TECANA AMERICAN UNIVERSITY

Bachelor of Science in Computer Science

INFORME I

Fundamentos de la Informática

Por la presente, doy fe que soy la única autora de esta investigación, y que su contenido es consecuencia de mi trabajo académico.

Eugenia Bahit

Buenos Aires, 19 de Octubre de 2020

ÍNDICE DE CAPÍTULOS

ÍNDICE DE GRÁFICOS

ÍNDICE DE TABLAS

INTRODUCCIÓN Y OBJETIVOS

Las ciencias informáticas representan las bases de la tecnología de la cual hoy día la humanidad depende, de forma no exclusiva, para su desarrollo diario. Sin embargo, lo que hoy se experimenta como tecnología informática dista de ser la aplicación del conocimiento científico de la ciencia que la precede, y se asemeja más a lo que Bunge podría haber considerado una *pseudociencia*. Y si se desea producir tecnología que mejore la calidad de vida de las personas sin generar nuevos problemas, es imperativo profundizar en el conocimiento científico sobre el que se funda la informática, a fin de cumplir con el propósito de toda ciencia: servir para el bien común. Por ello, el objetivo principal de este trabajo es presentar los pilares sobre los que se concibe la informática como ciencia.

Como objetivos específicos, en el primer capítulo se busca determinar la forma en la que un ordenador entiende y procesa la información, mientras que en el segundo, se pretende comprender dicho procesamiento desde la perspectiva de los componentes físicos que lo hacen posible. El tercer capítulo propone explicar la función de los sistemas operativos como gestores de los recursos provistos por dichos componentes, mientras que los capítulos cuatro y cinco persiguen explicar los nexos entre usuarios y ordenadores por un lado, a través de los lenguajes con los que se escriben los programas que le facilitan procesar la información, y por otro, entre las bases de datos y los programas, como herramienta que le facilita la gestión y análisis de la misma.

CAPÍTULO I

FUNDAMENTOS DE LA INFORMÁTICA

El objetivo de este capítulo es introducir los aspectos básicos de la informática, que emanan de las principales teorías sobre las que se construye esta ciencia.

Conceptos básicos

Según la Real Academia Española puede definirse a la ***informática*** como al conjunto de conocimientos científicos y técnicas empleadas para automatizar el procesamiento de la información mediante un ***ordenador***, entendiendo como tal al equipo electrónico que mediante una serie de programas permite el almacenaje, procesamiento y transmisión de la información para resolver problemas específicos. En este contexto, la información puede interpretarse en tres aspectos:

- El aspecto *sintáctico*, que hace referencia a los símbolos con los que se construye un mensaje.
- El aspecto *semántico*, referido al sentido que el mensaje posee.
- Y el aspecto *pragmático*, relativo al uso que se hace del mensaje.

El concepto de información empleado previamente, en lo relativo al tratamiento automatizado y transmisión de mensajes, es el aspecto sintáctico que hace referencia al concepto de información discreta tratado en la Teoría de la Información de Claude E. Shannon. La *teoría de la información* (también

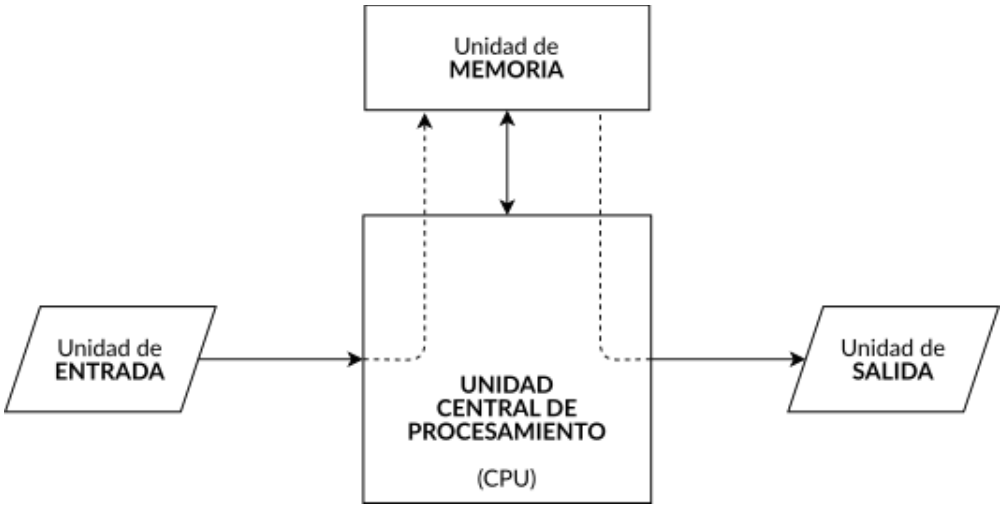
conocida como *teoría de la comunicación*, o *teoría matemática de la comunicación*), es la ciencia que estudia el tratamiento y transmisión de la información, así como los mecanismos para medirla. En esta teoría, la ***información*** se define como el conjunto de símbolos interrelacionados que componen un mensaje, independientemente de su contenido semántico y pragmático.

Funcionamiento y arquitectura del ordenador

Para procesar la información, los ordenadores utilizan un componente denominado *unidad de procesamiento*. Dicho componente tiene la capacidad de ejecutar instrucciones que son almacenadas en una *unidad de memoria*, de forma secuencial.

Cuando la unidad de procesamiento recibe la información, lo hace a partir de una *unidad de entrada*. Para procesar dicha información, recurre a la unidad de memoria para recuperar las instrucciones necesarias para llevar a cabo las operaciones para procesar la información, y almacena tanto la información recibida (a la espera de ser procesada) como la información procesada (a la espera de salida). Finalmente, entrega los resultados computados a una *unidad de salida*, encargada de mostrar la información ya procesada.

Este diseño de cómputo que define la forma en la que estos cuatro componentes se organizan para procesar la información, se conoce como ***arquitectura de Von Neumann***, y se resume simbólicamente en el Gráfico 1.



1. Gráfico: *Diagrama de bloques de la arquitectura Von Neumann. Fuente: elaboración propia.*

Representación interna de la información

Se comentó anteriormente que en términos informáticos, la información es un conjunto de símbolos independiente de su uso y significado. Estos símbolos constituyen la *representación externa* de la información. Dentro de un ordenador, la información que se procesa y almacena, son los *datos* empleados para su *representación interna*.

Cualquier tipo de información, bien sea cadenas de caracteres y números, o bien, imágenes, sonidos o animaciones (video), puede ser representada internamente en un ordenador. Los datos empleados para representar dicha información se generan sobre la base del *sistema binario*, cuya unidad mínima de información es el *bit*, una conjunción abreviada del término *«bi*nary digi*t»*. En términos concretos, el *bit*

es un *dígito binario* tal que solo puede asumir dos valores posibles, el 0 o el 1.

Los datos obtenidos a partir de la transformación de la información a datos binarios se denomina *código binario*. El código binario es una cadena de dígitos binarios combinados. Existen 2^n combinaciones posibles de dígitos binarios para grupos de n bits, por lo que para grupos de 8 bits existen $2^8 = 256$ combinaciones posibles. En un sistema de codificación de caracteres, cada combinación se utiliza para representar un carácter. Por ejemplo, en un sistema de codificación de caracteres que utilice grupos de 3 dígitos binarios para representar las letras vocales mayúsculas del alfabeto castellano, existirían $2^3 = 8$ combinaciones posibles de las cuáles 3 no estarían asignadas, como se muestra en la Tabla 1.

