Capítulo 1

Qué es un algoritmo

En programación, un **algoritmo** establece, de manera genérica e informal, la secuencia de pasos o acciones que resuelve un determinado problema informático.

Los algoritmos constituyen la documentación principal que se necesita para poder iniciar la fase de codificación de un programa y, para representarlos, se utiliza, fundamentalmente, dos tipos de notación: pseudocódigo y diagramas de flujo (ordinogramas). El diseño de un algoritmo es independiente del lenguaje que después se vaya a utilizar para codificarlo.

1.1. Pseudocódigo

El **pseudocódigo** es un lenguaje de programación algorítmico; es un lenguaje intermedio entre el lenguaje natural y cualquier lenguaje de programación específico, como son: C, FORTRAN, Pascal, etc. No existe una notación formal o estándar de pseudocódigo, sino que, cada programador puede utilizar la suya propia. Ahora bien, en los algoritmos de ejemplo de este tutorial, la mayoría de las palabras que se utilizan son una traducción literal –del inglés al castellano– de las palabras que se usan en lenguaje C para escribir las instrucciones de los programas. Por tanto, el pseudocódigo empleado en este tutorial es, mayormente, *"C En Español (**CEE**)"*. Se pretende de esta forma facilitar al estudiante la codificación posterior de los algoritmos de los ejemplos al lenguaje C. La codificación de un algoritmo consiste en traducirlo a un lenguaje de programación específico, C en nuestro caso.

EJEMPLO Si se desea crear un programa que calcule la suma de dos números enteros cualesquiera introducidos por el usuario y, después, muestre por pantalla el resultado obtenido:

```
Introduzca el primer número (entero): 3
Introduzca el segundo número (entero): 5
La suma es: 8
```

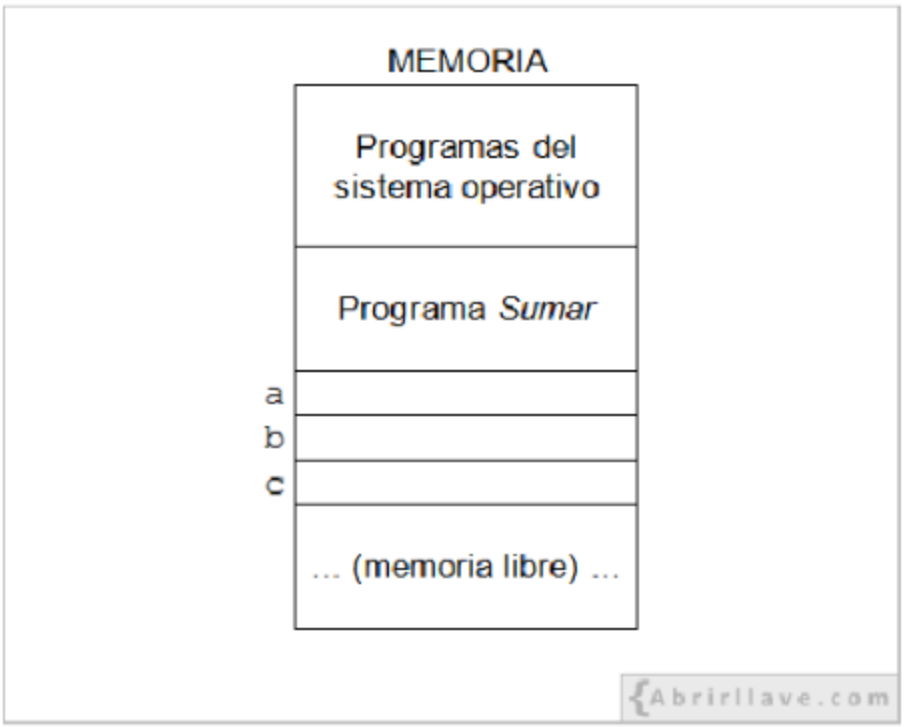
Se puede escribir el algoritmo siguiente:

```
algoritmo Sumar

variables
   entero a, b, c

inicio
   escribir( "Introduzca el primer número (entero): " )
   leer( a )
   escribir( "Introduzca el segundo número (entero): " )
   leer( b )
   c ← a + b
   escribir( "La suma es: ", c )
fin
```

Un algoritmo escrito en pseudocódigo siempre se suele organizar en tres secciones: cabecera, declaraciones y cuerpo. En la sección de cabecera se escribe el nombre del algoritmo, en este caso **Sumar**. En la sección de declaraciones se declaran algunos de los objetos que va a utilizar el programa. En el tutorial de lenguaje C de Abrirllave se estudian en detalle los distintos tipos de objetos que pueden ser utilizados en un programa, tales como: variables, constantes, subprogramas, etc. De momento, obsérvese que, en este ejemplo, las variables **a**, **b** y **c** indican que el programa necesita tres espacios en la memoria principal de la computadora para guardar tres números enteros. Cada una de las variables hace referencia a un espacio de memoria diferente.



