Programación de dispositivos electrónicos

Unidad 1

Conceptos básicos de programación- Lenguaje C

Héctor Albano Diego Codevilla

Conceptos básicos sobre programación

- Un **algoritmo** *es* una secuencia precisa de operaciones (pasos) que resuelven un problema.
- Un **programa** es la transcripción de un algoritmo en un lenguaje de programación, capaz de ser procesado por una computadora, el cual controla su funcionamiento a la hora de resolver un problema.
- Las instrucciones se le deben dar en un cierto **lenguaje,** que la computadora sea capaz de comprender.
- Las computadoras utilizan un lenguaje llamado "**lenguaje máquina**" o "código máquina" el cual es difícil tanto a la hora de escribir el programa como de corregirlo.
- Por eso, en la práctica se emplean lenguajes más parecidos al lenguaje humano, llamados "lenguajes de **alto nivel**". Normalmente, estos son muy parecidos al idioma inglés, aunque siguen unas reglas mucho más estrictas.

Lenguajes de programación

- ***Lenguaje máquina.*** Son solo entendidos por la computadora ya que sus instrucciones son cadenas binarias. Si quisiera programar en este lenguaje tendría que conocer qué combinación de unos y ceros corresponde a cada instrucción, los diferentes códigos binarios para representar números y letras, conocer el mapa de la memoria, etc. Esto trae como inconvenientes: la dificultad de codificación, la poca fiabilidad, la dificultad grande para verificar y poner a punto y, además, sólo ejecutable en el procesador específico.

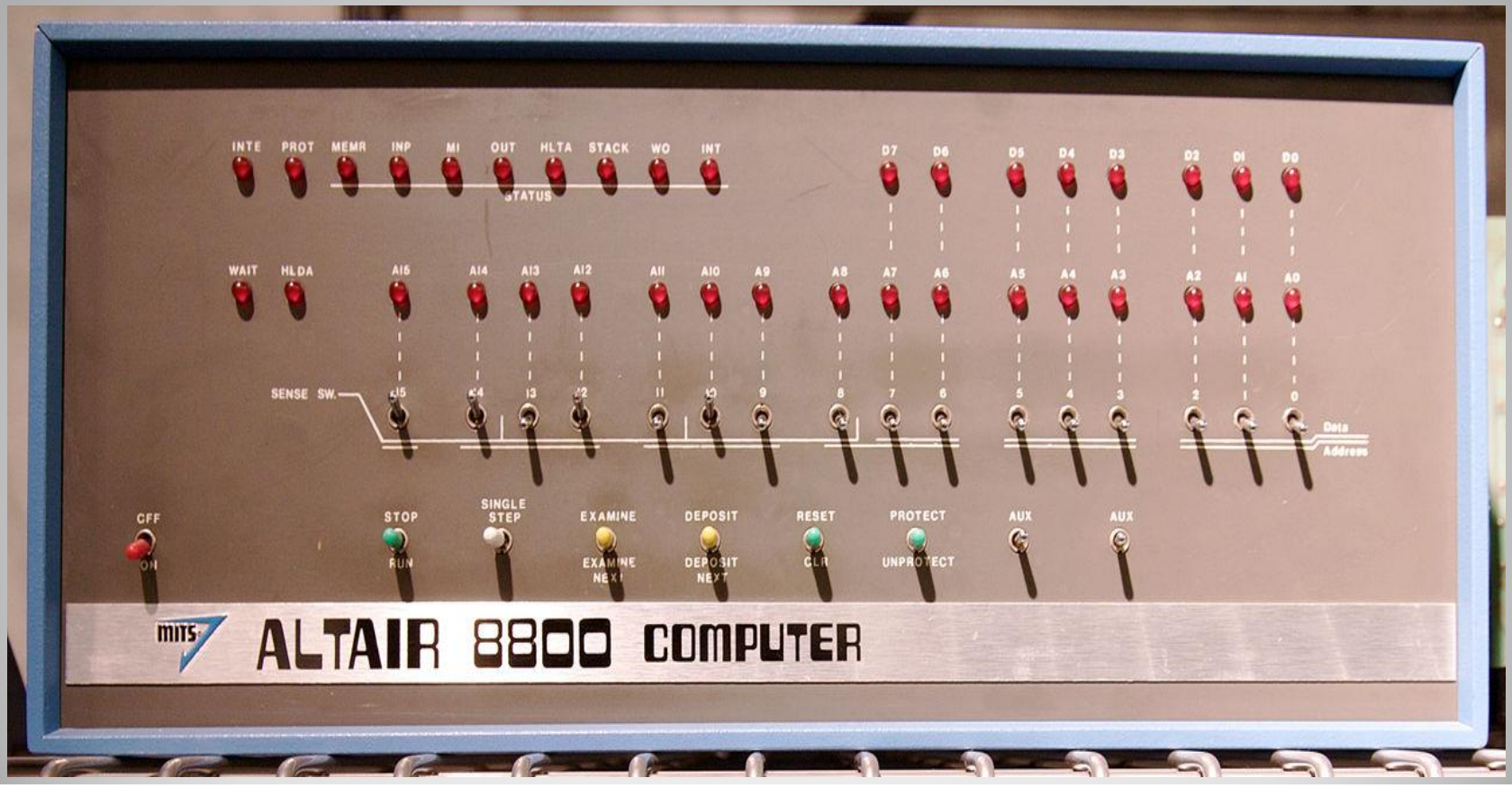
Por ejemplo para mostrar en la pantalla "Hello world!“ en una PC debería escribir:

```
00100001 00001010 00000000 00000000 00001100 00010000 00000000 00000110
01101111 01110010 01101100 01100100 00001000 00010000 00000000 00000110
01101111 00101100 00100000 01010111 00000100 00010000 00000000 00000110
01001000 01100101 01101100 01101100 00000000 00010000 00000000 00000110
00000000 00010000 00000000 00000110 00010000 00000000 00000000 00000000
00000001 00000000 00000000 00000000 00000100 00000000 00000000 00000000
10000000 00000000 00000000 00000000 00000001 00000000 00000000 00000000
10000000 00000000 00000000 00000000
```

