UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Escuela Nacional de Estudios Profesionales
"ARAGON"

INGENIERIA

2
29

Computadoras y Programación Basica en Ingenieria

TESIS PROFESIONAL
Que para obtener el Título de:
INGENIERO EN COMPUTACION

Presentan:
ARCE GARCIA JOSE LUIS
JARDON AVILA GRACIELA

Febrero 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

VNIVERSIDAD NACIONAL
AVTÓNOMA DE
MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN
DIRECCION

GRACIELA JARDON AVILA
P R E S E N T E .

En contestación a su solicitud de fecha 26 de enero del año en curso, presentada por **José Luis Arce García** y usted, relativa a la autorización que se le debe conceder para que el señor profesor, **Ing. LUIS LORENZO JIMENEZ** pueda dirigirle el trabajo de Tesis denominado **"COMPUTADORAS Y PROGRAMACION BASICA EN INGENIERIA"**, con fundamento en el punto 6 y siguientes del Reglamento para Exámenes Profesionales en esta Escuela, y toda vez que la documentación presentada por usted reune los requisitos que establece el precitado Reglamento; me permito comunicarle que ha sido aprobada su solicitud.

Aprovecho la ocasión para reiterarle mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPIRITU"
San Juan de Aragón, Edo. de Méx., Febrero 3 de 1993
EL DIRECTOR

M en I CLAUDIO C. MERRIFIELD CASTRO

c c p Lic. Alberto Ibarra Rosas, Jefe de la Unidad Académica.
c c p Ing. Juan Gastaldi Pérez, Jefe de Carrera de Ingeniería en Computación.
c c p Ing. Manuel Martínez Ortiz, Jefe del Departamento de Servicios Escolares.
c c p Ing. Luis Lorenzo Jiménez, Asesor de Tesis.

UNIVERSIDAD NACIONAL AVENTMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN
DIRECCION

JOSE LUIS ARCE GARCIA
P R E S E N T E .

En contestación a su solicitud de fecha 26 de enero del año en curso, presentada por **Graciela Jardón Avila** y usted, relativa a la autorización que se le debe conceder para que el señor profesor, **Ing. LUIS LORENZO JIMENEZ** pueda dirigirle el trabajo de Tesis denominado **"COMPUTADORAS Y PROGRAMACION BASICA EN INGENIERIA"**, con fundamento en el punto 6 y siguientes del Reglamento para Exámenes Profesionales en esta Escuela, y toda vez que la documentación presentada por usted reune los requisitos que establece el precitado Reglamento; me permito comunicarle que ha sido aprobada su solicitud.

Aprovecho la ocasión para reiterarle mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPIRITU"
San Juan de Aragón, Edo. de Méx., Febrero 3 de 1992
EL DIRECTOR

M. en A. CLAUDIO C. MERRIFIELD CASTRO

c c p Lic. Alberto Ibarra Rosas, Jefe de la Unidad Académica.
c c p Ing. Juan Gastaldi Pérez, Jefe de Carrera de Ingeniería en Computación.
c c p Ing. Manuel Martínez Ortiz, Jefe del Departamento de Servicios Escolares.
c c p Ing. Luis Lorenzo Jiménez, Asesor de Tesis.

CCMC/AIR!elc.

CONTENIDO

Quiero agradecer a mis padres Guillermo y Ma. del Carmen, a mis hermanos, profesores y a todas aquellas personas que colaboraron para la realización de mis estudios.

JOSE LUIS

Quiero expresar mi agradecimiento a las autoridades y profesores de la ENEP ARAGON, en especial a todas aquellas personas que directa o indirectamente colaboraron no sólo en la realización de esta Tesis sino en toda nuestra preparación profesional.

A mis padres Ma. de la Luz y Pedro Luis, a mi hermana Judith por el apoyo y motivación que me dieron para la culminación de mis estudios.

Y especialmente a mi esposo Javier que con su amor y comprensión me apoyó y ayudó para la realización de este trabajo, así como a Stephany nuestra hija quien me alentó para concluir esta Tesis.

GRACIELA

INTRODUCCION

Cada vez que observamos las innovaciones tecnológicas en el campo de la computación, y en otros campos, vemos los sueños de los usuarios hechos realidad; sin embargo, nuestro quehacer cotidiano está conformado por la investigación, el análisis, la deducción y la experiencia, es por ello que al elaborar esta tesis, pusimos todo nuestro empeño, dedicación y conocimientos, esperando que sea de utilidad para todos y cada uno de los estudiosos de un tema que es apasionante "la computación".

Este trabajo de investigación forma parte de una ambicioso programa que están realizando los profesores y alumnos pasantes de la Escuela Nacional de Estudios Profesionales Aragón (ENEP ARAGON) de la Universidad Nacional Autónoma de México, el cual tiene como objetivos, por una parte que un número mayor de estos alumnos se titulen y por otro que se realicen proyectos que colaboren a la actualización y mejor preparación de los estudiantes que utilicen el material.

La intención de esta tesis es dar al estudiante de primer semestre del Area de Ingeniería de la materia "Computadoras y Programación Básica en Ingeniería" un panorama de lo que es el campo de la computación, cubriendo temas que aunque tratados en forma somera dan al alumno una idea clara de los conceptos que se utilizan.

El contenido de este libro está dividido en seis capítulos, en el primero se da una visión general del desarrollo histórico de las computadoras, abarcándose además conceptos básicos de computación, los diversos lenguajes de programación y el impacto de las computadoras en la sociedad y en nuestro país, así como sus tendencias a corto y mediano plazos.

El segundo capítulo se refiere a la arquitectura y funcionamiento de las computadoras digitales, donde se analizan las transformaciones y operaciones de números de los sistemas decimal, binario, octal y hexadecimal, los diferentes tipos de códigos que existen actualmente, los componentes de una computadora, así como los criterios para la selección de un equipo y las sistemas operativos.

En la tercera parte de la tesis se estudian los conceptos de algoritmos, secuencia, programa, programación estructurada, seudocódigo y estructuras de control.

El cuarto y quinto temas tratan sobre los lenguajes de programación Basic y Pascal, donde se tratan las generalidades e instrucciones más comunes de estos dos lenguajes, así como el manejo de archivos.

En el capítulo seis, se realiza una recopilación de los paquetes de computadora de uso más común.

Enseguida, se presenta la bibliografía utilizada para la elaboración de este trabajo de investigación, la cual debido a que el campo de la computación está evolucionando aceleradamente, es difícil tener a disposición todo lo que se ha publicado dentro del área de la computación; sin embargo, tratamos de que la literatura consultada fuera en su mayoría de reciente publicación, para que el estudiante que así lo desée, profundice en alguno o algunos de los temas aquí tratados.

CAPITULO I

DESARROLLO HISTORICO DE LAS COMPUTADORAS

1.1 CONCEPTOS BASICOS DE COMPUTACION

En el campo de la computación existen términos informáticos que es indispensable conocer previamente para el buen entendimiento de este ámbito. A continuación se mencionan algunos de ellos para entrar en materia y conforme se adentre en el presente trabajo, se irán definiendo otros conceptos para comprender mejor los temas que se tratan.

BIT

Un bit es una contracción de dígito binario (en inglés, binary digit), unidad mínima de información que es capaz de representar físicamente una máquina o un soporte de datos, que es uno de los dos dígitos 0 y 1, usados en la notación binaria.

Físicamente un bit se caracteriza por una celda de almacenamiento, un pulso, un punto magnético. De manera conceptual, puede concebirse un bit como un estado de bombilla eléctrica que puede estar encendida o apagada. Por convención, el estado "encendido" es el número 1, mientras que "apagado" corresponde al 0.

Los bits se agrupan en unidades que son procesadas y almacenadas al mismo tiempo por la computadora. Dependiendo de su longitud y su función, a estos grupos se le denominan caracteres, bytes o palabras.

BYTE

Es una contracción del inglés Binary Term, agrupación fundamental de información binaria formada por ocho bits. Es la unidad mínima que puede direccionarse, pero no la unidad mínima que puede tratarse. Es una unidad común de almacenamiento en un sistema de cómputo. Equivale a 8 bits o a un carácter de información. 50,000 bytes equivalen a 50,000 caracteres. Los bytes se emplean para hacer referencia al tamaño del hardware, el software o la información.

PALABRA

Es el número de bits que constituye una unidad común de información en una computadora en particular. La mayoría de las primeras computadoras usaban palabras que eran de un byte u ocho bits de longitud. A estos tipos, que aún prevalecen, se les llamó máquinas de ocho bits. Existen computadoras personales de 16 y 32 bits. La mayoría de las computadoras

UNIDAD DE SALIDA

Dispositivo por medio del cual la computadora transmite información al usuario.

UNIDAD CENTRAL DE PROCESO

Es el control central o "cerebro" de la computadora. La conforman la unidad de Control y Procesamiento, también se considera como la parte de un sistema de computación que contiene aquellos circuitos que controlan las interpretaciones y ejecuciones de instrucciones.

La unidad central de proceso realiza los pasos siguientes :

- Sigue la posición de las instrucciones que se están ejecutando.
- Recupera de la memoria las instrucciones y las interpreta o decodifica.
- Ejecuta las instrucciones utilizando las unidades de hardware de aritmética y lógica, de prueba y corrimiento.
- Dirige el hardware de entrada/salida cuando el programa requiere instrucciones de E/S.

UNIDAD DE CONTROL

Componente de hardware que dirige a la computadora y a las actividades periféricas; la unidad de control de un procesador realiza las funciones primarias de la computadora. Localiza, analiza y dirige la ejecución de todas las instrucciones de un programa. Las unidades de control periféricas, obedeciendo las señales del UCP (Unidad Central de Proceso), realizan las transferencias físicas de información entre los periféricos y la UCP. Esta unidad tiene la función de coordinador de la computadora, es decir, dirige y supervisa la operación de todo el sistema.

UNIDAD DE PROCESAMIENTO O UNIDAD ARITMETICA Y LOGICA

La unidad aritmética y lógica es un conjunto de circuitos de una computadora que realiza las funciones aritméticas (suma, resta, multiplicación y división), las funciones lógicas (Y, NO y O) y las funciones de comparación (> mayor que; < menor que; = igual que, >= mayor o igual que; <= menor o igual que), los resultados de los cálculos se almacenan nuevamente en la memoria y los de las comparaciones son utilizados por las instrucciones de prueba que siguen a las de la comparación.

La unidad aritmética y lógica contiene acumuladores, sumadores, circuitos comparadores, registros para conservar operandos y resultados; así como circuitos de corrimiento y secuencia para realizar multiplicación, división, y otras operaciones matemáticas. También ejecuta la mayoría de los comandos de procesamiento interno de la computadora.

grandes emplean palabras de 32 y 64 bits.

COMPUTADORA

Es un dispositivo electrónico capaz de tratar la información según las instrucciones que le suministra un programa diseñado por el usuario a altas velocidades. En esencia, la computadora está formada por dos grandes partes: Hardware y Software.

HARDWARE

Son los equipos mecánicos, electromecánicos y electrónicos que forman la estructura física de la computadora. El hardware de la máquina es el encargado de efectuar físicamente los procesos de captación de información, operaciones aritméticas y lógicas, almacenamiento de información y obtención de resultados; para cada una de estas instrucciones existe, dentro de la computadora, un elemento que fue construido especialmente para realizarlas.

SOFTWARE

Son todos aquellos programas que están escritos en un lenguaje apropiado a la estructura física de las computadoras y con los cuales es posible utilizarlas. Es la estructura lógica de la computadora.

Una computadora es un sistema de hardware (son los circuitos electromecánicos que constituyen el sistema de computación), que cumple operaciones aritméticas, manipula datos (generalmente en forma binaria) y toma decisiones.

REPRESENTACION SIMBOLICA

Hay varios tipos de computadoras, pero cada una puede ser descompuesta en las mismas unidades funcionales.

Cada unidad cumple funciones específicas, y todas las unidades trabajan juntas para realizar las instrucciones dadas en el programa.

Las unidades de las que consta una computadora son las siguientes:

UNIDAD DE ENTRADA

Método, medio o dispositivo utilizado para introducir información en la computadora o para transferirla de un sistema de computación del mundo exterior. Ejemplos: Teclado, discos, cintas, etc.

UNIDAD DE SALIDA

Dispositivo por medio del cual la computadora transmite información al usuario.

UNIDAD CENTRAL DE PROCESO

Es el control central o "cerebro" de la computadora. La conforman la unidad de Control y Procesamiento, también se considera como la parte de un sistema de computación que contiene aquellos circuitos que controlan las interpretaciones y ejecuciones de instrucciones.

La unidad central de proceso realiza los pasos siguientes :

- Sigue la posición de las instrucciones que se están ejecutando.
- Recupera de la memoria las instrucciones y las interpreta o decodifica.
- Ejecuta las instrucciones utilizando las unidades de hardware de aritmética y lógica, de prueba y corrimiento.
- Dirige el hardware de entrada/salida cuando el programa requiere instrucciones de E/S.

UNIDAD DE CONTROL

Componente de hardware que dirige a la computadora y a las actividades periféricas; la unidad de control de un procesador realiza las funciones primarias de la computadora. Localiza, analiza y dirige la ejecución de todas las instrucciones de un programa. Las unidades de control periféricas, obedeciendo las señales del UCP (Unidad Central de Proceso), realizan las transferencias físicas de información entre los periféricos y la UCP. Esta unidad tiene la función de coordinador de la computadora, es decir, dirige y supervisa la operación de todo el sistema.

UNIDAD DE PROCESAMIENTO O UNIDAD ARITMETICA Y LOGICA

La unidad aritmética y lógica es un conjunto de circuitos de una computadora que realiza las funciones aritméticas (suma, resta, multiplicación y división), las funciones lógicas (Y, NO y O) y las funciones de comparación (> mayor que; < menor que; = igual que, >= mayor o igual que; <= menor o igual que), los resultados de los cálculos se almacenan nuevamente en la memoria y los de las comparaciones son utilizados por las instrucciones de prueba que siguen a las de la comparación.

La unidad aritmética y lógica contiene acumuladores, sumadores, circuitos comparadores, registros para conservar operandos y resultados; así como circuitos de corrimiento y secuencia para realizar multiplicación, división, y otras operaciones matemáticas. También ejecuta la mayoría de los comandos de procesamiento interno de la computadora.

UNIDAD DE MEMORIA PRINCIPAL

Es un dispositivo en el cual deben estar almacenados los programas que son ejecutados por la unidad central de procesamiento, así como los datos sobre los cuales se realizan las operaciones específicas del programa.

La memoria principal es comúnmente en una secuencia lineal de bits agrupados en bytes o palabras de longitud fija.

Esta unidad sirve de almacén temporal de información, ya sea que entre al sistema, o sea producto de cálculos internos.

DIAGRAMA DE BLOQUES DE UNA COMPUTADORA

La operación de una computadora puede ser globalmente descrita por los siguientes pasos:

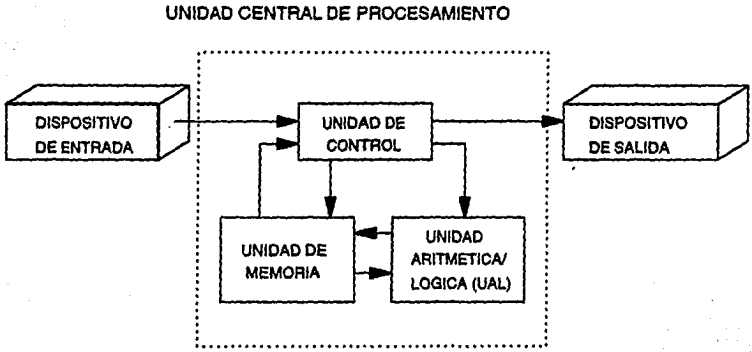
- Acepta información (programas y datos) a través de la unidad de entrada y la transfiere a la memoria.
- La información almacenada en la memoria es traída a la unidad aritmética y lógica para ser procesada.
- La información procesada se deja en la computadora a través de la unidad de salida.
- Todas las actividades dentro de la máquina están bajo el mando de la unidad de control.

En todos estos pasos la computadora es dirigida a través de un programa que fue almacenado previamente en la memoria. Un programa contiene un conjunto de instrucciones para dirigir a la computadora en la realización de una tarea.

Cada instrucción en memoria es traída a la unidad central de proceso (UCP), donde se ejecuta la operación específica.

Además de las instrucciones, es necesario usar algunos datos como operandos, los cuales también son almacenados en memoria.

En la figura 1.1 se muestra un diagrama de bloques de una computadora. En él podemos distinguir los siguientes componentes:



COMPUTADORAS ANALOGICAS, DIGITALES E HIBRIDAS

COMPUTADORAS DIGITALES

Son aquellas computadoras que procesan información representada por combinaciones de datos discretos o discontinuos; y todas las formas de información cifradas en binario, pueden ser aceptadas a perpetuidad por un sistema. Los sistemas basados en digitales realizan secuencias de operaciones aritméticas y lógicas, no solamente con datos sino con su propio programa.

Las características de la computadora digital son las siguientes: operación completamente automática y a muy alta velocidad, gran capacidad de memoria para almacenar información, ejecutar las instrucciones especificadas, programación sencilla para la recuperación de información, resolución numérica de variables obteniendo la precisión deseada, y la capacidad de tomar decisiones lógicas.

Las desventajas que presenta son que se opera en forma discontinua y discretizada, o sea en intervalos de tiempo específicos, ya que se trabaja con dígitos, y los cálculos entre intervalos pueden implicar errores apreciables y dificultad para obtener una solución estable. Si se desean resultados gráficos, como se opera con números se aumenta el tiempo de máquina, lo que también ocurre cuando el problema incrementa sus dimensiones y complejidad. La escala de tiempo en la que ocurre un proceso no se puede alterar.

Sin embargo, se pueden resolver problemas de cualquier tamaño si se cuenta con tiempo de máquina suficiente.

COMPUTADORAS ANALOGICAS

Son computadoras que aceptan y procesan señales electrónicas y análogas a las del mundo real. Las entradas y salidas de las computadoras analógicas son señales continuas tales como las fluctuaciones de voltaje; este hecho contrasta con la computadora digital, que sólo puede aceptar información cifrada en binario.

Se llama así por su analogía, que existe entre situaciones físicas, químicas, biológicas, etc., además acepta datos de aparatos de medición en forma de variaciones continuas (termómetro, barómetro, sismógrafo, multímetro, etc.).

Esta computadora es una colección de dispositivos electrónicos que pueden efectuar operaciones matemáticas básicas como suma, resta, multiplicación, división y generación de funciones. Es muy útil en el estudio de sistemas de variación en el tiempo, ya que la computadora simula fácilmente el comportamiento dinámico de cualquier sistema. Por esta razón la computadora analógica es básicamente una herramienta para la investigación y también para la enseñanza.

La computadora analógica tiene la característica de poder cambiar la escala de tiempo y hacer lentas las soluciones rápidas o acelerar las soluciones lentas, lo que resulta en un mejor análisis y un costo menor. Estos casos se presentan por ejemplo para analizar explosiones que ocurren en una fracción de segundo y que tienen que hacerse más lentas en la computadora para estudiarlas.

En algunos sistemas físicos es imposible o muy peligroso estudiar las condiciones críticas de operación del sistema, como en el caso de un reactor nuclear o de un red eléctrica. Sin embargo, el modelo de computadora puede llevarse al límite de destrucción y proporcionar la única manera segura de analizar con detalle el comportamiento del sistema en situaciones críticas.

En resumen, la computadora analógica, tiene la ventaja de alta velocidad de operación, facilidad de cambio de parámetros y observación instantánea de los efectos de estos cambios de manera que el operador puede experimentar realmente el modelo en estudio. El equipo opera en paralelo, de manera que el tiempo de solución es independiente de la complejidad del problema; además se puede cambiar la escala de tiempo.

Los resultados se presentan en forma gráfica y no se aumenta el tiempo de máquina con esta representación. No hay necesidad de aprender lenguajes de programación.

Se tienen desventajas en el tamaño de los problemas que se pueden resolver, en la precisión de los componentes del problema y en tener una capacidad de memoria limitada, lo que impide considerar decisiones lógicas con las soluciones obtenidas, para decidir la secuencia de cálculo posterior. Tampoco se cuenta con métodos de programación automáticos.

Algunas aplicaciones de las computadoras analógicas se usan en las industrias eléctrica, del acero, procesos químicos.

COMPUTADORAS HIBRIDAS

Son computadoras digitales que procesan señales analógicas; una computadora híbrida realiza un procesamiento digital en entradas analógicas que han sido transformadas a la forma digital. Estas computadoras son utilizadas en el control de procesos y en robótica.

La computadora híbrida es una combinación de computadoras analógicas y digital, con un sistema adecuado de comunicación y operación, ya que la analógica opera básicamente en forma continua y en paralelo, mientras que la digital trabaja en forma discretizada y secuencial.

El problema de comunicación no es sólo la conversión de analógica a digital y viceversa, sino también la solución de los complejos problemas de tiempo para asegurar que la transferencia de información entre los dos sistemas se realice eficientemente. Las computadoras híbridas se pueden utilizar en problemas de ecuaciones diferenciales o sistemas de ecuaciones diferenciales ordinarias o parciales, especialmente si se aplican funciones de variables múltiples y análisis estadísticos.

1.2 DESARROLLO HISTORICO DE LAS COMPUTADORAS DIGITALES

INVENTO DE LA ESCRITURA Y ARITMETICA

Desde tiempos remotos, el hombre se ha visto en la necesidad de tener algún elemento que lo auxilie en la labor de identificar y cuantificar sus pertenencias. En un principio comenzó por utilizar los dedos de sus manos, como un medio para contar sus animales, objetos de caza, etc., no obstante que este método lo ayudaba, frecuentemente era insuficiente, por lo cual ideó usar piedras, palillos, marcas en los troncos de los árboles y cavernas, como indicadores de esa cuantificación. El proceso consistía en realizar una correspondencia uno a uno entre sus pertenencias y los elementos del medio utilizado.

A medida que el hombre fue dominando el medio ambiente, adquirió cada vez más pertenencias y el método de correspondencia resultó evidentemente ineficaz; por lo cual, hubo que idear alguna herramienta que le permitiera realizar esta labor.

Entre los primeros elementos que el hombre desarrolló encontramos:

EL ABACO

El ábaco, es probable que haya sido el primer dispositivo mecánico para contar. Se ha podido determinar que su antigüedad se remonta cuando menos 2,600 años A. C., en China. Su eficiencia ha resistido la prueba del tiempo. Ya que sigue utilizándose tanto para explicar a los escolares los principios para contar como en las modernas aplicaciones de negocios.

TABLAS DE LOGARITMOS. (1614)

La dificultad para realizar operaciones de multiplicación y división motivó a John Napier a crear un nuevo método que redujera notablemente ese trabajo; fue así como surgieron las tablas de logaritmos, a través de las cuales es posible calcular multiplicaciones en forma sencilla y rápida, y con ellas puede traducirse la multiplicación a sumas de logaritmos y la división a restas de logaritmos, pero había que generar las tablas y sus antilogaritmos e imprimirlas, lo cual representaba un enorme trabajo, que fue realizado por otro compañero de Napier, H. Briggs; no obstante los esfuerzos realizados, las tablas tuvieron errores que fueron detectados tiempo después.

REGLA DE CALCULO. (1630)

Poco tiempo después de que Napier inventó los logaritmos, surgió otro nuevo invento, menos exacto, pero más fácil de utilizar, la regla de cálculo, inventada por el matématico inglés William Oughtred; trabaja con base en medir longitudes entre dos reglitas que guardan relación entre sí, utilizando la escala logarítmica. Esta herramienta ha sido sumamente usada, inclusive hoy en día, ya que aproxima con suficiente exactitud los resultados de las operaciones que se realizaban con ella, y no es sino hasta esta década que empezó a ser desplazada por las calculadoras electrónicas de bolsillo.

LA MAQUINA DE PASCAL. (1642)

Blaise Pascal, filósofo y matemático francés inventó la primera máquina de sumar y restar. Consistía en un sistema de ruedas engranadas, en cada una de las cuales estaban marcados los dígitos del cero al nueve; cada vez que una rueda completaba una vuelta, la siguiente a la izquierda caminaba un elemento y así sucesivamente, dando como resultado la suma de varias cantidades, la cual se desplegaba después en casillas colocadas sobre cada rueda de la máquina. La llamó "Pascalina"

LA MAQUINA DE LEIBNIZ. (1681)

Gottfried Wilhelm Leibniz, filósofo y matemático alemán, desarrolló y mejoró el dispositivo creado por Blaise Pascal, logrando que la máquina fuese capaz de realizar las cuatro operaciones básicas, es decir, la suma, la resta, la multiplicación y la división en forma mecánica, esto es por palancas y engranajes.

TARJETA PERFORADA. (1804)

En 1804, Joseph Marie Jacquard, inventó una serie de tarjetas conteniendo perforaciones para atender patrones uniformes en la construcción de ropa. Jacquard introduce la automatización y es el creador del sistema de tarjeta perforada.

LAS MAQUINAS DE BABBAGE. (1812-1834)

Uno de los más notables contribuyentes al desarrollo de las máquinas para cálculos, fue el inglés Charles Babbage, quien obtuvo el apoyo del gobierno para construir una máquina que fuera capaz de efectuar cálculos complejos sin la intervención del ser humano y de esta forma eliminar los errores en que frecuentemente se incurría. Esta máquina trabajaba con base en el "método de las diferencias" y fue creada para corregir los errores de las tablas de logaritmos; sin embargo, este trabajo no pudo concluirse, ya que el Gobierno Británico después de haber gastado 17,000 libras le suspendió la subvención.

Babbage propuso también la construcción de una máquina analítica para colocar tablas astronómicas matemáticas y otras tablas con rapidez y seguridad.

Desgraciadamente la tecnología de principios del siglo XVIII no estaba suficientemente avanzada para producir los componentes que necesitaban. El diseño básico de Babbage incorporaba muchas de las características de una computadora moderna, tenía que ser capaz de sumar, restar, multiplicar y dividir ejecutando una serie de operaciones en una determinada secuencia, reteniendo los resultados de un cálculo para usarlos en el siguiente y "ramificándose" eligiendo entre alternativas. Debería tener un "almacén" o memoria de un millar de números de 50 dígitos y poder imprimir sus resultados.

Lady Ada Augusta, condesa de Lovelace trabajó con Babbage y es conocida como "la primera programadora" debido a su trabajo sobre los tipos de instrucciones que tendría que darse a la máquina analítica para hacerla funcionar.

EL ALGEBRA DE BOOLE. (1854)

El matemático George Boole desarrolló, en 1854, un sistema para representar las proposiciones lógicas por medio de símbolos matemáticos. Con estos emblemas y unas pocas reglas, podía determinarse si una proposición era, en sentido lógico, verdadera o falsa. Sus métodos no tuvieron entonces amplia aceptación, pero constituyen en la actualidad la base de la capacidad lógica de las computadoras.

TARJETAS PERFORADAS. (1880-1890)

El reunir los datos de los censos, era en el siglo XIX un proceso sumamente laborioso. Para compilar los datos del censo de 1880 en Estados Unidos, se necesitaron siete años y resultaba obvio que en poco tiempo, tomaría, más de diez años compilar los del siguiente. La solución del problema fue presentada por el Dr. Herman Hollerith, quien desarrolló un sistema mediante el cual los datos del censo se perforaban a mano en tarjetas y se contaba con una tabuladora construida por él para tal efecto, ya que la mayoría de las respuestas eran "si" o "no"; podrían ser representadas por la presencia o ausencia de una perforación en un lugar determinado de la tarjeta.

Con este sistema, para el censo de 1890, se necesitaron únicamente dos años y medio para reunir los datos, a pesar de que la población se había incrementado en un 25%. Las tarjetas utilizadas para este fin contaban con cuarenta y cinco columnas perforables en forma circular. Hollerith fundó la Tabulating Machine Company para fabricar equipo de tarjetas perforadas y

venderlo a empresas y al gobierno. En 1837 se realizó el primer censo en Rusia con los Tabuladores de Hollerith. En 1911 se fusiona con otras compañías, creándose la Computer-Tabulating-Recording Company. En 1924 cambia su nombre por el de International Business Machines Corporation (I.B.M.). En 1928, la I.B.M. inició el uso de tarjetas perforadas, con orificios rectangulares, lo que permitió aumentar a ochenta las columnas.

Hollerith desarrolló, además, máquinas capaces de ordenar automáticamente dichas tarjetas comparándolas entre sí y escribiendo los resultados en forma legible.

POULSEN. (1900)

Valdemar Poulsen inventa en 1900 la cinta magnética y el tambor magnético.

MAQUINAS REGISTRADORAS. (XIX)

Durante el siglo XIX las necesidades de empresas comerciales, especialmente en los Estados Unidos, hicieron que se desarrollaran las calculadoras electromecánicas y las máquinas registradoras, para llevar a cabo procesos de registro contable.

EL BULBO. (XX)

En el siglo XX, con el descubrimiento de la electricidad, se facilitó notablemente la realización de un gran número de inventos, entre los cuales se encuentra el bulbo, que fue considerado un elemento importante en el desarrollo de las computadoras.

MAQUINA DE TURING. (1937)

Alan M. Turing matemático inglés, en 1937 demostró que con un pequeño número de instrucciones simples se podía resolver cualquier problema. De acuerdo con lo anterior, una máquina rápida es capaz de resolver problemas complejos. El modelo matemático que Turing propone (conocida ahora como la "Máquina de Turing"), inicia la Teoría Matemática de la Computación.

LA LOGICA DE LA CONMUTACION. (1938)

Claude Shannon ingeniero electricista, aplicó los métodos del álgebra booleana a la representación sistemática de complejas redes de conmutación. Los resultados obtenidos por Shannon simplificaron la enseñanza y la investigación en el campo del diseño de circuitos del tipo que luego habría de usarse en la computadora moderna.

LA "MARK 1". (1937-1944)

La primera máquina que realizó el sueño de Babbage fue la MARK 1, o ASCC (Automatic Sequence Controlled Calculator), construida en la Universidad de Harvard por Howard Aiken,

a finales de los años 30 y principios de los 40, con el apoyo de I.B.M.; fue la primera computadora electromecánica automática. Era capaz de ejecutar largas secuencias de operaciones codificadas previamente, registradas en una cinta perforada de papel y calculaba los resultados con ayuda de las unidades de almacenamiento (memoria), de control y aritméticas. No obstante, la máquina MARK 1 era relativamente lenta, ya que su velocidad de operación dependía de la rapidez de sus numerosos componentes (alrededor de 750,000), constituidos por ruedas engranadas operadas automáticamente y accionados por switches del mismo tipo, alimentados por una corriente eléctrica; fue empleada durante quince años para realizar cálculos astronómicos.

LA MAQUINA ABC. (1939)

La primera persona en diseñar y construir una máquina de cálculo fue John Atanasoff en la Iowa State University. Deseaba una máquina que resolviera ecuaciones lineales simultáneas. En 1939, Atanasoff y Cliford Berry construyeron una máquina a la que llamaron ABC, de Atanasoff Berry Computer (Computadora de Atanasoff y Berry). Los principales componentes electrónicos de la ABC eran 300 tubos de vacío, o bulbos. La máquina podía resolver un conjunto de 29 ecuaciones simultáneas con 29 variables. La ABC fue la primera computadora electrónica digital. No era, sin embargo, una computadora de propósito general a gran escala. Podía hacer sólo un trabajo: resolver tipos limitados de problemas matemáticos.

LA "ENIAC" Y LA "Z4". (1943-1945)

Las primeras computadoras electrónicas en el mundo de propósito general a gran escala fueron desarrolladas en la Universidad de Pennsylvania, por el Dr. John W. Mauchly y J.P. Eckert, y en el German Aircraft Research Institute por Konrad Zuse; la máquina americana conocida como ENIAC (Electronic Numerical Integrator and Calculator) no tenía partes mecánicas, utilizaba en su lugar bulbos, alrededor de 18,000; pesaba más de treinta toneladas, con 30 m. de longitud, 3 m. de alto y 90 cm de profundidad y consumía 150 Kw de electricidad, era capaz de ejecutar 5,000 operaciones por segundo y fue usada en la resolución de problemas de balística y aeronáutica.

Su mayor mérito fue tener gran cantidad de elementos electrónicos y hacerlos funcionar simultáneamente. Era sumamente grande y usaba demasiados bulbos, por lo que en poco tiempo, se calentaba en extremo.

No obstante, podía completar en un día aquellos procesos que requerían 30 días en las computadoras electromecánicas.

La máquina alemana llamada "Z4", fue resultado de dos proyectos anteriores ("Z2" y "Z3"), y destruida en la segunda guerra mundial. Se utilizó para desarrollar bombas que eran lanzadas desde aviones. Su inventor trabajó aislado a otros pioneros de la computación; sin embargo, reinventó independientemente la idea de Babbage de una máquina programada. Al final de la guerra sus captores no pudieron hacer uso apropiado de sus conocimientos, ya que él hablaba poco inglés y sus interrogadores poco alemán; por consiguiente, no pudo convencerlos de la importancia de su trabajo, el cual quedó olvidado por más de diez años.

LA EDVAC. (1945-1952)

En la misma época surgió un ciclo de conferencias sobre "Teoría y Técnicas de las Computadoras Electrónicas Digitales". En este curso se presentó una serie de ideas elaboradas por un grupo de investigadores dirigidos por John Von Newmann.

En ellas se analizaron los problemas de diseño de la computadora y se hicieron propuestas para un nuevo tipo de máquinas que fuera más pequeña y potente que la ENIAC.

El mismo equipo que desarrolló la ENIAC, Eckert y Mauchly, construyó una segunda máquina mayor que ésta, con el nombre de EDVAC (Electronical Discrete Variable Automatic Computer) entre 1945 y 1952, capaz de realizar operaciones aritméticas con números binarios y almacenar instrucciones internamente.

Un poco más adelante la Compañía Remington Rand fundada por ellos mismos, desarrolló la UNIVAC 1 (Universal Automatic Computer), que fue la primera computadora de uso comercial, apareciendo en el año 1951. Entre sus características principales se encuentran el uso de cinta magnética para la entrada y salida de datos, aceptar y procesar datos alfabéticos y numéricos, así como el uso de un programa especial capaz de traducir programas en un lenguaje particular a lenguaje de máquina.

LA MAQUINA DE PROGRAMA ALMACENADO: LA ARQUITECTURA DE VON NEUMANN

En 1946 Von Neumann y Eckert Mauchly desarrollaron la máquina "EDVAC", primera computadora con programa almacenado en memoria.

El concepto de programa almacenado consiste en que las instrucciones y datos son acumulados, entremezclados en un mismo medio de almacenamiento.

También importante en la máquina de Von Neumann es el registro contador de programa (PC) que es usado para efectuar la siguiente instrucción a ser ejecutada. Este registro es incrementado cada vez que una instrucción se obtiene de la memoria; en tal forma que siempre contenga la dirección de la siguiente instrucción a consumarse.

La máquina de Von Neumann consiste de cinco partes:

1.- LA UNIDAD DE ENTRADA
2.- LA UNIDAD DE SALIDA
3.- LA UNIDAD DE MEMORIA
4.- LA UNIDAD DE CONTROL
5.- LA UNIDAD ARITMETICA Y LOGICA

Otra característica básica de esta máquina es que el flujo de información (datos e instrucciones) de la unidad de entrada a la memoria y de la unidad de memoria a la unidad de salida es a través de la unidad aritmética y lógica.

1.3 GENERACION DE COMPUTADORAS DIGITALES. LENGUAJES DE PROGRAMACION

Como se mencionó anteriormente, la computación ha tenido un desarrollo histórico, debido a avances científicos y tecnológicos que se han dado a través del tiempo.

Dentro de este desarrollo histórico se ha clasificado a la evolución de las computadoras en generaciones.

A continuación se describen las características de las generaciones de las computadoras.

PRIMERA GENERACION. (1946-1959)

Estas máquinas formaron lo que se llamó la primera generación de computadoras, utilizaban bulbos de vacío (dispositivo electrónico formado por dos electrodos encerrados en un tubo al vacío), como componentes básicos de sus circuitos internos; como consecuencia eran demasiado voluminosas (en realidad eran gigantescas), consumían mucha energía y producían tanto calor que fue preciso establecer rígidos controles en cuanto al aire acondicionado y temperatura, no eran tan confiables como se había esperado, eran rápidas pero no lo suficiente, tenían capacidad de almacenamiento interno pero limitada (Cuadro 1.1).

Eran capaces de realizar hasta 1,000 instrucciones por segundo y capacidad de memoria de hasta 20,000 posiciones (palabras).

SEGUNDA GENERACION. (1959-1964)

La segunda generación estuvo determinada por la invención del transistor (dispositivo electrónico que actúa como un interruptor, ya que decide el paso o no de la corriente entre dos puntos en función de la tensión aplicada a un tercero). Tecnología SSI (Short Scale of Integration, escala pequeña de integración).

Debido a su tamaño pequeño y al hecho a que los transistores disipaban menos calor que los tubos de vacío; fue posible el procesamiento más compacto, por lo que hubo una reducción sustancial del consumo de energía y del volumen ocupado por las máquinas, aumento de fiabilidad y velocidad de cálculo de las instrucciones (un millón de instrucciones por segundo).

En 1960 surge la primera máquina conocida como minicomputadora, por ser más pequeña que las computadoras, se caracteriza por poseer recursos limitados de hardware y por ser realmente más barata.

TERCERA GENERACION. (1964-1970)

La tercera generación de computadoras apareció a mediados de la década de los '60, utilizando por primera vez procesadores fabricados por circuitos integrados. Tecnología LSI (Large Scale Integration, integración a gran escala); un circuito integrado albergaba hasta 20,000 componentes en una superficie de 25 mm^2 y memoria de películas magnéticas. Estos circuitos integrados hicieron por la tercera generación lo que los transistores por la segunda.

