Tipos de base de datos

Existen 3 tipos de base de datos los cuales son: jerárquico, de red y relacional.

Tipo jerárquico

Una base de datos de tipo jerárquico utiliza jerarquías o árboles para la representación lógica de los datos. Los archivos son organizados en jerarquías, y normalmente cada uno de ellos se corresponde con una de las entidades de la base de datos. Los árboles jerárquicos se representan de forma invertida, con la raíz hacia arriba y las hojas hacia abajo (*Figura 1*).

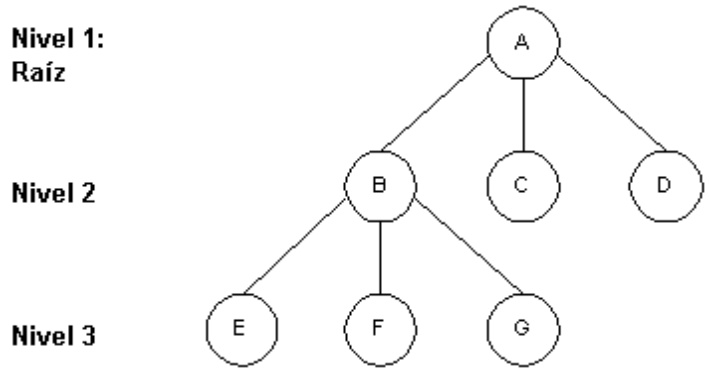

Figura 1 *Estructura de un árbol jerárquico*

Una base de datos de tipo jerárquico recorre los distintos nodos de un árbol en un *preorden* que requiere tres pasos:

- Visitar la raíz.
- Visitar el hijo más a la izquierda, si lo hubiera, que no haya sido visitado.
- Si todos los descendientes del segmento considerado se han visitado, volver a su padre e ir al punto 1.

Cada nodo del árbol representa un tipo de registro conceptual, es decir, una entidad. A su vez, cada registro o segmento está constituido por un número de campos que los describen − las propiedades o atributos de las entidades. Las relaciones entre entidades están representadas por las ramas. En la *Figura 2.* cada departamento es una entidad que mantiene una relación de uno a muchos con los profesores, que a su vez mantienen una relación de uno a muchos con los cursos que imparten.

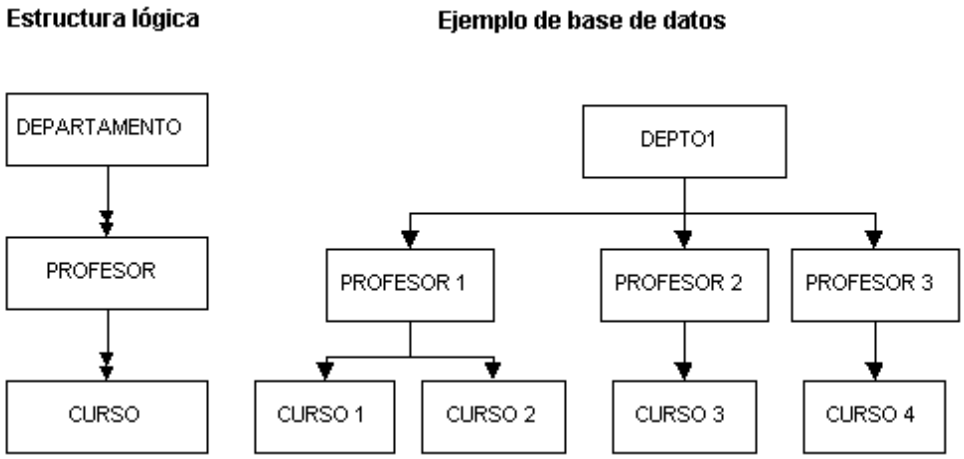

Figura 2. *Base de datos jerárquica. Estructura lógica y ejemplo*

A modo de resumen, enumeramos las siguientes características de las bases de datos jerárquicas:

- Los segmentos de un archivo jerárquico están dispuestos en forma de árbol.
- Los segmentos están enlazados mediante relaciones uno a muchos.
- Cada nodo consta de uno o más campos.
- Cada ocurrencia de un registro padre pueden tener distinto número de ocurrencias de registros hijos.
- Cuando se elimina un registro padre se deben eliminar todos los registros hijos (integridad de los datos.
- Todo registro hijo debe tener un único registro padre excepto la raíz.

