

Capítulo 2.
Conceptos básicos
del lenguaje C

Índice del capítulo

1. Introducción	5
2. Objetivos	6
3. Tipos de datos del lenguaje C	7
3.1. Identificadores	7
3.2. Tipos de datos	8
3.2.1. Números enteros	8
3.2.2. Números reales	9
3.2.3. Caracteres	10
3.2.4. Constantes	10
4. Comentarios	12
5. Funciones de entrada/salida	13
5.1. La función printf ()	13
5.1.1. Especificadores de formato	14
5.1.2. Modificadores de especificadores de formato	15
5.2. La función scanf ()	16
5.2.1. El operador &, operador indirección	18
6. Algunas funciones de entrada y salida de datos	20
6.1. getchar ()	20
6.2. getch ()	20
6.3. getche ()	20
6.4. putchar ()	20
6.5. putch ()	20
7. Expresiones y operadores	21
7.1. Expresiones	21
7.2. Operador de asignación, =	21
7.3. Operadores aritméticos	22
7.3.1. Operadores suma, +	22
7.3.2. Operador resta, -	22
7.3.3. Operador signo, -	23
7.3.4. Operador multiplicación, *	23
7.3.5. Operador división, /	23
8. Conversiones en las operaciones aritméticas y en las expresiones	24
9. Operaciones de relación	25

10. Operadores lógicos	27
11. Otros operadores	28
11.1. Operador modulo, %	28
11.2. Los operadores Incremento ++ y Decremento —	29
11.3. Operadores combinados	30
12. Prioridad de los operadores	31
13. Bibliografía	33
14. Ejercicios propuestos	34

1. Introducción

El lenguaje C fue diseñado en los laboratorios de AT&T por D. Ritchie mientras desarrollaba el sistema operativo UNIX junto a Ken Thompson. El campo de acción del lenguaje C hoy por hoy es ilimitado y se puede encontrar en cualquier área de la informática o de la ingeniería. Cada vez son más y más las aplicaciones desarrolladas por medio del lenguaje C.

Seguramente la fama del lenguaje C le viene dada, entre otras razones, por estas características:

- Es un programa muy potente a pesar de su sencillez, ya que con la ayuda de sus grupos de sentencias y sus bibliotecas se pueden crear fácilmente programas de todo tipo.
- Una vez compilados los programas escritos en C resultan muy eficientes, rápidos y ocupan muy poca memoria.
- El lenguaje C es muy portátil. Por lo tanto los programas escritos en lenguaje fuente en una máquina pueden ser llevados a otras y pueden compilarse y ejecutarse sin cambios.
- Aunque se trata de un lenguaje de alto nivel, pueden utilizarse conceptos muy cercanos a los recursos físicos (registros generales, bits, dispositivos de salida/entrada), siendo muy interesante de cara a determinadas aplicaciones.

Por otro lado, los programadores deben comportarse muy exacta y disciplinadamente, de modo que todas las personas del grupo de trabajo puedan entender los programas.

2. Objetivos

Los objetivos que perseguimos en este capítulo son los siguientes:

- Aprender cómo el lenguaje C utiliza distintos tipos de datos.
- Utilizar las funciones básicas de presentación de datos en pantalla y recogida de datos desde el teclado.
- Realizar nuestro primer programa en C.
- Aprender a escribir todo tipo de expresiones matemáticas y/o lógicas mediante la utilización de operadores adecuados.

3. Tipos de datos del lenguaje C

Los datos alfanuméricos y las variables son los datos básicos que se manipulan en cualquier programa. Un ordenador, mediante las instrucciones de un programa, puede efectuar muchas tareas diferentes. Puede sumar unos números u ordenar nombres, calcular la órbita de un cometa, escribir cartas, ofrecer la interpretación de una imagen digital, etc. Para llevar a cabo estas tareas nuestro programa necesita los datos de la información que vamos a manejar. Por ello deberemos, al escribir en C, definir esos datos. Tratándose de variables y constantes deberemos aclarar de qué tipos de datos se tratan para que el compilador les asigne un emplazamiento en la memoria y una cadena de bits.

Los datos numéricos en ocasiones son números enteros. En otros casos, en cambio, se trata de números reales. Para expresar si son números reales o enteros se utilizan diferentes expresiones. Para manejar enteros se utilizan *formatos de coma fija* mientras que para los reales se utilizan *formatos de coma flotante*.

Los números enteros se expresan mediante código binario. A veces se utiliza un bit para expresar el signo (1 positivo, 0 negativo).

