Concepto y características generales, ventajas para el tratamiento de la información

1. Introducción.

Una definición poco estricta de lo que es un ordenador podría ser que "es una máquina electrónica que recibe y procesa datos para convertirlos en información útil".

> Uno de los componentes básicos en un sistema informático son los **datos** que se puedan introducir en él y cómo hace este sistema para almacenarlos y usarlos posteriormente o mostrarlos de nuevo.

Por lo tanto, una de las tareas básicas que hacen los ordenadores es almacenar la información que les proporcionamos para poder ser procesada posteriormente. Esta información puede ser de muchos tipos diferentes (texto, imágenes, vídeos, música...) pero lo realmente importante será **"cómo la almacena el ordenador"** para poder tratarla posteriormente de manera eficiente y generar más información.

1.1. Los datos

Los datos son representaciones de aspectos del mundo real, y se suelen recoger para hacer cálculos, mostrarlos, organizarlos, etc., con el objetivo de que, posteriormente alguien pueda hacer algo: tomar decisiones, generar nuevos datos ...

Si no se es muy estricto, se podría decir, que en un sistema informático cualquiera, las únicas tareas que se desarrollan, consisten en almacenar datos para procesarlos por medio de un programa que:

- o bien aportará algún tipo de información.
- o bien los utilizará para generar otros datos.

1.1.1. Características de los datos

Entre las características interesantes sobre los datos destacan sobre todo tres aspectos:

- A quién van dirigidos
- La posibilidad de reutilizarlos
- Que se puedan compartir

- **Destinatario de datos**

Si se intenta ser un poco más práctico, se verá que realmente, los datos tendrán una forma u otra en función del **destinatario** al que vayan dirigidos:

- Datos destinados a los **humanos**: generalmente los datos destinados a los humanos requerirán que tengan alguna estructura concreta, con unos formatos determinados, con textos decorados de alguna manera. Aparecerán títulos, caracteres en negrita, etc.; no suele ser necesario conocer el significado que tienen los datos, ya que la interpretación se deja al lector.
- Datos destinados a los **programas**: los programas generalmente no necesitan que los datos den la información sobre cómo se deben representar, basta con que sean fácilmente identificables, que quede claro de qué tipo son y que haya alguna manera de determinar qué significan para poderlos tratar automáticamente.

- **Reutilización de los datos**

Muy a menudo los datos se querrán reutilizar para poder realizar tareas diferentes. Un error corriente suele ser almacenarlos específicamente para hacer una tarea concreta, ya que esto puede provocar, que posteriormente, sea mucho más complicado usarlos para hacer otras tareas.

Por lo tanto, es básico disponer de un sistema de almacenamiento que permita conseguir que los datos puedan ser reutilizados fácilmente y si puede ser, que sean reutilizados tanto para las personas como para los programas.

- **Compartición de los datos**

En el pasado, con los ordenadores centrales, la información se generaba y se procesaba en el mismo lugar. Pero la aparición de los ordenadores personales, la eclosión de las redes y, sobre todo, el éxito de Internet, ha creado toda una serie de problemáticas que hasta el momento no existían: los datos generados en un lugar ahora pueden ser consumidos en un lugar totalmente diferente, como:

- en sistemas operativos totalmente diferentes.
- en máquinas que pueden funcionar de maneras muy diversas.

Por tanto, en un sistema informático moderno se debe tener en cuenta esta posibilidad a la hora de almacenar datos. Existe la posibilidad de que estos datos sean compartidos; por tanto, hay que almacenarlos de alguna manera para que no haya problemas al usarlos en sistemas diferentes.

1.2. Almacenamiento de datos en ordenadores

Dada su arquitectura, los ordenadores almacenan la información en binario y, por tanto, toda la información que se puede almacenar siempre se representará en unos y ceros (1, 0). Esto hace, que para representar cualquier tipo de datos (imágenes, vídeos, texto . . .), haya que hacer algún proceso que convierta los datos a una representación en formato binario.

Tradicionalmente, en los ordenadores los datos se organizan de dos maneras:

- Datos de texto
- Datos binarios

1.2.1. Datos binarios

Almacenar los datos de forma binaria es la forma natural de almacenar datos en ordenadores. Estrictamente hablando, los datos binarios están en el formato que usa el ordenador, ya que sólo son una tira de bits uno tras otro. Por lo tanto, normalmente, un ordenador no tendrá que hacer ningún proceso especial para almacenar y leer datos binarios.

Los datos en formato binario tienen una serie de características que las hacen ideales para los ordenadores:

- Generalmente están optimizados para ocupar el **espacio** necesario
- Los ordenadores las **leen fácilmente**.
- Pueden tener **estructura**.
- Es relativamente fácil añadir **metadatos**.

Si un programa quiere usar los datos binarios, normalmente sólo necesitará conocer el tamaño en bits y, sobre todo, conocer **de qué manera se ha almacenado** la información. Para almacenar el número 150 hace falta convertir este valor decimal a su representación en binario y almacenarlo. Es trivial comprobar que puede ser almacenado en un solo byte

(8 bits) como se puede ver en la Tabla 1.1

Tabla1. 1. Representación del número 150 en binario

valor decimal	valor binario
150	10010110

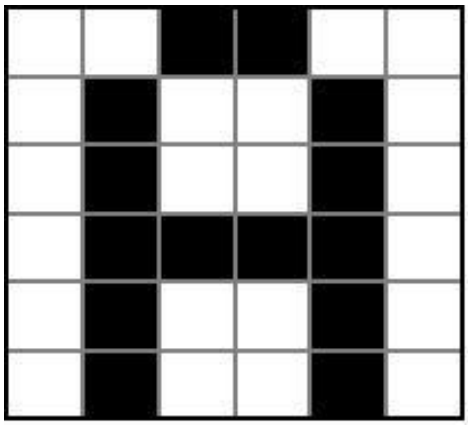
Una ventaja añadida de la representación de números en binario, es que ya están disponibles inmediatamente para hacer cálculos numéricos, ya que realmente se trata de números. No habrá que hacer ninguna transformación para poder usar este número en cualquier cálculo.

- **Metadatos**

Muy a menudo, los datos no se almacenan directamente tal como están, sino que se procesan para optimizarlos; se refiere a almacenar información sobre su contenido o a aplicar procedimientos de optimización. Estas optimizaciones son transparentes para el usuario final, que visualizará los datos normalmente.

Una de las maneras más sencillas de representar una imagen en un ordenador consiste en representar cada uno de los puntos de color que la forman. O sea, que sólo hay que decir de qué color será cada uno de los puntos para poder almacenar la imagen en un archivo (Figura 1.1).

Figura 1.1. Representación de un gráfico en un ordenador



Podemos utilizar un método sencillo para representar la imagen anterior, como podría ser definir si cada uno de los puntos es de color blanco (0) o negro (1). La imagen podrá ser representada de esta manera:

001100
010010
010010
011110
010010
010010

En realidad, un ordenador no almacenará la información así, sino de manera lineal, ignorando los saltos de línea. Una representación más cercana a como lo haría realmente un ordenador sería esta:

00110001001001001001111001001001010

Representar la información de esta manera hace que las imágenes ocupen mucho espacio y por este motivo normalmente se utilizan métodos para optimizar su almacenamiento.

Una forma de optimizar el espacio ocupado por la imagen, podría ser, darse cuenta de que hay varias repeticiones de los colores. De modo que se podría intentar aprovechar esta característica para conseguir un archivo binario más pequeño.
Se podría hacer, que en vez de especificar los puntos uno por uno, si hay una repetición, se pudiera especificar el número de veces que se repite el color. De este modo un punto blanco aislado se representará normalmente, pero si se encuentran cuatro puntos blancos, en vez de almacenar 0000, se puede representar con 40(4 blancos)
El resultado de aplicar este procedimiento a la misma imagen nos dará:

2021301201201201204120120120120 10

Este procedimiento tiene la ventaja añadida de que con el nuevo sistema los datos ocupan un 10% menos de espacio (33 caracteres) que antes (36 caracteres).
A pesar de que con el nuevo sistema no se almacenan todos los puntos, un programa puede conseguir fácilmente representarlos en pantalla siguiendo las especificaciones.

Se puede ver, que en la representación binaria hay toda una serie de valores que estrictamente no son datos de la imagen (los números 2, 3 y 4), sino que son datos que hacen referencia a la forma en que se han almacenado los datos. Estos datos se denominan " **metadatos** ".

Los **metadatos** son datos sobre los datos.

El uso de metadatos optimiza el almacenamiento de información, pero a la vez hace que la compartición de la información contenida en el fichero sea mucho más compleja. Pero eso sí, es necesario que el programa que quiera recuperar la información, conozca el procedimiento que se ha utilizado, o no obtendrá los datos correctos.