Las reglas de integridad en el modelo jerárquico prácticamente se reducen a la ya mencionada de eliminación en cadena de arriba a abajo. Las relaciones muchos a muchos no pueden ser implementados de forma directa. Este modelo no es más que una extensión del modelo de ficheros.

Como ejemplos de base de datos basados en este enfoque podemos citar el IMS de IBM Corporation y el SYSTEM 2000 de Intel Corporation.

Base de datos tipo red.

Este modelo fue el resultado de estandarización del comité CODASYL. Aunque existen algunas bases de datos de red que no siguen las especificaciones CODASYL, en general, Una base de datos CODASYL es sinónimo de base de datos de red. El modelo de red intenta superar las deficiencias del enfoque jerárquico, permitiendo el tipo de relaciones de muchos a muchos.

Una estructura de datos en red, o estructura *plex*, es muy similar a una estructura jerárquica, de hecho no es más que un súper conjunto de ésta. Al igual que en la estructura jerárquica, cada nodo puede tener varios hijos pero, a diferencia de ésta, también puede tener varios padres. La *Figura 3* muestra una disposición plex. En esta representación, los nodos C y F tienen dos padres, mientras que los nodos D, E, G y H tienen sólo uno.

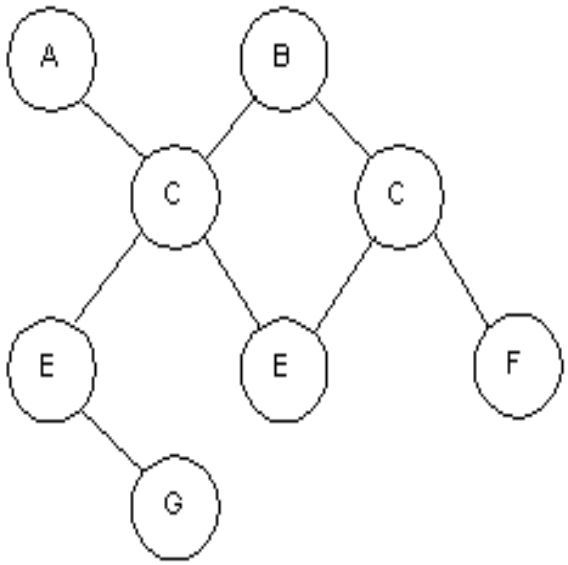

Figura 3 *Estructura de datos de red*

El concepto básico en el enfoque de red es el *conjunto* (`set'), definido por el comité CODASYL. Un conjunto está constituido por dos tipos de registros que mantienen una relación de muchos a muchos. Para conseguir representar este tipo de relación es necesario que los dos tipos de registros estén interconectados por medio de un registro conectivo llamado *conjunto conectivo*. Los conjuntos poseen las siguientes características:

- El registro padre se denomina *propietario* del conjunto, mientras que el registro hijo se denomina *miembro*.
- Un conjunto está formado en un solo registro propietario y uno o más registros miembros.
- Una *ocurrencia de conjuntos* es una colección de registros, uno de ellos es el propietario y los otros los miembros.
- Todos los registros propietarios de ocurrencias del mismo tipo de conjunto deben ser del mismo tipo de registro.
- El tipo de registro propietario de un tipo de conjunto debe ser distinto de los tipos de los registros miembro.
- Sólo se permite que un registro miembro aparezca una vez en las ocurrencias de conjuntos del mismo tipo.
- Un registro miembro puede asociarse con más de un propietario, es decir, puede pertenecer al mismo tiempo a dos o más tipos de conjuntos distintos. Esta situación se puede representar por medio de una estructura Mult. Anillo.
- Se pueden definir niveles múltiples de jerarquías donde un tipo de registro puede ser miembro en un conjunto y al mismo tiempo propietario en otro conjunto diferente.