o en hexadecimal mas sencillo de leer:

```
21 0a 00 00 0c 10 00 06
6f 72 6c 64 08 10 00 06
6f 2c 20 57 04 10 00 06
48 65 6c 6c 00 10 00 06
00 10 00 06 10 00 00 00
01 00 00 00 04 00 00 00
80 00 00 00 01 00 00 00
80 00 00 00
```

Una computadora personal muy popular entre los años 74 y 75 permitía ingresar el código binario de cada instrucción accionando interruptores.



- ***Lenguaje de bajo nivel* (ensamblador).** Dependen de la máquina donde se ejecuta y es difícil de programar, pero es mas fácil de codificar que el lenguaje máquina. Se requiere un programa ensamblador para traducir al código de maquina.

  El mismo programa en Assembler de PC

```
section     .text
global      _start
_start:
mov     edx,len
mov     ecx,msg
mov     ebx,1
mov     eax,4
int     0x80
mov     eax,1
int     0x80
section     .data
msg     db  'Hello, world!',0xa
len     equ $ - msg
```

- ***Lenguajes de alto nivel*.** Son independientes de la máquina, no dependen del diseño del hardware, son muy portables. Más fáciles de programar y entender. La sintaxis usada está más cerca del lenguaje humano que de la máquina. Inconvenientes: Tiempo de ejecución mayor y no se aprovechan los recursos internos de la máquina eficientemente. Ej C, C++, Visual Basic, Java, Pascal, etc. En el caso del C se requiere de un compilador para llevarlo al código de máquina.

  El mismo programas escrito en C

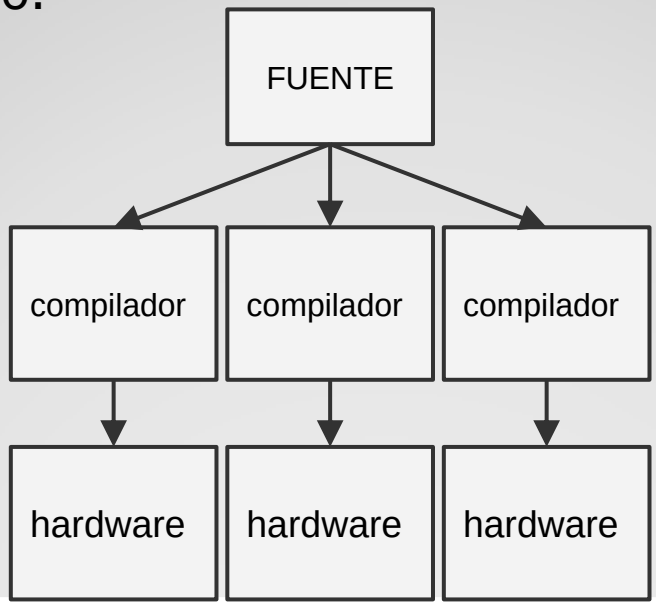
```
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
}
```

Ensambladores, compiladores e intérpretes

- Las órdenes que nosotros hemos escrito (lo que se conoce como "programa **fuente**") deben convertirse a lo que el ordenador comprende (obteniendo el "programa **ejecutable**"). Si elegimos un lenguaje de bajo nivel, como el ensamblador (en inglés Assembly, abreviado como Asm), la traducción es sencilla, y de hacer esa traducción se encargan unas herramientas llamadas **ensambladores** (en inglés Assembler).
- Cuando el lenguaje que hemos empleado es de alto nivel, la traducción es más complicada, y a veces implicará también recopilar varios fuentes distintos o incluir posibilidades que se encuentran en bibliotecas que no hemos preparado nosotros. Las herramientas encargadas de todo esto son los **compiladores**.

