UNIVERSIDAD CATOLICA DE SANTIAGO DEL ESTERO

DEPARTAMENTO ACADEMICO SAN SALVADOR

INGENIERIA EN COMPUTACION

INFORMATICA

Año: 1998

Programa

Unidad 1: Informática

Concepto. Diferencia entre dato e información. La información y la toma de decisiones. La informática como disciplina científica. Areas de aplicación de la informática.

Unidad 2: Arquitectura del computador

El computador: concepto. Calsificación. Generación.

Elementos que lo componen. Unidad central de proceso.

Unidad de control: funciones que realiza.

Unidad aritmética y lógica. Operaciones lógicas básicas.

Memoria: clasificación.

Periféricos. Concepto. Clasificación. Periféricos de entrada/salida y de almacenamiento.

Software: concepto. Clasificación.

Sistemas de numeración. Conversión.

Unidad 3: Heurística y Algoritmación

Técnicas para la solución de problemas. Definición de algoritmo. Propiedades.

Método Heurístico. Diagramas de flujo. Simbología. Estructuras lógicas de control básicas.

Diagramas lineales. Ciclos. Vectores. Matrices.

Unidad 4: Sistema Operativo

Entornos de trabajo en RED y en computadoras aisladas.

Comandos del sistemas operativo DOS:

- Para el manejo de directorios.

- Para el manejo de archivos.
- Para el manejo de impresora.
- Para el respaldo de la información.

Unidad 5: Lenguaje de programación PASCAL

Características del lenguaje. Formato de las sentencias. Constantes. Variables: tipos. Expresiones y operadores: jerarquias.

Tipos de sentencias: de entrada/salida, de bifurcación, de control repetitivas, de transferencia de control.

Funciones: tipos.

Estructuras de datos: vectores y matrices. Sentencias para el manejo de las mismas.

Unidad 6: Representación de los Algoritmos: Tablas de Decisión

Problemas de computación. Etapas en el proceso de solución de problemas con las computadoras.

Representación de algoritmos.

Tablas de decisión: definición, estructura, metodología de construcción.

Tipos de condiciones. Tipos de tablas. Condiciones independientes. Condiciones dependientes. Situaciones compuestas.

Procedimiento para la verificación de tablas de decisión. Conversión de tablas de decisión a programas: programación directa. Transformación a diagrama de flujo: método de Pollack. Ventajas de uso de tablas de decisión.

Unidad 7: Prueba, Documentación y Mantenimiento de Programas

Prueba de programas: definición. Casos de prueba exitoso y no exitoso. Pruebas de escritorio. Principios de laprueba.

Documentación: concepto. Objetivos internos y externos. Documentación de programas: carpeta de programas. Documentación interna y externa.

Manual de operaciones. Manual del usuario.

Mantenimiento: concepto. Técnicas y herramientas. Tipos de mantenimiento: correctivo, preventivo y extensivo.

Indice

UNIDAD

UNO

INFORMATICA

INFORMATICA

Concepto: Vocablo de origen Francés, compuesto por dos palabras: INFORMACION−AUTOMATICA, siendo la INFORMATICA disciplina del tratamiento automatico y racional de la informacion.

Es la disciplina que se dedica a la obtencion y tratamiento de la informacion *clara, precisa, confiable y oportuna* para el planeamiento, la organización y la toma de deciones empresiarias, a traves de tecnicas y teorias.

LA INFORMATICA ABARCA TODAS LAS RAMAS DE LA ACTIVIDAD HUMANA, decenas de millones de computadoras personales se encuentran en las estaciones de trabajo de oficinas, fabricas, escuelas, hogares, hospitales, agencias del gobierno, bancos y tiendas, ademas de los laboratorios. Y ademas de estos sistemas de aplicación general, la gente se encuentra por todos lados con computadoras invisibles de aplicacion general: en los aparatos domesticos, en los automoviles y en los relojes de pulso. De hecho si en este momento dejaran de funcionar todas las computadoras, no podrian moverse los aviones, trenes y muchos automoviles y elevadores, los semaforos y telefonos serian inutiles, y el mundo se veria sumido en una confusion total.

DATO (del Latin DATUM): Hecho aislado que se caracteriza por no tener significado propio, el dato se utiliza como materia prima de la informacion.

INFORMACION: Es el conocimiento que se tiene sobre algo, se esta informado a medida que se tiene conocimiento.

DATO	INFORMACION
• −Punto de partida de la investigacion.	• −Conjunto de DATOS.
• −Hecho aislado.	• −Conocimiento que se tiene sobre algo.
• −No tiene significado.	• −Si tiene significado por que es un conjunto de datos que agrupados significan algo.

Los *datos* son hechos, son la materia prima de la *información*. Los datos se representan por medio de símbolos, pero solo pueden considerarse como información en un sentido muy limitado. La información consiste en conocimientos importantes producidos como resultado de las operaciones de procesamiento de datos. *Toda la información consta de datos, pero no todos los datos producen información específica e inteligente*.

DATOS INFORMACION

LA INFORMACION y LA TOMA DE DECISIONES

La *información* sirve para tomar *deciosiones* con vistas a un accionar concreto (presente o futuro), y se obtiene realizando operaciones sobre datos. Su elaboracion permite tomar conocimiento de algún aspecto de la realidad, desconocido, lo cual disminuye la incertidumbre existente antes de tomar una decision.

Por ejemplo, supongamos que una persona tiene que comprar un mismo articulo en tres comercios diferentes. Antes de concretar su compra, en su mente podrá tener las siguientes alternativas:

Si en el comercio A el precio es más barato, *entonces* compro en A

Si en el comercio B el precio es más barato, *entonces* compro en B

Si en el comercio C el precio es más barato, *entonces* compro en C

Dicha persona por los medios apropiados obtendrá el valor de venta de dicho artículo en cada comercio (*datos*) como ser en el A: $20; en el B: $19 y en el C: $21. Luego realizará operaciones de comparación entre los datos a fin de determinar cual es el precio más bajo. Entonces habrá elaborado la informacion que le interesaba, supuestamente constituida por la representacion simbolica en B venden más barato, la cual permitirá tomar la decisión de comprar en el comercio B entre las tres alternativas posibles. Obviamente de no haber elaborado la información, y si compra sin más en A ó en C, la decisión hubiese sido mal tomada, *si el objetivo es el menor costo*.

Se debe cumplir lo siguiente:

- Tiempo de respuesta: la información debe estar en el momento oportuno para la toma de decisiones.
- Exactitud y precisión: la información debe ser exacta.
- Completa: debe tener los datos requeridos por el empresario.
- Correlativa: entendible y clara.
- Formato: debe ser leible y entendible.

LA INFORMATICA COMO DISCIPLINA CIENTIFICA

La Informática es una ciencia (esta compuesta por conocimientos de validez universal y utiliza para su estudio el método científico, las ciencias que dan origen a la Informática son: La Teoría General de Sistemas, La Cibernética, La Teoría de la Información) que se ha tratado como tal desde hace pocos años, a ella se le asocian una serie de hechos y descubrimientos anteriores que han servido para que hoy en día sea una de las ciencias a la que el hombre le esta dedicando mayor atención e importancia.

La palabra ***computar*** viene del Latín, significa genéricamente *contar, calcular* (aunque hoy día esta asociada a una máquina determinada), el proceso de datos es tan antiguo como el HOMBRE. Podemos decir que la primera herramienta para computar o primera computadora, fueron los dedos de la mano. Manteniendo los dedos de la mano en determinada posición, pudo memorizar y comunicar números. Puesto que la suma y la resta implican contar, aumentando o reduciendo el número de dedos estirados, fué posible sumar y restar.

Todas estas técnicas manuales son *posicionales,* como también lo es nuestro sistema decimal. No es casual que dígito −palabra latina que significa dedo− se use también para indicar cada una de las posiciones de los números, y se hable de computadoras *digitales.*

La etapa siguiente de la computación pudo haber empezado con la representación simbólica de números mediante montoncitos de DIEZ guijarros o piedritas. Del uso de estas ultimas para dicho fin, ha derivado la palabra *cálculo,* siendo que en Latín la palabra piedras es *calculus.*

De gran trascendencia fue la posterior invención del ***ábaco***. Se realizaba practicando varios surcos paralelos iguales sobre la superficie de la tierra o arena, esta última a veces contenida en una bandeja. Por cada objeto contado se coloca una piedra en el surco que esta más a la derecha (unidades). Cada vez que en este había diez piedras, se quitan las mismas y se reemplazan por una sola piedra colocada en el surco siguiente de la izquierda (decenas), obteniéndose así el equivalente a diez dedos del hombre. Cuando se tenían diez piedras en el surco de las decenas, también se reemplazaban por una sola piedra colocada en el surco siguiente de la izquierda (centenas).

De esta forma, con unas pocas piedras, se podía contar conjuntos tan grandes de elementos como fuese necesario.

Posteriormente el ábaco fue motivo de varias mejoras que le dieron mas velocidad de computo y portabilidad. Los surcos fueron reemplazados por finas varillas de madera paralelas, sujetas a una base, sobre las cuales se podían ensartar las cuentas (3500 a. C. aproximadamente). En el 2600 a. C. aparecen, en la China y en el Japón, el Suan−Pan y el Soroban, respectivamente. También por esa apoca apareció un sistema numérico INDOARABIGO del que surgió el sistema decimal, este permitió operar de la misma forma que con el ábaco pero con lápiz y papel; los hindúes predominan en las matemáticas hasta el siglo XII. En América precolombina, los Mayas usaban el ábaco, y varios sistemas posicionales que incluían el numero cero.

A finales del siglo XVI, el matemático escocés John Napier (inventor de los logaritmos naturales), ideo un dispositivo, basado en palillos que contenían números, capaz de multiplicar y dividir en forma automática. También ideo un calculador con tarjetas que permitía multiplicar y que tomó el nombre de ESTRUCTURAS DE NAPIER.

En el año 1623, Schickard (alemán), construye para Kepler, un reloj de calculo, basado en ruedas contadoras, para hacer sumas y restas de hasta seis dígitos.

Pocos años después, Blaise Pascal (matemático y filósofo Francés), en el año 1642, inventa una pequeña *calculadora mecánica,* con engranajes, capaz de sumar números y totalizar su resultado. Se llegaron a construir 50 de estas máquinas, a las que se denominó Máquina aritmética de Pascal.

En 1650, Patridge, basándose en los descubrimientos de Napier, inventó la regla de cálculo, pequeña regla

deslizante sobre una base fija en las que figuraban diversas escalas para la realización de diversas operaciones. Muy usado hasta los años setenta, cuando las calculadoras electrónicas lo sustituyen.

En el año 1808, Jacquard, Francés, perfecciona el uso de cartones perforados para cambiar a voluntad el dibujo que teje un telar automático. Podemos considerarla como la primera máquina mecánica programada.

En 1822, Charles Babbage, matemático inglés y profesor de la universidad de Cambridge, diseñó su *máquina de diferencias* que calcula mecánicamente valores numéricos sucesivos de polinomios de 2° grado, con 8 cifras decimales para construir tablas. Debido a las diferencias tecnológicas de la época, esta máquina no llegó a fabricarse.

En 1833, Babbage diseñó su *máquina analítica*, similar a la computadora actual, pues disponía de programa, memoria, unidad de control, periféricos de entrada y periféricos de salida (esquema que Aitken utilizó). La idea de su fabricación surge de la necesidad de realizar automáticamente tablas de logaritmos y funciones trigonométricas. Esta máquina tampoco llega a construirse, por las mismas razones a la anterior. Por este diseño, a Charles Babbage se lo considera el PADRE DE LA INFORMÁTICA.

Augusta Ada Bayron, es considerada la *primer programadora de computadoras* por su trabajo en la prueba de la Máquina Analítica de Babbage, escribió: la máquina puede efectuar un examen para determinar si ha tenido lugar un evento entre varios posibles, y seguir después el camino que convenga... puede hacer aquello que sepamos ordenarle que haga.

En 1854, George Boole, matemático inglés, desarrolló la teoría del Algebra de Boole, que permitió a sus sucesores la representación de circuitos de conmutación y el desarrollo de la llamada *Teoría de los Circuitos Lógicos*.

Sobre el año 1885, Herman Hollerith, funcionario de la Oficina del Censo de los Estados Unidos, observa que la mayoría de las preguntas tenían como respuesta *sí* o *no*, lo que lo hizo idear en 1886 una tarjeta perforada para contener la información de las personas censadas y una máquina capaz de leer y tabular dicha información, construyó su *Máquina Censadora* o *Tabuladora*, tardándose solo tres años en realizarse el censo y perforándose una cantidad de 56 millones de tarjetas.

En el 1896, Hollerith fundó la Tabulating Machine Company, para fabricar y vender su invento, que más tarde se fusionó con otras empresas para formar lo que hoy es IBM (International Business Machines).

El proceso de datos con tarjeta perforada comenzaba a aplicarse en las grandes corporaciones, mucho antes de que inventaran las computadoras.

En 1936, Alan Turing, matemático inglés, desarrollo la teoría de una máquina capaz de resolver todo tipo de problemas, llegando a la construcción teórica de la *Máquina de Turing*. Una Máquina de Turing es la forma de representar un proceso a partir de su descripción.

En 1937, Howard Aitken, de la Universidad de Hardvard, desarrolla la idea de Babbage junto con científicos e ingenieros de IBM. Como resultado se construye la primera computadora electromecánica basada en relés, denominada: Calculadora Automática de Secuencia Controlada (Automatic Sequence Controlled Calculator, ASCC), y que se llamó: MARK I.

La Mark I se terminó de construir en 1944, tenia elementos de entrada (tarjetas y cintas perforadas), memoria principal, unidad aritmética, unidad de control y elementos de salida.

Tenia 17 m de largo por 2 m de alto, pesaba unas 70 toneladas, estaba constituida por 700000 piezas móviles y un cableado de una longitud de 800000 m. Sumaba dos números en menos de un segundo y los multiplicaba

en seis segundos.

En 1946, John Mauchly y Prosper Eckert Jr. terminan de construir la ENIAC (Electrical Integration Automatic Computer) para generar tablas de balística. Constaba de 18000 válvulas y 1500 relés. Esta máquina era unas 1000 veces más rápida que la Mark I. La ENIAC operó durante unos 10 años.

UNIDAD

DOS

ARQUITECTURA DEL COMPUTADOR

El COMPUTADOR

Concepto: es un artefacto electrónico cuya función principal se puede resumir en dos palabras: *procesar información*, tarea que realiza a gran velocidad, siguiendo las instrucciones indicadas por un programa.

Se ingresan datos, a traves de los distintos Dispositivos de Entrada, a la computadora, los procesa y nos presenta los resultados a traves de los Dispositivos de Salida.

CLASIFICACION

<table>
<tr><td rowspan="7">Clasificación

De

Computadoras</td><td>(a)Por los tipos que trata</td><td rowspan="7">Analógicos.</td><td></td><td>Digitales.</td></tr>
<tr><td></td><td></td><td>Híbridos.</td></tr>
<tr><td></td><td rowspan="5">(b)Por su tamaño</td><td>Microcomputadoras.</td></tr>
<tr><td></td><td></td><td>Minicomputadoras.</td></tr>
<tr><td></td><td></td><td>Macrocomputadoras.</td></tr>
<tr><td></td><td></td><td>Supercomputadoras.</td></tr>
</table>

- Es posible construir computadoras para contar números o para medir relaciones físicas. Una computadora digital es aquella que cuenta directamente los números, que pueden representar letras, símbolos especiales, etc. Las computadoras analógicas miden magnitudes físicas que se distribuyen en una escala continua, como ser la temperatura o la presión. Por ejemplo: la bomba de gasolina de una estación de servicio contiene un procesador analógico que convierte las mediciones de flujo de combustible en valores de volumen y precio. En ocasiones se combinan las características de las máquinas analógicas y de las digitales para crear un sistema de computo híbrido. Por ejemplo: en la unidad de cuidado intensivo de un hospital, se mide la función cardiaca, la temperatura y otros signos vitales de los pacientes mediante dispositivos analógicos. Enseguida esas mediciones se convierten en números y se alimentan a un componente digital que supervisa los signos vitales del paciente y llama la atención de una enfermera si se detectan lecturas anormales.
- Las computadoras modernas varían de tamaño físico desde las que ocupan totalmente un cuarto, hasta aquellas cuya CPU cabe en la uña de un dedo. Las microcomputadoras o micros son los equipos más pequeños para usos generales. Pero pueden realizar las mismas tareas que muchas computadoras mayores. Se las suele utilizar en las corporaciones para apoyar la toma de decisiones, entre los periodistas y viajeros son muy populares las portátiles ya que las utilizan para procesamiento de texto en cualquier lugar momento. Las minicomputadoras o minis también son sistemas pequeños de aplicación general, pero a diferencia de la mayor parte de las micros, generalmente atienden a varios usuarios. Son por lo común más poderosas y costosas que las micros. En tamaño físico, las minis van desde modelos de escritorio hasta unidades del tamaño de un archivero pequeño. Es en el campo de las comunicaciones donde se las utiliza

con mas frecuencia y donde a alcanzado su mayor desarrollo, por ejemplo: para la reserva de plazas, transmisión de mensajes, etc. Las macrocomputadoras proporcionan en un lugar central, el poder del proceso requerido en una organización, puede ser utilizado como el anfitrión central de una red de procesamiento distribuido de datos. Comunica y ejerce control sobre procesadores satélites más pequeños. Las supercomputadoras, son los sistemas más grandes, rápidos y costosos del mundo. Se las usan en aplicaciones con enorme cantidad de datos, y donde el tiempo de respuesta debe ser lo mas corto posible. Areas de aplicación: en el terreno militar estratégico, en la predicción del tiempo, sismos, física nuclear, etc.

GENERACIONES DE COMPUTADORAS

Desde que en la primera mitad de la década del '50 se empezaron a utilizar las computadoras con fines comerciales, estas han evolucionado hasta el punto en que se pueden distinguir generaciones claramente identificables. El método que nos permite decidir en que momento termina una generación y comienza otra se basa en dos (2) características: una, la tecnología utilizada para la construcción de las computadoras; dos, la arquitectura de los sistemas.

- Primera generación de computadoras ('46 / `58): computadoras construidas con válvulas electrónicas (lámparas de vacío), por lo que su tamaño era muy grande, su mantenimiento complicado y desprendían grandes cantidades de calor. La tasa de averías era elevada y su costo era un freno oponiéndose a su expansión. El software apenas existía en esta generación, se programaba en lenguaje máquina.

Dispositivo de E / S: tarjetas de papel, cinta perforada.

Memoria Principal: tambor magnético.

Memoria Secundaria: cinta magnética.

Software: programación en lenguaje simbólico de máquina (Assembler), que permita expresar los códigos binarios de las instrucciones de máquina mediante símbolos de nuestro alfabeto, para facilidad del programador.

- Segunda generación de computadoras ('58 / `65): se impone el transistor, más confiable de menor tamaño, menor disipación de calor y más rápido que la válvula para cambiar de estado. Se alcanzan velocidades de procesamiento de centenares de miles de instrucciones por segundo. Las máquinas disminuyen de tamaño y costo.

Dispositivo de E / S: tarjetas perforadas, cintas magnéticas de alta velocidad, impresoras.

Memoria Principal: núcleo magnético, desde el '60, memoria de película delgada.

Memoria Secundaria: cintas y discos magnéticos.

Software: aparecen los sistemas operativos de tiempo compartido y se generalizan los lenguajes de programación de alto nivel (COBOL, APL, PL/1, este último para usuarios de IBM).

- Tercera generación de computadoras ('65 / `74): el desarrollo de los circuitos integrados en pequeña y mediana escala de integración, y de plaquetas impresas con caminos de cobre para soportarlos, permitieron equipos más compactos, confiables y económicos. Se generaliza el uso de computadoras para los más diversos tipos de actividades, en esta generación aparecen las minicomputadoras.

Dispositivos de E / S: unidades de respuesta hablada.

Memoria Secundaria: discos rígidos.

Software: se trabaja en multiprogramación; varios programas que se ejecutan simultáneamente, el tiempo compartido y la memoria virtual. En el '65 aparece BASIC para usos de iniciación en computación. Hacia el '70 nace el lenguaje PASCAL, para la programación estructurada. En el '71 aparece el sistema operativo UNIX, de los laboratorios Bell, es eficaz económico y sencillo, adaptable a las más diversas arquitecturas.

