Conceptos básicos

Contenido

Algoritmo

La palabra *algoritmo* procede de la deformación de la forma latina del nombre del matemático persa ``*Abu Ja'far Mohamed ibn Musa* ***al Khowarizmi***'' *(825 D.C.)*. Si consultamos la definición actual de la palabra en un diccionario enciclopédico podemos encontrarnos con algo así: Conjunto ordenado y finito de operaciones que permite obtener la solución de un problema.

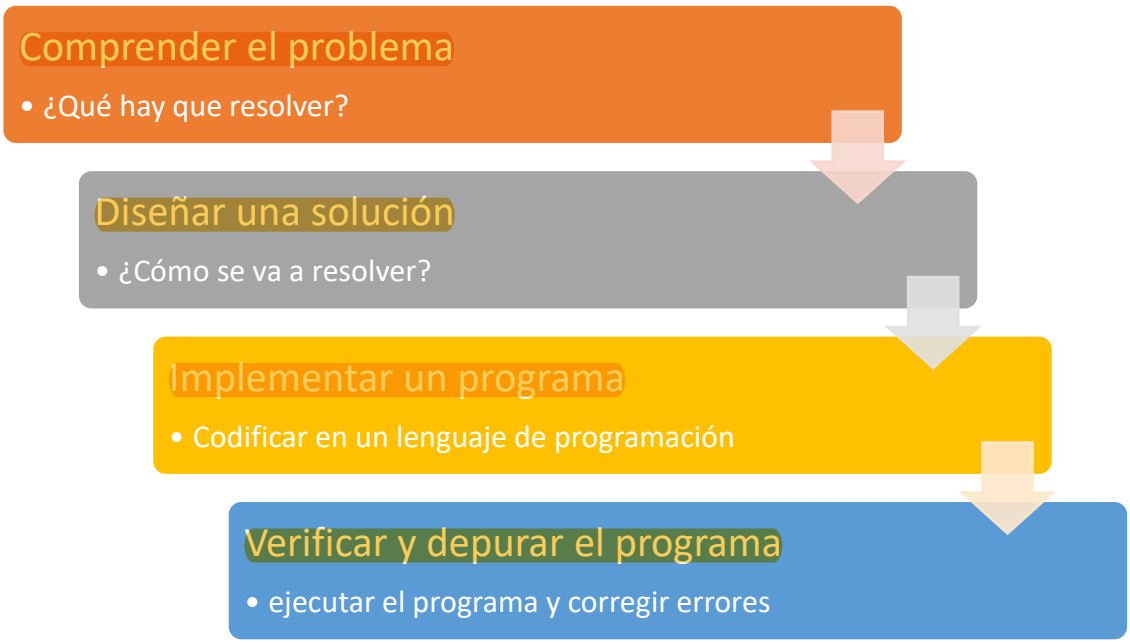
Ahora bien, aplicado a la informática y en concreto a la realización de programas, podemos verlo como la descripción precisa de una sucesión de instrucciones que permite a un computador llevar a cabo un trabajo para solucionar un determinado problema.

Programa

Conjunto de instrucciones ordenadas **escritas en un lenguaje de programación** para que un ordenador lleve a cabo una determinada tarea.

Fases en el desarrollo de un programa

La siguiente figura ilustra las fases que se deben seguir para desarrollar un programa



Lenguajes de programación

Las instrucciones que constituyen un programa hay que escribirlas de forma que el ordenador las entienda, para ello hay que emplear un lenguaje de programación.

El lenguaje que usa el ordenador es el código máquina y es muy difícil para nosotros, por lo que emplearemos lenguajes más parecidos al lenguaje humano llamados **lenguajes de alto nivel.** Son lenguajes en los que se pueden expresar los algoritmos de una forma que al cerebro humano le resulta más cercana. Ejemplos de lenguajes de alto nivel son: C/C++, Python, Java, PHP, etc.

También existen los **lenguajes de bajo nivel**, son más cercanos al ordenador, pero resulta más difícil trabajar con ellos ya que no disponen del nivel de abstracción de los lenguajes de alto nivel.

