Tema 2: Lenguaje C: Elementos Básicos

Introducción a la programación I

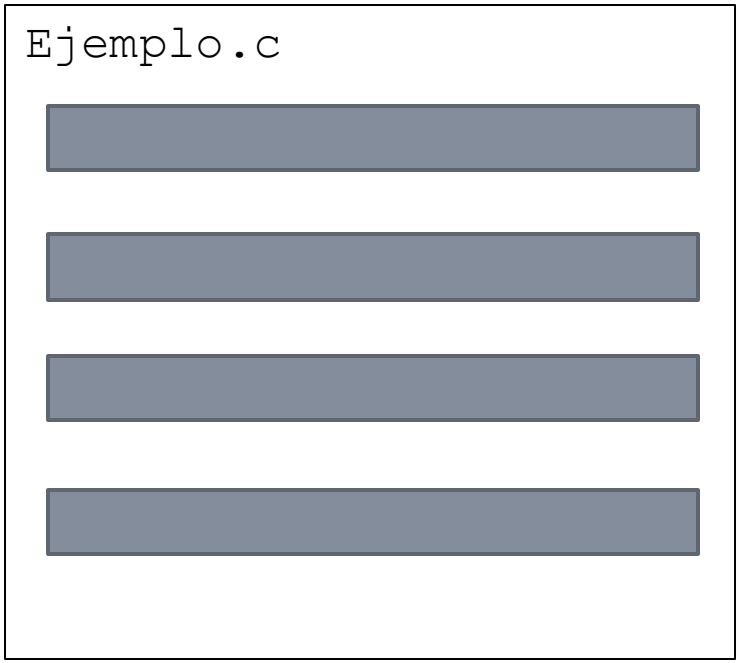
Ana Isabel Sierra de las Heras
Marcos Novalbos
Francisco Javier Garcia Algarra
Rodrigo Alonso Solaguren-Beascoa
Alfonso Castro Escudero

Índice

1. Estructura general de un programa en C
2. Elementos de un programa en C
3. Variables y Tipos de Datos en C
4. Entrada/Salida: printf() y scanf()
5. Constantes
6. Ejercicios

1. Estructura general de un programa en C

- Un programa en C es un conjunto de instrucciones que realizan una o más acciones.
- Un programa en C puede incluir:
  - Directivas de preprocesador
  - Declaraciones globales
  - La función principal main
  - Funciones definidas por el usuario
  - Comentarios del programa

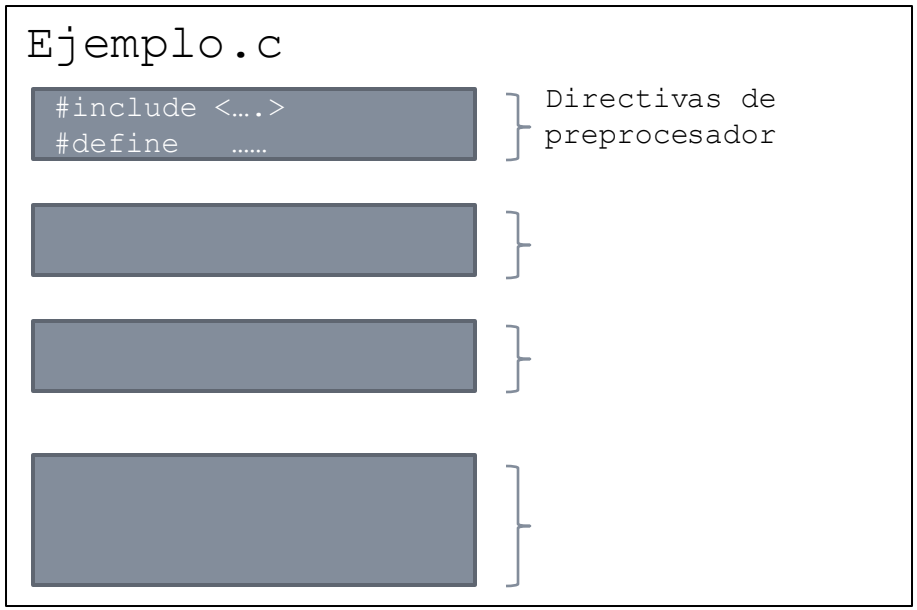


1.1 Directivas de preprocesador

- Las directivas de preprocesador son instrucciones que se le pasan al compilador.
- Todas las directivas del preprocesador comienzan con el signo "**#**" y no terminan en "**;**" ya que no son instrucciones del lenguaje C.
- **#include**
  - Directiva `#include` indica al compilador que lea un archivo denominado "cabecera" de la biblioteca de C (conjunto de funciones predefinidas) y que lo inserte en la posición donde se encuentra dicha directiva.
  - Estas instrucciones son de la forma:

```
#include <nombre_libreria.h>
            o
#include "nombre_libreria.h"
```

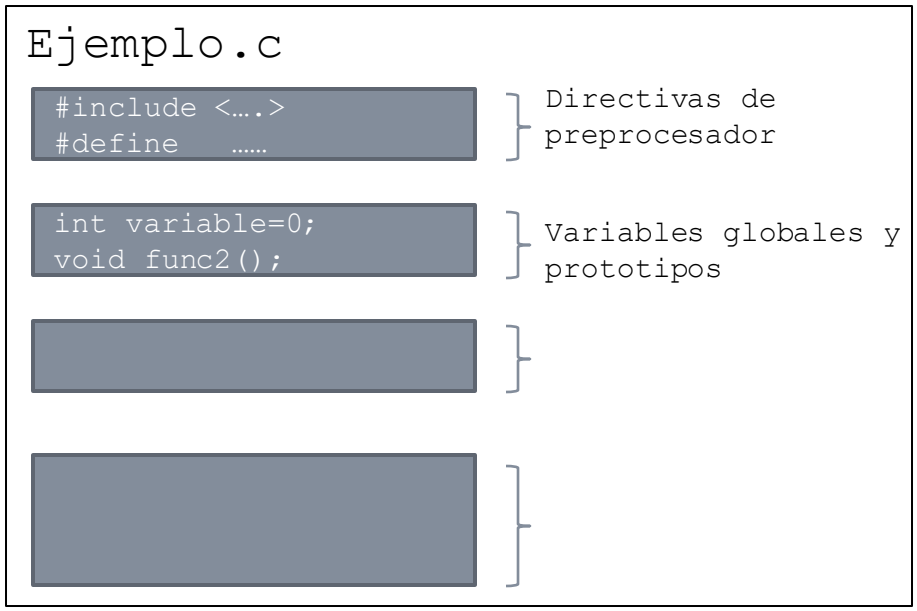
- **#define**
  - La directiva `#define` indica al preprocesador que defina un ítem de datos o una función para el programa C.
  - Así, la directiva `#define TAM 10` sustituirá el valor `10` cada vez que el símbolo `TAM` aparezca en el programa



1.2 Declaraciones globales

- Las declaraciones globales indican al usuario que las constantes o variables así declaradas son comunes a todas las funciones del programa

- La zona de declaraciones globales puede incluir:
  - Declaraciones de variables
  - Declaraciones de prototipos de funciones