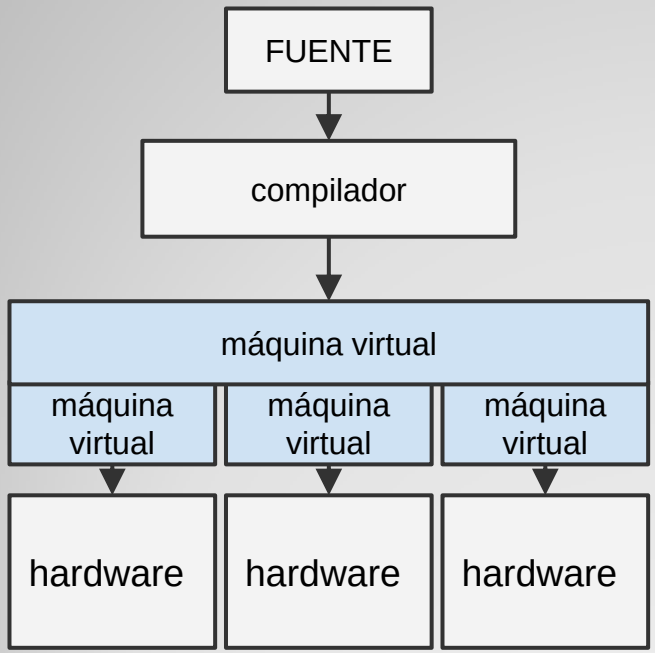
- El programa ejecutable obtenido con el compilador se podría hacer funcionar en otra computadora similar al que habíamos utilizado para crearlo, sin necesidad de que ese otra computadora tenga instalado el compilador.
- Por ejemplo, un programa hecho en Pascal, tendríamos un fichero fuente llamado SALUDO.PAS. Este fichero no serviría de nada en un ordenador que no tuviera un compilador de Pascal. En cambio, después de compilarlo obtendríamos un fichero SALUDO.EXE, capaz de funcionar en cualquier otro ordenador que tuviera el mismo sistema operativo, aunque no tenga un compilador de Pascal instalado.



Existe un compilador para cada lenguaje y para cada tipo de máquina

- Un **intérprete** es una herramienta parecida a un compilador, con la diferencia de que en los intérpretes no se crea ningún "programa ejecutable" capaz de funcionar "por sí solo", de modo que si queremos distribuir nuestro programa a alguien, deberemos entregarle el programa fuente y también el intérprete que es capaz de entenderlo, o no le servirá de nada. Cuando ponemos el programa en funcionamiento, el intérprete de encarga de convertir el programa en lenguaje de alto nivel a código máquina, orden por orden, justo en el momento en que hay que procesar cada una de las órdenes. Por ejemplo: Phyton,Ruby y PHP.

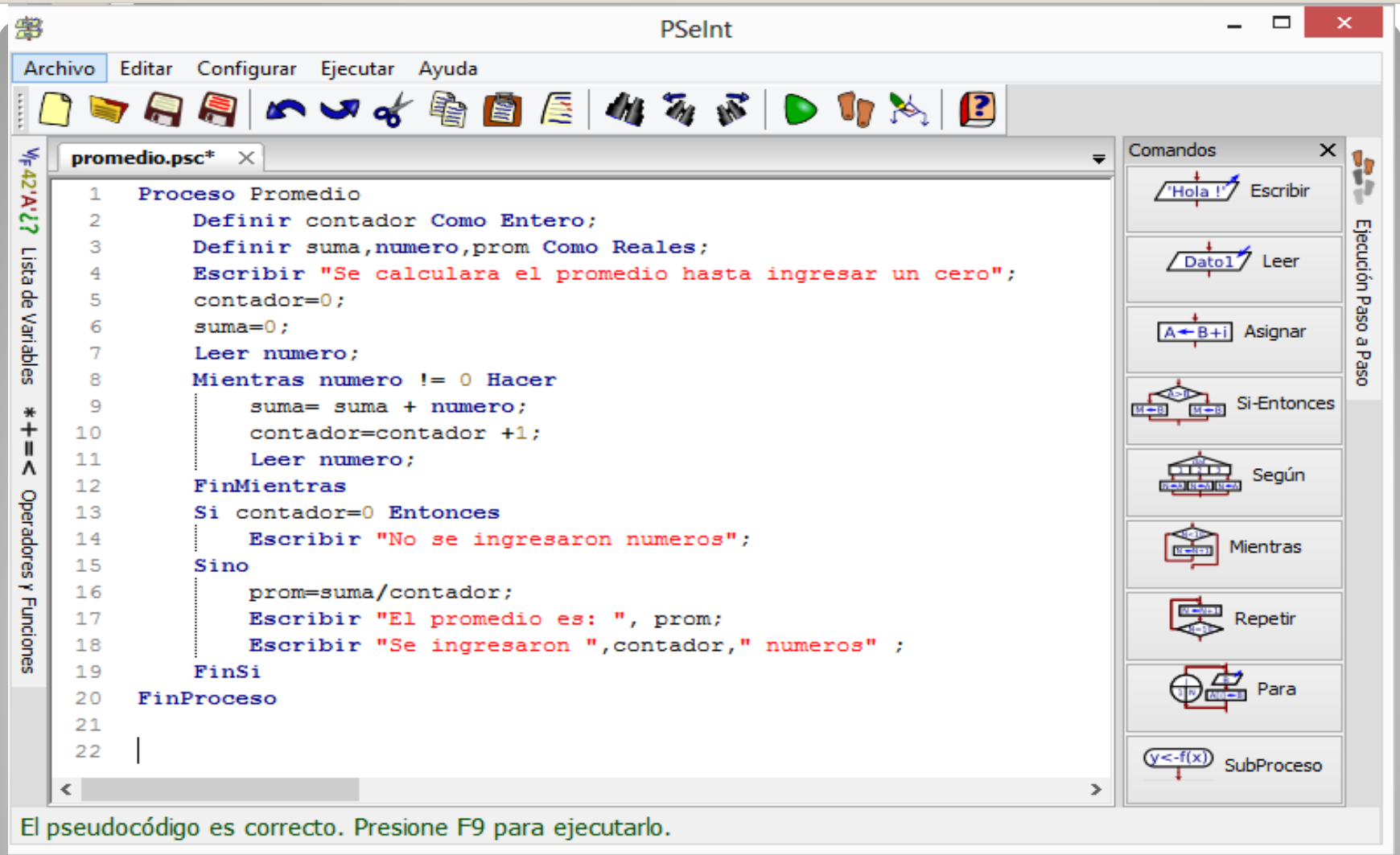
- Existen lenguajes que no se compilan para obtener un ejecutable para un ordenador concreto, sino un ejecutable "genérico", que es capaz de funcionar en distintos tipos de ordenadores, a condición de que en ese ordenador exista una "**máquina virtual**" capaz de entender esos ejecutables genéricos. Esta es la idea que se aplica en Java: los fuentes son ficheros de texto, con extensión ".java", que se compilan a ficheros ".class". Estos ficheros ".class" se podrían llevar a cualquier ordenador que tenga instalada una "máquina virtual Java" Esta misma idea se sigue en el lenguaje C#.