- **Datos estructurados**

Los datos, en la forma en que los generamos los humanos, no están en un formato que facilite el tratamiento automático por parte de un ordenador. Por este motivo, a menudo, los datos que deben ser procesadas por los ordenadores, se convierten en algún formato que sea más idóneo para su tratamiento. Lo más corriente es tratar los datos para que tengan algún tipo de estructura.

Los tipos de datos estructurados son agrupaciones de otros tipos de datos (normalmente tipo más sencillos).

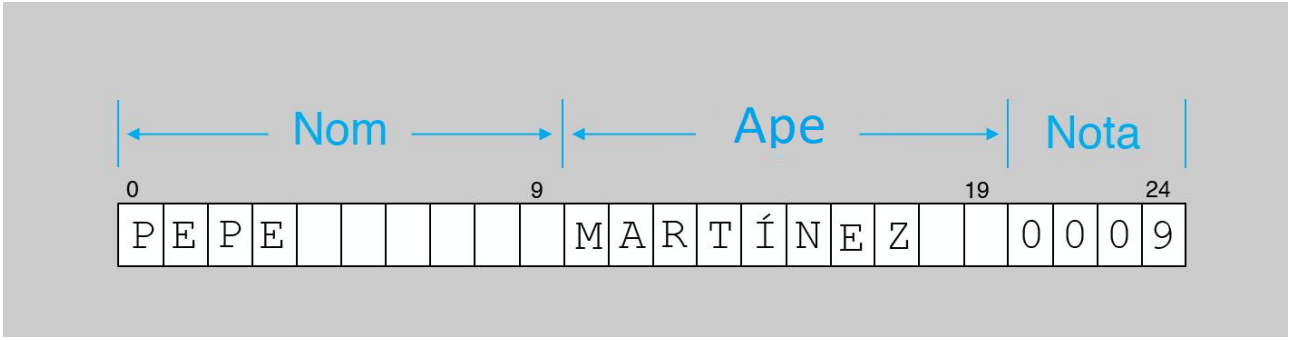
```
1  struct alumno {
2          char nombre [10];
3          char apellido [10];
4          int nota;
5  }
```

La manera más corriente de estructurar datos binarios, suele ser tenerlos agrupados en registros que contienen la información repetitiva de un dato en concreto. Es habitual que los lenguajes de programación tengan alguna manera de definir datos estructurados. Por ejemplo, para estas tareas, **el lenguaje C** utiliza *struct*.

Si se puede acceder a cada uno de los registros del fichero, se puede acceder de nuevo a los datos de un alumno, podemos identificar rápidamente cada parte de los datos: el nombre, apellido o la nota; además sabemos si los datos deben ser interpretados como números o como texto.

El ejemplo de la figura 1.2 muestra que se puede identificar a qué dato corresponde cada uno de los caracteres. Los diez primeros son el nombre, los 10 siguientes son el apellido y los cuatro siguientes son el número entero (32 bits).

Figura 1.2. Representación de datos en una estructura utilizando caracteres



Generalmente, estos datos estructurados se almacenan en forma de listas o conjuntos de registros, por lo que el desarrollador del programa podrá acceder a los datos de todos los alumnos simplemente recorriendo los diferentes registros uno por uno.
Los datos estructurados facilitarán que las aplicaciones los puedan tratar de manera automática.

- **Problemas**

Los datos binarios también tienen problemas asociados a la hora de ser compartidos, ya que, en el mundo moderno, los datos se deben compartir en máquinas en las que no han sido generados, que pueden tener Sistemas operativos diferentes, pueden ser máquinas totalmente diferentes, etc. Los datos binarios optimizados para el SISTEMA en que se generan no siempre serán bien entendidos por los otros SISTEMAS

Estructura de los datos

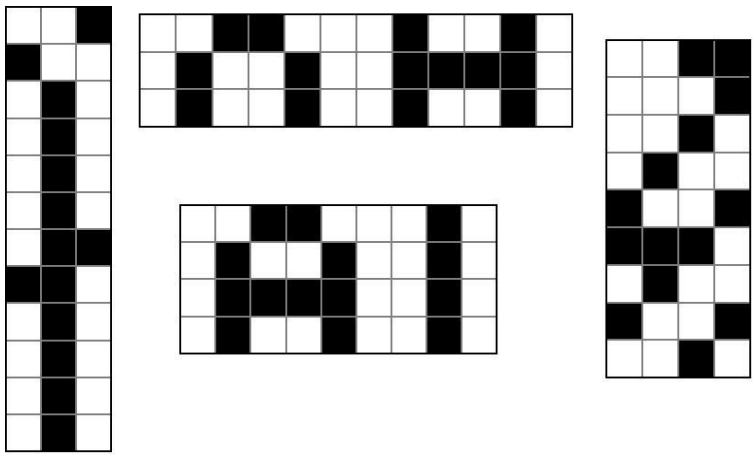
Uno de los problemas de dar estructura a los datos, es que esta estructura sólo la

entenderán los programas que tengan información sobre la estructura. Al definir cuáles son los datos que utilizará el programa, se define qué tamaño tendrá cada campo y de qué manera se guardarán los datos dentro del archivo binario.

Si tomamos el ejemplo que hemos visto en la figura 1.1, para que un programa pueda representar la imagen de manera correcta, es necesario que tenga suficiente información para hacerlo:

- Primero que sepa que lo que se ha representado es una imagen.
- También debe saber que se va a guardar cada punto de color con un solo carácter.
- Es básico que conozca la equivalencia de colores que hemos hecho: 0 es blanco, 1 es negro.
- Y necesita saber que la imagen es de 6 caracteres de largo por 6 caracteres de ancho, o el resultado será muy diferente de lo que era inicialmente (figura 1.3)
- Si se ha representado la imagen utilizando el sistema optimizado, debe conocer que los valores numéricos superiores a 1 indican que este valor no es un color, sino que son el número de repeticiones del color siguiente.

Figura 1.3. Intentos de representar el gráfico sin conocer las dimensiones



Todas estas cosas son las que hay que conocer para poder usar los datos de un ejemplo sencillo; por tanto, puede imaginarse qué pasaría con un ejemplo más complejo.

Forma de lectura del procesador

Del mismo modo que en los lenguajes humanos hay idiomas que se escriben de izquierda a derecha y de otros de derecha a izquierda, todos los procesadores no almacenan la

información de la misma manera (técnicamente se hace referencia en la orden de lectura en las direcciones de memoria).

Hay dos grandes SISTEMAS para almacenar la información en ordenadores:

- Big endian: los datos se escriben en el orden en que se crean. Así, para escribir hola en el ordenador se almacenaría h, o, l, a. Este sistema es el que utilizan los procesadores de Motorola.
- Little endian: los datos se guardan de menos relevante a más relevante: a, l, o, h. Este sistema es el que utilizan los procesadores de Intel.

Lo más habituales es que los ordenadores sólo utilicen uno de los dos SISTEMAS, aunque los hay que pueden funcionar con ambos indistintamente (ARM, PowerPC, PA-RISC...).

Esto no es importante cuando los datos se pasan entre ordenadores que funcionan con el mismo tipo, pero es un aspecto vital que hay que tener en cuenta, si los ordenadores que se pasan la información son de tipos diferentes, ya que los datos binarios pasados de un sistema al otro pueden ser totalmente mal interpretados por culpa de que se almacenan internamente de manera diferente.

Lectura para humanos

Un problema distinto es que los datos en formato binario están pensados para ser leídos por máquinas (figura 1.4) pero no por humanos, por lo que son ideales para ser almacenadas en máquinas; van bien para la comunicación de información entre máquinas, pero en cambio para que un humano las pueda utilizar deberá tener un programa específico para leerlas.

Figura 1. 4. El formato binario no está pensado para ser leído por humanos

Formatos binarios estándares

Se han definido algunos estándares de ficheros binarios, permitiendo que cualquiera pueda hacer programas que puedan interpretar estos archivos.

Esto es lo que ocurre, por ejemplo, con los archivos de imágenes JPG, PNG, GIF…, que pueden ser leídos por diferentes programas ya que su especificación es pública.

- **JPG** - Estándar ISO/IEC 10918
- **GIF** - Especificación del W3C GIF89a
- **PNG** - Estándar ISO/IEC 15948

Y, además, no sirve cualquier programa, sino que es necesario que el programa entienda la estructura de los datos que contiene el archivo. Por ejemplo, los datos generados por Microsoft Word no pueden ser abiertos con el programa de dibujo Gimp porque no está preparado para entenderlos.

Si, quien ha desarrollado el programa, no ha hecho pública de qué manera se guardan los datos binarios que se generan, será muy difícil compartir datos con otros usuarios; obliga a que todos los usuarios dispongan del mismo programa.

1.2.2. Datos de texto

Para solucionar el problema de que “haya que recurrir a programas específicos para recuperar los datos que hay en un fichero”, una posibilidad es hacer lo más obvio, hacer lo mismo que han hecho los humanos durante siglos. Los humanos al escribir ya están utilizando una codificación y, por tanto, si se utiliza para todos los datos la misma codificación, tendremos los datos en un formato fácil de usar y entender que no tendrá problemas para ser leído por los programas.

En archivos binarios el componente de información más pequeño es el bit, en ficheros de texto el componente más pequeño es el carácter.

Los archivos de texto almacenan la información letra por letra, de una manera similar a como lo haría un humano al escribir. Esto hace que se esté generando una información que se podrá leer de la misma manera que se lee un documento de papel.

Para un ordenador no hay mucha diferencia a la hora de almacenar los archivos de texto o archivos binarios, ya que los archivos de texto también son tiras de bits. La diferencia es que esta vez los bits están agrupados de una manera estándar y conocida: un código de caracteres.

- **Códigos de caracteres**

Representar los datos en un ordenador en forma de texto, implica que, para poder representar una palabra cualquiera en el ordenador, previamente deberá ser codificada para poderla representar en binario (recordemos que los ordenadores sólo pueden representar datos en binario). Esta codificación, suele consistir en determinar una cantidad de bits predefinida para marcar un carácter y posteriormente se asocia un valor numérico a cada uno de los caracteres.

> Obsérvese que para un ordenador el espacio en blanco es un carácter más.

La equivalencia entre los caracteres y sus valores numéricos, no se puede hacer de manera aleatoria, ya que se estaría creando el mismo problema que hay con los datos binarios. Si se quiere conseguir que los datos se puedan leer en diferentes sistemas, hay que seguir algún tipo de norma conocida por todos. Para esto aparecieron los **estándares de codificación de caracteres**.

Problema de la codificación de caracteres

Por ejemplo, si tuviéramos un idioma que sólo tuviera 5 caracteres (las vocales A, E, I, O, U), se podrían codificar todos los caracteres de este idioma utilizando sólo 3 bits.
Por tanto, una posible codificación de las letras del idioma podría ser la de la Tabla 1.2.

Tabla 1.2. Posible codificación de letras

carácter	valor decimal	valor binario
A	0	000
E	1	001
I	2	010
O	3	011
U	4	100
espacio	5	101

Esto nos permitiría codificar la frase "**AI AI AI**" de esta manera:

000010101000010101000010

Pero ante el mismo problema otra persona podría elegir una combinación diferente, como la de la Tabla 1.3.

carácter	valor decimal	valor binario
espacio	0	000
A	1	001
E	2	010
I	3	011
O	4	100
U	5	101

Tabla 1. 3. Alternativa diferente de codificación

Esto provocaría que al comunicar la frase "**AI AI AI**" generada con el primer sistema, en el segundo sistema se descodifica:

000 -> Espacio 010 -> E 101 -> U

Y interpretaría que el mensaje es "**EU EU EU**".

El procedimiento de tener una tabla con los valores numéricos asociados y simplemente hacer la conversión, es el procedimiento más habitual, pero también se dan casos en los que la codificación tiene que cumplir algún tipo de reglas o restricciones a la hora de hacer la conversión.

ASCII

Uno de los primeros estándares que fue adoptado mayoritariamente es **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange), que se puede ver en la Tabla 1.4. ASCII codifica cada uno de los caracteres con siete bits y define qué valor numérico se corresponde con cada uno de los caracteres de la lengua inglesa.

Tabla1. 4. Caracteres imprimibles del ASCII

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

La codificación que se hace en ASCII, es relativamente sencilla: simplemente se compara en la tabla cada uno de los caracteres del texto para codificar, y se obtiene el valor numérico correspondiente en binario.

Por ejemplo, para codificar la palabra “Hola” en un ordenador que esté funcionando con el código ASCII deberemos convertir cada uno de los caracteres en su equivalente numérico

Tabla1. 5. Conversión de caracteres en binario utilizando ASCII

carácter	decimal	binario
H	72	1001000
o	111	1101111
l	108	1101100
a	97	1100001

El primer problema que se encontró para el ASCII era, que sólo estaba pensado para el inglés y, por tanto, no se disponía de caracteres de uso corriente en otras lenguas: ç, à, á, ñ, etc. Por lo tanto, para poder expandirse a otras zonas se creó un ASCII expandido o

extendido, que incrementó el número de bits a 8, y gracias a este bit extra, se podían codificar los caracteres específicos de cada idioma que el inglés no tenía. De esta manera se permitía crear textos en otros idiomas que utilizaran el alfabeto latino.

Esto, hizo que aparecieran muchas variedades de ASCII, especializadas en un grupo de idiomas (ISO 8859-1, ISO 8859-2, etc.).
Pero como cada idioma utilizaba los valores nuevos para añadir sus caracteres propios, la información representada utilizando uno de estos "ASCII", no siempre se veía bien en otro de los "ASCII".
Además, ASCII y ASCII-extendido, sólo estaban pensados para idiomas que utilizaran el alfabeto latino y, por tanto, los idiomas no basados en el alfabeto latino tenían que recurrir a otras codificaciones.

Unicode

Unicode, es un intento de sustituir los códigos de caracteres existentes en uno genérico, que sirva para todas las lenguas, y por tanto supere todos los problemas de incompatibilidad que se producían en entornos multilingües y permita añadir los caracteres no latinos.
La idea básica de Unicode, es dar a cada uno de los símbolos un identificador único universal, de manera, que se puedan utilizar en el mismo documento idiomas diferentes, sin que ello comporte problemas de representación.
La adopción de Unicode resuelve de una vez todos los problemas de representación de caracteres en ficheros de texto.

> A pesar de sus ventajas, Unicode también recibió críticas de la comunidad anglófona, porque hacía que los textos acabaran ocupando más del doble que en ASCII.

Unicode define tres formas de codificación básicas UTF (**U**nicode **T**ransformation **F**ormat), que puede ver en la Tabla 1.6.

Tabla 1.6 Formas de codificación de Unicode

nombre	
U TF-8	Sistema basado en un byte con algunos símbolos de longitud variable
U TF-16	Sistema de longitud variable basada en dos bytes
U TF-32	Sistema de longitud fija que utiliza 32 bits para cada carácter

Unicode ha sido adoptado de manera general por la mayoría de sistemas operativos modernos. Actualmente casi todos los sistemas operativos utilizan alguna variedad de Unicode (el Linux suele utilizar UTF-8 y el Windows adapta UTF-16).

- **Compartición de información**

Gracias al uso de estándares de códigos de caracteres, la información en forma de texto es más fácilmente compartida que la información binaria, ya que los códigos de caracteres que utilizan los sistemas para representar el texto, son conocidos, y pueden ser implementados libremente.

Por lo tanto, almacenar los datos en formato de texto aporta dos grandes ventajas:

- Los pueden usar una gran cantidad de programas que ya existen (editores de texto, navegadores, etc.).
- Pueden ser leídos por humanos.
- Con uno de los programas más simples que hay, un editor de texto, se puede crear un documento que se podrá compartir con cualquier persona que entienda el idioma en que ha sido escrito. Y como todos los sistemas operativos incorporan de serie programas capaces de compartir archivos, si se envía el archivo a alguien, éste no tendrá ningún problema para interpretar los datos cuando las reciba.

Problemas

Generar información en archivos de texto también tiene algunos problemas:

- Ocupan más espacio en disco que los binarios.
- Hay múltiples códigos de caracteres diferentes.
- La forma en que los tratan los diferentes sistemas operativos.
- Falta de estructuración de los datos.

Pero a pesar de los problemas, estos son mucho menos importantes que los que tenemos para compartir ficheros binarios. Por lo tanto, **los archivos de texto, son la forma más sencilla de asegurarnos de que podemos compartir la información que contienen con otras personas**.

- **Espacio en el disco**

Uno de los problemas que tienen los archivos de texto es que la información ocupa mucho más espacio del que ocuparía si se almacenara en formato binario. Como podemos ver en la tabla (Tabla 1.7), si queremos almacenar el número **150** en un ordenador el resultado será diferente en función del formato elegido.

Tabla1. 7. Diferencia entre almacenar en formato de texto y binario

formato	representación interna
formato binario	10010110
formato textual	00110001 00110101 00110000

Para almacenar el número en formato **binario** simplemente se convierte en binario y se podrá almacenar en un byte (8 bits); si se quiere almacenar en formato de **texto,** utilizando ISO-8869-1, se deberán guardar por separado cada uno de los tres caracteres (1, 5 y 0). Con el segundo sistema serán necesarios 3 bytes (24 bits): el triple de espacio!

Pero, además, si la información se guarda en formato binario, necesario para hacer algún cálculo, ya se puede usar de inmediato, ya que se almacena realmente el número; si tenemos el número en formato de texto, tendremos que convertir a su equivalente numérico antes de poder hacer cualquier operación matemática.

Múltiples códigos de caracteres

Aunque Unicode intenta que este problema desaparezca, algunos sistemas operativos todavía usan múltiples códigos de caracteres, y los usuarios tienen información antigua guardada en códigos de caracteres antiguos. Por lo tanto, la compartición de datos aún provoca problemas, que pueden ser desde la simple representación incorrecta de algún

carácter, hasta corromper el texto o hacer que la lectura de la información sea imposible.

Figura 1.5. Problema de codificación incorrecta de caracteres


QuÃ¨ Ã©s l'IOC?

dijous, 10 de febrer de 2011 09:29

La raÃ³ de ser de l'Institut Obert de Catalunya es troba plenament vinculada als objectius del Departament d'Ensenyament

En primer lloc, l'extensiÃ³ de l'educaciÃ³, facilitant que pugui arribar al mÃ xim de persones superant les limitacions de l'espai i del temps.

En segon lloc, una educaciÃ³ que ha de posar els interessos, les necessitats i les possibilitats de progrÃ©s de les persones en el centre d'un model educatiu flexible i adaptable.

En tercer lloc, una proposta educativa que, en colÂ·laboraciÃ³ amb les altres, impulsa la formaciÃ³ al llarg de la vida.

I, finalment, el treball compartit per presentar una oferta educativa de qualitat i amb voluntat de millora, que vagi incorporant les innovacions en les tecnologies de la informaciÃ³ i de la comunicaciÃ³ amb l'objectiu de donar un millor servei educatiu.

Si el objetivo es “hacer que los datos se puedan reutilizar tanto para programas como para personas”, los caracteres erróneos se deben evitar. Como se ve en la Figura 1.5, los problemas en algunos casos pueden ser poco importantes si se quiere que una persona entienda lo que pone, pero pueden ser un problema muy grande para un programa, ya que su capacidad de interpretación es muy inferior.

La adopción de Unicode en la mayoría de sistemas operativos, está haciendo que este problema se esté reduciendo, y el hecho de que la cantidad de codificaciones de caracteres sea muy inferior a la cantidad de códigos binarios, hace que los datos en formato de texto se consideren fácilmente compartibles.

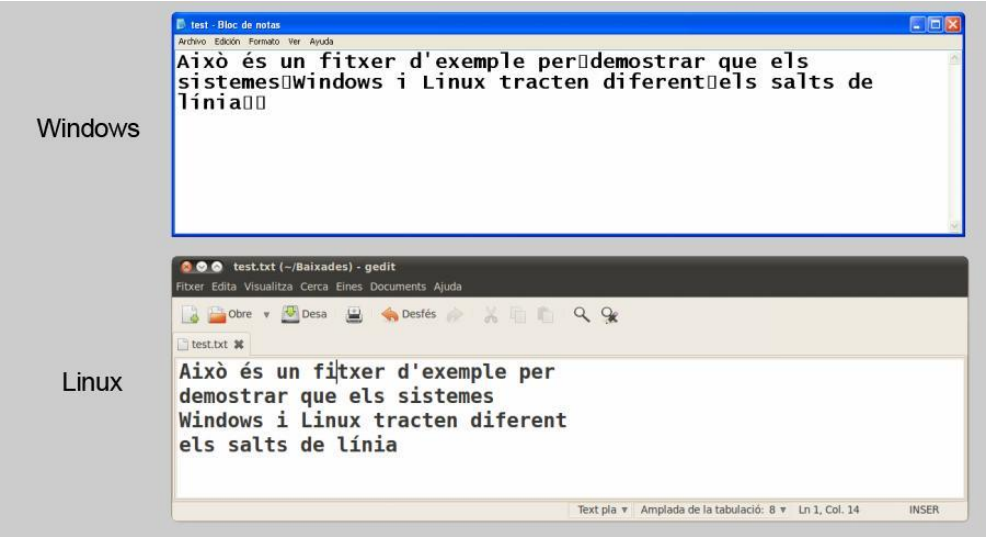
Si alguien hubiera almacenado información durante los años setenta (aunque se tuviera la capacidad de leer el soporte en el que se guardaron), difícilmente se podría recuperar algo de los datos binarios que se encontraran, ya que los programas que las generaron ya no existen o no funcionan con los sistemas operativos modernos; en cambio es posible que los datos almacenados en formato de texto sí se pudieran recuperar.

Representación de caracteres no textuales

Otro problema que suele haber en la lectura de datos de texto cuando se hace en diferentes sistemas operativos, suele estar relacionada con “cómo se hace el tratamiento de los caracteres no textuales”.