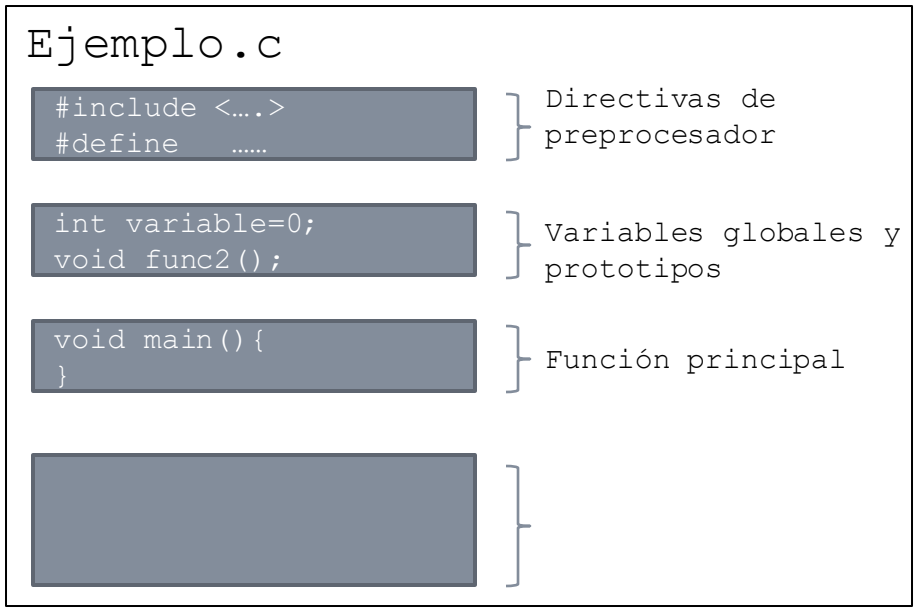


1.3 Función principal `main()`

- Cada programa de C tiene una función **`main()`** que es el punto inicial de entrada al programa
- Es la primera función invocada por el programa
- Su estructura es:

```
void main ()
{
  Bloque de instrucciones;
}
```

- Las instrucciones de C situadas en el cuerpo de la función `main()`, o de cualquier función, deben terminar en "**;**"



1.4 Funciones definidas por el usuario

- Un programa C es una colección de funciones.
- C proporciona funciones de diferente tipo:
  - Predefinidas que se encuentran en algún fichero cabecera de la biblioteca de C
  - Definidas y codificadas por el programador.
- Se invocan por su nombre y los parámetros que tenga.
- Después de que la función es llamada, se ejecuta el código asociado con dicha función y a continuación se retorna al punto desde el que la función se invocó.
- En C, las funciones definidas por el usuario requieren una declaración o prototipo en el programa, que indica al compilador el nombre por el que será invocada.
- Las funciones de algún archivo cabecera requieren que se incluya ese archivo donde está su declaración

Ejemplo.c

```
#include <….>
#define   ……
```
Directivas de preprocesador

```
int variable=0;
void func2();
```
Variables globales y prototipos

```
void main(){
}
```
Función principal

```
void func2(){
…….
}
```
Codificación de funciones propias

1.5 Comentarios

- Un comentario es cualquier información que se añade a su archivo fuente.
- Los comentarios "de bloque" en C comienzan por la secuencia de caracteres `/*` y terminan con `*/`
- También se pueden utilizar comentarios "de línea" utilizando dos "slash" seguidos ("//") antes una línea. El comentario acaba cuando acaba dicha línea.

Ejemplo.c

```
#include <….>
#define   ……
```
Directivas de preprocesador

```
/* comentarios */
```

```
int variable=0;
void func2();
```
Variables globales y prototipos

```
/* comentarios */
```

```
void main(){
}
```
Función principal

```
/* comentarios */
```

```
void func2(){
…….
/* comentarios */
// comentario
}
```
Codificación de funciones propias

1.6 Ejemplo: Estructura general de un programa en C

```
/* Nombre programa: Area de un circulo de radio 3
   Autor:
   Fecha: 1-10-2022 */

/* Directivas de preprocesador */
#include <stdio.h>
#define PI 3.14
#define RADIO 3

/* Declaraciones globales */
void AreaCirculo3 ();
float area=0.0f;

/* Función Principal main */
void main (int argc, char **argv){
     AreaCirculo3 ();
}

/* Codificación de funciones */
void AreaCirculo3 (){
     area = PI* RADIO* RADIO;
     printf("El area de un circulo de radio 3 es: %f", area);
}
```

1.6 Ejemplo: Estructura general de un programa en C

```
/* Nombre programa: Area de un circulo de radio 3
   Autor:
   Fecha: 1-10-2022 */

/* Directivas de preprocesador */
#include <stdio.h>
#define PI 3.14

/* Declaraciones globales */
float AreaCirculo (float radio);

/* Función Principal main */
void main (int argc, char **argv){
     float MiRadio = 4;
     float MiRadio=0.0f,MiArea=0.0f;
     MiArea = AreaCirculo (MiRadio);
     printf("El area de un circulo de radio %d es: %f", MiRadio, MiArea);
}

/* Codificación de funciones */
float AreaCirculo (float Radio){
     float Area=0.0f;
     Area = PI* RADIO* RADIO;
     return (Area);
}
```

2. Elementos de un programa en C

- Los elementos básicos de un programa C son:
  - Identificadores
  - Palabras reservadas
  - Comentarios
  - Signos de puntuación
  - Separadores
  - Archivos cabecera

2. Elementos de un programa en C

- **Identificadores**
  - Básicamente son los nombres de las variables y las funciones
  - Un identificador es una secuencia de caracteres, letras, dígitos y subrayados.
  - El primer carácter tiene que ser una letra o un subrayado.
  - Las letras mayúsculas y minúsculas son diferentes.
  - Pueden tener cualquier longitud, pero el compilador ignora a partir de 32.
  - No pueden ser palabras reservadas
- **Palabras reservadas**
  - Son las palabras que tienen un significado semántico para el lenguaje C
  - Las palabras clave, combinadas con la sintaxis formal de C, conforman el lenguaje de programación C.
  - No pueden ser utilizadas como identificadores.

2. Elementos de un programa en C

- **Palabras reservadas**
  - Palabras reservadas en el estándar C89

main no es una palabra reservada es el nombre de una función, pero se la tratará como así lo fuera

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

2. Elementos de un programa en C

- **Palabras reservadas**
  - Palabras reservadas añadidas en el estándar C99 (1)

_Bool	_Imaginary	restict
_Complex	_inline	

  - Además, muchos compiladores de C han añadido varias palabras clave para explotar de manera más eficiente el sistema operativo (organización de memoria o acceso a interrupciones).
  - Lista de palabras reservadas ampliadas más frecuentes

asm	_ds	_huge	pascal
cdecl	_es	interrupt	_ss
_cs	far	near	

(1): En marzo de 2000, ANSI adoptó la norma ISO / IEC 9899: 1999. Este estándar se conoce comúnmente como C99.

2. Elementos de un programa en C

- **Comentarios**
  - Los comentarios de boque se encierran entre /* y */
  - Éstos pueden extenderse a lo largo de varias líneas
  - Los comentarios de línea se inician con // y acaban al finalizar dicha línea.
  - Los comentarios son ignorados por el compilador.
- **Signos de puntuación**
  - Todas las instrucciones deben terminar en “;”
  - Otros signos de puntuación son:
    - ! % ^ & * ( ) - + = { } [ ] \ ; ‘ : < > ? , . / “

2. Elementos de un programa en C

- **Separadores**
  - Los separadores son espacios en blanco, retorno de carro y avances de línea.

