



# Tema 2: Lenguaje C: Elementos Básicos

Introducción a la programación I

Marcos Novalbos  
Elena García Gamella

# Índice

---

1. Estructura general de un programa en C
2. Elementos de un programa en C
3. Compilación de un programa en C
4. Variables y Tipos de Datos en C
5. Constantes

# 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

Ejemplo.c



## 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

### Ejemplo.c

```
#include <...>  
#define .....
```

} Directivas de preprocesador

```
int variable=0;  
void func2();
```

} Variables globales y prototipos

```
.....
```

```
.....
```

## 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

### Ejemplo.c

```
#include <...>  
#define .....
```

} Directivas de  
preprocesador

```
int variable=0;  
void func2();
```

} Variables globales y  
prototipos

## 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 “;”

### Ejemplo.c

```
#include <...>  
#define .....
```

} Directivas de  
precompilación

```
int variable=0;  
void func2();
```

} Variables globales y  
prototipos

```
void main(){  
}
```

} Función principal

## 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 de estructura general de un programa en C

---

```
/* Nombre programa: Area de un circulo de radio 3
   Autor:
   Fecha: 1-10-2020 */

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

/* Declaraciones globales */
void AreaCirculo3 ();
float area;

/* 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);
}
```

## 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**

- Un identificador es una secuencia de caracteres, letras, dígitos y subrayados.
- El primer carácter tiene que ser una letra (no 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, pero se la tratará como así lo fuera

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	wigned	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 <sup>(1)</sup>

<code>_Bool</code>	<code>_Imaginary</code>	<code>restrict</code>
<code>_Complex</code>	<code>_inline</code>	

- 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

<code>asm</code>	<code>_ds</code>	<code>_huge</code>	<code>pascal</code>
<code>cdecl</code>	<code>_es</code>	<code>interrupt</code>	<code>_ss</code>
<code>_cs</code>	<code>far</code>	<code>near</code>	

(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 bloque 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.

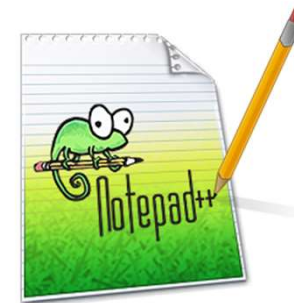
- **Archivos cabecera**

- 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`

### 3. Compilación de un programa en C

---

- La creación de un ejecutable de un programa en C sigue los siguientes 3 pasos:
  - Generación de las líneas de código (construcción o codificación del programa).
  - Compilación del programa
  - Enlazado del programa con las funciones de biblioteca que necesita
- Herramientas
  - Entorno cygwin
    - [Descarga de paquete preparado](#)
    - Compilador basado en gcc
  - Editor Notepad ++



`$ gcc circulo.c -o circulo.exe`



## 3. Compilación de un programa en C

### Breve repaso sobre comando útiles de Linux

- **pwd**
  - El comando `pwd` muestra cuál es el directorio de trabajo actual, en otras palabras, le dice al usuario dónde está posicionado dentro del árbol de directorios del sistema
- **ls**
  - El comando `ls` muestra el contenido del directorio actual. Por defecto, los archivos ocultos no se muestran. Éste es seguramente el comando que más se utiliza
  - Se pueden añadir opciones a `ls`, por ejemplo
    - `ls -a` muestra todos los archivos, incluyendo los ocultos (cuyo nombre comienza por un punto),
    - `ls -l` muestra un listado detallado, con la última fecha de modificación de cada archivo, el tamaño, etc.,
    - `ls -h` muestra el tamaño de los ficheros en bytes, Kb, Mb, etc.
  - Todas las opciones disponibles, tanto para `ls` como para el resto de comandos se pueden consultar mediante las páginas del manual, con el comando `man` seguido del comando del cual se quiere obtener información

## 3. Compilación de un programa en C

---

### Breve repaso sobre comando útiles de Linux

- **cd**
  - El comando `cd` (change dir) permite cambiar de directorio. Si se utiliza tal cual, sin ningún tipo de argumento, cambia al directorio de trabajo del usuario (HOME). Si se utiliza seguido de una ruta, cambia al directorio que se indica.
- **mkdir**
  - Se pueden crear directorios con el comando `mkdir`.
- **rmdir**
  - Permite borrar directorios siempre que estos estén vacíos.

## 3. Compilación de un programa en C

---

### Breve repaso sobre comando útiles de Linux

- **cat**
  - El comando cat muestra por pantalla el contenido de un fichero y, cuando termina, el usuario está otra vez de vuelta en la línea de comandos (prompt)
- **more**
  - El comando more hace lo mismo que cat, a diferencia de que muestra el fichero pantalla a pantalla, es decir, llena de texto la pantalla y se espera a que el usuario pulse la tecla <espacio> para pasar a la siguiente.
- **less**
  - El comando less es el más versátil de los tres, ya que permite moverse hacia delante y hacia atrás dentro del fichero, utilizando los cursores o las teclas de “AvPág” y “RePág”:

## 3. Compilación de un programa en C

---

### Breve repaso sobre comando útiles de Linux

- touch
  - El comando touch permite crear un fichero vacío. Con cualquier editor de texto se puede crear un fichero vacío, pero con touch es especialmente cómodo y rápido
- vi
  - Editor de ficheros
- mv
  - Este comando permite mover archivos entre directorios, y renombrarlos
- rm
  - Este comando permite eliminar archivos

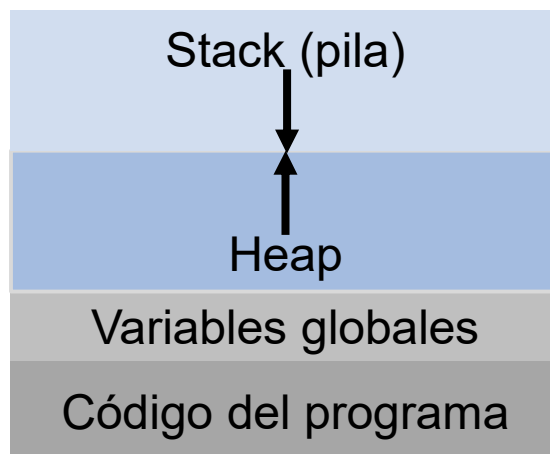
Nota: todos estos comandos se estudian en profundidad en la asignatura de LRSO..

## 3. Compilación de un programa en C

---

### ■ Mapa de memoria

- Un programa compilado en C crea una estructura de memoria con cuatro zonas o regiones lógicas diferentes de la memoria del ordenador cada una de las cuales se utilizan para funciones específicas.



- La pila se usa para muchas cosas durante la ejecución del programa:
  - Mantiene las direcciones de vuelta para las llamadas a funciones
  - Estado de la CPU en los cambios de contexto
  - Variables locales
- El heap es la región de memoria libre que usa el programa para la asignación de memoria dinámica.

## 4. Variables y Tipos de Datos en C

---

- **Una variable** es una posición 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 (",")

```
int i, j, k ;
```

```
double beneficio, gasto, precio;
```

- El nombre de una variable no tiene nada que ver con el tipo.

## 4. Variables y Tipos de Datos en C

---

- 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.
- Los tipos de datos básicos en C son:
  - Carácter **char**
  - Entero **int**
  - Real de simple precisión **float**
  - Real de doble precisión **double**
  - Sin tipo **void**
    - Declara explícitamente que una función no devuelve ningún valor
    - Crea punteros genéricos (de cualquier tipo)

## 4. 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



## 4. Variables y Tipos de Datos en C

Tipo	Tamaño en bits	Intervalo
char	8	0 .. 255
short int	16	(16 bits) -32.766 .. 32.767
int	32	(32 bits) -2.147.483.648 ... 2.147.483.647
long	32 (x86_32)	-2.147.483.648 .. 2.147.483.647
long	64 (x86_64)	-9.223.372.036,854.775.808 ... 9.223.372.036.854.775.807
float	32	$3,4 * 10^{-38} .. 3,4 * 10^{+38}$
double	64	$1,7 * 10^{-308} .. 1,7 * 10^{+308}$

## 4. 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.
- Para la representación de estos caracteres utiliza el código ASCII. Dichos códigos ASCII toman valores del intervalo 0 a 255
- 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 character = 'b';
character = character - 32
```

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

ASCII: American Standard Code for Information Interchange

## 4. Variables y Tipos de Datos en C

## Tabla códigos ASCCI

Caracteres ASCII de control			Caracteres ASCII imprimibles				ASCII extendido (Página de código 437)									
00	NULL	(caràcter nulo)	32	espacio	64	@	96	`	128	Ç	160	á	192	Ł	224	Ó
01	SOH	(inicio encabezado)	33	!	65	A	97	a	129	ü	161	í	193	Ł	225	ß
02	STX	(inicio texto)	34	"	66	B	98	b	130	é	162	ó	194	ŧ	226	Ô
03	ETX	(fin de texto)	35	#	67	C	99	c	131	â	163	ú	195	ƒ	227	Õ
04	EOT	(fin transmisión)	36	\$	68	D	100	d	132	ä	164	ñ	196	—	228	ö
05	ENQ	(consulta)	37	%	69	E	101	e	133	à	165	Ñ	197	+ †	229	Ö
06	ACK	(reconocimiento)	38	&	70	F	102	f	134	å	166	ª	198	ˆ	230	µ
07	BEL	(timbre)	39	'	71	G	103	g	135	ç	167	º	199	˜	231	þ
08	BS	(retroceso)	40	(	72	H	104	h	136	ê	168	¿	200	ℓ	232	p
09	HT	(tab horizontal)	41	)	73	I	105	i	137	ë	169	®	201	⌋	233	ú
10	LF	(nueva línea)	42	*	74	J	106	j	138	è	170	¬	202	⌈	234	Û
11	VT	(tab vertical)	43	+	75	K	107	k	139	ï	171	½	203	⌋	235	Ü
12	FF	(nueva página)	44	,	76	L	108	l	140	î	172	¾	204	⌋	236	ý
13	CR	(retorno de carro)	45	-	77	M	109	m	141	ì	173	¡	205	=	237	Ý
14	SO	(desplaza afuera)	46	.	78	N	110	n	142	Ä	174	«	206	≠	238	~
15	SI	(desplaza adentro)	47	/	79	O	111	o	143	Å	175	»	207	□	239	‘
16	DLE	(esc.vínculo datos)	48	0	80	P	112	p	144	É	176	•	208	◊	240	≡
17	DC1	(control disp. 1)	49	1	81	Q	113	q	145	æ	177	◌̇	209	∅	241	±
18	DC2	(control disp. 2)	50	2	82	R	114	r	146	Æ	178	◌̈	210	Ê	242	≡
19	DC3	(control disp. 3)	51	3	83	S	115	s	147	ô	179	◌̉	211	Ě	243	¼
20	DC4	(control disp. 4)	52	4	84	T	116	t	148	ö	180	◌̊	212	Ě	244	¶
21	NAK	(conf. negativa)	53	5	85	U	117	u	149	ò	181	◌̋	213	ı	245	§
22	SYN	(inactividad sinc)	54	6	86	V	118	v	150	û	182	◌̌	214	İ	246	÷
23	ETB	(fin bloque trans)	55	7	87	W	119	w	151	ù	183	◌̍	215	İ	247	°
24	CAN	(cancelar)	56	8	88	X	120	x	152	ÿ	184	©	216	İ	248	·
25	EM	(fin del medio)	57	9	89	Y	121	y	153	Ö	185	◌̎	217	Ј	249	˙
26	SUB	(sustitución)	58	:	90	Z	122	z	154	Û	186	◌̏	218	┐	250	˘
27	ESC	(escape)	59	;	91	[	123	{	155	ø	187	◌̐	219	█	251	˚
28	FS	(sep. archivos)	60	<	92	\	124		156	£	188	◌̑	220	■	252	˛
29	GS	(sep. grupos)	61	=	93	]	125	}	157	Ø	189	€	221	▬	253	˜
30	RS	(sep. registros)	62	>	94	^	126	~	158	×	190	¥	222	▬	254	■
31	US	(sep. unidades)	63	?	95	_			159	f	191	γ	223	▬	255	nbsp
127	DEL	(suprimir)														

## 4. Variables y Tipos de Datos en C

---

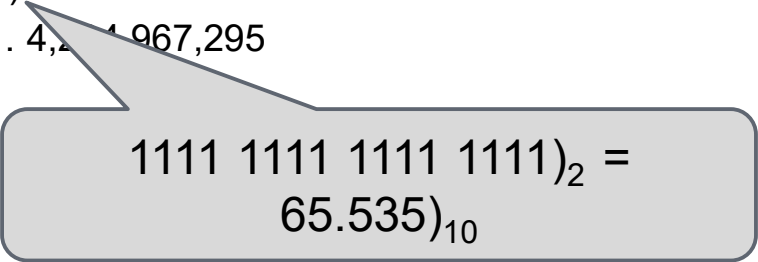
### ▪ **int (entero)**

- 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 de dicho número
  - 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

## 4. Variables y Tipos de Datos en C

### ■ Int (entero)

- Los enteros se almacenan internamente en 2 o 4 bytes de memoria.
- El uso del 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


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

## 4. 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.

$$N = s \cdot M \cdot r^E$$

$$\begin{array}{cc} \text{signo} & \text{exponente} \\ \underbrace{\quad} & \underbrace{\quad} \\ + 6.02 \cdot 10^{-23} \\ \underbrace{\quad} & \underbrace{\quad} \\ \text{mantisa} & \text{base} \end{array} \quad \begin{array}{cc} \text{signo} & \text{exponente} \\ \underbrace{\quad} & \underbrace{\quad} \\ + 1.01110 \cdot 2^{-1101} \\ \underbrace{\quad} & \underbrace{\quad} \\ \text{mantisa} & \text{base} \end{array}$$



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

## 4. 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

## 4. 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.



## 4. 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.

## 4. Variables y Tipos de Datos en C

---

### ■ Parámetros formales

- Si una función va a usar argumentos, debe declarar las variables que van a aceptar esos valores de los argumentos. Esas variables son los parámetros formales de la función.
- Se comportan como cualquier otra variable local de la función.

```
/* Función que calcula el área de un cuadrado */  
int AreaCuadrado (int lado) {  
    int Area;  
    area = lado * lado;  
    return (area)  
}
```

## 4. 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.

## 4. 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*/
}
```

## 4. 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*/
}
```

```
Alfonso Castro@DESKTOP-A3B7ER7 ~/2020_2021/Tema2
$ ./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
```

## 4. 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*/
}
```

## 4. 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*/
}
```

```
Alfonso Castro@DESKTOP-A3B7ER7 ~/2020_2021/Tema2
$ ./VariblessGlobales.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
```

## 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 apóstrofes (**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`**

- El cualificador `const` permite dar nombres simbólicos a constantes.
- Su valor no puede ser modificado por el programa

```
const tipo_nombre = valor;  
#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. Entradas y salidas

---

- 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);
```

## 6. Entradas y salidas

---

- 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 lectura y escritura 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);
```

## 6. Entradas y salidas

---

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

## 6. Entradas y salidas

---

- 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.

## 6. Entradas y salidas

---

- 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;  
}
```

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



## 6. Entradas y salidas. 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

## 6. Entradas y salidas. printf

---

- Las especificaciones de formato siguen el siguiente patrón:

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

## 6. Entradas y salidas. 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) String 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 %

## 6. Entradas y salidas. 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

## 6. Entradas y salidas. 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.

## 6. Entradas y salidas. printf

### ▪ [precisión]

.número	<ul style="list-style-type: none"><li>• Para especificadores de enteros (d, i, o, u, x, X), la precisión especifica el número mínimo de dígitos que se escribirán.<ul style="list-style-type: none"><li>• Si el valor a escribir es más corto que este número, el resultado se rellena con ceros a la izquierda.</li><li>• El valor no se trunca incluso si el resultado es más largo.</li><li>• Una precisión de 0 significa que no se escribe ningún carácter para el valor 0.</li></ul></li><li>• Para los especificadores e, E y f: este es el número de dígitos que se imprimirán después del punto decimal.</li><li>• Para especificadores g y G: este es el número máximo de dígitos significativos que se imprimirán.</li><li>• Para s: este es el número máximo de caracteres que se pueden imprimir.<ul style="list-style-type: none"><li>• De forma predeterminada, todos los caracteres se imprimen hasta que se encuentra el carácter nulo final.</li></ul></li><li>• Para el tipo c, no tiene ningún efecto.</li><li>• 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.</li></ul>
.*	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

## 6. Entradas y salidas. 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)

## 6. Entradas y salidas. printf

- Ejemplos:

**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. Variables y Tipos de Datos en C

---

- 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.

## 4. Variables y Tipos de Datos en C

---

- 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. Variables y Tipos de Datos en C

- 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?

## 6. Entradas y salidas. 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.

## 6. Entradas y salidas. 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.
- Un carácter "blanco" puede ser
  - Uno o más espacios en blanco
  - Un tabulador ('t')
  - Un cambio de línea ('n').
- Después de esto, 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.



CENTRO UNIVERSITARIO  
DE TECNOLOGÍA Y ARTE DIGITAL