GENERACION DE COMPUTADORAS DIGITALES
LENGUAJES DE PROGRAMACION

GENERACION Y APLICACIONES	TECNOLOGIAS Y ARQUITECTURA	UNIDADES PERIFERICAS	LENGUAJES DE PROGRAMACION Y ALFABETO	SISTEMA OPERATIVO Y ADMINISTRACION	ASPECTOS CUANTITATIVOS Y FACILIDADES	MODELOS
PRIMERA 1946 - 1959 INSTRUMENTO DE CALCULO	TUBOS DE VACIO	LECTORAS Y PERFORADORAS DE TARJETAS Y CINTAS DE PAPEL, ETC.	ENSAMBLADORES PRIMITIVOS NUMERICO		MEMORIA CENTRAL 1000 A 8000 PALABRAS PROCESO 10^4 OPS/SEG PRECIO DE 1 A 2.5 MILLONES DE DLS.	IBM - 850 BENDIX 6 - 15 UNIVAC 5590 BULL - PT IBM - 709
SEGUNDA 1959 - 1964 PROCESADOR DE DATOS INSTRUMENTO DE CALCULO	TRANSISTORES FERRITAS	LECTORAS Y PERFORADORAS DE TARJETAS, IMPRESORAS Y CINTAS MAGNETICAS	ENSAMBLADORES PRIMARIOS COMPILADORES (FORTRAN, ALGOL) NUMEROS, LETRAS Y ALGUNOS CARACTERES ESPECIALES	RUDIMENTARIO CONTROL DE PERIFERICOS INICIA Y TERMINA TAREAS PRIMITIVA PLANEACION DE PRODUCCION CON PROCESOS MASIVOS	MEMORIA CENTRAL 8000 A 32000 PALABRAS PROCESO: 10^5 OPS/SEG PRECIO DE 0.1 A 100 MILLONES DLS. EXISTENCIA DE BIBLIOTECAS	CDC-160 IBM 7090 IBM 1401 BORROUGHS 5500 BCA 305 BENDIX 6-20 CDC 3600 CEC 6600
TERCERA 1964 - 1970 SISTEMA DE INFORMACION	CIRCUITOS INTEGRADOS Y MEMORIA DE PELICULA MAGNETICA ARQUITECTURA MULTIPROGRAMACION MULTIPROCESO SISTEMAS DE INTERRUPCION OPTIMIZACION DE CODIGOS	CINTAS Y DISCOS MAGNETICOS TERMINALES DE VIDEO Y TELETIPOS	LENGUAJES DE ALTO NIVEL COBOL PL/1 BASE DE DATOS (DMS) NUMEROS, LETRAS Y CARACTERES ESPECIALES	MANEJO DE DISCOS MULTIPROCESO, MEMORIA DINAMICA Y VIRTUAL, ETC. COMPLEJA Y ESPECIALIZADA	MEMORIA CENTRAL 64 A 256 K PALABRAS PROCESO: 10^6 OPS/SEG M SECUNDARIA 10 ^4 CARACTERES PRECIO DE 5*10^ A 10 8 DLS. EDICION Y PRUEBA INTERACTIVA DE PROGRAMAS	IBM 360 BURROUGS 6700 PDP 10 PDP 11 UNIVAC 1106 CYBER 170
CUARTA 1970-1980 SISTEMAS DE COMUNICACION DE INFORMACION PARA NEGOCIOS Y USO PERSONAL	MICROELECTRONICA MEMORIA MOS (METAL OXIDE SYLICATES) ARQUITECTURA PROCESO DISTRIBUIDO USO DE MICROPROCESADORES	TERMINALES INTELIGENTES DISCOS Y CINTAS MAGNETICAS EQUIPO DE GRAFICACION LECTORES OPTICOS Y DIGITALIZADORES	BASES DE DATOS DISTRIBUIDAS LENGUAJES INTERACIVOS, DESCRIPTIVOS Y GRAFICOS PROCEDURABLES (COGEN-LINK) IRRESTRICTO MAYUSCULAS, MINUSCULAS SIMBOLOS MATEMATICOS ALFABETO ARABE, JAPONES, ETC.	PROCESO SIN INTERRUPCIONES COMUNICACION ENTRE MAQUINAS RUTINAS DE RECUPERACION, ETC. SIMPLE PARA EQUIPOS PERSONALES. COMPLEJA PARA REDES DE PROCESO DISTRIBUIDO	MEMORIA CENTRAL 64 A 10^7 CARACTERES PROCESO 10^7 OPS/SEG MEMORIA SECUNDARIA 10 ^ 10 CARACTERES PRECIO DE 10 ^ 10 ^ 8 DLS. METAPROCESADORES, CORREO ELECTRONICO, MANEJADORES DE TEXTO, ETC.	IBM 4330 UNIVAC 1100 BURROUGS UGHS 6900, 7900 PRIME 650 MP 3100 VAX APPLE TR 80 IBM-PC
QUINTA 1990 -	ESTAS MAQUINAS SE CARACTERIZAN POR LA UTILIZACION DE ENJAMBRES PROCESADORES MICROSCOPICOS OPERANDO SIMULTANEAMENTE PARA RECIBIR Y CLASIFICAR INFORMACION, POR SU CAPACIDAD BASICA DE INFERENCIA Y GENERACION DE CONOCIMIENTOS Y ESQUEMAS GENERALES A PARTIR DE INFORMACION PARTICULAR ASI COMO SU ESTRECHA RELACION CON EL HOMBRE. SU SISTEMA DE CONTROL SIGUE Y UTILIZA LOS PRINCIPIOS DE LAS MAQUINAS LISP Y DEL FLUJO DE DATOS. EL DESARROLLO DE LA COMPUTACION NO-NUMERICA COMO PRINCIPAL APLICACION. COMUNICACION ENTRE MAQUINAS.				

Existe reducción de dimensiones de las instalaciones.

Los problemas de compatibilidad de las computadoras de la segunda generación quedaron eliminados casi por completo en la tercera; pero éstas diferían radicalmente de las de la segunda.

El cambio fue revolucionario, no evolutivo, y provocó pesadillas de conversión a miles de usuarios de computadoras. Con el tiempo, la conversión se consideró como el costo del progreso.

Esta generación presentó nuevas tecnologías en software de sistemas, los sistemas operativos y los sistemas manejadores de bases de datos.

CUARTA GENERACION. (1970-1980)

A principios de la década de los '70, la manufactura de circuitos integrados llega a ser tan avanzada que se logran incorporar miles de componentes activos en volúmenes de una fracción de pulgada, a esto se le llama integración a gran escala (VLSI Very Large Scale Integration) de circuitos, de al menos 100,000 transistores en 25 mm^2 la que es el siguiente eslabón de la cadena de desarrollo: bulbo/transistor/circuito integrado.

Estos nuevos circuitos están más densamente integrados que los de los sistemas anteriores, por lo que han incrementado la velocidad de procesamiento interno de las computadoras.

A partir de este tipo de integración surgen los microprocesadores, los cuales son dispositivos que tienen todas las funciones de la unidad central de proceso (UCP); es decir, están formados de un circuito integrado a gran escala programable, que contiene todos los elementos requeridos para procesar datos codificados en forma binaria; esto es, un microprocesador puede ejecutar operaciones aritméticas y lógicas básicas, tan bien como los ejercicios del mismo tipo que ejecuta la UCP de cualquier computadora convencional.

Un microprocesador complementado con circuitos de suministro de poder, interfases de control de entrada-salida y memoria, consitituyen lo que es una microcomputadora.

Las microcomputadoras se diferencian de las minicomputadoras por tener un tamaño de palabra más pequeño, un conjunto de instrucciones más limitado, tiempo de ciclo de memoria más lento, un menor costo, mínimo consumo de energía, así como controles para aplicaciones específicas.

Esta etapa se caracteriza por la especialización de las aplicaciones de la informática, entre las que destacan las telecomunicaciones, el tratamiento electrónico de la imagen gracias al cual se pueden crear, manipular e interpretar imágenes por medio del ordenador, es el proceso empleado, por ejemplo:

- En la generación de las imágenes enviadas por las sondas espaciales a la Tierra;

- Las bases de datos (colecciones de datos interrelacionados y estructurados que se almacenan independientemente del programa utilizado y que permiten evitar problemas tales como los de la reduplicación de la información contenida en los archivos);

- La inteligencia artificial (rama de la informática que, superando el nivel del cálculo aritmético, se especializa en el tratamiento lógico de la información);

- El desarrollo de sistemas expertos (que se aplican ya a la medicina, la ingeniería, etc.);

- El desarrollo de autómatas o robots capaces incluso de reconocer formas e interactuar con el medio en el que desarrollan su actividad y cuya creciente aplicación en los procesos industriales han generado una nueva rama de la técnica, llamada robótica, y otras más.

QUINTA GENERACION. (1990)

La quinta generación de computadoras deberá estar formalmente establecida durante la década de los '90. Las tecnologías VLSI (Very Large Scale Integration-Muy Alta Escala de Integración), se refieren al gran número de componentes electrónicos (transistores, etc.) interconstruidos en un microcircuito.

La complejidad VLSI varía aproximadamente de 100,000 a 1,000,000 de transistores; y la SLSI (Super Large Scale Integration- Super Gran Escala De Integración) se refiere a las pastillas de densidad ULTRA-ALTA, que contiene más de un millón de transistores, que pondrán al alcance de cualquier persona las grandes computadoras de más reciente desarrollo.

Las fibras ópticas, los videodiscos y otras tecnologías, que por el momento se encuentran en los laboratorios de investigación, se utilizarán en la construcción de los sistemas de cómputo.

Las técnicas de la inteligencia artificial estarán incorporadas en todas las aplicaciones. A fines de siglo, una computadora podrá alternar de manera inteligente con una persona.

El concepto de las máquinas se basa en cuatro elementos fundamentales: el módulo de resolución de problemas; el dispositivo de gestión de las bases de conocimientos (es decir, aquella parte del sistema que alberga los conocimientos de los especialistas en la materia y en la que la información está representada mendiante reglas de producción o redes semánticas); una interfase de lenguaje natural (p. ej., el inglés, y que es el que permitirá la interacción con el usuario); y, finalmente, un módulo de programación.

Algunos fabricantes de computadoras han anunciado una cuarta generación y unos cuantos consideran que sus máquinas son de la quinta generación. Esto es más bien un truco de ventas que la realidad.

Las primeras tres generaciones se distinguieron por importantes cambios tecnológicos en la electrónica: El uso de los tubos de vacío, después los transistores y por último los circuitos integrados.

La cuarta generación está surgiendo con más lentitud en forma de componentes de computadoras y software avanzado; pero es probable que esta generación no resulte oficial hasta el siguiente avance tecnológico.

Esto no quiere decir que han pasado dos décadas sin innovaciones importantes en la tecnología de las computadoras.

Se ha avanzado aún más en la miniaturización de los circuitos en comunicaciones de datos, diseño de hardware y software de computadoras y en dispositivos de entrada y salida.

1.4 TIPOS DE COMPUTADORAS POR SU TAMAÑO

COMPUTADORA EN UNA PASTILLA (4,8,16 BITS)

Son aquellas computadoras que están constituidas en una sola pastilla; un ejemplo, sería un juguete manual educativo que puede utilizar un microprocesador en una sola pastilla.

MICROPROCESADOR (8,16 BITS)

Es un procesador usado en las microcomputadoras, computadoras, y dispositivos periféricos y en casi todos los productos industriales y de gran consumo que emplea una computadora.

MICROCOMPUTADORA (8,16 BITS)

Fueron las primeras computadoras en utilizar como procesador una sola pastilla microprocesadora. Las primeras microcomputadoras, diseñadas y fabricadas principalmente para aficiones o para uso doméstico, recibieron el nombre de computadoras personales. Las computadoras personales y las pequeñas computadoras para negocios, son ejemplo de éstas. Normalmente las microcomputadoras de 8 bits dan servicio a una sola terminal de usuario y tienen un máximo de 64 Kbytes de memoria. Las de 16 bits pueden dar servicio a varias terminales de usuarios y casi siempre tienen un máximo de 1 millón de bytes de memoria.

SUPERMICRO O MICROMINI (16 BITS)

Son los procesadores de minicomputadoras microminiaturizados a base de uno o más microcircuitos.

MINICOMPUTADORA (32 BITS)

Computadora de pequeño a mediano tamaño; las minicomputadoras se encuentran en un punto intermedio entre las microcomputadoras y las mainframes, y ofrecen una amplia variedad de capacidades. Los sistemas a base de minicomputadoras pueden servir desde unas cuantas, hasta varios cientos de terminales de usuario al mismo tiempo.

MAINFRAME (32 BITS)

Al principio todas las computadoras eran mainframes, ya que éste era simplemente otro término para el gabinete que contenía el UCP. Aún se emplea esta palabra para referirse al albergue principal de la UCP; sin embargo, también se conoce el nombre de mainframe como computadora grande.

Existen mainframes de pequeña, mediana y grande escala, que pueden manejar varios miles de terminales en línea. Tienen aproximadamente desde un millón a 64 millones de bytes de memoria principal y tienen potencial de almacenamiento en línea de disco de más de cientos de miles de millones de bytes. Las mainframes de escala media alta emplean computadoras más pequeñas como procesadores frontales, terminales para conectar sus redes de comunicación.

Las mainframes de escala pequeña y las superminis coinciden en cuanto a su capacidad.

Los vendedores de mainframes originalmente fueron: Burroughs, Control Data, General Electric, Honeywell, IBM, NCR y UNIVAC. Todos los vendedores desarrollaron cuando menos una o más series de minicomputadoras y/o microcomputadoras. Para el siglo XXI, la mainframe de la década de los '90 será un modelo de escritorio.

1.5 LENGUAJES DE PROGRAMACION

DEFINICION DE LENGUAJE

Un lenguaje es un conjunto de símbolos y/o gestos junto con un conjunto de reglas (gramática/sintaxis) de acuerdo con las cuales pueden combinarse estos elementos en forma sistemática para comunicar pensamientos o ideas.

Por lo general estos símbolos se combinan con sub-unidades denominadas palabras y después las palabras se combinan en oraciones.

PROGRAMA

Es un conjunto de instrucciones escritas en alguno de los lenguajes de programación, por medio de los cuales el usuario logra que la computadora realice todas las operaciones o decisiones señaladas en dichas instrucciones que una computadora puede reconocer y que han sido ordenadas en una secuencia lógica para ejecutar una tarea particular.

LENGUAJE DE PROGRAMACION

Es el que sirve para escribir instrucciones a la computadora; incluye vocabulario, reglas o conversiones que rigen la forma o secuencia en que se escriben las instrucciones para su ejecución en las computadoras. Los principales tipos de lenguajes utilizados son:

- Lenguaje máquina
- Lenguaje de bajo nivel (ensamblador)
- Lenguaje de alto nivel

Los lenguajes de programación son todos los que permiten comunicar entre usuarios y máquinas, por ejemplo: FORTRAN (Formula Translator), BASIC (Beginner's All-purpose Symbolic Instruction Code), COBOL (Common Business Oriented Language), ALGOL (Algorithmic Language), PL/I (Programming Language I).

LENGUAJE DE MAQUINA o BINARIO

Los lenguajes de máquina son aquellos que están escritos en lenguajes directamente inteligibles por la computadora, ya que sus instrucciones son cadenas binarias (cadenas o series de caracteres -bits- 0 y 1), que especifican una operación y las posiciones (dirección) de memoria implicadas en la ejecución; se denominan instrucciones de máquina o código máquina. El código de máquina es el conocido código binario.

Un lenguaje de máquina no tiene ni gramática ni estructura de oraciones. El código de máquina no requiere traducirse antes de su ejecución. Los lenguajes de máquina utilizan direcciones absolutas y notaciones de máquina para representar códigos de operación.

LENGUAJE ENSAMBLADOR

Este lenguaje usa términos nemotécnicos, que son abreviaturas fáciles de recordar, para representar las instrucciones que se quieren ejecutar, facilitando de esta forma la generación de instrucciones. Cada lenguaje ensamblador está basado en un lenguaje de máquina particular; es decir, cada computadora tiene su propio lenguaje ensamblador y la mayoría de las instrucciones del lenguaje ensamblador tienen una relación uno a uno con las instrucciones correspondientes en el lenguaje de máquina.

Incluso, los formatos de las instrucciones del lenguaje ensamblador son muy parecidos a los de las instrucciones del lenguaje de máquina, de aquí que se considere al primero como una versión simbólica del segundo.

Las instrucciones del lenguaje ensamblador permiten al programador controlar directamente el hardware de la computadora en la misma forma que lo puede hacer con el lenguaje de máquina sin tener que lidiar con las dificultades propias de este último lenguaje como son:

- La codificación directa en lenguaje de máquina dificulta la modificación de un programa.

- Familiaridad con la representación binaria de los códigos de operación.

- Familiaridad con los modos de direccionamiento.

- El tener que considerar direcciones de los operandos en lugar de variables simbólicas.

El programa traductor que convierte el programa fuente en lenguaje máquina es llamado ensamblador; las funciones básicas del ensamblador son:

- Producir un programa objeto en lenguaje de máquina a partir del programa fuente.
- Asignación de memoria principal.
- Generación de enlaces a subrutinas y a rutinas de entrada/salida.
- Manejo de símbolos en una tabla.
- Asociación de direcciones simbólicas con direcciones reales.

Comparado con los lenguajes de alto nivel que se verán a cotinuación el lenguaje ensamblador permite hacer un uso más eficiente de los recursos del computador como son los registros del procesador central, la memoria y los aparatos de e/s.

LENGUAJES DE ALTO NIVEL

Son lenguajes de programación que permiten, con una sola instrucción, hacer una serie de operaciones que en otro lenguaje deben desglozarse y especificarse una tras otra.

Estos lenguajes son más fácilmente entendibles por los usuarios porque es un idioma más parecido al lenguaje natural que ellos usan. Ventajas de los lenguajes de alto nivel:

- Facilidad de aprendizaje.
- Facilitan la depuración de errores.
- Facilidad de mantenimiento y documentación.
- Son transportables.

PROGRAMA FUENTE

Es aquella forma que tiene el programa tal como lo escribe el programador. Por lo general se escribe en un lenguaje de alto nivel y después a través de un programa compilador o intérprete, es traducido a un lenguaje de máquina (objeto) que la computadora pueda entender y comprender.

Los programas fuentes contienen tres elementos esenciales:

- Descripciones de los datos que definen los datos manipulados.
- Descripciones de procedimientos en donde se definen las acciones a realizar.
- Comentarios y notas que permiten a otras personas a seguir la lógica de la codificación

(documentación interna).

PROGRAMA OBJETO

El programa objeto es el que prepara un ensamblador o compilador después de actuar sobre un programa fuente (de alto nivel) escrito por un programador; es decir, es el programa fuente traducido al lenguaje de máquina.

PROGRAMA EJECUTABLE

Es un proceso de computadora que interpreta una instrucción de computadora y realiza las operaciones específicas por la instrucción. Un programa ejecutable se define, cuando se compila un programa fuente; el compilador genera un programa objeto; este programa objeto se liga o enlaza por medio de un editor de enlace (LINK) para que pueda ser corrido en un sistema operativo en particular.

COMPILADORES

Es un programa que sirve para traducir un lenguaje de programación simbólico de alto nivel a un lenguaje de máquina que sea comprensible para la computadora.

Si existen errores en el programa fuente, el compilador ayuda a depurar el problema. El código compilado se cargará y ejecutará más rápido que el código fuente.

INTERPRETES

Es un programa diseñado para ejecutar en la computadora un lenguaje de alto nivel. El intérprete rastrea cada línea del código binario de programa fuente (alto nivel) y lo transforma (interpreta) en código de máquina cada vez que el programa se ejecuta. Traduce cada instrucción del programa y de inmediato la ejecuta. Los programas complicados que utilizan intérpretes tienden a correr con mucha lentitud.

LENGUAJES COMPILADOS E INTERPRETADOS

De acuerdo con la forma como el programa fuente o programa en lenguaje de alto nivel es traducido a lenguaje de máquina, los lenguajes se clasifican en compilados e interpretados.

En el proceso de compilación el programa fuente es traducido a un programa objeto con código en lenguaje máquina. El programa que hace la conversión es llamado un compilador. El programa objeto es posteriormente enlazado con las bibliotecas necesarias, cargado en memoria y ejecutados.

En el programa de interpretación el programa fuente es primero traducido a un programa objeto cuyo código no es el de máquina sino un código llamado intermedio, usando

subrutinas. El programa intérprete es más pequeño que el programa compilador; sin embargo, el tiempo de ejecución de un programa interpretado es mas grande que el tiempo de ejecución de un programa objeto compilado.

RUTINA

Una rutina es un conjunto de instrucciones de la computadora digital en secuencia diseñada para realizar una función específica: resolver un problema, recuperar datos específicos o controlar un sistema. Una rutina puede contener dos o más instrucciones, pero la mayoría abarca de 40 a 300. En cuanto a su función, las rutinas pueden ser especificadas, generales o muy generales. Una rutina puede ser independiente, pero lo más común es que se utilice en interacción con otras.

SUBRUTINA

Es un grupo de instrucciones de computadora diseñado para ser usados por otras rutinas, con el fin de lograr un objetivo específico. Por lo general, una subrutina se emplea más de una vez.

1.6 CLASIFICACION DE LOS LENGUAJES DE ACUERDO CON EL AREA DE APLICACION.

De acuerdo con el área de aplicación, los lenguajes de alto nivel son clasificados en:

COMERCIALES

Son lenguajes usados en programas que hacen un uso extenso de operaciones de manipulación de archivos de datos y de generación de reportes. Los programas en estos lenguajes se caracterizan por hacer poco uso del procesador y un gran uso de operaciones de e/s, para leer y escribir archivos. El ejemplo típico es COBOL (Common Business Oriented Language).

CIENTIFICOS

Estos lenguajes son usados en programas que manipulan datos numéricos, en estos casos se hace un gran uso del procesador y poco empleo de los aparatos de e/s. El ejemplo clásico de estos lenguajes es Fortran (Formula Translator).

INTERACTIVOS

Son lenguajes que permiten al usuario interactuar con la computadora. Para ello el usuario o

programador utiliza una terminal de video, a través de ella hace cambios y correcciones durante la ejecución del programa. APL es el lenguaje interactivo por excelencia.

DE BLOQUES

Son lenguajes en los que cada programa es organizado como un conjunto anidado de bloques, lo cual permite la programación estructurada que se caracteriza por el uso de sólo tres estructuras de control: la secuencia, la selección y la iteración. Los lenguajes PASCAL, ALGOL (Algorithmic Language) caen dentro de este tipo.

DE SIMULACION

Son lenguajes diseñados especialmente para estudiar modelos de sistemas reales. Entre este tipo de lenguajes están el GASP, el GPSS y el SIMSCRIPT.

OTRAS APLICACIONES

EN LA CIENCIA Y LA INVESTIGACION

Los ingenieros y los científicos utilizan sistemáticamente la computadora como una herramienta en la experimentación y diseño. Estos usos podrían incluir a: un ingeniero electromecánico que utiliza y analiza la aerodinámica de un automóvil; un ingeniero industrial que hace un análisis estadístico con una muestra de trabajo; un físico que genera números aleatorios para entrada experimental; un diseñador que analiza gráficas de computadoras para dibujar una vista tridimensional de una parte mecánica.

EN EL CONTROL DE PROCESOS

Las computadoras utilizadas en el control de proceso captan datos en un ciclo continuo de retroalimentación; en este ciclo el proceso genera datos que se vuelven entradas para las máquinas. Conforme la computadora recibe datos y los interpreta, inicia acciones para controlar el proceso en marcha.

Las computadoras de control de procesos vigilan y controlan el equipo de calefacción, ventilación y acondicionamiento de aire en edificios, de manera que el consumo de energía se minimice y la comodidad personal se optimice.

EN LA EDUCACION

Las computadoras pueden interactuar con los estudiantes y mejorar el proceso de aprendizaje. Hardware relativamente poco costoso, capaz de comunicación multidimensional (sonido, impresión, gráficas y color) ha favorecido un crecimiento muy desmedido de la computadora como herramienta educativa tanto en el hogar como en el salón de clases.

EN HOSPITALES

La computadora ha penetrado también en los hospitales. Con ella puede controlarse eficazmente las constantes vitales de los enfermos y, al mismo tiempo, centralizar los cuidados de toda la institución. Pero quizá su aplicación más prometedora en medicina sea el diagnóstico y la terapéutica correspondiente, ante los que el médico decide.

Las bibliotecas automatizadas son también de gran valor en medicina. Asimismo, el registro mecanizado de las enfermedades de una población puede permitir detectar sus causas, localizar epidemias y, en general, potenciar el desarrollo de la medicina preventiva, a la que día con día se da mayor importancia.

A LA COMPUTACION PERSONAL

Las personas están adquiriendo pequeñas computadoras personales, poco costosas como entretenimiento y para variadas aplicaciones domésticas. El crecimiento de esa área ha sobrepasado aún las predicciones de la mitad de la década de los '70.

Las aplicaciones representativas de la computación personal incluyen el manejo de cuentas de cheques, agenda telefónica, juegos de video, instrucción auxiliada por computadora y procesamiento de palabras.

A LOS SISTEMAS DE INFORMACION/PROCESAMIENTO DE DATOS

La mayor parte de la fuerza actual de computación está dedicada a los sistemas de información y al procesamiento de datos. En esta categoría se incluyen todos los usos de la computadora en los negocios.

Son ejemplos de estas aplicaciones los sistemas de nómina, los sistemas de reservación de líneas aéreas, los sistemas de información para estudiantes, los sistemas de facturación a pacientes de hospitales, los sistemas de mantenimiento de vehículos, los sistemas de automatización de oficinas, etc.

1.7 CARACTERISTICAS DE LOS LENGUAJES DE PROGRAMACION

Es indudable que cada lenguaje de programación tiene sus características que lo distinguen de otros; sin embargo, aún con propiedades diferentes, los lenguajes de programación son diseñados con base en un conjunto de conceptos fundamentales que sirven de base para estudiarlos. Enseguida se analiza cómo son soportados a nivel de Hardware. Los datos, las operaciones, el control de la secuencia, el control de los datos, la memoria y el medio ambiente.

DATOS

Los tipos de datos pueden ser manejados directamente por operaciones primitivas de hardware de la computadora son los números enteros (o números de punto fijo), los números reales (o números de punto flotante), las cadenas de bits y de caracteres de longitud fija. Otro tipo de datos manejados por el hardware son los programas representados en lenguaje de máquina y cuya estructura es la de una secuencia de localizaciones de memoria.

Cada localización contiene una o más instrucciones, las que a su vez están compuestas de un código de operación y de indicadores de operandos.

OPERACIONES

Las operaciones que aparecen en las instrucciones de lenguaje de máquina de una computadora son las primitivas, que pueden ser ejecutadas por el hardware de la misma. Entre las más importantes se encuentran las siguientes:

- Operaciones primitivas aritméticas.
- Suma entera o real.
- Multiplicación.
- División.
- Resta.
- Operaciones para checar atributos de los datos.
- Chequeo de números positivos.
- Chequeo de números negativos.
- Chequeo de ceros.
- Operaciones para controlar los aparatos de entrada/salida.

CONTROL DE LA SECUENCIA

Este concepto se refiere al control de la secuencia en la cual las instrucciones del programa deben ser ejecutadas. La secuencia natural de ejecución es el orden en el que las instrucciones están dentro del programa en lenguaje máquina.

Como se sabe la unidad de procesamiento central contiene un registro contador de programa (PC) el cual posee la dirección de la siguiente instrucción a ser ejecutada. En el proceso de obtención de la instrucción (FETCH) el registro contador de programa es incrementado para apuntar a la instrucción sucesiva.

Si una instrucción en lenguaje de máquina altera el orden natural de ejecución enviando al intérprete a ejecutar una instrucción cuya dirección es x, el registro contador de programa es cargado con el valor x lo que indicará al intérprete que ese valor es la dirección de la próxima dirección.

TIPOS DE MEMORIA

MEMORIA

Es un almacenamiento de trabajo de la computadora. Considerada como el principal trabajo de una computadora, ya que todo procesamiento tiene lugar en la memoria.

La memoria es el recurso más importante de la computadora; determina tanto la complejidad, como el número de programas diferentes que pueden ejecutarse al mismo tiempo.

MEMORIA RAM

La memoria RAM (Random Access memory - memoria de acceso aleatoria) se utiliza en la memoria interna de las microcomputadoras para almacenar en forma temporal programas de aplicaciones, datos para cálculos posteriores y resultados de procesamiento de computadoras. La memoria de acceso aleatorio se borra o vuelve a posicionar cuando falla la energía (es volátil), cuando la computadora se apaga y a menudo en la carga.

MEMORIA ROM

La memoria ROM (Read Only Memory - memoria de sólo lectura), es una pastilla de memoria permanente para almacenamiento de programas. Las instrucciones y la información, o ambas, se almacenan en la ROM en el momento de su fabricación y no pueden alterarse en el uso normal de las computadoras ni por pérdida de energía (no es vólatil). La memoria de sólo lectura ROM se utiliza en los módulos que se puedan conectar en las calculadoras, computadoras para juegos y ciertas computadoras personales.

MEMORIA EPROM

La memoria EPROM (Erasable and Programable Read Only Memory -memoria de lectura exclusiva reprogramable), es un arreglo de memoria de un circuito integrado, que se fabrica con un patrón inicialmente de ceros y unos lógicos. Tiene escrito una instrucción específica y un patrón de información, lo cual hace el fabricante utilizando un programador especial de hardware. En este tipo de memoria es posible almacenar instrucciones e información, o ambas, provenientes de una computadora. También son utilizados para almacenar software permanente.

MANEJO DE MEMORIA

Hay tres grandes componentes de almacenamiento de datos en una computadora, éstos son:

- Memoria principal
- Registro de propósito general del procesador central
- Memoria secundaria o memoria de discos o cintas

Sin embargo, a excepción del hardware para la implementación del concepto de memoria virtual y reubicación no hay mecanismo de manejo de memoria soportados por la circuitería de la computadora.

MEDIO AMBIENTE OPERATIVO

El medio ambiente operativo de la computadora consiste de aparatos periféricos y otros tipos de aparatos de e/s que relacionan a la computadora con el medio ambiente del usuario.

1.8 IMPACTO DE LAS COMPUTADORAS EN LA SOCIEDAD Y TENDENCIAS DEL DESARROLLO DE LOS SISTEMAS DE COMPUTO A CORTO PLAZO. LA COMPUTACION EN NUESTRO PAIS: SITUACION ACTUAL Y PERSPECTIVAS A MEDIANO PLAZO

Durante 1990 el mercado mexicano de informática experimentó cambios radicales que marcaron el inicio de una década totalmente diferente a la pasada. La apertura de fronteras fue uno de los principales catalizadores de dicho cambio, ya que ayudó a intensificar aún más la competencia, favoreciendo al usuario; si la década de los ochenta se distinguió por mantener un mercado de vendedores, la de los noventa logrará convertirlo en uno de compradores.

La apertura fue acompañada por otros cambios importantes, tales como la simplificación de las compras del sector público, mercado que registró la tasa de crecimiento más alta de los últimos años.

La apertura dio origen a una fase de incertidumbre y alta turbulencia. En primer lugar, poco antes y poco después de la apertura, la incertidumbre hizo que el usuario fuese más cauteloso en sus compras, ocasionando un bache en el crecimiento de la demanda durante el segundo trimestre, más pronunciado que de costumbre.

En segundo lugar, la apertura favoreció el establecimiento de nuevos oferentes; la gran mayoría introdujeron marcas de prestigio internacional que estaban parcial o totalmente ausentes en el mercado mexicano, tales como ACER, ALR, APPLE, ATT, COMPAQ, COMPUADD, DELL y HYUNDAI.

La presencia de nuevos jugadores hizo que la oferta aumentara en forma significativa, particularmente en la parte alta del mercado, donde la oferta era limitada y los diferenciales de precios altos. El número de modelos de máquinas de alto rendimiento como las 386sx, 386 y 486, aumento rápidamente.

Junto con la incertidumbre y la turbulencia, durante 1990 continuaron apareciendo señales inequívocas de un mercado que avanza hacia la madurez. Los nuevos usuarios siguen y seguirán siendo una fuerza muy importante de crecimiento y desarrollo, ya que todavía existe una enorme masa de individuos y empresas que no han tenido contacto con la tecnología

informática.

Sin embargo, los usuarios experimentados cada día son más, y están contribuyendo a que aumente la demanda de más y mejores productos y servicios de tecnología avanzada. El alto crecimiento que registraron las 286's y las 386sx acompañadas de interfaz gráfica del año pasado, es sólo una muestra de esta tendencia hacia la sofisticación del mercado.

Para 1994 esperamos que la turbulencia e incertidumbre disminuyan y que las tendencias a la maduración del mercado se consoliden. Continuarán llegando nuevos jugadores y veremos las primeras marcas en aprietos; sin embargo, los cambios en la oferta serán mínimos en comparación con lo que sucedió en 1990. El evento más espectacular será, por otro lado, el hecho de que el parque instalado rebasará el millón de "PC" durante 1992, lo que simbólicamente representa una parte en la historia de nuestro mercado.

Esta maduración traerá aparejada la exigencia de un valor agregado mayor por parte del usuario. Las marcas lograrán ser competitivas en la medida en que se complementen los precios agresivos con la oferta de máquinas de calidad en configuraciones completas que generalmente incluirán software, en particular de interfaz gráfica. Uno de los segmentos más beneficiados con este proceso de maduración será el de las redes y el software para redes, así como los nuevos servicios que empiezan a vender a través de ellas.

La intensificación de la competencia y el crecimiento esperado de la economía mexicana para los próximos años prometen mantener el mercado mexicano de informática en una situación próspera y de mucho vigor. Como veremos a continuación, tal vez las tasas de crecimiento no sean tan espectaculares como al final de los ochenta, pero se matendrán altas en comparación con el resto de la economía.

El mercado de equipo de informática creció a una tasa compuesta anual de 10% durante la década de los ochenta; entre 1986 y 1989, la tasa se aceleró, alcanzando el 16% anual. Las microcomputadoras mostraron mayor dinamismo ya que durante esta década crecieron a una tasa impresionante de 38% anual en comparación de la tasa negativa de 1.5% anual que registraron en conjunto las minis y las macros.

Sin duda, la tecnología micro es más apropiada para el mercado mexicano y su alto crecimiento hizo que para 1989 las ventas de micros representaran el 66% del mercado nacional de equipo informático.

La juventud del mercado de las micros en México, se muestra por las drásticas fluctuaciones en niveles muy altos que se registraron en los ochenta, en comparación con el crecimiento de las minis y las macros que siguieron más de cerca a la economía mexicana.

En los próximos años, sin embargo, debemos esperar para las micros tasas altas y más estables, las cuales disminuirán en forma paulatina; esto es señal de que el mercado está alcanzando mayores niveles de madurez y de que aún está muy lejos de la saturación.

En general, el mercado de equipo creció a una tasa compuesta de 12% entre 1990 y 1993, llegando a ser de $1,106 millones de dólares a finales de este periodo. Las micros aumentarán a una tasa de 15% y las minis-macros de 5%.

El mercado de software en México creció a tasas aún más aceleradas que el mercado de

equipo. Entre 1984 y 1989 registró una tasa compuesta de 24% anual. Lógicamente, el software para minis y macros, 33% contra 19%. Asimismo, este mercado para micros muestra las mismas tendencias hacia la maduración que el de las micros con tasas altas que disminuyen en forma paulatina.

El mercado de software aumentó a una tasa de 11% entre 1990 y 1993, llegando a ser de $279 millones de dólares al término de 1993; el software para micros crecerá a una tasa compuesta de 15% y el software para minis y macros de 8%.

El mercado de servicios tales como mantenimiento y capacitación también se incrementó mucho durante los ochenta, registrando una tasa compuesta de 21% anual. Se esperó que entre 1990 y 1993 los servicios crecieran a una tasa de 18% anual para llegar a ser de $485 millones de dólares.

Es muy probable que el crecimiento del software y los servicios durante la década de los ochenta este siendo sobrestimado, particularmente en el segmento de las micros. Opinamos que nuestro mercado muestra un rezago considerable que limita el crecimiento y la efectividad de los usuarios.

Si aceptamos que la sofisticación del mercado es la tendencia clave de los noventa, es de suma importancia que desarrollemos el software y los servicios para satisfacer las necesidades de los consumidores.

Con un parque instalado de más de 1 millón de PC para finales de 1991, por ejemplo, las oportunidades para los oferentes de software y servicios aumentaron exponencialmente. El reto es contribuir a que los usuarios de la informática compitan en el mercado internacional, al que cada día están mas expuestos por la inserción de nuestra economía en la economía mundial. Esto dependerá de que se ofrezcan los bienes y servicios informáticos con precios, calidad y rendimiento similares a los que consumen nuestros competidores.

UN FUTURO DOMINADO POR ESTANDARES. Los estandares exclusivos han desaparecido, siendo reemplazados por soluciones más populares y uniformes de múltiples proveedores. No obstante, la necesidad de compatibilidad con la base instalada retarda los avances tecnológicos. Los precios del mercado abierto son más bajos, con fuerte competencia en software de aplicación. Las ventajas mayores las tienen firmas pequeñas y sensibles a las necesidades de los usuarios o grandes empresas que producen en volumen. El lejano oriente gana fuerza en todo el mundo a costa de IBM.

UN FUTURO IMPULSADO POR NUEVA TECNOLOGIA. Los avances continúan hasta el séptimo año, pantallas planas, reconocimiento del habla, procesamiento de imágenes, extraordinario color, asistentes expertos y documentos multimedia le dan una nueva definición a la interacción con el usuario. Las fibras ópticas aceleran la transmisión de imágenes, mientras el procesamiento paralelo y nuevas arquitecturas incrementan tremendamente el rendimiento. Compañías nuevas y agresivas impulsan la innovación; Los grandes proveedores se limitan a soportar la base instalada existente.