- Cuarta generación de computadoras ('74 /...?): el desarrollo de chips en muy grande escala de integración con millones de transistores (Pentuim tiene 3.200.000) ha permitido el advenimiento de microprocesadores que hoy en día superan en velocidad al 360 de IBM (computadora para uso universal) referente de la tercera generación. Ellos permiten fabricar microcomputadoras personales baratas, que han invadido todos los ámbitos, con una gran variedad de periféricos fabricados para las mismas. Por otra parte, las computadoras personales, cada vez en mayor grado, se comunican entre sí a través de módem y redes globales.

No hay acuerdo general acerca del año establecido como el comienzo de esta generación, ni tampoco cuando termina, o sea si estamos en la cuarta o quinta generación.

- Proyecto de la quinta generación: en 1982 en Japón se anunció el proyecto de quinta generación de computadoras, a desarrollar en un lapso de diez años, con el fin de crear una arquitectura basada en una *máquina paralela de diferencias lógicas* y el software correspondiente para la misma.

Se planteó un nuevo modelo de computador, que pretendía innovar en gran medida las técnicas informáticas, en especial en los siguientes aspectos: *velocidad de procesamiento, operaciones con lógica simbólica* y *aplicaciones orientadas hacia la Inteligencia Artificial*.

Shapiro y Warren en comentarios aparecidos en una revista en Marzo del '93, plantean que la computación paralela requiere un mercado que necesite grandes cálculos numéricos regulares que hoy son suficientemente simples. Todavía no existe una demanda de computadoras paralelas para realizar una clase más amplia de problemas, a nivel de requerimientos industriales. El boom que provocó la inteligencia artificial en los años '80 a declinado, reduciéndose el número de investigadores en esta área, y los que siguen en ella no requieren un poder computacional mayor. Tampoco hoy se necesitan las aplicaciones basadas en Sistemas Expertos.

Sostienen que los investigadores de la quinta generación no han logrado determinar demasiadas aplicaciones que pongan en relieve el poder computacional de las máquinas construidas.

ELEMENTOS QUE LO COMPONEN

Retomando la definición antes expuesta de computadora, debemos tener en cuenta cuales son esos *dispositivos:* son de *entrada* si permiten el ingreso de información para ser procesada (el scanner, el mouse, el teclado), de *salida* si permiten mostrar la información procesada (el monitor, la impresora), o de *entrada y salida* simultáneamente, es decir, que pueden recibir o brindar información (el disco rígido). Además cuenta con CPU que es la *Unidad Central de Proceso*, es la parte fundamental de la computadora y controla su funcionamiento. Realiza todo el proceso de tratamiento de datos conforme a las instrucciones del programa.

Estructura del computador:

CPU

La CPU (a quien también llamaremos microprocesador o procesador) constituye el cerebro del computador, se

encarga de tomar la información que recibe de diferentes fuentes, efectuar los procesos necesarios a dicha información y enviar el resultado al destino que se le indicó.

El microprocesador dispone de tres vías de comunicación (lo que llamaremos BUS):

- Bus de Dirección: conducen de CPU a memoria principal (MP) cada combinación de unos y ceros que indica dónde localizar las instrucciones o datos en MP, es *unidireccional*.
- Bus de Datos: en cada lectura de MP conducen de ésta hacia la CPU tantos datos a operar como instrucciones; y en una escritura conducen desde la CPU hacia MP datos resultantes. Son pues, *líneas bidireccionales*.
- Bus de Control: son *unidireccionales individuales* para que la CPU dé órdenes (como leer y escribir en MP) y para que ella reciba señales (como la que origina la MP para indicar lectura efectivizada).

La CPU puede dividirse en varios bloques, de los cuales vamos a tomar los dos más importantes para analizar el funcionamiento básico:

- La CU o UC (*Control Unit* − Unidad de Control).
- La ALU o UAL (*Arithmetic and Logic Unit* − Unidad Aritmética y Lógica)

Otros autores consideran que la MP forma parte de la CPU, aumentándose a los dos elementos mencionados anteriormente.

Esquema interno de la CPU:

La UC tiene a su cargo el secuenciamiento de las acciones necesarias que deben realizar los circuitos involucrados en la ejecución de cada instrucción, según el código de la misma; y también tiene a su cuidado el orden de ejecución de un programa, conforme como éste fue establecido. En otras palabras, controla y coordina el conjunto de operaciones que se realizan, interpretando para ello las instrucciones que componen el programa.

La secuencia lógica que la unidad de control debe realizar para ejecutar una instrucción es la siguiente:

- Localizar y extraer de la MP la instrucción correspondiente.
- Transferir la instrucción de la MP a la UC.
- Determinar qué tipo de operación se debe ejecutar.
- Ejecutar la instrucción, enviando las señales de control u órdenes a los elementos pertinentes.
- Supervisar la operación anterior para determinar si ha finalizado correctamente.
- Localizar la siguiente instrucción a ejecutar.

Para llevar a cabo estas operaciones la UC utiliza los siguientes elementos:

Reloj: consiste en un circuito eléctrico capaz de generar una sucesión de pulsos a intervalos de tiempo constante. El intervalo entre dos pulsos se denomina ciclo, así se controla la duración de las distintas instrucciones.

Contador de programa: (también denominado registro contador de instrucciones) su misión es controlar el orden de ejecución de las instrucciones de programa. La PC almacena la dirección de la próxima instrucción a ejecutarse.

Registro de instrucción: guarda la instrucción cuando se extrae de la MP y se mantiene mientras se realiza la decodificación o interpretación.

Decodificador: analiza el tipo de operación que se encuentra en el campo de operación de la instrucción.

Secuenciador: es el generador de órdenes (denominadas microórdenes) que sincronizadas con el reloj y distribuidas a los elementos necesarios permiten la ejecución de la instrucción.

La ALU sirve para realizar las operaciones aritméticas o lógicas que le ordena la UC, siendo auxiliada por registros acumuladores para guardar transitoriamente datos y resultados.

Mientras que la UC es la encargada de ordenar operaciones de lectura−escritura de registros y de memoria, así como las operaciones que debe realizar la ALU; ésta última no puede emitir orden alguna.

Por lo tanto, la ALU no ejecuta instrucciones, no puede ordenar las operaciones correspondientes a los pasos que requiere la ejecución de una instrucción. Sólo realiza uno de ellos: la operación aritmética o lógica que la instrucción ordena, cuándo así lo requiere la UC. Mediante una operación de la ALU, a partir de uno o dos números (materia prima) se puede obtener un número(resultado) que antes no existía.

La palabra Lógica de las siglas ALU puede llevar a suponer inteligencia, siendo que las operaciones lógicas de la ALU son las operaciones de AND, OR, negación, etc.

Las operaciones lógicas básicas son de AND, OR, NOT, pudiéndose obtener a partir de ellas las de NAND y NOR.

MEMORIA PRINCIPAL

La memoria es uno de los componentes fundamentales de las computadoras, ya que sin ellas éstas no podrían procesar información de ninguna manera, por que no tendrían un medio de almacenamiento de información temporario.

La MP almacena bits en *celdas* independientes, aisladas entre sí, que contiene un byte (8 bits) de información. Cada celda se localiza en el conjunto mediante un número binario, que constituye su *dirección*, o indican su *posición* en ese conjunto. Este número no se puede alterar, pues está establecido circuitalmente.

El acceso a la MP es *aleatorio* (es la forma directa de acceder, seleccionar o ubicar algo), implica que cualquier posición puede encontrarse en igual tiempo, sin búsqueda alguna. Vale decir que el tiempo de acceso es el mismo para cualquier dirección.

Las dos acciones que se pueden realizar con una memoria son:

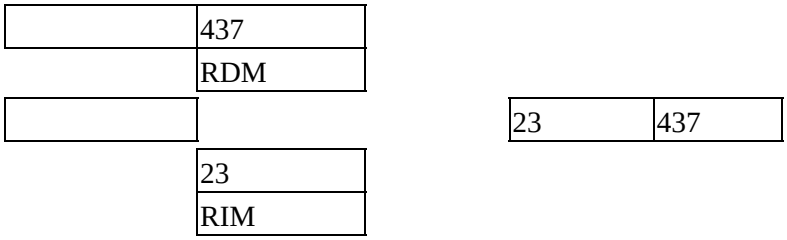
- Leer o extraer información de una celda de memoria.
- Grabar o escribir información en una celda de memoria.

Para realizar estas dos funciones es necesario dos registros, que son:

- RDM (registro de dirección de memoria).
- RIM (registro de intercambio de memoria).

Proceso de lectura:

Para acceder a la celda que se desea, debe introducirse su dirección en el RDM, cuando llega una orden de lectura, el contenido del RDM pasa por un complejo circuito de selección que permite acceder a la palabra en memoria solicitada y lleva el contenido de la misma al RIM.



Proceso de escritura:

Ahora se quiere escribir un dato, la dirección de la celda a seleccionar se introduce en el RDM. El dato a escribir se introduce en el RIM. Se da una orden de escritura y el contenido del RIM es llevado a la celda seleccionada por el RDM y queda grabado en ella.

Es importante observar que la lectura de una celda no es destructiva pero no así la escritura.

CARACTERISTICAS:

- Capacidad De La Memoria: *La capacidad de almacenamiento de una memoria es la cantidad total de bytes que puede guardar*. En el presente, lo común es que en cada celda se guarde un byte de información. Entonces una memoria con N celdas tendrá una capacidad de N bytes. Dicho número es siempre potencia entera de dos: N=2k.

La capacidad se expresa en las siguientes unidades y múltiplos:

El byte (B) es la unidad menor, siendo sus múltiplos el Kilobyte (KB), el Megabyte (MB), el Gigabyte (GB) y el Terabyte (TB).

Puesto que 210 = 1024, es la potencia de 2 más cercana a 1000 (kilo), entonces se considera 1024 bytes = 1KB.

El múltiplo siguiente es 1024 veces mayor: 1MB =1024 x 1KB = 1.048.576 Bytes, número cercano a 1.000.000 (*mega*).

Siguen los múltiplos:

1GB = 1024 x 1MB (supera a mil millones = *Giga*)

1TB = 1024 x 1GB (supera el millón de millones = *Tera*)

- Tiempo De Acceso: Se define así al tiempo que tarda el computador en leer o escribir una celda de memoria estos tiempos son muy pequeños y suelen medirse en Nanosegundos o en Microsegundos. El Nanosegundo es una unidad de tiempo que significa una mil millonésima de segundo (10−9 segundo), o sea mil veces menor que una millonésima de segundo. Una millonésima de segundo (10−6 segundo) es un Microsegundo.

En el presente los microprocesadores internamente realizan una operación en contados Nanosegundos. De ahí que esta unidad de tiempo tenga sentido en el ámbito de la computación, para evitar escribir 0,000000001 seg.

- Costo: Esta en función de la capacidad y del tiempo de acceso a memoria. Las memorias más caras son las más rápidas.

- Volatilidad: son aquellas memorias que por sus característica físicas pierden la información almacenada cuando se corta el suministro de corriente eléctrica, sea por apagado del equipo o por accidente.

Las primeras computadoras fabricadas comercialmente ya tenían MP con acceso random, o sea que eran Random Access Memory (RAM), y cualquier celda podía ser escrita o leída todas las veces que se quería. Desde entonces decir memoria RAM es sinónimo de memoria de lectura y escritura y además de memoria volátil.

La cuestión de las denominaciones se complica, desde que, para una porción de la MP, a partir de los años '70 se emplean chips de memoria ROM (Read Only Memory − Memoria de Sólo Lectura), estos chips desde entonces contienen un programa para el arranque inicial de las computadoras. La memoria ROM también es de acceso Random, como exige que sea una memoria principal, aunque su tiempo de acceso puede ser más largo que el de la RAM.

La memoria ROM es una memoria no volátil, o sea que almacena información en forma permanente, no necesita energía eléctrica para mantener guardados los datos, si para leerlos.

Típicamente la porción ROM de MP de una PC está en uno o varios chips ROM.

Por lo tanto:

DISPOSITIVOS PERIFERICOS

Se denominan dispositivos periféricos a todas las unidades que se encargan del intercambio de información entre el exterior y la unidad central. No forman parte de la CPU, de ahí que sea necesario adaptar la información precedente del exterior para que sea inteligible por la computadora, así como adecuar la información suministrada por el computador para que sea tratada por los periféricos. La denominación de *periféricos* proviene de su posición, vinculada al mundo exterior en su relación con la porción central o interna, constituida por el microprocesador y la MP.

Clasificación:

Dispositivos Periféricos		Periféricos de comunicación	De entrada	
		Su utilización		De salida
	Según su periferia	Memorias secundarias o auxiliares.		
		Periferia local: dispositivos ubicados a pocos metros de la CPU.		
		Periferia remota: dispositivos situados a grandes distancias, conectados al computador por algún medio de comunicación, como la línea telefónica.		
	Según su modo de acceso a los datos	Acceso secuencial: no se puede acceder a un dato grabado en estos dispositivos sin haber pasado por todos los anteriores.		
		Acceso directo: se puede acceder a cualquier dato de una manera inmediata, sin necesidad de pasar por todos los anteriores (ramdom).		

La actuación de las unidades de entrada/salida esta controlada por la CPU. Una computadora puede tener varias unidades de E / S que a su vez pueden controlar varios periféricos del mismo tipo.

	Canales.
Dispositivos de entrada/salida	
	Controladores de Periféricos.

Cuando la CPU efectúa una operación de transferencia de información a un periférico, lanza todas las órdenes de transferencia al canal y continúa ejecutando otras instrucciones del programa. En adelante es misión exclusiva del canal la gestión de las operaciones que han sido delegadas.

Los controladores periféricos se encargan de gestionar uno o varios periféricos del mismo tipo, deben por lo tanto poder:

- Interpretar las instrucciones que reciben y/o entregan del/al computador.
- Controlar al periférico asociado, decodificar la operación que se le ordena ejecutar: escritura, lectura. Informar el estado del periférico.

Si dos o más procesadores solicitan simultáneamente el mismo periférico, la unidad de entrada/salida tiene varias opciones para decidir cuál de ellos se atenderá primero.

Periféricos de Comunicación	• Lectora de tarjetas: la tarjeta perforada es una cartulina estructurada en 80 columnas y 12 filas. Un caracter se representa en una columna por una cantidad de perforaciones. La velocidad de perforación oscila entre 50 y 500 tarjetas por minuto. La velocidad de lectura, en cambio, entre las 100 y las 2000 tarjetas por minuto.	
		• Cinta perforada: cinta de papel donde cada caracter se representa por una serie de perforaciones. Supera la capacidad de almacenamiento de la tarjeta.
		• Terminales: dispositivo periférico de una doble función de I/O. El teclado actúa como dispositivo de entrada y el monitor como uno de salida.
		• Impresoras: periférico de salida que se emplea para obtener listados en papel de determinado tipo de información.

		• Plotteres: periféricos de salida que a partir de un programa realiza los planos de un diseño. Se utiliza mucho en oficinas de ingeniería como salida final de los sistemas, diseño asistido y fabricación asistida por el computador.
Memorias Auxiliares	• Cinta magnética: primer tipo de memoria secundaria utilizada. Se emplean generalmente como medio de almacenamiento de copias de seguridad. Acceso secuencial. Cintas grandes (15 Gb), cartuchos, cassettes (200 a 300 Mb).	
		• Disco magnético: pieza de metal a la que se le a aplicado una capa magnética especial. Una unidad de disco tiene usualmente varios discos dispuestos verticalmente a una distancia de 2 o 3 cm.
		• Tambor magnético: variación del disco, a lo largo del tambor hay muchas cabezas de lectura y escritura *fijas*. Cada una puede leer o escribir en una pista. Tienen menor capacidad que los discos, pero su tiempo de acceso es menor.
		• Discos flexibles: aparecieron en el año '77. Los datos se almacenan en una lámina delgada de plástico circular y flexible, cuyo espesor no supera los dos nanómetros (0,002 mm). Esta lámina esta recubierta por una película metálica magnética. La lámina es muy flexible y débil y por lo tanto se encuentra dentro de una funda plástica, cuya estructura depende del tamaño del diskette: 5 ¼", 3 ½" ó los viejos de 8".

SOFTWARE

La computadora por sí sola no puede hacer ningún trabajo, solo es capaz de realizar la tarea que se le mande.

Para que pueda realizar ese trabajo es necesario que el hombre le dé instrucciones. Este conjunto de instrucciones, agrupadas en programas, constituye el software (también denominado *componente lógico del sistema informático*), que es pensado y realizado por el hombre. En su sentido más amplio, el software es el conjunto de programas que se utilizan en un ordenador.

Clasificación (*Software de Base y de aplicación* y *Software Dañino*):

Software de base y de aplicación: en el software de base están comprendidos los sistemas operativos y los lenguajes de programación, es el principal encargado de transformar la máquina *desnuda* en otra con facilidades y potencialidades propias.

El software de aplicaciones son los programas que desarrolla o adquiere un usuario para que un sistema de computación realice las tareas que requiere.

- Sistemas operativos (SO): puede definirse como un conjunto de programas que controlan la operación automática de un sistema de computación, actúa como interfaz entre el usuario y el hardware, con dos funciones complementarias:

- Para que sea una máquina fácil de programar y operar.
- Para administrar los recursos de dicho sistema, a fin de optimizar su funcionamiento, detectar errores e intentar salvarlos.

Sin embargo, el SO no lleva a cabo ninguna función útil por sí mismo, sino que proporciona un entorno.

Objetivo: proporcionar un entorno en el cual el usuario pueda ejecutar sus programas en forma cómoda (objetivo principal) y eficiente (secundario). Otro objetivo es ocultar al usuario los detalles del hardware (se logra a través de lo que se llama *máquina virtual*).

El SO puede dividirse en módulos que administran los *cuatro recursos* de un sistema de cómputo: CPU, MP, PERIFERICOS y ARCHIVOS, por lo tanto, el SO es un asignador de recursos (controla y coordina el uso del hardware entre los diversos programas de aplicación de los usuarios.). En *multiprogramación* (multitasking − multitarea) varios programas presentes en MP dan lugar a procesos que se disputan los cuatro recursos citados. El SO optimiza el funcionamiento del conjunto, oficiando de árbitro, posibilitando que al mismo tiempo, mientras que la CPU ejecuta uno de los programas del usuario, se realicen también procesos de E/S correspondientes a otros programas de usuario, almacenados en MP.

Los SO pueden clasificarse como sigue:

- SO Monotarea: sólo pueden controlar la ejecución de una tarea del usuario por vez. Simplemente cargan y ubican en MP la aplicación en curso, posibilitando que pueda usar los recursos del sistema. Cuándo parece un comando del tipo EXIT, el SO da por finalizada la aplicación, y se encarga de encadenar la siguiente. Ej. : DOS.
- SO Multitarea: permite la multiprogramación, cargando y ubicando en MP diversas aplicaciones, proporcionando a cada aplicación la posibilidad de usar los recursos del sistema. Ej. : UNIX, OS/2 y WINDOWS NT.

Existen dos clases de SO Multitarea:

- SO de tiempo compartido: (time sharing) que asignan a cada aplicación el tiempo que periódicamente, en cada ronda de procesos, tiene asignado para que la CPU ejecute parte del mismo.
- SO en tiempo real: cada tarea se ejecuta durante un tiempo que depende de determinados eventos que ocasionan la *conmutación* de una tarea a otra, volviendo más tarde a completarse el proceso de una

actividad inconclusa.

- Lenguajes de programación: con el fin de facilitar el trabajo del programador, surge la necesidad de que el ordenador entienda un lenguaje diferente al suyo. Un lenguaje de programación es un conjunto de símbolos, caracteres y reglas de utilización que permiten a la gente comunicarse con las computadoras.

Al igual que los lenguajes humanos, tales como el inglés o el español, los lenguajes de programación poseen una estructura (*gramática o sintaxis*) y un significado (*semántica*).

Los lenguajes de programación se pueden clasificar en dos grupos fácilmente identificables:

- Lenguajes de BAJO NIVEL: el procesador solo puede ejecutar instrucciones simples, llamadas *instrucciones de máquina*. Una instrucción de máquina es un conjunto de ceros y unos, un *programa máquina* es un conjunto de instrucciones de máquina escritas con ciertas reglas sintácticas que constituyen el *lenguaje máquina*. El programa escrito en lenguaje máquina es el único que entiende la computadora.

Una instrucción máquina seria: 1000110000110001.

Los lenguajes ensambladores para muchos autores constituyen el software de la segunda generación de los lenguajes de programación. La diferencia con el anterior reside en el medio en que se presentan las instrucciones para el programador. En lugar de las pesadas series de 1 y 0, el leguaje ensamblador utiliza símbolos reconocibles, llamados *mnemotécnicos* para representar las instrucciones.

