

PROGRAMACIÓN EN LENGUAJE C. APUNTES

Introducción

- C es un lenguaje estructurado de nivel medio, ni de bajo nivel como ensamblador, ni de alto nivel como Ada o Haskell. Esto permite una mayor flexibilidad y potencia, a cambio de menor abstracción.
- No se trata de un lenguaje fuertemente tipado, lo que significa que se permite casi cualquier conversión de tipos. No es necesario que los tipos sean exactamente iguales para poder hacer conversiones, basta con que sean parecidos.
- No lleva a cabo comprobación de errores en tiempo de ejecución, por ejemplo no se comprueba que no se sobrepasen los límites de los arrays. El programador es el único responsable de llevar a cabo esas comprobaciones.
- Tiene un reducido numero de palabras clave, unas 32 en C89 y 37 en C99.
- C dispone de una *biblioteca estándar* que contiene numerosas funciones y que siempre está disponible, además de las extensiones que proporcione cada compilador o entorno de desarrollo.

En resumen, es un lenguaje muy flexible, muy potente, muy popular, pero que no protege al programador de sus errores.

Tipos de datos

C ofrece una colección de tipos de datos bastante limitada, aunque no por ello poco funcional. Dicha colección se compone de los siguientes tipos:

- char:
 - **Contenido:** Puede contener un caracter del conjunto de caracteres locales
 - **Tamaño:** Normalmente 1 byte.
- int:
 - **Contenido:** Un número entero
 - **Tamaño:** El determinado por la arquitectura para números enteros. En arquitecturas Intel/x86 es 4 bytes
- float:
 - **Contenido:** Un número en coma flotante
 - **Tamaño:** El determinado por la arquitectura para números en coma flotante. En arquitecturas Intel/x86 es 4 bytes

- double
 - **Contenido:** Un número en coma flotante de precisión doble
 - **Tamaño:** El determinado por la arquitectura para números en coma flotante de doble precisión. En arquitecturas Intel/x86 es 8 bytes

Variables

DEFINICIÓN: **Variable:** *Espacio de memoria, referenciado por un identificador, en el que el programador puede almacenar datos de un determinado tipo.*

Declarar una variable es indicar al compilador que debe reservar espacio para almacenar valores de un tipo determinado, que serán referenciados por un identificador determinado. En C debemos declarar **todas** las variables antes de usarlas, establecer el tipo que tienen y, en los casos que sea necesario, darles un valor inicial.

A la hora de declarar una variable debemos tener en cuenta diversas restricciones :

- Los nombres de variables se componen de letras, dígitos y el caracter de subrayado _.
- El primer caracter del nombre debe ser una letra o el carácter de subrayado.
- Las letras mayúsculas y minúsculas son distintas en C.
- Las palabras reservadas del lenguaje no se pueden usar como nombres de variable.

Constantes

Las constantes son valores fijos que no pueden ser modificados por el programa. Pueden ser de cualquier tipo de datos básico (punteros incluidos).

También podemos marcar en nuestro programa que queremos que una variable sea constante. Para ello utilizamos la palabra reservada `const` tal que:

```
const int dummy = 321;    /* declaramos que dummy vale y valdrá siempre 321 */
```

No tiene demasiado sentido declarar una variable de tipo `const` sin darle valor inicial, pero algunos compiladores permiten hacerlo.

Constantes numéricas:

Aparte de constantes enteras tipo 234 y en coma flotante de la forma 10.4, a veces, sobre todo al trabajar a bajo nivel, resulta más cómodo poder introducir la constante en base 8 (octal) o 16 (hexadecimal) que en base 10. Dado que es corriente usar estos sistemas de numeración, C permite especificar constantes enteras en hexadecimal u

octal. Una constante hexadecimal empieza por 0x seguido del número esa base.

Iguálmente una constante octal comienza por 0:

const int hex = 0x80A; /* 2058 en decimal */

const int oct = 012; /* 10 en decimal */

Constantes de caracteres:

Cuando se escribe un carácter cualquiera entre comillas, el valor de esa constante es el que se especifica en la tabla de códigos ASCII:

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Uso del preprocesador para definir constantes simbólicas

Con el preprocesador, del que hablaremos más adelante, también se pueden definir constantes, aunque en este caso son realmente alias. Al compilar se sustituye tal cual un valor por otro, tenga o no sentido, y no se reserva memoria para el valor. Por ejemplo, para sustituir en el código fuente el número 10 por el alias LONGITUD, se escribe:

```
#define LONGITUD 10
```

La directiva `#define` tiene otra poderosa característica: el nombre de macro puede tener argumentos. Cada vez que el compilador encuentra el nombre de macro, los argumentos reales encontrados en el programa reemplazan los argumentos asociados con el nombre de la macro. Por ejemplo:

```
#define MIN(a,b) (a < b) ? a : b
```

```
main()
{
    int x=10, y=20;

    printf("EL minimo es %d\n", MIN(x,y) );
}
```

Cuando se compila este programa, el compilador sustituirá la expresión definida por `MIN(x,y)`, excepto que `x` e `y` serán usados como los operandos. Así después de que el compilador hace la sustitución, la sentencia `printf` será ésta:

```
printf("El minimo es %d\n", (x < y) ? x : y);
```

Como se puede observar donde se coloque `MIN`, el texto será reemplazado por la definición apropiada. Por lo tanto, si en el código se hubiera puesto algo como:

```
x = MIN(q+r,s+t);
```

después del preprocesamiento, el código podría verse de la siguiente forma:

```
x = ( q+r < s+t ) ? q+r : s+t;
```

Operadores

En C existen una gran variedad de operadores, que se pueden agrupar de la siguiente manera:

- Operadores aritméticos.
- Operadores relacionales.
- Operadores lógicos.
- Operadores a nivel de bit (bitwise operators).
- Operadores especiales.

Operadores aritméticos

Los operadores aritméticos nos permiten, básicamente, hacer cualquier operación aritmética, que necesitemos (ejemplo: suma, resta, multiplicación, etc). En la siguiente tabla se muestran los operadores de los que disponemos en C y su función asociada.

Nota: Todos ellos aceptan operandos de cualquier tipo excepto el Módulo, Incremento y Decremento, que sólo acepta operadores enteros.

Los incrementos y decrementos se pueden poner de forma prefija y postfija:

a) Forma prefija: preincremento y predecremento

Cuando un operador de incremento o decremento precede a su operando, se llevará a cabo la operación de incremento o de decremento antes de utilizar el valor del operando. Veámoslo con un ejemplo:

```
int x,y;  
x = 2004;  
y = ++x;  
/* x e y valen 2005. */
```

b) Forma postfija: postincremento y postdecremento

En el caso de los postincrementos y postdecrementos pasa lo contrario: se utilizará el valor actual del operando y luego se efectuará la operación de incremento o decremento.

```
int x,y  
  
x = 2004;  
y = x++;  
/* y vale 2004 y x vale 2005 */
```

Operadores relacionales

Al igual que en matemáticas, estos operadores nos permitirán evaluar las relaciones (igualdad, mayor, menor, etc) entre un par de operandos (en principio, pensemos en números). Los operadores relacionales de los que disponemos en C son:

> Mayor que
>= Mayor o igual que
< Menor que
<= Menor o igual que
== Igual
!= Distinto

El resultado de cualquier evaluación de este tipo, es un valor "cierto" (true) o "falso" (false). La mayoría de lenguajes tienen algún tipo predefinido para representar estos valores (boolean, bool, etc); sin embargo en C, se utilizan valores enteros para representar esto:

falso (false)	0
cierto (true)	cualquier valor distinto de 0, aunque normalmente se usará el 1

Volviendo a los operadores relacionales, el resultado de una evaluación será un valor entre 0 y 1, que indicará como hemos dicho, la falsedad o certeza de esa relación.

Operadores lógicos

Como operadores lógicos designamos a aquellos operadores que nos permiten "conectar" un par de propiedades (al igual que en lógica):

```
numero = 2701;  
  
if ( EsPrimo(numero) && (numero > 1000) ){  
  
    /* Ejecutaremos este código si numero */
```

/* es primo y numero es mayor que 100 */

}

Los operadores lógicos de los que disponemos en C son los siguientes:

Tabla: Operadores lógicos.

Operador

Acción

&&

Conjunción (Y)

||

Disyunción (O)

!

Negación

Operadores a nivel de bit

Como operadores a nivel de bit entendemos aquellos que toman los operandos como un conjunto de bits y operan con esos bits de forma individual.

Tabla 3.5: Operadores a nivel de bit
Operador

Acción

&

AND a nivel de bit.

|
OR a nivel de bit.

^
XOR a nivel de bit.

-
Complemento.

<<
Desplazamiento a la izquierda.

>>
Desplazamiento a la derecha.

A continuación describiremos cada uno de estos operadores brevemente.

El **operador AND (&)**: El operador AND compara dos bits; si los dos son 1 el resultado es 1, en otro caso el resultado será 0. Ejemplo:

```
c1 = 0x45 --> 01000101
c2 = 0x71 --> 01110001
-----
c1 & c2 = 0x41 --> 01000001
```

El **operador OR (|)**: El operador OR compara dos bits; si cualquiera de los dos bits es 1, entonces el resultado es 1; en otro caso será 0. Ejemplo:

```
i1 = 0x47 --> 01000111
i2 = 0x53 --> 01010011
-----
i1 | i2 = 0x57 --> 01010111
```

El **operador XOR (^)**: El operador OR exclusivo o XOR, dará como resultado un 1 si cualquiera de los dos operandos es 1, pero no los dos a la vez. Ejemplo:

```
i1 = 0x47 --> 01000111
i2 = 0x53 --> 01010011
-----
i1 ^ i2 = 0x14 --> 00010100
```

El **operador de complemento (~)**: Este operador devuelve como resultado el complemento a uno del operando:

```
c = 0x45 --> 01000101
-----
~c = 0xBA --> 10111010
```

Los **operadores de desplazamiento a nivel de bit (<< y >>)**: Desplazan a la izquierda o a la derecha un número especificado de bits. En un desplazamiento a la izquierda los bits que sobran

ACADEMIA CARTAGENA99
C/ Cartagena 99, B° C , 28002 Madrid
91 51 51 321
academia@cartagena99.com.es

Cartagena99

www.cartagena99.com.es

por el lado izquierdo se descartan y se rellenan los nuevos espacios con ceros. De manera análoga pasa con los desplazamientos a la derecha.