El formato para expresar números reales en la memoria, generalmente, es el formato binario de coma flotante. El número real se expresa mediante dos números en la memoria: **la mantisa** y **el exponente**, las cuales se representan mediante los correspondientes códigos binarios.

$$14,5585 = 0,145585 * 10^2 \implies (0,145575, 2)$$

El lenguaje C posee un sistema para identificar todos los datos que necesite el ordenador. De modo que podemos definir cualquier tipo de dato siguiendo ese sistema. Si el dato es constante el compilador de C nos dirá de qué tipo de dato se trata y si son variables deberán estar expresadas mediante una **palabra clave** en una sentencia de expresión. En la siguiente sección veremos cómo se lleva a cabo todo esto.

3.1. Identificadores

Como ya hemos mencionado, utilizaremos diferentes clases de datos, pero para procesar esos datos deberemos ponerles un nombre. Para ello utilizaremos los identificadores. De modo que *los nombres que se adjudican a las variables y a las funciones* son **sus identificadores**. Para definir los identificadores se utilizan **caracteres alfabéticos** (de la A la Z, mayúsculas y minúsculas), **números** (0-9), y **carácter especial** `_`. Por ejemplo:

Variable

Nombre_de_mi_funcion_9

Hay que tener en cuenta que los nombres de variables que vienen a continuación son diferentes:

monte, Monte, MONTE, monTE

La longitud máxima de un identificador puede ser cualquiera pero el compilador tan solo identifica los primeros 31 caracteres. El primer carácter obligatoriamente debe ser alfabético o el carácter “_” (guión bajo).

3.2. Tipos de datos

El lenguaje C reconoce los datos básicos siguientes:

- Números enteros
- Números reales
- Números reales de doble precisión.
- Caracteres

3.2.1. Números enteros

Como ya sabemos los números enteros no tienen decimales. Pueden definirse números enteros de diferentes características mediante diferentes cantidades de bits y diferentes usos del signo. De este modo identificamos:

- Números sin signo.
- Números cortos.
- Números largos.

Para guardar números sin signo se utilizan dos bytes de la memoria. La palabra clave es **unsigned int** y su valor está comprendido entre 0 y 65535.

Para guardar los números cortos con signo también se utilizan dos bytes de la memoria. La palabra clave es **short int** y están entre -32.768 y 32767. La palabra clave **short** no es obligatoria y si no se utiliza se supone que es un número muy corto con signo. Utilizando la palabra clave **unsigned** pueden definirse números sin signo.

Para guardar los números largos con signo se utilizan cuatro bytes de la memoria. La palabra clave es **long int** y están entre -2.147.438.648 y 2.147.483.647. También pueden ser sin signo y para ello debe entonces utilizarse la palabra clave **unsigned**.

3.2.1.1. Declaración de números enteros

La declaración de números enteros se efectúa así:

palabra clave	nombre	Valor inicial <i>(no es obligatorio)</i>
int	mi_variable	= -1;
long	otra_variable	= 68000;
unsigned int		= 250;
unsigned long	otras_cosas	= 60000;

Como hemos podido ver, para declarar un número entero puede haber tres secciones, *la palabra clave, el identificador del número o el nombre* y, aunque no es absolutamente necesario *el valor inicial*. La expresión siempre se cierra utilizando el signo; (punto y coma).

3.2.2. Números reales

Estos números son necesarios ante todo en los cálculos científicos. Utilizan el formato de coma flotante. La diferencia entre los de tipo real estriba en la precisión.

Los del tipo **float** tienen 7 cifras decimales, necesitan 32 bits (4 bytes), de ellos 8 se utilizan para expresar el exponente y 24 para la mantisa. Los del tipo **double** tienen 15 cifras decimales, necesitan 64 bits (8 bytes), también podemos definir los del tipo **long double** y en este caso se necesitan 80 bits (10 bytes).

Tipo de dato	palabra clave	número de bytes	límites
<i>Caracteres sin signo</i>	unsigned char	1	0/255
<i>Números enteros sin signo</i>	unsigned int	2	0/65.535
<i>Números enteros sin signo</i>	unsigned long	4	0/2 ³²
<i>Caracteres</i>	char	1	-128/127
<i>Números enteros</i>	short int	2	-32.768/32.767
<i>Números enteros</i>	long int	4	-2.147.483.648 / 2.147.483.647
<i>Números reales</i>	float	4	10 ³⁸ /10 ⁻³⁸
<i>Números reales</i>	double	8	1.7 10 ⁻³⁰⁸ /1.7 10 ³⁰⁸
<i>Números reales</i>	long double	10	3.4 10 ⁻⁴⁹³² /1.1 10 ⁴⁹³²

Tabla 1: Varios tipos de datos del lenguaje C.

3.2.2.1. Declaración de números reales

Al hacer la declaración de números reales también se diferencian tres áreas: la palabra clave, el identificador o el nombre y el valor inicial. He aquí varios ejemplos:

palabra clave	nombre	Valor inicial <i>(no es obligatorio)</i>
float	juan_var	= 6.63e-23;
double	otra_var	= 68.3456;

3.2.3. Caracteres

Este tipo de dato define un número entero sin signo entre 0 y 255. Normalmente este número entero se guarda en un byte. Generalmente el ordenador, utilizando el código ASCII, convierte los números en caracteres y a la inversa.

3.2.3.1. Declaración de caracteres

Las tres secciones descritas anteriormente también se utilizan para hacer la declaración de caracteres. A continuación pueden verse unos ejemplos:

palabra clave	nombre <i>(no es obligatorio)</i>	Valor inicial
char	mi_caracter	= 'S';
char	semana;	

3.2.4. Constantes

El lenguaje C reconoce las siguientes constantes:

- Constantes enteras
- Constantes reales
- Caracteres constantes

3.2.4.1. Constantes enteras

Un número entero escrito en lenguaje C es una constante entera, es decir, sin punto decimal y sin exponente. Por ejemplo:

40	40 int
45L	45 long
020	20 utilizando código octal
0x20	20 utilizando código hexadecimal

3.2.4.2. Constantes reales

Un número real escrito en lenguaje C es una constante real. Generalmente las constantes reales las escribimos así:

12.5	12.5 float
25.	25 float
12.5e0	12.5 double
12.5e5	12.5 x 10 ⁵ double
12.5e-5	12.5 x 10 ⁻⁵ double

3.2.4.3. Caracteres constantes

Los caracteres en lenguaje C se escriben entre apóstrofes.

'a'	El carácter a
-----	---------------

Hay otras constantes para expresar los caracteres especiales (utilizadas sobre todo para escribir los resultados). Estos caracteres especiales se escriben mediante secuencias de escape:

\n	carácter de final de línea
\t	tabulador
\b	atrás (backspace)
\r	return
\f	salto de página
\\	línea atrás
\'	apóstrofe
\"	comillas

3.2.4.4. El modo más indicado para definir constantes: #define

Mediante esta sentencia del preprocesador podemos asignar un nombre a cada constante. Hay muchas razones para hacer esto así. Los programas son más legibles y además si hay que cambiar por cualquier razón el valor de la constante tan solo deberemos hacerlo en el sitio donde esté definida.

```
#define PI          3.1415
#define IVA        0.16
```

Los pasos necesarios para definir constantes son los siguientes:

- Escribir en un fichero todas las sentencias definidas mediante la sentencia **#define** y nombrarlo adecuadamente. Por ejemplo constant.h
- Escribir en el comienzo de cada programa #include "constant.h".

De este modo el preprocesador utilizara las sentencias reunidas en el fichero constant.h.

4. Comentarios

Los comentarios se utilizan para poder leer más fácilmente los programas. Por lo tanto utilizamos los comentarios para explicar el porqué de las instrucciones y sus significados. Al compilar el programa fuente, el compilador salta todo aquel comentario que detecte y no lo tiene en cuenta. En lenguaje C los comentarios se escriben así:

```
/* Esto es una explicación y no una instrucción */
```

5. Funciones de entrada/salida

Estas funciones se utilizan ante todo para introducir datos mediante el teclado y para visualizarlos en la pantalla. También se utilizan para lograr la comunicación entre otros dispositivos; por ejemplo para utilizar la impresora. En esta sección examinaremos las funciones `printf()` y `scanf()`. La primera la utilizamos para visualizar datos en la pantalla y la segunda para introducir datos mediante el teclado.

5.1. La función `printf()`

La función `printf()` se utiliza para hacer visibles los datos en la pantalla. El formato de esta función es el siguiente:

```
printf ("Especificador 1, Especificador 2,...", variable_1, variable_2,...);
```

El especificador es una cadena de caracteres que definen cómo han de visualizarse los datos. Son unos caracteres especiales que deben escribirse siempre entre comillas, y se les denomina especificaciones de formato. Por ejemplo:

```
int besos = 7;  
printf ("dame %d besitos", besos);
```

Si ejecutásemos este programa leeríamos lo siguiente en la pantalla:

```
Dame 7 besitos.
```

Tal y como puede verse en el ejemplo, en una cadena de caracteres aparecen los caracteres `%d`. A estos caracteres especiales se les denomina **especificadores de formato** definen el tipo de datos que vamos a visualizar. En este caso en el lugar en que aparecen los caracteres `%d` entra el valor de la variable `besos` y es su contenido lo que se visualiza. En esta cadena de control se pueden introducir los especificadores de formato que queramos pero cada uno de ellos deberá tener bien definido su valor. Por ejemplo,

```
int pan = 7;  
float mas = 1.5;  
printf ("Dame %d panes o por lo menos %f ", pan, mas);
```