Existe un único compilador para la máquina virtual y una máquina virtual para cada tipo de máquina.

Pseudocódigo

- A pesar de que los lenguajes de alto nivel se acercan al lenguaje natural, que nosotros empleamos, es habitual no usar ningún lenguaje de programación concreto cuando queremos plantear los pasos necesarios para resolver un problema, sino emplear un lenguaje de programación ficticio, no tan estricto, muchas veces escrito incluso en español. Este lenguaje recibe el nombre de **pseudocódigo.** Un ejemplo es el siguiente:



PseInt http://pseint.sourceforge.net/

El Lenguaje C.

- Fue creado por Denis Ritchie en los Laboratorios Bell Telephone en 1972.
- Tenia como finalidad ser usado para crear el sistema operativo UNIX.
- Es poderoso y flexible de ahí su rápida difusión.
- Fue estandarizado por el American National Standards Institute (ANSI)
- Es un lenguaje de nivel medio: combina elementos de lenguajes de alto nivel con la funcionalidad del ensamblador. Esto le permite hacer cosas que otros lenguajes de alto nivel no pueden, como la manipulación de bits, bytes, direcciones y es tan fácil de usar como cualquier otro lenguaje de alto nivel.
- Se puede incorporar órdenes en lenguaje ensamblador en medio de un programa escrito en C (el programa ya no se podrá llevar a otros ordenadores que no usen el mismo lenguaje ensamblador).
- Es muy portable, es decir, es posible adaptar el software escrito para un tipo de ordenador en otro e incluso con diferentes sistemas operativos.
- Es un lenguaje de pocas palabras tiene solo 32 palabras clave.
- Es modular, usa mucho funciones.
- Tiene bibliotecas con funciones incorporadas, esto aumenta su potencia.
- A veces es complicado encontrar errores.

Pasos para crear un programa en C.

- **Programa:** Algoritmo (secuencia no ambigua, finita y ordenada de instrucciones para la resolución de un determinado problema) traducido a un lenguaje de programación, de modo que un ordenador es capaz de ejecutarlo.

- **Programación:** Elaboración de un programa de manera que éste sea:
  - **Correcto** Un programa será correcto ***si hace lo que debe hacer***, de modo que se deben especificar de manera muy clara cuáles son los datos sobre los que se trabajarán y lo que se debe hacer con ellos. Todo debe ser ***documentado y probado*** antes de desarrollarlo.
  - **Eficiente** Debe consumir la menor cantidad de recursos (tiempo y/o memoria) posible.
  - **Claro** Es muy importante la claridad y legibilidad de todo programa, ya que facilitará al máximo la tarea de mantenimiento posterior del software.
  - **Modular** Los programas suelen subdividirse en subprogramas (módulos), para reducir la complejidad de aquella parte que se está implementando y facilitar la reutilización de código.