- **Archivos cabecera**
  - Un archivo de cabecera es un archivo especial que contiene las declaraciones de objetos y funciones de biblioteca que son añadidos en el lugar donde se insertan.
  - Un archivo de "cabecera" se inserta con la directiva #include

2. Ejercicio. Identificadores en C

- ¿Cuál de los siguientes identificadores es válido en c

```
W
MiCasa
Mi Casa
Mi_Casa
15
printf
Printf
main
int
5peras
Peras
_Peras
*Peras
```

2. Ejercicio. Identificadores en C

- ¿Cuál de los siguientes identificadores es válido en c

```
W -> Correcto.
MiCasa -> Correcto.
Mi Casa -> Incorrecto tiene un espacio.
Mi_Casa -> Correcto.
15 -> Incorrecto es un número.
printf -> Correcto, pero no recomendado ya que es el nombre de una función de la librería stdio.h
Printf -> Correcto, pero no recomendado ya que es el nombre de una función de la librería stdio.h.
Main -> Correcto, pero no recomendado ya que es el nombre de la función principal.
int -> Incorrecto es el nombre de un tipo de variable.
5peras -> Incorrecto comienza por un número.
Peras -> Correcto.
_Peras -> Correcto.
*Peras -> Incorrecto comienza por asterisco.
```

3. Variables y Tipos de Datos en C

- **Codificación y almacenamiento de datos en la memoria del ordenador**
  - Los lenguajes de programación de propósito general como el C, almacenan la información en la memoria utilizando grupos de 1, 2, 4 y 8 bytes (8, 16, 32 y 64 bits) dependiendo de los tipos de datos que se desean almacenar y de la arquitectura interna del ordenador

- **Una variable** es una o varias posiciones de memoria con un nombre determinado y que se usa para mantener un valor que puede ser modificado por el programa.

- Todas las variables han de ser declaradas para ser utilizadas.

- La forma general de declarar una variable es:

```
tipo lista_de_variables;
```

  - `tipo` debe ser un tipo de datos válido con algún modificador y `lista_de_variables` puede consistir en uno o más nombres de identificadores separados por comas (“`,`”)
  - Se recomienda **encarecidamente** inicializar la variable a un valor conocido (Nulo ó cero).

3. Variables y Tipos de Datos en C

- Ejemplos:

```
int i=0, j=0, k=0 ;
double beneficio=0.0, gasto=0.0, precio=0.0;
float altura=0.0f, peso=0.0f;
```

- El nombre de una variable no tiene nada que ver con el tipo.
- El **tipo de una variable** determina cuánto espacio ocupa en el almacenamiento y cómo se interpreta el patrón de bits almacenado.
- Cada **tipo de dato** ocupa diferente número de posiciones de memoria
- El tamaño en bits asignado a un tipo de datos depende de la arquitectura de la computadora y del compilador utilizado

3. Variables y Tipos de Datos en C

- Los tipos de datos en C se clasifican en:
  - **Tipos básicos**. Son tipos aritméticos que se clasifican en:
    - Tipos enteros
    - Tipos en punto flotante
  - **Tipos enumerados.** Son tipos aritméticos y se usan para definir variables que solo pueden asignar ciertos valores enteros discretos a lo largo del programa
  - **Tipo void**. Indica que ningún valor está disponible. Así:
    - Declara explícitamente que una función no devuelve ningún valor.
    - Declara explícitamente que una función no admite ningún parámetro.
    - Crea punteros genéricos (de cualquier tipo). Representa la dirección de un objeto que no tiene ningún tipo predeterminado.
  - **Tipos derivados**. Incluyen:
    - Tipos punteros
    - Tipos array
    - Tipo estructura
    - Tipo unión
    - Tipo función

3. Variables y Tipos de Datos en C

- Los tipos de datos básicos enteros se clasifican en:
  - **`char`** `Caracteres`
  - **`int`** `Enteros`

- Los tipos de datos básicos coma flotante se clasifican en:
  - **`float`** `Real de simple precisión`
  - **`double`** `Real de doble precisión`

- Sin tipo void
  - Declara explícitamente que una función no devuelve ningún valor
  - Crea punteros genéricos (de cualquier tipo)

3. Variables y Tipos de Datos en C

- **Modificación** de tipos básicos (modificadores)
  - Salvo el tipo void el resto pueden tener modificadores que les preceden y que modifican dicho tipo de dato.
  - Así, el modificador altera el valor de un tipo, pudiendo ser los modificadores los siguientes:
    - signed
    - unsigned
    - long
    - short

3. Variables y Tipos de Datos en C

- Los tipos básicos enteros, sin y con modificadores, más representativos son

Tipo	Tamaño en bits	Intervalo
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long (long int)	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long int	8 bytes	0 to 18446744073709551615

3. Ejercicio1. Variables y Tipos de Datos en C

- Codificar un programa para comprobar los valores del anterior cuadro

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <float.h>

int main(int argc, char** argv) {

    printf("El numero de bits de un char es         :   %d\n", CHAR_BIT);
    printf("El numero de bytes de un char es        :   %d\n", sizeof(char));
    printf("El menor valor de un signed char es     :   %d\n", CHAR_MIN);
    printf("El mayor valor de un signed char es     :   %d\n", CHAR_MAX);
    printf("El mayor valor de un unsigned char es   :   %d\n\n", UCHAR_MAX);
    printf("El numero de bytes de un int es         :   %d\n", sizeof(int));
    printf("El menor valor de un int es             :   %d\n", INT_MIN);
    printf("El mayor valor de un int es             :   %d\n", INT_MAX);
    printf("El mayor valor de un unsigned int es    :   %u\n\n", (unsigned int) UINT_MAX);
    printf("El numero de bytes de un short es       :   %d\n", sizeof(short));
    printf("El menor valor de un short es           :   %d\n", SHRT_MIN);
    printf("El mayor valor de un short es           :   %d\n", SHRT_MAX);
    printf("El mayor valor de un unsigned short es  :   %d\n\n", (unsigned short) USHRT_MAX);
    printf("El numero de bytes de un long es        :   %d\n", sizeof(long));
    printf("El menor valor de un long es            :   %ld\n", (long) LONG_MIN);
    printf("El mayor valor de un long es            :   %ld\n", (long) LONG_MAX);
    printf("El mayor valor de un unsigned long es   :   %lu\n", (unsigned long) ULONG_MAX);

    return 0;
}
```

3. Variables y Tipos de Datos en C

- **char (carácter)**
  - C procesa datos de tipo carácter y los símbolos tipográficos utilizando el tipo de dato char.
  - Las variables tipo char se almacenan internamente en 1 byte de memoria (8 bits).
  - Para la representación de estos caracteres utiliza el código ASCII. Dichos códigos ASCII toman valores enteros del intervalo 0 a 255 (con 8 bits en binario se pueden tener 2^8=256 distintos números) .
  - Los caracteres se almacenan internamente como números y por lo tanto se pueden realizar operaciones aritméticas con datos tipo char.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
01	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
02	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
03	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
04	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
05	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
06	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
07	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- Ejemplo:

```
char caracter = 'b';
caracter = caracter - 32
```

Convierte b (código ascii 98) a B (código ASCII 66)

ASCII: American Standard Code for Information Interchange (valores hexadecimales)

3. Variables y Tipos de Datos en C

Tabla códigos ASCII

(valores decimales)

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

3. Ejercicio. Variables y Tipos de Datos en C

- Mostrar utilizando un programa de C la representación interna del carácter 'A'. Transformar el carácter 'A' (mayúscula) al carácter 'a' (minúscula) sin usar funciones externas.

3. Ejercicio1. Variables y Tipos de Datos en C

- Mostrar utilizando un programa de C la representación interna del carácter ‘A’. Transformar el carácter ‘A’ (mayúscula) al carácter ‘a’ (minúscula) sin usar funciones externas.

