Texto para la asignatura

Computación 1

Introducción a la Informática para estudiantes de Ingeniería

- curso 2010 -

Material generado para la asignatura
"Computación 1" de Facultad de Ingeniería
por el Grupo CeCal del Instituto de Computación
-Universidad de la República-
Año lectivo 2010
Montevideo - Uruguay

Índice

1 Alcance de este material

Este material forma parte del curso de Introducción a la Computación para el Ciclo Básico de Ingeniería que se dicta en la Facultad de Ingeniería de la Universidad de la República.

En él se pretende dar un panorama de lo que es la informática como herramienta para ingenieros. El texto pretende servir de ayuda al estudiante pero no está pensado para reemplazar la asistencia a clase ya que allí se desarrollará cada tema con mayor profundidad, se darán ejemplos de lo expuesto y propondrán ejercicios que hagan reflexionar sobre los conceptos vertidos.

La informática afecta todas las áreas del desenvolvimiento profesional, ya no sólo los cálculos de ingeniería sino la gestión administrativa de la empresa, de los proyectos, la forma de presentación de la información a terceros, etc. Ya no es una opción sino una herramienta que debe ser utilizada para obtener niveles de competitividad -calidad, precio, eficiencia- aceptables que hagan del Ingeniero o de su empresa una opción a tener en cuenta en la contratación de trabajos.

Debe quedar claro que la validez de este material, a menos de la información histórica, lo es para el momento y en el contexto en el cual fue escrito. Los avances continuos en el área informática lo pueden hacer obsoleto en términos de meses. Es por ello que tratamos de verter conocimientos genéricos que sabemos han de perdurar en el tiempo más que el detalle circunstancial.

Material generado por el Grupo “Centro de Cálculo” del Instituto de Computación originalmente para el curso del año 2005 por los docentes Leticia Pérez y Juan González. Actualizado en 2010 por los docentes del Grupo.

2 Arquitectura de Computadoras

2.1 Representación de datos en computadoras

Los computadores digitales actuales se basan en una tecnología electrónica que permite representar los datos mediante combinaciones de circuitos. Estos elementos básicos sólo admiten dos estados (encendido o apagado) representados por el nivel de tensión a su salida.

La información que se representa mediante la combinación de elementos que sólo admiten dos estados se denomina información binaria. Cada uno de los elementos de la información binaria recibe el nombre de bit (binary digit) y se codifica mediante el empleo de los símbolos 0 o 1.

Dicho de otro modo, cualquier dato que se deba procesar en un computador digital deberá estar representado por un código formado por una secuencia de ceros y unos. La codificación consiste en establecer reglas que definan una correspondencia entre los datos y la secuencia de bits que constituye su código.

Los datos que proporcionamos a los computadores pueden ser de distintos tipos: cadenas de caracteres, números enteros, números reales, fechas, etc.

Veremos brevemente cómo los datos son representados internamente en un computador para poder trabajar con ellos. Existen varios criterios genéricos para establecer esta correspondencia, que dan lugar a tipos diferentes de representaciones internas.

2.1.1 Tamaños privilegiados

Bit: Es la unidad elemental de información que se usa en computación, es un objeto que toma sólo dos valores posibles: 0 ó 1: el BIT (dígito binario)

Estudiaremos entonces, la representación interna de los datos como la expresión de los distintos tipos en función de estructuras de bits.

Byte: Unidad de información formada por 8 bits.

Los múltiplos de Byte son Kilobyte = 1024 bytes, MegaByte = 1024 KB, GigaByte = 1024 MB, TeraByte = 1024GB, etc. Se usa el número 1024 por ser una potencia de dos (2^{10}) que es la base del sistema de numeración binario.

Palabra: Unidad de información formada por 2 bytes = 16 bits

Doble Palabra: Unidad de información formada por 2 palabras = 4 bytes = 32 bits

2.1.2 Representaciones Alfanuméricas

Los caracteres (letras, símbolos, y dígitos numéricos) se representan por medio de una serie de códigos de n-bits. De esta forma, a cada carácter se le asigna, arbitrariamente, un número. Hay distintos juegos de códigos como:

- EBCDIC (7 bits): Antiguo, utilizado en ordenadores IBM. Actualmente está en desuso.
- ASCII (7 bits): Antiguo, aunque considerado como estándar.
- ASCII (8 bits): Es ampliamente usado, incluye tratamiento de algunos caracteres de lenguas europeas (acentos, ç, ñ, etc.), si bien estos no son estándar.

- UniCode (16bits): es el más moderno, desarrollado como nuevo estándar. Es una ampliación del código ASCII de 8 bits diseñada para codificar distintos juegos de caracteres: latino, griego, árabe, kanji, cirílico, etc., el precio de tanta flexibilidad es ocupar el doble de espacio que ASCII: es de 16 bits.

La forma más habitual de representar el código ASCII, y en general todos los sistemas de codificación de caracteres, es a través de una matriz cuyas columnas están asociadas a los 3 (o 4) bits más significativos del código (0 a 7), y sus filas a los 4 bits menos significativos (0 a F) tal como se muestra en la siguiente tabla:

De los 128 valores posibles del código (7 bits) 10 se utilizan para los dígitos decimales (del 30 hex al 39 hex), 26 para las letras mayúsculas (del 41 hex al 5A hex), 26 para letras minúsculas (del 61 hex al 7A hex), 34 para símbolos especiales (espacio, !, #,$,%,/,&,+,-,*, etc.), los 32 primeros se denominan genéricamente "caracteres de control" y se utilizan esencialmente en la comunicación de datos y con fines de dar formato a los textos en impresoras y pantallas.

Tabla 1: codificación ASCII de 7 bits

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

Tabla 2: Ejemplos de codificación de caracteres

Carácter	Valor Binario	Valor Hexadecimal	Valor Decimal
k	0110 1011	6b	107
K	0100 1011	4b	75
6	0011 0110	36	54
?	0011 1111	3f	63

2.2 Esquema general de una computadora

Una computadora consta de diferentes partes con funciones específicas. A continuación describiremos un esquema general de cómo se conforma una computadora típica tratando de abstraernos de detalles de implementación.

2.2.1 Procesador (CPU: Control Process Unit)

Elemento encargado del procesamiento de instrucciones. En este caso es la unidad que ejecuta los programas de aplicación en general. Hay también procesadores específicos para ciertas tareas especialmente usados en automatismos de máquinas y equipos.

Los principales parámetros de un procesador son:

Cantidad de instrucciones que puede ejecutar. Hay dos grandes familias: CISC (Complex Instruction Set Computers) y RISC (Reduced Instruction Set Computers). Si bien "pocas instrucciones" significa que operaciones complejas requieran varios ciclos de reloj para ser concluidas, la tecnología actual permite que circuitos de pocas instrucciones puedan funcionar a velocidades más altas que los de muchas instrucciones.

Largo de palabra del procesador. Cantidad de bytes que el procesador puede manejar simultáneamente. A mayor cantidad de bytes más información podrá procesar en un solo ciclo de reloj. Los equipos de arquitectura PC comenzaron con un procesador de 8 bits y hoy están en 64 bits.

Velocidad del procesador. Usualmente medida en Giga Hertz (mil millones de ciclos de reloj por segundo), donde ciclo es equivalente a instrucción de procesador conocidas como microinstrucciones.

2.2.2 Coprocesador

Se encarga del proceso de un cierto conjunto de instrucciones especializadas. Si no existe las resuelve el Procesador Central mediante algoritmos implementados en software. Históricamente el coprocesador matemático era un opcional, hoy por razones de costo ya viene incorporado al procesador central. Otros coprocesadores pueden encontrarse en las tarjetas de vídeo (pueden procesar imágenes en forma automática a demanda) y en las tarjetas de audio.

2.2.3 Memoria

Un computador es básicamente un equipo capaz de realizar cálculos. En la medida que la capacidad de cálculo crece se hace necesario almacenar información para que esté disponible por el procesador y para que éste pueda a su vez almacenar los nuevos resultados. Sin la memoria, en sus diversas formas, el computador no puede hacer (ni ser) más que una calculadora de bolsillo.

Los parámetros más relevantes de la memoria son:

Volátil o permanente. Volátil es aquella memoria que conserva la información asignada mientras sea alimentada por una fuente de energía, si se corta el suministro eléctrico toda la memoria adquiere un único estado y por lo tanto se pierde toda información. Cuando es permanente se puede cortar el suministro de energía y la información permanecerá por tiempos muy prolongados o infinito.

Velocidad de acceso. Todas las memorias tienen dos tiempos de acceso: para grabar datos y para leer datos. Leer es más rápido que grabar y en general memorias de igual tecnología mantienen entre ambas una relación lineal por lo cual suele mencionarse uno sólo de estos parámetros.

De lectura y grabación y sólo lectura. Hay memorias en las que el usuario final puede grabar datos y luego leerlos y otras en que sólo puede leer. También hay algunas variantes en donde la grabación no es algo que se espere que el usuario final pueda hacer con frecuencia pero que en caso de ser necesario puede hacerlo.

Capacidad de almacenamiento. Define el tamaño de la memoria, la cantidad de unidades de memoria (bits o bytes) que puede albergar.

A continuación describimos los diferentes tipos de memoria que podemos encontrar en un computador típico.

2.2.3.1 Memoria RAM: Random Access Memory

Memoria de acceso directo en modo lectura-grabación donde se almacena la información mientras está en ejecución un programa. Puede ser el propio programa, parte de él o los datos del programa.

En las computadoras con sistema operativo multitareas, en la memoria RAM hay una mezcla de información de varios programas que se están ejecutando simultáneamente, cada uno de ellos en un espacio de direcciones reservado (como se muestra en la Ilustración 1).

Es una memoria volátil, si se corta el suministro de energía su contenido se pierde.

P1	P1	P1	P2	P3
P3	LIBRE	P4	P5	P6
LIBRE	P7	P7	LIBRE	LIBRE

Ilustración 1: representación de la ocupación de la memoria RAM por diferentes programas y datos

2.2.3.2 Memorias ROM

2.2.3.2.1 Memoria ROM: Read Only Memory

Memoria de acceso directo en modo *sólo lectura* donde se almacenan datos y/o programas. Éstos se cargan a memoria RAM para ser procesados. Es muy utilizada para instalar pequeños programas para equipos electrónicos. Es del tipo *permanente*: su contenido no se pierde si se corta el suministro de energía.

2.2.3.2.2 Memoria EPROM y EEPROM: Erasable and Programmable Read Only Memory

Memoria de acceso directo en modo *sólo lectura* pero que en ciertas condiciones se puede reescribir. Las del tipo EPROM se borran con luz ultravioleta. Las EEPROM (Electric EPROM) se pueden borrar mediante impulsos eléctricos. Sirven para almacenar pequeños programas y datos.

En las computadoras tradicionalmente se almacenan sistemas operativos mínimos y programas muy primitivos de gestión de dispositivos básicos. Habitualmente se la conoce como la BIOS (Basic Input Output System). Es del tipo *permanente*: su contenido no se pierde si se corta el suministro de energía.

Este tipo de memoria es la que almacena los programas de nuestros electrodomésticos: lavarropas, hornos microondas, equipos de audio / DVD, alarmas, etc.

2.2.3.3 Memoria Caché

Memoria extremadamente rápida (varias veces más rápida que la RAM) que sirve de área de trabajo del procesador. Allí se almacena la información que el procesador accede con mayor frecuencia o que forma parte de un conjunto de instrucciones relacionadas entre sí. Es del tipo volátil: su contenido se pierde si se corta el suministro de energía.

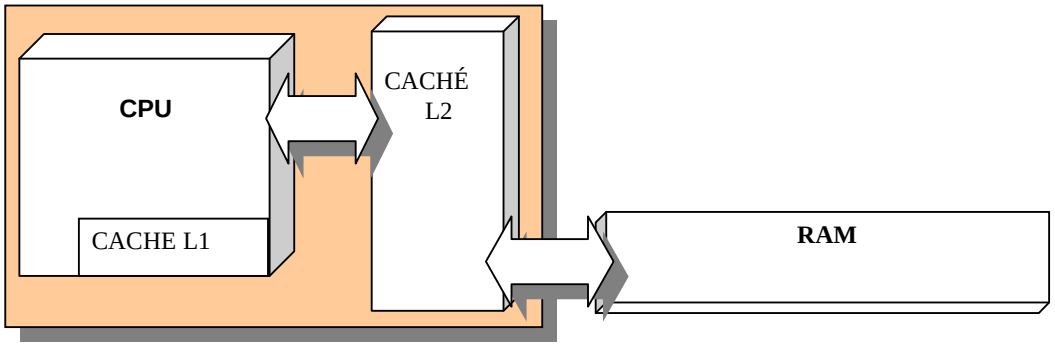


Ilustración 2: Memoria caché: vinculación con la CPU y RAM

Existen dos tipos de memoria caché denominadas L1 y L2. La tipo L1 está alojada en el propio procesador, la L2 está fuera del procesador, es más lenta y menos costosa que la L1.

2.2.3.4 Memoria CMOS

Memoria de la placa principal (motherboard) en la que se almacenan las características y configuración del hardware instalado en la computadora. Es del tipo volátil: su contenido se pierde si se corta el suministro de energía. Para mantener la información cuando la computadora está apagada e incluso desconectada se la alimenta con una pila incluida en la placa madre del computador.

2.2.3.5 Memoria Virtual

La memoria RAM es costosa y puede limitar el desempeño global del hardware. Una aplicación que sea mayor (medida en bytes) que la memoria disponible para ejecutar programas no podrá ser ejecutada. Por otra parte no parece inteligente comprar memoria RAM en tales cantidades que me asegure que no importa qué tareas ni cuántas ejecute, nunca me faltará memoria para albergarlas. La falta de memoria es más frecuente aún cuando hablamos de sistemas operativos multitareas pues para cada tarea (programa) que se ejecuta debe reservar cierta cantidad de memoria.

Para solucionar esta situación se diseñó un mecanismo denominado Memoria Virtual que ejecuta el sistema operativo, el cual permite a una máquina utilizar alguna porción de disco como una forma de almacenamiento temporal que se utilizará en los casos en que la memoria RAM no sea suficiente. El costo de ello es una pérdida notoria en la velocidad de ejecución de las tareas cuando parte de la memoria RAM se graba en el disco y/o cuando se recupera del disco. La ventaja es que se podrán ejecutar todas las tareas que sean necesarias o a lo sumo limitado por el espacio disponible en el disco duro.

La porción de disco que se utiliza como extensión de la memoria RAM de la máquina se denomina “área de intercambio” (swap area). En general el usuario puede indicar cuánta memoria de disco utilizará como extensión, para ello se considerarán las tareas a realizar con la máquina. Una estación de trabajo Unix tendrá típicamente entre dos y cuatro veces la memoria RAM (si tenemos 1 GB de memoria RAM se asignan entre 2 y 4 GB de memoria Virtual en disco).

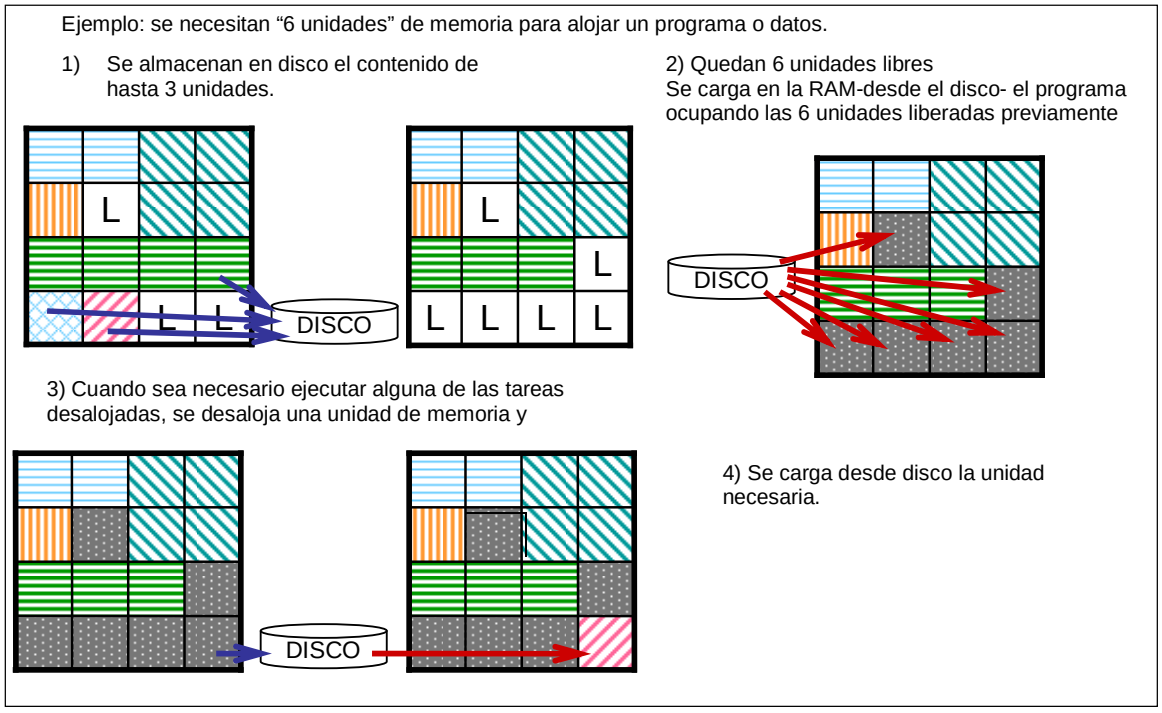


Ilustración 3: Ejemplo del mecanismo de la Memoria Virtual

2.2.4 Bus de datos

Conjunto de circuitos que permiten la transferencia de información entre los diferentes componentes de la computadora (por ej. entre CPU y memoria caché, entre memoria caché y RAM, entre CPU y la tarjeta controladora del monitor, entre la CPU y el teclado, etc.). El Bus está constituido por una parte de hardware y otra de software (en memoria ROM). Existen a nivel de PC's varios tipos de BUS estándar: ISA, EISA, PCI, Microcanal, VL-BUS, AGP. Otros BUSes pueden servir para conectar periféricos como el SCSI y el USB.

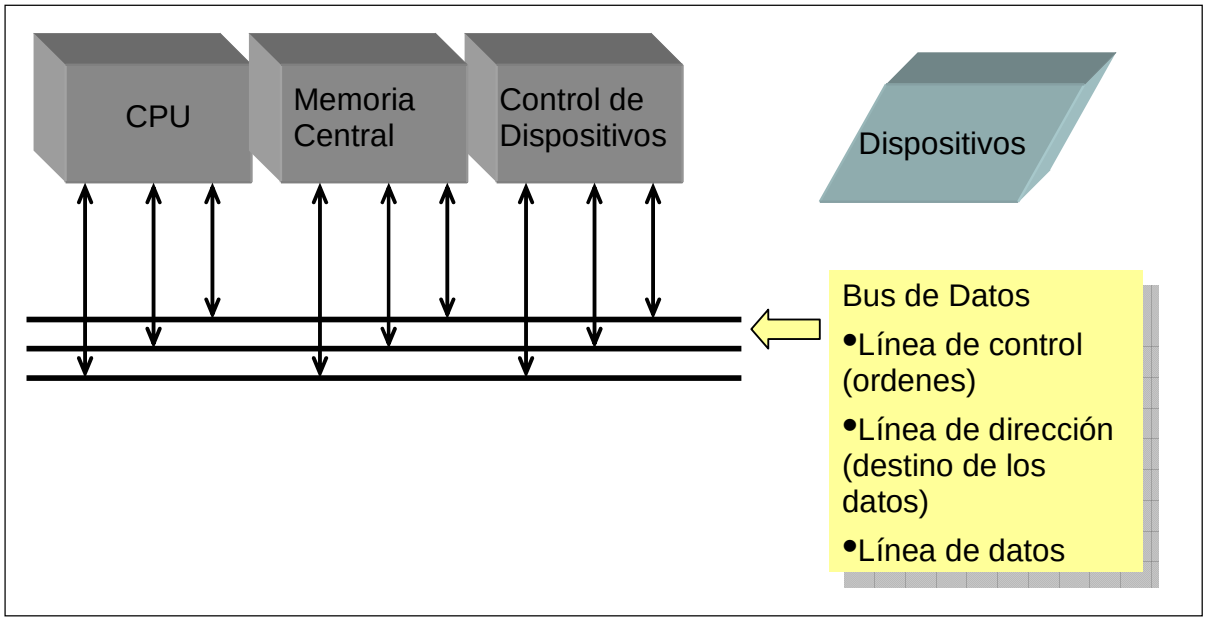


Ilustración 4: Esquema general de comunicaciones dentro de una computadora

2.2.5 Reloj de la computadora

Es la unidad que marca el tiempo de ejecución de cada instrucción y cada operación dentro de la CPU. Todas las actividades de la computadora (no la de los periféricos pero sí la transferencia entre ellos y con la CPU) están regidas por los pulsos (tiempos) que marca este reloj, por eso se utiliza como una forma de medida de la capacidad de proceso de la computadora. Cada pulso provoca la ejecución de una micro instrucción del procesador. Un procesador de 100 MHz debería procesar 100 millones de micro instrucciones por segundo.

Formalmente no es exacto que la capacidad de proceso dependa únicamente de la velocidad del reloj, cuando hablamos de computadoras de igual arquitectura es una forma de comparación razonable, pero en diferentes arquitecturas en general hay muchos otros parámetros involucrados. Entre otros factores que inciden está la cantidad de micro instrucciones que se precisan para realizar una cierta instrucción (por ejemplo para hacer una suma se precisa recuperar los sumandos desde la memoria RAM, sumarlos y luego poner el resultado en memoria RAM), cada paso se hace en un ciclo de reloj, a mayor complejidad de las instrucciones mayores suelen ser la cantidad de micro instrucciones necesarias (ciclos de reloj) y si los procesadores son diferentes pueden diferir la cantidad de micro instrucciones para ejecutar una misma instrucción.

2.2.6 Controladores de dispositivos

Son el hardware específico que permite el manejo de los periféricos. Así por ejemplo tenemos las tarjetas controladoras de discos IDE o las de discos SCSI, de impresoras, de monitores, de puertos seriales, de puertos USB, etc.

Algunos circuitos de control vienen por defecto en la placa madre y otros por separado para permitir que se instale el más adecuado a las necesidades de cada usuario. Por ejemplo, todas las placas madre incluyen los controladores de teclado pero los controladores de vídeo se pueden comprar por separado según las necesidades debido a la variedad de sus características.

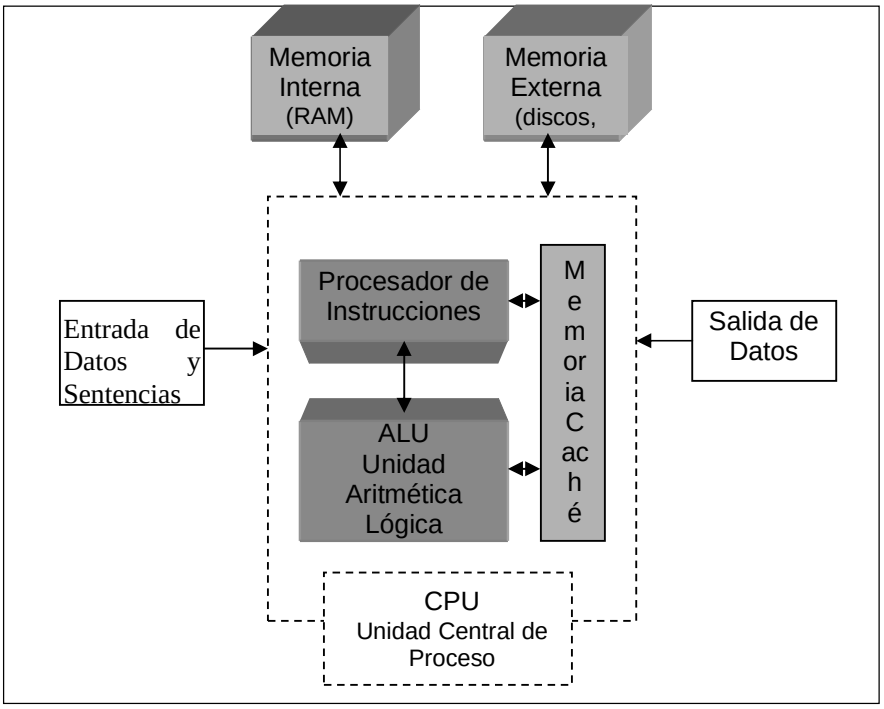


Ilustración 5: Esquema general de un procesador

2.2.7 Discos y archivos

Hay muchos tipos y jerarquías de recursos en informática. Dentro de ellos encontramos uno que sobresale debido al especial papel que cumple en su relación con el usuario final. Se trata de los discos; en sus múltiples variantes son la memoria estable de la computadora, aquella que permanece por tiempo ilimitado (o casi) a nuestra disposición independientemente del equipo.

2.2.7.1 Discos de datos

Son dispositivos que permiten almacenar información de forma que ésta no se pierde si la computadora es apagada.

Los discos almacenan la información, en una estructura física de círculos concéntricos llamados pistas. Cada pista está subdividida en sectores que conforman unidades de almacenamiento físico. Esta estructura es "dibujada magnéticamente" en la superficie del disco durante el proceso de formateo.

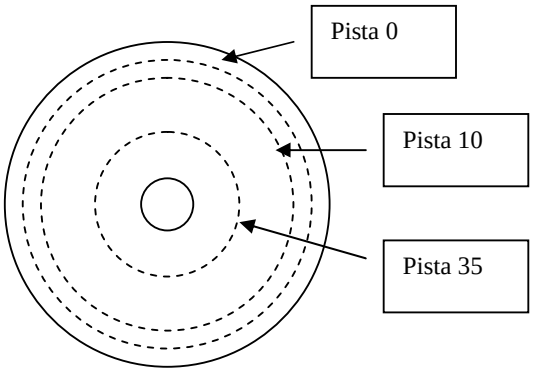


Ilustración 6: Esquema básico de un disco

Así por ejemplo un disquete de 3,5 pulgadas y de alta capacidad (1440 KB), formateado para trabajar con Windows se graba en ambas caras, cada una tiene 80 pistas con 18 sectores de 512 bytes: 2 * 80 * 18 * 512 = 1474560 Bytes = 1440 KB.

La pista cero es donde habitualmente se almacena el "índice" de archivos contenidos (FAT por File Allocation Table) y el área donde eventualmente se graba el sistema operativo en caso de que se vaya a utilizar esa unidad para arrancar el computador.

Se los puede clasificar de varias maneras, por los principios técnicos que les permiten retener la información o por su funcionalidad.

Desde el punto de vista técnico tenemos los discos magnéticos y los ópticos.

El principio técnico utilizado por los discos magnéticos es el mismo que el de las cintas y casetes de audio y vídeo. Una fina película de material magnetizable y un cabezal capaz de afectar y detectar la polarización de cada parte de la superficie.

En los discos ópticos la superficie se “talla” de tal modo que un haz de luz láser que incida sobre esa superficie será reflejado hacia un lector o no, con esa cualidad podemos representar ceros y unos con los que construir información binaria.

Duros y fijos (Hard Disk): son aquellos que están conectados en forma permanente a la computadora. Sus parámetros típicos son la velocidad de acceso y su capacidad. Por lo general el sistema operativo reside en algún disco fijo conectado a la computadora. Sus características físicas: diámetro, uno o varios discos de material rígido, velocidad de rotación, velocidad de movimiento del cabezal para ubicar la "pista" correcta son algunos de los determinantes de su rendimiento.

Duros y removibles (*Removable Hard Disk):* técnicamente son iguales a los anteriores pero están provistos de una carcasa que les permite ser instalado y extraído con suma facilidad para ser luego llevados a otra computadora. Han perdido algo de terreno frente al CD-ROM, pero sigue siendo útil para solucionar problemas de seguridad y confidencialidad de la información.

Flexibles o disquetes (Floppy disk): son de mucha menor capacidad comparados con los anteriores y se utilizan por lo general para transportar información. Su característica física (un disco de material flexible que no puede rotar a la misma velocidad que los rígidos) determina que posea menor densidad de datos y que opere a velocidades notablemente inferiores que los discos duros. Además para preservarlo de un desgaste innecesario la unidad se enciende estrictamente cuando se requiere lo que agrega un parámetro más a los tiempos de acceso incrementándolo. Este tipo de discos han caído prácticamente en desuso.

Los siguientes son de tecnología óptica con lectores láser:

CD-ROM: son discos de “sólo lectura” de tecnología de lectura óptica (vía láser). Debido a su bajo costo y alta capacidad de almacenamiento se utilizan para la distribución de software e información multimedia. Un disco estándar puede almacenar hasta 800 MB de información.

CD-R – CD Recordable (originalmente CD-WORM -Write Once Read Many): Unidades que también graban CD’s por única vez que luego pueden ser leídos múltiples veces. La importancia es que el formato de almacenamiento de los datos sea compatible con el estándar de CD-ROMs.

CD-RW – CD Read/Write: Unidades que se pueden grabar múltiples veces (CD Regrabable).

DVD-ROM (Digital Versatile Disk o Digital vides Disk): Es el sucesor del CD-ROM. Alcanza densidades de grabación varias veces superiores a ellos permitiendo en sus versiones más sencillas almacenar hasta 4.700 MB y son adoptados como soporte por la industria del vídeo.

Blu-ray (Blu-ray Disc): Es un nuevo formato de disco óptico sucesor del CD y el DVD que se utiliza entre otras cosas para almacenar vídeo de alta definición. Su capacidad de almacenamiento llega a 25 GB para los discos de una capa.

Habrá observado el lector que tanto en CDs como DVDs se incorpora habitualmente la extensión ROM. Ello es un formalismo que permite distinguir el uso que se le da al dispositivo. Cuando se utiliza para almacenar datos (no audio o video en los formatos convencionales) se considera a estos dispositivos una variante de memoria ROM (Read Only Memory). Con la popularización de las grabadoras de esos dispositivos pierde sentido hablar de dispositivos ROM.

Hay muchas variantes que no son consideradas estándares y permiten en general grabar mayores volúmenes de información. Aquí resumimos las más utilizadas.

2.2.8 Dispositivos de memoria estable

En los últimos años se han desarrollado dispositivos de memoria que no responden a ninguno de los dos anteriores. Estos son utilizados en una gran diversidad de dispositivos: cámaras de fotos digitales, teléfonos celulares, dispositivos de audio (comúnmente conocidos como reproductores MP3), dispositivos de memoria conectables a puertos USB (conocidos como pendrives), tarjetas inteligentes, etc.

Este tipo de memoria se conoce como "memorias flash". Se caracterizan por:

- no emplear ningún mecanismo mecánico para su lectura y/o grabación, lo que hace que sean dispositivos silenciosos.
- permitir que múltiples posiciones de memoria sean escritas o borradas en una misma operación, lo que hace que puedan funcionar a velocidades superiores a otros tipos de memorias.
- mantener su contenido sin energía.
- resistencia a los golpes.
- bajo consumo de energía.
- tener tamaño pequeño.

Existen multiples tipos de memorias flash como por ejemplo: CompactFlash, SmartMedia, Memory Stick, SD, MiniSD, MicroSD, xD-Picture Card, etc. Actualmente, la mayoría de PCs tienen incorporadas ranuras que permiten utilizar una gran variedad de tarjetas de memorias.

2.2.9 Archivos

En los discos la información se almacena en estructuras lógicas denominadas particiones, directorios y archivos que el sistema operativo construye sobre los discos como forma de administrar el espacio disponible. La forma de hacerlo varía con cada sistema operativo pero en general se utiliza una estructura jerárquica (en forma de árbol). Cada nodo u hoja del árbol puede ser un ARCHIVO o un DIRECTORIO.