Si ejecutásemos este programa leeríamos lo siguiente en la pantalla:

```
Dame 7 panes o por lo menos 1.5
```

5.1.1. Especificadores de formato

Cuando queremos visualizar un dato, las instrucciones que debemos dar a la función `printf ()` varían según el tipo de dato de la variable. De modo que para visualizar números enteros debemos utilizar la instrucción `%d`, y la función `%c` para visualizar caracteres. A estas instrucciones se les da el nombre de **especificadores de formato** o **especificadores de tipos de datos**.

He aquí los especificadores de datos que se utilizan:

Especificador	Significado
<code>%d</code>	Número Entero
<code>%c</code>	Carácter
<code>%s</code>	Cadena de caracteres
<code>%e</code>	Número real de coma flotante Notación exponencial
<code>%f</code>	Número real de coma flotante Notación decimal
<code>%g</code>	<code>%f</code> o <code>%e</code> , el que sea más corto
<code>%u</code>	Numero entero sin signo
<code>%o</code>	Número octal sin signo
<code>%x</code>	Número hexadecimal sin signo

1. Ejemplo

En este ejemplo queremos visualizar en la pantalla unos mensajes utilizando la función `printf ()`.

```
#include <stdio.h>

/* Utilizando la directiva #define definimos la constante PI */
#define PI 3.1415

main ()
{
    /* declaración de variables */

    /* En esta sección se definen dos variables y se les da el valor inicial */
    int mujer = 5;
    float vino = 13.5;

    /* Aquí comienza el programa */
```

```

/*
Utilizando los tres printf () siguientes se visualizan tres mensajes en la pantalla.
Tened en cuenta que en lugar donde están %d y %f entran los valores de las variables mujer y vino.
*/
printf ("esas %d mujeres bebieron %f botellas de vino \n",mujer, vino);

printf ("El valor de la constante PI es: %f \n", PI);

printf ("Eso es todo \n");
}

```

Después de ejecutar este programa leeríamos lo siguiente en la pantalla:

```

Esas 5 mujeres bebieron 13.5 botellas de vino
El valor de la constante PI es: 3.1415
Eso es todo

```

5.1.2. Modificadores de especificadores de formato

Los modificadores, unidos a los especificadores, cambian el formato de la salida. Se sitúan entre el signo % y los caracteres que definen el tipo de dato. He aquí unos modificadores de uso habitual:

- – Lo que vayamos a escribir se escribirá en el espacio asignado, comenzando a escribirlo a partir de la izquierda.
- **número** : La mínima anchura del espacio asignado para escribir.
- **.número o**: Precisión. Número de decimales que debemos escribir a la derecha del punto en el caso de los números de coma flotante. En las cadenas de caracteres es el número de caracteres que debemos escribir.

En este ejemplo hay dos espacios para escribir números y, si en la variable *edad* está guardado el valor 33, el mensaje que veríamos en pantalla aparecería así:

```
printf ("Mi edad es %2d.", edad);
```

En este segundo ejemplo hay cuatro espacios para visualizar los números enteros y por lo tanto el mensaje aparecería de este modo:

```
printf ("Mi edad es %4d.", edad);
```



```
#include <stdio.h>

main ()
{
    /* Declaración de variables */

    char nombre [20];

    /* Aquí comienza el programa */

    /* Visualiza en pantalla el mensaje entre comillas */
    printf (" por favor escriba su nombre: \n\n");

    /* Pide que el usuario introduzca datos mediante el teclado y los almacena en la variable nombre */
    scanf ("%s", nombre);

    /*
    Visualiza el mensaje entre comillas, en el lugar de %s pon los datos que se guardan en la variable
    nombre.
    */
    printf ("%s está estudiando lenguaje C en la Escuela de Ingenieros de Bilbao", nombre);
}
```

Al ejecutar este programa veríamos esto en pantalla:

```
Por favor escriba su nombre:
Fermin
Fermin está estudiando el lenguaje C en la
Escuela de Ingenieros de Bilbao
```

3. Ejemplo

Escribir un programa que tomando la edad de una persona calcule su equivalente en días.

```
#include <stdio.h>
#define DIAS_ANUALES 365
main ()
{
    /* Declaración de variables */

    int aniversarios, dias;
```

```
/* El programa comienza aquí */

/* Pedir la edad de la persona */

printf ("Por favor introduzca su edad en años: \n\n");
scanf ("%d", &aniversarios);

/* conversión */
dias = aniversarios * DIAS_ANUALES;

/* Visualizar en días la edad de la persona */
printf (" Su edad en días es: %d\n", dias);
}
```

Después de ejecutar este programa veríamos el siguiente resultado en la pantalla:

```
Por favor introduzca su edad en años
10
Su edad en días es: 3650
```

5.2.1. El operador &, operador indirección

¿Por qué tenemos que utilizar este operador? ¿Porqué no podemos utilizar el nombre de la variable? En lenguaje C todas las variables tienen una dirección y un contenido. El contenido es el valor que en ese momento toma la variable y la dirección es la posición de la memoria donde se encuentra ese contenido. Cuando queremos utilizar la variable mediante este operador se logra la dirección donde está guardada. La función `scanf ()` lo pide así, no hay que utilizar la variable en sí sino su dirección. Cuando este operador se sitúa delante de una variable nos estamos refiriendo a la dirección de esa variable

Este operador le otorga una gran potencia al lenguaje C y de aquí en adelante deberemos utilizarlo cada vez más.

4. Ejemplo

En este ejemplo se visualizan en pantalla el contenido y la dirección de una variable utilizando la función `printf ()`.

```
#include <stdio.h>  
main ()  
{  
    /* Declaración de variables */  
    int numero = 2;  
    /*  
        Mediante esta instrucción se visualizan el contenido y la dirección de la variable numero  
    */  
  
    printf (" Valor = %d, Dirección = %d \n\n", numero, &numero);  
}
```

6. Algunas funciones de entrada y salida de datos

El lenguaje C ofrece mediante su biblioteca diversas funciones de entrada y salida de datos, algunas de las cuales se presentan en este apartado. Vamos a analizar cinco de ellas, de las cuales tres son funciones de entrada (*getchar ()*, *getche ()* y *getch ()*) y las otras dos de salida (*putchar ()* y *putch ()*). La utilización de estas funciones se explica en los ejemplos de la siguiente sección.

6.1. *getchar ()*

Esta función toma un dato del teclado y se lo da a un programa que está ejecutándose. Espera hasta que se pulsa la tecla return.

6.2. *getch ()*

Esta función toma un dato del teclado y se lo da a un programa que está ejecutándose. El carácter no se visualiza y no hay que pulsar la tecla return.

6.3. *getche ()*

Esta función toma un dato del teclado y se lo da a un programa que está ejecutándose. No hay que pulsar la tecla return y además visualiza el eco del carácter.

6.4. *putchar ()*

Toma un dato de un programa en ejecución y lo visualiza en pantalla.

6.5. *putch ()*

Toma un dato de un programa en ejecución y lo visualiza en pantalla pero no lo escribe desde el comienzo de la línea sino en la última posición.

7. Expresiones y operadores

7.1. Expresiones

Las expresiones se obtienen combinando adecuadamente mediante el uso de operadores las variables y las constantes examinadas hasta el momento. Las expresiones nos permiten realizar cualquier tipo de operación. Por lo tanto, combinando las variables y las constantes con los operadores se consiguen las expresiones.

Por ejemplo la expresión necesaria para hacer la conversión de grados de temperatura Fahrenheit a grados Celsius es:

```
celsius_temp = (faren_temp - 32) * 5 / 9;
```

Donde *celsius_temp* y *faren_temp* son variables definidas de antemano y =, -, * y / son operadores.

En las siguientes secciones se examinarán los operadores más habituales.

7.2. Operador de asignación, =

En lenguaje C el signo igual no significa que las cosas sean iguales. Es un operador para asignar valores. Por ejemplo,

```
bmw = 2002;
```

Esta sentencia no dice que *bmw* sea igual a *2002* sino que le hemos asignado a la variable *bmw* el valor *2002*. El que está a la izquierda del signo es el nombre de la variable y el que está a la derecha el valor que le damos a la variable. Este operador de asignación debe utilizarse frecuentemente mientras escribimos un programa y por lo tanto su utilización debe entenderse bien.

Otra sentencia de asignación interesante es la siguiente:

```
i = i + 1;
```

Esta expresión no tiene sentido desde un punto de vista matemático, pero debe entenderse así: toma el valor de la variable *i*, lo incrementa y guarda nuevamente su resultado en la variable *i*. La utilización de este tipo de expresiones en programas escritos para ordenador es absolutamente normal a pesar de su aparente inconsistencia desde el punto de vista de una expresión matemática.

7.3. Operadores aritméticos

En esta sección examinaremos los operadores que utilizamos para llevar a cabo operaciones matemáticas.

7.3.1. Operadores suma, +

El operador de adición suma los valores que tiene a izquierda y derecha. Por ejemplo mediante esta instrucción se visualiza el número 24 en pantalla.

```
printf ("%d", 4 + 20)
24
```

Los operandos pueden ser tanto variables como constantes. Si en el ejemplo anterior los operandos eran constantes, en el siguiente son variables:

```
int var_uno = 7;
int var_dos = 9;
int suma;
suma = var_uno + var_dos;
printf ("suma = var_uno + var_dos = %d + %d = %d", var_uno, var_dos, suma);
```

Si escribimos una sección de programa con estas características durante su ejecución podríamos ver el resultado siguiente en pantalla:

```
suma = var_uno + var_dos = 7 + 9 = 16
```

Como se puede ver mediante esta expresión se guardan en la variable *suma* el valor de *var_uno* más el valor de la variable *var_dos*.

7.3.2. Operador resta, -

Para realizar una sustracción el operador resta el valor de la derecha del de la izquierda. Por ejemplo,

```
resta = 224.0 - 24.0;
```

En la variable *resta* se guarda el valor 200.0.

También aquí los operandos pueden ser tanto constantes como variables.

7.3.3. Operador signo, -

El operador de signo cambia el signo del valor situado a su izquierda. Veamos varios ejemplos:

```
felipe = -13;
```

En este ejemplo se guarda el número menos 13 en la variable *felipe*.

```
felipe = -juan;
```

En este segundo ejemplo se introduce en la variable *felipe* el valor de la variable *juan* pero con el signo cambiado.

7.3.4. Operador multiplicación, *

Este operador multiplica el valor de la izquierda por el valor de la derecha.

Por ejemplo,

```
centimetros = 2.54 * pulgadas;
```

En este caso se guarda en la variable denominada *centimetros* 2.54 multiplicado por el valor de la variable *pulgadas*.

7.3.5. Operador división, /

Este operador divide el valor de la izquierda por el de la derecha. Por ejemplo,

```
cuatro = 12.0 / 3.0;
```

Después de realizar esta operación el valor de 4.0 se guarda en la variable denominada *cuatro*.

Las divisiones funcionan de diferente manera según los tipos de datos. Si los operandos son números enteros el resultado también es un número entero. Si los operandos son números reales el resultado también será un número real. Si el tipo de dato de los operandos es diferente el número entero se convierte automáticamente en real y la operación se hace entre números reales.

Al hacer divisiones entre números enteros no se tiene en cuenta el resto. Por lo tanto,

```
cuatro = 13 / 3;
```

En la variable *cuatro* se guarda el número entero 4 después de efectuarse la división.

8. Conversiones en las operaciones aritméticas y en las expresiones

En los programas escritos en lenguaje C, y por lo que se refiere al tipo de los operandos, éstos no tienen por qué ser compatibles. Pero hay que tener en cuenta que esto puede ser una fuente de errores y por lo tanto al hacer el diseño del programa es conveniente tener en cuenta los tipos de operandos que se manejan.

Si los operandos de las operaciones son de diferentes tipos se produce una conversión de datos implícita siguiendo esta clasificación:

char fi short fi int fi long fi float fi double

De todas formas y para evitar errores es conveniente que los operandos tengan el mismo tipo de datos.

Por ejemplo examinemos qué ocurre con la división:

```
#include <stdio.h>
main ()
{
    printf ("División con números enteros 5 / 4 = %d \n", 5 / 4);
    printf ("División con números enteros 6 / 3 = %d \n", 6 / 3);
    printf ("División con números enteros 7 / 4 = %d \n", 7 / 4);
    printf ("División con números reales 7. / 4. = %2.2f \n", 7. / 4.);
    printf ("División con números enteros y reales 7. / 4 = %2.2f \n", 7. / 4);
}
```

El resultado de este programa sería el siguiente:

```
División con números enteros 5 / 4 = 1
División con números enteros 6 / 3 = 2
División con números enteros 7 / 4 = 1
División con números reales 7. / 4. = 1.75
División con números enteros y reales 7. / 4 = 1.75
```

Como puede verse al hacer una división entre números enteros también se consigue un número entero. Si el tipo de dato de los números es diferente, la conversión se produce automáticamente.