- Lenguajes de ALTO NIVEL: permiten programar sin necesidad de conocer el funcionamiento interno de la máquina. Los lenguajes de alto nivel poseen una potencia considerable en sus instrucciones, de modo que una instrucción en lenguaje de alto nivel equivale a varias en lenguaje máquina. Así por ejemplo: M=A+B(i,j)+C*LOG(X) puede suponer muchas instrucciones, sin embargo la instrucción es fácilmente entendible por el programador.

Los lenguajes de alto nivel pueden ser compiladores e intérpretes. Cuando un ordenador ejecuta un programa en lenguaje intérprete, traduce las instrucciones paralelamente a su ejecución; mientras que si lo hace en lenguaje compilado, el programa antes de ser ejecutado es traducido previamente a código máquina.

Entre los lenguajes de alto nivel de uso más frecuente:

BASIC (Beginners All Porpuse Symbolic Instruction Code)

FORTRAN (FORmula TRANslator)

COBOL (Common Business−Oriented Language)

PASCAL

C

PL/1 (Programming Language 1)

ALGOL (ALGOrithmic Language)

LOGO

Software dañino (*virus*): el software dañino está constituido por los virus, que son programas. Un programa virus consta de dos partes. Una de ellas al ser ejecutada por la CPU, sirve para la autocopia del programa, con el fin de autoreproducirse tantas veces como pueda, para pasar de un computador a otro. La otra parte al ser ejecutada produce los daños para el que ha sido concebido el virus.

El daño de un virus puede tener como objetivo el borrado de información (archivos o discos completos), la alteración de tablas o parámetros que utiliza el sistema operativo, o directamente la modificación de archivos con programas del sistema operativo. También pueden destruir hardware, como ser obligar al cabezal del disco rígido que salte continuamente entre dos posiciones extremas (lo cual produce la pérdida de la información contenida en el disco), o intensificar la acción del haz de rayos catódicos, para producir un daño irreparable en la pantalla del monitor.

SISTEMAS DE NUMERACION

Las computadoras no utilizan el sistema de numeración decimal, sino el sistema de numeración en base dos (2), denominado sistema *binario*. En el sistema binario, cada dígito representa solamente dos posibles valores: 0 ó 1, note que estos valores también forman parte del sistema decimal. Sin embargo, muchas veces para especificar direcciones de memoria, utilizamos otro sistema de numeración: de base 8 (*Octal*), o de base 16 (*Hexadecimal*). El más utilizado en el mundo de las PC's es este último.

Cada dígito de un número hexadecimal representa 16 valores posibles, la afirmación anterior puede llevar a la conclusión de que cada dígito puede tener un valor del 0 al 15, pero esto llevaría a una gran confusión. Si uno lee el número 15, afirmará que está compuesto por dos dígitos, por lo tanto dicho número no es efectivo y se decidió utilizar los números del 0 al 9 y las letras de la A hasta la F, para representar los valores del 0 al 9 y del 10 al 15 respectivamente.

Debe recordarse que el sistema decimal es un sistema numérico *posicional*. En estos sistemas, cada símbolo, además del número de posiciones que representa considerado aisladamente, tiene un significado o peso distinto, según la posición que ocupa en el grupo de caracteres del que forma parte.

Si bien en la práctica apreciamos mecánicamente la magnitud de un número decimal con solo visualizarlo, sabemos que la cantidad de elementos que él simboliza se determina sumando los productos que se obtienen de multiplicar el valor individual de cada dígito por el *peso* (valor) de la posición que ocupa (unidades, decenas, etc.). Así:

1303=1x1000+3x100+0x10+3x1

Siendo:

10=1x10=101; 100=10x10=102; 1000=100x10=103; ...etc.

O sea que *el peso de cada posición es el de la anterior multiplicada por la base (diez), resultando así los pesos potencias crecientes de la base*.

¿Cuál es el valor posicional de cada dígito en los sistemas binario y hexadecimal?

Por ejemplo, los 4 bits que forman el número binario 11012 tendrán los pesos (expresados en decimal) que se indican en negrita arriba de cada bit:

8 4 2 1

1 1 0 12;	Siendo: 2=1x2; 4=2x2; 8=4x2; etc.

Asimismo, el valor de cada posición es una potencia de la base. En este caso, que la base es dos, se tiene:

1=20; 2=21; 4=22; 8=23; etc.

Análogamente, en hexadecimal el peso (en decimal) de cada dígito se especifica sobre el mismo en negrita para el número 1A3F16:

4096 256 16 1	Siendo:	16=1x16;		256=16x16;		4096=256x16;	etc.
1 A 3 F16	y también:	1=160;	16=161;	256=162;		4096=163; etc.	

Conversiones Entre Bases Numéricas

¿Cómo se convierten a base diez los números de otras bases?

Teniendo en cuenta el peso (en decimal) de cada dígito, se los suma, obteniendo como resultado el valor representado en base diez.

8 4 2 1	
1 1 0 12=	(1x8+1x4+0x2+1x1)10= (8+4+1)10= 1310

La suma de los productos de cada dígito por su peso, es la misma que se realiza en el sistema decimal cuando se descompone un número en unidades, decenas, centenas, etc.

1000 100 10 1	
8 2 0 710=	(8x1000+2x200+0x10+7x1)10= (8000+200+7)10

Regla práctica para pasar de binario (o de cualquier base) a decimal:

Ejemplos:

16 8 4 2 1	
1 0 1 1 02=	(16+4+2)10= 2210
16 1	
1 016=	(1x16)10= 1610
512 64 8 1	
1 0 3 78=	(1x512+3x8+7x1)10= 54310

¿Cómo se hace la conversión de decimal a otras bases?

13 2

1 6 2

0 3 2

1 1

de Decimal a Binario

1 1 0 12= 1310

16140 16

0140 1008 16

12=C 48 63 16

0 15=F 3=3

de Decimal a Hexadecimal

3 F 0 C16= 1614010

El mismo procedimiento es el que se utiliza para pasar de Decimal a Octal.

¿Cómo se pasa directamente de Binario a Hexadecimal y viceversa?

El pasaje de Binario a Hexadecimal y en sentido contrario, se hace mecánicamente y en forma directa, sin necesidad de realizar ningún cálculo.

Cuando una base es potencia de otra, como lo son las bases 16 y 2 (16=24) se demuestra, que simplemente se reemplaza cada dígito por su equivalente numérico en cada base, basta tener en cuenta las equivalencias de los cuartetos binarios y los números del **0** al **F**.

Regla práctica para pasar de Binario a Hexadecimal:

Ejemplos: convertir a Hexadecimal los siguientes números Binarios 10110010, 1010000011110111, 1100011101.

1 0 1 1 0 0 1 0	=B216
B 2	
1 0 1 0 0 0 0 1 1 1 1 0 1 1 1	=A0F716
A 0 F 7	
0 0 1 1 0 0 0 1 1 1 0 1	=31D16
3 1 D	

Regla práctica para pasar de Hexadecimal a Binario:

Ejemplos: convertir los números Hexadecimales B0A y 10.

B0A16	= 1 0 1 1 0 0 0 0 1 0 1 02
	B 0 A
1016	= 0 0 0 1 0 0 0 02
	1 0

La siguiente tabla sirve para realizar conversiones entre los sistemas Hexadecimal, Binario y Decimal, sin la necesidad de realizar tediosos cálculos.

Conversión Hexadecimal/Binario/Decimal

N° Hexa	N° Binario	Dígito Decimal 000x	Dígito Decimal 00x0	Dígito Decimal 0x00	Dígito Decimal x000
0	0000	0	0	0	0
1	0001	1	16	256	4.096
2	0010	2	32	512	8.192
3	0011	3	48	768	12.288
4	0100	4	64	1.024	16.384
5	0101	5	80	1.280	20.480
6	0110	6	96	1.536	24.576
7	0111	7	112	1.792	28.672
8	1000	8	128	2.048	32.768
9	1001	9	144	2.304	36.864
A	1010	10	160	2.560	40.960
B	1011	11	176	2.816	45.056
C	1100	12	192	3.072	49.152
D	1101	13	208	3.328	53.248
E	1110	14	224	3.584	57.344
F	1111	15	240	3.840	61.440

Para explicar el funcionamiento de esta tabla, utilizaremos un par de ejemplos, realizando conversiones entre los diferentes sistemas.

Para convertir el número hexadecimal BC16 al sistema decimal debe hacer lo siguiente: moverse en la tabla a la fila donde está la letra B, en la primer columna y buscar el valor en la columna correspondiente a las decenas, en este caso (columna 00x0). Luego moverse en la fila donde se encuentra C en la primer columna, y buscar el valor del dígito decimal en la primer columna (que es la de las unidades, 000x). Sumando los dos valores (176+12), se obtendrá el número decimal correspondiente (188).

Para convertir el número hexadecimal BC16 al sistema binario se debe hacer lo siguiente: moverse en la tabla a la fila C, en la primer columna y buscar el valor correspondiente a C en la columna de números binarios. Repetir el proceso anterior pero en la fila B. Una vez convertidos todos los dígitos hexadecimales en sus valores binarios (siempre tomando los dígitos de derecha a izquierda), se deben anotar los valores binarios uno detrás del otro y se obtendrá el número binario correspondiente. Para el ejemplo, el número binario correspondiente al número hexadecimal BC16 será 10111100.

UNIDAD

TRES

HEURISTICA y ALGORITMACION

ALGORITMO

Definición: es una serie de operaciones detalladas a ejecutar paso a paso, y que conducen a la resolución de un problema. En otras palabras, *un algoritmo es un conjunto de reglas para resolver una cierta clase de problema o una forma de describir la solución de un problema*. Una receta de cocina para hacer cordero asado es un algoritmo.

Un algoritmo es el medio por el que se explica como debe resolverse un problema, mediante aproximaciones paso a paso.

Para describir algoritmos de computadoras se han diseñado lenguajes de programación. Cada una de las acciones de las que consta un algoritmo se llamará sentencia y estas deben ser escritas en términos de cierto lenguaje comprensible para el ejecutor (máquina), que es el lenguaje de programación.

El conjunto formado por la representación de datos utilizada y el algoritmo en sí, se conoce usualmente con el nombre de *programa*. En esencia, un programa *es la descripción de un proceso en un cierto lenguaje*, o dicho de otra manera: la secuencia de acciones entendibles por la computadora que conducen a realizar una tarea determinada y el correcto tratamiento de los datos.

Diagramas De Flujo:

La resolución de todo problema exige tres grandes elementos: *datos de problema, resultados solicitados* y *algoritmo de resolución*.

Los datos del problema son información de partida. Los resultados constituyen la información de salida y estarán íntimamente relacionados con la información de entrada. El algoritmo de resolución es el conjunto de operaciones (matemáticas, lógicas, etc.) o manipulaciones que se deben realizar con los datos para llegar a la obtención de los resultados.

Los algoritmos se suelen representar en forma narrativa, pero cuando tienen su aplicación más directa es cuando se convierten en *diagramas o gráficos de programación*, y son la representación gráfica de la solución del problema que se desea mecanizar. El *diagrama de programación* es la representación gráfica de unos procedimientos y de la secuencia u orden en que deben ejecutarse; en resumen *la representación gráfica de la solución de un problema o de un procedimiento*.

Símbolos utilizados en los diagramas:

Símbolos Principales	*Función*
	Terminal (representa el comienzo, *inicio* y el final, *fin* de un programa. Puede representar también una parada o interrupción programada que sea necesario realizar en un programa).
	Entrada/Salida (cualquier tipo de operación de introducción de datos en la memoria desde los *periféricos de entrada* o registro de la información procesada en un *periférico de salida*).
	Proceso (cualquier tipo de operación definida que pueda originar cambio de valor, formato o posición de la información almacenada en memoria: operaciones aritméticas, etc.).
Si No	Decisión (indica operaciones lógicas de comparación entre datos − normalmente dos − y en función del resultado de la misma determina cuál de los distintos caminos alternativos del programa se debe seguir; normalmente tiene dos salidas − respuestas SI o NO − pero puede tener tres o más según los casos).
Símbolos Principales	*Función*
Salidas Múltiples	Decisión múltiple (en función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado).

	Conector (sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama).
	Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones).
	Línea conectora (sirve de unión entre dos símbolos).
	Conector (conexión entre dos puntos del diagrama situados en páginas diferentes).
	Llamada a subrutina o a un proceso predeterminado (una subrutina es un módulo independiente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y regresa al programa principal).
	Pantalla (se utiliza en ocasiones en lugar del símbolo de E/S).
	Impresora (se utiliza en ocasiones en lugar del símbolo de E/S).
	Teclado (se utiliza en ocasiones en lugar del símbolo de E/S).

Es conveniente que los diagramas (cuando así lo exijan) utilicen notaciones normalizadas para operaciones de cualquier tipo, como las que se reflejan en la siguiente tabla:

Operaciones Aritméticas	*Significado*
	Movimientos de unas posiciones de memoria a otras o cambios en campos de información.
+	Suma.
−	Resta.
*	Multiplicación.
/	División.
O bien "	Exponenciación.

Operación De Relación	*Significado*
<	Menor que.
=	Igual
>	Mayor que.
"	Mayor o igual que.
"	Menor o igual que.
<>	Diferente (menor o mayor que).
"	No igual que (diferente).

ESTRUCTURAS LOGICAS BASICAS

Concepto previo: Diagramas de flujos estructurados: en general contará con las siguientes fases:

- INICIO.
- ENTRADA DE DATOS.
- PROCESO.
- SALIDA DE DATOS.
- FIN.

Aunque las 5 partes existen prácticamente siempre, el orden de las fases 2, 3 y 4 puede variar e incluso confundirse unas fases con otras.

Las estructuras lógicas básicas necesarias para confeccionar un programa se reducen a tres: *secuenciales*, *selectivas* y *repetitivas*.

- *Estructuras secuenciales*: constan esencialmente de sucesivos pasos, uno detrás de otro.

- *Estructuras selectivas*: requieren una prueba o comparación de ciertas condiciones seguidas de rutas alternativas en el programa.

Falso Verdadero Verdadero

No Si Si

No Falso

- *Estructuras repetitivas*: cada una de las estructuras tiene una sola flecha de entrada y una sola flecha de salida, y cada una de ellas es legible de arriba hacia abajo (diseño *top down*).

Falso Verdadero

Verdadero Falso

DIAGRAMAS LINEALES

Son aquellos en los que no existen instrucciones de bifurcación, también se denominan secuenciales, dado que siguen exactamente la secuencia especificada.

Es normal que estos programas consten de las siguientes fases:

- Lectura o entrada de datos.
- Proceso.
- Impresión o salida de resultados.

Ejemplo: diagrama de flujo para lectura de tres números M, N, P en una sola operación e impresión de los mismos y de su suma en una impresora.

Datos de entrada: M, N, P.

Proceso: Suma: = M + N + P.

Salida de resultados: Impresión de M, N, P y Suma.

DIAGRAMAS CICLICOS

Son aquellos diagramas en los que un grupo de instrucciones se ejecuta un número determinado de veces −de modo cíclico− hasta que se cumple una cierta condición que indica el fin de las ejecuciones de dichas instrucciones. El conjunto de instrucciones que se repiten cíclicamente se denomina *bucle*, *lazo* o *ciclo*.

La estructura de un diagrama cíclico suele constar de los siguientes bloques o fases:

- Entrada de datos e instrucciones previas.
- Lazo o bucle (conjunto de instrucciones que se repiten y ejecutan un número determinado de veces).
- Instrucciones finales o resto del proceso.
- Salida de resultados.

Estos programas precisan instrucciones de bifurcación o control que permitan la entrada y salida del bucle.

Ejemplo: supongamos que los datos del ejemplo anterior provienen de un fichero o archivo que tiene un número determinado de grupos de tres números, y que se desea obtener la suma e impresión de todos los grupos.

En el caso de tratarse de un fichero, estos suelen contener siempre un campo en su última ficha que se denomina fin de fichero (FF) y que cuando se encuentra en la lectura supone que se ha alcanzado el final del fichero y por consiguiente no se debe seguir leyendo.

El diagrama de flujo modificado es:

Si

No

DIAGRAMAS ALTERNATIVOS

Son aquellos que permiten la ejecución de diferentes operaciones, dependiendo de que se cumplan (o no) determinadas condiciones que se producen en los datos de entrada o durante el proceso. Según la condición que se cumple se realiza una serie de instrucciones diferentes.

Ejemplo: ingresar un número y decir si éste es cero, par o impar.

No Si

No Si

REGLAS PARA LA CONSTRUCCION DE DIAGRAMAS DE FLUJOS:

El proceso para la construcción de un diagrama de flujo no supone un método rígido, pero se pueden enunciar unas reglas de tipo general, útiles para cualquier nivel:

- Todo diagrama debe tener un principio (INICIO) y un fin, al objeto de que pueda ser utilizado como submódulo de otro módulo de nivel superior.
- Las líneas de conexión o de flujo deben ser siempre rectas y, si es posible que sean sólo verticales y horizontales (no cruzarse ni inclinarse), para conseguir lo anterior se debe recurrir a conectores, numerados convenientemente y sólo en los casos estrictamente necesarios.
- Las líneas que enlazan los símbolos entre sí deben estar todas conectadas. Cada línea o flecha debe entrar en un bloque, en un símbolo de decisión, terminar en FIN o unirse a una flecha.
- Se deben dibujar todos los símbolos de manera que se pueda seguir el proceso visualmente de arriba hacia abajo (diseño *top down*) y de izquierda a derecha.
- Realizar un gráfico claro y equilibrado, procurando que el flujo central del diagrama sea la parte central de la hoja del papel.
- Evitar la utilización de terminología específica de un lenguaje de programación o máquina, sobre todo en las expresiones donde se tiene tendencia a ello.
- Se debe dejar un bloque o dos de procesos libres al comienzo del diagrama, para reservar posiciones de

memoria para variables, acumuladores, inicialización de subíndices de listas y tablas, etc.

- Indicar con comentarios al margen, las variables utilizadas y su descripción, procurando no abusar de su uso. Diferenciar las variables propias del proceso de las pseudovariables o variables ficticias (contadores, conmutadores, etc.).
- En las operaciones lógicas recurrir preferentemente a la lógica positiva que a la lógica negativa (es siempre más claro si A = B que si no es A <> B).
- A cada bloque o símbolo se accede por arriba y/o por la izquierda y se sale por abajo y/o por la derecha. Las entradas pueden ser varias pero la salida es única, excepto en los casos de símbolos de decisión.
- Realizar todas las anotaciones o comentarios marginales del diagrama para que éste sea comprensible no sólo por la persona que lo ha elaborado sino también por cualquier persona ajena al mismo, sobre todo con el paso del tiempo y para cuando se necesite una modificación o actualización del diagrama.
- Siempre que sea posible es conveniente que el diagrama no sobrepase la página; si no es posible, numerar adecuadamente las hojas del diagrama y utilizar los correspondientes conectores de páginas que indiquen sin dudas la dirección correcta del flujo (de donde viene y a donde se dirige).

VECTORES

Un vector es un conjunto ordenado de elementos que contiene un número *fijo* de ellos. Los vectores son *matrices unidimensionales*, sus elementos deben ser del mismo tipo (enteros, reales, caracteres).

Un vector es:

(2, 3, 4, 8, 9, 10, 15, 17)

y representa una lista de elementos.

La *longitud* de un vector es el número total de elementos que posee. Cada elemento de un vector se representa por el nombre del vector y un índice o subíndice. Para el ejemplo anterior, si X es el nombre del vector, los elementos son:

X1=2, X2=3, X3=4, X4=8, X5=9, X6=10, X7=15, X8=17

El índice o subíndice también suele escribirse entre paréntesis o corchetes, ya que así se escriben normalmente en los lenguajes de programación. Entonces X1 se representaría: X(1) o X[1].

Una representación gráfica del vector anterior puede ser la siguiente:

2	3	4	8	9	10	15	17

X[1] X[2] X[3] X[4] X[5] X[6] X[7] X[8]

MATRICES

Una matriz con un solo subíndice se denomina *unidimensional*. Una matriz con dos o más índices se llama *multidimensional*. El número de índices necesario para especificar un elemento de una matriz se denomina *dimensión*.

Las matrices más útiles son las de dos dimensiones, que se asemejan a una tabla o a un conjunto de filas (primer índice) y columnas (segundo índice).

Los elementos de una matriz de dos dimensiones se representan por dos expresiones separadas por una coma y encerradas entre corchetes o paréntesis.

A[I, J] o A(I, J)

Se refieren al elemento de la fila I y la columna J.