2.2.9.1 Partición

Una partición de disco es una segmentación del disco físico en una o más partes lógicas de tal forma que el sistema operativo luego vea un disco "lógico" por cada una de ellas. Cada partición tendrá un área de disco contigua con su tamaño físico y su FAT (File Area Table) y podrá o no ser un disco que sirva para arrancar el sistema operativo (tendrá o no un "boot area"). Cada partición tiene un determinado sistema de archivos. Un sistema de archivos es un conjunto de reglas que determina las características y el comportamiento del sistema operativo respecto a los archivos. Ejemplos de sistemas de archivos son FAT (DOS, Windows 95/98), FAT32 (Windows 2000/XP/VISTA/7), NTFS (Windows 2000/XP/VISTA/7), UFS (Unix File System, común a los sistemas operativos Unix). FAT32 utiliza estructuras de 32 bits pero es básicamente igual a FAT. NTFS es el recomendado por Microsoft para Windows 2000/XP y permite entre otras cosas que las carpetas y/o archivos puedan tener atributos de *seguridad* que indican al sistema operativo si pueden o no ser leídos, grabados y/o ejecutados según las políticas de acceso definidas, es muy parecido a UFS que se considera su origen. Para cada sistema de archivos el sistema operativo deberá utilizar algoritmos de acceso y control distintos. Diferentes particiones pueden coexistir al mismo tiempo en un mismo disco físico.

La unidad de grabación en bytes también es una característica del sistema de archivos que se puede configurar al momento de definir y formatear la partición. Por defecto UTF (por citar un caso) define bloques de 512 bytes. Esto quiere decir que si se quiere grabar un archivo de 40 bytes se ocupará un bloque de 512 (más la correspondiente entrada en la FAT). Si se pretende guardar 1200 bytes ocuparemos tres bloques de los cuales el último no estará aprovechado al máximo. Las ventajas y desventajas de utilizar distintos tamaños de bloques dependen del tipo y volumen de la información y en general es analizado por profesionales en informática.

2.2.9.2 Archivo

Un ARCHIVO es desde el punto de vista del sistema operativo un conjunto de bytes. Estos bytes pueden codificar cualquier tipo de información, desde un texto hasta una imagen digitalizada.

2.2.9.3 Directorio

Un DIRECTORIO es la raíz de un sub árbol. Bajo un subdirectorio se guardarán archivos y eventualmente contendrá otros directorios llamados subdirectorios de éste.

Esta estructura se genera en el dispositivo antes de empezar a utilizarlo en el proceso de dar formato.

Una forma de indicarle al sistema operativo Windows 95/98/NT/2000/XP/ VISTA/7 el nombre completo de un archivo es la siguiente:

[DISCO]**:**[CAMINO]\[ARCHIVO]**.**[EXTENSIÓN]

donde

DISCO es una letra que indica en qué disco (unidad) está el archivo. Este puede ser una partición de un disco físicamente conectado al equipo o un directorio de un disco de accesible a través de la red al que en nuestro equipo identificaremos con esa letra de unidad.

CAMINO es una lista de nombres de directorios separados por el carácter \. Indican el sub árbol donde se encuentra el archivo. Cuando el camino completo está constituido por "\" exclusivamente, se dice que se está en el directorio "raíz" del disco.

ARCHIVO es el nombre del archivo, el cual está formado por caracteres alfanuméricos

EXT es la "extensión" del nombre del archivo, la cual está formada por n caracteres. Si bien podemos poner cualquier extensión a un archivo (u omitirla) es importante respetar las convenciones preestablecidas ya que muchos programas (en particular el sistema operativo) saben qué programa es el apropiado para procesar un archivo a través de su extensión: xls: planilla excel, doc: documento de MS-Word, txt: archivo de texto plano, jpg: imagen en formato jpeg, htm ó html: página web en formato html, etc.

Ejemplos:

- `a:\directorio1\directorio2\prueba.txt`
- `e:\usuarios\pepe\doc\so.doc`
- `c:\archivos de programa\Procesadores de Texto\Word\winword.exe`

Si el archivo estuviera en otra máquina sobre la cual se tienen los permisos adecuados, la forma de apuntar al mismo sería:

\\nombre_máquina\directorio_público\subdirectorios\archivo.txt

Hay sistemas operativos que carecen del concepto de “extensión del nombre de un archivo”, como el Unix que además distingue (considera distintas) letras mayúsculas y minúsculas. Otros como el Windows 9x/NT/2000/XP/VISTA/7 soportan nombres "largos", distinguen mayúsculas de minúsculas con ciertas limitaciones pero mantienen el criterio de reconocer el contenido de un archivo por su extensión.

3 Sistemas Operativos

Toda actividad realizada en una computadora es producto de la ejecución de un programa.

Un programa es una secuencia de instrucciones que la computadora es capaz de entender y ejecutar. Dichas instrucciones ejecutadas por la CPU hacen uso de los recursos (memoria, disco, teclado, impresora, modem, monitor, etc.) que dispone la computadora.

El sistema operativo (SO) es un programa o conjunto de programas que se encargan de administrar los recursos que la computadora tiene y hacerlos accesibles a los programas que el usuario ejecuta.

3.1 Proceso de arranque (Boot process)

Las computadoras poseen en memoria no volátil (ROM o EPROM – al que se conoce como BIOS -) un programa que es ejecutado automáticamente por ellas al ser encendidas. Este programa realiza una verificación del hardware básico (memoria RAM, teclado, monitor, discos, etc.) utilizando para ello la información de configuración almacenada en la propia BIOS y por el usuario en la memoria CMOS. Una vez finalizado el chequeo, el SO va a leer del dispositivo que le indique la CMOS, se encarga de leer las instrucciones del SO desde ese dispositivo de almacenamiento, lo coloca en memoria RAM y luego ordena a la CPU que ejecute la primer instrucción del SO con lo cual de aquí en más será el SO quien administre el equipo.

El SO terminará de configurar dispositivos más específicos, cargará en memoria RAM rutinas de administración de recursos (más sofisticadas y optimizadas que las que se encontraban en la EPROM), eventualmente algún programa residente y quedará esperando las órdenes del usuario.

Esto implica que el SO es el primer programa (o uno de los primeros) que es cargado y ejecutado por la computadora y típicamente realizarán como mínimo las siguientes tareas:

- verificar el hardware disponible (especialmente aquel que no verifica el BIOS)
- cargar en memoria los programas que controlan dicho hardware
- configurar en forma optimizada algún dispositivo de entrada / salida (por ej.: monitor, teclado, tarjeta de sonido)
- quedar a la espera de órdenes del usuario y / o ejecutar algún programa

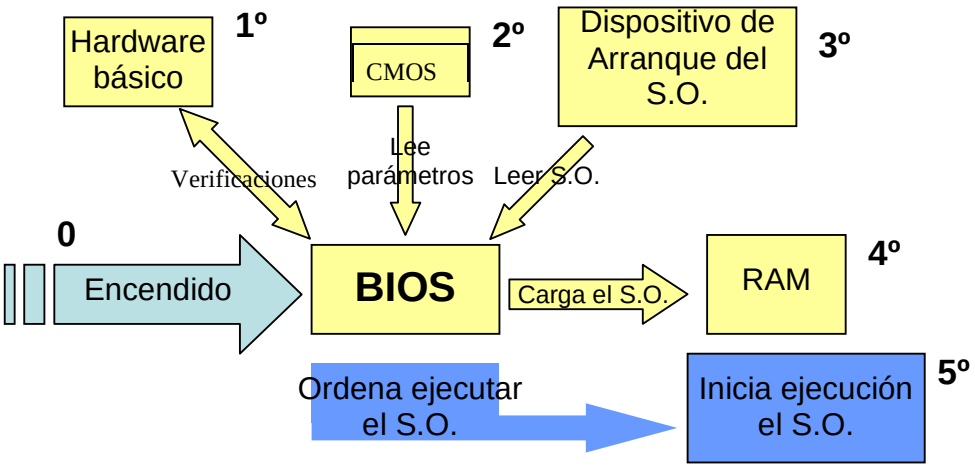


Ilustración 7: diagrama del proceso de arranque de un computador

Actualmente es frecuente que se tengan varios sistemas operativos instalados en forma simultánea. En estos casos en nuestra descripción el lugar del sistema operativo lo ocupa un programa que despliega al usuario un menú del cual debe seleccionar el sistema operativo final que desea ejecutar. Entre las opciones más frecuentes (los escenarios son meramente ilustrativos de las razones por las cuales se puede tener esas opciones) pueden encontrarse:

Linux - Windows 7	Para quienes desean probar Linux pero al mismo tiempo poder usar todo el software de Windows 7 (no en forma simultánea, o uno u otro).
Windows Vista - Windows 7	Caso típico de quien desea migrar su instalación en Windows Vista a Windows 7 pero quiere estar tranquilo de no perder funcionalidad.
Windows Vista – Windows 7 papá – Windows 7 niños	Alguien que está migrando de Windows Vista a Windows 7 pero quiere estar tranquilo de que sus hijos no alteren su entorno de trabajo.

3.2 Funciones del SO

Una vez que el usuario selecciona qué programa quiere usar, éste es leído desde el disco u otro dispositivo de almacenamiento por el SO y a continuación le indica a la computadora que lo ejecute.

En los SO modernos toda la interacción entre los programas y el hardware es realizada exclusivamente vía el SO. Es decir, si un programa quiere acceder a un dispositivo (recurso) X conectado a la computadora, le pide al SO que realice la operación *A* sobre el dispositivo *X*.

Esto tiene las siguientes ventajas:

Independiza a los programas del hardware, no importa qué modelo de disco, monitor o ratón tenga el computador, si el SO lo sabe utilizar (porque fue configurado para ello) entonces cualquier programa escrito para ese SO lo podrá utilizar.

Permite controlar quién y cuándo puede acceder a determinado recurso, verificando los permisos de acceso a los recursos y haciendo que distintos programas utilicen en forma sincronizada los recursos evitando y/o controlando el acceso simultáneo al mismo dispositivo.

Podemos encontrar que en ciertas circunstancias se programan aplicaciones que utilizan cierto hardware directamente sin pasar por el SO Esto se hace en casos donde es estrictamente necesario por motivos de optimización, esas aplicaciones pueden tener problemas si las queremos luego trasladar a otras máquinas con configuraciones diferentes. Un ejemplo típico son algunos juegos de PCs muy avanzados que para cargarse desalojan al propio SO tomando el control total del equipo.

Entre las funciones del SO está la gestión de la memoria y la implementación de la Memoria Virtual según se describió en el capítulo Arquitecturas.

Hay muchos SO y pueden ser clasificados de muchas formas, una de las más comunes es la siguiente en función de si permiten y en qué condiciones su uso por uno o varios usuarios:

Monousuario: no se hace diferencias entre distintos usuarios. Aunque la computadora pueda ser utilizada por muchas personas el SO no hace diferencia entre ellas. No hay control de acceso a los recursos. (DOS, Windows 3.1)

Multiusuario: el sistema diferencia entre usuarios y mantiene configuraciones diferentes para cada uno. Típicamente los usuarios deben identificarse para poder utilizar la computadora (Unix, Windows NT/2000/XP). El acceso a los recursos es controlado en función de las propiedades del usuario y del recurso.

Debemos distinguir en esta última categoría a aquellos SO que permiten trabajar simultáneamente a más de un usuario de los que sólo permiten que trabaje uno por vez. Según esta subclasificación en la primera categoría entran Unix y la mayoría de los SO de mainframes. En la segunda quedarían Windows NT/2000/XP/ VISTA/7.

Otra clasificación es según permitan ejecutar más de una tarea simultáneamente.

Monotarea: permite ejecutar un sólo programa a la vez (DOS). Hoy se piensa en sistemas operativos monotarea sólo para situaciones muy específicas donde se pretende que todo el potencial del equipo esté dedicado a ejecutar una única aplicación como por ejemplo en algunos equipos de control industrial.

Multitarea: el SO es capaz de ejecutar muchos programas a la vez. (Unix, Windows 2000/XP/2003/ VISTA/7)

3.3 Procesamiento en paralelo

Hoy es frecuente que se ofrezcan en plaza computadores cuyo hardware admite la instalación de más de un procesador. En equipos destinados a desempeñarse como servidores comerciales es frecuente hablar de 2 a 4 procesadores pero en el ámbito científico se puede llegar a decenas de miles de procesadores.

Son muchas las aplicaciones de tales configuraciones, en el ámbito comercial se busca la suma de potencia de proceso sin mayores pretensiones. En el ámbito académico se determina con precisión cuales son las circunstancias y las mejores formas de explotar la paralelización de procesos.

Las formas de resolver el trabajo cooperativo de varios procesadores son variadas y la escogida dependerá de los objetivos y de los recursos (hardware y sistema operativo) disponibles.

Un modelo de paralelización es la ejecución de un sistema operativo en cada procesador y la coordinación del trabajo de los mismos por un programa que se ejecuta en uno de ellos. La alternativa es que el propio sistema operativo esté diseñado para utilizar varios procesadores.

En cualquier caso debemos considerar que la coordinación de las tareas implica un costo, por ello no se puede esperar que la potencia de cálculo aumente tanto como procesadores se pongan, siempre que se incorpora un procesador hay un costo en ciclos de CPU que se paga por su gestión.

En el ámbito científico es frecuente la utilización de múltiples procesadores para aplicaciones de cálculo numérico. Algunos ejemplos de los modelos numéricos más demandantes de potencia de proceso (y más conocidos) son las simulaciones de evolución del clima, simulación de explosiones atómicas, comportamiento de corrientes marinas, simulación de propagación de plagas, etc.

Cuando se realizan procesos en paralelo hay dos alternativas básicas en cuanto al manejo de la memoria: que sea una sola compartida por todos los procesadores o que cada procesador tenga su propia memoria.

Cada alternativa tiene sus ventajas y desventajas y debe ser evaluado este aspecto al decidir cómo paralelizar una aplicación a los efectos de sacar el mayor provecho de la arquitectura de múltiples procesadores.

La computadora XT5-HE de Cray es considerada actualmente (junio de 2010) como la computadora más potente del planeta; tiene 224.162 núcleos de seis procesadores cada uno, es gestionada con el sistema operativo Linux y tiene una velocidad máxima de 1759 teraflops. El flop es la medida de rendimiento y equivale a una operación de punto flotante por segundo. Un teraflop equivale a un billón de operaciones de punto flotante por segundo.

3.4 Interfaz Hombre – Máquina

Cada sistema operativo tiene alguna forma de interactuar con el usuario, el conjunto de elementos utilizados para ello definen la interfaz del sistema operativo. Nos referimos aquí al protocolo (conjunto de reglas) que el usuario y la máquina deben usar para comunicarse. Por ejemplo: dónde estará el botón para cerrar una ventana, el que la minimiza, cuándo debe hacer “doble click” y cuándo “un click”, qué pasará si oprime el botón derecho, el aspecto formal de las ventanas (no detalles de color o grosor de líneas).

Un SO puede tener más de una interfaz, si bien no es lo más común. Se han hecho esfuerzos a nivel mundial para definir modelos de interfaz estándar que permitan utilizar varios sistemas operativos con la impresión

para el usuario de que son iguales, el resultado ha sido un modelo de interfaz denominado MOTIF, otro modelo es el CDE (Common Desktop Environment, Ambiente común de escritorio).

Debe quedar claro que Sistema Operativo e interfaz gráfica NO son la misma cosa; hay sistemas operativos que traen asociada una interfaz gráfica la cual no es posible cambiar en sus características más importantes (sólo en detalles poco significativos), pero otros permiten que el usuario elija con cuál desea trabajar (por ejemplo equipos Unix).

En el siguiente esquema se muestra como el usuario se vincula con la máquina a través de una serie de capas de software que le facilitan el control de la computadora sin tener que preocuparse de detalles menores. Siempre existe una "interfaz con el usuario", aún cuando ésta sea la simple pantalla de 24 líneas por 80 columnas el usuario accede a los comandos del Sistema Operativo e interactúa con sus aplicaciones mediante una interfaz.

En este esquema el teclado, el ratón y el monitor son sólo los medios físicos de la interfaz, no los consideramos parte de la misma pues nos referimos a los elementos del software que conforman el protocolo de comunicación.

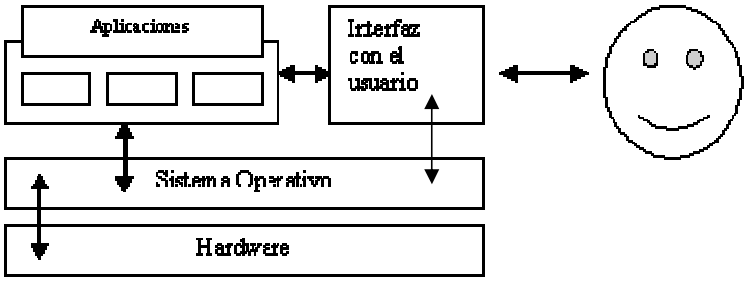


Ilustración 8: Esquema de la relación entre usuario, aplicaciones y SO

4 Herramientas de software

4.1 Introducción al concepto de herramientas de software

La informática da un entorno de trabajo a diferentes actividades y como tal ofrece las herramientas adecuadas para cada caso. Las hay de muy diversos tipos, en función de qué tan específicas a la aplicación final sean se pueden clasificar en:

Programas (software) de base

Programas (software) de aplicación

Los programas de aplicación a su vez se pueden clasificar como Herramientas de propósito general y Herramientas específicas.

4.1.1 Programas de base

Son aquellos que dan el entorno necesario para que el usuario desarrolle su trabajo. Podemos considerar como tales a los siguientes:

- Sistema Operativo
- Conectividad con redes
- Interfaz gráfica

4.1.2 Programas de aplicación

Aplicaciones específicas son aquellas que resuelven temas particulares del usuario. Son programas con un propósito determinado que pueden ser hechos por el propio usuario.

4.1.3 Programas de propósito general

- Editores de texto
- Procesadores de texto
- Planillas electrónicas
- Administrador del correo electrónico
- Agenda de tareas
- Navegador de Internet

4.1.4 Herramientas específicas

- Liquidación de sueldos y jornales
- Mantenimiento de stock de productos
- Gestión de cuentas corrientes de clientes
- Facturación
- Seguimiento de expedientes

4.1.5 Descripción de algunas herramientas de propósito general

4.1.5.1 Editores y procesadores de textos

Permiten ingresar texto a la computadora y modificarlo cuantas veces sea necesario. El texto puede ser esencialmente de dos tipos:

a) datos para ser leídos y procesados por un programa

El editor de textos en general permite el ingreso de información en modo “texto puro” - también conocido como "texto plano" o ASCII puro - compuesto exclusivamente por caracteres visibles por el usuario, se utiliza por ejemplo para ingresar datos a un programa. Algunos editores específicos para ciertos lenguajes realizan controles de la sintaxis adecuados al lenguaje. El resultado es un archivo de texto “plano” donde sólo se ven caracteres uniformes en tipo de letra y tamaño. Típicamente los únicos caracteres no visibles como tales en estos archivos son el “*fin de línea*”, “*retorno de carro*”, “*tabulador*” y “*fin de archivo*” de los que se ve su resultado.

b) datos documentales para ser leídos por otros individuos.

Estos documentos permiten que el resultado sea de fácil lectura, que destaque los ítems importantes de los secundarios y que permita la incorporación de gráficas, fotografías, etc.

Poseen una amplia gama de tipos de caracteres, tamaños y formas de presentación, capacidad de armar y mantener la estructura completa de un libro manteniendo la homogeneidad de capítulos, párrafos y sangrías, actualización automática del índice, mantenimiento de la numeración de ilustraciones, corrección automática de ortografía y reglas sintácticas en varios idiomas, etc.

Pueden generar documentos en diferentes formatos aparte del propio. Los más comunes son el formato de texto plano, en este caso se pierde prácticamente todo formato quedando sólo el texto y los correspondientes saltos de línea, formato HTML que luego puede ser visualizado con un Navegador de Internet, RTF (Rich Text Format) y PDF (Portable Document File)

El sistema operativo Windows trae incorporado uno de cada tipo, el notepad es un editor de texto que sirve por ejemplo para programar o para generar archivos de datos y el wordpad que sirve para generar documentos sencillos.

Se pueden invocar desde Inicio ➔ ejecutar ➔ Notepad (ó Wordpad).

4.2 Lenguajes de programación

Un lenguaje, definido en forma genérica, es un conjunto de símbolos y reglas que dicen cómo se construyen las estructuras del lenguaje con los símbolos del mismo.

Las computadoras tienen su propio lenguaje, conocido como lenguaje binario o de máquina, el cual es ejecutado directamente por los circuitos electrónicos que ellas poseen.

Cada arquitectura de máquina tiene pues un lenguaje de máquina propio, ya que cada empresa diseña los circuitos según su capacidad tecnológica y el destino final del producto que crea. Así por ejemplo el código máquina de un PC es diferente del que tienen máquinas como las Macintosh, IBM AS/400, IBM RS6000, etc. Incluso máquinas de un mismo fabricante tienen lenguajes de máquina diferente: IBM AS/400 e IBM RS6000 son dos ejemplos, la arquitectura del primero está especialmente diseñada para aplicaciones comerciales y el segundo tiene un diseño más orientado a cálculo intensivo.

Los PCs son un caso particular en cuanto que han evolucionado pero han mantenido siempre el mismo lenguaje de máquina básico.

Los *lenguajes* de máquina son sumamente engorrosos y primitivos, para utilizarlos hay que concentrarse mucho en el lenguaje y en la arquitectura de la máquina. Diseñar aplicaciones con un nivel de abstracción muy alto (por ej.: modelar el comportamiento de las mareas en el Río de la Plata, un sistema de cuentas corrientes, un sistema de pago de sueldos y jornales, etc.) es sumamente complejo, más allá de la aplicación misma, pues el programador pierde de vista el modelo al tener que concentrarse en aspectos muy menores de la implementación. Para resolver eso se crearon, y se siguen creando, lenguajes de programación que permitan generar aplicaciones de la forma más eficiente posible.

Un lenguaje de programación es entonces un conjunto de símbolos y reglas que permiten construir una secuencia de instrucciones que puede ser entendida y ejecutada por una computadora. Permiten ordenar a la computadora la ejecución de una cadena de instrucciones, tomar decisiones, manejar archivos y periféricos.

Se puede realizar una primera clasificación según las características de las aplicaciones que generan en:

Científicos Fortran, Basic, APL, Matlab

Comerciales Cobol, RPG, xBase (dBase, Clipper, FoxPro), Mantis, SQL.

Propósito general C, C++, Java. En este caso a la potencia de un lenguaje se le incorporan bibliotecas que resuelven los aspectos "científicos", los "comerciales" u otros cualesquiera

Cada uno ofrece ventajas para el trabajo en el área específica para la cual fue creado.

Existen otros de aplicación más específica y otros cuya aplicación específica está por encima de todas ellas, como por ejemplo los programas de generación de aplicaciones multimedia. En este caso nada tiene que ver la utilización final del material (que puede ser científica, comercial o artística) sino las características propias de los elementos a procesar.

Algunos elementos que componen un lenguaje son:

- Símbolos y reglas sobre su utilización
- Estructuras de control
- Bibliotecas
- Compilador y/o intérprete
- Encadenador (link editor) de bibliotecas

4.2.1 Compiladores

La definición vista anteriormente expresa un objetivo de los lenguajes de programación, pero no dice cómo es posible que la máquina (que sólo entiende lenguaje de máquina) puede entender lo que nosotros escribimos en un lenguaje de programación.

El compilador es un programa que toma de un archivo las sentencias generadas por el programador, verifica que se cumplan las reglas determinadas para el lenguaje y convierte todo a *lenguaje de máquina* llamado *código objeto* o *binario*. Éste código se genera dependiendo de la arquitectura y del sistema operativo que tenga la máquina.

El compilador es pues un programa **traductor** entre nuestro *lenguaje de programación* y el *lenguaje de máquina*. Habrá un compilador para cada combinación de lenguaje de programación y lenguaje de máquina. Dada la cantidad de tipos de máquinas diferentes, es posible que se limite la cantidad de versiones de un compilador a aquellas máquinas en las cuales hay mayor interés. Un caso particular lo constituye el lenguaje C (y su sucesor C++) que tienen compiladores para todas las máquinas. Ello se debe a varias razones, entre ellas destacamos el hecho de que mucho del software escrito en todo el mundo, especialmente sistemas operativos como el UNIX, están escritos en un muy alto porcentaje en C. Si hay un compilador C para una máquina es posible crearle un sistema operativo Unix con relativo poco esfuerzo.

El compilador entonces traduce nuestro programa. Pero aún así éste no es ejecutable aún. El código generado le dice a la máquina lo que nuestra aplicación hace, pero por ejemplo no resuelve la invocación a funciones que realizan tareas específicas (por ej.: tratamiento de señales de audio, de imágenes, de cálculo de estructuras, etc.) ni el acceso al monitor, al teclado o al disco. Esas tareas rutinarias y predefinidas están en bibliotecas en forma de módulos de código binario.

El encadenador (link editor) de bibliotecas ensambla nuestro código objeto con código que se encuentra en esas bibliotecas que resuelven tareas específicas. Siempre que se compila se debe ejecutar luego el encadenador, habitualmente se ejecuta automáticamente después del compilador si la compilación indica que no se detectaron errores en esa fase.

La encadenación de nuestro programa con todos los módulos por él invocado genera programas ejecutables que pueden tener tamaños importantes. Tanto como para que no puedan ser cargados en memoria para ser luego ejecutados. Es muy posible por otra parte que el usuario no requiera la ejecución de todos los módulos en una misma invocación.

Para hacer frente a esta problemática se han creado los módulos de encadenamiento dinámico (DLL: Dynamic Link Library - biblioteca de encadenamiento dinámico -) que son cargados a memoria y ejecutados siempre y cuando se les necesite, de lo contrario permanecen en disco. Esto permite tener un programa más chico en términos de Bytes en la memoria RAM pero que a medida que lo vamos ejecutando se "agranda" para cubrir todos los requerimientos. El efecto negativo: cuando invoquemos un módulo que está en disco deberemos esperar un tiempo extra mientras se carga a memoria RAM la primera vez; el efecto positivo: dejar libre memoria RAM para que tengamos más aplicaciones ejecutando simultáneamente.

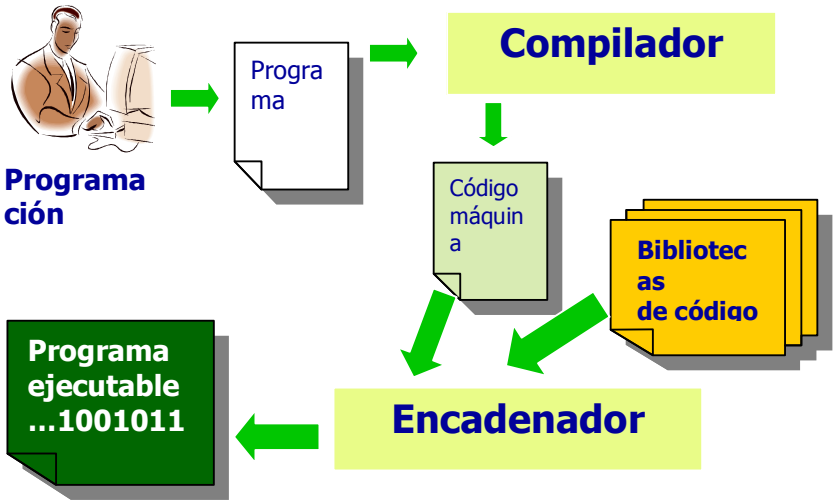


Ilustración 9: Ciclo de compilación de un programa

4.2.2 Intérpretes

Un intérprete es un programa que toma cada una de las sentencias del código fuente, analiza que sea correcta y si lo es interpreta (ordena la ejecución a la máquina) lo que allí se indica. Este proceso de análisis, interpretación y ejecución es más lento que la ejecución de código objeto.

Como ventaja más visible tenemos una mayor facilidad para el desarrollo del programa. Esto ha llevado a la existencia de “entornos de programación” que durante el desarrollo del programa permiten que se ejecute mediante un intérprete, y que antes de ponerlo en “producción” se compila.

Existen compiladores que en vez de generar código - máquina generan un código “intermedio” que es interpretado por un intérprete más eficiente (entre otras cosas porque las reglas del lenguaje -sintaxis- ya fueron analizadas). Algunos generan un ejecutable donde se incluye al propio intérprete, lo cual, a los efectos del usuario es como tener un único programa ejecutable en vez de tener por separado el programa y el intérprete.

Debe distinguirse aquí el entorno de desarrollo de aplicaciones del programa que estrictamente interpreta el código escrito por el programador. Ese programa es conocido como “Runtime” ya que es necesario en tiempo de ejecución. En general los proveedores de estos programas cobran por separado la herramienta de desarrollo de los “runtime” correspondientes.

Notas:

- Diferentes arquitecturas de procesador implican programas ejecutables diferentes
- Iguales arquitecturas de procesador con sistemas operativos diferentes también implican programas ejecutables diferentes.
- Los lenguajes que generan programas para ser ejecutados por un intérprete pueden obviar esas diferencias si tenemos el intérprete adecuado a la máquina.

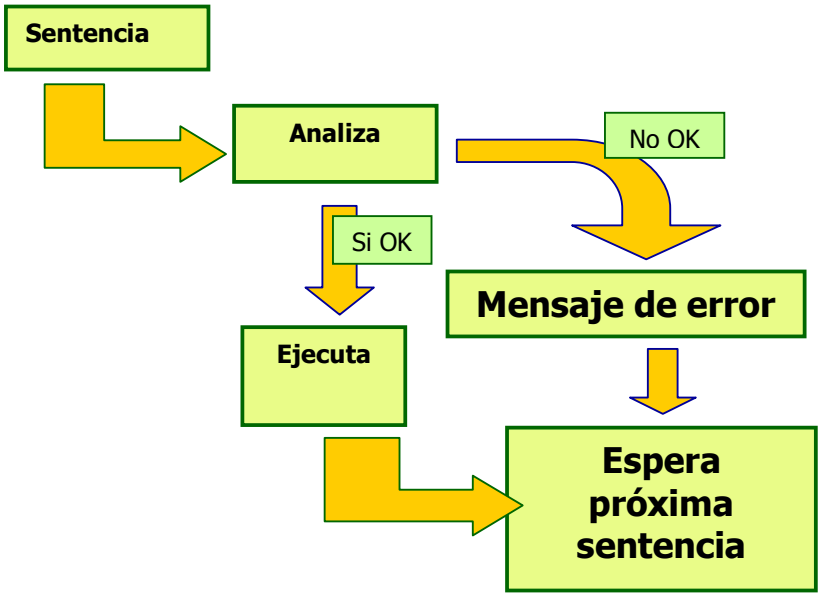


Ilustración 10: Ciclo de Interpretación de un programa

4.2.3 Los compiladores JIT

4.2.3.1 JAVA

Con el advenimiento de Internet se introdujo un lenguaje interpretado que se presentaba como una excelente solución para el desarrollo de aplicaciones multiplataformas (que pueden ejecutarse en diferentes combinaciones de hardware-sistema operativo). Ese lenguaje se llama Java y es interpretado por un runtime que se llama Máquina Virtual Java (JVM) por proveer un entorno con características de gestión de memoria y seguridad equivalentes a una máquina.

El que fuera interpretado despertó inmediatas críticas por ser notoriamente más lento que las aplicaciones compiladas. La solución que se dio fue instalar en la máquina virtual JAVA un compilador a lenguaje de máquina que funcionara de la siguiente manera:

Cuando se solicita la ejecución de un programa se verifica si existe una versión compilada, si existe se carga en memoria y se ejecuta, la velocidad de ejecución es la de un código compilado (de hecho es compilado) más algo de tiempo utilizado por la Máquina Virtual Java (JVM) para gestionar la ejecución. En caso de no estar compilado (es la primera vez que se solicita su ejecución) se compila, se guarda la versión compilada y luego se carga y ejecuta. En este caso el tiempo de ejecución es notoriamente mayor que la simple ejecución pero se da sólo en la primera ejecución.

Ese compilador se conoce como compilador JIT por “Just In Time” ya que su función es compilar en el momento que sea necesario.

La ventaja de este esquema es que quien distribuye la aplicación no tiene que preocuparse por la máquina en que se ejecutará. El usuario tampoco ya que si tiene la JVM instalada ésta se encarga de resolver la compilación y lograr así la mayor velocidad posible para la plataforma.

4.2.3.2 El modelo .NET de Microsoft

Microsoft sintió el impacto comercial que implicaba la tecnología Java y lanzó al mercado la tecnología de aplicaciones .NET.

