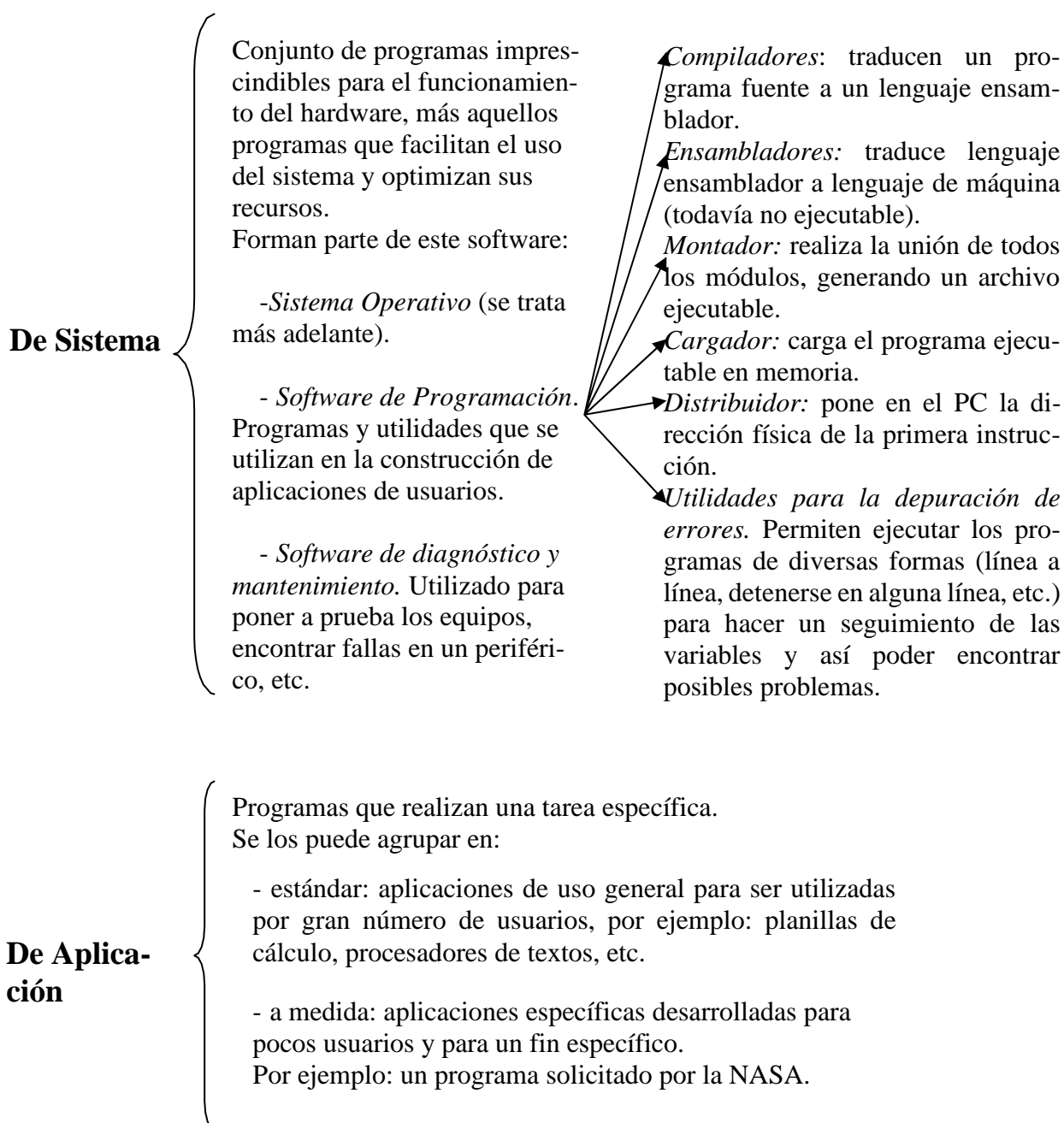


1. Introducción.

El software está constituido por un conjunto de elementos intangibles llamados programas que el hardware ejecuta.

Dentro del software se incluyen: lenguajes de programación, interfaces de usuarios, sistemas operativos, las herramientas o utilidades, etc.

Una primera clasificación nos permite distinguir dos grandes grupos:



2. Archivos

2.1 Conceptos básicos.

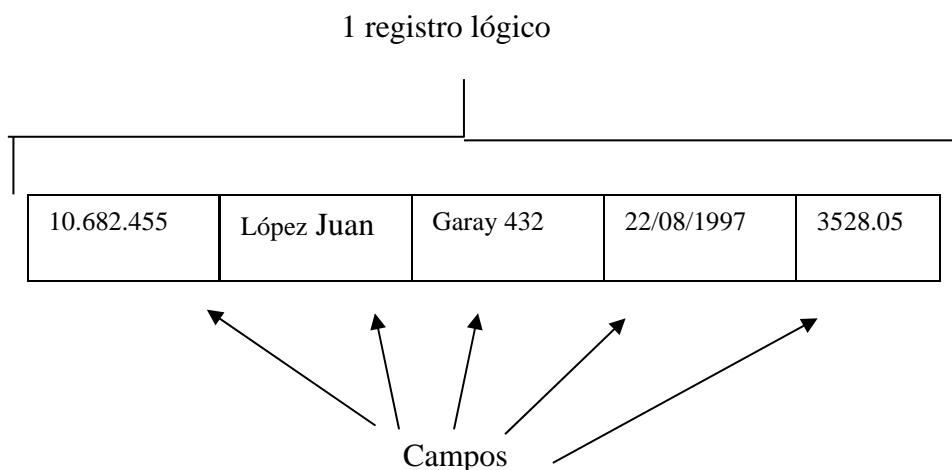
La necesidad de almacenar y procesar grandes volúmenes de datos utilizando como herramienta el computador dio lugar al uso de los denominados ficheros o archivos informáticos.

Un archivo o fichero es un conjunto ordenado de datos que tienen entre sí una relación lógica y residen en soportes de información, también llamados memorias secundarias auxiliares.

Un archivo está compuesto por estructuras de datos más simples llamadas **registros**. Todos los registros de un archivo son del mismo tipo, es decir, tienen la misma estructura.

Cada registro está formado por **campos**, los cuales pueden ser de diferentes tipos, conteniendo información referente a una característica en particular dentro del archivo.

Por ejemplo en un archivo de personal, cada registro contiene los campos con los datos de cada empleado (número de DNI, nombre y apellido, dirección, fecha de ingreso, sueldo, etc.).



Se llama **registro lógico** al conjunto de información identificable acerca de uno de los elementos del archivo.

Se llama **registro físico o bloque** a la cantidad de información que se transfiere físicamente en cada operación de acceso (lectura o escritura) sobre el archivo.

2.2 OPERACIONES CON FICHEROS

CREACIÓN	Primera operación sobre un fichero donde se describen los datos y sus características. Se diseña el archivo.
CONSULTA O RECUPERACIÓN	Se realiza a nivel de registro para obtener la información contenida en él, por ejemplo consultar el domicilio de un empleado.
MANTENIMIENTO O ACTUALIZACIÓN	Una vez creado el archivo puede ser necesario realizar distintas operaciones a nivel de registro: <ul style="list-style-type: none"> • <i>Inserción</i> de un registro nuevo, por ejemplo: se incorpora un empleado nuevo. • <i>Modificación</i> de un registro por cambios en uno o varios campos del mismo, por ejemplo: cambia el domicilio de un empleado. • <i>Eliminación o borrado de un registro</i>, por ejemplo un empleado que se da de baja. Puede hacerse de 2 formas: <ol style="list-style-type: none"> 1. Por marca o borrado lógico: colocar en un campo un valor que será interpretado por los programas de aplicación como registro sin validez. 2. Eliminación real: hacer inaccesible el registro o bien ocupar su espacio con otros registros.
BORRADO O DESTRUCCIÓN	Se elimina la información y la estructura del archivo. Finaliza la existencia del archivo.

La mayoría de las operaciones sobre ficheros (lectura, escritura o modificación, borrado) implica realizar una **búsqueda** de un registro determinado dentro del mismo, existen varios procedimientos para localizar registros en un archivo:

- **Búsqueda secuencial:** Se recorren todos los registros del archivo desde el principio al fin, hasta encontrar el solicitado. Aplicable en archivos con pocos registros o cuando los registros no tienen ningún orden.
- **Búsqueda binaria o dicotómica.** Solo aplicable si el fichero está ordenado.
Se lee el registro que está en el centro del archivo, si la clave buscada es menor que la del registro leído, se desecha la segunda mitad del fichero y se considera solo la primera, o a la inversa; luego se lee el registro central del tramo no desechado y se vuelve a repetir el proceso hasta encontrar el buscado o hasta obtener un tramo vacío.
- **Búsqueda por bloques.** Se considera el archivo lógicamente dividido en bloques, se determina primero en que bloque se encuentra el registro, para lo cual se lee el último registro de cada bloque, hasta encontrar el buscado u otro mayor a él, en cuyo caso se pasa a buscar el registro en el bloque anterior. Hallado el bloque se busca secuencialmente en él.

Otras operaciones, no tan usuales sobre ficheros son:

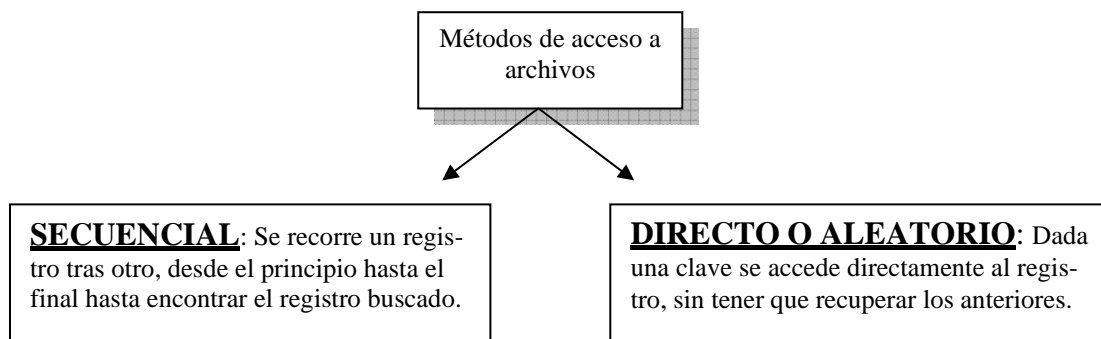
- **Ordenación** (sort) o clasificación de archivos, consiste en reubicar los registros de tal forma que queden ordenados con respecto a los valores de uno o varios campos denominados clave de ordenación, algunos métodos son:
 1. *Inserción*: Se construye una lista ordenada con los valores de las llaves de los registros, luego se va leyendo secuencialmente y se escriben los registros correspondientes en el archivo de salida. Si se inserta una nueva llave, se insertará en el lugar que le corresponda, quedando la lista ordenada.
 2. *Mezcla (merge)*: Se procede a una mezcla reiterada de secuencias del fichero ya ordenadas. Sea N el número de valores llave distintos del fichero. Partimos de N segmentos de un registro cada uno, desordenados; estos segmentos se mezclan dos a dos, produciendo $N/2$ segmentos ordenados de dos registros. Este proceso se repite hasta ordenar el archivo completo.
Si un archivo está ordenado y almacenado en un soporte direccionable, serán mucho más rápidas las consultas que se realicen por medio el campo que rige la ordenación.
- **Concatenación**. Dado 2 registros de igual estructura se genera otro en el que figuran todos los registros del primero y a continuación todos los del segundo.
- **Intersección**. De dos registros de igual estructura se obtiene otro donde figuren los registros comunes a ambos.
- **Fusión**. De dos archivos de igual estructura ordenados por una misma clave, se obtiene como resultado otro archivo que contiene los registros de ambos y mantiene la ordenación.
- **Actualización**. Consiste en modificar un archivo (maestro) por medio de otro archivo (de movimientos) que contiene altas, bajas y modificaciones que hay que realizar sobre el archivo maestro para ponerlo al día.

2.3 TIPOS DE ARCHIVOS

Clasificación según la longitud de los registros	LONGITUD FIJA	La suma de los caracteres de todos los campos es constante. Todos los registros del archivo tienen la misma longitud.
	LONGITUD VARIABLE	Cada registro del archivo puede tener una longitud distinta y esta oscila entre un mínimo y un máximo. Se reserva al comienzo de cada registro una palabra para anotar su longitud.
	DELIMITADOS	La longitud del registro es variable y no es posible conocer en cuanto difieren unos de otros. El sistema incluye un carácter especial para indicar el fin del registro.
	INDEFINIDOS	La longitud es totalmente variable. El programa del usuario localiza el principio y fin de cada registro.
Clasificación según el uso que se hace de los archivos	PERMANENTES	<p>Contienen información necesaria para el funcionamiento de una aplicación. Su vida es larga.</p> <ul style="list-style-type: none"> • Archivos maestros o de situación. Refleja el estado actual de los datos, se actualiza constantemente para reflejar cada nueva situación. Ej. Estado de cuentas de un banco. • Archivos constantes. Su información permanece prácticamente inamovible, en general se utilizan de consulta. Ej. Archivo de códigos postales. • Archivos históricos. Contienen datos que fueron actuales en tiempos anteriores. Se obtienen de los maestros cuando se dejan fuera de uso para futuros estudios estadísticos o consultas. Ej. Archivo de las cuentas canceladas.
	TEMPORALES	<p>Contienen información necesaria para un proceso específico. Tienen una vida efímera y una vez realizada su función se cancelan. Se pueden clasificar en:</p> <ul style="list-style-type: none"> • <i>Intermedios</i>. Se utilizan para pasar información de un proceso a otro. • <i>De maniobras</i>. Se utilizan para no perder información generada por un proceso que por falta de espacio en memoria principal no se puede conservar. • <i>De resultados</i>. Se genera a partir de los resultados finales de un proceso que van a ser transferidos a un dispositivo de salida. Ej. Un fichero de impresión.

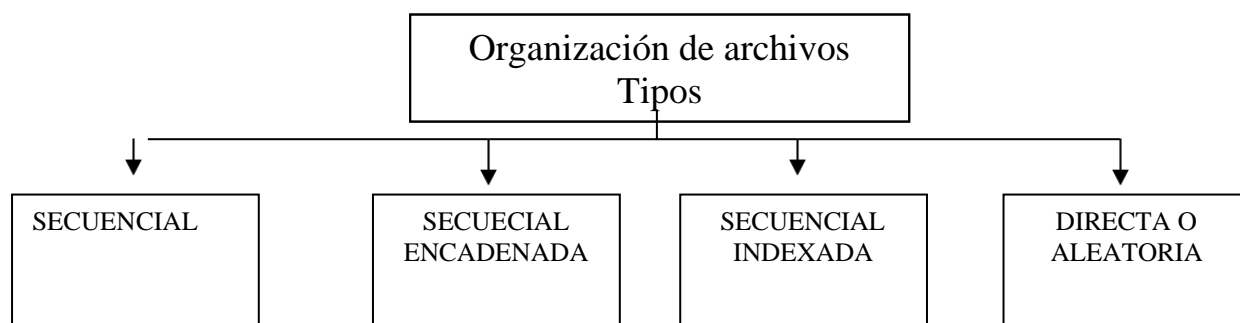
2.4 ORGANIZACIÓN DE ARCHIVOS

Usualmente el computador necesita acceder a los archivos ya sea para recuperar la información o para grabarla. El **método de acceso** determina como pueden recuperarse los registros.



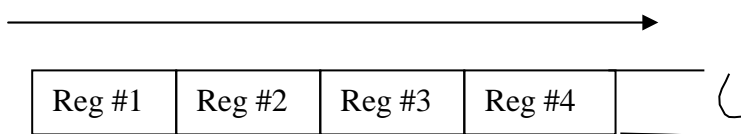
El acceso a un archivo está íntimamente ligado a la forma como están dispuestos los registros en el soporte material, por ejemplo un archivo con organización secuencial no podrá ser accedido de forma directa. Cuando se crea un archivo es necesario especificar qué organización tendrá, ya que esto va a determinar que tipo de acceso podemos utilizar.

Los tipos de organización de archivos son básicamente:



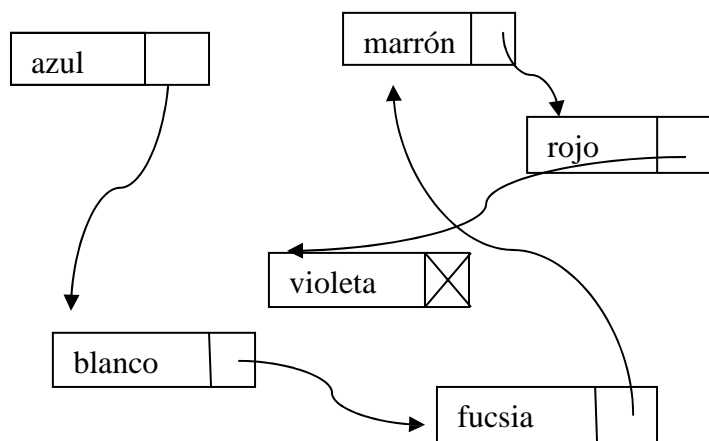
2.4.1 ORGANIZACIÓN SECUENCIAL

Los registros se almacenan uno después de otro, sin dejar espacio entre ellos y ordenados según una clave de clasificación.



2.4.2 ORGANIZACIÓN SECUENCIAL ENCADENADA.

Cada registro contiene, además de los campos de información, un puntero con la dirección del registro siguiente según el orden lógico del archivo, el puntero del último registro contiene una dirección nula.



Las direcciones físicas donde se encuentran los registros son arbitrarias, pero los punteros permiten recorrer el archivo en su secuencia lógica.

2.4.3 ORGANIZACIÓN SECUENCIAL INDEXADA

Un archivo con esta organización consta de tres zonas o áreas:

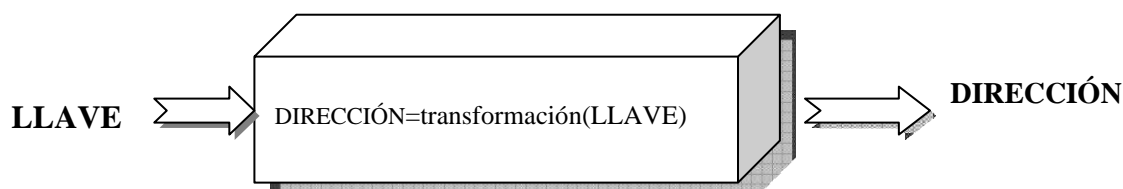
Zona o área	Contenido
de registros	Posee una organización secuencial pura. Contiene todos los registros de datos, ordenados según un campo clave. Está dividida en una serie de segmentos o tramos lógicos, formados por registros consecutivos.
de índices	Es una estructura, con organización secuencial pura, creada y gestionada por el sistema, con una cantidad de registros muy inferior al total de registros del archivo. Cada registro está formado por solo dos campos: un campo <i>clave</i> o <i>llave</i> (contiene la clave más alta de cada segmento) y otro campo <i>dirección</i> (contiene la dirección de comienzo de dicho segmento). Por cada segmento o tramo en la zona de registros, existe un registro en la zona de índices. El sistema accede primero a la zona de índices y a través de esta accede directamente a un segmento de la zona de registros.
de desbordamiento (overflow)	Contiene los nuevos registros que no pueden ser agregados al área de registros, ya que para ello hay que reorganizar el archivo por tratarse de una estructura secuencial pura. En esta zona los registros están desordenados, ya que cada registro nuevo se añade al final de la misma.

Ejemplo de un fichero secuencial indexado:

Zona de índices		Zona de registros			
Clave	Dirección		<i>Legajo</i>	<i>Apellido y nombre</i>	<i>Domicilio</i>
10482	1	1	8640	Lopez Juan	Carlos Calvo 1832
28453	4	2	9522	Areco Mirta	Azcuénaga 345
32546	6	3	10482	Lordi Marcelo	Arieta 3467
		4	12478	Blanco Pedro	Illia 2412
		5	28453	Zamudio Pablo	Pichincha 765
		6	30125	Avalo Liliana	Varela 321
		7	31877	Zavala Nora	Ombú 467
		8	32012	Miguez Pablo	Cabildo 1234
		9	32489	Caruso José	Santa Fé 456
		10	32546	Marin Pedro	Lavalle 54

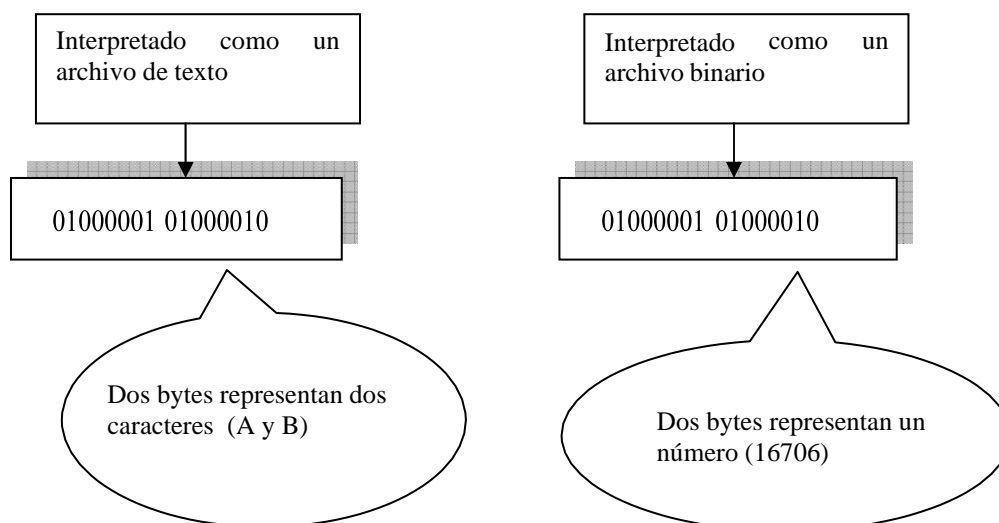
2.4.4 ORGANIZACIÓN DIRECTA O ALEATORIA.

En la organización indexada, el índice relaciona la llave con la dirección, a expensas de usar tablas adicionales que ocupan lugar y requieren mantenimiento. En la organización directa o aleatoria, la posición de un registro en el almacenamiento masivo, se calcula aplicando una fórmula o algoritmo matemático al valor del campo clave.



2.5 Archivos de texto- archivos binarios

Un archivo almacenado en un dispositivo de almacenamiento es una secuencia de bits que puede ser interpretado por un programa de aplicación como un archivo de texto o un archivo binario.



- **Archivos de textos:** es un archivo de caracteres, los números enteros, de punto flotante o cualquier otra estructura de datos, para almacenarlos deberán ser convertidos a sus formatos equivalentes de caracteres.

Si una cadena de caracteres se envía a la impresora, esta toma 8 bits, los interpreta como un byte y lo decodifica en el sistema de codificación de la impresora (ASCII o EBCDIC).

Si es un carácter imprimible, la impresora lo imprimirá, de no serlo, se realizará otra acción, por ejemplo la impresión de un espacio, el avance de una línea, etc.

- **Archivos Binarios:** Es una colección de datos (enteros, punto flotante, un carácter, etc.) almacenados en el formato interno de la computadora.

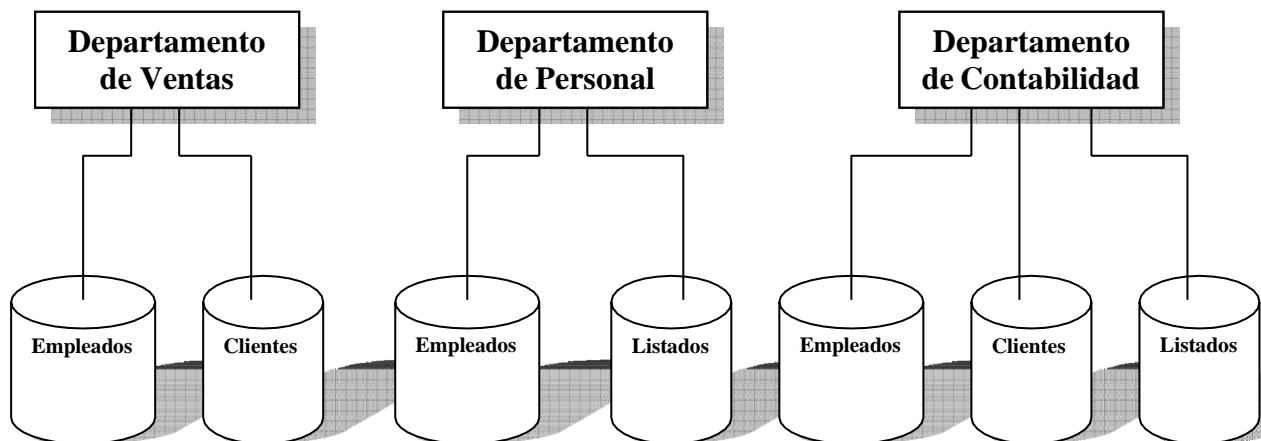
Estos datos son significativos solo si son interpretados correctamente por un programa.

Si son numéricos 2 o más bytes se consideran un elemento de datos, por ejemplo una PC que utiliza 2 bytes para almacenar un entero. Si los datos son textuales un carácter se representa con 1 byte.

3. BASES DE DATOS

Si se considera que una de las aplicaciones más importantes de una computadora es el almacenamiento, recuperación y mantenimiento de grandes cantidades de información, los archivos como aplicación convencional no es del todo eficiente. Los archivos se diseñan de acuerdo con los programas, donde el programador decide si debe haber archivos, cuántos deben ser, qué organización tendrán, qué información contendrán, qué programas actuarán sobre ellos y cómo lo harán. Esta modalidad tiene la ventaja de ser lo suficientemente eficiente, ya que el archivo está pensado para el programa que lo va a usar. Sin embargo, con un sistema tradicional de archivos, cada sector de una misma empresa será el responsable de crear y mantener los datos necesarios, aún cuando éstos estén duplicados. De esta manera, se pueden encontrar los siguientes problemas:

Problemas	Características
Actualización de la Información	La actualización puede resultar costosa cuando se tiene información total o parcialmente duplicada en archivos diferentes. Esto conduce a <i>inconsistencia de datos</i> .
Redundancia	Consiste en tener datos que no aportan información, porque pueden ser deducidos de otros.
Rigidez en la búsqueda	No siempre el modo de acceso que tiene el archivo es el más eficiente, no pudiendo cambiarse.
Dependencia con los programas	Cualquier cambio en la estructura del archivo implica una modificación de los programas que lo tratan.
Confidencialidad y seguridad	La confidencialidad consiste en evitar el acceso a determinados usuarios. La seguridad consiste en que los datos no puedan ser modificados por usuarios no autorizados. Ambas cosas deben hacerse <i>por programa</i> .



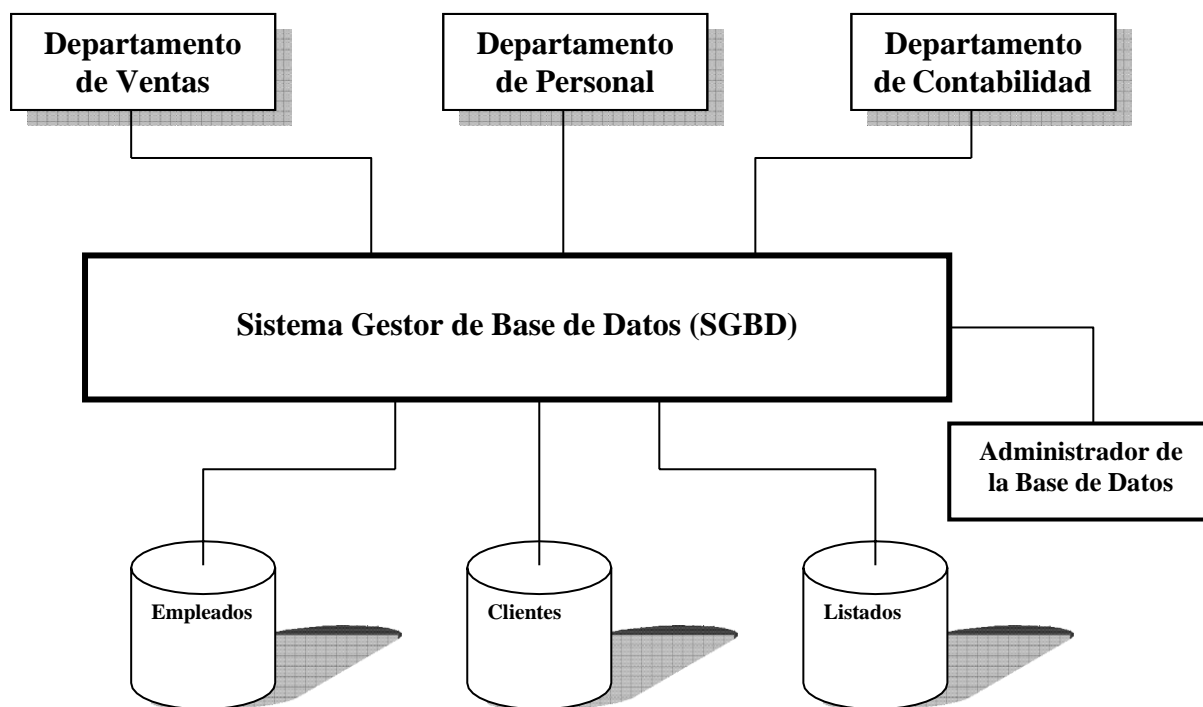
Organización con un sistema de archivos tradicionales

Las *bases de datos* surgen como una alternativa válida a los sistemas de archivos, que facilita la manipulación de grandes cantidad de información, para hacerla segura y accesible a una variedad de usuarios para una variedad de propósitos.

Una base de datos es un sistema formado por un conjunto de datos organizados de tal manera que se controla el almacenamiento de datos redundantes, los datos resultan independientes de los programas que los usan y se pueden acceder a ellos de diversas formas.

Los requisitos de una buena base de datos son:

- Varios usuarios accediendo a la base de datos y cada uno accederá a determinada información.
- Se controlará el acceso de los usuarios asegurando confiabilidad y seguridad.
- Los datos se almacenan sin redundancia, excepto en casos especiales (redundancia aceptable).
- Se accede de distintas maneras, flexibilizando las búsquedas.
- Deben existir mecanismos concretos de recuperación de información en caso de fallos.
- Se puede cambiar el soporte físico sin repercusión en los programas que usan la base.
- Se puede modificar los contenidos, las relaciones o agregar nuevos datos sin afectar los programas que usan la base de datos.
- Existe una interfaz de la base de datos que permite usarla de forma cómoda y sencilla.



Organización con un Sistema de Gestión de Base de Datos

3.1 CONCEPTOS BÁSICOS DE LAS BASES DE DATOS

Se definen conceptos básicos de Bases de Datos:

► *Entidad*

Una *entidad* es un objeto real o abstracto con características particulares, capaces de hacerse distinguir de otros objetos, y del cual se almacena información en la base de datos. Una *entidad* toma como significado a conceptos u objetos que tienen una importancia en el sistema u organización. Está formada por un conjunto de ítems de datos o *atributos*.

Si tomamos el ejemplo de una base de datos que guarde la información del Departamento de Ingeniería de la UNLaM acerca de los alumnos, los docentes, las materias y las aulas donde se dictan, se tomaría como entidades ALUMNO, PROFESOR, MATERIA y AULA.

► *Atributos*

Un *atributo* es una unidad básica e indivisible de información acerca de una entidad que sirve para identificarla o describirla.

Si continuamos con el ejemplo anterior, atributos de la entidad ALUMNO son: DNI, Apellido y nombre, fecha de nacimiento, domicilio, carrera que cursa (Informática, Industrial o Electrónica), teléfono, código de materia que cursa, etc.

► Registros

En una base de datos, la información de cada entidad se almacena en *registros*, y cada atributo, en campos de dicho *registro*. Existen distintos tipos de *registros* dentro de la misma base, ya que cada entidad necesitará una estructura distinta.

En una base de datos habrá tantos tipos de *registros* como entidades haya, mientras que en un archivo sólo hay un tipo único de *registro*.

Siguiendo con el ejemplo anterior, la estructura del registro de la entidad ALUMNO va a ser muy distinta del registro de la entidad MATERIA, ya que ambas entidades tienen distintos atributos.

► Superclave, clave candidata, clave principal o primaria y clave foránea o ajena.

Se llama **superclave** de una entidad a un atributo o conjunto de atributos que permite identificar de forma única a un registro de una entidad.

Si de una *superclave* no se puede obtener ningún subconjunto que a su vez sea *superclave*, se dice que dicha *superclave* es **clave candidata**.

De todas las *claves candidatas* existentes, el diseñador de la base de datos, escogerá una que individualizará de forma inequívoca a cada *registro* de la *entidad*. Esta clave se denomina **clave principal o primaria**.

Cuando existe una referencia entre dos entidades, esto es cuando un campo o conjunto de campos de una de las entidades es la clave de otra, se las llama clave foránea o ajena. Esta clave foránea permite localizar una entidad a partir de otra.

En el ejemplo anterior, en la entidad ALUMNO algunas *superclaves* pueden ser:

- ✓ DNI
- ✓ Apellido y nombre
- ✓ DNI + Apellido y nombre
- ✓ DNI + Apellido y nombre + fecha de nacimiento
- ✓ Apellido y nombre + domicilio + teléfono
- ✓ ...

Las *claves candidatas* serían todas aquellas no permitan encontrar un subconjunto que pueda ser a su vez *superclave*:

- ✓ DNI
- ✓ Apellido y nombre

Entre estas *claves candidatas*, se debe escoger DNI, ya que podría haber dos alumnos con el mismo apellido y nombre. Entonces DNI es la *clave primaria o principal*.

La entidad ALUMNO tiene un atributo **código de materia que cursa**. En la entidad MATERIA, el código de la materia es la clave primaria o principal. Por lo tanto, en la entidad ALUMNO el atributo **código de materia que cursa** es una *clave foránea*, ya que relaciona a esta entidad con la entidad MATERIA.

► Relaciones

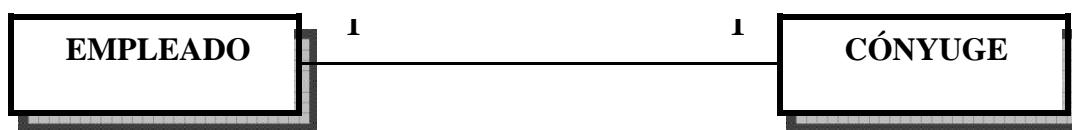
Las entidades por sí solas no describen la realidad de un sistema de información. Además de identificar objetos, hay que establecer las asociaciones existentes entre ellos. Esto es una *relación*: la existencia de algo común entre entidades.

Siguiendo con el ejemplo, existe la *relación* entre las entidades ALUMNO – PROFESOR (dictado de clases), PROFESOR – MATERIA (dictado de la misma), MATERIA – AULA (correspondencia en un determinado día y horario), etc.

► Grado

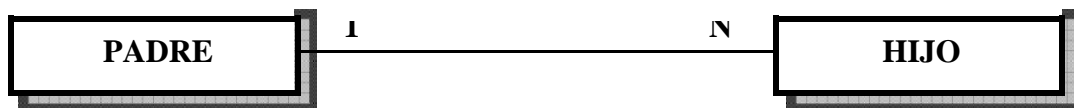
El grado de una relación representa la participación en la relación de cada una de las entidades afectadas. El grado siempre se evalúa de a dos entidades. Existen tres tipos posibles:

- ✓ **1:1 (una a una):** A cada registro de una entidad le corresponde no más de un registro de la otra y viceversa. Es **biunívoca**.



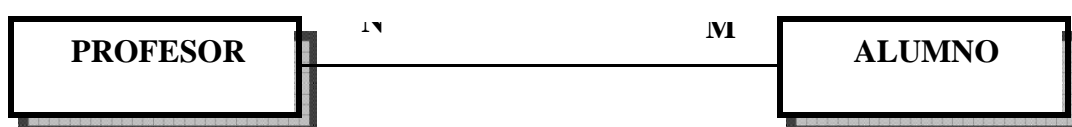
La relación EMPLEADO – CÓNYUGE (un empleado puede estar casado con una única persona)

- ✓ **1:N (una a muchas):** A cada registro de la primera entidad le pueden corresponder varios registros de la segunda, y a cada registro de la segunda le corresponde no más de uno de la primera.



La relación PADRE – HIJO (un padre puede tener muchos hijos, pero éstos sólo tendrán un padre)

- ✓ **N:N (muchas a muchas):** A cada registro de la primera entidad le pueden corresponder varios registros de la segunda y viceversa.



La relación PROFESOR – ALUMNO (un profesor da clase a muchos alumnos y un alumnos tiene varios profesores)

► **Cardinalidad**

La *cardinalidad* de una base de datos es el número de entidades que se relacionan entre si en la base de datos. En el ejemplo de la base de datos de la UNLaM, la cardinalidad es tres, ya que la entidad PROFESOR se relaciona con ALUMNO (el profesor dicta clase a alumnos), con MATERIA (el profesor dicta UNA MATERIA) y con AULA (el profesor dicta clase en un aula).

► **Esquema de una base de datos**

El *esquema de una base de datos* es la definición de la estructura lógica de ésta, esto es, la especificación de cada uno de los registros que la integran, indicando los campos que la componen y las relaciones que las ligan.

3.2 **LENGUAJE DE BASE DE DATOS**

Las bases de datos utilizan para su creación o manejo dos lenguajes específicos:

- **DDL** (Data Definition Language): Este es un lenguaje de *descripción de datos*, que como cualquier lenguaje de alto nivel, necesita un traductor para generar código objeto a partir del código fuente.
- **DML** (Data Manipulation Language): Es un lenguaje de *manipulación de datos*. El *DML* puede ser usado de dos formas. Por un lado, se incluyen sentencias de *DML* en programas escritos en lenguajes de alto nivel (COBOL, Basic, C, etc.), por lo que al primero se lo llama *lenguaje huésped*, y al segundo, *lenguaje anfitrión*. En este caso, como el compilador del lenguaje anfitrión no reconoce las sentencias en DML, se incluye un precompilador en el Sistema de Gestión de Bases de Datos (SGBD). La segunda forma del DML es mediante el uso de programas que contengan exclusivamente sentencias propias de este lenguaje.

3.3 **SISTEMA DE GESTIÓN DE BASE DE DATOS**

Se denomina *Sistema de Gestión de Base de Datos* (SGBD o DBMS, *Data Base Management System*), al conjunto de software destinado a la creación, gestión, control y manipulación de la información sobre una base de datos. Los SGBD tienen como propósito registrar y mantener información.

Un SGBD debe permitir:

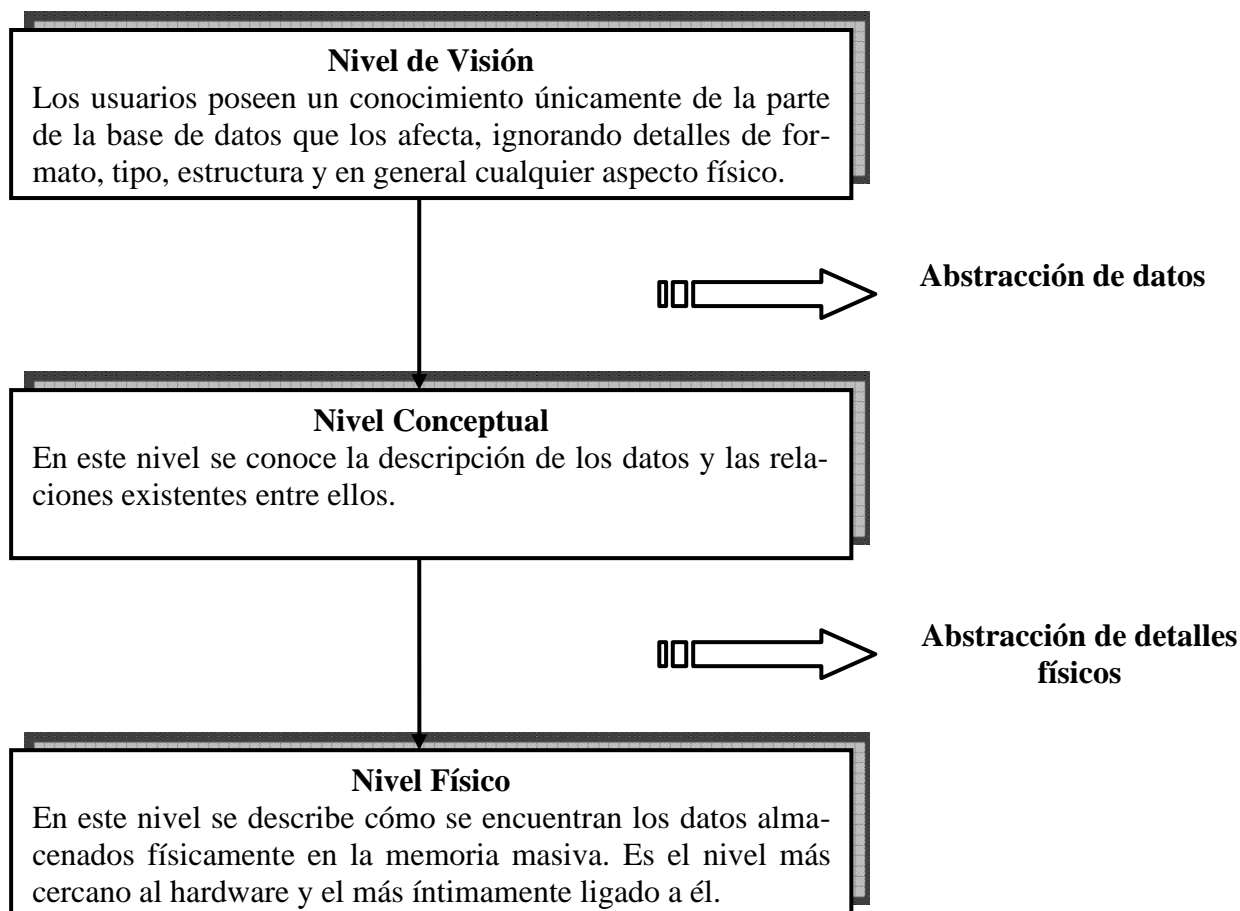
- **Definir el esquema de la base de datos**, describiéndolo mediante un conjunto de instrucciones con el lenguaje DDL.
- **Acceder a la información desde un lenguaje de alto nivel**, para lo cual se utiliza el lenguaje DML.
- **Acceder a la información en modo conversacional**, incorporando una interfaz de usuario a través de la cual introducir sentencias DDL o DML directamente desde una terminal para obtener información interactiva.
- **Gestionar los archivos**, función realizada por el módulo Gestión de Archivos, que es el que

se encarga de la comunicación con el Sistema Operativo.

- ***Realizar funciones varias***, tales como control de usuarios, recuperación de la base en caso de fallas, organización física de la base de datos, control de seguridad, privacidad de la información y gestión de accesos concurrentes.

3.4 ABSTRACCIÓN DE LA INFORMACIÓN

El SGBD debe proporcionar información a usuarios y desarrolladores a distintos niveles, representando cada uno de ellos una abstracción de datos.



3.5 TIPOS DE BASE DE DATOS

► *Bases de datos jerárquicas*

Fueron las primeras en aparecer. En este tipo de base de datos sólo se pueden crear estructuras jerárquicas (estructuras en árbol). Están formadas por una colección de registros unidos a través de relaciones que pueden ser de uno a uno o de uno a muchos, aunque no se pueden definir relaciones de muchos a muchos. Las relaciones se implementan físicamente utilizando *punteros*.

Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos, permitiendo crear estructuras estables y de gran rendimiento.

► *Bases de datos en red*

Estas bases son muy parecidas a las jerárquicas. Se permite cualquier tipo de relaciones, pero se distingue entre las de red simple (no permiten relaciones de muchos a muchos u es la más común) y las de red complejas. Cualquier sistema puede ser representado en este tipo de base de datos.

► *Bases de datos Relacionales*

Una base de datos relacional está formada por tablas. Una tabla es una estructura bidimensional formada por una sucesión de registros. Si se imponen ciertas restricciones, se las puede tratar como relaciones matemáticas (de allí el nombre *relacionales*).

Una tabla es como el usuario ve sus datos. Se divide horizontalmente en filas y verticalmente en columnas. Una fila representa un registro (comúnmente llamado **tupla**), los cuales contienen toda la información necesaria sobre la entidad de la tabla. Cada columna contiene información referente a un único campo o *atributo*.

Se define como clave a un atributo o conjunto de atributos que identifican a una tupla de forma unívoca y sin redundancia.

Las tablas deben cumplir los siguientes requisitos:

- ✓ Todas las tuplas de una tabla son del mismo tipo. Para almacenar registros distintos se usan tablas distintas.
- ✓ Cada columna se identifica por su nombre.
- ✓ En ninguna tabla aparecen nombres de columnas repetidos.
- ✓ En ninguna tabla existen tuplas duplicadas (filas idénticas).
- ✓ El orden de los registros es indiferente, ya que en cualquier momento mediante una sentencia en DML se pueden recuperar en un orden en particular.
- ✓ En cada tabla hay una **llave** (clave principal o primaria) formada por uno o más campos.
- ✓ El contenido de las tablas es independiente del almacenamiento físico de los datos.

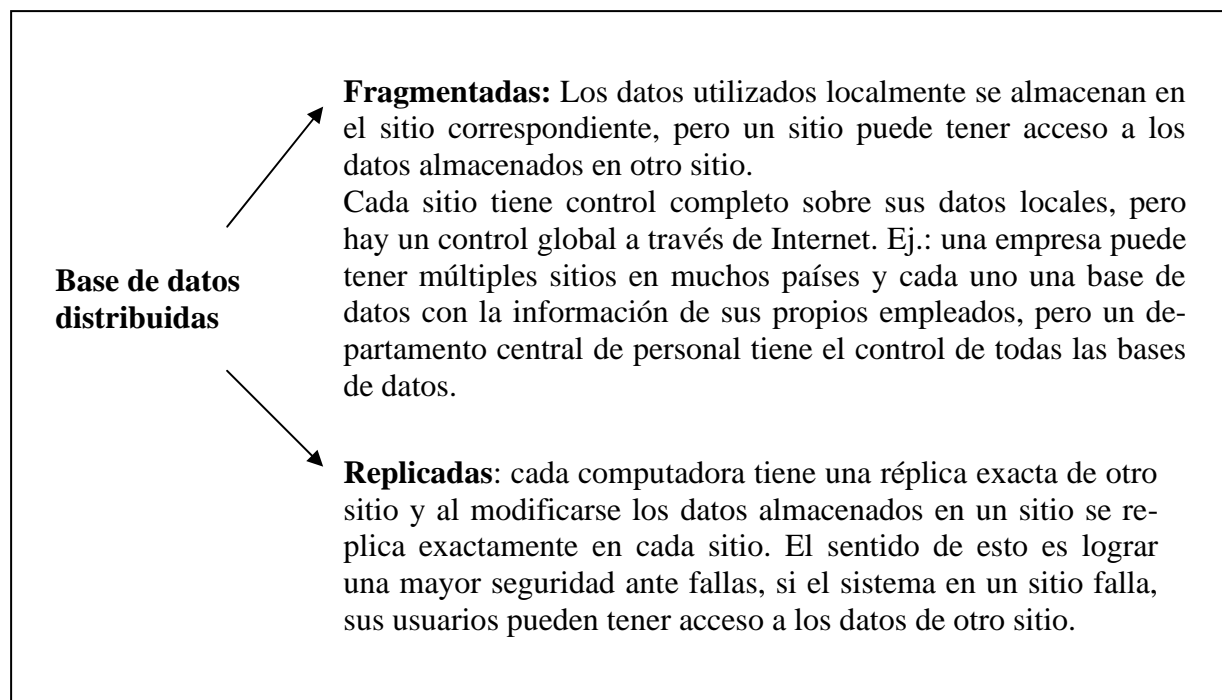
El modelo relacional es **uno de los más utilizados** en el diseño y gestión de base de datos, fundamentalmente por dos causas:

- Se basa en un número reducido de conceptos que lo hacen fácil de entender, diseñar, cambiar, administrar y acceder.
- Posee un lenguaje de definición y manipulación muy potente y flexible: **SQL (Structured Query Language)**.

Ejemplos comerciales de este tipo de base de datos es Oracle, MySQL, SQLServer, Access, etc.

► *Bases de Datos Distribuidas*

Está basado en el modelo relacional. Los datos se almacenan en varias computadoras (llamadas también *sitios*) conectadas a través de Internet (o una red privada de área amplia). Éstos pueden estar fragmentados y cada fragmento almacenado en un sitio o duplicadas en cada sitio:



► Bases de datos Orientadas a objetos

Las bases de datos relacionales tienen como unidad de dato más pequeña la intersección de una tupla y un atributo, sobre esto basan la vista específica de sus datos pero actualmente diversas aplicaciones necesitan realizar búsquedas viendo los datos como una estructura, por ejemplo un registro compuesto de campos.

Las bases de datos orientadas a objetos permiten que las aplicaciones accedan a los datos estructurados tratando de mantener las ventajas del modelo relacional.

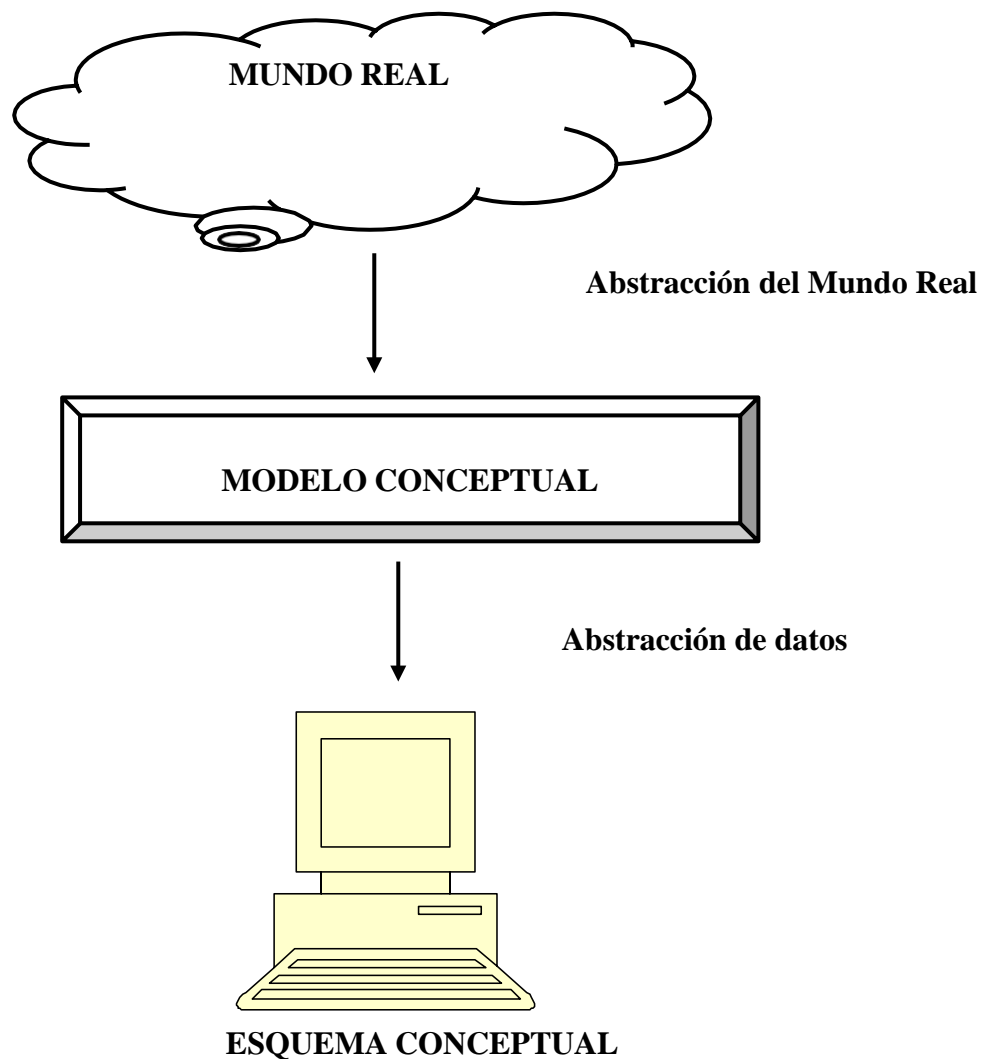
En una base de datos orientada a objetos se definen los objetos y sus relaciones, donde cada objeto puede tener atributos que pueden expresarse como campos.

Por ejemplo, en una organización, se puede definir tipos de objetos tales como empleados, secciones, clientes. La clase empleados puede definir los atributos de un objeto empleado (apellido y nombre, D.N.I., salario, etc.) y como se puede tener acceso a los mismos. El objeto departamento puede definir los atributos del departamento y como puede accederse a ellos. Asimismo, la base de datos puede crear una relación entre un objeto empleado y un objeto departamento (un empleado trabaja en un departamento).

3.6 MODELO DE DATOS

Cuando se desea confeccionar una base de datos debe realizarse un proceso que partiendo del mundo exterior, lo conceptualice de manera tal que lo transforme en un conjunto de ideas y de definiciones que conformen una imagen fiel del mundo real. Esto no es más que una abstracción del mundo que nos rodea. A esta imagen del mundo exterior se lo llama **modelo conceptual**.

Una vez definido el modelo conceptual, se debe transformar en una descripción de datos, atributos y relaciones que se denominan **esquema conceptual**. Finalmente este esquema debe ser traducido a estructuras almacenables en soportes físicos.



En algunos casos, si la complejidad del problema no es muy grande y la experiencia de quien lo realiza es alta, puede pasarse del mundo real al esquema conceptual sin pasar por el modelo conceptual.

Un **modelo de datos** puede definirse como un grupo de herramientas conceptuales para describir los datos, sus relaciones, su semántica y sus limitaciones, facilitando la representación del mundo real en nuestro mundo informático.

Un *modelo de datos* contiene dos tipos de propiedades:

- ✓ **Propiedades estáticas:** No varían con el tiempo, quedando especificadas por las estructuras. Se define mediante el lenguaje DDL.
- ✓ **Propiedades dinámicas:** Varían con el tiempo, ya que son las operaciones. Se definen mediante el lenguaje DML. Las operaciones pueden ser de selección (localizar un dato) o de acción (recuperación y actualización).

3.7 USUARIOS DE LA BASE DE DATOS

Tipo de Usuario	Características	Responsabilidades
Programador de aplicaciones	<ul style="list-style-type: none"> ▪ Escribe programas que utilicen las bases de datos. ▪ Diseñan aplicaciones para apoyar al usuario final. 	<ul style="list-style-type: none"> ▪ Realizar funciones de creación, recuperación, borrado o modificación de datos.
Usuario final	<ul style="list-style-type: none"> ▪ Accede a la base desde una terminal. 	<ul style="list-style-type: none"> ▪ Utiliza acceso a través de DML directamente. ▪ Utiliza programas utilitarios creados por el programador.
Administrador de la base de datos	<ul style="list-style-type: none"> ▪ Encargado del control general del sistema de base de datos. 	<ul style="list-style-type: none"> ▪ Decidir el contenido de la información. ▪ Decidir las estructuras de almacenamiento y la estrategia de acceso. ▪ Vincularse con el resto de los usuarios de la base de datos. ▪ Definir controles de autorización y validación. ▪ Definir estrategia de respaldo y recuperación de datos por fallas del sistema. ▪ Controlar el rendimiento y utilización de la base. ▪ Responder a los cambios en los requerimientos.

3.8 INDEPENDENCIA DE LOS DATOS

Los sistemas de archivos son dependientes de los datos. Esto significa que la forma de almacenamiento y acceso a la información dependen de los requerimientos de la aplicación. Se dice que una aplicación es dependiente de los datos cuando *es imposible cambiar la estructura de almacenamiento y las estrategias de acceso* sin afectar la aplicación.

Sin embargo, en el sistema de base de datos, es *necesario* que los datos sean independientes de las aplicaciones, fundamentalmente por dos razones:

- ✓ Aplicaciones diferentes requerirán vistas diferentes por parte de los usuarios de los mismos datos.
- ✓ El administrador de la base de datos debe tener libertad para cambiar la estructura de almacenamiento y la estrategia de acceso, en respuesta al cambio de necesidades sin alterar todas las aplicaciones existentes.

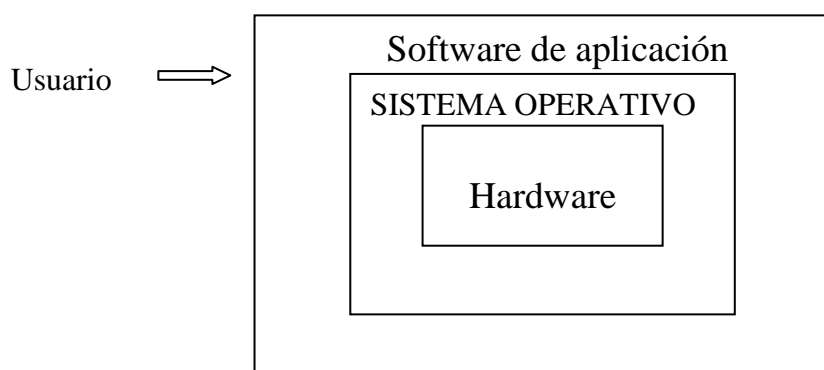
La independencia de los datos es un objetivo esencial de los sistemas de bases de datos.

4. Sistemas Operativos

4.1 Introducción

El software es el elemento que permite que un computador pueda almacenar, procesar y recuperar información, para lo cual se trata de aislar al usuario de la complejidad del hardware, añadiendo una capa de software sobre el hardware puro que se encarga de gestionar todos los elementos del sistema y alivianar de ese modo el trabajo con el computador.

Esta capa de software tan ligado al hardware recibe el nombre de Sistema Operativo.



Un **Sistema Operativo** es un conjunto de programas que controlan el funcionamiento del hardware ocultando sus detalles al usuario, permitiéndole así trabajar con el computador de una manera más fácil y segura.

Por otra parte, el computador posee un conjunto de elementos, necesarios para cumplir con su trabajo, llamados recursos, que deben ser racionalmente distribuidos y utilizados para un mejor rendimiento. El Sistema Operativo es el administrador de los recursos ofrecidos por el hardware.

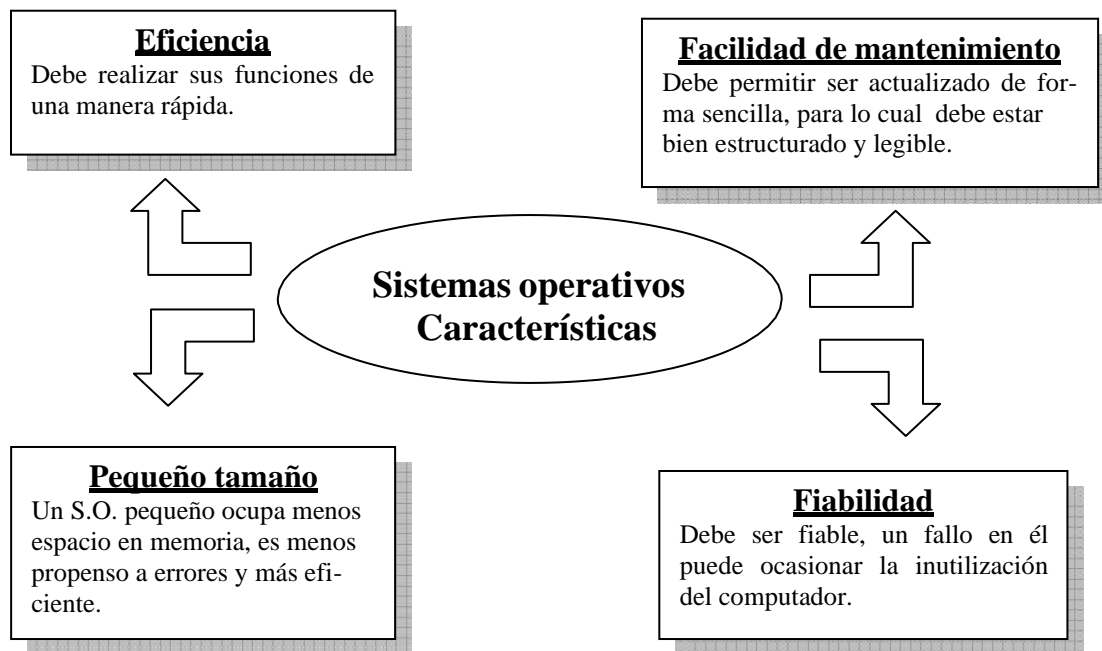
Los principales recursos de un computador son: el procesador, la memoria principal, los dispositivos periféricos y la información (los datos).

Sin un sistema operativo, un computador nunca podría empezar a funcionar, por lo tanto cuando se enciende un computador lo primero que ha de ocurrir es la carga del S. O. en la memoria principal.

Primero se ejecuta un programa de autodiagnóstico de encendido, que identifica todos los dispositivos de hardware conectados, luego se ejecuta el *cargador* inicial, que carga un programa de autoarranque más eficiente, el mismo busca el S.O. y carga parte del mismo (parte residente) en la memoria principal.

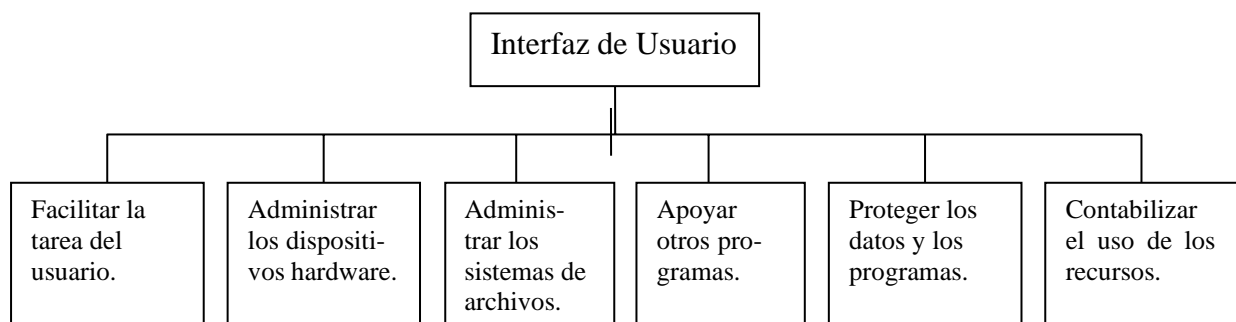
Una vez que el ordenador ha puesto en marcha el Sistema Operativo mantiene parte de él en su memoria en todo momento.

4.2 Características deseables de un sistema operativo



4.3 Funciones de los sistemas operativos

Las funciones que realiza un sistema operativo dependen del tipo de S.O. (monotarea, multitarea, etc.), pero existen algunas que se pueden considerar en todos los S.O:



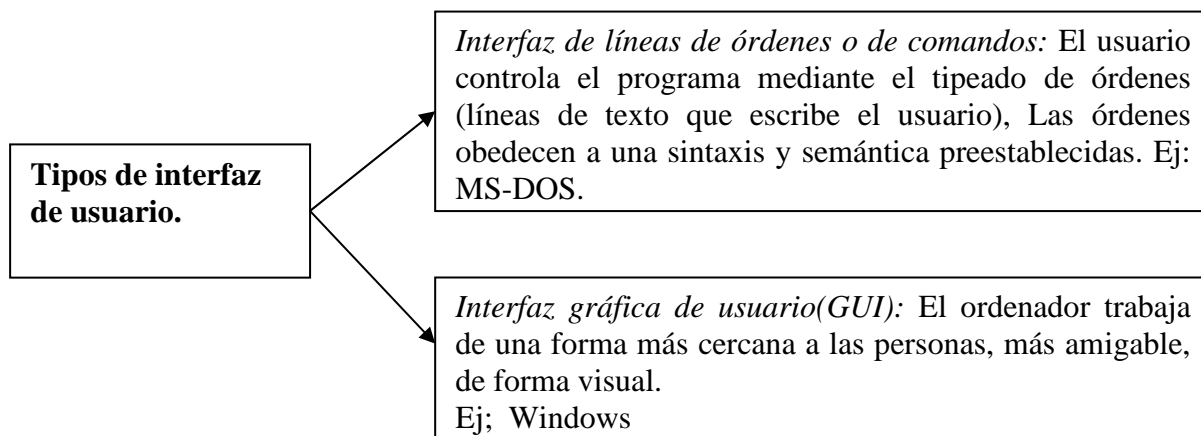
4.3.1 Facilitar la tarea del usuario

Podemos considerar que cada una de estas tareas es realizada por un módulo del S.O. y existe un módulo distinguido, **Kernel**, el núcleo del S.O., que sirve a todos los demás.

El Kernel es el módulo del S.O. de más bajo nivel, más ligado al hardware, siempre permanece en la memoria principal y entre otras cosas se encarga del manejo de interrupciones, la asignación de trabajos al procesador y proporciona una vía de comunicación entre los programas.

Para comunicarse con el ordenador, los programas incluyen módulos para definir la interfaz del usuario. Cada programa de aplicación dispone de sus propios módulos de interfaz con el usuario y el S.O también dispone de su propia interfaz de usuario.

Las funciones principales del S.O. son controladas por el núcleo, pero la interfaz de usuario es controlada y establecida por el *entorno (shell)* o *intérprete de órdenes*, programa independiente del sistema operativo que acepta solicitudes de los usuarios (procesos) y las interpreta para el resto del S.O. Muchas shells diferentes pueden usar el Kernel de un mismo S.O.



4.3.2 Administración el hardware

Los programas, durante su ejecución, necesitan utilizar determinados recursos de hardware, como ser la memoria, el monitor, unidades de disco, etc. El S.O. realiza la gestión y administración de los mismos. La administración del hardware por parte del S.O. abarca:

GESTIÓN DEL PROCESADOR

Programa: conjunto de instrucciones, escritas por un programador y almacenadas en memoria masiva.

Proceso: es un programa en ejecución, que se ha iniciado pero no se ha terminado. Puede estar en uno de los siguientes estados durante su existencia:

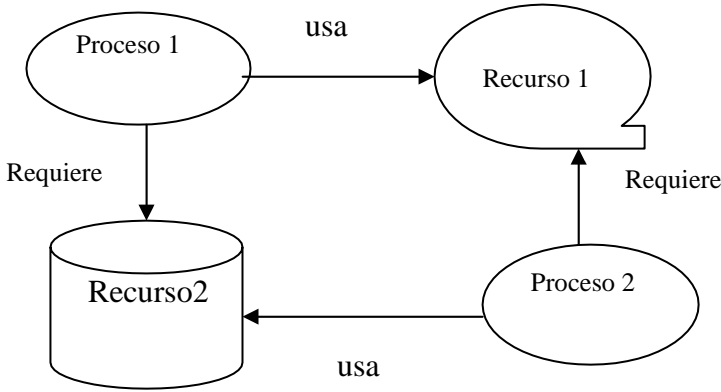
- *Estado ejecutable:* el proceso reside en memoria principal y está preparado para continuar su ejecución.
- *Estado de ejecución:* está siendo atendido por el procesador.
- *Estado bloqueado:* tiene operaciones de E/S pendientes o en espera de algún recurso que no está disponible.

Un programa por sí es un ente pasivo, mientras que un proceso es un ente activo.

Las tareas asociadas a la gestión del procesador por parte del SO. son:

1. Preparación de programas. Transfiere los programas ejecutables del usuario desde la memoria masiva a la memoria principal a partir de una determinada dirección de memoria por medio de un programa llamado *cargador*.

2. Asignación de recursos. La demanda de recursos durante la actividad del computador casi siempre supera a los recursos realmente disponibles. Ante este problema, los sistemas operativos disponen de una *política de*

	<p><i>asignación de recursos.</i> Un ejemplo de política simple podría ser: “el primero que solicite un recurso será el primero en ser atendido”. Esta política puede llevar a situaciones de interbloqueo (deadlock) o punto muerto, que ocurren cuando 2 procesos se bloquean mutuamente al solicitar insistentemente recursos que están asignados al otro.</p>  <pre> graph TD P1([Proceso 1]) -- usa --> R1((Recurso 1)) P2([Proceso 2]) -- usa --> R2[(Recurso2)] P1 -- Requiere --> R2 P2 -- Requiere --> R1 </pre> <p>Se han desarrollado diversas políticas que evitan o resuelven este problema. Por ejemplo, no permitir que un proceso inicie su ejecución hasta que los recursos requeridos estén disponibles. Otra solución sería limitar el tiempo que un proceso puede ocupar un recurso.</p> <p>3. Planificación del procesador. A través de una política de asignación de tiempos, se controla el uso del procesador por parte de los distintos procesos. Por ejemplo, mediante el uso de una cierta jerarquía de prioridades.</p> <p>4. Relanzamiento de procesos. Ante interrupciones fortuitas en la ejecución de un programa o por que se están ejecutando varios procesos a la vez y se tienen que turnar, se deben tomar las medidas necesarias (por ej. hacer una copia del estado de todos los registros) para poder reproducir el estado que tenía el proceso cuando se vuelva a reactivar.</p>
GESTIÓN DE LA MEMORIA	<p>Este módulo se encarga de asignar ciertas posiciones de la memoria principal a los diferentes programas o partes de los programas que la necesiten mientras el resto de los datos y programas se mantienen en almacenamiento secundario. Se establecen zonas de seguridad para evitar colisiones entre las distintas áreas de memorias asignadas a cada programa.</p> <p>Existen varias técnicas de asignación de memoria principal, según sea un S.O. monoprogramado (un solo programa en ejecución) o multiprogramado (ejecución de varios programas simultáneamente).</p> <p>La forma más común de gestión de memoria en la actualidad se basa en crear una memoria virtual, lo cual permite al usuario hacer programas de una capacidad muy superior a la que físicamente tiene el computador, teniendo como límite el espacio que se reserve en disco para ella y no el de la memoria principal. En el disco se mantiene un archivo con la imagen del programa completo, dividido en páginas o segmentos y en memoria sólo la página o segmento que en ese momento debe estar en ejecución, intercambiando las páginas o segmentos entre disco y memoria principal.</p>

GESTIÓN DE ENTRADAS/SALIDAS	Los distintos dispositivos periféricos poseen características muy diferentes. El Sistema Operativo, logra que el programador de aplicaciones pueda realizar las operaciones de E/S de una forma independiente de las particularidades de los dispositivos, por ejemplo es el S.O. el que determina los bloques y sectores de los discos que se utilizan.
------------------------------------	--

4.3.3 Administración del sistema de ficheros

Los sistemas operativos agrupan la información (datos y programas) dentro de compartimientos lógicos, denominados ficheros o archivos, para almacenarla en los soportes de almacenamiento masivo. Aunque la noción de archivo constituye un concepto lógico, los archivos hacen referencia a un grupo de información físicamente almacenada.

El sistema operativo posibilita que el usuario tenga una visión lógica de los archivos, aislándolo de los problemas físicos del almacenamiento, no tiene que utilizar direcciones físicas, puede actuar sobre un archivo simplemente indicando su nombre y utilizando determinadas órdenes del lenguaje de control del sistema operativo.

Hay archivos cuya información puede ser compartida por múltiples usuarios y otros tienen carácter privado, por lo tanto, cada archivo generalmente tiene un conjunto de **privilegios de acceso**, que indican en que medida se puede compartir su información. El Sistema Operativo controla que estos privilegios no sean violados.

El administrador de archivos puede controlar la asignación de nombres a los archivos y es el responsable de hacer los respaldos. Se encarga de gestionar volúmenes de datos, a través de la creación de directorios.

El conjunto de módulos del S.O. que se encarga de la gestión de archivos y directorios se conoce como *sistema de ficheros*.

4.3.4 Apoyo a otros programas

El Sistema Operativo proporciona servicios a otros programas, liberando al programador de ciertas tareas engorrosas que realiza en forma automática: transferencia de información entre soportes, revisión del espacio disponible en memoria y los soportes de información, ordenación de datos en archivos, etc.

4.3.5 Protección

Al ejecutarse varios programas simultáneamente en el computador es necesario algún mecanismo de protección entre los programas, básicamente contra dos tipos de sucesos:

1. Errores de los programas. El S.O. debe ser capaz de detectar los errores de los programas de aplicación y diagnosticarlos, cancelando el programa y enviando el correspondiente mensaje al usuario.
2. Abuso intencionado de los recursos del sistema. Cuestión difícil de resolver por los sistemas operativos.

La protección se realiza en todos los niveles, pero se presta especial cuidado a la seguridad de la memoria principal y soportes de almacenamiento masivo. El módulo de gestión de memoria principal del S.O. asigna a cada proceso una porción de memoria estableciendo fronteras de separación y luego asigna un grado de protección a cada porción según la naturaleza del proceso. El núcleo del S.O. tiene el nivel más alto de protección y los programas de aplicación poseen el más bajo.

4.4.4 Contabilidad del uso de los recursos

Los sistemas operativos multiusuarios mantienen la contabilidad del gasto de recursos que realiza cada usuario, por ejemplo, tiempo de CPU consumido por cada proceso, uso de los soportes de almacenamiento masivo y de papel impreso por cada usuario.

4.4 Tipos de Sistemas Operativos

MONOTAREA O SERIE.	<p>Primeros sistemas operativos, en ellos hasta que no finalizaba la ejecución de un programa no empezaba a ejecutarse otro.</p> <p>El rendimiento alcanzado con estos S. O. era muy bajo, debido a la existencia de tiempos en los cuales el procesador no realizaba ningún trabajo útil (por ejemplo, mientras se realizaban E/S).</p>
MULTITAREA O MULTIPROGRAMADOS.	<p>Capaces de ejecutar más de un programa al mismo tiempo. Son los más usados en la actualidad.</p> <p>Dependiendo como gestiona el procesador podemos clasificarlos en:</p> <ul style="list-style-type: none"> • <i>Cooperativa.</i> Existe una cooperación entre el S.O. y los programas de aplicación. Los procesos periódicamente detectan si otro proceso necesita el procesador, de ser así el proceso en ejecución deja el control del procesador al siguiente programa. El S.O. no tiene el control del procesador de forma autónoma al decidir que programa se ejecutará, depende de lo que determine la aplicación. • <i>Con asignación de prioridades.</i> El S.O. mantiene una lista con los procesos que intervienen en cada momento, con sus prioridades. El S.O. puede cambiar la prioridad de un proceso si lo necesita. Es el responsable de pasar el control del procesador a un proceso o a otro, según prioridades.
MONOUSUARIO.	<p>Simples, permiten la conexión de un único usuario en un instante dado, por lo cual no necesitan realizar la gestión y control de varios usuarios. Pueden ser monotarea o multitarea.</p>
MULTIUSUARIO.	<p>Más de un usuario accede al computador al mismo tiempo.</p> <p>El S.O. debe ser también multitarea y establecer mecanismos de identificación, autenticación y control de los distintos usuarios.</p> <p>Además cada usuario puede ejecutar varios programas simultáneamente. Ej. UNIX.</p>

MULTIPROCESO.	<p>Dos o más procesadores interconectados trabajando simultáneamente, formando un único ordenador.</p> <ul style="list-style-type: none"> • <i>Multiproceso asimétrico</i>, un procesador principal controla el comportamiento global de todos los demás. Inconveniente: el procesador principal puede sobrecargarse en las tareas de administración. Si se agregan procesadores no se consigue un aumento lineal de las prestaciones. Existen versiones de UNIX para procesamiento asimétrico. • <i>Multiproceso simétrico</i>. No existe un procesador controlador único. Si se agregan procesadores se consigue aumentar linealmente la capacidad del sistema. Inconveniente: su implementación exige la actualización y rediseño de los sistemas operativos y de los compiladores. Existen versiones de UNIX y WINDOWS NT para procesamiento simétrico.
EN TIEMPO REAL.	<p>Para el control de aplicaciones en Tiempo Real (en las cuales el factor tiempo es crucial). Los S.O. en tiempo real deben ser capaces de responder a determinados eventos en plazos de tiempos previamente determinados. Muy usados en control industrial, control de vuelo, etc.</p>

4.5 Ejemplos de Sistemas Operativos

Algunos ejemplos de S.O. son:

- LINUX.
- La familia de S.O. WINDOWS (2000, CE, XP, VISTA, SERVER 2003, etc.).
- SOLAIS.
- OS/400.
- IRIX.

5. Lenguajes de programación

5.1 Introducción

Construir programas directamente utilizando el lenguaje de la máquina presenta grandes dificultades, para superar esto se han desarrollado otros lenguajes adecuados para programar cualquier computadora, más fáciles de aprender y utilizar, reduciendo la posibilidad de cometer errores y que puedan ser traducidos al lenguaje que entiende una máquina concreta.

Un lenguaje de programación es un conjunto de símbolos junto a un conjunto de reglas para combinar dichos símbolos que se usan para escribir programas.

Como cualquier tipo de lenguaje, se compone de un *léxico* predefinido (conjunto de símbolos permitidos), una *sintaxis* predefinida (reglas que indican como realizar la construcción del lenguaje) y una *semántica* (reglas que permiten determinar el significado de cualquier construcción del lenguaje).

Para que una computadora pueda ejecutar un programa escrito en un determinado lenguaje de programación, es necesaria primero una *traducción* del programa al lenguaje que entiende dicha computadora (lenguaje de máquina), según una serie de etapas.

5.2. Lenguaje de máquina

Fue el primer lenguaje utilizado en la programación de computadoras. Es el único lenguaje que entiende directamente un computador, por lo cual su estructura está totalmente ligada a los circuitos de la máquina y muy alejada de la forma de expresión y análisis de los problemas propios del hombre.

Frente a lo complicado que resulta programar en lenguaje de máquina, el código de máquina hace posible que el programador aproveche al máximo los recursos existentes.

Sus principales características son:

- Las instrucciones se expresan en binario como cadenas de ceros y unos, pudiéndose utilizar códigos intermedios (octal y hexadecimal), esto hace a los programas difíciles de entender y modificar.
- Los datos se referencian por medio de las direcciones de memoria donde se encuentran (no usan nombres de variables).
- Las instrucciones realizan operaciones muy simples, debiendo el programador expresar cada una de las operaciones que desee realizar solo con las instrucciones elementales que dispone.
- Poca versatilidad para la redacción de las instrucciones, formato muy rígido en cuanto a la posición de los distintos campos.
- Poca *portabilidad*, un programa en lenguaje de máquina solo se puede ejecutar en el procesado para el cual está destinado, por estar íntimamente ligado a la CPU del ordenador.
- No puede incluirse comentarios que ayuden a la lectura del mismo.

A continuación se muestra un programa en lenguaje de máquina de un ordenador de 16 bits de palabra:

Instrucción de máquina en binario
0001 0001 0011 1011
1101 0011 1001 1110
1010 0101 1100 1101
1001 0111 0011 1111

5.3. Lenguaje ensamblador o simbólico

Es el primer intento de sustituir el lenguaje de máquina por uno más cercano al hombre.

Si bien presenta la mayoría de los inconvenientes del lenguaje de máquina (repertorio reducido de instrucciones, rígido formato de las instrucciones, baja portabilidad y fuerte dependencia del hardware), presenta las siguientes ventajas con respecto al mismo:

- Se permite el uso de comentarios entre líneas de instrucciones.

- Evita los códigos numéricos usando una notación simbólica o nemotécnica (generalmente constituidos por las abreviaturas de las operaciones en inglés) para representarlos. Por ejemplo: la suma se representa en la mayoría de los ensambladores por ADD.
- Direccionamiento simbólico. Los datos pueden identificarse con nombres, por ejemplo: FECHA, IMPORTE, IVA, etc., en lugar de direcciones binarias absolutas.

Al igual que el lenguaje de máquina goza de la ventaja de mínima ocupación de memoria y mínimo tiempo de ejecución en comparación con el resultado de la compilación del programa escrito en otros lenguajes. El lenguaje ensamblador hace corresponder a cada instrucción en ensamblador una instrucción en código máquina. Esta traducción es realizada por un programa traductor llamado **ensamblador**.

Los programadores usan este lenguaje para afinar partes importantes de programas escritos en lenguajes de más alto nivel.

El lenguaje ensamblador da al programador el control total de la máquina, permitiéndole generar un código compacto, rápido y eficiente. Ejemplo de un fragmento de código en lenguaje ensamblador:

INICIO:	ADD	B, 1
	MOV	A, B
	CMP	A, E
	JE	FIN
	JMP	INICIO
FIN:	END	

5.4. Lenguajes de alto nivel

El esfuerzo por hacer la tarea de programación independiente de la máquina condujo al desarrollo de lenguajes de alto nivel. Sus características principales son:

- Independencia de la arquitectura física del computador, lo cual permite utilizar un mismo programa en diferentes equipos (portabilidad), sin necesidad de conocer el hardware específico.
- Requiere de una traducción al lenguaje de máquina de la computadora donde va a ejecutarse.
- Se aproximan al lenguaje natural, para que el programa se pueda leer y escribir de una forma más sencilla. Las instrucciones vienen expresadas mediante texto, permitiendo incluir comentarios.
- Por lo general, una sentencia da lugar, al ser traducida a varias instrucciones en lenguaje de máquina.
- Se incluyen rutinas de uso frecuente, como las de Entrada/Salida, funciones matemáticas de uso frecuente (seno, coseno, conversión de entero a real), que figuran en una especie de librería del lenguaje, las cuales se pueden utilizar sin necesidad de programarlas cada vez.
- El lenguaje de alto nivel, a diferencia del de máquina y el ensamblador, no permite aprovechar totalmente los recursos internos de la máquina.

El principal problema que presentan los lenguajes de alto nivel es la gran cantidad de ellos que existen actualmente y sus diferentes versiones.

Algunos de los más conocidos son: Fortran, Cobol, C++, Java, Lisp, Prolog, Modula-2, Visual Basic, PHP, XML.

Todas estas características ponen de manifiesto un acercamiento a las personas y un alejamiento de la máquina, por eso los programas de alto nivel no pueden ser directamente interpretados por las computadoras, siendo necesario una traducción previa a lenguaje de máquina. Para ello se utilizan programas traductores, previamente desarrollados para cada computador, que se encargan de realizar dicho proceso.

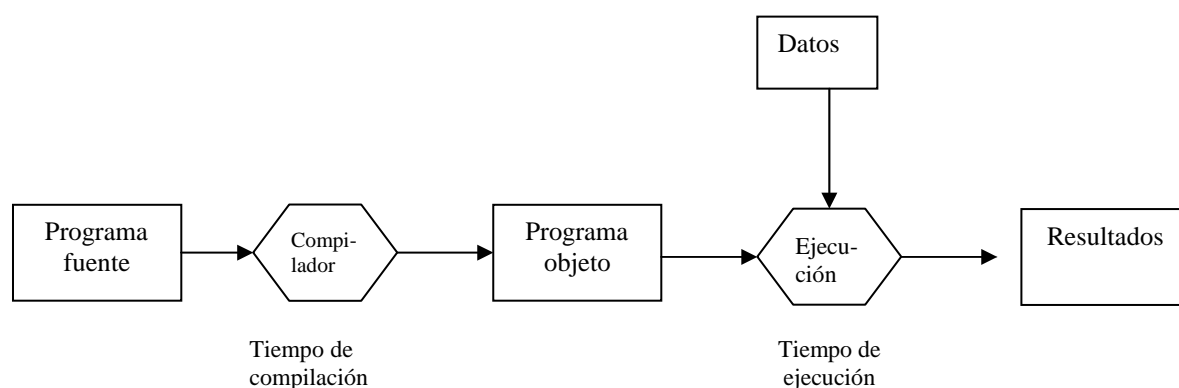
5.5. Traductores. Compiladores e intérpretes

Un **traductor** es un metaprograma que toma como entrada un programa (o parte de él) escrito en lenguaje simbólico alejado de la máquina denominado **programa fuente** y proporciona como salida otro programa semánticamente equivalente, escrito en un lenguaje comprensible por la máquina denominado **programa objeto**.

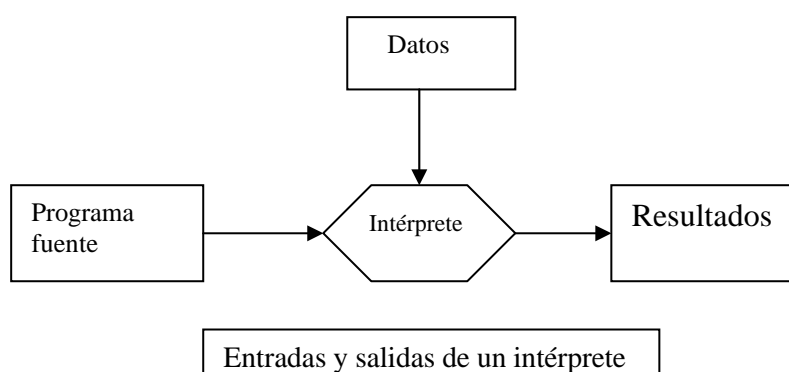
Veremos dos tipos de traductores, los compiladores y los intérpretes, los cuales realizan la tarea de manera muy distinta.

Un **compilador** traduce completamente un programa fuente, generando un programa objeto escrito en lenguaje de máquina. En este proceso de traducción, se informa al usuario de errores en el programa fuente, sólo se crea el programa objeto si no se detectaron errores.

Compilación y ejecución de un programa:



Un **intérprete** permite que un programa fuente escrito en un determinado lenguaje vaya traduciéndose y ejecutándose directamente sentencia a sentencia por la computadora. Capta una sentencia fuente, la analiza y la interpreta, teniendo lugar su ejecución inmediata.



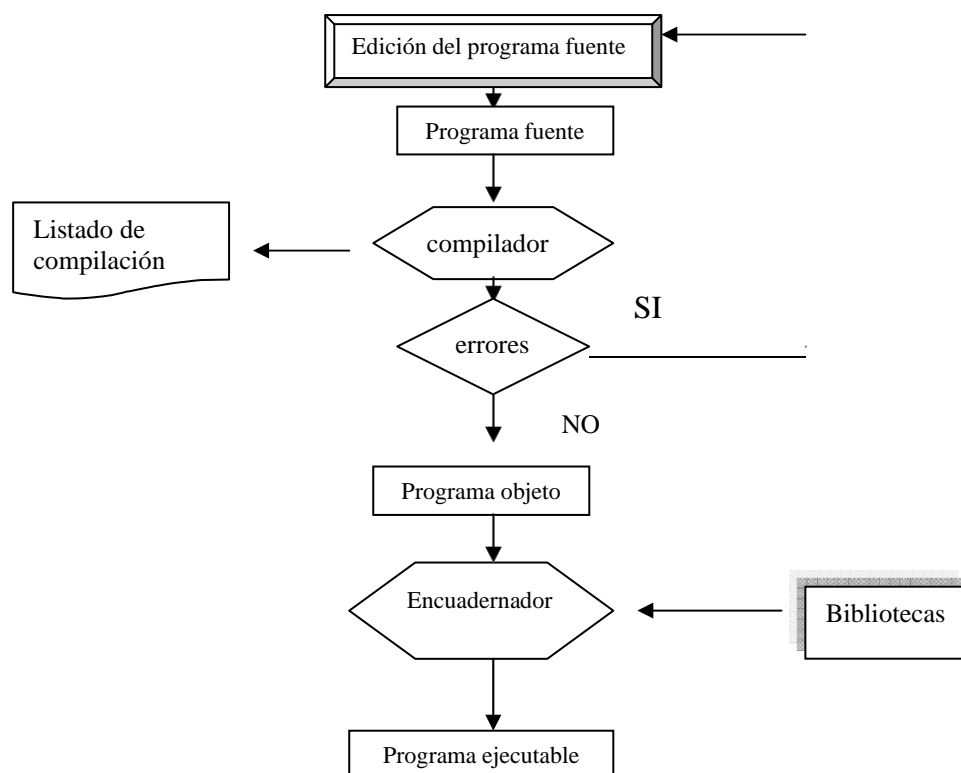
El intérprete no crea ningún programa objeto, por lo cual cada vez que se deba ejecutar el programa, se deberá traducir, en cambio, con un compilador, aunque sea más lenta, la traducción sólo debe realizarse una vez. La principal ventaja del intérprete respecto a los compiladores, es que resulta más fácil depurar los programas, ya que permite interrumpir en cualquier momento para conocer por ejemplo, los valores de las distintas variables o la instrucción fuente que acaba de ejecutarse, por estos motivos se los prefiere para fines pedagógicos.

5.5. Proceso de compilación.

Previo al proceso de compilación de un programa, se crea el programa fuente usando cualquier editor de texto. Para que la compilación sea posible el programa fuente debe residir en memoria simultáneamente con el compilador.

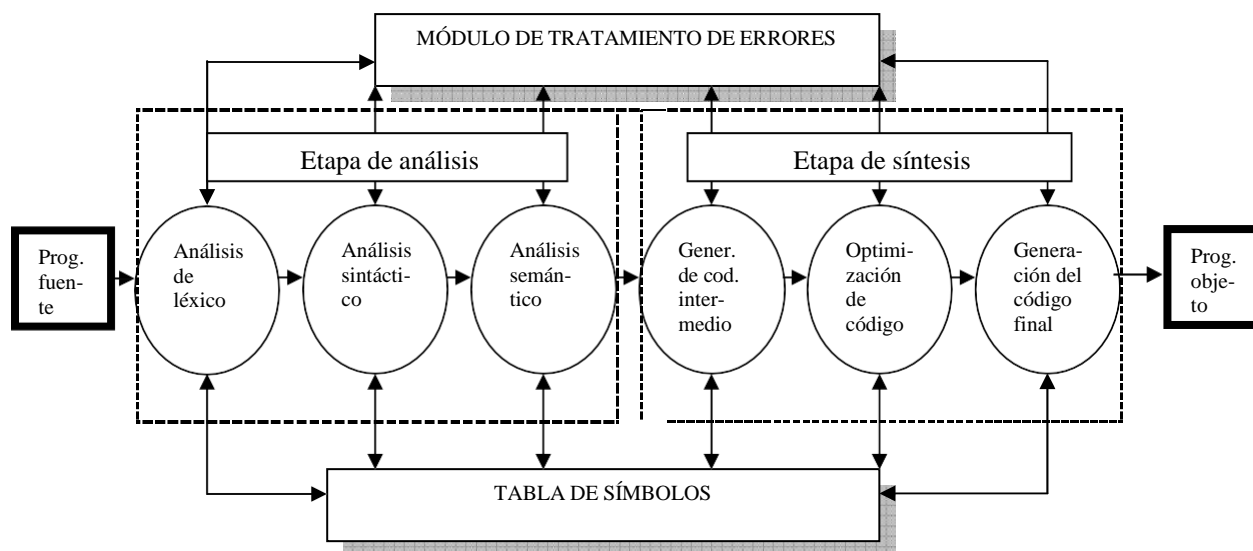
Si como resultado de la compilación se notifican errores, no se genera programa objeto, pero se genera un informe (listado de compilación) indicando la naturaleza de los mismos. Este último permitirá, volviendo al programa editor, corregir los errores y empezar de nuevo el proceso.

Terminada la compilación y obtenido el programa objeto, se lo somete a un proceso de montaje donde se enlazan los distintos módulos que lo componen (en caso de programas que poseen subprogramas, los cuales pueden ser compilados separadamente) y se incorporan las denominadas rutinas de bibliotecas. Este montaje es realizado por un programa denominado **montador o encuadernador o editor de enlace o linker**.



Proceso de compilación de un programa

El proceso de compilación consta en general de dos *etapas* fundamentales: **análisis del programa fuente** y **síntesis del programa objeto**. Cada consiste en la realización de varias *fases* según el siguiente esquema:



Etapa de análisis - Fases

Lexicográfico: realizado por el *analizador lexicográfico o escáner*.

Se examina el programa fuente de izquierda a derecha, eliminando información supérflua (comentarios, tabulaciones, etc.) y reconociendo las unidades básicas de información (unidades léxicas o tokens) pertenecientes al lenguaje. Un token es una cadena de caracteres que tiene un significado propio en el lenguaje, ejemplo: palabras reservadas, los identificadores de variables, etc.).

Además de reconocer cada tokens, almacena en la tabla de símbolos información el mismo para ser usada en otras fases.

De no existir errores, el programa queda representado por :

- La descripción del símbolo en la tabla de símbolos.
- Una secuencia de símbolos (tira de tokens), conteniendo cada símbolo y una referencia a la ubicación de dicho símbolo en la tabla de símbolos.

Los errores detectados en esta fase son debido a la detección de cadenas de caracteres que no se ajustan a las descripción de ningún tokens, por ejemplo, en C la sentencia:

A=5, no es correcta (como final de sentecia se utiliza ; y no ,) Este error seria detectado por el analizador de léxico.

Sintáctico: Realizado por el *analizador sintáctico o parser*.

La sintaxis especifica como deben escribirse los programas en base a un conjunto de reglas sintácticas que determinan la gramática del lenguaje.

Se examina si sus estructuras (expresiones, asignaciones, etc.) aparecen dispuestas de forma correcta de acuerdo a la gramática del lenguaje. Ejemplo, en C:

Imp * 18,5 = incre es sintácticamente incorrecta (antes del operador de asignación solo puede haber un único identificador).

	<p><u>Semántico:</u> Se encarga de estudiar la coherencia semántica del código fuente a partir de la identificación de las construcciones sintácticas y de la información de la tabla de símbolos. Detecta construcciones sin un significado correcto.</p> <p>Ejemplo:</p> <p style="text-align: center;">impor = "hola"; donde impor fue declarada de tipo real</p> <p>Esta sentencia no es semánticamente correcta, pues no se puede asignar a una variable real una cadena de caracteres.</p>
Etapas de síntesis- Fases	<p><u>Generación del código objeto.</u> Se traduce el resultado de la etapa de análisis a un lenguaje intermedio propio del compilador, con características propias del compilador. Este código intermedio debe ser independiente de la máquina, fácil de producir y fácil de traducir a un lenguaje de máquina. Está presente en los compiladores construidos para ser implementados en distintas máquinas, todos los módulos son iguales para todas las máquinas (aumenta la portabilidad) y será particularizado para cada familia de ordenadores en las fases siguientes.</p> <p><u>Optimización del código.</u> Se optimiza el código intermedio, adaptándolo a las características del procesador al que va dirigido.</p> <p><u>Generación del código objeto:</u> Se traduce el código intermedio optimizado al código final, es decir, al lenguaje de máquina (a veces a ensamblador) del procesador al que el compilador va dirigido.</p>
Tabla de símbolos	<p>El compilador la utiliza internamente para introducir determinados datos que necesita (tipo de cada variable o estructura de dato, el número de argumentos de cada función, la memoria asignada a cada variable, etc.). Interviene en todas las fases del proceso de compilación.</p>
Módulo de tratamiento de errores	<p>Permite determinar las reacciones que se deben producir ante la aparición de cualquier tipo de error. Comprende la realización de dos acciones:</p> <p>1. Diagnóstico del error. Trata de buscar su localización exacta (línea del programa donde se detectó) y la posible causa (tipos de error), para ofrecer al programador un mensaje del diagnóstico.</p> <p>2. Recuperación del error: al detectar un error, cada fase debe tratarlo de alguna forma para poder continuar la compilación y permitir la detección de más errores, aunque no se genere código objeto.</p> <p>Los tipos de errores que puede presentar un programa son:</p> <ul style="list-style-type: none"> - Lexicográficos, sintácticos y semánticos (ya estudiados). - Errores lógicos: son debido a la utilización de un algoritmo incorrecto. Se detectan en la fase de prueba utilizando un juego de prueba bien estudiado. Por ejemplo, un programa donde deben calcularse intereses por compra con tarjetas de créditos y lo realiza también para compras al contado. - Errores de ejecución: Son errores relacionados con desbordamientos, operaciones matemáticamente irresolubles (por ejemplo: división por cero).

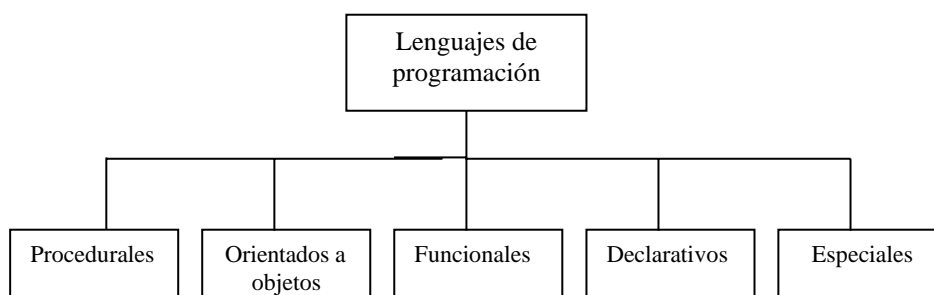
5.5. Clasificación de los lenguajes de programación.

Podemos clasificar los lenguajes de programación según su desarrollo histórico paralelo a la evolución de las computadoras:

1. Primera generación. Lenguajes de máquina.
2. Segunda generación. Ayudas a la programación como son los ensambladores.
3. Tercera generación: lenguajes de alto nivel imperativo. Como ser Pascal, Modula-2, Fortran, Cobol, C y Ada.
4. Cuarta generación (4G): Lenguajes o entornos de programación orientados básicamente a aplicaciones de gestión y bases de datos, como SQL, Natural, etc.
5. Quinta generación: lenguajes orientados a inteligencia artificial, como Lisp y Prolog.

Clasificación de los lenguajes de alto nivel

Se puede decir que el principal problema que presentan los lenguajes de programación de alto nivel es la gran cantidad que existen actualmente en uso, además de las diferentes versiones que se han desarrollado de algunos de ellos. Es difícil establecer una clasificación general de los mismos, ya que hay lenguajes que pertenecen a más de uno de los grupos establecidos. Una clasificación muy extendida, atendiendo a la forma de trabajar de los programas y a la filosofía con que fueron concebidos, es la siguiente:

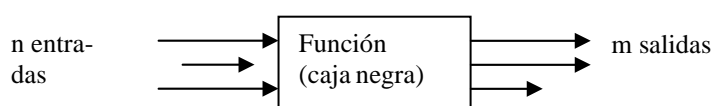


- **Lenguajes procedurales o imperativos.** Utilizan el método tradicional de programación. Siguen el mismo método empleado por el hardware para ejecutar un programa (buscar y traer, decodificar y ejecutar). La programación imperativa describe paso a paso un conjunto de instrucciones que deben ejecutarse para solucionar un problema particular. Cuando un programador necesita resolver un problema usando un lenguaje procedural debe saber cuál es el **procedimiento** a seguir.

Una instrucción cambia los valores almacenados en las posiciones de memoria o los mueve a alguna otra parte o es una instrucción de control que localiza la siguiente instrucción a ejecutarse, por eso el nombre de **lenguaje imperativo**: cada instrucción es un comando para que el sistema realice una tarea específica.

Pascal, C, Basic, Modula-2 y Fortran son algunos de los lenguajes procedurales más importantes.

- Lenguajes orientados a objetos.** Su metodología difiere totalmente de la programación procedural. Se puede pensar en un elemento de datos en ambos lenguajes como un objeto y un programa puede imaginarse como una secuencia de operaciones que desea realizar en el objeto. En la programación procedural, los objetos están completamente separados e independientes de las operaciones, se almacenan en el computador y diferentes programas se aplican a ellos, son pasivos, no tienen ninguna operación definida para ellos, los programadores definen las operaciones y las aplican a los objetos. En cambio, en la programación orientada a objetos, los objetos y las operaciones que se aplican a ellos están ligadas. El programador primero define un objeto y los tipos de operaciones que pueden aplicarse a ese objeto, pudiendo luego usar esta combinación e invocar algunas o todas las operaciones definidas, los objetos son activos. Algunos de los lenguajes orientados a objetos desarrollados son Smalltalk, Java, C++.
- Lenguajes funcionales:** los programas están constituidos únicamente por funciones, entendiendo el concepto de función según su definición matemática, y no como simples subprogramas de los lenguajes imperativos. En este contexto, una función es una caja negra que correlaciona una lista de entradas con una lista de salidas.

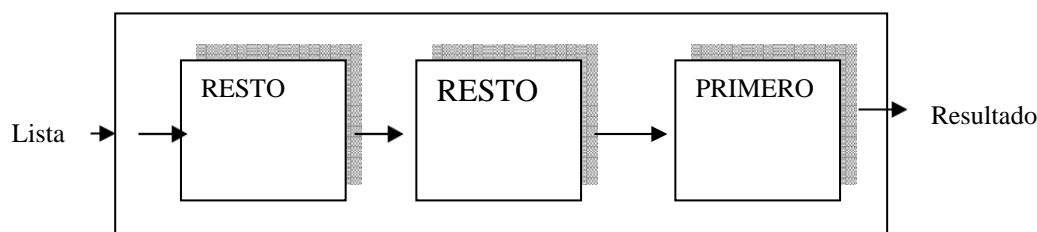


Por ejemplo, la *suma* puede considerarse una función con n entradas y solo una salida; toma las n entradas, las suma y crea el resultado de la suma.

Un lenguaje funcional realiza lo siguiente:

1. predefine una serie de funciones primitivas (atómicas) que puede usar cualquier programador.
 2. Permite al programador combinar funciones primitivas para crear funciones nuevas.
- Ejemplo:

Definimos una función primitiva llamada PRIMERO que extrae el primer elemento de una lista y otra llamada RESTO que extraiga todos los elementos excepto el primero. Un programa puede definir una función que extraiga el tercer elemento de una lista combinando las dos funciones:



TERCERO

Una característica propia de este lenguaje es la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o iteración (las repeticiones de instrucciones se llevan a cabo por medio de funciones recursivas).

Un lenguaje funcional tiene ventajas sobre uno procedural, fomenta la programación modular y permite al programador hacer nuevas funciones, factores que ayudan al programador a crear programas grandes y menos propensos a errores a partir de programas ya probados. Ejemplo: LISP.

- **Lenguajes declarativos (lógicos).**

Los programas están contruidos únicamente por expresiones lógicas, es decir, que son ciertas o falsas, en oposición a una expresión interrogativa (una pregunta) o imperativa (una orden).

Se basa en la lógica formal definida por los matemáticos griegos y desarrollada posteriormente en lo que se llama *cálculo de predicado de primer orden*.

El razonamiento lógico se basa en la deducción. Se dan algunas instrucciones o hechos que se supone son verdaderos, la lógica utiliza reglas rigurosas de razonamiento lógico para deducir nuevas instrucciones.

Por ejemplo, la famosa regla de deducción en lógica es:

SI (A es igual a B) y (B es igual a C), entonces (A es igual a C)

Usando esta regla y los dos hechos siguientes,

Hecho 1:	Sócrates es un ser humano	→	A es igual a B
Hecho 2:	Un humano es mortal	→	B es igual a C

Podemos deducir un nuevo hecho:

Hecho 3:	Sócrates es mortal.	→	A es igual a C.
----------	---------------------	---	-----------------

Este tipo de programación permite la división de tareas, el programador es el encargado de suministrar la parte lógica, hechos y reglas que representan conocimientos, un intérprete se ocupa de establecer un orden en que se han de aplicar las reglas.

Los programadores deben conocer todos los hechos en el dominio de su tema u obtenerlos de expertos en el campo, además de ser expertos en lógica para definir las reglas en forma adecuada.

Estos lenguajes presentan un problema: un programa es específico para un dominio en particular, por que la recopilación de todos los hechos en un programa hace que el programa se vuelva enorme, por esta razón este tipo de programación está limitada a campos específicos, por ejemplo Inteligencia Artificial. El lenguaje declarativo más famoso es Prolog.

- **Lenguajes especiales. Una visión a la WEB y al futuro.**

Algunos lenguajes que han aparecido durante las últimas décadas, no pueden encuadrarse en las categorías analizadas, por ser combinaciones de dos o más modelos o pertenecer a una tarea específica.

Internet ha sido el disparador de nuevos lenguajes tales como el **HTML** que es el lenguaje de programación de las páginas WEB para hipertexto. Es un pseudolenguaje que incluye marcas que sirven como sugerencias de formato y vínculos a otros archivos. El mismo constituye una codificación bastante simple, basada en marcadores (TAGs).

Un archivo HTML (página) se guarda en el servidor y puede descargarse mediante un explorador, el cual elimina las etiquetas y las interpreta como sugerencias de formatos o vínculos a otros archivos.

Un lenguaje de marcado como el HTML le permite incrustar instrucciones de formato en el archivo mismo, las instrucciones se almacenan con el texto, de forma que cualquier explorador puede leer las instrucciones y dar formato al texto de acuerdo con la estación de trabajo que se está usando.

De la misma manera, cuando se hace necesario proveer de funciones adicionales a un servicio web se recurre a PERL (Practical Extraction and Report Language), lenguaje práctico de extracción e informes, un lenguaje de alto nivel con una sintaxis similar al lenguaje C (pero más eficiente) o al lenguaje Python, ideal para principiantes, debido a que en general es más fácil de aprender, su código es más limpio, multiplataforma, admite tanto programación modular como orientada a objetos.

Por otra parte Java, bajo la dirección de SUN, constituye la idea de la programación abierta y universal para las aplicaciones de escritorio, pero todavía los estándares visuales (C, Basic y Delphi), son demasiado poderosos como para desplazarlos.

La popularidad de los lenguajes de programación (Oct 08)

Cada año se realizan estudios en los que se estima cuál es el lenguaje de programación más popular. Este año (2008), al igual que en los últimos, Java ocupa la primera posición, seguido por C y C++, estos suman ya casi el 50% de las aplicaciones desarrolladas.

Hay que ir a la posición 5 para encontrarnos el primer lenguaje destinado al desarrollo web, PHP, con 8,94% de las aplicaciones desarrolladas se considera un año más como el lenguaje de desarrollo web por excelencia.

