TEMA 1. Informática básica

Introducción

El Tema 1 nos servirá para dos fines. En la sección 1.1 Informática básica, realizaremos una breve presentación de la noción de "Informática". Veremos que la misma es muy amplia y que comprende muchos elementos distintos. Enunciaremos algunos de ellos, presentaremos los que se tratarán en secciones posteriores, y descartaremos otros por no pertenecer al ámbito de la asignatura en la que nos encontramos. También definiremos la noción de "Sistemas Informáticos", que da nombre a nuestra asignatura, y presentaremos una breve descripción de los contenidos de este curso.

En la parte restante del Tema 1 (secciones 1.2, 1.3, 1.4 y 1.5) nos centraremos ya en un tipo concreto de sistema informático, el ordenador, sobre el cual ilustraremos alguna de sus principales características. Hay que tener en cuenta que ésta no es una asignatura de "Estructura de Computadores", sino más bien de "Sistemas Operativos", y por lo tanto no entraremos en un nivel muy elevado de detalles. Sin embargo, sí que creemos necesario mencionar los "Fundamentos Estructurales y de Funcionamiento" (sección 1.2), donde nos centraremos en la codificación de la información en código binario o hexadecimal, destacando también los "Componentes básicos de un ordenador" (sección 1.3), "El Disco Duro" (sección 1.4) y "El sistema de arranque de un ordenador. Particiones y Volúmenes" (sección 1.5).

1.1 Informática básica

Podemos definir Informática, siguiendo el diccionario de la RAE, como el "conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de ordenadores". De una forma más sintetizada, podemos identificar la Informática con "información automática".

Por lo tanto, un "Sistema Informático" lo podemos entender como una solución completa a un problema concreto de tratamiento automatizado de información.

Esta solución, por lo general, contempla una parte física (el "hardware" o máquinas sobre las que se trabaja) y una parte "software" (formada por los programas o utilidades de que se hace uso en la máquina). Por último, los sistemas informáticos suelen

requerir por lo general de una interacción humana, que se suele identificar con la que demanda unos ciertos servicios.

Ejemplos de sistemas informáticos, por lo tanto, serían el usuario que en su ordenador personal trata de redactar un texto o leer el correo electrónico, alguien que en su teléfono móvil apunta un número en su agenda, o un operario de una fábrica que debe controlar ciertas máquinas a través de un programa informático; o podemos también pensar en sistemas mucho más sencillos como un programador de un termostato o de un electrodoméstico cualquiera.

Nuestra asignatura no pretende entrar en el diseño y definición de sistemas informáticos, puntos que se verán más adelante durante la titulación, sino como asignatura de iniciación a los sistemas informáticos, introducirnos en el conocimiento y uso de los elementos básicos en todo sistema.

Sólo a modo de ejemplo, podemos observar que el diseño completo de un sistema informático requiere la definición de elementos de muy diferente naturaleza, como pueden ser:

- Equipos: Ordenadores, periféricos, sistemas de comunicación (a este punto dedicaremos la sección 1.2).
- Procedimientos: Sistemas Operativos (a este punto dedicaremos el tema 3), aplicaciones de gestión, ofimática, sistemas de comunicación, documentación, seguridad (copias, acceso, protección).
- Personal: necesidades de personal, especialización, procedimientos de usuario, formación (inicial y continuada).
- Otros: estudio económico (coste inicial, mantenimiento, de personal, vida del sistema).

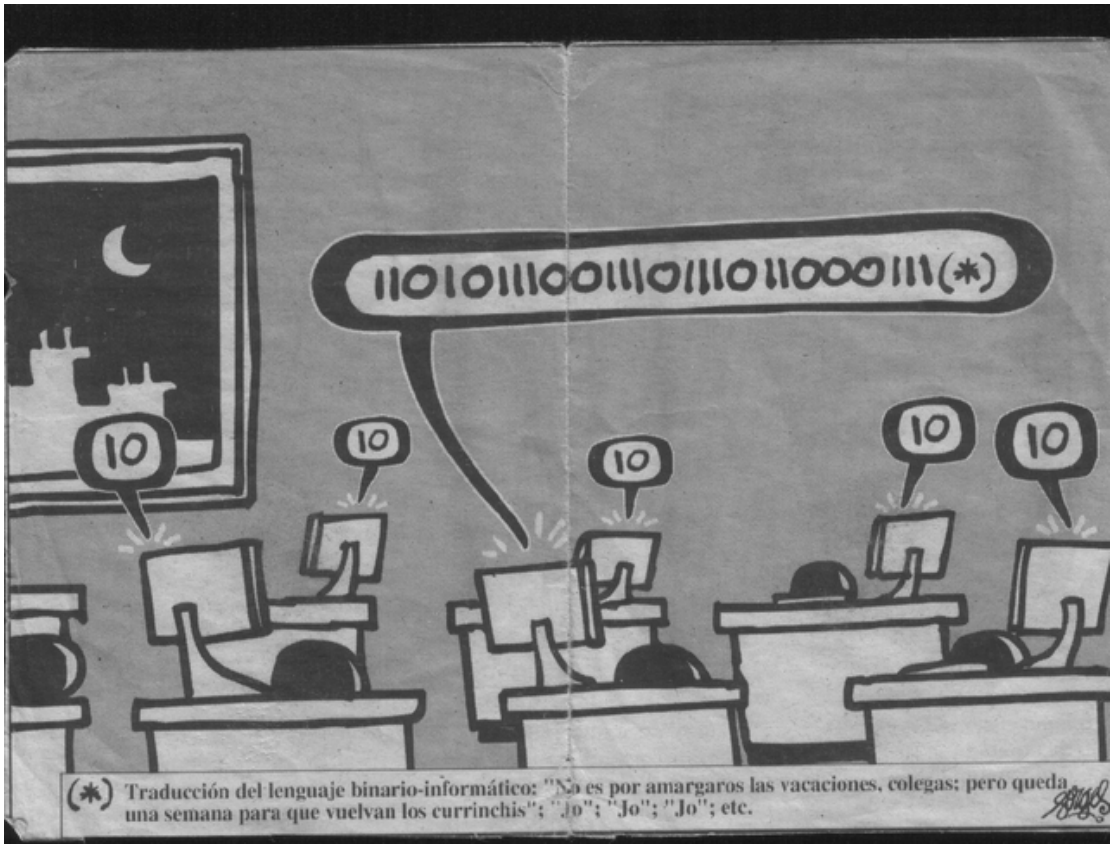
Entre estos distintos componentes se pueden encontrar cuestiones de hardware, de software, lenguajes de programación, bases de datos, sistemas de comunicación, análisis y diseño de sistemas, estrategias de formación…

La creación de sistemas complejos normalmente no se lleva a cabo por una única persona, sino por un equipo de personas, cada una de ellas especializada en una determinada área.

1.2 El ordenador. Fundamentos estructurales y de funcionamiento

A partir de la definición hecha de Informática como "información automática", nuestro primer objeto de interés va a ser conocer cómo la información es almacenada en un sistema informático. Para ello

explicaremos la diferencia entre señales analógicas y señales digitales. Posteriormente, presentaremos la estructura básica de un ordenador, con el fin de saber cómo la información almacenada en un dispositivo puede ser procesada.



1.2.1 Almacenamiento de información: señal digital.

Los ordenadores actuales se basan en una tecnología electrónica digital, en comparación a otros dispositivos que utilizan señal analógica.

Podemos identificar la idea de digital con "discreto": en un sistema digital las variables de entrada y salida son magnitudes (en general señales eléctricas) discretas o que se toman como tales, y se procesan como valores discretos. Es decir, sólo un número entero y concreto de valores son posibles.

Por el contrario, podemos identificar la idea de analógico con "continuo": en un sistema analógico sus variables de entrada y salida son magnitudes (en general, señales eléctricas) continuas y se procesan como valores continuos, es decir, pueden alcanzar cualquier valor dentro de un espectro determinado.

Un ejemplo de señal analógica sería la fotografía tradicional con película. La luz es captada e impresa sobre la película de forma continua, sin distinguir regiones o cuadros. Sin embargo, las cámaras fotográficas en la actualidad son en su mayoría digitales, ya que la señal se ha discretizado. Por ejemplo,

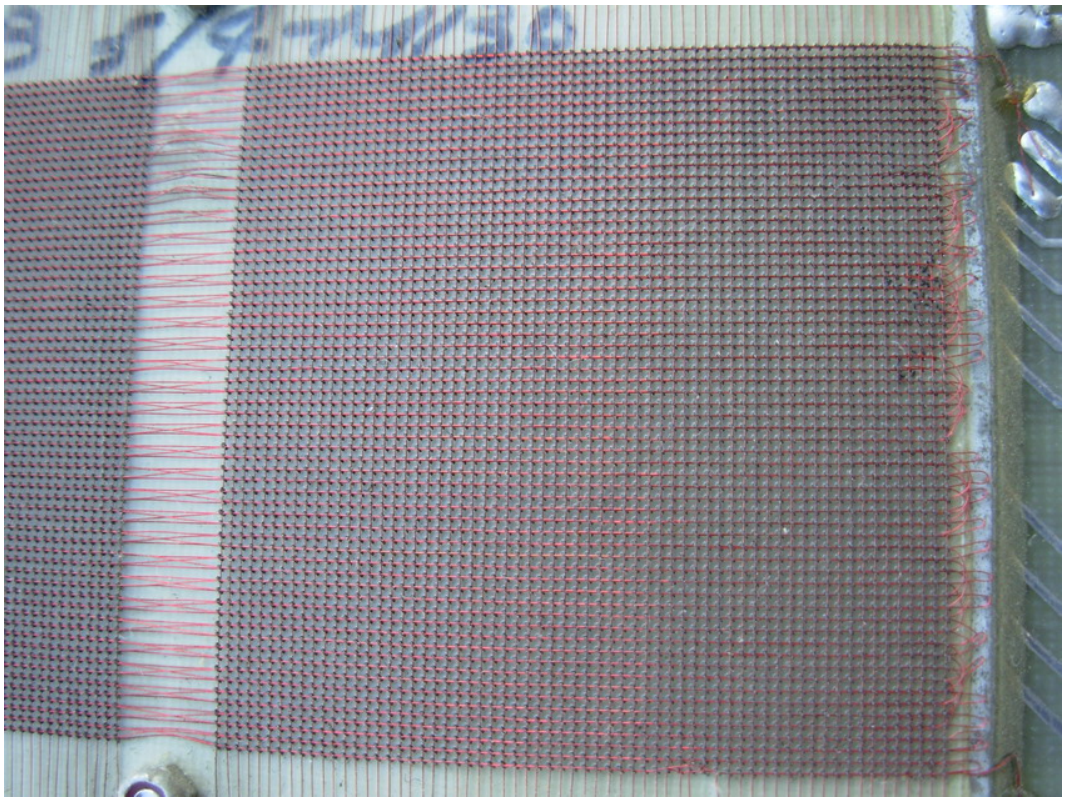
en una resolución de 14 megapíxeles la señal se ha discretizado en una matriz de 4592 * 3056 píxeles, con cada píxel capturando una intensidad de rojo, verde y azul (RGB) correspondiente. Si aumentáramos el tamaño de una foto cualquiera, veríamos como la misma está "discretizada" o "pixelada". La captura de la fotografía se basa en los impulsos electromagnéticos que produce el objeto fotografiado sobre el sensor de la cámara (que es el encargado de grabar la misma para cada píxel). Otro ejemplo similar sería la señal analógica (continua) que se graba en las cintas de audio o discos de vinilo, que en la actualidad se han convertido en discos compactos o "mp3" donde la señal de nuevo está "discretizada".

La informática por lo general se basa en la utilización de señales digitales. La tecnología actual permite construir con cierta facilidad todo tipo de dispositivos discretos que distingan entre dos estados diferentes (los cuales etiquetaremos como "0" y "1"). Así pues, nuestra tecnología digital gestiona entradas y salidas y las procesa admitiendo únicamente dos valores diferentes (aún así, el concepto genérico de digital se refiere a un número finito que puede ser más de dos). Desde el punto de vista teórico, el hecho de admitir más de dos estados en un sistema digital haría inmensamente más eficiente y potente dicho sistema (por ejemplo, los sistemas cuánticos se basan en el control de más de dos estados de cada uno de sus elementos, pero por el momento siguen siendo dispositivos teóricos en los que a nivel práctico, el controlo de varios estados resulta muy costoso), pero por el momento nos vemos limitados a trabajar con bits (binary digits) que sólo aceptan 2 estados (podemos pensar en ellos como "encendido" y "apagado").

Por este motivo toda la información que se almacena y maneja en los ordenadores es información codificada en formato binario. Podemos pensar en la información que tenemos almacenada en una memoria digital. Esa memoria está compuesta por un gran conjunto de elementos (podemos pensar en casilla o celdas) que sólo son capaces de distinguir dos estados diferentes. Cualquier información que queramos almacenar en ella habrá que codificarla de alguna manera a base de cadenas de dichos dos estados. Toda la información, números, texto, colores, sonido, imagen… hay que codificarla a una secuencia de dos estados distintos que identificamos con 0 y 1.

Llamaremos bit (binary digit) a la unidad básica (y mínima) de información en un sistema digital (y también en informática). Un bit puede representar un 0 ó un 1.

En la imagen inferior podemos observar una memoria de una computadora (del año 1980) en la cual el tamaño de los bits se puede comprobar a simple vista. La memoria contiene 4096 celdas o bits, y su tamaño es aproximadamente 4 * 4 cm.



Semejante fragmento de memoria nos serviría para almacenar 2 elevado a 4096 valores. Si bien puede parecer un número muy elevado, en un sistema con bytes (explicamos el significado de "byte" en el siguiente párrafo) de 8 bits nos permitiría almacenar únicamente 512 caracteres.

En general un bit es un fragmento de memoria tan pequeño (sólo permite almacenar dos valores) que cualquier sistema informático utilizará fragmentos de memoria un poco mayores, o agrupaciones de bits, que le permitan almacenar cantidades de información relevantes. A dichos fragmentos les llamaremos bytes.

Un byte es una secuencia de bits contiguos. Su tamaño depende del código de caracteres (o el tamaño de palabra) que esté definido en nuestro sistema.

Por lo general, se considera que un byte está formado por 8 bits, aunque en algunos sistemas la longitud de palabra (y por tanto el byte) tiene otro tamaño. Debido a esta asignación de un byte como 8 bits, en castellano se suele utilizar la palabra octeto para sustituir a la palabra anglosajona byte.

Siguiendo con esa asignación, un byte (de 8 bits) puede contener $2^8 = 256$ valores, y por tanto puede representar 256 informaciones diferentes, que serían las siguientes:

00000000	00000001	00000010	00000011	00000100	00000101	00000110	00000111
00001000	00001001	00001010	00001011	00001100	00001101	00001110	00001111
00010000	00010001	00010010	00010011	00010100	00010101	00100110	00010111
00011000	00011001	00001010	00011011	00011100	00011101	00011110	00011111
00100000	00100001	00100010	00100011	00100100	00100101	00100110	00100111

00101000	00101001	00101010	00101011	00101100	00101101	00101110	00101111
00110000	00110001	00110010	00110011	00110100	00110101	00100110	00110111
00111000	00111001	00101010	00111011	00111100	00111101	00111110	00111111
01000000	01000001	01000010	01000011	01000100	01000101	01000110	01000111
01001000	01001001	01001010	01001011	01001100	01001101	01001110	01001111
01010000	01010001	01010010	01010011	01010100	01010101	01100110	01010111
01011000	01011001	01001010	01011011	01011100	01011101	01011110	01011111
01100000	01100001	01100010	01100011	01100100	01100101	01100110	01100111
01101000	01101001	01101010	01101011	01101100	01101101	01101110	01101111
01110000	01110001	01110010	01110011	01110100	01110101	01100110	01110111
01111000	01111001	01101010	01111011	01111100	01111101	01111110	01111111
10000000	10000001	10000010	10000011	10000100	10000101	10000110	10000111
10001000	10001001	10001010	10001011	10001100	10001101	10001110	10001111
10010000	10010001	10010010	10010011	10010100	10010101	10100110	10010111
10011000	10011001	10001010	10011011	10011100	10011101	10011110	10011111
10100000	10100001	10100010	10100011	10100100	10100101	10100110	10100111
10101000	10101001	10101010	10101011	10101100	10101101	10101110	10101111
10110000	10110001	10110010	10110011	10110100	10110101	10100110	10110111
10111000	10111001	10101010	10111011	10111100	10111101	10111110	10111111
11000000	11000001	11000010	11000011	11000100	11000101	11000110	11000111
11001000	11001001	11001010	11001011	11001100	11001101	11001110	11001111
11010000	11010001	11010010	11010011	11010100	11010101	11100110	11010111
11011000	11011001	11001010	11011011	11011100	11011101	11011110	11011111
11100000	11100001	11100010	11100011	11100100	11100101	11100110	11100111
11101000	11101001	11101010	11101011	11101100	11101101	11101110	11101111
11110000	11110001	11110010	11110011	11110100	11110101	11100110	11110111
11111000	11111001	11101010	11111011	11111100	11111101	11111110	11111111

Lo que un conjunto de ceros y unos signifique depende de la manera en que nosotros hayamos decidido codificar nuestra información (por ejemplo, de la codificación de caracteres que utilice nuestro sistema, o de la tabla de caracteres vigente), pero es importante tener presente que la información guardada en un sistema digital (en una memoria) siempre lo estará, en última instancia, en el formato anterior.

1.2.2. Codificación de caracteres.

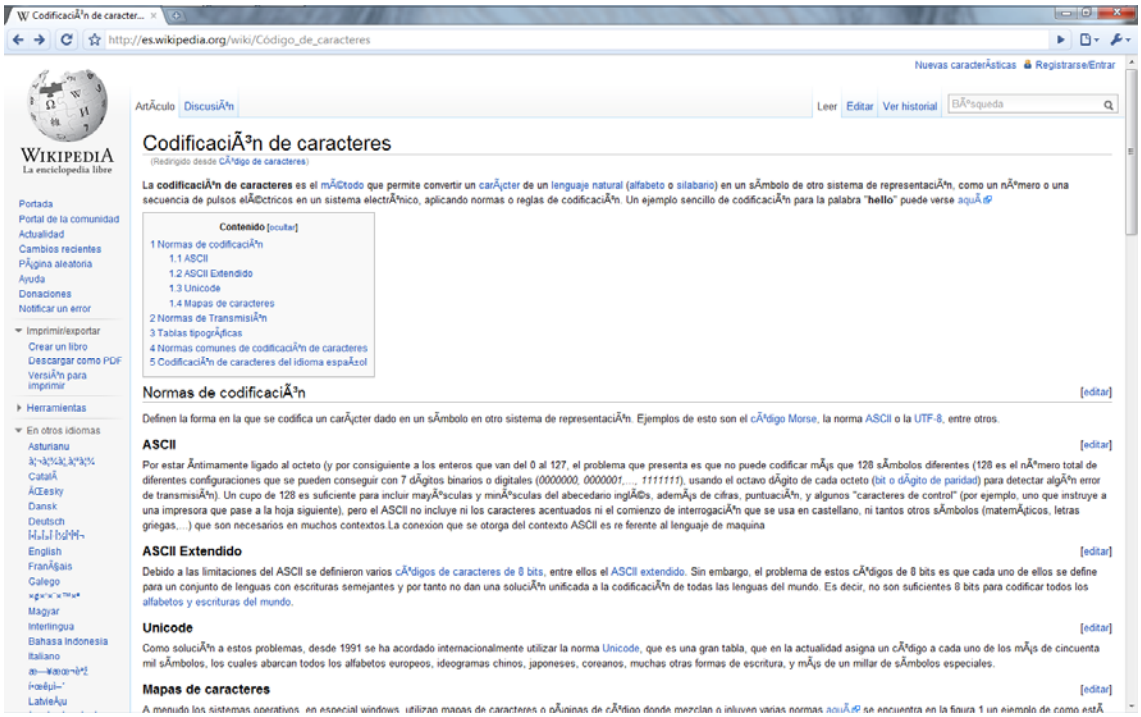
Generalmente, en el disco duro de un ordenador, la información se almacena en archivos. Un archivo es un conjunto de bits (y también de bytes) que tiene un nombre único, y que para el sistema operativo de nuestro ordenador se entiende como una unidad lógica. Es importante resaltar aquí la diferencia entre la representación del archivo en el disco duro, que ocupará un número determinado de bytes (o de unidades de memoria física) y la representación del mismo para el sistema operativo, para el cual el archivo es una unidad.

Estos archivos pueden ser de muy diversos tipos (imágenes, sonidos, vídeos, ejecutables, archivos de configuración, archivos propios del sistema operativo…). Lo que pueda significar un bit o un byte en un disco duro depende directamente del tipo de archivo en que nos encontremos. De este modo, un byte en un fichero de imagen puede hacer referencia al color de un píxel, mientras que ese mismo byte en una base de datos puede representar un número cualquiera, o en un fichero de un procesador de textos puede contener información sobre la tipografía utilizada (Arial, Times, Verdana…).

Existe un tipo de archivos particulares que se conocen como "ficheros de texto" que se caracterizan porque contienen exclusivamente información textual (caracteres) y no llevan información sobre el formato del mismo (tamaño de la fuente, tipo, color, márgenes…). Estos archivos son de amplio uso, ya que permiten guardar códigos fuente de programas, archivos de configuración de equipos, ficheros con grandes cantidades de datos…

Más adelante profundizaremos en el tratamiento de archivos de cualquier tipo en el disco duro. Por el momento, sólo nos vamos a interesar por cómo la información (en este caso texto sin formato, o texto plano) es almacenada en los citados archivos de texto.

Cuando escribimos el carácter "A" y lo guardamos en nuestro ordenador en un fichero de texto plano (por ejemplo, en el bloc de notas, pero no en "Microsoft Word", que le aportaría formato al mismo), "alguien" (nosotros, el programa en uso o el sistema operativo) habrá determinado un sistema de codificación para guardar "A" como un conjunto de ceros y unos, que representan la codificación del carácter en dicho sistema de codificación. Por tanto, es imprescindible que cuando leamos lo que hemos guardado lo hagamos con un programa que utilice el mismo sistema de codificación de caracteres que se usó al guardarlo. En caso contrario, podría suceder lo que observamos en la siguiente captura de pantalla de un navegador. ¿Qué ha sucedido con todos los caracteres acentuados o no estándar?



Antes de que entremos en más detalle sobre las codificaciones de caracteres es importante fijar una serie de conceptos que definimos a continuación. La diferencia entre ellos puede parecer sutil, pero es necesario comprenderla para poder entender los procesos de codificación y decodificación de caracteres que tienen lugar sobre ficheros de texto plano o sin formato (las siguientes definiciones no son estándar, y quizá en algunos textos las encuentres de forma distinta, pero es importante que distingas los tres conceptos que se presentan, ya que es bastante común confundirlos, haciendo la conversión de caracteres mucho más complicada de entender de lo que en realidad es).

- Repertorio de caracteres: es un conjunto de caracteres sin ninguna estructura. Lo que hacemos por medio de un repertorio de caracteres es definir los caracteres que van a ser importantes para nosotros en nuestra codificación. Sí que es relevante notar cómo un repertorio de caracteres puede depender de la región en que nos encontremos (no será lo mismo, posiblemente, un repertorio de caracteres para Asia que un repertorio de caracteres centroeuropeos o de Europa Occidental). En un sencillo ejemplo, podríamos considerar como un repertorio de caracteres el conjunto {a, e, i, o, u, A, E, I, O, U, á, é, í, ó, ú}.

- Tabla de caracteres: una tabla de caracteres puede entenderse como una aplicación que a cada carácter (de un repertorio de caracteres) le asigna un número entero (no negativo). El número entero suele ser conocido como "código de posición" (en la tabla de caracteres) o "número de código". El anterior repertorio de caracteres podría ser representado ahora por la siguiente tabla de caracteres: {a=1, e=2, i=3, o=4, u=5, A=6, E=7, I=8, O=9, U=10, á=11, é=12, í=13, ó=14, ú=15}, pero también por otras tablas de caracteres distintas (por ejemplo, en orden inverso, o sólo por números pares).

- Codificación de caracteres: una codificación de caracteres es un algoritmo o regla que permite convertir los códigos de posición de una tabla de caracteres en octetos (o secuencias de octetos). La codificación más sencilla es convertir cada código a su representación binaria {a = 00000001, b = 00000010, c = 00000011…}. Esto funciona bien para tablas de menos de 256 caracteres. Sin embargo, es muy fácil encontrar otras codificaciones diferentes; por ejemplo, invirtiendo el orden de los bits: {a = 10000000, b = 01000000, c = 11000000…}. Para tablas con más de 256 elementos los algoritmos son más complejos, ya que se hace necesario más de un octeto.

Los computadores de los años 80 utilizaban procesadores de 8 bits (un octeto), y eso hizo que la mayor parte de las codificaciones de caracteres se ajustaran a esas dimensiones (es decir, que cada palabra ocupe un octeto). Es importante notar que toda codificación que ocupe un octeto va a permitir representar un máximo de 256 caracteres. En este contexto, una de las tablas o normas de codificación que se hicieron más populares fue el ASCII (American Standard Code for Information Interexchange). Siendo un estándar de origen occidental (más concretamente norteamericano) sirve para codificar caracteres latinos (por supuesto, no contiene caracteres cirílicos, hebreos, árabes…).

Sus principales características serían las siguientes. Su repertorio de caracteres está compuesto sólo por 128 caracteres y símbolos de control. De éstos, 32 corresponden a caracteres de control, algunos de los cuales se usaban en la época para enviar señales a dispositivos periféricos tales como impresoras, y el resto se corresponden con caracteres estándar (mayúsculas, minúsculas, signos de puntuación…). Se puede encontrar la definición concreta de los operadores de control en http://es.wikipedia.org/wiki/ASCII.

La tabla de caracteres ASCII con sus códigos de posición sería la siguiente:

b7 → b6 → b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits	b4 ↓	b3 ↓	b2 ↓	b1 ↓	Row ↓ Column →	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	/
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

El repertorio de caracteres ASCII consta sólo de 128 caracteres. Si observamos que $2^7 = 128$, con 7 bits sería suficiente para poder codificar la tabla de caracteres ASCII. Sin embargo, internamente, y ya que la longitud de palabra del ordenador es el octeto, cada carácter se codifica con 8 bits. Para ello el proceso (o algoritmo) de codificación sería el siguiente. A cada carácter se le asignan los 7 bits correspondientes a la codificación en binario de su posición en la tabla de caracteres. Por ejemplo, al carácter "O", cuya posición en la tabla es 79 (puedes contar las columnas, o convertir 4 – 15 de hexadecimal a digital para conocerlo), le asignamos los 7 bits de su representación binaria ($79 = 1*2^0 + 1*2^1 + 1*2^2 + 1*2^3 + 0*2^4 + 0*2^5 + 1*2^6$), es decir, 1001111. Para determinar el bit de paridad, que colocaremos en primera posición, contamos el número de "1"s en la representación binaria de "O" (en este caso 5), y la convertimos en par (de ahí el nombre de bit de paridad), añadiendo un nuevo 1. Finalmente, la codificación del carácter "O" queda como "11001111". Si el carácter a representar fuese "P", cuya posición en la tabla es $80 = 0*2^0 + 0*2^1 + 0*2^2 + 0*2^3 + 1*2^4 + 0*2^5 + 1*2^6$ y cuyo bit de paridad será 0 (ya que la codificación anterior contiene dos "1" y ya es par), sería 01010000.

El repertorio (y la tabla) de caracteres ASCII se convirtió en un estándar de facto, pero pronto se vieron sus limitaciones, sobre todo por el corto número de caracteres que se pueden codificar en la misma (se puede observar, por ejemplo, que no se pueden encontrar ni vocales acentuadas "á, é, í…", ni la letra "ñ", ni la "u" con diéresis "ü", propios del castellano, ni muchos otros caracteres de otros idiomas).

Se hizo necesaria la definición de nuevos repertorios de caracteres, que utilizaran al menos toda la capacidad del octeto (256 valores) para codificar caracteres. Debido al papel preponderante que había adquirido ASCII en el mercado de los ordenadores, las tablas de caracteres que surgieron después coincidieron en incluir los caracteres "representables" ASCII (es decir, desde

el 33 hasta el 128) codificados de la misma forma que en la norma ASCII (así nos aseguramos que cualquier texto que sólo contenga caracteres de la norma ASCII se podrá guardar y leer en cualquier norma de codificación de la misma forma).