Como ejemplos de DBMSs comerciales basados en el modelo de red cabe citar el DMS 1100 de UNIVAC; el IDMS, de Cullinane; el TOTAL, de Cincom; el EDMS, de Xerox; el PHOLAS, de Philips; el DBOMP, de IBM, y el IDS, de Honeywell. Tanto el modelo jerárquico de datos como el de red permiten únicamente operaciones y facilidades navegacionales primitivas.

Base de datos de tipo relacional

El modelo relacional de datos supuso un gran avance con respecto a los modelos anteriores. Este modelo está basado en el concepto de *relación*. Una relación es un conjunto de *n*−tuplas. Una tupla, al contrario que un segmento, puede representar tanto entidades como interrelaciones13 N:M. Los lenguajes matemáticos sobre los que se asienta el modelo relacional, el álgebra y el cálculo relacionales, aportan un sistema de acceso y

consultas orientado al conjunto. La repercusión del modelo en los DBMSs comerciales actuales ha sido enorme, estando hoy en día la gran mayoría de los gestores de bases de datos basados en mayor o menor medida en el modelo relacional.

El concepto de *modelo de datos* en sí surgió al mismo tiempo que el modelo relacional de datos fuera propuesto por su creador, Ted Codd, después de que los modelos jerárquico y de red estuvieran en uso. Posteriormente, estos dos modelos fueron definidos independientemente de los lenguajes y sistemas usados para implementarlos. Con anterioridad no eran más que colecciones de estructuras de datos y lenguajes sin una teoría subyacente definida. En cuanto al modelo relacional, no se puede decir que sea en sí un modelo semántico de datos. Su enorme éxito no se debe a que permite de forma implícita operaciones conceptualmente abstractas sobre los datos, sino a los altos niveles de fiabilidad e integridad que aporta en el manejo de grandes cantidades de datos.

Desde su comienzo en 1970 y durante mucho tiempo después, los sistemas gestores de bases de datos relacionales (*RDBMS : Relational Database Management System*) estuvieron restringidos al ámbito de los mainframes y mini−ordenadores. Con la irrupción masiva en el mercado de los micro−ordenadores, aparecieron algunas implementaciones de RDBMSs que intentaban emular las propiedades de los grandes sistemas, aunque no contaban con la mayor parte de las características necesarias para ser denominados "relacionales", especialmente en lo que se refiere al cumplimiento de las reglas de integridad relacional.

Hoy en día contamos con RDBMSs para micro−ordenadores que sí pueden ser considerados plenamente relacionales y que, si bien no llegan alcanzar las prestaciones de los grandes sistemas en cuanto a velocidad de ejecución, seguridad, integridad de datos, recuperación y estabilidad, no tienen nada que envidiar a éstos cualitativamente, y sus deficiencias se deben sobre todo al tipo de máquina en el que funcionan y a los sistemas operativos que estas máquinas utilizan.

Lo que realmente marca la diferencia entre los sistemas relacionales y los sistemas anteriores es el hecho de que su creador, Ted Codd, basó expresamente su funcionamiento sobre un modelo matemático muy específico: el álgebra relacional y el cálculo relacional, así como la progresiva adopción, por parte de su creador y algunos colaboradores, de un número de Reglas de Integridad Relacional y de Formas Normales.

La definición formal y exhaustiva más actualizada del modelo se encuentra en (Codd 1990). Además existe un buen número de obras que tratan el modelo desde diversas perspectivas; Entre éstos destacamos la obra, ya clásica, de C. J. Date (Date 1990. En este apartado resumiremos los conceptos más importantes del modelo relacional. Lo que exponemos a continuación es, en esencia, un resumen de la obra de Codd (Codd 1990).

Ventajas e inconvenientes del modelo relacional

Durante la exposición en los apartados anteriores y el capítulo anterior de las bases de datos en general y el modelo relacional en particular, hemos comentado las características más sobresalientes de este tipo de sistemas de información. Las ventajas de utilizar un RDBMS podrían ser resumidas en las siguientes:

- Compatibilidad y estandarización.
- Fiabilidad.
- Garantía de independencia de los datos.
- Existencia de numerosos sistemas comerciales entre los que escoger y consiguiente apoyo técnico.
- Conectividad garantizada con los lenguajes de programación estándar.