	Columna1	Columna2	Columna3	Columna4	
Fila1					
Fila2					A[2,3]
Fila3					A[3,1]
Fila4					

Matriz de dos dimensiones

La matriz es un conjunto de elementos, todos del mismo tipo, en los que el orden de los elementos es significativo, y en el que se necesita más de un índice para definir un elemento. Las matrices pueden ser de dos, tres o cuatro dimensiones.

UNIDAD

CUATRO

SISTEMA OPERATIVO

ENTORNOS DE TRABAJO

En Red

MULTIPROCESAMIENTO: surgen a raíz de la gran cantidad de información que no puede ser tratada por un solo procesador. Se utilizan varios procesadores que trabajan conjuntamente, distribuyendo sus cálculos y compartiendo la información. Se descomponen los algoritmos en subalgoritmos, cada uno tratado por un procesador, al final se juntan los datos y se obtiene al resultado final.

Existen dos esquemas para la construcción de sistemas de multiprocesamiento:

- Sistemas Fuertemente Acoplados: los procesadores comparten la memoria principal y el reloj. La comunicación se realiza a través de la memoria (ej.: servidor con terminales bobas).
- Sistemas Débilmente Acoplados (Sistemas Distribuidos): los procesadores no comparten ni la memoria ni el reloj, cada procesador cuenta con su memoria local. La comunicación se realiza a través de redes.

Razones para la construcción de sistemas distribuidos:

- Compartimiento de recurso; el usuario puede acceder a los recursos del sistema.
- Compartimiento de cargas.
- Aceleración de los cálculos; se dividen los cálculos entre las instalaciones y se ejecutan concurrentemente.
- Confiabilidad; si falla una instalación, las demás continúan operando.
- Comunicación.

En Computadoras Aisladas

SISTEMAS MONOUSUARIO: (computadoras personales) no son ni multiusuario ni multitarea. El hardware es de bajo costo.

Objetivo: maximizar la utilización de la CPU y de los periféricos, brindar comodidad y rapidez de respuesta a los usuarios.

DOS

El DOS es uno de los sistemas operativos de microcomputadoras más difundidos del mercado. Está diseñado para ser utilizado en computadoras IBM y compatibles.

DOS es la abreviación de las palabras inglesas *Disk Operating System*, que significa Sistemas Operativo de Disco. Como su nombre lo indica, busca fundamentalmente facilitar el uso de discos para manipular grandes volúmenes de información.

En el mercado existen varias compañías que ofrecen diferentes tipos de DOS. Entre los más conocidos están el PC−DOS desarrollado por IBM, el DR−DOS desarrollado por Digital Research y el MS−DOS desarrollado por Microsoft.

Comandos

Uno de los medios que provee el sistema operativo para comunicarse con el computador son los *comandos*. Un comando es una orden que debe ser digitada en el teclado del computador para ejecutar una tarea específica.

El lugar donde se introducen los comandos es la línea de comandos. La línea de comandos puede ser usada cuando se presenta en pantalla el símbolo del sistema seguido del cursor. El cursor es una pequeña línea intermitente que aparece en la línea de comandos. Indica la posición del próximo carácter que será digitado desde el teclado. A medida que se va digitando un comando, el cursor se va desplazando hacia la derecha.

C:\>_

Símbolo del sistema Cursor

Línea de comandos:

C:\>_

Cuando el símbolo del sistema aparece en la pantalla, significa que el computador está listo para recibir una orden por medio de un comando. Para que el comando sea ejecutado debe presionarse la tecla ENTER (INTRO) después de digitarlo.

Anatomía de un Comando: un comando es una palabra clave, que puede ser una palabra completa o una abreviatura de una o más palabras; puede estar seguida de elementos que especifican información adicional necesaria para la ejecución del comando.

En un comando pueden distinguirse los siguientes elementos:

- El nombre de un comando.
- Los parámetros del comando.
- Los modificadores del comando.

- Nombre: identifica la tarea o acción que se desea ejecutar. Es el elemento del comando que debe ser especificado en primer lugar. Por ejemplo: *ver*, *type*, *copy*, etc.

- Parámetros: especifican información adicional requerida por el comando. Los parámetros generalmente identifican a los archivos, directorios, unidades de disco, etc. que están involucrados en la ejecución del comando. Los parámetros se escriben luego del nombre del comando. Por ejemplo: el comando *type* examina el contenido de un archivo, así el nombre del archivo constituye un parámetro del comando. Si se desea examinar el contenido del archivo llamado RESUMEN, se deberá escribir el siguiente comando:

Type resumen

Cuando un comando requiere de más de un parámetro, estos deberán ser especificados en el orden requerido, si el orden es alterado, la ejecución del comando no producirá el resultado esperado.

Existen comandos que la especificación de algunos de sus parámetros es opcional. Si no se especifica, el comando considerará ciertos valores, como valores por defecto.

- Modificadores: permite alterar la forma en que un comando va a ejecutar una acción. Un modificador es especificado en un comando por medio de una barra diagonal (/) seguida de una letra o número. Por ejemplo: el comando *dir* permite listar los nombres de los archivos que están en el directorio. Cuando la lista es muy larga, algunos de los nombres no aparecerán en la pantalla y serán desplazados por otros nombres. Si se especifica el modificador */P* al escribir el comando *dir*, la lista de nombres se desplegará pantalla por pantalla.

COMANDOS DEL SISTEMA OPERATIVO: DOS

Convenciones que se utilizarán:

El nombre del comando se presenta en color.

Los parámetros se presentan en negro. Los que se usan con mayor frecuencia son:

- Unidad: nombre de la unidad de diskette o unidad de disco duro. Siempre es una letra seguida de dos puntos (:).
- Camino: la ruta de acceso que deberá utilizar el sistema operativo para localizar un archivo o directorio.
- Archivo: nombre de un archivo.
- Texto: cadena de caracteres. Generalmente encerradas entre comillas ( ).
- Fuente: ubicación de archivos que serán usados como información de entrada requerida para la ejecución de un comando. Puede ser: nombre de archivo, nombre de directorio, identificación de la unidad de disco o alguna combinación de estos.
- Destino: la ubicación de los resultados producidos por la ejecución de un comando. Puede ser: nombre de archivo, nombre de directorio, identificación de la unidad de disco o alguna combinación de estos.

Los parámetros y modificadores encerrados entre corchetes [ ] son opcionales.

Los símbolos como ; , = , / , : , , , etc., deben ser escritos igual a como aparecen en la sintaxis del comando.

En ocasiones los valores que puede tomar un parámetro están restringidos. En esos casos se usa el símbolo | para separar las diferentes alternativas. Debe escribirse un solo valor sin el símbolo.

Debido a que existen varias versiones del sistema operativo MS−DOS, es posible que algún comando de los aquí explicados no sea soportado en alguna versión.

Para el manejo de directorio

CHDIR o CD

USO: cambia el directorio actual a un directorio diferente o presenta el nombre del directorio actual.

VERSION: 2.0 y siguientes.

SINTAXIS:

CHDIR [unidad:] [camino]

CHDIR [..]

CD [unidad:] [camino]

CD [..]

COMENTARIOS: al ejecutar el comando con dos puntos (..), se cambia al directorio padre del directorio actual.

Si se ejecuta el comando sin parámetros, se presenta la unidad y el nombre del directorio actual.

Ejemplos:

Para cambiar el directorio actual al directorio DATOS, ejecutar:

CD DATOS

Para identificar el directorio actual, ejecutar:

CD

MKDIR o MD

USO: crea un directorio.

VERSION: 2.0 y siguientes.

SINTAXIS:

MKDIR [unidad:] nombre

MD [unidad:] nombre

COMENTARIOS: el parámetro unidad indica la unidad donde será creado el nuevo directorio, si no se especifica este parámetro se considera la unidad actual.

El parámetro nombre identifica el nombre del directorio nuevo.

Ejemplos: para crear el directorio llamado DATOS en el directorio raíz de la unidad C, escribir:

MKDIR C: \DATOS

La otra forma consiste en cambiar del directorio actual al directorio raíz, y luego escribir

MKDIR DATOS

RMDIR o RD

USO: permite eliminar directorios.

VERSION: 2.0 y siguientes.

SINTAXIS:

RMDIR [unidad:] nombre

RD [unidad:] nombre

COMENTARIOS: los parámetros unidad y nombre identifican al directorio que se va a eliminar. El directorio no debe contener ningún archivo (inclusive ocultos y del sistema) o subdirectorio, en caso contrario el comando despliega un mensaje de error.

Ejemplo: para eliminar el directorio DATOS, primero debe verificarse que el directorio esté vacío, para esto ejecutar el siguiente comando:

DIR DATOS /A

Si el directorio datos está vacío se presenta en pantalla dos entradas al directorio de archivos, identificados como . y ..:

El volumen en unidad C no tiene etiqueta

Número de serie del volumen es 1D17−8C4C

Directorio de C: \DATOS

<DIR> 07−27−94 1:03 PM

<DIR> 07−27−94 1:03 PM

2 archivo (s) 0 bytes

298442752 bytes libres

Una vez verificado que el directorio DATOS está vacío, ejecutar el siguiente comando, desde el directorio padre de DATOS, para eliminar el directorio:

RMDIR DATOS

TREE

USO: permite desplegar la estructura de los directorios ubicados en un disco o en camino de acceso.

VERSION: 3.2 y siguientes.

SINTAXIS:

TREE [unidad:] [camino] [/F] [/A]

COMENTARIOS: el parámetro unidad identifica al disco cuya estructura de directorios será desplegada. Si no se especifica este parámetro, se considerará a la unidad actual.

El parámetro camino permite especificar un camino de acceso para el cual será presentado su estructura de archivo.

El modificador /F muestra junto a la estructura, el nombre de los archivos que están en cada directorio.

El modificador /A obliga al comando a usar caracteres de texto para mostrar la estructura de directorios. Si, por el contrario, el modificador no es especificado, el comando usa caracteres gráficos (caracteres ASCII del 128 al 255).

Ejemplo: supóngase que se dispone de un diskette, en la unidad a:, que tiene los siguientes subdirectorios:

DATOS

DATOS\ANO92

DATOS\ANO93

DATOS\ANO94

Para desplegar en pantalla la estructura de directorios en disco, ejecutar el siguiente comando:

TREE A:

La estructura de directorios se presenta en pantalla:

A:		
	DATOS	
		ANO92
		ANO93
		ANO94

DELTREE

USO: elimina toda la bifurcación de directorios sin que Ud. tenga primero que vaciar cada uno de los

directorios.

VERSION: 6.0 y siguientes.

SINTAXIS:

DELTREE [unidad:] [camino]

COMENTARIOS: los parámetros unidad y camino, realizan las mismas acciones que el comando anterior.

XCOPY

USO: permite copiar archivos y directorios completos.

VERSION: 3.2 y siguientes.

SINTAXIS:

XCOPY fuente [destino] [/A | /M] [/D: fecha] [/P] [/S] [/E] [/V] [/W]

COMENTARIOS: el parámetro fuente identifica la ubicación y nombre del archivo o grupo de archivos que van a ser copiados. Puede especificarse la unidad de disco, el directorio y el nombre del archivo.

El parámetro destino identifica la ubicación y nombre del archivo o grupos de archivos en donde residirán las copias de los archivos especificados en parámetro fuente.

El modificador /A copia solamente aquellos archivos cuyo atributo de archivo modificado esté activado. El atributo de archivo modificado permanece activo después de la copia.

El modificador /M copia solamente aquellos archivos cuyo atributo de archivo modificado esté activado. El atributo de archivo modificado es desactivado después de efectuar la copia.

El modificador /D: fecha copia solamente aquellos archivos cuya fecha de modificación sea igual o superior a la fecha especificada como parámetro.

El parámetro /P permite confirmar la copia de cada archivo en el sitio de destino.

El modificador /S considera todos los subdirectorios del directorio origen para efectuar la copia. No se consideran aquellos directorios que estén vacíos. Al finalizar la copia, el destino tendrá una estructura de directorios idéntica a la de origen.

El modificador /E permite considerar los directorios vacíos cuando se copian subdirectorios. Este modificador debe ser especificado junto con el modificador /S.

El modificador /V verifica que los datos sean copiados libres de errores.

El modificador /W permite introducir una pausa antes de iniciar una copia de archivos. Para iniciar el proceso de copia, se puede presionar cualquier tecla. Para abortar la operación se presiona la siguiente combinación de teclas: CONTROL+C o CONTROL+BREAK.

El comando XCOPY no copia archivos que tengan activo el atributo de archivo oculto o el atributo de archivo del sistema.

Para el manejo de archivos

APPEND

USO: permite el acceso a archivos que se encuentran en directorios diferentes al directorio actual de trabajo. Por defecto cuando se solicita el acceso de un archivo (sin especificar el directorio) el sistema operativo solo examina el directorio actual de trabajo.

VERSION: 3.2 y siguientes.

SINTAXIS:

APPEND [[unidad:] camino [;...]] [/X[:ON | :OFF]] [/PATH][:ON |:OFF] [/E]

COMENTARIOS: los parámetros unidad y camino especifican el directorio que va a ser incluido para realizar la búsqueda de archivos.

Se puede incluir más de un directorio al mismo tiempo, reparando cada camino por un punto y coma (;).

Si se ejecuta el comando sin parámetros (APPEND ;) se muestra en pantalla los directorios de búsqueda actualmente definidos.

El modificador /X especifica cuando usar los directorios definidos por el comando. /X:ON usa los directorios definidos para acceder a los archivos de datos así como para ejecutar programas. /X:OFF usa los directorios definidos para acceder solamente a archivos de datos. El valor por defecto de este modificador es OFF.

En ciertas ocasiones, cuando se especifica el nombre de un archivo, se anexa información acerca del directorio en el que se encuentra. Si /PATH:OFF es especificado, se usan los directorios definidos solo para acceder a archivos que no incluyen información sobre el directorio en donde está ubicado. Si /PATH:ONN es especificado, se utilizan los directorios definidos para acceder a cualquier archivo. El valor por defecto de éste modificador es ON.

El modificador /E, cuando es especificado, almacena una copia de lo directorios definidos en una variable de ambiente llamada APPEND. Este modificador puede ser usado solamente la primera vez que se ejecuta el comando.

Ejemplo: el siguiente comando establece al directorio C:\DATOS como directorio de búsqueda:

APPEND C:\DATOS

De esta forma se le indica al sistema operativo que si no consigue un archivo en el directorio actual, tratar de ubicarlo en el directorio C:\DATOS. Si posteriormente se ejecuta el siguiente comando:

APPEND

El siguiente mensaje aparece en pantalla:

APPEND = C:\DATOS

ATTRIB

USO: despliega o altera los atributos de uno a más archivos. Los atributos son características particulares de los archivos que restringen su uso de diferentes formas.

VERSION: 3.0 y siguientes.

SINTAXIS:

ATTRIB [+R] [+A | −A] [+S | −S] [+H | −H] [[unidad:] [camino] archivo] [/S]

COMENTARIOS: los parámetros unidad, camino y archivo especifican el o los archivos cuyos atributos van a ser consultados o alterados.

El modificador +R activa el atributo que establece que el archivo es de sólo lectura, por lo que no podrá ser ni modificado ni eliminado. El modificador −R desactiva el atributo de sólo lectura.

El modificador +A activa el atributo de archivo modificado. Esto permite llevar un control para hacer procedimientos de respaldo selectivo, en que los archivos son copiados solamente si fueron modificados desde la última vez que se efectuó un respaldo. Este parámetro es usado por comandos BACKUP, RESTORE y XCOPY. El modificador −A desactiva el atributo de archivo modificado.

El modificador +S activa el atributo de archivo del sistema. Los archivos con este atributo no son presentados en el directorio. El modificador −S desactiva el atributo de archivo del sistema.

El modificador +H activa el atributo de archivo escondido, entonces no será mostrado por el sistema operativo en la lista de archivos de un directorio. El archivo sólo podrá ser usado si se conoce con exactitud su nombre. El modificador −H desactiva el atributo de archivo escondido.

El modificador /S hace que el comando procese los archivos que se encuentran en todos los subdirectorios del camino especificado como parámetro.

Este comando permite el uso de comodines para procesar más de un archivo en la misma ejecución. Los comodines deben usados en el parámetro archivo.

Ejemplos: supóngase que se tiene un archivo llamado PERSONAL.DAT en C:\DATOS. Para ver los atributos del archivo, se debe ejecutar el comando:

ATTRIB C:\DATOS\PERSONAL.DAT

Para activar el atributo de sólo lectura del archivo:

ATTRIB +R PERSONAL.DAT

Para activar el atributo de sólo lectura de todos lo archivos con extensión DAT en el directorio C:\DATOS y en todos sus subdirectorios se utilizan comodines en la ejecución del comando, junto con el modificador /S.

ATTRIB +R C:\DATOS*.DAT /S

COPY

USO: copia un archivo o grupo de archivos a otro directorio en un mismo disco u otro disco.

VERSION: 1.0 y siguientes.

SINTAXIS:

COPY [/A | /B] fuente [/A | /B] [+ fuente [/A | /B]] [+ ...]] [destino [/A | /B]] [/V]

COMENTARIOS: el parámetro fuente indica la ubicación y nombre del archivo o grupo de archivos que van a ser copiados. Puede especificarse la unidad de disco, el directorio y el nombre del archivo.

El parámetro destino identifica la ubicación y nombre del archivo o grupo de archivos en donde residirán las copias de los archivos especificados en el parámetro fuente. Los archivos en el destino, que tengan nombres en común con algunos de la fuente, son reemplazados por el archivo de la fuente.

El modificador /A indica que tanto el archivo que se procede como los archivos que suceden al modificador, son de tipo ASCII. El modificador afecta a todos los archivos que le sucedan hasta que encuentre un modificador /B. Cuando es especificado el modificador /A para algún archivo fuente, el comando COPY sólo copia el archivo hasta encontrar un carácter de fin de archivo (CONTROL+Z). Ahora, si es para un archivo destino, se agrega al final del archivo, un carácter de fin de archivo.

El modificador /B indica que tanto el archivo que le precede como los archivos que le suceden, son de tipo binario. El modificador afecta a todos los archivos que le suceden hasta encontrar un modificador /A.

El modificador /V verifica si los datos fueron copiados libre de errores.

El comando COPY puede ser usado para unir varios archivos. Para esto debe especificarse varios parámetros fuente, intercalando entre ellos el signo + . El archivo especificado en destino contendrá una copia de la unión de todos los archivos. El comando considera que los archivos a unir son del tipo ASCII, a menos que se indique lo contrario.

Para copiar de manera rápida todos los archivos y subdirectorios de un directorio, es recomendable usar el comando XCOPY.

Ejemplo: para copiar un archivo llamado NUEVO.DAT, que se encuentra en el directorio C:\NUEVO, al directorio C:\DATOS, ejecutar el siguiente comando:

COPY C:\NUEVO\NUEVO.DAT C:\DATOS

El comando responde con el siguiente mensaje:

1 archivo (s) copiado (s)

Si se desea copiar todos los archivos con extensión DAT del directorio C:\DATOS al directorio raíz de la unidad A, escribir lo siguiente:

COPY C:\DATOS*.DAT A:\

El comando despliega en pantalla una lista con los archivos que va copiando, al final se presenta el número total de archivos copiados.

DEL o ERASE

USO: elimina un archivo o un grupo de archivos de un disco.

VERSION: 1.0 y siguientes.

SINTAXIS:

DEL [unidad:] [camino] archivo /P

ERASE [unidad:] [camino] archivo /P

COMENTARIOS: los parámetros unidad, camino y archivo identifican al archivo o grupo de archivos que se van a eliminar. Se puede hacer uso de comodines para especificar un grupo de archivos.

El modificador /P hace que el comando confirme la eliminación de cada archivo.

Ejemplo: para eliminar los archivos con extensión DAT, en el directorio C:\DATOS, escribir:

DEL C:\DATOS*.DAT

Si en la situación anterior, se desea que se pregunte por cada archivo para confirmar su eliminación se debe escribir lo siguiente:

DEL C: \DATOS*.DAT /P

DIR

USO: permite examinar la lista de archivos y subdirectorios, de un directorio. Puede mostrar, por cada archivo, su tamaño, el día y la hora de su última modificación.

VERSION: 1.0 y siguientes.

SINTAXIS:

DIR [unidad:] [camino] [archivo] [/P] [/W] [/A[[:]atributos]] [/O [[:]orden]] [/S] [/B] [/L]

COMENTARIOS: el nombre del directorio que se desea examinar está especificado en los parámetros unidad y camino. Para examinar un archivo o un grupo de archivos, se puede especificar el parámetro archivo. Se puede hacer uso de comodines para hacer referencia a un grupo de archivos.