UN FUTURO QUE PERTENECE A LAS REDES. El uso de computadoras separadas e

independientes va disminuyendo, a medida que máquinas conectadas a redes se convierten en procesadoras paralelas distribuidas. Acceso a datos independientemente de su ubicación y el procesamiento cooperativo son los principales paradigmas de software. El software para grupos (groupware) es popular. Hasta en empresas de computación pequeñas que participan en el mercado mundial de sistemas de información.

UN FUTURO ECONOMICAMENTE AL ALCANCE. Disminuyen los gastos en tecnología, pues los usuarios piensan muy bien antes de realizar inversiones; dominan la lógica económica de adoptar plataformas distribuidas de bajo costo. Las nuevas aplicaciones más populares son las destinadas al usuario final. Las economías de escala y buena distribución son elementos claves para el éxito, el estancamiento mundial propicia el orgullo nacionalista.

UN FUTURO DOMINADO POR GRANDES EMPRESAS. Intensa consolidación reduce el número de proveedores de extensas líneas de productos de información. La necesidad de obtener altos beneficios de las inversiones de investigación y desarrollo propicia los enfoques exclusivos, dando lugar a mejor tecnología, integración y funcionalidad. Los usuarios prefieren las soluciones empaquetadas y los sistemas críticos para la misión desarrollados centralmente. Los fabricantes dominan a sus proveedores y a sus clientes y elevan el costo de entrada al mercado.

CAPITULO II

ARQUITECTURA Y FUNCIONAMIENTO DE LAS COMPUTADORAS DIGITALES

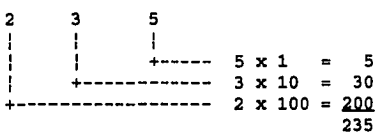
2.1 TRANSFORMACION DE NUMEROS ENTRE LOS SISTEMAS DECIMAL, BINARIO, OCTAL Y HEXADECIMAL

CONCEPTO DE SISTEMA DE NUMERACION

Un sistema numérico consiste de un conjunto ordenado de símbolos llamados dígitos con relaciones definidas para adición (+), substracción (-), multiplicación (x) y división (/). La base de este sistema es el número total de dígitos permitidos en el sistema numérico.

Cualquier número en un sistema numérico, tiene una parte entera y una parte fraccionaria, separada por un punto, existen algunos casos en los cuales una de estas dos partes no existe.

Nuestro sistema numérico decimal consta de 10 dígitos: 0,1,2,3,4,5,6,7,8,9. Utilizando estos 10 símbolos podemos escribir números tan grandes como queramos, debido a la notación posicional, mediante la cual cada dígito tiene un valor relativo y un peso según su posición. De esta forma, cuando ponemos tres dígitos seguidos, como 235, queremos indicar el número doscientos treinta y cinco:



De no considerar el valor relativo de las cifras, a partir del nueve necesitaríamos usar un número infinito de símbolos, lo cual es imposible.

La notación posicional permite además realizar fácilmente las operaciones elementales de suma, resta, multiplicación o división. En cualquier otro sistema no posicional, será difícil realizar esas operaciones; por ejemplo, piénsese en los números romanos, que sí permiten representar números grandes, pero con los cuales no se pueden hacer operaciones.

Sistema de numeración es el conjunto de reglas que permiten nombrar y escribir cualquier número, a partir de un número finito de símbolos y consta de una base.

BASE de un sistema de numeración es el número por el que hay que multiplicar a una unidad inferior para obtener la inmediata superior. Es el cardinal del conjunto de símbolos. Es decir, siendo "b" la base, la unidad inferior siempre tiene de peso $b^0 = 1$, la inmediata superior tendrá un peso de $1 \times b = b$, la inmediata superior será el producto de la anterior por b: $b \times b = b^2$, etc.

$$b^3 \quad b^2 \quad b^1 \quad b^0 = 1$$

En un sistema de numeración de base "b" existen b símbolos diferentes, que van desde 0 hasta b-1:

$$0,\ 1,\ 2,\ 3,\ \ldots\ ,\ b-1$$

Nuestro sistema decimal usa los símbolos:

$$0,1,2,3,4,5,6,7,8,9.$$

Cuando la base es mayor que 10, se usarán además otros símbolos que son:

$$0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,\ldots$$

donde la A simboliza el 10, la B el 11, la C el 12, etc. La base en la que está escrito un número la indicaremos como subíndice de ese número. Por ejemplo: 123_6.

TEOREMA FUNDAMENTAL DE LA NUMERACION

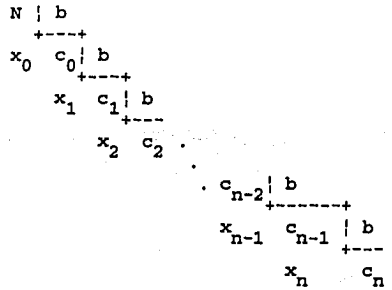
El teorema fundamental de la numeración dice así:

Dado un sistema de numeración de base b, con base > 1 cualquier número natural N puede descomponerse de la forma:

$$N = x_{n-1} \cdot b^{n-1} + \ldots + x_2 \cdot b^2 + x_1 \cdot b^1 + x_0$$

siendo n el número de cifras del número N en ese sistema, donde los coeficientes x_i son menores que la base b.

Demostración: Para ello dividiremos N y los sucesivos cocientes obtenidos entre b.



por lo que:

$$N = c_0 b + x_0 \quad (1)$$

$$c_0 = c_1 b + x_1 \quad (2)$$

$$c_1 = c_2 b + x_2 \quad (3)$$

$$\ldots\ldots\ldots\ldots$$

$$c_{n-2} = c_{n-1} b + x_{n-1} \quad (4)$$

$$c_{n-1} = c_n b + x_n \quad (5)$$

donde cada cociente es menor que el dividendo:

$$N > c_0 > c_1 > c_2 > \ldots > c_{n-2} > c_{n-1} > c_n$$

y el último cociente será $c_n < b$ que no permite más divisiones.

Si sustituimos el valor de las expresiones (2) y (1) tenemos que:

$$N = (c_1 b + x_1) b + x_0 = c_1 b^2 + x_1 b + x_0$$

sustituyendo ahora la expresión (3) resulta:

$$N = (c_2 b + x_2) b^2 + x_1 b + x_0 = c_2 b^3 + x_2 b^2 + x_1 b + x_0$$

si repetimos el proceso con las demás ecuaciones, llegaremos al resultado buscado.

$$N = x_{n-1} \cdot b^{n-1} + \ldots + x_2 \cdot b^2 + x_1 \cdot b + x_0$$

CONVERSION DE UN SISTEMA DE NUMERACION A OTRO CON NUMEROS ENTEROS

Se aplica el teorema fundamental de la numeración.

EJEMPLO: Escribir en la base decimal el número 4123_5

$$4123_5 = 4 \cdot 5^3 + 1 \cdot 5^2 + 2 \cdot 5^1 + 3 = 4 \cdot 125 + 1 \cdot 25 + 2 \cdot 5 + 3 = 538_{10}$$

```
          ¦
          +--- n-1= 4-1=3, el núm de n=4 cifras
```

También es posible obtenerlo aplicando el **algoritmo de Ruffini**, que no es más que otra forma de hacer las operaciones del teorema fundamental de la numeración, o sea:

1. Copiar arriba el número y debajo, a la izquierda, la base.

```
   ¦ 4   1   2   3
 5 ¦
---+--------------
   ¦
```

2. El primer número de arriba de la izquierda, copiarlo debajo.

```
   ¦ 4   1   2   3
 5 ¦
---+--------------
   ¦ 4
```

3. Multiplicar este número por la base, colocar el resultado en la siguiente columna y sumar los elementos de la columna.

```
   ¦ 4   1   2   3
 5 ¦     20
---+---------------
   ¦ 4   21
```

4. Repetir el paso 3 con la siguiente cifra, y así sucesivamente hasta el final. El número obtenido en el extremo inferior derecho es el resultado.

```
   ¦ 4   1   2   3
 5 ¦     20 105 535
---+---------------
   ¦ 4   21 107 538
```

CONVERSION DE UN NUMERO EN BASE 10 A LA BASE b

Ahora el procedimiento es el inverso: hacer las divisiones sucesivas del número entre la base hasta que el cociente sea menor que la base. Los números obtenidos como éstos componen el número en base b, pero colocados en orden inverso al que se han ido obteniendo.

EJEMPLO: Convertir a base 5 el número 538_{10}

```
538 ¦ 5
    +---+
038  107¦ 5
       +---+
      07  21¦ 5
          +---
  3    2   1  4
```

luego: $538_{10} = 4123_5$

CONVERSION DE UN NUMERO EN UNA BASE CUALQUIERA b A OTRA BASE CUALQUIERA b'

Se funadamenta en utilizar como base auxiliar intermedia la base decimal; es decir, pasaremos el número en base b a base 10, y de ésta, a la base b'.

EJEMPLO: Convertir a base 12 el número 4123_5

$$4123_5 = 4 \cdot 5^3 + 1 \cdot 5^2 + 2 \cdot 5^1 + 3$$
$$= 4 \cdot 125 + 1 \cdot 25 + 2 \cdot 5 + 3 = 538_{10}$$

```
538 | 12
    +----+
058   44 | 12
         +---
 10    8    3
 (A)
```

luego: $4123_5 = 538_{10} = 38A_{12}$

Nótese que:

- Al convertir un número a una base mayor, el número de cifras disminuye, puesto que cada una participa con un mayor peso.

- Al convertir un número a una base menor, el número de cifras aumenta, puesto que cada una participa con un menor peso.

CONVERSION DE UN NUMERO EN BASE b A LA BASE 10 CON NUMEROS FRACCIONARIOS

Simplemente hay que aplicar el teorema fundamental de la numeración, que se amplía de la siguiente forma:

Llamando n al número de cifras enteras y n' al número de cifras fraccionarias, cualquier número N puede descomponerse en la forma:

$$N = \underbrace{x_{n-1} \cdot b^{n-1} + \ldots + x_2 \cdot b^2 + x_1 \cdot b^1 + x_0}_{\text{Parte entera}} + \underbrace{d_1 \cdot b^{-1} + d_2 \cdot b^{-2} + \ldots + d_{n'} \cdot b^{-n'}}_{\text{Parte fraccionaria}}$$

EJEMPLO: Convertir a la base decimal el número 101.1101_2

$$101.1101_2 = 1\cdot2^2+0\cdot2+1+1\cdot2^{-1}+1\cdot2^{-2}+0\cdot2^{-3}+1\cdot2^{-4} =$$

$$4 + 0 + 1 + 0.5 + 0.25 + 0.0624 = 5.8125_{10}$$

CONVERSION DE UN NUMERO EN BASE 10 A LA BASE b

Se realizará en dos partes:

a) La parte entera se convierte como se ha explicado anteriormente.

b) La parte fraccionaria se convierte de la siguiente manera:

Se multiplica la parte fraccionaria por la base a la que se pretende pasar, y la parte entera del resultado es el primer dígito fraccionario obtenido. Se toma la parte fraccionaria de dicho resultado y se le vuelve a multiplicar por la base.

Se repite el método sucesivas veces hasta obtener una parte fraccionaria nula (en cuyo caso, el cambio de base es exacto), o hasta que se tenga como aproximación una cantidad suficiente de dígitos.

EJEMPLO: Convertir a la base 2 el número 5.8125_{10}

Parte entera: 5

```
5 | 2
  +---+
1   2 | 2
      +---
    0   1
```

$5_{10} = 10_2$

Parte fraccionaria:

```
0.8125 x 2 = 1.6250      1 (primer dígito fraccionario)
0.625  x 2 = 1.25           izquierdo)
0.25   x 2 = 0.5
0.5    x 2 = 1.0
```

$0.815_{10} = 0.1101_2$

Por tanto, $5.8125_{10} = 101.1101_2$

CONVERSION ENTRE LAS BASES BINARIA, OCTAL Y HEXADECIMAL

El sistema binario (b = 2) dispone de dos dígitos que son 0 y 1. Estos dígitos se llaman abreviadamente bits. Es muy importante este sistema de numeración en el campo de la computación, porque en él se basa toda la lógica interna de una computadora.

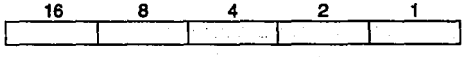
Para hacer conversiones de números binarios a números escritos en las bases octal o hexadecimal, y a la inversa, se pueden aplicar los procedimientos antes estudiados, pero para estos dos casos concretos hay otros métodos más rápidos y sencillos, como se muestra a continuación.

Estos procedimientos servirán para la conversión entre una base binaria y otra que sea potencia de 2 ($b = 2_n$), y viceversa.

CONVERSION RAPIDA DE BINARIO A DECIMAL

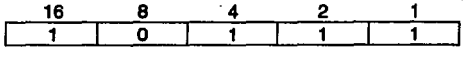
Cuando el número binario es de pocas cifras puede hacerse mentalmente con sólo tener en cuenta los pesos de cada dígito.

Peso de cada dígito



EJEMPLO: Que número es el 10111_2

Peso de cada dígito



$$10111_2 = 16 + 0 + 4 + 2 + 1 = 23_{10}$$

CONVERSION RAPIDA DE DECIMAL A BINARIO

Cuando el número decimal es pequeño puede hacerse mentalmente con sólo tener en cuenta los pesos de cada dígito. Y lo único que debemos pensar es cómo descomponer el número dado sumando ciertos pesos de los dígitos binarios.

EJEMPLO:

Pasar a base 2 el número 13_{10}

Como 13 = 8 + 4 + 1, pondremos 1 en esas casillas y cero en las otras.

Peso de cada dígito

8	4	2	1
1	1	0	1

Entoces, $13_{10} = 110_{12}$

CONVERSION DE BINARIO A OCTAL (b = 8)

Tomar el número binario y hacer grupos de tres cifras, de derecha a izquierda, conviertiendo cada grupo de éstos a su equivalente octal.

Tomando grupos de tres cifras por ser $8 = 2^3$

EJEMPLO: Convertir a octal el número 11010111_2

011 010 111

En octal: 3 2 7

Luego $11010111_2 = 327_8$

CONVERSION DE OCTAL A BINARIO

Es el proceso inverso al anterior. Consiste en convertir cada cifra del número en las tres cifras binarias correspondientes. Proceso que conviene hacer mentalmente, pues haciendo divisiones sucesivas sería más largo.

EJEMPLO: Convertir a binario el número 4307_8

	4 3 0 7
En binario	100 011 000 111
Luego	$43078 = 100011000111_2$

CONVERSION DE BINARIO A HEXADECIMAL (b = 16)

Tomar el número binario y formar grupos de cuatro cifras, de derecha a izquierda, para luego convertir cada grupo de éstos a su equivalente hexadecimal. Se forman grupos de cuatro cifras por ser $16 = 2^4$.

EJEMPLO Convertir a hexadecimal el número 11010111_2

	1101 0111
En decimal:	13 7
En hexadecimal:	D 7

Luego $11010111_2 = D7_{16}$

CONVERSION DE HEXADECIMAL A BINARIO

Es el proceso inverso al anterior. Consiste en convertir cada cifra del número escrito en hexadecimal en las cuatro cifras binarias correspondientes.

EJEMPLO Convertir a binario el número $43F9_{16}$

	4	3	F	9
En binario:	0100	0011	1111	1001

Luego $43F9_{16} = 100001111111001_2$

CONVERSION DE HEXADECIMAL A OCTAL Y VICEVERSA

Se utiliza como base intermedia la base dos y después se transforma a la base deseada.

EJEMPLO

Hexadecimal	8	A	
Binaria	1000	1010	
Luego:	10001010=010	001	010
Octal	2	1	2

$(8A)_{16} = (212)_8$

EJEMPLO

Octal	2	1	4
Binaria	010	001	100
Luego:	0100011100 =	1000	1100

$(214)_8 = (8D)_{16}$

TABLA GENERAL DE EQUIVALENCIAS

BINARIO	OCTAL	DECIMAL	HEXADECIMAL
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10
10001	21	17	11
10010	22	18	12
10011	23	19	13
10100	24	20	14
.	.	.	.
.	.	.	.
.	.	.	.

2.2 OPERACIONES EN LOS SISTEMAS DE NUMERACION

SUMA

En primer lugar, analicemos cómo se realiza la operación suma en la base decimal para que este criterio nos sirva como modelo de referencia.

```
arrastres        1 1                2 + 7 = 9
                  7 8 4 2           4 + 3 = 7
                +                   8 + 4 = 2 y llevo 1
                  7 4 3 7     7 + 7 + 1 = 5 y llevo 1
                 1 5 2 7 9            1 = 1
```

Como se observa, para sumar dos números, si la suma excede al valor de la base existe un acarreo o arrastre de una unidad en la columna izquierda siguiente.

En cualquier otra base se hará de forma análoga, esto es, se suman de derecha a izquierda los números de cada una de las columnas, incluyendo el posible arrastre, como si se tratase de la base decimal y, seguidamente, el resultado obtenido N se convierte a la base deseada.

Aplicando al caso de dos sumandos resulta:

- Si N < b, el resultado final es el mismo en la base b, es decir, N.
- Si N => b, se descompone N en dos sumandos, de manera que:

$$N_{10} = (1 \cdot b + x)_{10} = 1xb$$

(se escribe x, y hay un arrastre 1 en la columna izquierda

EJEMPLO: En base 8:

```
arrastres    1 1
             1 7 4 2(8)          2 + 3 =  5(10)
          +                      4 + 6 = 10(10)
             5 0 6 3(8)    1 + 7 + 0 =  8(10)
             ---------
             7 0 2 5(8)    1 + 1 + 5 =  7(10)
```

```
 5(10)  =          =  5(8)
10(10)  = 8 + 2(10) = 12(8)  (2 y llevo 1)
 8(10)  = 8 + 0(10) = 10(8)  (0 y llevo 1)
 7(10)  =          =  7(8)
```

EJEMPLO: En base 2

arrastres 1 1 1 1

$$\begin{array}{rr} & 1\ 0\ 1\ 1_{2} \\ + & 1\ 1\ 0\ 1_{2} \\ \hline & 1\ 1\ 0\ 0\ 0_{2} \end{array}$$

$1 + 1 = 2_{10}$
$1 + 1 + 0 = 2_{10}$
$1 + 0 + 1 = 2_{10}$
$1 + 1 + 1 = 3_{10}$
$1 = 1_{10}$

$2_{10} = 2 + 0_{10} = 10_{2}$
$2_{10} = 2 + 0_{10} = 10_{2}$
$2_{10} = 2 + 0_{10} = 10_{2}$
$3_{10} = 2 + 1_{10} = 11_{2}$
$1_{10} = = 1_{2}$

En base 2, la tabla de sumar es muy simple:

$0 + 0 = 0_{2}$
$0 + 1 = 1_{2}$
$1 + 0 = 1_{2}$
$1 + 1 = 10_{2}$ (0 y llevo 1)
$1 + 1 + 1 = 11_{2}$ (1 y llevo 1)

EJEMPLO: En base 16

arrastres 1

$$\begin{array}{rr} & 1\ E\ 9\ 2_{16} \\ + & 5\ 0\ A\ 3_{16} \\ \hline & 6\ F\ 3\ 5_{16} \end{array}$$

$2+3 = 5_{10} = = 5_{16}$
$9+A = 19_{10} = 16+3_{10} = 13_{16}$
$1+E+0 = 15_{10} = = F_{16}$
$1+5 = 6_{10} = = 6_{16}$

RESTA

Analicemos cómo se realiza la operación resta en la base decimal suponiendo que el minuendo sea siempre mayor o igual que el sustraendo:

```
  7 8 4 2     2 < 7: (2 +10) - 7 = 5   y llevo 1
-                    4 - (2 + 1) = 1
  4 4 2 7            8 - 4       = 4
arrastres             7 - 4       = 3
  ---------
  3 4 1 5
```

A la vista de estas operaciones observamos lo siguiente:

Cuando la cifra del minuendo (término superior) es menor que la del sustraendo (término inferior) incluyendo el posible arrastre anterior, se suma la base a la cifra del minuendo y se resta del sustraendo (incluido el posible arrastre anterior), creando un arrastre de una unidad en la columna siguiente de la izquierda.

Cuando la cifra del minuendo es mayor que el sustraendo más el posible arrastre anterior, sencillamente se resta el minuendo del sustraendo (incluido el arrastre anterior dentro del sustraendo).

En cualquier otra base se hará de forma análoga, como se muestra en los ejemplos siguientes:

EJEMPLO: En base 8

```
    5 7 4 2(8)   2 < 3:    (2+8) - 3 = 7   y llevo 1
-                4 < (6+1)  (4+8)-7 = 5   y llevo 1
    1 0 6 3(8)            7 - (0+1) = 6
arrastres          1 1      5 - 1   = 4
    -----------
    4 6 5 7(8)
```

EJEMPLO: En base 2

```
    1 1 0 1 1(2)          1 - 1 = 0
-                         1 - 0 = 1
      1 1 0 1(2)  0 < 1   (0+2)-1 = 1   y llevo 1
arrastres         1<(1+1) (1+2)-2 = 1   y llevo 1
    1 1 --------          1 - 1   = 0
    0 1 1 1 0(2)
```

En base 2, la tabla de restar es muy simple:

$$0 - 0 = 0_2$$
$$0 - 1 = 1_2 \quad \text{(y llevo 1)}$$
$$1 - 0 = 1_2$$
$$1 - 1 = 0_2$$
$$1 - (1 + 1) = 1_2 \quad \text{(y llevo 1)}$$

EJEMPLO: En base 16

$$\begin{array}{rr} & 8\ E\ 1\ 2_{16} \\ - & \\ & 5\ 0\ A\ 3_{16} \\ \text{arrastres} & \quad 1\ 1 \\ \hline & 3\ D\ 6\ F_{16} \end{array}$$

$2 < 3 \qquad (2+16) - 3 = 15_{10} = F_{16}$ y llevo 1

$1 < (10+1) \qquad (1+16) - (10+1) = 6_{16}$ y llevo 1

$14 - (0+1) = 13_{10} = D_{16}$

$8 - 5 = 3_{10} = 3_{16}$

SUBSTRACCION POR COMPLEMENTO

El complemento de unos de un número binario es simplemente la diferencia entre uno y cada uno de los dígitos del número, puesto que 1 - 0 = 1 , 1 - 1 = 0, la complementación en el sistema binario, consiste simplemente de escribir un uno por un cero y un cero por un 1. Luego el complemento de 10010 es 01101 y el complemento de 11101 es 00010.

Para restar por el sistema de complementos de uno más uno y se hace caso omiso de uno inicial (acarrear al extremo).

Utilicemos este método en el ejemplo previo de 110011 (=51) menos 11101 (=29):

$$\begin{array}{cccccc} 1 & 1 & 0 & 0 & 1 & 1 \\ + & 0 & 0 & 0 & 1 & 0 \quad \text{(COMPLEMENTO DE 11101)} \\ \hline 1 & 1 & 0 & 1 & 0 & 1 \\ + & & & & & 1 \quad \text{(ACARREAR AL OTRO EXTREMO)} \\ \hline & 1 & 0 & 1 & 1 & 0 \ (22)_{10} \end{array}$$

La respuesta, 10110 (=22) coincide con la obtenida previamente por substracción binaria convencional. Como por ejemplo, considérese el siguiente problema:

COMPLEMENTANDO

```
  1  0  0  1  0  1 = 37    1  0  0  1  0  1
-    1  1  0  0  1 = 25  +    0  0  1  1  0
  0  0  1  1  0  0 = 12    1  0  1  0  1  1
                                          1
                          ----------------
                             0  1  1  0  0
```

Los ceros adicionales frente a las respuesta binarias, naturalmente, no tienen significación.

La mayor parte de las computadoras digitales efectúan la substracción binaria sumando los complementos a unos. Puesto que la suma se lleva a cabo por referencias a la tabla de adición binaria almacenada en la memoria y la complementación de inspección (también una función de memoria), de hecho se elimina toda la aritmética real, en esta forma.

MULTIPLICACION (BINARIA)

Abordemos ahora la multiplicación, que solamente veremos en el sistema binario, ya que en las otras bases tiene un escaso interés práctico, además de una cierta complejidad.

Las reglas de la multiplicación binaria son:

```
0 x 0 = 0  ¦  1 x 0 = 0
0 x 1 = 0  ¦  1 x 1 = 1
```

```
                    10.11   (multiplicando)
                  x 110.1   (multiplicador)
                     1011
                    0000
                   1011
                  1011
acarreo (suma)   1111
                 10001.111
```

Aplicando las reglas anteriores se observan las siguientes conclusiones prácticas:

- Cuando el multiplicador es 1, se copia el multiplicando.
- Cuando el multiplicador es 0, se copian ceros.

La multiplicación binaria, y cualquier multiplicación, también se puede efectuar mediante adiciones repetidas: para multiplicar A x B, simplemente se suma A asimismo el número de veces indicado por B (en forma decimal). De manera semejante la división se puede efectuar por sustracciones repetidas.

```
110    X    11  =
 A          B


  110
+ 110     11(3) veces
+ 110
10010
```

La computadora puede realizar con sumas todas las operaciones aritméticas.

DIVISION BINARIA

La división en cualquier sistema es la inversa de la multiplicación. Es el proceso que determina cuántas veces puede restarse un número (divisor) de otro número (dividendo), quedando aún un remanente positivo. Esto es exactamente lo que queremos decir cuando nos preguntamos cuantas veces cabe un número (divisor) en otro (dividendo).

Por definición, siempre puede ser efectuada una división restando el divisor de el dividendo un número de veces hasta que una resta más dejará un remanente negativo.

El número de veces que pueda hacer esto, es el resultado o cociente.

DIVIDIR 478 POR 94.

METODO CONVENCIONAL (DIVISION LARGA)

```
   5     COCIENTE
94 /478
   470          RESTA
     8      REMANENTE
```

En forma semejante puede lograrse la división binaria: por ejemplo, la división del binario 110111 (=55) por 101 (=5) arroja el binario 1011 (=11) utilizando la división larga.

```
   1011      COCIENTE
101/110111   DIVIDENDO
  101
  00111
    101
    0101
     101
     000     REMANENTE
```

EJEMPLO: En Octal

```
    3 7
121/472,1
- 363
  107 1
 -106 7
    2
```

REPRESENTACION DE NUMEROS DE MUCHAS CIFRAS

Las computadoras actuales están preparadas para trabajar a un nivel interno, con números enteros del tamaño de 1 ó 2 bytes, como ocurre en las PC (computadoras personales).

Mediante programas añadidos se puede trabajar con números mayores, como suele ocurrir cuando utilizamos un lenguaje de alto nivel, como el BASIC, el COBOL o el PASCAL.

Los datos en una computadora se almacenan en general en formato binario, según las siguientes configuraciones de cifras:

a) Con 1 byte se pueden formar $2^8 = 256$ combinaciones de 0 y 1.

- Si son números enteros sin signo irán del 0 al 255.
- Si son números enteros con signo irán del -128 al 127, incluyendo el número 0 (la mitad son los negativos, y la otra mitad son los positivos incluyendo el 0).

b) Con 2 bytes se pueden formar $2^{16} = 65,536$ combinaciones de 0 y 1.

- Si son números enteros sin signo irán del 0 al 65,535.
- Si son números enteros con signo irán del -32,768 al 32,767, incluyendo el número 0.

En los lenguajes de alto nivel, el formato de los enteros utilizado suele ser actualmente de 2 bytes. Si son números con signo van, como ya hemos señalado, desde -32,768 hasta 32,767, lo que puede ser un rango de valores muy pequeño en muchos casos.

Para conseguir un rango mayor, podemos elegir mayores posiciones de memoria para su almacenamiento, y manejar enteros mayores. Por ejemplo, con cuatro bytes podemos conseguir números con signo entre, aproximadamente, -2000 millones y 2000 millones, que ya es un rango sensiblemente mayor que el anterior, pero todavía no cubre muchos casos prácticos; por ejemplo, dentro del mundo de las finanzas o de la astronomía. El rango de los números enteros en la computadora se resume en la siguiente tabla:

	CON SIGNO	SIN SIGNO
1 BYTE	[0, 255]	[-128, 127]
2 BYTE	[0, 65 535]	[-32 768, 32 767]
4 BYTE	[0, 4 294 967 296]	[-2 147 483 647, 2 147 483 647]

Otra representación que además permite trabajar con números fraccionarios, es el formato conocido como notación exponencial o científica.

Veamos a continuación en qué consiste esta representación.

Cuando un número es muy grande o muy pequeño, se suele abreviar escribiéndole en notación exponencial, también llamada coma flotante. Así:

$$148\ 000\ 000\ 000\ 000 = 148 \cdot 10^{12}$$

$$0.000\ 000\ 000\ 012 = 012 \cdot 10^{-10}$$

Según donde se coloque el punto (coma) decimal hay muchas representaciones. Por esto, se define la notación exponencial normalizada como aquélla en la que a continuación del punto decimal y a su derecha se coloca la primera de las cifras significativas (o sea, la primera cifra no nula).

Por ejemplo:

$$123.456 = 0.123456 \times 10^{3}$$

$$0.0023 = 0.23 \times 10^{-2}$$

FORMATO DE COMA FLOTANTE

El formato de coma flotante en cualquier sistema de numeración tiene la forma siguiente:

$$\text{MANTISA} \cdot \text{BASE}^{\text{EXPONENTE}}$$

Estos números, en una base dada, tienen dos partes: la mantisa, que indica el valor de los dígitos significativos, y el exponente, que señala el lugar donde se coloca la coma "decimal", por esto se denomina "coma flotante".

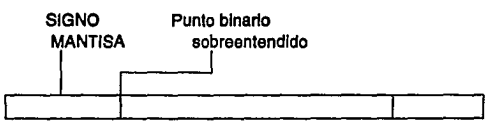
Dependiendo de la computadora utilizada y del propio lenguaje de programación, los formatos internos de punto flotante son muy diversos. Nosotros realizaremos la descripción de un caso que suele ser bastante general, y que es el más aplicado en las PC. Este sistema consiste en almacenar el número en forma exponencial binaria.

Hablaremos de dos formatos en punto flotante: simple precisión y doble precisión. Su diferencia únicamente suele afectar al número de cifras significativas de la mantisa que se almacenan; que en el primer caso resultan ser siete cifras significativas, y en el segundo, 16.

Hablaremos primeramente del formato de punto flotante de simple precisión. Suele ocupar 4 bytes, cuya información se divide en tres partes:

- **Signo del número** (signo de la mantisa), que basta con 1 bit (0 para el signo + y 1 para el -).
- **Mantisa**: Para la que reservaremos 23 bits (que junto con el signo anterior son 3 bytes).
- **Exponente del número**: Para nosotros será de 8 bits, incluyendo el signo del exponente.

Puesto que el número se almacena en formato normalizado, el punto binario está sobreentendido justo delante de la mantisa [0.mantisa].



El campo exponente (incluido el signo de éste) suele representarse por su característica, que se obtiene tomando el valor del exponente verdadero del número y sumándole un

desplazamiento, que se calcula como 2^{n-1}, llamando n al número de bits del campo exponente.

característica = exponente verdadero + desplazamiento

desplazamiento = 2^{n-1} (n = núm. de bits del campo exponente)

En nuestro caso, n = 8, luego el desplazamiento es 128.

Por esto, también se dice que el exponente se escribe en código exceso a 128. El campo exponente de 8 bits puede así representar 256 números (2^8), que admite exponentes desde -128 a +127, lo cual significa que la computadora puede almacenar números en punto flotante con un rango de valores absolutos comprendido entre:

$$2^{-128} \quad y \quad 2^{+127}$$

es decir, aproximadamente,

$$10^{-38} \quad y \quad 10^{+38}$$

La relación entre el exponente vardadero y su característica se observa también en la siguiente tabla:

Exponente verdadero	-128	-127 ... -1	0	1 ... 127
Característica	0	1 ... 127	128	129 ... 255

En los números de doble precisión, el formato inicial es el mismo, y lo único que suele variar es el tamaño de la mantisa que es mayor. De este modo, al no variar normalmente el tamaño del campo exponente, el rango de valores es el mismo.

EJEMPLO:

Obtener el formato de punto flotante en simple precisión del número A = -215.265

Primeramente lo convertimos a formato binario puro:

Parte entera:

```
215 | 2
    +-----+
015   107 | 2
      +---+
   1    07   53 | 2
             +---+
        1    13   26 | 2
                  +---+
             1    06   13 | 2
                       +---+
                  0    1    6 | 2
                            +---+
                       0    3 | 2
                              +---
                            1   1
```

$$215_{10} = 11010111_{2}$$

Parte fraccionaria:

```
0.625 x 2 = 1.25
0.25  x 2 = 0.5
0.5   x 2 = 1.0
```

$$0.625_{10} = 0.101_{2}$$

Entonces,

$$215.625_{10} = 11010111.101_{2}$$

Luego

$$A = -11010111.101_{2}$$

Y en forma exponencial normalizada será

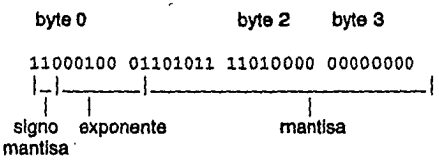
$$A = -0.11010111101 \times 2^{+8}$$

El exponente verdadero es +8, y la característica será:

$$8 + 128 = 136_{10} = 10001000_2$$

El signo de la mantisa lo indicaremos así: "0" si es positivo, "1" si es negativo. En nuestro caso es negativo, luego bit 1 para el signo.

Entonces con el formato anterior, la memoria contendrá:

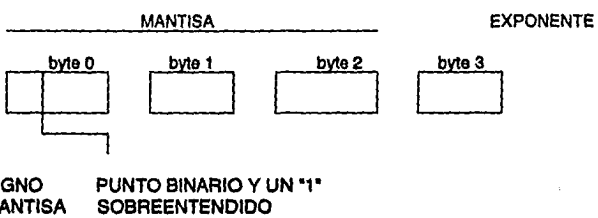


El formato real usado no es exactamente el anterior, pues tiene unos pequeños cambios que vamos a indicar:

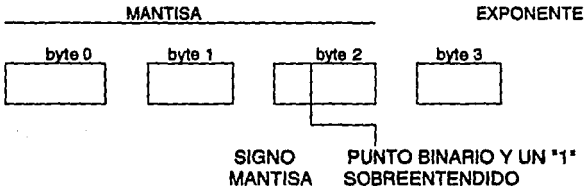
1. En el formato anterior podemos ahorrarnos un bit. Puesto que la mantisa está normalizada, cuando a partir del punto se coloca una cifra significativa, al ser un formato binario, esa primera cifra significativa siempre es "1" y, por tanto, se puede dar por supuesto. Entonces, en el campo mantisa daremos por sobreentendido que delante de él (a continuación del bit de signo), irá siempre el punto binario y un "1" de primera cifra significativa [0.1 mantisa].

2. Y el otro cambio afecta al orden de colocación:

   - El exponente se pone al final, en el último byte:



Los bytes de la mantisa, incluyendo el signo, están intercambiados entre sí: el byte 0 donde el byte 2, y el byte 2 donde está el byte 0.



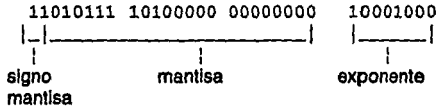
Con el ejemplo anterior se tenía:

SIGNO MANTISA: 1

MANTISA (sin el signo): 1101011 11010000 00000000

EXPONENTE (característica): 10001000

Eliminando el primer "1" de la mantisa, que se da por sobreentendido y colocando el signo delante tenemos:



E intercambiando los bytes de la mantisa, tenemos finalmente:

```
00000000 10100000 11010111 10001000
```

O en hexadecimal: `00 A0 D7 88.`

2.3 CODIGOS

La transferencia de información de un dispositivo a otro de la computadora, desde un punto de vista físico, es la presencia de un impulso eléctrico que va de un lugar a otro.

Por lo tanto, analizando físicamente lo que se transmite como un número, es una secuencia de impulsos que representa una cantidad binaria (base 2).

Pero el hecho de que sólo números pueden ser interpretados por la computadora, no significa que únicamente información numérica se pueda manejar. Es posible hacer que los números representen letras; para realizarlo, es necesario formular el conjunto de caracteres válidos y asignarle a cada carácter del conjunto un número o clave.

En principio cualquier número se puede asociar a algún elemento del conjunto; sin embargo, es conveniente establecer ciertas reglas para utilizarlos.

Un código, es un sistema de símbolos para representar datos o instrucciones en una computadora.

Los códigos, dependiendo del número de elementos que contengan necesitan un determinado número de bits para ser representados.

Un código que utilice seis bits, es el de menor longitud posible si se quiere que contenga letras, dígitos y algunos caracteres especiales y de puntuación como: , . " : + etc., que representan un conjunto de caracteres apropiado al lenguaje común.

En la medida en que se han incluido nuevos caracteres a los conjuntos, el código de seis bits ha resultado insuficiente por lo que se han generado nuevos códigos de 7 y 8 bits.

Entre los diferentes códigos de máquina que existen actualmente se pueden mencionar:

CODIGO BINARIO

Conocido también como código natural de todas las computadoras (aunque existen casos aislados de máquina que utilizan código ternario), el cual sólo emplea un bit para su representación y es la agrupación de todos ellos la que determina los demás códigos.

CODIGO HEXADECIMAL

Cuenta con 16 posibilidades del 0 al 9 y de la A a la F, que se utilizan como simplificación para legibilidad del código binario por la relación numérica existente entre la base 2 y la base 16, esto es, $2^4 = 16$, lo que permite que una agrupación de bits exprese su equivalencia en binario.

$$(2^4)^0 = 16^0 = 1,\ (2)^{4x1} = (2^4) = 16^1,\ (2^4)^2 = (2^8) = 16^2 \text{ etc.}$$

Los códigos binario y hexadecimal no representan un código de máquina en el sentido de que no es posible especificar todo un conjunto de caracteres válido en dichos códigos, sino que se necesita una equivalencia numérica para ciertos caracteres; por ejemplo, no se puede prrecisar en hexadecimal o en binario el caracter "$" como tal, sino que se requiere de una expresión binaria o hexadecimal del código de máquina que determine dicho caracter.

Los códigos de máquina capaces de representar conjuntos de caracteres "completos", esto es, grupos capaces de contener los elementos necesarios para hacer extensivos a la computadora el manejo de textos, expresiones aritméticas, etc., y que por lo tanto constan de los elementos del alfabeto, dígitos del 0 al 9, operadores aritméticos y caracteres de puntuación o especiales, son entre otros los siguientes:

CODIGO BCD

Código BCD "Binary Coded Decimal", un sistema de representación decimal que utiliza 4 bits para representar cada dígito decimal; este código no permite la representación de caracteres alfabéticos.

CODIGO BCL

Código BCL "Burroughs Common Language", un sistema de función alfanumérica que usa 6 bits para mostrar sus elementos.

CODIGO EBCDIC

Código EBCDIC "Expanded Binary Coded Decimal Interchanged Coded", que emplea 8 bits para definir los elementos del conjunto y es el código utilizado por omisión para la Burroughs B-6700.

CODIGO ASCII

Código ASCII "American Standard Character Interchanged Information", que emplea 7 bits para describir el conjunto de caracteres y es usado comúnmente para la transmisión de datos en unidades periféricas como los teletipos o pantallas.

Es conveniente hacer notar que el hecho de que un código permita 64, 128 ó 256 posibles elementos a describir no significa que estos códigos agoten las posibilidades, esto es el código EBCDIC, no consta de los 256 caracteres posibles a describir sino que utiliza sólo un subconjunto de representaciones para determinar su conjunto de caracteres válidos pero es posible incluir nuevos elementos al conjunto, dado que existen representaciones en el código, que no han sido usadas, o bien que no corresponden aún a algún elemento del conjunto y son considerados como caracteres inválidos.

Existe, además, una relación entre el código y la longitud de palabra dentro de una computadora dada.

Dentro de una palabra de memoria, se pueden almacenar varios bytes, esto es, depende del número de bits que ocupe cada byte o caracter y del código que se esté manejando.

Supongamos que se está trabajando con una computadora Burrouhgs, modelo B6700, la cual tiene una longitud de palabra de 48 bits para almacenar información en ellos y cuatro bits de control.

Bits de Control	Bits de almacenamiento											
51	47	43	39	35	31	27	23	19	15	11	7	3
50	46	42	38	34	30	26	22	18	14	10	6	2
49	45	41	37	33	29	25	21	17	13	9	5	1
48	44	40	36	32	28	24	20	16	12	8	4	0

Si estuviéramos manejando código binario se podría tener dentro de ella hasta 48 bytes (del 0 al 47), en código haxadecimal 12 bytes, en BCL 8 bytes, en ASCII 6 bytes y desperdiciaríamos 6 bits, en EBCDIC 6 bytes exactamente.

```
BINARIO   1 BIT  = 1 BYTE     48 por palabra
HEX       4 BITS = 1 BYTE     12 por palabra
BCD       4 BITS = 1 BYTE     12 por palabra
BCL       6 BITS = 1 BYTE      8 por palabra
ASCII     7 BITS = 1 BYTE      6 por palabra
EBCDIC    8 BITS = 1 BYTE      6 por palabra
```

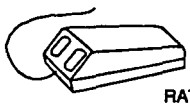
Se puede observar ahora que los códigos de máquina son la interpretación numérica de elementos numéricos y no numéricos de los lenguajes naturales y sus caracteres gramaticales que permiten la generalización del uso de las computadoras.

2.4 COMPONENTES DE UNA COMPUTADORA. SOPORTE FISICO (HARDWARE)

DISPOSITIVOS DE ENTRADA

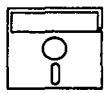
Los sistemas de cómputo emplean dispositivos de entrada, que son máquinas diseñadas para la captura de datos.

COMPUTADORA PERSONAL



RATON

CINTA MAGNETICA



DISCO FLEXIBLE

FIGURA 2.1

Algunos dispositivos de entrada permiten la comunicación directa entre los seres humanos y la máquina, mientras que otros requieren la grabación de los datos en un medio de entrada, por ejemplo en un material magnetizable.

Con frecuencia se utilizan dispositivos que leen datos grabados en plástico flexible con un recubrimiento especial (disco flexible) o en cinta magnética. El teclado de una estación de trabajo conectado directamente a una computadora se dice que está en línea con ella.

Existen dispositivos de entrada directa: el ratón, el lápiz de entrada, la pantalla sensible al tacto, el micrófono, tableta digitalizadora, scanner, compactdisc, modem, panel de proyección, audioport. Todos estos dispositivos son componentes que interpretan la información, y permiten la comunicación entre las personas y entre los sistemas de computo.

UNIDAD DE PROCESO

El corazón de todo sistema de cómputo es la unidad de proceso, que contiene los elementos de almacenamiento primario, aritmética-lógica y control.

A.- La sección de almacenamiento primario.

La sección de almacenamiento primario (llamada también memoria principal) se utiliza para cuatro funciones. Tres de ellas se relacionan con los datos que se están procesando.

a. Los datos se introducen a un área de almacenamiento de entrada y permanecen en ese lugar hasta el momento que se van a procesar.

b. Un espacio de memoria de trabajo es como una hoja de papel para hacer cuentas y contendrá los datos que se están procesando, así como los resultados intermedios de dicho procesamiento.

c. Un área de almacenamiento de salida guarda los resultados finales del procesamiento hasta que pueden ser liberados.

d. Además de estas áreas relacionadas con los datos, la sección de almacenamiento primario también contiene un área de almacenamiento de programas, que guarda las instrucciones de procesamiento.

Las distintas áreas que destinan las cuatro funciones generales mencionadas no se fijan por límites físicos incorporados en la sección de almacenamiento primario, sino que pueden variar en las diferentes aplicaciones.

Por lo tanto, un espacio físico determinado puede contener datos de entrada en una aplicación, resultados para salida en otra e instrucciones de procesamiento de una tercera. El programador que escribe las instrucciones de la aplicación (o los programas de administración de los recursos de cómputo preparados por otros programadores) determinan la forma en que se va a utilizar el espacio para cada tarea.

B.- La sección aritmética-lógica.

Juntas las secciones aritmética-lógica y de control constituyen la unidad central de procesos (UCP). Todos los cálculos y todas las comparaciones se realizan en la sección aritmética-lógica de la UCP. Una vez que los dispositivos de entrada introdujeron los datos a la memoria principal, se guardan y se transfieren, cuando es necesario, a la sección mencionada, donde se lleva a cabo el procesamiento.

En el almacenamiento primario no se lleva a cabo ningún proceso. Los resultados intermedios que se generan en la unidad aritmética-lógica se sitúan temporalmente en un área designada de memoria de trabajo hasta que se vuelva a necesitar.

Así los datos pueden pasar de almacenamiento primario a la unidad aritmética-lógica y volver a la memoria muchas veces antes que se termine el proceso. En este momento los resultados finales se envían a una sección de almacenamiento de salida y de ahí a un dispositivo de salida.

C.- La sección de control.

Al seleccionar, interpretar y ordenar la ejecución de las instrucciones del programa, la sección de control de la UCP mantiene el orden y dirige la operación de todo el sistema. Aunque la sección de control no procesa datos, actúa como un sistema nervioso central para los demás componentes manipuladores de datos de la computadora.

Al comenzar el procesamiento, se elige la primera instrucción del programa y se pasa del área de almacenamiento de programas a la sección de control. Ahí se le interpreta y se envían señales a otros componentes para que se lleven a cabo las instrucciones del programa, una tras otra, hasta terminar el proceso.

DISPOSITIVO DE ALMACENAMIENTO SECUNDARIO

En casi todas las computadoras se emplean dispositivos de almacenamiento secundario (o auxiliar) para complementar la limitada capacidad de almacenamiento de la sección de almacenamiento primario. Los dispositivos de almacenamiento secundario están en línea con el procesador.

Aceptan datos o instrucciones del programa del procesador, los conservan y los vuelven a introducir en el procesador cuando se necesitan para llevar a cabo tareas de procesamiento.

Los discos rígidos suelen encontrarse permanentemente sellados dentro de sus dispositivos de almacenamiento, pero los discos flexibles se cambian con facilidad. Los datos o instrucciones almacenados en un disco flexible permanecen intactos después de sacar el disco, pero dado que el procesador ya no tiene acceso directo y automático a ellos, se dice que está fuera de línea.

DISPOSITIVOS DE SALIDA

Al igual que las unidades de entrada, los dispositivos de salida son instrumentos que interpretan información y permiten la comunicación entre los seres humanos y las computadoras.

Estos dispositivos convierten los resultados que produce el procesador y que están en código de máquina en una forma susceptible de ser empleada por las personas (por ejemplo: informes impresos o desplegados en pantallas) o como entrada para otras máquinas que formen parte de un ciclo de procesamiento distinto.

En los sistemas personales de cómputo, los dispositivos de salida más populares son las

pantallas de exhibición y las impresoras de escritorio, plotter. Los dispositivos que aceptan las salidas de sistemas mayores generalmente son impresoras más grandes y rápidas, muchas estaciones de trabajo en línea y unidades de cinta magnética y disco rígido.

A las unidades de entrada/salida y almacenamiento secundario se les llama en ocasiones dispositivos periféricos (o simplemente periféricos) porque a menudo se les coloca cerca de la unidad de proceso.

ALMACENAMIENTO Y REPRESENTACION DE LA INFORMACION

Existen dos clases de almacenamiento de la computadora. Una se conoce como memoria principal y la cual reside en la unidad central de proceso y sirve para almacenar los datos e instrucciones que van a procesarse.

La memoria principal puede encontrarse en uno o varios bloques y se divide en localidades llamadas "direcciones de memoria". El acceso a la memoria es por direcciones y van de la cero a la N.

Cada dirección es capaz de retener un caracter, esto es, una letra, un dígito, o un caracter especial como un punto o un guión.

La seguda clase se conoce como memoria secundaria; se refiere a un "almacén" auxiliar o periférico. Son dispositivos capaces de contener información que puede ser recuperada.

AGRUPACION DE ALMACENAMIENTO

ARCHIVO

REGISTRO 1
REGISTRO 2
REGISTRO 3
.
.
.
.
REGISTRO n

REGISTRO

CAMPO 1	CAMPO 2	...	CAMPO n

CAMPO

CARACTER 1	CARACTER 2	...	CARACTER n

Estos dispositivos pueden ser discos o cintas magnéticas y sirven para guardar grandes volúmenes de datos que no serían costeables y en muchas ocasiones ni siquiera posibles de almacenar en la memoria principal.

La información se conserva en agrupaciones de requisitos llamados archivos.

La memoria de la computadora retiene datos de dos clases básicas:

1. Datos para ser operados, tales como información para una nómina, información textual, etc.

2. Datos que son un conjunto de instrucciones que indican, de manera específica, lo que se va a hacer con la información. A este conjunto de instrucciones se le da el nombre de "programa" de computadora.

2.5 SISTEMA OPERATIVO

Un sistema operativo es un conjunto integrado de programas que residen en la memoria principal cuya función primordial es la administración y control de los recursos del computador. Entre estos elementos destacan, por su importancia, la memoria principal, el procesador central (UCP), los dispositivos periféricos de entrada/salida (E/S), así como la información y datos de los archivos del sistema.

El actuar entre una interfase entre el usuario y el hardware de la computadora es otra de las funciones de este sistema. La complejidad de la máquina virtual es relativamente sencilla de operar. Por ejemplo, la operación de entrada/salida en la máquina real puede ser extremadamente compleja y puede requerir programación sofisticada; sin embargo, el sistema operativo releva al usuario del problema de entender esta complejidad y le presta una máquina virtual mucho más fácil de usar e igualmente poderosa que la máquina original.

El usuario interactúa con el sistema operativo a través de un lenguaje de control de tareas (JOB CONTROL LENGUAGE O JCL), o también usando el lenguaje de control del operador, en tanto los programas lo hacen a través de llamadas al supervisor (SVC), también nombrados requerimientos al ejecutivo (ER). Cabe mencionar que el sistema operativo también es conocido como supervisor o programa ejecutivo.

Finalmente, otra función importante del sistema operativo es inicializar el hardware del computador para que esté disponible para ejecutar el primer programa. Cuando el computador es puesto a trabajar al inicio de un determinado día, es necesario cargar el sistema operativo de disco o memoria principal.

Esto se conoce como carga inicial del programa (IPL) y consiste en un mecanismo de hardware o firware para leer un pequeño programa (BOOTSTRAP) el cual a su vez instala el sistema operativo en la memoria. El procesador, la memoria principal y los periféricos también son inicializados para aceptar las primeras tareas durante el proceso de inicialización.

2.6 DOCUMENTACION DEL SISTEMA

Existen tres grupos de personas que necesitan conocer la documentación del programa:

- Programadores
- Operadores
- Usuarios

Los requerimientos necesarios para cada uno de ellos suelen ser diferentes.

En entornos interactivos, las misiones del usuario y operador pueden ser las mismas; po lo que, la documentación del sistema se puede concretar a:

- Manual del usuario
- Manual de mantenimiento

MANUAL DEL USUARIO

El manual del usuario debe cubrir al menos los siguientes puntos:

- Ordenes necesarias para usar el sistema
- Nombres de los archivos externos a los que accesa el programa
- Formato de todos los mandatos de mensajes de error o informes
- Opciones en el funcionamiento del sistema
- Descripción detallada de la función realizada por cada módulo del sistema
- Descripción detallada de cualquier salida producida por el sistema

MANUAL DE MANTENIMIENTO

El manual de mantenimiento es la documentación requerida para mantener un programa durante su ciclo de vida.

Se divide en dos categorías:

- Documentación interna
- Documentación externa

Documentación Interna

Cubre los aspectos de cada programa que conforme el sistema; esta información está contenida en los comentarios. Algunos tópicos a considerar son:

- Cabecera del Programa. Nombre del programador, fecha, breve descripción de la función a realizar
- Nombres significativos para describir identificadores
- Comentarios relativos a la función de cada módulo del programa
- Claridad de estilo y formato. Una sentencia por línea, sangrado, líneas en blanco para separar módulos, etc.
- Comentarios significativos

Documentación Externa

Son documentos ajenos al programa fuente, y deberán incluir:

- Listado actual del programa fuente
- Especificación del programa
- Explicaciones de fórmulas complejas
- Especificaciones de los datos a procesar archivos externos, incluyendo el formato de las estructuras de éstos

2.7 TIPOS DE PROCESAMIENTOS

PROCESAMIENTO EN LOTES

Este procesamiento también es conocido con el anglicismo "BATCH"; implica la carga de un bloque de datos para que sufran un proceso predefinido, a diferencia de los procesos interactivos, en donde se tiene un diálogo entre el programa y el usuario, y donde generalmente se van introduciendo y procesando unidades aisladas de información.

El proceso en lotes generalmente se lleva a cabo programa por programa y requiere por tanto de un sistema operativo muy simple y de áreas de memoria pequeñas.

Es posible que cuando se trabaje en lotes con una computadora, se quiera hacer enlances de programas; esto es, una vez que se haya ejecutado un programa de principio a fin, se cargue

y ejecute automáticamente el siguiente programa en secuencia; un sistema operativo debe tener esta mínima capacidad, esto puede lograrse con el apoyo de comandos o directrices que controlan la actividad del sistema operativo.

La tendencia para aprovechar cada vez más el potencial de la computadora, ha provocado una creciente innovación de procedimientos que se han sumado en los nuevos sistemas operativos, de manera que la filosofía de procesar en lotes programa por programa se ha quedado atrás, fortaleciéndose la utilización de sistemas operativos capaces de atender varios programas concurrentes en la memoria principal.

PROCESAMIENTO EN TIEMPO REAL

Un sistema de tiempo real es un sistema dedicado completamente a actividades de control de procesos físicos (químicos, industriales, mecánicos, etc.), para garantizar tiempos de respuesta previamente especificados aún en condiciones de máxima carga y es diseñado en un sistema con configuración en línea el cual se define como un sistema de cómputo donde los datos van directamente desde el punto donde son generados (terminal de video o dispositivo de control) al computador y ahí la información de salida va directamente del computador al punto en el que se le necesite.

El factor importante en este caso es el tiempo de respuesta, ya que la contestación del computador al evento o proceso físico que controla, debe ser lo suficientemente rápida para influir en el desarrollo del proceso.

PROCESAMIENTO EN TIEMPO COMPARTIDO

Este es un sistema en el cual el tiempo del procesador central es distribuido entre un conjunto de usuarios en forma tal que proporcionan servicios de computación y tiempo de respuesta adecuada (relativamente corta) a todos ellos, dándole a cada usuario la impresión de que el computador está solamente a su servicio. En este sistema, también conocido como un sistema de demanda, se asigna a cada usuario la atención del procesador central o UCP cuando le toca su turno.

MULTIPROGRAMACION

Es la ejecución de dos o más programas concurrentemente en una sola unidad central de proceso.

La manera como se desarrolla la multiprogramación es a través de la división del tiempo del UCP en periodos de tiempo pequeño que son asignados a los diferentes procesos en ejecución concurrente.

MULTIPROCESO

Este tipo de proceso de datos consiste en tener más de una unidad central de proceso en un sistema de cómputo, las cuales trabajan al mismo tiempo.

Pueden estar conectadas directamente a través de canales o sólo lógicamente en función de las entradas y las salidas.

TELEPROCESO

Esta modalidad de proceso consiste en realizar el proceso de datos desde un lugar alejado del centro de cómputo. Esto es, que en un sitio lejano del computador central se le dan los comandos de control y los datos para ser procesados en el computador central y normalmente el resultado del proceso se regresa al lugar donde fue enviado el trabajo.

Para realizar el teleproceso se requiere hacer una transmisión de la información en el lugar de origen hasta el procesador central y viceversa; para esto, se utilizan medios de comunicación como líneas telefónicas, microondas, etc., y también se necesita un dispositivo que modula y demodula la señal, conocido como modem. Esto es, transforma la señal digital manejada por los computadores, las señales senoidales que puedan ser transmitidas y viceversa.

2.8 ASPECTOS A CONSIDERAR AL SELECCIONAR UN EQUIPO DE COMPUTO.

Para que la computación proporcione los mejores resultados, es necesario que se tomen en cuenta una serie de consideraciones previas a la adquisición tanto de equipo como de programas, los cuales nos llevarán a una automatización exitosa de las áreas que nos interesan de acuerdo con las necesidades del proceso de información que hayan sido identificados.

A continuación se presentan algunas consideraciones:

- La regla más importante antes de adquirir un equipo de cómputo es tener una idea clara de nuestras necesidades.

- Realizar un estudio del hardware de la máquina, considerando los siguientes puntos:

    - Modelo del UCP
    - Capacidad de la memoria principal y secundaria
    - Impresoras
    - No. de terminales
    - Unidades de respaldo

- Realizar un estudio del software del equipo, considerando los siguientes puntos:

    - Costo aproximado de cada aplicación
    - Existencia de aplicaciones
    - Sistema Operativo
    - Rutinas de utilería
    - Lenguajes

- Efectuar un estudio de los costos de equipos considerando los siguientes puntos:

  - Costo del equipo
  - Costo del software
  - Gastos de instalación
  - Derechos de importación

- Realizar un estudio de los servicios y costos de mantenimiento, tomando en cuanta:

  - Horario de mantenimiento
  - Tiempo de respuesta
  - Mantenimiento de equipo
  - Mantenimiento de software
  - Venta de software

El conocimiento detallado de estas opciones nos permitirá adquirir de una forma más segura el equipo de cómputo que mejor se adapte a nuestras necesidades.

CAPITULO III

ALGORITMOS Y DIAGRAMAS DE FLUJO

3.1 CONCEPTO DE ALGORITMO, SECUENCIA Y PROGRAMA

ALGORITMO

Una secuencia de pasos lógicos necesarios para llevar a cabo una tarea específica, como la solución de un problema se conoce como algoritmo. Además de lograr estos objetivos, un buen algoritmo debe poseer las siguientes características:

DEFINICION

Cada paso debe ser preciso; esto es, nada se puede dejar a la aventura. Los resultados finales no dependen de quién esté siguiendo el algoritmo. En este sentido, un algoritmo es análogo a una receta. Ejemplo: dos cocineros que trabajan independientemente con base en una buena receta deben preparar platillos esencialmente idénticos.

FINITUD

El proceso siempre debe terminar después de una cantidad finita de pasos. Un algoritmo no puede ser abierto.

ENTRADA/SALIDA

Entrada es la información que para ser procesada va a ser leída (datos); esta información trabajada será el resultado (salida) de un proceso.

EFECTIVIDAD

El algoritmo debe ser lo suficientemente general como para enfrentarse a cualquier contingencia.

En la figura 3.1 se ilustra un algoritmo que soluciona el problema sencillo de la suma de dos números. Dos programadores independientes que trabajen sobre esta secuencia pueden generar programas que muestren algunas diferencias en cuanto a estilo. Sin embargo, dados los mismos datos, los programas deben obtener el mismo resultado.

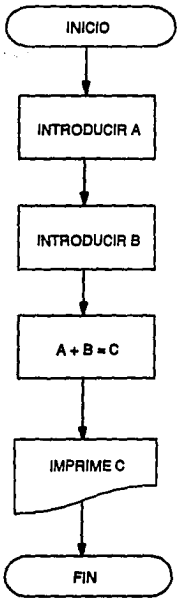


FIGURA 3.1

Las descripciones paso a paso redactadas en español y mostradas en la figura son una manera de expresar un algoritmo. Estos comentarios son particularmente útiles en problemas pequeños o para especificar las tareas principales de un proyecto grande de programación. Sin embargo, en las representaciones detalladas de los programas complicados, pronto se tornaran inadecuados. Por esta razón, se han elaborado alternativas visuales mas útiles, a las que se le ha llamado diagramas de flujo.

DIAGRAMAS DE FLUJO

Un diagrama de flujo es una representación visual o gráfica de un algoritmo. Este diagrama emplea una serie de bloques y de flechas; cada una de las cuales representa una operación en particular o un paso en el algoritmo. Las flechas indican la secuencia en la que se realizan las operaciones (Figura 3.2).

FIGURAS GEOMETRICAS NORMALMENTE EMPLEADAS EN LOS DIAGRAMAS DE FLUJO

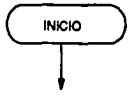
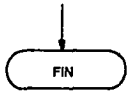
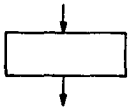
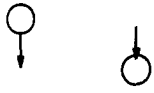
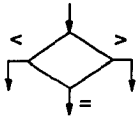
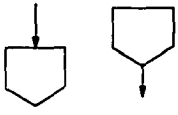
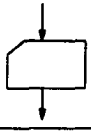
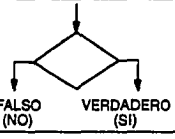
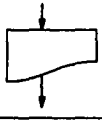
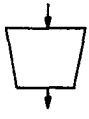
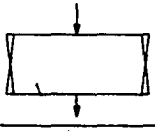
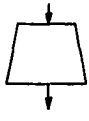
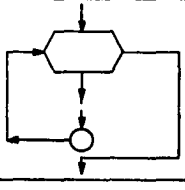
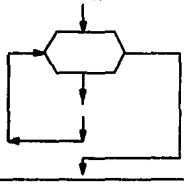
FIGURA	SIGNIFICADO	FIGURA	SIGNIFICADO
	PRINCIPIO DE PROGRAMA		FIN DEL PROGRAMA
	OPERACION ALGEBRAICA		CONECTOR
	DECISION NUMERICA		CONECTOR DE CAMBIO DE PAGINA
	LECTURA DE DATOS POR TARJETA		DECISION LOGICA
	RESULTADOS POR IMPRESORA DE LINEAS		ENTRADA DE INFORMACION POR CUALQUIER MEDIO
	ENTRADA Y SALIDA INMEDIATA DE INFORMACION		SALIDA DE INFORMACION POR CUALQUIER MEDIO
	OPERACION REPETITIVA O CICLICA		OPERACION REPETITIVA O CICLICA

FIGURA 3.2

CONCEPTO DE SECUENCIA

El algoritmo involucra pasos simples que deben ejecutarse uno después del otro. Como se mencionó, algoritmo es una secuencia de pasos, lo cual significa que:

(1) Los pasos se ejecutan uno a la vez.
(2) Cada paso se realiza exactamente una vez. Ninguno se repite y ninguno se omite.
(3) El orden en que los pasos se hacen es el mismo en el que están escritos.
(4) La terminación del último paso implica la terminación del algoritmo.

CONCEPTO DE PROGRAMA

Un programa es un conjunto de instrucciones que le dicen a la computadora qué hacer. A todos los programas necesarios para hacer funcionar una computadora en particular se le conocen con el nombre de software.

3.2 CONCEPTOS BASICOS DE PROGRAMACION ESTRUCTURADA

DISEÑO MODULAR

Los programas se pueden dividir en pequeños subprogramas o módulos que se escriben y prueban separadamente. A este procedimiento se le conoce con el nombre de diseño modular.

La característica más importante de los módulos es la de ser tan independientes y autosuficientes como sea posible. Además, casi siempre se diseñan para realizar una función específica y bien definida. Como tales, a menudo son breves (menos de 50 líneas) y con una finalidad muy clara.

Uno de los elementos principales de programación utilizados en la representación de cada módulo es la subrutina. Una subrutina es un conjunto de instrucciones de cómputo que realizan una tarea.

Un programa principal llama a estos módulos a medida que se necesitan. Por tanto, el programa principal dirige cada una de estas partes de manera lógica.

Los módulos tienen una entrada y una salida. Se pueden tomar decisiones dentro de un módulo que tenga repercusión en todo el flujo, pero el salto debe ser únicamente hacia el programa principal.

El diseño modular tiene algunas ventajas. El uso de unidades pequeñas y autocontenidas

facilita la proyección y la comprensión de la lógica subyacente para el programador y el usuario.

El desarrollo de un programa se puede efectuar con mayor facilidad, ya que cada módulo se crea aisladamente. De hecho, en proyectos muy grandes trabajan muchos programadores en unidades aisladas; asimismo, el usuario puede mantener una biblioteca de módulos que más tarde aprovecha en otros programas. El diseño modular también aumenta la facilidad de depuración y búsqueda de errores en un programa, ya que éstos se pueden aislar más fácilmente.

Por último, el mantenimiento y la modificación de los programas se facilitan. Esto es muy importante, ya que se pueden desarrollar nuevos módulos que realicen tareas adicionales e incorporarse fácilmente en un esquema coherente y organizado.

DISEÑO DESCENDENTE (TOP DOWN) O DISEÑO ARRIBA-ABAJO

El diseño descendente es un proceso de desarrollo sistemático que empieza con las instrucciones más generales de los objetivos de un programa para enseguida dividirlos sucesivamente en segmentos más detallados; por lo que el diseño va de lo general a lo específico.

La mayoría de los diseños descendentes empieza con una explicación en el idioma natural del programa en sus términos más generales. Esta descripción se divide en algunos elementos que definen las operaciones principales del programa. Enseguida, cada uno de los componentes principales se divide en subelementos. Este procedimiento se continúa hasta identificar módulos bien definidos.

Además de proporcionar un procedimiento para dividir el algoritmo en unidades bien definidas, el diseño descendente tiene algunas ventajas más. En particular, asegura que el producto final comprenda todo. Al comenzar con una definición vaga y agregando detalles progresivamente, es menos probable que el programador pase por alto operaciones importantes.

PROGRAMACION ESTRUCTURADA

La programación estructurada, estudia cómo se genera el código real del programa, de tal modo que sea fácil de entender, corregir y modificar.

Nadie ha coincidido en lo que significa la programación estructurada. En este contexto, se define como un conjunto de reglas que aconsejan buenos hábitos de estilo a los programadores. Las siguientes reglas son las más generales, y en las que concuerdan más personas:

1.- Los programas deben constar únicamente de tres estructuras fundamentales de control: secuencia, selección y repetición. Los científicos de la computación han demostrado que los programas se pueden construir a partir de estas estructuras básicas.

2.- Cada una de las estructuras debe tener únicamente una salida y una entrada.

3.- Se debe evitar las transferencias incondicionales GOTO (transferencia de control).

4.- Las estructuras se deben identificar claramente con comentarios y ayudas visuales como sangrías y líneas, espacios en blanco.

La programación estructurada tiene las siguientes ventajas:

1. Soluciones cuya lógica se sigue fácilmente.
2. Tiempo de depuración y prueba reducido.
3. Mayor productividad del programador.
4. Fácil de leer y comprender.
5. En una amplia gama de lenguajes y en diferentes sistemas fácil de codificar.
6. Eficiente, aprovechando al máximo los recursos del computador.
7. Modularizable.
8. Transportable.

El efecto de la programación estructurada se mide mejor en programas largos y complicados, en los que las relaciones lógicas deben estar definidas claramente.

Los programas grandes que adoptan estos principios son mantenidos más fácilmente y son más sencillos de modificar cuando es necesario aumentarlos.

METODO DE WARNIER

Este método se basa en el empleo de llaves de distintos tamaños que relacionan entre si todas las tareas y operaciones.

Un diagrama de Warnier consta de:

- Estructuración de datos de salida y entrada, representando los registros y archivos lógicos.
- Representación del proceso o algoritmo.

La figura 3.3 muestra un Diagrama de Warnier que representa un programa que lee cinco veces tres números y los imprime ordenados ascendentemente indicando en cada ordenación si los números fueron introducidos ordenados o no.

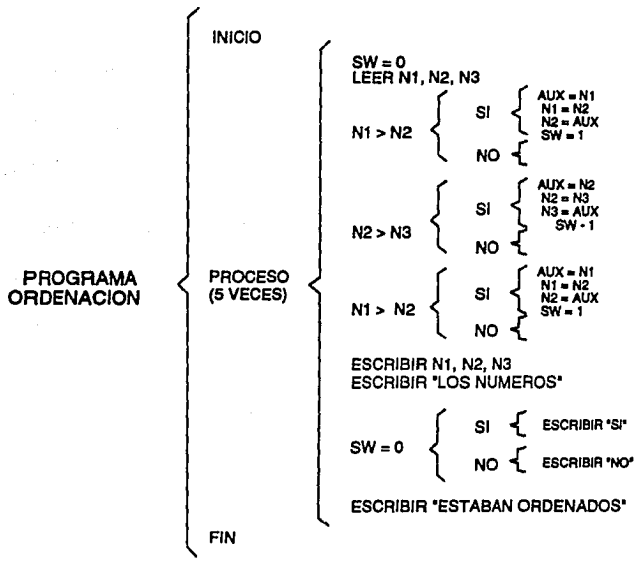


FIGURA 3.3

METODO DE JACKSON

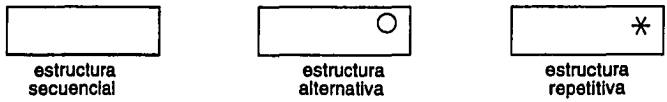
Se trata de un método de representación de programas en forma de árbol denominado diagrama arborescente de Jackson.

Un diagrama de Jackson consta de:

- Definición detallada de los datos de entrada y salida incluyendo los archivos lógicos utilizados.

- Representación del proceso o algoritmo.

La simbología utilizada se basa en el empleo de rectángulos horizontales que pueden presentar los siguientes aspectos:



La lectura del diagrama se hace recorriendo el árbol en preorden (RID), lo que supone realizar:

- Situarse en la raíz (R).
- Recorrer el subárbol izquierdo (I).
- Recorrer el subárbol derecho (D).

Cada subárbol se recorre igualmente en preorden hasta llegar a las hojas o nodos terminales del árbol.

La figura 3.4 es un diagrama de Jackson que representa un programa que lee cinco veces tres números y los imprime ordenados ascendentemente, indicando en cada ordenación si los números fueron ordenados o no al introducirse.

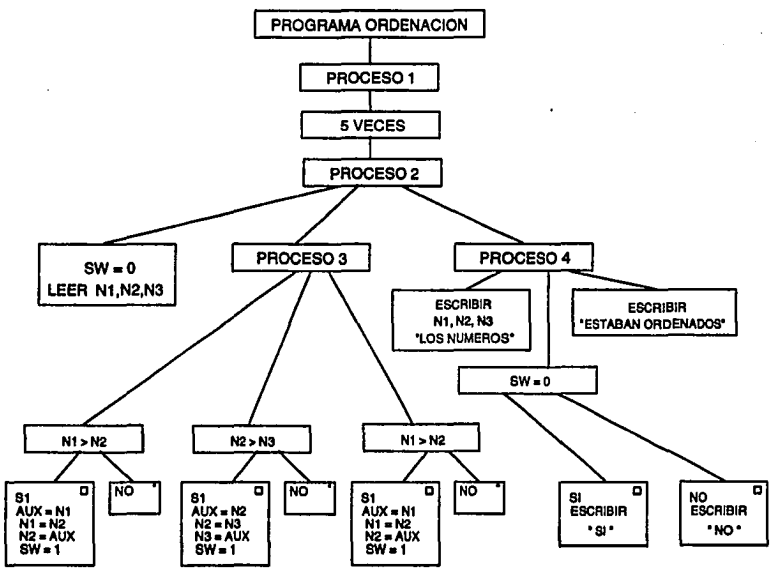


FIGURA 3.4

METODO DE BERTINI

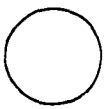
Al igual que Jackson, la representación de programas es en forma de árbol, denominada diagrama arborescente de Bertini.

Un diagrama de Bertini consta de:

- Definición detallada de los datos de entrada y salida, incluyendo los archivos lógicos utilizados.

- Representación del proceso o algoritmo.

La simbología usada se basa en el empleo de réctangulos horizontales y círculos.



La lectura del diagrama se hace recorriendo el árbol en preorden (RDI), lo que supone realizar:

- Situarse en la raíz (R).
- Recorrer el subárbol derecho (D).
- Recorrer el subárbol izquierdo (I).

Cada subárbol se recorre igualmente en orden inverso hasta llegar a las hojas o nodos terminales del árbol.

La figura 3.5 es un diagrama de Bertini que representa un programa que lee cinco veces tres números y los imprime ordenados ascendentemente, indicando en cada ordenación si los números fueron ordenados o no al ser introducidos.

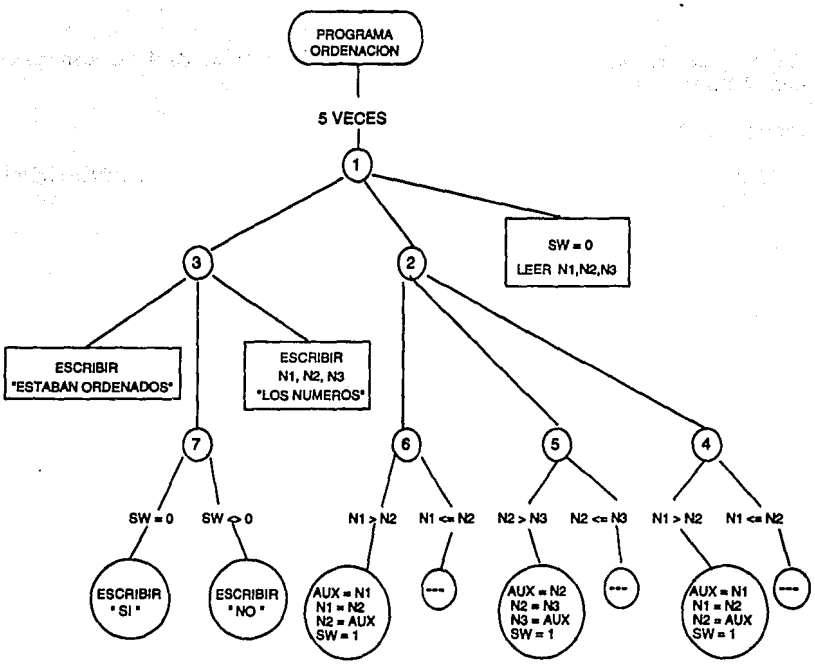


FIGURA 3.5

3.3 SEUDOCODIGO. ESTRUCTURAS DE CONTROL: SECUENCIA, SELECCION RAMIFICACION Y REPETICION

DEFINICION DE SEUDOCODIGO

Una manera diferente de expresar un algoritmo que difiere tanto de los diagramas de flujo como del código es con el seudocódigo. Este método utiliza instrucciones semejantes a las del idioma común, en lugar de los símbolos gráficos de los diagramas de flujo. En la figura siguiente se ilustra la representación en seudocódigo de las estructuras fundamentales de control.

Ejemplos:

Para sumar 2 números.

```
Inicio de los cálculos
Introducción del valor A
Introducción del valor B
Suma el valor de A y de B y el resultado será C
Imprimir el valor de C
Fin de cálculos
```

Algoritmo para sumar N números:

```
Escribir "cuántos números tiene la lista"
leer N
contador <--- 0
suma     <--- 0
* Comienzo Bucle
 Repetir lo siguiente mientras contador < N
        leer número
        sumar    <--- suma + número
        contador <--- contador + 1
 Fin repetir
* Fin de bucle
  Escribir "la suma de" N "números es:", suma
```

ESTRUCTURAS DE CONTROL: SECUENCIA, SELECCION, RAMIFICACION Y REPETICION

CONTROL

Un concepto de la programación estructurada señala que cualquier programa puede representarse mediante tres secuencias de control de bloques: la secuencia de procesamiento, la condicional IF/THEN/ELSE y la secuencia de bucles. La figura 3.6 muestra estas tres secuencias de control de bloques.

En la secuencia de procesamiento, se realiza una serie de procesamientos, en los que una tarea precede a la siguiente. La secuencia IF/THEN/ELSE especifica una operación condicional en la que el programa debe escoger dos caminos alternativos. Estos pueden conducir a una sola sentencia o a un módulo de instrucciones. El último bloque de control es la secuencia de bucles, pieza fundamental de muchas actividades de procesamiento y debe documentarse adecuadamente.

SECUENCIA. Son pasos simples que deben ejecutarse uno después del otro.

SELECCION. Se utiliza para determinar si un paso particular debe ejecutarse o no.

REPETICION. La operación de "reciclado" repite un conjunto de instrucciones y se le llama ciclo.

RAMIFICACION. Las estructuras múltiples se emplean cuando la condición puede tomar n valores enteros para realizar una de las n acciones.

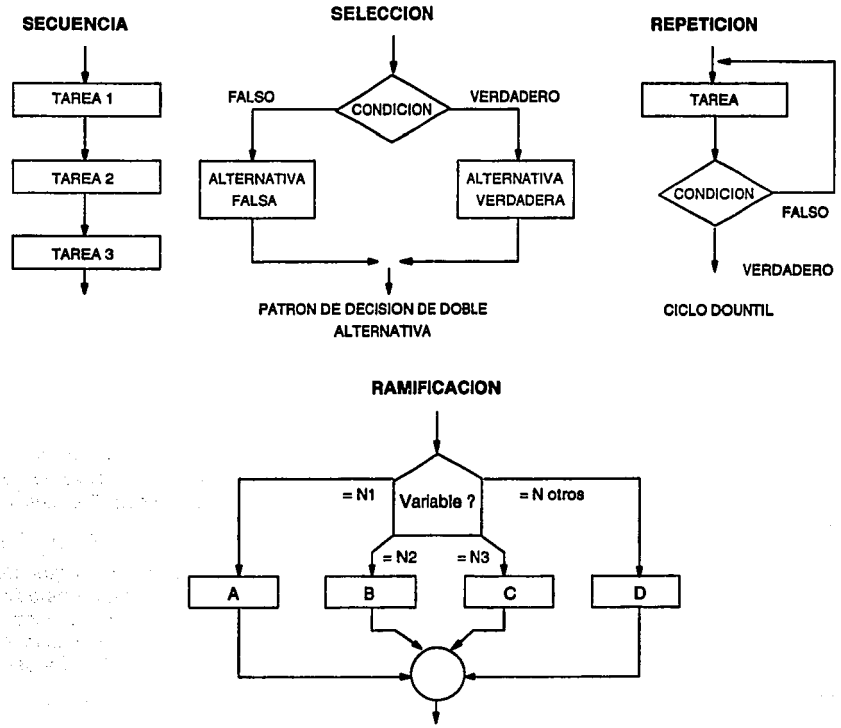


FIGURA 3.6

TEMA IV

LENGUAJES DE PROGRAMACION BASIC Y PASCAL

4.1 GENERALIDADES

Del mismo modo que los hombres se valen de lenguajes para comunicarse entre sí, también las computadoras hacen lo mismo para comunicarse con otros dispositivos (las impresoras por ejemplo), los operadores humanos e incluso con otras máquinas.

En la actualidad hay varios lenguajes de computadoras. Dos de ellos son el lenguaje BASIC y el PASCAL.

El BASIC fue creado a mediados del los años 60's por el Dr. John Kemeny del Dartmouth College, su uso se ha expandido para permitir el manejo de aplicaciones industriales, de gestión y científicas.

Originalmente sólo existía una versión del BASIC, pero ahora se cuenta con varias (BASIC, BASICA, GWBASIC, TURBO BASIC etc.), comercializadas por los fabricantes de computadoras. Estas nuevas versiones son bastante similares, incorporando cada fabricante instrucciones especiales seleccionadas para resaltar la eficiencia de su BASIC.

La palabra BASIC, es un acrónimo de "BEGINER'S ALL-PURPOSE SYMBOLIC INSTRUCTIONAL CODE" (Código Instruccional Simbólico Todo-Propósito para Principiantes). El BASIC posee su propia sintaxis y mientras se ejecutan los programas el intérprete BASIC exigirá que se cumplan las reglas de sintaxis y detectará errores. Aunque el BASIC no es un lenguaje completamente estructurado, conduce a la aplicación de muchos conceptos de programación estructurada.

El lenguaje de programación PASCAL, es un lenguaje de alto nivel y propósito general, desarrollado por el profesor suizo Niklaus Wirth como un idioma para enseñar la programación de un modo disciplinado y sistemático.

Una versión preeliminar del lenguaje apareció en 1968 y el primer compilador totalmente completo se dio a conocer a finales de 1970. Desde entonces muchos compiladores han sido construidos y están disponibles para diferentes máquinas. El nombre de PASCAL fue elegido en honor de Blaise Pascal por ser un brillante científico y matemático y por haber inventado la primer máquina de calcular mecánica en el mundo.

En cuanto a la estructura de un programa se puede decir que BASIC no cuenta con una estructura definida, es decir que su estructura es libre. Cada línea en el programa deberá empezar con un número (llamado número de renglón o etiqueta) y la última instrucción en el programa deberá ser la instrucción END, con la cual termina el programa.

Para incorporar un comentario dentro de un programa BASIC recurriremos a la instrucción REM, así cualquier línea que inicie con REM, no será ejecutada. Una representación generalizada de un programa BASIC sería:

```
10 REM COMENTARIO.....
20 INSTRUCCION
30 INSTRUCCION
-----------
-----------
-----------
100 INSTRUCCION
110 END
```

Por lo tanto, el lenguaje PASCAL en cambio sí cuenta con una estructura definida para la creación de un programa y es la siguiente:

```
PROGRAM <identificador> (lista de parámetros de archivos)

LABEL                Declaración de etiquetas
------
CONST                Declaración de constantes
------
TYPE                 Declaración de tipos
------
VAR                  Declaración de variables
------
PROCEDURE O FUNCTION  <identificador>  (lista de parámetros)
------
------
BEGIN
------
------
------
END
```

Las secciones de declaración (LABEL, CONST, TYPE y PROCEDURE y/o FUNCTION), no tienen que existir todas en un programa.

La instrucción PROGRAM especifica el nombre del programa y la de sus parámetros.

Los identificadores representan los objetos de un programa (constantes, variables, tipos de datos, procedimientos, funciones, utilidades, programas y campos de registro). Un identificador es una secuencia de caracteres válidos.

Los comentarios, dentro de un programa PASCAL, los integraremos con el siguiente formato: (* comentario ... *). Otra observación importante dentro de un programa Pascal es la puntuación; así un punto (.)después del último END en un programa se usa para marcar el final de todo el programa; un punto y coma (;) se usa para separar dos instrucciones

consecutivas dentro del programa.

Es conveniente que los programadores principiantes dominen los conceptos de diseño estructurado incorporados en muchos lenguajes populares actualmente. Los conceptos ordenados también proporcionan un medio de autodiciplinarse y ayudan a crear programas que sean lógicos y fáciles de depurar.

El diseño estructurado se ha mejorado y refinado a través del tiempo, y se puede decir que los pasos principales en este diseño estructurado son:

1.- Descomponer el problema dando lugar a una estructura modular, de manera que cada módulo de actividades importantes se trate individualmente.

2.- Minimizar el uso de sentencias de bifurcación incondicional (GO TO).

3.- Establecer una estructura jerárquica para los módulos del programa.

La aplicación de los principios del diseño de programación estructurada al desarrollo de software se denomina programación estructurada. Y en el presente trabajo aplicaremos estos principios para el aprendizaje del Lenguaje de Programación BASIC y PASCAL.

4.2 ELEMENTOS DEL LENGUAJE BASIC y PASCAL

CONSTANTES

Las constantes son valores estáticos que el programa utiliza durante su ejecución. Hay dos tipos de constantes:

a).- De cadena

b).- Numéricas

Una constante de cadena es una secuencia de 0 a 255 caracteres alfanuméricos encerrados entre comillas. Los siguientes son ejemplos de estas constantes:

```
"hola"      "25.00"      "88"      "E.N.E.P."
```

Las constantes numéricas pueden ser positivas o negativas Las siguientes son ejemplos de constantes numéricas:

```
- 1000        5        0.25      - 23012.24
```

Existen tres tipos de constantes numéricas: enteras, punto fijo y punto flotante.

Constante entera.- Son números enteros sin punto decimal.

Constante de punto fijo.- Son números reales positivos o negativos que contienen puntos decimales.

Constantes de punto flotante.- Son números positivos o negativos representados en forma exponencial. Una constante de punto flotante consiste en un entero con signo opcional o de punto fijo, seguido por la letra E y un entero con signo opcional (el exponente). Por ejemplo:

```
235988E-7   =   0.0000235988

2359E6      =   2359000000
```

En BASIC las constantes pueden ser declaradas en cualquier parte del programa antes de ser utilizadas, si no se declaran asumen un valor de cero. Ejemplo:

```
10 REM EJEMPLO01
20 PI        = 3.141692
30 CARACTER = 'HOLA'
40 ANCHO     = 8
50 LARGO     = 8
60 ALTURA    = 8
.
.
.
```

En cambio en PASCAL las constantes se tienen que manifestar en la sección de declaración del programa y mediante la palabra reservada CONST. Ejemplo:

```
(* EJEMPLO01 *)
PROGRAM EJEMPLO01 (DATA, OUTPUT)
CONST
PI          = 3.141692
CARACTER    = `HOLA`
ANCHO , LARGO , ALTURA = 8
```

VARIABLES Y DECLARACIONES

Las variables son nombres utilizados que se han seleccionado para representar los valores empleados en un programa. El valor de una variable puede ser asignado específicamente o resultar de cálculos en su programa.

Las variables representan tanto valores numéricos como cadenas.

Una variable es, en realidad, una posición de memoria con nombre. El nombre de la posición (un identificador) se llama nombre de la variable; el valor almacenado en la posición se denomina valor de la variable.

En BASIC los caracteres de declaración de tipo indican que representa una variable. Se reconocen los siguientes caracteres de declaración de tipo:

Caracter	Tipo de Variable
$	Variable de cadena
%	Variable entera
!	Variable de precisión sencilla
#	Variable de precisión doble

Ejemplos:

```
N$    =  "ADRIAN"
ENT%  =  8
VA1   =  48.3
VA2   =  573487583484354543.231
```

El tipo predeterminado para un nombre de variable numérica es precisión sencilla.

Existen variables de tipo arreglo, donde un arreglo es un grupo o tabla de valores referenciados con el mismo nombre de variable. Cada elemento dentro de un arreglo se relaciona como una variable arreglo; es decir, un entero o una expresión entera indexada. El índice se encierra entre paréntesis. Un nombre de variable arreglo tiene tantos índices como dimensiones contenga el arreglo. Ejemplo:

```
V(10)       referencia un valor en un arreglo uni-dimensional.
V(1,4)      referencia un valor en un arreglo bi-dimensional.
```

En el lenguaje PASCAL todas la variables de un programa deben ser declaradas antes de ser utilizadas. Las variables se deben manifestar en la sección de declaraciones del programa, y mediante la palabra reservada VAR. En esta declaración se deben incluir todos las variables usadas en un programa y los tipos de datos asociados a ellas.

```
VAR
VARIABLE1   :   TIPO
VARIABLE2   :   TIPO
.
.
.
VARIABLEN   :   TIPO
```

VARIABLEN : TIPO

Donde TIPO define los valores que una variable puede adquirir y las operaciones que en ella se pueden realizar. Y podremos tener los siguientes tipos:

```
INTEGER     .-   Para variables enteras.
REAL        .-   Para valores reales.
CHAR        .-   Para uso de caracteres (ASCII).
BOOLEAN     .-   Para valores booleanos (true y false).
STRING      .-   Para datos de cadena.
```

Ejemplo:

```
VAR
DATOS    : INTEGER ;
RAIZ     : REAL    ;
APELLIDO : STRING  ;
LETRA    : CHAR    ;
```

Además de estos tipos de datos simples existen otros datos simples conocidos como tipos subrango y enumerados. Estos tipos de datos sólo requieren elementos simples de datos y no una organización específica. Sin embargo, PASCAL incluye tipos de datos que requieren una organización de datos denominados estructuras de datos (conjuntos, arrays y registros). Por el momento sólo nos ocuparemos de los tipos de datos enumerados y subrango, también conocidos como datos definidos por el usuario. Un tipo enumerado se declara mediante la enumeración o listado entre paréntesis de sus valores que también se llaman sus constantes; el formato de un tipo enumerado es:

```
TYPE
NOMBRE  =  ( constante 1, constante2, ... constanteN )
```

Ejemplo:

```
TYPE
VEHICULOS  = ( MOTO, AUTOBUS, TREN, AVION )  ;
FRUTAS     = ( MANZANAS, PERAS, UVAS )       ;
DIAS       = ( LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO,
               DOMINGO )    ;
```

De igual forma, las variables pueden ser de tipo enumerado. Así, por ejemplo, se pueden declarar:

```
VAR
CLASE  = VEHICULOS
POSTRE = FRUTAS
```

Los tipos de datos SUBRANGO se definen especificando dos constantes de tipo ordinal, que actúan como límite inferior y superior del conjunto de datos de ese tipo. Su formato es el siguiente:

```
TYPE
NOMBRE = Límite inferior ... Límite superior
```

Ejemplo:

```
TYPE
INTERVALOS   =  -100 .. 100   ;
LETRASMIN    =  'a'.. 'z'     ;
```

Se pueden crear variables cuyos valores se restrinjan a un subrango. Ejemplo:

```
TYPE
INTERVALO     =  -100 .. 100   ;
LETRASMIN     =  'a'.. 'z'     ;
VAR
REDUCIDO       : INTERVALO     ;
X              : REAL          ;
```

OPERADORES

Los operadores realizan operaciones matemáticas o lógicas sobre los valores. Los operadores provistos por BASIC están divididos en tres categorías:

1.- Aritméticos
2.- Relacionales
3.- Lógicos (booleanos)

Operadores aritméticos.- Los siguientes son los operadores aritméticos.

Operador	Operación	Ejemplo
^	Exponenciación	A ^ B (en BASIC)
*	Multiplicación	A * B
/	División	A / B
+	Suma	A + B
-	Resta	A - B

El orden de procedencia es el siguiente:

^	Exponenciación
* /	Multiplicación y división
+ -	Adición, sustracción y negación

En las operaciones entre paréntesis son realizadas primero. Dentro del paréntesis se mantiene el orden usual de prioridades.

Los siguientes son ejemplos de expresiones algebráicas y sus equivalentes codificados.

Expresión algebráica	Expresión codificada
$\frac{A - B}{C}$	(A - B) / C
$\frac{A\,B}{C}$	A * B / C
$\frac{A + B}{C}$	(A + B) / C

Operadores relacionales.- Estos operadores le permiten comparar dos valores. El resultado de la comparación es verdadero o falso. Este resultado puede ser entonces utilizado para tomar una decisión referente al flujo del programa.

Operador	Relación	Expresión
=	Igualdad	A = B
<>	Desigualdad	A <> B
<	Menor que	A < B
>	Mayor que	A > B
<=	Menor o igual que	A <= B
>=	Mayor o igual que	A >= B

Cuando los operadores aritméticos y relacionales se combinan en una expresión, el aritmético se ejecuta primero.

Para comparar cadenas se utilizan los mismos operadores relacionales.

Operadores lógicos.- Los operadores lógicos realizan pruebas a relaciones múltiples. Este operador da un resultado en bit, que es verdadero (diferente a cero) o falso (igual a cero). En una expresión las operaciones lógicas son efectuadas después de las operaciones matemáticas y relacionales. El resultado de una operación lógica se determina de la manera como se muestra en la siguiente tabla:

Operación	Valor	Valor	Resultado
NOT	X		NOT X
V			F
F			V
AND	X	Y	X AND Y
	V	V	V
	V	F	F
	F	V	F
	F	F	F
OR	X	Y	X OR Y
	V	V	V
	V	F	V
	F	V	V
	F	F	F

INSTRUCCIONES Y DECLARACIONES

Las instrucciones describen las acciones algorítmicas que pueden ser ejecutadas. En general, las instrucciones se clasifican en ejecutables (especifican operaciones de cálculos aritméticos y entradas/salidas de datos) y no ejecutables (no realizan acciones concretas, ayudan a la legibilidad del programa, pero no afectan en la ejecución del mismo).

Cada instrucción ejecutable se traduce (por el compilador) en una o más instrucciones de lenguaje de máquina, que se copian en el archivo objeto y posteriormente se ejecutan.

Las declaraciones por otra parte, describen el significado y el propósito de cada identificador definido por el usuario, no se traducen a instrucciones en lenguaje máquina y no aparecen en el archivo objeto.

Las instrucciones se clasifican, atendiendo a su tipo y número en:

- Simples
- Estructuradas

Instrucciones Simples.- Son instrucciones que no contienen otra instrucción. Se clasifican en: asignación y goto.

Instrucciones Estructuradas.- Son instrucciones compuestas de otras instrucciones que se ejecutan en secuencia (compuestas), condicionalmente (condicionales o selectivas), o repetitivamente (repetitivas).

En BASIC las instrucciones van precedidas por un número de línea las cuales son un medio de identificación y definen el orden en que se ejecutan las instrucciones. El número de línea se sitúa al principio de ésta y tiene un formato de número entero.

En PASCAL las instrucciones aparecen en el cuerpo del programa a continuación de la palabra reservada BEGIN.

LA INSTRUCCION DE ASIGNACION

La instrucción de asignación se utiliza para asignar (almacenar) valores o variables.

La asignación es una operación que sitúa un valor determinado en una posición de memoria.

En BASIC la instrucción LET señala a una variable el valor de una expresión. Su formato es el siguiente:

```
LET VARIABLE = EXPRESION
```

La palabra LET es opcional; es decir, el signo igual es suficiente cuando se indica a una expresión a un nombre de variable. La instrucción LET es poco utilizada, ya que las versiones

recientes de BASIC pueden prescindir de ella. Ejemplo:

```
10 LET  D  = 8
20 LET  A  = 6.5
30 LET  X  = A + D
40 Y       = 8 / 14
```

En PASCAL el operador "=" se sustituye por el simbolo := , que se denomina operador de asignación. Su formato es:

VARIABLE := EXPRESION

Ejemplo:

```
D   :=  8
A   :=  6.5
X   :=  A + D
Y   :=  8 / 14
```

Ahora ya tenemos los elementos suficientes para elaborar un programa completo tanto en BASIC como en PASCAL.

Ejemplo:

En BASIC.

```
10  REM  AUTOR    : G.JARDON Y J.L.ARCE
20  REM  FECHA    : FEB/93
30  REM  FUNCION : CALCULAR EL AREA DE UN RECTANGULO
40  REM  EJEMPLO 03
50  REM
60  ANCHO   = 4
70  LARGO   = 5
80  AREA    = ANCHO  *  LARGO
90  END
```

En PASCAL.

```
(*   AUTOR    :  G. JARDON  Y  J.L ARCE  *)
(*   FECHA    :  FEB/93                  *)
(*   FUNCION :  CALCULAR EL AREA DE UN TRIANGULO *)
(*                                                *)
PROGRAM EJEMPLO 03 ;
VAR
ANCHO  : INTEGER ;
```

```
LARGO   : INTEGER ;
AREA    : INTEGER ;
(* *)
BEGIN
ANCHO   := 4 ;
LARGO   := 5 ;
AREA    := ANCHO   *   LARGO
END.
```

4.3 INSTRUCCIONES DE ASIGNACION Y FUNCIONES INTRINSECAS ELEMENTALES

Las funciones son uno de los elementos básicos en programación. Una función es un subprograma que recibe como argumento o parámetro datos de tipo numérico, o no numérico, y devuelve un resultado.

Las funciones son un medio muy adecuado para resolver operaciones complejas matemáticas o de otras disciplinas afines, así como tratar diferentes tipos de datos definidos y realizar con ellos procesos complejos. Existen dos grandes grupos de funciones:

- Funciones predefinidas internas o standar
- Funciones definidas por el usuario

Por el momento nos concentraremos en las primeras, describiendo las más comunes.

Las funciones tanto en BASIC como en PASCAL son similares por lo que sólo, resaltaremos cuando exista alguna diferencia en su uso en uno u otro lenguaje.

Funciones trigonométricas

```
SIN(X)      Calcula el seno trigonométrico de x

COS(X)      Calcula el coseno trigonométrico de x

TAN(X)      Calcula la tangente trigonométrica de x (Sólo en BASIC)

ARCTAN(X)   Calcula el arcotangente ( Sólo en PASCAL )
```

Aquí el ángulo X se expresa como medida de radianes. En este sistema de medición, 360 grados equivalen a PI radianes por 2; es decir, un grado es igual a 0.017453 radianes y un radian es igual a 57.29578 grados.

Si queremos calcular las funciones trigonométricas con el ángulo X expresado en grados, usaremos las funciones siguientes:

```
SIN(0.01745 * X)

COS(0.01745 * X)

TAN(0.01745 * X)
```

Funciones trigonométricas exponenciales.

```
EXP(X) Calcula la función exponencial y elevada a la x (Sólo en BASIC)
LOG(X) Calcula el logaritmo natural (base e) de x
```

Potencias

El BASIC tiene una función de raíz cuadrada, denotada por SQR(X). Ejemplo:

```
10  Y = SQR(2.0)
```

Entonces Y será igual a 1.414214

Otro método alterno para obtener la raíz de X número es elevarlo a la potencia fraccionaria deseada. Ejemplo:

```
10 Y = X ^ (1/2)
```

Aunque la función SQR(X) opera con mayor velocidad, es preferible usarla en estos casos. El método alterno es más flexible; por ejemplo, podemos extraer la raíz cúbica de X como:

```
10 Y = X ^ (1/3)
```

O elevar X a la potencia 5.389

```
10 Y = X ^ 5.389
```

En PASCAL la función cuadrada SQR(x) equivale a elevar al x al cuadrado o de otro modo X * X.

La función SQRT(x) es la raíz cuadrada del argumento.

Parte entera del argumento.

En BASIC :

El mayor entero que es menor o igual a x se denota por INT(X). Por ejemplo, el que es menor o igual que 5.46789 es 5, por lo tanto:

```
INT(5.46789)        Devuelve el valor 5
```

En PASCAL :

```
TRUNC (x)

TRUNC ( 5.46789 )    Devuelve el valor 5
```

Devolución de la parte entera de un decimal.

En BASIC :

De manera análoga el mayor entero menor o igual que -3.4 es -4. En consecuencia:

```
INT(-3.4)            Devuelve el valor  - 4
```

Igual en PASCAL

Nota: En el caso de números positivos, la función INT(X) suprime la parte decimal. Pero en el caso de números negativos, no hace exactamente lo mismo. Sin embargo, para suprimir la parte decimal de un número positivo o negativo, aplicaremos la función.

En BASIC :

```
FIX (x)

FIX(5.46789)         Devuelve el valor 5

FIX(-3.4)            Devuelve el valor  - 3
```

En PASCAL :

```
ROUND(x)

ROUND(4.44)          Devuelve el valor 4
```

Valor absoluto.

El valor absoluto de x viene denotado por ABS(X). Ejemplo:

```
ABS(9.23)            Devuelve el valor 9.23

ABS(-4.1)            Devuelve el valor 4.1
```

Igual en PASCAL

Mostraremos un ejemplo del uso de la función SQR(x) de PASCAL y del operador (^) de exponenciación de BASIC.

En BASIC :

```
10 REM  AUTOR      : G. JARDON  Y  J. LUIS ARCE
20 REM  FECHA      : FEB/93
30 REM  FUNCION    : CALCULO DEL AREA DE UN CIRCULO
40 REM               EJEMPLO 04
50 PI      = 3.1416
60 RADIO   = 88
70 AREA    = PI * ( RADIO ^ 2 )
80 END
```

En PASCAL :

```
(*  AUTOR      : G. JARDON  Y  J. LUIS ARCE  *)
(*  FECHA      : FEB/93                      *)
(*  FUNCION    : CALCULO DEL AREA DE UN CIRCULO *)
(*   *)
PROGRAM EJEMPLO 04 ;
CONST
PI     =  3.1416 ;
RADIO =  88      ;
VAR
AREA   : REAL ;
(*  *)
BEGIN
AREA = PI * SQR(RADIO)
END.
```

4.4 INSTRUCCIONES DE ENTRADA Y SALIDA

Los datos se pueden almacenar en memoria de tres formas diferentes: asociados con constantes, asignadas a una variable con una instrucción de asignación o una instrucción de lectura. Ya hemos examinado las dos primeras, el tercer método, la instrucción de lectura, es el más indicado, si se desea manipular datos cada vez que se ejecute el programa. Además, esta lectura permite asignar valores desde dispositivos hasta archivos externos (por ejemplo un teclado o una unidad de disco) en memoria se denomina operación de ENTRADA o LECTURA.

A medida que se realizan cálculos en un programa se necesitan visualizar los resultados. Esta operación se conoce como operación de SALIDA o ESCRITURA.

LA ESCRITURA DE RESULTADOS (SALIDA)

Los programas para ser útiles deben proporcionar información de salida (resultados). Esta salida toma información de la memoria y la sitúa (almacena) en: pantalla, en un dispositivo de almacenamiento (disco duro o flexible), o en un puerto de E/S (puerto serie para comunicaciones o impresoras, normalmente paralelos).

EN BASIC :

Instrucción PRINT

La instrucción PRINT puede utilizarse para:

- Imprimir los valores de las variables
- Imprimir resultados de alguna expresión
- Imprimir mensajes
- Saltar renglones

Su sintaxis es la siguiente:

```
PRINT  [ lista de expresiones ] [,] [;]
```

BASIC divide la línea en una zona de impresión de 14 espacios, la posición de cada elemento impreso se determina por la puntuación utilizada para separar los elementos de la lista.

Separador	Posición de impresión
,	Comienza en la zona siguiente
;	Inmediatamente después del último valor

EN PASCAL :

En PASCAL las instrucciones de escritura son:

```
WRITELN ( ítem , ítem ... ) ;

WRITE ( ítem , ítem ... ) ;
```

Donde:

ítem es el objeto que se desea visualizar; un valor literal (entero o real, un caracter, una cadena o un valor lógico), una constante con nombre, una variable.

La instrucción WRITLN se usa para hacer que la computadora imprima algún material. Al ejecutar una instrucción WRITLN, la computadora imprimirá literalmente todo lo incluido entre apóstrofos e imprimirá los valores de cualesquiera variables o expresiones.

Se usan comas para separar las elementos a imprimir. Ejemplo:

```
WRITELN ( 'El área del rectángulo es ', AREA ) ;
```

Hará que la computadora imprima literalmente el mensaje:

```
El área del rectángulo es
```

Si el valor de la variable área fuese 42 se obtendría entonces:

```
El área del rectángulo es 42
```

Cuando la computadora ejecuta una instrucción WRITELN, imprime la salida especificada y después baja el cursor al inicio de la siguiente línea.

La instrucción WRITELN se debe usar cuando se desea que la salida de cada instrucción aparezca sólo en una línea.

En cambio la instrucción WRITE se debe usar cuando se desea que la salida de varias aparezcan en la misma línea. Cuando la computadora ejecuta una instrucción WRITE, el cursor no baja a la línea siguiente; en vez de ello, permanece donde terminó la impresión y espera allí la siguiente instrucción WRITE. Ejemplo:

EN BASIC :

```
10 REM  AUTOR    : GRACIELA JARDON  Y  J. LUIS ARCE
20 REM  FECHA    : FEB/93
30 REM  FUNCION : CONVERSION DE YARDAS A PIES
40 REM             EJEMPLO 05
50 YARDAS = 8
40 PIES    = YARDAS * 3
50 PRINT YARDAS ; " YARDAS SON "
60 PRINT PIES ; " PIES"
70 END
```

El resultado sería:

```
8 YARDAS SON
24 PIES
```

Ejemplo:

```
10 REM  AUTOR    : GRACIELA JARDON  Y  J. LUIS ARCE
20 REM  FECHA    : FEB/93
30 REM  FUNCION : CONVERSION DE YARDAS A PIES
40 REM             EJEMPLO 06
50 YARDAS = 8
60 PIES    = YARDAS * 3
70 PRINT YARDAS ; " YARDAS SON " ; PIES ; " PIES"
80 END
```

El resultado sería:

```
8 yardas son 24 pies
```

EN PASCAL :

```
(* AUTOR     :  GRACIELA JARDON  Y  J. LUIS ARCE    *)
(* FECHA     :  FEB/93                              *)
(* FUNCION   :  CONVERSION DE YARDAS A PIES         *)
PROGRAM EJEMPLO 07 ;
CONST
YARDAS = 8 ;
VAR
PIES  : REAL ;
(*                                                  *)
BEGIN
PIES = YARDAS * 3  ;
WRITELN ( YARDAS, ' YARDAS SON ') ;
WRITELN ( PIES , 'PIES')
END.
```

El resultado sería:

```
8 YARDAS SON
24 PIES
```

Ejemplo:

```
(* AUTOR     :  GRACIELA JARDON  Y  J. LUIS ARCE    *)
(* FECHA     :  FEB/93                              *)
(* FUNCION   :  CONVERSION DE YARDAS A PIES         *)
PROGRAM EJEMPLO08 ;
CONST
YARDAS = 8  ;
VAR
PIES  : REAL ;
(*                                              *)
BEGIN
PIES = YARDAS * 3  ;
WRITE ( YARDAS , ' YARDAS SON ')  ;
WRITE ( PIES , ' PIES ')
END.
```

El resultado sería:

```
8 YARDAS SON 24 PIES
```

LA ENTRADA DE DATOS (LECTURA)

La operación de lectura permite proporcionar datos durante la ejecución de un programa. Los datos que se pueden leer son: enteros, reales, caracteres, y cadenas. No se puede leer un boolean o un elemento de tipo enumerado.

EN BASIC :

READ Y DATA

Las instrucciones READ y DATA se usarán simultáneamente. El propósito de la instrucción READ es leer valores de una instrucción DATA y asignarlos a variables. Su sintaxis es la siguiente:

```
READ variable1, variable2, ...
```

Las variables de la instrucción READ pueden ser numéricas o de cadena y, los valores leídos deben estar de acuerdo con los tipos de variables especificados. Una sola instrución READ puede lograr acceso a una o más instrucciones DATA. Para volver a leer instrucciones DATA desde el principio, utilice la instrucción RESTORE.

El propósito de la instrucción DATA es almacenar constantes numéricas y de cadena a las que se logra acceso mediante las instrucciones READ de un programa. Su sintaxis es:

```
DATA constante1, constante2, ...
```

Ejemplo:

```
10  REM  AUTOR    :  GRACIELA JARDON A.  Y  JOSE LUIS ARCE G.
20  REM  FECHA    :  FEB/93
30  REM  FUNCION :  MOSTRAR EL USO DE LAS INSTRUCCIONES READ Y
40  REM               DATA
50  REM
60  READ N$, H, R
70  G = H * R
80  PRINT N$, G
90  DATA "JUAN PEREZ", 35, 2
100 END
```

El resultado será:

```
JUAN PEREZ  70
```

LA INSTRUCCION INPUT

Es de gran utilidad hacer que la computadora le pida información durante la ejecución de un programa. Y ello se consigue por medio de la instrución INPUT. Su sintaxis es:

```
INPUT [;] [cadena de señal] lista de variables
```

Donde:

- Cadena de señal es una solicitud para que se suministren datos durante la ejecución del programa.

- Lista de variables contiene la(s) variable(s) que almacena los datos suministrados. Cada elemento de datos en la cadena de señal debe estar encerrada entre comillas, seguida por un punto y coma o coma y el nombre de la variable a la cual será asignado.

Si se especifica más de una variable, los elementos de datos deben estar separados por comas. Los valores de las variables incluidas pueden ser numéricos o de cadena, incluyendo variables de índice. Ejemplo:

EN PASCAL :

En PASCAL las instrucciones de lectura son READ y READLN, y su formato es:

```
READ (VAR1 , VAR2 ...) ;

READLN (VAR1, VAR2 ...) ;
```

Las instrucciones READLN y READ esperan hasta que se pulsa la flecha intro (return o enter) antes de asignar un valor a la variable.

Una instrucción READLN, como una instrucción de asignación, se usa para asignar un valor a una variable. Sin embargo, a diferencia de una instrucción de asignación, que recibe el valor que ha de calcularse del programa mismo, una instrucción READLN permite que se introduzca un valor desde una fuente externa.

La diferencia entre READ y READLN es semejante a la que se da entre WRITE y WRITELN. Después de ejecutar una instrucción READLN el cursor baja a la línea siguiente de la pantalla. En seguida de una instrucción READ; sin embargo, el cursor permanece en la misma línea.

Ejemplos:

EN BASIC :

```
10  REM  AUTOR    : GRACIELA JARDON  Y  J. LUIS ARCE
20  REM  FECHA    : FEB/93
30  REM  FUNCION  : CONVERTIR PIES DE ENTRADA A PULGADAS
40  REM             EJEMPLO 09
50  INPUT "INGRESE NUMERO DE PIES " ; PIES
60  PULGADAS = PIES * 12
70  PRINT "LAS PULGADAS EQUIVALENTES SON: "; PULGADAS
80  END
```

Si ingresáramos a la pregunta cinco ; el resultado sería:

```
INGRESE NUMERO DE PIES 5
LAS PULGADAS EQUIVALENTES SON : 60
```

EN PASCAL :

```
(*  AUTOR    : GRACIELA JARDON  Y  J.LUIS ARCE    *)
(*  FECHA    : FEB/93                             *)
(*  FUNCION  : CONVERTIR PIES DE ENTRADA A PULGADAS *)
PROGRAM EJEMPLO 10  ;
VAR
PIES      : REAL  ;
PULGADAS  : REAL  ;
(*                                                *)
BEGIN
WRITE ( 'INGRESE NUMERO DE PIES ')  ;
READLN (PIES) ;
PULGADAS := PIES * 12  ;
WRITELN ( 'LAS PULGADAS EQUIVALENTES SON : ', PULGADAS)
END.
```

Si ingresáramos cinco a la pregunta; el resultado sería:

```
INGRESE NUMERO DE PIES 5
LAS PULGADAS EQUIVALENTES SON : 60
```

IMPRESION EN PAPEL

EN BASIC :

En BASIC se utiliza la instrucción LPRINT que es lo mismo que PRINT excepto que las salidas van a la impresora.

EN PASCAL :

En PASCAL se emplean las instrucciones: WRITE (lst ...) o bien WRITELN (lst ..) que mandarán la salida a la impresora.

4.5 REPRESENTACION DE LAS ESTRUCTURAS DE CONTROL

El concepto de flujo de control a través de un programa se refiere al orden en que se ejecutan las acciones individuales de un programa.

Aunque el flujo normal de un programa es lineal, existen métodos que permiten salir del flujo lineal; como son las instrucciones de transferencia incondicional (GOTO), a través del uso de las estructuras de control selectivas (IF_THEN_ELSE , ON-GOTO , CASE) y como sabemos las computadoras están preparadas para realizar tareas repetitivas que se logran mediante las estructuras de control repetitivas (WHILE , FOR REPEAT).

A continuación describiremos cada una de estas instrucciones:

LA INSTRUCCION GOTO

Su propósito es transferir incondicionalmente el control de la secuencia del programa a un número de línea específico.

EN BASIC :

Su formato general es :

```
GOTO número de renglón
```

En donde el número de renglón pertenece al número de renglón de alguna otra instrucción.

Ejemplo :

```
100 CANTIDAD = 8
120 PRECIO = 20000
130 COSTO = CANTIDAD * COSTO
140 PRINT  "EL COSTO ES " ; COSTO
150 GO TO 500
.
.
.
500 PRINT "RENGLON A QUE SE TRANSFIRIO EL CONTROL "
```

EN PASCAL :

Para utilizar esta instrucción en PASCAL se requiere que se declaren etiquetas y la forma en que se declaran es:

```
LABEL
(ETIQUETA 1 ) , (ETIQUETA 2 ) , ..., (ETIQUETA N ) ;
```

Las etiquetas están constituidas por un número no mayor de cuatro dígitos. Se usan para especificar niveles de declaraciones a las cuales uno quiere regresar o pasar en un momento dado por medio de la instrucción GOTO .

Ejemplo :

```
PROGRAM EJEMPLO GOTO ;
LABEL
100 , 200 ;
CONTS
CANTIDAD = 8
PRECIO   = 20000
VAR  COSTO     : REAL
BEGIN
.
.
.
100 COSTO := CANTIDAD * PRECIO
WRITELN ( 'EL COSTO ES ', COSTO ) ;
GOTO 200
.
.
.
200 WRITELN('ETIQUETA A QUE SE TRANSFIRE EL CONTROL');
.
.
.
END.
```

Esta instrucción puede ocasionar problemas en el control de flujo de nuestro programa, y en una programación estructurada no es recomendable utilizarla. Sólo se deberá emplear en casos realmente necesarios para salir de un cruce de caminos a tomar.

LA INSTRUCCION IF..THEN

La intrucción GO TO que representamos anteriormente proporciona un medio para transferir, incondicionalmente, el control a otra parte del programa. En algunos casos, se requiere que la transferencia sea condicional; es decir, deberá ejecutarse en algunos casos pero en otros no. Su diagrama representativo se muestra en la figura 4.1.

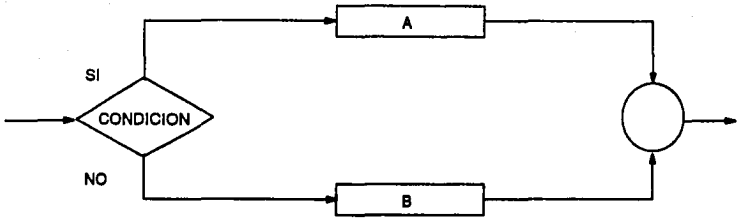


FIGURA 4.1

La estructura o sentencia IF funciona de la siguiente forma:

1.- Se evalúa la expresión lógica.

2.- Si la expresión toma el valor true (verdadero), se ejecutará la sentencia A y el control pasará a la sentencia inmediata siguiente a la IF_THEN_ELSE (no se ejecuta la sentencia B).

3.- Si la expresión toma el valor false (falso), entonces sólo se ejecutará la sentencia B y el control pasa de nuevo a la siguiente instrucción del programa.

En muchos casos se desea que una determinada acción sólo se ejecute si una cierta condición es verdadera y no realizar ninguna acción si la condición es false; en este caso se omite del formato la cláusula ELSE.

```
IF condición
THEN
sentencia
```

Los símbolos de relación que se utilizan dentro de una expresión son:

```
=    Igual a

<>   Diferente de

<    Menor que

>    Mayor que

<=   Menor o igual que

>=   Mayor o igual que
```

EN BASIC :

Su sintaxis es:

```
IF expresión [,] THEN instrucción [ELSE instrucción ]

IF expresión [,] GO TO No. de línea [[,] ELSE instrucción]]
```

Si el resultado de la expresión es diferente de cero (verdadero lógico), la instrucción THEN se ejecuta o el número de línea después de GO TO se ejecuta.

Si el resultado de la expresión es cero (falso), se ignora el THEN o el número de línea de GO TO, se ejecuta el número de línea de ELSE, si esta presente.

Ejemplo:

```
10 REM  AUTOR     : GRACIELA JARDON  Y  JOSE LUIS ARCE
20 REM  FECHA     : FEB/93
30 REM  FUNCION   : DETERMINAR SI UN NUMERO DE ENTRADA ES
40 REM              POSITIVO O NEGATIVO
50 REM              EJEMPLO 11
60 INPUT "NUMERO"; X
70 IF X > 0 THEN 30 ELSE 80
80 PRINT X ; "ES POSITIVO"
90 PRINT "EN ESTA OPCION" ; X ;"ES POSITIVO"
100 GO TO 90
110 PRINT X ; "ES NEGATIVO"
120 PRINT " O BIEN ES CERO "
130 END
```

EN PASCAL :

En numerosas ocasiones, en lugar de realizar sólo una acción cuando se cumpla o no se cumpla una determinada condición, se deben realizar diferentes acciones. Esto se efectúa mediante instrucciones compuestas.

Una instrucción compuesta es un conjunto de instrucciones separadas por puntos y comas y encerradas entre las palabras BEGIN y END.

Ejemplo:

```
(*  AUTOR     : GRACIELA JARDON  Y  JOSE LUIS ARCE      *)
(*  FECHA     : FEB/93                                  *)
(*  FUNCION   : DETERMINAR SI UN NUMERO LEIDO ES        *)
(*              POSITIVO O NEGATIVO                     *)
PROGRAMA EJEMPLO 12 ;
VAR
X  :  REAL ;
BEGIN
READ ('TECLEE UN NUMERO', X ) ;
IF X > 0
THEN
BEGIN
WRITELN (X, 'ES POSITIVO') ;
WRITELN ('EN ESTA OPCION', X , 'ES POSITIVO)
END
ELSE
BEGIN
WRITELN ( X , 'ES NEGATIVO') ;
WRITELN ( 'O BIEN ES CERO ')
END.
```

Para colocar los puntos y comas, se considera a la instrucción IF-THEN-ELSE como una sola instrucción.

Ejemplo:

Unas camisas se venden a 50 nuevos pesos cada una, si se compran más de tres, y a 60 nuevos pesos si se compran menos de tres. Escribir un programa que lea un número de entrada de camisas compradas e imprima el costo total.

EN BASIC :

```
10 REM  AUTOR     : GRACIELA JARDON  Y JOSE LUIS ARCE
20 REM  FECHA     : FEB /93
30 REM  FUNCION   : CALCULAR COSTO DE CAMISAS
40 REM              EJEMPLO 13
50 INPUT "ESCRIBA EL NUMERO DE CAMISAS: ; CANTIDAD
60 IF CANTIDAD > 3 THEN 90
70 COSTO = CANTIDAD * 60
80 GO TO 100
90 COSTO = CANTIDAD * 50
100 PRINT CANTIDAD ; "CAMISAS CUESTAN : " , COSTO
110 END
```

EN PASCAL :

```
(*  AUTOR     : GRACIELA JARDON AVILA  Y  JOSE LUIS ARCE *)
(*  FECHA     : FEB /93                                  *)
(*  FUNCION   : CALCULO DEL COSTO DE CAMISAS             *)
PROGRAM EJEMPLO14 ;
VAR
CANTIDAD  : INTEGER  ;
COSTO     : REAL     ;
BEGIN
WRITE(`ESCRIBE EL NUMERO DE CAMISAS');
READLN (CANTIDAD) ;
IF CANTIDAD > 3
THEN
COSTO  := CANTIDAD * 50
ELSE
COSTO  := CANTIDAD * 50
WRITELN (CANTIDAD, `CAMISAS CUESTAN ' , COSTO )
END.
```

ESTRUCTURA DE ALTERNATIVA MULTIPLE (ON -GOTO Y CASE)

El diagrama representativo para una estructura de alternativa múltiple se muestra en la figura 4.2.

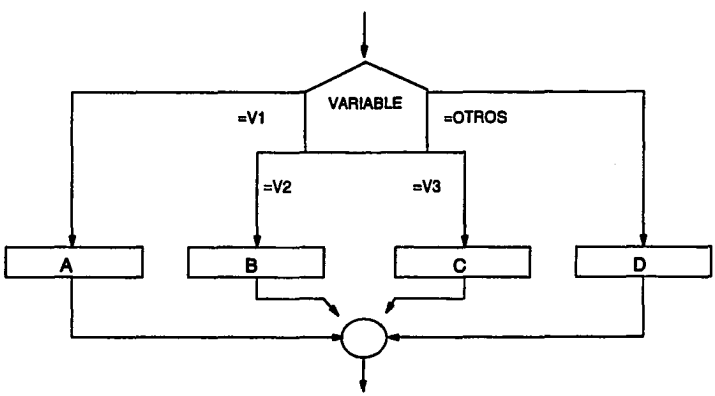


FIGURA 4.2

No obstante que las instrucciones GOTO e IF_THEN_ELSE son las más usuales para alterar la secuencia normal de ejecución, las instrucciones ON_GOTO y CASE, pueden alterar también la secuencia de ejecución.

EN BASIC :

Las instrucción ON_GOTO se utiliza para elegir entre varias alternativas y su formato general es:

```
ON expresión GOTO número renglón, número renglón, etc...
```

Al ejecutar la computadora procede como sigue:

1. Evalúa la parte entera de la expresión.
2. Si el resultado fuese igual a uno, transfiere el control al primer número que aparece en el formato de la instrucción; si fuera igual a dos al segundo y así sucesivamente.
3. Si el resultado fuese menor que uno o bien, fuera mayor que la posición ordinal del último número de renglón, la computadora mandará mensaje de error.

EN PASCAL :

La sentencia CASE se utiliza para elegir entre diferentes alternativas. Una sentencia CASE se compone de varias sentencias simples. Cuando CASE se ejecuta, una y sólo una de las sentencias simples se selecciona y ejecuta. Su formato general es:

```
CASE selector OF

   lista  constante 1  : sentencia 1 ;

   lista  constante 2  : sentencia 2 ;

      .
      .
      .
      lista  constante n  : sentencia n ;

          OTHERWISE

          sentencia x
END
```

Las reglas de ejecución son:

1.- La expresión selector se evalúa y se compara con las listas de uno o más posibles valores de selector separados por comas y, pueden ser constantes o tener un valor subrango. Sólo se ejecuta una sentencia. Si el valor del selector está en la lista de constantes 1, se realiza la sentencia 1 y el control pasa seguidamente a la primera instrucción después del end. Cada instrucción puede ser a su vez una instrucción Pascal simple o compuesta.

2.- La cláusula OTHERWISE es opcional.

3.- Si el selector no está comprendido en ninguna lista de constantes y no existe la cláusula OTHERWISE puede ocurrir un error.

4.- El selector debe ser de tipo ordinal (integer, char, boolean o enumerado).

EJEMPLO :

Los programas deben presentar con mucha frecuencia menús. Un menú es un conjunto de opciones que presenta al usuario, entre las cuales debe elegir una de ellas. Y dependiendo de la decisión se realizarán una serie de acciones. Un menú puede ser por ejemplo:

```
1  = NOMBRE CLIENTE
2  = DIRECCION
3  = TELEFONO
4  = CIUDAD
5  = FIN
```

Realizar el programa que permita elegir una opción (1, 2, 3, 4, 5)

EN BASIC :

```
10 REM  AUTOR    :GRACIELA JARDON   Y  JOSE LUIS ARCE
20 REM  FECHA    : MAR/92
30 REM  FUNCION  : PROGRAMA DE ELECCION MULTIPLE
40 REM             EJEMPLO 14
50 INPUT "TECLEE SU OPCION :" ; SELECCION$
60 IF SELECCION < 1 .OR. SELECCION > 5 GO TO 450
70 ON SELECCION GOTO 100, 200, 300, 400,500
100 INPUT "SU NOMBRE :"; NOMBRE$
110 GO TO 500
200 INPUT "SU DIRECCION: "; DIRECCION$
210 GO TO 500
300 INPUT "SU TELEFONO:"; TELEFONO$
310 GO TO 500
400 INPUT "SU CIUDAD:" ; CIUDAD$
410 GO TO 500
450 PRINT "OPCION INVALIDA"
460 GO TO 10
500 END
```

EN PASCAL :

```
PROGRAM EJEMPLO15 ;
  VAR
     SELECCION : INTEGER      ;
     NOMBRE    : STRING [40] ;
     DIRECCION : STRING [30] ;
     TELEFENO  : STRING [12] ;
     CIUDAD    : STRING [20] ;
  BEGIN
     WRITELN ('TECLEE SU OPCION') ;
     READLN (SELECCION) ;
     CASE SELECCION OF
           1  :  BEGIN
                       WRITE('SU NOMBRE: ') ;
                       READLN (NOMBRE)
                 END;
           2  :  BEGIN
                       WRITE ('SU DIRECCION:') ;
                       READLN (DIRECCION)
                 END;
           3  :  BEGIN
                       WRITE ('SU TELEFONO:') ;
                       READLN (TELEFONO)
                 END;
           4  :  BEGIN
                       WRITE ('SU CIUDAD: ') ;
                       READLN (CIUDAD)
                 END;
           5  : (* NO REALIZA NADA *)
                       OTHERWISE
                       WRITELN ('ELECCION NO VALIDA')
                 END
END.
```

LA INSTRUCCION FOR ...

En números ocasiones se puede desear un bucle que se ejecute un número determinado de veces, y cuyo número se conozca por anticipado. Para aplicaciones de este tipo se utiliza la instrucción FOR. La instrucción FOR requiere se conozca por anticipado el número de veces que se ejecutarán las instrucciones del interior del bucle. Su diagrama representativo se muestra en la figura 4.3.

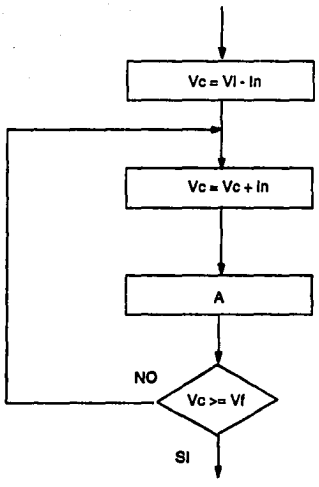


FIGURA 4.3

EN BASIC :

Las instrucciones FOR y NEXT constituyen una pareja; ninguna de ellas puede utilizarse aisladamente. La instrucción FOR encabeza el circuito y la instrucción NEXT lo termina. Su función es la de realizar instrucciones un número especifico de veces en un bucle. Su formato es el siguiente:

```
FOR variable = x TO y  [ STEP z ]
   .
   .
   .
NEXT
```

Donde :

- variable es utilizada como contador
- X, Y y Z son expresiones numéricas
- STEP z especifica el incremento o decremento del contador para cada iteración.

EN PASCAL :

El formato de la instrucción FOR es:

```
FOR variable := valor inicial  TO valor final DO
     sentencia  ;
```

ó

```
FOR variable := valor inical DOWNTO valor final DO
     sentencia  ;
```

Al ejecutarse por primera vez la sentencia FOR, el valor inicial se asigna a "variable" que se denomina variable de control, y a continuación se realiza la instrucción del interior del bucle. Si la sentencia es compuesta, entonces el formato de FOR es:

```
FOR  V :=  Vi   TO    Vf  DO
BEGIN
     sentencia 1 ;
     sentencia 2 ;
     .
     .
     .
     sentencia n
       END;
```

Se puede utilizar DOWNTO para decrementar el contador en lugar de TO.

Las reglas de funcionamiento de FOR son :

1.- Las variables de control, valor inicial y valor final deben ser todas del mismo tipo, pero el tipo real no está permitido. Los valores iniciales y finales pueden ser tanto expresiones como constantes.

2.- Antes de la primera ejecución del bucle a la variable de control se asigna el valor inicial.

3.- La última ejecución del bucle normalmente ocurre cuando la variable de control es igual al valor final.

4.- Cuando se usa la palabra reservada TO, la variable de control se incrementó en cada iteracción. Si se utiliza DOWNTO la variable se decrementa.

5.- Es ilegal intentar modificar el valor de la variable de control, el valor inicial y el valor final dentro del bucle.

6.- El valor de la variable de control se queda indefinido cuando se termina el bucle.

Ejemplo:

Programa que sume los enteros de la serie:

$$1 + 2 + 3 + 4 + ... + 20000$$

EN BASIC :

```
10 REM  AUTOR    : GRACIELA JARDON  Y  JOSE  LUIS ARCE
20 REM  FECHA    : FEB/93
30 REM  FUNCION  : CALCULAR LA SUMA DE LA SERIE DADA
40 REM             EJEMPLO 16
50 SUMA = 0
60 FOR N = 1 TO 20000 STEP 1
70 SUMA = SUMA + N
80 PRINT "LA SUMA DE LOS PRIMEROS "; N ; "NUMEROS ES"; SUMA
90 NEXT N
100 END
```

EN PASCAL :

```
(*  AUTOR  : GRACIELA JARDON AVILA  Y  JOSE LUIS ARCE *)
(*  FECHA  : MAR/93                                   *)
(*  FUNCION: SUMAR LA SERIE DADA                      *)
PROGRAM EJEMPLO 17;
 CONTS
   MAXIMO = 20000 ;
 VAR
   N, SUMA  : INTEGER ;
BEGIN
  SUMA := 0
  FOR N  := 1 TO MAXIMO  DO
    SUMA  := SUMA + N ;
    WRITE ('LA SUMA DE LOS ', N , 'PRIMEROS NUMEROS ES IGUAL
              A ') ;
    WRITELN (SUMA)
END.
```

La instrucción FOR puede ir en circuitos anidados. El requisito para utilizar circuitos anidados es que no se traslapen entre sí; es decir, que ninguno de ellos esté parcialmente fuera del otro.

LA INSTRUCCION WHILE ...

La estructura repetitiva WHILE es aquella en la que el número de iteracciones no se conoce por anticipado y el cuerpo del bucle se repite mientras se cumple una determinada condición. Por esta razón, a estos bucles se les denomina bucles condicionales. El diagrama se muestra en la figura 4.4.

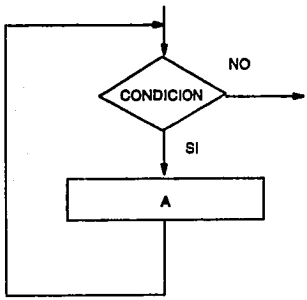


FIGURA 4.4

Las reglas de funcionamiento de la instrucción son las siguientes:

1.- La condición (expresión lógica) se evalúa antes y después de cada ejecución del bucle. Si la condición es verdadera, se ejecuta el bucle, y si es falsa, el control pasa a la sentencia siguiente del bucle.

2.- Si la condición se evalua a falso cuando se ejecuta el bucle por primera vez, el cuerpo del bucle no se ejecutará nunca. En este caso se dice que el bucle se ha ejecutado cero veces.

3.- Mientras la condición sea verdadera el bucle se ejecutará. Esto significa que el bucle se realizará indefinidamente a menos que "algo"" en el interior del bucle modifique la condición haciendo que su valor pase a falso. Si la expresión nunca cambia de valor, entonces el bucle no termina nunca y se denomina bucle infinito o sin fin.

EN BASIC :

```
WHILE expresión
.
.
.
WEND
```

EN PASCAL :

a). Sentencia simple:

```
WHILE expresión lógica  DO
sentencia ;
```

b). Sentencia compuesta:

```
WHILE expresión lógica  DO
BEGIN
       sentencia 1
       .
       .
       .
       sentencia n
END;
```

EJEMPLO :

Calcular el área de un círculo incrementando el radio de 1 en 1. Terminar el programa cuando el área sea mayor a 500.

EN BASIC :

```
10 REM  AUTOR    : GRACIELA JARDON A. Y JOSE LUIS ARCE
20 REM  FECHA    : MAR/93
30 REM  FUNCION : CALCULO DEL AREA DE UN CIRCULO
40 REM             EJEMPLO 17
50 PI = 3.1416
60 RADIO = 1
70 WHILE AREA <= 500
80     AREA = PI = * R ^ 2
90     PRINT "PARA EL RADIO;" ; RADIO
100    PRINT :EL AREA ES:" ; AREA
110    RADIO = RADIO + 1
120 WEND
140 END
```

EN PASCAL :

```
(* AUTOR    : GRACIELA JARDON AVILA  Y  JOSE LUIS ARCE *)
(* FECHA    : MAR/93                                    *)       (*
FUNCION : CALCULO DEL AREA DE UN CIRCULO          *)
PROGRAM EJEMPLO 18 ;
 CONST
      PI = 3.1416 ;
```

```
VAR
   RADIO  : REAL ;
   AREA   : REAL ;
   RADIO2 : REAL ;
BEGIN
   RADIO := 1
   WHILE AREA <= 500 DO
      BEGIN
         RADIO2 := RADIO * RADIO
         AREA = PI * RADIO2
         WRITELN ('PARA EL RADIO: ', RADIO )
         WRITELN (EL AREA ES: ', AREA)
         RADIO = RADIO + 1
      END;
END.
```

LA INSTRUCCION REPEAT ...

Los ciclos REPEAT-UNTIL son muy similares a los ciclos WHILE en el grado de versatilidad que ofrecen. De hecho, hay muchas situaciones en las que funciona tan bien una estructura de ciclo como la otra, y es una cuestión de preferencia personal cual se escoja. También se dan casos en los que una de estas estructuras de ciclos y -no la otra- debe ser usada. El diagrama general de funcionamiento se muestra en la figura 4.5.

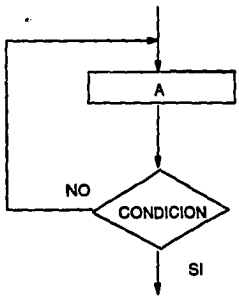


FIGURA 4.5

Las reglas de funcionamiento son:

1.- La condición (expresión lógica) se evalúa al final del bucle, después de ejecutarse todas las sentencias.

2.- Si la expresión lógica es falsa, se sale del bucle y se ejecuta la siguiente sentencia a UNTIL.

3.- La sintaxis no requiere BEGIN y END.

EN BASIC :

```
No existe esta instrucción.
```

EN PASCAL :

El formato general de la instrucción es:

```
REPEAT
 sentencia 1
   .
   .
   .
  sentencia n
UNTIL expresión lógica
```

EJEMPLO :

Calcular la suma de un número arbitrario de enteros de entrada.

EN PASCAL :

```
(*  AUTOR     : GRACIELA JARDON   Y JOSE LUIS ARCE *)
(*  FECHA     : MAR/93                              *)
(*  FUNCION   : CALCULO DE NUMEROS ENTEROS          *)
PROGRAM EJEMPLO 19;
  VAR
      SUMA, NUMERO : INTEGER ;
      RESPUESTA : CHAR ;
BEGIN
  SUMA := 0
  REPEAT
      WRITE ('TECLEE NUMERO ') ;
      READLN (NUMERO);
      SUMA := SUMA + NUMERO ;
      WRITE('TECLEE  S  PARA CONTINUAR,  N  PARA PARAR:');
      READLN (RESPUESTA)
  UNTIL RESPUESTA = 'N';
  WRITE ('LA SUMA ES : ', SUMA)
END.
```

4.6 ARREGLOS UNIDIMENSIONALES Y BIDIMENSIONALES

Los datos manejados anteriormente han sido los denominados datos simples. En un gran número de problemas es necesario manejar un conjunto de datos, más o menos grande, que estén relacionados entre sí, de tal forma que constituyen una unidad para su tratamiento. Por ejemplo, si se quiere manipular una lista de 100 nombres de personas, es conveniente tratar este conjunto de datos de forma unitaria en lugar de utilizar 100 variables, una para cada dato simple.

Un conjunto de datos homogéneos que se tratan como una sola unidad se denomina estructura de datos.

Si una estructura de datos reside en la memoria central de la computadora, se denomina estructura de datos interna. Recíprocamente si reside en un soporte externo, se llama estructura de datos externa.

La estructura de datos interna más importante desde el punto de vista de utilización es el arreglo (tabla), que existe en la práctica totalidad de los lenguajes de programación.

Una tabla consiste en un número fijo, finito y ordenado de elementos, todos del mismo tipo y bajo un nombre común para todos ellos.

Se denominan componentes a los elementos de una tabla.

La posición de cada componente dentro de la tabla está determinada por uno o varios índices. A cada componente se puede acceder de forma directa indicando sus índices y el nombre de la tabla.

Una tabla se puede estructurar en una, dos o más dimensiones según el número de índices necesarios para acceder a sus elementos. Por lo tanto, la dimensión de una tabla es el número de índices que utiliza.

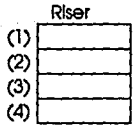
Longitud o tamaño de una tabla es el número de componentes que contiene.

El tipo de una tabla es el tipo de sus componentes.

Los componentes de una tabla se utilizan de la misma forma que cualquier otra variable de un programa, pudiendo por lo tanto intervenir en instrucciones de asignación, entrada/salida, etcétera.

Por ejemplo, un arreglo puede compararse con un edificio de apartamentos. El nombre del arreglo es el nombre del edificio. Los elementos del arreglo son los apartamentos, cada uno de los cuales tiene un número índice que corresponde al de un apartamento.

Así como Riser(4) podría ser o representar el apartamento 4 de un edificio llamado Riser.



Para especificar un elemento concreto de un arreglo, se debe dar el nombre del arreglo y el índice. Ejemplo:

El arreglo que contiene 8 nombres de personas se muestra en la figura 4.6.

LUIS	JOSE	ROSA	JUAN	TERE	TOÑA	LOLA	JAVI

Figura 4.6

Arreglo : La estructura de datos representada.

Nombre del arreglo:	ALUMNOS.
Componentes:	ALUMNOS(1), ALUMNOS(2)... ALUMNOS(8).
Indice:	Los números del 1 al 8 que direccionan cada componente.
Dimensión:	Una.
Longitud:	Ocho.
Tipo :	Alfanumérica.

Los arreglos se clasifican segun su dimensión en:

- Unidimensionales
- Bidimensionales

Arreglos unidimensionales.

Son arreglos de una dimensión también denominados vectores.

Tienen un solo índice. Cada componente del vector se direcciona mediante su nombre seguido del número correspondiente al índice entre paréntesis.

El ejemplo anterior nos muestra una tabla de este tipo, en la cual la tercera componente que contiene el valor "ROSA" se denota por:

```
ALUMNOS(3)
```

En los diferentes lenguajes de programación hay que declarar los arreglos antes de su utilización.

EN BASIC :

Con la instrucción DIM se especifica el tamaño de un arreglo. Así para definir el tamaño de un arreglo A(J), donde J = 1...1000, introducimos la instrucción:

```
10 DIM A(1000)
```

Con esta instrucción le indicamos a la computadora que espere variables A(0), A(1), ... , A(1000) en el programa y que reserve espacio en la memoria para 1000 unidades.

Es posible definir varios arreglos por medio de una instrucción DIM. Por ejemplo:

```
10 DIM A(1000), B$(5)
```

Define el arreglo A(0)...A(1000), y el arreglo de cadena B$(0)... B(5).

EN PASCAL :

El formato para declarar un arreglo en PASCAL es:

```
TYPE
   nombre-array = ARRAY [ tipo subíndice ] OF TIPO
```

Donde:

Nombre-array .- Identificador válido.
Tipo subíndice .- Puede ser un tipo ordinal: boolean o char, un tipo enumerado o un tipo subrango. Existe un elemento por cada valor del tipo subíndice.
Tipo .- Describe el tipo de cada elemento.

Ejemplo:

Un array X de ocho elementos reales

```
TYPE
  X = ARRAY [1..8] OF real ;
```

Variables tipo ARRAY

Las declaraciones de tipo de dato ARRAY no crean ninguna variable específica tipo ARRAY. En realidad la declaración TYPE proporciona información perteneciente a un grupo de ARRAYS como un todo.

Las variables tipo arrays son creadas utilizando la declaración VAR, cuyo formato general es:

```
VAR
  nombrearrays : nombretipo ;
```

Donde :

nombrearrays.- nombre para la variable
nombretipo.- igual nombre que el utilizado en TYPE

Ejemplo:

```
TYPE
  Valores = ARRAY [-10..10] OF REAL ;
VAR
  precios : Valores;
```

El subíndice de la variable ARRAY Precios debe ser un entero entre -10 y 10, cada elemento de ARRAY contiene un valor real.

Ejemplo:

```
Const
  Max  =  500 ;
Type
  Texto = ARRAY [ 1.. Max] OF CHAR ;
Var
  Palabras, letras  : Texto
```

La declaración VAR en este ejemplo crea dos variables arrays, cuyos nombres son palabras y letras. Consultando las declaraciones Const y Type se observa que los subíndices de estos arrays deben ser un entero entre 1 y 500; cada elemento de estos arrays contendrán un valor tipo caracter.

El número de elementos de un array viene definido por la diferencia entre los índices mayor y menor más uno. El array Precios contiene 10 (-10) + 1 = 21 elementos. Deberá tener cuidado al declarar variables, no exceder la capacidad de memoria de su máquina cuando se declaran arrays.

Los tipos de datos array pueden ser cualquier dato válido.

Ejemplo:

```
VAR ESTADO : ARRAY [1..1000] OF BOOLEAN ;
```

Los componentes de este array son:

```
ESTADO [1]
ESTADO [2]
.
.
.
ESTADO [1000]
```

Los datos pueden ser de tipo enumerado.

Ejemplo:

```
  TYPE
    LUCESTRAFICO = (ROJO, VERDE, AMBAR);
VAR
    TRAFICO : ARRAY [1..4] OF LUCESTRAFICO ;
```

EJEMPLO:

Programa que lea las calificaciones de un alumno en 10 asignaturas, las almacene en un arreglo unidimensional, calcule e imprima su media.

EN BASIC :

```
10 REM  AUTOR   : GRACIELA JARDON AVILA  Y  JOSE LUIS ARCE
20 REM  FECHA   : MAR/93
30 REM  FUCION  : USO DE ARREGLO UNIDIMENSIONAL
40 REM            EJEMPLO 20
50 DIM NOTAS(10)
60 FOR I = 1 TO 10
70 PRINT "NOTA No. "; I
80 INPUT NOTAS(I)
90 NEXT I
100 SUMA = 0
110 FOR I = 1 TO 10
120 SUMA = SUMA + NOTAS(I)
130 NEXT I
140 MEDIA = SUMA / 10
150 PRINT "NOTA MEDIA:" ; MEDIA
160 END
```

EN PASCAL :

```
(* AUTOR    : GRACIELA JARDON AVILA  Y  JOSE LUIS ARCE *)
(* FECHA    : MAR/93                                   *)
(* FUNCION : USO DE ARREGLO UNIDIMENSIONAL             *)
PROGRAM EJEMPLO21 ;
  VAR
      NOTAS  : ARRAY[1..10] OF REAL
      SUMA, MEDIA : REAL ;
      I : INTEGER ;
  BEGIN
      FOR I:= 1 TO 10 DO
         BEGIN
            WRITE('NOTA No. ', I) ;
            READLN(NOTAS[I])
         END;
      SUMA := 0 ;
      FOR I := 1 TO 10 DO
        SUMA := SUMA + NOTAS[I] ;
        MEDIA := SUMA / 10 ;
        WRITE ('NOTA MEDIA : ', MEDIA)
   END.
```

ARREGLOS BIDIMENSIONALES

Son arreglos de dos dimensiones. También se denominan matrices. Tienen dos índices, por lo cual cada componente de la matriz se direcciona mediante su NOMBRE seguido de los dos índices separados por coma y entre paréntesis.

Ejemplo:

Matriz de 6 filas y 8 columnas conteniendo el número de alumnos matriculados en cada grupo de un centro docente por asignatura. Las filas corresponden a los grupos y las columnas a las asignaturas, como se muestra en la figura 4.7.

1	35	30	32	32	34	35	34	28
2	40	33	40	37	36	39	40	29
3	25	23	26	21	24	24	25	15
4	33	33	33	32	34	30	32	20
5	45	44	45	44	43	40	44	33
6	24	20	22	22	24	25	24	12
	1	2	3	4	5	6	7	8

Figura 4.7

Donde:

Nombre de la tabla:	MATRICULA
Componentes:	MATRICULA(1,1), MATRICULA(1,2),...MATRICULA(6,8)
Indices:	Los números del 1 al 6 para filas y los números 1 al 8 para columnas.
Dimensión:	Dos
Longitud:	6 * 8 = 48.
Tipo:	numérica entera.

La componente MATRICULA(4,6) almacena el número de alumnos matriculados en el grupo número 4 en la asignatura número 6, que en el ejemplo representado son 30 alumnos.

EN BASIC :

Los arreglos bidimensionales se declaran con la instrucción DIM indicando en los subíndices el número de renglones y el número de columnas. Ejemplo:

```
10 DIM  A(5,4)
```

Indica que se define un arreglo de cinco renglones y cuatro columnas.

EN PASCAL :

Al igual que en los arrays de una dimensión, los arrays bidimensionales se crean con declaraciones TYPE y VAR cuando un programa se codifica en PASCAL , como se muestra en la figura 4.8.

1	T(1,1)	T(1,2)	T(1,3)					T(1,8)
2								
3				T(3,4)				
4								
5	T(5,1)							T(5,8)
	1	2	3	4	5	6	7	8

Figura 4.8

Se deben indicar:

1.- Nombre del array.
2.- Tipo del array.
3.- El rango permitido por cada subíndice.

El formato es:

a).

```
TYPE
 identificador = ARRAY [índice1, índice2] OF tipo elemento;
```

b).

```
TYPE
  identificador = ARRAY[índice] OF ARRAY[índice2] OF tipo
```

Ejemplo:

```
TYPE
  TABLA = ARRAY[1..25,1..4] OF REAL;
VAR
  GRADOS : TABLA ;
```

Reserva 100 posiciones de memoria datos reales 25 filas por 4 columnas.

Grados [i,j] se refiere a la fila i y a la columna j

EJEMPLO:

Calcular la media de una lista de veinticinco alumnos de una clase de informática con notas en cuatro asignaturas, como se muestra en la figura 4.9.

NUMERO ESTUDIANTE	ALGORITMOS	PASCAL	BASE DE DATOS	INGENIERIA DE SOFTWARE
1	7.20	6.00	7.00	5.00
2	8.40	5.80	7.40	4.50
3	9.60	4.20	6.00	3.20
4	1.30	2.50	4.80	2.40
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
25	5.20	6.00	7.80	4.00

Figura 4.9

EN BASIC :

```
10 REM  AUTOR    : GRACIELA JARDON   Y JOSE LUIS ARCE
20 REM  FECHA    : MAR/93
30 REM  FUNCION : FUNCIONAMIENTO DE ARREGLO BIDIMENSIONAL
40 REM             EJEMPLO 22
50 DIM NOTAS(25,4)
60 FOR I = 1 TO25
70 PRINT "INTRODUCCION NOTAS DEL ESTUDIANTE"; I
80 PRINT "Y CALCULO DE SU MEDIA"
90 SUMA = 0
100 PRINT "INTRODUZCA LAS NOTAS DE LAS CUATRO ASIGNATURAS"
110 FOR J = 1 TO 4
120 INPUT NOTAS(I,J)
130 SUMA = SUMA + NOTAS(I,J)
140 NEXT J
150 PRINT
160 MEDIA = SUMA / 4
170 PRINT "LA NOTA MEDIA DEL ESTUDIANTE"; I ; "ES" ; MEDIA
180 NEXT I
190 END
```

EN PASCAL :

```
(* AUTOR    : GRACIELA JARDON  Y  JOSE LUIS ARCE  *)
(* FECHA    : MAR/93                                *)
(* FUNCION : USO DE ARREGLO BIDIMENSIONAL     *)
PROGRAM EJEMPLO 22;
VAR
  NOTAS   :  ARRAY [1..25, 1..4] OF REAL ;
  I, J    : INTEGER ;
  SUMA, MEDIA : REAL ;
BEGIN
  FOR I := 1 TO 25 DO
  BEGIN
      WRITE ('INTRODUCCION NOTAS DEL ESTUDIANTE ',I);
      WRITELN('Y CALCULO DE SU NOTA MEDIA ');
      SUMA := 0
      WRITELN ('INTRODUZCA LAS NOTAS DE LAS CUATRO
            ASIGNATURAS');
      FOR J := 1 TO 4 DO
        BEGIN
            READ (NOTAS[I,J]);
            SUMA := SUMA + NOTAS [I,J]
        END;
      READLN ;
      MEDIA := SUMA / 4 ;
      WRITELN ('LA NOTA MEDIA DEL ESTUDIANTE ',I,
            'ES ', MEDIA)
      END
END.
```

4.7 FUNCIONES Y SUBPROGRAMAS

FUNCIONES

Una función calcula y regresa un solo resultado basado en uno o más valores que se le pasen. Ya se han considerado varias funciones integradas como abs() y sin(X); para satisfacer las necesidades de un problema dado el programador puede también definir funciones propias.

EN BASIC :

El lenguaje BASIC permite al usuario definir funciones por medio de la instrucción DEF. La sintaxis es:

```
DEF FN_NOMBRE [ARGUMENTO]  EXPRESION
```

Donde :

NOMBRE debe ser un nombre de variable válido. Este nombre precedido por el FN se convierte en el nombre de la función.

ARGUMENTOS consiste en los nombres de las variables en la definición de la función que deben ser reemplazadas en el momento de llamar la función. Los elementos de la lista se separan por comas.

EXPRESION es una expresión que realiza la operación de la función; está limitada a una instrucción.

En matemáticas, las funciones suelen definirse especificando una o más fórmulas. Por ejemplo, a continuación damos las fórmulas que definen tres funciones F(X), G(X) y H(X):

```
F(X) = ( X ^ 2 - 1 ) ^ ( 1 / 2 )

G(X) = 3 * x ^ 2 - 5 * X - 15

H(X) = 1 / ( X - 1 )
```

Note que cada función se nombra con una letra, es decir, F, G, H respectivamente. El BASIC permite definir funciones como las precedentes y utilizarlas por nombre en todo el programa.

Por ejemplo, si queremos definir la función (x) anterior, emplearemos la proposición:

```
10 DEF FNF ( X ) = ( X ^ 2 - 1 )  ^  ( 1 / 2 )
```

Para definir la función G(X) anterior usamos la proposición:

```
20 DEF FNG(X) = 3 * X ^ 2 - 5 * X - 15
```

Observe que en cada caso con la letra (F o G) identificaremos la función. Supongamos que queremos calcular el valor de la función F para X = 12.5. Una vez definida la función, el cálculo puede definirse a la computadora como FNF(12.5).

Ese cálculo se empleará en todo el programa y nos ahorrará el esfuerzo de reteclear la fórmula correspondiente a la función en cada caso. Puede usarse cualquier nombre válido de variable como nombre de función. Por ejemplo, una función INTERESES se definirá mediante la proposición:

```
10 DEF FNINTERESES(X) = ...
```

Ejemplo:

Programa para tabular el valor de la función siguiente para:

X = 0, 0.1, 0.2 0.3, ..., 10.0

F(X) = 5 EXP(-2X)

```
10 REM AUTOR    : GRACIELA JARDON A.  Y  JOSE LUIS ARCE G.
20 REM FECHA    : MAR/93
30 REM FUNCION : MOSTAR EL USO DE FUNCIONES
 40 REM            EJEMPLO 23
 50 DEF = FNF(X) = 5 * EXP(-2 * X)
 60 FOR X = 0 TO 10 STEP 0.1
 70 PRINT X, FNF(X)
80 NEXT X
90 END
```

EN PASCAL :

El formato de una función es:

```
FUNCTION NOMBRE ( P1, P2, ...) : TIPO

( DECLARACIONES LOCALES Y SUBPROGRAMAS )
```

EJEMPLO:

EN BASIC:

```
10 REM AUTOR     : GRACIELA JARDON  Y  JOSE LUIS ARCE
20 REM FECHA     : MAR/93
30 REM FUNCION   : MOSTRAR EL USO DE FUNCIONES DEFINIDAS
40 REM           : EJEMPLO 26
50 X = 3
60 DEF FNCUBO(X) = X ^ 3
70 PRINT " EL CUBO DE  3 ES " ; FNCUBO(X)
80 END
```

EN PASCAL :

```
   (* AUTOR     : GRACIELA JARDON  Y JOSE LUIS ARCE  *)
   (* FECHA     : MAR/93                             *)
   (* FUNCION   : MOSTRAR EL USO DE FUNCIONES DEFINIDAS *)
PROGRAM EJEMPLO 26;
  VAR
    N : INTEGER
  FUNCTION CUBO (X : INTEGER ) : INTEGER ;
    BEGIN
      CUBO := X * X * X
    END;
 (*                                                  *)
BEGIN
  T := CUBO (3) ;
  WRITELN ('EL CUBO DE 3 ES  ', T) ;
  WRITELN ('4 AL CUBO ES  : ', CUBO(4) ) ;
END.
```

SUBPROGRAMAS

En los problemas simples es posible escribir programas sin mucha planeación previa. En los problemas más difíciles, sin embargo, es importante usar un enfoque sistemático para diseñar los programas.

Utilizando la programación descendente se empieza haciendo un bosquejo del cuerpo principal. Si el problema es muy complicado, será necesario trazar varios bosquejos del cuerpo principal, en donde cada uno irá completando cada vez más detalles.

Estos bosquejos sucesivos se conocen como refinamiento por pasos y se muestran en la figura 4.10.

EJEMPLO:

EN BASIC:

```
10 REM AUTOR     : GRACIELA JARDON  Y  JOSE LUIS ARCE
20 REM FECHA     : MAR/93
30 REM FUNCION   : MOSTRAR EL USO DE FUNCIONES DEFINIDAS
40 REM           : EJEMPLO 26
50 X = 3
60 DEF FNCUBO(X) = X ^ 3
70 PRINT " EL CUBO DE  3 ES " ; FNCUBO(X)
80 END
```

EN PASCAL :

```
  (* AUTOR     : GRACIELA JARDON  Y JOSE LUIS ARCE  *)
  (* FECHA     : MAR/93                             *)
  (* FUNCION   : MOSTRAR EL USO DE FUNCIONES DEFINIDAS *)
PROGRAM EJEMPLO 26;
  VAR
    N : INTEGER
  FUNCTION CUBO (X : INTEGER ) : INTEGER ;
    BEGIN
      CUBO := X * X * X
    END;
 (*                                                  *)
BEGIN
  T := CUBO (3) ;
  WRITELN ('EL CUBO DE 3 ES  ', T) ;
  WRITELN ('4 AL CUBO ES  : ', CUBO(4) ) ;
END.
```

SUBPROGRAMAS

En los problemas simples es posible escribir programas sin mucha planeación previa. En los problemas más difíciles, sin embargo, es importante usar un enfoque sistemático para diseñar los programas.

Utilizando la programación descendente se empieza haciendo un bosquejo del cuerpo principal. Si el problema es muy complicado, será necesario trazar varios bosquejos del cuerpo principal, en donde cada uno irá completando cada vez más detalles.

Estos bosquejos sucesivos se conocen como refinamiento por pasos y se muestran en la figura 4.10.

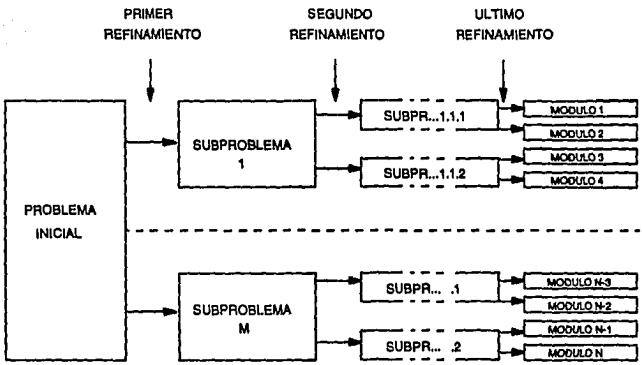


FIGURA 4.10

La utilización de esta técnica de diseño, tiene los siguientes objetivos básicos:

- Simplificación del problema y de los subprogramas resultantes de cada descomposición.

- Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.

- El programa final queda estructurado en forma de bloques o módulos, lo que hace más sencilla su lectura y mantenimiento.

EN BASIC :

Siempre que en un programa se requiera dos o más veces la misma operación, una sola subrutina puede servir para resolverlos.

De esta forma, se puede evitar la duplicidad de codificaciones. La transferencia de control a la subrutina se puede lograr con la instrucción GOSUB cuyo formato es:

```
GOSUB n
```

En donde n representa el número del renglón en que principia la subrutina particular. Por ejemplo, la instrucción:

```
10 GOSUB 460
```

Transfiere el control al renglón 460. Virtualmente, esta instrucción equivale a GOTO, con la única excepción de que, en el caso de GOSUB, la computadora recuerda en qué lugar estaba cuando se le ordenó transferir el control al renglón 460. Esto es, en cuanto encuentra la instrucción RETURN, cuyo formato es, simplemente:

```
RETURN
```

Transferirá de regreso, el control a la instrucción inmediata siguiente a GOSUB. El siguiente dibujo muestra el funcionamiento de GOSUB, como se muestra en la figura 4.11.

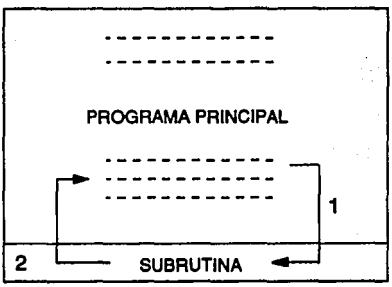


FIGURA 4.11

De lo anterior se deduce que es posible llamar una subrutina desde diferentes etapas del programa mediante la instrución GOSUB y, en vista de que la computadora recuerda el lugar en que estaba, regresa a él sin riesgo de confusión.

Para ilustrar la manera de utilizar subrutinas en BASIC, consideraremos la evaluación de la siguiente expresión:

```
X = N! / ( J!  ( N - J)! )
```

En donde N! = 1 x 2 x ... x (N-1) x N , y 0! = 1.

Suponga que nuestro programa va a leer N y J. El factorial de K se puede calcular por medio de las instrucciones siguientes:

```
10 REM CALCULO DEL FACTORIAL DE K = F
20 F = 1
30 IF K = 0 THEN 70
40 FOR L = 1 TO K
50 F = F * L
60 NEXT L
70 RETURN
```

El siguiente programa muestra la versión completa para calcular la anterior expresión:

```
10 REM AUTOR     : GRACIELA JARDON  Y  JOSE LUIS ARCE
20 REM FECHA     : MAR/93
30 REM FUNCION   : USO DE SUBRUTINAS
50 READ N , J
60 REM CALCULO DEL FACTORIAL DE N
70 K = N
80 GOSUB 800
110 N1 = F
130 REM CALCULO DEL FACTORIAL DE J
150 K = J
160 GOSUB 800
170 J1 = F
180 REM CALCULAR EL FACTORIAL DE N - J
210 K = N - J
220 GOSUB 800
230 REM RESULTADOS
260 X = N1 / J1 * F )
270 PRINT N , J , X
280 GO TO 1000
300 REM SUBRUTINA
800 F = 1
810 IF K = 0 THEN 850
820 FOR L = 1 TO K
830 F = F * L
840 NEXT L
850 RETURN
900 DATA 8 , 3
1000 END
```

En un solo programa puede haber más de una subrutina. De hecho, una de ellas puede llamar a otra, dando lugar a subrutinas entrelazadas.

EN PASCAL :

Un procedimiento es un subprograma que realiza una tarea específica. Puede recibir cero o más valores del programa que llama y devolver cero o más valores a dicho programa llamador. Un procedimiento está compuesto de un grupo de sentencias a las que se asigna un nombre (identificador) y constituye una unidad de programa.

Al igual que cualquier otro identificador en PASCAL, todos los procedimientos se deben declarar dentro del cuerpo del programa. La declaración de un procedimiento no indica a la computadora que ejecute las instrucciones dadas, sino que indica a la maquina cuáles son estas instrucciones y dónde están localizadas cuando sea necesario. El formato es el siguiente:

a).

```
PROCEDURE  nombreproc ;
   declaraciones locales
BEGIN
  cuerpo del procedimiento
END;
```

b).

```
PROCEDURE nombreproc (lista parámetros formales) ;
   declaraciones locales
BEGIN
   cuerpo del procedimiento
END ;
```

Donde:

nombreproc.-	identificador válido.
lista parámetros formales.-	parámetros formales del
procedimiento;	sirven para pasar información del procedimiento a la unidad de programa que le invoca; tiene el formato:
lista1:	tipo 1 ; lista 2 : tipo 2 ; ...
lista:	uno o varios identificadores.

Los procedimientos se llaman dentro de un programa o de otro procedimiento directamente por su nombre, de acuerdo con los formatos a). o b). Ejemplo:

- Procedimiento ESTRELLAS: visualiza 15 asteriscos

```
PROCEDURE ESTRELLAS ;
BEGIN
  WRITE ('***************')
END;
```

- Programa que incorpora el procedimiento ESTRELLAS

```
PROGRAM RECUADRO ;
     PROCEDURE ESTRELLAS ;
        BEGIN
   WRITE ('***************')
  END;
   BEGIN
  ESTRELLAS ;
     WRITE ('MENSAJES') ;
  ESTRELLAS
   END.
```

Cuando se llama a un procedimiento, se instruye a la computadora para ejecutar las sentencias que constituyen el cuerpo de éste. Como ya se ha comentado para llamar a un procedimiento, basta simplemente escribir en la sección ejecutable del programa principal el nombre del procedimiento que desea realizar la computadora. Al ejecutar la computadora la sentencia que llama al procedimiento, se localiza este procedimiento y se ejecutan todas las instrucciones del cuerpo de ese procedimiento, como se muestra en la figura 4.12.

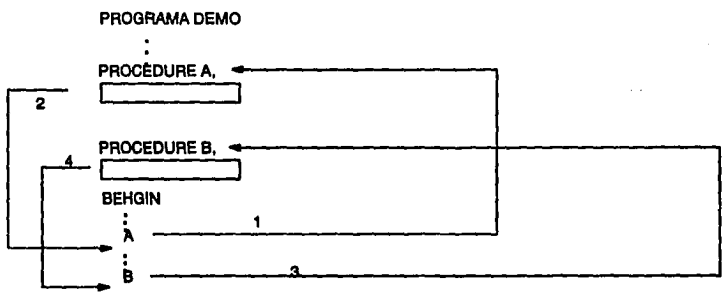


FIGURA 4.12

Cuando el cuerpo del procedimiento se termina de ejecutar, se retorna al programa principal y se realiza la sentencia inmediatamente a la siguiente que llamó al procedimiento.

Si en lugar de un procedimiento el programa contiene varios procedimientos, el orden de escritura de los mismos no tiene importancia si no se invocan unos a otros. En caso de que dos procedimientos se llamen entre sí, se debe seguir la siguiente regla de escritura:

Si subA y subB son subprogramas definidos en la misma unidad de programa y si subA es

llamado por subB, entonces subA debe estar definido antes de subB.

Ejemplo:

Calcular el área de un cilindro dado.

área = 2 * pi * radio * altura

```
(* AUTOR     : GRACIELA JARDON   Y  JOSE LUIS ARCE  *)
(* FECHA     : MAR/93                               *)
(* FUNCION   : CALCULO DEL AREA DE UN CILINDRO      *)
PROGRAM EJEMPLO 28 ;
CONST
   PI = 3.1416 ;
 VAR
   RADIO, ALTURA : REAL
 AREA            : REAL


PROCEDURE LEER ;
BEGIN
  WRITE ('INTRODUCIR RADIO ') ;
    READLN (RADIO) ;
    WRITE ('INTRODUCIR ALTURA') ;
    READLN (ALTURA)
END;
PROCEDURE AREA ;
  BEGIN
    AREA :=  PI * RADIO * ALTURA
END;
PROCEDURE SALIDA ;
BEGIN
  WRITELN ('LA ALTURA DEL CILINDRO ES ', ALTURA) ;
    WRITELN ('EL RADIO DEL CILINDRO ES ', RADIO) ;
  WRITELN ('EL AREA DEL CILINDRO ES  ', AREA ) ;
END;

BEGIN
  LEER ;
    WRITELN ;
  AREA ;
    WRITELN ;
    SALIDA ;
END.
```

CAPITULO V

MANEJO DE ARCHIVOS CON BASIC Y PASCAL ESTRUCTURADO

5.1 INTRODUCCION AL MANEJO DE ARCHIVOS

Los objetos tratados por un programa, que hemos visto hasta ahora, tienen dos restricciones importantes; por un lado, la cantidad de datos que pueden almacenar es bastante reducida, por ser limitada a la memoria central del computador; por otro lado su existencia está condicionada al tiempo que dure la ejecución del programa; es decir, cuando termina el programa, todos sus datos desaparecen de la memoria central.

Para abordar un aspecto importante de la programación, que trata de la manipulación y almacenamiento de datos para futuros usos, se utilizan las estructuras de datos externas denominadas ficheros o archivos.

Los archivos no están contenidos en la memoria central del computador, sino que residen en soportes externos que abren comunicación con el mismo al ser solicitada.

Su nombre corresponde al concepto clásico de conjunto de fichas conteniendo información relativa a un mismo tema. Los soportes donde residen estos archivos pueden ser una carpeta, un armario, etc., existiendo algunas reglas o criterios de clasificación y manipulación.

Desde el punto de vista informático, un archivo es algo similar, residente en un soporte de información externo como puede ser un disco o una cinta magnética.

Un archivo o fichero se compone de registros, siendo éstos la unidad de acceso y tratamiento.

Esta estructura es fundamental, debido a que nos permite almacenar cualquier tipo de información; como por ejemplo datos, textos, gráficos, programas, etc.

CONCEPTOS Y DEFINICIONES

Un archivo o fichero es un conjunto de informaciones estructuradas en unidades de acceso denominadas registros, todos del mismo tipo.

Un registro (registro lógico) es una estructura de datos formada por uno o más elementos denomidados campos, que pueden ser de diferentes tipos y que a su vez, pueden estar compuestos por subcampos.

Se denomina clave (identificativo o llave) a un campo especial del registro que sirve para identificarlo.

Algunos archivos en sus registros no tienen campo clave, mientras que otros pueden tener varios, denominados éstos, clave primaria, secundaria, etc.

Otro concepto es el de bloque (registro físico), correspondiente a la cantidad de información que se transfiere en cada operación de lectura o escritura sobre un archivo. Su tamaño depende de las características físicas del computador utilizado.

Se denomina factor de bloqueo al número de registros lógicos que contiene cada bloque.

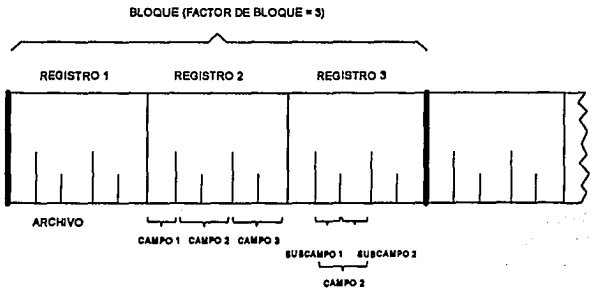


Figura 5.1

Características de archivos.

Las principales características de esta estructura son:

- Independencia de las informaciones respecto de los programas.
- Un archivo puede ser accedido por distintos programas en distintos momentos.
- La información almacenada es permanente.
- Gran capacidad de almacenamiento.

Clasificación de los archivos según su uso.

Los archivos se clasifican según su uso en tres grupos:

- Permanentes o maestros. Contienen información que varía poco.

- De movimientos. Se crean para actualizar los archivos maestros. Sus registros son de tres tipos: altas, bajas y modificaciones.

- De maniobra o trabajo. Tienen una vida limitada, normalmente menor que la duración de la ejecución de un programa. Se utilizan como auxiliares de los anteriores.

Organización de archivos.

Los archivos se organizan para su almacenamiento y acceso, según las necesidades de las aplicaciones que los van a utilizar.

- Secuencial.

- Directa o aleatoria.

- Secuencial indexada.

Cada uno tiene sus ventajas y desventajas sobre el otro. Los archivos secuenciales son más eficientes en espacio de almacenamiento, están diseñados para almacenar información "permanente" que cambia con poca frecuencia, se dificulta el movimiento de datos dentro del archivo y el tiempo de acceso a un registro determinado es muy lenta.

Por el contrario, los archivos aleatorios utilizan más espacio en disco, pero permiten modificar más fácilmente la información y se tiene un acceso directo a un elemento del archivo sin recorrerlo totalmente permitiendo un acceso más rápido, este tipo de archivos se usan cuando la información cambia continuamente.

En general, en un archivo secuencial es más fácil escribir y leer la información, pero no modificarla; por lo tanto no son los mejores para la mayoría de las aplicaciones.

Antes de utilizar un archivo de datos se tiene que analizar la información que se desea obtener de los datos almacenados en un archivo, la velocidad de acceso necesario y si los datos a almacenar cambian continuamente, para decidir el tipo de archivo que mejor cumple con estas necesidades.

5.2 ARHIVOS DE ACCESO SECUENCIAL Y DE ACCESO ALEATORIO

ARCHIVO DE DATOS SECUENCIALES

Un archivo secuencial es simplemente una serie de caracteres sin ningún formato. En el cual para poder realizar una lectura se tiene que hacer siempre desde el primer renglón y avanzar de uno por uno hasta llegar al renglón que se busca.

Para realizar una escritura se tiene que hacer sobre un archivo nuevo, o al final del archivo, nunca en un lugar intermedio.

Para realizar el acceso a un archivo secuencial se tiene que efectuar lo siguiente:

1.- Apertura del archivo.
(de lectura, de escritura, o para agregar datos)

2.- Se almacena o se procesa la información.

3.- Y se cierra el archivo.
(Esto es indispensable para no perder la información)

ARCHIVO DE DATOS ALEATORIOS

Un archivo de acceso aleatorio es un archivo dividido en secciones llamadas registros que permiten cambiar datos fácilmente.

Esta estructura también permite un acceso rápido, ya sea que se encuentren al principio o al final del archivo.

Este tipo de archivo tiene su mayor aplicación en los casos en que los datos cambian frecuentemente.

El uso de este tipo de archivos requiere mayor planeación en su diseño, pero esto ayudará a disminuir el trabajo de programación.

Para realizar un acceso a los archivos aleatorios se deberá hacer lo siguiente:

1.- Abrir el archivo.

2.- Definir el formato del registro de trabajo.

3.- Realizar la operación deseada (escritura y lectura) utilizando el registro definido.

4.- Cerrar el archivo.

Como se puede ver es parecida la secuencia de operación con los archivos secuenciales, sólo que aquí se tiene que definir el registro de trabajo, y que las escrituras y lecturas se puedan realizar al mismo tiempo.

5.3 PROCESAMIENTO DE ARCHIVOS. INSTRUCCIONES PARA EL MANEJO DE ARCHIVOS

Después de que los archivos han sido creados, se gestionan por el sistema operativo de la computadora (MS-DOS, OS/2, UNIX, MPE, ETC).

El sistema operativo almacena y accede a archivos, genera directorios, copia, renombra y transfiere archivos entre discos, memoria y dispositivos de E/S. Esto significa que puede emplear las utilidades de su sistema operativo para manipular un archivo después de que se crea.

Los archivos se pueden almacenar en disquetes, unidades de discos, cintas, discos ópticos, etc.

Las instrucciones para el manejo de archivos tanto en BASIC como en PASCAL son las siguientes:

ARCHIVOS SECUENCIALES EN BASIC

Apertura de un archivo:

Se puede abrir un archivo de tipo secuencial de dos modos que son:

```
OPEN nomarch  FOR  modo  AS  (#) numarch
```

o

```
OPEN  modo2, (#) numarch, nomarch
```

Donde:

nomarch: es el nombre del archivo con el que se almacenará en disco
modo: es uno de los tres modos permitidos:

modo	modo2	Tipo de acceso
OUTPUT	"O"	Salida secuencial
INPUT	"I"	Entrada secuencial
APPEND	"A"	Agregar en forma secuencial

Si un archivo existente se abre para salida, los datos originales serán destruidos y las escrituras siguientes siempre empezarán al principio del archivo borrado o nuevo.

En cambio en modo agregar, las siguientes escrituras se harán sin borrar el archivo existente.

numarch: define un número para distinguir el archivo de los demás archivos abiertos, este número se usa para realizar las lecturas y escrituras.

Ejemplo:

```
200 OPEN "DATOS.SEC" FOR OUTPUT AS 3
```

O

```
200 OPEN "O", #3, "DATOS.SEC"
```

```
300 OPEN "ENTRADA.SEC" FOR INPUT AS #4
```

O

```
300 OPEN "I", #4, "ENTRADA.SEC"
```

ESCRITURA A UN ARCHIVO SECUENCIAL

Después de abrir un archivo secuencial de salida puede utilizar las siguientes instrucciones para escribir en el:

```
PRINT #
PRINT # USING
WRITE #
```

Estas instrucciones funcionan igual que las de escritura a pantalla sólo que aquí la escritura se envía al archivo con el # que lo identifica.

El lugar donde se iniciarán las escrituras en el archivo estará definido por el modo de apertura, (output, al inicio), (append, al final), y a partir de esta posición en forma secuencial.

Ejemplo:

```
110 NOM$ = "HOLA"
120 PRINT #2 , NOM$, I;
```

Escribirá en el archivo: HOLA

```
110 UNO$ = "ENEP"
120 DOS$ = "ARAGON"
130 PRINT #2, UNO$;DOS$
```

Escribirá en el disco: ENEPARAGON

Para evitar que quede pegado se usará:

```
130 PRINT #2, UNO$; " , ";DOS$
```

Así escribira:

```
ENEP , ARAGON
```

O de otra forma:

```
130 WRITE #2, UNO$, DOS$
```

Así automáticamente dividirá estas dos variables.

Escribiendo:

```
"ENEP","ARAGON"
```

Esta última opción es la mejor para escrituras a un archivo secuencial como veremos en la lectura.

LECTURA DE UN ARCHIVO SECUENCIAL

La lectura de datos se puede realizar sobre cualquier archivo que está abierto de entrada (INPUT) y similarmente a las instrucciones de escritura; existen las instrucciones de lectura, las cuales también se parecen a las de lectura de teclado:

```
INPUT #
LINE INPUT #
```

El formato de estas instrucciones es:

```
INPUT  # numarch,  var1  [, var2, ...]
```

Las lecturas a un archivo secuencial siempre se realizan a partir del primer renglón y así sucesivamente hasta llegar al final del archivo.

El tipo de las variables que se van a leer debe corresponder con las que están almacenadas en el archivo o marcará el error "type mismatch" (tipos diferentes).

Para separar los datos almacenados en un archivo y diferenciar uno de otro se utilizan los espacios, las comas, las comillas y los brincos de línea; de esta forma se prefiere que los datos se escriban con el estatuto WRITE #, o con PRINT #, colocando los datos en forma apropiada. Ejemplos:

Si se tiene almacenado / CARLOS,JUAN,PEDRO,JOSE /

Y se ejecuta:

```
100 INPUT #1 , A$ , B$
```

Y si se tuviera : / CARLOS,"JUAN,PEDRO,JOSE" /

Tendríamos resultados diferentes.

La instrucción LINE INPUT # sólo tiene un delimitador, el fin de línea y siempre se lee sobre una variable de tipo string.

Así se pueden leer todos los datos de un archivo sin importar las comillas espacios y comas que tenga por renglón. Su formato es:

```
LINE INPUT  #numarch, varstring
```

USO DE LAS FUNCIONES DE ESTADO DE ARCHIVO SECUENCIALES

Existen tres funciones en BASIC que le indicarán cómo se encuentra en cualquier momento el archivo. Estas son:

```
LOF(numarch)        LOC(numarch)        EOF(numarch)

LOF : mantiene la longitud en bytes (letras), del archivo
      asociado con numarch.

LOC : mantiene un número de posición relativa dentro del
```

```
        archivo en términos de grupos de 128 bytes.

   EOF : nos ayuda a descubrir si hemos llegado al fin del
         archivo o no.
```

CIERRE DE UN ARCHIVO SECUENCIAL

Este paso final en el uso de archivos secuenciales es muy importante, ya que si no se realiza adecuadamente se pueden perder datos, debido a que éstos no se almacenan en disco hasta que se llena un espacio de memoria principal llamado buffer. La instrucción es:

```
CLOSE [ numarch, numarch ]
```

Si no se define un numarch se cierran todos los archivos abiertos, y si se indican, sólo se cerrarán éstos.

También es necesario cerrar un archivo, antes de abrir de otro modo el archivo secuencial. Ejemplo:

```
1.- Se abre el archivo de salida.

2.- Se escriben datos en él.

3.- Se abre el mismo archivo de entrada.
```

En este momento se pueden perder datos; se debe ejecutar antes la instrucción CLOSE.

ARCHIVOS SECUENCIALES EN PASCAL

REGISTROS

Antes de conocer las instrucciones para gestión de archivos en PASCAL, estudiaremos el concepto de archivos y el método de declarar y procesar dichos registros en este lenguaje.

Un registro es un tipo de datos estructurado (denominado dato récord) que consta de un conjunto de elementos que pueden ser del mismo tipo o de tipos diferentes.

Los componentes de un registro se denominan campos. Cada campo tiene un nombre llamado identificador de campo, que es algún identificador elegido por el programador cuando se declara el tipo de registro y un tipo que se especifica cuando se declara el tipo de dato récord.

El formato es:

```
TYPE
      tipo-reg = RECORD
                    lista id1 : tipo 1 ;
                    lista id2 : tipo 2 ;
                    .
                    .
                    .
                    lista idn : tipo n
                 END;
```

Donde:

tipo-reg nombre de la estructura o dato registro.

lista id lista de uno o más nombres de campos separados por comas.

tipo puede ser cualquier tipo de dato estándard o definido por el usuario.

Ejemplo:

```
 TYPE
EMPLEADO  = RECORD
               NOMBRE       : STRING [30];
               EDAD         : INTEGER ;
               DOMICILIO    : STRING [40] ;
               SALARIO      : REAL
            END
```

Estas declaraciones han creado un tipo de datos llamado EMPLEADO con cuatro campos: Nombre, Edad, Domicilio y Salario.

VARIABLES TIPO REGISTRO

Tras utilizar la declaración TYPE se ha creado un tipo de dato registro, pero no se ha creado una variable de ese tipo. Esto se consigue con la declaración VAR y el formato es:

```
VAR
nombrereg : nombretipo
```

Donde:

nombretipo es el mismo que el utilizado en la correspondiente declaración type.

nombrereg nombre de una variable particular registro.

Ejemplo:

```
VAR
 TRABAJADOR  : EMPLEADO ;
```

La variable TRABAJADOR tiene la estructura especificada en la declaración del tipo de registro EMPLEADO y se representa gráficamente así:

Variable empleado

Nombre:	Graciela Jardón Avila
Edad:	29
Domicilio:	1501 No. 150
Salario:	4000.00

Variable Fecha

Mes:	Junio
Día:	27
Año:	1993

Junio	27	1993
Mes	Día	Año

Figura 5.2

ACCESO A LOS CAMPOS DE UN REGISTRO

Se puede acceder a cada campo de un registro directamente utilizando un designador o selector de campo de la forma:

```
Nombrereg.Nombrecampo
```

Los datos mostrados anteriormente en empleado mediante una secuencia de sentencias de asignación:

```
Empleado.Nombre     :=  'JOSE LUIS ARCE'  ;
Empleado.Edad       :=  34 ;
Empleado.Domicilio  := 'BRASILIA 11' ;
Empleado.salario    :=  2656.00 ;
```

Una vez que los datos están almacenados en un registro, se pueden manipular de igual forma que otros datos en memoria. Ejemplo:

```
WRITE (EMPLEADO.NOMBRE)
```

OPERACIONES SOBRE REGISTROS

Los procedimientos de lectura y escritura no permiten más que números, caracteres o cadenas. Un registro al ser una estructura compuesta es preciso efectuar las operaciones de lectura y escritura individualmente.

Otra operación que se puede realizar entre registros es la asignación (copia del contenido de un registro en otro registro del mismo tipo). Si A y B son variables registro del mismo tipo, la sentencia:

```
A  :=   B
```

Copia todos los valores asociados con el registro A al registro B.

Ejemplo:

El siguiente programa define un tipo registro (cliente) y a continuación rellena (pone valores en los campos) en la variable correspondiente. Otra variable del mismo tipo se asigna a la primera variable y los campos de la segunda variable se imprimen uno a uno.

```
PROGRAM  VISREG ;
  TYPE
   DATOS = RECORD
                NOMBRE        : STRING [80] ;
           DIRECCION     : STRING [80] ;
              EDAD          : INTEGER ;
            SALDO         : REAL
              END ;

VAR
      AUX, CLIENTE   : DATOS ;


BEGIN
   (* rellenar campo *)
WRITE ('INTRODUZCA NOMBRE DEL CLIENTE :');
  READLN (CLIENTE.NOMBRE);
CLIENTE.DIRECCION  := 'ARAGON 14'
WRITE ('INTRODUZCA EDAD DEL CLIENTE ') ;
READLN  (CLIENTE.EDAD) ;
CLIENTE.SALDO := 356000
(* transfiere datos al registro aux *)
```

```
    AUX := CLIENTE ;
    (* visualizar los campos de aux *)
    WRITELN ('NOMBRE :      ', AUX.NOMBRE) ;
    WRITELN ('DIRECCION :   ', AUX.DIRECCION) ;
    WRITELN ('EDAD :        ', AUX.EDAD ) ;
    WRITELN ('SALDO :       ', AUX.SALDO )
END.
```

Este método de lectura/escritura campo a campo es engorroso. PASCAL proporciona la sentencia WITH, que facilitará el proceso de lectura/escritura de los registros.

LA SENTENCIA WITH

La tarea de escribir el selector de campo completo cada vez que se referencia un campo de un registro es tedioso, sobre todo si el número de campos es grande.

La sentencia WITH permite referenciar el nombre del registro en su cabecera, y posteriormente para llamar algún campo sólo se necesita el nombre del campo y no el selector de campo completo, con el nombre del registro (por ejemplo, EDAD en lugar de CLIENTE.EDAD). La sentencia WITH sólo ofrece un medio fácil de manipular diferentes campos de la misma variable tipo de registro.

Su formato es:

```
WITH Varregistro  DO
  BEGIN
      sentencias que hacen referencia a campos de
      varregistro
  END
```

Donde:

varregistro	nombre o nombres de registros
sentencias	relacionadas con los campos

Ejemplos:

A).

```
TYPE
        PRUEBA = RECORD
                   F1  : INTEGER  ;
                   F2  : STRING  [8]
    END ;

 VAR
    X  := PRUEBA ;
```

Un modo de asignar valores a un registro y luego visualizarlo con WITH puede ser:

```
WITH   X   DO
   BEGIN
         F1  :=  20 ;
         F2  :=  'JARDON'
   END;
WRITELN ( X.F1, X.F2 )
```

B).

```
WITH empleado  DO
   WRITELN (nombre)
```

Equivale a: WRITELN (EMPLEADO.NOMBRE)

Para poder utilizar un archivo primeramente se tiene que declarar y este paso consta de dos operaciones:

1.- Definir una variable de tipo archivo TEXT.

2.- Asociar a esta variable el nombre de un archivo en disco sentencia ASSING.

Definir una variable de tipo TEXT. El formato es:

A).

```
VAR
      NOMBREVAR  : TEXT  ;
```

B).

```
VAR
      NOMBREVAR  :  FILE OF  CHAR ;
```

Al contrario que otras estructura, la longitud de una variable archivo no se precisa.

ASIGNACION DE ARCHIVOS

La operación de asignar un archivo establece una correspondencia entre variable tipo archivo con un archivo externo situado en un disco. El formato es:

```
ASSIGN ( VAR F: nombre ; string )
```

Llamada al procedimiento:

```
ASSIGN (F, nombre)
```

Donde:

F — nombre interno del archivo por el que se conoce el archivo dentro del programa.

nombre — nombre externo con el que se conoce el archivo por el sistema operativo. Es una cadena que se define como una variable cuyo valor se lee interactivamente; puede ser una unidad: nombre. extensión o bien una unidad con nombres de caminos y archivos.

Ejemplos:

1.

```
CONST
     Nombrearc  =  'program8.txt'
   VAR
   Miarch  : text ;
 BEGIN
   assing  (Miarch, Nombrear ) ;
```

2.

```
 VAR
     Datos  : text ;
   Nombre : String  [20] ;
  ...
    Write (escriba el nombre del archivo externo') ;
    Readln (Nombre) ;
    Assign (Datos, nombre) ;
...
```

APERTURA DE UN ARCHIVO

Después de haber asignado un archivo debe ser abierto. Esta operación se realiza por uno de los dos procedimientos predefinidos: REWRITE Y RESET.

RESET

Abre un nuevo archivo existente para una operación de lectura. Si se intenta llamar a RESET y el archivo especificado no existe, se producirá un error de E/S. Formato:

```
RESET (NombreArch)
```

Donde:

NombreArch variable tipo archivo.

REWRITE

Crea y abre un nuevo archivo. Si el archivo ya existe, REWRITE borra su contenido; en caso contrario, el archivo queda abierto para una operación de escritura. Formato:

```
REWRITE (f)
```

ESCRITURA DE UN ARCHIVO

Una vez que se ha abierto un archivo para escritura, las sentencias (procedimientos) WRITE y WRITELN sirven para escribir datos en el nuevo archivo. El formato es:

```
WRITE (f, v1, v2, ...)
```

Donde:

```
f     variable tipo archivo.
```

```
WRITELN (f1, v1, v2, ... vn)
```

Donde:

```
v1, v2, ...      variables del tipo de datos
```

LECTURA DE UN ARCHIVO

Los procedimientos READ y READLN se utilizan para la lectura de los datos en un archivo de tipo texto. El formato es:

```
READ ( f . v1, v2, ...)          READ (f, v1, v2, ...)
```

Donde:

f variable archivo de texto

v1, v2, ... variable de tipo char, integer, real o string

Ejemplo:

```
 VAR
Total, Horas   : real ;
Archivo        : text ;
Mensaje        : string [30] ;
 BEGIN
Assing ( Archivo, 'Demo') ;
Reset  ( Archivo ) ;
...
Readln ( archivo, mensaje, Horas) ;
```

FUNCIONES EOLN/EOF

La función EOLN devuelve el estado de fin de línea de un archivo. Es una función de tipo lógico. Formato:

```
EOLN [(VAR f: text)]
```

Llamada a la función: EOLN(f)

Donde:

```
f  variable de archivo de texto.
```

EOLN(f) devuelve true si en la posición actual del archivo está la marca fin de línea, o bien si EOF(f) es true; en caso contrario devuelve false.

La función EOF, fin de archivo, devuelve el estado de un archivo de texto. Es una función de tipo lógico que indica si el fin de archivo se ha almacenado; devuelve true si se ha almacenado, false en caso contrario.

AÑADIR DATOS A UN ARCHIVO

El procedimiento APPEND abre un archivo existente para añadir datos al final del mismo. El formato es:

```
APPEND (f)
```

Donde:

(f) variable de archivo de texto que debe haber sido asociada con un archivo externo mediante assign.

Si el archivo no existe, se produce un error, y si ya estaba abierto, primero se cierra y luego se reabre.

Ejemplo:

```
  PROGRAM Agregar ;
VAR
  Fich : Text ;
  BEGIN
  Assign   (Fich, 'texto.txt') ;
    Rewrite  (Fich) ;
    Writeln  (Fich, 'línea primitiva') ;
    Close    (Fich);

  Append   (Fich)  ;
    Writeln  (Fich, 'línea añadida') ;
    Close    (Fich)
END.
```

ARCHIVOS ALEATORIOS EN BASIC

Al igual que en los archivos secuenciales la apertura se puede realizar de dos formas, pero este tipo de archivos sólo tienen un modo de operación. Y se realiza de la siguiente manera:

```
OPEN nomarch AS  numarch [ LEN = longreg ]
```

o

```
OPEN "R" , [#] numarch, nomarch [, longreg ]
```

Como se puede observar no se tiene que especificar si es de entrada o de salida, ya que se pueden realizar las dos operaciones sobre este tipo de archivos. Ejemplo:

```
120 OPEN "R" , 3 , "CLIENTES.DAT"
```

O

```
120 OPEN "CLIENTES.DAT" AS 3
```

De esta forma la longitud de los archivos siempre será de 128 bytes.

Para utilizar de una mejor manera los archivos se puede definir la longitud de los registros y no desperdiciar espacio de disco.

```
120 OPEN "R" , 2, "ARTICULOS" , 48
```

O

```
120 OPEN "ARTICULOS" AS 2 LEN = 48
```

DEFINICION DEL FORMATO DEL REGISTRO

El segundo paso después de abrir el archivo es "asignar campo" a los buffer (s) que usará el programa; es decir, definir los datos dentro del registro junto con su longitud. Esto se hace utilizando la instrucción: FIELD.

Ejemplo:

CAMPO	LONGITUD
fecha	6
nombre	20
dirección	20
saldo cuenta	8

Entonces el estatuto sería:

```
130 FIELD 3, 6 AS FECHA$, 20 AS NOM$, 20 AS DIRE$, 8 AS SAL$
```

El primer número es el que distingue a este archivo y las siguientes parejas de números indican (long. as nomcampo).

Estos nombres de campos son los que van a realizar las escrituras y lecturas.

ESCRITURA DE DATOS A MEMORIA INTERMEDIA

Después de definir el formato del registro se tienen que utilizar las siguientes instrucciones LSET Y RSET para colocar los datos que se desean almacenar posteriormente en disco.

El formato es:

```
LSET nomcampo = datostr
```

o

```
RSET nomcampo = datostr
```

Si la longitud datostr es de menor longitud que nomcampo previamente definido en FIELD, se completan con blancos al final en LSET, y al inicio en RSET.

Por el contrario si es de mayor longitud se eliminarán las letras que sobren al final en LSET, o las del inicio con RSET.

Ejemplo:

Se tiene la siguiente definición de registro de un archivo:

```
210 OPEN "INVENT" AS 1
220 FIELD 1, 6 AS NUM$, 20 AS DESC$
```

Y se asigna lo siguiente:

```
230  LSET DESC$ = "TABLERO BASICO"
```

Esto se almacenará así:

```
/TABLERO BASICO  ...../
```

De otra forma si se tuviera:

```
230 RSET DESC$  =  "TABLERO BASICO"
```

Se almacenaría así:

```
/..... TABLERO BASICO/
```

Cambiando el valor de asignación sucedería lo siguiente:

```
230 LSET DESC$ = "TABLERO DE MEMORIA DE LA COMPUTADORA"
```

Quedaría así:

```
/TABLERO DE MEMORIA D/
```

O

```
230 RSET DESC$ = "TABLERO DE MEMORIA DE LA COMPUTADORA"
```

Quedaría así:

```
/IA DE LA COMPUTADORA/
```

CONVERSION DE NUMEROS EN CADENAS

Debido a que los campos que nos ayudan para escribir la información en el disco son de tipo string (letras), si se desea almacenar datos numéricos es necesario cambiarlos de tipo numérico a string.

Este proceso se hace con las siguientes funciones:

```
MKI$  : Convierte un valor entero a un string de 2 bytes
        (2 letras).

MKS$  : Convierte un valor de precisión sencilla en un
        string (de 4 letras).

MKD$  : Convierte un valor de doble precisión en un string
        de (8 letras).
```

Por el contrario cuando lea del archivo se tendrán datos tipo string que se desean convertir a números; esto se hace así:

```
CVI  : STR(2)   A  valor entero.

CVS  : STR(4)   A  valor de precisión simple.

CVD  : STR(8)   A  valor de doble precisión.
```

ESCRITURA DE REGISTROS EN UN ARCHIVO ALEATORIO

Después de construir completamente los registros (después de haber asignado todos los campos con los datos a escribir), el registro se envía a disco con la siguiente instrucción:

```
PUT [#] numarch  [, número de registro ]
```

El parámetro numarch es el que se le asignó con OPEN.

Cuando no se especifica el número de registro la escritura se hará en el lugar en donde esté el apuntador del archivo. Si se especifica un número, en ese número, se hará la escritura.

Ejemplo:

```
300  PUT # 3, 20
```

LECTURA DE ARCHIVOS ALEATORIOS

Para recuperar los registros previamente almacenados en un archivo aleatorio, se utiliza la instrucción GET, con formato:

```
GET [#] numarch [,número de registro]
```

El parámetro numarch es el definido en OPEN y el número de registro, si no se especifica, se utilizará en donde está el apuntador del archivo, y si se especifica, ese es el que se leerá.

Los datos leídos quedarán en el registro definido en FIELD.

Ejemplo:

```
300 GET #2, 10
```

Esto traerá al registro, los datos almacenados, en el décimo registro en el disco.

Una siguiente lectura traería a registro los datos del onceavo registro.

Después de hacer un GET ya se pueden utilizar los valores que tienen los campos del registro para la actividad deseada.

ARCHIVOS DE ACCESO ALEATORIO EN PASCAL

Los archivos aleatorios están constituidos en elementos o registros cuyo tipo puede ser cualquiera. A los elementos de estos archivos se accede directamente, al no situarse éstos en posiciones físicamente consecutivas, sino en posiciones lógicas. Esta es la razón por la que se les denomina archivos de acceso aleatorio o directo.

Los elementos de los archivos aleatorios son de igual tamaño y el término acceso directo significa que es posible acceder directamente a no importa cúal elemento con sólo especificar su posición. Un archivo de acceso aleatorio se almacena, en formato binario comprimido, por lo que no es visualizable en pantalla.

El proceso de operaciones con archivos con tipo o acceso aleatorio es similar al ya tratado en los archivos secuenciales, con algunas diferencias que estudiaremos.

Las etapas necesarias para la creación, lectura y escritura de un archivo aleatorio son:

1.- Definir el tipo de componentes del archivo y declarar una variable de archivo como escritura de tipo file (declaraciones TYPE y VAR).

2.- Establecer un enlace entre la variable tipo archivo y el nombre del archivo en disco (procedimiento ASSING).

3.- Abrir un archivo nuevo o uno ya existente (procedimiento REWRITE o RESET). En cualquier caso, el archivo se abre tanto para lectura como para escritura.

4.- Marcar una posición específica en el archivo para la siguiente operación de lectura o escritura (SEEK). Leer el registro (READ) o escribir un nuevo registro en esa posición (WRITE).

5.- Cerrar el archivo al terminar las operaciones de lectura/escritura (CLOSE).

DECLARACION DE UN TIPO DE ARCHIVO

La declaración de un archivo de acceso aleatorio se efectúa con la ayuda de las palabras reservadas FILE OF. Ejemplos:

1.-

```
TYPE   CLIENTE  = RECORD
                      CODIGO       : STRING [10] ;
                           NOMBRE       : STRING [40] ;
                           DIRECCION    : STRING [30] ;
                           SALDO        : REAL ;
                           EDAD         : INTEGER
                       END;
  VAR
    FICH_CLIENTE     : FILE OF   CLIENTE  ;
    UN_CLIENTE       : CLIENTE  ;
    CAR              : CHAR ;
```

2.-

```
TYPE  LISTA  =  ARRAY  [0...10] OF INTEGER  ;
      ITEM   =  RECORD
                  NOMBRE  : STRING  [20] ;
                     CODIGO  : INTEGER ;
                  PRECIO  : REAL
                END;
TIPOFICH1  = FILE OF  REAL ;
TIPOFICH2  = FILE OF LISTA ;
TIPOFICH3  = FILE OF ITEM;
```

ASIGNACION DE ARCHIVO

El procedimiento de asignación es idéntico al ya conocido.

```
ASSING (NOMBRE_ARCHIVO _INTERNO, NOMBRE_ARCHIVO_EXTERNO)
```

Ejemplo:

```
ASSING (FICH_CLIENTE, 'CLIENTE.DAT')
```

APERTURA DEL ARCHIVO

Esta operación se efectúa por uno de los procedimientos ya conocidos: **RESET, REWRITE.**

RESET	necesita la existencia del archivo.
REWRITE	crea y abre un archivo; si ya existía, su contenido se destruye.

OPERACIONES DE LECTURA, ESCRITURA Y FIN DE ARCHIVO

Los archivos aleatorios no están constituidos por líneas; en consecuencia, los procedimientos standard READLN, WRITELN y EON no se pueden utilizar en archivos con tipo.

Los únicos procedimientos que se pueden utilizar para lectura y escritura son:

```
READ        WRITE        EOF
```

Los formatos respectivos son:

```
READ (nombre_de_fichero, lista de elementos)
WRITE (nombre_de_fichero, lista de elementos)
EOF (nombre_de_fichero )
```

En cada operación de lectura y escritura el puntero asociado al archivo se incrementa automáticamente de modo que se posiciona en el registro siguiente.

CIERRE DE ARCHIVO

El cierre de un archivo con tipo se efectúa de igual modo que para un archivo de texto.

```
CLOSE (nombre_del _archivo)
```

MANTENIMIENTO DE ARCHIVOS ALEATORIOS

Las operaciones típicas en el mantenimiento de archivos son:

- Creación
- Acceso
- Consulta
- Actualizar

CREACION DE UN ARCHIVO ALEATORIO

Crear un archivo de registro de direcciones:

```
 PROGRAM CREAR ;

TYPE
  DIREC = RECORD
    NOMBRE : STRING [30] ;
       CALLE  : STRING [25] ;
       CIUDAD : STRING [15] ;
       TELEFONO: STRING [10]
 END;
ARCHIVO = FILE OF DIREC;

 VAR
   DIR      : ARCHIVO ;
   ELEMENTO : DIREC ;
RESPUESTA : CHAR ;

 BEGIN
   ASSING (DIR, 'AGENDA.DAT') ;(* asigna dir a AGENDA.DAT *)
 REWRITE (DIR)                  (* abre el archivo dir *)
 WITH ELEMENTO DO
    REPEAT
    WRITE ('NOMBRE :');
       READLN(NOMBRE);
```

```
        WRITE ('CALLE: ');
        READLN(CALLE) ;
        WRITE ('CIUDAD :') ;
        READLN (CIUDAD) ;
        WRITE ('TELEFONO : ');
        READLN (TELEFONO) ;
        WRITE (DIR,ELEMENTO) ;     (* ESCRITURA DE UN REGISTRO *)
        WRITELN ('DESEA OTRO ELEMENTO S/N');
        RESPUESTA : UPCASE (READKEY)
      UNTIL RESPUESTA = 'N' ;
       CLOSE (DIR)
END.
```

ACCESO ALEATORIO A UN ARCHIVO

Al igual que con otras estructuras de datos, un archivo, con frecuencia se debe modificar o actualizar después de haber sido creado. Se puede desear modificar uno o más campos de un registro existente, borrar un registro, insertar un nuevo registro o simplemente visualizar los valores actuales de los campos de un registro.

PROCEDIMIENTO SEEK

Sitúa el puntero de posición del archivo en el número de registro correspondiente. Formato:

```
SEEK (nombrearchivo, numreg)
```

Donde:

nombrearchivo	nombre del archivo (no puede ser secuencial)
numreg	número de posición del registro, sobre la base de que el primer registro es cero.

Ejemplo;

```
SEEK (INVENTARIO, 15)
```

Mueve el puntero de posición del archivo al registro 15, que en realidad es el decimosexto registro.

```
POSITION(F)    Y MAXPRO(F)
```

Estas funciones sólo pueden usarse con archivos aleatorios.

POSITION(F) regresa el valor entero de la posición del puntero inmediatamente después de utilizar la sentencia SEEK.

Ejemplo:

```
SEEK(F,K)
POSITION(F)
```

La función Position regresar el valor de k.

La función Maxpos(f) regresa el valor entero del último registro utilizado.

AGREGAR REGISTROS

Para poder añadir datos a un archivo es preciso que éste exista; es decir, se necesita comprobar su existencia; si no existe el archivo, la operación única a realizar será escribir en él y no añadir datos. Para añadir datos se detecta el final del archivo con la función MAXPOS(F) y luego posicionarle en ese punto con el procedimiento SEEK. Ejemplo:

```
SEEK (nombre_arch, MAXPOS(nombrearch)) ;
```

Sitúa el puntero del archivo detrás del último elemento existente, por lo que basta con escribir en el archivo para añadir un elemento.

```
SEEK (nombre_archivo, MAXPOS(nombre_archivo));
WRITE (nombre_arch, articulo);
```

A continuación mostraremos 2 programas en lenguaje PASCAL los que fueron corridos en la HP 1000.

PROGRAMA 1

```
PROGRAMA GENERACIONDEACTAS (INPUT, OUTPUT);
(*GRACIELA JARDON AVILA     *)
(*JOSE LUIS ARCE GARCIA     *)
VAR NOMBRE : STRING[30];
   I, NUMALUMNOS : INTEGER;
   NOTA, SUMA, MEDA, SUMEDIA, MEDIAGRUPO : REAL;
BEGIN (*PP*)
   WRITELN('         LISTADO DE CALIFICACIONES');
   WRITELN('         -------------------------');
   WRITELN();
```

```
    WRITELN('NOMBRE DEL ALUMNO            NOTA MEDIA');
    WRITELN('-----------------            ----------');
    NUMALUMNOS :=  0;
    SUMAMEDIA :=  0;
    WRITELN('ESCRIBA NOMBRE DEL ALUMNO O FIN');
    READLN(NOMBRE);
    WHILE NOMBRE <> 'FIN' DO
       BEGIN (*1*)
          NUMALUMNOS  := NUMALUMNOS * 1;
          SUMA : = 0;
          FOR I := 1 TO 6 DO
             BEGIN (*2*)
                WRITELN('ESCRIBA NOTA NUMERO' I);
                READLN(NOTA);
                SUMA := SUMA + NOTA;
             END; (*2*)
          MEDIA := SUMA / 6;
          SUMAMEDIA := SUMAMEDIA + MEDIA;
          WRITELN(NOMBRE,MEDIA);
          WRITELN('ESCRIBA EL NOMBRE DEL ALUMNO O FIN');
          READLN(NOMBRE);
       END; (*1*)
    MEDIAGRUPO := SUMAMEDIA / NUMALUMNOS;
    WRITELN();
    WRITELN('NOTA MEDIA DEL GRUPO :', MEDIA GRUPO);
END. (*PP*)
```

```
    LISTADO DE CALIFICACIONES
    -------------------------

NOMBRE DEL ALUMNO           NOTA MEDIA
-----------------           ----------
ESCRIBA NOMBRE DEL ALUMNO O FIN
ESCRIBA NOTA NUMERO            1
ESCRIBA NOTA NUMERO            2
ESCRIBA NOTA NUMERO            3
ESCRIBA NOTA NUMERO            4
ESCRIBA NOTA NUMERO            5
ESCRIBA NOTA NUMERO            6
LUIS 9.333334E+00
ESCRIBA NOMBRE DEL ALUMNO O FIN
ESCRIBA NOTA NUMERO            1
ESCRIBA NOTA NUMERO            2
ESCRIBA NOTA NUMERO            3
ESCRIBA NOTA NUMERO            4
ESCRIBA NOTA NUMERO            5
ESCRIBA NOTA NUMERO            6
ROGELIO 9.000000E+00
ESCRIBA NOMBRE DEL ALUMNO O FIN

NOTA MEDIA DEL GRUPO  : 9.166668E+00
```

PROGRAMA 2

```
PROGRAM COMIDA (INPUT,OUTPUT)
(*GRACIELA JARDON AVILA     *)
(*JOSE LUIS ARCE GARCIA     *)
TYPE
   COMIDA = (ZANAHORIA, PAPA, JITOMATE, AGUACATE, MANZANA, PLATANO, NARANJA,
             MENUDO, POZOLE, TACOS, MOLE, FRESA, DURAZNO, MANGO, UVAS,
             FILETE, CARNITAS, APIO, ESPINACAS, PIZZA, TAMALES);
   ALIMENTOS = SET OF COMIDA;
```

```
VAR
   A : INTEGER;
   VEGETALES, FRUTAS, GUISOS : ALIMENTOS
   ELEMENTO                  : COMIDA;BEGIN
WRITELN ('  ';8,'CLASIFICACION DE COMIDA');
WRITELN;
VEGETALES ;= [ZANAHORIA, PAPA, JITOMATE, AGUACATE, APIO, ESPINACAS];
FRUTAS ;= [ MANZANA, PLATANO, NARANJA, FRESA, DURAZNO, MANGO, UVAS ];
GUISOS ;= [ MENUDO, POZOLE, TACOS, MOLE, FILETE, CARNITAS, PIZZA, TAMALES);
WRITELN('TIPO      ','  VEGETALES  ','      FRUTAS    ','     GUISOS');
FOR ELEMENTO ;=ZANAHORIA TO TAMALES DO
   BEGIN
   CASE ELEMENTO OF
   ZANAHORIA, ; WRITE( 'ZANAHORIA');
   PAPA,      ; WRITE( 'PAPA     ');
   JITOMATE,  ; WRITE( 'JITOMATE ');
   AGUACATE,  ; WRITE( 'AGUACATE ');
   MANZANA,   ; WRITE( 'MANZANA  ');
   PLATANO,   ; WRITE( 'PLATANO  ');
   NARANJA,   ; WRITE( 'NARANJA  ');
   MENUDO,    ; WRITE( 'MENUDO   ');
   POZOLE,    ; WRITE( 'POZOLE   ');
   TACOS,     ; WRITE( 'TACOS    ');
   MOLE,      ; WRITE( 'MOLE     ');
   FRESA,     ; WRITE( 'FRESA    ');
   DURAZNO,   ; WRITE( 'DURAZNO  ');
   MANGO,     ; WRITE( 'MANGO    ');
   UVAS,      ; WRITE( 'UVAS     ');
   FILETE,    ; WRITE( 'FILETE   ');
   CARNITAS,  ; WRITE( 'CARNITAS ');
   APIO,      ; WRITE( 'APIO     ');
   ESPINACAS, ; WRITE( 'ESPINACAS');
   PIZZA,     ; WRITE( 'PIZZA    ');
   TAMALES    ; WRITE( 'TAMALES  ');
END;
(*                                                                    *)
(* A CONTINUACION SE HACE LA CLASIFICACION, ESCRIBIENDO V(VERDADERO))
   CUANDO EL ALIMENTO PERTENECE AL CONJUNTO INDICANDO O UNA F (FALSO)
   CUANDO NO SEA ELEMENTO DE DICHO CONJUNTO.                        *)
(*                                                                    *)
IF ELEMENTO IN VEGETALES THEN
   WRITE (' V  ','       ','       ';5)
ELSE
   WRITE (' F  ','       ','       ';5)
IF ELEMENTO IN FRUTAS THEN
   WRITE ('    ','   V   ','       ';5)
ELSE
   WRITE ('    ','   F   ','       ';5)
IF ELEMENTO IN GUISOS THEN
   WRITE (' V  ','       ','   V   ';5)
ELSE
   WRITE (' F  ','       ','   F   ';5)
END;
(* END FOR *);
END.
```

CLASIFICACION DE COMIDA

TIPO	VEGETALES	FRUTAS	GUISOS
ZANAHORIA	V	F	F
PAPA	V	F	F
JITOMATE	V	F	F
AGUACATE	V	F	F
MANZANA	F	V	F
PLATANO	F	V	F
NARANJA	F	V	F
MENUDO	F	F	V
POZOLE	F	F	V
TACOS	F	F	V
MOLE	F	F	V
FRESA	F	V	F
DURAZNO	F	V	F
MANGO	F	V	F
UVAS	F	V	F
FILETE	F	F	V
CARNITAS	F	F	V
APIO	V	F	F
ESPINACAS	V	F	F
PIZZA	F	F	V
TAMALES	F	F	V

CAPITULO VI

INTRODUCCION A LOS PAQUETES DE USO COMUN

En la actualidad existe una gran variedad de paquetes que ayudan en infinidad de tareas; en este capítulo mencionaremos algunos de estos paquetes describiendo en forma general su aplicación:

6.1 PAQUETES DE DIBUJO Y DISEÑO

Seguramente usted habrá visto las iniciales CAD del inglés Computer Aid Design que significan Diseño Ayudado por Computadora, una aplicación que se empezó a desarrollar hace varios años y cuyo objetivo original fue auxiliarse de la computadora para realizar cualquier tipo de diseño, no sólo el arquitectónico o mecánico.

Otra de las aplicaciones que se ha popularizado en el ámbito de los CADS es el Dibujo Ayudado por computadora. A continuación mencionaremos algunos de estos paquetes.

AUTOCAD

El paquete AUTOCAD es un paquete con funciones completas, que pueden ser adaptadas a las exigencias particulares del usuario. Su potencia, la ventaja de que a su vez es programable y su flexibilidad para que se le incorporen muchos cientos de productos afines, hacen que se le pueda considerar como un programa de norma.

Su eficiencia para trabajos en dos dimensiones es excepcional. Incluso es muy recomendable para la confección de planos arquitéctonicos en dos dimensiones y desde luego, para dibujos de ingeniería en general. El control de su escala permite aumentar o disminuir a voluntad el dibujo terminado. Asimismo, las dimensiones de lo que se ha dibujado se pueden obtener automáticamente de forma directa de la imAgen proyectada en el monitor.

Aunque el AUTOCAD ofrece un número limitado de funciones tridimensionales, el mismo no es un programa CAD completo para trabajar en tres dimensiones; en esta modalidad se pueden representar volúmenes que tengan bien definidas sus tres coordenadas (x, y, z). Es decir, se puede trabajar con un volumen que cuente con tres dimensiones.

Fabricante Autodesk, Inc.

CADKEY

Primordialmente el CADKEY ha sido desarrollado para los ingenieros mecánicos, pero es suficientemente rico en funciones para que se le pueda utilizar en cualquier rama del diseño de ingeniería y arquitectura.

Este es un paquete para el dibujo técnico en dos dimensiones, pero que particularmente se distingue por el gran auxilio que presta en el diseño y el dibujo tridimensional. Incluso al trabajarlo en el formato bidimensional, permite que cualquier perfil se convierta en tridimensional si le incorpora la tercera coordenada.

Cuando al programa se le incorpora el producto AUTOSHADE, los objetos tridimensionales que se ven proyectados con líneas continuas adquieren una apariencia real con primeros planos, cambios de tonos y sombras (la fuente de iluminación la sitúa el usuario a voluntad y la puede ir cambiando sucesivamente para buscar los mejores efectos plásticos).

Fabricante : Cadkey Inc.

DRAWBASE

Es un paquete diseñado específicamente para el diseño y dibujo arquitectónico. El paquete Drawbase es para la oficina profesional y no para el usuario ocasional.

Asimismo, de forma integral el paquete cuenta con una base de información y funciones asociadas en lo relacionado con la construcción, que permiten la confección de listas de materiales, presupuestos, especificaciones, etc.

El trabajar el paquete demanda que el usuario continuamente cambie la vista de la tableta digitalizadora hacia el monitor durante el proceso creativo, pues exclusivamente se utiliza el teclado de la PC para incorporar texto o acotaciones numéricas. La operación para archivar es sumamente engorrosa, puesto que cada dibujo tiene que ser enviado a cinco archivos. Adicionalmente, cada dibujo que tiene que ser archivado debe estar proyectado en la pantalla durante la duración de todo el proceso.

Adicionalmente, para archivar es necesario que todos los dibujos sean convertidos a la escala de 1, aunque no suele ser precisamente la escala a la que se imprimirán los dibujos posteriormente. Pero a pesar de estos inconvenientes Drawbase es una valiosísima herramienta para el diseño arquitéctonico.

Fabricante : Skok System Inc.

AEGIS DRAW PLUS

Un paquete de uso múltiple con una resolución de 640 x 400 puntos y 16 colores simultáneos en pantalla.

Este paquete cuenta con una retícula que permite generar un diseño con mayor facilidad de lo acostumbrado, modulable según los requerimientos de cada caso; asimismo, tiene un juego de

reglas pautadas según las unidades de medidas existentes, desde las unidades decimales a las inglesas, y con una serie de herramientas - líneas, curvas, círculos, etc. - que le brinda la posibilidad de realizar un dibujo preciso.

En lo que se refiere al nivel de acercamiento, este programa efectúa ampliaciones milimétricas de cada sección del dibujo, posibilitando la observación con absoluto detalle y el trazado de nuevos elementos en un área mínima.

Fabricante : Aegis Development.

DYNAMIC CAD

Es un paquete muy poderoso usado tanto en aplicaciones de ingeniería eléctrica como en arquitectura. Su capacidad para ubicar texto en el diseño, ajustándolo al tamaño necesario es excelente así como su manejo de funciones isométricas, ideales para dibujos de tipo mecánico (presentación en forma automática de un objeto desde diferentes puntos de vista).

Este paquete tiene una biblioteca con todos los elementos gráficos que utilizan los ingenieros en electrónica; el único inconveniente radica en que para solicitarlo es necesario conocer un código alfanumérico que corresponde a cada elemento, lo cual obliga a recurrir continuamente al manual en busca de esos códigos, sin teclear también se puede solicitar el dibujo a escala de esos elementos. Este paquete cuenta con herramientas de dibujo de precisión (líneas, curvas, círculos, etc.).

Fabricante : Microlllusions Inc.

PCLO

Está orientado, fundamentalmente, a la creación de planos para circuitos impresos de tipo electrónico, por lo que su aplicación se destina a ingenieros que deseen lograr tales planos con el fin de fotografiarlos y mediante el proceso correspondiente, definir las placas de circuitos impresos.

Cada una de las herramientas con que cuenta el programa, se presenta en diez grosores distintos, lo que aumenta las posibilidades de precisión para el trazado; complementariamente, también se incluyen puntos de montaje, de soldadura, patrones de tierra, nudos y agujeros.

El PCLO genera un número ilimitado de planos sin peligro de saturación, pero sólo dos de ellos se despliegan en pantalla al mismo tiempo. Se tiene la posibilidad de almacenar hasta nueve planos en un solo disco.

Otra característica muy importante de este paquete es la rutina que genera en forma automática las vías de conexión entre puntos, sobre la base de un trazo de líneas paralelas, y que logra delimitar el área en la que se debe generar el dibujo.

Fabricante : Softcircuits.

LOGIC WORKS

Este paquete permite diseñar y probar circuitos lógicos electrónicos, así como generar un plano que ocupe un área de hasta 12 metros cuadrados. Con el Logic Works es factible emplear una serie de segmentos de línea, otra de dispositivos predefinidos; también puede simular la operación real del circuito: se simula incluso la alimentación de dicho circuito con algun valor hexadecimal, con switches alternos o con un reloj de pulso, se obtiene simultáneamente una gráfica que define la salida del circuito y los puntos de retroalimentación.

Complementariamente este paquete contiene las funciones características de corte, copiado e inserción, para editar el circuito en la forma deseada. También tiene un gran número de compuertas lógicas (NOT, AND, OR, NAND etc.) y con los flip-flops D y J, aún de contadores, registro, multiplexores, decodificadores y relojes de pulso. Así el usuario puede crear sus propios circuitos y bibliotecas de circuitos.

Fabricante : Capilano Computers System.

6.2 PAQUETES PROCESADORES DE PALABRAS Y PAQUETES TIPOGRAFICOS.

Nadie ha estimado el número de palabras que la computadora promedio procesa en su vida, pero la cantidad pudiera estar fácilmente en los billones. En la era de las interfases gráficas y de las hojas de cálculo tridimensionales, el procesamiento de texto es todavía la función principal de la mayoría de las computadoras personales, y el mercado más competitivo del software para PC.

Hace algunos años, los analistas auguraron que el advenimiento del procesador de texto iría seguido por la oficina sin papeles. En lugar de esto, la computadora ha llevado directamente al mayor aumento de consumo de papel desde los principios de la impresión.

A medida que los procesadores de palabra aumentan en potencia y flexibilidad, han hecho posible que se impriman documentos con una complejidad y elegancia que solamente estaba disponible en las imprentas profesionales no hace mucho tiempo. Y también han hecho posible dar formato a documentos del tamaño de libros en cuestón de minutos. Pero la más reciente generación de procesadores de texto puede hacer algo más. Estos le permiten crear un documento automátizado del tipo nunca antes posible. Un documento automátizado puede recolectar datos de una hoja de cálculo o base de datos, transformar los datos de una gráfica, y luego emitir comandos que harán que el documento se imprima a sí mismo o que se envíe a sí mismo, mediante el correo electrónico, a una computadora remota para un procesamiento automático adicional.

Los principales protagonistas de este revolucionario fenómeno son firmas de reconocido prestigio dentro de la industria de la computación: La WordPerfect con su avanzado program WORKPERFECT en la version 5.0 ; la firma Microsof Corp. con el suyo WORD en su version 4.0, la Lotus Development Corp. con su correspondiente programa MANUSCRIP y la Micro Internacional con sus respectivos programs WORDSTAR 5.0 y WORDSTAR 2000, la Aldus Corp. con el fabuloso PAGEMAKER y la Xerox con su impresionante VENTURA PUBLISHer. Asimismo, la conocida firma Apple Macintosh sigue desarrollando su propio grupo de estos

avanzados programas.

Dentro de los procesadores de texto más conocidos tenemos:

AMI PROFESSIONAL

El AMI Professional version 2.0 de Lotus Word Processing, contiene todas las propiedades de un procesador de palabra poderoso y completo, tales como el uso de diferentes tipos de letras, estilos, tamaños, espacios de interlíneas y alineamiento de párrafos.

Además incluye la posibilidad de establecer la fecha como referencia y ligar texto de un documento, así como deshacer los últimos cambios de edición.

Como parte del ambiente Windows es posible ligar documentos, así como copiar información, tablas o imagenes, dibujos y editar gráficas dentro del documento y hasta permite, sin necesidad de utilizar otro programa, crearlas e integrarlas.

MICROSOFT WORD

Microsoft word versión 5.0 de Microsoft Corp., es considerado el mejor procesador de texto basado en caracteres para crear hojas de estilo y formatos repetitivos, y tiene un sólido pero complejo apoyo para compendios plegables.

MICROSOFT WORD FOR WINDOWS

Microsoft word for windows versión 1.1 de Microsoft Corp., es un procesador de texto excepcionalmente fuerte y tiene un innovador grupo de herramientas de macro y fusión para automátizar el control sobre la información, tiene una compatibilidad total con Windows 3.0.

WORD PERFECT

El Word Perfect versión 5.1 de Wordperfect Corp., tiene más características y conveniencias que cualquier otro procesador de texto, y su interfaz opcional basada en menús también lo hace uno de los más fáciles de usar. No ofrece compendios plegables, y sus hojas de estilo no están bien implementadas, pero casi todo lo demás es de primera clase, incluyendo una fusión de correspondencia totalmente programable y enlaces a procesamiento automático de documentos y correo electrónico mediante el programa Word Perfect Office.

VENTURA

Ventura es un paquete de diseño muy popular, tiene gran capacidad de manejo de textos y gráficos. Está pensado para ser una herramienta muy completa tanto en la elaboración de grandes publicaciones de 200 ó 300 páginas como para hacer originales de publicidad, circulares, boletines, etc.

6.3 HOJAS ELECTRONICAS

Una hoja electrónica de cálculo se asemeja a una hoja de trabajo de contabilidad y está compuesta por un conjunto de renglones y columnas. A la intersección de un renglón con una columna se le llama celda o casilla.

Es posible almacenar, manipular, calcular y analizar datos tales como números, texto y fórmulas en una hoja de cálculo. Los gráficos se pueden agregar directamente a la hoja de cálculo. Elementos gráficos como líneas, rectángulos, cuadros de texto y botones, se pueden agregar a las hojas de cálculo modernas.

Entre las hojas de cálculo más populares se encuentran:

FRAMEWORK

Es un paquete integrado y se compone de procesador de textos, hoja de cálculo, base de datos, gráficas y comunicaciones. Incorpora un lenguaje de programación propio FRED, lo que lo hace adaptable a un gran número de necesidades.

EXEL

Exel es uno de los primeros programas de una nueva generación de paquetes diseñados para operar bajo ambiente windows. Y es la aplicación más flexible, más atractiva y poderosa en hoja de cálculo que existe en el mercado.

LOTUS

Lotus 1-2-3 para DOS versión 3.4 de Lotus Develepmen, utiliza más de 90 iconos inteligentes, el usuario podrá incrementar su productividad obteniendo un acceso fácil y rápido a los comandos más usados, como son los de impresión, fechas, macros, gráficos en 3a. dimensión, tipo pastel, etc.

SYMPHONY

Symphony, hoja de cálculo, gráficos y macros, es un programa de gran utilidad para analistas financieros, contadores, directivos de pequeñas empresas, ingenieros, secretarias, etc., pues combina la potencia y la sencillez de manejo. En realidad Symphony no es sólo un programa, sino cinco programas especializados, en forma de un paquete integrado. Los cinco ambientes operativos de que consta son: Hoja de Cálculo, Gestor de bases de datos, procesador de textos, gráficos y comunicaciones. Además dispone de un lenguaje de programación.

6.4 MANEJADOR DE BASES DE DATOS

Una base de datos es un conjunto de información organizada para facilitar su consulta. Por ejemplo, una guía de teléfonos, un horario de trenes, registros de propiedad inmobiliaria e

inventarios son bases de datos.

Antes de que existiera la posibilidad de procesar datos de forma electrónica, las bases de datos se plasmaban en papel guardando en carpetas, que a su vez, se guardaban en armarios. Los ficheros debían extraerse de los armarios, era necesario rellenar formularios y los papeles corrían el riesgo de mezclarse.

Con las paquetes de bases de datos, todas las actividades pueden realizarse electrónicamente, de forma que la información puede archivarse y recuperarse con presición y agilidad.

Dentro de los paquetes de bases de datos más conocidos tenemos:

DBASE

Es un sistema de gestión de bases de datos, una potente herramienta para controlar la información. Sus programas permiten almacenar, relacionar, manipular y recuperar volúmenes importantes de información rápida y efectivamente.

El lenguaje de dDASE puede considerarse como el estándar para la programación de base de datos en la PC, lo que hace atractivo para los que consideren la reducción o integración de sus bases de datos de sistemas grandes. Sin embargo, existen problemas para la ejecución de multiusuarios de dBASE IV lo que convierte a este paquete en una pobre elección para las aplicaciones de multiusuarios grandes.

FOXPRO

FoxPro 2.0 aparece como el manejador de bases de datos más veloz en el manejo de archivos índices. Contiene una calculadora, agenda, ejecuta programas, etc.

Otros paquetes de este tipo son:

- Versión 3.5 de Paradox de Borland International Inc;
- Data Ease versión 4.2 de Data Ease International Inc;
- Knowledge Man versión 3.0 de Micro Data Ease System Inc.;
- R:Base versión 3.1c de Microrim;
- Superbase 4 de Software Publishing Corp.;
- entre otros.

6.5 PAQUETES ESPECIALES

MICRO TSP (TIME SERIES PROCESSOR)

MICRO TSP proporciona herramientas de regresión y pronóstico aplicables en computadoras personales compatibles; se puede hacer una relación estadística sobre sus datos y usar esta relación para pronosticar los valores futuros de los datos. Las áreas de aplicación pueden

incluir:

- Pronósticos de ventas.
- Análisis de costos y pronósticos.
- Análisis financiero.
- Pronóstico del mercado de valores.
- Planeación de la producción.

MICRO TSP es una versión para computadoras personales compatibles de un conjunto de programas para manipular series de tiempo, originalmente desarrolladas en el paquete Time Series Processor para computadoras grandes.

EL COORDINADOR

El Coordinador facilita y aumenta la productividad del trabajo en equipo. Con esta herramienta se puede estabier una red fluida de comunicaciones entre departamentos completos, diferentes divisiones, oficinas remotas y ejecutivos y empleados en viaje de negocios, así como con sus proveedores y otras organizaciones con las que se trabaja. Además, permite a los ususarios controlar y dar un seguimiento eficaz a las actividades.

Muestra agendas de un día y hasta de todo el año. Entrega mensajes de una red local, entre redes con múltiples administradores de archivos, entre redes dispersas, y entre estaciones individuales.

CORREO ELECTRONICO

El correo electrónico más que una aplicación es un concepto y tiene la finalidad de intercambiar mensajes electrónicos en forma remota pudiendo anexar archivos diversos (programas, formas, bases de datos, etc.) a los mismos. Podemos decir que el correo electrónico es un complemento al cómputo organizacional y debe permitir un acercamiento entre diversos grupos de trabajo para facilitar seguimiento a proyectos y asuntos con el objetivo de incrementar la productividad de la organización.

Entre los paquetes de correo electrónico más comúnes tenemos:

Microsoft Mail

Es un paquete para redes de PC que permite el envío de mensajes, compartir información y la administración del sistema de mensajería. Mediante este correo electrónico se puede enviar hojas de cálculo, fotografías, sonido, objetos de multimedia y textos. Se permite asignar prioridades a los mensajes para que el interlocutor se dé cuenta de su importancia y los conteste, además de la confirmación de recepción. El envío de mensajes se puede hacer a nivel mundial con solo alimentar la lista de interlocutores, además cuenta con la función de búsqueda de nombre para agilizar la localización de direcciones.

Da Vinci Email

Da Vinci Systems Corp. ofrece un correo electrónico que se maneja con una simple y probada combinación de menús, ventanas y soporte opcional de mouse. Usa un sistema jerárquico de

carpetas y subcarpetas para poder guardar mensajes, se puede configurar para archivar automáticamente ciertos tipos de mensajes en las carpetas que se especifiquen. Permite a los usuarios anexar todos los archivos necesarios, planillas de cálculo, gráficas, documentos, etc.

6.6 PAQUETES EXISTENTES EN EL MERCADO MEXICANO

Como no es nuestra intención decribir todos los paquetes que existen en el mercado, sino sólo presentar en forma general los paquetes con los que el estudiante o profesionista se puede ayudar en sus actividades, a continuación mencionamos el software más conocido dentro del mercado en México.

Sistema operativo para computadoras personales o estaciones de trabajo:

- MS-DOS
- UNIX
- XENIX

Interfaz al usuario en computadoras personales o en estaciones de trabajo:

- MS-WINDOWS
- OSF/MOTIF
- XWINDOWS

Bases de datos:

- DBASE
- INFORMIX
- SQL BASE
- PARADOX
- INGRES

Lenguajes de programación:

- BASIC
- COBOL
- CLIPPER
- ORACLE
- PASCAL
- TURBO PASCAL
- PL1
- FORTRAN
- XBASE
- SQL FORM
- ASSEMBLER
- RPG

Lenguajes de cuarta generación:

- ACTOR
- SQL WINDOWS
- SISINF

Procesadores de palabras:

- AMI PROFESIONAL
- WINDOWS WRITE
- WORD
- WORD FOR WINDOWS
- MULTIMATE
- WORD STAR
- WORD PERFEC

Hojas electrónicas:

- LOTUS
- EXEL
- WING Z
- SUPERCAL

Graficación administrativa:

- POWERPOINT
- FREELANCE
- COREL DRAW

Graficación técnica:

- CAD CAM
- AUTOCAD

Generador de prototipos:

- KNOWLEDGEPRO
- TOOLBOOK

Desktop publishing:

- VENTURA PUBLISHER
- PAGEMAKER
- FRAMEMAKER

Control de proyectos:

- MS PROYECT FOR WINDOWS
- TIME LINE
- HARVARD TOTAL

Correo electrónico:

- DVINCI
- MSMAIL
- COORDINADOR
- C OBJECT OFFICE

Sistemas ejecutivos de información:

- PILOT/LIGHTSHIP
- FORESY & TREES
- EXPRESS/EIS

Paquetes integrados:

- FRAMEWORK
- SYMPHONY
- UNIPLEX
- WORKS

Paquetes antivirus:

- NORTON ANTIVIRUS (NAV)
- SCAN ANTIVIRUS
- NETSCAN PARA REDES
- PS-CILLIN

BIBLIOGRAFIA

Arechiga R., Corchado J., Domínguez R., González S. Fundamentos de Computación. Instituto Politécnico Nacional, Dirección de Estudios Profesionales. México.

Farina Mario V. Diagramas de Flujo. Ed. Diana. México, 1976.

Goldstein L. J., Goldstein M. Introducción al Sistema Operativo, Programación y Aplicaciones en Basic. Ed. Prentice-Hall Hispanoamericana, S. A. México, 1986.

Hunt R., Shelley J. Computadoras y Sentido Común. Ed. Prentice-Hall Internacional, S. A. México, 1987.

Monro Donald M. Basic básico. Introducción a la Programación. Ed. McGraw-Hill. México, 1984.

Orilia L. S. Basic Estructurado. Ed. McGraw-Hill, México, 1987.

Orozco Maldonado L. Manual Pascal Sistema HP 1000. Escuela Nacional de Estudios Profesionales Aragón, Universidad Nacional Autónoma de México. México, 1990.

Robledo Sosa C. Introducción a las Computadoras. Ed. Tlamatini S. A. México, 1986.

Salvat. Los Ordenadores. Ed. Salvat. Barcelona España, 1979.

Siechert C., Wood C. La Potencia de: PC/DOS. Ed. McGraw-Hill. México, 1988.