```
#include <stdio.h>

void main(int argc, char** argv) {

   char Letra = 'A';

   printf ("la letra es %c\n", Letra);
   /* un char es un numero y el numero es el codigo ascii del char */
   printf ("la letra es %d\n", Letra);
   /* Si se suma 32 a una letra mayuscula se tiene la minuscula */
   printf ("la letra minuscula es %c\n", Letra+32);
}
```

```
$ ./Tema2_2_TiposC.exe
la letra es A
la letra es 65
la letra minuscula es a
```

3. Variables y Tipos de Datos en C

- **int (entero)**
  - Los enteros se almacenan internamente en 2 ó 4 bytes de memoria.
  - El uso del modificador "signed" en enteros está permitido, pero es superfluo porque la declaración implícita de entero asume números con signo.
  - La diferencia entre un entero con signo o sin signo es como se interpreta el bit más significativo
    - El intervalo de un unsigned short int (16 bits) es 0 .. 65.535
    - El intervalo de un unsigned int (32 bits) 0 .. 4,294,967,295
  - Su uso habitual es:
    - Aritmética de enteros
    - Bucles for
    - Conteo

3. Variables y Tipos de Datos en C

- **int (entero)**
  - Los enteros se almacenan internamente en 2 o 4 bytes de memoria.
  - El uso del modificador "signed" en enteros está permitido, pero es superfluo porque la declaración implícita de entero asume números con signo.
  - La diferencia entre un entero con signo o sin signo es como se interpreta el bit más significativo
    - El intervalo de un unsigned short int (16 bits) es 0 .. 65.535
    - El intervalo de un unsigned int (32 bits) 0 .. 4,2[illegible]967,295
  - Su uso habitual es:
    - Aritmética de enteros
    - Bucles for
    - Conteo

$1111\ 1111\ 1111\ 1111)_2 = 65.535)_{10}$

3. Variables y Tipos de Datos en C

- **Representación interna de un int (entero)**
  - Los **números enteros positivos** se almacenan en el formato asociado con el **sistema binario** tal como se ha visto previamente.
  - Para los **números enteros negativos**, el formato en el que se almacenan es **complemento a 2** que consiste en:
    - Tomar el numero positivo en binario
    - Cambiar los ceros por unos
    - Sumar uno al resultado
  - Ejemplo representación de negativos en complemento a 2 con 4 bits
    - 3 (decimal) = 0011 -3 (decimal) = 1101
    - 6 (decimal) = 0110 -6 (decimal) = 1010

3. Ejercicio. Variables y Tipos de Datos en C

- Mostrar utilizando un programa de C la representación en hexadecimal de dos números enteros tanto positivos como negativos. (%x es el especificador del printf que permite presentar los 1 y 0 de una posición de memoria en hexadecimal)

3. Ejercicio2. Variables y Tipos de Datos en C

- Mostrar utilizando un programa de C la representación en hexadecimal de dos números enteros tanto positivos como negativos. (%x es el especificador del printf que permite presentar los 1 y 0 de una posición de memoria en hexadecimal)

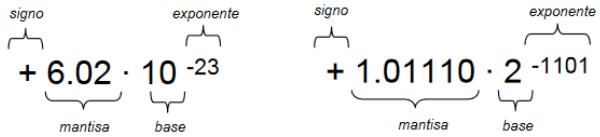
```
#include <stdio.h>

void main(int argc, char** argv) {

   int NumeroPositivo = 5;
   int NumeroNegativo = -5;

   printf ("La representacion interna del numero positivo es: %x\n",NumeroPositivo);
   printf ("La representacion interna del numero negativo es: %x\n",NumeroNegativo);

}
```

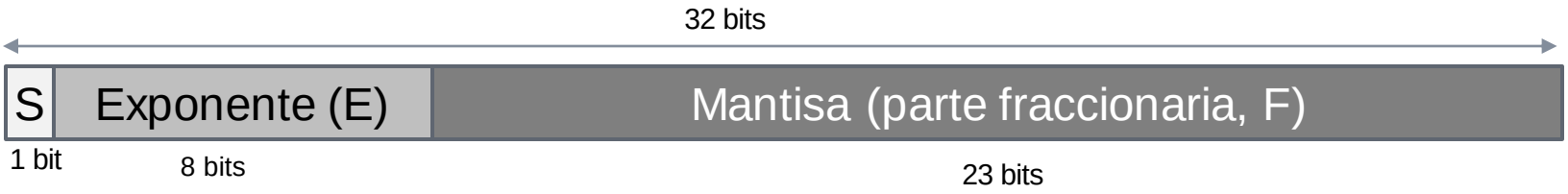
```
$ ./Tema2_2_TiposC.exe
La representacion interna del numero positivo es: 5
La representacion interna del numero negativo es: fffffffb
```

3. Variables y Tipos de Datos en C

- **float y double (reales)**
  - En el formato estándar ANSI/IEEE 754-1985 para un número binario de simple precisión, el bit de signo (S) es el que se encuentra más a la izquierda, el exponente (E) incluye los siguientes 8 bits y la mantisa o parte fraccionaria (F) incluye los restantes 23 bits.

$$N = s \,.\, M \,.\, r^{E}$$





  - En doble precisión (64 bits) : 1 bit de signo, 11 de exponente y 52 de mantisa

3. Variables y Tipos de Datos en C

- Tipos básicos coma flotante sin y con modificadores más representativos son

Tipo	Tamaño en bits	Intervalo
float	4 bytes	1.2E-38 to 3.4E+38
double	8 bytes	2.3E-308 to 1.7E+308
long double	16 bytes	3.4E-4932 to 1.1E+4932

- Los valores máximos y mínimos de cada uno de los tipos básicos se encuentran recogidos como constantes definidas en distintas librerías (limits.h y float.h)
- La cantidad de bytes requerida para un long double puede variar considerablemente entre diferentes arquitecturas y compiladores

3. Ejercicio 3. Variables y Tipos de Datos en C

- Codificar un programa que calcule el área de un la base de un cilindro y su volumen, pidiéndole al usuario el radio y la altura y utilizando un float para el radio y un double para la altura

3. Ejercicio 3. Variables y Tipos de Datos en C

- Codificar un programa que calcule el área de un la base y el volumen de un círculo, pidiéndole al usuario el radio y la altura y utilizando un float y un double para la altura

```
#include <stdio.h>

void main() {
    // Variables para el radio y altura
    float radio=0.0, areaCirculo=0.0;
    double altura=0.0, volumenCilindro=0.0;

    // Constante para el valor de PI
    const double PI = 3.14159265359;

    // Solicitar al usuario que ingrese el radio del círculo
    printf("Ingrese el radio del círculo: ");
    scanf("%f", &radio);

    // Calcular el área del círculo (usando float)
    areaCirculo = PI * radio * radio;