1. Tabla: *Ejemplo de tabla de codificación de caracteres con base 2, para un sistema hipotético de codificación de letras vocales mayúsculas. Fuente: elaboración propia.*

000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
A	E	I	O	U	no asignados		

Pero la misma codificación, podría ser utilizada en otro sistema, para representar las vocales minúsculas, como se muestra en la Tabla 2.

2. Tabla: *Ejemplo de tabla de codificación de caracteres con base 2, para un sistema hipotético de codificación de letras vocales minúsculas que utiliza los mismos códigos del sistema de la Tabla 1. Fuente: elaboración propia.*

000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
a	e	i	o	u	no asignados		

A fin de facilitar el intercambio de datos entre distintos ordenadores, la codificación de caracteres se encuentra estandarizada. Uno de los primeros estándares aparecidos es el *American Standard Code for Information Interchange* (ASCII), que utiliza combinaciones de 7 bits.

1001000	1001111	1001010	1000001
H	O	J	A

2. Gráfico: *Representación binaria de la palabra "HOJA" según el estándar ASCII. Fuente: elaboración propia.*

El estándar más actual es *Unicode*. Dicho estándar asigna un código único a cada uno de los caracteres del total de los alfabetos en el mundo. Para codificar los caracteres emplea el formato UTF (*Unicode Transformation Format*). Este formato utiliza combinaciones de entre 8 y 32 bits. Ofrece compatibilidad directa con el estándar ASCII, reservando los primeros 128 caracteres a los definidos en dicho estándar.

01001000	01001111	01001010	01000001
H	O	J	A

3. Gráfico: *La representación binaria de la palabra "HOJA" según el formato UTF-8 del estándar Unicode, rellena con un 0 a la izquierda para completar la cadena de 8 bits. Fuente: elaboración propia.*

Lógica simbólica

La lógica es la disciplina que estudia la forma del razonamiento. Se encarga de establecer reglas que definen y permiten diferenciar, aquello que es *«formalmente*

correcto», es decir, que su forma encaja con ciertos principios lógicos. Su estudio hace posible establecer la validez formal —de su forma— de enunciados y razonamientos pero no así su veracidad. Esto significa que un enunciado válido no necesariamente es verdadero. Por ejemplo, el siguiente razonamiento es formalmente válido pero su conclusión es falsa así como sus premisas:

Todos los patos son hombres.
Todos los hombres son árboles.
Luego, todos los patos son árboles.

Al sustituir cada uno de los términos por símbolos semánticamente abstractos, la falsedad no se hace tan evidente como su forma:

Todo A es B.
Todo B es C.
Luego, todo A es C.

Por ello, se elige este procedimiento de abstracción para validar enunciados lógicos. Este proceso de abstracción simbólica se conoce como *lógica matemática* (o simbólica) y deriva de la necesidad de simplificar el lenguaje natural para su análisis objetivo, en la lógica de primer orden.

Para cumplir dicho objetivo, la lógica matemática se compone de elementos tales como *variables sentenciales*, utilizadas para sustituir enunciados (letras como p , q , r , …, son de uso habitual aunque cualquier letra es admitida); operaciones lógicas de *conjunción* ($\wedge$) y *disyunción* ($\vee$); y otros símbolos para la *negación* de enunciados y variables ($\neg$), la *agrupación* de sentencias y

enunciados ($(a \wedge b) \vee (\neg b \vee c)$), y para indicar *implicaciones lógicas* ($(a \wedge b) \Rightarrow c$) y doble implicación o *bicondicionalidad* ($(a \wedge b) \Leftrightarrow c$).

En el ámbito del álgebra de conjuntos, se incorporan además, símbolos necesarios para llevar a cabo operaciones de conjunto, pudiendo encontrar llaves para agrupar elementos de un conjunto ($A = \{1, 2, 3, 4, 5, 6\}$), y operadores aritméticos y de comparación como el signo < (menor que), > (mayor que), ≤ (menor o igual que) , ≥ (mayor o igual que), entre otros.

Estas operaciones son evaluadas mediante *tablas de verdad* cuyos encabezados contienen cada uno de los componentes del enunciados y cada fila los diferentes valores de verdad para estos (verdadero o falso, representados por las letras V y F respectivamente).

La tabla 3 muestra la validación del enunciado $P \wedge Q$, tomando separadamente los valores de verdad para P , Q y el valor de verdad para la conjunción de ambos componentes en sus diversas combinaciones.

3. Tabla: *Tabla de verdad. Fuente: elaboración propia.*

P	Q	$P \wedge Q$
V	V	V
V	F	F
F	F	F
F	V	F

CAPÍTULO II

COMPONENTES DE UN ORDENADOR

El objetivo de este capítulo es revisar los componentes de un ordenador desde la perspectiva necesaria para el procesamiento de la información.

Memoria y disco duro

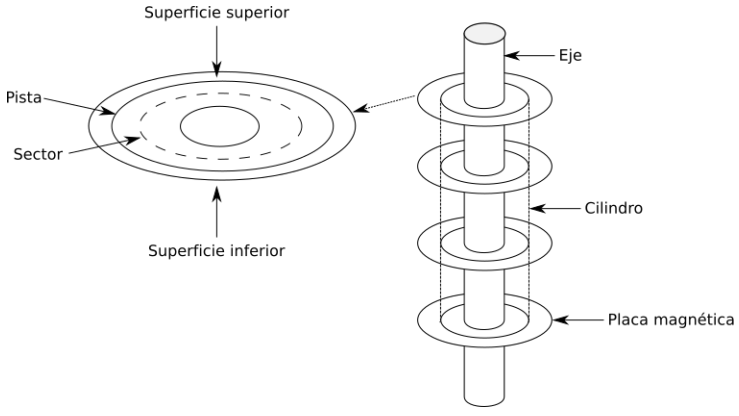
Una unidad de *memoria* consiste en un conjunto de n *celdas* con capacidad para almacenar 1 bit cada una. Las celdas se organizan en grupos interconectados llamados *registros* con capacidad para almacenar cualquier tipo de información binaria de *k*-bits llamadas *palabras*. Por lo general, el tamaño de cada palabra (número de bits) empleado en las unidades de memoria, es múltiplo de 8, lo que se conoce como *byte*. Por ello, la capacidad de almacenamiento de una unidad de memoria en ordenadores comerciales se mide en bytes, puesto que representa la cantidad de palabras (información) que puede ser almacenada.

Tanto si se quiere leer como escribir una palabra, se necesita conocer el registro en el que dicha palabra se aloja (en caso de lectura) o en el que será alojada (en caso de escritura). Para ello, cada registro dispone de un código de identificación denominado *dirección* (al que suele referirse como dirección de memoria).

La cantidad de direcciones de una memoria está determinada por la cantidad de palabras que puedan alojarse. Para una unidad de memoria de i celdas, organizadas en grupos de k bits, se pueden almacenar n palabras (i/k) de

k bits cada una. Dado que solo existen 2 variantes posibles (0 y 1) se necesitarán combinaciones de m bits para conseguir abarcar el total de direcciones necesarias, determinado por la raíz cuadrada del total de palabras tal que $m=\sqrt{(n)}$. Esto se deduce a partir de que con m bits se obtienen 2^m direcciones posibles.

Los ordenadores utilizan dos tipos de memorias: una memoria principal, conocida como **RAM** (*Random Access Memory —Memoria de acceso aleatorio—*). Un tipo de memoria a cuyas celdas se puede acceder para recuperar o escribir información desde cualquier ubicación y sin un orden específico. El tiempo que demanda acceder a dicha celda es independiente de su ubicación, en contraposición a las memorias secuenciales donde la salida de una celda es la entrada de la siguiente; y una memoria de solo lectura, conocida como **ROM** (*Read Only Memory — Memoria de solo lectura—*). El acceso también es aleatorio, pero a diferencia de la RAM, no se destruye tras la lectura. La información en este tipo de memorias es cargada por el fabricante.



4. Gráfico: *Estructura de disco magnético. Fuente: elaboración propia.*

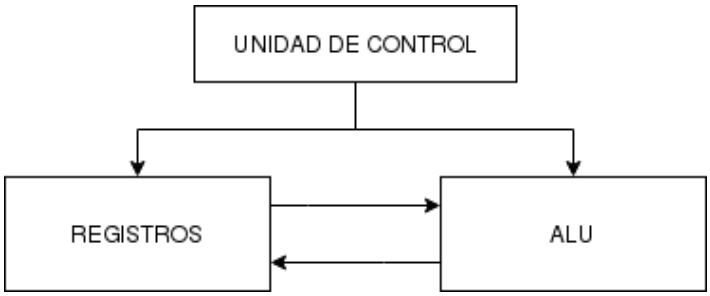
Discos magnéticos: Los discos magnéticos se conforman de unas placas ovales de metal liso. Cada una de ellas se encuentra recubierta en ambas caras, por una lámina magnética. En cada una de estas superficies, se guarda información en pistas concéntricas, mientras que van girando alrededor de un eje montado sobre un motor cuya rotación media es de 7200 rpm. El conjunto de pistas conectadas a lo largo de las placas se conoce como *cilindro*. Cada pista, a su vez, se divide en sectores. Dado que las operaciones de lectura y escritura se llevan a cabo en los límites de un sector, cuando el tamaño de la información a ser almacenada en un sector es menor que el tamaño del sector, el espacio sobrante se rellena con el último byte almacenado. Para calcular la capacidad de almacenamiento total de un disco se debe considerar: la cantidad de bytes que pueden almacenarse por sector (s); la cantidad de sectores que hay en una pista (p); la cantidad de pistas por superficie (t); y la cantidad de superficies (m). Tal que la capacidad en bytes estará determinada por el producto de $s \times p \times t \times m$.

Discos de estado sólido (SSD): Los discos de estado sólido utilizan un sistema basado en el de las memorias *flash*, el cual es similar al de una RAM dinámica (DRAM) pero no volátil. Al no incluir componentes mecánicos, las operaciones de lectura y escritura ahorran el tiempo insumido por los cabezales de los discos magnéticos para hacer girar las placas, posicionarse en el lugar correspondiente y escribir o recuperar la información. Esto hace que además, el consumo de energía sea mucho menor que el de los discos magnéticos.

Unidad central de procesamiento (CPU)

La unidad central de procesamiento (CPU por sus siglas en inglés) es la encargada de llevar a cabo el procesamiento de la información en un ordenador, ejecutando las instrucciones que le son dadas por los programas. Se compone de tres partes:

1. Un *conjunto de registros* cuyos datos se emplean durante la ejecución de instrucciones.
2. Una *unidad lógico aritmética* (ALU, por sus siglas en inglés) que se encarga de llevar a cabo microoperaciones sobre los datos.
3. Una *unidad de control* que supervisa el transporte de la información, el conjunto de registros y la ALU.



5. Gráfico: *Componentes de una CPU. Fuente: elaboración propia.*

La CPU (también llamada *procesador*) puede interpretar y ejecutar un conjunto de códigos y operaciones específicos para el tipo de procesador. Dicho código se compone de cadenas de dígitos binarios con la siguiente estructura: (i) código de operación a ser ejecutada; (ii) la dirección de memoria donde el/los operandos se almacena; (iii) la dirección de memoria donde guardar el resultado de la

operación; y (iv) la dirección de memoria en la que se encuentra la siguiente instrucción. A no ser que ocurra una excepción, la dirección de memoria será consecutiva (si X es la actual, la siguiente es $X+1$).

Unidad de E/S y transporte

Internamente en un ordenador, es necesario transportar la información desde la memoria principal al procesador y a las unidades de salida (S), y desde el procesador y las unidades de entrada (E) a la memoria.

A fin de optimizar los tiempos de transporte de dichos datos, los bits que componen una palabra se transmiten de forma simultánea a través de un grupo de cables al que se conoce como *bus*. Esto implica que el ancho del bus debe ser el mismo que el de las palabras que transporte. A este tipo de bus se lo conoce como *bus de datos*.

Dado que las palabras se almacenan en direcciones específicas de la memoria, otro bus es necesario para transportar dicha dirección. A este se lo conoce como *bus de dirección de memoria* (MAB -*Memory Address Bus*-).

Finalmente, será necesario un bus que transporte las operaciones (comandos tales como `READ`, `WRITE`, `START`, `STOP` y `SEEK` [leer, escribir, comenzar, parar y buscar]) desde el procesador a las unidades de E/S. A este bus se lo conoce como *bus de control*.

CAPÍTULO III

SISTEMAS OPERATIVOS

El objetivo de este capítulo es comprender las funciones de los sistemas operativos y cómo estas son llevadas a cabo para procesar la información. Se abarcarán dichas especificaciones desde su concepción general, y no desde perspectivas particulares de cada sistema, aunque se mencionen algunos ejemplos.

Tipos de sistemas operativos y funciones

En la actualidad, es posible encontrar nueve tipos de sistemas operativos, los cuales se diferencian por el tipo de equipo al que van dirigidos. En orden de magnitud, pueden encontrarse sistemas operativos para los siguientes tipos de equipos computacionales:

Para centrales de cómputo (*mainframe*): destinados a manejar procesos de ordenadores de gran tamaño y capacidad de almacenamiento. Ej: OS/390, Linux.

Para servidores: destinados a distribuir recursos en redes de ordenadores. Ej:Linux, OpenBSD, FreeBSD, Windows Server, Solaris, entre otros.

Para equipos multiproceso: destinados a equipos con múltiples CPUs y/o múltiples núcleos. Ej: Linux, Windows.

Para ordenadores personales (PC): destinados a proveer recursos a un único usuario. Ej: Linux, Windows, OpenBSD, FreeBSD, Mac OS X.

Para ordenadores de mano: destinados a pequeños dispositivos como tabletas, PDAs, teléfonos móviles, etc. Ej: Android, iOS, Replicant, etc.

Sistemas operativos embebidos: destinados a ser alojados directamente en la ROM de equipos que no admiten instalación de programas (como DVDs de salón, reproductores MP3, Tvs, coches). Ej: eLinux (Embedded Linux), QNX.

Para *nodosensores*[1]: destinados a manejar redes de nodos diminutos con sensores ambientales, interconectados por medio de conexiones inalámbricas. Ej: TinyOS.

Sistemas de tiempo real: destinados a manejar procesos de equipos que requieren una respuesta en un tiempo determinado, como aviones, misiles, etc. Ej: eCos.