En esta tecnología se reproduce en lo esencial el modelo Java y su JVM pero se lleva un poco más lejos. Se especifica un lenguaje interpretado de un nivel intermedio entre el lenguaje de programación y el de máquina

denominado MSIL (Microsoft Intermediate Language) y la correspondiente máquina virtual CRL (Common Languaje Runtime) para ejecutarlo; esa máquina virtual también tiene un compilador JIT que funciona igual que en el caso JAVA.

La diferencia más relevante desde el punto de vista de este curso es que en el caso de Java el lenguaje de programación es uno sólo (Java) mientras que en el caso de tecnologías .NET se puede programar en varios lenguajes. Luego éstos son compilados al lenguaje común intermedio y éste será a su vez el que compile el runtime.

5 Introducción a Matlab

5.1 Finalidad de Matlab

Es una herramienta creada específicamente para el desarrollo de modelos numéricos.

Esta simple definición dice mucho acerca de las características que se deben esperar de este producto:

- Capacidad de representar fácilmente (con cierta naturalidad) modelos numéricos
- Posibilidades de utilizar funciones matemáticas avanzadas y en lo posible optimizadas
- Flexibilidad para el desarrollo de un modelo
- Rigurosidad en las estructuras para evitar ambigüedades (en la ejecución y/o en la interpretación que una persona pudiera hacer)
- Capacidad de modularización y reutilización de módulos

Matlab cumple con estas premisas.

Su unidad de representación numérica son las matrices de números, en punto flotante doble precisión. Si trabajamos con números complejos, Matlab sabrá cómo tratarlos cuando se presenten.

Esto significa que a diferencia de un herramienta orientada a representar escalares, en los cuales hay que programar cualquier operación que se desee realizar con matrices (o recurrir a bibliotecas), Matlab permite en forma muy natural la especificación de esas operaciones tratando a los escalares como casos particulares de matrices de dimensiones 1 x 1.

Matlab incorpora por defecto una importante biblioteca de funciones matemáticas avanzadas, pero además su estructura permite que el usuario genere sus propias bibliotecas no habiendo diferencias al momento de utilizar una función propia de Matlab con la utilización de las que defina el usuario, extendiendo así las funcionalidades del lenguaje. De hecho la mayor parte de las bibliotecas de Matlab están programadas en Matlab.

Es posible adquirir bibliotecas específicas para tratamiento de imágenes, señales, audio, etc., las que una vez instaladas se usan como parte del lenguaje.

5.1.1 Entorno interactivo

Denominamos así al entorno que permite una interacción (diálogo) entre el usuario y el programa.

En este sentido Matlab tiene un entorno, que llamaremos consola, en el que el usuario puede ejecutar cualquier comando o función y obtener una respuesta inmediata de Matlab.

Al entorno interactivo se lo invoca ejecutando Matlab (o usando la interfaz gráfica correspondiente) y se sale de él con el comando quit.

En dicho entorno podemos solicitar ayuda con el comando *help*. Ejecutado sólo, muestra una lista de temas sobre los cuales el usuario puede solicitar ayuda poniendo *help tema*. En caso de necesitar ayuda específica sobre un capítulo o una función específica se pondrá *help función*.

Podemos utilizar este entorno para resolver cosas muy puntuales, pero no menos potentes, en general como parte de un proyecto mayor.

Podemos generar juegos de datos especiales:

- una matriz de números aleatorios (rand)

- una matriz identidad (eye)
- una matriz triangular inferior o superior a partir de una matriz dada (triu, tril)
- una matriz de Hilbert (hilb), etc.

En estos casos cuando decimos matriz estamos implícitamente extendiendo los conceptos a los vectores (fila o columna) y escalares, ya que como se explicó antes éstos son casos particulares de las matrices. Así por ejemplo rand(1,1) me devolverá un número aleatorio.

5.1.1.1 Formato de la información que se despliega

A los efectos de obtener un adecuado despliegue se puede recurrir al comando format. El mismo afecta sólo la forma en que se muestra la información, no así la manera en que se almacenan y opera con los datos.

El comando help format nos muestra las siguientes opciones:

```
» help format

 FORMAT Set output format.
    All computations in MATLAB are done in double precision.
    FORMAT may be used to switch between different output
    display formats as follows:
      FORMAT         Default. Same as SHORT.
      FORMAT SHORT   Scaled fixed point format with 5 digits.
      FORMAT LONG    Scaled fixed point format with 15 digits.
      FORMAT SHORT E Floating point format with 5 digits.
      FORMAT LONG E  Floating point format with 15 digits.
      FORMAT SHORT G Best of fixed or floating point format with 5 digits.
      FORMAT LONG G  Best of fixed or floating point format with 15 digits.
      FORMAT HEX     Hexadecimal format.
      FORMAT +       The symbols +, - and blank are printed
                     for positive, negative and zero elements.
                     Imaginary parts are ignored.
      FORMAT BANK    Fixed format for dollars and cents.
      FORMAT RAT     Approximation by ratio of small integers.

    Spacing:
      FORMAT COMPACT Suppress extra line-feeds.
      FORMAT LOOSE   Puts the extra line-feeds back in.
```

Se puede ejecutar en cualquier momento de la sesión, no afecta las variables ni las operaciones, sólo la forma en que se ve la información. De las opciones más usadas destacamos:

- `Long` despliega 16 dígitos, 14 decimales
- `Short` despliega 4 dígitos, 4 decimales
- `Compact` elimina saltos de líneas en blanco haciendo más compacta la presentación

5.1.1.2 Tipos de datos en Matlab

Esencialmente Matlab sólo reconoce números, pero permite operar como si tuviera variables lógicas y cadenas de caracteres (string).

Las variables lógicas valen 0 ó distinto (~=) de 0

Si = 0: se considera FALSO

Si ~= 0:se considera VERDADERO. En particular Matlab pone 1 cuando el resultado de una comparación es Verdadero (y se recomienda que el usuario utilice la misma convención para claridad de sus programas).

Las cadenas de caracteres son en realidad vectores (matrices de 1 fila) de números, donde cada casilla del vector contiene el valor ASCII del carácter correspondiente. Observe el siguiente ejemplo:

```
» a = 'hola'
a =
hola
» a = a + 1
a =
  105  112  109   98
» a
a =
  105  112  109   98
» a = char(a)
a =
ipmb
» a
a =
ipmb
```

El que sean vectores de números no es un detalle menor a la hora de ser pasados como parámetros al invocar funciones, uso que veremos más adelante.

5.1.1.3 Desplegar y aceptar información

Desplegar información en la consola:

Disp (' texto a desplegar ')

Error (' mensaje de error ')

Despliega el mensaje antecedido de varios signos "??????" e interrumpe la ejecución.

Variable = input (' texto a desplegar ')

Despliega el mensaje y queda esperando a que el usuario ingrese un dato válido.

; El ";" al final de una línea evita que Matlab despliegue el resultado de ejecutar el comando correspondiente.

Las funciones *disp*, *error* e *input* solamente deben utilizarse al trabajar directamente en la consola o en scripts. No se debe utilizar este tipo de funciones al implementar una función.

5.1.1.4 Diario de una sesión

Suele ser muy útil poder guardar los comandos (y sus resultados) de una sesión en un archivo a los efectos de poder referenciarlos más tarde. La forma de hacerlo es con el comando `diary` que graba todo lo que se despliega en la consola en un archivo. Si no se especifica nada el archivo se llamará `diary`, de lo contrario se puede ordenar "`diary pepe`" y generará un archivo pepe con el contenido de lo que se ejecute hasta que se cierre la sesión de matlab o se ordene "`diary off`."

5.1.1.5 Operaciones básicas

Tabla 3: Las operaciones aritméticas básicas

+	Suma	
-	Resta	
*****	Producto	
^	Potenciación	
'	Transpuesta	
****	División izquierda	por ej: A x = b ➔ x = A \ b
/	División derecha	por ej: x A = b ➔ x = b / A

.	Operación a coordenada	En caso de que una operación matricial se anteceda con un punto pasará a operar celda a celda

Como explicamos antes estas operaciones se realizan en forma matricial, por lo tanto cada una de ellas asume que los operandos son matrices que cumplen las condiciones necesarias para que la operación sea realizada, de no ser así se tendrá un mensaje de error. Pero entonces cuando deseamos operar con escalares hay que hacer algo específico para indicarlo.

Veamos el siguiente ejemplo:

```
» a = [1 2 3 4]
a =
     1     2     3     4
b = a
b =
     1     2     3     4
» a * b
??? Error using ==> *
Inner matrix dimensions must agree.
» b = a'
b =
     1
     2
     3
     4
» a*b
ans =
    30
» a.*b
??? Error using ==> .*
Matrix dimensions must agree.
» a.*b'
ans =
     1     4     9    16
» a^2
??? Error using ==> ^
Matrix must be square.
» a.^2
ans =
     1     4     9    16
```

Analice el alumno las respuestas y resultados.

5.1.1.6 Variables de entorno

Se puede saber qué variables están definidas, su dimensión y cuanto espacio ocupan. Esta información la brinda el comando whos y su salida tiene el siguiente aspecto:

```
» whos
  Name      Size          Bytes  Class

  a         1x3              24  double array
  ans       1x1              16  double array (complex)
  b         1x19            152  double array
  h         2x19            304  double array
  ii        1x1               8  double array

Grand total is 62 elements using 504 bytes
```

5.1.1.7 Operadores relacionales y operadores lógicos

Tabla 4: Operadores relacionales y lógicos

	Operadores relacionales
<	Menor que
< =	Menor o igual que
>	Mayor que
> =	Mayor o igual que
= =	Igual
~ =	No igual

	Operadores lógicos
&	Y
|	O (inclusivo)
~	No

5.1.1.8 El operador ":"

Sirve para indicar una sucesión de números.

Una forma es: valor_inicial : desplazamiento : valor_final

Se creará una sucesión de números iniciada en valor_inicial, al que se sumará sucesivamente el valor desplazamiento hasta alcanzar (o superar) el valor_final.

Una forma simplificada se da en el caso de que se quiera indicar que el desplazamiento es 1 (uno):

valor_inicial : valor_final

Esta forma es muy usada para indicar porciones de un vector o porciones de filas/columnas en una matriz. Un extremo de este último caso es poner sólo el símbolo **:**. En este caso se asume que se está indicando desde el primer elemento hasta el último.

5.1.1.9 Guardar y recuperar variables de trabajo

Se puede en cualquier momento guardar en un archivo las variables con que se está trabajando a los efectos de luego recuperarlas. El comando para hacerlo es

save fname v1 v2 ...

Se creará un archivo con nombre fname y extensión .mat (fname.mat) donde se almacenarán las variables v1 v2 . . .

Si no se especifican variables se guardan todas las variables que estén en uso en ese momento.

La recuperación se puede hacer con el comando `load fname`

Vea la sintaxis completa con el comando help save y help load.

Tabla 5: Valores especiales

Valor	Significado
`pi`	Número pi (3,14…)
`i, j`	Valor $\sqrt{-1}$ (unidad imaginaria)
`inf`	Infinito
`NaN`	Not a Number
`clock`	`Representa la hora actual. Si la fecha fuera las 23:04:44.9 del 7/11/2001 el resultado sería:` `1.0e+003 * [2.0010 0.0110 0.0070 0.0230 0.0040 0.0449]`
`date`	Despliega la fecha actual

eps	Precisión en punto flotante de la computadora que se está usando. Es el menor valor en que pueden diferir dos números. En un PC Eps = 2.2204e-016
ans	Es un valor calculado o generado por una expresión pero no almacenado en una variable específica.

5.1.2 Entorno de programación

5.1.2.1 Scripts

Un script (en muchos textos en español se denomina guión) es una colección de comandos matlab guardados en un archivo con extensión ".m".

Se ejecuta tal como si lo digitara el operador comando a comando y una vez que termina devuelve el control a la consola o al programa que lo llamó. Las variables que utiliza el script son las que tiene el entorno interactivo en el momento que es ejecutado y si define nuevas variables éstas serán luego visibles desde la consola.

Su utilidad es la de ejecutar un trozo de programa, eventualmente repetitivo, sacándolo del programa principal para que el código sea más claro.

Puede utilizarse para definir y/o encender las variables iniciales o el entorno (comando format).

Se debe tener especial cuidado en las variables que utiliza pues el script podría modificar inadvertidamente variables que no sabemos, con los consiguientes problemas. Una técnica para enfrentar con éxito esta situación es prefijar las variables definidas en cada script con el nombre del mismo (por ej. si el script se llama código_n, sus variables se llamarán código_n_variable1, etc.).

5.1.2.2 Funciones de usuario

Se denominan así a funciones que esencialmente son iguales a las provistas por Matlab, simplemente que las programa el usuario para solucionar sus problemas específicos.

Sobre las funciones descansa buena parte de la potencia de un lenguaje, pues de su habilidad para implementar este concepto dependerá la posibilidad de crear bibliotecas que satisfagan las necesidades del usuario y faciliten la construcción de programas.

Desde el punto de la programación una función se parecerá en mucho a un script: es una sucesión de comandos o sentencias en lenguaje Matlab. Pero su nombre proviene del hecho de que reviste características propias de las funciones matemáticas: dada una serie de valores de entrada que serán asignados a las variables de la función, obtendremos un resultado.

Su sintaxis en matlab es bastante sencilla, se pone al inicio de la función

Function objeto _ retorno = nombre _ función (parámetro_1, parámetro_2,…)

Function[parámetro _ salida_1,parámetro _ salida_2,…]=nombre _ función (parámetro_1, parámetro_2, …)

El objeto de retorno es el objeto en el que se devolverá el valor funcional. En el caso de Matlab no necesariamente es un número, puede ser un vector o una matriz.

El nombre de la función es aquel por el cual será invocada nuestra función. Más allá de detalles de sintaxis no debe coincidir con el nombre de funciones propias de Matlab.

Los parámetros son los valores a asignar a las variables de la función con los que se evaluará la misma.

Las funciones que provee Matlab están organizadas en directorios. El usuario puede crear sus directorios con funciones. Cada función se autodocumenta: los comentarios puestos al comienzo de una función hasta la primer sentencia ejecutable se considera como la ayuda de esa función y se desplegará con el comando help nombre_de_función. Se puede mostrar el contenido de un directorio con funciones Matlab con help nombre _ directorio. Esto nos permite rápidamente saber qué funciones componen esa "biblioteca". Por lo visto se

dispone de todas las herramientas para extender la funcionalidad de Matlab de forma que se integren nuestras aplicaciones a las del lenguaje.

5.1.2.3 Alcance de variables de una función

Concepto de variable local y global.

Matlab considera que cada función encapsula todas las variables que define y usa y que en ningún caso afecta las variables del entorno: ni del programa que la invocó ni de las funciones que a su vez invoque. Por eso se dice que el alcance de las variables es local al programa o función que la define.

Las nuevas versiones de Matlab permiten la definición de variables globales (help global). Por razones de claridad se acostumbra que las variables globales se pongan con todas sus letras en mayúsculas.

Estas variables pueden ser accedidas y modificadas por cualquier módulo que sea invocado directa o indirectamente desde el programa.

Ejemplo de script e invocación a funciones

ProgPri.m

```
% EVAL_FUN
% Evaluación de funciones
format compact;

Tolerancia = 10^(-6)

f_x = 'x * cos(x) - log(x)'

a = 10^(-15); b = 10; delta = .1;
% grafico la función
estado = graffun( f_x, a, delta, b);

% raíces en: [1,2]

disp('Raíz calculada por nuestro sistema de bipartición:')
a = 1; b = 2;
raiz = bipart( f_x, a, b, Tolerancia)
disp('La raíz de Matlab:')
%raiz_M = fzero( f_x, [a, b], Tolerancia, 0 );
raiz_M = fzero( f_x, a, Tolerancia);
raiz_M
```

Graffun.m

```
function estado = graffun( fun, a, deltax, b );
% método para graficar funciones
% Se recibe una función en el string fun.
% Se graficarán sus valores entre los puntos a y b con un
% deltax.

vx = []; vfx = [];
x = a; fa = eval( fun );
for x = a: deltax: b,
   vx = [ vx, x ];
   vfx = [ vfx, eval( fun ) ];
end
plot( vx, vfx )
grid on
estado = 1;
```

Bipart.m

```
function x0 = bipart( fun, a, b, Tolerancia );
% método de bipartición para funciones
% se recibe una función en el string fun a la cual se le
% buscará la raíz en el intervalo ( a, b)

iter = 0;
x = a; fa = eval( fun );
x = b; fb = eval( fun );
x = ( a + b ) / 2; fx = eval( fun );
if fa * fb > 0 % no hay cambio de signo ==> no hay raiz
   disp('La función tiene = signo en todo el intervalo')
   return;
end
while abs(x)/ (2^(iter)) > Tolerancia,
   iter = iter + 1;
   if fa * fx < 0,
      b = x; fb = fx;
   else
      a = x; fa = fx;
   end
   x = ( a + b)/2; fx = eval( fun );
end
x0 = x;
```

Resultado de la ejecución

```
» eval_fun

Tolerancia =

  1.0000e-006

f_x =

x * cos(x) - log(x)

Raíz calculada por nuestro sistema de bipartición:

raiz =

   1.3476

La raíz de Matlab:

raiz_M =

   1.3476
```

5.1.3 Algunas herramientas

Para una visión completa de las herramientas de Matlab sugerimos que se utilice el comando help en la consola de Matlab. Aquí mostramos algunas que entendemos pueden ser de interés inmediato para los alumnos.

5.1.3.1 Polinomios

Un polinomio es una función de una sola variable que se puede expresar en la siguiente forma general:

$$f(x) = a_0 x^N + a_1 x^{N-1} + a_3 x^{N-3} + a_2 x^{N-2} + \ldots + a_{N-2} x^2 + a_{N-1} x + a_N$$

Aquí decimos que la variable es x y los coeficientes se representan con los valores de a0, a1, etc. El grado de un polinomio es igual al valor más alto empleado como exponente cuyo coeficiente sea distinto de cero.

5.1.3.1.1 Evaluación de polinomios

Un polinomio en Matlab se implementa como un vector fila compuesto por los coeficientes (incluyendo los que valgan 0).

Para evaluarlo utilizamos la función polyval que evalúa ese vector reemplazando la variable x por el valor dado como segundo parámetro.

```
» a = [ 3 -0.5 0 1 -5.2 ]
a =
    3.0000   -0.5000         0    1.0000   -5.2000
» x = 2
x =
     2
» f = polyval(a, x)
f =
   40.8000
» x = 0:1:10
x =
     0     1     2     3     4     5     6     7     8     9    10
» f = polyval(a, x)
f =
  1.0e+004 *
  Columns 1 through 7
   -0.0005   -0.0002    0.0041    0.0227    0.0735    0.1812    0.3781
  Columns 8 through 11
    0.7033    1.2035    1.9322    2.9505
```

5.1.3.1.2 Operaciones

Si tenemos dos vectores, a y b, podemos realizar con ellos las operaciones de suma, resta, multiplicación por un escalar, multiplicarlos y dividirlos entre sí.

Para ello debemos tener en cuenta la forma en que Matlab representa los polinomios.

5.1.3.1.2.1 Suma de polinomios

Para las operaciones de suma y resta los dos polinomios deben tener igual cantidad de elementos, aún cuando por ser de diferente grado, el menor deba completarse con ceros.

Si tenemos los polinomios:

$g(x) = x^4 - 3x^2 - x + 2.4$

$f(x) = 4x^3 - 2x^2 + 5x - 16$

$s(x) = g + h$

en Matlab se representan de la siguiente manera

```
g =  [ 1 0 -3 -1 2.4 ]
h =  [ 0 4 -2  5 -16 ]
» g + h
ans =
    1.0000    4.0000   -5.0000    4.0000  -13.6000
```

La resta se plantea de igual modo

5.1.3.1.2.2 Producto de polinomios

El producto por un escalar es igual al producto de un vector por un escalar:

```
» 3*g
ans =
    3.0000         0   -9.0000   -3.0000    7.2000
```

El producto y el cociente de polinomios plantean una situación más complicada. Para resolverla Matlab provee funciones específicas.

Producto:

```
» prod = conv( h, g)
prod =
  Columns 1 through 7
         0    4.0000   -2.0000   -7.0000  -14.0000   -3.4000   38.2000
  Columns 8 through 9
   28.0000  -38.4000
```

5.1.3.1.2.3 Cociente de polinomios

```
» [ q, r ] = deconv(prod, g)
q =
   0   4  -2   5  -16
r =
   0   0   0   0   0   0   0   0   0
```

Donde q es el cociente y r el residuo. Si no se recibe el resultado en un vector de dos elementos sólo devuelve el vector correspondiente al cociente.

5.1.3.1.2.4 Raíces de polinomios

Existe una función para hallar las raíces de un polinomio

```
» roots(g)
ans =
   1.6508
  -1.2374 + 0.4829i
  -1.2374 - 0.4829i
   0.8240
```

Matlab provee de una función que implementa cierto algoritmo de cálculo que halla las raíces de polinomios con una precisión determinada, puede no ser la que le sirve al usuario. Éste deberá verificar si la precisión y el tiempo de cálculo se adecuan a su problema

También provee de la función inversa a la anterior que permite hallar un polinomio dadas las raíces que nos interesa que tenga. Como un polinomio de tales características posiblemente no sea único el usuario debe evaluar si las soluciones que obtiene son las convenientes.

```
% consideremos el polinomio de segundo grado [ 4 2 1 ] que tiene las
% mismas raíces que [ 1 0.5 0.25 ]
» poly( roots( [ 4 2 1 ] ) )
ans =
    1.0000    0.5000    0.2500
% Analizando el resultado de cada función
»  r1 = roots( [ 4 2 1 ] )
r1 =
  -0.2500 + 0.4330i
  -0.2500 - 0.4330i
» poly( r1 )
ans =
  1.0000    0.5000    0.2500
```

5.1.3.2 Gráficos en Matlab

(material generado a partir de textos de Shoichiro Nakamura)

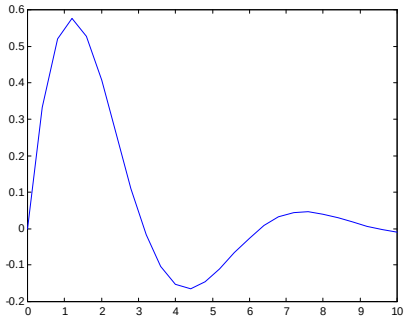
5.1.3.2.1 Gráficos en 2D -2 dimensiones-

Una gráfica 2D es esencialmente la visualización en el plano de pares de valores (x, y) donde y = f(x). En su forma más sencilla podemos entonces graficar con el comando

plot(x, y)

donde x es un vector con las coordenadas x e y el de coordenadas y.

Este comando generará una línea continua que unirá los diferentes puntos (x, y) del plano.




```
x = (0:0.4:10)';
y = sin(x).*exp(-0.4*x);
plot(x,y)
```

Se pueden especificar una marca específica para señalar los puntos

```
plot(x, y, 'marca')
```

Donde marca puede ser:

Tipo de marca	Símbolo
Punto	.
Más	+
Estrella	*
Círculo	o
Marca x	x

Una vez resuelto el problema inicial surgirán múltiples necesidades que tienen su solución en comandos y/o opciones de los mismos.

5.1.3.2.1.1 Desplegar más de una gráfica simultáneamente

Se puede usar el mismo comando plot ya que admite graficar tantos pares (x, y) como el usuario ingrese.

Así por ejemplo: plot(x1, y1, x2, y2, x3, y3) desplegará tres gráficas de líneas en el mismo sistema de coordenadas.

Para distinguir las tres líneas podemos añadir una marca distintiva a cada una:

plot(x1, y1, 'o', x2, y2, '+' , x3, y3, '*')

Otra forma sería distinguirlas utilizando trazos diferentes, que podemos elegir de la siguiente tabla:

Tipo de línea	Símbolo
Continua	-
Guiones	--
Punteada	:
Guiones y Puntos	-.

La forma de utilizar esta tabla sería:

```
Plot( x1, y1, '-', x2, y2, '- -' , x3, y3, ':')
```

Color de línea	Símbolo
Rojo	**r**
Amarillo	**y**
Magenta	**m**
Turquesa	**c**
Verde	**g**
Azul	**b**
Blanco	**w**
Negro	**k**

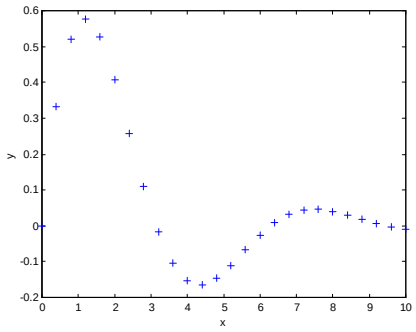
Una tercera forma de identificar las curvas es mediante colores. Matlab por defecto asigna un color diferente a cada curva que despliega, si el usuario desea controlar los colores puede hacerlo utilizando la siguiente tabla de colores:

El comando quedaría plot (x1, y1, 'r', x2, y2, 'y' , x3, y3, 'g')

Es posible combinar marcas y colores.

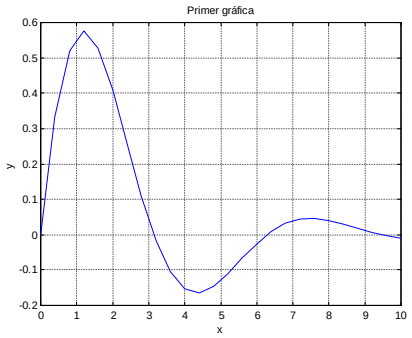
Si se indica que un gráfico tendrá marcas en los puntos la gráfica no tendrá una línea que una los puntos. Si deseamos que además de las marcas exista la línea debemos dar las correspondientes órdenes en forma consecutiva

```
x = (0:0.4:10)';
y = sin(x).*exp(-0.4*x);
plot(x,y,'+')
xlabel('x'); ylabel('y')
```

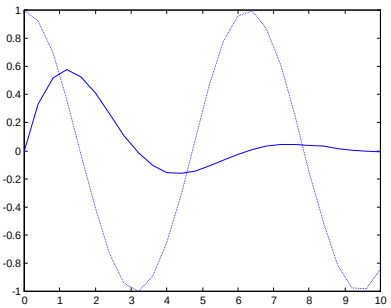


5.1.3.2.1.2 Títulos, etiquetas y grillas

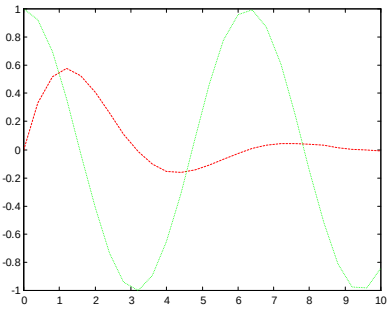
```
% Para generar el gráfico con una línea
% plot(x,y)
% al volver a hacer plot se borran los
% datos previos, así que vuelvo a poner
% las etiquetas de los ejes
xlabel('x'); ylabel('y')
% ponemos un título sobre el gráfico
title('Primer gráfica')
% hago dibujar una grilla
grid on
```




```
y2 = cos(x);
% represento la nueva gráfica con una línea
% punteada
plot(x,y2,':')
% vemos que el nuevo plot borró la gráfica
% anterior y los datos de las etiquetas
% fuerzo que se conserven los datos que ya
% están en la gráfica
hold on
% despliego la primer gráfica
plot(x,y)
```



```
% se puede evitar el uso del comando hold
% graficando las dos curvas con un mismo
% comando plot
hold off
plot(x,y,'--r',x,y2,':g')
```



5.1.3.2.1.3 Escala de los ejes

Si bien Matlab maneja en forma muy apropiada las escalas de los ejes se pueden manejar a voluntad con el comando axis.

5.1.3.2.1.4 Gráficas en escala logarítmica

En muchos casos es conveniente utilizar una escala logarítmica para representar la información. Para ello existen tres variantes cuyo comportamiento, en cuanto a opciones y sintaxis es idéntico a plot.

semilogx	usa una escala logarítmica en el eje x
semilogy	usa una escala logarítmica en el eje y
loglog	usa una escala logarítmica en ambos ejes

5.1.3.2.1.5 Otros comandos para gráficas

A continuación exponemos algunos comandos que suelen ser útiles en diversas oportunidades. Para obtener ayuda sobre los mismos recurra al comando help o a alguno de los textos indicados en la bibliografía.

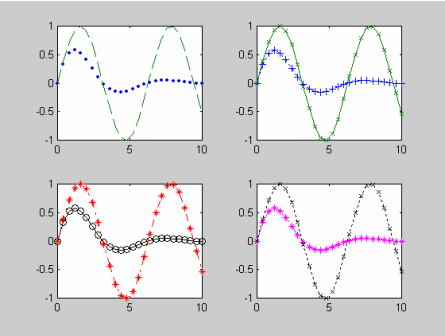
Comando	Descripción
`Clf`	Borra el contenido de la ventana de gráficos
`Cla`	Borra solamente las curvas
`Grid on / off`	Pone una grilla o la elimina
`Hold on /off`	Ordena que se conserve o no lo que está graficado
`Title`	Pone un título
`Text`	Permite poner un texto en un lugar especificado como x, y
`Ggtext`	Permite poner un texto en un lugar del gráfico que se indica con el puntero del ratón
`Plot3`	Genera gráficos en 3D. Son válidos en general todos los comandos y opciones vistos para 2D
`Axis`	Determina valores mínimos y máximos para cada eje. Es especialmente útil cuando se presentan varias curvas en la misma gráfica

5.1.3.2.1.6 Gráficos múltiples y otros

Es posible subdividir el espacio de gráficos para generar varias gráficas en el mismo, cada una de ellas con características propias.

Para ello se usa el comando Subplot (m, n, k) que divide la pantalla en m x n gráficas y usa el lugar k de la misma para hacer que trabaje el comando plot.

```
» subplot( 2, 2, 1)
» plot ( x, y, '.', x, y1, '--')
» subplot( 2, 2, 2)
» plot ( x, y, '+:', x, y1, 'x-')
» subplot( 2, 2, 3)
» plot ( x, y, 'o-k', x, y1, '*-.r')
» subplot( 2, 2, 4)
» plot ( x, y, '*--m', x, y1, 'x:k')
```



Aparte de los tipos de gráficos vistos, Matlab posee potentes comandos para la generación de gráficos en coordenadas polares, mallas, superficies con texturas y efectos de iluminación, etc.

5.2 Manipulación de archivos

Los archivos sirven para almacenar información en medios estables (discos, cintas magnéticas, disquetes, etc.). Al hacerlo es posible tener copias de los datos y eventualmente compartirlos con otras personas que talvez trabajan con herramientas (hardware y/o software) distintas a las nuestras.

Es en la utilidad de compartir datos donde adquiere relevancia la correcta utilización de los formatos de grabación de la información. Si grabamos algo sin saber cómo lo generamos, estamos dificultando la tarea de quien luego quiera usarlo.

Así por ejemplo si generamos un archivo con una matriz mediante el comando save de Matlab, está todo muy bien para quienes usan Matlab, pero quienes usen programas codificados en C, Fortran, etc. no tendrán la misma suerte.

Matlab prevé formas de grabar y leer archivos con formatos propios (save), pero además tiene comandos que permiten generar archivos con formatos más “personalizados”. A ellos nos dedicaremos ahora.

El comando save permite grabar datos que pudieran ser luego leídos por otros programas. Pero es muy rígido.

Es más adecuado a los efectos de intercambiar información usar el comando fprintf.

5.2.1.1 fprintf

El comando fprintf (por file print format) permite dar formato a la información a desplegar o grabar.

Si se usa en el entorno interactivo desplegará en pantalla la información con el formato indicado. Eventualmente puede tratarse de un texto explicativo.

fprintf(‘Ejemplo de despliegue en pantalla’)

El formato general del comando es

fprintf (id_archivo, formato, datos)

id_archivo es el identificador de archivo que devuelve el comando fopen

formato es el formato que se dará a los datos, puede incluir texto, constantes y el formato de los datos que tiene la misma sintaxis que el lenguaje C que son d, i, o, u, x, X, f, e, E, g, G, c, s. Además se pueden usar los caracteres de control \n,\r,\t,\b,\f para generar linefeed, carriage return, tab, backspace, y formfeed respectivamente.

datos una lista de los datos a los que se asignará el formato especificado.