    // Solicitar al usuario que ingrese la altura del cilindro
    printf("Ingrese la altura del cilindro: ");
    scanf("%lf", &altura);

    // Calcular el volumen del cilindro (usando double)
    volumenCilindro = PI * radio * radio * altura;

    // Imprimir los resultados
    printf("El área del círculo es: %.2f\n", areaCirculo);
    printf("El volumen del cilindro es: %.2lf\n", volumenCilindro);
}
```

3. Variables y Tipos de Datos en C

- **Tipos de variables según donde se declaran**
  - Variables locales.
    - Se declaran dentro de las funciones.
  - Parámetros formales
    - Se declaran en la definición de las funciones
  - Variables globales
    - Se declaran fuera de todas las funciones

5. Variables y Tipos de Datos en C

- **Variables locales**
  - Se declaran dentro de las funciones.
  - Pueden ser utilizadas solo en las instrucciones que estén dentro del bloque en el que se han sido declaradas.
  - Solo existen mientras se está ejecutando el bloque de código en el que se han declarado.
  - Se crea cuando se entra en el bloque y se destruye cuando se sale de él.
  - Utilizando el modificador `static`, las variables locales pueden retener sus valores entre distintas llamadas a las funciones donde están definidas
  - Las variables locales se guardan en la pila.

5. Variables y Tipos de Datos en C

- **Variables locales**
  - El nombre de una variable local puede repetirse en diferentes bloques, teniendo un significado completamente diferente.

```
void function1(){
  float altura;
  altura  = 1.75;
}
```

  - La palabra reservada `auto` se usa para declarar que una variable local existe solamente mientras estemos dentro de la subrutina o bloque de programa donde se declara, pero, dado que por defecto toda variable local es auto, no suele usarse.
  - Por conveniencia y limpieza del código, se declaran todas las variables locales de un bloque al principio del mismo, después de { , antes de las instrucciones, aunque se pueden declarar en cualquier punto del bloque.

5. Variables y Tipos de Datos en C

- **Variables globales**
  - Se declaran fuera de todas las funciones
  - Se conocen a lo largo de todo el programa y se pueden usar en cualquier parte del código
  - Mantienen su valor durante toda la ejecución del programa
  - Se almacenan en una zona de memoria fija establecida por el propio compilador.

5. Variables y Tipos de Datos en C

- Ejemplo variables

¿Cuál sería la salida por terminal del programa?

```
#include <stdio.h>
int cuenta; /*cuenta es una variable global*/

void Func1 (void);
void Func2 (void);

int main(void){
   cuenta = 10;
   printf ("A la entrada del main cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
   Func1 ();
   printf ("A la salida del main cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
   return 0;
}
void Func1 (void){
   int temp=0;
   temp = cuenta;
   Func2 ();
   printf ("A la salida de Func1 temp es %d y cuenta es %d\n", temp,cuenta); /*muestra el valor de temp*/
}
void Func2 (void){
   int cuenta=0;
   cuenta = 15;
   printf ("A la salida de Func2 cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
}
```

5. Variables y Tipos de Datos en C

- Ejemplo variables

```
#include <stdio.h>
int cuenta; /*cuenta es una variable global*/

void Func1 (void);
void Func2 (void);

int main(void){
   cuenta = 10;
   printf ("A la entrada del main cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
   Func1 ();
   printf ("A la salida del main cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
   return 0;
}
void Func1 (void){
   int temp=0;
   temp = cuenta;
   Func2 ();
   printf ("A la salida de Func1 temp es %d y cuenta es %d\n", temp,cuenta); /*muestra el valor de temp*/
}
void Func2 (void){
   int cuenta;
   cuenta = 15;
   printf ("A la salida de Func2 cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
}
```

```
usuario@desktop$
$ ./VariblesGlobales.exe
A la entrada del main cuenta es 10
A la salida de Func2 cuenta es 15
A la salida de Func1 temp es 10 y cuenta es 10
A la salida del main cuenta es 10
```

5. Variables y Tipos de Datos en C

- Ejemplo variables

```
#include <stdio.h>
int cuenta; /*cuenta es una variable global*/

void Func1 (void);
void Func2 (void);

int main(void){
   cuenta = 10;
   printf ("A la entrada del main cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
   Func1 ();
   printf ("A la salida del main cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
   return 0;
}
void Func1 (void){
   int temp;
   temp = cuenta;
   Func2 ();
   printf ("A la salida de Func1 temp es %d y cuenta es %d\n", temp,cuenta); /*muestra el valor de temp*/
}
void Func2 (void){
   /* int cuenta;*/
   cuenta = 15;
   printf ("A la salida de Func2 cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
}
```

5. Variables y Tipos de Datos en C

- Ejemplo variables

```
#include <stdio.h>
int cuenta; /*cuenta es una variable global*/

void Func1 (void);
void Func2 (void);

int main(void){
   cuenta = 10;
   printf ("A la entrada del main cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
   Func1 ();
   printf ("A la salida del main cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
   return 0;
}
void Func1 (void){
   int temp;
   temp = cuenta;
   Func2 ();
   printf ("A la salida de Func1 temp es %d y cuenta es %d\n", temp,cuenta); /*muestra el valor de temp*/
}
void Func2 (void){
   /* int cuenta;*/
   cuenta = 15;
   printf ("A la salida de Func2 cuenta es %d\n", cuenta); /*muestra el valor de cuenta*/
}
```

```
usuario@desktop$
$ ./VariblesGlobales.exe
A la entrada del main cuenta es 10
A la salida de Func2 cuenta es 15
A la salida de Func1 temp es 10 y cuenta es 15
A la salida del main cuenta es 15
```

4. Entrada/Salida

- El archivo de cabecera de la biblioteca de C que proporciona las facilidades de entrada y salida es **`stdio.h`**
- Salida
  - La función **`printf()`** visualiza datos en la salida estándar (monitor)
  - Transforma los datos que están almacenados en binario dentro del computador a ASCII.

```
int  printf("Cadena de formato",[Lista de variables o expresiones]);
```

- Entrada
  - La función más utilizada para la entrada de datos a través de la entrada estándar (teclado) es **`scanf()`**

```
int  scanf("Cadena de formato",  Lista de direcciones);
```

4. Entrada/Salida

- Las funciones "printf" y "scanf" no tienen un número de parámetros fijo. Este tipo de funciones se denominan "cabeceras/prototipos de formato dinámico", y usan uno de los parámetros de entrada (normalmente el primero) para describir el resto parámetros.

- En el caso de "printf" y "scanf", esos parámetros son descritos usando una cadena de caracteres (una frase) en la que se incluye la escritura y lectura de variables con un formato especial.

- IMPORTANTE: Los tipos de datos deben coincidir con los descritos en la cadena de entrada. En caso contrario puede haber errores (en algunos casos, graves).

- Un ejemplo de uso sería el siguiente:
  - Si se quiere imprimir el valor de dos variables enteras llamadas “var1” y "var2", se podrá realizar de la siguiente manera:

```
printf("las variables enteras valen: %d %d \n",var1, var2);
```

4. Entrada/Salida. printf()

- La función **`printf()`** visualiza datos en la salida estándar (monitor)
- Transforma los datos que están almacenados en binario dentro del computador a ASCII.