El ejemplo más conocido es “el diferente tratamiento de los saltos de línea” que hacen los sistemas Windows y las diferentes variedades de Unix y Linux. Para representar los saltos de línea en el texto, los sistemas operativos utilizan alguno de los caracteres no imprimibles del código de caracteres y, por tanto, de forma "transparente al usuario", pueden representar el texto tal como se ha escrito; dos de los sistemas operativos más populares, Windows y Unix, lo hacen de manera diferente. Mientras que Unix utiliza un solo carácter para indicar el salto de línea, LF (**L**ine **F**eed), Windows en usa dos: CR (**C**arriage **R**eturn) y LF (**L**ine **F**eed). El resultado es, que al abrir un archivo en un sistema en Windows que se ha generado en un sistema Unix, los saltos de línea han desaparecido y en su lugar aparece un rectángulo que representa el carácter LF.

Figura 1.6 Diferencia del tratamiento en los saltos de línea entre Windows y Linux



Actualmente muchos programas ya detectan este problema y lo compensan automáticamente de forma transparente al usuario.

Lectura de datos automatizada

Los programas de ordenador todavía no son muy buenos interpretando los datos si son en texto narrativo y, por tanto, generalmente conviene que los datos que deberán ser tratadas por programas de ordenador estén definidos con algún tipo de **estructura** para facilitar dicho tratamiento.

Concepto y características generales, ventajas para el tratamiento de la información

Se han inventado sistemas para hacer que los datos de los ficheros de texto puedan ser estructurados. Uno de los formatos que se ha utilizado durante mucho tiempo para exportar datos estructurados (contenidos en bases de datos u hojas de cálculo) a texto, ha sido el formato CSV (**C**omma **S**eparated **V**alues); simplemente, se limita a separar cada uno de los registros de la estructura en líneas, y los campos se separan con comas. Además, para poder definir los tipos de datos, rodea de comillas los datos de texto, mientras que no se ponen comillas en los numéricas.

```
"Manuel", "Perez", "Garcia", 8
"Pedro", "González", "Parada", 5
"María", "Pozos", "Calle", 7
```

CSV es una manera sencilla de guardar datos estructurados en formato de texto que permite a un programa identificar los diferentes datos que contiene cada registro y además interpretar de qué tipo son.

Además, una ventaja añadida de CSV es, que es relativamente fácil añadir más datos a un fichero que esté en formato CSV, ya que sólo hay que editar el texto y respetar las reglas de separar los datos con comas y hacer un salto de línea para cada registro.
Por lo tanto, un programa puede deducir fácilmente a partir del ejemplo anterior que los datos son como se ve en la Tabla 1.8.

Tabla1. 8. Interpretación del significado del archivo CSV

dato	resultado
Manuel	Dato de texto porque está entre comillas
Perez	Dato de texto porque está entre comillas
Garcia	Dato de texto porque está entre comillas

Pero los sistemas que estructuran datos en ficheros de texto, también tienen problemas; si necesitamos añadir más datos en cada registro es casi seguro que obligará a hacer cambios en el programa que los tratará. El programa necesita saber qué datos hay en cada una de las columnas para poder trabajar y, por tanto, si modificamos las columnas puede malinterpretar los datos.

Problema de añadir datos en un CSV

Por ejemplo, si añadimos un dato nuevo a la persona en del archivo anterior:

```
"Sr.", "Manuel", "Perez", "Garcia", 8
"Sr.", "Pedro", "González", "Parada", 5
"Sra", "María", "Pozos", "Calle", 7
```

Al utilizar el mismo programa que antes, éste puede pensar que los nombres de las personas son "Sr" y "Sra" y que las notas son "Garcia", "Perez", etc.

Pero añadir datos no es el único problema que tenemos, ya que representar los datos de esta manera arruina la lectura de la gente que no use los programas específicos para ellos. ¿Quién puede saber que el número informa sobre una nota y no es otra cosa? A cualquiera de vosotros se os puede ocurrir una manera sencilla de evitar este problema, que consiste en añadir una primera fila que indique información sobre cada uno de los datos que se representan a continuación. Desarrollar un programa que interprete estos datos y que se asegure que no se producen errores es bastante complejo.

```
"Nombre", "Apellido1", "Apellido2", "Nota"
"Manuel", "Perez", "Garcia", 8
"Pedro", "González", "Parada", 5
"María", "Pozos", "Calle", 7
```