Pasos para la elaboración y ejecución de un programa:

Los pasos a seguir los podemos resumir de la siguiente manera:

1 Escribir el código fuente con un editor de texto sin formato.

2 Compilar el fichero fuente

3 Si se producen errores de sintaxis (o warnings) volver al editor y eliminar los errores de sintaxis.

4 Si no hay errores se obtendrá el código objeto y el enlazador construirá el archivo ejecutable.

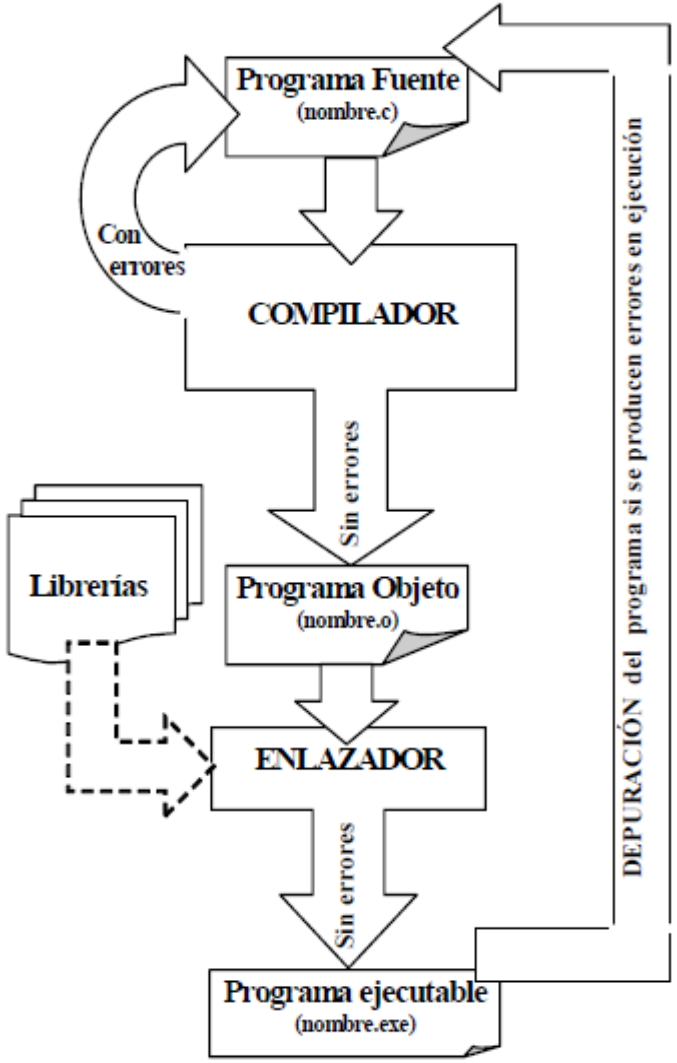
5 Una vez tengamos el archivo ejecutable, será el sistema operativo el encargado de colocar el programa en la memoria central y ejecutarlo.

6 Comprobar el funcionamiento del programa.

7 Si se detecta errores o un mal funcionamiento del programa, activar el debugger para analizar el programa y ejecutarlo sentencia a sentencia.

8 Una vez que hayamos encontrado la causa del error, volveremos al editor y lo corregimos.

9 El proceso de compilar, enlazar y ejecutar el programa lo repetiremos hasta que no se produzcan errores.



Programa fuente: Es una serie de enunciados o comandos escrito en un lenguaje de alto nivel que se usan para darle instrucciones a la computadora. El mismo es un texto sin formato y se lo guarda con extensión c. Necesita ser traducido a código máquina para poder ser ejecutado.

Compilador: Programa encargado de traducir los programas fuentes escritos en un lenguaje de alto nivel a lenguaje máquina y de comprobar que las llamadas a las funciones de librería se realizan correctamente. Genera un archivo objeto con extensión obj.

Programa (o código) objeto: Es el programa fuente traducido (por el compilador) a código máquina. Aún no es directamente ejecutable.

Librerías son una colección de códigos (funciones) ya programados y traducidos a código máquina (código objeto), listo para utilizar en un programa y que facilita la labor del programador.

Linker (montador o enlazador): Es el programa encargado de insertar al programa objeto el código máquina de las funciones de las librerías (archivos de biblioteca) usadas en el programa y realizar el proceso de montaje, que producirá un **programa ejecutable** .exe.

Programa Ejecutable: Traducción completa a código máquina, realizada por el enlazador, del programa fuente
y que ya es directamente ejecutable.

El CodeBlocks es un entorno completo de desarrollo con tiene el editor de texto, el compilador y el enlazador. Se lo puede descargar desde: http://www.codeblocks.org/downloads/26 y el archivo a seleccionar es: codeblocks-17.12mingw-setup.exe.