Uno de los intentos de aumentar la norma ASCII fue la conocida como ASCII extendida. Existen múltiples códigos ASCII extendidos. Su principal característica es que contienen los caracteres del 33 al 128 idénticos a los de ASCII (los caracteres de control en las posiciones 1 a 32 pueden variar). Los caracteres 129 a 256 se utilizan para codificar caracteres propios de distintas regiones o lenguas. Algunas de estas codificaciones son la ISO-8859-1 (también conocida como ISOLatin1), cuyos caracteres imprimibles puedes encontrar en http://es.wikipedia.org/wiki/ISO-8859-1 (que coincide en todo el rango 1 – 128 con la tabla ASCII), la 850 (DOSLatin1), página de códigos que se utiliza en la consola de MSDOS para Europa Occidental, cuya codificación se puede encontrar en http://aspell.net/charsets/codepages.html, o Windows 1252 (WinLatin1), que también se puede encontrar en http://aspell.net/charsets/codepages.html. Es interesante comprobar algunos caracteres especiales (entre las posiciones 129 y 256) en cada una de ellas y ver si coinciden o no, eso podría explicar algunos fallos de visualización que podemos encontrar en algunos ficheros de texto o páginas web.

La falta de un estándar globalmente aceptado y la diversificación de normas de codificación llevaron a la creación del repertorio de caracteres Unicode. Unicode es un estándar de representación de caracteres que pretende facilitar el trabajo con caracteres de múltiples idiomas (incluyendo lenguas muertas o lenguas basadas en ideogramas como chino, coreano, japonés) o disciplinas técnicas (como matemáticas, física, química, ingeniería).

Veamos ahora con un poco más de detalle la tabla de caracteres para una parte de Unicode. En primer lugar vamos a presentar el repertorio de caracteres y su posición (es decir, la tabla de caracteres) y más adelante presentaremos los algoritmos y reglas de codificación de dichos caracteres. No incluimos los repertorios y tablas en este texto por ser demasiado extensa. Se pueden encontrar en http://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF. Si navegas por esta página podrás observar que hay al menos otras 15 tablas como la anterior (cada una con 4096 caracteres), dando lugar a lo que se conoce como BMP (Basic Multilingual Plane), y algunas otras más. La tabla anterior sólo contiene $16^3 = 4096$ caracteres (a día de hoy se calcula que Unicode cuenta con más de 100.000 símbolos ya definidos, y dispone de espacio para al menos 2^{20} símbolos).

Unicode, tal y como la hemos introducido, es una tabla de caracteres (o, en cierto modo, un conjunto de tablas de caracteres). Pero todavía no hemos dicho nada de la codificación de estos caracteres en sistema binario.

El número de caracteres que permite Unicode es tan elevado que el mismo ha dado lugar a distintas formas de codificación. Las tres más extendidas son:

1. UTF - 8. Codificación orientada a byte (de 8 bits u octeto) con símbolos de longitud variable (cada símbolo ocupará en memoria de 1 a 4 octetos). La codificación UTF – 8 permite representar no sólo 256 caracteres (un octeto) sino cualquier carácter que esté en Unicode. Esto es posible debido a la idea de codificación de longitud variable. Si queremos representar un carácter que esté entre los símbolos 0 y 127 de Unicode (que coinciden con los ASCII), la codificación utilizará en memoria un único octeto. Si queremos representar un carácter que esté fuera de dicho rango, su codificación utilizará de 2 a 4 octetos, 16 a 32 bits.

2. UTF – 16. Codificación de 16 bits de longitud variable (cada símbolo ocupará en memoria 1 ó 2 segmentos de 16 bits). Por tanto, desde el punto de vista de memoria, es bastante peor que UTF – 8 (por lo general, un texto que no contenga caracteres especiales, ocupará el doble que en UTF - 8). Sin embargo, permite codificar con dicha longitud de palabra 2^{16} = 65536 caracteres, lo cual nos asegura que prácticamente cualquier carácter que queramos almacenar va a estar disponible. Gracias a que es de longitud variable, si pretendemos codificar algún carácter que no esté entre los 65536 disponibles pero sí en las tablas Unicode (hasta la posición 10FFFFF), aumentará la longitud de palabra a 2 bytes.

3. UTF – 32. Codificación de longitud fija. Dispone de 2^{32} (más de 4 * 10^9) símbolos. Cada carácter ocupa en memoria 32 bits (4 octetos o bytes).

Abre el enlace correspondiente a la primera tabla del BMP (http://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF) y observemos algunos detalles que nos permitirán repasar algunos de los conceptos vistos hasta ahora.

- En primer lugar, si compruebas la filas que van desde la "0000" hasta la "0070" (¿qué número representa 0070 en formato hexadecimal?) y las columnas de la "0" hasta la "F", podrás ver que los 128 primeros caracteres de Unicode, tal y como ya habíamos anunciado, son idénticos a los de ASCII (y a los de todas las codificaciones que hemos visto hasta ahora). Por ejemplo, el símbolo "e", que en la tabla de caracteres ocupa la posición "0065" en hexadecimal, diríamos que ocupa la posición 101 (en sistema decimal) de la tabla de codificación Unicode o la posición "1100101" en binario. Como ya anunciamos anteriormente, la posición en la tabla no tiene por qué coincidir con la codificación en binario. Su codificación binaria en UTF – 16 sería "00000000 01100101", y en UTF – 8 sería "01100101" (el último octeto de la representación UTF - 16).