En el cuerpo están descritas todas las acciones que se tienen que llevar a cabo en el programa, y siempre se escriben entre las palabras **inicio** y **fin**. La primera acción:

```
escribir( "Introduzca el primer número (entero): " )
```

Indica que se debe mostrar por pantalla el mensaje que hay entre comillas dobles. Después, mediante la acción:

```
leer( a )
```

Se está indicando que el programa esperará a que el usuario teclee un número entero, el cual se almacenará en el espacio de memoria representado por la variable **a**. El mismo proceso se tiene que seguir con el segundo número, que se guardará en el espacio de memoria representado por la variable **b**.

```
escribir( "Introduzca el segundo número (entero): " )
leer( b )
```

Acto seguido, la acción:

```
c ← a + b
```

Indica que en el espacio de memoria representado por la variable **c** se debe almacenar la suma de los dos números introducidos por el usuario del programa. Para terminar, el resultado de la suma se mostrará por pantalla con la acción:

```
escribir( "La suma es: ", c )
```

1.3. Cualidades de un algoritmo

Para cualquier problema dado no existe una única solución algorítmica; es tarea de la persona que diseña un algoritmo encontrar la solución más optima, ésta no es otra que aquella que cumple más fielmente las cualidades deseables de todo algoritmo bien diseñado:

- **Finitud**. Un algoritmo siempre tiene que finalizar tras un número finito de acciones. Cuando el algoritmo **Sumar** sea ya un programa, la ejecución de éste siempre será la misma, ya que, siempre se seguirán las acciones descritas en el cuerpo del algoritmo, una por una, desde la primera hasta la última y en el orden establecido.
- **Precisión**. Todas las acciones de un algoritmo deben estar bien definidas, esto es, ninguna acción puede ser ambigua, sino que cada una de ellas solamente se debe poder interpretar de una única manera. Dicho de otra forma, si el programa que resulta de un algoritmo se ejecuta varias veces con los mismos datos de entrada, en todos los casos se obtendrán los mismos datos de salida.
- **Claridad**. Lo normal es que un problema se pueda resolver de distintas formas. Por tanto, una de las tareas más importantes del diseñador de un algoritmo es encontrar la solución más legible, es decir, aquella más comprensible para el ser humano.
- **Generalidad**. Un algoritmo debe resolver problemas generales. Por ejemplo, el programa **Sumar** deberá servir para realizar sumas de dos números enteros cualesquiera, y no, solamente, para sumar dos números determinados, como pueden ser el 3 y el 5.

- **Eficiencia.** La ejecución del programa resultante de codificar un algoritmo deberá consumir lo menos posible los recursos disponibles del ordenador (memoria, tiempo de CPU, etc.).
- **Sencillez.** A veces, encontrar la solución algorítmica más eficiente a un problema puede llevar a escribir un algoritmo muy complejo, afectando a la claridad del mismo. Por tanto, hay que intentar que la solución sea sencilla, aun a costa de perder un poco de eficiencia, es decir, se tiene que buscar un equilibrio entre la claridad y la eficiencia. Escribir algoritmos sencillos, claros y eficientes se consigue a base de práctica.
- **Modularidad.** Nunca hay que olvidarse del hecho de que un algoritmo puede formar parte de la solución a un problema mayor. Pero, a su vez, dicho algoritmo debe descomponerse en otros, siempre y cuando, esto favorezca a la claridad del mismo.

La persona que diseña un algoritmo debe ser consciente de que todas las propiedades de un algoritmo se transmitirán al programa resultante.

1.4. Codificación

Una vez que los algoritmos de una aplicación han sido diseñados, ya se puede iniciar la fase de **codificación**. En esta etapa se tienen que traducir dichos algoritmos a un lenguaje de programación específico,

Para codificar un algoritmo hay que conocer la sintaxis del lenguaje al que se va a traducir. Sin embargo, independientemente del lenguaje de programación en que esté escrito un programa, será su algoritmo el que determine su lógica. La **lógica de un programa** establece cuáles son sus acciones y en qué orden se deben ejecutar. Por tanto, es conveniente que todo programador aprenda a diseñar algoritmos antes de pasar a la fase de codificación.

Tipos de datos

Los datos que utilizan los programas se pueden clasificar en base a diferentes criterios. Uno de los más significativos es aquel que dice que todos los datos que utilizan los programas son simples o compuestos.

Un **dato simple** es indivisible (atómico), es decir, no se puede descomponer.

EJEMPLO Un `año` es un dato simple.

```
Año...: 2006
```

Un `año` se expresa con un número entero, el cual no se puede descomponer. Sin embargo, un **dato compuesto** está formado por otros datos.

EJEMPLO Una `fecha` es un dato compuesto por tres datos simples (`día`, `mes`, `año`).

```
Fecha:
   Día...: 30
   Mes...: 11
   Año...: 2006
```

A los datos compuestos también se les conoce como **datos estructurados**, ya que, son datos que se forman al agruparse otros. Por consiguiente, de los datos simples se dice que no tienen estructura.