En general, un RDBMS cumple con los requisitos que expusimos al principio del Capítulo 4, por lo que parece una elección razonable. El RDBMS que hemos utilizado para nuestra implementación, Microsoft Access, cumple todas ellas, estando considerado, en su versión 8 (Access 97) como uno de los RDBMSs para estaciones de trabajo bajo plataforma Win32 más sólidos y versátiles del mercado, ofreciendo todas las

garantías de conectividad y estabilidad deseables, así como uno de los motores de bases de datos más rápidos.

Sin embargo, también hemos de ser conscientes de los aspectos negativos, o más bien limitaciones, que conlleva la adopción un modelo de datos con una veintena de años. Existen una serie de desventajas bien conocidas del modelo relacional de datos, que se ponen de manifiesto especialmente cuando lo comparamos con otros modelos más nuevos (p. ej. el modelo orientado al objeto o las modernas implementaciones basadas en marcos). Las más obvias son las siguientes:

- Imposibilidad de representar conocimiento en forma de reglas.
- Inexistencia de mecanismos de herencia de propiedades (y por supuesto de métodos).
- Falta de poder expresivo (por ejemplo, para representar jerarquías.
- Dificultad para gestionar datos no atómicos (por ejemplo, los valores estructurados de una estructura de rasgos.
- Incompatibilidad entre los tipos de estructuras de datos que se transfieren o inadaptación de impedancia (*impedance mismatch*).

Los cuatro primeros aspectos afectan directamente a la representación léxica, mientras que el último es un problema meramente técnico que no detallaremos y que no presenta el modelo de datos orientado al objeto que hemos mencionado.

Como vimos en el apartado 4.4, los formalismos de representación léxica modernos hacen uso extensivo del concepto de herencia mediante mecanismos provenientes de los esquemas de representación basados en marcos. En este sentido son superiores en poder expresivo a una base de datos relacional, pero sin embargo no ofrecen las facilidades de manejo de datos masivos que una base de datos garantiza. El RDBMS que utilizaremos implementa una función avanzada del modelo relacional: La noción de tipo / subtipo, mediante la cual se puede recrear una jerarquía, aunque no con el poder expresivo de los lenguajes basados en marcos como el que mostraremos a continuación y con el que hemos implementado nuestra ontología. Se trata únicamente de un mecanismo de auto referencia (*self−joint*) mediante el que se puede interrelacionar una relación determinada consigo misma. Utilizaremos este mecanismo para establecer nuestra "jerarquía" de dimensiones y subdimensiones dentro de un campo léxico.

En resumen, un RDBMS supone una plataforma estable y compatible, con limitaciones en sus capacidades y poder expresivo. En este estado de cosas, pensamos que un cuidado diseño (modelado conceptual) puede vencer muchas de estas desventajas y aprovechar al máximo todas las ventajas mencionadas. La evolución del modelo relacional pasa por los modelos semánticos de datos, o de cuarta generación. Estos modelos, influenciados por los sistemas de información de la IA, trataron de dotar de *significado* a las estructuras de datos. En el siguiente capítulo describiremos un modelo de datos semántico, el de Entidad/Relación (Chen 1976) que nos servirá para mostrar el modelado conceptual de nuestra base de datos. Consideramos esta línea de investigación como la verdaderamente revolucionaria en el terreno de las bases de datos, ya que ha permitido el desarrollo de sistemas de representación muy avanzados, entre ellos, y sobre todo, el modelo de orientación al objeto.20

No nos detendremos a analizar estos novedosos modelos aunque resultan ciertamente atractivos, especialmente tras las últimas revisiones de los estándares CORBA y la fijación del método unificado (Booch & Rumbaugh 1995) como estándar de modelado de datos.

En cualquier caso, para entender estos modelos de datos es necesaria una perspectiva de los esquemas de representación típicamente usados para desarrollar bases de conocimiento, porque la influencia de los últimos sobre los primeros es evidente y porque no es posible llegar a entender su alcance sin comprender las técnicas de IA puras de las que provienen. Hemos preferido contemplar el desarrollo de estos modelos de datos dentro del espectro de influencias mutuas entre las dos facetas de la representación de conocimiento que venimos estudiando: las bases de datos y las bases de conocimiento

Ejemplos y aplicaciones de base de datos

- **Pintura de casas Mary Richards, usa una base de datos realizada en ACCESS**

Mary Richards es una pintora profesional de casas: posee y opera una pequeña compañía compuesta por ella misma, otro pintor profesional y, cuando es necesario, pintores contratados por medio tiempo. Mary ha estado en el negocio a lo largo de diez años y se ha ganado una buena reputación como pintora de gran calidad y que trabaja por un precio razonable (ni barato ni excesivo. Consigue gran parte de sus trabajos con clientes que ya la han contratado antes y por referencias personales. Además obtiene algún trabajo por medio de contratistas de edificios y de diseñadores profesionales de interiores.

Los clientes la recuerdan y le hablan para decirle que alguno de sus conocidos les gusto el trabajo que ella les realiza y desean que les haga algo parecido en sus casas, pero como ella pinta mas de 50 casa al año le es difícil recordar el lugar de la casa y a su cliente.

Con el propósito de ayudar a su memoria y mantener una mejor secuencia en sus registros de negocios Mary buscó a un asesor para desarrollarle una base de datos que ella empleara en su computadora personal. La base de datos almacena los registros concernientes a los clientes, trabajos y fuentes en tablas. Como en la fig. 1

El trabajo de un programa denominado sistema de organización de la base de datos es almacenar y obtener los datos para las tablas. Sin embargo, cuando tales datos están en forma de tablas no son muy útiles para Mary. Ella preferiría saber como se relacionan entre sí los clientes, los trabajos y las referencias. Por ejemplo, le gustaría saber cuales trabajos ha hecho para un cliente en particular o cuales han sido recomendados por una persona en particular.

Para resolver su problema el asesor una base de datos que procesa formas de entrada para los datos y produce reportes. Figura 2. Aquí teclea en la forma el nombre de un cliente o su número de calle, entonces la aplicación recupera la información apropiada y la despliega en una forma como la quiere ella.

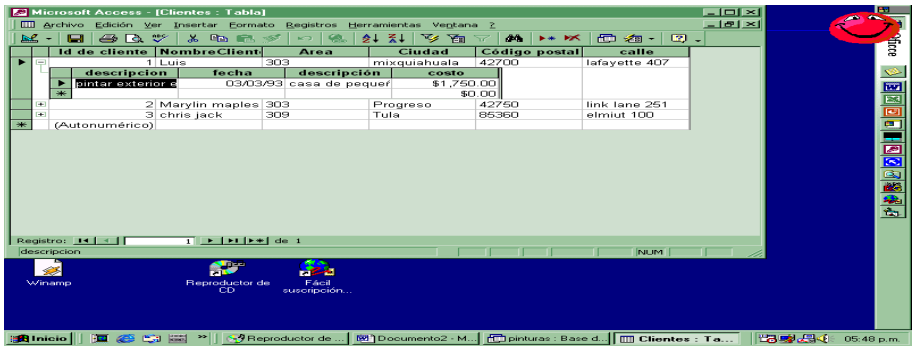

Figura 1

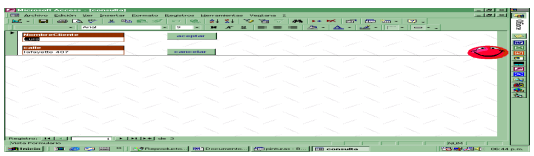

Figura 2

- **Biblioteca escolar de una primaria utiliza una base de datos realizada en visual basic**

El encargado de la biblioteca atendía a muchos usuarios y tenia que registrar los libros que se utilizaban, los prestamos en que lugar se encontraban los libros, revistas dependiendo de su clasificación y algunas personas no sabían utilizar el fichero de información por lo cual tenía que hacerlo personalmente y le restaba tiempo de sus actividades a si que el solicito se le hiciera un programa en el cual pudiera registrar las entradas de los usuarios y salidas, los prestamos y devoluciones de libros, la ubicación de los mismos y tener registrados a todos los afiliados a la biblioteca para acceder a ellos de forma más rápida y saber en que momento renovar las credenciales.