5.2.1.2 fopen

El comando fopen permite abrir un archivo indicando algunas opciones. No es posible usar un archivo sin antes abrirlo. Un caso particular es la consola que puede verse como un archivo que se abre en el momento

de ejecutar Matlab.

fid = fopen(nombre_archivo, permisos)

fid es una variable en la que se guardará el identificador de archivo que será usado luego por los comandos fprintf , fscanf y fclose.

Nombre_archivo indica el nombre del archivo a procesar. Se especifica como una cadena de caracteres y debe respetar las particularidades del sistema operativo donde se use.

Permisos indica cómo se va a usar el archivo. La tabla de permisos es la siguiente:

```
        'r'     read
        'w'     write (create if necessary)
        'a'     append (create if necessary)
        'r+'    read and write (do not create)
        'w+'    truncate or create for read and write
        'a+'    read and append (create if necessary)
        'W'     write without automatic flushing
        'A'     append without automatic flushing
```

5.2.1.3 fscanf

Para leer un archivo con formato se usa el comando

fscanf (fid, format, size)

fid es el identificador de archivo devuelto por el comando open

format cumple las mismas reglas que el format de escritura y utiliza la misma sintaxis que el lenguaje C

size dice cuántos elementos (no bytes) se deben leer. Se puede indicar como:

- N N elementos
- Inf (infinito) indica que se lea hasta que se termine el archivo
- [M, N] indica que lo leído será almacenado en una matriz de dimensiones M x N

5.2.1.4 fclose

Los archivos una vez usados deben ser cerrados por los programas, de no hacerlo entre las cosas que pudieran pasar podemos citar el que el archivo no pueda ser usado luego por otro programa. En el caso de que el archivo se hubiera abierto con permisos de escritura y se ejecutaran comandos fprintf podría suceder que los últimos datos enviados a grabar no queden efectivamente en el archivo.

La sintaxis del comando es

```
fclose( fid )
```

Ejemplo 1

```
filas = 10;
columnas = 10;
A = ones( 10, 10);
fid = fopen('C:\tmp\datos.txt','w');
for i = 1:filas
   fprintf(fid,'%d ',A(i,:));
   fprintf(fid,' \n');
end
fclose( fid);
```

Ejemplo 2

```
x = 0:.1:1; y = [x; exp(x)];
fprintf('%6.2f  %12.8f\n',y);
  0.00    1.00000000
  0.10    1.10517092
  0.20    1.22140276
  0.30    1.34985881
  0.40    1.49182470
  0.50    1.64872127
  0.60    1.82211880
  0.70    2.01375271
  0.80    2.22554093
  0.90    2.45960311
  1.00    2.71828183
```

En este caso no se ejecutan open ni close pues se dirige la salida a la consola.

5.3 Vectores estructurales en Matlab

Hasta ahora hemos tratado con vectores y matrices (arreglos) en los cuales todos los elementos son definidos de igual tipo (enteros, reales, caracteres, etc.).

Los vectores estructurales permiten correlacionar vectores de distinto tipo. Sus componentes son estructuras.

Ejemplo: supongamos que queremos crear una base de datos de los alumnos del curso de Computación I que contenga la siguiente información de ellos:

- nombre y apellido
- documento de identificación civil
- dirección electrónica
- resultados de las pruebas parciales

Cada "ítem" de cada estudiante corresponde a un tipo de dato, es un campo (field) al que se le asigna un nombre de campo (field name). En el ejemplo tenemos cuatro campos: los tres primeros son de tipo texto (son datos cualitativos o categóricos) y el último está compuesto por números (datos cuantitativos) sobre los cuales se pueden establecer operaciones aritméticas.

Una estructura consiste en toda la información de un único alumno.

Un vector estructural (de aquí en adelante: vector-e) es un vector que contiene la información de varios alumnos.

Ejemplo: estudiante (vector-e de 1 fila y 2 columnas). Observar que toda la estructura de datos correspondiente a un estudiante se considera una única fila.

	Estudiante 1	**Estudiante 2**
Nombre	Juan Pérez	Juana Gómez
CI	1:999.999-0	2:111.111-9
e-mail	jperez@fing.edu.uy	jgomez@adinet.com.uy
resultados	25, 38	30, 29

5.3.1 Cómo crear vectores estructurales

Existen dos formas tal como con los vectores habituales: en forma implícita ingresando los datos y en forma explícita con la función struct definiendo sus características.

Los vectores estructurales usan el punto "." para indicar el campo del vector-e al que se quiere acceder, la sintaxis sería:

nombre_vector(indice).nombre_campo

5.3.1.1 Definición implícita

Para nuestro ejemplo crearemos el vector-e ingresando la información de cada uno de los estudiantes:

```
student.nombre    = 'Juan Perez';
student.CI        = '1:999.999-0';
student.email     = 'jperez@fing.edu.uy';
student.parciales = [25 38];
```

Si queremos visualizar la información ingresada, basta digitar el nombre del vector-e (student):

```
» student
student =
       nombre: 'Juan Perez'
           CI: '1:999.999-0'
        email: 'jperez@fing.edu.uy'
    parciales: [25 38]
```

Para agregar el segundo alumno debemos cambiar el índice del vector-e (en el primero el índice estaba implícito y era 1):

```
student(2).nombre    = 'Juana Gomez';
student(2).CI        = '2:111.111-9';
student(2).email     ='jgomez@adinet.com.uy';
student(2).parciales = [30 29];
```

De esta forma hemos extendido el vector-e a dos componentes. Matlab nos informa de la estructura de nuestro vector-e mediante los siguientes comandos:

```
» student
student =
1x2 struct array with fields:
    nombre
    CI
    email
    parciales
» size(student)
ans =
     1     2
```

Existe un comando específico para vectores-e:

```
» fieldnames(student)
ans =
    'nombre'
    'CI'
    'email'
    'parciales'
```

¿Qué pasa si se omite información para algún campo?

Matlab asigna a ese campo una matriz nula.

```
» student(3).nombre = 'Josefa Diaz';
» student(3).CI     = '2:001.111-8';
» student(3).parciales = [40 39];
» student(3)

ans =

       nombre: 'Josefa Diaz'
           CI: '2:001.111-8'
        email: []
    parciales: [40 39]
```

En el caso de no tener información dentro de un campo numérico podemos usar el NaN (not a number) que usábamos para vectores numéricos:

```
» student(3).parciales = [40 NaN];
» student(3).parciales

ans =

    40   NaN
```

Otra alternativa es que si Josefa Díaz no rindió el segundo parcial, directamente no se incluye esa información en el campo correspondiente. Debe tenerse esto en cuenta en el caso de tratamientos posteriores, por ej. si va a calcularse un promedio de puntos el NaN no puede procesarse como número.

```
» student(3).parciales = 40;
» student(3).parciales

ans =

    40
```

Si queremos volver al estado de valores anteriores podemos hacerlo utilizando el índice del campo correspondiente:

```
» student(3).parciales(2) = NaN;
» student(3).parciales

ans =

    40   NaN
```

Los datos de los vectores-e pueden usarse como lo hacíamos con los vectores habituales, así por ejemplo podemos asignar el contenido del campo de un vector-e a una variable cualquiera:

```
» variable_x = student(3).parciales(1)

variable_x =

    40
```

5.3.1.2 Definición explícita

La definición explícita de estructuras de datos se hace con el comando struct que permite definir la estructura y eventualmente asignar valores iniciales a esa estructura, que luego pueden ser cambiados, borrados o agregarse otros.

```
» s = struct('nombre', {'Juan Perez'; 'Juana Gomez'}, 'CI',{'1:999.999-
0';'2:111.111-8'},'email',{'';''},'parciales',{[25 38];[30 29]})
s =
2x1 struct array with fields:
    nombre
    CI
    email
    parciales
```

El campo email se inicializó en "nulo".

Una visualización de los datos para el primer estudiante nos da:

```
» s(1)
ans =
       nombre: 'Juan Perez'
           CI: '1:999.999-0'
        email: ''
    parciales: [25 38]
```

5.3.2 Formas de acceder a los datos contenidos en un vector-e

Para acceder a los datos de un campo de todos los estudiantes:

```
» s.nombre
ans =
Juan Perez

ans =
Juana Gomez
```

Para acceder a todos los campos de un estudiante:

```
» s(2)
ans =
       nombre: 'Juana Gomez'
           CI: '2:111.111-8'
        email: ''
    parciales: [30 29]
```

Para acceder a un campo de un estudiante en particular:

```
» s(2).nombre
ans =
Juana Gomez
```

Para cambiar el valor de un campo particular:

```
» s(2).CI = '3:000.001-8';
```

5.3.2.1 Los comandos setfield y getfield

Estos comandos permiten modificar y recuperar datos de los campos de una forma distinta a las vistas hasta el momento.

5.3.2.1.1 setfield

Permite poner datos en un campo cualquiera.

```
» s(1)
ans =
```

```
       nombre: 'Juan Perez'
           CI: '1:999.999-0'
        email: ''
    parciales: [25 38]
» s = setfield( s, {2},'parciales', [20 21]);
» s(2)
ans =
       nombre: 'Juana Gomez'
           CI: '3:000.001-8'
        email: ''
    parciales: [20 21]
```

5.3.2.1.2 getfield

El comando getfield permite recuperar el contenido de un campo de un vector-e.

```
» F = getfield( s, {2}, 'nombre')
F =
Juana Gomez
```

5.3.2.1.3 Utilidad de setfield y getfield

Si bien ya vimos que podíamos poner y recuperar datos en un vector-e en una forma "natural" para la sintaxis de Matlab, estos dos comandos vistos por último permitirían realizar estas operaciones considerando los nombres de los campos como instancias de variables.

Ejemplos posibles podrían ser:

```
» nombre_campo = 'nombre';
» for i= 1:2
F = getfield( s, {i}, nombre_campo)
end
F =
Juan Perez
F =
Juana Gomez
```

Observemos que el campo desplegado depende del contenido de la variable **nombre_campo**. Con esta capacidad podríamos hacer una función que dado un nombre de campo cualquiera imprima el contenido del vector-e para ese campo. La función sería única.

Operaciones con los campos definidos

Con variables cuantitativas o numéricas se pueden realizar las operaciones usuales:

```
» s(2).parciales
ans =
    30    29
» mean(s(2).parciales)
ans =
   29.5000
» [sum(s(1).parciales) sum(s(2).parciales)]
ans =
    63    59
```

Con variables categóricas o cualitativas se pueden aplicar sólo algunas funciones:

```
» sortrows([s(1).nombre  s(2).nombre])
ans =
Juan PerezJuana Gomez
» length(s(2).parciales)
ans =
```

```
     2
```

También se puede programar usando los campos como información para realizar tareas. Por ejemplo en caso de las pruebas de utilización de nombre de campo desde una variable:

```
» for i= 1:length( s )
F = getfield( s, {i}, nombre_campo)
end
F =
Juan Perez
F =
Juana Gomez
```

Otro caso de ejemplo:

```
» for i= 1:length(s(2).parciales)
v(i) = s(2).parciales(i);
end
» v
v =
    30    29
```

6 Metodologías de Programación

6.1 Resolución de problemas

El proceso de resolución de un problema por medio de una computadora se compone de tres pasos básicos:

- Análisis del problema y evaluación de potenciales soluciones para el mismo
- Encontrar un algoritmo que solucione el problema
- Codificación en un lenguaje de programación.

6.1.1 Determinación de un modelo del problema

En el análisis de un problema dado se debe recoger información del contexto (sistema) que permita comprender cuáles son los elementos importantes y cuáles los accesorios para tratar de generar una simplificación del sistema que sirva a los efectos de buscar soluciones a lo planteado. Este proceso permite generar un "modelo" del problema que es una simplificación conveniente del mismo. A ese modelo se tratará de buscar la solución y luego se verifica que las soluciones a los problemas del modelo son también convenientes al sistema real.

En particular los ingenieros trabajan sobre modelos matemáticos, algebraicos y analíticos. Así el análisis de la capacidad de generación de energía solar en el Uruguay se puede convertir en un modelo matemático donde se reflejen las principales características del clima, consumo, ángulo de incidencia del sol a lo largo del año, etc.

6.1.2 Análisis de soluciones posibles

Para analizar un problema y buscar posibles soluciones para el mismo, se aplican distintas técnicas y estrategias.

Una vez que se tiene claro el objetivo se diseña una estrategia para alcanzar el mismo. En especial la técnica conocida como *Divide y Reinarás* que se basa en dividir el problema en otros de menor grado de dificultad cuya resolución sea más sencilla y en lo posible conocida. Se aplica el mismo mecanismo a los sub-problemas mientras sea necesario hasta que se llega a sub-problemas con solución conocida.

Un algoritmo es un procedimiento detallado paso por paso para resolver un problema. Las instrucciones deben ser claras y sin ambigüedades y lo bastante específicas para ejecutarse y terminarse en un número finito de pasos.

La codificación es el proceso de traducir un algoritmo en un lenguaje de programación.

Una vez que se obtiene una codificación en un lenguaje (Programa Fuente) , según se programe en lenguajes compilados o interpretados:

Comienza el proceso de COMPILACIÓN / ENLACE / EJECUCIÓN a cargo de la computadora. El *Programa Fuente* se compila generando un Programa Objeto que al enlazarse con las bibliotecas genera un programa ejecutable listo para ser ejecutado con datos de entrada para dar como resultado datos de salida.

Se ejecuta a través del intérprete apropiado.

6.1.2.1 Algoritmo

Antes de conocer las tareas a realizar en cada fase, vamos a considerar el concepto y el significado de la palabra algoritmo.

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe **al-Khwārizmī**, nombre de un

matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.

Un algoritmo es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos. Pueden expresarse a través de fórmulas, diagramas, lenguaje natural, etc.

Un algoritmo debe presentar las siguientes características:

- Preciso: indicar el orden de realización de cada paso.
- Definido: si se sigue muchas veces, obtiene el mismo resultado cada vez.
- Finito: tiene fin, un número determinado de pasos.

Ejemplos de algoritmos son: instrucciones para andar en bicicleta, una receta de cocina, obtener el máximo común divisor de dos números, etc.

6.1.2.2 Algoritmo computacional

Informalmente se define como cualquier procedimiento computacional bien definido que toma valores como entrada y produce valores de salida. Son una secuencia de pasos que transforma esos datos de entrada en datos de salida.

Los algoritmos computacionales deben ser analizados para predecir los recursos que consumirán. Ocasionalmente memoria (especialmente RAM) y medios de comunicación, pero el recurso más importante que se debe medir es el tiempo que tomará en ejecutarse el programa que implemente dicho algoritmo.

Los análisis de algoritmos se concentran principalmente en determinar el peor de los casos para el cual se ejecutará el algoritmo, se indican principalmente dos razones para ello:

El saber el peor de los casos nos garantiza que nuestro algoritmo nunca tomará más tiempo del que ya sabemos.

Para algunos algoritmos, el peor de los casos ocurre más frecuentemente que el mejor de los casos, por ejemplo dentro de un manejador de base de datos.

Al tiempo estimado del algoritmo debemos considerar la forma en que se programe el mismo. El utilizar correctamente, o no, el lenguaje para programar un mismo algoritmo puede generar soluciones que tarden tiempos sustancialmente distintos.

6.2 Formas de representar un algoritmo

Un algoritmo se puede expresar de distintas formas, entre ellas se destacan:

- En forma gráfica, usando diagramas de flujo
- Codificado a través de un pseudo-código
- Codificado en un lenguaje de programación determinado

6.2.1 Diagrama de flujo de datos

Es una herramienta que representa gráficamente la secuencia lógica de los pasos de un proceso que implementa un algoritmo determinado. Se utilizan para la visualización de la estructura de un proceso. Esto nos facilita la visión descriptiva de la ejecución del programa.

Se utilizan símbolos geométricos bien definidos, para indicar las diferentes partes o acciones dentro del flujo de pasos de un programa. Sus principales elementos se pueden agrupar en:

Inicio y fin de un programa.

Operaciones de entrada y salida de datos (I/O), aritméticas y lógico-aritméticas.

Decisiones lógicas.

Flujo de la ejecución.

Los símbolos utilizados han sido normalizados por el Instituto Norteamericano de Normalización (ANSI por sus siglas en inglés) y los más frecuentemente empleados se muestran a continuación:

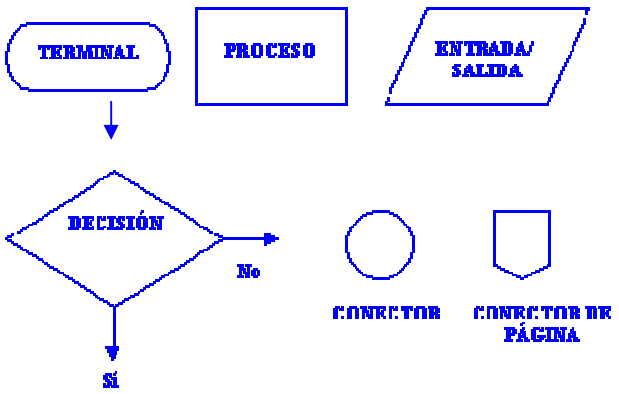


Ilustración 11: Principales elementos de un diagrama de flujo de datos

6.2.2 Pseudo-código

El pseudo-código es una herramienta en la que los pasos del algoritmo se escriben en palabras del lenguaje natural (en nuestro caso tomadas del inglés o español) facilitando así tanto la escritura como la lectura del mismo. En esencia, el pseudo-código se puede definir como un lenguaje de especificación de algoritmos, es un medio para expresar la lógica de un algoritmo.

A continuación se muestran algunos ejemplos de palabras para construir algoritmos en pseudo-código, es una convención que a los efectos del curso es conveniente. En cada caso se pueden adoptar convenciones que se adapten mejor a las circunstancias.

Tabla 6: Ejemplo de pseudo-código en español

Código	UTILIZACIÓN
ABRE	Abre un archivo
CIERRA	Cierra un archivo
CASO [ENOTROCASO]	Selección entre múltiples alternativas. “Enotrocaso” indica las acciones a realizar si no se cumple ninguno de los casos especificados
LEER	Leer un dato del teclado
ESCRIBE	Visualiza un dato en pantalla
HAZ	Inicia la iteración HAZ – HASTA
HASTA	Cierra la iteración HAZ – HASTA
MIENTRAS	Inicia la iteración mientras
PARA CADA	Inicia un número fijo de iteraciones
SI ENTONCES [SINO]	Selección SI - ENTONCES – SINO
INICIO	Inicia un bloque de instrucciones
FIN	Finaliza un bloque de instrucciones
NO	Niega la condición que le sigue
O	Disyunción lógica
Y	Conjunción lógica
{}	Inicio y fin de comentario
<=	Asignación

6.3 Objetivos de la programación

En la preparación de un programa, el programador puede tener que escoger entre soluciones alternativas en muchos puntos. Cada elección debe hacerse para satisfacer los objetivos y restricciones de la tarea de programación particular. Aquí asumiremos como apropiados para toda tarea de programación los siguientes objetivos:

- Exactitud
- Claridad
- Eficiencia

6.3.1 Exactitud

Un objetivo obvio en la escritura de cualquier programa de computador es que tiene que satisfacer su especificación exactamente. A menudo, a causa de la complejidad de la labor del programa, y de un entendimiento o cuidado inadecuados de parte del programador, un programa falla en satisfacer alguna parte de su especificación. Un programador tiene que ser en todo momento cuidadoso de la exactitud o pertinencia del programa para su propósito especificado.

Un factor clave en el logro de exactitud es la simplicidad. Escogiendo el algoritmo o técnica más simple disponible, es más probable que un programador pueda ver si satisface o no los requerimientos de la especificación del programa, y es menos probable que la describa incorrectamente en su programa. La innecesaria complejidad no cumple propósito alguno en la programación de computadores.

Algunos programas son, por supuesto, inherentemente complejos. Para tales programas, el programador debe adoptar un tratamiento sistemático que controle y limite la complejidad de la que tiene que ocuparse en cada etapa.

6.3.2 Claridad

Un programa es necesariamente tan complejo como el algoritmo que describe. Sin embargo, es importante que la forma en que el algoritmo esté descrito, por el texto del programa, no sea más complicada de lo que es necesario. La claridad del programa es una ayuda importante para el programador mismo en el diseño y limpieza del programa; y para otros que puedan tener que leer y alterar el programa en alguna etapa posterior.

La claridad del programa es lograda casi en la misma forma que para cualquier texto escrito, tal como un ensayo o un libro en los cuales se requiere:

a) Separación lógica del texto en partes comprensibles (capítulos, secciones, etc.) que reflejen la distinción entre los temas que describen, y su presentación en una secuencia lógica que refleje las relaciones entre ellas

b) Selección cuidadosa de las características del lenguaje, usadas en cada parte para expresar su sentido propuesto tan precisamente como sea posible

c) Selección cuidadosa de las palabras usadas para denotar los objetos y conceptos involucrados

d) La inclusión de comentarios y preámbulos para clarificar el texto principal cuando sea necesario;

e) Un aprovechamiento de los dispositivos para presentación de textos, tales como las líneas en blanco y la sangría para enfatizar la relación entre partes componentes de textos.

Un programador podría ser tan diligente en el uso de éstas técnicas para lograr claridad como el autor de cualquier texto. En muchos casos, la utilidad de un programa es determinada tanto por la claridad de su texto como por las cualidades del algoritmo que describe.

6.3.3 Eficiencia

El costo de ejecutar un programa de computador, es medido normalmente en términos de:

a) El tiempo tomado por el computador para llevar a cabo la secuencia de operaciones involucradas

b) La cantidad de memoria de computador usada en hacerlo.

En muchos ambientes de ejecución (en la máquina) el programa competirá con otros programas por el uso de esos recursos del computador, y es por lo tanto lógico minimizar los requerimientos de los mismos para cada uno.

El tiempo tomado para ejecutar el programa es directamente proporcional al número de operaciones que el procesador tiene que realizar al hacerlo. El programador debe, por tanto, escoger un algoritmo que minimice las operaciones implicadas, y tener cuidado de evitar cualquier operación redundante al expresar el algoritmo como un programa de computador.

La memoria usada por el programa durante su ejecución es determinada por la cantidad de datos que tienen que ser guardados y por el número de instrucciones del procesador requeridas para definir el programa, ya que éstas también tienen que ser guardadas en la memoria. Para minimizar el espacio de memoria usado por su programa, el programador debe considerar los datos manipulados y el número de operaciones especificadas por el programa.

Para algunos programas, o partes de programas, el uso eficiente del tiempo o del almacenamiento puede ser crítico; para otros puede serlo menos. El programador debe estar enterado de cualquiera de tales requerimientos de eficiencia cuando escriba su programa.

6.3.4 Expresiones lógicas (booleanas)

Las expresiones lógicas (booleanas) se usan para determinar si un conjunto de una o más expresiones es verdadero o falso y el resultado de su evaluación es un valor de verdad (verdadero o falso). Este tema es tratado en profundidad como "Álgebra de Boole". Los operandos de una expresión booleana pueden ser:

expresiones relacionales: que comparan dos valores y determinan si existe o no una cierta relación entre ellos

expresiones lógicas: que se relacionan mediante operadores lógicos

Las expresiones relacionales permiten determinar si una relación dada se verifica entre dos valores. La forma general de una expresión relacional es:

expresión-1 *operador-de-relación* expresión-2

donde:

- ***expresión-1*** es una expresión numérica, de cadena de caracteres o fecha
- **operador-de-relación** es uno de los siguientes:

<	Menor que
< =	Menor o igual que
>	Mayor que
> =	Mayor o igual que
=	Igual
~ =	No igual (distinto de)

- **expresión-2** es una expresión del mismo tipo que expresión-1

Si la comparación se realiza entre cadenas de caracteres la comparación es del tipo alfabética, donde el orden es (de menor a mayor):

- El carácter "espacio"

- Dígitos numéricos
- Caracteres alfabéticos en mayúsculas (conjunto básico)
- Caracteres alfabéticos en minúsculas (conjunto básico)
- Caracteres alfabéticos particulares de otros idiomas

Los caracteres especiales se intercalan entre todos los anteriores.

Para comprender mejor cómo se ordenan es conveniente observar la tabla de caracteres ASCII (7 bits) (página 2-11 - Tabla 1: codificación ASCII de 7 bits).

Cualquier expresión que devuelve verdadero o falso como valor es lo que llamamos una expresión lógica y puede combinarse con otras expresiones lógicas mediante operadores lógicos para formar nuevas expresiones lógicas.

Los operandos de una expresión lógica pueden combinarse mediante los operadores siguientes:

- **AND** (Y) Este operador produce el valor *Verdadero* si ambos operandos son *Verdadero*. Si cualquiera de los dos operandos es *Falso*, entonces el resultado será *Falso*.

Su correspondiente tabla de verdad es la siguiente:

Tabla 7: tabla de verdad del operador lógico AND

Expresión-1	Expresión-2	**AND**
V	**V**	**V**
V	**F**	**F**
F	**V**	**F**
F	**F**	**F**

- **OR** (O) Este operador realiza una operación O-inclusivo. El resultado es *Verdadero* si cualquiera de los dos operandos, o ambos son *Verdadero*. En caso contrario, es *Falso*.

Tabla 8: tabla de verdad del operador lógico OR

Expresión-1	Expresión-2	**OR**
V	**V**	**V**
V	**F**	**V**
F	**V**	**V**
F	**F**	**F**

El siguiente operador lógico tiene la particularidad de aplicarse a un único operando (es unario).

- **NOT** (NO) Este operador produce el valor *Verdadero*, si su operando es *Falso*, y el valor *Falso*, si su operando es *Verdadero*. El operador NOT siempre se aplica a la expresión lógica que le sigue.

Tabla 9: tabla de verdad del operador lógico NOT

Expresión	**NOT**
V	**F**
F	**V**

El orden de aplicación de los operadores lógicos cuando se encuentran varios de ellos en una expresión lógica compleja es (de mayor a menor) el siguiente:

NOT – AND – OR

A los efectos de facilitar la memorización es correcto asociar estos operadores a los operadores matemáticos - (signo), producto y suma según la siguiente tabla:

Tabla 10: relación entre operadores lógicos y matemáticos

Operador Lógico	Operador Matemático
NOT	- (signo)
AND	*
OR	+

De este modo la siguiente expresión no tiene ambigüedades de interpretación:

```
X AND NOT Y OR X AND Z AND Y
```

Una evaluación correcta puede ser:

- **NOT** Y
- X **AND** NOT Y
- X **AND** Z **AND** Y
- X AND NOT Y **OR** X AND Z AND Y

Trate el alumno de resolver la expresión propuesta asignando valores de Verdadero y/o Falso a cada una de las variables X, Y, Z y utilizando las tablas de verdad correspondientes.

6.4 Introducción al manejo de constantes y variables

Los programas de computación manejan datos que sometidos a un tratamiento, permiten obtener nuevos datos (información). Para que el ordenador obtenga datos, hay que conocer esos datos y saber donde están, es decir, hay que tenerlos diferenciados. Hay dos aspectos a tener en cuenta. Saber donde están los datos a manejar y diferenciar perfectamente unos de otros en la memoria.

Todo proceso maneja dos grandes grupos o tipos de datos: constantes y variables.

6.4.1 Constantes

Son aquellos datos que tienen un valor fijo y conocido de antemano por el ordenador. De esta manera para todo proceso que necesite una constante hay que definirla previamente y luego simplemente utilizarla. Las constantes pueden ser: números, letras, fechas.

6.4.2 Variables

Las variables son una posición de memoria a la que asignamos un nombre y en la que podremos almacenar y recuperar datos.

Las variables son aquellos datos cuyo contenido puede variar durante la ejecución del proceso. Las variables, no son conocidas por el ordenador, sino que cada programa tiene que definir las que necesite. Una variable es un trozo de memoria que se utiliza de forma particular.

Definir una variable significa indicar al ordenador en qué zona de la memoria se encuentra y qué tipo de dato va a contener, o sea que una variable tiene asociada una dirección, un nombre, un tipo, un valor y un alcance.

***dirección*:** Al declarar una variable se le reserva una localidad de memoria que permanece incambiada y sólo es conocida por la computadora.

***nombre*:** Se lo otorga el programador al declararla y permanece incambiado. Las variables se refieren por su nombre.

***tipo*:** Define el dominio y el tipo de operaciones que se pueden hacer sobre la variable. Se lo otorga el programador en el momento de declararla y permanece incambiado.

***valor*:** Se lo otorga el programador por medio de sentencias de asignación.

***alcance*:** Queda dado por el lugar de la declaración de la variable y define los segmentos del programa en donde la variable existe (puede ser utilizada).

6.4.2.1 Asignación

nomvar **:=** *expresión*;

La asignación permite cambiar el valor (almacenado en la localidad de memoria correspondiente) de una variable.

La operación significa colocar como nuevo valor (sustituyendo el anterior) de la variable identificada como nomvar el valor resultado de evaluar la expresión.

El tipo del valor resultado de la evaluación de la expresión debe ser del tipo declarado para la variable.

6.4.2.2 Estructuras de control

Son mecanismos para definir el orden en que se ejecutan las instrucciones de un programa. Estas permiten modificar el flujo de ejecución de las instrucciones de un programa. Se pueden clasificar en Secuenciales, de Selección y de Iteración.
En este material se tratará los ejemplos en notación Matlab dado que las estructuras se pueden encontrar en forma similar en el resto de los lenguajes computacionales.

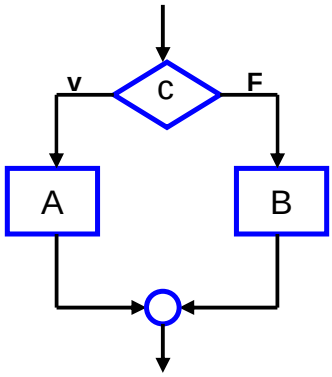
6.4.2.2.1 Secuencia

La secuencia es el mecanismo de estructuración más simple del que disponen los lenguajes de programación. Indica que las instrucciones de un programa se ejecutan una después de la otra, en el mismo orden en el cual aparecen en el programa. Se representa gráficamente como una caja después de otra, ambas con una sola entrada y una única salida.

Las cajas A y B pueden ser definidas para ejecutar desde una simple instrucción hasta un módulo o programa completo.

6.4.2.2.2 Selección

También conocida como la estructura SI-CIERTO-FALSO (también SI-ENTONCES-SINO), plantea la selección entre dos alternativas con base en el resultado de la evaluación de una condición o predicado; equivale a la instrucción IF de todos los lenguajes de programación y se representa gráficamente de la siguiente manera:



En el diagrama de flujo anterior, C es una condición que se evalúa; A es la acción que se ejecuta cuando la evaluación de este predicado resulta verdadera y B es la acción ejecutada cuando indica falso. La estructura de control que implementa esta idea es el IF-THEN-ELSE. A y/o B también pueden ser cualquier estructura básica o conjunto de estructuras.

- **Instrucción: IF-THEN-ELSE**

Idea: Si se cumple una determinada condición se ejecutará una cierta secuencia de instrucciones, pero si la condición no se cumple se ejecutará una secuencia de instrucciones diferente.

if *<<expresión booleana >>*
    instrucciones 1
else *instrucciones 2*

end

- Se evalúa la expresión lógica (booleana). Si el resultado es verdadero (TRUE) se ejecuta la *instrucción 1* (o secuencia de instrucciones). Si el resultado es falso (FALSE) se ejecuta la *instrucción 2* (o secuencia de instrucciones).
- El **ELSE** y su instrucción asociada es opcional, es decir, existe la variante **IF-THEN** que ejecuta una instrucción únicamente si la condición es cierta (TRUE).
- Las instrucciones IF – THEN - ELSE pueden anidarse, es decir, colocarse dentro de otras instrucciones IF - THEN - ELSE

if *<<expresión booleana 1>>*
    if*<<expresión booleana 2>>*
        instrucciones 1
    else *instrucciones 2*
    end
else *instrucciones 3*
end

La mayoría de los lenguajes actualmente en uso tienen una instrucción que indica el fin de la sentencia IF de forma que sea claro el punto a partir del cual el flujo de ejecución se retoma ya sea que la sentencia IF haya salido por Verdadero o Falso. Puede ser de la forma “END”, “ENDIF”, “END IF”, etc.

- **Instrucción: ELSEIF / CASE (SWITCH)**

En la mayoría de los lenguajes de programación modernos es posible implementar una estructura que permite seleccionar una de entre muchas opciones y ejecutar para ella un bloque específico. Tiene diferentes formas, aquí presentaremos las más conocidas.

ELSEIF

if *condición_1*

    instrucciones 1

elseif *condición_2*

    instrucciones 2

elseif *condición_3*

    instrucciones 3

else

    instrucciones 4

end

La cantidad de condiciones (ELSEIF) en general puede ser tal que a los efectos prácticos no se considera limitada.

Esta forma de plantear una serie de condiciones es una formalización del anidamiento de IFs que aporta a la claridad del programa.

SWICH CASE

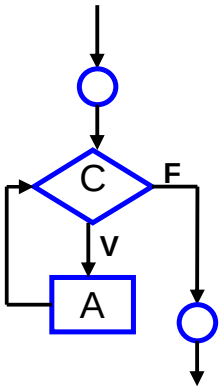
switch *variables*
 case *valor_1*
  instrucciones 1
 case *valor_2*
  instrucciones 2
 otherwise
  instrucciones 3
end

6.4.2.2.3 Iteración

Los cálculos más usuales implican la repetición de un cierto número de acciones. Por consiguiente todos los lenguajes de programación proporcionan estructuras de control que permiten al programador especificar bucles sobre un conjunto de instrucciones.

Instrucción: WHILE
Idea: **Mientras** se cumpla una cierta condición debe repetirse la ejecución de una cierta instrucción (o conjunto de instrucciones).

while *<<expresión booleana>>*
  instrucciones
end



Se evalúa la expresión booleana. En caso de que el resultado sea verdadero (TRUE) se ejecuta la instrucción. Se repite el proceso hasta que el valor de la expresión booleana sea falso (FALSE), momento en el cual el control pasará a la instrucción siguiente al WHILE.

Instrucción: FOR
Idea: Ejecutar una instrucción (o conjunto de instrucciones) un número exacto de veces que se conoce antes de comenzar la iteración.

for *<<variable de conteo>>* **=** *<<valor inicial>>* **:** *<<valor final>>*
  instrucciones
end

Al ejecutarse se asigna el valor inicial a la variable de conteo. Si el valor de la variable de conteo es menor o igual que el valor final, se ejecuta la instrucción, se sustituye el valor de la variable de conteo por su sucesor y se repite el control contra el valor final. En caso que el valor de la variable de conteo sea mayor que el valor final, se pasa el control a la instrucción siguiente al FOR.

- La variable de conteo de un ciclo FOR puede ser de cualquier tipo ordinal. Del mismo tipo deben ser los valores iniciales y finales.
- La variable del control del ciclo no se puede cambiar dentro del ciclo.
- Los valores iniciales y finales de la variable de conteo de un FOR se pueden modificar dentro del cuerpo de un ciclo FOR sin cambiar el número de veces que se ejecuta el ciclo.
- Después de la ejecución de una instrucción FOR, el valor de la variable de conteo queda indefinido.
- Toda instrucción FOR puede ser escrita con una estructura WHILE equivalente

6.5 Programación estructurada

Programación Estructurada (PE) es una técnica en la cual la estructura de un programa, esto es, la interpretación de sus partes se realiza tan claramente como es posible mediante el uso de tres estructuras lógicas de control:

- Secuencia: Sucesión simple de dos o más operaciones
- Selección: Bifurcación condicional de una o más operaciones
- Iteración: Repetición de una operación mientras se cumple una condición

Estos tres tipos de estructuras lógicas de control pueden ser combinados para producir programas que manejen cualquier tarea de procesamiento de información

Una característica importante en un programa estructurado es que puede ser leído en secuencia, desde el comienzo hasta el final sin perder la continuidad de la tarea que cumple el programa, lo contrario de lo que ocurre con otros estilos de programación.

Esto es importante debido a que, es mucho más fácil comprender completamente el trabajo que realiza una función determinada, si todas las instrucciones que influyen en su acción están físicamente contiguas y encerradas por un bloque.

La facilidad de lectura, de comienzo a fin, es una consecuencia de utilizar solamente tres estructuras de control

La PE tiene un teorema estructural, o teorema fundamental, el cual afirma que cualquier programa, no importa el tipo de trabajo que ejecute, puede ser elaborado utilizando únicamente las tres estructuras básicas (secuencia, selección, iteración).

6.5.1 Instrucción BREAK – Alteración leve a la Programación Estructurada

Se utiliza como una forma "racional" de alterar la secuencia natural de instrucciones en una iteración. Su efecto es interrumpir la secuencia y pasar a ejecutar la sentencia siguiente a la estructura de iteración en la cual se encuentra. En el contexto del curso de computación 1 no se debe usar esta instrucción.

6.5.2 Ventajas de la programación estructurada

Con la PE, elaborar programas de computador sigue siendo una labor que demanda esfuerzo, creatividad, habilidad y cuidado. Sin embargo, con este nuevo estilo podemos obtener las siguientes ventajas:

1. Los programas son más fáciles de entender. Un programa estructurado puede ser leído en secuencia, de arriba hacia abajo, sin necesidad de estar saltando de un sitio a otro en la lógica, lo cual es típico de otros estilos de programación. La estructura del programa es más clara puesto que las instrucciones están más ligadas o relacionadas entre sí, por lo que es más fácil comprender lo que hace cada función.

2. Reducción del esfuerzo en las pruebas. El programa se puede tener listo para producción normal en un tiempo menor del tradicional; por otro lado, el seguimiento de las fallas o depuración (debugging) se

facilita debido a la lógica más visible, de tal forma que los errores se pueden detectar y corregir más fácilmente.

3. Reducción de los costos de mantenimiento.

4. Programas más sencillos y más rápidos.

5. Aumento en la productividad del programador.

6. Los programas quedan mejor documentados internamente**.**

6.5.3 Caso de estudio

El siguiente ejemplo (tomado del curso de Programación I), si bien es sencillo permite visualizar algunos conceptos dados en clase.

Algoritmo

1. Pensar un número
2. Sumarle 3
3. Multiplicar el resultado del PASO 2 por el número 2
4. Restar 4 al resultado del PASO 3
5. Dividir el resultado del PASO 4 entre el número 2
6. Restar el número original del PASO 1 al resultado del PASO 5
7. Escribir el resultado del PASO 6

Posibles codificaciones (programaciones) para este algoritmo podrían ser:

Codificación 1

```
1. Paso_1 = 6
2. Paso_2 = Paso_1 + 3
3. Paso_3 = Paso_2 * 2
4. Paso_4 = Paso_3 – 4
5. Paso_5 = Paso_4 / 2
6. Paso_6 = Paso_5 – paso_1
7. imprimir Paso_6
```

Codificación 2

```
imprimir ( ( ( (Paso_1 + 3) * 2) – 4) / 2 ) – Paso_1 )
```

Codificación 3

```
1. Nro_pensado = 6
2. Resultado = Nro_pensado + 3
3. Resultado = Resultado * 2
4. Resultado = Resultado – 4
5. Resultado = Resultado / 2
6. Resultado = Resultado – Nro_pensado
7. imprimir Resultado
```

Las tres resuelven el mismo problema. Analizando cada una podemos observar que:

Codificación 1

- Es muy clara para entender qué hace cada paso.

- En cada etapa se utiliza una variable nueva lo que da tranquilidad de no estar sobrescribiendo datos. En programas muy extensos hay que cuidarse de que una variable no sea usada en otro lugar del programa.
- Derrocha memoria tal vez innecesariamente.

Codificación 2

- Es muy concisa. En una línea realiza todos los pasos propuestos por el algoritmo.
- No es fácil comprender lo que hace a simple vista si bien su notación matemática permite deducirlo sin dificultad.
- Utiliza el mínimo de variables por lo que utiliza muy poca memoria.

Codificación 3

- Usa muy poca memoria.
- Es clara en cuanto a lo que hace en cada paso

7 Recursión

7.1 Introducción

Se dice que un objeto o un procedimiento es recursivo si se define en función de sí mismo. El concepto de recursividad esta presente constantemente en nuestra vida diaria. A continuación enumeramos algunos ejemplos de ello:

1. La TV: Pensemos en la siguiente situación; un canal de TV pone al aire la imagen de un presentador de noticiario. Imaginemos que pasa si este individuo tiene a su lado un televisor y en el mismo se sintoniza la señal de este canal. La imagen del locutor con su televisor se repite tantas veces como la resolución lo permita.
   En este caso la imagen se puede definir en forma recursiva, esto es:

Fig. 3.1 A recursive picture.

Ilustración 12: Una imagen recursiva

Imagen de Canal X {
Imagen del locutor
Imagen de Canal X}

2. La estructura de nuestras familias es otro ejemplo. Si pensamos en la relación filial tenemos que una familia se define como:

Familia{
Hijos A1 . . . An
Familia del Padre
Familia de la Madre
}

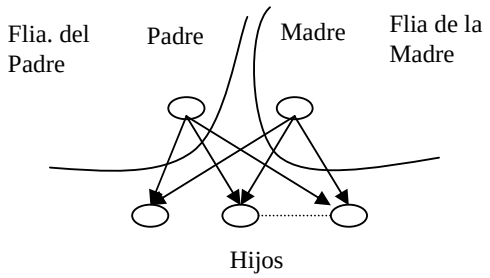


Ilustración 13: Un grafo recursivo

3. Los números naturales:
   - 0 (cero) es un número Natural
   - todo número sucesor de un Natural es un Natural

De esta forma vemos que una de las virtudes de la recursividad es la definición de un conjunto infinito de objetos mediante una proposición finita.

También encontramos ejemplos de procedimientos recursivos.

4. Cuando buscamos una ficha determinada dentro de un fichero, podemos pensar en la búsqueda de la siguiente manera :

   Extraemos una ficha

   Si es la ficha buscada finalizamos, sino buscamos en el resto del conjunto

```
Funcion Buscar [Ficha_buscada, Conj_fichas] → [V,F]
   si Conj_fichas == []
        Buscar = F
   sino
      Extraigo una ficha A de Conj_fichas
      si (A == Ficha_buscada) entonces
              Buscar = V
      sino
              Buscar = Buscar ( Ficha_buscada,{Conj-fichas – A})
      finsi
   finsi
fin
```

5. En matemáticas encontramos otros ejemplos de procedimientos recursivos como el factorial y la sumatoria:
   *fact(n) = n * fact(n-1)*

$$\sum_{t=1}^{n} 2^t = 2^1 + \sum_{t=2}^{n} 2^t$$

Vamos a hablar de ahora en más de que un programa recursivo P se compone de un conjunto de sentencias *S* y una invocación recursiva *P*

P = {S, P} (1)

De esta forma tenemos dos tipos de programas recursivos

- los directamente recursivos como (1)
- y los indirectamente recursivos:

P = {S1, Q}
Q = {S2, P}

7.1.1 La recursión como herramienta matemática

1. El concepto de recursión es una herramienta básica en matemática, un ejemplo de ello es la demostración por inducción completa.

Para probar que una propiedad M es cierta sobre el conjunto de los Naturales lo demostramos en 3 etapas:

Fase 1- Base de Recurrencia: demostrar que M es cierto para el 0 (esto se hace en base a las propiedades de los naturales)
Fase 2- Recurrencia: Se supone M es cierta para el natural ***n*** (hipótesis de recurrencia o hipótesis inductiva). Se demuestra que M es cierta para ***n+1*** (tesis).
Fase 3- Conclusión: M es válida para todo natural.

2. Asimismo, es utilizada para la definición recurrente de conjuntos (como el ejemplo de los números naturales visto en la introducción).

   Un conjunto puede ser definido de según dos técnicas:

   - Explícitamente, enumerando los elementos del conjunto

     Ej. {1, 2, 3, 4}

   - Implícitamente (por comprensión), enunciando una propiedad que caracteriza a los elementos del conjunto.

     Ej. {X / X es múltiplo de 2}

   - Por recurrencia

   La primera sólo es posible de ser usada en caso que el cardinal del conjunto es un número pequeño. En

este sentido, la definición por recurrencia de un conjunto tiene la característica de tener un enfoque constructivo del mismo. Es decir que definimos algunos elementos en el paso base y los nuevos son construidos en el paso inductivo utilizando los primeros.

Una definición recurrente de un conjunto consta de 3 partes:

Paso Base- se describen por extensión o comprensión los elementos del conjunto que se usan para generar el resto.

Recurrencia- Se indica mediante una formulación recurrente el mecanismo de creación de nuevos elementos a partir de elementos ya creados.

Conclusión- Indica que conforman el conjunto sólo aquellos elementos creados en el paso base y de recurrencia.

Un ejemplo de definición recursiva de un conjunto fue vista en la introducción para los números naturales.

Se puede ver de este ejemplo que la regla de recurrencia toma valores sobre el conjunto y devuelve valores del mismo conjunto. También es conocida como los constructores del conjunto.

7.1.1.1 Funciones recursivas

Como se ha visto anteriormente una función se define especificando su nombre, el nombre de los conjuntos iniciales - dominio y codominio (o imagen)- y el mapeo que esta realiza entre valores de ambos conjuntos.

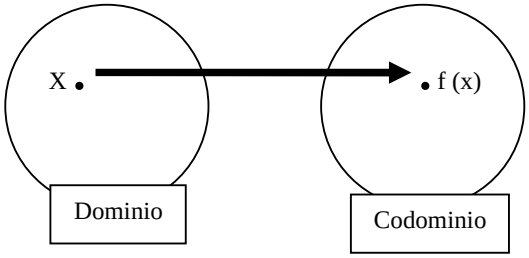


Ilustración 14: funciones recursivas como conjuntos

Esta definición se realiza mediante la creación de parejas (x,f(x)) la cual se puede efectuar por extensión (las parejas se ponen en una tabla) , por comprensión (mediante reglas) o por recurrencia. Estas últimas son las que nos van interesar de aquí en más. Veremos como una función se puede definir recurrentemente cuando el dominio ha sido definido también por recurrencia
Un ejemplo trivial es la función factorial.

- para la cual seleccionamos un nombre **fact**
- los conjuntos iniciales

  Dominio N^+
  Codominio N^+

la expresión para el factorial de n se define como
F(n) = 1 x 2 x..... x n

Si recordamos de la Introducción la definición recurrente del conjunto N, podemos escribir la definición recurrente de la función factorial de la siguiente manera:

paso base: la función factorial F para los valores que forman la base de N vale:

F(1) = 1

paso recursivo: queremos definir F(n + 1) en función de F(n). Según la definición de factorial sabemos que:

F(n + 1) = 1 x 2 . . . x n x (n + 1) = F(n) x (n + 1) por la relación de recurrencia.

Se puede ver que agregando como paso base F(0) = 1 se mantiene la relación de recurrencia.

La formulación recursiva en Matlab para este programa es:

```
% calcula el factorial en forma recursiva
% uso: res = fact_rec(n)
% con n>=0
function res = fact_rec(n)
if n == 0
   res = 1;
else
   res = n * fact_rec(n-1);   % llamada recursiva de la función
end
```

y la versión iterativa del mismo es:

```
% calcula el factorial en forma iterativa
% uso: res = fact_it(n)
% con n >= 0
function res = fact_it(n)
if  (n == 0)
     res = 1;
else
   res = 1;
   for  i = 2:n,
     res = res * i;
   end
end
```

o mejor aún:

```
% calcula el factorial usando operadores de matlab
% uso: res = fact(n)
% con n >= 0
function res = fact(n)
     res = prod([1:n])
end
```

En este punto vemos que tenemos dos componentes, la primera es la definición de la función y la segunda es la solución algorítmica, lo importante ahora a estudiar cómo es que se hace en la máquina el cálculo de una función de tipo recursivo. Veamos qué sucede al hacer la siguiente invocación en Matlab:

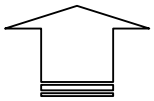
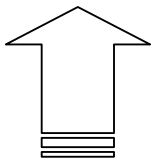
```
>>res=fact_res(3);
     6
```

En el llamado, como el valor de n = 3 se ejecuta la porción recursiva de la misma, es decir:

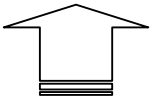
```
fact_rec(3)=3*fact_rec(2);
                  ||
                  2*fact_rec(1);
                         ||
                         1*fact_rec(0);
                                  ||
                                  fact_rec(0)=1    paso base

                    fact_rec(1)=1*1=1

              fact_rec(2)=2*1=2

fact_rec(3)=3*2=6
```

Siendo un poco más explícitos, un llamado a una función recursiva mantiene un conjunto de pilas en las cuales se conserva el valor de las variables y el lugar donde deberá retornar luego de finalizados los cálculos.

Ilustración 15: comportamiento de la pila durante la ejecución del programa para el ejemplo previo

N	función	La pila
1er paso		
3	3*fact(2)	
2º paso		
2	2*fact(1)	
3	3*fact(2)	
3er paso		
1	1* fact(0)	
2	2* fact(1)	
3	3* fact(2)	

En este paso la invocación a fact(0) no genera una nueva llamada recursiva y se resuelve en el paso base. Se podría considerar que se pasa a la etapa del desarmado de la recursión, en la cual los elementos que están en las pilas, variables locales y llamadas a funciones se empiezan a extraer de la misma.

N	Función	se calcula	La pila
4º paso			
2	2*fact(1)	fact(1) = 1*1 = 1	
3	3*fact(2)		
5º paso			
3	3*fact(2)	fact(2) = 2*1 = 2	
6º paso			
		fact(3) = 3*2*1 = 6	

Una conclusión que nos deja este ejemplo es que para cada "paso" de la resolución se ejecuta una nueva versión del mismo programa. Esto implica una nueva asignación (completa) de espacio en memoria. En ciertos problemas debemos considerar si la cantidad de memoria requerida en conjunción con la cantidad de veces que se espera sea llamada la función no provocará que el sistema operativo cancele la aplicación por no poder asignarle más memoria.

7.1.1.2 Ejemplo de recursión: los números de Fibonacci

No siempre es conveniente la utilización de una solución recursiva para resolver en un computador un problema cuya definición es recursiva. Un motivo lo vimos en el ejemplo anterior (problemas de memoria), en el siguiente veremos otro problema:

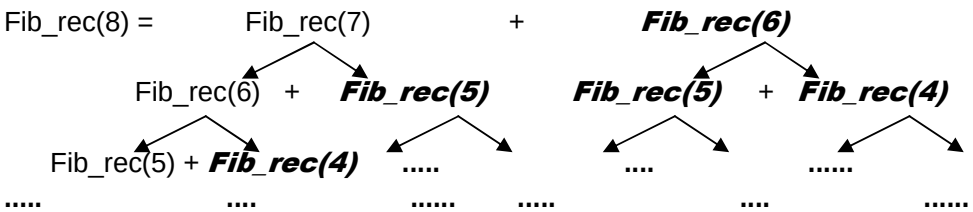
7.1.1.3 Fib: $N^+ => N^+$

```
Fib(1)=1
Fib(2)=2
Fib(N)=Fib(N-1) + Fib(N-2)
```

Tiene la siguiente solución recursiva:

```
Function y = Fib_rec(x)
if( x == 1 )
   y = 1
elseif ( x == 2 )
   y = 2
else
   y = fib_rec(x-1) + fib_rec(x-2)
end
```

Esta solución tiene gran redundancia de cálculos; en el siguiente esquema están marcados con letra ***negrita***:



La solución iterativa evita estos cálculos repetitivos, ya que guardamos en variables auxiliares el resultado de los cálculos anteriores:

```
Function y = Fib_it(x)
if(x == 1)
   y = 1
elseif (x == 2)
   y = 2
else
  x1 = 1;
  x2 = 2;
  for i = 3:x
       aux = x1 + x2;
       x1 = x2;
       x2 = aux;
   end
y = x2
end
```

7.1.1.4 Uso de recursión en cadenas (vectores)

La recursión es particularmente útil para el manejo de cadenas o vectores, produciendo soluciones compactas, fáciles de entender y sin la necesidad del uso de índices auxiliares.
Vamos a definir algunas funciones previas:

Nombre	Dominio	Especificación
Primer	Cadena ➔ elemento	Dada una cadena devuelve el 1er elemento de la cadena
Último	Cadena ➔ elemento	Dada una cadena devuelve el último elemento de la cadena
Resto	Cadena ➔ Cadena	Dada una cadena devuelve una cadena que contiene todos los elementos de la cadena de entrada menos el primero
Vacía	Cadena ➔ { V, F }	Dada una cadena indica si está vacía o no

```
Function y = vacia(c)
y =(c == []);
```

```
Function x = primer(c)
if vacia(c)
    x = []; % podría devolverse un valor que no forme parte de
            % los resultados posibles, de forma de indicar un
            % error. En este caso podría ser -1.
else
    x = c(1);
end
```

```
Function x = ultimo(c)
if vacia(c)
    x = []; % podría devolverse un valor que no forme parte de
            % los resultados posibles, de forma de indicar un
            % error. En este caso podría ser -1.
else
    x = c(lenght(c));
end
```

```
Function cn = resto(c)
if vacia(c)
    cn = [];
else
    cn = c(2:length(c));
end
```

Casos a resolver

a) Invertir una cadena tal que dada X = [A, B, C, D] obtener Y = [D, C, B, A]

```
Function y = reverse(x)
if vacia(x)
    y = [];
else
   y = [ ultimo(x) reverse(x(1:length(x)-1)) ]
end
```

b) Ver si una cadena es palíndrome (capicúa)

Ejemplos:

Palindrome ([1 2 1]) ➔ T

Palindrome ([1 2]) ➔ F

```
Function y = palindrome(c)
Y = ( c == reverse(c) )          %indirectamente recursiva
```

```
Function y = palindrome(c)
L = length(c);
if (l == 0 || l == 1)    % se puede poner tambien como
(vacia(c) || vacia(resto(c))
    val = 1;
else
   val = (primer(c) == ultimo(c) && palindrome(x(2:l-1)) )
end
```

c) Aplicar una función f a todos los elementos de un vector

Ejemplo: dado [1 2 3] devolver [f(1) f(2) f(3)]

```
function y = f_rec(func,x)
if vacia(x)
   y = [];
else
  y = [ feval(func,primer(x))  f_rec(func,resto(x)) ];
end
```

Esto arma un vector que se compone de aplicar f al primer elemento de X y de aplicar la función f al resto de X.

8 Sistemas de numeración

Se puede definir un sistema de numeración como el conjunto de símbolos y reglas que se utilizan para la representación de cantidades. Existe un elemento fundamental que caracteriza a todos los sistemas de numeración, y que se denomina base del sistema de numeración. Dicha base es el número de símbolos que se utilizan para realizar la representación.
Los sistemas de numeración se clasifican en dos clases: posicionales y no posicionales.

8.1 Sistemas no posicionales

En los sistemas de numeración, la representación de una cantidad siempre se efectúa mediante cadenas de símbolos. Si el significado de cada símbolo no depende del lugar que ocupa en la cadena, entonces dicho sistema de numeración recibe el nombre de *no posicional*.
Los números romanos son un sistema de numeración con algunas características de los sistemas no posicionales. La cadena XXXIII equivale al valor 33. Se observa que el símbolo X aparece tres veces en la cadena y todas ellas mantiene su valor: 10 unidades, sin importar el lugar que ocupa.

8.2 Sistemas posicionales

Si el significado de los símbolos varía en función de la posición que ocupen en la cadena, entonces dicho sistema de numeración recibe el nombre de *posicional*.
Los ejemplos más importantes de sistemas de numeración posicionales son el decimal o de base 10, creado por los árabes y utilizado por las culturas occidentales, y el binario o de base 2, que es el que utiliza el computador para representar la información y con el que es capaz de trabajar

8.2.1 Sistema decimal

Es el sistema habitual de numeración, su base es diez pues utiliza 10 dígitos diferentes (0, 1, 2, 3, 4, 5, 6, 7, 8 y 9)
Las cifras se nombran de la forma:

$$N \equiv n_r ..... n_4 n_3 n_2 n_1 n_0 \quad n_i \in (0,1,2,3,4,5,6,7,8,9)$$

y representan el número:

$$N = \sum_0^r n_i * 10^i = n_r * 10^r + ... + n_4 * 10^4 + n_3 * 10^3 + n_2 * 10^2 + n_1 * 10^1 + n_0 * 10^0$$

donde 10 es la base.
Ej: $1357 = 1 \times 10^3 + 3 \times 10^2 + 5 \times 10^1 + 7 \times 10^0$

8.2.2 Sistema binario

Este sistema usa únicamente dos cifras: el **0** y el **1**. Es el método habitual en los sistemas electrónicos digitales.
Las cifras se nombran de la forma:

$$\mathbf{B} \equiv b_r ..... b_4 b_3 b_2 b_1 b_0 \qquad b_i \in (0,1)$$

Conversión a decimal: $B = \sum_0^r b_i * 2^i = b_r * 2^r + ... + b_4 * 2^4 + b_3 * 2^3 + b_2 * 2^2 + b_1 * 2^1 + b_0 * 2^0$

donde 2 es la base.
Ej: $101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$ (decimal)

8.2.3 Sistema octal

Este sistema, utiliza 8 dígitos diferentes (0, 1, 2, 3, 4, 5, 6, 7) y su base es 8

Las cifras se nombran de la forma: $O \equiv o_r ..... o_4 o_o o_2 o_1 o_0 \qquad o_i \in (0,1,2,3,4,5,6,7)$

Conversión a decimal: $O = \sum_0^r o_i * 8^i = o_r * 8^r + ... + o_4 * 8^4 + o_3 * 8^3 + o_2 * 8^2 + o_1 * 8^1 + o_0 * 8^0$

8.2.4 Sistema Hexadecimal

Este sistema, utiliza 16 dígitos diferentes (0, 1, 2, 3, 4, 5, 6, 7 ,8, 9, 10, A, B, C, D, E, F) y su base es 16. Los diez primeros son los números decimales y tienen el mismo significado que en la numeración decimal. Los seis últimos son letras que representan: A=10, B=11, C=12, D=13, E=14 y F=15.

Las cifras se nombran de la forma:

$H \equiv h_r ..... h_4 h_3 h_2 h_1 h_0 \qquad h_i \in (0,1,2,3,4,5,6,7,8,9, A, B, C, D, E, F)$

Conversión a decimal:

$H = \sum_0^r h_i * 16^i = h_r * 16^r + ... + h_4 * 16^4 + h_3 * 16^3 + h_2 * 16^2 + h_1 * 16^1 + h_0 * 16^0$

E07F = $14x16^3 + 0x16^2 + 7x16^1 + 15 * 16^0 = 57471$ (decimal).

8.2.5 Conversión de base de numeración

8.2.5.1 De base b a base 10

Como se mostró en la definición de cada sistema de numeración, para convertir un número en base b a un número en base 10 basta multiplicar cada dígito por la potencia de b correspondiente a la posición que ocupa empezando por la derecha y se suman todos los resultados.

Ej.: El número 1101 en base 2 es $1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 = 13$

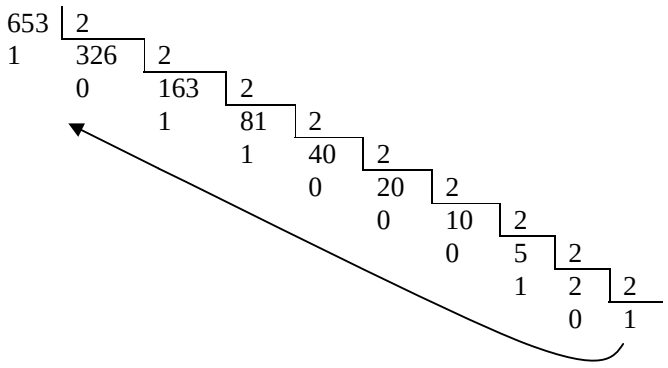
8.2.6 De base 10 a base b

Para llevar un número en base 10 a base b, se divide el número por b y se toma el resto. Se vuelve a dividir el cociente obtenido en la división anterior por b y se toma nuevamente el resto. Esta operación se repite hasta que el cociente resultante sea menor que b.

Por último se escribe, en este orden, el último cociente, el último resto, el penúltimo resto, el antepenúltimo resto,...

Ejemplo: Convertir 653 decimal a binario.

$653_{10} = 1010001101_2$



Caso particular: bases 8 y 16.

La base 8 (octal) y la base 16 (hexadecimal) tienen una íntima relación con la base 2. Puesto que $8 = 2^3$ cada dígito octal corresponde a 3 dígitos binarios. El procedimiento entonces para convertir un número binario en número octal es dividir en grupos de 3 bits a partir del punto binario y asignarle a cada grupo el dígito octal correspondiente.

Ejemplo: convertir 11001010011_2 a base 8

```
11  001  010  011 = 3123
 3    1    2    3
```

La conversión de base 8 a base 2 se hace a la inversa, convirtiendo en binario cada dígito octal, así:

732_8 es

$7_8 = 111_2$

$3_8 = 011_2 \Rightarrow 732_8 = 111011010_2$

$2_8 = 010_2$

El equivalente hexadecimal de un número binario se obtiene dividiendo al primero en grupos de 4 bits y asignando a cada grupo el dígito hexadecimal correspondiente.

Ejemplo:

```
1101 1011 1000 0110 = DB86
  D    B    8    6
```

La conversión de hexadecimal a binario se hace a la inversa convirtiendo cada dígito hexadecimal a binario.

Tabla 11: Equivalencias entre los sistemas numéricos decimal, binario, octal y hexadeximal

Decimal	Binario	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9

Decimal	Binario	Octal	Hexadecimal
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13

9 Representación interna de datos numéricos

9.1 Números enteros: representación en punto fijo

9.1.1 Enteros sin signo

9.1.1.1 Binarios puros

Los enteros sin signo, matemáticamente conocidos como Números Naturales, (siempre positivos) poseen la representación más simple: como números binarios. Se representan en binario con un número fijo de bits, con n bits tendremos 2^n números representables en el rango (0 a 2^n-1).
Los tamaños usuales para representar los enteros sin signo son:

- el byte (8 bits, 0 a 255)
- la palabra de 2 bytes (16 bits, 0 a (2^{16})-1)
- la palabra de 4 bytes (32 bits, 0 a (2^{32})-1).

Tipo	Máx. cantidad de nros representables
1 byte	255
2 bytes	65.535
4 bytes	4.294.967.295
8 bytes	18.446.744.073.709.551.615

9.1.1.2 Aritmética Binaria

Las operaciones elementales de este tipo son las cuatro usuales para los números enteros (**+ - * /**).

Tabla 12: Casos especiales en las operaciones aritméticas básicas

Suma	**Resta**	**Multiplicación**	**División**
0 + 0 = 0	0 - 0 = 0	0 x 0 = 0	0 / 0 = Error *
0 + 1 = 1	0 - 1 = 1 *	0 x 1 = 0	0 / 1 = 0
1 + 0 = 1	1 - 0 = 1	1 x 0 = 0	1 / 0 = Error *
1 + 1 = 0 *	1 - 1 = 0	1 x 1 = 1	1 / 1 = 1
* = con 1 de acarreo	* = con un préstamo de b		* la división por 0 no tiene sentido

En la suma de 2 enteros sin signo, se aplica el algoritmo usual para los números binarios como en el ejemplo de la izquierda:

```
Acarreos:      11
      25    11001
     +74  1001000
-------------------
      99  1100001
```

9.1.2 Enteros con signo

9.1.2.1 Valor absoluto y signo

En esta representación se utiliza el bit más a la izquierda como bit de signo y los restantes bits representan el valor absoluto del número en binario.

Msb lsb

b_{n-1}	b_{n-2}	...	b_1	b_0

b_{n-1} *= Signo. (0 es positivo, 1 es negativo)*
b_{n-2} *...* b_1 b_0 *= Valor absoluto*

Ejemplo: (en 4 bits) 0110 ⇒ 6
1110 ⇒ -6

Para n bits el rango del nro. representado es:

$$-(2^{n-1}-1) \le N \le 2^{n-1}-1$$

Ventajas:

- El cambio de signo es inmediato, se reduce a modificar un bit.
- El rango de representación es simétrico, tiene igual cantidad de números positivos que negativos.