Seguidamente, se van a estudiar cinco tipos de datos:

- Entero
- Real
- Lógico
- Carácter
- Cadena

De ellos, tan solo el tipo cadena es compuesto. Los demás son los tipos de datos simples considerados **estándares**. Esto quiere decir que la mayoría de los lenguajes de programación permiten trabajar con ellos. Por ejemplo, en C es posible utilizar datos de tipo entero, real y carácter, sin embargo, los datos de tipo lógico no se pueden utilizar, ya que, no existen en este lenguaje.

2.1. Datos de tipo numérico

Como su propio nombre indica, un **dato de tipo numérico** es aquel que puede tomar por valor un número. Existen dos tipos de datos numéricos básicos:

- Entero
- Real

EJEMPLO El número de **`asignaturas aprobadas`** por un estudiante en la universidad es un dato de tipo entero, mientras que, su **`nota`** en el examen de una asignatura en concreto puede ser de tipo real.

```
Asignaturas aprobadas.......: 4
Nota del examen de física...: 7,5
```

2.2. Datos de tipo entero

Un **dato de tipo entero** es aquel que puede tomar por valor un número perteneciente al conjunto de los números enteros (*Z*), el cual está formado por los números naturales, sus opuestos (números negativos) y el cero.

Z = { ..., -3, -2, -1, 0, 1, 2, 3, ... }

EJEMPLO La **edad** de una persona y el **año** en que nació, son dos datos de tipo entero.

```
Edad...: 29
Año....: 1976
```

Z es un conjunto infinito de números enteros, y como el ordenador no puede representarlos todos, un dato de tipo entero solamente puede tomar por valor un número perteneciente a un subconjunto de Z. Los valores máximo y mínimo de dicho subconjunto varían según las características de cada ordenador y del compilador que se utilice.

En pseudocódigo, para indicar que un dato es de tipo entero se utiliza la palabra reservada:

```
entero
```

En todos los lenguajes de programación existe un conjunto de palabras que tienen un significado especial, a estas palabras se las llama *reservadas*.

2.3. Datos de tipo real

Un **dato de tipo real** es aquel que puede tomar por valor un número perteneciente al conjunto de los números reales (*R*), el cual está formado por los números racionales e irracionales.

EJEMPLO El **peso** de una persona (en kilogramos) y su **altura** (en centímetros), son datos que pueden considerarse de tipo real.

```
Peso.....: 75,3
Altura...: 172,7
```

R es un conjunto infinito de números reales, y como el ordenador no puede representarlos todos, un dato de tipo real solamente puede tomar por valor un número perteneciente a un subconjunto de *R*. Los valores de dicho subconjunto varían según las características de cada ordenador y del compilador que se utilice.

En pseudocódigo, para indicar que un dato es de tipo real se utiliza la palabra reservada:

```
real
```

2.4. Datos de tipo lógico

Un **dato de tipo lógico** es aquel que puede tomar por valor únicamente uno de los dos siguientes:

```
{ verdadero, falso }
```

Los valores **verdadero** y **falso** son contrapuestos, de manera que, un dato de tipo lógico siempre está asociado a que algo se cumpla o no se cumpla.

EJEMPLO El **estado** de una barrera de paso de trenes es un dato que puede considerarse de tipo lógico, por ejemplo, asociando **verdadero** a que esté subida y **falso** a que esté bajada.

```
Estado...: falso
```

falso indica que la barrera está bajada.

2.5. Datos de tipo carácter

Un **dato de tipo carácter** es aquel que puede tomar por valor un carácter perteneciente al conjunto de los caracteres que puede representar el ordenador.

En pseudocódigo, el valor de un dato de tipo carácter se puede representar entre *comillas simples* (') o *dobles* ("). Pero, en este tutorial, se van a utilizar solamente las comillas simples, al igual que se hace en C.

EJEMPLO En un examen con preguntas en las que hay que seleccionar la respuesta correcta entre varias opciones dadas (a, b, c, d, e), la **`respuesta correcta`** de cada una de las preguntas es un dato de tipo carácter.

```
Respuesta correcta a la pregunta 3...: 'c'
```

En pseudocódigo, para indicar que un dato es de tipo carácter se utiliza la palabra reservada:

```
caracter
```

2.6. Datos de tipo cadena

Un dato de tipo cadena es aquel que puede tomar por valor una secuencia de caracteres.

En pseudocódigo, el valor de un dato de tipo cadena se puede representar entre *comillas simples* (') o *dobles* ("). Sin embargo, en este tutorial, se van a utilizar solamente las comillas dobles, al igual que se hace en C.

EJEMPLO El **`título`** de un libro y el **`nombre`** de su autor, son datos de tipo cadena.

```
Título...: "La Odisea"
Autor....: "Homero"
```

- **`"La Odisea"`** es una cadena de 9 caracteres.
- **`"Homero"`** es una cadena de 6 caracteres.