- En segundo lugar, puedes comprobar los caracteres que van desde la fila "0800" hasta la "0F00". Son los caracteres que en la tabla Unicode ocupan la posición desde la 129 hasta la 256. A priori, son aquellos cuya codificación debería ocupar sólo un octeto (un byte) en codificación UTF-8. En realidad la codificación UTF-8, al ser de longitud variable, debe incluir bits de paridad en cada octeto, y ello hace que, por ejemplo, en la práctica, el carácter "ñ" no se codifique como "F1" en hexadecimal, o "11110001", sino como "C3B1", o "11000011 10110001" (lo puedes comprobar, por

ejemplo, con el "Pspad"). La razón por la que "11110001" pasa a convertirse en "11000011 10110001" se puede encontrar en http://es.wikipedia.org/wiki/UTF-8 (en realidad no nos preocupa tanto el cómo se codifican, sino que el motivo para hacerlo así es para poder definir un código de longitud variable). Sería interesante comparar los símbolos que ocupan en la tabla Unicode las posiciones del 129 al 256 con los equivalentes de ISO-8859-1 o Windows-1252.

- Las filas siguientes, de la "0100" en adelante, contienen los símbolos Unicode a partir del 256. Si los observas verás que algunos de ellos son de uso muy extendido. Su codificación requiere más de un octeto. Sin embargo UTF-8 permite codificar los mismos gracias a su longitud variable. Por ejemplo, si queremos codificar la cadena "dě" en UTF-8 (en UTF–16 no supondría ningún problema, al encontrarnos dentro de su rango de símbolos) nos encontraremos con la siguiente secuencia de bits (d = 100, ě = 283): 01100100 11000100 10010101.

Como en todas las codificaciones de longitud variable, el problema ahora es intentar decodificar la anterior cadena "dě". ¿Cómo sabe el decodificador de la misma que debe considerar "01100100" como un primer carácter y "11000100 10010101" como un segundo carácter? ¿Por qué no decodificarlo como una cadena de tres caracteres? ¿O considerar los dos primeros octetos como un símbolo y el tercero como otro? Éste es el gran problema de las codificaciones de longitud variable. Sin información adicional, es imposible saber cuáles son los símbolos que debemos producir al decodificar.

Para eso hemos hecho antes el "truco" de convertir la "ñ", "F1" en hexadecimal, o "11110001", en "C3B1" o "11000011 10110001". En codificación UTF–8 sabemos que siempre que un octeto empieza por "110xxxxx" (como la "ñ" o la "ě" anterior), estamos ante un símbolo de 2 octetos (y por lo tanto debemos leerlo y decodificarlo junto al siguiente símbolo). Si aún necesitáramos usar tres octetos para algún carácter, el primero de ellos comenzaría por "1110xxxx". El segundo (y en su caso el tercer) octeto siempre comenzarían por "10xxxxxx" (de nuevo, como es el caso en los segundos octetos de la "ñ" o la "ě"). Así se explica que los caracteres del 129 al 256 (cuya representación binaria en UTF–8 sería "1xxxxxxx") no puedan ser representados con un único octeto, sino con dos octetos.

Si bien la forma de codificación en UTF–8 puede resultar un poco complicada de entender, es importante retener las ideas de que existen codificaciones de longitud variable, de que Unicode en sus distintas codificaciones se está convirtiendo en un estándar ampliamente aceptado en la informática, y por supuesto de la importancia de codificar y decodificar con respecto a la misma norma de codificación para recuperar los textos originales.

Sólo como curiosidad, en la página web http://www.sslug.dk/~chlor/utf-8/ se pueden encontrar los símbolos Unicode entre las posiciones 129 y 2047, con su codificación correspondiente en hexadecimal y en binario (por ejemplo, puedes buscar la "ñ" o la "ě" y compararlas con las codificaciones que hemos propuesto antes).

El hecho de que los ordenadores se basen en una "tecnología binaria" ha motivado que, al hablar de unidades de información (por analogía de memoria), trabajemos siempre en potencias de 2 y, por ello, los típicos prefijos del Sistema Métrico Decimal, Kilo, Mega, Giga… tienen una pequeña diferencia cuando hablamos de memoria:

1 Kb (Kilobyte)	2^{10} bytes = 1024 bytes
1 Mb (Megabyte)	2^{20} bytes = 1024 Kb
1 Gb (Gigabyte)	2^{30} bytes = 1024 Mb
1 Tb (Terabyte)	2^{40} bytes = 1024 Gb

¿Serías capaz de calcular cuantos caracteres en codificación ISO–8859–1 podemos alojar en un Mb, o en un Gb?

¡ATENCIÓN!: La diferencia (de considerar 1024 en lugar de 1000) es sólo al hablar de unidades de información o memoria. Kilo, Mega, Giga… son prefijos usados en el Sistema Métrico Decimal y alrededor de un sistema informático hay muchas cosas que se miden que no son cantidades de memoria. Sirva como ejemplo la velocidad de un procesador (2'4 Ghz, por ejemplo).