

Introducción a Lenguaje C

Jornadas de Octubre 2009

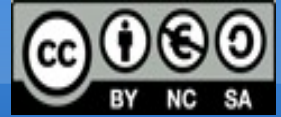
Grupo de Usuarios de Linux

Universidad Carlos III

Tania Pérez



Introducción a Lenguaje C



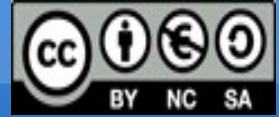
El lenguaje C es:

- Estructurado
- Portable.
- Flexible, veloz y potente.
- Fácil modificación.
- Compilado.

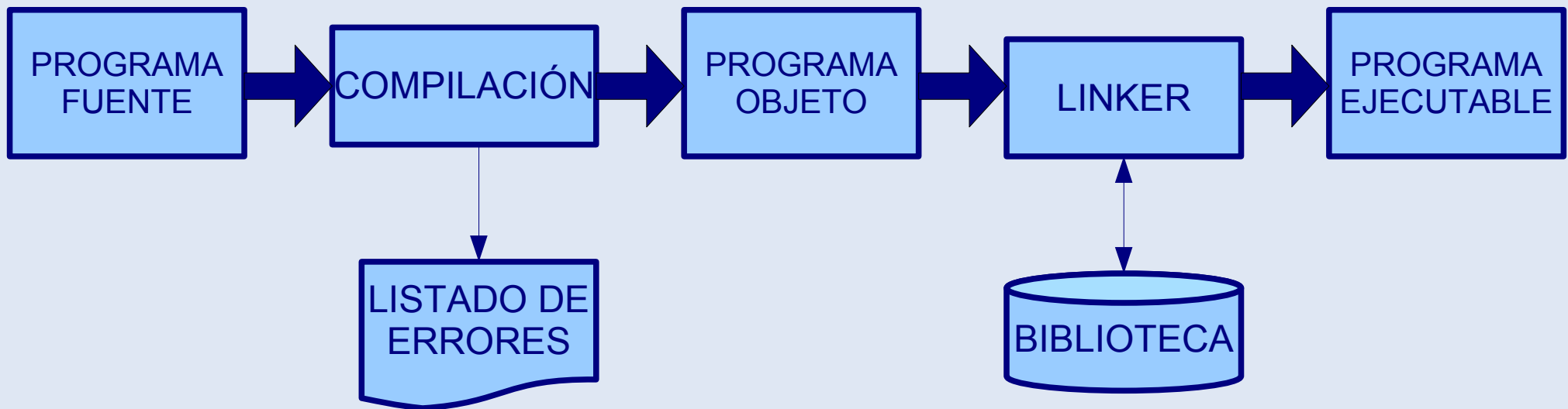
Algunas características de C

- Case Sensitive
- Las sentencias terminan con el símbolo punto y coma ‘;’
- Se pueden establecer **bloques** de instrucciones mediante el uso de llaves: los símbolos abre-llave ‘{’ y cierra-llave ‘}’.

Introducción a Lenguaje C



- Compilador de C



Estructura de un programa en C



Introducción a Lenguaje C



■ Primer programa en C

```
/* Primer programa en C */
```

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

```
int main () {
```

```
    printf ( " ¿Quieres ser un gulero? \n ");
```

```
    printf ( " Te esperamos en el Ed. Sabatini
```

```
        Los segundos viernes de cada
```

```
        Mes, 2.3.C05 \n");
```

```
    return 0;
```

```
}
```

Comentario

Acceso a biblioteca
de funciones

Función principal
del programa

Instrucción de salida
por pantalla

Segundo programa en C

```
/* Segundo programa en C */
```

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

```
void main () {
```

```
    int numero;
```

```
    numero = 9;
```

```
    Printf ( "Valor de la variable numero: %d \n ",numero);
```

```
}
```

Declaración de variables

Asignación de valor
a variables

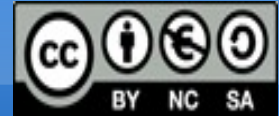
Instrucción de
salida por pantalla

Tercer programa en C

```
/* Tercer programa en C */  
  
#include <stdio.h>  
  
void main (void) {  
  
    int x;  
  
    printf ("introduce el valor de x: ");  
    scanf ( "%d",&x);  
    Printf ("el valor introducido de x es: %d ", x );  
}
```

→ Instrucción de entrada de datos por teclado

Introducción a Lenguaje C



Tipos de datos en C

Type	Description	Size	Domain
char	Signed character/byte. Characters are enclosed in single quotes.	1	-128..127
double	Double precision number	8	ca. $10^{-308}..10^{308}$
int	Signed integer	4	$-2^{31}..2^{31} - 1$
float	Floating point number	4	ca. $10^{-38}..10^{38}$
long (int)	Signed long integer	4	$-2^{31}..2^{31} - 1$
long long (int)	Signed very long integer	8	$-2^{63}..2^{63} - 1$
short (int)	Short integer	2	$-2^{15}..2^{15} - 1$
unsigned char	Unsigned character/byte	1	0..255
unsigned (int)	Unsigned integer	4	$0..2^{32} - 1$
unsigned long (int)	Unsigned long integer	4	$0..2^{32} - 1$
unsigned long long (int)	Unsigned very long integer	8	$0..2^{64} - 1$
unsigned short (int)	Unsigned short integer	2	$0..2^{16} - 1$

Introducción a Lenguaje C

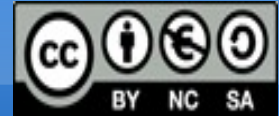


TABLA DE CÓDIGOS ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

TYPDEF

- Permite redefinir el nombre de un tipo de datos existente para poder usar el nuevo nombre como si fuera un tipo estándar de C.

`typedef [tipo_datos] [nuevo_nombre];`

Ejemplos:

- `typedef int entero;`
- `typedef float real;`
- `typedef char character;`

DECLARACIÓN DE CONSTANTES

- CONSTANTE = Valor fijo que no puede ser alterado a lo largo del programa.
- Se pueden definir constantes de cualquier tipo de datos simples.

Const [tipo de datos] [nombre de la constane] = [valor];

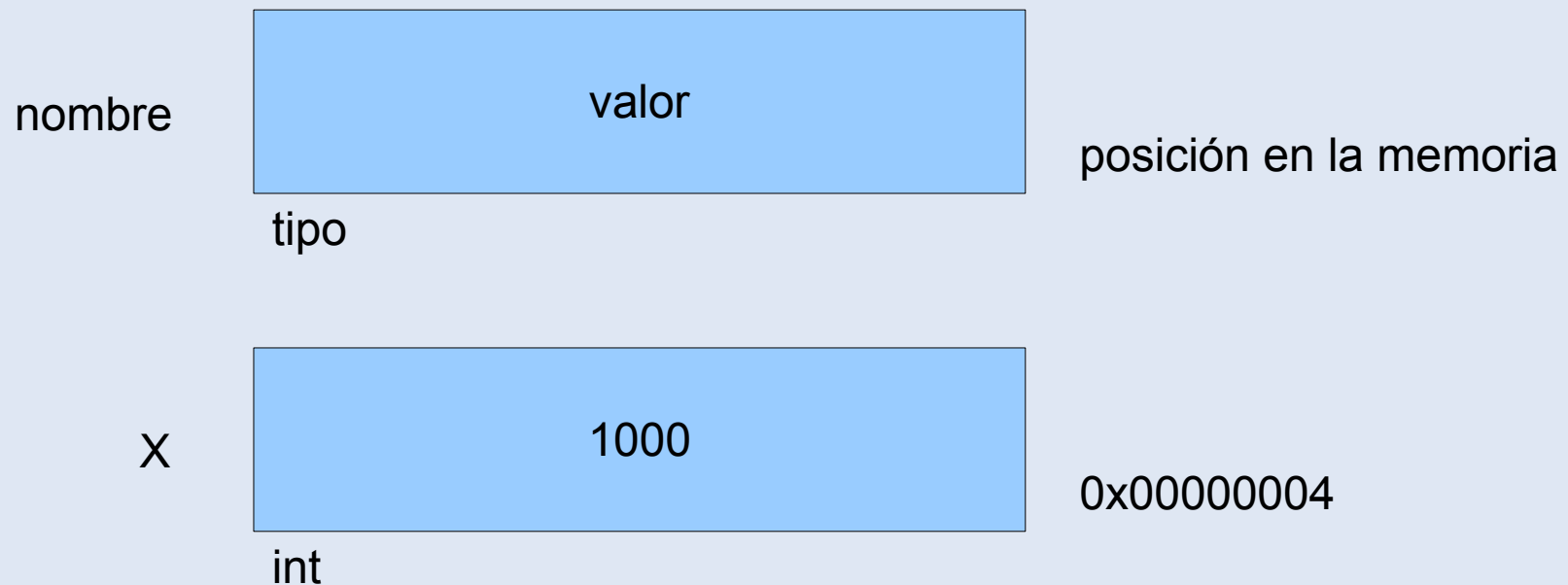
#define [nombre de la constante] [valor]

Ejemplos:

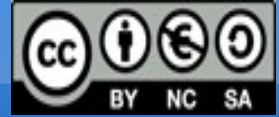
- #define MAX 1000
- Const int DIAS = 365;

DECLARACIÓN DE VARIABLES

- VARIABLE = Posiciones de memoria a las que se les asigna un nombre (identificador) para poder referenciarla y se usan para mantener valores que puedan ser modificados durante la ejecución del programa.



Introducción a Lenguaje C



- Para usar cualquier variable, es decir, asignarle un valor, antes debe ser declarada.
- Se pueden declarar en varias partes del programa.
 - Al principio del programa: **Variables globales**
 - Dentro de una función : **Variables locales**
 - En la definición de funciones: **parámetros formales**

[tipo de datos] [nombre variable 1, nombre variable2, ...];

Ejemplo:

- char opcion, letra;
- int dias;

Introducción a Lenguaje C



Operadores aritméticos y de asignación

© carlospes.com	
<i>Prioridad de los operadores aritméticos, de índice de un array, de llamada a una función, relacionales, lógicos, de asignación y de conversión de tipo (de mayor a menor) en C:</i>	
() []	Llamada a una función e índice de un array
+ - ++ -- ! (<tipo>)	Signo más, signo menos, incremento, decremento, negación y conversión de tipo
* / %	Multiplicación, división, módulo
+ -	Suma y resta
< <= > >=	Menor que, menor o igual que, mayor que, mayor o igual que
== !=	Igual que y distinto que
&&	Conjunción
	Disyunción
= += -= *= /= %=	Operadores de asignación

PRIMERAS FUNCIONES DE E/S POR CONSOLA

printf

- La función printf() imprime por pantalla el texto, las constantes y las variables que se indiquen.

int printf("cadena_de_control", tipo arg1, tipo arg2, ...);

Ejemplo:

```
int main(){
    char *nombre = "Maria";
    int edad= 21;
    printf("Hola me llamo %s y tengo %d anyos" , nombre, edad);
    return 0;
}
```

Introducción a Lenguaje C



■ printf formato

Cadena Formato	Tipo de Expresión	Resultado Impreso
%d %i	short, int	número entero
%ld %li	long	
%u	unsigned short, unsigned int	número entero sin signo
%lu	unsigned long	
%o	[unsigned] short, [unsigned] int	número entero octal sin signo
%lo	[unsigned] long	
%x %X	[unsigned] short, [unsigned] int	número entero hexadecimal sin signo
%lx %lX	[unsigned] long	
%f	float, double, long double	número real en coma fija
%e %E %g %G	float, double, long double	número real en notación científica
%c	char, unsigned char	se imprime un símbolo ASCII
%s	char*	se imprime una cadena de símbolos
%p	void*	se imprime el valor de un puntero

Scanf

- Se utiliza para leer los datos por teclado.

`int scanf ("%x%x2...", &arg1, &arg2, ...);`

- Devuelve como valor de retorno el número de conversiones de formato realizadas con éxito.

Ejemplo:

```
int main(){
    int edad;
    int sueldo;
    scanf("%d%f",&edad,&sueldo);
    printf("Sue edad es &d, y su sueldo es %f" , edad, sueldo);
    return 0;
}
```

Introducción a Lenguaje C



■ Scanf formato

Cadena Formato	Tipo de Expresión	Resultado Impreso
%d %i	short, int	número entero
%ld %li	long	
%u	unsigned short, unsigned int	número entero sin signo
%lu	unsigned long	
%o	[unsigned] short, [unsigned] int	número entero octal sin signo
%lo	[unsigned] long	
%x %X	[unsigned] short, [unsigned] int	número entero hexadecimal sin signo
%lx %lX	[unsigned] long	
%f	float, double, long double	número real en coma fija
%e %E %g %G	float, double, long double	número real en notación científica
%c	char, unsigned char	se imprime un símbolo ASCII
%s	char*	se imprime una cadena de símbolos
%p	void*	se imprime el valor de un puntero

Introducción a Lenguaje C



- **getchar()** → En caso de éxito lee un sólo carácter introducido por teclado y lo devuelve convertido en un unsigned int.
- **putchar()** → Permite introducir un solo carácter pasado como argumento, por pantalla.
- **getch()** y **getche()** → Esperan hasta que se pulse la letra e, inmediatamente devuelve el valor sin esperar a que se pulse ENTER. <conio.h>
- **gets** y **puts()** → Se usan para leer cadenas de caracteres por teclado hasta que se pulsa enter (gets) o para imprimir una cadena por pantalla (puts)
- **fflush (stdin)** → Para evitar la lectura de caracteres de control, limpia el buffer de teclado. Es recomendable utilizarla antes de cualquier operación de lectura de un carácter.

Estructuras de Control

- Sentencias Condicionales
 - `if, if-else`
 - `switch-case`
 - Operador Condicional ?

- Sentencias Iterativas o Bucles
 - `while`
 - `do-while`
 - `for`

■ Sentencia IF

...

```
if(expresion_logica)  
    sentencia;
```

...

Si `expresion_logica` se evalúa como verdadera (1) entonces se ejecuta `sentencia`. Si `expresion_logica` se evalúa como falsa (0) entonces `sentencia` no se ejecuta y el programa continua ejecutándose en la siguiente línea.

■ Sentencia IF – ELSE

...

```
if (expresion_logica)
```

```
    sentencia_1;
```

```
else
```

```
    sentencia_2;
```

...

Si `expresion_logica` se evalúa como verdadera (1) entonces se ejecuta `sentencia_1`. Si `expresion_logica` se evalúa como falsa (0) entonces se ejecuta `sentencia_2`.

Introducción a Lenguaje C



Ejemplo:

```
#include <stdio.h>
main () {
    int edad,acceso;
    acceso = 1;
    printf("Introduzca su edad por teclado: ");
    scanf ("%d",&edad);
    if ( edad < 18 )
    {
        printf("No tiene edad suficiente para acceder");
        acceso = 0;
    }
    else if ( edad == 18 )
        printf("Bienvenido y enhorabuena por su mayoría de edad");
    else
        printf ("Bienvenido");
}
```

■ Sentencia switch – case

```
...
switch (expresion_entera)
{
    case expresion_cte_1:
        sentencia_1;
        [break;]
    case expresion_cte_2:
        sentencia_2;
        [break;]
    ...
    case expresion_cte_n:
        sentencia_3;
        [break;]
    [default:
        sentencia_default;]
}
...
```

- Si `expresion_entera` es igual a alguna de la expresiones constantes `{expresion_cte_1, ..., expresion_cte_n}` entonces se ejecuta la sentencia correspondiente de entre las sentencias `{sentencia_1, ..., sentencia_n}`.
- Si `expresion_entera` no coincide con ninguna de las expresiones constantes `{expresion_cte_1, ..., expresion_cte_n}` se ejecuta la sentencia `sentencia_default` asociada con la sentencia default.
- Las sentencias `break` saltan el resto de la sentencia condicional `switch`.

Ejemplo:

```
#include <stdio.h>
main () {
    int opcion,x;
    printf("Introduzca la opcion deseada por teclado: ");
    scanf ("%d",&opcion);
    switch (opcion)
    {
        case 1: printf("Ha elegido la opción 1");
        case 2: printf("Ha elegido la opción 2");
        default: printf( "Opción incorrecta");
    }
}
```

■ While

...

```
while (expresion_logica)  
    sentencia;
```

...

Mientras `expresion_logica` se evalúe como verdadera (1) se ejecuta `sentencia` repetidamente. Cuando `expresion_logica` se evalúe como falsa (0) entonces la ejecución continua en la siguiente sentencia de programa.

Ejemplo:

```
#include <stdio.h>
main () {
    int var,contador;
    contador=0;
    printf("¿Cuántos asteriscos desea escribir ? ");
    scanf ("%d", &var);
    while (var>contador)
    {
        printf("*");
        contador ++;
    }
}
```

■ Do-while

...

do

 sentencia;

while (expresion_logica);

...

- Inicialmente se ejecuta `sentencia` una vez. Luego mientras `expresion_logica` se evalúe como verdadera (1) `sentencia` se ejecuta repetidamente. Cuando `expresion_logica` se evalúe como falsa (0) entonces la ejecución continua en la siguiente sentencia de programa.

Ejemplo:

```
#include <stdio.h>
main () {
    int var,contador;
    contador=0;
    printf("¿Cuántos asteriscos desea escribir ? ");
    scanf ("%d", &var);
    do
    {
        printf("*");
        contador ++;
    }while (var > contador);
}
```

- Aunque var=0 siempre se ejecuta al menos una vez el bloque de sentencias.

■ For

...

```
for(sentencia_inicializacion; expresion_logica; sentencia_actualizacion)  
    sentencia;
```

...

- Inicialmente se ejecuta `sentencia_inicializacion`. A continuación se evalúa `expresion_logica`. Si se evalúa como verdadera (1) se ejecuta `sentencia`, y luego `sentencia_actualizacion`. Luego se vuelve a evaluar `expresion_logica` y si es verdadera se vuelve a ejecutar `sentencia`, y luego `sentencia_actualizacion`. Esto se hará repetidamente hasta que `expresion_logica` se evalúe como falsa (0), en ese punto el bucle finaliza y la ejecución continua en la siguiente sentencia de programa.

Ejemplo:

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

    int var,contador;
    contador=0;
    printf("¿Cuántos asteriscos desea escribir ? ");
    scanf ("%d", &var);
    for (contador=0; var>contador; contador++)
    {
        printf("*");
    }

}
```

■ Funciones

- ✓ Permite la división de problemas genéricos en tareas más sencillas.
- ✓ Subalgoritmo que es utilizable por otros algoritmos.
- ✓ Realiza una tarea determinada.
- ✓ Trabaja en base a unos datos que se le suministran (argumentos).
- ✓ Puede devolver o no un resultado.
- ✓ Todo programa al menos debe contener al función `main()`.

Ventajas del uso de funciones

- ✓ Disminuye el tamaño del programa
- ✓ Evita duplicación de sentencias para procesos que realicen las operaciones con diferentes datos
- ✓ Permite mayor claridad/legibilidad de los programas fuentes.
- ✓ Facilita las modificaciones posteriores de los programas.
- ✓ Facilita la depuración de errores.
- ✓ Todo subalgoritmo puede ser probado independientemente del conjunto general del programa
- ✓ Se pueden confeccionar bibliotecas de funciones o módulos de uso general
- ✓ Ayuda a la productividad y eficiencia del programador (puede utilizar código ya hecho)

SINTAXIS DE LA DEFINICIÓN DE UNA FUNCIÓN

```
tipo_del_parámetro_de_salida NombreFunción ( declaración_lista_parámetros )  
{  
    Declaración de variables  
    Sentencias  
    Devolución de resultados [return(valor_de_vuelta); o simplemente return;]  
}
```

Función que calcula el cuadrado de un número:

```
#include <stdio.h>
int cuadrado (int x)
{
    x*=x;
    return x;
}
Main () {
```

Definición de la
función cuadrado

```
    int num,cuadrado;
    printf("Introduce un numero: ");
    scanf ("%d", &num);
```

```
    c=cuadrado(num);
    printf("El cuadrado de %d es : %d\n",c);
```

Llamada a la función
cuadrado