El programa que le realizaron cuenta con un menú principal desde el cual puede acceder a la información en el momento que la necesita *Fig. 3*

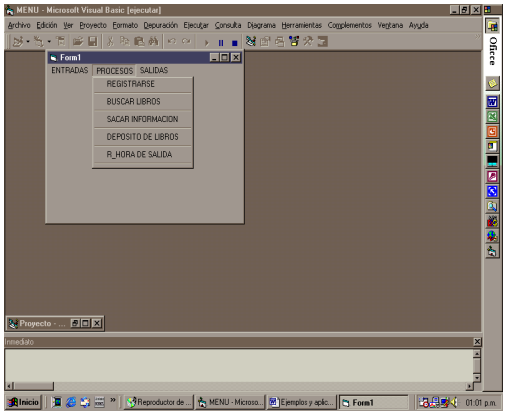

Figura 3

esta base de datos tiene tablas donde se encuentra almacenada la información de la biblioteca y estas están relacionadas con otras para obtener consultas y saber todos los datos de una persona *FIG. 4*

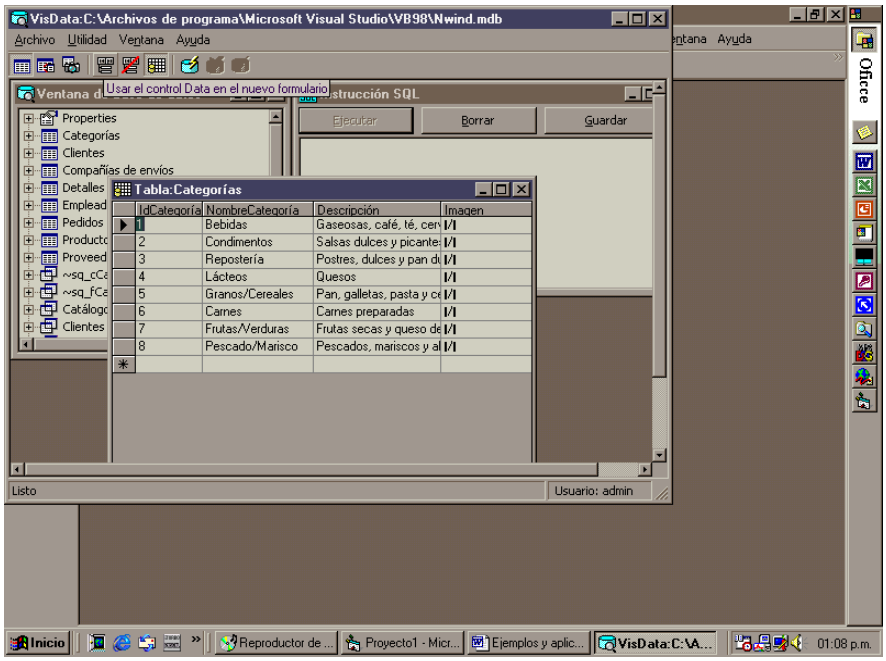

Fig.4

Con este programa creado mediante la aplicación de visual Basic el bibliotecario pudo resolver su problema.

Técnicas de diseño de programas

Elementos básicos de un programa

Las reglas para combinar los elementos básicos de un lenguaje forman la sintaxis del lenguaje. Existen un numero determinado de palabras reservadas, que solo pueden ser utilizadas de un modo limitado: *instrucciones o sentencias de programación.*

Los elementos básicos cuya correcta combinación permite construir un programa son:

- **Palabras clave e identificadores**
- **Constantes**
- **Variables**
- **Expresiones**
- **Sentencias de asignación**

Palabras clave e identificadores

***Las palabras clave o reservadas* constituyen las instrucciones (ordenes, comandos, sentencias, funciones y operadores) intrínsecas al lenguaje de programación y son la parte fundamental de su sintaxis.**

***Los identificadores tienen un significado predefinido (por ejemplo, en pascal abs., ibnput, reset)* o bien ser elegidos por el programador como son los casos de los nombres de variables, programas. Etc.**

Las palabras reservadas no pueden ser elegidas como identificadores o nombres de variables. Son palabras reservadas

BASIC Pascal

Goto, read, input, end... and, array, type, case, begin...

Constantes

Una constantes es una cantidad cuyo valor no cambia durante el proceso, es decir, es un elemento fijo de datos.

Para expresar una constante es preciso escribir su valor, pej. −25, 4. la mayoria de los lenguajes permiten diferentes tipos de constantes, siendo las más comunes: enteros, decimales, caracteres y constantes booleanas o lógicas.

Constante entera

Una constante entera es un número con un valor entero positivo o negativo

5 −124 +12458

Las comas y espacios no se deben utilizar para separar grupos de dígitos en enteros. El número 31.245 se debe escribir 31245.

Constante real

Un decimal o constante real es un número escrito con un punto decimal

31.43 −0.56 32.0

Los números reales se pueden expresar en notación de punto (coma) fijo y en notación de punto flotante.

punto fijo 3.141592 0.000002544 −324.05

Punto flotante. 0.3141592e+1 4.5e−8 −12.5879e2

Constante decimal: e+ n

Donde n es la potencia de diez a la que se tiene que elevar la constante decimal. En realidad esta constante es un tipo particular de las constantes reales

e + 5 equivalente a 105

de punto (coma) fijo y en notación de punto flotante.

es un carácter perteneciente al conjunto de caracteres disponibles, los caracteres son letras mayúsculas y minúsculas, dígitos, símbolos de puntuación y otros símbolos. Las constantes de caracteres se organizan en series o secuencias de caracteres denominadas *cadenas* en pascal se escriben encerrando en un signo de comillas o apostrofes

`A' `B' `HOLA `

En lenguaje basic las cadenas se escriben entre comillas

A B * HOLA

Constante booleana

La constante booleana puede tener dos valores posibles: Verdadero y falso son muy útiles en programación.

Variables

Representan elementos que pueden cambiar durante la ejecución de un programa. Las variables se refieren en los programas por nombres simbólicos o identificadores. Dependiendo del lenguaje, existen diferentes tipos de variables, tales como enteras, reales, caracteres, etc.

Una variables, o mejor un cierto tipo, puede tomar sólo valores de ese tipo. Una variables de carácter, pej. Puede tener como valores sólo caracteres. Cualquier intento de asignar un valor de distinto tipo a la variable producirá un error.

Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación −operadores−, paréntesis y nombres de funciones especiales.

Las expresiones matemáticas tienen igual sentido pej.

X (Y+5) − 4*Z +R

Expresiones aritméticas

Son análogas a las formulas matemáticas. Las variables y constantes implicadas son munericas y las operaciones se expresan por los operadores

	potencia
+ •	**Suma**
–	**Resta**
*	**Multiplicación**
/	**División**
\	**División entera**
MOD	**Modulo de entero (resto)**

Los paréntesis se utilizan para agrupar términos y asegurarse que las operaciones se ejecuten en el orden correcto.

Las operaciones se realizan por orden de prioridad comenzando por la más alta. En caso de igualdad en prioridad, se procesan los operadores en orden de izquierda a derecha. Además de los operadores es posible utilizar funciones matemáticas estándar del sistema o definidas por el usuario.

Expresiones booleanas

Una expresión booleana es donde existen entre otros elementos, operadores de relación o lógicos y su valor es siempre verdadero o falso. Un sistema para generar expresiones booleanas es combinar operadores de este tipo y relacionales con otros elementos.

Se pueden expresar con *operadores de relación o comparación.*

Los operadores de relación se utilizan para comparar expresiones. El formato general es:

Expresión 1* OPERADOR RELACIONAL *expresión 2

Las reglas de prioridad se aplican también en este caso. Todos los operarores de relación tienen menor prioridad que los operadores aritméticos.

Sentencias de asignación

Para ejecutar cálculos se necesitan sentencias que indiquen a la computadora que acciones ha de ejecutar. La herramienta básica es la *sentencia de asignación* son una parte fundamental de casi todos los lenguajes de programación, permiten asignar el valor de una expresión a una variables.

Nos puede tener una expresión en el lado izquierdo. La sentencia de asignación no debe confundirse con una ecuación matemática o la igualdad aritmética. Ejemplo:

A + 5: B − 6

No es un formato correcto, aunque la ecuación matemática

A + 5 = B − 6

Arreglos unidimensionales y bidimensionales

Los arreglos son una colección de variables del mismo tipo que sé referencia utilizando un nombre común. Un arreglo consta de posiciones de memoria contigua. La dirección más baja corresponde al primer elemento y la más alta al último. Un arreglo puede tener una o varias dimensiones. Para acceder a un elemento en particular de un arreglo se usa un índice.

El formato para declarar un arreglo unidimensional es:

tipo nombre_arr [**tamaño**]

Por ejemplo, para declarar un arreglo de enteros llamado *listanum* con diez elementos se hace de la siguiente forma:

int listanum[10];

En C, todos los arreglos usan cero como índice para el primer elemento. Por tanto, el ejemplo anterior declara un arreglo de enteros con diez elementos desde **listanum[0]** hasta **listanum[9]**.

La forma como pueden ser accesados los elementos de un arreglo, es de la siguiente forma:

listanum[2] = 15; /* Asigna 15 al 3er elemento del arreglo listanum*/

num = listanum[2]; /* Asigna el contenido del 3er elemento a la variable num */

Los arreglos bidimensionales también se conocen como multidiemensioanales, en el cual el número de dimensiones (índices) que se deben utilizar en un arreglo depende del problema que debemos resolver y las características del lenguaje que utilicemos.

Un arreglo bidimensional es un conjunto de datos heno, finito y ordenado, donde se hace referencia a cada elemento por medio de dos índices. El primero de los índices utiliza generalmente para indicar renglón, y el segundo indicar columna. Un arreglo bidimensional también para definirse como un arreglo de arreglos.

Declaración de arreglos bidimensionales

Se declaran los arreglos especificando número de renglones y de columnas, junto a cada tipo de componentes, con **líminfr** y **límsupr** se declara el tipo del índice de los renglones y cuantos renglones tendrá el arreglo. Con **líminfr** se declara el tipo de indice de las columnas y cuantas columnas tendrá el arreglo.

El lenguaje C no realiza comprobación de contornos en los arreglos. En el caso de que sobrepase el final durante una operación de asignación, entonces se asignarán valores a otra variable o a un trozo del código, esto es, si se dimensiona un arreglo de tamaño *N*, se puede referenciar el arreglo por encima de *N* sin provocar ningún mensaje de error en tiempo de compilación o ejecución, incluso aunque probablemente se provoque el fallo del programa. Como programador se es responsable de asegurar que todos los arreglos sean lo suficientemente grandes para guardar lo que pondrá en ellos el programa.

C permite arreglos con más de una dimensión, el formato general es:

tipo nombre_arr [**tam1**][**tam2**] ... [**tamN**];

Por ejemplo un arreglo de enteros bidimensionales se escribirá como:

int tabladenums[50][50];

Observar que para declarar cada dimensión lleva sus propios paréntesis cuadrados.

Para acceder los elementos se procede de forma similar al ejemplo del arreglo unidimensional, esto es,

tabladenums[2][3] = 15; /* Asigna 15 al elemento de la 3ª fila y la 4ª columna*/

num = tabladenums[25][16];

A continuación se muestra un ejemplo que asigna al primer elemento de un arreglo bidimensional cero, al siguiente 1, y así sucesivamente.

```
main()

{

int t,i,num[3][4];

for(t=0; t<3; ++t)

for(i=0; i<4; ++i)

num[t][i]=(t*4)+i*1;

for(t=0; t<3; ++t)

{

for(i=0; i<4; ++i)

printf("num[%d][%d]=%d ", t,i,num[t][i]);

printf("\n");

}

}
```

En C se permite la inicialización de arreglos, debiendo seguir el siguiente formato:

tipo nombre_arr[**tam1**][**tam2**] ... [**tamN**] = {lista−valores};

Por ejemplo:

int i[10] = {1,2,3,4,5,6,7,8,9,10};

int num[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};