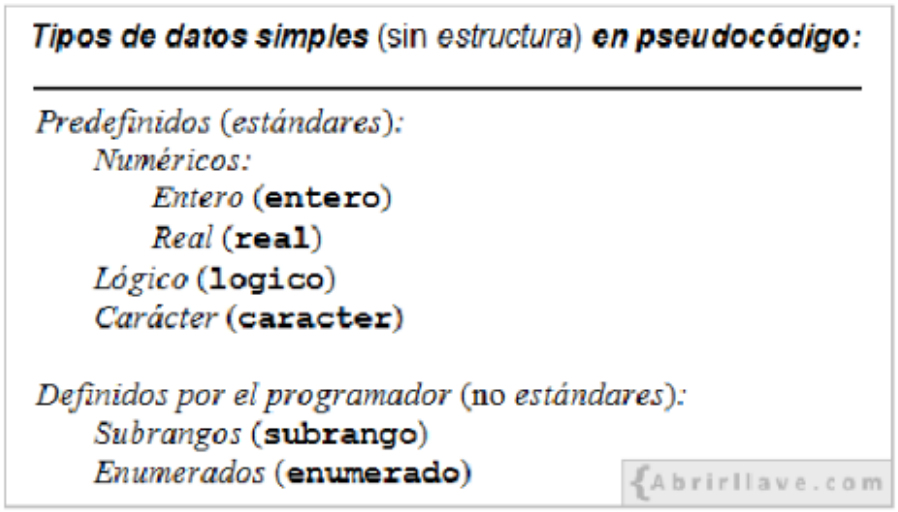
Fíjese que, en la cadena **`"La Odisea"`**, el carácter espacio en blanco también se cuenta.

En pseudocódigo, para indicar que un dato es de tipo cadena se utiliza la palabra reservada:

```
cadena
```

2.7. Clasificación de los tipos de datos simples

Los tipos de datos simples se clasifican en predefinidos y definidos por el programador. La clasificación completa es:



Los **tipos de datos simples predefinidos** (estándares) son aquellos proporcionados por los lenguajes de programación. Pero, el programador también puede definir sus propios tipos de datos simples (subrangos y enumerados), los cuales se estudiarán más adelante.

Todos los datos simples son ordinales, excepto el dato de tipo real. Un **dato ordinal** es aquel que puede tomar por valor un elemento perteneciente a un conjunto en el que todo elemento tiene un predecesor y un sucesor, excepto el primero y el último. Por ejemplo, el valor 5, perteneciente al conjunto de los números enteros, tiene como predecesor al 4, y como sucesor al 6. Sin embargo, entre dos números reales siempre hay un número infinito de números.

Identificadores, variables y constantes

Para diseñar algoritmos en pseudocódigo, se pueden utilizar los siguientes elementos:

- Tipos de datos
- Variables
- Constantes
- Operadores
- Expresiones
- Instrucciones

Todos ellos están relacionados entre sí. En este capítulo se va a ver la relación que existe entre los tipos de datos, las variables y las constantes.

3.1. Identificadores

La mayoría de los elementos de un algoritmo escrito en pseudocódigo se diferencian entre sí por su nombre. Por ejemplo, los tipos de datos básicos se nombran como: `entero`, `real`, `logico` y `caracter`.

Cada uno de ellos es un identificador. Un **identificador** es el nombre que se le da a un elemento de un algoritmo (o programa). Por ejemplo, el tipo de dato `entero` hace referencia a un tipo de dato que es distinto a todos los demás tipos de datos, es decir, los valores que puede tomar un dato de tipo entero, no son los mismos que los que puede tomar un dato de otro tipo.

Los identificadores `entero`, `real`, `logico` y `caracter` están predefinidos, forman parte del lenguaje algorítmico. No obstante, en un algoritmo también pueden existir identificadores definidos por el programador. Por ejemplo, un algoritmo puede utilizar variables y constantes definidas por el programador. Además, los algoritmos también se deben nombrar mediante un identificador.

En pseudocódigo, a la hora de asignar un nombre a un elemento de un algoritmo, se debe de tener en cuenta que todo identificador debe cumplir unas reglas de sintaxis. Para ello, en

nuestro pseudocódigo CEE (*"C en Español"*), vamos a seguir las mismas reglas de sintaxis que existen en lenguaje C:

1. Consta de uno o más caracteres.
2. El primer carácter debe ser una letra o el carácter guion bajo (_), mientras que, todos los demás pueden ser letras, dígitos o el carácter guion bajo (_). Las letras pueden ser minúsculas o mayúsculas del alfabeto inglés. Así pues, no está permitido el uso de las letras 'ñ' y 'Ñ'.
3. No pueden existir dos identificadores iguales, es decir, dos elementos de un algoritmo no pueden nombrarse de la misma forma. Lo cual no quiere decir que un identificador no pueda aparecer más de una vez en un algoritmo.

De la segunda regla se deduce que un identificador no puede contener caracteres especiales, salvo el carácter guion bajo (_). Es importante resaltar que las vocales no pueden llevar tilde ni diéresis.

Los identificadores son sensibles a minúsculas y mayúsculas.

EJEMPLO `Mes` y `mes` son considerados identificadores distintos.

Es aconsejable que los identificadores tengan un significado afín a lo que representan.