El modificador /P muestra información en la pantalla hasta que se llene, si queda más información por mostrar ésta se despliega cuando se presione cualquier tecla.

El modificador /W presenta un listado condensado de los archivos, mostrando sólo su nombre y extensión.

El modificador /A permite filtrar la lista de archivos, mostrando solamente aquellos que tengan ciertos atributos.

El modificador /O permite controlar el orden en que son listados los nombres de los archivos y directorios. Si se omite el parámetro orden, los archivos son listados en orden alfabético por nombre, luego de listar los directorios, también en orden alfabético.

El modificador /S presenta los archivos de todos los subdirectorios asociados al directorio especificado.

El modificador /B hace que el comando despliegue solamente los nombres de los archivos y subdirectorios, sin incluir otro tipo de información.

El modificador /L presenta tanto los nombres de los archivos como de los directorios usando letras minúsculas.

Ejemplo: para mostrar la lista de archivos y subdirectorios del directorio actual, pantalla por pantalla, ejecutar el siguiente comando:

DIR /P

Si todos los archivos no pueden ser presentados en una pantalla, en la última línea de la misma aparece el siguiente mensaje:

Presione cualquier tecla para continuar ...

EDIT

USO: permite la creación y modificación de archivos de texto. Dispone de una interfaz amigable, que facilita el proceso de edición de archivos de texto simple.

VERSION: 5.0 y siguientes.

SINTAXIS:

EDIT [[unidad:] [camino] archivo] [/B] [/G] [/H] [/NOHI]

COMENTARIOS: los parámetros unidad, camino y archivo identifican al archivo de texto. Si el archivo ya existe, el editor presenta su contenido en la pantalla. Si no existe el editor crea uno nuevo.

El modificador /B presenta el editor en blanco y negro.

El modificador /G mejora el rendimiento del editor para computadoras que poseen monitores CGA.

El modificador /H hace que el editor use el máximo número de líneas por pantalla que puede soportar el monitor del computador.

El modificador /NOHI permite usar el editor en un monitor que no soporte presentación de colores de alta intensidad.

Ejemplo: para editar el archivo C:\CARTA1.TXT, escribir:

EDIT C:\CARTA1.TXT

MORE

USO: permite desplegar el contenido de un archivo o el resultado de un comando, una pantalla a la vez.

VERSION: 2.0 y siguientes.

SINTAXIS:

MORE < [unidad:] [camino] archivo

COMENTARIOS: los parámetros unidad, camino y archivo identifican al archivo cuyo contenido será desplegado pantalla por pantalla.

Para avanzar de una pantalla a otra, presionar cualquier tecla. El siguiente mensaje:

−−Más−−

aparece al final de cada pantalla, indicando que queda más información por desplegar.

Ejemplo: para desplegar el contenido del archivo VENTAS98.DAT ejecutar el siguiente comando:

MORE < VENTAS98.DAT

REN o RENAME

USO: cambia el nombre de un archivo o grupo de archivos.

VERSION: 1.0 y siguientes.

SINTAXIS:

RENAME [unidad:] [camino] archivo1 archivo2

REN [unidad:] [camino] archivo1 archivo2

COMENTARIOS: los parámetros unidad, camino y archivo1 identifican al archivo o grupo de archivos cuyo nombre será cambiado. Para especificar un grupo de archivos puede hacerse uso de los comodines.

El parámetro archivo2, especifica el nuevo nombre o los nuevos nombres de los archivos indicados en el parámetro archivo1. Si el archivo especificado en el parámetro archivo2 existe, el comando despliega un mensaje de error y no cambia el nombre.

Ejemplo: para cambiar el nombre del archivo DATOS_A.DAT por DATOS_B.DAT, ejecutar el comando:

REN DATOS_A.DAT DATOS_B.DAT

Si desea cambiar la extensión de todos los archivos con extensión DAT del directorio actual por la extensión TXT, ejecutar:

REN *.DAT *.TXT

REPLACE

USO: permite el reemplazo selectivo de uno o más archivos del directorio destino por archivos de un directorio de origen con igual nombre.

VERSION: 3.2 y siguientes.

SINTAXIS:

REPLACE [unidad1:] [camino1] archivo [unidad2:] [camino2] [/A] [/P] [/R] [/W]

REPLACE [unidad1:] [camino1] archivo [unidad2:] [camino2] [/P] [/R] [/S] [/W] [/U]

COMENTARIOS: los parámetros unidad1, camino1 y archivo identifican al archivo o grupos de archivos que servirán de reemplazo. Para especificar un grupo de archivos puede hacerse uso de los comodines.

Los parámetros unidad2 y camino2 identifican la posición de los archivos que serán reemplazados por los especificados en los parámetros del punto anterior.

El modificador /A hace que sólo se agreguen archivos nuevos en el directorio destino. Si un archivo del origen ya existe en el directorio destino, éste no es reemplazado.

El modificador /P permite confirmar cada operación de reemplazo o adición de archivos.

El modificador /R permite hacer operaciones de reemplazo sobre archivos que sean de sólo lectura.

El modificador /S extiende la búsqueda de archivos en el directorio destino, a todos sus subdirectorios.

El modificador /W introduce una pausa, para que se pueda insertar un diskette en la unidad correspondiente e iniciar la búsqueda de los archivos de origen.

El modificador /U hace el reemplazo solamente de aquellos archivos del directorio destino cuya fecha sea anterior al de los archivos de origen.

Ejemplo: si se quiere reemplazar los archivos que están en el disco A por los archivos ubicados en el directorio C:\DATOS, ejecutar el siguiente comando:

REPLACE C:\DATOS*.* A:

TYPE

USO: muestra el contenido de un archivo de texto.

VERSION: 1.0 y siguientes.

SINTAXIS:

TYPE [unidad:] [camino] archivo

COMENTARIOS: los parámetros unidad, camino y archivo identifican al archivo de texto cuyo contenido será mostrado. No pueden usarse comodines para especificar un grupo de archivos.

Ejemplo: para mostrar el contenido del archivo DATOS.TXT ubicado en el directorio actual, ejecutar el siguiente comando:

TYPE DATOS.TXT

Para el manejo de impresora

GRAPHICS

USO: habilita al computador para que imprima el contenido de una pantalla, cuando se utiliza un adaptador o tarjeta de video con capacidades gráficas.

VERSION: 3.2 y siguientes.

SINTAXIS:

GRAPHICS [tipo] [[unidad:] [camino] archivo] [/R] [/B] [/LCD] [/PRINTBOX: STD % PRINTBOX: LCD]

COMENTARIOS: el parámetro tipo indica el tipo de impresora que se utiliza para imprimir el contenido gráfico de la pantalla.

Los parámetros unidad, camino y archivo identifican al archivo que almacena la información referente a los tipos de impresora soportadas por el comando, si no se especifica esta información, se utilizará el archivo GRAPHICS.PRO.

El modificador /R permite imprimir la imagen tal cuál aparece en pantalla, es decir, caracteres en blanco sobre fondo negro. Si no se especifica éste modificador se imprimen los caracteres en negro sobre fondo blanco.

El modificador /B se utiliza para imprimir el fondo en color, el modificador sólo puede ser especificado para impresoras del tipo COLOR4 y COLOR8.

El modificador /LCD permite imprimir imágenes que se despliegan en pantallas de cristal liquido, usado por la mayoría de las computadoras portátiles.

El modificador /PRINTBOX: permite seleccionar el recuadro de impresión. El valor STD establece un tamaño proporcional a las dimensiones de la pantalla, mientras que el valor LCD establece un tamaño proporcional a una pantalla de cristal liquido. El modificador LCD tiene el mismo efecto.

El comando GRAPHICS prepara al computador para imprimir una pantalla de gráficos. Para imprimir una imagen que se presenta en pantalla, presionar las teclas SHIFT+PTRSCRN.

PRINT

USO: imprime el contenido de uno o más archivos de texto.

VERSION: 2.0 y siguientes.

SINTAXIS:

PRINT [/D: dispositivo] [/B: tamaño] [/U: max1] [/M: max2] [/S:tiempo] [/Q: cola] [/T] [[unidad:] [camino] archivo [...] [/C] [/P]

COMENTARIOS: los parámetros unidad, camino y archivo identifican a los archivos de texto que serán impresos.

El modificador /D: dispositivo identifica al puerto de comunicación en donde está conectado el dispositivo de impresión. Los valores que puede tomar son: LPT1, LPT2 y LPT3 para los puertos paralelos; y COM1, COM2, COM3 y COM4 para los puertos seriales.

El modificador /B: tamaño especifica el tamaño, en bytes, de la zona de memoria que es usada para los datos antes de ser impresos (*buffer*). Los valores que puede tomar oscilan entre 512 y 16.384. El valor por defecto es de 512.

El modificador /U: max1 permite especificar el tiempo que debe esperar el comando por la disponibilidad de la impresora. Si al pasar este tiempo la impresora no está disponible se cancela el proceso de impresión. El parámetro max1 viene expresado en pulsaciones de reloj electrónico (*ticks*), una pulsación corresponde aproximadamente a 1/18 de segundo. Los valores del parámetro pueden oscilar entre 1 y 255, si no se especifica se considera un valor de 1.

El modificador /M: max2 especifica el tiempo que podrá tardar el comando para imprimir un carácter, si se sobrepasa ese tiempo se despliega un mensaje de error. Éste parámetro viene dado en pulsaciones de reloj electrónico.

El modificador /S: tiempo especifica el tiempo que es asignado al proceso de impresión. El comando PRINT se ejecuta en segundo plano, es decir, que mientras se imprime un archivo se pueden ejecutar comandos. Para ello el sistema operativo debe planificar la proporción de tiempo que será asignado a cada tarea. Mientras mayor sea el valor de tiempo, la impresión se efectuará más rápidamente, pero el tiempo de ejecución de otros comandos será mayor. Éste parámetro viene dado en pulsaciones de reloj electrónico.

El modificador /Q: cola especifica un número máximo de archivos que pueden estar en la cola de impresión, esta cola contiene los nombres de los archivos que esperan por ser impresos. Los valores que puede tomar el parámetro cola oscilan entre 4 y 32. Si no se especifica éste parámetro se considera un valor de 10.

El modificador /T elimina todos los archivos de la cola de impresión.

El modificador /C permite eliminar selectivamente archivos de la cola de impresión. Si es colocado después de un nombre de archivo se elimina de la cola el archivo que precede al modificador y los siguientes hasta conseguir un modificador /P.

El modificador /P permite agregar archivos a la cola de impresión. Si es colocado después del nombre de un archivo, se agrega a la cola el archivo que precede al modificador y los siguientes hasta encontrar un modificador /C.

Para expresar el estado actual de la cola de impresión ejecutar el comando sin especificar ningún parámetro.

Ejemplos: para establecer que el dispositivo de impresión está conectado al puerto LPT1, ejecutar el siguiente comando:

PRINT /D: LPT1

Para establecer un límite de 28 archivos en la cola de impresión, escribir el siguiente comando:

PRINT /: 28

Para imprimir el archivo CARTA1.TXT, ejecutar:

PRINT CARTA1.TXT

Para eliminar de la cola de impresión el archivo CARTA1.TXT y agregar los archivos CARTA2.TXT y CARTA3.TXT, ejecutar el siguiente comando:

PRINT CARTA1.TXT/C CARTA2.TXT /P CARTA3.TXT

Para respaldo de información

BACKUP

USO: permite hacer copias de seguridad en discos, de uno o varios archivos. Las copias de seguridad pueden hacerse usando diskettes o usando un disco duro. La copia puede realizarse entre discos que pueden tener formatos diferentes.

VERSION: 2.0 y siguientes.

SINTAXIS:

BACKUP fuente unidad−destino: [/S] [/M] [/A] [/F] [:tamaño] [/D:fecha] [/T: hora] [/L [: [unidad:] [camino] notas]]

COMENTARIOS: el parámetro fuente indica la ubicación de los archivos a ser respaldados. Puede estar constituido por el nombre de un archivo, el nombre de un directorio, la identificación de una unidad de disco o alguna combinación de éstos. Pueden usarse comodines para especificar un grupo de archivos.

El parámetro unidad−destino indica la unidad de disco que va a ser usada para almacenar las copias de seguridad.

El modificador /S le indica la comando que efectúe respaldo de todos los subdirectorios incluidos en el directorio especificado en el parámetro fuente.

El modificador /M permite copiar solamente aquellos archivos que han sido modificados desde que se hizo el último respaldo. Además, elimina el atributo de archivo modificado de los archivos originales.

El modificador /A hace que los archivos sean agregados al disco en donde se están haciendo las copias, sin borrar ningún archivo que exista previamente.

El modificador /F: tamaño permite dar formato al disco que se encuentra en la unidad de destino, cuando sea necesario. El parámetro tamaño especifica la capacidad del disco al cual se le va a dar formato.

El modificador /D: fecha permite copiar sólo aquellos archivos que fueron alterados en la fecha especificada o luego de ésta.

El modificador /T: hora permite copiar sólo aquellos archivos que fueron modificados en la hora especificada o luego de ésta. Éste modificador debe ser usado junto con el modificador /D.

El modificador [/L [: [unidad:] [camino] notas]] crea un archivo en donde se registran las diferentes operaciones de respaldo como la fecha y la hora, identificación de los diskettes y los nombres de los archivos respaldados. Si no se proporciona un camino de acceso, el archivo se almacena en el directorio raíz de la unidad de disco fuente. En caso de especificar una unidad, ésta deberá ser un disco duro. El parámetro notas indica el nombre del archivo; si no es especificado, se usará por defecto el nombre BACKUP.LOG.

El comando BACKUP genera dos archivos en cada diskette, llamados BACKUP.nnn y CONTROL.nnn, donde nnn corresponde al número de diskette usado para realizar el respaldo. Así el primer diskette usado para crear el respaldo tendrá los archivos BACKUP.001 y CONTROL.001. El segundo diskette tendrá los archivos BACKUP.002 y CONTROL.002, y así sucesivamente hasta completar la copia de todos los archivos.

Los archivos respaldados sólo pueden ser recuperados usando el comando RESTORE.

Ejemplo: supóngase que se desea efectuar un respaldo de todos los archivos del directorio DATOS que está en la unidad C, en diskette en la unidad A. Para hacer el esto, ejecutar el siguiente comando:

BACKUP C:\DATOS A:

Inmediatamente aparece en pantalla el siguiente mensaje:

Inserte el diskette de seguridad 01 en unidad A:

¡ADVERTENCIA! Archivos en unidad destino A:\ directorio raíz serán borrados

Presione cualquier tecla para continuar ...

Al presionar cualquier tecla, aparecerá el mensaje:

*** creando copias de seguridad en la unidad A: ***

Diskette Número: 01

CHKDSK

USO: presenta un reporte sobre el estado del disco. El reporte incluye información sobre fragmentación de archivos y errores en el sistema de archivos.

VERSION: 1.0 y siguientes.

SINTAXIS:

CHKDSK [unidad:] [[camino] archivo] [/F] [/V]

COMENTARIOS: el parámetro unidad identifica la unidad de disco que se desea revisar con el comando.

Los parámetros camino y archivo identifican al archivo o grupo de archivos que van a ser revisados para detectar fragmentación. Se pueden usar comodines para especificar un grupo de archivos. Si se omite éste parámetro, el comando revisará todo el disco.

El modificador /S corrige los errores en el disco. Éstos errores se refieren a porciones de archivos que fueron

removidos del directorio, pero ocupan espacio en disco. Cuando se localiza algún error en el disco, el comando despliega un mensaje similar al siguiente:

13 unidades de asignación perdidas se encontraron en 4 cadenas.

¿Desea convertir las cadenas perdidas en archivos (s/n)?

En caso de que responda s (sí), el comando recupera cada cadena perdida en un archivo con el nombre de la forma FILEnnn.CHK y los coloca en el directorio raíz de la unidad. Éstos archivos pueden ser revisados para ver si tienen alguna información importante. Si se responde n (no), el comando elimina directamente las cadenas perdidas.

El modificador /V lista los nombres de cada archivo de cada directorio de la unidad que está siendo revisada.

Ejemplo: para revisar el disco C, ejecutar el comando:

CHKDSK C:

Se despliega un reporte similar al siguiente:

Volumen DISCO1 creó 07−28−1993 11:32a

Número de serie del volumen es 1D17−8C4B

106614784	bytes en 8 archivo (s) oculto (s)
5208064	bytes en 8 archivo (s) oculto (s)
249856	bytes en 2529 archivo (s) de usuario
92438528	bytes en 2529 archivo (s) de usuario
8718336	bytes disponibles en disco
2048	bytes en cada unidad de asignación
52058	total de unidades de asignación en disco
4257	unidades de asignación disponibles en disco
655360	bytes en memoria total
577200	bytes libres

COMP

USO: determina si el contenido de dos archivos o grupos de archivos, es el mismo. En caso de existir alguna diferencia, se despliega en pantalla el byte que difiere y su ubicación en ambos archivos.

VERSION: 3.3 y siguientes.

SINTAXIS:

COMP [unidad1:] [camino1:] [archivo1] [unidad2:] [camino2] [archivo2] [/A] [/D] [/L] [/N = número] [/C]

COMENTARIOS: los parámetros unidad1:, camino1 y archivo1 identifican al primer archivo o grupo de archivos que van a ser comparados.

Los parámetros unidad2:, camino2 y archivo2 identifican al segunda archivo o grupo de archivos que van a ser comparados.

Se puede hacer uso de comodines, en los parámetros archivo1 y archivo2, para especificar grupos de archivos.

Por defecto el comando COMP presenta los bytes que difieren en código hexadecimal. Para ver las diferencias en caracteres ASCII, se debe especificar el modificador /A. O si por el contrario, se desea presentar las diferencias en código decimal, especificar el modificador /D.

El modificador /L, al ser especificado, incluye el número de línea donde se encuentran las diferencias.

El comando COMP no permite compara archivos de diferentes tamaños a menos que se use el modificador /N. El número de líneas a compara debe especificarse en el parámetro número.

El modificador /C', cuando es especificado, permite ignorar las diferencias entre mayúsculas y minúsculas.

Ejemplo: si se desea compara dos archivos llamados DATOS1.DAT y DATOS2.DAT, debe escribirse el siguiente comando:

COMP DATOS1.DAT DATOS2.DAT

Mientras hace la comparación se despliega el siguiente mensaje:

Comparando DATOS1.DAT DATOS2.DAT ...

Si no existen diferencias el comando responde:

Comparación de archivos SI

En caso de existir diferencias se despliega un mensaje como el siguiente por cada diferencia detectada:

Error de comparación en OFFSET D

Archivo1 = 47

Archivo2 = 64

Error de comparación en OFFSET 22

Archivo1 = 76

Archivo2 = 62

Error de comparación en OFFSET 36

Archivo1 = 65

Archivo2 = 61

Si los archivos no tienen el mismo tamaño, el comando despliega el siguiente mensaje:

Archivos son de tamaño diferentes

DISKCOMP

USO: compara el contenido de dos diskettes pista por pista.

VERSION: 3.2 y siguientes.

SINTAXIS:

DISKCOMP [[unidad1:] [unidad2:]] [/1] [/8]

COMENTARIOS: los parámetros unidad1 y unidad2 identifican a las unidades que contienen a los diskettes cuyos contenidos van a ser comparados. No pueden especificarse indicaciones de discos duros. Si se necesita usar la misma unidad para los dos diskettes, se puede usar, especificando la misma en ambos parámetros; el comando solicita durante el proceso de comparación, que se intercambien los diskettes. Si se omite la especificación de alguna de las unidades, se toma la identificación de la unidad actual.

El modificador /1 compara únicamente la primera cara de cada diskette (cara 0).

El modificador /8 compara únicamente los primeros 8 sectores de cada pista, en ambos diskettes.

El comando no puede comparar diskettes de diferentes tipos, si los diskettes son de tipos diferentes, el comando despliega un mensaje de error.

Para comparar dos diskettes usando la unidad B, escribir:

DISKCOMP B: B:

se despliega un mensaje como el siguiente:

Inserte el PRIMER diskette en unidad B

Presione cualquier tecla para continuar ...

Luego de realizar dicha acción, se presenta información acerca del formato del diskette:

Comparando 80 pistas

18 sectores por pistas, 2 cara (s)

Un instante más tarde se pide al usuario que inserte el segundo diskette en la unidad B:

Inserte el segundo diskette en unidad B:

Presione cualquier tecla para continuar ...

Es posible que se deban intercambiar los diskettes más de una vez.

DISKCOPY

USO: copia el contenido de un diskette en otro. El comando da formato al diskette de destino, si es necesario.