```
}
```

Declaración de una función

- Consiste en la declaración de todas las características de una función excepto el código de la misma
- Sintaxis de la declaración de una función:

tipo_del_parámetro_de_salida NombreFunción (declaración_lista_parámetros)

Ejemplo: int **cuadrado**(int n);

- En la lista de parámetros no es necesario especificar los identificadores
- En C no necesita declararse una función si la primera llamada a la misma se produce después de la definición
- Los prototipos se pueden escribir en los llamados ficheros de cabecera (.h) y luego incluirlos en nuestro programa fuente con `#include`

- Paso de parámetros
 - Paso por valor
 - Se pasa una copia temporal de la variable
 - Se puede modificar el valor de la copia, pero no el valor de la variable original
 - En todos los ejemplos anteriores el paso de parámetros es por valor
 - Paso por referencia
 - Se pasa la dirección en memoria de la variable
 - Se puede modificar el valor de la variable original pasada
 - Requiere el uso de punteros

VECTORES

- Un vector es una colección de un número determinado de elementos del mismo tipo.
- Los elementos de un vector se almacenan de forma consecutiva en la memoria principal del ordenador.
- Sintaxis de la declaración de un array unidimensional en C

tipoBase nombreArray[expresionConstante];

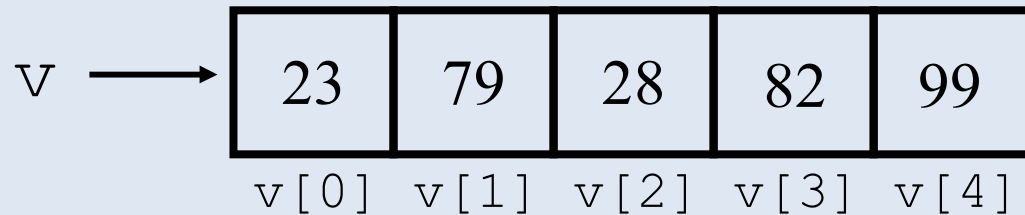
Ejemplo: int vector[5];

- Para acceder a cada componente del array se escribe el nombre seguido de la posición o índice, entre corchetes, de esa componente dentro del vector

Introducción a Lenguaje C



- Los arrays comienzan en el elemento 0 y termina en el elemento n-1.
- Para desplazarnos por un vector, siempre hay que declarar un índice cuyo valor será 0 hasta n-1.
- Para listar los elementos de un array es recomendable utilizar un bucle for.



```
int v[5] = {23,79,28,82,99}
```

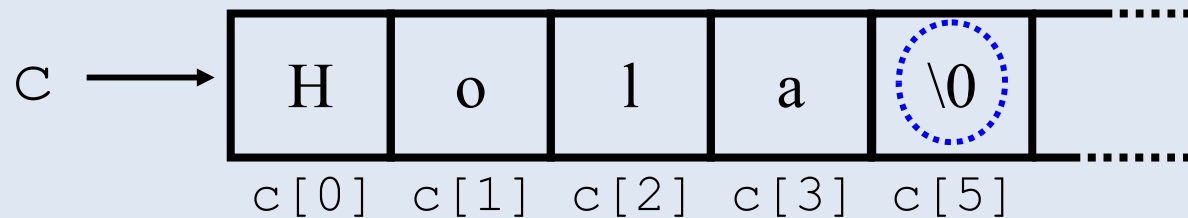
- **Ejemplo: Lectura y escritura de un array de enteros**

```
void main(void)
{
    int v[10]; // array de 10 enteros
    int i;
    for (i=0; i<10; i++)
    {
        printf("Valor %d: ", i);
        scanf("%d", &v[i]);
    }
    printf("Valores leidos:\n");
    for (i=0; i<10; i++)
        printf("Componente %d: %d\n", i, v[i]);
}
```

■ Cadenas de caracteres

Una cadena es un array unidimensional acabado en el caracter nulo '\0'.

- Una cadena de n caracteres necesita un array de n+1 elementos, puesto que hay que añadir el carácter '\0' que indica el fin de la cadena



- En la inicialización se puede omitir el tamaño del array: `char cadena[]="hola";`
- Existen numerosas funciones de librería en C para el tratamiento de cadenas, cuyos prototipos se encuentran en `<string.h>`

Dudas y sugerencias, consultar a:

Grupo de Usuarios de Linux – Carlos III de Madrid (Leganés):

- Lista correo GUL UC3M:
gul@gul.uc3m.es
- DESPACHO GUL: 2.3C05 (Ed. Sabatini)
(Segundo viernes de cada mes)

Muchas gracias por venir