Primer programa en C

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

int main(void)
{
   printf("Hola!\n");
   return 0;
   getch();
}
```

#include <..> es una directiva del preprocesador, indica que debe incluir, antes de la compilación el contenido del archivo indicado. En este caso, el archivo stdio.h. Este archivo es un archivo de cabecera de la librería stdio el cual contiene prototipos, macros y constantes usados por la librería. Esta librería esta incluida en cualquier compilador que usemos y nos permite llamar a la función printf.

printf es la función que se encarga de mostrar un texto en pantalla. Es la responsable de que hayamos necesitado escribir ese "include" al principio del programa.

int main (void) main debe existir siempre e indica el punto en el que realmente comenzará a funcionar el programa. Después de "main" van dos llaves { y }, que delimitan el bloque más importante: el cuerpo del programa, int delante indica que la función main va a devolver un número entero y (void) indica que a la función no le entrego ningún dato.

getch() es una función que espera que se presione una tecla, se encuentra dentro de la librería conio.h, para poder agregarla se deberá incluir el archivo de cabecera correspondiente. Se puede agregar a un programa para evitar que salga del mismo hasta que se presione una tecla cualquiera.

Cada enunciado termina con ;

return 0 es el valor que devuelve la función

```c
/*Programa para calcular el producto de dos números*/
#include<stdio.h>
#include<conio.h>
int producto (int x, int y);/*esto es el prototipo de la función*/
int main(void)
{
   int a,b,c;
   /*pide el primer numero*/
   printf("Entre un numero entre 1 y 100 : ");
   scanf("%d",&a);
   /*pide el segundo numero*/
   printf("\nEntre otro numero entre 1 y 100 : ");
   scanf("%d",&b);
   /*calcula el producto y lo muestra*/
   c=producto(a,b); /*Aca se usa la función*/
   printf("\n%d veces %d es = %d \n",a,b,c);
   getch();
   return 0;
}
/*Esto es la definición de la función*/
/*Funcion que cuando la llamo regresa el producto*/
int producto(int x,int y)
{
   return (x * y);
}
```

- Los programas trabajan con distintos tipos de datos que pueden ser números o caracteres.
- Y los mismos a su vez pueden ser variables o constantes a lo largo del programa.

Variables

Una variable es un nombre para identificar una (o varias) posiciones de memoria donde el programa guarda los distintos valores de una misma entidad.

- El nombre de las variables pude incluir letras, números y el carácter _.
- El primer carácter debe ser letra.
- Las mayúsculas y las minúsculas son distintas letras.
- No se pueden usar palabras clave como nombres de variables.
- No se pueden usar mas de 31 caracteres.
- Se aconseja usar letras minúsculas.
- Se aconseja usar nombres que tengan que ver con la función en el programa.

Palabras clave:

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

Tipo de Variable Punto Flotante

TIPO	BYTES	VALOR MINIMO	VALOR MAXIMO
float	4	3.4E-38	3.4E+38
double	8	1.7E-308	1.7E+308
long double	10	3.4E-4932	3.4E+4932

Las variables de punto flotante son SIEMPRE con signo, y en el caso que el exponente sea positivo puede obviarse el signo del mismo.

Tipos de Variables Enteras

TIPO	BYTES	VALOR MINIMO	VALOR MAXIMO
signed char	1	-128	127
unsigned char	1	0	255
signed short	2	-32.768	+32.767
unsigned short	2	0	+65.535
signed int	2	-32.768	+32.767
unsigned int	2	0	+65.535
signed long	4	-2.147.483.648	+2.147.483.647
unsigned long	4	0	+4.294.967.295

Si se omite el calificador delante del tipo de la variable entera, éste se adopta por omisión (default) como "signed".

La norma ANSI C no establece taxativamente la cantidad de bytes que ocupa cada tipo de variable, sino tan sólo que un "long" no ocupe menos memoria que un "int" y este no ocupe menos que un "short", los alcances de los mismos pueden variar de compilador en compilador

Variables tipo carácter

El lenguaje C guarda los caracteres como números de 8 bits de acuerdo a la norma ASCII extendido, que asigna a cada carácter un número comprendido entre 0 y 255 (un byte de 8 bits) Es común entonces que las variables que vayan a alojar caracteres sean definidas como: char c;.

Sin embargo, también funciona de manera correcta definirla como int c;.

Las variables del tipo carácter también pueden ser inicializadas en su definición, por ejemplo es válido escribir: char c=97; para que c contenga el valor ASCII de la letra "a", esto nos obliga a recordar dichos códigos. Existe una manera más directa de asignar un carácter a una variable; la siguiente inicialización es idéntica a la anterior: char c = 'a';.

Si delimitamos un carácter con comilla simple, el compilador entenderá que debe suplantarlo por su correspondiente código numérico. Lamentablemente existen una serie de caracteres que no son imprimibles, en otras palabras que cuando editemos nuestro programa fuente (archivo de texto) nos resultará difícil de asignarlas a una variable ya que el editor las toma como un COMANDO y no como un carácter. Un caso típico sería el de "nueva línea" ó ENTER.

Con el fin de tener acceso a los mismos es que aparecen ciertas secuencias de escape convencionales. Su uso es idéntico al de los caracteres normales así para resolver el caso de una asignación de "nueva línea" se escribirá: char c = '\n';.

El código ASCII

sigla en inglés de American Standard Code for Information Interchange
(Código Estadounidense Estándar para el Intercambio de Información)

Caracteres de control ASCII

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

Caracteres ASCII imprimibles

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

ASCII extendido

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

Héctor Albano Diego Codevilla