VERSION: 2.0 y siguientes.

SINTAXIS:

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

COMENTARIOS: el parámetro unidad1, identifica la unidad que contiene el diskette que se desea copiar, denominado diskette de origen.

El parámetro unidad2, identifica la unidad que contiene el diskette en donde se va a copiar la información, denominado diskette de destino. Se puede usar una misma unidad para copiar dos diskettes, especificando la misma identificación en los dos parámetros. Durante el proceso de copia, el comando solicitará que se intercambien los discos.

El modificador /1 hace que se copie solamente la primera cara del diskette (cara 0).

El modificador /V permite verificar la copia, asegurando que el contenido de los diskettes sea el mismo.

No se puede utilizar el comando para copiar discos duros, sirve solamente para copiar diskettes.

Ejemplo: para copiar un diskette que está en al unidad A, a un diskette que está en la unidad B, escribir el siguiente comando:

DISKCOPY A: B:

Inmediatamente aparece el siguiente mensaje:

Inserte diskette de origen en unidad A:

Inserte diskette de destino en unidad B:

Presione cualquier tecla para continuar ...

Luego aparece un mensaje especificando el formato del disco que se está copiando:

Copiando 80 pistas

18 sectores por pista, 2 cara (s)

RESTORE

USO: permite la recuperación de archivos que forman parte de copias de seguridad efectuadas usando el comando BACKUP.

VERSION: 2.0 y siguientes.

SINTAXIS:

RESTORE unidad1: unidad2: [camino [archivo]] [/S] [/P] [/B: fecha] [/A: fecha] [/E: hora] [/L: hora] [/M] [/N] [/D]

COMENTARIOS: el parámetro unidad1 identifica la unidad donde residen las copias de seguridad de los archivos.

El parámetro unidad2 identifica la unidad de destino.

El parámetro camino indica el subdirectorio en donde serán almacenados los archivos que se restauren. El nombre de subdirectorio debe corresponder al directorio original, desde donde se hicieron las copias de seguridad.

El parámetro archivo identifica el archivo o grupo de archivos que serán restaurados. Puede hacerse uso de los comodines para especificar un grupo de archivos.

El modificador /S permite considerar todos los subdirectorios en el proceso de restauración.

El modificador /P permite la confirmación al momento de restaurar archivos de sólo lectura o archivos que hayan sido modificados desde la última vez que se hizo una copia de seguridad.

El modificador /B: fecha restaura solamente los archivos modificados en la fecha indicada o antes de la misma.

El modificador /A: fecha restaura solamente los archivos modificados en la fecha indicada o después de la misma.

El modificador /E: hora restaura solamente los archivos modificados en la hora indicada o antes de la misma.

El modificador /L: hora restaura solamente los archivos modificados en la hora indicada o después de la misma.

El modificador /M restaura solamente los archivos que fueron modificados después de efectuar la última copia de seguridad.

El modificador /N restaura los archivos que no se encuentren previamente en el directorio de destino.

El modificador /D permite mostrar la lista de archivos que serán restaurados al disco destino, si se ejecuta el comando con los parámetros y modificadores actuales. Si se especifica éste modificador, ningún archivo es restaurado.

Ejemplo: retomemos el ejemplo del comando explicado anteriormente, ahora lo que se quiere hacer es restaurar los archivos:

RESTORE A: C:\DATOS*.*

El sistema operativo solicita que introduzca el primer diskette de la copia de seguridad en la unidad A, mediante el siguiente mensaje:

Inserte diskette copia de seguridad 01 en unidad A:

Presione cualquier tecla para continuar ...

Inmediatamente aparece la fecha en que realizó la copia de seguridad y los nombres de los archivos que recupera del diskette:

** se hizo copia de seguridad de los archivos 11−10−1994**

*** restaurando archivos de unidad A: ***

diskette: 01

\DATA\DATOS1.DAT

\DATA\DATOS2.DAT

\DATA\DATOS3.DAT

Es importante que el directorio de destino sea el mismo que el directorio desde donde se realizó las copias de respaldo. En caso contrario, se despliega el siguiente mensaje:

¡ADVERTENCIA! No se encontraron archivos para restaurar

RECOVER

USO: permite recuperar información de un disco que presenta defectos en algunos de sus sectores.

VERSION: 2.0 y siguientes.

SINTAXIS:

Para recuperar un archivo:

RECOVER [unidad:] [camino] archivo

Para recuperar un disco completo:

RECOVER unidad:

COMENTARIOS: los parámetros unidad, camino y archivo identifican al archivo que será recuperado. Solo las partes del archivo que no están en ningún sector defectuoso serán recuperadas. No se pueden usar comodines para especificar un grupo de archivos.

Si solamente se especifica el nombre de la unidad, todos los archivos son recuperados. Los archivos recuperados estarán ubicados en el directorio raíz, bajo el nombre de FILEnnnn.REC, donde nnnn representa un número de 4 dígitos, que se ira incrementando a medida que se vayan recuperando archivos.

Ejemplo: para recuperar el archivo DATOS.TXT ubicado en directorio principal del diskette en la unidad A, ejecutar el siguiente comando:

RECOVER A:\DATOS.TXT

SYS

USO: transfiere los archivos del sistema y el procesador de comandos a un diskette.

VERSION: 1.0 y siguientes.

SINTAXIS:

SYS [unidad1:] [camino] unidad2:

COMENTARIOS: los parámetros unidad1 y camino especifican la ubicación de los archivos del sistema IO.SYS y MSDOS.SYS, y el procesador de comandos CAOMMAND.COM. Los archivos IO.SYS y MSDOS.SYS son archivos ocultos.

El parámetro unidad2 identifica la unidad en donde está el diskette que recibirá los archivos. Éstos son copiados en el directorio raíz.

UNDELETE

USO: ofrece la posibilidad de recuperar archivos eliminados previamente, por medio del comando DEL.

VERSION: 5.0 y siguientes.

SINTAXIS:

UNDELETE [[unidad] [camino] archivo] [/LIST | /ALL] [/DOS | /DT]

COMENTARIOS: los parámetros unidad, camino y archivo identifican al archivo que desea recuperar. Se puede hacer uso de comodines para especificar un grupo de archivos.

El modificador /LIST despliega una lista de los archivos borrados que pueden ser recuperados según los valores de los parámetros sin recuperar ninguno.

Cuando un archivo es eliminado, el sistema operativo sustituye la primera letra del nombre del archivo por un caracter especial. Si el modificador /ALL es especificado, el comando usará el registro de archivos borrados. Si el modificador no está especificado, el comando solicita al usuario que escriba la primera letra de cada archivo a recuperar.

El modificador /DOS recupera solamente aquellos archivos que el sistema operativo ha marcado como borrados.

El modificador /DT recupera solamente aquellos archivos que aparecen en el registro de archivos eliminados.

Es posible que el comando no pueda recuperar un determinado archivo, especialmente si después de su eliminación, se crearon, modificaron o eliminaron otros archivos.

Ejemplo: se quiere recuperar el archivo DATOS98.TXT del directorio C:\DATOS, para recuperar el archivo, ejecutar el siguiente comando:

UNDELETE C:\DATOS\DATOS98.TXT

Inmediatamente aparece en pantalla el siguiente mensaje:

Directorio: C:\DATOS

Espec. De archivo: DATOS98.TXT

No se encontró arch. De regis/eliminación

Directorio de MS−DOS contiene 1 archivos eliminados.

De los cuales, 1 se podrán recuperar.

Usando el directorio MS−DOS.

?ATOS98 TXT 3 12−14−98 4:42p ...A

Restablecer (s/n)?

Si se presiona la letra S a continuación aparece el siguiente mensaje:

Por favor escriba la primera letra de ?ATOS98.TXT

Luego de suministrar la letra faltante, en caso de ser D, aparece el siguiente mensaje:

Archivo fue restablecido con éxito.

UNIDAD

CINCO

LENGUAJE DE PROGRAMACION: PASCAL

CARACTERISTICAS DEL LENGUAJE

El PASCAL se ha convertido en un lenguaje dirigido a al enseñanza de la programación, aunque últimamente está siendo utilizado para el desarrollo de programas complejos. El Pascal introduce los conceptos de la buena programación. Debido a que sus instrucciones son palabras en inglés, éstas se pueden memorizar con facilidad y pueden formarse instrucciones más complejas, componiendo instrucciones simples. Es un lenguaje altamente estructurado que ayuda al programador a organizar bien las ideas antes de empezar a escribir un programa.

Las siguientes características son propias de Turbo Pascal 5.5 a 7.0:

- Posee palabras reservadas (*absolute, interface, string, unit, uses*, etc.).
- Un identificador puede contener caracteres subrayados _ , (paga_mes, nombre_apellido, interés_anual, etc.).
- Constantes: enteras (se pueden escribir en notación hexadecimal, con prefijo $). De cadenas (son compatibles con tipos string y pueden contener caracteres de control y otros caracteres no imprimibles).
- Las declaraciones de constantes, tipos, variables, procedimientos y funciones pueden ocurrir cualquier número de veces, en cualquier orden en un bloque.
- Las variables se pueden declarar en direcciones de memoria absoluta utilizando una cláusula *absolute*.
- Las variables tipo *string* se pueden indexar como arrays, para acceder individualmente a caracteres de una cadena.
- Incorpora operadores lógicos como: XOR, SHL y SHR.
- Los operadores NOT, AND, OR y XOR se pueden utilizar como operandos tipo entero para ejecutar

operaciones lógicas *bitwise*.

- El operador +, se puede utilizar para comparar cadenas, al igual que los operadores relacionales.
- Incorpora el operador @, que sirve para obtener la dirección de una variable, un procedimiento o una función.
- Incorpora unidades para programación modular y compilación separada.
- Incorpora procedimientos y funciones específicas de tratamientos de archivos (APPEND, ERASE, FILESIZE, RENAME, SEEK, etc.).
- Incorpora las siguientes funciones y procedimientos:

- Addr
- GetMen
- Move
- Sptr
- Concat
- Halt
- Ofs
- Seg
- Copy
- Hi
- ParamCount
- SizeOf
- CSeg
- Inc
- ParamStr
- SSeg
- DSeg
- Insert
- Pi
- Str
- Dec

- Int
- Pos
- Swap
- Delete
- Lenght
- Ptr
- UpCase
- Exit
- Lo
- Random
- Val
- FillChar
- Mark
- Randomize
- Frac
- MaxAvail
- Release
- FreeMen
- MemAvail
- RunError

FORMATO de las SENTENCIAS

Asignación:
<*identificador*>: = <*expresión*>;

Rango: = alto − bajo;
Cuenta: = cuenta + 1;

Compuesta:
Begin <*sentencias*>

end;

Begin

z: = 5;

GetReal (`Valor', ValorReal);

WriteLn (ValorReal);

End

Selectiva (*if*):

if <*condición*> **then**

<*sentencia*>

[**else**

<*sentencia*>];

if (Cad1 <> Cad2) **then**

Begin

Cad1: = Cad1 + Cad2;

WriteLn (`Nueva Cadena', Cad1);

End;

Selectiva (*if−anidada*):

if <*condición*> **then**

Sentencia

Else if <*condición*> **then**

Sentencias

Else if <*condición*> **then**

Sentencia

Else

Sentencia

if (x>y) **then**

if (x>z) **then**

Write (x);

Else

Write (z);

Else

if (y>z) **then**

Write (y);

Else

Write (z);

WriteLn (`es el mayor');

Selectiva múltiple (*case*):

Case <*selector*> **of**

<*lista de constantes*>

<*sentencias 1*>

<*lista de constantes*>

<*sentencias 2*>

...

[**else**

<*sentencia por defecto*>]

End;

Case (N) **of**

1, 2 : WriteLn (`Primero');

3, 4, 5; : WriteLn (`Bronce')

7 : WriteLn (`Fin');

else

WriteLn (`Final');

End;

Sentencia repetitiva *for*:

- **For** <*índice*>: = <*valor inicial* > **to** <*valor final* > **do** <*sentencia*>

```
For k: = 1 to 10 do
WriteLn (`cuerpo del bucle');
```

```
For j: = 1 to 10 do
Begin
    J: = J + Z;
    WriteLn (`pasos', pasos: 2);
    Paso: = paso + 1;
End;
WriteLn (`Fuera del bucle');
```

- **For** <*índice*>: = <*valor final*> **downto** <*valor inicial*> **do** <*sentencia*>

```
For i: = 10 downto 1 do
Begin
Resultado: = i * Resultado;
WriteLn (`Bucle', i, resultado);
End;
```

Sentencia repetitiva *while*:

While <*condición*> **do**

<*sentencia*>

<*sentencia*> = *sentencia simple*

Sentencia compuesta

```
Paso: = 1;
While (paso<=10) do
Begin
    WriteLn (`Pasos:', paso: 2);
    Paso: = Paso + 1;
End;
WriteLn (`Fuera del Bucle');
```

Sentencia repetitiva *repeat*:

```
Repeat
<sentencia>
Until <condición>;
```

```
Paso: = 1;
Repeat
    WriteLn (`pasos', paso: 2);
    Paso: = Paso + 1;
Until (Paso >10);
WriteLn (`Fuera del bucle');
```

VARIABLES y CONSTANTES

Constantes predefinidas:

True	Verdadero
False	Falso
Maxint	Entero mayor disponible en máquina.

Declaración de variables:

Var
X, Y, Z: real;
I, J, K: integer;
Digito: 0..9;

Variables absolutas:
Var
Cad: string [40];
LongCad: Byte absolute Cad;

Variables predefinidas:	
Variable	*Tipo*
Input	Archivo Text
Ouput	Archivo Text
ExitCode	Integer
FileMode	Byte
InOutRes	Integer
RandSeed	LongInt
StackLimit	Word

Constantes con tipos:
Tipo simple
Const
Maximo: integer = 425;
Factor: real = −12.5
Const
Min: integer = 0;
Max: ineger = 100;
Type
Lista= **array** [Min..Max] **of** integer;
La declaración LISTA no es válida ya que Min y Max son constantes de tipo.
Tipo cadena
Const
Cabecera : string [10] = `Ciudades';
Nueva Linea : string [2] = #13#10;
Respuesta : string [5] = `Si';
Tipo estructurado
Const
Digitos : **array** [0..9] **of** char =(`0', `1', `2', `3', `4', `5', `6', `7', `8', `9');
Const
Digitos : array [0..9] of char =`0123456789';
Type
Matriz : array [0..1, 0..1, 0..1] of integer;
Const

```
Lista : Matriz =(((0,1), (2,3)), ((4,6) ,(7,9)))
```

Tipo registro

```
Type
Punto = record
X, Y: real;
End;
Vector = array [0..1] of Punto;
```

Tipo objeto

```
Type
Punto = object
        X, Y: integer;
      End;
Const
Origen : Punto = (X: 0, Y: 0);
```

Tipo conjunto

```
Type
Digitos = set of 0..9;
Letras = set of `A'...`Z';
Const
Pares : Digitos = [0, 2, 4, 6, 8];
Vocales : Letras = [`A', `E', `I', `O', `U'];
```

Tipo puntero

```
Type
Direccion = (Izda, Dcha, Arriba, Abajo);
PNodo = ^NodoT;
NodoT = Record
          ...
        End;
Const
S1 : string [4] = `Abajo';
N1 : NodoT = (siguiente: nil; symbolo: @S1; valor: Abajo);
```

Tipo procedimiento

```
Type
ProcError = procedure (CodigoError: integer);
Precedure ErrorPorOmision (CodigoError: integer) far;
Begin
WriteLn (`Error', CodigoError, `, `);
End;
Const
ManipuladorErrores : ProcError = ErrorPorOmision;
```

OPERADORES

Aritméticos:

Operador	Sintaxis	Significado
+	+Expresión	Positivo (unitario).
+	Expresión1+Expresión2	Suma (binaria).
−	−Expresión	Negativo (unitario).
−	Expresión1−Expresión2	Resta (binario).
*	Expresión1*Expresión2	Multiplicación.
/	Expresión1/Expresión2	División real.
DIV	Expresión1 DIV xpresión2	División entera.
MOD	Expresión1 MOD Expresión2	Resto (módulo).

Lógicos:

Operador	Sintaxis	Significado
NOT	NOT Expresión	Complemento.
AND	Expresión1 AND Expresión2	AND.
OR	Expresión1 OR Expresión2	OR inclusiva.
XOR	Expresión1 XOR Expresión2	OR exclusiva.

Relacionales:

Operador	Sintaxis	Significado
=	Expresión1=Expresión2	Expresiones son iguales.
<>	Expresión1<>Expresión2	Expresiones no son iguales.
<	Expresión1<Expresión2	Expresión1 es menor que Expresión2.
<=	Expresión1<=Expresión2	Expresión1 es menor o igual que Expresión2.
>	Expresión1>Expresión2	Expresión1 es mayor que Expresión2.
>=	Expresión1>=Expresión2	Expresión1 es mayor o igual que Expresión2.

De conjunto:

Operador	Sintaxis	Significado
=	Conjunto1=Conjunto2	Conjunto1 y Conjunto2 son idénticos. Cada elemento en el Conjunto1 está contenido en el Conjunto2, y cada elemento en Conjunto1 está contenido en Conjunto1.
<>	Conjunto1<>Conjunto2	Uno de los conjuntos contiene al menos un elemento que no está en el otro conjunto.
<=	Conjunto1<=Conjunto2	Cada elemento del Conjunto1 está también en Conjunto2.
>=	Conjunto1>=Conjunto2	Cada elemento de Conjunto2 está también en Conjunto1.
IN	Elemento IN Conjunto2	El elemento Elemento se encuentra en Conjunto2.

−	Conjunto1−Conjunto2	Diferencia.
+	Conjunto1+Conjunto2	Unión.
*	Conjunto1*Conjunto2	Intersección.

Dirección:

@ Dirección de variable, procedimiento o función.

Concatenación:

+ concatena dos cadenas.

Prioridad de operadores:

Prioridad	Operadores
1 (alta)	@, NOT, unitario, +, −
2	*, /, DIV, MOD, AND, SHL, SHR
3	Binario, +, −, OR, XOR
4 (baja)	=, <>, <, >, <=, >=, IN

FUNCIONES

Funciones aritméticas:

Nombre	Sintaxis	Descripción
Abs	Abs (x);	Devuelve el valor (positivo) absoluto de su argumento.
ArcTan	ArcTan (x: real): real;	Arco Tangente expresado en radianes.
Cos	Cos (x: real): real;	Coseno del argumento en radianes.
Exp	Exp (x: real): real;	Potencia exponencial del argumento (ex).
Frac	Frac (x: real): real;	Parte decimal de un número real.
Int	Int (x: real): real;	Parte entera del argumento.
Ln	Ln (x: real): real;	Logaritmo neperiano (base e) del argumento.
Pi	Pi: real;	Valor de Pi (3.14159...).
Sin	Sin (x: real): real;	Seno del argumento.
Sqr	Sqr (x);	Cuadrado del argumento.
Sqrt	Sqrt (x: real): real;	Raíz cuadrada del argumento.

Ejemplos:

Y	: = Abs (x)*2;	Z	: = Int (345.678);
Z	: = ArcTan (1.75);	T	: = Ln (1.25);
Tan	: = Sin (x) / Cos (x);	Z	: = ArcTan (Pi);
Pot	: = Exp (3);	Z	: = Sin (Pi);
R	: = Frac (−245.123);	F	: =Sqr (3.45);

F	: = Sqrt (1.2345);

Funciones de transferencias:

Nombre	Sintaxis	Descripción
Chr	Chr (x: byte): char;	Devuelve el carácter correspondiente al código ASCII.
Ord	Ord (x): LongInt;	Número ordinal de un tipo ordinal.
Round	Round (x: real): LongInt;	Redondea un valor real a entero largo.
Trunc	Trunc (x: real): LongInt;	Trunca un valor de tipo real a entero.

Ejemplos:

Write (Chr(i));	Round (5.449)	*Devuelve 5*
Ord (`A');	Trunc (−3.14)	*Devuelve −3*
	Trunc (6.5)	*Devuelve 6*

Procedimientos o funciones ordinales:

Nombre	Sintaxis	Descripción
Dec	Dec (var x [; n: LongInt]);	Decrementa una variable.
Inc	Inc (var x [; n: LongInt]);	Incrementa una variable.
High	High (x);	Devuelve el valor más alto en el rango del argumento que debe ser un tipo array o un tipo cadena.
Low	Low (x);	Devuelve el valor más bajo (array o cadena).
Odd	Odd (x: LongInt): boolean;	
Pred	Pred (x);	Predecesor del argumento (x del tipo ordinal.).
Succ	Succ (x);	Sucesor del argumento (x del tipo ordinal).

Ejemplos:

```
Dec (z);
Inc (z);
```

```
For (i: = 0) to (High (x)) do
S: = S + x[i];
```

```
For (Dia: = Low (ADia)) to (High (Adia)) do
Begin
    ...
End;
```

Pred (z);	*Devuelve `4'*
Succ (z);	*Devuelve 1947*

Tratamiento de cadenas:

Nombre	Sintaxis	Descripción

Concat	Concat (s1, s2, ..., sn: string);	Concatena (une) cadenas.
Copy	Copy (s: string; Pos, Long: integer): string;	Copia una cadena dada.
Length	Length (s: string): integer;	Longitud de una cadena.
Pos	Pos (Patron, Fuente: string): integer;	Posición de la primera ocurrencia de una subcadena.

Ejemplos:

s: = Concat (`xyz', `LUIS');

s: = Copy (s, 2, 3);

t: = Length (Cad);

Asignación dinámica:

Nombre	Sintaxis	Descripción
MaxAvail	MaxAvail: LongInt;	Tamaño del bloque disponible mayor del montículo (heap).
MemAvail	MemAvail: LongInt;	Cantidad de memoria libre en el montículo (heap).

Funciones diversas:

Nombre	Descripción	Sintaxis
Hi	Hi (x): byte;	Byte de mayor pero del argumento.
Include	Include (**var** s: **set of** t; i: t);	Incluye un elemento en un conjunto.
Lo	Lo (x): byte;	Byte de menor pero del argumento.
Move	Move (**var** Fuente, Dest; Cuenta: word);	Copia un número especificado de bytes contiguos de un rango fuente a un rango destino.
ParamCount	ParamCount: word;	Devuelve el número de parámetros pasados en la línea de órdenes.
ParamStr	ParamStr: (Indice): string;	Parámetro de la línea de órdenes.
Random	Random [(Rango: word)];	Devuelve un número aleatorio.
Sizeof	Sizeof (x): word;	Número de bytes ocupado por el argumento.
Swap	Swap (x);	Intercambia los bytes más altos y menos altos del argumento.
TypeOf	TypeOf (x): Pointer;	Devuelve un puntero a una tabla de métodos virtuales de tipos objetos.
UpCase	UpCase (ch: char): char;	Convierte un caracter a mayúsculas.

Ejemplos:

Randomize;

for (i: = 1) **to** (10) **do**

```
Write (Random (MaxInt): 8);

WriteLn (`I =', I: 5, `Byte alto =', Hi (I):3);

Include (s, I);

WriteLn (`Byte bajo =', Lo (n): 6);

Move (car[1], car[50], 50);

for (i: = 1) to (ParamCount) do

WriteLn (i: 2, `:', ParamStr (i));

WriteLn (`Real ...', Sizeof (real));

w: = 240; w: = Swap (w); WriteLn (`w =', w);

WriteLn (`car: UpCase');

for (ch: = `a') to (`z') do

WriteLn (ch, UpCase (ch), ` ');
```

Entradas/salidas:

Nombre	Descripción	Sintaxis
Eof	Eof (**var** F): boolean;	Devuelve el estado fin ade archivo.
Eoln	Eoln [(**var** F: text)]: boolean;	Devuelve el estado fin de línea de un archivo de texto.
FilePos	FilePos (**var** F): LongInt;	Posición actual de un archivo con o sin tipos.
IOResult	IOResult: Integer;	Estado de la última operación de Entrada/Salida realzada.
SeekEof	SeekEof [(**var** F: text)]: boolean;	Estado fin de archivo de un archivo.
SeekEoln	SeekEoln[(**var** F: text)];	Estado fin de línea de un archivo.

Ejemplos:

```
While not Eof (Tf) do

Begin

If Eoln (Tf)

Then WriteLn;

If Length (s) > 0

Then Rename (F, S);
```

End;

Existe: = (IOResult = 0);

While not SeekEoln (Tf) **do**

Begin

ReadLn (Tf, Ch);

Write (ch);

End;

ESTRUCTURAS DE DATOS

Una estructura de datos es una colección de datos organizados de modo particular. Pueden ser de dos tipos: *estructuras de datos estáticas* y *estructuras de datos dinámicas*.

Las primeras son aquellas en las que se asigna una cantidad fija de memoria cuando se declara la variable.

En numerosas ocasiones se necesitan colecciones de datos que crezcan y reduzcan su tamaño en memoria a medida que el programa progresa, éstas se denominan estructuras dinámicas de datos.

Pascal define las estructuras de datos a través de declaraciones de tipos. Nos ocuparemos de uno de los más usados: *array*.

Un Array es una estructura de datos en la que se almacena una colección de datos del mismo tipo (por ej., los salarios de los empleados de una empresa); es decir, un Array es una lista de un número finito n de elementos del mismo tipo que se caracteriza por:

- Almacenar los elementos del Array en posiciones de memoria continua.
- Tener un único nombre de variable que representa a todos los elementos, y éstos a su vez se diferencian por un índice o subíndice.
- Acceso directo o aleatorio a los elementos individuales del Array.

Los Array se clasifican en :

- Unidimensionales (vectores o listas).
- Multidimensionales (tablas o matrices).

ARRAYS UNIDIMENSIONALES: VECTORES

Un Array unidimensional (vector o lista) es un tipo de datos estructurados compuesto de un número *finito* de elementos, *tamaño fijo* y *elementos homogéneos*. Finitos indica que hay un último elemento, tamaño fijo significa que el tamaño del Array debe ser conocido en tiempo de compilación, homogéneo significa que todos los elementos son del mismo tipo.

Los elementos del Array se almacenan en posiciones contiguas de memoria, a cada una de las cuales se puede acceder directamente.

DECLARACION:

type

nombre_array = array [*subíndice*] of *tipo*;

- *nombre_array* identificador válido.
- *subíndice* puede ser tipo ordinal: boolean o char, un tipo enumerado o un tipo subrango.
- *tipo* describe el tipo de cada elemento del vector, los cuales deben ser del mismo tipo.

var

vector: nombre_array;

- *vector* nombre elegido para la variable.

NOTA:

- Los tipos estándar Real e Integer no pueden ser utilizados como un tipo *subíndice*, sin embargo un subrango de enteros puede ser un tipo subíndice.
- El *tipo* de elemento puede ser cualquier tipo definido por el usuario.

OPERACIONES CON VECTORES: los vectores (Arrays) no se pueden leer/escribir en una sola operación o sentencia. La lectura o escritura de un Array se debe hacer elemento a elemento, y para realizar éstas operaciones se deben leer o visualizar los componentes de un Array mediante estructuras repetitivas.

Lectura de un vector:

- Bucles for:

```
for i: = 1 to 100 do

ReadLn (notas [i]);
```

- Bucles while:

```
i: =1;

while (i<=100) do

begin

Read (notas [i]);

i: = i+1;

end;
```

- Bucles repeat:

```
i: =1;

repeat
```

```
Read (notas [i]);

i: = i+1;

until (i>100);
```

Escritura de un vector:

Para visualizar el contenido de un vector en pantalla se utilizan los mismos recorridos vistos en el punto anterior, sustituyendo la sentencia Read o ReadLn por Write o WriteLn. En el primer caso aparecerán los elementos uno al lado del otro y en el segundo uno debajo del otro.

Copia de vectores:

Una operación que se suele dar en ocasiones es la copia de los elementos de un vector en otro vector. Supongamos por ejemplo, que los vectores Alfa y Beta se declaran con:

```
type

ListaReal = array [1..5] of real;

var

Alfa, Beta: ListaReal;
```

Si el vector Alfa tiene asignados valores, éstos se pueden copiar en el vector Beta por la sentencia:

```
for i: = 1 to 5 do

Beta [i]: = Alfa [i];
```

Esta asignación de elementos del vector Alfa al vector Beta se puede hacer de modo más simple mediante la sentencia:

```
Beta: = Alfa;
```

Regla: en gral., un array puede ser asignado a otro array solo cuando ambos tienen el mismo tipo y el mismo tamaño. Esto significa que deben ser declarados por el mismo identificador o identificadores equivalentes.

```
type

VectarA = array [1..20] of real;

VectorB = VectorA;

VectorC = VectorB;
```

ARRAYS BIDIMENSIONALES: MATRICES

Un array bidimensional (tabla o matriz) es un array con dos índices que deben ser ordinales o tipos subrango.

Para localizar o almacenar un valor en el array se deben especificar dos posiciones (dos subíndices), uno para

la fila y otro para la columna. Los elementos se referencian con el formato:

T [3, 4] *elemento de la fila 3 y columna 4.*

DECLARACION:

type

identificador = array [*índice1, índice2*] of *tipo_elemento*;

ó

type

identificador = array [*índice*] of array [*índice2*] of *tipo_elemento*;

var

tabla: identificador;

MANIPULACION DE TABLAS: el orden más natural de procesar los vectores es el orden secuencial: del primero al último elemento. En el caso de las tablas o de los arrays multidimensionales, existen diferentes órdenes para su recorrido. Los más usuales son: recorridos por filas y recorridos por columnas.

Recorrido por fila:

- Bucle for:

```
for Fila: = 1 to 3 do

for Columna: = 1 to 4 do

ReadLn (A [Fila, Columna]);
```

- Bucle while:

```
Fila: = 1;

while (Fila<=3) do

begin

Columna: = 1;

while (Columna<=4) do

begin

Read (A [Fila, Columna]);

Columna: = Columna+1;
```

```
end;

Fila: = Fila+1;

end;
```

Recorrido por columna:

- Bucle for:

```
for Columna: = 1 to 4 do

for Fila: = 1 to 3 do

ReadLn (A [Fila, Columna]);
```

- Bucle while:

```
Columna: = 1;

while (Columna<=4) do

begin

Fila: = 1;

while (Fila<=3) do

begin

Read (A [Fila, Columna]);

Fila: = Fila +1;

end;

Columna: = Columna +1;

end;
```

UNIDAD

SEIS

REPRESENTACION DE ALGORITMOS: TABLAS DE DECISION

TABLAS DE DECISION

DEFINICION:

Las tablas de decisión (TD) constituyen una herramienta poderosa para definir la lógica de un problema complejo. Constituyen un medio de representación de la información en forma tabular, cuyo objetivo

primordial es el de aportar información en un formato que sea fácil de leer y comprender.

Una TD es una herramienta que permite presentar en forma concisa las reglas lógicas que hay que utilizar para decidir acciones a ejecutar en función de las condiciones y la lógica de decisión de un problema específico.

Una TD se representa en un cuadro de cuatro bloques:

MATRIZ DE CONDICIONES	**ENTRADA DE CONDICIONES (Situación)**
MATRIZ DE ACCIONES	**ENTRADA DE ACCIONES (Decisión)**

Representación gráfica de una Tabla de Decisión

La *matriz de condiciones* recoge las condiciones de todo o parte del problema, las preguntas que deben probarse para alcanzar una decisión. La *matriz de acciones* refleja todas o parte de las acciones del problema a realizar. La *entrada de las condiciones* permiten reflejar en la TD si se cumple o no tal condición o si es indiferente, es decir, sea cual su fuere condición de entrada, no tiene influencia sobre la acción que debería ejecutarse. La *entrada de las acciones* indica efectuar la acción correspondiente a un conjunto de condiciones cumplimentadas.

Reglas de decisión:

Cada combinación de entrada de condiciones y su correspondiente entrada de acciones constituyen una relación denominada *regla de decisión*. Existen tantas reglas como pares distintos de entrada de condiciones/acciones. El *número de reglas de decisión* debe cubrir todos los casos posibles, sin repeticiones ni omisiones.

La representación de una TD completa es la siguiente:

Situación

	C1	S	S	S	N		N
	C2	S	N	N	S		N
Condiciones							
	CN	N	N	S	N		S
	A1	X			X		X
	A2		X				X
Acciones							
	AN	X			X		

Regla de decisión

S! Si.

N! No.

X! acción de ejecución.

En blanco! no se realiza.

Las TD siempre se leen de izquierda a derecha y las reglas de arriba hacia abajo.

Las líneas horizontales y verticales dobles sirven como límites o frontera: las condiciones aparecen encima de la doble línea horizontal y los tratamientos debajo de dicha doble línea.

Ejemplo: TD que contiene los pedidos de una empresa y emite facturas a los clientes de acuerdo con las siguientes hipótesis:

- Los pedidos pueden ser de Madrid o exteriores a Madrid, y en cualquier caso se expedirá la factura calculando el precio total de acuerdo con las siguientes condiciones.
- Pedido exterior a Madrid: Imprimir etiquetas de direcciones para envío por correo, cargar gastos en la factura.
- Pedido de Madrid: Si el pedido es superior a 50.000 pesetas hacer un descuento del 5%.

Envío a Madrid	S	S	N	N
Pedido superior a 50.000	N	S	S	N
Imprimir etiqueta			X	X
Calcular precio pedido	X	X	X	X
Añadir portes			X	
Descuento del 5%		X		

La lectura de la TD arroja los siguientes resultados:

- Si el envío es de Madrid y no superior a 50.000 pesetas entonces *calcular el precio pedido* para imprimir factura por ese precio.
- Si el envío es a Madrid y sí es superior a 50.000 pesetas, entonces *calcular precio pedido* y hacer un *descuento del 5%*.
- No es de Madrid el envío y sí es superior a 50.000 pesetas, entonces *imprimir etiquetas*, *calcular precio pedido* y *añadir portes* para emitir factura total.
- No es de Madrid el envío y no es superior a 50.000 pesetas, entonces *imprimir etiqueta* y *calcular precio pedido* para emitir factura total.

TIPOS DE TABLAS DE DECISION

Las TD se pueden clasificar atendiendo a dos criterios:

- *Tipos de entrada de condición*
    - Limitadas.
    - Ampliadas.
    - Mixtas.
- *Procesos complejos (encadenamiento o enlace de tablas)*

- Abiertas.
- Cerradas.

Tablas de decisión limitadas: son aquellas en las que la evaluación de las condiciones está limitada a dos posibles valores: SI o NO, o a la reducción de ambas, el guión (−), en síntesis:

- Las casillas situadas frente a las condiciones se rellenan con SI, NO, o bien, indiferente (−).
- Las casillas situadas frente a las acciones sólo contienen cruces (X).

Tablas de decisión de entradas ampliadas o extendidas: son aquellas tablas donde las distintas condiciones pueden tener un rango de alternativas superiores a dos valores (SI o NO).

Ejemplos claros de aplicación de TD ampliadas son:

- Puesta al día de un fichero. Alternativas posibles son: altas, bajas y modificaciones.
- Valores numéricos múltiples correspondientes a un solo elemento.

1 peseta = 1/135 dólares = 1/220 libras = 1/60 francos...

- Valores acotados.

Compras entre 0 y 1.000 pesetas.

Compras entre 1.000 y 5.000 pesetas.

Compras entre 5.000 y 10.000 pesetas.

EJEMPLO: TD para las ventas de unos grandes almacenes donde el cliente puede pagar al contado o con tarjeta de crédito, y el pedido se remitirá a domicilio cobrándole portes según las siguientes condiciones:

- Si el importe de la compra < 5.000 pesetas ! cargar portes.
- Si el importe de la compra está entre 5.000 y 10.000 pesetas ! cargar 75% de portes.
- Si el importe de la compra está entre 10.000 y 50.000 pesetas ! cargar 50% de los portes.
- Si el importe de la compra > 50.000 pesetas ! no cobrar portes.

En cualquier pedido se incluirá un IVA del 12%.

Hay que tener en cuenta que el número de condiciones es elevado, son 6, lo que significa que tenemos 26 combinaciones o reglas de decisión diferentes, por otra parte las condiciones son excluyentes (si el pedido es <5.000 pesetas, nunca podrá cumplir las otras condiciones).

Al elevado número de condiciones habría que sumarle las acciones, lo que hace que la TD limitada relacionada a éste problema sea muy compleja, entonces se recurre a una TD ampliada.

Tiene tarjeta de crédito.	SI	NO	SI	NO	SI	NO	SI	NO
Importe de la compra.	<5.000		5.000 y 10.000		10.000 y 50.000		>50.000	
Pago de contado.		X		X		X		X
Cargar cuenta.	X		X		X		X	
Cargar IVA.	X	X	X	X	X	X	X	X
Cargar portes.	100%		75%		50%		0%	

Tablas de decisión mixtas: se denominan así a las tablas que resultan de la combinación de las tablas vistas anteriormente.

EJEMPLO: una aplicación de éste tipo de tabla puede ser para la expedición de billetes de ferrocarril (dos clases 1ª y 2ª) y con dos listas de espera (1ª y 2ª).

Se debe:

- Comprobar plazas libres en 1ª y 2ª.
- Petición del viajero.
- Acepta cambio de clase si no existe la solicitada.

La TD ampliada resultante sería la siguiente:

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
Billetes disponibles.	Solo 1ª.			1ª y 2ª.		Solo 2ª.			No hay plazas.			
Petición del viajero.	1ª	2ª	2ª	1ª	2ª	1ª	2ª	2ª	1ª	1ª	2ª	2ª
Cambio de clase.	–	S	N	–	–	S	N	–	S	N	S	N
Expedición billete 1ª.	X	X		X								
Expedición billete 2ª.					X	X		X				
Lista de espera 1ª.							X		X	X	X	
Lista de espera 2ª.			X						X		X	X

ENCADENAMIENTO DE TABLAS DE DECISION

Cuando un problema de lógica es complejo, la TD equivalente resultará compleja. En estos casos es preferible dividir el problema en problemas secundarios de modo que a cada uno de ellos dé lugar a la creación de una tabla particular. Posteriormente las diferentes tablas se encadenan o enlazan para la solución del problema global.

Tablas de decisión abiertas: estas tablas presentan una conexión al principio de la tabla llamada (A), sin necesidad de retorno a la tabla que llama; después de la ejecución de la tabla (A).

Se clasifican en dos grandes grupos:

Encadenamiento en cascada

Encadenamiento en anillo

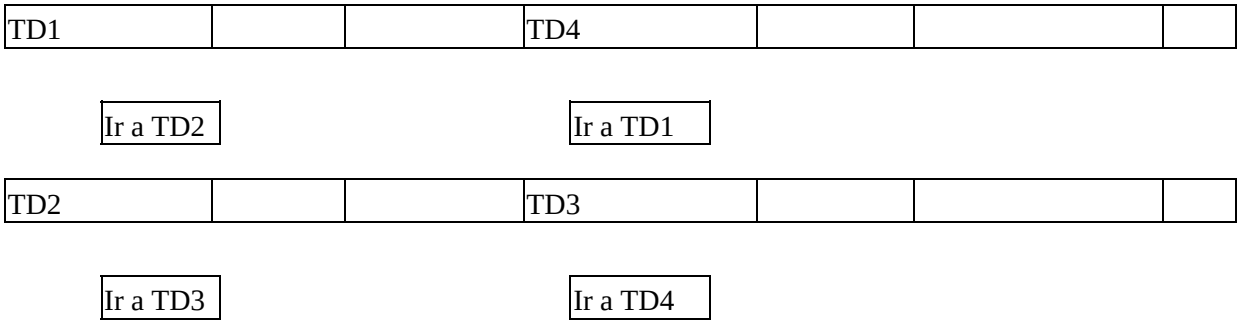
Encadenamiento en cascada

TD1					

	Ir a TD2				
TD2					

	Ir a TD3				
TD3					

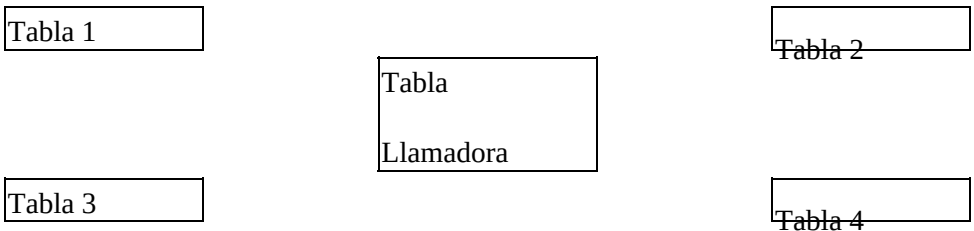
Encadenamiento en anillo



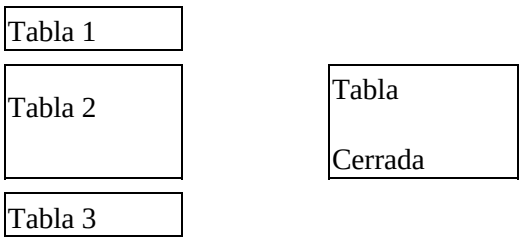
Tablas de decisión cerradas: son aquellas en que una vez ejecutada la tabla llamada, devuelve el control a la tabla que la llamó.

Existen numerosos casos de tablas cerradas:

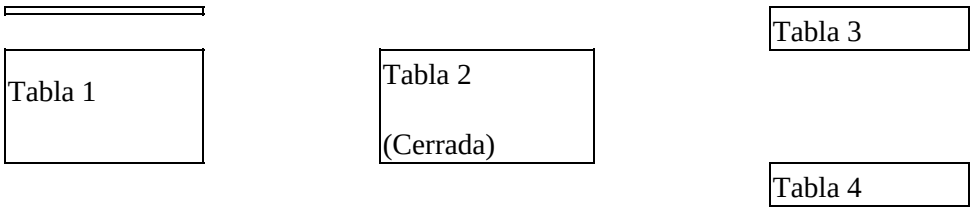
- Se puede llamar desde una tabla a varias tablas cerradas.



- Una tabla cerrada puede ser llamada desde diferentes tablas.



- Una tabla cerrada puede llamar a su vez a otras tablas cerradas.



Las condiciones que se deben cumplir para que pueda existir un encadenamiento son:

- Cada celda se debe identificar mediante una etiqueta (número o serie de caracteres).
- La tabla invocada se ejecutará totalmente.
- Cada regla de decisión debe indicar lo que se ha de hacer a continuación (*bifurcar*, *hacer*, *terminar*, etc.).

CONSTRUCCION DE TABLAS DE DECISION

Son necesarios los siguientes pasos:

- Estudio de condiciones, acciones y sus relaciones.
- Rellenar la TD.
- Analizar la TD.
- Simplificar la TD.
- Estudio de condiciones, acciones y relaciones: tras la lectura del enunciado del problema se localizan las condiciones, que se han de extraer del enunciado, mediante la búsqueda de palabras similares a Si, cuando, sino, etc..

Las acciones se han de localizar en el enunciado con los verbos que implican acción, es decir hacer algo: realizar, ejecutar, sumar, restar, etc.

- Rellenar la tabla: Una vez identificadas las condiciones y las acciones se comienza la escritura de la TD de acuerdo a las siguientes reglas:

Escritura de las condiciones:

  - Realizar un listado de todas las condiciones.
  - Clasificar las condiciones siguiendo el orden lógico −si existe− en el que se consideran. Así en el caso de tres condiciones, órdenes lógicas pueden ser:

N	N	N	N	S	S	S	S	O bien	S	S	S	S	N	N	N		
N	N	S	S	N	N	S	S		S	S	N	N	S	S	N	N	N
N	S	N	S	N	S	N	S		S	N	S	N	S	N	S		N

  - El orden lógico de considerar las condiciones es de arriba hacia abajo.

Una elección lógica del orden de las condiciones simplifica considerablemente la escritura de las reglas de la tabla.

Escritura de las acciones:

  - Realizar el listado de todas las acciones a tomar.
  - Clasificar las acciones mediante un orden cronológico −si existe− según el orden en que deben considerarse. En ocasiones el orden no es importante, pero en otras puede afectar el desarrollo de las acciones.

  - El orden de considerar las acciones es de arriba hacia abajo.

Escritura de las reglas de decisión de las TD limitadas:

- *Escritura de la entrada de condiciones (situaciones):*

  - Las condiciones se deben formular Si (S), No (N) o indiferente (−).
  - Seguir un orden lógico en las entradas. Por ejemplo, primero todos los Sies y a continuación hacer que vayan apareciendo todos los Noes progresivamente (o viceversa). Si una TD contiene n condiciones ! debe tener 2n reglas, casos de no utilizar signos de indiferencia.

- *Escritura de la entrada de acciones:*

Se van colocando cruces (X) frente a las acciones que hay que ejecutar para cada situación o conjunto de condiciones considerada.

Escritura de las reglas de decisión de las TD ampliadas:

Se van escribiendo las condiciones de modo que a cada una de ellas le corresponderá un determinado número de alternativas. Para evitar olvidar reglas es conveniente seguir un método para rellenar las tablas.

Para rellenar las entradas de acciones bastará ir colocando las precisiones relativas a las acciones que se han de ejecutar en las casillas correspondientes.

- Análisis y requisitos de una tabla de decisión: antes de convertir una TD en programa se necesita efectuar un análisis de sus reglas y comprobar que la TD diseñada cumple con los siguientes requisitos:

    - Completa.
    - No tiene redundancias.
    - No tiene contradicciones.

Una TD es *completa* cuando se han establecido todas las situaciones posibles (todas las combinaciones matemáticamente posibles de las entradas o estados de las condiciones).

Una TD *no tiene redundancias* cuando no existen ningunas situaciones idénticas que conduzcan tratamientos idénticos. Existe *redundancia* cuando aparece dos o más veces la misma entrada de condiciones con igual tratamiento.

Una TD *no es contradictoria* cuando siendo redundantes los tratamientos son los mismos. Es *contradictoria* cuando a un mismo estado de condiciones le corresponden tratamientos diferentes.

Si el conjunto de reglas supera 2n, existirán errores.

- Simplificación de una tabla de decisión: La TD es denominada *propia* por indicarse en ella todo el conjunto de condiciones posibles. En el caso de que una o varias combinaciones de condiciones den lugar a las mismas acciones es posible simplificar la TD obteniéndose en ese caso la TD *impropia* o *reducida*, que tiene menor número de reglas.

Éste método consiste en agrupar las reglas simples en reglas compuestas (*reglas simples* son las que no contienen ninguna indiferencia y *reglas compuestas* son las que contiene al menos una indiferencia).

Así por ejemplo en la tabla:

Reglas

	R1	R2	R3	R4	R5	R6	R7	R8
C1	S	S	S	S	N	N	N	N
C2	S	S	N	N	S	S	N	N
C3	S	N	S	N	S	N	S	N
A1	X	X	X	X	X		X	X
A2		X		X				
A3	X		X			X	X	X
A4	X				X			X

Las reglas 2 y 4 producen las mismas acciones:

S S

S N

N S

Por consiguiente en la condición C2, Sí o No, se producen las mismas acciones; eso significa que se pueden simplificar las dos reglas R2 y R4 dejándolas reducidas a una sola R2,4 con la segunda condición representada por un guión (−).

S S S

− equivale a S N

S N S

De igual forma las reglas 3 y 7 producen las mismas acciones:

S N

N N

S S

Por consiguiente se pueden sustituir por una sola regla, R3,7:

–

N

S

Así pues, la TD resultante sería:

Reglas

	R1	R2	R3	R4	R5	R6
C1	S	S	−	N	N	N
C2	S	−	N	S	S	N
C3	S	N	S	S	N	N
A1	X	X	X	X		X
A2		X				
A3	X		X		X	X
A4	X			X		X

Si son más de dos reglas las que contienen iguales acciones, existe un teorema denominado del paraguas, que permite la sustitución de varias reglas simples por una compuesta, siempre que las reglas simples tengan las mismas acciones: se unen mediante un arco aquellas reglas en que solo cambia una condición. Si todos los arcos forman un circuito cerrado, se puede obtener una regla compuesta que sustituya a las reglas simples.

En la TD siguiente es posible simplificar las cuatro reglas simples en una sola compuesta, debido a que se cumple el teorema anterior.

	R1	R2	R3	R4
C1	S	S	S	S
C2	S	S	S	S
C3	S	N	S	N
C4	S	S	N	N
A1				
A2	X	X	X	X

La tabla resultante se obtiene viendo las condiciones que permanecen fijas y poniendo indiferencias en las condiciones que varían.

	R1234
C1	S
C2	S
C3	–
C4	–
A1	
A2	X

La utilización del teorema es solo una opción para que la simplificación no lleve tanto tiempo, pero se la puede realizar (en una forma más segura) agrupando primero las reglas 1 y 2, 3 y 4. Como se puede seguir simplificando se agrupan las nuevas reglas ya simplificadas en una sola regla.

La simplificación de una TD se debe realizar siempre que sea posible.

CONVERSION DE TABLAS DE DECISION A PROGRAMAS: PROGRAMACION DIRECTA

Las reglas de decisión se convierten en instrucciones de los lenguajes. En el caso de la siguiente regla:

	R1	
C1	S	
C2	N	
C3	S	En COBOL se codificaría así:
A1	X	
A2	X	

IF C1 AND NOT C2 AND C3 HACER A1 A2 ELSE...

Esta técnica es cómoda pero tiene el inconveniente de preguntar en la misma pasada varias veces por la misma condición.

Si se han definido tablas encadenadas, cada una de ellas dará lugar a subprogramas abiertos o cerrados según el tipo de tabla.

En el caso de tablas abiertas, la acción *ir a la tabla n* se traduce según el caso, por una instrucción de bifurcación incondicional del tipo GOTO (BASIC o Pascal) GO TO (COBOL), o bien por una instrucción de llamada a subprograma o subrutina mediante la instrucción CALL en COBOL, BASIC, etc.

Si son tablas cerradas, la acción *ejecutar la tabla n* se traduce por una instrucción de llamada de subprograma cerrado (instrucción PERFORM en COBOL).

TRANSFORMACION DE LA TABLA DE DECISION A DIAGRAMA DE FLUJO: METODO DE POLLACK

El método se basa en la localización de las indiferencias de cada regla, siguiendo los siguientes pasos:

- Cálculo del número de reglas simples a las que equivale cada regla (una regla con n indiferencias equivale a 2n reglas simples) y escríbalo al final de cada columna.
- Por cada condición se calcula un coeficiente sumando los valores calculados en el paso anterior y se registra en una columna a la derecha de la TD, el coeficiente de cuenta mide el número de reglas en el que la condición en estudio no interviene (su valor será tanto más importante cuanto menor sea su valor). *Se evalúa en primer lugar la condición cuyo coeficiente sea menor.*
- Si existen varias condiciones con una cuenta mínima igual, se elige un nuevo valor (coeficiente) que es el que mide en valor absoluto la diferencia entre el número de SI y NO que aparecen en dichas condiciones, eligiéndose el de mayor valor absoluto.
- Si subsiste la igualdad se elige indiferentemente cualquiera de las condiciones.
- Se repite el proceso (pasos 1 a 4) hasta la última condición.

Sea por ejemplo una TD de 4 condiciones y 7 reglas con las indiferencias siguientes:

	R1	R2	R3	R4	R5	R6	R7
C1	N	S	–	S	S	N	S
C2	S	N	S	S	–	N	N
C3	–	–	–	N	S	–	N
C4	S	N	N	S	S	–	S

Paso 1

R4 y R7 son reglas simples ! su coeficiente es igual a 1.

R1, R2 y R5 son reglas compuestas con una indiferencia, equivale cada una a dos reglas simples ! a cada una le corresponde un coeficiente igual a 2.

R3 y R6 son reglas compuestas con dos indiferencias, equivale cada una a cuatro reglas simples ! a cada una la corresponde un coeficiente igual a 4.

Paso 2

Condiciones	Indiferencias	Coeficientes
C1	R3	4
C2	R5	2
C3	R1, R2, R3, R6	2+2+4+4=12
C4	R6	4

Se elige la condición C2 por ser la de coeficiente mínimo.

	R1	R2	R3	R4	R5	R6	R7	*Coeficientes*
C1	N	S	–	S	S	N	S	4

C2	S	N	S	S	–	N	N	2
C3	–	–	–	N	S	–	N	12
C4	S	N	N	S	S	–	S	4
Condiciones	2	2	4	1	2	4	1	

Al evaluar la condición C2 (S y N), la TD se transforma en dos tablas derivadas de la eliminación de la condición C2. La primera tabla contiene las reglas en las que la condición C2 tiene el valor S y la segunda el valor N (las indiferencias tienen los valores dobles S y N).

Paso 3

Las evaluaciones siguientes a partir de C2 son para C1 y C4 por orden de coeficiente (C1 para C2 con evaluación N y C4 para evaluación S).

C2 con evaluación S

Regla	Coeficiente	Condiciones	Indiferencia	Coeficiente
R1	2	C1	R3	4
R3	4	C3	R1, R3	2+4=6
R4	1	C4	Ninguna	0
R5	1			

La siguiente condición es C4 (coeficiente 0).

C2 con evaluación N

Regla	Coeficiente	Condiciones	Indiferencia	Coeficiente
R2	2	C1	Ninguna	0
R5	1	C3	R2, R6	2+4=6
R6	4	C4	R6	4
R7	1			

La siguiente condición es C1 (coeficiente 0).

S N

	R1	R3	R4	R5	Coeficientes		R2	R5	R6	R7	Coeficientes
C1	N	–	S	S	4	C1	S	S	N	S	0
C3	–	–	N	S	6	C3	–	S	–	N	6
C4	S	N	S	S	0	C4	N	S	–	S	4

Paso 4 y repeticiones

A partir de C2 se analizarán los dos posibles estados o respuestas S, N.

Para C2 con evaluación S se toma la condición C4 ya que los coeficientes y las condiciones son:

Coeficientes	C1 !4

Condiciones	R1 !2

C3 !6	
C4 !0	

R3 !4
R4 !1
R5 !1

Para C2 con evaluación N se toma la condición C1 ya que los coeficientes y las condiciones son:

Coeficientes	C1 !0
	C3 !6
	C4 !4

Condiciones	R2 !2
	R5 !1
	R6 !4
	R7 !1

El diagrama resultante será:

S N

S N S N

	R1	R4	R5	Coeficientes
C1	N	S	S	0
C3	–	N	S	2

	R2	R5	R7	Coeficientes
C3	–	S	N	2
C4	N	S	S	0

- Estado S.

	R3			R6
C1	–		C3	–
C3	–		C4	–

- Estado N.

A partir de la condición C4 se derivarán C1 y de la condición SI (de C1), C3 con todos sus datos. De la condición C1 se derivarán C4 y de la condición SI (de C4) la condición C3.

Diagrama de flujo derivado de C4.

N

S

N

S

N

Diagrama de flujo derivado de C1.

N

S

N

S

N

S

Para obtener el diagrama de flujo bastará ir uniendo los diagramas de flujos parciales.

Diagrama de flujo total

S N

N N

C4 C1

S S

N N

C1 C4

S S

C3 C3

R5 R4 R1 R3 R5 R7 R2 R6

Las reglas indicarán las acciones a realizar en cada caso según la TD inicial.

La codificación resultante en pseudocódigo será:

SI C2

ENTONCES

SI C4

ENTONCES

SI C1

ENTONCES

SI C3

ENTONCES R5

SINO R4

FIN−SI

SINO R1

FIN−SI

SINO R3

SINO

SI C1

ENTONCES

SI C4

ENTONCES

SI C3

ENTONCES R5

SINO R7

FIN−SI

SINO R2

FIN−SI

SINO R6

FIN−SI

FIN−SI

VENTAJAS DEL USO DE TABLAS DE DECISION

Las TD constituyen una herramienta muy poderosa para definir la lógica de un proceso. Constituyen un medio de representación de la información en forma tabular, cuyo objetivo primordial es el de aportar información en un formato que sea fácil de leer y comprender.

El uso de las TD (aunque no tan extendido como los diagramas de flujo) suele constituir en algunas ocasiones una técnica para capturar datos y la primera operación de análisis del problema. Una vez planteada la TD correspondiente se suele realizar la construcción del diagrama de flujo o algoritmo correspondiente. En ocasiones las TD sustituyen a los diagramas de flujos.

UNIDAD

SIETE

PRUEBA, DOCUMENTACION Y MANTENIMIENTO DE PROGRAMAS

PRUEBA DE PROGRAMAS

En todo proyecto de software, aproximadamente el 50% del tiempo y más de la mitad del costo son empleados en probar el programa o sistema. Una cuestión muy importante es la correcta definición del término PROBAR.

Cuando se prueba un programa se desea agregarle un valor, lo que significa aumentar su calidad o confiabilidad, lo que a su vez significa encontrar y eliminar errores.

De aquí que no se debe probar un programa para mostrar que funciona; es conveniente comenzar con la suposición de que el programa contiene errores y luego probar el programa para encontrar tantos como sea posible.

En consecuencia una buena definición de prueba de programas sería:

Prueba es el proceso de ejecutar un programa con el fin de encontrar errores.

Si nuestra meta es demostrar que el programa no tiene errores, tenderemos a seleccionar datos de prueba con baja probabilidad de hacer que el programa falle.

En cambio, si nos proponemos demostrar que el programa tiene fallas, nuestros datos de prueba tendrán una mayor probabilidad de lograrlo.

Ello implica que *la prueba es un proceso destructivo*, lo que explica por que resulta tan difícil.

Un caso de *prueba exitoso* es el que descubre un error y un caso de *prueba no exitoso* es el que hace que el programa produzca un resultado correcto.

Dada la definición de prueba el paso siguiente es determinar si es posible probar un programa para encontrar todos sus errores. En general es imposible encontrar todos los errores de un programa.

La primera actividad relacionada con la prueba o verificación es de muy bajo rendimiento, se trata de la prueba de escritorio, en la que el propio autor auxiliado por algún colaborador realiza la actividad totalmente informal en la que revisa el producto. Los resultados son necesariamente pobres.

La segunda actividad está relacionada con los códigos generados para los programas, se trata de la inspección del código, en la que participan el autor del código, una persona la que hace las veces de moderador o secretario y un especialista del área de control de calidad.

Para realizar la verificación de un producto, el especialista diseña un conjunto de datos con lo que se realiza la prueba funcional.

Esta prueba determina si la función que especifica un producto coincide con las funciones implementadas en ese producto.

Hay dos enfoques posibles para la prueba funcional del software: prueba del tipo caja negra y prueba del tipo caja blanca.

- PRUEBA DEL TIPO CAJA NEGRA: es llamada también prueba producida por los datos o pruebas producidas por entradas/salidas.

Aquí se desentiende del comportamiento y estructura interna del programa.

Su interés se dirige a encontrar circunstancias en las cuales el programa no se comporta de acuerdo a sus especificaciones. Los datos de prueba se preparan de acuerdo a las especificaciones, el criterio es probar la entrada de datos exhaustivamente.

Para detectar errores de este tipo habría que usar no solamente los datos de entrada válidos sino también todos los datos de entrada posibles. Esto demuestra que la prueba de entrada exhaustiva es imposible.

Dos consecuencias de estas son que:

- No se puede probar un programa para garantizar que está libre de errores.
- Una de las consideraciones fundamentales con respecto a la prueba de programas es la de economía.

El objetivo debe ser: maximizar el número de errores encontrados en un número finito de casos de prueba.

- PRUEBA DEL TIPO CAJA BLANCA: también llamadas lógicas, permiten examinar la estructura interna de un programa. Es una prueba exhaustiva de secuencias, es decir, si se ejecutan todas las posibles secuencias del flujo del control del programa probablemente se podrá decir que está completamente probado.

En esta técnica hay dos posibles inconvenientes:

- El número de secuencias lógicas distintas en un programa es muy grande, con lo que la prueba exhaustiva de secuencia (al igual que la de entrada) no resulta práctica, o es directamente imposible de realizar.
- Es que aunque se haya probado cada secuencia posible de flujo de control, este podría estar todavía plagado de errores.

Para esto hay tres explicaciones:

- Que esta prueba no garantiza de ninguna manera que el programa cumpla con su especificación.
- Un programa puede ser incorrecto por causa de secuencias faltantes.
- Podrá no detectar errores de sensitividad de los datos (caso del valor absoluto).

En conclusión, aunque la prueba exhaustiva de entrada es superior a la de secuencias, ninguna de ellas resulta ser una estrategia útil por que ambas son impracticables.

Sin embargo, se pueden combinar ambos métodos para llegar a una estrategia razonable, aunque no sea perfecta, para la prueba de programas.

PRINCIPIOS DE LA PRUEBA

- La definición del resultado esperado a la salida del programa es una parte integrante y necesaria de un caso de prueba.

Por ello un caso de prueba debe constar de dos componentes:

- descripción de los datos de entrada al programa.
- descripción precisa de la salida esperada para estos datos de prueba.

- Un programador debe evitar probar sus propios programas.

Esto es por que resulta muy difícil para un programador que ha sido constructivo durante todo el tiempo, cambiar súbitamente y adoptar una actitud totalmente destructiva con respecto a su mismo programa.

Además el programa puede contener errores debidos a una falsa interpretación por parte del programador del enunciado del problema.

No se dice que es imposible para un programador llegar a probar sus propios programas, lo que se afirma es que la prueba se hace con mayor efectividad si la realiza otra persona.

- