UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

**PROGRAMA ALTERNATIVO PARA
LA MATERIA DE CIBERNETICA
Y COMPUTACION I.**

T E S I S

QUE PARA OBTENER EL TÍTULO DE

ACTUARIO.

P R E S E N T A :

ALFREDO LOPEZ MORALES

MEXICO, D.F. **1995**

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO.

FACULTAD DE CIENCIAS.

PROGRAMA ALTERNATIVO PARA LA MATERIA DE CIBERNETICA Y COMPUTACION I.

T E S I S

QUE PARA OBTENER EL TITULO DE ACTUARIO.

P R E S E N T A .

ALFREDO LOPEZ MORALES.

ASESOR : M. EN C. ELISA VISO GUROVICH.

PROF. TIT. "A" TC.

UNIVERSIDAD NACIONAL
AVPNºMA DE
MEXICO

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

"PROGRAMA ALTERNATIVO PARA LA MATERIA DE CIBERNETICA Y COMPUTACION I"

realizado por ALFREDO LOPEZ MORALES

con número de cuenta 7322849-9 , pasante de la carrera de ACTUARIO

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis	M. EN C. ELISA VISO GUROVICH
Propietario	DR. JAVIER PAEZ CARDENAS
Propietario	MAT. JULIO CESAR GUEVARA BRAVO
Propietario	M. EN C. VIRGINIA ABRIN BATULE
Suplente	M. EN C. ALEJANDRO RAUL REYES ESPARZA
Suplente	

Consejo Departamental de Matemáticas

DEDICATORIA.

A LA MEMORIA DE MIS PADRES :

REFUGIO Y LUCINA.

GRACIAS A SUS PRINCIPIOS, HE LOGRADO CONDUCIRME POR LA VIDA EN FORMA ADECUADA.

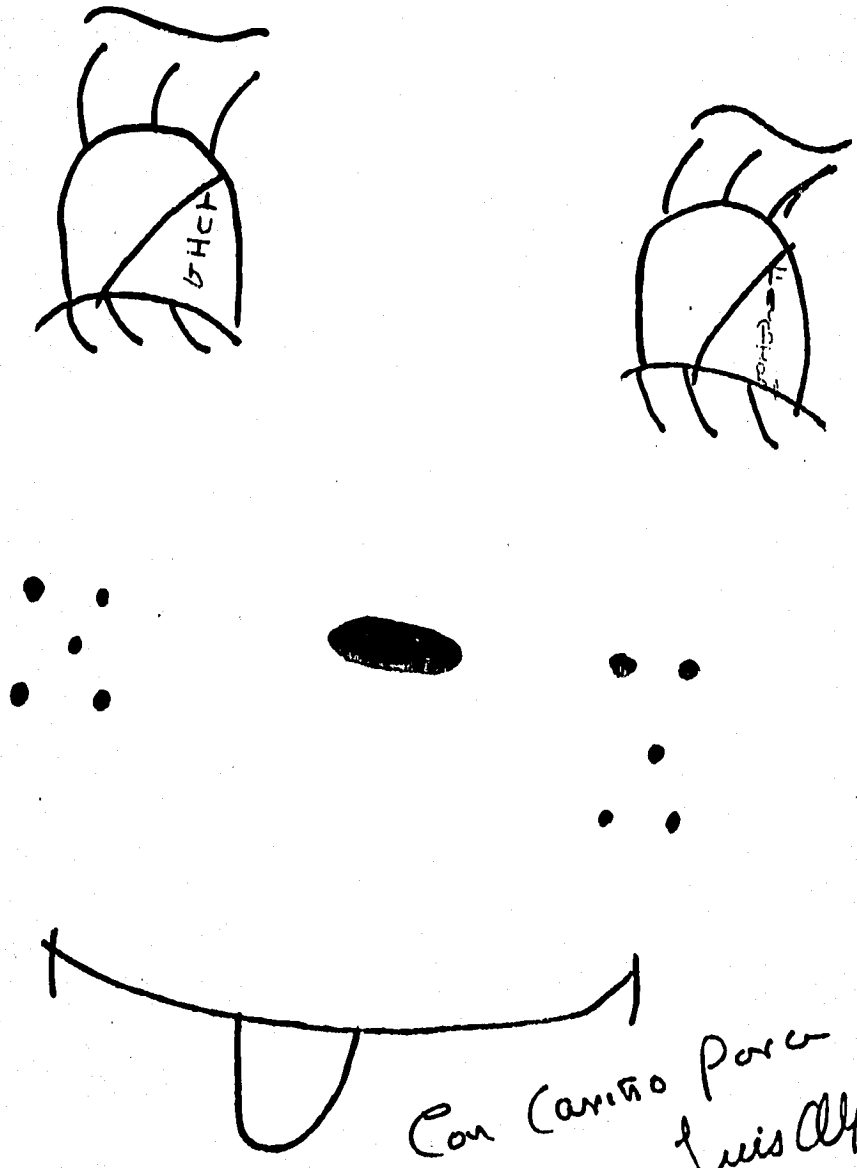
A MIS HIJOS :

FABIOLA Y LUIS ALFREDO.

GRACIAS A SU CARACTER Y CHISPA, HE LOGRADO VENCER UN RETO MAS EN LA VIDA.

A MIS PROFESORES :

GRACIAS A LA PACIENCIA QUE ME TUVIERON AL TRANSMITIRME SUS CONOCIMIENTOS.



Con Cariño para
Fabiola y Luis Alfredo
Alfredo.

INDICE DE CONTENIDO.

INTRODUCCION

En los umbrales del siglo XXI es imposible vivir sin el conocimiento y utilización de la información; la necesitamos para decidir las medidas adecuadas para conducirnos de una mejor forma; la información la requiere la administración pública, la privada, la sociedad civil, los investigadores, los científicos, los conductores de un automóvil, etc.. La información se encuentra por lo general almacenada en bases de datos de alguna computadora; para poderla utilizar es indispensable saber utilizar una computadora, i.e., en este siglo y en el futuro, el que no sepa manejar una computadora está "muerto"; éste es un argumento suficiente para que nuestros alumnos de bachillerato aprendan el uso de la computadora, en un curso introductorio que les dé las bases para su posterior especialización.

Necesitamos enseñarle al alumno el funcionamiento de las computadoras, la forma de comunicarnos con ella, ya sea para resolver un problema, escribir un texto, consultar información etc.. Como vehículo para estos conocimientos consideraremos al lenguaje de programación Pascal; La razón de utilizar Pascal es muy sencilla: fue diseñado para enseñar programación en forma estructurada; esto no quiere decir que tiene limitaciones; al contrario, a pesar de ser sencillo en su concepción y estructura, es un lenguaje muy poderoso y al alumno le facilitará el aprendizaje posterior de programación en cualquier otro lenguaje de computación.

La idea de introducir un procesador de texto es con la finalidad de que el alumno se familiarice con el uso de la computadora, a través de la aplicación más común hoy en día. Al introducir un texto diseñado por él mismo, se dará cuenta que la computadora es un instrumento de trabajo, útil y sencillo de usar.

Estoy convencido que el procesador de texto funciona muy bien como primer contacto del alumno con la computadora, ayudando a que el alumno tome confianza con la misma y le pierda temor.

En el capitulo uno se describe la historia de la computación; en la que se abarca desde el desarrollo de los sistemas de numeración hasta la evolución de las modernas computadoras. En el capitulo dos se describen los componentes físicos de la computadora y se da un ejemplo de su funcionamiento interno. En el tres se mencionan diferentes sistemas operativos y se desarrollan los comandos más comunes del sistema operativo DOS. En el cuatro se describe el editor de Turbo Pascal versión 3.1. En el cinco se desarrolla una metodología para transcribir la solución de un problema a un lenguaje de programación estructurado. En el capitulo seis se dan los elementos para un curso introductorio del lenguaje Pascal en ambiente Turbo versión 3.1.

CAPITULO UNO

HISTORIA DE LA COMPUTACION.

INTRODUCCION.

Para apreciar el estado actual de la computación, así como su futuro, es importante valorar el esfuerzo que ha realizado la humanidad en su desarrollo.

1.1 DE LOS DEDOS A LOS HUESOS DE NAPIER.

La historia del cálculo numérico se remonta al hombre primitivo, quien, necesariamente, utilizó los dedos de las manos y pies para contar; el arte rupestre que se ha encontrado en cuevas en diversas partes del mundo sugiere la utilización de marcas como el segundo paso de los procesos contables; el hombre primitivo, al fin y al cabo nómada, tuvo la necesidad de llevarse dichos registros; para esto, ideó diferentes formas tales como : una cuerda con nudos a intervalos regulares, piedras o guijarros de diferentes tamaños, tablillas de arcilla marcadas de cierta forma, etc.

Los romanos utilizaron una tarja, que era una tabla de madera acanalada por el centro en forma horizontal hacia abajo, para realizar transacciones mercantiles; la tabla era separada por el centro cuando finalizaba la negociación y cada romano tenía un registro de lo pactado.

Contra lo que se cree, fueron los egipcios y los hindúes quienes primero inventaron el ábaco, casi simultáneamente, hacia el siglo X a.C.; Los egipcios eran conocidos por el uso de surcos en la arena para registrar y representar los números con guijarros. Cuando pusieron sus guijarros y sus surcos en una caja inventaron el primer ábaco. Los chinos primeramente utilizaron el Suan Pan, que era una tablilla de cuentas muy parecida a la tarja romana; perfeccionaron el ábaco tal

como lo conocemos hoy, dos mil años después en el siglo XII d.C. El ábaco aún hoy está en uso y es muy fácil de utilizar. Actualmente se realizan competencias entre usuarios del ábaco vis, usuarios de calculadoras modernas, en las cuales al ábaco no le va tan mal.

1.1.1 SISTEMAS NUMERICOS.

Uno de los más grandes pasos que ha dado la humanidad es sin duda la representación del número de objetos por medio de símbolos. La primera representación fue el sistema de número concreto, es decir : representar cuatro manzanas era diferente a cuatro borregos, la cantidad era la misma y los objetos distintos, por tanto tenían diverso significado; un gran paso fue sin duda el proceso de abstracción en el cual un símbolo representa la misma cantidad numérica independientemente de su valor apreciativo. Se puede clasificar a los sistemas numéricos en dos tipos : de tarjas, en el que el mismo símbolo era usado una y otra vez, y el sistema de código, en el que cada número tiene asignado diferente símbolo (ver tabla 1.1). Algunos sistemas están basados en la notación posicional. Otra característica importante de los sistemas numéricos es la base, o número de dígitos, que utilizan (ver tabla 1.2).

El concepto de notación posicional, en la representación de cantidades por medio de símbolos, es de tal importancia que voy a establecer una comparación de un sistema numérico con notación posicional y el sistema de notación romano .

CODIGO	TARJA	CODIGO
1	I	A
2	II	B
3	III	C

TABLA 1.1 SISTEMA DE TARJA Y DE CODIGO.

PUEBLO	BASE NUMERICA	TIPO DE SISTEMA
CHINO	2	TARJA
MESOPOTAMICO	3	
EGIPCIO	5	CODIGO
MAYA	20	TARJA
BABILONIO	60	TARJA

TABLA 1.2 SISTEMA NUMERICO Y TIPO DE CODIGO.

Los romanos simbolizaron los números por medio de letras mayúsculas : I,V,X,L,C,D y M, los cuales representan a los números 1,5,10,50,100,500 y 1000, respectivamente, en sistema decimal; la formación de los restantes números se basa en la aplicación de un conjunto de reglas, que se listan a continuación.

1.1.2.- REGLAS DE LA NOTACION ROMANA.

1.- Pueden repetirse los simbolos I,X,C y M, hasta tres veces; al repetirlas se repite su valor. Ejemplo : II = 2, XXX = 30, MMM = 3000.

2.- Todo símbolo colocado a la derecha de otro mayor, agrega su valor al de ese símbolo mayor. Ejemplo : VI = 6, LX = 60, MD = 1500.

3.- Todo símbolo colocado a la izquierda de otro mayor resta su valor al de ese símbolo mayor, como sigue : el I antes de V y de X; el X antes de L y C; el C antes de D y M. Ejemplo : IV = 4, XL = 40, CM = 900. Todo símbolo colocado entre dos mayores que él, resta su valor al que está a su derecha. Ejemplo : XIV = 14, CXL = 140.

4.- No se repiten los símbolos : V, L y D; tampoco se escriben a la izquierda de otro símbolo mayor. Ejemplo : VV <> 10, VX <> 5, LD <> 450, (Donde <> significa diferente).

5.- Una barra colocada sobre un símbolo, incrementa su valor por mil.

1.1.3.- SISTEMA DE NOTACION POSICIONAL BASE DOS.

Para establecer las diferencias entre el sistema romano y uno de notación posicional, voy a desarrollar el sistema base dos, el cual se va a utilizar al describir el funcionamiento interno de la computadora.

En los últimos años del siglo XIX, le correspondió a la dama inglesa Lady Lovelace dar el siguiente paso en el desarrollo de los sistemas numéricos; diseñó el sistema con valor de posición base dos que utiliza los símbolos cero y uno; con este sistema se puede representar cualquier número de base decimal.

El sistema base dos utiliza los símbolos 0 y 1; para diferenciarlos del sistema base 10 los denotaremos como 0_2 y 1_2; que representan al 0 y al 1 en base decimal respectivamente; para representar los otros números vamos a proceder como sigue : ya que no existen más símbolos disponibles, se hace necesario utilizar uno de los símbolos otra vez en una nueva posición, en la cual el símbolo representará un valor más grande. Cada posición asocia un "peso" al dígito. En base 10, por ejemplo, en cada posición podemos colocar uno de 10 símbolos (los dígitos del 0 al 9). Cuando decimos que el sistema decimal es posicional lo que queremos decir es que el peso del dígito depende de su posición. Numeramos a las posiciones de derecha a izquierda, empezando del cero; el peso de la posición 0 es 1. El peso de la posición 1 es 10; el peso de la posición 2 es 100; y así sucesivamente. Cada vez que nos movemos a la izquierda una posición el peso de esa nueva posición es el de la posición de la derecha multiplicada por 10, que es la base. En el caso del sistema base 2, tenemos el mismo concepto, nada más que el peso de cada posición se va multiplicando por 2. Por esa razón, para representar el 2 escribimos 10_2 , es decir, un grupo de dos y ningún grupo de uno. De la misma forma representamos el 3 por 11_2 , esto es una posición de dos y una de uno. Para representar el 4 utilizamos un símbolo en una nueva posición y escribimos 100_2, lo que significa: un grupo de cuatro, ningún grupo de dos y ningún grupo de uno y así sucesivamente. Con esta técnica podemos representar cualquier número (ver tabla 1.3).

La notación romana es un sistema que carece de un orden metodológico para su construcción; está basado en un conjunto de reglas; para aquellos acostumbrados a la notación posicional les parece engañosa, pues un número de menor cantidad numérica utiliza más símbolos que un número mayor; por ejemplo el número XXXVI aparenta ser mayor que C, dado que aquél tiene más símbolos para su representación. Y por otro lado, es un sistema de notación no posicional dado que el lugar que guardan los símbolos no denotan un valor por sí solos; el sentido de cantidad lo tiene toda la expresión . Por tanto el sistema romano es una técnica apta para el registro de, por ejemplo, los capítulos de los libros, los volúmenes de las enciclopedias, años, y en general, de cantidades pequeñas y "distinguidas".

Es importante resaltar que dicha notación romana era necesaria y útil para los romanos; resolvía sus problemas y sirvió para establecer los sistemas de numeración actuales.

En los sistemas de notación posicional, un número con más símbolos representa una cantidad mayor que uno con menos símbolos; por ejemplo : el número 110101_2 es mayor que 1101_2 . Siempre es posible representar un número con la técnica expuesta anteriormente; y siempre se pueden representar en otros sistemas numéricos de notación posicional. Dada la importancia de la notación posicional y a riesgo de ser repetitivos, insistimos en que, en este tipo de sistemas, consideramos las posiciones de derecha a izquierda; de tal forma que el primer dígito de derecha a izquierda se encuentra en la posición cero, el segundo en la uno, ..., el i-ésimo en la posición i-1. En notación posicional el valor de cada dígito depende de la posición que ocupa; para determinar el valor de los dígitos procedemos como sigue :

sea p la posición del digito.

b la base en la que se encuentra el número y

d el dígito.

la expresión : db nos da el valor del dígito d, en la base b en la p-ésima posición, por ejemplo : en el número 1.101001_2 el valor de los dígitos se encuentran expresados en la tabla 1.4.

DECIMAL	BASE DOS	SIGNIFICADO
0	0_2	NINGUN GRUPO DE UNO.
1	1_2	UN GRUPO DE UNO.
2	10_2	UN GRUPO DE DOS, NINGUN GRUPO DE UNO.
3	11_2	UN GRUPO DE DOS, UN GRUPO DE UNO.
4	100_2	UN GRUPO DE CUATRO, NINGUN GRUPO DE DOS NI DE UNO.

TABLA 1.3 : REPRESENTACION DE LOS NUMEROS BINARIOS.

DIGITO	POSICION	VALOR
1	0	$1(2^0) = 1$
1	1	$1(2^1) = 2$
0	2	$0(2^2) = 0$
1	3	$1(2^3) = 8$
0	4	$0(2^4) = 0$
1	5	$1(2^5) = 32$
1	6	$1(2^6) = 64$
1	7	$1(2^7) = 128$

TABLA 1.4 : VALOR DE LOS DIGITOS DEL NUMERO : 11101001_2

1.1.4. SISTEMAS ANTIGUOS DE NUMERACION.

El sistema numérico chino (sistema de código) tenía originalmente el dos como base, pero desapareció hace mucho tiempo. Los chinos diferenciaban los valores acompañados de un sufijo para representar las decenas, centenas y los millares.

El sistema egipcio (sistema de código) tiene significado tanto numérico como cultural : la vara de trigo representa los dígitos del 1 al 9 ; el talón de un pie el número 10; la cuerda de cáñamo enrollada, de valiosa utilidad, el 100;...y un hombre en postura de admiración representa un millón ¿ Quién no ?

La base babilónica (sistema de tarjas), 60, es la de mayor base conocida por la humanidad; por extraño que parezca hoy en dia, la utilizamos para medir las horas y los ángulos. También recibe el nombre de cuneiforme, del latín cuneus, que significa cuña, dado que los símbolos que utilizaban era en forma de cuña. Tal vez su aportación más importante es que su sistema es posicional e implícitamente utilizaron un símbolo parecido al cero.

Los mayas (sistema de tarjas) sabían que un buen sistema necesita un cero. Su sistema fue vigesimal y utilizaron el valor de posición.

Del sistema hindú (sistema de valor de la posición), sus símbolos son los que usamos hoy. Los moros, que conocían el mundo arábigo y el hindú, introdujeron el sistema numérico en sus grandes universidades. En los últimos años del siglo X, Gerbert de Aurillac (el Papa Silvestre II) deseaba estudiar en las grandes universidades árabes de Córdoba y Sevilla, en las cuales no permitían el ingreso a católicos; Gerbert declaró que no quería pertenecer al cristianismo, por lo que le fue permitido entrar; vivió entre los árabes, estudió y se graduó en la universidad y

posteriormente retornó al mundo cristiano, aportando así el sistema numérico indo-arábigo.

El sistema hindú era un sistema de código que tenía incorporado el concepto de valor de posición y la idea de cero. La ventaja de un sistema numérico que contiene el valor de posición es que proporciona un método para representar la magnitud de un número dado, sin importar qué tan grande sea.

Desde la invención del ábaco hasta principios del siglo XVII, no se desarrolló ningún aparato para el cálculo numérico. Le correspondió al escocés John Napier desarrollar la teoría de los logaritmos; y los números resultantes fueron grabados en barras blancas a fin de facilitar su manipulación, especialmente la multiplicación y la división. Estas barras se conocieron como los huesos de Napier.

1.2 MAQUINAS DE CALCULO AUTOMATICO.

El siglo XVII se caracterizó por la consolidación de la burguesía con sus sistemas de producción y comercialización, así como por el desarrollo de la astronomía, la construcción de cartas de navegación. Asimismo tomaban gran importancia los primeros bancos comerciales y se empezaban a recaudar los impuestos en forma sistemática. Existía la necesidad de realizar cálculos más exactos y con mayor rapidez; se contaba con la tecnología de los precisos relojes por medio de engranes y los rodillos dentados de las cajas musicales.

Ante este panorama de necesidades la historia registra la contribución de tres personajes : Wilhelm Schickard, Blaise Pascal y Goltfried Wilhelm Von Leibnitz, los cuales desarrollaron mecanismos de cálculo tanto automático como semiautomático.

La información que se tiene acerca de Schickard, de nacionalidad alemana, data de 1620-24, a partir de la correspondencia que mantuvo con el científico J. Kepler. La máquina realizaba las operaciones de suma y resta en forma automática, y las de multiplicación y división en forma semiautomática; su sistema mecánico consistía en engranes para el acumulador, incluía un mecanismo para introducir números en forma manual, lo que permitía efectuar el producto con mayor rapidez; el proyecto incluía desarrollar un sistema de rodillos grabados con las multiplicaciones de Napier. Desgraciadamente no se pudo concluir pues un incendio terminó con la máquina y una epidemia con él y su familia.

La mayoría de los historiadores le dan el mérito a Blaise Pascal, francés, de ser la primera persona en inventar una máquina de cálculo automática en el año de 1640; el proyecto de Pascal partió de la necesidad que tenía su padre de realizar operaciones. Pascal desarrolló un sistema de rodillos por medio de engranes con el cual realizaba las operaciones de suma y resta. El cuentakilómetros de los automóviles utilizan el principio ideado por Pascal en su máquina de cálculo.

El trabajo del alemán Leibnitz se centró en facilitarle los cálculos a los científicos de la época; el sistema de cálculo automático diseñado por Leibnitz realizaba las operaciones de multiplicación y división en forma automática, y de esta forma facilitaba los cálculos de los logaritmos neperianos. La contribución de Leibnitz al desarrollo de las máquinas de cálculo automático fue que podía introducir un dígito de la multiplicación en forma anticipada; con una vuelta a la manivela, ese número podía sumarse al número existente en el acumulador.

El trabajo desarrollado por Schickard, Pascal y Leibnitz demostró que la aritmética se podía automatizar.

1.3 DE LA TARJETA PERFORADA AL CHIP.

1.3.1. EL MODELO DE C. BABBAGE.

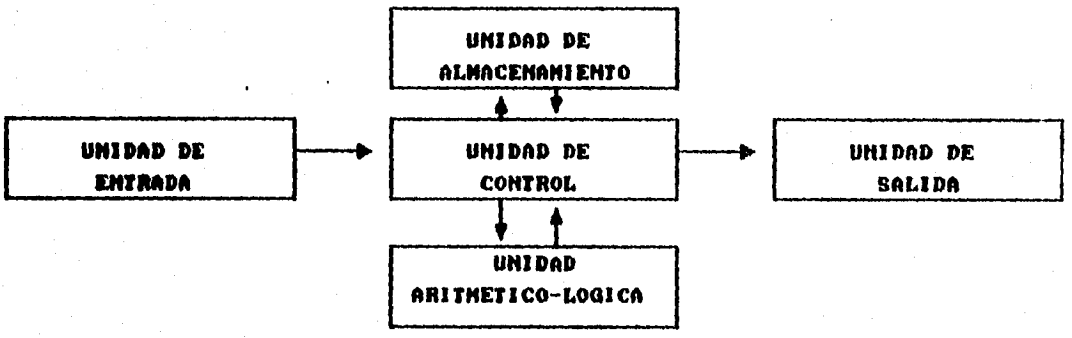
El siguiente gran paso que dio la humanidad en el desarrollo do sistemas de cálculo automático se remonta al siglo XIX; Ya existia la idea de diseñar mecanismos de control de procesos de tal forma que un sistema de producción se redujera a una serie de pasos realizados en orden y de manera rutinaria. Le correspondió al francés Joseph M. Jacquard diseñar un sistema de tarjetas perforadas para el control de los sofisticados dibujos de los telares.

En 1832 el inglés Charles Babbage REVOLUCIONO el mundo de los procesos de cálculo automático, no por el diseño de sus dispositivos de cálculo mecánico en si, pero si por las ideas y los conceptos implementados en ellos, los cuales se adelantaron a toda la ciencia y tecnología de su época; de tal forma que después de la muerte su mérito fue reconocido en un informe, de 1879, del Comité Merryfield, el cual señalaba : "... aparte de la cuestión del posible ahorro de trabajo en las operaciones actuales... la existencia de tal instrumento colocaría dentro de nuestro alcance mucho que, si realmente no es imposible, está demasiado cerca de los límites de la capacidad y paciencia humanas como para poder ser llevados a la práctica.".

Nada más la primera parte de su proyecto se concluyó; la máquina de diferencias era capaz de generar tablas logaritmicas y astronómicas de seis posiciones. La técnica que utilizó fue el cálculo de diferencias sucesivas; de ahí el nombre de la máquina. La segunda fase consistía en el diseño de la máquina llamada analítica accionada con vapor; el control del funcionamiento operaba con tarjetas perforadas como las de Jacquard, contaba con secciones de : almacenamiento de datos, de cálculo aritmético, de entrada de datos y una de salida; Babbage ideó un instrumento de cálculo programable, es decir, una máquina de proceso automático que realizaba una serie de operaciones aritméticas y de decisión sin intervención humana.

Nada más y nada menos que el esquema básico de las modernas computadoras electrónicas, (esquema 1.5).

A Charles Babbage se le llama padre del cálculo electrónico digital moderno. Una gran impulsora de las ideas de Babbage fue Lady Ada Lovelace, matemática inglesa que desarrolló en esa época el sistema binario. ¿ Coincidencia ?



ESQUEMA 1.5 : ESQUEMA DE LA MAQUINA ANALITICA IDEADA POR C. BABBAGE.

Hollerith patentó en 1884 su máquina tabuladora de "conteo rápido", la que se utilizó en los censos de los U.S.A. de 1890; la máquina de Hollerith trabajaba por medio de tarjetas perforadas de la siguiente manera : tenía una tarjeta matriz

perforada con el significado de todos los resultados posibles; el perforador utilizaba un instrumento parecido al pantógrafo; señalaba la zona de la tarjeta que correspondía al dato y la perforaba; con este método se perforaron 56 millones de tarjetas. El procedimiento para cuantificar los datos consistía en introducir cada tarjeta en una lectora que por medio de un sistema de pasadores recorría la tarjeta desde la parte superior buscando una perforación para hacer contacto con un recipiente de mercurio, cerrando así un circuito eléctrico, lo que originaba un avance en el conteo; entonces la ranura correcta se abría, de tal forma que la información podía clasificarse automáticamente para la estadística.

El diseño de Hollerith es mencionado por su sistema de tarjetas perforadas, mas no por la forma de contabilizar los datos a partir de un circuito eléctrico. Si consideramos que las computadoras funcionan con impulsos eléctricos, se tiene que revalorizar su diseño; en 1896 fundó la Tabulating Machine Company, la cual vendió en 1911. En 1924 esta compañía se fusionó para formar la International Business Machines Corporation (IBM).

En el siglo XX las ideas de Babbage se concretaron con la invención de dispositivos de cálculo automático electromecánicas y electrónicas desarrollados siguiendo el esquema básico de Babbage. Dada la gran potencialidad de estos sistemas y los periodos de guerra, se mantuvieron como secretos de Estado y pocas personas las conocieron.

En 1925, V. Bush, en el Instituto Tecnológico de Massachusetts (MIT), construyó una máquina analógica en que los números se representaban mediante un fenómeno físico tal como una variación de voltaje. En 1930, L. J. Comrie en Inglaterra y W. J. Eckert en los Estados Unidos (USA) utilizaron equipo de procedimiento comercial de datos para calcular datos astronómicos; G.R. Stibitz, en USA, diseñó

una máquina semiautomática del tipo de relevadores, con una máquina de teletipo conocida como la "Computadora Compleja". En 1930, el alemán Konrad Zuse trabajó en el área de computadoras programables de uso general; poco se conoce del diseño de su sistema, aunque se sabe que es el primer proyecto que utiliza el sistema binario. En la U. de Harvard se desarrolló la llamada Mark I, capaz de resolver ecuaciones diferenciales. Diseñada por Howard Aiken, construida por la IBM, era un instrumento de cálculo electromecánico que incluía los componentes del esquema de Babbage, exceptuando la toma de decisiones, y trabajaba en sistema decimal; posteriormente se modificó para que incluyera tubos electrónicos y otros refinamientos, pero no tenía programas almacenados y se operaba casi completamente mediante interruptores mecánicos.

En 1945, J. Mauchly y J.P. Eckert diseñaron la primera computadora digital electrónica llamada la ENIAC en la U. de Pennsylvania, con fines militares; era una máquina gigantesca, con 18 mil tubos electrónicos (bulbos) y 6 mil interruptores; toda su aritmética se efectuaba mediante impulsos electrónicos; con una velocidad 15 mil veces más rápida que la MARK I; las órdenes se daban por medio de contactos eléctricos; carecía de programas almacenados; pero antes de su terminación se concibió la idea de programa almacenado que adoptaron los ingenieros de Cambridge y Manchester en Inglaterra.

J. Von Newmann estableció en 1945-1947 en la U. de Princeton la base teórica de los ordenadores modernos. El y su equipo de trabajo desarrollaron una computadora con programas almacenados en memoria y desarrolló los conceptos de diagramas de flujo, la separación de funciones de computadora en unidades y el uso de lenguaje máquina. La primera computadora que incorpora el concepto de programa almacenado fue la EDSAC, desarrollada en la U. de Manchester. Era más

pequeña que la ENIAC, pero seis veces más rápida. Las órdenes se leían de una cinta de papel perforada con el código apropiado.

1.3.2. GENERACIONES DE COMPUTADORAS.

El desarrollo de los dispositivos de cálculo electrónico se da a grandes pasos; día con día se diseñan equipos con procesadores más potentes, memorias de mayor capacidad de almacenamiento, sistemas operativos más eficientes, presentaciones de menor tamaño.

La primera generación, de 1946-1952, se caracterizó por la computadora de tubos de vacío (bulbos), de gran tamaño; en comparación con los modelos actuales, eran lentas y embarazosas; utilizaban demasiada energía eléctrica, por tanto generaban gran cantidad de calor, por lo que necesitaban espacios demasiado grandes habilitados con aire acondicionado.

La segunda generación, que va desde 1952 a principios de los años sesenta, marca la miniaturización de las computadoras, un incremento altamente significativo en la confiabilidad de las operaciones, cálculos más rápidos, y una mayor memoria; gracias al invento del transistor y su uso en las computadoras, desarrollado por el laboratorio de la compañía telefónica Bell (En U.S.A.). El transistor se utilizó en lugar de los bulbos.

Los tubos al vacío tenían fallas frecuentes, por lo que no se le tenían confianza a los resultados obtenidos por las computadoras. Por ejemplo, para tener una confianza de 99.9 % de que el resultado sea válido, se necesita que cada componente de la computadora (los bulbos) realice 10^{16} operaciones sin cometer error, esto es, lo que equivale a que una mecanógrafa no cometa un error en un

millón de años de trabajo continuo. Con la utilización de los transistores en lugar de los bulbos se evitaron las fallas en los resultados por problemas de hardware.

La tercera generación de computadoras se registra, a finales de los años sesenta; con el empleo de los circuitos integrados (CI). Los CI se encuentran constituidos por circuitos eléctricos completos, impresos en una placa pequeña llamada chip. Se implementaron los lenguajes de programación : Algol, Fortran y Cobol, entre otros, para uso científico técnico y comercial respectivamente.

La cuarta, quinta, ... generación de computadoras marca un alto grado de sofisticación electrónica, que influye en la construcción de supercomputadoras para el desarrollo científico, computadoras de todos los tamaños, con el desarrollo de lenguajes de programación para todo tipo de aplicación, sean éstas educativas, mercantiles, recreativas, tecnológicas, etc..

CAPITULO DOS

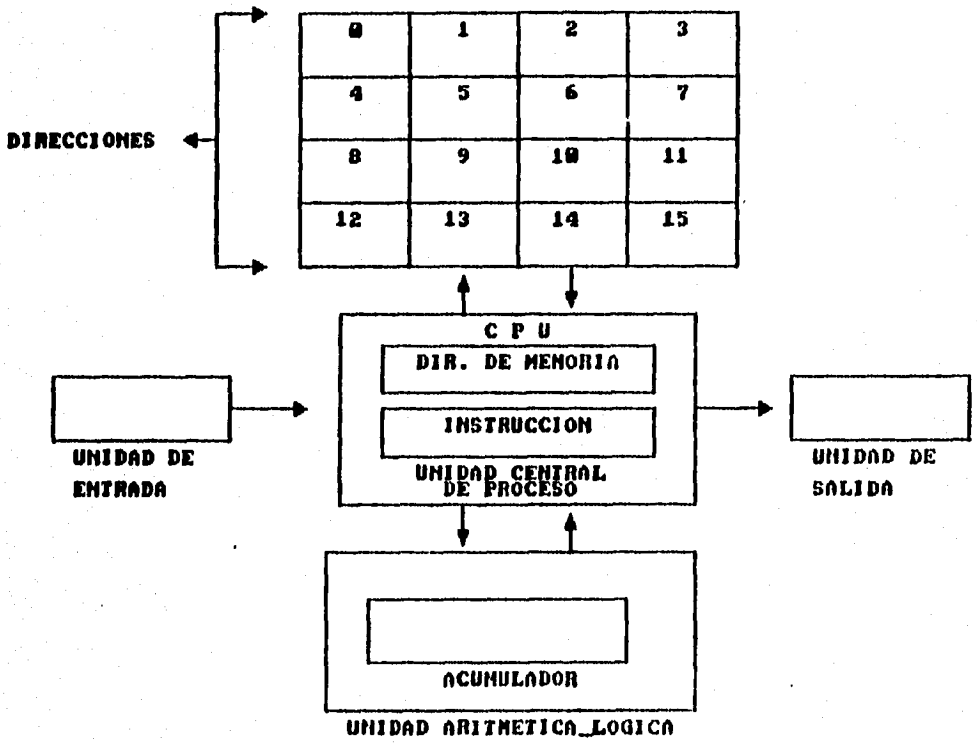
ESTRUCTURA Y FUNCIONAMIENTO DE LA COMOUTADORA.

INTRODUCCION.

En los capitulos 3,4 y 6 aprenderemos a comunicarnos con las computadoras y a interactuar con ellas. Con este aprendizaje le perderemos el "temor" a las computadoras; pero nos quedará la duda de cómo trabajan esas "cajas negras". En esta unidad aprenderemos los conceptos del funcionamiento interno de la computadora, asi como sus componentes y la forma en que interactúan entre si, para realizar las operaciones que se le indican.

2.1.- CARACTERISTICAS DE LAS COMPUTADORAS

Una computadora es una máquina de origen electrónico con una o más unidades de proceso y equipos periféricos controlados por programas almacenados en su memoria, que pueden realizar una gran variedad de trabajos. Todas las computadoras digitales son básicamente dispositivos que pueden transmitir, almacenar y manipular información (datos). Una computadora puede procesar distintos tipos de datos; éstos pueden ser datos numéricos, alfanuméricos (nombres, direcciones, etc.), datos gráficos (mapas, dibujos fotografías, etc.) y sonido (música, lectura de textos, etc.). A todos los componentes físicos que constituyen la computadora se les llama HARDWARE. Al ser una máquina programable debe contar con información que le indique de qué forma utilizar sus unidades físicas para llevar a cabo el trabajo. Esta información es lo que denominamos soporte lógico o SOFTWARE, es decir, al conjunto de programas que hacen uso y/o gestionan las partes físicas de la computadora se les denomina software.,



ESQUEMA 2.1 : COMPONENTES DE UNA COMPUTADORA.

Los sistemas operativos son elementos constitutivos de software, pero no todo software es un sistema operativo; existen programas diseñados para todo tipo de aplicación, que se sirven del sistema operativo, para poder ser ejecutados.

2.2.- COMPONENTES FISICOS DE UNA COMPUTADORA.

la parte física de la computadora, el hardware, está constituida de una unidad central, de la que forman parte la memoria, la unidad aritmética-lógica y la unidad de control (conocida como CPU), así como unidades periféricas, que toman o expulsan los datos hacia o desde la unidad central a través de unos soportes de los mismos (ver esquema 2.I). Cada una de ellas tiene una tarea específica dentro del funcionamiento de la computadora.

La unidad central de proceso (CPU por sus siglas en inglés) es la encargada de organizar y coordinar el trabajo de las otras unidades. La podemos considerar como el "cerebro" de la computadora; la unidad aritmética-lógica efectúa cálculos aritméticos, toma decisiones lógicas, etc.; la memoria permite almacenar datos y programas en direcciones bien identificadas; por medio de la unidad de entrada se le envían datos a la computadora; la unidad de salida posibilita a la computadora enviarnos informes de datos de entrada y/o procesados.

2.2.1.- EL DISPOSITIVO DE ENTRADA.

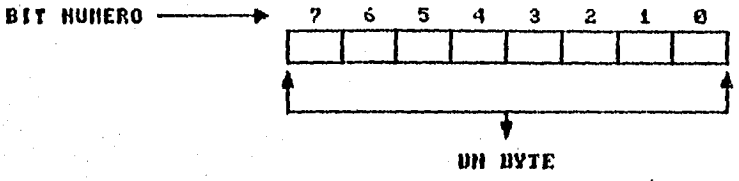
Cuando a un estudiante le piden que resuelva un ejercicio de matemáticas, le informan las condiciones iniciales asociadas con ese ejercicio; es decir, le están dando información de "entrada". El dispositivo que emplea para recibir esa información pueden ser los ojos, los oídos. La computadora también necesita que se le dé información, la cual se introduce por un dispositivo de entrada. Este dispositivo puede

ser un teclado, un lector óptico (como el utilizado en los supermercados), un scanner, un lápiz óptico, y cualquier otro medio para introducir datos a la computadora.

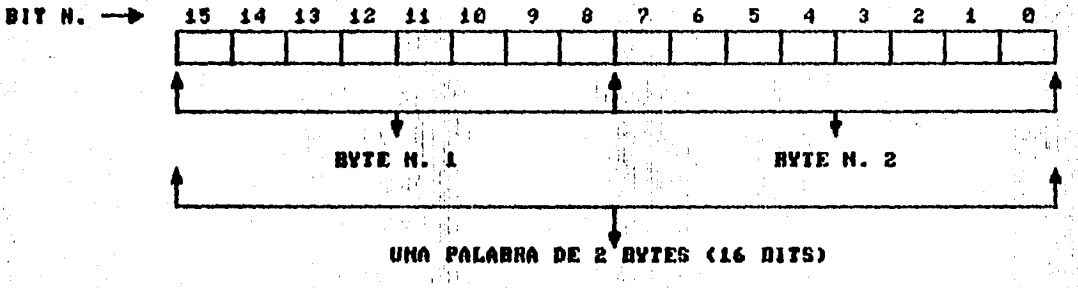
2.2.2.- LA UNIDAD DE MEMORIA

Este dispositivo cumple una misión primordial en la computadora : almacena datos e instrucciones. La memoria de la computadora es como un conjunto de casilleros y cada uno se encuentra numerado en una forma adecuada que permite su fácil localización; comenzando con el número 0 (cero) hasta el último casillero; el tamaño de la memoria lo determina el número del último casillero; para llamarlo con lenguaje más apropiado al número de cada casillero lo llamaremos dirección (ver esquema 2.1). En cada dirección de la memoria se almacenan datos e instrucciones, los cuales son codificados como una combinación única de ceros y unos. Estos ceros y unos se llaman bits, que es una abreviación de las palabras en inglés : binary digit (dígito binario). Un dispositivo electrónico representa cada bit, el cual se encuentra en el estado "apagado" (cero) o "encendido" (uno).

Las computadoras pequeñas tienen organizada la memoria en conjuntos de 8 bits, llamados Bytes, los bits se numeran de derecha a izquierda (en notación posicional), comenzando con la posición 0 (cero) más a la derecha y terminando con 7 (el bit más a la izquierda) (ver esquema 2.2). Por lo general en un byte de memoria se almacena un carácter (por ejemplo : una letra, un dígito, o un símbolo de puntuación), entonces una instrucción consta de más de un byte.



ESQUEMA 2.2 : UN BYTE FORMADO POR 8 BITS.



ESQUEMA 2.3 : PALABRA DEL PROCESADOR 80286.

LES	ALGORITMO
LEE 12 LEE 13	1.- ENTRADA DE LOS VALORES DE LA LONGITUD Y ANCHURA.
CAR 12 MUL 13 ALM 14	2.- CALCULO DEL AREA = LONGITUD * ANCHURA.
IMP 12 IMP 13 IMP 14	3.- SALIDA DE LOS VALORES DE LONGITUD, ANCHURA Y AREA.
FIN	4.- FINAL.

TABLA 2.6 : EL PROGRAMA LES Y EL ALGORITMO DE SOLUCION.

Las computadoras grandes tienen organizada la memoria en grupos de bytes llamadas palabras, dependiendo del procesador (CPU)* , una palabra puede estar constituida por 2,4,8 bytes, por ejemplo : en el procesador 80286, una palabra la constituyen dos bytes, es decir 16 bits (ver esquema 2.3); los bytes dentro de una palabra están numerados de la misma forma que los bits; el byte do la derecha es el byte 0 (cero o bajo) y el de la izquierda es el byte 1 (ó alto).

El tamaño de la memoria de una computadora se expresa como un múltiplo de 2^{10} = 1024 byte, esto es, 1k (1 kilo byte); las computadoras pequeñas tienen memorias de tamaño comprendido entre 64 k y varios megabytes, donde 1 megabyte equivale a $2^{10} * 2^{10}$ bytes = 1024 k bytes; por ejemplo : si una computadora tiene 256 k bytes de memoria, entonces se puede almacenar en su memoria 256 * 1024 = 262,144 caracteres y/o instrucciones.

2.2.2.1 TIPOS DE MEMORIA

En una computadora personal podemos clasificar tres tipos de memoria básicamente : ROM, RAM y auxiliar; cada una con características propias; podemos resaltar las más importantes :

i) MEMORIA ROM :

ROM es abreviación de las palabras en inglés Read Only Memory, esto es, memoria solamente para lectura; como su nombre lo indica en esta memoria sólo se puede leer y no se puede escribir en ella. La memoria ROM se reserva para ciertos programas que son esenciales para la operación de la computadora, los cuales son activados automáticamente al encender la máquina. Estos programas se graban en

---

* Ver la sección la UNIDAD CENTRAL DE PROCESO.

la memoria ROM durante el proceso de su fabricación y el usuario no puede modificarlos.

ii) MEMORIA RAM :

La memoria RAM, por su abreviación de las palabras en inglés : Random Access Memory, es memoria de acceso aleatorio. La computadora lee y escribe sobre ella; los programas que "corre" un usuario se "cargan" en la memoria RAM, así como los datos que se introducen y/o se modifican son grabados en forma momentánea en esta memoria. Sin embargo, apenas se apaga la computadora, toda la información en RAM se borra, y por tanto no se puede almacenar información permanentemente, lo que explica la necesidad de utilizar tanto discos flexiblos como los fijos para almacenamiento de programas y datos. La razón principal de utilizar la memoria RAM es su gran rapidez de acceso (toma aproximadamente una millonésima de segundo almacenar o recuperar un carácter en RAM).

Como se mencionó en la sección anterior, la capacidad de la memoria se mide en k bytes; muchas de las PC modernas tienen al menos 256 k bytes de memoria RAM. Los programas actuales requieren de varios mega-bytes en memoria RAM para su funcionamiento; aunque sólo 64 k bytes de esta memoria expandida pueden estar activos a la vez.

iii) MEMORIA DE DISCO AUXILIAR :

A los discos "flexibles" y duros se les llama memoria auxiliar, dado que son dispositivos adicionales a la configuración básica de la computadora; además, permite que programas y datos se almacenen permanentemente en ellos, es decir, si se apaga la computadora los datos grabados en la memoria auxiliar se mantienen. Los discos "flexibles" pueden almacenar de 360 k bytes a 1.4 mega-bytes, dependiendo de

la densidad de grabación. Los discos duros o fijos almacenan de 10 a mil mega-bytes; permiten un acceso mucho más rápido que los flexibles, pero menor a la memoria RAM.

Los discos están organizados en círculos concéntricos llamados pistas; cada una se encuentra dividida en sectores. Para grabar información en los discos es necesario "imprimirle" marcas magnéticas, esto se logra al formatear el disco. La pista que se encuentra más al interior contiene información sobre los datos del disco : El directorio de los archivos incluidos, la hora y la fecha de su última modificación; la localización del primer sector y una tabla de asignación de archivos (FAT), la que registra que sectores se encuentran en uso y a que archivo corresponden.

2.2.3.- LA UNIDAD CENTRAL DE PROCESO (CPU).

Los procesos que permiten a las personas resolver un problema son "controlados" por una parte del cerebro. La computadora tiene una unidad de proceso central (CPU), que controla el procedimiento completo y coordina la interacción de los componentes de la computadora que trabajan simultáneamente. El CPU lee y/o realiza las instrucciones almacenadas en la memoria; las instrucciones se ejecutan una a una y se activan las diferentes unidades de la computadora según sea la instrucción, por ejemplo : si la instrucción es hacer un cálculo, se activa la unidad aritmética-lógica. En una computadora, la CPU es un microcircuito integrado de semiconductores, cuya longitud es de aproximadamente una pulgada. El primer circuito integrado (CI) fue desarrollado en los comienzos de los años sesenta. El CI reducía el tamaño de los capacitores, diodos y transistores y los colocaba en una oblea de silicio puro. En 1970 la Cía. INTEL desarrolló su primer micro procesador (CPU) de 8 bits, el 8008, a partir

AÑO	MICRO PROCESADOR	PROPIEDADES
1978	8008	8 BITS, UN BYTE.
1974	8080	USO DE PROPOSITO GENERAL.
1978	8086	COMPATIBLE CON EL 8080; MAS AVANZADO.
1978	8088	VARIACION DEL 8086; COMPATIBILIDAD CON DISPOSITIVOS ACTUALES DE E/S; LA IBM LO ADOPTO PARA SUS PC.
1978	8087	COPROCESADOR MATEMATICO; ALTA VELOCIDAD Y PRECISION EN CALCULOS MATEMATICOS.
1984	80286	APLICACION CON ALTA PRECISION; COMPATIBLE CON EL 8086/8088; GESTION DE MEMORIA; MECANISMOS DE PROTECCION; GESTION DE TAREAS Y SOPORTE DE MEMORIA VIRTUAL; 16 BITS.
1984	80386	COMPATIBLE CON 8086/8088 Y 80286; 32 BITS.
1984	80287/80387	COPROCESADORES MATEMATICOS DE ALTO NIVEL, PRECISION EN CALCULOS MATEMATICOS.

TABLA 2.4 : EVOLUCION DE PROCESADORES.

INSTRUCCION	SIGNIFICADO
LEE "d"	ORDEN DE ENTRADA QUE LEERA DATOS DESDE UN DISPOSITIVO DE ENTRADA Y LOS INTRODUCIRA EN LA DIRECCION "d" DE LA MEMORIA. EJEMPLO : LEE 15 SIGNIFICA : LEE LOS DATOS Y GUERDALOS EN LA DIRECCION 15 DE LA MEMORIA.
IMP "d"	ORDEN QUE ENVIARA LOS DATOS DE LA DIRECCION "d" DE LA MEMORIA AL DISPOSITIVO DE SALIDA.
CAR "d"	CARGAR LA INFORMACION EN EL ACUMULADOR DE LA UNIDAD ARITMETICA-LOGICA DESDE LA DIRECCION "d" DE LA MEMORIA.
ALM "d"	ALMACENA LA INFORMACION CONTENIDA EN EL ACUMULADOR EN LA DIRECCION DE MEMORIA "d".
SUM "d"	SUMA LA INFORMACION DE LA DIRECCION DE MEMORIA "d" AL AL VALOR DEL ACUMULADOR.
RES "d"	RESTA LA INFORMACION DE LA DIRECCION DE MEMORIA "d" DEL VALOR DEL ACUMULADOR.
MUL "d"	MULTIPLICA LA INFORMACION DEL ACUMULADOR POR EL VALOR DE DE LA DIRECCION DE MEMORIA "d".
DIV "d"	DIVIDE LA INFORMACION DEL ACUMULADOR POR EL VALOR DE LA DIRECCION DE MEMORIA
FIN	TERMINA EL PROGRAMA.

TABLA 2.5 : CONJUNTO DE INSTRUCCIONES LES.

de esta fecha la Intel se ha esforzado en construir microprocesadores más eficientes y universales (ver tabla 2.4).

2.2.4 LA UNIDAD ARITMETICO-LOGICA (UAL)

Cuando el estudiante se dispone a realizar los cálculos pedidos en un ejercicio matemático, activa la zona de comprensión del cerebro; de la misma forma la computadora activa la unidad aritmético-lógica de la computadora (UAL). Esta unidad se encarga de ejecutar las operaciones aritméticas y lógicas; los circuitos electrónicos de que consta solamente realizan un número limitado de operaciones básicas, la realización de complicadas operaciones es consecuencia de un extenso aprovechamiento de las instrucciones básicas combinadas eficientemente entre sí; los resultados de estos cálculos se colocan temporalmente en una parte especial de la UAL llamada acumulador. Las operaciones las podemos clasificar como :

i) Instrucciones de cálculo aritmético, para realizar las operaciones de suma y resta.

ii) Instrucciones de cálculo lógico, que comprenden las comparaciones del contenido de dos zonas de memoria y los saltos de ejecución de un grupo de instrucciones a otro.

iii) Instrucciones de copia, para leer una zona de la memoria central y copiarla en otra distinta.

2.2.5.- EL DISPOSITIVO DE SALIDA.

Cuando las personas quieren transmitir información a otros congéneres pueden hacerlo mediante la palabra y la escritura; de la misma manera la computadora

utiliza medios físicos para comunicar información al usuario; los más comunes son : los monitores, las impresoras, los sintetizadores de voz.

2.3.- FUNCIONAMIENTO INTERNO DE LA COMPUTADORA.

2.3.1.- LENGUAJE DE MAQUINA.

Para apreciar el funcionamiento del hardware de la computadora, utilizaré un ejemplo de cómo obtener el área de un rectángulo en lenguaje ensamblador simulado, que es muy próximo al lenguaje de máquina utilizado por la computadora.

El lenguaje de máquina es el **dialecto** natural de la computadora, el cual consiste en una colección de instrucciones muy detalladas y crípticas que controlan la circuitería interna de la máquina (hardware), codificado con ceros y unos. Pocos programas se escriben en lenguaje de máquina por dos razones importantes :

i) El lenguaje es muy incómodo para trabajar.

ii) La mayoría de computadoras tienen su repertorio propio de instrucciones, es decir, un programa codificado en lenguaje máquina difícilmente se puede ejecutar en otra computadora de otro tipo, sin efectuar modificaciones importantes; es decir, no es un lenguaje portátil.

2.3.2.- LENGUAJE ENSAMBLADOR SIMULADO (LES).

El lenguaje ensamblador es el más cercano al lenguaje de máquina; el ensamblador tampoco es portátil, pero es mnemónico y, por tanto, más fácil de codificar. Para visualizar el proceso de la computadora seguiremos de cerca un programa (para obtener el área de un rectángulo) escrito en lenguaje ensamblador. El

cálculo del área de un rectángulo es tratado con detalle en los capítilos 5 y 6 de algoritmos y programación en Pascal, respectivamente.

El ensamblador que vamos a utilizar se llama LES, (Lenguaje Ensamblador Simulado), el cual, ilustrará el flujo de control por el interior de la computadora (el hardware).

Todo lenguaje necesita un conjunto de órdenes o instrucciones (las instrucciones de LES se pueden ver en la tabla 2.5). En todas las instrucciones "**d**" significa una dirección o localización en la memoria RAM.

La memoria de la computadora debe ser capaz de soportar cada una de las instrucciones del programa, así como los datos de trabajo, los resultados intermedios y finales generados por el proceso indicado por el programa. La memoria de nuestra computadora artificial tiene 16 bytes de memoria RAM; donde cada dirección soporta una instrucción o un dato; las primeras nueve posiciones están destinadas a las instrucciones, las direcciones 12 y 13 están destinadas para los datos de entrada y la 14 para el cálculo del área del rectángulo.

En la tabla 2.6 se puede ver el algoritmo de solución y el programa en LES. El programa debe "cargarse" en la memoria RAM de la computadora, donde permanecerá hasta ser "corrido".

El esquema 2.7 representa el programa cargado en la memoria RAM; los valores de los datos se encuentran esperando en la unidad de entrada.

Cuando el CPU recibe la orden de ejecutar el programa, se pone en marcha la orden contenida en la dirección 0 (cero). Examinará una instrucción cada vez y la ejecutará.

LEE 12, activará la unidad de entrada y registrará el dato en la dirección 12, esto es, lee la longitud 30 y depositala en la posición 12 (ver esquema 2.8). De la misma forma, **LEE 13** registrará la anchura (20), en la dirección 13 (ver esquema 2.9).

CAR 12, "cargará" el número que se encuentra en la dirección 12 en el acumulador, i.e., la CPU lee el valor que se encuentra en la dirección 12 (longitud = 30) y lo escribe en su cuaderno de notas, se comunica con la UAL y registra el número en el acumulador (esquema 2.10).

MUL 13, multiplicará el valor en la posición 13 (anchura = 20) por el valor que se encuentra en el acumulador (longitud = 30) y el resultado (600) lo deposita en el acumulador (esquema 2.11).

ALM 14, registrará el valor que se encuentra en el acumulador (área = 600), en la dirección 14 de la memoria RAM (esquema 2.12).

Las órdenes **IMP** enviarán el contenido de las direcciones **12,13 y 14** a la unidad de salida (esquema 2.13).

El esquema 2.14 muestra el estado que guarda nuestra computadora imaginaria al término de sesión.

Se pueden resaltar dos aspectos importantes, uno de los cuales se encuentra relacionado con el acumulador de la UAL y el otro con el lenguaje LES.

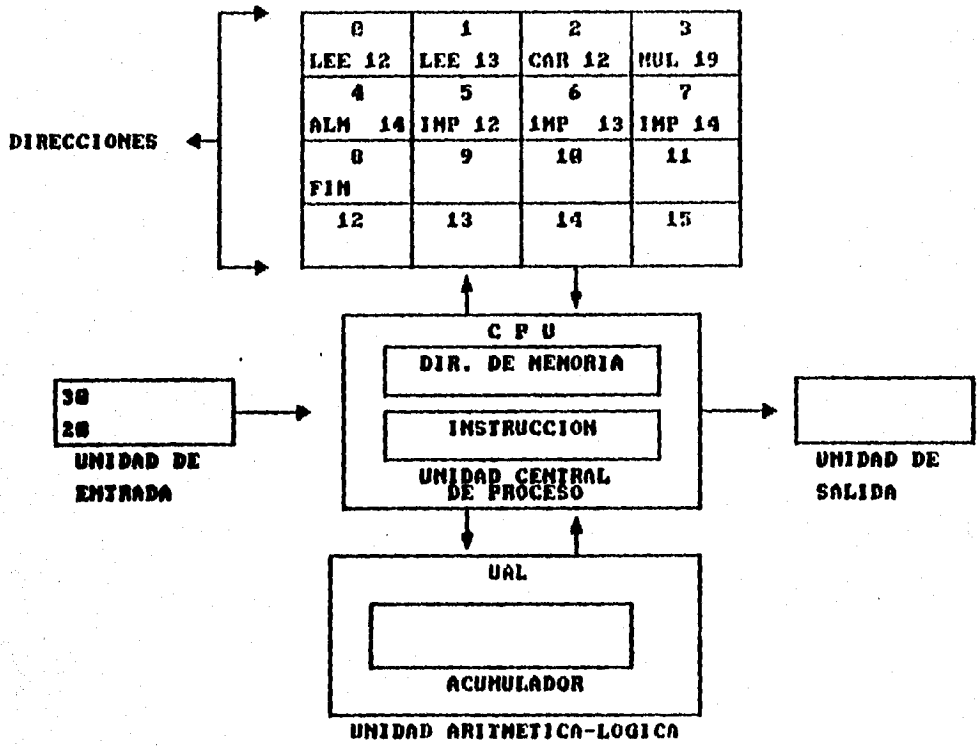
Como se puede apreciar, la función del acumulador es importante para los procesos de la operatividad aritmética de la computadora; supongamos que el acumulador no existe y analicemos el funcionamiento de la computadora sin este componente; para ello sigamos el programa en LES; entonces debemos omitir la instrucción CAR 12 que se relaciona con el acumulador.

La intención del programa es obtener el área de un rectángulo; la orden MUL 13 indica que hay que multiplicar el contenido de la dirección 13 con algo, ¿ con qué ? ¿quién sabe ?, de alguna forma le tenemos que indicar qué valores se tienen que multiplicar tenemos que modificar la orden; MUL 12 13, será nuestra nueva orden para el producto de dos valores que se encuentran en las direcciones 12 y 13; interpretamos MUL 12 13 como multiplica el valor en la posición 12 por el de la 13, parece que con esta instrucción solucionamos la dificultad; ¿en qué localidad almacenamos el dato? Si lo registramos en la 12 o en la 13 perdemos uno de esos valores (el cual se puede utilizar en otra llamada de LES), entonces necesariamente tenemos que utilizar otra dirección en la memoria; por tanto hay que indicarlo en la instrucción, i.e., hay que modificar la instrucción otra vez ; MUL 12 13 14, indicará : multiplica el valor de la posición 12 por la trece y el resultado almacénalo en la 14. ¡Por fin encontramos la

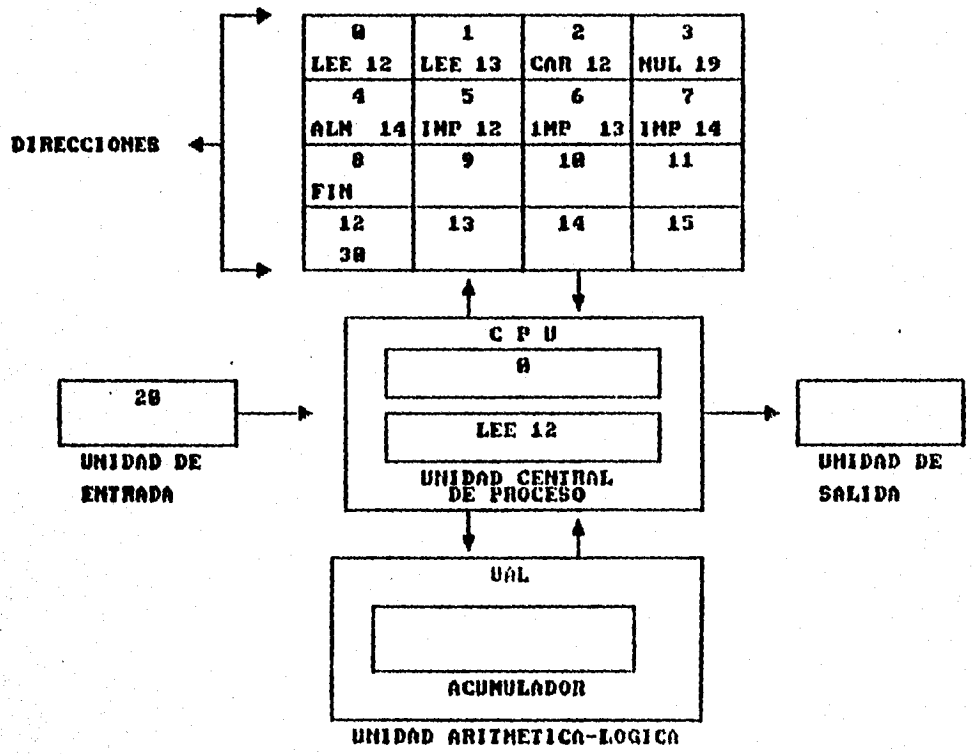
solución! Pero si consideramos que un procedimiento normal de cálculo puede contener 10^9 operaciones básicas, entonces necesitamos aproximadamente una memoria RAM de tamaño ¡ 1000 mega-bytes !, para registrar los resultados temporales y finales. El costo de omitir el acumulador de la UAL es enorme; el acumulador realiza una función de extrema importancia en el funcionamiento de las computadoras.

Como se puede observar, para escribir un programa en lenguaje ensamblador es necesario : escribir el programa de solución del problema detalladamente, con el objetivo de "manejar" adecuadamente los componentes físicos de la computadora, es decir, el hardware. Una ventaja importante que se tiene al escribir programas en lenguaje máquina o ensamblador consiste en que el programador tiene un control absoluto sobre la computadora. A los programas escritos de esta forma se les llama programas de bajo nivel.

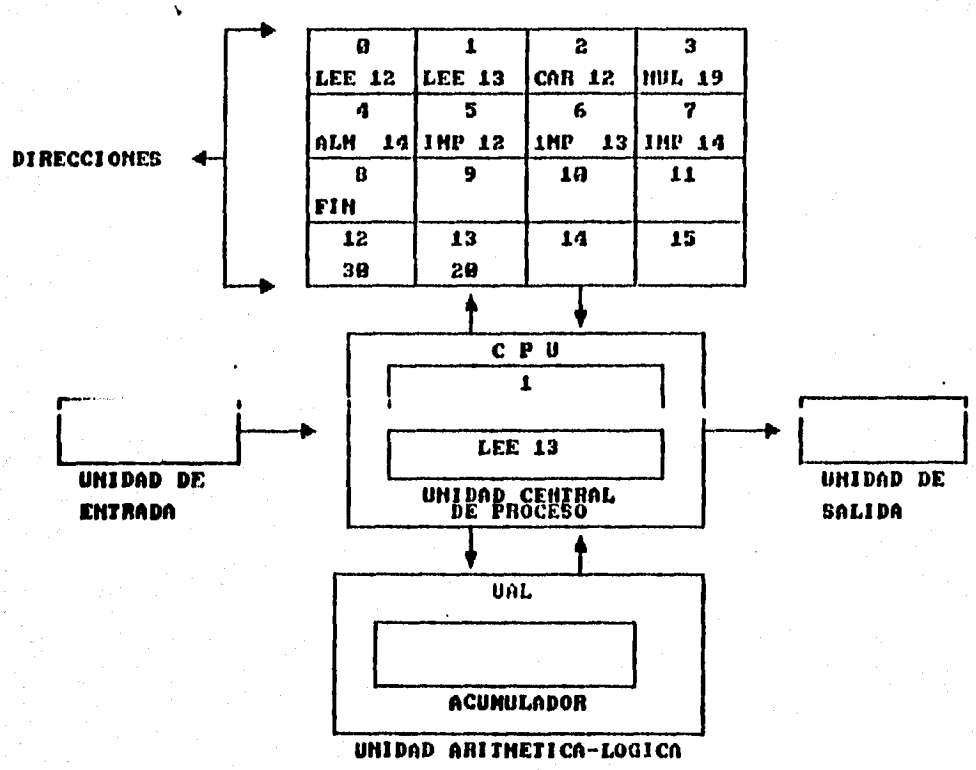
Es más conveniente escribir programas en lenguajes de alto nivel (para los cuales existen traductores que los codifican en lenguaje máquina) por su portabilidad y su escritura no es tan detallada. Entre los lenguajes de alto nivel tenemos: Pascal, Fortran, Lisp, Basic, C, Cobol.



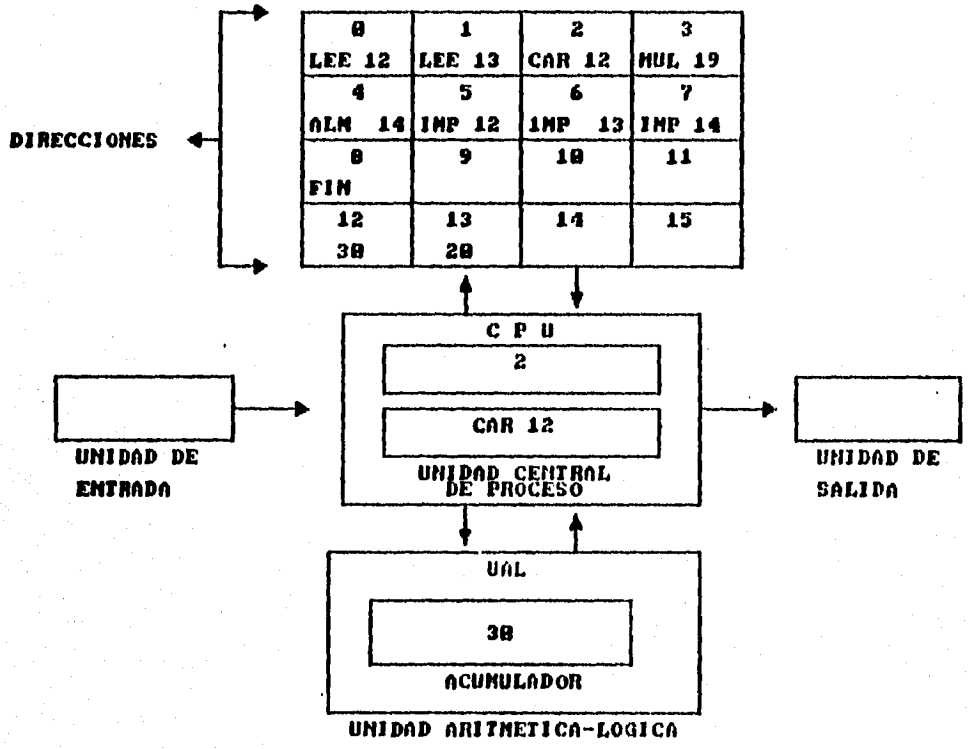
ESQUEMA 2.7 : EL PROGRAMA SE ENCUENTRADA CARGADO EN RAM.



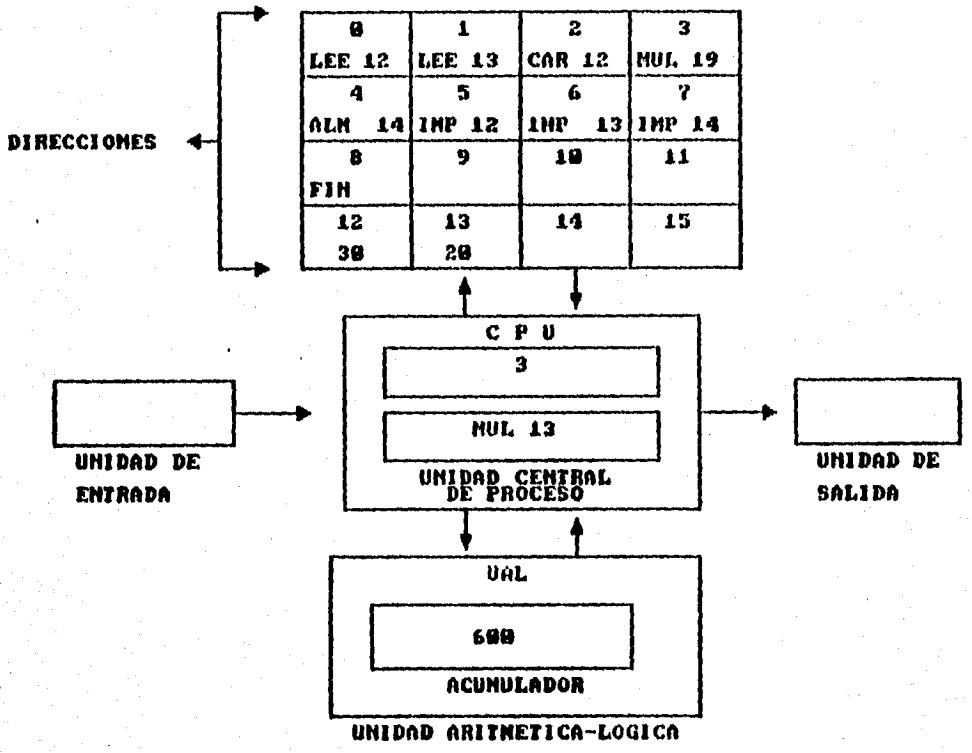
ESQUEMA 2.8 : EL PROGRAMA SE EMPIEZA A EJECUTAR CON LA ORDEN LEE 12
EN LA POSICION 0 (CERO)



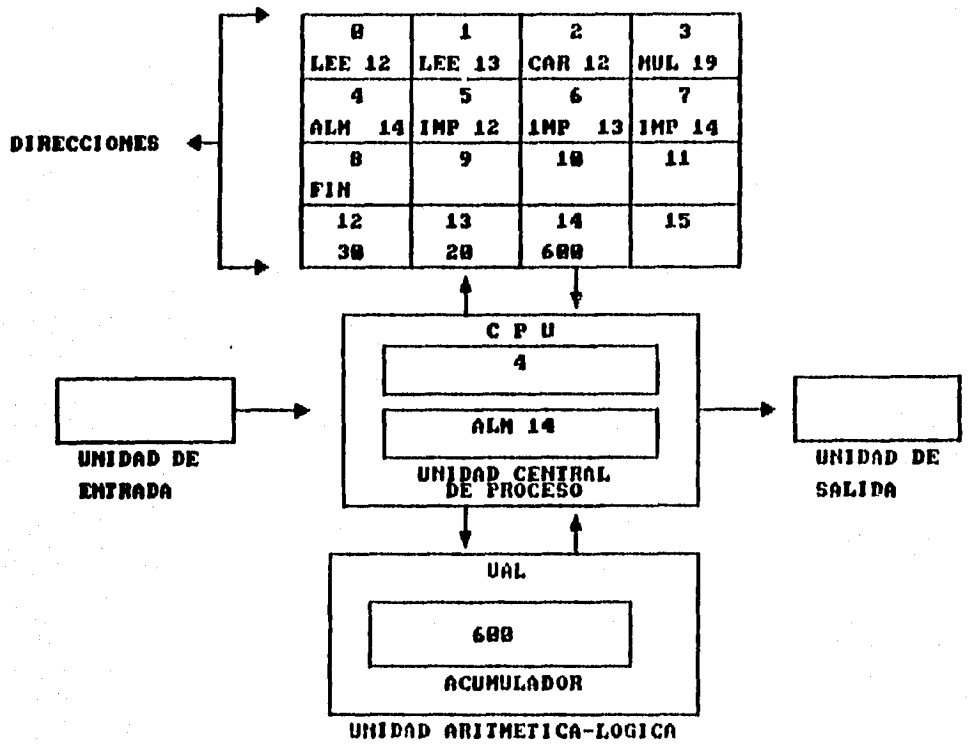
ESQUEMA 2.9 : SE EJECUTA LA ORDEN LEE 13.



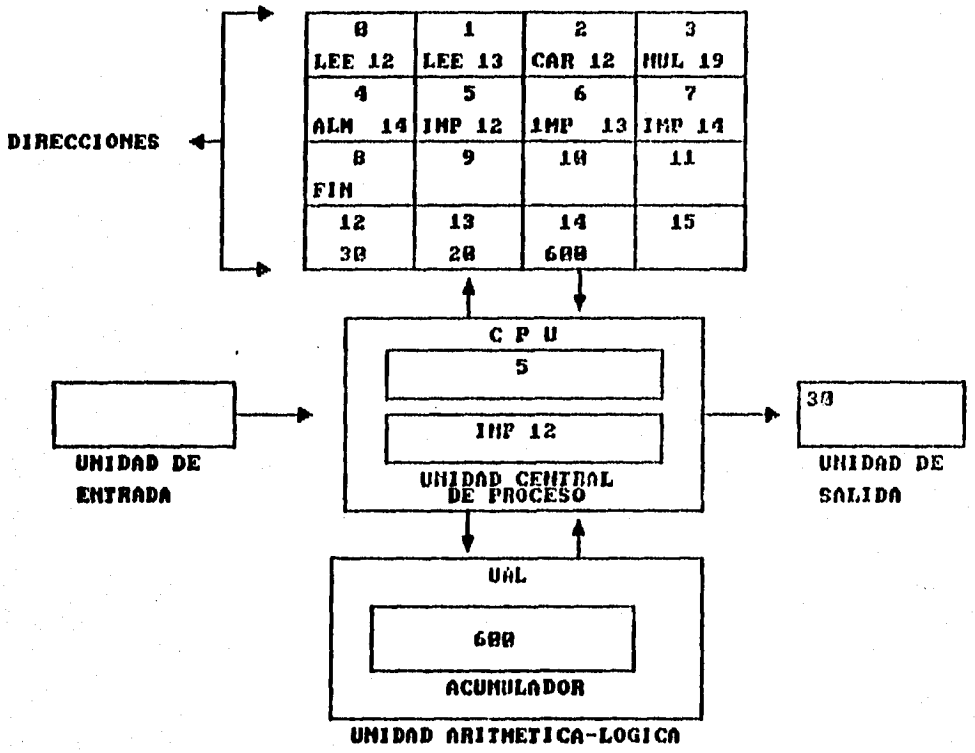
ESQUEMA 2.10 LA INSTRUCCION CAR 12, CARGA EL VALOR 30,
DE LA POSICION 12 EN EL ACUMULADOR.



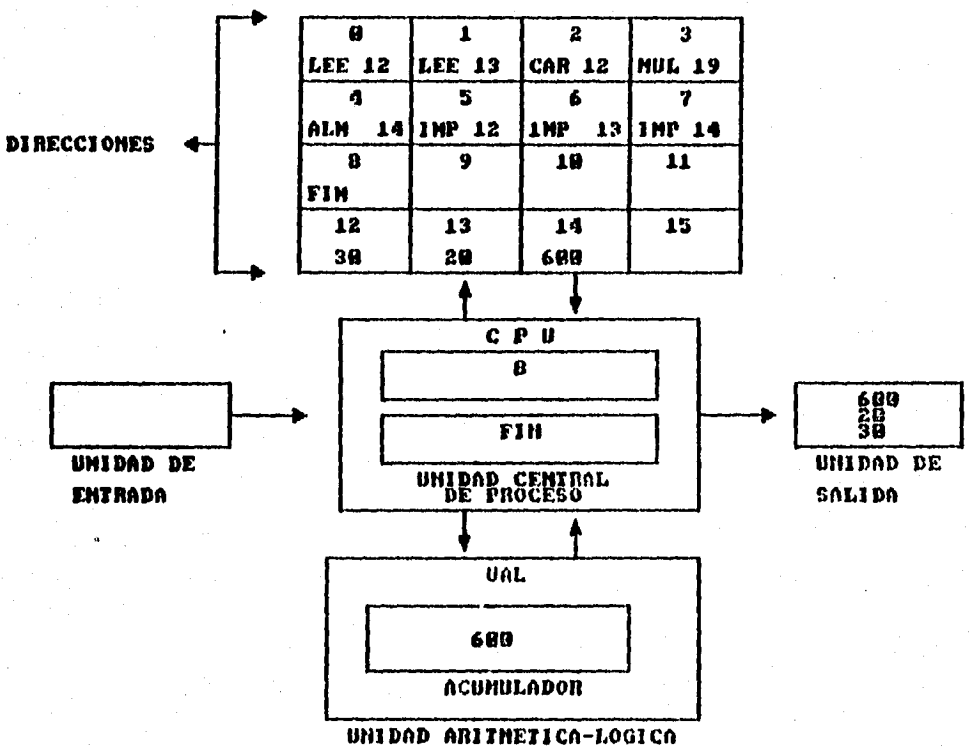
ESQUEMA 2.11 EL VALOR EN EL ACUMULADOR (30) ES MULTIPLICADO POR EL VALOR (20) DE LA DIRECCION 13, PARA OBTENER 600 EN EL ACUMULADOR.



ESQUEMA 2.12 : EL CONTENIDO EN EL ACUMULADOR ES ALMACENADO EN LA POSICION 14



ESQUEMA 2.13 : LOS VALORES DE LAS POSICIONES 12,13 Y 14 SE IMPRIMEN



ESQUEMA 2.14 : EL ESTADO FINAL QUE GUARDA LA COMPUTADORA AL EJECUTAR LA ORDEN FIN.

CAPITULO TRES

SISTEMAS OPERATIVOS.

3.1 SISTEMA OPERATIVO.

Para comprender la importancia que tiene el concepto de sistema operativo, remitámonos a la forma en la que "operaban" las computadoras de las primeras generaciones.

Las primeras máquinas de .cálculo (automáticas y semiautomáticas) eran gestionadas por el usuario desde un tablero con enchufes, esto es , para programar la computadra en lenguaje de máquina (el único que existía) las órdenes se transmitían conectando un sin número de enchufes. Posteriormente el control de la computadora se realizaba por medio de una consola, se asignaba un solo usuario en cada ocasión y por un tiempo determinado, la información se introducía por medio de lectoras de tarjetas (1950); existía lo que se llama mono programación, es decir, un programa en cada sesión.

Dado el alto costo de la monoprogramación se crearon Diferentes puestos altamente especializados : de programación, operadores y personal de mantenimiento. El "operador" controlaba el sistema, cargaba los programas, obtenía los resultados etc.; el programador dejó de tener acceso directo a la computadora; el procedimiento de trabajo consistía en : los programadores daban los trabajos al operador, éste los reunía y los ejecutaba uno detrás de otro y recogía los resultados obtenidos, para entregarlos a los programadores. Una modificación importante en la administración de las computadoras, consistió en agrupar aquellos programas que requerían los mismos dispositivos físicos, esto es, escritos en el mismo lenguaje (Fortran, Cobol, Algol); de esta forma el traductor y el sistema operaban más eficientemente, con el consecuente ahorro de tiempo.

El trabajo del "operador" era repetitivo, por lo que se optó por diseñar un programa para realizar el trabajo en forma automática; dicho programa se denominó Monitor Residente, que puede ser considerado como el primer sistema operativo; el nombre sugiere que se encontraba almacenado constantemente en la computadora. Al encender la computadora se daba control al programa monitor, éste daba el control al primer programa y al terminar su proceso, se transferia el control al siguiente programa, y así sucesivamente.

El programa monitor estaba constituido por las siguientes partes:

i) El secuenciador automático de trabajo.

ii) El intérprete de las tarjetas de control.

iii) Controladores software de entrada/salida.

Para la ejecución de un programa se debía armar un paquete de tarjetas perforadas, donde habia distintos grupos de tarjetas; en el siguiente orden : tarjetas de control, del programa, de control, de datos y de control; de esta forma el monitor residente identificaba el inicio de sesión, el programa en si, la terminación de éste, los datos y el fin del programa.

El desarrollo de periféricos de entrada y salida, así como el de la computadora en general , implicó el desarrollo de sistemas operativos más eficaces que satisficieron las múltiples necesidades de los usuarios.

Para encontrar una definición de sistema operativo, analicemos las palabras que lo forman y procedamos a encontrar una definición adecuada.

Sistema :

Conjunto de personas, máquinas y cosas que, ordenadamente relacionadas entre sí, contribuyen a lograr un determinado objetivo.

Operativo :

Personas, máquinas y cosas que trabajan conjuntamente, de acuerdo a un proceso determinado, para conseguir el objetivo deseado.

De las definiciones anteriores obtenemos la de sistema operativo :

SISTEMA OPERATIVO :

Conjunto de programas que ordenadamente organizados entre sí, contribuyen a que la computadora lleve a cabo su trabajo correctamente.

Los sistemas operativos cubren dos objetivos fundamentales :

i) Gestionar de forma eficiente los recursos, esto es, administrar los recursos ofrecidos por el hardware óptimamente para alcanzar un eficaz rendimiento de los mismos.

ii) Facilitar el trabajo al usuario, es decir, ocultar los detalles del hardware, ofreciendo al usuario una forma sencilla y flexible de acceso al mismo.

Del segundo objetivo se desprende el concepto de Máquina Virtual, es decir, el sistema operativo al esconder el aspecto físico de la computadora al usuario , le muestra una máquina que tiene la virtud (el poder) para realizar un proceso aunque en sí no lo produzca. A la máquina virtual también se le conoce como Máquina Extendida.

3.2.- ESTRUCTURA DE LOS SISTEMAS OPERATIVOS.

Existen distintas estructuras de los sistemas operativos según las necesidades que se requieran satisfacer.

3.2.1.- ESTRUCTURA MONOLITICA.

Los primeros sistemas operativos se encontraban constituidos por un solo programa compuesto por un conjunto de subrutinas entrelazadas entre sí, de tal forma que cada una podía llamar a cualquier otra. Este tipo de sistema operativo se le llama de estructura monolítica; sus principales características son :

i) Construcción del programa final a base de módulos compilados separadamente que se unen a través del editor de enlace.

ii) Buena definición de parámetros de enlace entre las distintas rutinas existentes.

iii) Carecen de protecciones y privilegios.

iv) Generalmente están hechos a la medida, por lo que son eficaces y rápidos en su ejecución y gestión.

3.2.2.- ESTRUCTURA JERARQUICA.

En función del desarrollo de los sistemas computacionales y las necesidades de los usuarios, se modificó la estructura de los sistemas operativos; pasando de una estructura monolítica a una dividida, de tal forma que cada parte se relaciona con cada una de las otras; la primera estructura de este tipo fue desarrollada por Dijkstra, con fines didácticos y se le llamo THE (Technische Hogeschoole, Eindhoven).

Sus principales características se pueden ver en la tabla 3.1. En esta estructura se basan la mayoría de los sistemas operativos actuales.Las zonas de

NIVEL	5	USUARIOS
NIVEL	4	ARCHIVOS
NIVEL	3	ENTRADA/SALIDA
NIVEL	2	COMUNICACIONES
NIVEL	1	MEMORIA
NIVEL	0	GESTION CPU
NIVEL	-1	HARDWARE

TABLA 3.1 SISTEMA JERARQUICO THE.

UN USUARIO	UN USUARIO	VARIOS USUARIOS	UN USUARIO	VARIOS USUARIOS
DOS/VS	CMS	UNIX	OS/2	VMS
HARDWARE VIRTUAL	HARDWARE VIRTUAL	HARDWARE VIRTUAL	HARDWARE VIRTUAL	HARDWARE VIRTUAL
CP	CP	CP	CP	CP
HARDWARE				

TABLA 3.2 : ESTRUCTURA DE MONITOR VIRTUAL.

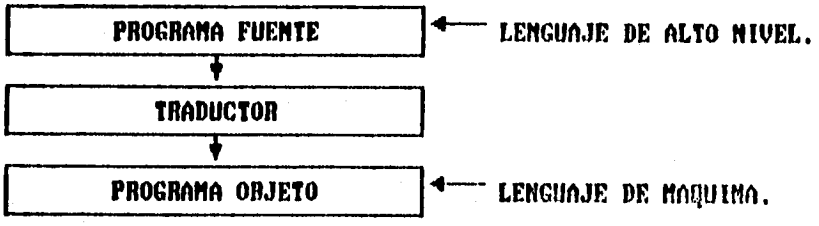


TABLA 3.3 : ESQUEMA GENERAL DEL PROCESO DE TRADUCCION.

menor nivel se encuentran protegidas de accesos indeseados, por tanto son más privilegiadas que las externas.

3.2.3.- ESTRUCTURA DE MAQUINA VIRTUAL.

El sistema operativo con estructura de máquina virtual presenta un interface a cada proceso, mostrando una máquina extendida al usuario. El sistema operativo de máquina virtual esta diseñado de tal forma que le presenta a los usuarios una gama de distintos ambientes de trabajo, es decir, muestra sistemas operativos diferentes.

El núcleo de estos sistemas operativos se denomina monitor virtual y tiene como objetivo efectuar la multiprogramación; presentando a los niveles superiores tantas máquinas virtuales como se necesiten. Estas máquinas extendidas, están diseñadas de tal manera que en cada una de ellas se puede ejecutar un sistema operativo diferente, que será el que la máquina virtual ofrezca al usuario (tabla 3.2).

3.2.4.- ESTRUCTURA CLIENTE-SERVIDOR.

El sistema operativo tipo cliente-servidor; es el tipo más reciente. Es de uso general, puede ser ejecutado en computadoras chicas y grandes; suministra mecanismos para la gestión de : procesos, memoria y comunicación entre procesos; maneja los conceptos de servidor y cliente, es decir, un programa de aplicación normal es un cliente que llama al servidor correspondiente para acceder a un archivo o realizar un proceso de entrada-salida. La estructura cliente-servidor es recurrente, en el sentido de que un proceso cliente puede actuar como servidor para otro distinto.

3.3 COMPILADORES E INTERPRETES.

Todo sistema operativo posee un conjunto de programas de proceso encargados de la ayuda a los programadores en la realización y puesta a punto de

los programas. Entre los programas de proceso del sistema operativo se encuentran los llamados traductores, que como su nombre lo indica, se encargan de codificar (traducir) los programas escritos por los usuarios (por lo general escritos en algún lenguaje de alto nivel) al lenguaje máquina, el cual es el único que entiende la computadora. El programa escrito por el usuario se le llama programa fuente y a su traducción equivalente en lenguaje máquina se le denomina programa objeto, (tabla 3.3). El programa objeto es el que "corre" la computadora.

Básicamente existen dos tipos de traductores, los compiladores y los intérpretes; su proceso de traducción es diferente, pero por ambos métodos se llega al mismo resultado.

Los compiladores son traductores encargados en transformar el programa fuente en programas objeto; su característica principal consiste en traducir el programa del usuario completamente a lenguaje máquina, esto es, si no existen errores de sintaxis* (en el programa fuente), entonces crea el programa objeto.

---

* Un error de sintaxis consisteen escribir una instrucción o sentencia incorrectamente.

Los traductores de tipo intérprete, codifican una instrucción a lenguaje máquina y la ejecuta inmediatamente, es decir, no esperan a la traducción de las siguientes líneas del programa.

3.4 TIPOS DE SISTEMAS OPERATIVOS

En la actualidad existe una gran variedad de sistemas operativos dependiendo del tamaño y uso que se le dé a la computadora. Hay sistemas diseñados a la medida, es decir para un uso particular, tales como los sistemas operativos contenidos en los juegos de nintendo si nintendo, atari y demás atolondramientos infantiles. La mayoría de los sistemas operativos son de uso general; en esta sección se describirá un sistema operativo multi-usuario (Unix), uno implementado en grandes sistemas (OS/400) y se revisará con detalle el sistema operativo DOS, el cual soporta el equipo con el que cuenta el CCHA.

3.4.1.- SISTEMA OPERATIVO UNIX.

El sistema operativo UNIX fue desarrollado, en 1969 en los laboratorios de la empresa Bell Telephone Laboratories, por Thompson & Ritchie para uso interno de la compañía; el UNIX administró una computadora PDP-7 y permitió la monoprogramación. En 1974 se introduce el sistema operativo UNIX en las Universidades norteamericanas. En 1977 aparece la primera versión comercial. Es tan potente el UNIX y de uso tan generalizado en ambiente red, que la mayoría de fabricantes de computadoras tiene su propia versión de UNIX (tabla 3.4)

ESTRUCTURA DEL SISTEMA OPERATIVO UNIX.

UNIX es un sistema multiusuario, en el que existe portabilidad para la implantación en distintas computadoras.

COMPAÑIA	VERSION
DIGITAL EQUIPMENT CORPORATION	ULTRIX
IBM	AIX
DATA GENERAL	DG/UX
NATIONAL SEMICONDUCTOR	GENIX
GOULD CONCEPT 32/6750	UTX
HEWLETT PACKARD	HP-UX

TABLA 3.4 : VERSIONES DEL SISTEMA OPERATIVO UNIX.

```
login : su_nombre
Password :
you have mail
$
$ mail
Hola, esta es una prueba de correo UNIX
$ ^D
login :
```

ESQUEMA 3.5 : INICIO Y FIN DE UNA SESION UNIX.

Se puede pensar que está formado por círculos concéntricos, donde el círculo más interno contiene el hardware, rodeándolo el núcleo (kernel), que interactua directamente con la máquina física, aislándolo de los usuarios. La segunda capa está formada por los comandos, que no son otra cosa que la interface entre los programas de aplicación y el núcleo del UNIX. La última capa contiene los programas de aplicación de los usuarios.

SESION UNIX.

El trabajo de un usuario en UNIX se organiza por sesiones, la cual comprende desde que la identificación del usuario, hasta que se despide del sistema (esquema 3.5).

Una sesión UNIX da principio con la identificación del usuario, con login:; el sistema operativo espera el nombre del usuario escrito con minúsculas, pues de otra forma no lo entiende.

La sentencia Password: está pidiendo la clave de acceso del usuario; la cual no se ve en el monitor; una vez que UNIX la reconoce el usuario puede interactuar con la computadora en ambiente UNIX.

El Sistema operativo como administrador y facilitador de trabajo transmite diferentes mensajes al usuario, tal como: You have mail (Usted tiene correspondencia electrónica).

El símbolo $ (%) denota el prompt, el cual indica que el sistema operativo se encuentra listo para recibir instrucciones o comandos; lo genera el intérprete de comandos o << shell >>, este símbolo se presentará tantas veces como se pulse la tecla <RETURN>.

El comando $ mail solicita un listado de la correspondencia electrónica del usuario (Hola, esta es una prueba del correo UNIX).

La instrucción $ ^D (Ctrl D), indica fin de sesión, UNIX.

En general el sistema operativo UNIX está diseñado para gestionar archivos y directorios con estructura jerárquica en forma de árbol. El núcleo de UNIX identifica cada proceso con un número el cual es único. El SHELL es una interface entre las órdenes (comandos), utilidades y programas que da el usuario y el sistema; el shell es un programa del sistema operativo, pero no forma parte de su núcleo; le corresponde es un lenguaje de programación de alto nivel.

En UNIX existe un directorio que es de uso exclusivo de un super-usuario llamado administrador del sistema, el cual contiene una serie de comandos, que nada más pueden ser llamados por el administrador.

El administrador del sistema es quien organiza y designa el arranque del sistema. Podemos señalar sus principales funciones, tales como:

-Comprobar si el sistema de archivos es correcto, en caso contrario restaurar los archivos dañados.

-Limpiar el registro de usuarios activos.

-Montar el sistema de archivos.

-Limpiar los directorios temporales.

-Arrancar los procesos iniciales.

-Presentar la fecha en la consola maestra.

-Enviar mensajes de aviso a las terminales en servicio.

-Terminar o interrumpir los procesos activos, excepto los de la consola maestra.

-Desmontar el sistema de archivos.

3.4.2.- SISTEMA OPERATIVO OS/400.

La computadora IBM Application System/400 (AS/400) apareció a finales de los años 80; posee el esquema de conectividad System Application Architecture (SSA) de IBM entre grandes computadoras, computadoras de tipo medio y computadoras personales, es decir, su arquitectura está diseñada en la conectividad; se encuentra organizada en unidades estructurales y lógicas conocidos como: objetos, bibliotecas, archivos y carpetas. Uno de los principales aspectos del sistema AS/400 es que trata todas las entidades como objetos, incluyendo las impresoras, bases de datos, programas y pantallas; el sistema almacena las descripciones de todos los objetos y cuando un usuario hace la referencia a un objeto, el sistema lo reconoce y lo busca, obteniendo la información completa de dicho objeto.

El sistema operativo de la computadora AS/400 es el Operating System/400 (OS/400); su principal característica es la facilidad de operación que le ofrece al usuario, permitiendole trabajar a partir de un menú y ayuda de todo tipo, es decir, existe un interface con el usuario. En el sistema operativo OS/400, o los usuarios pueden controlar el flujo de trabajos para maximizar el sistema o el sistema toma el control automático de los trabajos de los usuarios.

El sistema operativo OS/400 cuenta con una caja de herramientas, (un conjunto de programas) integrados al sistema para servicio del usuario; entre las herramientas mencionamos :

1.- Utilidades de entrada al sistema. Es un editor de texto, que permite la escritura de programas y consta de un diccionario para la corrección de sintaxis.

2.- Diseño de pantallas. Facilita el diseño de pantallas y de menús, siendo compiladas inmediatamente.

3.- Gestor para el desarrollo de programas. Es una aplicación que permite trabajar con bibliotecas y objetos, así como exportar un programa a otra biblioteca para su utilización de otros usuarios.

4.- Gráficos. Se trata de una herramienta para el diseño de gráficos.

5.- Utilidad de archivos de datos. Facilita la edición y visualización de bases de datos creando programas de entrada/salida de usuario.

6.- Consulta. Permite la creación de un programas, principalmente para extraer informes, mostrando los resultados de cálculos diversos, así como la actualización de bases de datos.

7.- Office Vision. Es el procesador de textos, diseñado para el trabajo rutinario de oficina; consta de las siguientes subherramientas:

7.1.- Calendario. Es una agenda electrónica que está habilitada para ejecutar un programa en un hora especificada de antemano.

7.2.- Editor de texto. Permite utilizar diferentes editores de texto: AS/400 office, WordPerfect, Display Write entre otros.

7.3.- Correo y mensajes. Permite el intercambio de información vía correo electrónico.

7.4.- Gestor de computadoras personales. Controla computadoras personales conectadas.

8.- Lenguajes de alto nivel. El sistema operativo OS/400 soporta diferentes compiladores de lenguajes de alto nivel estándar (RPG/400, COBOL/400, SQL/400 y C/400 entre, otros) que pueden ser ejecutados en otros sistemas (IBM-370, AS/400 o PS/2).

3.4.3.- SISTEMA OPERATIVO DOS.

El sistema operativo Disk Operating System, DOS (sistema operativo de disco) es un sistema monotarea y monousuario diseñado para la gama de computadoras personales (PC); lanzadas al mercado por la IBM en 1981; es utilizado por una gran diversidad de sistemas compatibles, construidos por infinidad de compañías en todo el mundo.

La estructura del sistema operativo DOS, la podemos visualizar como un conjunto de círculos concéntricos, donde el círculo más anidado contiene una parte de software llamado BIOS (Basic Input Output System) residente en la ROM. El siguiente nivel corresponde al núcleo del sistema operativo que reside constantemente en memoria RAM; está formado por el intérprete de comandos (COMMAND.COM) que contiene un conjunto de comandos residentes en RAM y dos archivos de los denominados ocultos que permiten ampliar y actualizar las rutinas de la ROM-BIOS.

El BIOS tiene dos funciones básicas :

i) Realizar una prueba de todo el equipo, donde se examina a todos los elementos conectados y el estado que guardan; la prueba se realiza al arrancar el sistema.

ii) Efectuar un interface entre el software de los niveles superiores y el hardware a través de una serie de rutinas, cada una con una función especifica.

EL intérprete de comandos del DOS llamado COMMAND.COM realiza funciones de interface entre el usuario y la computadora; el COMMAND.COM contiene un conjunto de comandos internos, esto es residen permanentemente en RAM. El DOS posee una serie de comandos llamados malamente externos, (más bien son transitorios) esto es, no se encuentran incluidos en el COMMAND.COM, para su uso es necesario que estén presentes en una unidad activa. Otra función del intérprete de comandos es el de mantener el indicador del DOS (prompt), el cual indica que el sistema operativo se encuentra listo para realizar una tarea; el prompt se representa con una letra mayúscula seguida por el simbolo ">"; la letra indica en que unidad activa se encuentra el usuario.

Un comando consta de tres partes : nombre del comando, parámetros y modificadores, escritos en este orden.

Nombre del comando.

Se escribe en primer término e indicará la acción que se desea que el DOS realice.

Parámetro.

Define el elemento sobre el que se va a realizar la acción del comando. Por ejemplo : el comando **DEL** (eliminar) requiere el nombre del archivo que se desea borrar. algunos comandos requieren más de un parámetro.

Modificadores.

Se representa por una diagonal derecha "/" seguida de una letra o número; indica la forma en que un comando realiza una tarea. Por ejemplo : el comando **DIR** (directorio) despliega todos los archivos almacenados en memoria auxiliar en forma rápida; si utilizamos la orden **DIR** con el modificador **/p** desplegará todos los archivos que quepan en una pantalla, y proseguirá con la lista cuando se le indique.

TRABAJANDO CON ARCHIVOS.

La información para su tratamiento computacional se organiza en archivos. Existen archivos con instrucciones (programas) y con información (bases de datos, textos).

A los archivos se les asigna nombre y extensión, respetando las siguientes reglas, para el nombre :

i) No deben tener más de ocho caracteres.

ii) Pueden constar solamente de las letras de la A a la Z, de los números del 0 al 9 y los siguientes caracteres especiales :

Subrayado.	_	Intercalación.	^
Tilde.	~	Exclamación derecho.	!
Porcentaje.	%	Ampersand.	&
Guión.	-	Llaves.	()

Paréntesis. () Número. #

Dolar. $

Ningún otro carácter especial es aceptado.

iii) No podrás contener espacios, comas, barras inversas y puntos (excepto el que separe el nombre del archivo de su extensión).

iv) No se podrán usar los siguientes nombres de archivo reservados : CLOCKS$, CON, AUX, COMn (donde n = 1..4), LPTn (donde n = 1..13), NUL y PRN.

Recomiendo las siguientes reglas para nombres de archivo; las cuales no son oficiales, pero en la práctica dan buen resultado :

i) El nombre consiste de un carácter alfabético, seguido de letras y/o números.

ii) Solo se permiten caracteres alfanuméricos.

iii) La mayoria de los sistemas operativos solo reconoce los primeros ocho caracteres.

iv) No se permite el uso de palabras reservadas[1] , para el nombre de archivos.La extensión por lo general indica la naturaleza del archivo, indicando si es un archivo especial o si pertenece a un software en particular (tabla 3.6). Los nombres de las extensiones, por lo general son asignados por el software con el que se trabaje.

---

[1] Palabra reservada. Es una palabra que tiene un significado especifico dentro del software, por ej. : AUX, CLOCK, DIR, NUL, ETC.

EXTENSION	SIGNIFICADO
SYS	ARCHIVO DEL SITEMA.
EXE	ARCHIVO EJECUTABLE
COM	ARCHIVO EJECUTABLE
PAS	ARCHIVO CODIFICADO EN PASCAL
DBF	ARCHIVO BASE DE DATOS (PARA EL MANEJADOR DE DBASE)
NTX	ARCHIVO DE INDICE DE BASE DE DATOS (PARA EL MANEJADOR DBASE)
CHI	ARCHIVO DE TEXTO
BAT	ARCHIVO CON COMANDOS DEL SISTEMA OPERATIVO.

TABLA 3.6 : EXTENSIONES DE ARCHIVOS.

La extensión no deberá tener más de tres caracteres. Se aplican las mismas reglas oficiales que para el nombre del archivo.

En algunos comandos se utilizan junto con los parámetros caracteres comodines, al igual que en la baraja el comodin representa las letras (baraja) que queramos. Existen dos comodines en el DOS :

i) El asterisco "*" que representa al menos una letra.

ii) El signo de interrogación final "?" que toma el lugar de un carácter únicamente.

Por ejemplo; si tenemos un conjunto de archivos con los siguientes nombres :

tarea01.pas	tarea06.pas	alumnos1.dbf	alumn601.dbf
tarea02.pas	tarea07.pas	alumnos2.dbf	alumn601.prg
tarea03.pas	tarea01.chi	alumnos3.dbf	alumn601.ntx
tarea04.pas	tarea02.chi	alumnos4.dbf	alumn602.dbf

tabla 3.7. Nombres de archivos.

Si queremos listar los archivos con extensión .pas tenemos que escribir : **DIR *.PAS**, aquí el carácter comodín asterisco "*" representa cualquier nombre de archivo, lo único que interesa es en este caso la extensión.

El COMANDO **DIR ALUMNOS?.DBF**, listará todos los archivos que sus primeros siete caracteres sean ALUMNOS y el octavo carácter puede ser cualquiera, la extensión es **.DBF**.

3.4.4.-COMANDOS DE MAYOR USO DEL SISTEMA OPERATIVO DOS.

Anteriormente se mencionó el interprete de comandos COMMAND.COM del DOS, se especificaron algunas de sus características más importantes. En este módulo explicaremos los comandos de mayor uso del sistema operativo DOS.

El tema que trata cada comando presenta, junto al nombre del mismo, un cuadro que indica las categorías a la que pertenece (tabla 3.8). Si un comando tiene asociado el símbolo "ε" indica que pertenece a esa categoría; el símbolo "NO ε", indica no pertenece a esa categoría.

∈ DOS
NO ∈ LOTES
NO ∈ CONFIG.SYS
∈ INTERNO
NO ∈ EXTERNO
∈ RED

TABLA 3.8 : CATEGORIA DE LOS COMANDOS DEL DOS.

El significado de las categorías es el siguiente :

-DOS. El comando es instrucción básica del sistema operativo DOS.

-LOTES. El comando es una instrucción de un archivo con extensión .BAT.

-CONFIG.SYS. Los comandos de esta categoría sirven para personalizar el sistema.

-INTERNOS. Son comandos incluidos en el intérprete de comandos COMMAND.COM del sistema operativo DOS.

-EXTERNO. Son comandos no incluidos en el COMMAND.COM; para su ejecución es necesario que estén almacenados en memoria auxiliar (disco) y presentes en el momento de invocarlos; los comandos externos tienen las extensiones .COM, .EXE o .BAT.

-RED. El comando trabaja en ambiente de red.

SOBRE LA SINTAXIS DE LOS COMANDOS.

La sintaxis de los comandos del DOS indica la forma correcta para escribir el nombre, los parámetros y los modificadores de los comandos; existe cierta uniformidad para escribirlos. Los elementos que se presentan en negrita (**A a, B b, C c,...Zz**) deben ser escritos exactamente en esa forma; los elementos que aparecen en letra estándar (A a, B b, C c,...Z z) indican información suministrada por el usuario. A continuación se presenta un ejemplo de sintaxis de un comando del sistema operativo DOS :

nombrecomando [+r | -r][unidad:][ruta]nombrearchivo[...]
[opciones]

Donde cada uno de los elementos tiene el siguiente significado :

nombrecomando

Indica el nombre del comando.

[]

Determina un elemento optativo. En caso de utilizarlo escriba los datos que se solicitan, NO los corchetes.

|

Separa dos opciones; aquí hay que escojer una u otra, pero no ambas.

unidad:

Especifica la unidad de disco (A:, B:, C: o la unidad correspondiente), en la que se encuentra el comando externo o el nombre_archivo sobre el cualva a . actuar el comando.

ruta

Especifica la dirección que el sistema deberá seguir.

nombrearchivo

Indica el nombre del archivo sobre el que va actuar el comando.

...

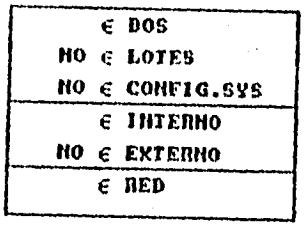
Denota que un parámetro o modificador puede ser utilizado más de una ocasión. Escribá sólo la información NO los puntos suspensivos.

opciones

Indica uno o más parámetros o modificadores opcionales para el comando.

A continuación se describen algunos de los comandos más utilizados en un curso introductorio en el CCHA. No son los más importantes ni los únicos. Algunos comandos no se presentan con todos sus parámetros o modificadores; pues son de uso no generalizado y no son necesarios para un curso inicial.

Para ver todas las opciones de los comandos consulte el manual del DOS. Los presentaremos en orden alfabético.



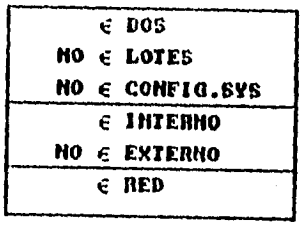
CLS (Limpiar pantalla)

Borra la pantalla La pantalla muestra el prompt y el cursor

SINTAXIS.

cls

COPY (Copiar)



Copia al menos un archivo, a la unidad especificada. Si se copia más de un archivo, el DOS despliega el nombre de cada uno al copiarlo. Se pueden copiar con nombre diferente.

SINTAXIS.

COPY [/a |/b] origen [/a |/b] [+origen [/a |/b] [+...]] [destino [/a |/b]] [/v]

PARAMETROS

origen

Representa la [unidad:][ruta] archivo origen, es decir indica la dirección así como el nombre del archivo o archivos que serán copiados.

destino

Denota la [unidad:][ruta] archivodestino, esto es, especifica el destino , así como el nombre del archivo o archivos en los que se depositan la(s) copia(s). Si no aparece la [unidad:] toma la que está activa en ese momento.

MODIFICADORES.

Para referencia de los modificadores : **/a**, **/b** y **/v** consulte un manual del DOS.

Ejemplo 1. Se copiará el archivo TAREAS.PAS de la unidad activa A: a la unidad B:. a) con el mismo nombre; b) con el nombre POLINOMI.PAS especificando la unidad de origen A:.

a) A>COPY TAREAS.PAS B:

b) A>COPY A:TAREAS.PAS B:POLINOMI.PAS

El ejemplo 1.b se puede escribir de la forma :

A>COPY TAREAS.PAS B:POLINOMI.PAS

Dado que la unidad activa es la A: y no es necesario nombrarla al escribir la orden.

Ejemplo 2. Se copiarán todos los archivos que se encuentran en la unidad A: con extensión .PAS a la unidad B:, la cual se encuentra activa.

B>COPY A:*.PAS

Ejemplo 3. Una forma de efectuar el respaldo de la información de un disco es :

A>copy *.* B:

Donde el disco origen se encuentra en la unidad por defecto A: y el disco destino en la unidad B:

Ejemplo 4. Si se desean concatenar (Adicionar, uno a continuación del otro) los archivos tarea01.pas, tarea02.pas y tarea03.pas en el archivo tareas.pas, tenemos que escribir la orden COPY de la siguiente forma :

A> COPY TAREA01.PAS + TAREA02.PAS + TAREA03.PAS TAREAS.PAS

Si no especifica el archivo destino, la concatenación se efectuará en el primer archivo (tarea01.pas).

DATE (Fecha)

	€ DOS
NO	€ LOTES
NO	€ CONFIG.SYS
	€ INTERNO
NO	€ EXTERNO
	€ RED

Despliega la fecha y permite modificarla, como se encuentre en ese momento en memoria o en el reloj de la máquina. El DOS registra la fecha actual de cada archivo que es creado o modificado.

SINTAXIS.

DATE [dd-mm-aa]

PARAMETRO.

dd-mm-aa

Donde dd, mm y aa representa dia, mes y ano respectivamente. Pueden ir separados por punto (.), guión (-) o diagonal a la derecha (/). El

orden del día, mes y año varía dependiendo de la configuración especificada para la orden COUNTRY en el archivo CONFIG.SYS. Los valores válidos para el parámetro dd-mm-aa son :

dd 1 a 31

mm 1 a 12

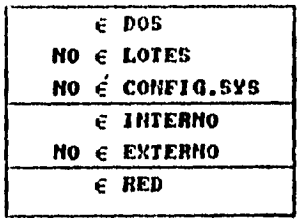
aa 80 a 99 ó 1980 a 2099

COMENTARIOS.

La importancia de registrar la fecha actual radica en que se tiene actualizada la fecha de la última modificación de un archivo o la creación, lo cual es importante para las referencias futuras del mismo.

DIR (Directorio).

Presenta una lista de los archivos y subdirectorios existentes en un directorio.



SINTAXIS.

dir [unidad:][ruta][nombrearchivo] **[/p]** [/w] [/a[[:]atributos]] [//o[[:]orden-declasificación]] **[/s]** [/b] [/l]

PARAMETROS

[unidad:][ruta][nombrearchivo].

Especifica la unidad, directorio archivo o conjunto de archivos que se desea ver.

MODIFICADORES.

/p

Lista una pantalla de archivos en cada ocasión; pulsar cualquier tecla para ver la pantalla que sigue y así sucesivamente.

/W

Presenta el nombre de los archivos junto con su extensión en forma de tabla; cinco archivos por linea.

COMENTARIOS.

El comando **dir** con los modificadores **/p** y /w presenta la información de etiqueta, volumen y ruta en la parte superior de la pantalla de la siguiente forma :

El volumen en la unidad C: tiene etiqueta ALFREDO.

El número de serie del volumen es 2E18-iADB.

Directorio de C:\CHI .

Después de listar los archivos despliega el número de archivos en el directorio y los bytes disponibles :

57 archivos 1851392 bytes libres.

EJEMPLOS :

Supongamos que estamos trabajando en un subdirectorio que se llama TAREAS que se encuentra en la unidad A:.

Ejemplo 1. La orden dir sin modificadores se ejecuta de la siguiente manera y produce el listado :

A> **dir ENTER**

El volumen en la unidad A: tiene etiqueta LUIS_ALFREDO

El número de serie del volumen es 2E18-IADB

Directorio de A:\TAREAS

tarea01	pas	456	09-22-78	6.18p
tarea06	pas	1345	12-01-89	3.34
alumnos1	dbf	73783	08-26-93	4.18p
alumn601	dbf	2323	11-30-91	4.56p
tarea02	pas	4555	06-23-94	9.25
tarea07	pas	3774	01-23-94	12.00
alumnos2	dbf	623636	04-27-94	2.25p
alumn601	prg	63623	06-17-93	8.25
tarea03	pas	5235	03-26-94	9.22
tarea01	chi	12343	06-30-93	3.25p

alumnos3 dbf 633 08-12-92 4.18p

alumn601 ntx 34445 02-28-94 9.23

tarea04 pas 8387 09-13-92 12.05

tarea02 chi 52365 03-26-94 3.25p

alumnos4 dbf 4646 11-23-94 2.48p

alumn602 dbf 2343 10-25-89 9.46

17 archivos 5392 bytes libres.

Los dos primeros elementos corresponden al nombre y extensión del archivo (nótese que aparecen separados por espacios), el tercero indica el número de bytes del archivo, y los dos últimos representan la fecha y la hora respectivamente de su última modificación.

Ejemplo 2. Si queremos listar los archivos que se encuentran en la unidad B: en el subdirectorio HOLA, por pantalla tenemos que escribir La orden **dir** con el modificador **/p** de la siguiente forma y obtenemos el listado :

A> **dir** B:\hola **/p** **ENTER**

El volumen en la unidad B: tiene etiqueta FABIOLA

El número de serie del volumen es 2E18-IADB

Directorio de B:\HOLA

. <DIR> 08-11-92 3.24p

.. <DIR> 08-11-92 3.24p

tarea01 pas 456 09-22-78 6.18p

tarea06 pas 1345 12-01-89 3.34

alumnos1 dbf 73783 08-26-93 4.18p

alumn601 dbf 2323 11-30-91 4.56p

tarea02 pas 4555 06-23-94 9.25

tarea07 pas 3774 01-23-94 12.00

Presione cualquier tecla para continuar...

De esta forma se continúa hasta listar todos los archivos especificados.

Ejemplo 3. Para listar todos los archivos con extensión .txt en la unidad por defecto, escribimos:

A>DIR *.txt ENTER

DISKCOPY (Copiar disquete)

Copia el contenido del disquete de la unidad de origen a otro disquete formateado o no. La información contenida en el disquete destino es destruida durante el proceso de copiado. Sólo se puede copiar de unidades de disco flexible, no se puede copiar a disco duro y

	∈ DOS
NO	∈ LOTES
NO	∈ CONFIG.SYS
NO	∈ INTERNO
	∈ EXTERNO
NO	∈ RED

los discos tienen que ser idénticos en cuanto a tamaño y densidad de grabación

SINTAXIS.

DISKCOPY [unidad1:unidad2]] [/1][/v]

PARAMETROS.

unidad1:

Precisa la unidad del disquete de origen.

unidad2:

Precisa la unidad del disquete destino.

MODIFICADORES.

/1

Copia solamente la cara 0 del disquete.

/v

Verifica que la copia sea correcta.

Ejemplo 1. Se desea copiar la información de un disquete insertado en la unidad A:, el disquete destino se encuentra en la B:. La orden se escribe como :

A>**DISKCOPY** A: B: **ENTER**

El DOS despliega el siguiente mensaje :

Insert source disk in drive A:

Insert target disk in drive B:

Strike any key when ready

Es decir, indica que se inserte el disquete que contiene la información que se desea copiar en la unidad A:, el disquete destino en la B:; si éste no se encuentra formateado, lo formateará y a continuación procederá a copiar la información. Al terminar el proceso de copiado se desplegará el siguiente mensaje :

Copy another (Y/N)? (¿ Quiere seguir copiando (S/N) ?)

Si se pulsa Y (Si), el DOS repetirá el proceso de copiado.

Ejemplo 2. Se desea copiar el contenido de un disquete, sólo se dispone de una unidad de disco flexible. La orden puede ser cualquiera de estas tres :

A>DISKCOPY A: A: **ENTER**

A>DISKCOPY A: **ENTER**

A>DISKCOPY **ENTER**

En todas las órdenes la unidad por defecto es la A:. En la primera se especifica explícitamente la unidad destino como la A:; en la segunda orden, **DISKCOPY** utiliza la unidad actual como destino y en la tercera asume la unidad activa para ambos, es decir, para origen y destino de la información.

COMENTARIOS.

Se puede copiar en una o dos unidades; en caso de utilizar una sola unidad, se intercambian constantemente los disquetes de origen y destino previa notificación del DOS. Si el disquete destino contiene información ésta será

remplazada por la del disquete origen. Si el disquete destino es formateado en el proceso de la orden **DISKCOPY**, se produce una copia idéntica tanto en hardware como en software, es decir, cada cara del disquete destino es magnetizada con el mismo número de pistas y sectores que el original.

ERASE (Eliminar)

La orden **ERASE** (**DEL**) se usa para borrar archivo o grupo de archivos en la ruta especificada.

	∈ DOS
NO	∈ LOTES
NO	∈ CONFIG.SYS
	∈ INTERNO
NO	∈ EXTERNO
	∈ RED

SINTAXIS.

ERASE [unidad:][ruta]nombrearchivo

DEL [unidad:][ruta]nombrearchivo

COMENTARIOS

Es más común utilizar la orden **DEL** que la **ERASE**, ambas tienen la misma sintaxis. Si no se especifica la unidad, la acción de la orden se ejecuta en la unidad por defecto. Se pueden utilizar los caracteres comodines (?,*) para eliminar conjuntos de archivos.

Ejemplo 1. Se desea eliminar un archivo que se llama BASURA.TON, ubicado en la unidad D:, la unidad activa es la A:; la orden la escribimos como :

A>**DEL** D:basura.ton

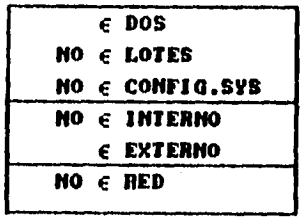
Si escribimos la orden sin especificar la unidad D:, el DOS buscará el archivo en la unidad activa (A:) al no encontrarlo mandará un mensaje indicando que el archivo no se encuentra.

Ejemplo 2. Se desean borrar todos los archivos de la unidad activa. Podemos proceder de dos formas: una es borrando archivo por archivo y la otra es con la ayuda del carácter comodín *.

a>**DEL *.***

El DOS contesta con una pregunta : ¿ está usted seguro (S/N)? (are you sure (Y/N) ?). Si se responde sí, el DOS procede a borrar todos los archivos en la unidad activa. Si no se trabaja con el disquete, es posible recuperar los archivos borrados.

FORMAT (Dar formato)



La orden FORMAT prepara un disco para su utilización posterior. Al dar formato a un disco el DOS reserva una pequeña parte para registrar la tabla de asignaciones y el directorio raíz.

SINTAXIS.

FORMAT [unidad:] [/S] [/1] [/8] [/V] [/B] [/f:tamaño]

PARAMETROS.

unidad:

Se refiere a la unidad en la que se encuentra el disquete o disco duro que va a ser formateado.

MODIFICADORES.

/S

Copia los archivos del sistema IO.SYS, DOS.SYS y COMMAND.COM; desde la unidad activa al disquete que se encuentre en la unidad especificada; si no los encuentra, entonces el DOS pide que se inserte un disquete que los contenga.

/1

Da formato a una sola cara del disquete.

/8

Formatea 8 sectores por pista un disco de 5 1/4 pulgadas, los cuales son compatibles con discos de versiones anteriores a la 2.0 del DOS.

/V

Determina la etiqueta del volumen, la que consta de once caracteres y sirve para identificar el disquete.

/B

Crea un disco con 8 sectores por pista y deja espacio para los archivos ocultos.

/f:tamaño

Formatea un disquete según el tamaño del mismo (tabla 3.9).

COMENTARIOS.

Si no se especifica la unidad, se formateará el disquete o disco duro que se encuentre en la unidad activa. El formateo se realizará dependiendo del tipo de unidad de la que se trate y tipo de disquete (ver tabla 3.9). El modificador /S crea un disquete con todos los archivos ocultos necesarios para inicializar el sistema. Los modificadores /8 y /V no son compatibles, es decir, se utiliza uno o el otro, pero no ambos. Si se omiten los modificadores /8 y /B el disco es formateado a 9 sectores por pista.

BYTES	TIPO DE DISCO
160 KB	DISQUETES DE 5 1/4 PULGADAS DE 160 KB, DE DOBLE DENSIDAD Y DE UNA SOLA CARA.
180 KB	DISQUETES DE 5 1/4 PULGADAS DE 180 KB, DE DOBLE DENSIDAD Y DE UNA SOLA CARA.
320 KB	DISQUETES DE 5 1/4 PULGADAS DE 320 KB, DE DOBLE DENSIDAD Y DE DOS CARAS.
360 KB	DISQUETES DE 5 1/4 PULGADAS DE 360 KB, DE DOBLE DENSIDAD Y DE DOS CARAS.
720 KB	DISQUETES DE 3 1/2 PULGADAS DE 720 KB, DE DOBLE DENSIDAD Y DE DOS CARAS.
1.2 MB	DISQUETES DE 5 1/4 PULGADAS DE 1.2 MB DE CUADRUPLE DENSIDAD Y DE DOS CARAS.
1.4 MB	DISQUETES DE 3 1/2 PULGADAS DE 1.4 MB,DE CUADRUPLE DENSIDAD Y DE DOS CARAS.
2.88 MB	DISQUETES DE 3 1/2 PULGADAS DE 2.88 MB, DE CUADRUPLE DENSIDAD Y DE DOS CARAS.

TABLA 3.9 : TIPO DE DISQUETES.

Ejemplo 1. Se desea formatear un disco nuevo. Se cuenta con dos unidades de disco. Tenemos que escribir :

A>**FORMAT** B:

Los disquetes formateados sin sistema operativo usualmente se utilizan como discos de trabajo.

Ejemplo 2. Si se desea tener un disco para inicializar el sistema, tenemos que escribir :

A>**FORMAT** B: /S

A>**FORMAT** B: /S /V

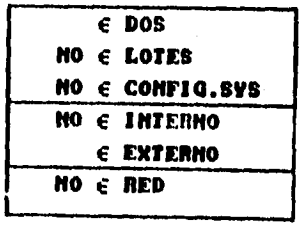
Con ambas opciones transferimos los archivos del sistema IO.SYS, DOS.SYS y EL COMMAND.COM. La segunda opción nos permite asignarle volumen al disquete, para su identificación.

Ejemplo 3. Queremos formatear un disco a 360 Kb. Disponemos de una máquina con procesador 80286 o mayor. entonces tenemos que escribir :

A>**FORMAT** B: /f:360kb

Si no se utiliza el modificador /f:tamaño es posible formatear un disquete de doble densidad y doble cara. Pero tendremos como resultado un disquete con muchos sectores dañados.

LABEL (Etiqueta)



La orden **LABEL** permite borrar, crear o modificar una etiqueta de volumen en un disco

SINTAXIS.

LABEL [unidad:][etiqueta]

PARAMETROS.

unidad:

Determina la unidad en la que se encuentra el disco que se le quiere asignar la etiqueta de volumen.

etiqueta

Especifica el nombre de la nueva etiqueta de volumen

COMENTARIOS.

Para el nombre para etiqueta de volumen se pueden utilizar once caracteres y se pueden aplicar las mismas reglas de nombre de archivo. Si no se especifica la unidad, se modificará la etiqueta de volumen del disco que se encuentre en la unidad activa. La etiqueta de volumen es la misma que se asigna con la opción */V* en la orden **FORMAT**

Ejemplo 1. Se quiere modificar el nombre de la etiqueta de volumen de un disquete que se encuentra en la unidad B:. Escribimos la orden LABEL de las siguientes formas :

LABEL B:

LABEL B:NUEVOVOLU

En la primera opción el DOS mandará el siguiente mensaje :

Volume in drive B: is HOLA

Volume label (11 characters, ENTER for none?)

Aquí se debe teclear el nombre deseado y pulsar ENTER. Si se pulsa ENTER sin escribir la etiqueta, se recibirá el mensaje :

Delete current volume label (Y/N)?

Es decir, pregunta si se quiere eliminar la etiqueta actual; si se pulsa (Y), entonces se borra. Con la segunda opción se transfiere el nombre NUEVOVOLU para etiqueta de volumen en forma automática.

MORE (más)

El comando **MORE** despliega un archivo de texto pantalla por pantalla.

	∈ DOS
NO	∈ LOTES
NO	∈ CONFIG.SYS
NO	∈ INTERNO
	∈ EXTERNO
NO	∈ NED

SINTAXIS.

a) **MORE** < [unidad:][ruta]nombrearchivo

b) nombre-comando nombrearchivo | **MORE**

PARAMETROS.

[unidad:][ruta]nombrearchivo

Indica la dirección y el nombre del archivo de texto que será desplegado por pantallas.

nombre-comando nombrearchivo

Especifica el nombre de un comando (**DIR, SORT o TYPE**) y el nombre del archivo de texto que se presentará pantalla por pantalla.

COMENTARIOS.

Los símbolos "<" y "|" se les conoce como caracteres de redireccionamiento y canalización, respectivamente. Con la redirección, en lugar de transmitir la información al dispositivo estándar de salida, se orienta hacia otro. EL carácter de canalización (|) se utiliza en la redirección, siempre que la orden **MORE** se utilice con otro comando. Para poder usar el símbolo de canalización, es necesario declarar la variable de ambiente "**TEMP**" 0 en el archivo AUTOEXEC.BAT. El comando **TYPE** se describe posteriormente.

Ejemplo. Se quiere ver el contenido del archivo de texto largo.pas, podemos escribir :

MORE < LARGO.PAS

TYPE LARGO.PAS | **MORE**

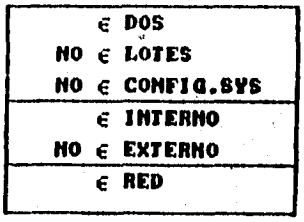
El comando MORE desplegará la primera pantalla y presentará el mensaje :

-- Más --

Entonces, pulsando cualquier tecla (menos PAUSE) continuará desplegando el contenido del archivo pantalla por pantalla.

RENAME (Renombrar)

Es más común utilizar el comando **RENAME** como **REN**. Se utiliza para renombrar uno o más archivos. Con **REN** no se puede cambiar el nombre de un subdirectorio.



SINTAXIS.

REN [unidad:][ruta] nombrearchivo1 nombrearchivo2

PARAMETROS.

[unidad:][ruta] nombrearchivo1

Indica la dirección del archivo que se va a renombrar.

nombrearchivo2

Especifica el nuevo nombre del archivo.

COMENTARIOS.

La orden **REN** se puede utilizar con los caracteres comodín * y ? para renombrar más de un archivo en la unidad activa; Los caracteres representados por los comodines en nombrearchivo1 serán asignados a nombrearchivo2.

Ejemplo 1. Se desea renombrar el archivo tarea01.pas por area.pas; en la unidad activa A:.

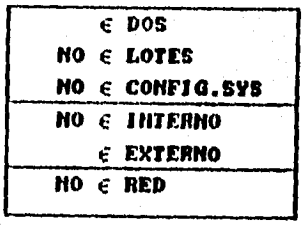
A>**REN** tarea01.pas area.pas

Ejemplo 2. Para cambiar la extensión BAK de todos los archivos en la unidad activa por la extensión RES, escribimos :.

A>REN *.bak *.res

SYS (Sistema)

El comando **SYS** permite transferir los archivos del sistema (IO.SYS y DOS.SYS) así como el COMMAND.COM



SINTAXIS.

SYS [unidad1:][ruta] unidad2:

PARAMETROS.

[unidad1:][ruta]

Indica la dirección en la que se localizan los archivos del sistema. Si no se especifica la dirección el DOS buscará en el directorio raíz de la unidad activa.

unidad2:

Se refiere a la unidad en la que se encuentra el disquete (disco) al que se le transferirán los archivos del sistema.

COMENTARIOS.

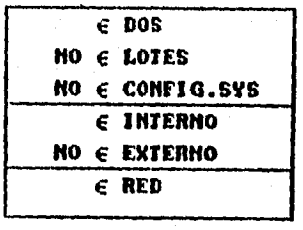
El disquete (disco) destino se debió formatear con cualquiera de los modificadores /S o /B; esto es, hay que asegurar la existencia de espacio disponible en el disquete, para los archivos del sistema.

Ejemplo 1. Se desea transferir los archivos del sistema al disco duro, donde la unidad activa es A:.

A>**SYS** C:

TIME (Hora)

El comando **TIME** permite modificar la hora del sistema o ajustar el reloj de la computadora.



SINTAXIS.

TIME [hh:mm:ss.xx]

PARAMETROS.

hh

Especifica la hora; los valores permitidos se encuentran entre 0 y 23 inclusive.

mm

Representa los minutos; los valores permitidos se encuentran entre 0 y 59 inclusive.

ss

Denota los segundos; los valores permitidos se encuentran entre 0 y 59 inclusive.

xx

Especifica las centésimas de segundo; los valores permitidos se encuentran entre 0 y 99 inclusive.

COMENTARIOS.

Es importante tener actualizada la hora del sistema y del reloj, pues cada vez que se modifica un archivo se registra la hora y de esta forma se puede tener información sobre las alteraciones que sufren los archivos. En algunas aplicaciones es importante registrar la hora exacta en la que se presenta un suceso, por ejemplo la ocurrencia de un temblor. Las computadoras con reloj integrado de fábrica almacenan la hora actual; las máquinas que no lo incluyen despliegan el siguiente mensaje al arrancar el sistema :

Current time is 00:00:12.34

Enter new time:

Es decir, la hora actual es 00:00:12.34, introduzca la nueva hora : , si se pulsa ENTER se deja la hora sin modificaciones.

Ejemplo 1. Si se desea actualizar la hora actual, entonces tenemos que escribir :.

A>TIME

Current time is 00:00:12.34

Enter new time:22:36

Al pulsar ENTER se registra la hora.

€	DOS
NO €	LOTES
NO €	CONFIG.SYS
€	INTERNO
NO €	EXTERNO
€	RED

TYPE (Escribir)

El comando **TYPE** muestra el contenido de un archivo de texto en pantalla sin pausa automática. El contenido del archivo no se puede modificar.

SINTAXIS.

TYPE [unidad:][ruta]nombrearchivo

PARAMETRO.

[unidad:][ruta]nombrearchivo

Indica la posición y el nombre del activo que será desplegado en el monitor.

COMENTARIOS.

Los archivos con extensión .EXE, .COM, es decir, los archivos ejecutables no se pueden visualizar con la orden **TYPE**; pues al ejecutar la orden, se presentan caracteres incomprensibles.

Ejemplo. Se desea ver el contenido del archivo de texto AREA.PAS, localizado en la unidad B:, la unidad activa es la C:. Se tiene que proceder de la siguiente manera.

C>**TYPE** B:AREA.PAS

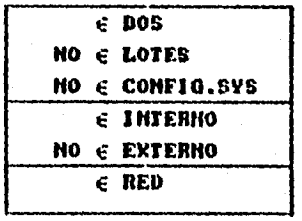
Si el archivo es demasiado extenso se puede utilizar la tecla de PAUSA, para ver el archivo. Otra forma es utilizar la orden **MORE**, con la cual tenemos que escribir :

C>**TYPE** B:AREA.PAS | **MORE**

El comando escrito de esta forma nos permite ver el contenido de un archivo de texto por pantalla.

VER (Versión)

El comando **VER** presenta la versión del sistema operativo DOS que se encuentra presente en memoria.



SINTAXIS.

VER

Ejemplo. Si se desea ver la versión del DOS activa simplemente escriba :

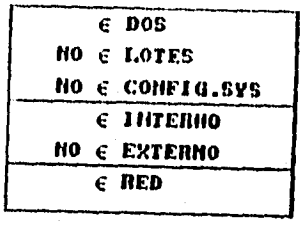
VER

Se presentará en pantalla el siguiente mensaje :

XX-DOS, versión 3.1

Donde XX representa la marca del fabricante.

VOL (Volumen)



El comando **VOL** presenta la etiqueta de volumen y su número de serie si es que existen.

SINTAXIS.

VOL [unidad:]

PARAMETROS.

unidad:

Representa la unidad en la que se encuentra el disco al que se le quiere observar la etiqueta de volumen.

COMENTARIOS.

Si se desea consultar la etiqueta de volumen del disco de la unidad activa no es necesario escribir el parámetro unidad:.

Ejemplo. Se quiere observar la etiqueta de volumen de un disco que se encuentra en la unidad B:, siendo la unidad activa la A:, tenemos que escribir:

A>**VOL** B:

EL DOS desplegará cualquiera de estos mensajes, dependiendo si el disco tiene o no etiqueta de volumen.

Volume in drive B: is FABIOLA

Volume in drive B: has no label.

Un aspecto importante para el conocimiento y uso eficiente de los comandos del DOS es la práctica correcta y constante de ellos.

UNIDAD CUATRO.

PROCESADOR DE TEXTOS.

4.1.- EDITOR DE TURBO PASCAL.

Un aspecto importante en el aprendizaje de la computación es recorrer el velo mágico que lo cubre, para esto es necesario que el alumno interactúe activamente con el objeto de conocimiento.

Al introducir en la computadora un texto diseñado por el mismo, el alumno se dará cuenta que la computadora es un instrumento más de trabajo diseñado para archivar y procesar la información. La idea de enseñar un editor de texto es con el fin de que el alumno le pierda el miedo a la computadora, es decir, que conozca el teclado, las unidades de disco, que se compenetre con la computadora y posteriormente codifique la solución de problemas en algún lenguaje de programación.

El editor de texto que se describirá es el de Turbo Pascal versión 3.0 el cual consta del editor y un compilador para correr los programas codificadas en lenguaje Pascal versión Turbo. El editor contiene una serie de comandos que nos permiten manipular y editar textos, con la combinación de dos o más teclas; por ejemplo, para salir del editor de Turbo Pascal y regresar al menú principal, la secuencia de teclas es CTRL-KD, es decir, hay que presionar al mismo tiempo las teclas de CTRL y K, a continuación pulsar D. En la tabla 4.1 se pueden ver las teclas de órdenes por defecto, incluyendo las teclas opcionales para IBM PC.

ACCION.	COMBINACION DE TECLAS DE CONTROL.	TECLAS OPCIONALES PARA IBM PC.
CONTROL DEL CURSOR.		
UN CARACTER A LA IZQUIERDA.	CTRL-S	← FLECHA IZQUIERDA.
UN CARACTER A LA DERECHA.	CTRL-D	→ FLECHA DERECHA.
SUBIR UNA LINEA.	CTRL-E	▲ FLECHA HACIA ARRIBA.
BAJAR UNA LINEA.	CTRL-X	▼ FLECHA HACIA ABAJO.
SUBIR UNA PAGINA DE PANTALLA.	CTRL-R	PAGE UP.
BAJAR UNA PAGINA DE PANTALLA.	CTRL-C	PAGE DOWN.
UNA PALABRA A LA IZQUIERDA.	CTRL-A	CTRL-FLECHA IZQUIERDA.
UNA PALABRA A LA DERECHA.	CTRL-F	CTRL-FLECHA DERECHA.
CORRER HACIA ARRIBA.	CTRL-W	NINGUNA.
CORRER HACIA ABAJO.	CTRL-Z	NINGUNA.
PRINCIPIO DE LINEA.	CTRL-QS	HOME.
FIN DE LINEA.	CTRL-QD	END.
PRINCIPIO DE LA PANTALLA.	CTRL-QE	CTRL-HOME.
FINAL DE LA PANTALLA.	CTRL-QX	CTRL-END.
PRINCIPIO DE ARCHIVO.	CTRL-QR	CTRL-PAGE UP.
FINAL DEL ARCHIVO.	CTRL-QC	CTRL-PAGE DOWN.
COMIENZO DE BLOQUE.	CTRL-QB	NINGUNA.
FIN DE BLOQUE.	CTRL-QK	NINGUNA.
RESTAURAR LA POSICION DEL CURSOR.	CTRL-QP	NINGUNA.
TABULAR.	CTRL-I	TAB.
AUTO TABULAR ON/OFF	CTRL-OI	NINGUNA.
INSERTAR O SUPRIMIR.		
INSERTAR ON/OFF.	CTRL-V	INS.
ESPACIO ATRAS Y SUPRIMIR.	BACKSPACE	BACKSPACE
SUPRIMIR CARACTER.	CTRL-G	DEL
SUPRIMIR PALABRA.	CTRL-T	NINGUNA
INSERTAR LINEA.	CTRL-N	NINGUNA
SUPRIMIR LINEA.	CTRL-Y	NINGUNA
SUPRIMIR HASTA FIN DE LINEA.	CTRL-QY	NINGUNA

TABLA 4.1 : TECLAS DE ORDENES DEL EDITOR POR DEFECTO EN TURBO PASCAL.

(CONTINUA SIGUIENTE PAGINA)

A C C I O N .	COMBINACION DE TECLAS DE CONTROL.	TECLAS OPCIONALES PARA IBM PC.
I N S E R T A R O B L O Q U E .		
MARCAR COMIENZO DE BLOQUE.	CTRL-KB	F7
MARCAR FINAL DE BLOQUE.	CTRL-KK	F8
MARCAR UNA PALABRA.	CTRL-KT	NINGUNA
OCULTAR/PRESENTAR BLOQUE.	CTRL-KH	NINGUNA
COPIAR BLOQUE MARCADO.	CTRL-KC	NINGUNA
MOVER BLOQUE MARCADO.	CTRL-KV	NINGUNA
LEER BLOQUE DE UN ARCHIVO.	CTRL-KR	NINGUNA
ESCRIBIR BLOQUE EN UN ARCHIVO.	CTRL-KW	NINGUNA
B U S C A R Y R E E M P L A Z A R .		
ENCONTRAR CADENA DE BUSQUEDA.	CTRL-QF	NINGUNA
REEMPLAZAR CADENA DE BUSQUEDA.	CTRL-QA	NINGUNA
REPETIR ENCONTRAR/REEMPLAZAR.	CTRL-L	NINGUNA
V A R I O S .		
INSERTAR CARACTER DE CONTROL (X)	CTRL-X	NINGUNA
ABORTAR ORDEN.	CTRL-U	NINGUNA
RESTAURAR LINEA.	CTRL-QL	NINGUNA
FIN DE EDICION.	CTRL-KD	NINGUNA

TABLA 4.1 : TECLAS DE ORDENES DEL EDITOR POR DEFECTO EN TURBO PASCAL.

Para utilizar el editor de Turbo Pascal se necesitan los archivos TURBO.COM y TURBO.MSG, este último no es necesario para utilizar el editor de Turbo Pascal para escribir textos, pero sí es útil para escribir programas en Pascal. Para iniciar una sesión con el editor podemos seguir los siguientes pasos :

1.- Inicializado el sistema, escribir TURBO y pulsar ENTER.

2.- El sistema Turbo despliega :

TURBO PASCAL system Version n.nn

PC-DOS

Copyright (C) 1983,84,85 BORLAND Inc.

Include error messages (Y/N) ?

Es decir (Cargar el mensajero de errores (S/N) ?

Si se pulsa Y (Si), el archivo TURBO.MSG se cargará en memoria y los errores de programación se referenciarán con mensaje descriptivo en lugar de un número.

3.- El menú principal de Turbo Pascal se puede ver en el esquema 4.2, el cual varía de una versión a otra.

Logged drive: C

Active directory: \PASCAL\TRABAJO

Work file:

Main file:

Edit Compile Run Save

Dir Quit compiler Options

Text : 0 bytes

Free : 62024 bytes

Esquema 4.2 : Menú principal de Turbo Pascal.

A continuación se explicará el funcionamiento de los comandos. Para invocar los comandos basta con pulsar la letra mayúscula de cada uno.

a) **Logged drive : C:**

Indica la unidad activa y permite modificarla. Al pulsar "**L**" el sistema despliega :

New drive:

Se especifica la unidad deseada; la que puede ser cualquiera de las incluidas en el equipo de cómputo, es decir, letras mayúsculas o minúsculas de la "A" a la "P".

b) **Active directory : \PASCAL\TRABAJO**

Especifica la dirección de trabajo activa; al pulsar "**A**" se despliega :

New directory:

Nos está solicitando la nueva dirección de trabajo, para lo cual hay que especificar la unidad y la ruta correspondiente.

c) **Work file :**

Especifica el nombre del archivo de trabajo, el cual debe respetar la regla descrita para nombre de archivo. Si no se especifica la extensión el sistema Turbo Pascal asume la extensión "PAS"; no se pueden utilizar las extensiones : .BAK. CHN .CMD y .COM pues tienen un significado especial en Pascal. Al pulsar "**W**" se despliega en el monitor :

Work file name :

Es decir, nos solicita el nombre del archivo de trabajo. Si el archivo existe, entonces lo carga en memoria, y si no, crea el archivo. Si existe un archivo en memoria, el sistema pregunta si se quiere salvar o no, y a continuación pregunta por el nombre del nuevo archivo; por ejemplo :

workfile c:\pascal\tarea.pas not saved. saved(Y/N)

Work file name :

d) **Main file :**

Dado que el sistema Pascal trabaja los programas fuente y objeto la mayor parte del tiempo en memoria RAM, si los archivos son muy extensos y la memoria RAM es pequeña, entonces es conveniente hacer programas más cortos enlazados todos ellos por un programa principal (Main file). Para más detalle consultar un manual de Turbo Pascal.

e) **Edit :**

Despliega el editor de Turbo Pascal en el monitor. Si no hay archivo activo primero despliega el comando Work file y a continuación presenta el editor (esquema 4.3). El que está conformado con una línea de estado constituida por :

e.1) **Line n :**

Denota la linea en la que se encuentra posesionado el cursor.

e.2) **Col n :**

Especifica la columna en la que se encuentra el cursor.

e.3) **Insert :**

Se encuentra el editor en modo de inserción, es decir, al escribir un carácter se desplaza hacia la derecha lo que se encuentre escrito; al presionar enter se salta a una nueva línea. El otro modo es "Overwrite", el cual escribe sin desplazar; al presionar Enter, el cursor se posesiona al principio de la misma línea.

e.4) **Indent :**

Las líneas se escriben en formato indentado, es decir, al presionar Enter el cursor se desplaza en la nueva línea a la columna que corresponde al principio del texto de la línea anterior.

e.5) **A:Nombre_archivo.ext :**

Denota la dirección y el nombre del archivo activo.

Line n Col n Insert Indent A:Nombre_archivo.ext

Esquema : 4.3. Línea de estado del editor.

f) **Compile :**

Compila el archivo de trabajo, es decir, transfiere el programa fuente en programa objeto. Si no hay archivo activo primero ejecuta el comando work file.

g) **Run :**

Corre el archivo activo, si no se encuentra compilado primero lo compila y después lo corre. Si no existe archivo de trabajo activo, ejecuta primero Work file y después corre el programa.

h) **Save :**

Salva el archivo activo. Si no existe archivo activo, al ejecutar el comando "Save" el sistema Turbo Pascal ejecuta las siguientes instrucciones :

Work file name : nombre_archivo [Enter]

Loading A:nombre_archivo.pas

Saving : A:nombre_archivo.pas

Donde nombre_archivo es un nombre de archivo suministrado por el usuario. Si existe un archivo anterior con el mismo nombre y extensión, deja el nombre del archivo y la extensión la renombra con .BAK.

i) **Dir :**

Despliega en pantalla los archivos existentes en la unidad activa en forma tabular. Soporta el uso de los caracteres comodin "*" , "?" y la especificación de la ruta de búsqueda.

j) **Quit :**

Permite salir del sistema Turbo Pascal y regresar al DOS. Si hay un archivo activo que no ha sido salvado pregunta si se quiere salvar o no antes de abandonar Turbo Pascal.

k) **compiler Options :**

Permite modificar la forma de compilar archivos; para mayor referencia consultar un manual de Turbo Pascal.

CAPITULO CINCO

TECNICAS PARA RESOLVER PROBLEMAS CON COMPUTADORA.

5.1 METODOLOGIA DE WARNIER-ORR.

Los problemas clásicos en el CCHA se limitan al algoritmo de la ecuación. de segundo grado, en el cual es muy fácil elaborar un diagrama de flujo. En este punto, lo importante es que el alumno comprenda que para poder hacer un programa de un problema específico, él lo puede resolver; entonces es necesario enseñarle métodos para que estructure su solución y de esta forma realizar el programa correspondiente; la técnica de los diagramas de flujo no es la mejor, en infinidad de casos en lugar de ser una solución complica ésta y la mayoría de las veces se pierde el control del programa, es decir, no se puede identificar fácilmente la secuencia lógica de un conjunto de instrucciones; recomendaría los diagramas de flujo para los ciclos básicos (if ... then, case, while y repeat) por su forma esquemática es de fácil comprensión. Es más recomendable la técnica de pseudocódigo, la cual conduce a la programación directa; también tiene sus limitaciones pues en ocasiones no permite inmediatamente la detección de errores de lógica; sería deseable establecer una metodología que permitiera escribir los programas en forma estructurada y que evitara los fatales errores de lógica.

Antes de describir la técnica, definiré algunos conceptos básicos, los cuales son importantes para comprender dicha metodología.

Cadena :

Secuencia de caracteres encerrado entre comillas (").

Conjunto :

Colección de objetos agrupados; en los cuales **SI** importa el orden. (En teoría de conjuntos no importa el orden.)

Constante :

Identificador (dato) cuyo valor no se modifica durante el proceso.

Dato :

Información necesaria, susceptible de procesar. Los datos pueden ser tanto constantes como variables. Los datos se relacionan como :

Repetición :

Datos de un mismo tipo, que se repiten; por ejemplo el número de cuenta (folio) de un alumno (empleado).

Secuencia :

Determina el orden en que se presentan los datos en el reporte.

Selección :

Se elige un conjunto u otro, pero no ambos.

Encabezado :

Parte superior de un reporte.

Problema :

Cuestión que se trata de resolver por medio de procedimientos científicos.

Reporte :

Informe de salida de un proceso.

Resultado :

Consecuencia de un proceso.

Variable :

Nombre genérico de un dato, que toma diferentes valores durante un proceso.

Var. Independiente (primaria) :

Son los datos proporcionados por el usuario.

Var. Dependiente (secundaria) :

Son los valores que se obtienen en un proceso operativo.

Refinamiento :

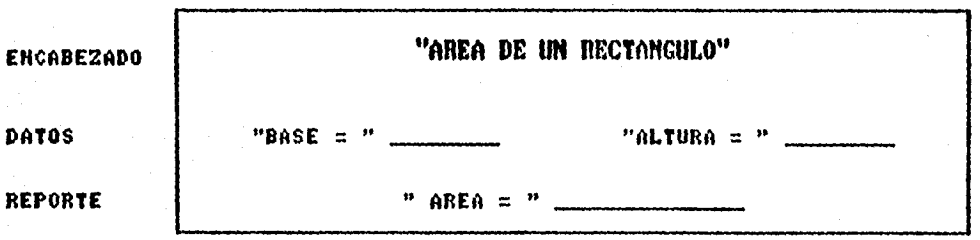
Eliminación de entes no necesarios en un proceso.

PROBLEMA 1 : OBTENER EL AREA DE UN RECTANGULO.

El método desarrollado por Warnier-Orr nos permite escribir la solución de un problema en una forma estructurada y lógica, en el cual se parte de atrás hacia adelante, es decir, del reporte hacia los datos. Para ilustrar la técnica utilizaré el ejemplo de cómo obtener el área de un rectángulo. (El reporte lo podemos ver en el esquema 5.1).

En la hoja de reporte se puede ver que está compuesta básicamente de tres partes: el encabezado formado por la cadena "AREA DE UN RECTANGULO", los datos

compuestos por las cadenas "base" , "altura" y dos líneas punteadas, y finalmente el resultado, el cual contiene la cadena "área =" y una línea punteada. Las líneas punteadas corresponden o a datos suministrados por el usuario o a datos obtenidos en un proceso de cálculo.



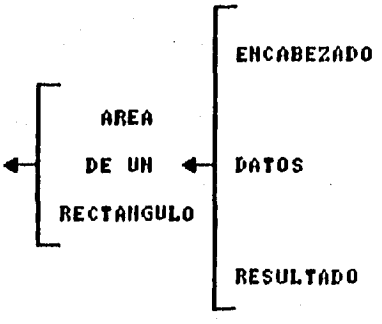
ESQUEMA 5.1 : HOJA DE REPORTE DEL AREA DE UN RECTANGULO.

La hoja de reporte la vamos a considerar como un conjunto de datos; por tanto, en concordancia con la notación de conjunto, escribimos (esquema 5.2) :

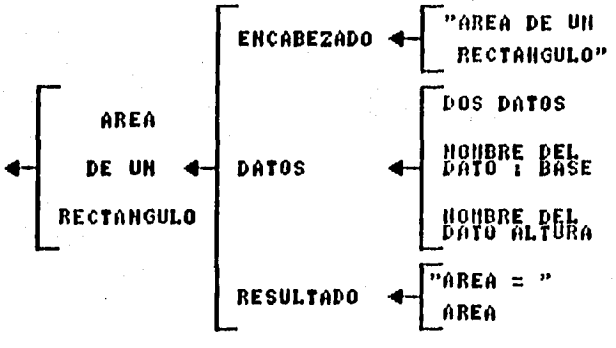
. El cierre de llave derecho no es necesaria, la eliminamos. Nos encontramos en el nivel superior de detalle, pues es donde menos especificaciones se efectúan. Los siguientes niveles los escribiremos a la derecha, abriendo la respectiva llave, para indicar al subconjunto al que pertenece la secuencia. Observando la hoja de reporte se puede especificar el siguiente nivel (encabezado, datos y resultado), en el cual hay que respetar la secuencia (el orden), esto es, lo escribiremos verticalmente (esquema 5.3).

En este nivel, respetando la secuencia natural que nos da el reporte, lo interpretamos como : primero va el encabezado seguido de los datos y por último el resultado.

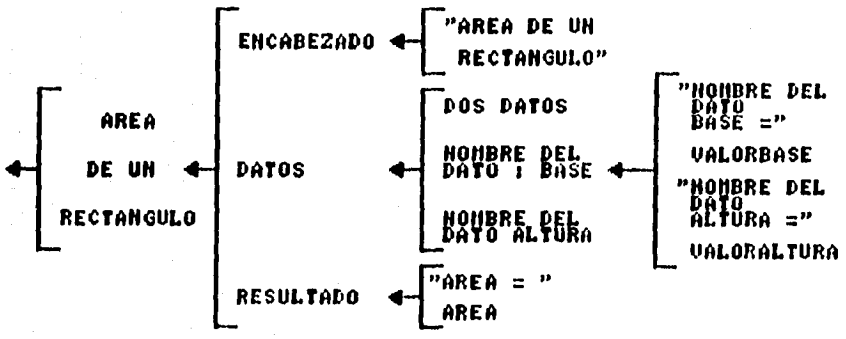
ESQUEMA 5.2 : HOJA DE REPORTE COMO CONJUNTO DE DATOS.



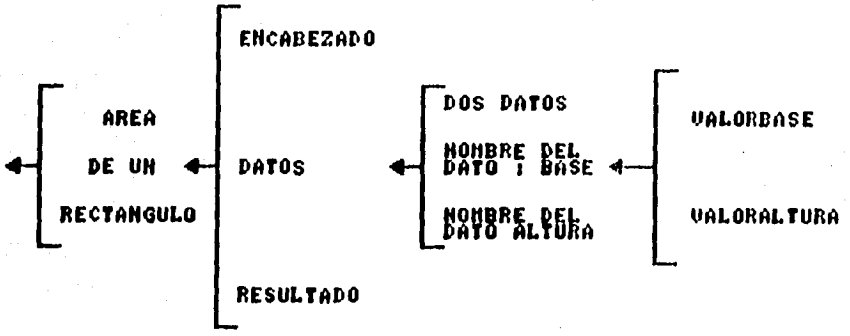
ESQUEMA 5.3 : NIVELES DE LA HOJA DE REPORTE.



ESQUEMA 5.4 : SUBNIVELES DE LA HOJA DE REPORTE.



ESQUEMA 5.5 : ESQUEMA LOGICO DE DATOS.



ESQUEMA 5.6 : ESQUEMA LOGICO DE ENTRADA.

En el siguiente nivel se desglosa lo que se pueda o se escribe la cadena correspondiente; todo con su respectiva llave para especificar a qué subconjunto pertenece (esquema 5.4).

Con este procedimiento se continúa hasta que cada subconjunto queda completamente detallado; en el esquema 5.5 podemos ver lo que se conoce como el " **Esquema lógico de datos (o del reporte)".**

El siguiente paso es generar el esquema lógico de la entrada de datos primarios (los datos suministrados por el usuario), para esto hay que proceder eliminado las cadenas, las constantes, las variables secundarias; aquí de lo que se trata es de que únicamente queden las variables primarias; si existe repetición de datos independientes, nada más se denotan hasta dos veces.

Efectuándo la depuración anterior obtenemos (esquema 5.6) :

El esquema lógico de entrada de datos indica que los únicos datos que tiene que suministrar el usuario son el valor de la base (valorbase) y de la altura (valoraltura), los cuales son las variables independientes.

Para obtener el esquema lógico del proceso, procedemos de la siguiente manera :

a) Consideramos el esquema lógico del reporte.

b) En cada llave escribimos en la parte superior la palabra principio y en la inferior final.

c) Anotar las palabras obtener y escribir de la siguiente manera :

Obtener se apunta donde van las variables de datos primarios (los datos suministrados por el usuario).

Escribir se anota donde van las cadenas, constantes y variables secundarias (variables dependientes).

Efectuando estas indicaciones obtenemos una versión preliminar del esquema lógico de proceso (esquema 5.7).

Para tener a punto el esquema lógico de proceso (solución) es necesario efectuar las siguientes consideraciones :

a) Proceder de la llave más a la derecha (de mayor refinamiento) a la de la izquierda una por una.

b) Del **final** hacia el **principio**.

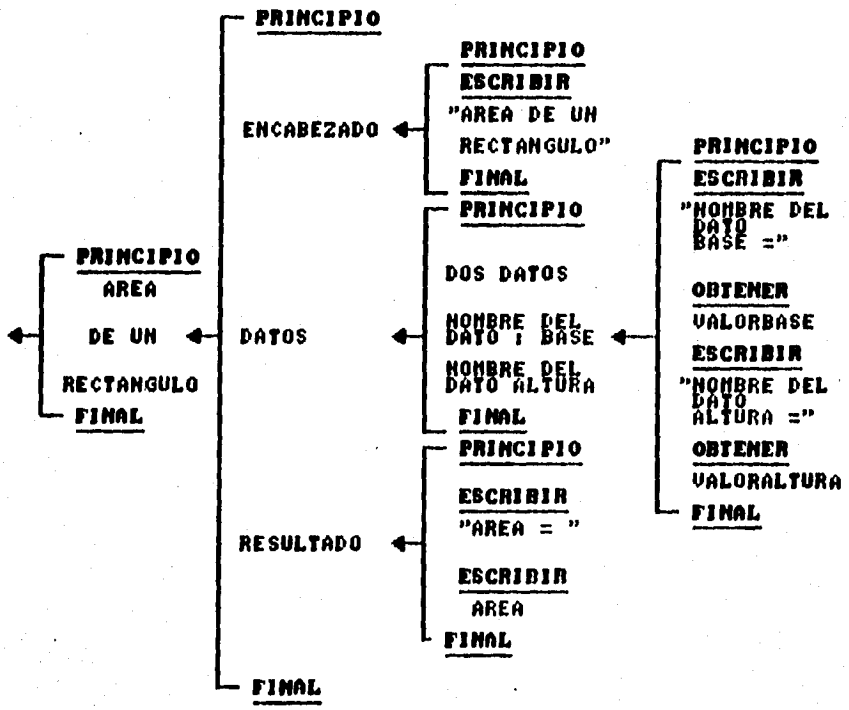
c) En el **final** anotar las consideraciones necesarias del proceso que termina.

d) En el **principio** especificar los requerimientos necesarios para el desarrollo del proceso.

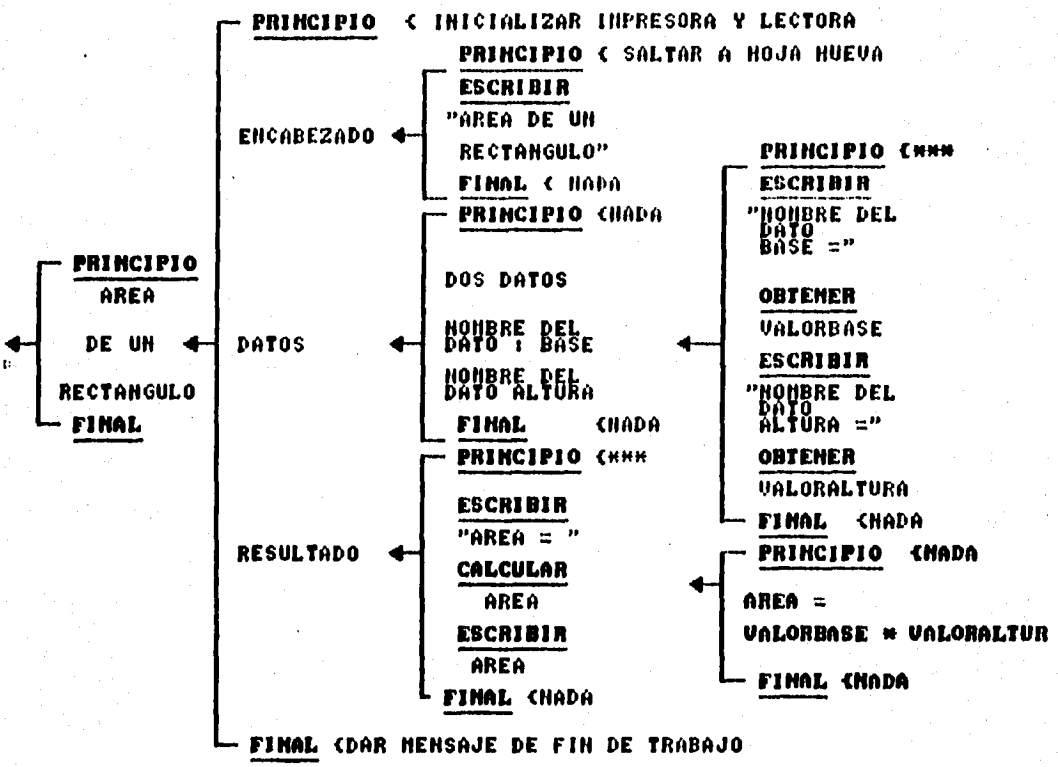
e) En las variables dependientes (secundarias) apuntar **Calcular** antes que la palabra **escribir**, abrir la llave correspondiente para detallar el proceso de "calcular"

f) Continuar con el método de Warnier-Orr, en el momento oportuno, es decir, no adelantar ni retrasar pasos, sino cuando la metodologia lo requiera.

Aplicando estas consideraciones obtenemos el esquema lógico del proceso (esquema 5.8).



ESQUEMA 5.7 : VERSION PRELIMINAR DEL ESQUEMA LOGICO DE PROCESO.



ESQUEMA 5.8 : ESQUEMA LOGICO DEL PROCESO.

Partiendo del esquema lógico del proceso se obtiene el "programa"; para esto, lo recorremos de derecha a izquierda, del principio hacia el final, considerando cada subconjunto y anotando los comentarios que aparecen en las palabras subrayadas.

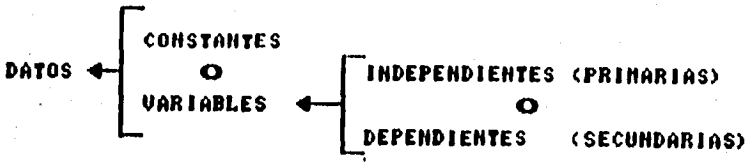
Aplicando lo anterior obtenemos el "programa" :

```
Inicializar impresora y lectora
saltar a hoja nueva
escribir "Area de un rectángulo"
saltar dos renglones
escribir "nombre del dato base"
obtener valorbase
escribir Nombre del dato altura
obtener valoraltura
saltar dos renglones
escribir "área ="
calcular área = valorbase * valoraltura
escribir área
Fin de trabajo
```

El "programa" que se obtuvo se encuentra escrito en forma estructurada y lógica, lo cual es un principio necesario para la buena programación.

Dada la naturaleza del ejemplo se omitieron algunos puntos importantes de la metodología de Warnier-Orr, los cuales son descritos a continuación.

a) Los datos pueden ser constantes o variables, éstas pueden ser primarias (independientes) o secundarias (dependientes); siguiendo la metodología de Warnier-Orr los datos los escribimos :



DATOS

Donde **O** representa el operador O (OR en inglés) exclusivo, el cual significa : o se elige la primera opción o la segunda pero, no ambas; por ejemplo :

O Luis Alfredo tiene ocho años o tiene doce años

Aquí lo que se quiere decir O Luis Alfredo tiene una edad o tiene la otra, pero no ambas al mismo tiempo.

b) Las repeticiones se denotan con paréntesis, debajo del objeto que las define, indicando el tipo de repetición :

calcula el área

(hasta fin de archivo de datos)

reporta

(mientras número sea menor que 99)

escribe

(desde inicio , hasta final)

TIPO DE DATOS INVOLUCRADOS EN UN PROCESO.

Para redondear el "programa" obtenido según la metodología de Warnier-Orr, es necesario especificar la naturaleza de las variables primarias y secundarias;

así como las restricciones de las mismas y el tipo de operaciones que se pueden realizar.

La naturaleza de los datos se refiere al tipo de datos que son, es decir, si son variables numéricas (enteras o reales), o alfabéticas.

Las restricciones de los datos se refiere, en el caso de variables numéricas, a los valores que pueden tomar, es decir, si son mayores que cero, que puede ser cualquier número, etc.

La naturaleza de los datos, sus restricciones y el tipo de operaciones realizables es lo que se conoce como estructura de datos del "problema".

En la solución de un problema es importante especificar la estructura de datos relacionada con dicho "problema".

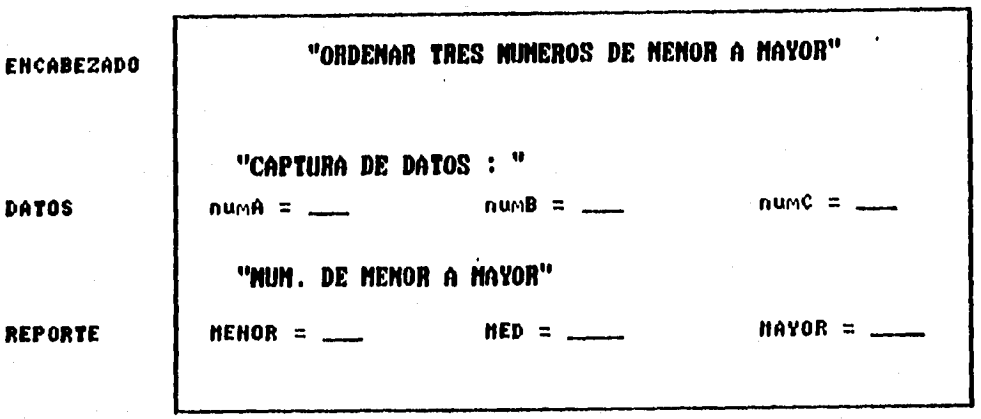
Por ejemplo en la solución del área de un rectángulo la estructura de datos es :

valorbase, valoraltura y área son números reales mayores que cero.

PROBLEMA 2 : ORDENAR TRES NUMEROS DE MENOR A MAYOR

Los ejemplos que se desarrollen en esta sesión tienen dos finalidades; la primera es aprender la metodología de Warnier-Orr y la segunda es dar los elementos necesarios para aprender las bases de la programación vía Turbo Pascal.

Con el problema 1 se pretende aprender las partes que componen un programa en Pascal, la estructura de datos y diferentes tipos de sentencias (proposiciones y enunciados).



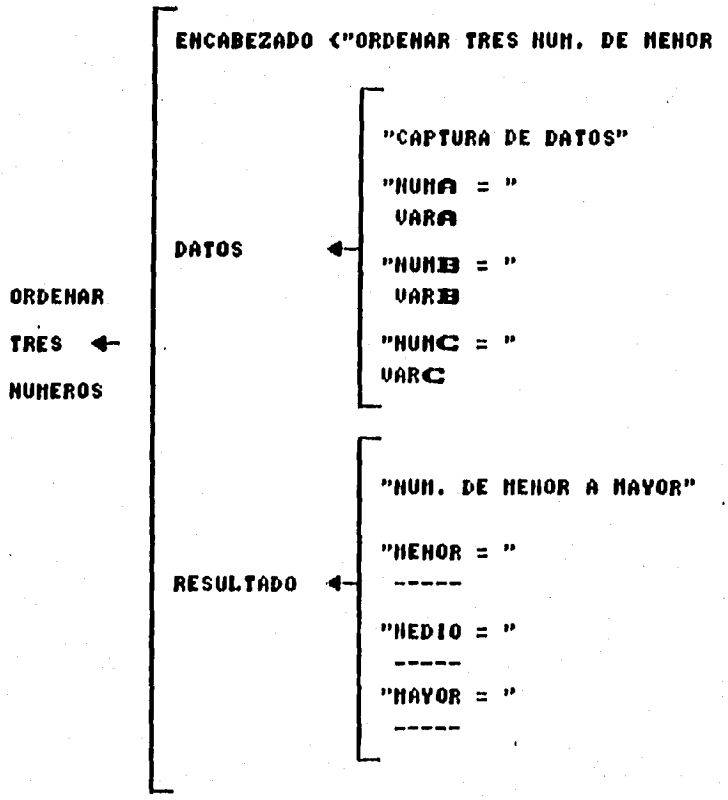
ESQUEMA 5.9 : HOJA DEL REPORTE DE ORDENAR TRES NUMEROS.

..La finalidad del problema 2, es aprender la sentencia de control "if then ". (El reporte lo podemos ver en el esquema 5.9).

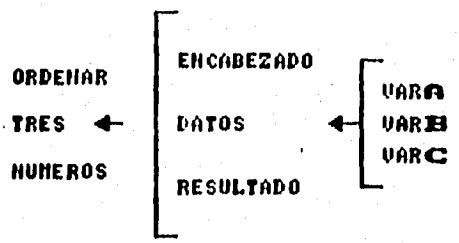
En el subconjunto de resultados del esquema jerárquico del reporte no se especificó ningún nombre de dato de variable de "salida", en su lugar se punteó (esquema 5.10), dado que para ordenar secuencias de números los datos de entrada se utilizan como datos de salida en el orden requerido. (El esquema lógico de entrada lo podemos ver en el esquema 5.11).

Un aspecto importante en la solución de problemas vía un lenguaje de programación es no inventar el hilo negro, es decir, si existen técnicas para la resolución de un problema especifico hay que utilizarlas.

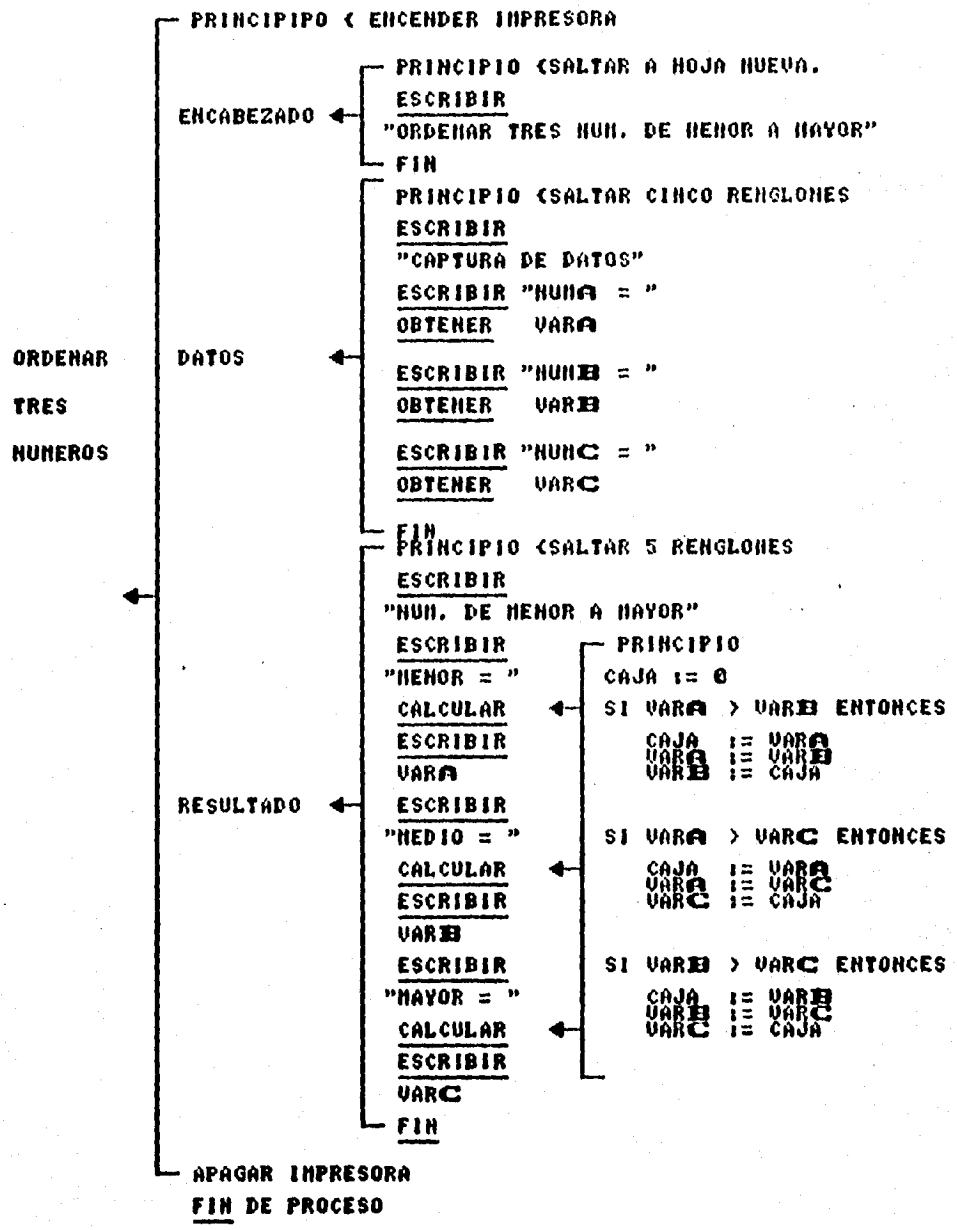
El método para ordenar la secuencia de tres números es el llamado "metodo de la burbuja"; es uno de los más pobres, en el sentido de que si la serie a ordenar es muy grande el proceso es muy lento, pero funciona bien en conjuntos de pocos datos.



ESQUEMA 5.10 : ESQUEMA JERARQUICO DEL REPOR



ESQUEMA 5.11 : ESQUEMA LOGICO DE ENTRADA.



ESQUEMA 5.12 : ESQUEMA LOGICO DEL PROCESO.

En el subconjunto de resultado del esquema lógico del proceso (esquema 5.12) se señaló la instrucción escribe varA, varB y varC, los valores de estas variables de "salida" no son los mismos que los de la entrada, dado que en el proceso de "calcular" se intercambiaron los valores y de esta forma, aunque las variables tienen el mismo nombre, el contenido es diferente.

METODO DE INTERCAMBIO O DE BURBUJA

La técnica de ordenación vía "burbuja" consiste en comparar dos elementos adyacentes y si la secuencia de control "si condición entonces" es verdadera, entonces se procede al intercambio de valores. Los pasos son :

1.- Comparar el primer elemento con el segundo; si están en orden, se mantienen en ese estado, en otro caso se intercambian.

2.- Se compara el segundo elemento con el tercero, si es necesario se intercambian.

3.- El método continúa hasta que todos los elementos se encuentren en orden.

4.- Repetir los pasos 1 a 3 tantas veces como sea necesario hasta que no existan intercambios.

En el proceso de intercambio se utilizó una variable auxiliar a la que denominamos caja, en la cual guardamos el valor de la primera variable; el valor de la segunda variable se asignó a la de la primera y finalmente a la segunda variable se le asignó el contenido de caja, es decir :

```
caja := var1
var1 := var2
var2 := caja
```

Con la técnica de escritorio (probar el algoritmo con papel y lápiz para cerciorarse que la solución no tiene errores de lógica) vamos a checar el proceso de intercambio :

Sea : var1 := 3; var2 := 1; caja := 0.

Variables	var1	var2	caja
Valores iniciales	3	1	0
caja := var1	3	1	3
var1 := var2	1	1	3
var2 := caja	1	3	3

De esta forma el intercambio se efectuó correctamente. ¿Qué pasa si no utilizamos la variable auxiliar caja? El proceso de intercambio sería :

var1 := var2

var2 := var1

Efectuando la prueba de escritorio obtenemos :

variables	var1	var2
valores iniciales	3	1
var1 := var2	1	1
var2 := var1	1	1

Como se puede observar se perdió un valor.

La estructura de datos del "problema" ordenar tres números es :

varA, varB y varC reales de cualquier tipo.

caja real, con caja = 0.

El "programa" ordenar tres números es :

Encender impresora

saltar hoja nueva

escribir "Ordenar tres números de menor a mayor"

saltar 5 renglones

escribir "Captura de datos"

escribir "núm A "

obtener varA

escribir "núm B"

obtener varB

escribir "núm C"

obtener varC

saltar 5 renglones

escribir "Número de menor a mayor"

escribir "menor = "

calcular

escribir varA

escribir "medio = "

calcular

escribir varB

escribir "mayor = "

calcular

escribir varc

apagar impresora

Fin de proceso.

Donde calcular se encuentra descrito como :

```
        caja := 0
    si varA > varB entonces
        caja := varA
        varA := varB
        varB := caja
    si varA > varC entonces
        caja := varA
        varA := varC
        varC := caja
    si varB > varC entonces
        caja := varB
        varB := varC
        varC := caja
```

PROBLEMA 3 : SECUENCIA DE FIBONACCI

Con la secuencia numérica de Fibonacci se pueden aprender las sentencias de control : desde (for), mientras (while) y hasta que (repeat).

En el caso de series de números, no se necesitan datos de entrada, por esto en la zona de datos del reporte se escribe "**NADA**" (esquema 5.13) sino valores iniciales de las variables y una regla para construir la serie.

Como en las series numéricas no existen datos de entrada, entonces no existe el esquema lógico de entrada. El esquema del reporte se puede ver en el esquema 5.14.

ENCABEZADO	"SECUENCIA DE FIBONACCI"
DATOS	" N A D A "
SALIDA	0, 1, 1, 2, 3, 5, 8, 13, 21, 34,...

ESQUEMA 5.13 : HOJA DE REPORTE DE LA SECUENCIA DE FIBONACCI.

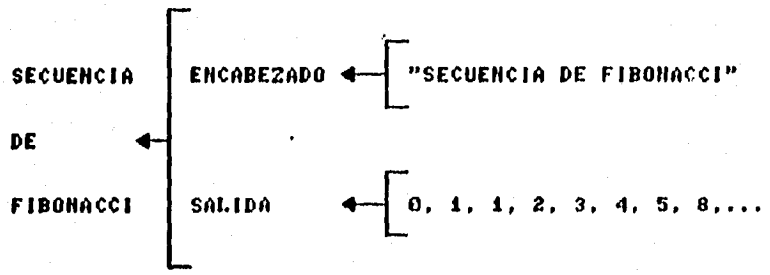
En el subconjunto de salida del esquema del proceso (esquema 5.15), la parte correspondiente a calcular" se encuentra marcada con tres asteriscos; esta parte se va a resolver para las sentencias de control : desde, mientras y hasta que. Primero se va a explicar la regla para obtener la secuencia de Fibonacci :

Variables	Penúltimo	último	suma
Valores iniciales	0	1	1
Asignación 1	1	1	2
Asignación 2	1	2	3
Asignación 3	2	3	5
Asignación 4	3	5	8
Asignación 5	5	8	13
.....			
Asignación n-ésima	último	suma	

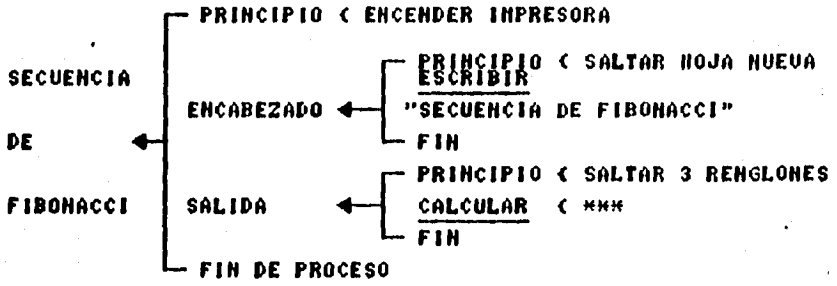
La regla de construcción es bastante sencilla :

1.- A las variables "penúltimo" y "último" se les asigna el valor de cero y uno, respectivamente.

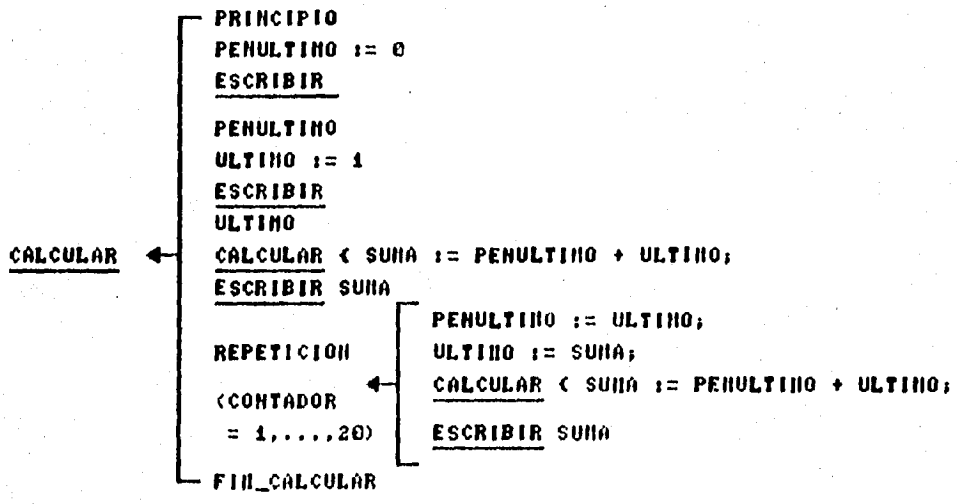
2.- La variable "suma" se obtiene de sumar "penúltimo" y "último".



ESQUEMA 5.14 : ESQUEMA DEL REPORTE.



ESQUEMA 5.15 : ESQUEMA DEL PROCESO.



ESQUEMA 5.16 : CALCULAR PARA EL PROCESO "DESDE" (FOR).

3.- Se efectúan las siguientes asignaciones durante todo el proceso:

penúltimo := último;

último := suma;

suma := penúltimo + último;

4.- La salida se obtiene con los valores iniciales de "**penúltimo**" y "**último**" y el valor de "**suma**" en cada asignación.

CALCULAR : SENTENCIA DE CONTROL "DESDE" (FOR).

Si se requiere repetir un proceso un número determinado de ocasiones se debe usar la estructura de control desde o, para (for en inglés).

La secuencia de control desde ejecuta las acciones un número especificado de veces; de modo automático controla el número de pasos.

La sintaxis es :

desde contador = valor_inicial **hasta** valor_final **hacer**

acciones

...

...

fin_desde

Entonces el subconjunto de salida calcular , para los primeros 20 números de Fibonacci según el ciclo de control desde, lo podemos ver en el esquema 5.16.

CALCULAR : SENTENCIA DE CONTROL "MIENTRAS" (WHILE).

El ciclo de control mientras (en inglés while o do while "hacer mientras"), repite una serie de actividades mientras se cumpla una condición; es decir si la condición se cumple, ejecuta las instrucciones. Antes de ejecutar una acción "checa" que la condición se cumpla y si ésta es cierta ejecuta el proceso.

La sintaxis es :

mientras condición hacer

acciones

...

...

fin_mientras.

El subconjunto de salida calcular, para los números menores a 5000 de la secuencia de Fibonacci; según el ciclo de control mientras lo podemos ver en el esquema 5.17.

```
                PRINCIPIO
                PENULTIMO := 0
                ESCRIBIR
                PENULTIMO
                ULTIMO := 1
                ESCRIBIR
                ULTIMO
CALCULAR  <-    CALCULAR ( SUMA := PENULTIMO + ULTIMO;

                                    ESCRIBIR SUMA
                REPETIR
                                    PENULTIMO := ULTIMO;
                (MIENTRAS    <-     ULTIMO := SUMA;
                 SUMA < 5000)       CALCULAR ( SUMA := PENULT
                FIN_CALCULAR
```

ESQUEMA 5.17 : CALCULAR PARA EL CICLO "MIENTRAS"

```
                PRINCIPIO
                PENULTIMO := 0
                ESCRIBIR
                PENULTIMO
                ULTIMO := 1
                ESCRIBIR
                ULTIMO
CALCULAR  <-    CALCULAR ( SUMA := PENULTIMO + ULTIMO;

                REPITE              ESCRIBIR SUMA
                (HASTA QUE
                                    PENULTIMO := ULTIMO;
                             <-     ULTIMO := SUMA;
                SUMA >= 5000)       CALCULAR ( SUMA := PENULT
                FIN_CALCULAR
```

ESQUEMA 5.18 : CALCULAR PARA EL CICLO "REPITE ... HASTA

CALCULAR : SENTENCIA DE CONTROL "REPITE ... HASTA" (REPEAT).

En el caso del ciclo de control mientras si la condición inicial no se cumple, no realiza ninguna acción, es decir si suma >= 5000, entonces no realiza el proceso.

En algunas ocasiones es deseable que primero se realicen las instrucciones y después se "cheque" la condición: es decir, si la condición es falsa (no se cumple), continúa el proceso hasta_que sea cierta . El ciclo de control repite ... hasta (en inglés, repeat ... until) posee esta estructura.

La sintaxis es :

```
repite
        acción
        acción
        ...
hasta_que condición
```

El subconjunto de salida calcular, para los números menores a 5000 de la secuencia de Fibonacci; según el ciclo de control repite ... hasta_que se ve en el esquema 5.18.

Como la secuencia de Fibonacci se obtiene con las mismas reglas, independientemente de la asignación de control que se utilice, la composición de datos es la misma para los tres ciclos de control. Por tanto la estructura de datos es :

- Penúltimo, último y suma son números enteros mayores o iguales que cero.

  Para el caso del ciclo For, contador es de tipo entero mayor que cero.

El "programa" de la secuencia de Fibonacci es :

```
Encender impresora
    saltar hoja nueva
    escribir "SECUENCIA DE FIBONACCI"
    saltar 3 renglones
    calcular ***
fin_de_proceso.
```

La parte correspondiente de "calcular" para los ciclos desde, mientras y repite lo escribimos a continuación :

Calcular para el ciclo desde (FOR) :

```
penúltimo := 0
escribir penúltimo
último := 1
escribir último
calcular suma := penúltimo + último
escribir suma
desde contador = 1 hasta 20 hacer
    penúltimo := último;
    último := suma;
    calcular suma := penúltimo + último;
    escribir suma
fin_desde
```

Calcular, según el ciclo mientras (WHILE) :

```
penúltimo := 0
escribir penúltimo
último := 1
escribir último
calcular suma := penúltimo + último
mientras suma < 5000 hacer
    escribir suma
    penúltimo := último;
    último := suma;
    calcular suma := penúltimo + último;
fin_mientras
```

Calcular, según el ciclo repite_hasta (REPEAT UNTIL) :

```
penúltimo := 0
escribir penúltimo
último := 1
escribir último
calcular suma := penúltimo + último
repite
    escribir suma
    penúltimo := último;
    último := suma;
    calcular suma := penúltimo + último;
hasta_que suma >= 5000
```

La metodología de Warnier-Orr nos permite encontrar la solución de un problema en forma estructurada y lógica, lo que lleva a eficientar los recursos de que se dispone; no importa el equipo que se utiliza, dado que los "programas" que se obtienen se pueden codificar en cualquier lenguaje de programación estructurado.

CAPITULO SEIS

INTRODUCCION A LA PROGRAMACION ESTRUCTURADA VIA "PASCAL

6.1 INTRODUCCION.

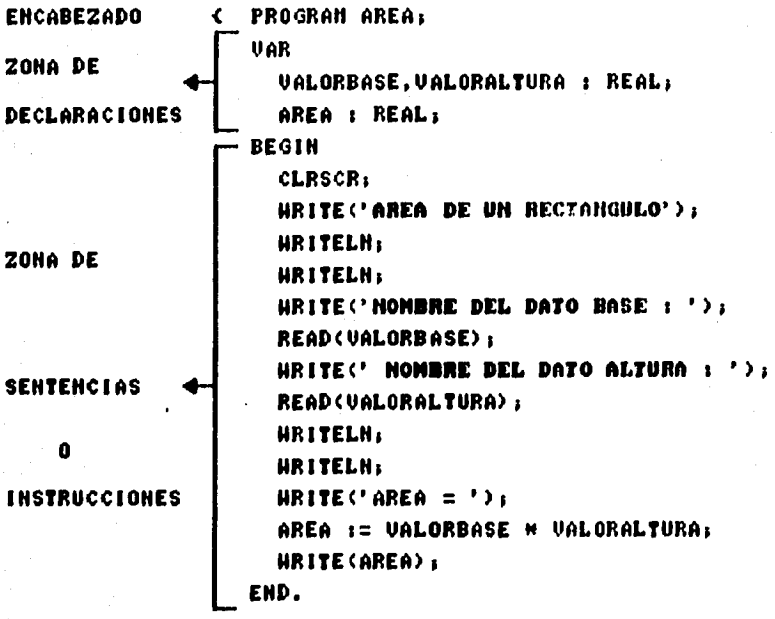
Algunas personas piensan que programar es mágico, difícil, un arte y un sinnúmero más de adjetivos, siendo todo lo contrario. Para programar, sólo se necesita encontrar una solución al problema, escribirla "ordenadamente" y consultar un manual del lenguaje de programación, esto último para conocer algunas reglas de sintaxis e interpretación.

Para el desarrollo de esta unidad se van a utilizar los "programas" obtenidos a partir de la metodología de Warnier-Orr. Para su ejecución es necesario utilizar el editor de Turbo Pascal versión 3.0 ó posteriores. Las características de las instrucciones de TURBO PASCAL las voy a explicar partiendo de ejemplos.

El "programa" del área de un rectángulo obtenido según la metodología de Warnier-Orr y su transcripción a Turbo Pascal se presentan en el esquema 6.1.

	program area;
	var
	valoraltura : real;
	valorbase,area : real;
	begin
saltar a hoja nueva	clrscr;
escribir "Area de un rectángulo"	write('Area de un rectángulo ');
saltar dos renglones	writeln;
	writeln;
escribir "nombre del dato base"	write('nombre del dato base');
obtener valorbase	read(valorbase);
escribir Nombre del dato altura	write('nombre del dato altura);
obtener valoraltura	read('valoraltura');
saltar dos renglones	writeln;
	writeln;
escribir "área ="	write('área = ');
calcular	area :=
area = valorbase * valoraltura	valorbase * valoraltura;
escribir area	write(area)
	end.
----------	----------
"Programa" según Warnier-Orr.	Programa codificado en Turbo Pascal.

Esquema 6.1 : "Programa" Warnier-Orr transcrito a Turbo Pascal.



ESQUEMA 6.2 : PARTES DE UN PROGRAMA EN PASCAL.

Si observamos el "programa" según Warnier-Orr y lo escrito entre las palabras begin y end se ve que existe una gran semejanza.

Aquí lo **importante** es que para codificar la solución de un problema en un lenguaje de programación, es determinante la forma lógica y estructurada en la que se escribió.

Analizando la codificación del "programa" en Turbo Pascal se puede ver que consta de tres partes : el encabezado, la zona de declaraciones y el lugar de sentencias o instrucciones (Esquema 6.2).

El encabezado o cabecera de un programa siempre es la primera línea de un programa y nombra a éste; en la zona de declaraciones se especifican los nombres de los datos, así como al tipo que pertenecen, y, finalmente, el lugar de las sentencias corresponde en sí a las instrucciones que ejecuta el programa. En la zona de sentencias se pueden apreciar las instrucciones clrscr, write, writeln y read, las cuales son definidas por Pascal; a continuación se describirán las instrucciones o subprogramas definidos por Pascal.

SUBPROGRAMAS DEFINIDOS POR PASCAL.

Un subprograma es una secuencia de instrucciones con nombre propio, que puede o no devolver algún valor. Pascal predefinió subprogramas, de los cuales se van a explicar los de mayor uso en un curso introductorio; para ver todos los subprogramas predefinidos por Pascal consulte su manual de referencia.

CLRSCR (Limpia pantalla).

SINTAXIS :

```
clrscr;
```

El procedimiento clrscr limpia la pantalla del monitor y posesiona el cursor en la esquina superior derecha.

GOTOXY (Desplaza el cursor a la columna y al renglón especificado).

SINTAXIS :

gotoXY(col_X , ren_Y);

col_X : Indica la columna del monitor, toma valores enteros de 1 a 24.

ren_Y : Indica el renglón en el monitor, toma valores enteros de 1 a 79.

Se puede considerar el monitor como un plano cartesiano, donde el eje de las X' (abscisas) son las columnas y el eje de las Y' (ordenadas) son los renglones. Entonces la orden gotoxy(col_X, ren_Y) desplaza el cursor a la posición señalada.

READ (Leer datos).

SINTAXIS :

read(lista_de_entrada);

lista_de_entrada : La lista de entrada es un conjunto de tipos de datos separados por coma (,); los datos pueden ser de diferentes tipos. Al terminar de leer la lista_de_datos el cursor permanece en la misma línea. Si se va a leer más de un dato hay que dejar un espacio en blanco entre ellos.

Ejemplo : En el programa de cálculo del área del rectángulo tenemos dos instrucciones read :

read(valorbase);

read(valoraltura);

Las cuales se pueden escribir como una sola :

read(valorbase,valoraltura);

Al capturar los datos, deben ir separados por espacio en blanco.

READLN (Leer datos y saltar renglón).

SINTAXIS :

readln(lista_de_entrada);

lista_de_entrada : El mismo comentario que read( );. la diferencia consiste en que readln al terminar de leer los datos el cursor salta de renglón.

WRITE (Escribe).

SINTAXIS :

write(lista_de_salida);

lista_de_salida : La lista de salida es un conjunto de tipos de datos separados por coma (,); los datos pueden ser de diferentes tipos. Al terminar de escribir la lista_de_datos el cursor permanece en la misma línea. Las cadenas de caracteres deben encerrarse entre apóstrofos; los datos de tipo real se pueden formatear de la siguiente forma:

Write(dato_real:num_digitos:num_decimales);

-dato_real : Dato_real es la variable de tipo real.

-num_digitos : Es un número entero que especifica el número de dígitos que se van a desplegar, considerando la parte entera, el punto y los decimales.

-num_decimales : Es un número entero que especifica el número de decimales que se van a desplegar.

Ejemplo : Del "programa" cálculo del área de un rectángulo, tenemos que :

```
write('área = ');

write(area);
```

Estas instrucciones las podemos escribir como una sola :

```
write('area = ',area:6:2);
```

La variable real "area" se encuentra formateada.

WRITELN (Escribe y salta renglón).

SINTAXIS :

```
writeln(lista_de_salida);
```

-lista_de_salida : Se aplica el mismo criterio que write();. Se diferencia en que writeln( ); al terminar de escribir salta al siguiente renglón.

La entrada de datos en Pascal se puede efectuar del teclado o de un archivo de datos, así como la salida puede ser al monitor, a un archivo o a la impresora. La entrada y salida estandard, es decir del teclado y monitor, lo efectuamos con las instrucciones read, readln, write y writeln, respectivamente. Dada la naturaleza de un

curso introductorio, la entrada y salida de datos de archivos no se va a tratar; el formato de salida por medio de la impresora de las instrucciones write y writeln es :

write(lst,lista_de_salida);

writeln(lst,lista_de_salida);

Donde lista_de_salida se definió anteriormente.

FUNCIONES ARITMETICAS.

Turbo Pascal cuenta con un conjunto de funciones aritméticas de uso frecuente :

ABS (Valor absoluto)

SINTAXIS :

abs(var_num);

abs : la función abs( ) regresa el valor absoluto de var_num.

var_num : Var_num es un dato de tipo Integer o Real.

Por ejemplo la sentencia :

local := abs(-3);

Le asigna a la variable local el valor de 3.

ARCTAN (Arco tangente).

SINTAXIS :

arctan(ángulo);

arctan : Regresa el arco tangente del "ángulo" en radianes.

ángulo : El ángulo puede ser Real o Integer.

COS (Coseno).

SINTAXIS :

cos(ángulo);

cos : Regresa el coseno del ángulo en número Real.

ángulo : El ángulo debe estar en radianes, puede ser Integr o Real.

EXP (Función Exponencial).

SINTAXIS :

exp(número);

exp : Calcula la exponencial de número ($e^{número}$)

número : El número puede ser Integer o Real.

FRAC (Fracción).

SINTAXIS :

frac(núm_real);

frac : Regresa la parte decimal, de un número Real, expresado en número Real.

INT (Entero).

SINTAXIS :

int(núm_real);

int : La función int( ) regresa la parte entera de un número real expresado en Real.

LN (Logaritmo natural).

SINTAXIS :

ln(var_núm);

ln : Calcula el logaritmo natural de var_núm.

var_núm : var_núm puede ser Integer o Real.

RANDOM (Aleatorio).

SINTAXIS :

random(var_ent);

random : Si se omite var_ent random( ) regresa un número pseudoaleatorio mayor o igual a cero y menor a uno; si se considera var_ent entonces regresa un número entero aleatorio entre cero y var_ent, sin incluir var_ent.

var_ent :Número de tipo Integer.

ROUND (Redondear).

SINTAXIS :

round(var_real);

round : Redondea la var_real al entero más próximo, respetando las siguientes reglas :

sea pf = "parte fraccionaria de var_real"

si pf < 0.5 , se redondea al entero menor próximo.

si pf >= 0.5, se redondea al entero mayor próximo.

var_real : Donde var_real es un dato de tipo Real.

SIN (Seno).

SINTAXIS :

sin(ángulo);

sin : Regresa el seno del ángulo.

ángulo : Puede ser un número Integer o Real, expresado en radianes.

SQR (Cuadrado).

SINTAXIS :

sqr(var_núm);

sqr : Regresa el cuadrado de var_núm.

var_num : Número Integer o Real.

SQRT (Raíz cuadrada).

SINTAXIS :

sqrt(var_núm);

sqrt : Calcula la raíz cuadrada de var_núm.

var_núm : Var_núm puede ser un número Real o Integer mayor que cero.

TRUNC (Truncar).

SINTAXIS :

trunc(var_real);

trunc : La función trunc( ) regresa la parte entera de var_real expresado en número Integer.

var_real : Número Real.

6.2 PARTES DE UN PROGRAMA EN PASCAL.

Como se mencionó, un programa en Pascal se divide en tres partes :

I) El encabezado o cabecera, es opcional, e identifica o nombra el programa.

II) La zona de definiciones o declaraciones, la cual es opcional. En esta parte se definen los datos así como al tipo al que pertenecen; los tipos de datos pueden ser definidos por Pascal o por el usuario; consta de :

- Definiciones de tipos.
- Definiciones de constantes.

- Definiciones de variables.

- Definiciones de etiquetas.

- Definiciones de subprogramas.

III) Las sentencias o instrucciones pueden ser de :

- Asignación.

- Control del flujo

La parte de sentencias debe empezar con la palabra reservada begin y terminar con end (en la tabla 6.3 se puede ver una lista de palabras reservadas); la palabra end indica la finalización del programa y va seguida de un punto (.). La única parte no opcional es la de sentencias.

absolute	external	nil	shr
and	file	not	string
array	for	of	then
begin	forward	or	to
case	function	packed	type
const	goto	procedure	until
div	if	program	var
do	in	record	while
downto	inline	repeat	with
else	label	set	xor
end	mod	shl	

La siguiente palabra soló es reservada en la versión 3.0 :

overlay

Las que siguen son reservadas sólo en 4.0 y 5.0 :

implementation	interrupt	uses
interface	unit	

Tabla 6.3 : Palabras reservadas en Turbo Pascal.

Se describirá cada una de estas zonas. Las palabras reservadas se escribirán con negritas. Las que deben ser proporcionadas por el usuario se anotarán en letra normal.

I LA CABECERA O ENCABEZADO DEL PROGRAMA.

Sintaxis.

program nombre_del_programa ;

Program : Es una palabra reservada que identifica al programa.

nombre_del_programa : Es un nombre que identifica la acción del programa.

; : El punto y coma indica la terminación del encabezado.

Ejemplo : program area;

La palabra area es el nombre_del_programa.

II LA ZONA DE DEFINICIONES O DECLARACIONES.

La zona de definiciones se encuentra entre la cabecera y la zona de sentencias; se encuentra conformada de :

Definiciones de tipos.

Definiciones de constantes.

Declaración de variables.

Declaración de etiquetas.

Definición de subprogramas.

En este trabajo se van a desarrollar las definiciones de constantes, la declaración de variables; de los tipos de datos definidos por el usuario se tratarán los tipos simples (subrango y enumerativo), de los de tipo estructurado solamente se verán los arreglos.

En la zona de definiciones se les asigna un nombre a los datos, así como al tipo de datos al que pertenecen, y, de esta forma, se les reserva una localidad en memoria para que coloquen en ella el o los valores que se utilizarán en el proceso correspondiente.

El nombre del dato, también conocido como identificador, debe respetar las siguientes reglas :

a) Debe comenzar con un carácter alfabético o un carácter de subrayado (_).

b) Seguido de hasta 126 caracteres alfabéticos, numéricos y de subrayado. Los caracteres alfabéticos pueden ser mayúsculas o minúsculas, es indistinto para Pascal. La mayoría de los programadores escribe los programas en minúsculas.

TIPOS.

Los tipos de datos definidos por Pascal son :

-**Boolean** : TRUE y FALSE.

-**Byte** : Valor entero entre 0 y 255.

-**Char** : Conjunto de caracteres ASCII más los caracteres cuyo valor interno va de 128 a 255.

-**Integer** : Valores enteros entre -32768 y 32767

-**Real** : Números reales.

-**Text** : Archivo externo del tipo Char.

El tipo Text no se va a definir en este trabajo, dado que no son materia de un curso introductorio en el CCHA. Para su consulta vea un manual de Pascal.

Boolean : El tipo Boolean toma los valores de verdad TRUE (cierto) y FALSE (falso). Generalmente se utiliza con sentencias condicionales y con los ciclos de control. Utiliza un byte de memoria.

Byte : El tipo byte se define solamente en Turbo. Ocupa un solo byte en memoria; por tanto, toma valores enteros en el rango de 0 a 2^8 - 1, es decir, de 0 a 255. El tipo de datos byte es un subconjunto de los de tipo integer, y se utiliza cuando se sabe que un dato nada más va a tomar valores en el rango definido, esto para ahorrar espacio en memoria.

Char : El tipo char toma los valores del conjunto ASCII (ver una tabla en un manual de referencia), más el conjunto extendido de caracteres con valor interno del 127 al 255. Utiliza un byte de memoria.

Integer : el tipo integer solamente toma valores enteros en un rango establecido que depende del procesador; por ejemplo, con el procesador 8088, el rango establecido es : -2^{15} a 2^{15} - 1, es decir de -32768 a 32767, dado que los datos tipo integer ocupan dos bytes de memoria.

Real : El tipo real ocupa n bytes en memoria dependiendo del procesador, para el 8088 toma 6 bytes y los valores reales se encuentran en el rango 1^{-99} a 1^{+99} con una precisión de 11 dígitos significativos, es decir, 11 decimales.

CONSTANTES

Las constantes son identificadores a los cuales se les asigna un valor fijo, el que permanece sin modificaciones durante el proceso. Existen varios tipos de constantes : las definidas por el usuario, con identificador que define el valor y las definidas por Pascal.

a) Las definidas por el usuario, a las que también se les conoce como constantes con nombre, se declaran en la parte de definición de constantes. Se escriben de la siguiente forma :

```
Cónst
    identificador = valor;
```

Donde identificador es el nombre de la constante. El signo de igual (=) asigna el valor al identificador. El valor es el ente que se le va a asignar al identificador, el cual puede ser entero, real, booleano, carácter o una cadena de caracteres. El punto y como (;) indica la terminación de la definición.

Ejemplo :

```
Const
    veces  = 10;
    prima  = 1.0003;
    bandera = true;
    letra  = 'A';
    fin    = 'Se terminó'
```

Nótese que los valores de los identificadores "letra" y "fin" se acompañan con apóstrofos (').

b) Las constantes con identificador literal describen un valor que puede ser booleano (true, false), entero (15, 6) o carácter ('a', '6', 'A'). Nótese que las constantes '6' y 6 son diferentes pues la primera representa al carácter '6' el cual no tiene valor numérico, mientras que el 6 es entero y representa un valor numérico.

c) las constantes definidas por Pascal tienen un valor definido por Pascal :

Identificador	Descripción.
MAXINT	Representa el valor 32767.
PI	Valor de pi 3.1415926536.

VARIABLES

Las variables son identificadores de datos que toman diferentes valores durante el desarrollo de un proceso; pueden ser de tipo : cadena, carácter, real, entero, byte o booleano. Se declaran en la parte de variables escribiendo la palabra var y a continuación se escribe el o los identificadores, seguido de dos puntos (:) el tipo de datos al que pertenecen, terminando con punto y coma (;). Si los identificadores son del mismo tipo pueden ir o no separados por coma (,).

Ejemplo 1 :

```
var
    senal   : Boolean;
    inicial : char;
    nombre  : string[30];
    indice  : integer;
```

```
i,j,k,l  : integer;
radio    : real;
lado10,
base_01,
altura   : real;
```

En la parte de definiciones de variables var se definen los identificadores así como su tipo, con la finalidad de asignarles una localidad y tamaño en memoria para recibir los valores que se les asigne en la zona de sentencias.

Ejemplo 2. En el problema de obtener el área de un rectángulo se definieron las variables :

```
var
    valorbase,valoraltura : real;
    area : real;
```

Nótese que el nombre del identificador señala la acción o función que tiene el identificador en el proceso, es decir, la asignación de nombres para los identificadores es deseable que éstos señalen la acción o papel que van a desempeñar. Al identificador se le puede dar cualquier nombre.

Ejemplo 3 :

```
var
    arana  : integer;
    piedra : char;
    nose   : real;
```

Aunque el nombre de los identificadores es válido no indica la acción que va a realizar y esto puede llevar a errores.

CONSTANTES CON TIPO

Las constantes con tipo son variables de tipo constante, que se declaran en la parte de constantes const; a las que se les asigna tanto el tipo al que pertenecen así como un valor. Sólo se pueden definir para Turbo Pascal.

Ejemplo :

```
const
interes  : byte = 80;
letra    : char = 'A';
```

III ZONA DE SENTENCIAS O INSTRUCCIONES

En la zona de instrucciones o sentencias se realiza la codificación de la solución del "problema"; para ello vamos a definir los operadores, las sentencias y a explicar algunos subprogramas contenidos en Pascal. Las sentencias se encuentran contenidas entre las palabras begin y end, esta última palabra con punto final (.).

SENTENCIAS

Existen tres tipos de sentencias : de asignación, compuestas y de control. Las sentencias de control se detallarán en la sesión de "ciclos y sentencias de control".

El punto y coma (;) se utiliza por regla general para separar dos sentencias, es decir, notifica que termina una instrucción y comienza otra. En algunas ocasiones se omite el punto y coma, lo que se explicará en el momento oportuno.

SENTENCIAS DE ASIGNACION.

En una sentencia de asignación se transfiere un valor o expresión a un identificador.

SINTAXIS.

identificador := expresión ;

Identificador es el nombre de la variable a la que se le va a transferir el contenido de la expresión. := es el símbolo de asignación. **Expresión** es lo que se va a asignar al identificador; La expresión puede ser una constante, una variable o una expresión del mismo tipo que la variable

Ejemplo 1. En la codificación del área de un rectángulo tenemos la asignación :

area := valorbase * valoraltura;

Donde "area" es el identificador y "valorbase * valoraltura" es la expresión, que en este caso es aritmética.

Debe existir compatibilidad en el tipo definido entre el identificador y la expresión, es decir, por ejemplo a una variable declarada como tipo char no se le puede asignar un valor entero.

SENTENCIAS COMPUESTAS.

Las sentencias compuestas se componen de sentencias de asignación, subprogramas y sentencias de control; se encuentran encerradas ente un begin y un end. Para los efectos del punto y coma (;), todo el grupo begin...end; funciona como una sola sentencia.

Ejemplo 1:

```
begin
    valorbase := 12;
    valoraltura := 14;
```

```
        area := valorbase * valoraltura;
    end;
```

En los ciclos de control es más común utilizar las sentencias compuestas.

Ejemplo 2 :

```
for var_ent := val_ini to val_fin do
  begin
    if condición then
      begin
        sentencia1;
        sentencia2;
      end (* if *)
    else
      sentencia3;
  end; (* for var_ent *)
```

Nótese que la palabra end que precede al vocablo else no se acompaña del punto y coma (;), dado que si se inserta se rompe la secuencia de la condición if_then_else.

OPERADORES

Un operador lo podemos definir como : Palabra clave (puede ser un símbolo) de una expresión, que indica un cálculo a realizar sobre una o más subexpresiones (operandos).

En Pascal existen operadores aritméticos, lógicos y relacionales (ver tabla 6.4). En Pascal no existe un operador para elevar a potencias.

Un operador es unario si actúa sobre un solo operando. Es binario si actúa sobre dos operandos. En general, es n-ario si actúa sobre n operandos.

Identificadores	Descripción
Operadores aritméticos.	
-	(Menos unario). Invierte el signo de un operando.
*	Multiplicación de número entero o real
/	División de números reales.
div	División entera.
mod	Operación módulo; el resulta después de la división entera.
+	Suma de números enteros o reales.
-	Resta de números enteros o reales.
Operadores lógicos o booleanos.	
not	Operación not booleana y entera.
and	Operación and booleana y entera.
or	Operación or inclusiva booleana y entera.
xor	Operación xor exclusivo booleano y entera.
Operadores relacionales.	
valor1 = valor2	TRUE si los valores son iguales.
valor1 <> valor2	TRUE si los valores son diferentes.
valor1 < valor2	TRUE si valor1 es menor que valor2.
valor1 <= valor2	TRUE si valor1 es menor o igual que valor2
valor1 > valor2	TRUE si valor1 es mayor que valor2.
valor1 >= valor2	TRUE si valor1 es mayor o igual que valor2.

Tabla 6.4 : Operadores de Pascal.

OPERADORES ARITMETICOS.

Los operadores aritméticos se escribirán en negritas, se presentarán en el siguiente orden : operador, el o los tipos de datos sobre los que actúa.

Los operadores shl y shr no se van a tratar en este trabajo, lo mismo que los operadores not, and, or y xor cuando se trabajan como datos de tipo entero, dado que todos estos operadores actúan sobre números enteros y modifican su equivalente en binario. Y corresponden a una extensión de TurboPascal. Para su referencia consulte un manual de Turbo Pascal.

- (menos unario) Integer o Real.

El operador menos unario, invierte el signo de su operando. Por ejemplo :

Si valor1 = 12, entonces -valor1 = -12.

Si valor2 = -3.34, entonces -valor2 = 3.34.

* (producto) Integer o Real.

SINTAXIS :

num1 * num2;

Se obtiene como resultado el producto de la expresión; el operador * ejecuta el producto de num1 y num2. Los operadores num1 y num2 pueden ser datos de tipo entero o real. El resultado es un número real o entero, dependiendo del tipo de los operandos.

En el producto, en la suma y la resta, si al menos uno de los operandos es **Real**, entonces el resultado será **Real**; en otro caso será **Entero**. En el ejemplo siguiente se muestra un programa completo que muestra el producto de datos tipo entero y Real.

```
program producto;
var (* declaración de variables *)
    ent1, ent2, resent : integer;
    val1, val2, resval1, resval2 : real;
begin
    ent1 := 2; (* asignación de número entero *)
    ent2 := 3;
    val1 := 3.5; (* asignación de número real *)
    val2 := 2.5;
    resent := ent1 * ent2; (* producto de enteros *)
    resval1 := ent1 * val1; (* prod. de entero y real*)
    resval2 := val1 * val2; (* producto de reales *)
    write('El producto de ent1 y ent2 es el Entero :');
    writeln(resent);
    write('El producto de ent1 y val1 es el Real :');
    writeln(resval1);
    write('El producto de val1 y val2 es el Real :');
    writeln(resval2);
end.
```

El ejemplo es demasiado explícito para requerir de más explicaciones.

/ (División) Integer o Real.

SINTAXIS :

num1 / num2;

El operador / ejecuta la división de num1 entre num2, el resultado es un número real; Los operadores num1 y num2 pueden ser datos de tipo entero o real. Num2 tiene que ser diferente de cero.

div (División entera) Integer.

SINTAXIS :

ent1 div ent2;

El resultado corresponde al cociente de la división de ent1 entre ent2. Los operadores ent1 y ent2 son datos de tipo entero. Ent2 tiene que ser diferente de cero.

mod (Operación módulo) Integer.

SINTAXIS :

ent1 mod ent2;

El operador mod realiza la división entre ent1 y ent2, y da como resultado el residuo de la división. Los operandos ent1 y ent2 son datos de tipo entero. Ent2 tiene que ser diferente de cero.

+ (Suma) Integer o Real.

SINTAXIS :

num1 + num2;

El operador + ejecuta la suma de num1 y num2; el resultado es un número real o entero. Los operandos num1 y num2 pueden ser datos de tipo entero o real.

- (Resta) Integer o Real.

SINTAXIS :

num1 - num2;

El resultado es un número real o entero, que se obtiene al restar num2 de num1. Los operandos num1 y num2 pueden ser datos de tipo entero o real.

OPERADORES BOOLEANOS O LOGICOS.

Turbo Pascal soporta cuatro operadores lógicos : not, and, or y xor, que a continuación explicaremos.

George Boole desarrolló una teoría algebraica en base a los enunciados que admiten solamente dos respuestas; a dichas respuestas les llamaremos valores de verdad, (verdad, falso), (sí, no), (1,0) y tres operadores : and, or y not; dicha teoría se conoce como Algebra Booleana.

Los enunciados que nos interesan los vamos a llamar proposiciones para diferenciarlos de aquellos enunciados que no admiten ningún valor de verdad como son los imperativos, los exclamativos y los interrogativos. Las dos respuestas de una proposición tienen la propiedad de ser mutuamente exclusivas, es decir, no se pueden presentar las dos respuestas al mismo tiempo; si una de las dos se da, la otra no se presenta. Por ejemplo :

La ballena es un mamífero. : verdad.

Los árboles caminan. : falso.

La naranja es una fruta. : true.

Los perros son aves. : false.

Por cuestiones de notación vamos a utilizar al uno como verdad y al cero como falso, que es como la computadora representa internamente estos valores. A las proposiciones las representaremos por letras mayúsculas del alfabeto. Por ejemplo :

A = " La ballena es un mamífero ".

B = " Los árboles caminan ".

A partir de dos o más proposiciones sencillas y con los operadores lógicos and y or podemos formar proposiciones compuestas. Por ejemplo :

1.- Luis Alfredo tomará parte en la representación O (or) ayudará en el vestuario.

2.- El pizarrón es verde Y (and) el gis es blanco.

En estos ejemplos se puede observar que dos proposiciones sencillas se juntaron por medio de los operadores "O" (or) y "Y" (and), respectivamente, para formar una proposición compuesta.

OPERADOR OR.

Efectuemos un análisis más cuidadoso del ejemplo uno para definir el significado lógico del operador "OR", así como para construir una tabla de verdad de dicho operador.

En el ejemplo 1, la proposición compuesta significa que : Luis Alfredo tomará parte en la representación o (OR) que Luis Alfredo ayudará en el vestuario o (OR) que Luis Alfredo participará en ambos eventos, es decir, el significado lógico del operador

OR es en el sentido incluyente, o sea, para que la proposición compuesta sea verdadera basta con que, por lo menos, sea cierta una proposición sencilla o que ambas lo sean.

En base al sentido lógico del operador "OR" construyamos una tabla de verdad.

Sea :

A = "Luis Alfredo tomará parte en la representación"

B = "Luis Alfredo ayudará en el vestuario"

Entonces la proposición compuesta la denotamos como : A OR B y definimos al operador OR como :

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

OPERADOR AND.

Para definir el significado lógico del operador AND, veamos el ejemplo dos : El pizarrón es verde Y (AND) el gis es blanco; para que la proposición compuesta sea cierta, las dos proposiciones sencillas deben ser ciertas al mismo tiempo, o sea, que el pizarrón tiene que ser verde Y (AND) el gis tiene que ser blanco. Si una de las dos es falsa o ambas son falsas, entonces la proposición compuesta es falsa.

En base al sentido lógico del operador AND construyamos una tabla de verdad.

Sea :

A = "El pizarrón es verde"

B = "El gis es blanco"

Entonces la proposición compuesta la denotamos como : A AND B y definimos al operador AND como :

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OPERADOR NOT.

El operador NOT actúa sobre una proposición, es decir, es una operación unaria. Si a una proposición se le añade el operador NOT obtenemos como resultado la negación o inverso de la proposición, es decir, que si la entrada es 0 el resultado es 1 y viceversa. Por ejemplo, si consideramos la proposición : Luisa es una persona alta, la negación seria : Luisa no (NOT) es una persona alta. Asignando símbolos a las proposiciones obtenemos :

Sea :

A = " Luisa es una persona alta".

La negación sería :

NOT(A) = "Luisa no es una persona alta".

El operador NOT lo definimos como :

A	NOT(A)
0	1
1	0

OPERADOR XOR. (OR EXCLUSIVO).

Para explicar al operador XOR consideremos su significado lógico con el siguiente ejemplo : O (XOR) Fabiola tiene 12 años O (XOR) tiene 20 años, la proposición significa que O (XOR) bien Fabiola tiene 12 años O (XOR) Fabiola tiene 20 años, es decir O (XOR) tiene una edad O (XOR) tiene la otra, pero no ambas al mismo tiempo.

En base al sentido lógico del operador XOR construyamos una tabla de verdad.

Sea :

A = "Fabiola tiene 12 años".

B = "Fabiola tiene 20 años".

Entonces la proposición compuesta la denotamos como : A XOR B y definimos al operador XOR como :

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

La salida es 1, solamente si alguna entrada es 1, pero no ambas.

En la siguiente tabla de verdad mostraremos las operaciones lógicas fundamentales :

A	B	AND	OR	XOR	NOT A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Donde 0 = "falso" y 1 = "verdadero".

6.3 CICLOS Y SENTENCIAS DE CONTROL

Las sentencias de control (los ciclos) forman una parte importante en el desarrollo de la solución de un "problema" vía la computadora.

Los operadores relacionales producen un resultado booleano; los operandos que se van a comparar deben ser del mismo tipo, con excepción de los datos Reales, Integer y Byte, que sí se pueden mezclar. En la tabla 6.4 se pueden ver los operadores relacionales

Los operadores relaciónales se encuentran fuertemente ligados a las sentencias de control; en la sintaxis de éstas la palabra condición se refiere a las sentencias relacionales, es decir, al operador relacional junto con los dos valores que compara.

El significado de las estructuras de control (desde, mientras y repite ... hasta_que) se explicaron en la unidad cinco y en la sesión 3 de la unidad seis; falta definir la sintaxis en Pascal, la cual se describirá a continuación.

SENTENCIA DE CONTROL WHILE (MIENTRAS).

```
while condición do
    acción;
```

Si la condición es falsa entonces no se ejecuta la acción.

Utilizando una sentencia compuesta la sintaxis es :

```
while condición do
    begin
        acción1
        acción2
        .....
    end;
```

A manera de ejemplo utilicemos el "programa" obtenido con la metodología de Warnier-Orr de la secuencia de Fibonacci :

EL "PROGRAMA" DE LA SECUENCIA DE FIBONACCI PARA EL CICLO WHILE SEGUN WARNIER-ORR ES :

```
Encender impresora
    saltar hoja nueva
    escribir "SECUENCIA DE FIBONACCI"
    saltar 3 renglones
    Calcular
        penúltimo := 0
        escribir penúltimo
        último := 1
        escribir último
        calcular suma := penúltimo + último
        mientras suma < 5000 hacer
                escribir suma
                penúltimo := último;
                último := suma;
                calcular suma := penúltimo + último;
        fin_mientras
fin_de_proceso.
```

LA CODIFICACION DE LA SECUENCIA DE FIBONACCI PARA EL CICLO WHILE EN PASCAL ES :

```
program fibonacci_mientras;
var (*Variables definidas en la*)
    pen,ult,suma : integer; (* estructura de dato*)
begin
    clrscr;(* Limpia pantalla*)
    write('SECUENCIA DE FIBONACCI');
    writeln; writeln; writeln; (* Saltar 3 renglones *)
    pen := 0;
    write(pen:4,', ');
    ult := 1;
    write(ult:4,', ');
    suma := pen + ult;
    while suma < 5000 do (* inicio ciclo mientras *)
        begin
            write(suma:4,', ');
            pen := ult;
            ult := suma;
            suma := pen + ult;
        end (* fin ciclo mientras*)
end. (* fin_fibonacci_mientras*)
```

SENTENCIA DE CONTROL FOR (DESDE)

La secuencia de control for, se encuentra diseñada para ejecutar un proceso un número determinado de veces.

La sintaxis es :

```
for contador = valor_inicial to valor_final do

    acción
```

En el caso del ciclo for con una sentencia compuesta, tenemos que la sintaxis es :

```
for contador = valor_inicial to valor_final do
  begin
      acción1
      acción2
      .....
  end;
```

EL "PROGRAMA" DE LA SECUENCIA DE FIBONACCI PARA EL CICLO FOR SEGUN WARNIER-ORR ES :

```
Encender impresora
    saltar hoja nueva
    escribir "SECUENCIA DE FIBONACCI"
    saltar tres renglones
    Calcular
            penúltimo := 0
            escribir penúltimo
            último := 1
            escribir último
            calcular  suma := penúltimo + último
            escribir suma
            desde contador = 1 hasta 20 hacer
                    penúltimo := último;
                    último := suma;
                    calcular suma := penúltimo + último;
                    escribir suma
            fin_desde
fin_de_proceso.
```

LA CODIFICACION DE LA SECUENCIA DE FIBONACCI PARA EL CICLO FOR EN PASCAL ES :

```
program fibonacci_desde;
var (*Variables definidas en la*)
    pen,ult,suma : integer; (* estructura de dato*)
    contador : integer;
begin
    clrscr;          (* Limpia pantalla*)
    write('SECUENCIA DE FIBONACCI');
    writeln; writeln; writeln; (* Saltar 3 renglones *)
    pen := 0;
    write(pen:4,', ');
    ult := 1;
    write(ult:4,', ');
    suma := pen + ult;
    write(suma:4,', ');
    for contador = 1 to 20 do (* inicio del ciclo *)
        begin          (* desde *)
            pen := ult;
            ult := suma;
            suma := pen + ult;
            write(suma:4,', ');
        end; (* fin ciclo desde)
end. (* fin_fibonacci_desde*)
```

SENTENCIA DE CONTROL REPEAT (REPITE)

La estructura de control repeat, primero realiza el proceso y después checa la condición; si ésta no se cumple, continúa con el proceso. La sintaxis del ciclo repeat en Turbo Pascal es :

```
repeat
    sentencia1
    sentencia2
until condición
```

Como se puede observar esta estructura carece de los paréntesis begin_end, las palabras reservadas repeat_until realizan la misma función, es decir, aceptan sentencias compuestas.

EL "PROGRAMA" DE LA SECUENCIA DE FIBONACCI PARA EL CICLO REPEAT_UNTIL SEGUN WARNIER-ORR ES :

```
Encender impresora
    saltar hoja nueva
    escribir "SECUENCIA DE FIBONACCI"
    saltar tres renglones
    calcular
        penúltimo := 0
        escribir penúltimo
        último := 1
        escribir último
        calcular suma := penúltimo + último
        repite
                escribir suma
                penúltimo := último
                último := suma
                calcular suma := penúltimo + último
        hasta_que_suma >= 5000
fin_de_proceso.
```

LA CODIFICACION DE LA SECUENCIA DE FIBONACCI PARA EL CICLO REPEAT_UNTIL EN PASCAL ES :

```
program fibonacci_repite;
var                    (* Variables definidas en la*)
    pen,ult,suma : integer; (* estructura de dato*)
bogin
    clrscr;            (* Limpia pantalla*)
    write('SECUENCIA DE FIBONACCI');
    writeln; writeln; writeln;  (* Saltar 3 renglones *)
    pen := 0;
    write(pen:4,', ');
    ult := 1;
    write(ult:4,', ');
    suma := pen + ult;
    repeat                 (* inicio del ciclo *)
        write(suma:4,', ');
        pen := ult;
        ult := suma;
        suma := pen + ult;     (* el punto y coma (;) *)
    until suma >=  5000;       (* algunos sistemas no lo *)
end. (* fin_fibonacci_repite*) (* permiten porque "corta" *)
                                   (* el Repeat...until *)
```

6.4 TIPOS DEFINIDOS POR EL USUARIO.

Hasta ahora hemos visto la declaración de constantes y variables, así como los ciclos básicos de control (if...then, for, while y repeat), con lo cual se puede efectuar una infinidad de aplicaciones. Sin embargo, la potencia de Pascal reside en la facilidad de definir tipos que se ajusten a las necesidades requeridas.

Para que una definición de tipo tenga efecto sobre un programa es necesario declarar al menos una variable de ese tipo; de esta forma el compilador reserva espacio en memoria y establece los límites necesarios para los valores de los datos así definidos y, de esta forma, el compilador se encuentra posibilitado para detectar asignaciones erróneas.

TIPO SUBRANGO.

En cierta ocasión revisando las edades de los pensionados del IMSS, se encontraron edades negativas, es decir, ¡existían pensionados que todavía no nacían !; se puede encontrar infinidad de situaciones en la vida real de esta naturaleza. El problema se puede superar estableciendo rangos (SUBRANGOS) en los cuales las variables se encuentren definidas, esto es, establecer valores mínimos y máximos dentro de los cuales las variables tomen valores válidos.

En el caso de los pensionados del IMSS podemos definir el tipo SUBRANGO edad_legal de la siguiente manera :

```
type
    edad_legal = 18..120;
var
    edad : edad_legal;
```

Además de los tipos estándar definidos por Pascal, tenemos un tipo definido por el programador, y, como se puede ver en la parte de variables, se definió la variable edad del tipo subrango edad_legal. Si se efectúa una asignación a la variable edad fuera del rango 18..120, el compilador señalará un error; por ejemplo :

```
edad := 15;
```

La variable edad sólo puede tomar valores en el rango especificado por edad_legal.

Los tipos subrango pueden ser de cualquier tipo excepto real. por ejemplo :

```
type
    alfabeto = 'A'..'Z';
    segundo = 1..60;
```

TIPO ENUMERATIVO.

En ocasiones es necesario definir (enumerar) solamente un conjunto de valores que las variables pueden tomar. Por ejemplo :

```
type
    vocal = (a,e,i,o,u);
    mes_nac = (ENE, FEB, MAR, ABR, MAY, JUN,
            JUL, AGO, SEP, OCT, NOV, DIC);
    dia_habil = (lun,mrt,mie,jue,vie);
var
    letra : vocal;
    mes   : mes_nac;
    dia   : dia_habil;
```

De esta forma declaramos las variables letra, mes y dia del tipo enumerativo : vocal, mes_nac y dia_habil, respectivamente, con lo que se puede hacer asignaciones de la siguiente forma :

```
letra := a;
mes   := ENE;
dia   := vie;
```

Los elementos de un tipo enumerativo se encuentran ordenados, por ejemplo : ENE < DIC y vie > jue; es decir, tienen un valor ordinal asociado y de esta forma se puede hablar del primer valor, el segundo valor, etc. A partir de esta propiedad de los datos de tipo enumerativo se pueden emplear las funciones : Pred, Succ, Ord, Odd y Chr (ver tabla 6.5).

Con las variables de tipo enumerativo y las funciones ordinales se pueden construir ejemplos de fácil lectura :

```
if ( dia = lunes ) then
    writeln(' Prepara los reportes ')
else
    writeln('Es el día ',ord(dia) + 1);
```

Función	Descripción
Chr	Convierte un parámetro del tipo Integer o Byte al carácter equivalente en ASCCI.
Odd	True si el valor ordinal es impar.
Ord	Devuelve el valor ordinal del parámetro.
Pred	Devuelve el valor decrementado del parámetro.
Succ	Devuelve el valor incrementado del parámetro.

Tabla 6.5 : Funciones ordinales predefinidas.

Dentro de los aspectos importantes que hay que señalar de los tipos enumerativos tenemos :

1.- El valor ordinal del primer elemento de un tipo enumerativo es cero, es decir, ORD(lun), ORD(ENE) y ORD(letra) toman el valor ordinal de uno.

2.- No se permite con los tipos enumerativos el uso de los procedimientos predefinidos estándar de entrada y

salida READ y WRITE; Si se utilizan, el compilador de Pascal detecta el error y regresa el mensaje : E/S no permitida (I/O not allowed)

3.- Se puede definir un tipo subrango de un tipo enumerativo ya declarado. Por ejemplo :

```
type
    mes_nac = (ENE,FEB,MAR,ABR,MAY,JUN,JUL,AGO,
               SEP,OCT,NOV,DIC);
    mes_primavera = MAR..MAY;
```

TIPOS ESTRUCTURADOS.

Pascal soporta el uso de los tipos estructurados, los cuales describen la estructura de los datos usados por el programador. Cuando se define un tipo estructurado, el compilador recibe la información y se prepara para asignar memoria y detectar errores. Se puede trabajar con toda la estructura o con los tipos individuales que la conforman. Pascal soporta cinco estructuras de datos : arreglos (array), registros (record), cadenas (string), conjuntos (set) y archivos (file). En este trabajo se van a tratar los arreglos (array), los cuales son los de mayor uso en otros lenguajes de programación; para ver las otras estructuras consulte un manual de Pascal.

ARREGLOS (ARRAY).

Para conceptualizar la estructura de un arreglo, hay que pensar en una tabla de variables ordenadas del mismo tipo, por ejemplo la hoja de calificaciones de estudiantes :

NOMBRE	CAL1	CAL2.....CALn	PROM
ALPIZAR	8.0	7.6. .....9.3	

BONILLA	4.6	6.3.	.....7.4
CORDERO	6.8	6.0.	.....7.5
FLORES	9.5	10.0.	.....9.0
JIMENEZ	7.6	6.3.....	.6.0
MORALES	8.0	4.6.	.....6.8
ZUÑIGA	8.4	6.7..	.....7.5

Si omitimos el nombre de los alumnos obtenemos un arreglo (tabla) donde las variables son las calificaciones de los alumnos, la cual se encuentra ordenada por columnas (cal1, cal2,...caln) y por renglones uno por cada alumno; las variables son del mismo tipo (reales). Donde n es un entero positivo.

En lugar de calificaciones, las columnas pueden ser por ejemplo las edades (el peso) de los trabajadores de un hospital y los renglones cada uno de los centros de salud de un estado.

Para definir un tipo estructurado array (arreglo) hay que determinar el número de elementos que lo conforman y el tipo base. El tipo base de un arreglo puede ser cualquiera de los tipos simples o un tipo array definido anteriormente, es decir, se puede declarar un array multidimensional. El número de elementos se encuentra determinado por su número índice y debe ser especificado cuando se declara el arreglo.

Los tipos estructurados array (arreglo) se definen en la zona de tipos (type).

SINTAXIS ARRAY (ARREGLO) :

```
type
  identificador = array[indice,indice2,...] of tipo;
```

-identificador : Es el nombre que se le da al tipo arreglo.

-array[ ]: Palabra que define el tipo arreglo.

-índice : Determina el número de elementos del arreglo, el cual puede ser de tipo subrango, es decir, que se puedan enumerar sus elementos; por ejemplo: los tipo booleano, entero y carácter.

-índice2 : Indica lo mismo que índice; es opcional y se pueden declarar tantos como sean necesarios; se separan por comas; el número de índices indica la dimensión del arreglo.

-of tipo : Indica el tipo de datos del arreglo, puede ser simple, o un tipo array definido anteriormente.

Para comprender la utilidad de los arreglos, vamos a obtener el promedio de los alumnos utilizando los tipos simples. Supongamos que n = 5, es decir, existen 5 calificaciones. Analizando la zona de declaración de variables del programa anterior, se observa que si el número de calificaciones (columnas) es muy grande, por ejemplo n = 1000 hay que declarar 1001 variables.

```
program promedio_simple;
var
    cal1,cal2,cal3,cal4,cal5,promedio : real;
    continuar : char;
        (* Si n = 1000, entonces hay que *)
        (* declarar 1001 variables *)
begin (* promedio_simple *)
    continuar := 'S';
    while upcase(continuar) = 'S' do
        (* la función upcase(car) devuelve car en mayúscula *)
        (* si car pertenece al rango 'a'..'z' *)
      begin
        clrscr;
        writeln('Escribe las 5 (n) calificaciones :');
        readln(cal1);
        readln(cal2);
        readln(cal3);
        readln(cal4);
        (* para n = 1000, hay que escribir 1000 *)
        (* sentencias de este tipo *)
        readln(cal5);
        promedio := (cal1 + cal2 + cal3 + cal4 + cal5) / 5;
        writeln ('El promedio es : ',promedio);
        write('Pulse << S >> para continuar, otra tecla P/SALIR ');
        readln(continuar);
      end
end.
```

Utilizando el tipo estructurado array (arreglo), la declaración se simplifica :

```
const
    cal_max = 1001; (* Declaración de constante *)
type
    caln = 1..cal_max; (* Declaración tipo subrango *)
    calificacion = array[caln] of real;
    (* Declaración de ARREGLO (ARRAY) *)
var
    califica : calificacion;

(* Declaración de la var. califica *)

(* tipo arreglo calificación *)
```

Suponiendo que hay 1000 calificaciones, entonces, ¿ por qué le asignamos a la constante cal_max el número 1001 ? La respuesta es muy sencilla : en la variable califica[1001] se va a depositar el promedio de cada alumno. Para hacer referencia a la primera calificación, a la segunda, a la n-ésima consideramos que :

califica[1] equivale a cal1
califica[2] equivale a cal2
.....
califica[n] equivale a caln

Es decir, califica[1] contendrá las calificaciones de la columna 1, califica[2] de la columna 2 y así sucesivamente.

Donde los números que se encuentran entre los corchetes son en este caso enteros positivos, lo que nos permite utilizar el ciclo for (desde) para recorrer todo el arreglo, por ejemplo el promedio se obtiene como :

```
promedio := 0;
for indice := 1 to cal_max do
   begin
      if indice = cal_max then
         (* indice toma el valor 1001 *)
         califica[indice] := promedio / (cal_max - 1)
      else
         promedio := promedio + califica[indice];
   end;
```

El tipo arreglo calificacion se puede declarar como :

```
type
      calificacion = array[1..1001] of real;
var
      califica : calificacion;
```

Con esta forma de declarar un array (arreglo) se escribe menos, pero al darle mantenimiento al programa (modificar el número de calificaciones 1001 en cada lugar en el que se encuentre escrito en el programa) se pierde mucho tiempo y puede conducir a errores. Con la primera forma se modifica una vez.

La forma en la que se definió el tipo arreglo calificación nos permite obtener el promedio de un alumno; para obtener el promedio de todos, es decir, recorrer todos los renglones, tenemos que declarar un arreglo de dos dimensiones y utilizar dos ciclos for (desde) anidados (un ciclo contenido en otro). Por ejemplo, la declaración del arreglo puede ser :

```
const
    cal_max = 1001; (* Declaración de constante *)
    num_alu = 50;
type
    caln = 1..cal_max; (* Declaración tipo subrango *)
    alun = 1..num_alu;
    calificacion = array[alun,caln] of real;
        (* Declaración de  ARREGLO de dos dimensiones *)
var
    califica : calificacion;
        (* Declaración de la var. califica *)
        (* tipo arreglo calificación *)
```

El primer índice (alun) representa los renglones, es decir, los alumnos y para recorrer toda la tabla se identifica con el ciclo for más externo; el índice caln representa las calificaciones (las columnas) y va indicar al ciclo for interno. En general, los índices van a ser representados de afuera hacia adentro.

El programa para calcular los promedios de los alumnos recorriendo toda la tabla es :

```
program promedio_arreglo;
const
   cal_max = 4; (* Declaración de constante *)
   num_alu = 3;
type
   caln = 1..cal_max; (* Declaración tipo subrango *)
   alun = 1..num_alu;
   calificacion = array[alun,caln] of real;
        (* Declaración de ARREGLO de dos dimensiones *)
var
   califica : calificacion;
   ind_ren,ind_col : integer;
   promedio : real;
        (* Declaración de la var. califica *)
        (* tipo arreglo calificación *)
begin
   clrscr;
(*     ------------- lectura de datos -----------    *)
   writeln(' LECTURA DE DATOS ');
   for ind_ren := 1 to num_alu do
   (* ciclo for externo, primer indice *)
      begin
         writeln('Alumno # : ',ind_ren:4 ); writeln;
         for ind_col := 1 to cal_max - 1 do
         (* ciclo for interno, segundo indice *)
            begin
  (*          ---------- CONTINUA SIG. PAG. ----------        *)
```

```
            write('CALIFICACION # : ',ind_col:4,' = ');
            readln(califica[ind_ren,ind_col]);
          end;
      end; (* lectura de datos *)
(*    ------------- CALCULO DEL PROMEDIO ---------- *)
   clrscr;
   writeln(' CALCULO DEL PROMEDIO ');
   for ind_ren := 1 to num_alu do
   (* ciclo for externo, primer indice *)
      begin
         promedio := 0;
         for ind_col := 1 to cal_max  do
         (* ciclo for interno, segundo indice *)
            begin
               if ind_col = cal_max then
                  begin
                    write('EL  PROMEDIO  DEL  ALUMNO    #  :  '
                    ,ind_ren:4,' ES : ');
                    califica[ind_ren,ind_col]  :=  promedio  /
                    (cal_max - 1);
                    writeln(califica[ind_ren,ind_col]:5:2);

                  end
               else
                  promedio  :=  promedio  +  califica[ind_ren,
                  ind_col];
            end; (* for ind_col *)
      end (* for ind_ren *)
end. (* program promedio *)
```

Como se puede observar, la declaración del tipo estructurado array permite efectuar programas más robustos y de un fácil mantenimiento.

ANEXO A.

OBJETIVOS GENERALES

El alumno :

a) Aprenderá a comunicarse con una computadora.

b) Se enfrentará a situaciones que involucren el uso de la computadora.

EL SENTIDO DEL CURSO Y SUS IMPLICACIONES

OBJETIVOS :

a) Conocerá los objetivos y las caracteristicas metodológicas del curso.

b) Conocerá el sistema de evaluación del curso.

CONTENIDO TEMATICO :

a) Objetivos y contenidos del curso.

b) Ubicación del curso dentro del área y su importancia en el mundo actual.

c) Sistemas de trabajo y evaluación.

TIEMPO APROXIMADO EN HORAS : una.

1.- HISTORIA DE LA COMPUTACION.

OBJETIVOS :

a) Conocerá los logros de diversas sociedades y personas que contribuyeron significativamente al campo del procesamiento de datos.

b) Conocerá las bases de la computación y su rápido desarrollo.

CONTENIDO TEMATICO :

a) De los dedos a los huesos de Napier.

b) Máquinas de cálculo automáticas.

c) De la tarjeta perforada al chip.

TIEMPO APROXIMADO EN HORAS : cuatro

2.- ESTRUCTURA Y FUNCIONAMIENTO DE LA COMPUTADORA.

OBJETIVOS :

a) Conocerá la estructura básica de una computadora.

b) Comprenderá el funcionamiento interno de la computadora

CONTENIDO TEMATICO :

a) Características de las computadoras.

b) Componentes físicos de una computadora.

c) Funcionamiento interno de la computadora.

TIEMPO APROXIMADO EN HORAS : ocho.

3.- SISTEMAS OPERATIVOS.

OBJETIVOS :

a) Aprenderá qué es un sistema operativo.

b) Conocerá qué hay diferentes sistemas operativos.

c) Conocerá los comandos de mayor uso del sistema operativo del equipo de cómputo existente en el CCHA...

CONTENIDO TEMATICO :

a) Sistema operativo.

b) Estructura de los sistemas operativos.

c) Compiladores e intérpretes.

d) Tipos de sistemas operativos.

e) Comandos del sistema operativo DOS.

TIEMPO APROXIMADO EN HORAS : 16

4 - PROCESADOR DE TEXTOS.

OBJETIVOS :

a) Aprender el uso de un editor de texto.

CONTENIDO TEMATICO :

a) Editor de Turbo Pascal.

TIEMPO APROXIMADO EN HORAS : 4

5.- TECNICAS PARA RESOLVER PROBLEMAS CON COMPUTADORA.

OBJETIVOS :

a) Valorará que para hacer un programa de un problema específico, es necesario saberlo resolver.

b) Aprenderá una metodología para escribir su solución de un problema con computadora.

CONTENIDO TEMATICO :

a) Metodología de Warnier-Orr.

TIEMPO APROXIMADO EN HORAS : ocho

6.- INTRODUCCION A LA PROGRAMACION ESTRUCTURADA VIA PASCAL.

OBJETIVOS :

a) Aprender las bases del lenguaje de programación PASCAL en el ambiente TURBO.

CONTENIDO TEMATICO :

a) Introducción.

b) Partes de un programa en Pascal.

c) Ciclos y sentencias de control.

d) Tipos definidos por el usuario.

TIEMPO APROXIMADO EN HORAS : 25

BIBLIOGRAFIA SUGERIDA A LOS ESTUDIANTES.

a) BASICA.

Goldstein, IBM-PC, Prentice-Hall.

Hennefeld J. Turbo Pascal con aplicaciones 3.0, 4.0 y 5.0, Grupo editorial Iberoamérica, 1991.

Reyes E., y Otros, Notas para el curso de computación, Academia de Matemáticas (Plantel Azcapotzalco).

San Martin R., Introducción a las computadoras, DGSCA.

b) CONSULTA.

Benice, D., Introducción a las computadoras y proceso de dastos, Prentice-Hall.

Kein W., Principios de los computadores.

Viso G., Introducción a la programación a través del lenguaje Pascal, Trillas, 1988.

Wilson A., Discoguia para IBM/PC, Osborne/McGraw-Hill, 1984.

Wood S., Turbo Pascal versión 3.0, Osborne/Mcgraw-Hill, 1990.

c) **SUJERENCIAS PARA LEER.**

Gonick L., Aprenda divirtiéndose computación, Harla.

Rusch R. La computadora máquina maravillosa, Editores Asociados.

Salvat, Los ordenadores, Colección G. T. No. 27.

BIBLIOGRAFIA.

Duffy T., Cuato herramientas de software, Grupo editorial Iberoamérica, 1990.

George H., Cibernética y biología, Alambra, 1968.

Heller S., Bits y bytes iniciación a la informática, Publicaciones Cultural, 1985.

Hennefeld J., Turbo Pascal con aplicaciones 3.0, 4.0 y 5.0, Grupo editorial Iberoamérica, 1991.

Hewlett P., Manual del usuario y referencia MS-DOS Versión 5.0, 1991.

Higgins D., Designing structured programs, Prentice-Hall, 1979.

Joyanes A., Fundamentos de programación algoritmos y estructura de datos, McGraw_Hill, 1990.

Murray III H., Pappas H., 80386/80286 Programación en lenguaje ensamblador, Osborne/McGrah-Hill, 1989.

Rose J., La revolución cibernética, Fondo de Cultura Económica, 1987.

Scheid F., Introducción a la ciencia de las computadoras, Serie Schaum McGraw-Hill, 1980.

Tenenbaum M., Augenstein J., Estructura de datos en Pascal, Prentice-Hall, 1986.

Viso G., Introducción a la programación a través del lenguaje Pascal, Trillas, 1988.

Wilson A., Discoguia para IBM/PC, Osborne/McGraw-Hill, 1984.

Wood S., Turbo Pascal versión 3.0, Osborne/Mcgraw-Hill, 1990.