Desventajas:

- Existen dos formas de representar el cero, 1000 y 0000 (para n=4).
- Las operaciones de suma y resta se complican al depender de los signos y las magnitudes.

9.1.2.1.1 Operaciones Aritméticas

Las operaciones no trabajan directamente con la representación sino que deben interpretarse en base a los signos relativos. El proceso requiere la comparación de los signos y las magnitudes para después realizar una suma o una resta.

Algoritmo de suma:

- Dados dos números binarios A B en representación valor absoluto y signo.
- Si los signos de A y B son iguales sumar las dos magnitudes. Asignar al resultado el signo en común.
- Si los signos de A y B son diferentes comparar las magnitudes
  - Si las dos magnitudes son distintas, restar la magnitud más pequeña a la más grande. Asignar al resultado el signo de la magnitud mayor.
  - Si las dos magnitudes son iguales restarlas.

A modo de ejemplo: (+25) + (-37) = - (-37) - (-25) = -12

La multiplicación y la división se tratan sin dificultad operándose por un lado con las magnitudes y por otro con los signos.

Existe la posibilidad de desbordamiento (overflow) en estas operaciones y se detecta cuando el resultado requiera n +1 bits siendo que la representación sólo utiliza n bits.

9.1.2.2 Complemento a uno

En esta representación los números positivos se representan en binario, y los números negativos se representan como el valor absoluto complementado bit a bit.

Para n bits el rango de la representación es: $-(2^{n-1}-1)...(2^{n-1}-1)$

Tabla 13: Ejemplo de Complemento a uno para n = 4

Nº	Representación	Representación	Nº
0	0000	1111	0
1	0001	1110	-1
2	0010	1101	-2
3	0011	1100	-3
4	0100	1011	-4
5	0101	1010	-5
6	0110.	1001	-6
7	0111	1000	-7

Ventajas:

- El cambio de signo se reduce al complemento lógico (cambiar ceros por unos y viceversa)
- El rango de representación es simétrico [$-2^{n-1}+1$, $2^{n-1}-1$]

Desventajas:

- El orden de los números en binario, no corresponde al orden de los números en base 10.
- Existen dos representaciones distintas para el cero (0000 y 1111).

9.1.2.2.1 Operaciones Aritméticas

La operación de suma es sencilla, no es necesario evaluar magnitudes y signo. Pero, debe tenerse en cuenta que si ocurre un acarreo este debe sumarse al dígitomás a la derecha del resultado.

```
Ej:        10            00001010
      +  (-3)            11111100
           7          1 00000110
Acarreo 1    00000110
                      1
             ----------
                0000111 ➔ 7
```

Existe la posibilidad de desbordamiento que deberá ser evaluada. Si los sumandos tienen signos opuestos nunca puede haber un error de desbordamiento. Si tienen el mismo signo y el resultado sale de signo opuesto, ha habido desbordamiento y el resultado es incorrecto. Habrá desbordamiento si y sólo si el acarreo al bit de signo es distinto del acarreo del bit de signo.

La multiplicación y la división son complicadas puesto que hay que considerar la posibilidad de que existan operandos complementados.

9.1.2.3 Exceso a "m"

En este sistema los números se incrementan en M y el resultado se representa luego en binario puro. El número X viene representado por X + M expresado en binario. Si se emplean n bits en la representación, se toma $M = 2^{n-1}$.

Por ejemplo para números de 8 bits el sistema se llama exceso a $128 = 2^{8-1}$ y los números se representan como su verdadero valor + 128 y en binario.

Representación de –3 con código de 8 bits: -3 ➔ $-3 + 128 = 125 = 01111101_2$

Así los números desde –128 a 127 se corresponden con los números desde 0 a 255, los cuales se pueden expresar como enteros de 8 bits

Ventajas:

- Existe una única representación para el cero.
- Conserva el orden de los números.

Desventajas

- No conserva la suma, (X1 + M) + (X2 + M) = (X1 + X2 + 2M)

9.1.2.4 Complemento a dos

En esta representación los números positivos se representan directamente en binario y para conseguir el código de los negativos, se complementa el valor absoluto y se los incrementa en uno. Así, negar un número pasa a ser un proceso de dos pasos. Primero cada 1 se reemplaza por un cero y cada 0 por un 1, como en el complemento a 1. Luego se le suma 1 al resultado.

Por ejemplo, para números de 8 bits se tiene:

Si 70 = 01000110, entonces para lograr -70 se hace el complemento a uno (negación bit a bit) de esta configuración y luego le sumo uno

```
01000110  (70)
10111001  (complemento a 1)
       +1
--------------------------
10111010  (-70)
```

Ventajas:

- Mantiene la suma (la suma con o sin signo es la misma operación, es decir que el algoritmo es el mismo).
  - A - B = A + (-B) = A + not B + 1
- Existe una única representación para el cero.

```
00000000 → 0
11111111 → not 0
       +1
----------------
00000000
```

representación (0) = representación (- 0)

- El cambio de signo es sencillo aunque ligeramente más complicado que en el complemento a 1: realizar el complemento lógico y añadir 1.

Desventajas

- El rango de representación es asimétrico [-2^{n-1}, $2^{n-1} - 1$]. Esto presenta el problema de que no se puede hacer el complemento de -2^{n-1} ya que daría el mismo código, lo que se supone que es un desbordamiento.

9.1.2.4.1 Operaciones Aritméticas

La suma y resta son más sencillas que con el complemento a 1: consisten en realizar la suma directa.

El procedimiento para sumar es muy simple y puede definirse como sigue: sumar los dos números incluyendo sus bits de signo y descartar cualquier acarreo de la posición de bits de signo (más a la izquierda)

Tabla 14: Ejemplos de complemento a dos

En todos los casos de la *Tabla 14: Ejemplos de complemento a dos* la operación que se realiza es una suma, incluyendo los bits de signo. Cualquier acarreo de la posición de bit de signo se descarta y los resultados negativos están de forma automática en complemento a 2.

+6	00000110	-6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111
+6	00000110	-6	11111010
-13	11110011	-13	11110011
-7	11111001	-19	11101101

Para restar dos números en complemento a 2 basta tomar el complemento a 2 del sustraendo (incluyendo el bit de signo) y se suma al minuendo (incluyendo el bit de signo). Se elimina el acarreo de la posición del bit de signo.

A - B = A + (-B) = A + not B + 1

Existe la posibilidad de desbordamiento en estas operaciones.

Sucede que si se trabaja en una representación de por ejemplo 8 bits e interesa sumar 70 + 80 = 150 se observara que la suma no es representable en 8 bits.

El desbordamiento no puede ocurrir en una suma de dos números con signos opuestos, puede ocurrir sólo si se suman dos números positivos o dos números negativos.

Puede detectarse el desbordamiento al observar el acarreo hacia la posición del bit de signo y el acarreo de la posición del bit de signo. Si estos acarreos no son iguales se ha producido desbordamiento

Tabla 15: Ejemplos de acarreo en Complemento a dos

Acarreos: 0 1			Acarreos: 1 0		
+ 70	0	1000110	- 70	1	0111010
+ 80	0	1010000	- 80	1	0110000
+ 150	1	0010110	- 150	0	1101010

En la multiplicación debemos escribir el algoritmo. Lo que se hace es multiplicar números positivos y luego colocarles el signo según las reglas algebraicas. El algoritmo que se usa es el mismo que en la escuela girando uno de los factores y luego sumar.

9.2 Números reales. Representación en punto flotante

La representación en punto flotante surge de la necesidad de representar números reales y enteros con un rango de representación mayor que el que nos ofrece el punto fijo.

Esta basada en la notación científica utilizada en física, química y matemática:

$$N = \pm f * b^{e}$$

Donde f se llama mantisa, e es un entero positivo o negativo denominado exponente y b es la base del sistema de numeración.

Ejemplo en base 10:

$3.14 = 0.314 * 10^{1} = 3.14 * 10^{0}$

$0.000001 = 0.1 * 10^{5} = 1.0 * 10^{-6}$

$1941 = 0.194 * 10^{4} = 1.941 * 10^{3}$

La representación en punto flotante es la versión para computadoras de esta notación, utilizando base 2. Solamente el signo s, la mantisaF y el exponente E se representan de manera física. De esta forma un número se representará utilizando n bits de la siguiente forma

s	*E*	*F*

donde: s es el bit de signo (0 positivo,1 negativo)

E es el exponente, representado con q bits en exceso a M ($M = 2^{q-1}$)

F es la mantisa, representada con p bits en binario.

1 + p + q = n (bits)

Dado que esta representación es ambigua (existen varias representaciones para un mismo número) se utiliza una versión más restringida que se llama ***normalizada***.

Se dice que un número de punto flotante esta ***normalizado*** si el dígito más significativo de la mantisa es diferente de cero o lo que es equivalente, que la mantisa sea máxima.

El número binario 00011010 no está normalizado dados los tres primeros dígitos 0 que contiene. Para normalizarlo se lo desplaza tres posiciones hacia la izquierda y se descartan así los primeros ceros para obtener 11010000. Los cambios realizados multiplicaron al número por 2^3, para mantener el mismo valor debe restársele 3 al exponente.

La mantisa de los números normalizados es de la forma: 1,F. donde el bit más significativo de la mantisa es un 1. Como todos los números normalizados tienen un uno en el bit más significativo se define una representación que omite este bit y sólo representa la porción después de la coma. Esta representación consiste en un 1 implícito, una coma implícita y luego la mantisa.

Ej: Representar $67 * 2^{-7}$ en punto flotante de 16 bits (signo, exponente de 5 bits y significante de 10 bits)

Mantisa: $67_{10} = 1000011_2$

Exponente: 5 bits, representación en exceso a M, $M = 2^{5-1} = 16$

$-7_{10} \rightarrow -7_{10} + 16_{10} = 9_{10} = 1001_2$

s = 0																
f = 0001000011.0																
e = 01001																
Representación:	0	0	1	1	1	1	0	0	0	0	1	1	0	0	0	0

s = 0		s = 0
f = 0001000011.0	Normalizando	F = 1.000011 Se mueve la coma 6 lugares pues se divide la mantisa por 2^6
e = 01001		E = 1001 + 110 = 1111 Sumamos 6 al exponente pues se multiplica por 2^6

Los números normalizados proporcionan la máxima precisión posible para los números de punto flotante. Un 0 no puede normalizarse porque no tiene un digito diferente de cero. Por lo general, se representa en punto flotante con únicamente ceros en la mantisa y en el exponente.

9.2.1 Estándar IEEE 754 de punto flotante

Existen infinitas formas de representar un número en punto flotante, ya sea por la cantidad de bits o por la representación (binarios puros, exceso a M, etc.) elegida para representar mantisa y exponente. Esto dificulta el intercambio de información entre distintas computadoras con arquitecturas diferentes.

Para que los números representados en punto flotante se puedan intercambiar entre distintas arquitecturas, se establece el estándar IEEE 754 que define el formato y las operaciones a utilizar con estos.

Tabla 16: Los tres formatos definidos por el estándar IEEE 754

	S (bits)	E (bits)	F (bits)	Total (bytes)
Simple precisión	1	8	23	4
doble precisión	1	11	52	8
precisión extendida	1	15	64	10

Estos tres formatos definen la cantidad de bits a utilizar en cada parte (mantisa, signo y exponente).

Los números se almacenan de la siguiente forma:

s	E	F

Luego el estándar define como se representará cada una de estas partes.

El exponente se representa utilizando exceso a M, donde la M se calcula como $2^{n-1} - 1$ (Utilizan un cálculo diferente del habitual, 2^{n-1}).

$M = 2^{8-1} - 1 = 127$ para simple precisión

$M = 2^{11-1} - 1 = 1023$ para doble precisión

La mantisa se representa como un binario puro.

El estándar también define que los números deberán estar normalizados.

Los números normalizados son de la forma: 1,F. donde el bit más significativo de la mantisa es un 1. Como todos los números normalizados tienen un 1 en el bit más significativo el estándar define una representación diferente que omite este bit.

Esta consiste en: un 1 implícito, una coma implícita y luego la mantisa.

Por lo tanto la representación a utilizar es de la forma:

$N = \pm(1,F) * 2^{e+127}$ para números de simple precisión

$N = \pm(1,F) * 2^{e+1023}$ para números de doble precisión

Surge un problema cuando el resultado de un cálculo tiene una magnitud menor que el número normalizado de punto flotante más pequeño que se puede representar en este sistema. Por esta razón se crean los números desnormalizados. Estos números tienen un exponente de cero y una mantisa dada por los siguientes 23 o 52 bits. El bit implícito a la izquierda se convierte ahora en cero. Se puede distinguir a los números normalizados de los desnormalizados porque los primeros no pueden tener un exponente cero

Los números desnormalizados sirven para operar con números cuyo modulo es menor que el modulo del menor número normalizado representable. Estos números asumen un 0 implícito en vez del 1 implícito de los números normalizados. Por lo tanto, cuando se tiene un número en notación de punto flotante desnormalizado estamos representando el número:

$n = \pm(0,F) * 2^{E-126}$ donde e es siempre cero = $\pm(0,F) * 2^{-126}$

Tabla 17: Algunos valores especiales en "punto flotante"

Número en Pto. Flotante	E (Exponente)	E (Mantisa)
Normalizados	0 < Exp < Max	Cualquier combinación de 1's y 0's
Desnormalizados	0000.....0	Cualquier combinación de 1's y 0's distinta de 0000.....0
Cero	0000.....0	0000.....0
Infinito	1111.....1	0000.....0
Not a Number	1111.....1	Cualquier combinación de 1's y 0's Distinta de 0000.....0

Algunos ejemplos utilizando simple precisión:

- Normalizados: $N = \pm(1,F) * 2^{e + 127}$
- Desnormalizados: $N = \pm(0,F) * 2^{E - 126} = \pm(0,F) * 2^{- 126}$

Tabla 18: Algunos valores especiales en "simple precisión"

s (Signo)	E (exponente)	E (Mantisa)	
0	00000000	00000000000000000000000	= 0
1	00000000	00000000000000000000000	= -0
0	11111111	00000000000000000000000	= Infinito
1	11111111	00000000000000000000000	= -Infinito
0	11111111	11001011100101100010010	= NaN
1	11111111	11010100101011101110011	= NaN
0	10000000	00000000000000000000000	$= +1\times 2^{128-127}\times 1.0$
1	10000001	10100000000000000000000	$= -1\times 2^{129-127}\times 1.101$
0	11111111	10110000000000000000000	= NaN
0	00000000	10000000000000000000000	$= +1\times 2^{-126}\times 0.1$

9.2.2 Aritmética de punto flotante

Las operaciones aritméticas con números de punto flotante son más complicadas que con números de punto fijo. Su ejecución requiere más tiempo y un hardware más complejo.

9.2.2.1 Sumas y Restas

Para sumar o restar dos números en punto flotante es necesario que los exponentes sean iguales.

La operación de suma o resta se realiza del siguiente modo:

- *Alinear las mantisas:*
  - Se desplaza hacia la derecha la mantisa que tiene el exponente más pequeño tantos lugares como la diferencia entre los exponentes.
- *Sumar o restar las mantisas.*
- *Normalizar el resultado*

9.2.2.2 Multiplicación

Para multiplicar dos números en punto flotante no es necesario alinear las mantisas.

La operación de multiplicar dos números expresados en punto flotante normalizados implica:

- *Sumar los exponentes.*
- *Multiplicar las mantisas.*
- *Normalizar el resultado*

9.2.2.3 División

Para dividir dos números en punto flotante no es necesario alinear las mantisas

Para llevar a cabo la división en punto flotante:

- *Dividir la mantisa del numerador por la mantisa del denominador.*
- *Restar los exponentes.*
- *Normalizar el resultado*

9.2.3 Errores de la representación en punto flotante

Para investigar las propiedades de este método de representación, consideremos una representación que utilice tres dígitos y signo para la mantisa, cuyo valor absoluto está comprendido entre $0.1 \leq |f| < 1$ o cero y un exponerte de dos dígitos y signo. También trabajaremos en base 10 para simplificar los cálculos.

Estos números varían en magnitud de + 0.100 * 10^{-99} a +0.999 * 10^{99} y se necesitan solamente cinco dígitos y dos signos para representarlos.

Mediante la representación en punto flotante se representan los números reales, aunque existen algunas diferencias importantes.

Dividamos la recta real en siete regiones.

1 Números negativos menores que –0.999 * 10^{99}

2 Números negativos entre –0.999 * 10^{99} y –0.100 * 10^{-99}

3 Números negativos entre –0.100 * 10^{-99} y 0

4 Cero

5 Números positivos entre 0 y 0.100 * 10^{-99}

6 Números positivos entre 0.100 * 10-99 y 0.999 * 10^{99}

7 Números positivos mayores que 0.999 * 10^{99}

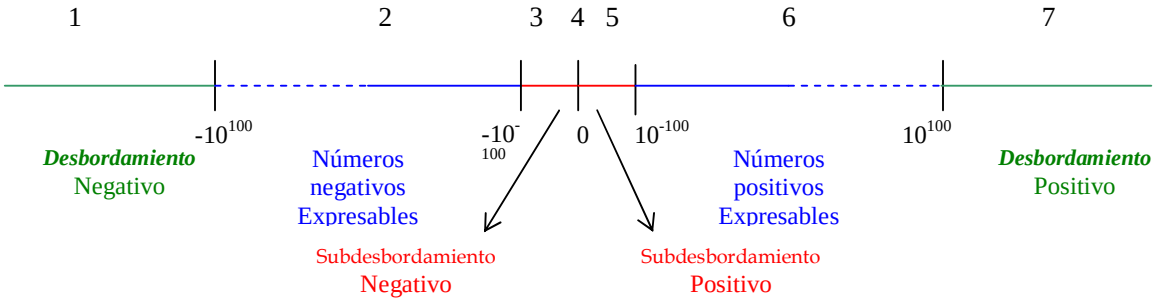


Ilustración 16: Representación de la recta de los números reales

Una diferencia importante entre el conjunto de los números representables en punto flotante por una mantisa de tres dígitos y un exponente de dos, y los números reales es que en los primeros no se puede representar ningún número en las regiones 1, 3, 5 o 7.

Si una operación aritmética diera como resultado un número en la región 1 o 7 se produciría un error de desbordamiento y el resultado sería incorrecto. La razón es la naturaleza finita de la representación. De manera similar no se puede representar ningún resultado de las zonas 3 o 5. Esto se llama error de subdesbordamiento (en inglés: underflow).

Otra diferencia importante entre los números reales y los números en punto flotante es su densidad. Mientras que los reales son densos, los números en punto flotante no lo son. Con la representación elegida se pueden representar exactamente 179000 números positivos, 179000 números negativos y el 0, dando un total de 358201. Es posible que el resultado de alguna operación no caiga dentro de estos números aunque sí pertenezca a la región 2 o 6. Si el resultado de una operación no se puede expresar en la representación elegida, se debe aproximar este resultado a un número representable. Las dos alternativas para aproximar el resultado se llaman redondeo y truncamiento.

En el primer caso se elige como aproximación el número más cercano expresable. En el caso de truncamiento se corta el número cuando se exceda de la cantidad de dígitos representables.

A modo de ejemplo:

Sea x = 2/3 = 0.6666666.... el número que se quiere representar

- Utilizando redondeo $\bar{x}$ = 0.667
- Utilizando truncamiento $\bar{x}$ = 0.666

Se observa entonces que al aproximar un número se comete un error. Llamaremos error absoluto a la diferencia entre el número que se quiere representar y el número efectivamente representado.

Error absoluto: $E_x = x - \bar{x}$

El espacio entre números adyacentes expresables no es constante a lo largo de las regiones 2 o 6.

La separación entre $0.998 * 10^{99}$ y $0.999 * 10^{99}$ es muchísimo mayor que la separación entre $+0.998 * 10^{2}$ y $0.999 * 10^{2}$. Por esta razón se define el error relativo debido a la aproximación.

Error relativo: $e_x = \dfrac{x - \bar{x}}{x}$

La explicación anterior se realizó en términos de una representación en particular y con un sistema en base 10, pero las observaciones son válidas para cualquier representación en punto flotante y con cualquier base.

10 Matrices Dispersas (Sparse matrix)

En muchas circunstancias el problema a resolver requiere formalmente de una, o varias, matrices pero la mayoría de los elementos de la misma son ceros. En esta circunstancia una gran cantidad de ceros están ocupando memoria. A los efectos de utilizar eficientemente un recurso tan costoso como la memoria (pensemos en matrices grandes) se puede almacenar la matriz en una estructura que ocupe menos espacio.

Esas matrices se denominan dispersas.

Existen técnicas para almacenar los datos distintos de cero y poder manipularlos en forma eficiente. Recuérdese que de por sí una matriz es una estructura sumamente eficiente tanto por el espacio de memoria que utiliza como por los tiempos de acceso a sus datos.

10.1 Tipos de formatos

Existen varias formas de representación: elemental o simple; comprimido por fila (CRS); comprimido por columna (CCS); comprimido por fila a bloque (BCRS),comprimido por diagonales (CDS), entre otros.

10.1.1 Formato elemental o simple

En esta técnica se generan 3 vectores en los cuales se almacenan respectivamente los valores (distintos de cero), las filas y las columnas.

Ejemplo

$$\begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 5 \\ 0 & 3 & 0 & 0 \end{pmatrix} \Rightarrow \begin{array}{l} D= [1\ 2\ 1\ 1\ 1\ 5\ 3] \\ F= [1\ 1\ 2\ 3\ 3\ 3\ 4] \\ C= [1\ 2\ 3\ 1\ 3\ 4\ 2] \end{array}$$

10.1.2 Formato CSR

En esta técnica se generan 3 vectores en los cuales se almacenan respectivamente los valores (distintos de cero), las columnas, y los índices de cada nueva fila.
Los vectores se generan de la siguiente forma: Un vector de punto flotante de tamaño *k* en el que se almacenan los valores de los coeficientes, siendo *k* la cantidad de coeficientes distintos de cero. Otro vector de tamaño *k* en el que se almacenan los números de columna de los elementos distintos de cero. Un vector de tamaño *n+1* siendo *n* la cantidad de filas de la matriz, en el cual se almacena la posición de la primera ocurrencia de cada fila.

Ejemplo

$$\begin{pmatrix} 1 & 0 & 0 & 4 \\ 0 & 2 & 0 & 3 \\ 7 & 0 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix} \Rightarrow \begin{array}{l} D= [1\ 4\ 2\ 3\ 7\ 6\ 5] \\ C= [1\ 4\ 2\ 4\ 1\ 3\ 4] \\ F= [1\ 3\ 5\ 7\ 8] \end{array}$$

De tal forma que F(i+1)- F(i)= numero de elementos no nulos de la fila i.

10.1.3 Formato CCS

Este formato es similar al anterior, se generan 3 vectores, valores (distintos de cero), filas, y los índices de cada nueva columna.
Esta es la forma en la que almacena Matlab.

10.2 Matrices dispersas en Matlab

Matlab tiene una función que permite crear matrices dispersas a partir de los vectores correspondientes, o generar la matriz dispersa a partir de la matriz completa. Este comando es **sparse.**

El comando help sparse de matlab expresa lo siguiente:

```
S = SPARSE(i,j,s,m,n,nzmax) uses the rows of [i,j,s] to generate an
    m-by-n sparse matrix with space allocated for nzmax nonzeros.  The
    two integer index vectors, i and j, and the real or complex entries
    vector, s, all have the same length, nnz, which is the number of
    nonzeros in the resulting sparse matrix S .  Any elements of s
    which have duplicate values of i and j are added together.
```

Ejemplo:

```
» a = eye(10)
a =
     1     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     0     1
» b = sparse( a )
b =
   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
   (6,6)        1
   (7,7)        1
   (8,8)        1
   (9,9)        1
  (10,10)       1
» whos
  Name      Size             Bytes  Class

  a        10x10               800  double array
  b        10x10               164  sparse array

Grand total is 110 elements using 964 bytes
```

Observe que la matriz b ocupa tan sólo 164 bytes frente a los 800 de la matriz a y que la cantidad de elementos del usuario son 110. En esa cantidad de elementos no figuran los que Matlab debe usar para mantener la estructura de la matriz dispersa, pero en los 164 bytes sí están considerados, ya que para los 10 datos son suficientes 80 bytes y se están usando 164.

En el caso de constuir una matriz dispersa como la reconoce Matlab la sintaxis es la siguiente:

S = sparse(i, j, s, m, n, nzmax)

Ejemplo

```
>> f=[1 2 3];
>> c=[1 2 3];
>> s=[1 1 1];
>> sparse(f,c,s)


ans =
   (1,1)        1
   (2,2)        1
   (3,3)        1
```

Donde i, j y s son los vectores que representan filas, columnas y datos respectivamente, m y n indican cantidad de filas y columnas que tendrán (si no se especifican se tomará m = max(i) y n = max(j); nzmax es la máxima cantidad de elementos distintos de ceros (NonZeroMax).

Todas las operaciones propias de Matlab (aritméticas, lógicas y de indexación) son aplicables en forma transparente a las matrices dispersas.

Las operaciones entre matrices dispersas dan matrices dispersas, las operaciones entre matrices completas dan matrices completas y en la mayoría de los casos las operaciones entre matrices dispersas y completas dan matrices completas excepto unos pocos casos donde de antemano se sabe que el resultado de la operación será una matriz dispersa.

Se puede saber si una matriz es dispersa con la función issparse(A) que devuelve 1 (TRUE) si es dispersa o 0 (FALSE) si no lo es.

El proceso inverso de convertir una matriz dispersa en una matriz completa se hace con el comando full.

A = full(S)

11 Sistemas de información

11.1 Breve cronología histórica

Esta sección tiene por finalidad mostrar brevemente la evolución de la informática, y sus orígenes como herramienta matemática y de automatización de procesos.

1833: El matemático inglés Charles **Babbage** diseñó una “máquina analítica” que pudiera realizar todas las operaciones matemáticas, programable mediante tarjetas de cartón perforado, Se ideó para la realización de tablas de logaritmos y funciones trigonométricas y nunca se construyó si bien su diseño fue utilizado varios años después para construir máquinas con la misma finalidad.

1936: El matemático inglés Alan M. **Turing** desarrolló la teoría de una máquina capaz de resolver todo tipo de problemas que tuviera una solución algorítmica. Construye en la teoría una "máquina de Turing", con ello se genera toda una disciplina matemática que trata de resolver algorítmicamente diferentes problemas matemáticos, basándose en que "un problema tiene solución algorítmica si existe una máquina de Turing capaz de representarlo".

1937: Howard H. **Aiken** de la Universidad de Harvard y su equipo junto con ingenieros de IBM desarrollan los conceptos de Babbage creando la primera computadora electromecánica. Se terminó de construir en 1944 y se conoció como **Mark-I**.

La ficha técnica de esta máquina:

Dimensiones : 16,6 por 2,6 metros

Peso : 70 toneladas

Componentes : 800.000 piezas móviles y 800.000 metros de cable

Capacidad de proceso: Suma de dos números: menos de un segundo, multiplicación de dos números: tres segundos.

1940: John Mauchly y John Presper Eckert junto con científicos de la Universidad de Pennsylvania crean la primera computadora electrónica conocida como **ENIAC**, la que entró en funcionamiento en 1945.

La ficha técnica de esta máquina:

Dimensiones : 111 metros cúbicos

Peso : 30 toneladas

Componentes : 17.468 válvulas de vacío, 50.000 conmutadores, 70.000 resistencias, 10.000 condensadores, 7.500 interruptores, 1.500 relés y un consumo de entre 100.000 y 200.000 vatios (suficiente para dejar a media luz a la ciudad de Filadelfia de entonces).

Capacidad de proceso: Suma de dos números: dos diezmilésimas de segundo, multiplicación de dos números: tres milésimas de segundo (era aproximadamente mil veces más rápida que la MARK-I).

1944: El ingeniero y matemático John **Von Neumann** desarrolla la idea de que el programa estuviera almacenado en la máquina del mismo modo que los datos y al mismo tiempo. Crea el modelo de Von Neumann de las computadoras de hoy.

1950-1960: Se inicia la fabricación en serie de las primeras computadoras.

1960-1970: A mediados de la década de los '60 se desarrolla la utilización de computadoras para aplicaciones comerciales. Eran grandes físicamente, caras y se utilizaban en la solución de problemas que justificaran los enormes costos de instalación y mantenimiento y que requerían muchas horas de cálculo e importantes volúmenes de información. Estaban destinados a la actividad militar, a grandes empresas

(bancos, aerolíneas, etc.) y actividad científica. Se les denomina **Mainframe** y ese nombre se aplica hoy a los equipos de mayor porte.

Entre las características de los Mainframe podemos citar que el desarrollo de software estaba limitado casi exclusivamente a las empresas que vendían los equipos, con lo que se tenían pocas posibilidades de elección y un encarecimiento de los costos de las instalaciones. Los Mainframes, desarrollados especialmente para aplicaciones comerciales, tienen características que los hacen especialmente aptos para esas tareas, en particular el manejo de grandes volúmenes de información y buenos tiempos de respuesta en las tareas de "tiempo real".

También se utilizan en aplicaciones científicas de gran porte. Para este último tipo de aplicaciones se han desarrollado otro tipo de Mainframe en los cuales se ha puesto énfasis en la potencia de cálculo; a esta línea de equipos se les denomina **Supercomputadoras** para diferenciarles y destacar justamente su carácter de equipo con gran capacidad de cálculo.

1970-1980: Surgen los **Mini computadores** como alternativas para empresas medianas. Básicamente eran equipos con iguales prestaciones/limitaciones que los mainframes pero con menor potencia y a precios considerablemente menores. Con equipos más económicos comienzan los primeros desarrollos de empresas de desarrollo de software, la oferta de éstos es más variada abarcando diferentes soluciones.

1980-1990: Surge el computador personal **(PC**) también conocido como **Micro computadora**. Originalmente visto como una herramienta muy cara para la pequeña empresa (por la relación costo/beneficio) y poco potente para la mediana empresa, los sucesivos avances tecnológicos cambiaron esto haciendo que los PC fueran lo suficientemente potentes y con mayor cantidad de herramientas disponibles como para dar mayores beneficios a las empresas a la vez que disminuían los costos de adquisición y mantenimiento.

Al contar con máquinas más potentes y económicas se desarrollan las redes locales dando soluciones económicas y muy flexibles que podían atender todas las áreas de una empresa, vinculando la información de forma que se pudiera tener acceso a todos los datos desde cualquier lugar físico. Esto antes era hecho con una computadora central y una red de terminales "tontas". La nueva propuesta es más económica y flexible por cuanto puede crecer en forma escalonada.

Se desarrollan varias líneas de "computadoras personales" con diferentes características. De ellas se destacan claramente dos: los PC "IBM compatibles" y los equipos de la línea "Apple". Los primeros, de costos sensiblemente inferior, se popularizan hasta tener la mayoría del mercado de equipos personales, los segundos, en base a una mayor capacidad de proceso, un sistema operativo de fácil utilización y mejoras notables en el tratamiento de imágenes se utiliza especialmente en ámbitos de diseño y actividad científica.

El desafío en esta década es compartir la información entre varias computadoras personales. Surgen las **redes de equipos PC**. Se intenta la interacción entre los ambientes de **Mainframe, Mini y PC's**; se busca el intercambio de información y la posibilidad de ejecución de procesos en más de una plataforma (interoperabilidad).

Se implementan las primeras aplicaciones comerciales en modo cliente-servidor.

1990-2000: En la década de los '90 se logra un importante nivel de **integración** de los diversos sistemas haciendo de la arquitectura **cliente-servidor** una herramienta fundamental. Surgen con mucha fuerza las herramientas para **trabajo en grupos a distancia**. Los costos bajan y las aplicaciones se hacen cada vez más sofisticadas aprovechando la disponibilidad de potencias de proceso cada vez mayores por menor costo.

Se difunden los **"sistemas abiertos"**, sistemas (hardware y/o software) que respetan estándares internacionalmente aceptados por las industrias. Con ellos es posible la interoperabilidad entre equipos de diferentes marcas.

Se habla de estaciones de trabajo para definir equipos de considerable potencia que resuelven todo lo que sea cálculo, utilizando la red para acceder a servicios específicos. El sistema operativo y la arquitectura del hardware dejan de ser tan relevantes.

Se comienza a utilizar Internet como medio de comunicación y comercial, permitiendo el intercambio de información de manera más rápida y a menor costo.

2000 en adelante: La gran disponibilidad de equipos, la capacidad de conectar "todo con todo" (computadoras, celulares, televisores), el aumento de ancho de banda y la reducción de costos, hacen que Internet se consolide como un medio de comunicación masivo global, revolucionando las comunicaciones e iniciando la llamada "era de la información".

11.2 Evolución del uso de la informática en la ingeniería (breve reseña)

La evolución tecnológica ha conllevado fuertes cambios en la forma de utilizar las computadoras.

Específicamente hablando de los ingenieros, en los inicios la elaboración de un programa era algo sumamente complejo y sólo se justificaba en casos de investigación o solución de problemas que por distintos motivos justificaran las enormes inversiones: enviar el hombre a la luna, programar la trayectoria de un misil, analizar el comportamiento de corrientes oceánicas o el clima a nivel continental son ejemplos en el área científica y militar.

Pocos ingenieros, aquellos que se vincularan con proyectos muy particulares, necesitarían programar las pocas computadoras de entonces. Además para ello bastaba aprender un lenguaje (de los pocos disponibles) y con él desarrollaba todos sus proyectos.

La difusión de las computadoras personales y estaciones de trabajo hace que hoy el ingeniero pueda recurrir a esta herramienta para afrontar situaciones muy diversas. Para ello dispone de una amplia gama de equipos, periféricos y programas que debe conocer a los efectos de utilizar el más apropiado en cada caso.

Es posible utilizar una computadora personal para todas las etapas de análisis y desarrollo de un proyecto obteniendo niveles de calidad y eficiencia muy buenos, imprescindibles para hacer frente a proyectos cada vez más ambiciosos.

Asimismo las computadoras estarán presentes en todos los ámbitos donde los ingenieros han de desarrollar su actividad: el diseño, estudio de factibilidad técnica y económica, análisis de recursos, control de la producción, análisis estadístico, de comportamiento de equipos, etc.

Obsérvese que no mencionamos una disciplina en particular pues todas están comprendidas.

11.3 Redes

11.3.1 Definiciones

Una red está constituida por "un conjunto de emisores y receptores conectados a un medio de transmisión". En general a los emisores y receptores se les denomina **nodos**, y a los medios que los interconectan se les denomina **enlaces**. Esta definición es deliberadamente muy general y no aplica solamente a las redes de computadoras. De esta forma podemos hablar de redes de distribución de agua, de energía eléctrica, de comunicaciones telefónicas, etc.

Por ejemplo:

- En la red telefónica consideramos que los nodos son los teléfonos particulares y las centrales telefónicas. Los enlaces son los cables telefónicos que recorren la ciudad (o entre diferentes ciudades).
- En la red de correo postal consideramos que los nodos son las oficinas postales y las casas. Los enlaces entre oficinas postales se realizan mediante trenes o aviones. Los enlaces entre la oficina postal y la casa se realiza mediante carteros.
- En cuanto a las redes de computadoras, los nodos son computadoras y los enlaces pueden ser cables, medios inalámbricos u otras formas de interconexión. Notar que los celulares que se conectan a Internet son en realidad computadoras, por lo que estarán considerados como nodos dentro de nuestra definición

de red.

11.3.1.1 Conmutación de circuitos vs. conmutación de paquetes

La red telefónica es desde uno de los puntos de vista posibles, una red de líneas conmutadas y directas.

Las directas se contratan para conectar en forma permanente dos puntos geográficamente distantes. Las conmutadas son aquellas que usamos habitualmente para comunicarnos. Al discar un número indicamos a la (o las) central que debe establecer un circuito (conmutar) entre nuestro teléfono y el identificado con el número discado. Es por ello que hablamos de **conmutación de líneas** o **conmutación de circuitos**.

Una alternativa a la conmutación de circuitos es la utilizada por el correo postal tradicional:

- Un usuario emisor que desea enviar una carta debe poner la carta en un sobre donde escribirá cierta información de envío (dirección del destinatario). Luego deposita la carta en el buzón y la misma es transportada por el cartero hacia la oficina postal.
- La oficina postal se encargará de enviarla a través de la red hacia otras oficinas postales. Dependiendo de dónde se encuentre el destinatario este viaje puede tener varios tramos terrestres o aéreos, pasando por varias oficinas postales, hasta que encuentre la que corresponda al destinatario.
- Un cartero finalmente llevará la carta de la oficina postal a la casa del destinatario.

El usuario emisor se desentendió del proceso en el momento en que depositó la carta en el buzón, confiando en que la red se encargará del resto del trabajo para que su carta llegue a destino. Las decisiones sobre qué camino tomar para llegar al destino no se eligen de antemano (como con la conmutación de circuitos) sino que se van tomando decisiones locales en cada oficina postal.

La red de computadoras por excelencia (Internet) utiliza en su núcleo un esquema similar a la red de correo postal, donde la unidad de envío se denomina **paquete** de información. Es por eso que a este tipo de redes se conoce como redes de **conmutación de paquetes**.

Esto es: cada computadora conectada a la red cuando quiere comunicarse con otra envía a la red un mensaje en el cual pone entre otros datos el destinatario, ese mensaje (conjunto de ceros y unos) es denominado **paquete** en la jerga de redes. Todas las máquinas conectadas reciben el paquete pero apenas analizada la dirección destino saben si deben leerlo todo (en caso de que sea el destino) o ignorarlo. Por supuesto que muchas máquinas enviando mensajes, o intentándolo, al mismo tiempo traería aparejada toda una gama de problemas. Los investigadores de las redes han estado trabajando activamente durante los últimos años, y se han desarrollado estrategias para reducir y/o controlar esas situaciones.

11.3.1.2 Protocolos de comunicación

Para que dos elementos se comuniquen, en este caso dos computadoras, deben cumplir ciertas reglas en común. A este conjunto de reglas se le denomina **protocolo**, y cada nodo de la red debe conocer los protocolos necesarios para poder comunicarse con los demás.

En nuestro ejemplo de la red de correo postal, el protocolo indica que se debe escribir la dirección del destinatario en el sobre y probablemente pegar un sello antes de poner el sobre en el buzón. Si el emisor no escribe esta dirección, o si no pega el sello, él no estará respetando el protocolo, y su carta nunca llegará a destino.

Volviendo a las redes de computadoras. Supongamos que queremos ingresar a la página web del curso. Abriremos nuestro navegador y escribiremos en la barra de direcciones:

http://www.fing.edu.uy/inco/cursos/comp1

Y unos instantes después la página se desplegará en la pantalla. ¿Qué fue lo que ocurrió mientras estábamos esperando? Nuestra computadora, una computadora en la facultad, y otro conjunto de computadoras intermedias tuvieron que dialogar y cumplir con todo un conjunto de protocolos para traernos una copia de la página web. Por ejemplo:

- La computadora intentará buscar cómo conectarse con el servidor de facultad, www.fing.edu.uy, esto se realiza mediante el protocolo DNS.
- La computadora le pedirá al servidor de facultad que le dé una copia de la página web del curso, esto se realiza mediante el protocolo HTTP.
- Cada uno de los mensajes que nuestra computadora intercambia con la de la facultad seguirá un proceso similar al del correo postal. Primero se le envía a la computadora de la central telefónica, que lo reenviará a otras computadoras hasta llegar al destino. Esto se realiza mediante el protocolo IP.

Estos son sólo tres ejemplos de los cientos de protocolos que se están ejecutando constantemente en cada computadora de la red. Otros problemas como: qué pasa si se pierde alguno de los paquetes que viajaba por la red, qué pasa si varias computadoras envían información a la vez, qué pasa si algunas computadoras están conectadas mediante cables y otras mediante enlaces inalámbricos, etc., también son resueltos mediante protocolos.

11.3.1.3 Jerarquía de protocolos

Dada la complejidad de todo el tema de tráfico de paquetes en las redes, a la variedad de implementaciones existentes, y a la cantidad de protocolos que se han construido, se han definido diferentes modelos en capas o niveles que permiten el manejo de alternativas en el momento de armar una configuración de red.

A los efectos de ser gráficos presentamos un posible modelo:

Capa física Está compuesta por los elementos emisores y receptores y el medio de transmisión. Especifica cosas tales como la frecuencia y potencia de las señales, tolerancias a pérdida y ruido en la señal, etc. Un ejemplo de esta capa es la especificación de redes ETHERNET.

Capa de enlace Tiene como objetivo corregir los errores que puedan producirse durante el envío de información en la capa física. Si un mensaje llega con información errónea (corrupto) se deberá descartar o corregir, o pedir al emisor que se lo envíe de nuevo. Si varias computadoras transmiten a la vez y hay colisiones, todos los mensajes que se hayan enviado llegarán corruptos al destino. Un ejemplo de esta capa es el protocolo CSMA/CD que se utiliza cuando varias computadoras transmiten información a través del mismo cable.

Capa de red Es la capa donde se logra que todas las computadoras de la red puedan comunicarse con todas las demás. En esta capa la información puede enviarse mediante conmutación de circuitos o de paquetes. El estándar de Internet en cuanto a capa de red es el protocolo IP, que utiliza conmutación de paquetes.

Capa de transporte Tiene como objetivo asegurarse que toda la información generada por un programa en el nodo A llegue a otro programa en el nodo B. Especifica cómo se componen los paquetes, tamaño, información y como se distribuye la misma dentro del paquete. El estándar de Internet en cuanto a capa de transporte es el protocolo TCP.

Capa de aplicación Esta capa incluye todos los protocolos que brindan funcionalidad directa al usuario. Por ejemplo: correo electrónico, páginas web, aplicaciones de chat, descarga de música y video, etc.

Ahora estamos en condiciones de comprender qué implica decir que una de las redes más utilizadas es la de tecnología TCP/IP sobre ethernet.

Algunas alternativas a la red TCP/IP sobre Ethernet que se utilizaron en algún momento fueron:

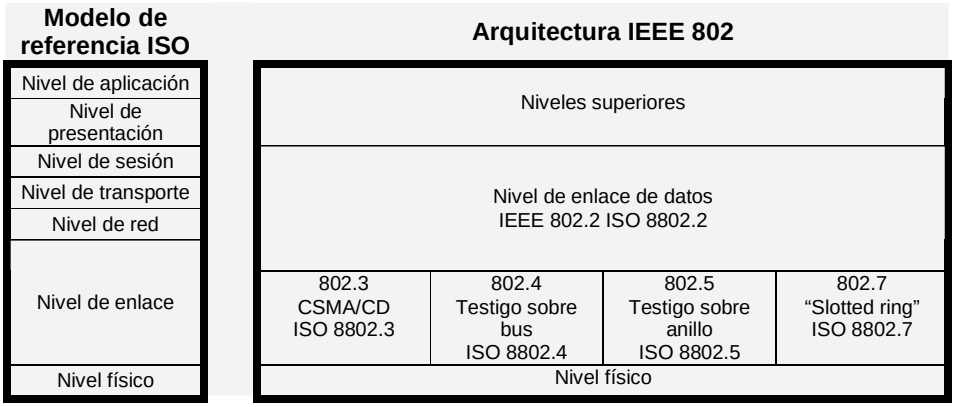
- NetBEUI sobre Ethernet (el protocolo de Microsoft)
- TCP / IP sobre FDDI
- NetBEUI sobre FDDI
- TCP / IP y NetBEUI sobre Ethernet

Si se desea conectar dos redes de tecnologías diferentes a nivel de Capa 1 y 2 se deben utilizar dispositivos específicos.

11.3.2 Modelos de red ISO e IEEE

Más allá de la jerarquía de ejemplo que mostramos, la industria ha generado sus propios estándares de referencia sobre cómo agrupar los diferentes protocolos en capas o niveles según las funcionalidades que proveen. Presentamos brevemente dos ejemplos de estos estándares.

Tabla 19: Dos de las formalizaciones de redes más conocidas son ISO y IEEE



Descripción del modelo ISO

Nivel Físico Garantiza el transporte de la información, es el medio físico de transmisión. La unidad es el bit.
Nivel de Enlace Responsable por el enrutamiento de los bloques de información garantizando que no tengan errores.
Nivel de Red Garantiza el enrutamiento de datos a través de diferentes nodos intermedios, asegurando la trayectoria y el control del flujo resolviendo problemas de saturación de tráfico en las diferentes rutas. La unidad es el paquete.
Nivel de transporte Se encarga de que los mensajes lleguen correctamente a sus destinatarios así como de la segmentación y rearmado de bloques de información que sean muy grandes. (dereccionamiento y secuenciación)
Nivel de sesión Inicia y controla el diálogo entre tareas remotas. La entidad de datos es la transacción.
Nivel de presentación Se encarga de que los datos lleguen a la red en un formato único para que el destinatario, independientemente de su Sistema Operativo y la aplicación pueda comprenderlos.
Nivel de Aplicación Define la forma en que se comunican las aplicaciones. Básicamente hay dos formas: a) ambas aplicaciones en modo conectado (activas) y b) aplicaciones en modo no conectado (tipo mensajería electrónica).

11.3.3 Clasificación de redes según su estructura física

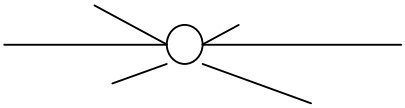
En esta clasificación se observará la estructura física en que la información circula a través de los medios de comunicación. Éstos medios pueden ser cables de cobre en diversas alternativas (TP, STP, UTP, coaxial, etc.), fibra óptica, microondas, etc..

Hacemos referencia a la forma en que los datos viajan debido a que es posible simular estructuras físicas con esquemas donde los datos en realidad hacen un recorrido diferente al aparente.

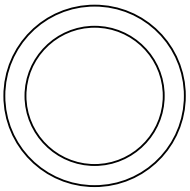
Algunas estructuras muy frecuentes son:

Bus

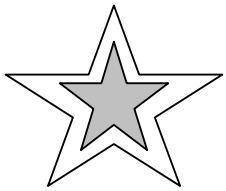
Estrella



Anillo



Anillo con forma de estrella; desde el punto de vista técnico - eléctrico - los datos viajan como en un anillo, desde el punto de vista de la disposición física de los equipos puede parecer una estrella. Esta conformación habitualmente se logra con equipos "concentradores" que gestionan el tráfico de la información



11.3.4 LAN (Redes de área local)

Una red surge al interconectar dos o más computadoras para compartir e intercambiar información. Con ello es posible que todos los usuarios conectados en algún lugar de la red puedan utilizar recursos comunes. Los recursos más compartidos son los discos y las impresoras. Pero es posible compartir otros. Además el entorno de red da la posibilidad de intercambio de información de muy diversas maneras entre los diferentes usuarios, desde el correo electrónico hasta el trabajo en grupo con herramientas específicas.

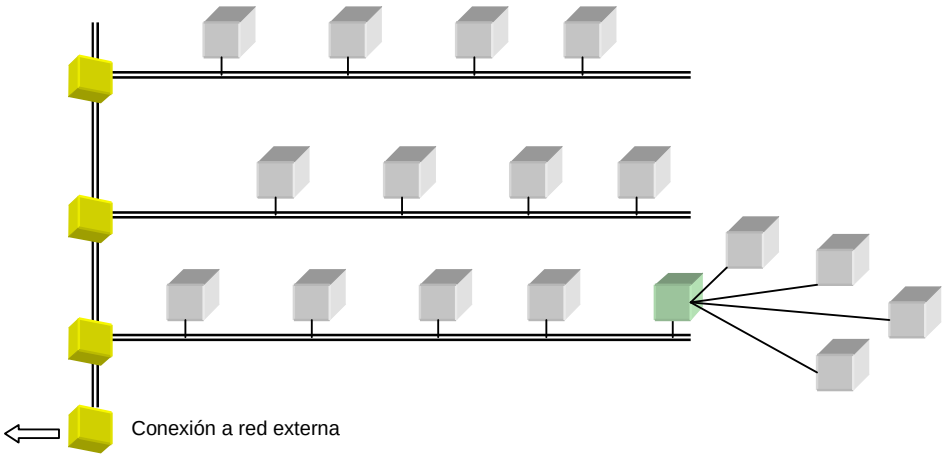


Ilustración 17: Esquema general de un cableado típico en un edificio

Las redes están compuestas por elementos electrónicos y programas y se clasifican de muy diversas formas,

según se consideren aspectos técnicos a nivel de los componentes electrónicos, a nivel del software (en alguno de los diferentes niveles), según la funcionalidad, etc.

Tomando uno de los criterios posibles (el papel que juegan los diferentes equipos) podemos distinguir dos tipos básicos de redes:

a) Redes con estructura servidores - estaciones de trabajo

Las redes con esta estructura se basan en que un equipo por lo menos se encarga de mantener la estructura lógica de la red y ofician de administradores de los recursos. Esos equipos se denominan servidores de red. Sus tareas típicas son el control de acceso de usuarios y verificación de permisos de esos usuarios para utilizar determinados recursos.

La administración de estas redes se efectúa desde usuarios específicos que tienen la potestad de asignar permisos de uso sobre los recursos de toda la red (computadoras, impresoras, etc.).

Ejemplos de estas redes son el NETWARE de NOVELL o el Windows 2000 Server y se conocen como redes “con dominio” ya que cada servidor tiene potestades sobre un cierto dominio de equipos.

b) Redes par a par (peer to peer)

Estas redes (también conocidas como punto a punto) se han extendido mucho en los últimos años. En ellas todas las máquinas conectadas tienen igual status, de ahí su nombre. Cada una de ellas puede autorizar el uso de sus recursos a los usuarios de otras máquinas de la red.

Ejemplo de estas redes son las que vienen por defecto con Windows 95/98/NT/2000/XP llamadas “grupos de trabajo”.

11.3.5 WAN (Redes de Área Extendida)

Consiste en la conexión de dos o más redes LAN. En general se asume que una LAN agrupa todos los equipos de una cierta área física, un edificio por ejemplo. Las redes WAN permiten que las redes LAN de dos áreas geográficas diferentes se conecten entre sí.

El siguiente ejemplo entra en la subclase denominada CAN “Campus Area Network” que tiene por finalidad unir las redes de dos o más edificios ubicados en un área relativamente chica.

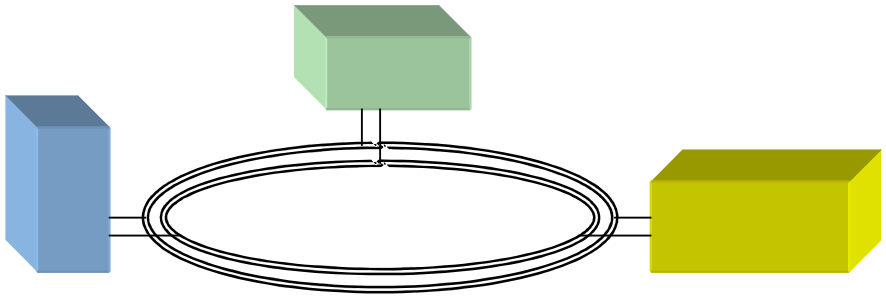


Ilustración 18: Esquema de cableado entre edificios usando un anillo doble de fibra óptica.

11.3.6 WAP – Protocolo de Redes Inalámbricas –

Es el protocolo definido como estándar por la industria de la telefonía celular. Si bien fue promovido por ésta en la especificación se consideraron otros dispositivos como las “palms”. Las utilidades de tal protocolo van

desde las personales hasta las empresariales. Como ejemplos podemos citar

Personales:

- servicios de banca on-line (mobile home-banking, bolsa, ...)
- venta y reserva de billetes (transportes, hoteles, etc)
- ticketing: espectáculos
- información de clima, tráfico, horarios, turística, ...
- escenarios de compra
- ...

Profesionales y empresariales:

- agendas corporativas WAP
- gestión de pedidos (fuerza de ventas)
- servicios de localización
- gestión de flotas
- servicios de mensajería
- tiendas virtuales
- comercio electrónico móvil
- ...

Básicamente el protocolo WAP es una adaptación del protocolo http de navegación en internet considerando pantallas con capacidades más reducidas que un monitor de computadora.

11.4 Roles en la interconexión

11.4.1 Hosts y terminales de datos

Cuando el uso de las computadoras comenzó a hacerse masivo, los equipos eran grandes y costosos. En general las empresas podían permitirse comprar solamente un equipo que fuera medianamente potente, y esta máquina tenía que permitir realizar todas las tareas de computación relativas a la empresa. En ese momento surgió el modelo de hosts y terminales de datos.

11.4.1.1 Host

Se denomina así a aquellas computadoras que realizan toda la funcionalidad (ejecutan toda la lógica) de la aplicación de referencia. Típicamente son hosts los equipos centrales (mainframes) de las grandes empresas y corporaciones a los cuales se accede desde terminales de datos (ver más adelante). Estos equipos suelen ser de gran porte, pero el término hoy en día es más bien conceptual y se puede aplicar a equipos pequeños. Por ejemplo se dice que el equipo donde se alojan las páginas web es el host internet de las mismas ya que es el encargado de que esas páginas se puedan navegar desde Internet. El alquilar un equipo para alojar páginas se conoce como "hosting".

11.4.1.2 Terminales de datos

Se entiende por tales a dispositivos que permiten al usuario conectarse con una computadora central (denominada host) y ejecutar procesos en ella.

El terminal en sí solamente puede ejecutar el protocolo de comunicación adecuado para intercambiar información con el host, todo el proceso de la información lo realiza esa computadora que denominamos host. Inicialmente las terminales que se conectaban al host solamente contaban con un dispositivo de entrada (teclado) y uno de salida (monitor). El procesamiento que podían realizar se limitaba exclusivamente a controlar el protocolo de comunicación con el host, usualmente almacenado en una memoria ROM, y no

eran capaces de realizar procesamiento alguno para el usuario. Por este motivo se les denominaba "terminales tontas" (dummy).

A medida que el precio del hardware fue bajando, comenzó a abandonarse el uso de las terminales tontas, sustituyendo las mismas por PC's que entre otros programas ejecutan los programas de protocolo de terminales (emulan terminales). Las ventajas de esto son múltiples, se pueden tener varias terminales de varios equipos al mismo tiempo por el costo de un PC, y simultáneamente se pueden ejecutar otras tareas propias de los PC's.

Típicamente los terminales ofrecían interfaces de 24 o 25 líneas por 80 columnas. En la década del 90 se popularizaron las terminales gráficas que también dependían de un host para ejecutar la lógica de cualquier aplicación pero que utilizaban con el usuario una interfaz gráfica completa. En este caso el terminal ejecuta el Manejador de Ventanas (window manager) - que es el Servidor - y las aplicaciones que se ejecutan en el host (servidor de aplicaciones) ejecutan clientes del Manejador de Ventanas.

11.4.2 Servidores y clientes

En este capítulo veremos las características de Servidores (servers), y Estaciones de trabajo (workstations)

La diferencia es meramente funcional, no hace referencia explícita a la potencia o periféricos, ni al sistema operativo que ejecuta un equipo.

11.4.2.1 Servidores

En general se trata de servidores de impresión, disco y de control de acceso de usuarios.

Otros servicios que pueden brindar:

- Correo electrónico
- CD-ROM
- Cámara de vídeo
- Páginas WEB (WWW)
- Bases de Datos
- Procesos de cálculo intensivo

Cualidades que habitualmente tienen:

Se les pide buena capacidad de almacenamiento en disco, memoria RAM suficiente para atender los requerimientos de varios usuarios conectados simultáneamente, eventualmente controles de seguridad, hardware específico para brindar más seguridad (redundancia de datos: replicación de discos y discos espejados, unidades de protección contra fallos eléctricos - UPS y estabilizadores -, mejores canales de acceso a los discos, etc.)

Más adelante daremos otra visión del concepto "servidor".

11.4.2.2 Estación de Trabajo, PCs y Puestos de Trabajo

En general se denomina "estación de trabajo" (o workstation en inglés) a equipos que están destinados a ser la herramienta de trabajo de una persona o un grupo de personas que desarrollan alguna actividad en particular, por ello suelen tener cualidades específicas.

Ejemplo de Estación de trabajo para un Ingeniero con funciones de diseño gráfico:

Sistema operativo con interfaz amigable, preferentemente GUI (Graphic User Interface), buena capacidad de memoria RAM para una ejecución rápida, periféricos específicos para la aplicación a la que se destina: plotter/impresora gráfica color, CD-ROM, cámara de vídeo, scanner, tarjetas adquisidoras de datos, etc. En general se la configura especialmente para cumplir ciertas tareas específicas.

En esencia una estación de trabajo no da servicios, pero perfectamente puede actuar como servidor en alguno de los ítems descriptos para Servidores.

PCs y Puestos de Trabajo

Al igual que las estaciones de trabajo ejecutan software y pueden comunicarse con otros PCs, estaciones de trabajo y servidores. La principal diferencia radica en que en esta línea de equipos en general encontramos una mayor oferta de equipos y componentes lo que posibilita que se pueda adquirirlos a precios sensiblemente menores, en tanto que hasta hace unos años las estaciones de trabajo eran computadoras notoriamente más potentes, su arquitectura era RISC y su sistema operativo generalmente UNIX. Son equipos muy flexibles que se pueden adaptar a múltiples actividades con diferentes niveles de potencia.

11.5 Arquitectura Cliente – Servidor

11.5.1 Introducción

Una vez que se tiene una red de computadoras conectadas entre sí se necesita construir programas que corran en dichas computadoras y que permitan intercambiar información de manera que se aproveche al máximo el potencial instalado. Dejamos de lado el caso de las terminales y hosts que ya fuera descrito.

La información intercambiada entre computadoras en una red se basa en el envío de mensajes. Teniendo en cuenta esto supongamos el siguiente escenario:

Tenemos dos computadoras C1 y C2 conectadas en una misma red, en C1 se ejecuta el programa P1 y en C2 se ejecuta el programa P2. Supongamos también que en determinado momento P1 necesita un servicio que brinda P2. Para esto P1, utilizando los servicios del SO para acceso a la red envía un mensaje a P2 en la computadora C2. En el mensaje se especifica qué servicio se requiere y posiblemente la forma de responderlo. Una vez que el mensaje es recibido por el SO de C2 este lo hace disponible para P2 y le avisa a P2 que tiene un mensaje nuevo. P2 realiza el servicio requerido y cuando termina le envía (siguiendo un procedimiento similar) un mensaje a P1 en C1 con la respuesta a su pedido.

C1 es la computadora Cliente y C2 es la computadora Servidora del servicio solicitado por P1. De la misma forma P1 es el programa Cliente del programa Servidor P2. Se ve entonces que cuando se hace referencia a un Servidor (o Cliente) se puede estar hablando de una computadora o un programa. Cuando se habla de una computadora, debemos tener claro que la cualidad de servidor está dada por él o los programas que en ella se ejecutan.

Observar también que C1, puede estar ejecutando otro programa R1 que es servidor de otro servicio. Por lo tanto C1 sería Cliente del servicio que brinda P2 y sería Servidor del servicio que brinda R1. Lo mismo se aplica a los programas.

Las acciones son siempre iniciadas por el cliente en tanto que el servidor está a la espera de pedidos de trabajo. En equipos con Sistema Operativo Unix los programas servidores se identifican generalmente con los llamados “daemons”. En entornos DOS-Windows son los llamados “programas residentes”. Clientes y Servidores pueden estar en la misma máquina o no.

Cliente / Servidor es la arquitectura de diseño de aplicaciones descripta anteriormente.

11.5.2 Descripción

Para que todo esto sea posible es necesario que:

C1 y C2 estén conectadas en la misma red o en redes interconectadas, obviamente también pueden estar el programa cliente y el servidor en la misma máquina.

Utilicen el mismo protocolo o exista entre ellas algún traductor (gateway) que transforme un protocolo en otro haciendo posible el diálogo.

Algunas de las ventajas de desarrollar aplicaciones de esta forma son:

C1 y C2 pueden (y es común que sean) computadoras diferentes, con sistemas operativos diferentes, de

fabricantes diferentes y costos diferentes.

En forma equivalente P1 y P2 pueden estar construidos utilizando lenguajes diferentes por programadores diferentes.

Permite dimensionar los puestos de trabajo para la función específica que van a cumplir. Por ejemplo: un servidor de BD debería tener mucha capacidad en sus discos, en cambio una estación de trabajo debe tener un buen monitor y potencia de cálculo.

Permite a los usuarios de las aplicaciones gran flexibilidad en lo que se refiere a la ubicación física de los equipos. Las distancias no se miden en Km, se miden en Bits/segundo (unidad de transferencia).

Algunos elementos a considerar al ejecutar aplicaciones C / S en red:

Se necesita tener un equipamiento de red que soporte el tráfico de información que las aplicaciones requieren.

Si la red está muy saturada (en términos de tráfico) todas las aplicaciones que acceden a servicios remotos verán afectada su performance. En este caso el "cuello de botella" de la performance no está en la velocidad de procesamiento en el cliente o el servidor sino en la del medio de comunicación entre las máquinas.

11.5.3 Ejemplos de aplicaciones Cliente / Servidor

Aplicaciones de Bases de Datos: El DBMS (Manejador de la Base de Datos) es el servidor y las aplicaciones de usuario son clientes de la misma.

Internet (WEB SITES): los sitios (sites) tienen aplicaciones servidores y los programas de navegación (browsers) son clientes. En este caso el browser implementa varios clientes: http, gopher, ftp, etc. los que se comunican con los diferentes servidores del "site".

Manejadores de entornos de ventanas (en inglés: Window Manager): los programas solicitan al manejador de ventanas (por ejemplo Windows) que despliegue una ventana con cierto título, texto y botones. Las características de color y aspecto general las da el manejador según la configuración del usuario. Una vez que el usuario active alguno de los botones el Manejador pasará la información al programa y éste decidirá, por ejemplo, que el manejador debe ahora cerrar esa ventana de diálogo.

La utilización de discos en una estructura de red: el equipo que "presta" el disco debe ejecutar un programa servidor y el que desea acceder a ese disco ejecuta un programa cliente.

Otros ejemplos de aplicación de la arquitectura C / S lo son los teléfonos celulares los cuales vienen provistos de aplicaciones cliente de páginas Web. Esto permite que se pueda navegar e interactuar con sitios Web de Internet.

11.6 Herramientas de software: segunda parte

11.6.1 Descripción de algunas herramientas de propósito general

11.6.1.1 Planilla electrónica

Una planilla de cálculo es una tabla de valores dispuestos en filas y columnas, la intersección de una fila y una columna define una celda. Cada celda puede contener un dato constante de tipo caracteres, numérico, fecha o una fórmula que relaciona otras celdas mediante sus direcciones absolutas o relativas a la posición de la celda en cuestión.

El origen es la clásica planilla en papel que se utiliza para múltiples actividades (contables, de ingeniería, etc.).

Un programa de planilla electrónica permite crear y manipular planillas con la computadora. Ejemplos de estos programas son: Lotus 1-2-3, Quattro y Excel.

Se puede definir qué tipo de datos hay en cada celda y cómo el valor que tiene una celda depende de otras.

Las relaciones entre celdas se llaman fórmulas, los nombres de las celdas se llaman etiquetas.

Se pueden ingresar los datos en las celdas, definir fórmulas que las conecten y modificar los datos para ver como cambian automáticamente las celdas dependientes; esto permite investigar resultados de acuerdo a diferentes situaciones (parámetros).

En las celdas se puede poner texto que haga a la planilla comprensible y presentable. Es posible que una planilla tenga una presentación tal que la convierta en un documento completo y pronto para ser impreso.

Se pueden generar macros (secuencias de instrucciones guardadas en la celda) para que al invocarlas se ejecute esa secuencia a modo de programa, permitiendo realizar tareas repetitivas sobre la planilla invocándolas con un par de teclas.

Cuando se guarda una planilla en un archivo guardamos los datos, la estructura y las macros.

Las planillas más conocidas tienen también otras posibilidades, por ejemplo:

- Reordenar la planilla de acuerdo a diferentes criterios (por ej. en orden alfabético).
- Producir gráficos en función de los datos de la planilla.
- Fórmulas de estadística y financieras predefinidas.
- Vinculación entre varias planillas (un cambio en una de ellas afecta a las demás).
- Extracción de datos de otra planilla.
- Extracción de datos de bases de datos.
- Diversas opciones para la visualización de la planilla que permiten cambiar tipos de letra, colores, formato de los números, etc.

Ventajas:

- Son fáciles de usar
- Se adaptan a la solución de una gama muy amplia de problemas
- Rapidez para modelar problemas sencillos
- Una vez modelado también es sencillo obtener presentaciones gráficas
- Capacidad de importar información en varios formatos e incluso conectarse a bases de datos

Desventajas:

- Rigidez: si hay cambios en el problema o en muchos datos puede ser complicado realizarlos
- No manejan cantidades muy grandes de datos (está acotada la cantidad de filas y columnas de la planilla)
- Redundancia: la necesidad de que un dato aparezca en varios lugares es difícil de solucionar eficientemente
- Si la complejidad del problema rebasa ciertos límites la planilla que lo resuelve puede ser sumamente compleja creando dificultades para operarla: quien opera la planilla debe saber dónde está cada dato pues hay una referencia espacial de los datos

Buena parte de estos inconvenientes devienen en que la lógica está embebida con los propios datos y dependiente de la estructura.

11.7 Archivos y bases de datos

El tratamiento de la información lleva en general asociado el manejo de datos almacenados en disco bajo la forma de archivos. Un archivo es la forma más elemental (primaria) de guardar información.

Es posible resolver diferentes tipos de aplicaciones (científicas, comerciales, administrativas, etc.) mediante la utilización de archivos. Cuando las aplicaciones requieren niveles de seguridad, velocidad de acceso/modificación de los datos, distribución en varios servidores, niveles de abstracción que permitan un rápido y cómodo desarrollo y mantenimiento de las aplicaciones, se recurre a herramientas de software que resuelven algunos o todos esos aspectos.

Los archivos están compuestos de registros cuya información está organizada en campos.

Un campo es una porción individual de información, un registro es un conjunto de campos, y el archivo es una colección de registros.

Por ejemplo: la guía telefónica es análoga a un archivo; contiene registros, cada uno de los cuales tiene tres campos: nombre, dirección y teléfono.

11.7.1 Bases de datos

Una base de datos es una colección de información organizada de manera que un programa de computación puede seleccionar porciones de datos deseadas (idealmente en forma rápida y eficiente). Esa información está bajo el control de un conjunto de programas que constituyen el **Sistema Manejador de Base de Datos** (DBMS por sus siglas en inglés) el cual gestiona el manejo del conjunto de archivos que dan soporte físico a la información.

A los efectos de diferenciar los conceptos decimos que un Banco de Datos es una colección de información no necesariamente ordenada y utilizable con cualquier finalidad.

11.7.1.1 Modelo Relacional

Es una técnica utilizada para el análisis de la información. Ésta se clasifica en Entidades y Relaciones. Para ejemplificar digamos que Entidad pueden ser PERSONA y BANCO, en tanto que una Relación puede ser la CUENTA_CORRIENTE de una Persona en un Banco.

PERSONA			
Cédula	**País**	**Nombre**	**Dirección**
122222	Uy	Rosario F.	Zufriategui 2230
133333	Ar	Martín G	Rodó 7879
144444	Uy	Ana S.	Canelones 234
155555	Uy	Jorge C.	Cuareim 1234

CUENTA_CORRIENTE		
Cédula	**Banco**	**Cuenta_C**
122222	BROU	2001
122222	BROU	2002
133333	CITY	3011
144444	CITY	3012
144444	BROU	2003

BANCO	
Nombre	**Datos**
CITY	Xxx
BROU	yyy
ABN	JJJ

PERSONA, CUENTA_CORRIENTE y BANCO son denominadas tablas. Cada tabla contiene información que se organiza en columnas que se corresponden con atributos de cada Entidad o Relación, y en filas (tuplas) que están constituidas por la información misma.

En el ejemplo se observa que la Persona identificada por la Cédula número 122222 tiene dos cuentas corrientes en el Banco BROU, cada cuenta corriente tiene un identificador único que es su número. En nuestro caso como estamos manejando la información de varios Bancos utilizaremos como identificador la clave Banco (+) Cuenta_Corriente; (+) significa concatenar, poner uno a continuación del otro .

Cuando un atributo, o conjunto de ellos, de una tabla es capaz de identificar a todo el resto de los atributos de una fila se le denomina clave de identificación.

11.7.1.2 Sistemas de manejo de Bases de Datos

Son conjuntos de programas que resuelven el tratamiento de datos en gran escala. Incluyen todo lo necesario para satisfacer requerimientos como manejar un enorme volumen de datos, seguridad de los mismos (manejar diferentes usuarios que sólo pueden ver y o modificar determinada parte de los datos), bases distribuidas en varios equipos, bases replicadas (varias copias de la misma), mantener información histórica de cambios que permitan reconstruir la base de datos en casos de pérdida de información, etc. Los Manejadores diseñados para implementar modelos de datos relacionales se conocen como RDBMS.

Pueden actuar como servidores para programas clientes que acceden a los datos a través de ellos. A finales de la década de 1980 se comienzan a estudiar Manejadores de Bases de Datos Orientadas a Objetos, conocidas como OODBMS. Estas permiten resolver aplicaciones que las Relacionales no, o que lo hacen a costos de programación muy altos. Por ser mucho más complejos en su diseño e implementación esos Manejadores son más lentos (instalados en máquinas iguales) que los Relacionales, por ello no los reemplazaron en las aplicaciones comerciales y de gestión administrativa (a las cuales se puede modelar muy bien con los RDBMS) sino que se reserva su utilización a casos muy concretos. De todas formas los Relacionales han tomado algunos principios del modelo de datos Orientado a Objetos dando lugar a una fusión que se conoce como ROODBMS (Manejadores de Base de Datos Relacionales Orientados a Objetos).

11.7.1.2.1 Transacción

Las bases de datos manejan un concepto denominado transacción que es fundamental en su esquema de funcionamiento. Podemos definir como transacción a un conjunto de tareas que deben realizarse en la Base de Datos de tal modo que ninguna de esas tareas en forma individual tiene sentido, o se realiza todo el conjunto o no se realiza ninguna.

Un ejemplo de ello puede ser un sistema (hipotético) de facturación. Una factura, al momento de emitirse debe realizar lo siguiente:

- Afectar el stock de cada producto contenido en la boleta.
- Afectar la cuenta corriente del cliente si se factura a crédito (siempre y cuando la verificación de que el crédito autorizado para ese cliente no es superado por la factura)
- Obtener un Número para la Factura según el esquema de la DGI.
- Registrar el movimiento de caja
- Imprimir la factura

Cualquier evento/circunstancia que impida que alguna de las acciones anteriores no pudiera concretarse causará que la Factura no se emita y por lo tanto ninguna de las tareas descriptas se deberá llevar a cabo.

Una vez descrito este concepto podemos analizar algunas de las funciones que tiene un DBMS.

11.7.1.2.2 Acceso Concurrente

Es el acto de que dos o más transacciones deseen acceder o modificar simultáneamente un mismo dato. El DBMS tiene definidas e implementadas políticas que resuelven esta situación satisfactoriamente.

11.7.1.2.3 Consistencia

Un DBMS observará que se cumplan ciertas reglas dictadas por el Administrador de la Base de Datos (DBA) y no dejará que ésta quede en un estado definido como inválido (inconsistente).

Por ejemplo: en un sistema bancario no es admisible que se pueda dar de baja a un titular de cuenta si éste tiene cuentas abiertas.

11.7.1.2.4 Integridad

Un DBMS no puede perder información. Si por algún motivo el soporte físico (disco o conjunto de discos) se daña, el DBMS debe poder reconstruir el estado consistente inmediato anterior, o sea, todas las transacciones deben haber terminado correctamente. Para ello se basa en respaldos de la información y en los archivos de

"log" donde el DBMS guarda por estricto orden todos los movimientos que afectaron la información de la Base desde el momento inmediato posterior al respaldo hasta que el sistema "cayó". Obviamente el archivo de "log" se guarda en un dispositivo diferente del utilizado para la Base de Datos, en general es una unidad de cinta magnética.

11.7.1.2.5 Seguridad en el acceso a los datos

El DBMS tiene capacidad de manejar diferentes clases de usuarios, los cuales deben identificarse ante él a los efectos de realizar diferentes acciones sobre los datos por él gobernados. Hay usuarios que pueden administrar las estructuras de la Base, otros pueden insertar, borrar o modificar información, otros sólo pueden ver cierta información (eventualmente toda).

11.7.1.2.6 Lenguaje de acceso SQL

Las bases de datos tienen siempre un lenguaje con el cual los programadores le ordenan las diferentes acciones a realizar. Actualmente todas las bases de datos comerciales deben tener al menos uno denominado SQL (Structured Query Language) creado por IBM en la década de 1970 y hoy reconocido como el estándar por los organismos mundiales de normas y estándares. En general los fabricantes de DBMS hacen extensiones al SQL para sacar provecho de las ventajas tecnológicas de sus productos, las cuales deben estar claramente indicadas como extensiones en la documentación. El usar un lenguaje común permite que para una cierta aplicación sea posible cambiar el “motor” de la base de datos a costos evaluables y en general no muy altos.

11.7.1.2.7 Bases de Datos Distribuidas

En aplicaciones donde la red abarca un área geográfica muy vasta (bancos, aerolíneas internacionales, etc.) es conveniente por muchos motivos que la información esté distribuida en varias computadoras aunque sean utilizados los datos como si estuvieran en una única máquina (por ej. en un Banco es lógico que las Cuentas de los clientes abiertas en una sucursal se almacenen en la misma sucursal). Los manejadores de base de datos que tienen la particularidad de poder manejar la información de esta forma se dice que permiten la implementación de Bases de Datos Distribuidas.

11.7.1.2.8 Replicación de Información

Hay situaciones en donde además de la distribución de la información es conveniente que en algunos nodos de la red se tenga copia de parte o toda la información de otro. Se habla de Replicación al conjunto de técnicas que permiten mantener réplicas de los datos (parciales o totales) en diferentes nodos. La habilidad de implementar alguna forma de réplica no es común a todos los DBMS.

11.7.1.2.9 Resumen

Como vemos son muchas las funciones que realiza un DBMS. Podemos resumir sus ventajas en:

- Manejo estructurado y eficiente de grandes volúmenes de información
- Gran potencia de manejo de la información
- Niveles de seguridad y velocidad de acceso (eficiencia) adecuados para aplicaciones críticas y en tiempo real
- Capacidad de gestionar información distribuida en diferentes equipos (eventualmente distantes geográficamente) y de mantener réplicas de los datos según se le indique
- Garantías de la integridad de la información almacenada y de que es consistente según las reglas fijadas
- Garantías de que la información será accedida sólo por aplicaciones autorizadas (seguridad de acceso)

Todo ello tiene como contrapartidas: un elevado precio de los DBMS, necesidad de personal altamente especializado que sea capaz de poner en funcionamiento óptimo todos estos mecanismos y equipos de muy buenas prestaciones para garantizar tiempos de respuesta adecuados. Originalmente los DBMS sólo

funcionaban en computadoras de gran porte (mainframes o supercomputadoras), hoy es posible encontrarlas en máquinas de mediano y pequeño porte (incluso en PC's) siendo la limitante la calidad de servicio que se desea obtener.

11.7.1.3 Administradores de Archivos xBase

Son sistemas de gestión de información que mediante la creación de índices de la información almacenada aseguran tiempos de respuesta aceptables y facilidades para la obtención y procesamiento de la información.

Carecen de sistemas de control y seguridad y están pensados para manejar pequeños volúmenes de información.

Su área de trabajo está limitada en general a los PC's y resuelven problemas de gestión administrativa y contable.

Se originan en una herramienta llamada dBase de la cual surgen luego: Clipper, Fox y Access. Todas ellas manejan formatos de archivos básicos en común por lo que se les conoce como pertenecientes a la familia de productos **xBase**. Intentan dar facilidades de acceso a los datos pero carecen de la mayoría de las características reseñadas para los DBMS.

Algunos incluso permiten el acceso a la información utilizando SQL como lenguaje, pero no debemos pensar que por poseer esa característica son DBMS. La utilidad de poseerlo es que se pueden desarrollar aplicaciones para PCs que utilicen los mismos comandos con un pequeño manejador de archivos o con una Base de Datos ubicada en un Servidor.

11.8 Vinculación entre varias herramientas: importación / exportación, OLE

Es muy frecuente tener que poner en un programa información que fue creada con otro.

Por ejemplo incluir en un informe escrito un gráfico, una imagen o una tabla de valores, o incluir en una planilla información de una base de datos.

Durante mucho tiempo se solucionó este requerimiento con herramientas de importación y exportación de datos. Las mismas son programas u opciones dentro de un programa que permiten guardar los datos que se están manejando en otro formato (exportar datos) diferente al considerado "natural" para esa aplicación. Típicamente todos los editores de texto y planillas electrónicas tienen una opción para guardar (*exportar*) un archivo como texto "plano" o "ASCII" (sin formato de letras, subrayados, etc.) y otra para leer (*importar*) archivos de esas características. También tienen la opción de manejar archivos que fueron creados por otros programas (importar datos) en su formato original y convertirlos al formato propio.

Estas soluciones son propias de cada programa, sus creadores le incorporaron estas opciones al hacerlos y si se desea que accedan a otro tipo de datos hay que hacer una versión nueva incorporándoselos o incorporar nuevos ejecutables que puedan ser ejecutados desde la aplicación.

Actualmente con Windows, hay una especificación denominada OLE (Object Linking and Embeding) que indica cómo dos aplicaciones deben intercambiar su información.

Con OLE se puede "embeber" un objeto dentro de otro. Por ejemplo en un texto se puede embeber un gráfico creado con una planilla electrónica (situación típica de quien debe presentar informes). Al hacerlo no estamos poniendo sobre el documento una imagen estática sino que se pone información adicional que permitirá luego que al hacer "doble click" sobre el gráfico sea invocado el programa correspondiente con que se generó la gráfica y los datos correspondientes para así poder proceder a modificarlo (el programa debe existir en la computadora y estar disponible).

Otra forma de incluir un objeto en otro es a través de los denominados **vínculos dinámicos**. La operación es muy similar a la anterior, pero el documento insertado sigue dependiendo del original: si en algún momento

se produce un cambio en él se verá reflejado automáticamente en el incrustado. Así podemos modificar una planilla electrónica y al mismo tiempo estaremos actualizando el documento final. Esta forma de insertar se denomina “pegado especial” en Word.

11.9 Gestión de proyectos

La automatización de la gestión de proyectos es utilizada cada vez con mayor frecuencia a los efectos de evaluar costos, analizar alternativas sobre cómo se desarrollarán las tareas de un cierto proyecto observando los recursos que en cada etapa deberán asignarse y la evaluación y seguimiento a lo largo del desarrollo mismo.

La elaboración de diagramas PERT y GANT en forma manual es tan costosa que sólo era abordada en proyectos de gran envergadura en los que se preveían situaciones críticas en cuanto al desarrollo de las mismas.

Hoy se cuenta con herramientas que facilitan enormemente esta tarea permitiendo la gestión de proyectos en forma muy conveniente.

Ello se refleja en la elaboración de cronogramas de actividades en los que la empresa puede optimizar el uso de recursos entre los diferentes proyectos y evaluar diferentes alternativas con muy poco esfuerzo y gran precisión.

Los programas que resuelven estos temas son muy flexibles ajustándose a las necesidades de los diferentes usuarios. Las unidades de tiempo, económicas, recursos humanos, etc. pueden ser configurados en cualquier momento de forma que se obtienen diferentes puntos de vista del proyecto en escasos segundos.

La asignación de tareas queda documentada y a cada responsable se le entrega un diagrama donde se especifican los tiempos y tareas asignadas a él.

Cada vez es más relevante en la contratación de trabajos a mediano y largo plazo que la empresa que aspira a realizar los trabajos muestre un cronograma detallado de la realización de los mismos, ello revela y documenta, un trabajo de evaluación de las diferentes etapas del proyecto, lo que brinda al contratante un mayor grado de confianza (y un documento) sobre el cumplimiento de los trabajos a contratar.

En estos diagramas se pueden detectar superposiciones de tareas, conflictos en el orden en que deben realizarse, carga de trabajo de cada unidad involucrada, etc.

Para las empresas que utilizan estas herramientas se abre un abanico de posibilidades para la correcta gestión del conjunto de los proyectos que se realizan o realizarán al poder analizar en detalle la utilización de los recursos de la empresa en el contexto de todos los proyectos.

11.10 Búsqueda bibliográfica: CCOD y EI

El resultado de las investigaciones en muy diversas áreas, así como información sobre los más diversos temas es divulgado en revistas, publicaciones periódicas, libros, etc.

Gracias a esto, existe un gran volumen de información que sirve como referencia para quienes deben encarar un proyecto nuevo a la que pueden recurrir buscando antecedentes de situaciones similares.

Para buscar entre todas esas publicaciones hay compañías que producen índices de las mismas.

Los mismos consisten en archivos de datos y programas para buscar sobre ellos y manipular los resultados de las búsquedas (ordenarlos por año, autor, idioma, etc.).

En los datos se incluye una descripción del trabajo, autor, editorial que lo publicó (para comprarlo), la fecha, etc.

En la Facultad de Ingeniería contamos con el CCOD (Current Contents on Diskette) y el EI (Engineering Index).

Su operación es muy sencilla, completamente a través de menús. Luego de realizadas las búsquedas, permiten grabar y/o imprimir el resultado de las mismas.

11.11 INTERNET, hipertexto, búsqueda de información

Internet es una red de redes de computadoras.

Cuando varias computadoras están interconectadas en red, pueden compartir recursos; cuando varias redes están conectadas entre sí, las computadoras que las integran pueden hacerlo también.

El modelo imperante en INTERNET es el Cliente / Servidor. Hay máquinas que dan servicios y máquinas clientes que requieren esos servicios, el modelo C / S permite que unos y otros sean totalmente independientes en sus características técnicas.

11.11.1 Servicios

Las computadoras conectadas a Internet pueden brindar diferentes servicios. Hay personas dedicadas a atender cada red que compone la Internet llamadas Administradores, ellos determinan y atienden los servicios que brindan sus máquinas.

Algunos de los servicios más conocidos son:

11.11.1.1 FTP (File Transfer Protocol)

FTP es un protocolo que permite la transferencia de archivos entre computadoras remotas. Mediante él se pueden copiar archivos desde o hacia otra computadora de la red.

Lo más usual es conectarse para copiar desde otra computadora hacia la propia (en la jerga: “bajar archivos”). Para usarlo uno debe saber el nombre y dirección (dominio) de la computadora remota, el directorio donde están los archivos y tener permisos de acceso adecuados. Muchos sistemas permiten acceso público mediante la utilización de una técnica denominada ***anonymous FTP***. Con ellas no es necesario tener un usuario propio en la máquina destino sino que se utiliza uno previsto para tales fines de nombre *anonymous* y cuya contraseña no existe, si bien se considera una demostración de "buenas costumbres" el poner la dirección electrónica de quien utiliza el servicio.

11.11.1.2 WWW

WWW es la sigla para World Wide Web (“Telaraña Mundial”). Este es el servicio que más impacto ha tenido en el público y con el cual se identifica la Internet.

El Web consta de conjunto de instalaciones llamadas “sites” (lugares) a los cuales pueden acceder computadoras de todo el mundo. Un site consiste en una computadora que ejecuta un programa que atiende pedidos para acceder información (servidor). Típicamente ésta consiste en archivos de hipertexto llamados páginas web que se pueden crear con cualquier editor común y corriente siguiendo ciertas reglas; el protocolo del servicio es ***http*** (**h**yper**t**ext **t**ransfer **p**rotocol)

El pedido se hace con un programa llamado Web Browser (p. ej.: Netscape, Internet Explorer, Mosaic, etc.) en una computadora conectada a la red Internet. Este programa le manda mensajes a los servidores pidiéndoles páginas Web.

Es muy fácil el uso de este servicio, prácticamente alcanza con saber utilizar el ratón. Las páginas Web están como se dijo en formato hipertexto lo cual permite incluir imágenes, diversos tamaños y tipos de letra y referencias a otras páginas. Cuando una página llega al browser el mismo la presenta en forma muy

agradable, incluyendo las imágenes y las referencias a otras páginas (usualmente en un color distinto del resto del texto para distinguirlas) las que se activan utilizando el ratón.

Hay servicios de búsqueda que tienen índices del contenido de las páginas y su dirección. Con ellos es más fácil obtener lo que se necesita. Son la versión electrónica de las guías telefónicas de páginas amarillas.

El Web es una forma sencilla y atractiva de poner información a disposición del mundo. Crece muy rápidamente y se está desarrollando su utilización comercial.

Actualmente los browsers incluyen herramientas auxiliares para componer, enviar, recibir y leer correo electrónico, hacer transferencia de archivos con ftp, etc., se han convertido en programas que implementan clientes para la mayoría de los servicios públicos que es posible encontrar en Internet.

Una variante del protocolo http es el https (hypertext transfer protocol secure) que establece una conexión http pero encripta la información que viaja entre el cliente y el servidor de modo que dificulta el análisis de los datos intercambiados, es el protocolo usado para realizar compras con tarjetas de crédito en Internet y para transacciones bancarias vía Internet entre otras aplicaciones.

11.11.1.3 Correo electrónico

Las aplicaciones de correo electrónico están hoy ampliamente difundidas. Toda red de computadoras permite de alguna forma el intercambio de mensajes entre los usuarios de la misma. Si los equipos tienen acceso a INTERNET, ya sea en modo dedicado o discado, entonces se tiene acceso al intercambio de correspondencia con todos los usuarios que estén conectados.

Típicamente se usa un programa cliente en el cual se escribe la dirección electrónica de correo del destinatario, una breve descripción del asunto por el cual le escribimos (subject) y luego el cuerpo del mensaje. La mayoría de los programas permiten incluir archivos de cualquier tipo junto con el mensaje, con lo que se pueden intercambiar imágenes, sonidos y datos. Una vez que concluimos el mensaje damos la orden de que nuestro programa cliente se comunique con el servidor de correo electrónico de nuestra organización y le entregue nuestro mensaje para que lo haga llegar al destinatario.

El correo electrónico es un servicio sumamente conveniente para facilitar y mejorar las comunicaciones corporativas. Es mejor que el teléfono pues no requiere que el destinatario esté presente para recibir el mensaje, mejor que el fax porque puede enviar la información en su formato original y mejor que el correo tradicional pues los mensajes tardan, en términos generales, unos pocos segundos en llegar a destino. Todo ello a costos sumamente bajos.

Una dirección de correo electrónico (e-address) en Internet es de la forma: usuario@dominio, donde “@” se lee “at”.

Cada vez se le incorporan al correo más funciones con lo cual el correo pasa a ser una forma de transporte de información de aplicaciones más complejas. Podemos mencionar entre esas aplicaciones:

Agenda. Si bien los programas de agendas personales de citas ya tienen bastante tiempo de desarrollados la difusión del correo hoy permite utilizar el correo electrónico para enviar automáticamente a todos los participantes de una cita la notificación correspondiente con detalles de día, hora y lugar en que ha de realizarse.

Servidores de citas. Son programas que sincronizan las agendas de citas de varios posibles participantes de modo que antes de proponer una cita ya sepamos si todos los involucrados dispondrán de tiempo para el momento que elijamos para proponer una reunión.

Flujos de trabajos automatizados (Work Flow). En actividades donde hay una fuerte demanda de gestión de expedientes es posible diseñar un sistema de seguimiento que gestione el envío del mismo desde un lugar de tratamiento a otro. O sea, se puede definir que un cierto tipo de expedientes debe recorrer un cierto camino entre oficinas, que en cada una de ellas se debe realizar cierta tarea y luego debe enviarse a otra determinada. El correo electrónico puede servir para que el envío se realice de forma rápida y segura entre cada nodo. Se puede controlar, monitorear y realizar estadísticas sobre el desarrollo de los trámites para detectar posibles problemas y tomar las acciones correctivas correspondientes.

11.11.1.3.1 Netiquette

Son personas las que se envían correo electrónico y personas también las que diseñan y ponen a disposición del mundo las páginas Web. A través del tiempo han surgido reglas tácitas de comportamiento y de uso de recursos en la Internet a las que se les llama netiquette - net etiquette - (etiqueta de la red).

Algunas de ellas son:

No está bien enviar correo sin poner "asunto" (subject). Eso hace perder tiempo a quien recibe el correo pues del "asunto" puede fácilmente deducir si es algo que debe leer ya o puede dejarlo para otro momento.
Escribir un texto en mayúsculas o con letras destacadas de alguna forma equivale a gritar.
El correo electrónico no transmite su rostro ni su estado de ánimo, al escribir se debe ser claro y preciso evitando en lo posible las ambigüedades. Al carecer de la comunicación gestual un chiste se puede interpretar como un agravio. De igual modo juegan su papel las diferencias culturales. En el monitor y a través de un e-mail no es fácil saber y comprender el entorno cultural de nuestro interlocutor, evitar chistes que aún inocentemente destraten pueblos, religiones, razas, costumbres, etc. es un buen principio.

Conviene familiarizarse con estas reglas para evitar malos entendidos y problemas. En la Facultad de Ingeniería se puede acceder al documento completo de las netiquettes en la dirección: http://www.albion.com/netiquette/index.html

También puede encontrarse información sobre falsos virus y cadenas de solidaridad en:

http://www.rompecadenas.com.ar

12 Tendencias

La informática ha dejado de ser algo utilizable por expertos. Cada vez más nos rodeamos de elementos informáticos disfrazados de elementos cotidianos.

La tendencia es integrar justamente objetos de uso habitual para que tengamos "todo bajo control" desde cualquier lugar que nos encontremos.

Así podemos escuchar cosas como:

12.1.1.1 Edificios inteligentes

Son edificios que se pretende tengan que una altísima automatización en todos sus aspectos y todo ello monitoreado y controlado en forma centralizada. Desde los sistemas de calefacción, telecomunicaciones internas, seguridad en áreas de acceso, garage, sistemas de alerta y combate de incendios, iluminación, etc. Seguramente habrá diferentes niveles de inteligencia, el primer paso es que el edificio tenga una adecuada red interna de comunicación de datos ya que para viabilizar proyectos como estos se trata que todos los elementos se puedan comunicar en red, y si es posible utilizando la misma red.

12.1.1.2 Internet del futuro (cuando el futuro es ahora)

Un modelo planteado para la Internet del futuro es que en ella, en algún lugar, estarán nuestros archivos de datos y que los programas que necesitemos los ejecutaremos en la propia Internet, no los tendremos cargados en nuestra "estación de trabajo", pagaremos por su uso y no deberemos preocuparnos por actualizarlos. Por ejemplo:

- Nuestra agenda. La podremos acceder desde cualquier lugar donde estemos y podrá ser usada por otras agendas (las que autoricemos) para coordinar citas.
- Nuestros documentos. El procesador de textos talvez esté en microsoft.com y nuestros datos en algún servidor suministrado por nuestra tarjeta de créditos o el supermercado del barrio.

Este modelo implica muchas cosas.

- No necesitamos en nuestros puestos de trabajo tener unidad de disco. Serían en esencia "terminales gráficas".
- Utilizando arquitectura Cliente / Servidor esto parece ser bastante ordenado y eficiente.
- Deberá aumentar el ancho de banda de transmisión de datos.
- Todas estas alternativas consideran que nuestros datos están en la red, así podemos accederlos desde cualquier lugar en cualquier momento. No están aún planteadas cuestiones como qué privacidad tendrán mis datos ni qué derechos me protegen.

12.1.1.3 Televisión interactiva

Ya están funcionando en Europa y EE.UU. televisores que pueden ser usados, por ejemplo en los hoteles para que el pasajero consulte su estado de cuenta, los espectáculos que hay en la ciudad, horarios de museos y medios de transporte.

12.1.1.4 Set – on – Box

Dispositivo que se conecta a la red telefónica y al televisor, posee un teclado y permite navegar en Internet. Convierte al televisor en un terminal conectado a Internet.

Este será el modelo de equipo que reemplazará al tradicional TV y al Computador de Hogar. Recordemos que ya está en funcionamiento la transmisión de señales de TV vía Internet y servicios de acceso a Internet utilizando el cable del proveedor de servicios de TV (ver más adelante Cable – MODEM).

12.1.1.5 Cable - Modem

Se refiere a la posibilidad de transmitir a través de la red de TV cable información binaria. Para hacerlo posible el usuario debe instalar un modem (especialmente diseñado) en el cable de señal de TV y luego conectarlo al computador.

12.1.1.6 Direct PC

Una posibilidad muy novedosa es conectar el computador a la antena de Direct TV para bajar información (el ancho de banda sería muy grande comparado con los de las líneas telefónicas) pero nosotros para enviar información lo haríamos a través de la línea telefónica. Esto se basa en que típicamente el usuario de Internet envía pocos datos y recibe muchos.

12.1.1.7 Telefonía móvil

Son pequeños terminales de Internet. Los anchos de banda que se están desarrollando prometen hacerlo con total comodidad. Como el modelo de red propuesto indica que todos nuestros datos y los programas para operarlos estarán en Internet no debemos preocuparnos por que tengan disco o grandes capacidades de proceso (el procesador también me lo pueden alquilar por tiempo de uso).

12.1.1.8 La telefonía urbana alámbrica

La telefonía urbana permite ya tener servicios donde el usuario decide la configuración de su conexión, por ejemplo: tener un canal de audio y uno de datos o dos de audio y uno de datos. Nosotros contrataremos no ya una línea sino un cierto ancho de banda y definiremos cómo los repartimos. Esta tecnología se denomina ISDN (Integrated Services Digital Network – Red Digital de Servicios Integrados) y en Uruguay está disponible desde el 2000.

Otra alternativa es la posibilidad de que la línea telefónica actual nos permita conectarnos en forma permanente a Internet y al mismo tiempo poder hacer y recibir llamadas telefónicas. Este servicio se denomina ADSL (Asynchronous Digital Subscriber Loop – Bucle de Abonado Digital Asincrónico).

13 Bibliografía Recomendada

Apuntes generados por los docentes de la materia, publicados por el CEI y en la página web del curso

13.1 Básica

Informática Básica, Eduardo Alcalde - Miguel García, Ed. McGraw Hall,
ISBN 84-481-1851-0.

Introducción a la Computación para Ingenieros, Steven C. Chapra, Raymond P. Canale, McGraw-Hill.
ISBN 968-422-487-7.

Informática, presente y futuro, Donald H. Sanders, Tercera edición de McGraw-Hill,
ISBN 0-07-054847-1.

Solución de problemas de ingeniería con Matlab, Delores M. Etter, Ed. Prentice Hall
ISBN 970-17-0111-9.

13.2 De consulta sobre tópicos específicos

Redes de Ordenadores, Andrew S. Tanenbaum, Ed. Prentice Hall,
ISBN 968-880-176-3.

Cableado de Redes. M. Schuartz, Ed. Paraninfo.
ISBN 84-283-2287-2.

Periféricos y Accesorios para la IBM-PC, PS/2 y compatibles, Peter Norton, Ed. Prentice Hall.
ISBN 968-880-321-9.

El libro de Internet, Douglas E. Comer, Ed. Prentice Hall.
ISBN 968-880-537-8.

Sistemas Operativos Modernos, Andrew S. Tanenbaum, Ed. Prentice Hall,
ISBN 968-880-153-4.

Capítulo 2 de Numerical Methods and Software, David Kahaner, Cleve Moler, Stephen Nash. Ed. Prentice-Hall International Editions.
ISBN 0-13-626672-X.

Capítulo 2 de: Numerical Methods, Germund Dahlquist, Åke Björck. Ed. Prentice-Hall Series in automatic computation.
ISBN 0-13-627315-7.

Introduction to Matlab for Engineers, William J. Palm III, Ed. McGraw Hill.
ISBN 0-07-047328-5.

Solución de problemas de ingeniería con Matlab, Delores M. Etter, Ed. Prentice Hall.
ISBN 970-17-0111-9.

Numerical Methods and Software, David Kahaner, Cleve Moler, Stephen Nash. Ed. Prentice-Hall International Editions.
ISBN 0-13-626672-X.

Numerical Methods, Germund Dahlquist, Åke Björck. Ed. Prentice-Hall Series in automatic computation. ISBN 0-13-627315-7.