9. Operadores de relación

Los operadores de relación se utilizan principalmente para expresar condiciones. He aquí los que vamos a aprender a utilizar:

OPERACIÓN	OPERADOR	FORMATO	EXPLICACIÓN
mayor	>	a > b	si a > b verdadero 1 si no falso 0
menor	<	a < b	si a < b verdadero 1 si no falso 0
igual	==	a == b	si a == b 1 si no 0
mayor o igual	>=	a >= b	si a >= b 1 si no 0
menor o igual	<=	a <= b	si a <= b 1 si no 0
diferente	!=	a != b	si a != b 1 si no 0
y	&&	a && b	si a && b 1 si no 0
o		a b	si a b 1 si no 0
no	!	! a	si !a 1 si no 0

5. Ejemplo

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

    int edad;

    edad = 15;
    printf ("¿eres más joven de 21 años ? %d", edad < 21);

    edad = 30;
    printf ("¿eres más joven de 21 años? %d", edad < 21);

}
```

Al ejecutar este programa nos dará lo siguiente:

¿eres más joven de 21 años? 1

¿eres más joven de 21 años? 0

Por lo tanto después de valorar si la condición es cierta se consigue el valor 1, si no 0. Cuando evaluamos en lenguaje C una expresión de condición, si el resultado que se logra es falso, el valor de la expresión es 0.

10. Operadores lógicos

Hemos comentado que entre las características del lenguaje C se encuentra la de que es capaz de manipular bits. Entre estos operadores destacan los operadores AND (&), OR (|) y XOR (^).

11. Otros operadores

El lenguaje C tiene 40 operadores más o menos y algunos se utilizan más frecuentemente que otros. En esta sección se examinan algunos de ellos.

11.1. Operador módulo, %

Tras hacer la división da como resultado el resto de los números enteros.

6. Ejemplo

```
#include <stdio.h>
#define SM 60
main ()
{
    /*convertir los segundos en minutos y segundos */

    int segundos, minutos, resto;

    printf ("Este programa, tomando los segundos, nos da cuantos minutos y segundo son \n");

    printf ("Introduzca los segundos a convertir = ");
    scanf ("%d", &segundos);

    minutos = segundos / SM;
    resto = segundos % SM;

    printf ("%d segundos son %d minutos y %d segundos. \n",
           segundos,
           minutos,
           resto);
}
```

11.2. Los operadores Incremento ++ y Decremento —

El operador de incremento ++ incrementa el valor de su operando (+ 1).

El operador de decremento — decreenta el valor de su operando (-1).

Estos dos operadores dan diferentes resultados según la posición. Si el operador se sitúa delante de la variable, al ejecutar la instrucción el valor de la variable se incrementa o decreenta antes de su uso. De colocarse detrás de la variable el valor de la variable es usado y luego se incrementa o decreenta. A continuación se da un ejemplo para aclarar el funcionamiento exacto de estos dos operadores.

7. Ejemplo

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

    edad = 15;
    printf ("Edad %d\n", edad);
    printf ("Edad %d\n", edad++);
    printf ("Edad %d\n", edad);
}
```

Después de ejecutar este programa esto es lo que se vería en pantalla:

```
edad = 15
edad = 15
edad = 16
```

8. Ejemplo,

```
#include <stdio.h>
main ()
{
    int edad;
    edad = 15;
    printf ("edad %d", edad);
    printf ("edad %d", ++edad);
    printf ("edad %d", edad);
}
```

Después de ejecutar este programa esto es lo que se vería en pantalla:

```
edad= 15
edad = 16
edad = 16
```

11.3. Operadores combinados

Al utilizar operadores combinados, el compilador de C logra un código más efectivo. El operador combinado se consigue escribiendo juntos la operación que deseamos realizar y el signo de asignación. Por ejemplo,

```
a = a + b
```

Mediante esta operación en la variable **a**, guardaremos el valor contenido en la variable **a** más el que hay en la variable **b**. Esta operación puede escribirse así mediante el operador combinado:

```
a += b
```

A continuación se da una lista de operadores combinados y las operaciones equivalentes que les corresponden.

Operador	Explicación	Uso
+=	a = a + b	a += b
*=	a = a * b	a *= b
-=	a = a - b	a -= b
/=	a = a / b	a /= b
%=	a = a % b	a %= b

12. Prioridad de los operadores

Supongamos que tenemos la siguiente expresión: ***minut = 25.0 + 120.0 * n / valor***

Esta expresión posee una suma, una multiplicación y una división. De modo que podemos preguntarnos: ¿cuál es la operación que se ejecutará en primer lugar?. Supongamos que la variable ***n*** tiene el valor de 6.0 y que la variable ***valor*** tiene el de 2.0.

Tal y como se aprecia claramente, ¡el orden tiene importancia! De cambiar el orden de la operación la variable ***minut*** logra valores totalmente diferentes.

Por lo tanto el lenguaje C debe tener el orden exactamente definido para realizar operaciones. Es por ello que a cada operador se le da su nivel de prioridad. Por ejemplo, los operadores de multiplicación y división tienen un nivel de prioridad más alto que el operador de sumas y por lo tanto se ejecutan antes. ¿Qué ocurre cuando hay dos operadores con el mismo nivel de prioridad? En este caso se ejecutan en el mismo orden en que figuran en la expresión (de izquierda a derecha). En la siguiente tabla se reflejan los niveles de prioridad de algunos operadores.

Nivel de prioridad	Operación	Orden de evaluación
1.	()	de izqu. a dcha.
2.	* / %	de izqu. a dcha.
3.	+ -	de izqu. a dcha.
4.	< <= > >=	de izqu. a dcha.
5.	= !=	de izqu. a dcha.
6.	&	de izqu. a dcha.
7.	L	de izqu. a dcha.
8.		de izqu. a dcha.
9.	&&	de izqu. a dcha.
10.		de izqu. a dcha.
11.	= *= /= += -= %=	de dcha. a izqu.

9. Ejemplo

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

```
main ()
```

```
{  
    int max, tanteo;  
  
    max = tanteo = - (2 +5) * 6 + (4 + 3 * (2 + 3));  
    printf ("max = %d \n", max);  
}
```

La expresión presente en este programa se evaluará mediante los siguientes pasos:

$$\text{max} = \text{tanteo} = -7 * 6 + (4 + 3 * (2 + 3))$$

$$\text{max} = \text{tanteo} = -7 * 6 + (4 + 3 * 5)$$

$$\text{max} = \text{tanteo} = -7 * 6 + (4 + 15)$$

$$\text{max} = \text{tanteo} = -7 * 6 + 19$$

$$\text{max} = \text{tanteo} = -42 + 19$$

$$\text{max} = \text{tanteo} = -23$$

La variable `tanteo` toma el valor de `-23` antes de que la tome la variable `max`.

13. Bibliografía

WAITE, M., PRATA, S., MARTIN, D. *Programación en C. Introducción y Conceptos Avanzados*. Ed. Anaya Multimedia.

14. Ejercicios propuestos

1. Encontrar los errores contenidos en este programa:

```
#include <stdio.h>

main
{
    flota g; h
    float valor, precio;
    g = e21;
    valor = precio * g
}
```

2. La escala de un termómetro ofrece la temperatura en grados Celsius. Crear un programa que los convierta en grados Fahrenheit.

$$F = 1.8 C + 32$$

F temperatura Fahrenheit
C temperatura Celsius

Después de calcular el resultado visualizarlo en este formato:

$$XXX.XX \text{ grados Celsius} = YYY.YY \text{ grados Fahrenheit}$$

3. Calcular mediante un programa la superficie de un círculo, de un triángulo y de un cuadrado. El programa pedirá los datos adecuados para cada figura.
4. Leer un número de tres cifras. Escribe en la pantalla un número nuevo de las mismas cifras pero, comparándolo con el anterior, con el orden de las cifras cambiado. Por ejemplo,

Entrada 1 2 3 fi Salida 3 2 1

5. La resolución de este sistema de ecuaciones,

$$A x + B y = C$$

$$D x + E y = F$$

se consigue mediante las siguientes formulas:

$$x = \frac{C E - B F}{A E - B D} \quad Y = \frac{A F - C D}{A E - B D}$$

Mediante un programa leer los coeficientes (A, B, C, D, E eta F) y escribir la resolución del sistema en la pantalla.

6. Leer un número entero de dos cifras del teclado y calcular la segunda, tercera y cuarta potencia del número. Visualizar el número con sus tres potencias guardando la siguiente forma:

$$Z \quad Z^{**2} \quad Z^{**3} \quad Z^{**4}$$

7. Dada la cantidad C de pesetas, calcular cuántos billetes y monedas de 1000 (de mil) 100 (de cien), de 50 (de cincuenta), de 25 (de veinticinco), de 5 (de cinco) y de 1 (de uno) hacen falta para totalizar C. La cantidad de billetes y monedas de diferentes tipos deberá ser la mayor posible.

8. Las masas M1 y M2 están a una distancia de dos R, calcular la fuerza atractora F que actúa sobre las dos partículas a causa de la gravitación. Dando las masas en kilogramos y la distancia en metros, la constante de la gravitación universal es $G = 6.67 \times 10^{-11}$ newtons por metro²/kg².

$$F = G \frac{M1 * M2}{R^2} \quad (\text{Nw})$$

9. Conseguir el equivalente (en grados, minutos y segundos) de un ángulo dado en radianes.

10. Leyendo una cara conseguir el volumen del cubo correspondiente y el volumen de la esfera más grande que pueda caber dentro de él.