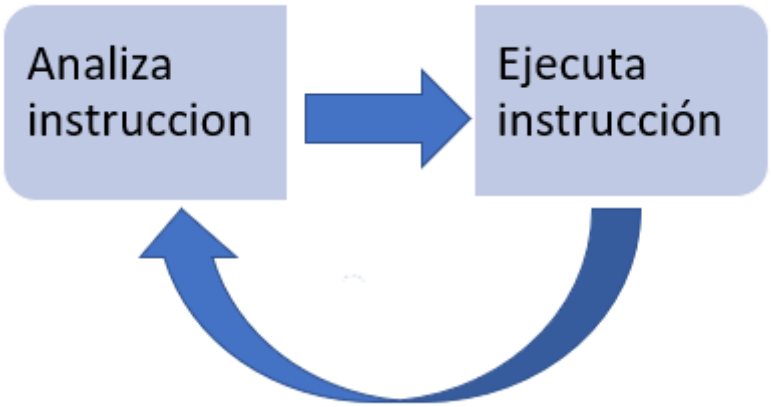
Un ordenador sólo entiende el código máquina. Hay dos tipos de programas que traducen lenguajes de alto nivel a código máquina: **intérpretes** y **compiladores**

Compiladores e intérpretes

El compilador analiza nuestro programa comprobando su sintaxis e indicando los errores de escritura, y **genera** el programa en lenguaje máquina. Puede que necesite un **enlazado** (linkado), en donde se le unen una serie de módulos de librería.

Al fichero que escribe el programador con las instrucciones se le llama **código fuente** y al fichero que genera el compilador se le llama **ejecutable**.

El intérprete es parecido a un compilador, pero no crea un fichero ejecutable, analiza y ejecuta un programa sentencia a sentencia.



En algunos lenguajes, es frecuente encontrar compiladores, pero no suelen existir intérpretes, es el caso del lenguaje C/C++, por ejemplo. Para otros lenguajes, lo habitual es trabajar con intérpretes y no con compiladores, como ocurre con Python, Ruby y PHP.

Pseudocódigo

Llamaremos **pseudocódigo** o **pseudolenguaje** a la combinación de las construcciones de un lenguaje formal de programación con proposiciones

informales expresadas en el lenguaje empleado por el creador del algoritmo - *en nuestro caso el español*-.

Si el algoritmo expresado en pseudocódigo está lo suficientemente detallado, la escritura del programa no consistirá más que en la traducción del pseudocódigo final obtenido **al lenguaje de programación elegido;** requiriendo esta tarea un esfuerzo relativamente pequeño por parte del programador.

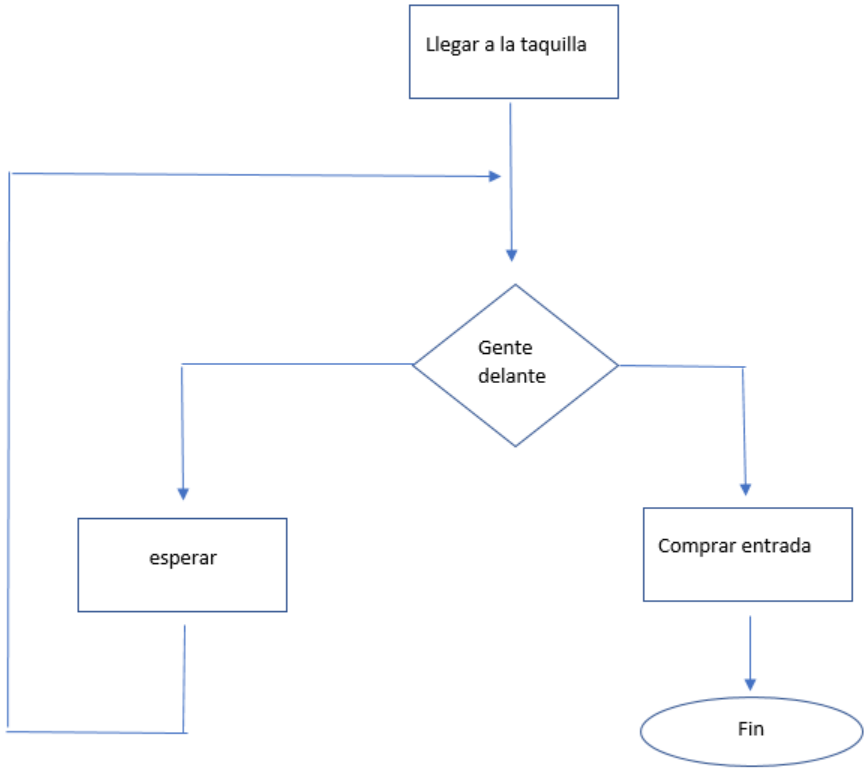
Ejemplo de pseudocódigo: pensemos en lo que ocurre cuando, para comprar una entrada en el cine, tras llegar a la taquilla, lo lógico es que esperemos en la cola hasta que nos llegue el turno, más o menos algo así:

```
algoritmo Comprar una entrada
   llegar a la taquilla
   mientras haya gente delante
      esperar a que llegue nuestro turno
   comprar entrada
fin_algoritmo
```

Si observamos el algoritmo anterior, nos daremos cuenta que aparece la palabra **mientras**, la cual proporciona la idea de repetición -*si hay cola, debemos esperar*- que es lo que hacemos realmente cuando llegamos a la taquilla y tenemos gente delante.

Diagramas de flujo

Un diagrama de flujo es un diagrama que describe un proceso o algoritmo informático. Los diagramas de flujo emplean formas geométricas junto con flechas para describir los pasos a seguir. Son una representación muy intuitiva para expresar un algoritmo. En ellos se resalta, principalmente, la lógica o el flujo de control del programa, es decir, lo que se hace en cada momento. Ejemplo de diagrama de flujo para el ejemplo del cine:



Representación de la información

Los ordenadores representan la información usando dos dígitos, 0 y 1 (sistema binario). Los dígitos de este sistema se denominan **bits**. Una agrupación de 8 bits constituye un **byte**.

Representación de números

Los números se representan utilizando una representación binaria

Decimal	**Binario**
0	00000000
1	00000001
15	00001111

Los números negativos se pueden representar de varias formas, una de las más comunes es la representación en complemento a 2. Los números reales se pueden representar de varias maneras, por ejemplo, en notación en coma flotante (con mantisa y exponente). Ejemplo utilizando 32 bits:

Signo Exponente Mantisa

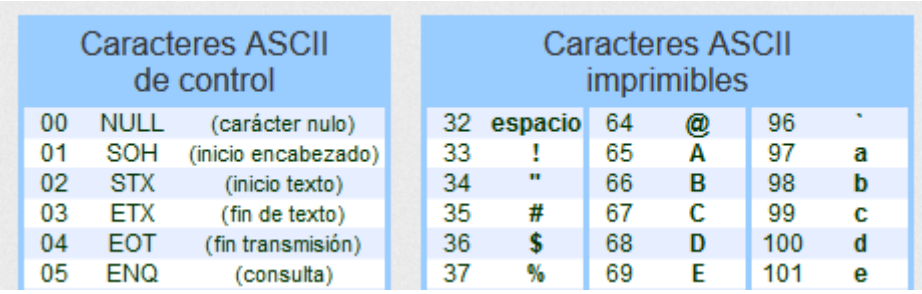
Bit 31 bits 30-23 bits 22-0

Representación de caracteres

El conjunto de caracteres codificable en un ordenador se denomina **juego de caracteres**. Inicialmente se usaba el juego de caracteres **ASCII**. Se usaban los 7 primeros bits de un byte para representar:

- letras
- dígitos del 0 al 9
- caracteres especiales y de puntuación
- 32 caracteres de control (salto de línea, etc.)

La siguiente figura muestra un fragmento de la tabla ASCII donde se puede ver para un determinado carácter cuál es su codificación ASCII (en decimal).



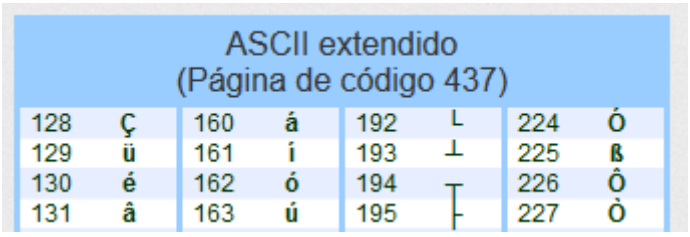
Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e

En total el juego de caracteres ASCII permite definir 127 caracteres. Este conjunto no permite representar algunos caracteres como las vocales acentuadas o la ñ, así que se propuso usar el octavo bit para extender el juego de caracteres y poder representar los caracteres especiales y surge el **ASCII extendido.** A continuación, se puede ver un fragmento.



ASCII extendido (Página de código 437)

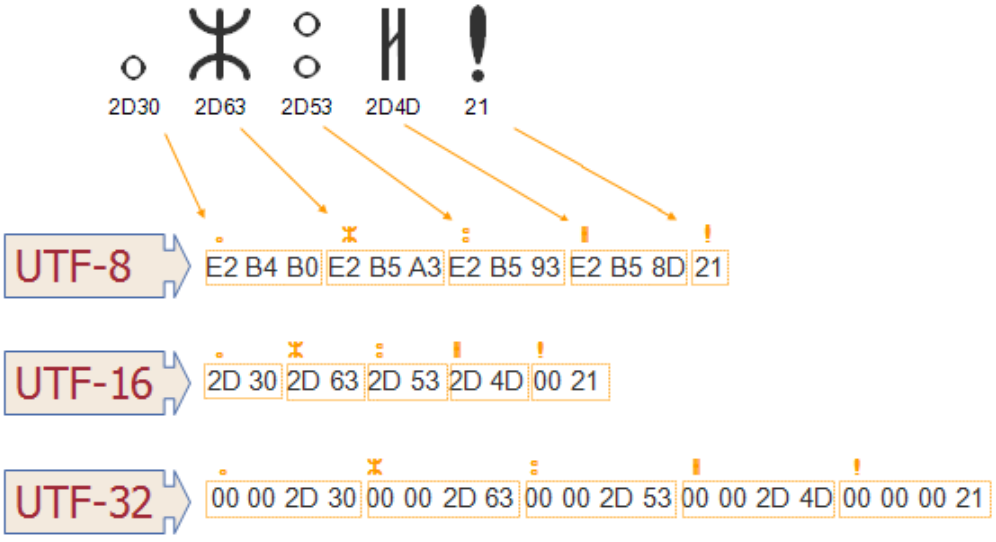
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò

Aun así, hay caracteres que no es posible representar, como por ejemplo los símbolos del alfabeto chino, etc. Es el momento en el que aparece el **Unicode**, es una evolución del ASCII que abarca todos los caracteres de las

ortografías del mundo. A diferencia del ASCII, Unicode puede codificar todos los idiomas, ideogramas, emojis, etc. El Unicode es un estándar que se utiliza para la codificación de caracteres de manera universal.

Una **codificación de caracteres** es la clave que convierte un determinado byte o una determinada secuencia de bytes en los caracteres específicos que la fuente representa como texto, es decir, define una tabla que asocia cada carácter con un número. En Unicode a ese número se le llama **punto de código (***code point***)**. Además, en Unicode existen distintas formas de codificar el mismo carácter, por ejemplo, la letra 'á' se puede representar mediante 2 bytes con una codificación y con cuatro en otra. Los formatos de Unicode son:

- UTF-32: emplea 4 bytes para todos los caracteres.
- UTF-16, utiliza 2 bytes para cualquier carácter en el BMP *(Basic Multilingual Plane*[1]) y 4 bytes para los caracteres complementarios.
- UTF-8: utiliza 1 byte para representar caracteres en el set ASCII, dos bytes para caracteres en otros bloques alfabéticos y tres bytes para el resto del BMP. Para los caracteres complementarios se utilizan 4 bytes.



[1] Contiene caracteres para casi todos los idiomas modernos y un gran número de símbolos

En el siguiente gráfico, la primera línea de números representa la ubicación de un determinado carácter en el set de caracteres codificados Unicode. En las otras líneas, se muestra el valor de los bytes (en hexadecimal) utilizados para representar dicho carácter en una codificación de caracteres específica.

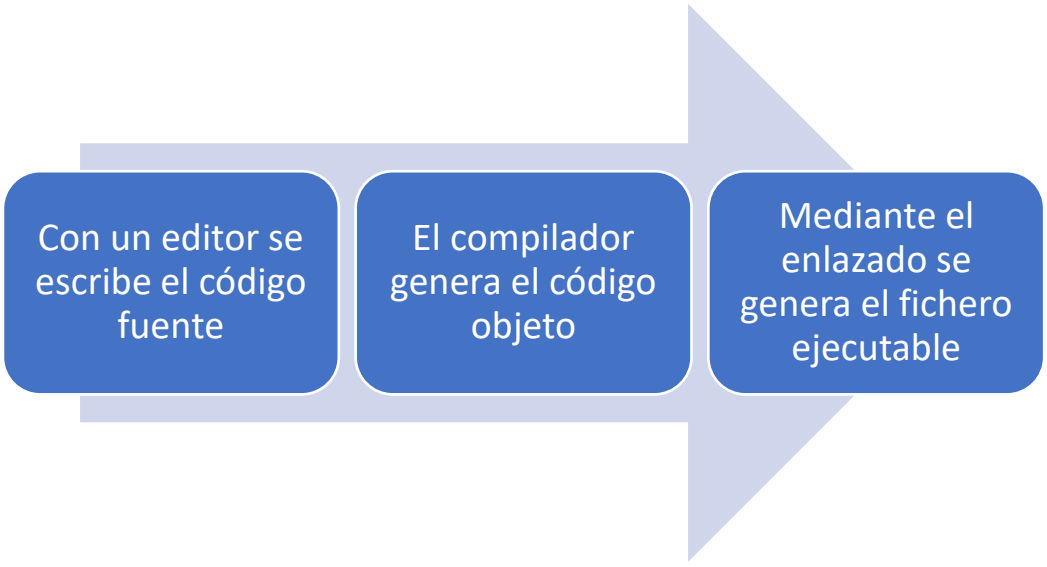
	B	𝔅
Punto de código	U+0042	U+0556
UTF-8	41	D5 96
UTF-16	00 41	05 56
UTF-32	00 00 00 41	00 00 05 56

Palabra

Es el número de bits que maneja una máquina de forma conjunta, es decir, los registros que hay en el procesador tienen capacidad para almacenar una palabra. El tamaño de la palabra depende del ordenador. Actualmente los tamaños más empleados son 32 (x86) y 64 bits (x64).

Creación de un programa ejecutable

Una vez tenemos escrito nuestro programa deberemos compilarlo y generar el programa ejecutable. El proceso se muestra en la figura. A partir del código fuente (nuestro programa) se obtiene un archivo ejecutable.

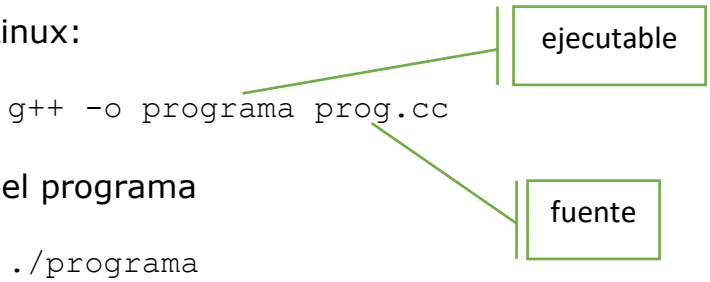


Una vez que el programa está compilado y enlazado y se ha creado el fichero ejecutable, se puede ejecutar escribiendo el nombre del fichero ejecutable.

En C++ lo hacemos de la siguiente manera:

- Con un editor de textos (kate, gedit, sublime, etc.) se escribe el programa. Es el código fuente. Supongamos que lo llamamos `prog.cc`
- Se compila llamando al compilador. Usaremos el g++ en sistema operativo Linux:

```
g++ -o programa prog.cc
```



- Se ejecuta el programa

```
./programa
```

Es importante destacar que solo se generará el fichero ejecutable si el fuente no tiene errores sintácticos. En caso contrario el compilador mostrará los errores para que puedan corregirse.

Características deseables de un programa

Toda persona que se dedica a la programación debería tener en mente una serie de pautas a seguir a la hora de construir un algoritmo.

Corrección

Un algoritmo siempre debe cumplir su especificación de manera rigurosa. Algunas de las causas que a menudo interfieren en esta labor son:

- La complejidad del propio algoritmo.
- Mala comprensión del problema por parte del programador.
- Descuidos a la hora de escribir el código por parte del programador.
- Una mala especificación del problema original.

Un buen criterio a la hora de escribir un algoritmo para resolver un problema consiste en empezar con una versión lo más sencilla posible del mismo, pero que sabemos a ciencia cierta que cumple la especificación. A partir de este momento nos podemos dedicar a intentar optimizar dicho algoritmo

para intentar que sea lo más eficiente posible, siempre cuidando de que se cumpla su especificación.

Claridad

El texto del algoritmo debe estar escrito de una manera clara y sencilla ya que un algoritmo escrito de manera clara es de gran ayuda, tanto para el propio autor, como para otras personas que lo puedan leer posteriormente para corregir errores o para modificarlo.

Para conseguirlo podemos seguir estas normas:

- Elegir nombres adecuados para las variables, subprogramas que construyamos, etc.
- Hacer un uso adecuado de los comentarios.
- Hacer un uso adecuado de los distintos elementos de composición del texto tales como:
    - Líneas en blanco.
    - Espacios en blanco.
    - Indentado o `*sangrado*' de las líneas.