```
/* Programa para obtener el tamaño de */
/* variables del C en bytes */
#include <stdio.h>
#include <conio.h>
int main( )
{
        printf( "\nchar \t\t%d bytes", sizeof( char ));
        printf( "\nint \t\t%d bytes", sizeof( int ));
        printf( "\nshort \t\t%d bytes", sizeof( short ));
        printf( "\nlong \t\t%d bytes", sizeof ( long ));
        printf( "\nunsigned char \t%d bytes", sizeof( unsigned char ));
        printf( "\nunsigned int \t%d bytes" , sizeof( unsigned int )) ;
        printf( "\nunsigned short \t%d bytes" , sizeof( unsigned short ));
        printf( "\nunsigned long \t%d bytes", sizeof( unsigned long));
        printf( "\nfloat \t\t%d bytes", sizeof ( float ));
        printf( "\nlong double \t%d bytes", sizeof ( long double ));
        printf( "\ndouble \t\t%d bytes \n", sizeof( double ));
        getch();
        return 0;
}
```

Declaración de las variables

Antes de usar una variable se la debe declarar informándole al compilador el nombre, el tipo de variable y ocasionalmente un valor específico, para que el compilador destine cierta cantidad de memoria para guardarlas.

nombre_de_tipo nombre_de_variable**=valor_de_variable**;
nom_de_tipo nom_de_var1**=val_de_var1**, nom_de_var2=**val_de_var2**;

```
int contador;
unsigned char i=0;
double porcentaje, iva_importado=10.1;
```

typedef permite usar un nombre alternativo a un tipo de dato existente, esto hace mas entendible al programa.

```
typedef int entero;
entero contador=0;
```

Conversión automática de tipos

Cuando dos o mas tipos de variables distintas se encuentran en una misma operación o expresión matemática , ocurre una conversión automática del tipo de las variables.
Reglas de conversión (previamente a la realización de dicha operación):

- 1) Las variables del tipo char o short se convierten en int
- 2) Las variables del tipo float se convierten en double
- 3) Si alguno de los operandos es de mayor precisión que los demás , estos se convierten al tipo de aquel y el resultado es del mismo tipo.
- 4) Si no se aplica la regla anterior y un operando es del tipo unsigned el otro se convierte en unsigned y el resultado es de este tipo.

En las asignaciones, por ejemplo valor1=valor2;
Posteriormente al cálculo del resultado de **valor2** (según las reglas antes descriptas), el tipo de este se iguala al del **valor1**. El resultado no se verá afectado si el tipo de **valor1** es igual o superior al del **valor2**, en caso contrario se efectuará un truncamiento o redondeo, según sea el caso. Por ejemplo, el pasaje de float a int provoca el truncamiento de la parte fraccionaria, en cambio de double a float se hace por redondeo.

Conversión explicita de tipos (casting)

Las conversiones automáticas pueden ser controladas a gusto por el programador, imponiendo el tipo de variable al resultado de una operación.
(<nombre de tipo>)<expresión>