```
int  printf("Cadena de formato",[Lista de variables o expresiones]);
```

- La función printf retorna un entero, igual al número de caracteres que se enviaron a la salida estándar (monitor) si no hay fallo y si hay fallo retorna un número negativo

- La cadena de formato contiene:
  - **Si se muestra el valor de una o varias variables:**
    - Los tipos de datos de la variable o las variables y las especificaciones de formato (forma de mostrarlos)
  - **Si no**
    - El texto que se envía a la salida externa

4. Entrada/Salida

- En el caso anterior, el primer parámetro es una cadena de caracteres, en la cual hay una palabra que comienza por "%", seguida de un carácter (especificador de formato).

- El carácter % indica que se rellenará con una variable pasada a printf por parámetros, y el siguiente carácter indicará el tipo de esa variable ("d", entero decimal). Dado que se pide imprimir una variable decimal, la variable "var1" debería haber sido declarada como un "integer". A continuación, se lista algunos de los tipos de datos más usados, soportados por printf:

Especificador	Salida	Ejemplo
%d , %i	Entero decimal con signo	392
%u	Entero decimal sin signo	7235
%f	Variable en formato coma flotante	392.65
%c	Caracter	a
%s	Cadena de caracteres	coche

4. Ejercicio. Entrada/Salida

- Ejemplo de uso con printf y varios tipos (float, int, char)
  - Escribir un programa que presente por pantalla: el carácter 'y', el entero 11 y el real 10.2.

4. Ejercicio. Entrada/Salida

- Ejemplo de uso con printf y varios tipos (float, int, char)
  - Escribir un programa que presente por pantalla: el carácter 'y', el entero 11 y el real 10.2.

```
#include <stdio.h>

int main(void){
   char c = 'y';
   int e = 11;
   float f = 10.2;

   printf ("Las variables valen: %c, %d, %f\n", c, e, f);

   return 0;
}
```

```
$ ./Tema2_2_TiposC.exe
Las variables valen: y, 11, 10.200000
```

4. Entrada/Salida

- Ejemplo de uso con printf y varios tipos (float, int, char)
  - Escribir un programa que presente por pantalla: el carácter 'y', el entero 11 y el real 10.2.

```
#include <stdio.h>

int main(void){
   char c = 'y';
   int e = 11;
   float f = 10.2;

   printf ("Las variables valen: %c, %f, %f\n", c, e, f);

   return 0;
}
```

¿Qué pasaría si en lugar de utilizar el especificador %d para el entero se utilizase %f?

4. Entrada/Salida

- Ejemplo de uso con printf y varios tipos (float, int, char)
  - Escribir un programa que presente por pantalla: el carácter 'y', el entero 11 y el real 10.2.

```
#include <stdio.h>

int main(void){
   char c = 'y';
   int e = 11;
   float f = 10.2;

   printf ("Las variables valen: %c, %f, %f\n", c, e, f);

   return 0;
}
```

Los 4 bytes de la variable entera e los interpretraría como un numero binario en coma flotante (un float)

```
$ ./Tema2_2_TiposC.exe
Las variables valen: y, 0.000000, 10.200000
```

4. Entrada/Salida. printf()

- Las especificaciones de formato siguen el siguiente patrón:

```
% [alineación] [ancho] [.precisión] [tamaño] especificador
```

4. Entrada/Salida. printf()

- Especificador

c	Imprime un carácter simple
d o i	Entero decimal con signo
e	Notación científica (mantisa/exponente) usando el carácter e
E	Notación científica (mantisa/exponente) usando el carácter E
f	Real en punto flotante (float)
lf	Real en punto flotante (double)
g	Usa la más corta de %e o %f
G	Usa la más corta de %E o %f
o	Octal con signo

s	(dirección de una cadena) Sttring de caracteres. Imprime los caracteres desde la dirección dada hasta encontrar el carácter nulo
u	Entero sin signo
x	Entero hexadecimal sin signo
X	Entero hexadecimal sin signo (mayúsculas)
P	(puntero) Imprime la dirección de un puntero
N	No se imprime nada
%	Imprime el carácter %

4. Entrada/Salida. printf()

- [alineación]

-	Justificación izda dentro del ancho del campo. Por defecto, justificación dcha
+	Fuerza a incluir delante del número el signo + o – incluso para los números positivos
Espacio	Si no se ha escrito signo , se inserta un espacio en blanco antes del valor
#	Si se usa con los especificadores o, x o X,, al valor de la variable se le precede de 0 0x o 0X respectivamente. Si se usa con los especificadores e, E y f, obliga a la salida a contener un punto decimal incluso si después no hay dígitos. Por defecto, si no hay dígitos detrás, no se escribe el punto decimal. Si se usa con los especificadores g o G, el resultado es el mismo que con e o E pero los ceros finales se eliminan
0	Rellena a la izda con 0 en lugar de espacios

4. Ejemplo. Variables y Tipos de Datos en C

- Codificar un programa para comprobar el tamaño en bytes y los valores máximo y mínimo de un float y de un double positivo o negativo.

```
#include <stdio.h>
#include <limits.h>
#include <float.h>

int main(int argc, char** argv) {

    printf("El numero de bits de un float es           :   %d\n", sizeof(float));
    printf("El mayor valor de un float positivo es     :   %g\n", (float) FLT_MAX);
    printf("El menor valor de un float positivo es     :   %g\n", (float) FLT_MIN);
    printf("El mayor valor de un float negativo es     :   %g\n", (float) -FLT_MAX);
    printf("El menor valor de un float negativo es     :   %g\n", (float) -FLT_MIN);
    printf("El numero de bits de un double es          :   %d\n", sizeof(double));
    printf("El mayor valor de un double positivo es    :   %g\n", (double) DBL_MAX);
    printf("El menor valor de un double positivo es    :   %g\n", (double) DBL_MIN);
    printf("El mayor valor de un double negativo es    :   %g\n", (double) -DBL_MAX);
    printf("El menor valor de un double negativo es    :   %g\n", (double) -DBL_MIN);
    printf("El numero de bits de un long double es     :   %d\n", sizeof(long double));
    printf("El mayor valor de un long double positivo es    :   %le\n", (double) LDBL_MAX);

    return 0;
}
```

4. Entrada/Salida. printf()

- [ancho]

número	Número mínimo de caracteres a imprimir. Si el valor a imprimir es más corto que este número, el resultado se rellena con espacios en blanco. El valor no se trunca incluso si el resultado es mayor
*	El ancho no se especifica en la cadena de formato, sino como un argumento de valor entero adicional que precede al argumento que se debe formatear.

4. Entrada/Salida. printf()

- [.precisión]

.número	• Para especificadores de enteros (d, i, o, u, x, X), la precisión especifica el número mínimo de dígitos que se escribirán.
	  • Si el valor a escribir es más corto que este número, el resultado se rellena con ceros a la izquierda.
	  • El valor no se trunca incluso si el resultado es más largo.
	  • Una precisión de 0 significa que no se escribe ningún carácter para el valor 0.
	• Para los especificadores e, E y f: este es el número de dígitos que se imprimirán después del punto decimal.
	• Para especificadores g y G: este es el número máximo de dígitos significativos que se imprimirán.
	• Para s: este es el número máximo de caracteres que se pueden imprimir.
	  • De forma predeterminada, todos los caracteres se imprimen hasta que se encuentra el carácter nulo final.
	• Para el tipo c, no tiene ningún efecto.
	• Cuando no se especifica ninguna precisión, el valor predeterminado es 1. Si el período se especifica sin un valor explícito de precisión, se supone 0.

.*	La precisión no se especifica en la cadena de formato, sino como un argumento de valor entero adicional que precede al argumento que se debe formatear

4. Entrada/Salida. printf()

- [tamaño]

h	El argumento se interpreta como un entero con o sin signo (solo aplica a especificadores de enteros i, d, o, u, s y X)
l	El argumento se interpreta como un entero largo (long int) con o sin signo para especificadores enteros (i, d, o, u, s y X) y como un carácter ancho o una cadena de caracteres amplia para los especificadores c y s
L	El argumento se interpreta como un largo doble (long double) (solo aplica a especificadores en punto flotantes (e, E, f, g y G)

4. Ejemplo. Entrada/Salida. printf()

- Uso de ancho, precisión y alineación con printf

```
int i = 123;
float = 1024.251
```

Variables enteras:

```
printf(":%5d:  \n",i);     --> :  123: Hay espacio suficiente
printf(":%-5d: \n",i);     --> :123  : Alineación izquierda
printf(":%05d: \n",i);     --> :00123: Llena con ceros
printf(":%2d:  \n",i);     --> :123:   No cabe, formato por defecto
```

Variables reales:

```
printf(":%12f:  \n",x);    --> : 1024.251000: Por defecto 6 dec.
printf(":%12.4f: \n",x);   --> :   1024.2510: .4 indica 4 decimales
printf(":%-12.4f: \n",x);  --> :1024.2510   : Alineación izquierda
printf(":%12.1f: \n",x);   --> :      1024.3: Redondea correctamente
printf(":%3f:  \n",x);     --> :1024.251000:  No cabe-> por defecto
printf(":%.3f: \n",x);     --> :1024.251:     Por defecto con 3 dec.

printf(":%12e:  \n",x);    --> :1.024251e+03: 12 caracteres
printf(":%12.4e: \n",x);   --> :  1.0243e+03: Idem. pero con 4 dec.
printf(":%12.1e: \n",x);   --> :    1.0e+03: Idem. pero con 1 dec.
printf(":%3e:  \n",x);     --> :1.024251e+03: No cabe. Por defecto
printf(":%.3e: \n",x);     --> :1.024e+03:    3 decimales.
```

4. Ejercicio. Entrada/Salida. printf()

- Ejemplo imprimir variables tipo float/double
  - Escribir un programa que visualice por pantalla una variable real definida como un float de valor 110.1 y otra definida como un double del mismo valor.

4. Ejercicio. Entrada/Salida. printf()

- Ejemplo variables tipo float/double
  - Escribir un programa que visualice por pantalla una variable real definida como un float de valor 110.1 y otra definida como un double del mismo valor.

```
#include <stdio.h>

int main(void){
   float a = 110.1;
   double b = 110.1;

   printf ("a: %f\n", a);
   printf ("b: %lf\n", b);

   return 0;
}
```

4. Ejercicio. Entrada/Salida. printf()

- Ejemplo variables tipo float/double
  - Escribir un programa que visualice por pantalla una variable real definida como un float de valor 110.1 y otra definida como un double del mismo valor.

```
#include <stdio.h>

int main(void){
   float a = 110.1;
   double b = 110.1;

   printf ("a: %f\n", a);
   printf ("b: %lf\n", b);

   return 0;
}
```

```
Alfonso Castro@DESKTOP-A3B7ER7 ~/2020_2021/Tema2
$ ./TipoVariables.exe
a: 110.099998
b: 110.100000
```

¿Por qué los valores son diferentes?

4. Entrada/Salida. scanf()

- La función **`scanf()`** lee, uno por uno, los caracteres procedentes de la entrada estándar (teclado).
- Los caracteres se convierten de acuerdo con las especificaciones de la cadena de formato y luego almacenados en las direcciones de memoria que se proporcionan a la función como parámetros entrada.

```
int  scanf("Cadena de formato" , Lista de direcciones);
```

- Todos los parámetros son obligatorios
- La "Cadena de formato" permite que se conviertan los caracteres de entrada en el formato esperado por las variables a las que se desea asignar un valor.
- La "Cadena de formato" está compuesta por una serie de especificadores de formato, similares a los empleados en la función printf.
- Debe haber un especificador de formato por cada dirección colocada, de lo contrario se podrá obtener resultados inesperados.

4. Entrada/Salida. scanf()

- La función **`scanf()`** por lo tanto leerá y transformará carácter por carácter el flujo de entrada, cuando encuentra un "blanco" se detiene, para proceder a asignar el valor convertido (según el formato especificado) en la dirección de memoria indicada en los parámetros.

- La función **`scanf()`** da por terminada la lectura de un dato cuando encuentra un espacio en blanco o el fin de línea (carácter '\n').

- Un carácter "blanco" puede ser
  - Uno o más espacios en blanco
  - Un tabulador (‘\t’)

- Si se desean leer varios datos (debe haber varios especificadores), se lee el primero y después se continua con el siguiente campo. Si al analizar un campo de entrada, los primeros caracteres corresponden a caracteres "blancos", éstos serán ignorados, procediéndose a trabajar a partir del primer carácter no blanco encontrado.

4. Entrada/Salida. scanf()

- La función `scanf()` devuelve como resultado un valor entero que representa el número de valores leídos correctamente. Si este valor es menor que el número previsto, o incluso cero, entonces es que ha ocurrido un error. Puede devolver también un EOF (End of File) si se alcanza el final del fichero antes de completar la lectura de datos.

- A continuación tienes un ejemplo de uso de `scanf()` en el que se lee un dato de tipo entero y se evalua el resultado de la función para determinar que el dato introducido es correcto. Observa que en el código se ha utilizado la variable `result` para almacenar el resultado de la función de lectura. Dicho resultado es un número, e indica el número de datos leídos correctamente por `scanf()`. Como nuestro objetivo es leer un dato de tipo `int`, comprobaremos si el resultado es distintito de 1 en cuyo caso el podemos determinar que el usuario ha introducido un dato no válido.

4. Ejercicio. Entrada/Salida. scanf()

- Codificar un programa que le pida al usuario que introduzca un entero y compruebe que el usuario ha introducido un entero.

4. Ejercicio. Entrada/Salida. scanf()

- Codificar un programa que le pida al usuario que introduzca un entero y compruebe que el usuario ha introducido un entero.

```
#include <stdio.h>


int main()
{
    int numero, result;
    printf("\n===> Por favor introduce un número entero\n");
    result = scanf("%d", &numero);
    if (result!=1)
       printf ("==> El dato introducido no es correcto. Debe ser un número entero\n");
    else
       printf(" ===> CORRECTO!!, el número leido es el entero %d\n",numero);
    return 0;
}
```

5. Constantes

- Una constante es un objeto cuyo valor no puede cambiar a lo largo de la ejecución de un programa
  - **Constantes enteras**
    - Son una sucesión de dígitos precedidos o no por el signo + o – dentro de un rango determinado

```
127,  -577
```

  - **Constantes reales**
    - Son una sucesión de dígitos con un punto delante, al final o en el medio y seguidos opcionalmente de un exponente

```
54.335     .75     37.     43e-4   -28E7   19.e+4
```

5. Constantes

- **Constantes carácter**
  - Es un carácter del código ASCII encerrado entre apóstrofos (**comillas simples**)

```
'A'     'l'
```

- **Constantes cadena**
  - Es una secuencia de caracteres encerrados entre comillas (**comillas dobles**).
  - En memoria, las cadenas se almacenan como una secuencia de caracteres ASCII más \0 o nulo que es definido en C mediante la constante NULL.

```
"Hola"     "123"     "12 de abril de 2005"
```

5. Constantes

- **Constantes definidas (simbólicas)**
  - Constantes que reciben nombres simbólicos mediante la directiva (#define)

```
#define NUEVALINEA '\n'
#define PI 3.1415929
```

- **Constantes enumeradas**
  - Las constantes enumeradas crean listas de elementos afines

```
enum estaciones {primavera, verano, otonio, invierno};
```

Cuando se procesa esa sentencia el compilador enumera los identificadores comenzando por 0.

5. Constantes

- **Constantes declaradas `const`**

  Si necesitamos almacenar un valor en una variable que nunca será cambiado el lenguaje C nos permite definir una constante con el calificador "const":

  ```
  const tipo nombre = valor;
  ```

  - El cualificador const permite dar nombres simbólicos a constantes de cualquier tipo soportado en C.
  - Su valor no puede ser modificado por el programa

  Una constante a diferencia de una macro ocupa un espacio durante la ejecución del programa. En el siguiente ejemplo, es importante notar que pi reserva espacio para un valor float durante la ejecución del programa, en cambio la macro PI cuando compilamos se sustituye todas las partes que aparece PI por el valor 3.1416.

  ```
  const float pi = 3.1416;
  #define PI 3.1415929
  ```

- **Constantes declaradas volatile**
  - La palabra reservada `volatile` actúa como `const`, pero su uso indica que el valor puede ser modificado además de por el propio programa, por el hardware o por el software del sistema. El uso de volatile le dice al compilador que no puede confiar en sus optimizaciones, es decir: si tiene que leer de memoria, lo hará; si tiene que escribir a ella, lo hará.

    (Nota: esto es porque a veces si se tiene que leer un valor desde memoria y se va a usar varias veces se intentará dentro de lo posible mantenerlo en un registro de la CPU para no tener que leerlo una y otra vez (puede suponer una fuente enorme de retrasos en el código). También se puede aprovechar para no escribir resultados intermedios si no son necesarios)

6. Ejercicio. Entrada/Salida

- Alineación y precisión con printf
  - Crea un programa que imprima números enteros y números en punto flotante alineados a la derecha y con una precisión de dos decimales.

6. Ejercicio. Entrada/Salida

- Alineación y precisión con printf
  - Crea un programa que imprima números enteros y números en punto flotante alineados a la derecha y con una precisión de dos decimales.

```
#include <stdio.h>
int main()
{
    int numero_entero = 42;
    float numero_flotante = 3.14159;
    printf("Entero: %10d\n", numero_entero);
    printf("Flotante: %10.2f\n", numero_flotante);
    return o;
}
```

```
$ ./printf01.exe
Entero:         42
Flotante:       3.14
```

6. Ejercicio. Entrada/Salida

- Tamaños y conversiones con printf
  - Escribe un programa que muestre en hexadecimal y octal un número entero ingresado por el usuario.

6. Ejercicio. Entrada/Salida

- Tamaños y conversiones con printf
  - Escribe un programa que muestre en hexadecimal y octal un número entero ingresado por el usuario.

```
#include <stdio.h>

int main() {
    int numero;

    printf("Ingrese un número entero: ");
    scanf("%d", &numero);

    printf("Decimal: %d\n", numero);
    printf("Hexadecimal: 0x%x\n", numero);
    printf("Octal: 0%o\n", numero);

    return 0;
}
```

```
$ ./printf02.exe
Ingrese un número entero: 15
Decimal: 15
Hexadecimal: 0xf
Octal: 017
```

6. Ejercicio. Entrada/Salida

- Usando tamaños y conversiones con scanf
  - Crea un programa que solicite una dirección IP en formato octal (por ejemplo, 052.013.014.010) y la convierta a decimal para mostrarla.

6. Ejercicio. Entrada/Salida

- Usando tamaños y conversiones con scanf
  - Crea un programa que solicite una dirección IP en formato octal (por ejemplo, 052.013.014.010) y la convierta a decimal para mostrarla.

```
$ ./printf03.exe
Ingrese una dirección IP en formato octal (xxx.xxx.xxx.xxx): 111.111.111.111
Dirección IP en decimal: 73.73.73.73
```

```
#include <stdio.h>

int main() {
    int ip_part1, ip_part2, ip_part3, ip_part4;

    printf("Ingrese una dirección IP en formato octal (xxx.xxx.xxx.xxx): ");
    scanf("%o.%o.%o.%o", &ip_part1, &ip_part2, &ip_part3, &ip_part4);

    printf("Dirección IP en decimal: %d.%d.%d.%d\n", ip_part1, ip_part2,
ip_part3, ip_part4);

    return 0;
}
```

6. Ejercicio. Entrada/Salida

- Escribir un programa que solicite al usuario en ancho y el largo de un rectángulo y visualice su área con cuatro decimales.

6. Ejercicio. Entrada/Salida

- Escribir un programa que solicite al usuario en ancho y el largo de un rectángulo y visualice su área con cuatro decimales

```
#include <stdio.h>
void main (){
        float x,y;

        printf(" Dame el largo de un rectangulo\n");
        scanf ("%f",&x);
        printf(" Dame el largo de un rectangulo\n");
        scanf ("%f",&y);
        printf(" El area es %10.4f", x*y);
}
```

6. Ejercicio. Entrada/Salida

- Escribir un programa en el que se introducen como datos de entrada la longitud de una carretera con tres números enteros que representan kilómetros, decámetros y metros respectivamente. Se ha de escribir, con un rótulo representativo, la longitud en metros.

6. Ejercicio. Entrada/Salida

- Escribir un programa en el que se introducen como datos de entrada la longitud de una carretera con tres números enteros que representan kilómetros, decámetros y metros respectivamente. Se ha de escribir, con un rótulo representativo, la longitud en metros

```
#include <stdio.h>
void main (){
	int kilometros, decametros, metros;

	printf("Introduzca kilometros, decametros y metros
");
	scanf("%d %d %d",&kilometros, &decametros,
&metros);
	metros = kilometros*1000+decametros*10+metros;
	printf (" Numero de metros es %d \n",metros);
}
```

6. Ejercicio. Entrada/Salida

- La equivalencia entre la masa y la energía se establece por la expresión de la teoría de la relatividad $E=mc^2$, donde d es la velocidad de la luz $c= 2.997925 \times 10^{10}$ cm/sg. Escribir un programa que lea una masa en gramos y obtenga la cantidad de energía producida cuando la masa se convierte en energía

6. Ejercicio. Entrada/Salida

- La equivalencia entre la masa y la energía se establece por la expresión de la teoría de la relatividad $E=mc^2$, donde d es la velocidad de la luz $c= 2.997925 \times 10^{10}$ cm/sg. Escribir un programa que lea una masa en gramos y obtenga la cantidad de energía producida cuando la masa se convierte en energía

```
#include <stdio.h>

int main() {
    float m, energia;
    const float c = 2.997925e10;

    printf("Introduzca masa: ");
    scanf("%f", &m);
    energia = m * c * c;
    printf("Energia en ergios: %e\n", energia);

    return 0;
}
```

CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL