h	&	054d	36h	6	070d	46h	F	086d	56h	V	102d	66h	f	118d	76h	v
007d	07h	•	(bel)	023d	17h	↨	(etb)	039d	27h	'	055d	37h	7	071d	47h	G	087d	57h	W	103d	67h	g	119d	77h	w
008d	08h	◘	(bs)	024d	18h	↑	(can)	040d	28h	(	056d	38h	8	072d	48h	H	088d	58h	X	104d	68h	h	120d	78h	x
009d	09h		(tab)	025d	19h	↓	(em)	041d	29h	)	057d	39h	9	073d	49h	I	089d	59h	Y	105d	69h	i	121d	79h	y
010d	0Ah		(lf)	026d	1Ah		(eof)	042d	2Ah	*	058d	3Ah	:	074d	4Ah	J	090d	5Ah	Z	106d	6Ah	j	122d	7Ah	z
011d	0Bh	♂	(vt)	027d	1Bh	←	(esc)	043d	2Bh	+	059d	3Bh	;	075d	4Bh	K	091d	5Bh	[	107d	6Bh	k	123d	7Bh	{
012d	0Ch	♀	(np)	028d	1Ch	∟	(fs)	044d	2Ch	,	060d	3Ch	<	076d	4Ch	L	092d	5Ch	\	108d	6Ch	l	124d	7Ch	|
013d	0Dh		(cr)	029d	1Dh	↔	(gs)	045d	2Dh	-	061d	3Dh	=	077d	4Dh	M	093d	5Dh	]	109d	6Dh	m	125d	7Dh	}
014d	0Eh	♫	(so)	030d	1Eh	▲	(rs)	046d	2Eh	.	062d	3Eh	>	078d	4Eh	N	094d	5Eh	^	110d	6Eh	n	126d	7Eh	~
015d	0Fh	☼	(si)	031d	1Fh	▼	(us)	047d	2Fh	/	063d	3Fh	?	079d	4Fh	O	095d	5Fh	_	111d	6Fh	o	127d	7Fh	⌂

Extended ASCII Chart (character codes 128 – 255; Codepage 850)

128d	80h	Ç	144d	90h	É	160d	A0h	á	176d	B0h	░	192d	C0h	└	208d	D0h	ð	224d	E0h	Ó	240d	F0h	-
129d	81h	ü	145d	91h	æ	161d	A1h	í	177d	B1h	▒	193d	C1h	┴	209d	D1h	Ð	225d	E1h	ß	241d	F1h	±
130d	82h	é	146d	92h	Æ	162d	A2h	ó	178d	B2h	▓	194d	C2h	┬	210d	D2h	Ê	226d	E2h	Ô	242d	F2h	‗
131d	83h	â	147d	93h	ô	163d	A3h	ú	179d	B3h	│	195d	C3h	├	211d	D3h	Ë	227d	E3h	Ò	243d	F3h	¾
132d	84h	ä	148d	94h	ö	164d	A4h	ñ	180d	B4h	┤	196d	C4h	─	212d	D4h	È	228d	E4h	õ	244d	F4h	¶
133d	85h	à	149d	95h	ò	165d	A5h	Ñ	181d	B5h	Á	197d	C5h	┼	213d	D5h	ı	229d	E5h	Õ	245d	F5h	§
134d	86h	å	150d	96h	û	166d	A6h	ª	182d	B6h	Â	198d	C6h	ã	214d	D6h	Í	230d	E6h	µ	246d	F6h	÷
135d	87h	ç	151d	97h	ù	167d	A7h	º	183d	B7h	À	199d	C7h	Ã	215d	D7h	Î	231d	E7h	þ	247d	F7h	¸
136d	88h	ê	152d	98h	ÿ	168d	A8h	¿	184d	B8h	©	200d	C8h	╚	216d	D8h	Ï	232d	E8h	Þ	248d	F8h	°
137d	89h	ë	153d	99h	Ö	169d	A9h	®	185d	B9h	╣	201d	C9h	╔	217d	D9h	┘	233d	E9h	Ú	249d	F9h	¨
138d	8Ah	è	154d	9Ah	Ü	170d	AAh	¬	186d	BAh	║	202d	CAh	╩	218d	DAh	┌	234d	EAh	Û	250d	FAh	·
139d	8Bh	ï	155d	9Bh	ø	171d	ABh	½	187d	BBh	╗	203d	CBh	╦	219d	DBh	█	235d	EBh	Ù	251d	FBh	¹
140d	8Ch	î	156d	9Ch	£	172d	ACh	¼	188d	BCh	╝	204d	CCh	╠	220d	DCh	▄	236d	ECh	ý	252d	FCh	³
141d	8Dh	ì	157d	9Dh	Ø	173d	ADh	¡	189d	BDh	¢	205d	CDh	═	221d	DDh	¦	237d	EDh	Ý	253d	FDh	²
142d	8Eh	Ä	158d	9Eh	×	174d	AEh	«	190d	BEh	¥	206d	CEh	╬	222d	DEh	Ì	238d	EEh	¯	254d	FEh	■
143d	8Fh	Å	159d	9Fh	ƒ	175d	AFh	»	191d	BFh	┐	207d	CFh	¤	223d	DFh	▀	239d	EFh	´	255d	FFh	

Cadenas de Caracteres

- En ANSI C no existen las cadenas de caracteres como tipo predefinido: Una cadena de caracteres no es más que un vector de caracteres.
- Para especificar un literal de cadena de caracteres se utilizan comillas dobles
  - “Esto es una cadena”
  - “’Esto’ también es una cadena”
- Las secuencias de escape son necesarias para introducir determinados caracteres dentro de una cadena:
  - “\”Cadena entre comillas\””

Formación de cadenas de caracteres

- Para construir cadenas de caracteres en las que mostrar datos se utilizan plantillas que se sustituirán por una representación adecuada de los valores del tipo indicado.

```
int edad=21;
printf ("La edad es %d \n",edad);
```

Plantilla	Tipo de dato
%c	char
%s	Cadena de caracteres
%d	int (en decimal)
%o	int (en octal)
%x	int (en hexadecimal)
%ld	long
%f	float
%lf	double
%Lf	long double

Entrada/Salida de Datos

- Para escribir por pantalla o leer desde teclado se utilizan las funciones printf y scanf que forman parte de la biblioteca estándar stdio.h . Tienen el siguiente formato:

  printf("plantilla de formato", lista de variables);

  scanf("plantilla de formato",lista de referencia a variables);

- Ejemplos mostrar datos:

```
char c;          int i;          float f;          double d;
…                …               …                 …
printf("%c",c);  printf("%d",i); printf("%f",f);  printf("%lf",d);
```

- Ejemplos leer datos:

```
char c;          int i;          float f;          double d;
…                …               …                 …
scanf("%c",&c);  scanf("%d",&i); scanf("%f",&f);  scanf("%lf",&d);
```

Salida de datos con formato

- Al representar un número o cadena podemos especificar como hacerlo con **`%-n.mX`**
- El número **`n`** indica el número de caracteres que se utilizarán como mínimo para representar el dato (núm total de dígitos en el caso de los números, tantos enteros como reales)
- El número **`.m`** indica el número máximo de caracteres que se utilizarán para representar el dato. En los reales, indica el número de decimales que se mostrarán.
- El signo – es opcional e indica que el texto ha de justificarse a la izquierda.
- **`X`** indica el tipo de dato.

Ejemplos de salida con formato

```
printf( "|%s|", "Hola mundo" )          |Hola mundo|
printf( "|%20s|", "Hola mundo" )        |          Hola mundo|
printf( "|%-20s|", "Hola mundo" )       |Hola mundo          |
printf( "|%20.4s|", "Hola mundo" )      |                Hola|
printf( "|%-20.4s|", "Hola mundo" )     |Hola                |

printf( "|%c|", 'a' )                   |a|
printf( "|%c|", 97  )                   |a|
printf( "|%c|", 'a' + 3 )               |d|


printf( "|%d|", 23423 )                 |23423|
printf( "|%10d|", 23423 )               |     23423|
printf( "|%-10d|", 23423 )              |23423      |

printf( "|%010d|", 23423 )              |0000023423|

printf( "|%10i|", 29387 )               |     29387|
printf( "|%10o|", 29387 )               |     71313|
printf( "|%10x|", 29387 )               |      72cb|

printf( "|%f|", 423.2348 )             |423.234800|
printf( "|%12f|", 423.2348 )           |  423.234800|
printf( "|%12.0f|", 423.2348 )         |         423|
printf( "|%12.2f|", 423.2348 )         |      423.23|
printf( "|%.2f|", 423.2348 )           |423.23|
printf( "|%e|", 423.2348 )             |4.232348e+02|
```

Ejercicios

- Supongamos que un producto cuesta 17345 euros. A cuantas pesetas equivale?. Implemente un programa que permita contestar a esta pregunta. (Recuerde 1 euro = 166.386 pesetas)

- Como debería hacer para implementar un programa que permita hacer la transformación de cualquier cantidad de euros ?

- Implemente un programa que lea dos valores enteros y muestre su suma, producto y división entera

- Escriba un programa que dados dos puntos p1=(x1,y1) y p2=(x2,y2), calcule la distancia euclídea entre ambos

Consejos Prácticos

- Escribir programa simples y claros
- Leer los manuales de la versión de C que se utilice
- Comenzar los programas con un comentario que describa su propósito
- Colocar un espacio después de cada coma (,)
- Asignar nombres significativos a las variables
- Colocar una línea en blanco entre la declaración de las variables y las instrucciones
- Colocar espacios en cada lado de un operador binario
- No incluir más de una instrucción por línea
- Terminar cada programa con \n

Errores Comunes

- No inicializar las variables
- Usar datos enteros y aplicar una división real
- Cadenas de desigualdades (x<z<y)
- Olvidar incluir stdio.h en un programa que usa printf ó scanf
- Omitir el punto y coma al final de una instrucción
- Usar el módulo (%) sobre operandos no enteros
- Dejar espacios en blanco entre los operadores ==, !=, <= y >=
- Confundir el operador de igualdad == con el de asignación =
- Partir los identificadores con espacios en blanco *(ma in)*