```
double d , e , f = 2.33 ; int i = 6 ;
e = f * i ;
d = (int) ( f * i ) ;
```

e es 13.98 y d es 13.00.
Si la operación es d = (int) f * i; el resultado será el mismo?

Constantes

Es una posición de almacenamiento de datos cuyo valor no cambia una vez compilado el programa.
Es conveniente usar letras mayúsculas al definirla para diferenciarla de las variables.
Ej unsigned int BASE 10;
const es un modificador del tipo de variable que le indica al compilador que dicho valor es una constante, si a lo largo del programa se lo quiere modificar, el compilador indica un error.
Ej const unsigned int BASE = 10;
También se pueden declarar constantes usando contenidos simbólicos. El compilador, en el momento de crear el ejecutable, reemplazará el símbolo por el valor asignado.
Se usa la directiva al compilador **#define** su sintaxis es la siguiente: *#define nombre_constante valor*
Ej #define BASE 10
A fin de tener control sobre el tipo de las constantes, se aplican la siguientes reglas :
• Una variable expresada como entera (sin parte decimal) es tomada como tal salvo que se la siga de las letras F o L (mayúsculas o minúsculas)
• Una variable con parte decimal es tomada siempre como DOUBLE, salvo que se la siga de la letra F ó L
• Si en cualquiera de los casos anteriores agregamos la letra U o u la constante queda calificada como UNSIGNED (consiguiendo mayor alcance
• Una variable numérica que comienza con "0" es tomado como OCTAL
• Una variable numérica que comienza con "0x" o "0X" es tomada como hexadecimal

Enunciados, expresiones y operadores

Un **enunciado** es una indicación completa que da instrucciones a la computadora para ejecutar alguna tarea. Por lo general ocupan una sola línea y terminan con un punto y coma. Ej x=2+3;
Un **enunciado compuesto** es un conjunto de dos o mas enunciados encerrados entre llaves.
Una **expresión** es cualquier cosa que evalúa un valor numérico.
Un **operador** es un símbolo que da instrucciones al C para que ejecute alguna operación o acción en uno o mas operandos.

Operador de asignación:
Es el signo =. Ej x=y; esto indica que el valor de y es asignado a x. Del lado derecho puede haber un expresión y del lado izquierdo solo puede haber una variable.

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVAL
=	igual a	a = b	13
op=	pseudocodigo	a += b	13
=?:	asig.condicional	a = (c>b)?d:e	12

Una operación aritmética o de bit cualquiera (simbolizada por OP)
a = (a) OP (b) ;
puede escribirse en forma abreviada como :
a OP= b ;
Por ejemplo
a += b ; /* equivale : a = a + b */
a -= b ; /* equivale : a = a - b */
a *= b ; /* equivale : a = a * b */
a /= b ; /* equivale : a = a / b */
a %= b ; /* equivale : a = a % b */

ASIGNACION CONDICIONAL:
var1 = (operación relacional o lógica) ? (var2) : (var3) ;
de acuerdo al resultado de la operación condicional se asignará a var1 el valor de var2 o var3. Si aquella es CIERTA será var1= var2 y si diera FALSO , var1 = var3 .
Por ejemplo, si quisiéramos asignar a c el menor de los valores a o b , bastará con escribir :
c = (a < b) ? a : b ;

OPERADORES ARITMETICOS

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVAL.
+	SUMA	a + b	3
-	RESTA	a - b	3
*	MULTIPLICACION	a * b	2
/	DIVISION	a / b	2
%	RESTO DIVISION	a % b	2
-	SIGNO	-a	2

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVAL
++	incremento	++i o i++	1
--	decremento	--i o i--	1

Pueden tener un solo operando o dos, tienen un solo operando los operadores de incremento decremento y signo. Los restantes tienen dos operandos.

x++; es equivalente a x=x+1;
x–; es equivalente a x=x-1;
Solo se puede aplicar a variables.

x++ modo postfijo modifica el operando después de usarlo
Ej x=10; y=x++; /*y=10 x=11*/

++x modo prefijo modifica el operando antes de usarlo
Ej x=10; y=++x; /*y=11 x=11*/

-x; le cambia el signo a x.

x%y; se usa para calcular el resto del cociente entre dos números enteros, no puede ser usado en variables del tipo flotante.
Ej x=11; y=x%3; /* y=2 */

OPERADORES DE MANEJO DE BITS

Estos operadores muestran una de las armas más potentes del lenguaje C , permiten modificar bit a bit las variables . Debemos anticipar que estos operadores sólo se aplican a variables del tipo char, short, int y long y no pueden ser usados con float o double.

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVAL.
&	AND	a & b	7
|	OR	a | b	9
^	XOR	a ^ b	8
<<	DESPLAZAMIENTO A LA IZQUIERDA	a << b	4
>>	DESPLAZAMIENTO A LA DERECHA	a >> b	4
~	NOT (COMPLEMENTO A UNO)	~a	1

OPERADORES RELACIONALES

Todas las operaciones relacionales dan sólo dos posibles resultados : verdadero o falso . En el lenguaje C, falso queda representado por un valor entero nulo (cero) y verdadero por cualquier número distinto de cero

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVAL
<	menor que	(a < b)	5
>	mayor que	(a >b)	5
< =	menor o igual que	(a <= b)	5
>=	mayor o igual que	(a >= b)	5
= =	igual que	(a == b)	6
! =	distinto que	(a != b)	6

OPERADORES LOGICOS

Estos operadores permiten realizar conectividad lógica entre operadores relacionales, los resultados obtenidos siembre serán verdadero o falso

SIMBOLO	DESCRIPCION	EJEMPLO	ORDEN DE EVAL
&&	Y (AND)	(a>b) && (c < d)	10
||	O (OR)	(a>b) || (c < d)	11
!	NEGACION (NOT)	!(a>b)	1

Precedencia de operaciones:
En primer lugar se realizarán las operaciones indicadas entre paréntesis. Si hay varios niveles de paréntesis se ejecutan de adentro hacia afuera.
Luego las restantes según el orden indicado en cada una de las tablas. Si tuvieran igual orden se analizan de izquierda a derecha.
Para evitar errores en los cálculos es conveniente usar paréntesis, sin limitación de anidamiento, los que fuerzan a realizar primero las operaciones incluidas en ellos . Los paréntesis no disminuyen la velocidad a la que se ejecuta el programa sino que tan sólo obligan al compilador a realizar las operaciones en un orden dado, por lo que es una buena costumbre utilizarlos.

Comentarios:
Un comentario puede empezar en una línea y terminar en otra distinta, así:

```
/* Este es un comentario
que ocupa más de una línea */
```

O también:

```
// Este es un comentario hasta fin de línea.
```