EJEMPLO El algoritmo de un programa que tiene las tareas de:

1º) Recoger por teclado dos datos de tipo entero (como datos de entrada).

2º) Realizar la suma de ellos.

3º) Mostrar por pantalla el resultado de la suma (como dato de salida).

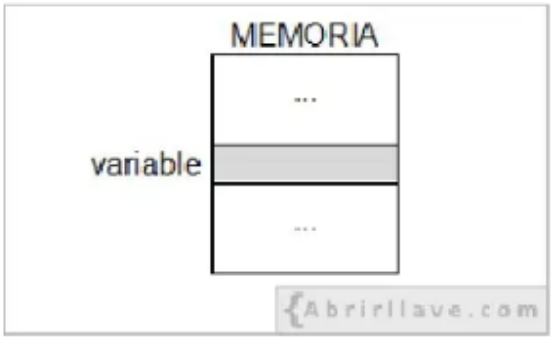
Estará mejor nombrado mediante el identificador **`SUMA`**, que, por ejemplo, mediante el identificador `Algoritmo_1`, aunque ambos identificadores sean válidos sintácticamente.

3.1.1. Palabras reservadas

Las **palabras reservadas** son identificadores predefinidos (tienen un significado especial). En todos los lenguajes de programación existe un conjunto de palabras reservadas. Por el momento, en pseudocódigo, se han estudiado las siguientes: **`cadena`**, **`caracter`**, **`entero`**, **`falso`**, `logico`, `real` y **`verdadero`**.

3.2. Variables

Una **variable** representa a un espacio de memoria en el cual se puede almacenar un dato. Gráficamente, se puede representar como:



Como ya se ha estudiado, los datos que utilizan los programas pueden ser de diferentes tipos. Así que, de cada variable se debe especificar –definir– el tipo de dato que puede almacenar.

EJEMPLO Si un programa va a utilizar un dato de tipo carácter, será necesaria una variable de tipo carácter, y en el espacio de memoria reservado para dicha variable se podrá almacenar cualquier carácter perteneciente al conjunto de los caracteres representables por el ordenador.

El programador, cuando desarrolla un programa –o diseña un algoritmo– debe decidir:

1. Cuántas son las variables que el programa necesita para realizar las tareas que se le han encomendado.
2. El tipo de dato que puede almacenar cada una de ellas.

Durante la ejecución de un programa, el valor que tome el dato almacenado en una variable puede cambiar tantas veces como sea necesario, pero, siempre, tomando valores pertenecientes al tipo de dato que el programador ha decidido que puede almacenar dicha variable, ya que, el tipo de dato de una variable no puede ser cambiado durante la ejecución de un programa.

3.2.1. Declaración de variables

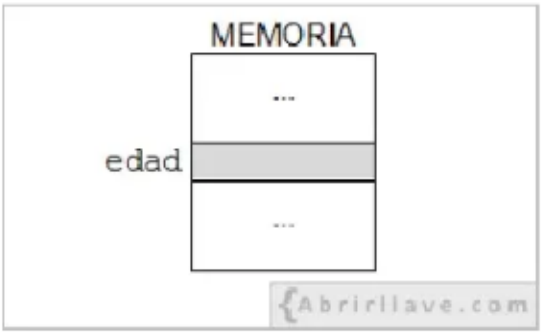
Para que un programa pueda hacer uso de una o más variables, estas deben ser declaradas previamente. Todas las variables de un programa se declaran de la misma forma, indicando de cada una de ellas:

- El tipo de dato que puede almacenar (mediante un identificador).
- Su nombre (mediante otro identificador).

EJEMPLO La declaración de una variable para almacenar la edad de una persona se escribe:

```
entero edad
```

Por tanto, en la memoria de la computadora se reservará un espacio para almacenar la **edad**:



En un programa no se pueden declarar varias variables con el mismo nombre, salvo excepciones que estudiaremos más adelante. Sin embargo, sí pueden existir varias variables del mismo tipo de dato.

Siguiendo con el ejemplo anterior, si también se quiere declarar una variable para almacenar su número de hijos, se debe escribir:

```
entero edad
entero numero_de_hijos
```

Las variables de un programa no tienen por qué estar contiguas en la memoria del ordenador:

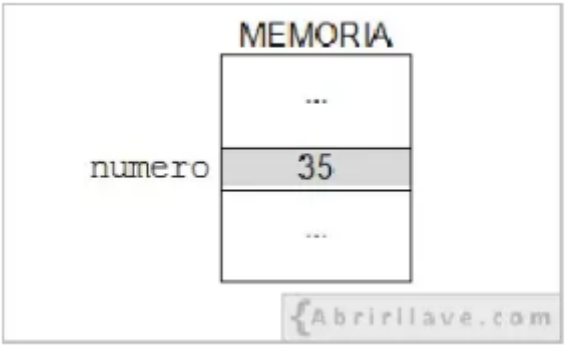
Puesto que las dos variables son del mismo tipo de dato, se pueden declarar en la misma línea separándolas por medio de una *coma* (,).

```
entero edad, numero_de_hijos
```

Opcionalmente, cuando se declara una variable, a esta se le puede asignar un valor inicial.

EJEMPLO Si se desea declarar una variable para almacenar un número entero y que, inicialmente, contenga el valor 35, se debe escribir:

```
entero numero = 35
```



3.3. Constantes

Una **constante** representa a un valor –dato almacenado en memoria– que no puede cambiar durante la ejecución de un programa.

En lenguaje C, una constante puede ser de tipo entero, real, carácter, cadena o enumerado. Las constantes de tipo enumerado se van a estudiar en el capítulo siguiente. En cuanto a las demás, se pueden expresar de dos formas diferentes:

- Por su valor.
- Con un nombre (identificador).

EJEMPLO Las siguientes constantes de tipo entero están expresadas por su valor:

```
-5

10
```

Para expresar una constante con un nombre, la constante debe ser declarada previamente.

3.3.1. Constantes de tipo entero

Una **constante de tipo entero** es aquella que representa a un valor –dato– perteneciente al subconjunto de *Z* representable por el ordenador.

EJEMPLO Suponiendo que el ordenador –utilizando dieciséis bits– pueda representar, en Complemento a 2, el siguiente conjunto de valores enteros:

{ -32768, -32767, ..., -1, 0, 1, ..., 32766, 32767 }

Algunos ejemplos de constantes de tipo entero son:

```
-32000

0

000077

+1111
```

Obsérvese que, además de los caracteres numéricos, dígitos del (**0**) al (**9**), también se puede hacer uso de los caracteres especiales (**+**) y (**-**) para indicar el signo de un número entero, el cual es positivo por omisión. Sin embargo, en pseudocódigo, y también en lenguaje C, es incorrecto usar los caracteres coma (,) y/o punto (.) para expresar constantes de tipo entero.

EJEMPLO Por tanto, es incorrecto escribir:

```
-32.000

0,0

+1,111.00
```

3.3.2. Constantes de tipo real

Una **constante de tipo real** es aquella que representa a un valor –dato– perteneciente al subconjunto de *R* representable por el ordenador.

EJEMPLO Algunos ejemplos son:

`8.12`

`000.333` (Los ceros a la izquierda no son significativos)

`+1111.809`

`-3200.` (También se puede escribir `-3200.0`)

`.56` (También se puede escribir `0.56`)

Obsérvese que, además de los caracteres numéricos, dígitos del (**0**) al (**9**), también se puede hacer uso de los caracteres especiales (**+**) y (**-**) para indicar el signo de un número real. Además, en lenguaje C y, por tanto, también en nuestro pseudocódigo CEE, obligatoriamente debe aparecer el carácter punto (**.**), o el carácter (**e**) o (**E**) seguido del exponente, del cual también puede indicarse su signo con los caracteres (**+**) y (**-**). Los signos del exponente y del número en sí, por omisión, son positivos.

EJEMPLO Las siguientes constantes de tipo real están expresadas correctamente:

```
-77e-3

+1111e+2

2000E+2

3040e2
```

EJEMPLO Algunos ejemplos de constantes de tipo real incorrectas son:

`-200` (No aparece el punto ni el exponente)

`-20,0` (No puede aparecer la coma)

`--111.` (No se puede duplicar el signo)

`-111..` (No se puede duplicar el punto)

`-111.11.` (No puede aparecer más de un punto)

`+22e` (Después del carácter (**e**) o (**E**) se debe escribir el exponente)

`+22ee6` (No se puede duplicar el carácter (**e**) o (**E**))

`+22e 6` (No se puede escribir el carácter espacio en blanco)

`38E-2.2` (El exponente debe ser una cantidad entera)

3.3.3. Constantes de tipo lógico

Una **constante de tipo lógico** es aquella que representa a un valor –dato– perteneciente al conjunto:

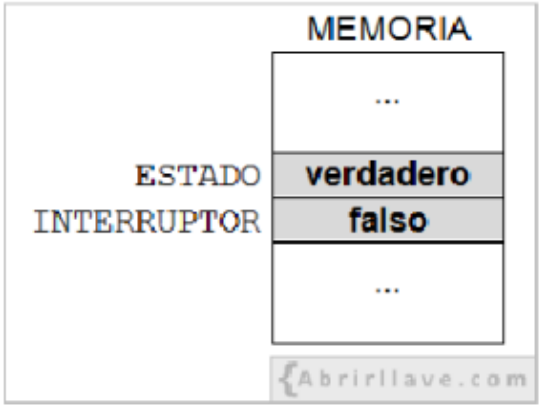
{ **verdadero**, **falso** }

verdadero y **falso** son palabras reservadas –identificadores– que, en sí mismas, representan a constantes de tipo lógico. En consecuencia, aunque se pueden definir constantes de tipo lógico, no suele ser habitual declarar constantes de este tipo de dato:

EJEMPLO

```
ESTADO = verdadero

INTERRUPTOR = falso
```



3.3.4. Constantes de tipo carácter

Una **constante de tipo carácter** es aquella que representa a un valor –dato– perteneciente al conjunto de caracteres que puede representar el ordenador.

EJEMPLO Las siguientes constantes de tipo carácter están expresadas por su valor:

```
'a'

'T'

'5'

'+'

'.'
```

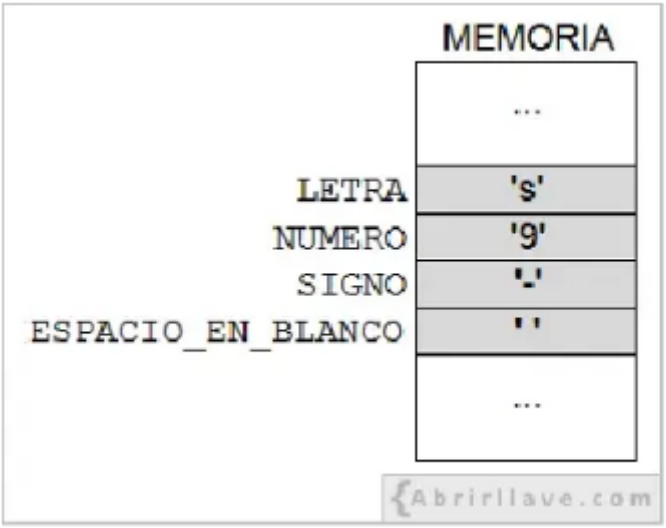
EJEMPLO Algunos ejemplos de declaración de constantes de tipo carácter son:

```
LETRA = 's'

NUMERO = '9'

SIGNO = '-'

ESPACIO_EN_BLANCO = ' '
```



3.3.5. Constantes de tipo cadena

Una **constante de tipo cadena** es aquella que representa a un valor –dato– de tipo cadena, es decir, representa a una secuencia de caracteres.

EJEMPLO Las siguientes constantes de tipo cadena están expresadas por su valor:

```
"Alejandro"
"Lucerna"
"Barcelona 2000"
```

EJEMPLO Algunos ejemplos de declaración de constantes de tipo cadena son:

```
NOMBRE = "Alejandro"
CIUDAD = "Lucerna"
OLIMPIADAS = "Barcelona 2000"
```

Operadores y expresiones

En un programa, el tipo de un dato determina las operaciones que se pueden realizar con él. Por ejemplo, con los datos de tipo entero se pueden realizar operaciones aritméticas, tales como la suma, la resta o la multiplicación.

EJEMPLO Algunos ejemplos son:

`111 + 6` (operación *suma*)

`19 - 72` (operación *resta*)

`24 * 3` (operación *multiplicación*)

Todas las operaciones del ejemplo constan de dos operandos –constantes enteras– y un operador. La mayoría de las veces es así, pero, también es posible realizar operaciones con distinto número de operadores y/u operandos.

EJEMPLO

`111 + 6 - 8` (tres operandos y dos operadores)

`- ( ( +19 ) + 72 )` (dos operandos y tres operadores)

`- ( -72 )` (un operando y dos operadores)

`- ( +43 )` (un operando y dos operadores)

En las operaciones del ejemplo se puede observar que los caracteres *más* (+) y *menos* (–) tienen dos usos:

1. Operadores suma y resta.
2. Signos de un número (también son operadores).

Los operadores de signo más (+) y menos (–) son **operadores monarios**, también llamados **unarios**, ya que, actúan, solamente, sobre un operando.

Los caracteres *abrir paréntesis* "**(**" y *cerrar paréntesis* "**)**" se utilizan para establecer la prioridad de los operadores, es decir, para establecer el orden en el que los operadores actúan sobre los operandos.

EJEMPLO Obsérvese la diferencia entre las siguientes operaciones:

`-19 + 72` (dos operandos y dos operadores)

`- ( 19 + 72 )` (dos operandos y dos operadores)

Los resultados de evaluarlas son, respectivamente:

53 (primero actúa el operador signo (–) y después el operador suma (+))

-91 (primero actúa el operador suma (+) y después el operador signo (–))

5.1. Expresiones aritméticas

De la evaluación de una **expresión aritmética** siempre se obtiene un valor de tipo entero o real. En las expresiones aritméticas se pueden utilizar los siguientes **operadores aritméticos**:

Operadores aritméticos en pseudocódigo	
Operador	***Descripción***
`+`	Suma
`-`	Resta
`*`	Multiplicación
`**`	Potencia (también se utiliza la *flecha arriba* ↑ o el *acento circunflejo* ^)
`/`	División real
`div`	División entera (también se utiliza el carácter *barra invertida* \)
`mod`	Módulo (resto de la división entera)
`+`	Signo más
`-`	Signo menos

Para poder evaluar correctamente las expresiones aritméticas del ejemplo, es necesario seguir un criterio de prioridad de operadores. En nuestro pseudocódigo CEE, la prioridad de los operadores aritméticos es:

Prioridad de los operadores aritméticos (de mayor a menor) en pseudocódigo	
Operadores	***Descripción***
`+ -`	Signos más y menos
`**`	Potencia
`* / div mod`	Multiplicación, división real, división entera y módulo
`+ -`	Suma y resta

5.2. Expresiones lógicas

De la evaluación de una **expresión lógica** siempre se obtiene un valor de tipo lógico (`verdadero` o `falso`). En las expresiones lógicas se pueden utilizar dos tipos de operadores:

- Relacionales.
- Lógicos.

Un **operador relacional** se utiliza para comparar los valores de dos expresiones. Estas deben ser del mismo tipo (aritméticas, lógicas, de carácter o de cadena).

EJEMPLO Algunos ejemplos son:

`22 > 13` (comparación de dos expresiones aritméticas)

`22.5 < 3.44` (comparación de dos expresiones aritméticas)

`verdadero = falso` (comparación de dos expresiones lógicas)

`'c' > 'f'` (comparación de dos expresiones de carácter)

`"coche" = "Coche"` (comparación de dos expresiones de cadena)

Proporcionan los valores:

`verdadero` (22 es mayor que 13)

`falso` (22.5 no es menor que 3.44)

`falso` (`verdadero` no es igual que `falso`)

`falso` (`'c'` no es mayor que `'f'`)

`falso` (`"coche"` no es igual que `"Coche"`)

Los operadores relacionales son:

Operadores relacionales en pseudocódigo	
Operador	***Descripción***
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
=	Igual que
<>	Distinto que

EJEMPLO La comparación entre valores de tipo cadena se realiza carácter a carácter.

"a" = "b" (se compara "a" con "b")

"bc" <> "kd" (se compara "b" con "k" y "c" con "d")

"126" < "9" (se compara "1" con "9")

"ab" <= "ab" (se compara "a" con "a" y "b" con "b")

"abb" >= "abc" (se compara "a" con "a", "b" con "b" y "b" con "c")

De estas expresiones se obtienen los valores:

falso ("a" no es igual que "b")

verdadero ("bc" es distinto que "kd")

verdadero ("1" es menor que "9")

verdadero ("ab" es menor o igual que "ab")

falso ("abb" no es mayor o igual que "abc")

Un **operador lógico** actúa, exclusivamente, sobre valores de expresiones lógicas. Los operadores lógicos son:

Operadores lógicas en pseudocódigo	
Operador	***Descripción***
y	Conjunción
o	Disyunción
no	Negación

El operador conjunción (**y**) y el operador disyunción (**o**) siempre actúan sobre dos operandos, mientras que, el operador negación (**no**) solamente actúa sobre un operando, o dicho de otra forma, es un operador monario.

El modo en que actúan los operadores lógicos se resume en las llamadas tablas de verdad, definidas por el matemático George Boole.

La tabla de verdad del **operador conjunción** (**y**) es:

Tabla de verdad del operador conjunción (y)		
<expresión_1>	**<expresión_2>**	**<expresión_1> y <expresión_2>**
verdadero	**verdadero**	**verdadero**
verdadero	**falso**	**falso**
falso	**verdadero**	**falso**
falso	**falso**	**falso**

La tabla de verdad del **operador disyunción** (**o**) es:

Tabla de verdad del operador disyunción (o)		
<expresión_1>	**<expresión_2>**	**<expresión_1> o <expresión_2>**
verdadero	**verdadero**	**verdadero**
verdadero	**falso**	**verdadero**
falso	**verdadero**	**verdadero**
falso	**falso**	**falso**

5.3. Expresiones de carácter

Aunque no existe ningún operador de caracteres, sí que existen expresiones de carácter. De la evaluación de una **expresión de carácter** siempre se obtiene un valor de tipo carácter.

EJEMPLO Dadas las siguientes declaraciones de constantes y variables en pseudocódigo:

```
CONSONANTE = 'S'

caracter letra = 'X'

caracter opcion = '3'
```

Algunas expresiones de carácter son:

```
opcion

letra

CONSONANTE

'a'
```

Los resultados de evaluarlas son:

```
'3'

'X'

'S'

'a'
```

5.4. Expresiones de cadena

De la evaluación de una **expresión de cadena** siempre se obtiene un valor de tipo cadena. Solamente existe un operador de cadena:

Operador de cadena en pseudocódigo	
Operador	***Descripción***
+	Concatenación

El **operador concatenación** (+) realiza la concatenación de dos operandos de tipo cadena, es decir, los encadena.

EJEMPLO Dadas las siguientes declaraciones de constantes y variables en pseudocódigo:

```
OLIMPIADA = "Atenas 2004"

PUNTO = "."

cadena nombre = "Pedro", apellido = "Cosín", rio = "Tajo"
```

Algunas expresiones de cadena son:

```
OLIMPIADA + PUNTO

nombre + " " + apellido

"Buenos días" + PUNTO

rio

nombre + " fue a las Olimpiadas de " + OLIMPIADA + PUNTO
```

Los resultados de evaluarlas son:

```
"Atenas 2004."

"Pedro Cosín"

"Buenos días."

"Tajo"

"Pedro fue a las Olimpiadas de Atenas 2004."
```