Para tarjetas inteligentes: destinados a realizar generalmente una única operación (o muy pocas funciones). Son sistemas propietarios[2] que se encuentran en los actuales "chips" de las tarjetas de crédito, débito y de prepago.

En cuanto a **funcionalidad**, los sistemas operativos (SO) se ocupan de: *gestionar procesos y memoria*; *controlar los dispositivos de entrada y salida* (E/S); proveer funciones de manejo para un *sistema de archivos*; y ofrecer mecanismos de *protección* y seguridad para el acceso a los datos. La forma en la que llevan a cabo estas funciones, es a través de *llamadas de sistema (system calls)* que varían de acuerdo a cada SO. Por ejemplo, en GNU/Linux, como Interfaz de Llamada al Sistema (*SCI – System Call Interface*) se emplea la biblioteca de funciones

1 Dado que no se ha podido hallar una traducción al castellano estandarizada para este tipo de sistemas, se emplea el término no oficial "nodosensor" para referirse al término inglés original, Sensor-node.

2 Al tratarse de sistemas propietarios no se mencionan ejemplos puesto que no son sistemas públicamente conocidos.

estándar de C de GNU, `glibc`. El objetivo de utilizar estas llamadas de sistema, es separar el sistema operativo de las aplicaciones de usuario.

GESTIÓN DE PROCESOS: Comúnmente, se entiende a un proceso como a un programa en ejecución. Para entender la diferencia entre programa y proceso, puede hacerse una analogía con la persona que compra un mueble para armar. En la caja, puede encontrar las instrucciones de cómo debe ser armado y los materiales con los cuáles hacerlo. La persona lee las instrucciones y las va ejecutando empleando los materiales hasta lograr un mueble listo para utilizar. En esta analogía, el papel con las instrucciones es el programa. Una entidad pasiva que solo se utiliza a modo de guía. Los materiales, son los datos de entrada. La persona, es el procesador. La acción de seguir las instrucciones para armar el mueble, es el proceso. Y el mueble listo para usar, los datos de salida.

Si dos personas armasen el mismo mueble, leyendo las mismas instrucciones, entonces habría dos procesos en ejecución de un mismo programa, llevados a cabo por dos procesadores. Si un vendaval golpeara las alas de la venta de la habitación donde se está armando el mueble, probablemente una de las personas decidiría que es más prioritario correr a cerrar la ventana, por lo que daría lugar de entrada a un nuevo proceso, dejando en pausa el anterior. Cuando la tarea prioritaria (cerrar la venta) hubiera acabado, retomaría la del armado del mueble. De esa forma, el procesador decide qué procesos y en qué momento, darles prioridad. Es decir, que administra los recursos para permitir la multitarea.

GESTIÓN DE MEMORIA: Dado que un único CPU puede ejecutar un único proceso a la vez pero que múltiples programas requieren de múltiples procesos, a cada proceso se le asigna su propio espacio en la memoria para que sean alojadas las instrucciones que serán ejecutadas, así como los datos necesarios, hasta tanto el proceso se lleve a cabo y deje de ser necesario. A fin de que cada proceso no interfiera con otro, el hardware provee mecanismos de protección que son controlados por el sistema operativo.

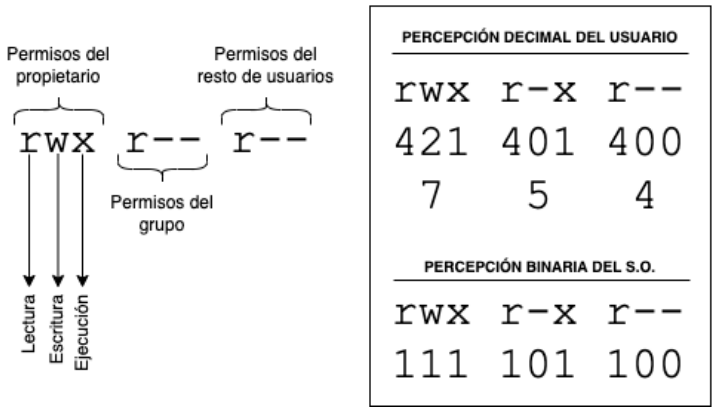
Cada proceso tiene una cantidad limitada de direcciones de memoria que puede utilizar, y por lógica, el máximo siempre debería ser inferior que la cantidad total disponible. Sin embargo, en las actuales arquitecturas de 32 y 64 bits, existe un espacio total de 2^{32} y 2^{64} *bytes* respectivamente, por lo que si un proceso necesitase más espacio de memoria, no podría hacerlo en la memoria principal. Para resolver este inconveniente, el sistema operativo utiliza una técnica conocida como *virtualización de memoria*, en la que mantiene una parte de las direcciones asignadas en la memoria principal, y otra en el disco y las va lanzando y realojando, en la medida que sean necesarias.

SISTEMA DE ARCHIVOS: La forma en la que los archivos se almacenan en los dispositivos de almacenamiento (hardware), es definida y controlada por el sistema operativo quien para lograrlo, ofrece un modelo abstracto e independiente del hardware. Para ello, la mayoría de los sistemas operativos para ordenadores personales, emplea el concepto de *directorio* y lo hace siguiendo una estructura jerárquica. El sistema operativo realiza las llamadas de sistema necesarias para las

operaciones de lectura y escritura de archivos, como abrir y cerrar archivos, alojarlos en disco, crear, eliminar, leer y escribirlos.

ENTRADA Y SALIDA: Todos los sistemas operativos poseen un subsistema de manejo para los dispositivos de entrada y salida. En algunos casos el programa de manejo será independiente del hardware y otros, como es el caso de los controladores, será específico para cada pieza de hardware.

PROTECCIÓN: Desde el momento en el que los ordenadores manejan archivos con información privada y/o confidencial de los usuarios, el sistema operativo debe proveer de mecanismos que controlen el acceso a dichos archivos. La protección puede estar orientada a todo el sistema (por ejemplo, cuando se protege el acceso al sistema mediante autenticación) y completada por una protección adicional de permisos sobre los archivos. En el gráfico 6 se muestra el sistema de permisos basado en un código de protección de 9 bits implementado en UNIX.



6. Gráfico: *Sistema de permisos implementado en sistemas Linux, UNIX y BSD. Fuente: elaboración propia.*

CAPÍTULO IV

FUNDAMENTOS DE PROGRAMACIÓN

El objetivo de este capítulo es abarcar los mecanismos que intervienen en el proceso de programación como análisis, abstracción de problemas y diseño de algoritmos, y los elementos y estructuras que componen a los lenguajes.

Análisis de problemas y diseño de algoritmos

En el ámbito de la ingeniería de software, un ***problema*** es un requerimiento concreto que necesita ser resuelto mediante un programa informático. Mientras que su ***análisis*** consiste en entender qué se necesita hacer para determinar cómo hacerlo, mediante qué operaciones, con cuáles valores de entrada y para arrojar cuáles valores de salida. Los ***algoritmos*** constituyen el paso a paso de la solución del problema.

Como características principales de los algoritmos, puede decirse qué: comienzan aceptando los datos de entrada que serán procesados; las instrucciones definidas deben ser completas, precisas y concretas; el tiempo total empleado por un algoritmo para llevar a cabo sus operaciones debe ser finito, así como la cantidad de instrucciones y el número de repeticiones cuando las hubiera; finalmente, debe producir al menos un resultado (valor de salida). Un factor clave de la programación es diseñar los algoritmos con una completa abstracción del lenguaje de programación, como se muestra en el ejemplo 1.

Ejemplo 1	*Análisis de registros del servidor HTTP de Apache.* *Fuente: elaboración propia*

Problema:

Se necesita visualizar una pila con las entradas del registro de errores del archivo *errors.log* del servidor HTTP de Apache del día jueves 20 de agosto de 2020. Cada entrada del registro es una línea que comienza por `[Thu Aug 20, 2020 <hora>,` aunque se incluyen registros de otras fechas.

Valor de entrada:

Archivo de registros (errors.log)

Valor de salida:

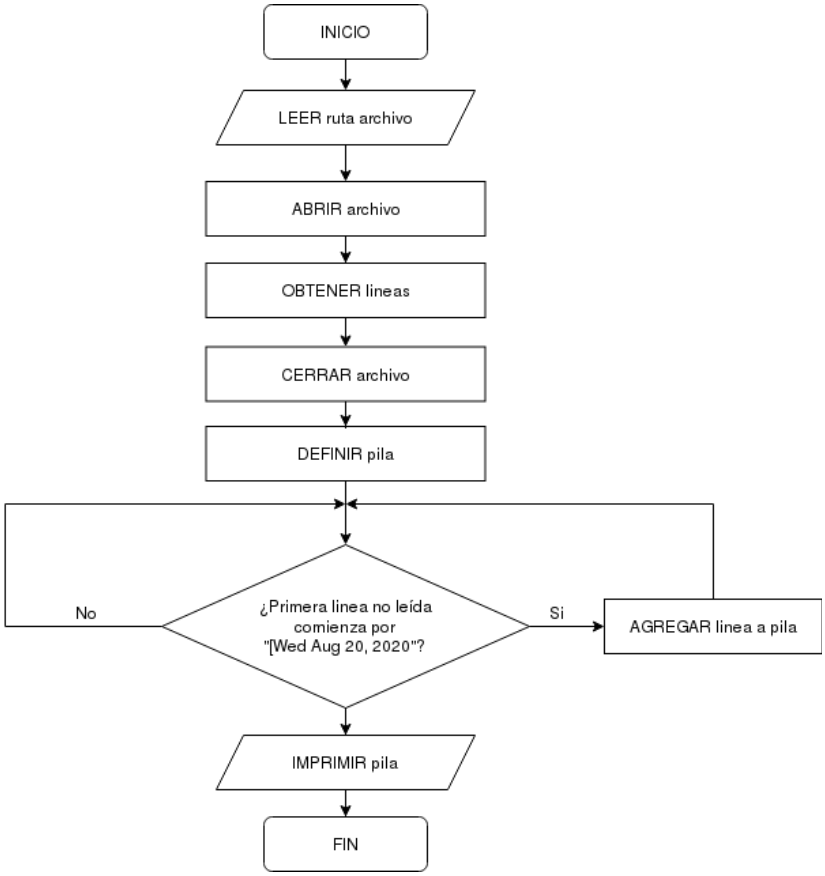
La pila con los registros del jueves 20 de agosto de 2020.

Procedimiento:

```
1.  INICIO
2.  LEER     ruta del archivo
3.  ABRIR    archivo
4.  OBTENER  líneas del archivo leído
5.  CERRAR   archivo
6.  DEFINIR  pila
7.  EVALUAR  ¿1ra línea no leída comienza por
             "[Wed Aug 20, 2020"? SI: ir a 7.
8.  AGREGAR  línea a pila
9.  Repetir 6 y 7 hasta que no queden más líneas sin leer
10. IMPRIMIR la pila en pantalla
11. FIN
```

Para facilitar su comprensión, un algoritmo puede representarse en un *diagrama de flujo* como el que se muestra en el gráfico 7. Estos diagramas emplean diversas figuras tales como: *paralelogramos*, para los valores de entrada y salida; *rectángulos*, para las operaciones; *rombos*, para los condicionales; y *rectángulos con bordes redondeados*, para indicar inicio y fin del algoritmo. Posteriormente, en la traducción a código fuente, aquellas operaciones no disponibles de forma nativa en el lenguaje, requerirán el diseño de un nuevo algoritmo. El gráfico 7

muestra el diagrama de flujo del algoritmo del ejemplo 1 y a continuación, su traducción a código fuente en lenguaje Python.



7. Gráfico: *Algoritmo para lectura y análisis de errores del servidor HTTP de Apache. Fuente: elaboración propia.*

Ejemplo 2	*Código fuente del algoritmo del Gráfico 7, en lenguaje Python. Fuente: elaboración propia.*

```
def obtener_registros(ruta_archivo):
    archivo = open(ruta_archivo, "r")
    lineas = archivo.readlines()
    archivo.close()
    pila = []
    for linea in lineas:
        if linea.startswith("[Wed Aug 20, 2020"):
            pila.append(linea)

    print(pila)
```

El *código fuente* es el conjunto de órdenes escritas en un determinado lenguaje de programación, que componen un programa. Los *lenguajes de programación* son lenguajes informáticos diseñados para que un ordenador pueda llevar a cabo operaciones concretas. Los *lenguajes informáticos* son lenguajes artificiales creados exclusivamente para ser entendidos por los ordenadores. Los de programación son los que permiten la ejecución de órdenes pero los hay de otros tipos. Se manejan en dos niveles básicos: *bajo nivel*, que se refiere al código máquina (código binario) y los de *alto nivel* que poseen una estructura humanamente legible e independiente del hardware.

Los hay *interpretados* (sus instrucciones son ejecutadas directamente sobre el sistema operativo por un intérprete) y *compilados* (aquellos que requieren ser traducidos a un lenguaje intermedio y empaquetados para su ejecución). Algunos funcionan en diversas plataformas y otros son exclusivos de sistemas concretos. Algunos aceptan solo un paradigma (técnica) de programación, mientras que otros, pueden aceptar más de uno.

Elementos y estructuras de los lenguajes

Si bien al momento de traducir un algoritmo a código fuente la sintaxis empleada dependerá exclusivamente del lenguaje de programación elegido, todo lenguaje posee una serie de elementos comunes —a todos los lenguajes— que se encuentran disponibles. Solo la sintaxis del lenguaje es lo que cambia. Por ejemplo, mientras que una variable *foo* de tipo entero con valor 15 , en Python

se define mediante `foo = 15`, en PHP se define anteponiendo un signo dólar al nombre de la variable y un punto y coma tras el valor, tal que `$foo = 15;` y en C, requiere la declaración de tipo, tal que `int foo = 15;`. Los elementos comunes a todos los lenguajes se mencionan a continuación.

Tipos de datos: los tipos de datos comunes son los números de base decimal (enteros y reales), cadenas de texto, y los valores de verdad (verdadero o falso). Algunos lenguajes como Python poseen un soporte dinámico para el tipo de datos (esto significa que al modificar el valor de una variable es posible también modificar su tipo). Otros lenguajes como C, poseen un soporte más fuerte para el tipo de datos (esto significa que las variables se encuentran protegidas de cambios accidentales en su tipo).

Ejemplo 3	*Tipos de datos en Python. Fuente: elaboración propia.*

```
entero = 15
real = 15.5
booleano = True
cadena_de_texto = "Hola mundo!"
```

Variables: las variables son nombres destinados a almacenar tipos de datos concretos, temporalmente en la memoria. Son ejemplos de variables los nombres *entero*, *real*, *booleano* y *cadena_de_texto* que se muestran en el ejemplo 3.

Operadores aritméticos: símbolos necesarios para llevar a cabo operaciones aritméticas básicas como la adición, sustracción, multiplicación y división, entre otras.

Ejemplo 4	*Operadores aritméticos en Python. Fuente: elaboración propia.*

```
adicion = 15 + 7.3
sustraccion = 15 - 7.3
division = 15 / 7.3
multiplicacion = 15 * 7.3
```

Operadores lógicos: símbolos necesarios para llevar a cabo las operaciones lógicas de conjunción, disyunción exclusiva e inclusiva, y negación.

Ejemplo 5	*Operadores lógicos en Python. Fuente: elaboración propia.*

```
conjuncion = (1 and 1)
disyuncion_inclusiva = (1 or 1)
disyuncion_exclusiva = (1 ^ 1)
negacion = not (1 and 0)
```

Operadores relacionales: símbolos empleados para evaluar la relación entre dos variables, tales como la igualdad, menor y mayor qué, distinto, entre otros.

Ejemplo 6	*Operadores relacionales en Python. Fuente: elaboración propia.*

```
igual_que = (1 == 0)
menor_que = (1 < 0)
mayor_que = (1 > 0)
distinto_que = (1 != 0)
menor_o_igual_que = (1 <= 0)
mayor_o_igual_que = (1 >= 0)
```

Comentarios: se trata de una forma de escribir "mensajes" no interpretados dentro del propio código fuente.

Ejemplo 7	*Comentarios en Python. Fuente: elaboración propia.*

```
# Esto es un comentario de una línea
variable = 1  # Esto es un comentario en línea
"""
    Esto es un comentario en bloque
```

```
    que puede ocupar más de una línea.

    También pueden emplearse tres comillas simples
    en lugar de tres comillas dobles.
"""
```

Estructuras de control de flujo: se trata de instrucciones que permiten alternar y controlar la ejecución de sentencias. Pueden encontrarse estructuras de control condicionales para evaluar sentencias contrarias (si / sino) o contradictorias (si / sino si), e iterativas (aquellas que repiten acciones). También se incluyen estructuras de control secuencial (funciones) y de control de excepciones, de las cuales se hablará y ejemplificarán posteriormente.

Ejemplo 8	*Estructuras de control condicional. Fuente: elaboración propia.*

```
# Evaluación de enunciados opuestos
if color == 'verde':
    print("Acción a realizar toda vez que el valor de",
        "la variable 'color' sea 'verde'.")
else:
    print("Acción a realizar toda vez que el valor de",
        "la variable 'color' NO sea 'verde'.")


# Evaluación de enunciados contradictorios
if color == 'verde':
    print("Acción a realizar toda vez que el valor de",
        "la variable 'color' sea 'verde'.")
elif color == 'amarillo':
    print("Acción a realizar toda vez que el valor de",
        "la variable 'color' sea 'amarillo'.")
elif color == 'rojo':
    print("Acción a realizar toda vez que el valor de",
        "la variable 'color' sea 'rojo'.")
else:
    print("Acción a realizar en cualquier otro caso.")
```

Ejemplo 9	*Estructuras de control iterativas. Fuente: elaboración propia.*

```
# Estructura 'while'
```

```
variable = 0
while variable < 10:
    print(variable * 10)
    variable += 1


# Estructura 'for'
for numero in range(0, 10):
    print(variable * 10)
```

Funciones: se trata tanto de operaciones incorporadas (disponibles de forma nativa) en el propio lenguaje, como de la posibilidad de definir operaciones propias. Las funciones son estructuras de control secuencial.

Ejemplo 10	*Funciones en Python. Fuente: elaboración propia.*

```
# Función definida por el usuario
def get_iva(bruto, alicuota):
    return bruto * alicuota / 100

# Función propia del lenguaje (print) que imprime el
# valor de retorno de la función definida 'get_iva'
print(get_iva(1500, 19))
```

Tokens: los tokens son palabras claves reservadas que en el lenguaje tienen un significado concreto y no pueden emplearse si no es para el fin con el que han sido creadas. Un ejemplo de ello es la instrucción empleada para retornar datos en las funciones ('return' en el ejemplo 10).

Captura de excepciones: se trata de estructuras de control que permiten capturar el momento en el que una excepción (error fatal) ocurre en el código y actuar en consecuencia.

Ejemplo 11	*Control de excepciones en Python. Fuente: elaboración propia.*

```
try:
    bruto = float(bruto)
except:
    bruto = 0
    print("'bruto' no puede ser convertido a número real")
finally:
    iva = get_iva(bruto, 19)
```

En el ejemplo 11 se intenta convertir la variable 'bruto' a número real (bloque *try*). Si no se puede, se establece el valor de la variable bruto en cero y lanza un mensaje de error (bloque *except*), finalmente, se calcula el valor del iva (bloque *finally*, opcional en Python).

Lectoescritura de archivos: informáticamente, la lectura y escritura de archivos se lleva a cabo mediante la apertura de un puntero en memoria, sobre el cual se realizan las diferentes operaciones de lectura y escritura, y finalmente, se cierra dicho puntero. Sin embargo, algunos lenguajes ofrecen procedimientos abreviados para estas tareas, como es el caso de Python, PHP, o Ruby, posiblemente entre otros.

Ejemplo 12	*Método abreviado de lectoescritura de archivos en Python. Fuente: elaboración propia.*

```
# Lectura de archivos
with open("/path/to/file", "r") as archivo:
    contenido = archivo.read()

# Escritura de archivos
with open("/path/to/file", "w") as archivo:
    archivo.write("contenido a escribir en el archivo")
```

Operaciones de entrada y salida (E/S): se refiere a las formas en las que el lenguaje presenta la información en pantalla (operaciones de salida) e interactúa con el usuario para recibir información (operaciones de entrada).

Ejemplo 13 *Operaciones de E/S en Python. Fuente: elaboración propia.*

```
bruto = input("Ingrese importe bruto: ")  # Entrada
iva = get_iva(bruto, 19)
print("El IVA de", bruto, "es", iva)      # Salida
```

Colecciones: las colecciones son en realidad un tipo de variable que permite almacenar más de un datos. En algunos lenguajes como en Python, las colecciones pueden almacenar datos de diferentes tipos. En los lenguajes de tipado fuerte (aquellos cuyas variables deben mantener el tipo de datos con el que han sido definidas), las colecciones almacenan datos de un único tipo, como es el caso de C. En el ejemplo 14 se muestran los tres tipos de colecciones que ofrece Python.

Ejemplo 14 *Colecciones en Python. Fuente: elaboración propia.*

```
# listas (pueden modificarse)
lista = [1, 2, 3, 'cadena de texto', [9, 99, 999], True]
print(lista[2])  # Salida: 3
lista[4][1] = 0.99  # Ahora 99 es 0.99
lista.append(1500)
print(lista[-1])  # Salida: 1500

# tuplas (NO pueden modificarse)
tupla = (1, 2, 3, 'cadena de texto', [9, 99, 999], True)
print(tupla[2])  # Salida: 3
tupla[0] = 9  # Dará error
tupla[4][1] = 0.99  # No dará error ya que tupla[4][1]
                    # es una lista

# Diccionarios (valores asociados a nombres de claves)
diccionario = {'color': 'rosa', 'hilos': 800}
print(diccionario['color'])
diccionario['hilos'] = 600
diccionario['material'] = 'satén'
```

CAPÍTULO V

FUNDAMENTOS DE BASES DE DATOS

El objetivo de este capítulo es hacer un breve recorrido por los conceptos básicos, arquitecturas, modelos de datos y lenguajes de consulta sobre los que se fundamentan las bases de datos.

Conceptos básicos y arquitecturas

Se puede definir una ***base de datos*** como un conjunto organizado de datos relacionados, cuyo procesamiento permite la obtención de información en un mismo contexto. El conjunto de programas, herramientas, bibliotecas y lenguajes que permiten el procesamiento de esos datos, es conocido como ***Sistema de Gestión de Bases de Datos*** (SGBD).

Los SGBD (o por sus siglas en inglés, *DBMS*), emplean una ***arquitectura cliente-servidor*** donde las funcionalidades del sistema de gestión se reparten en dos módulos: un ***módulo cliente***, que facilita la visualización de los datos, y un ***módulo servidor***, que se encarga de gestionar el almacenamiento de los datos y proveer de los mecanismos necesarios para acceder a los datos y procesarlos.

En los SGBD primitivos, ambas funcionalidades se encuentran *centralizadas* en el mismo equipo de cómputo. En arquitecturas más complejas, diferentes clientes se conectan mediante una red a un mismo servidor. Esto se conoce como *arquitectura cliente-servidor básica*. En arquitecturas aún más complejas donde se

manejan volúmenes de datos mucho mayores, también los servidores se encuentran distribuidos en múltiples equipos. Esto se conoce como *arquitectura distribuida*.

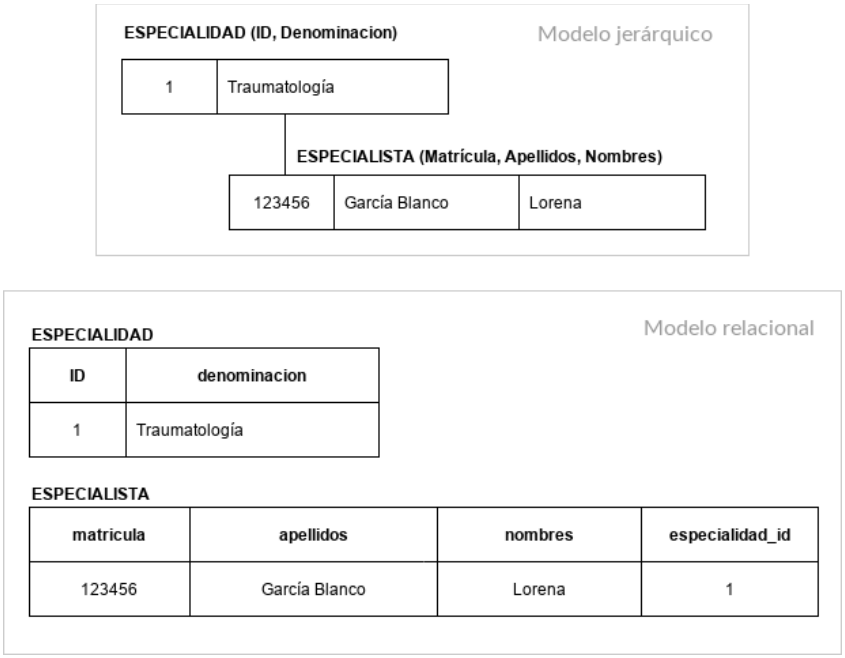
Modelos de datos

Un *modelo de datos* es un concepto abstracto que define la forma en la que los datos son representados y utilizados. Existen tres **niveles de abstracción**:

1. El nivel más bajo de abstracción, es aquel que define la forma en la que los datos son organizados y almacenados internamente. En este nivel, se encuentran los ***modelos de datos físicos*** que aíslan por completo estos detalles de los datos representados.
2. En el nivel opuesto, el de mayor abstracción, se encuentran los ***modelos de datos conceptuales*** que definen conceptos del mundo real (entidades), sus características distintivas (atributos), y la forma en la que se relacionan entre sí (relaciones). El modelo más popular en este nivel, es el *modelo entidad-relación (ER)*.
3. Finalmente, en un nivel intermedio, se encuentran los ***modelos de datos figurativos***[3] cuyo objetivo es definir la estructura de los datos, ofreciendo un mayor detalle de estos. En este nivel intermedio pueden encontrarse cuatro tipos de modelos, entre los que se destacan el *modelo relacional* y

3 Habitualmente se emplea en castellano el término «*representacional*». El mismo es una deformación de traducción del término inglés «*representational*». Según la definición de dicho término en el Diccionario de Cambridge, compartiría significado con el término «figurativo» según el diccionario de la Real Academia Española.

su sucesor, el *modelo de datos basado en objetos*, y con menor frecuencia sus antecesores, el modelo de datos jerárquico y el modelo de datos en red. En el gráfico 8 se puede apreciar la diferencia entre un modelo de datos jerárquico y uno relacional, para el mismo esquema de base de datos.



8. Gráfico: *Modelo de datos jerárquico vs modelo de datos relacional. Fuente: elaboración propia.*

Sobre el modelo entidad-relación: Puede definirse una *entidad* como una cosa de existencia real o imaginaria, y a sus *atributos,* como aquellas propiedades que la describan. En el sentido más puro, no puede considerarse atributo a aquel dato "necesario" si este no describe a la entidad que define.

Cada uno de estos atributos tiene un valor determinado para cada entidad. Estos pueden ser *compuestos* si es posible dividirlos en partes más pequeñas, o *atómicos* en caso contrario. Un ejemplo de valor compuesto es el atributo *domicilio* de una entidad física o jurídica, el cual puede dividirse en los atributos *calle*, *numeración*,

planta, *puerta*, etc. Cada atributo simple de una entidad particular, se encuentra asociado a un *dominio de valores* que determina los valores posibles de dicho atributo para la entidad correspondiente. Matemáticamente, un atributo A de una entidad E cuyo conjunto de valores posibles se denota por V, se define por una función de E que resulta en el conjunto de potencia P de los subconjuntos que conforman V, tal que $A : E \rightarrow P(V)$.

En las asociaciones entre entidades, pueden encontrarse diversos tipos de relaciones, conjuntos de relaciones e instancias de relaciones. Un *tipo de relación* R entre n tipos de entidades ($E_1, E_2, \ldots, E_n$) define un *conjunto de relaciones* entre entidades de esos tipos. Matemáticamente, el conjunto de relaciones R es un conjunto de *instancias de relación* r_i que asocian entidades individuales ($e_1, e_2, \ldots, e_n$) y donde cada una de estas entidades individuales e_j en la instancia de relación r_i es un miembro del conjunto de entidades E_j para el que se cumple que j es mayor o igual a 1 y menor o igual a n, tal que $1 \leq j \leq n$.

Lenguaje de consulta estructurado (SQL)

SQL es el lenguaje que se emplea para acceder a las bases de datos y manipular tanto su estructura como la información almacenada. Posee un estándar definido que se centra en tres componentes: **DML** (parte del lenguaje que permite insertar, recuperar, modificar y eliminar datos de una base de datos), **DDL** (empleado para manipular la estructura de la base de datos), y **DCL** (parte del lenguaje destinada al control de acceso a los datos).

En el ejemplo 15 se muestran instrucciones en el lenguaje SQL de los SGBD MariaDB y MySQL® para estos tres componentes. Las dos primeras crean una base de datos y dos tablas respectivamente. Las instrucciones del bloque DCL crean un usuario al que le otorgan permisos completos para acceder y manipular las tablas de dicha DB. Finalmente, un ejemplo para insertar, seleccionar, actualizar y eliminar datos respectivamente.

Ejemplo 15	*Ejemplos del lenguaje de consulta SQL empleado por los SGBD MariaDB y MySQL®. Fuente: elaboración propia.*

```
# DDL (lenguaje de definición de datos)
CREATE DATABASE agenda_medica;

CREATE TABLE agenda_medica.especialidad (
  especialidad_id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  denominacion VARCHAR(25) NOT NULL
) ENGINE=InnoDB;

CREATE TABLE agenda_medica.especialista (
  especialista_id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  apellidos VARCHAR(50) NOT NULL,
  nombres VARCHAR(50),
  especialidad_id INT(11) NOT NULL,
  FOREIGN KEY (especialidad_id)
    REFERENCES especialidad(especialidad_id)
) ENGINE=InnoDB;

# DCL (lenguaje de control de datos)
CREATE USER gestion IDENTIFIED BY 'supersecretpassword';
GRANT ALL ON agenda_medica.* TO gestion;

# DML (lenguaje de manipulación de datos)
INSERT INTO  especialidad(denominacion)
VALUES       ('Traumatología'), ('Cirugía'), ('Psiquiatría');

SELECT especialidad_id WHERE denominacion = 'Traumatología';

UPDATE  especialidad
SET     denominacion = 'Cirugía General'
WHERE   especialidad_id = 2;

DELETE FROM especialidad WHERE especialidad_id > 2;
```

CONCLUSIONES

De los capítulos precedentes es posible afirmar que a lo largo de estos se ha cumplido con el objetivo general de exponer los pilares sobre los que se fundamentan las ciencias informáticas, y que a partir de los mismos es posible llegar a las siguientes conclusiones:

A) Que respecto al primer objetivo específico, del capítulo uno se desprende que informáticamente, la información es representada a través de combinaciones de bits; que en sentido concreto, estos se representan a su vez por variaciones de carga eléctrica; y que los mismos, son analizados mediante operaciones lógicas y que por lo tanto, su estudio constituye un factor indispensable para comprender cómo los ordenadores analizan la información y ejecutan decisiones a partir de dicho análisis.

B) Que las mencionadas operaciones lógicas, según se deduce del segundo capítulo, se realizan de forma coordinada por una unidad central de procesamiento (CPU) que recibe y envía la información desde y hacia una memoria principal volátil (RAM) y que su persistencia requiere de dispositivos con capacidad de almacenar las nombradas cargas eléctricas, como por ejemplo, los discos duros.

C) Que en cuanto al objetivo específico de explicar el funcionamiento de los sistemas operativos como gestores de los recursos mencionados, del capítulo tres se desprende que es tarea de los mismos, tanto la asignación de memoria a cada proceso en ejecución, como la habilidad para proveer mecanismos de protección

que impidan que diferentes procesos interfieran en espacios de memoria ajenos; que además, deben proveer mecanismos para hacer persistir y manipular la información persistida en las unidades de almacenamiento, así como interfaces para que los programas puedan interactuar con el hardware.

D) Que en cuanto al objetivo específico de comprender los lenguajes de programación que permiten escribir los programas que facilitan el procesamiento de la misma, tal como se menciona en el capítulo cuatro, se desprende que los procesos de diseño y análisis de problemas son esenciales para escribir instrucciones claras, precisas y concretas, y que si bien dichas instrucciones responden a la sintaxis de cada lenguaje en particular, los mismos ofrecen elementos que son comunes a todos.

E) Que finalmente, en relación al objetivo específico de establecer un nexo entre las bases de datos y los programas, como herramientas que facilitan al usuario el análisis y gestión de la información, si bien el capítulo quinto permite establecer dicha relación e inducir la importancia que las mismas representan al usuario actual a la hora de comprender la información que desea procesar, del mismo también se deduce por lo concluido en capítulos previos, que las bases de datos no constituyen un fundamento de las ciencias informáticas, sino una herramienta reemplazable como cualquier otro programa, que puede ser tanto mejorado como sustituido por una u otra tecnología, sin que ello ponga en pugna alguna de las bases sobre las que la informática se sostiene como ciencia.

BIBLIOGRAFÍA

[0] Bansal, P. (2020). *Operating Systems* (6th ed., pp. 1-116). Meerut: Satyendra Rastogi Mitra.

[1] Darie, C., & Watson, K. (2003). *The Programmer's Guide to SQL* (1st ed., pp. 1-7). New York: Apress.

[2] Elmasri, R., & Navathe, S. (2017). *Fundamentals of Database Systems* (7th ed., pp. 31-50, 63-68, 72, 86, 149-157, 367-368). India: Pearson India.

[3] Gregersen, E. et al. *Database*. Encyclopædia Britannica. (2020). Recuperado el 19 de septiembre de 2020, de https://www.britannica.com/technology/database.

[4] Gupta, M. (2020). *Discrete Mathematics* (19th ed., pp. 1-59). Meerut: Satyendra Rastogi Mitra.

[5] ITL Education Solutions Limited (2004). *Introduction to Computer Science* (1st ed., pp. 366-389). Nueva Deli: Pearson Education.

[6] Mano, M (1993). *Computer System Architecture* (3rd ed., pp. 50-58). New Jersey: Pearson Education.

[7] Nisan, N. & Schocken, S. (2008). Chapter 5. En *The Elements Of Computing Systems* (Kindle Edition). Cambridge, Masachuset: MIT Press.

[8] Oxford Research Centre Ltd, & Bahit, E (2020). *Python para principiantes, edición 2020* (2nd ed., pp. 9-27). London: Oxford RC Publisher.

[9] Rajaraman, V. & Adabala, N. (2015). *Fundamentals Of Computers* (6th ed.). Deli: PHI Learning.

[10] Real Academia Española. *Informática*. Diccionario de la Lengua Española (23rd ed.). (2020). Recuperado el 19 September de 2020, de https://enclave.rae.es/recursos/diccionarios/dle/palabras/informática.

[11] Shannon, C. & Weaver, W. (1999). *The Mathematical Theory Of Communication* (1st ed., pp. 3-9). Urbana: University of Illinois Press.

[12] Stallings, W. (2013). *Computer Organization And Architecture* (9th ed., pp. 17-19). New Jersey: Prentice Hall.

[13] Tanenbaum, A. & Bos, H. (2015). *Modern Operating Systems* (4th ed., pp. 35-73). Amsterdam: Pearson.

[14] Van Der Lubbe, J. (1997). *Information Theory* (1st ed., p. 1). Netherlands: VSSD and Cambridge University Press.

[15] Villar, J. R. (2015). *Fundamentos De Informática (1st ed., pp. 6-8, 16-25).* Oviedo: Departamento de Informática, Universidad de Oviedo. Recuperado el 26 de agosto de 2020 de: https://bit.ly/3gBVkM8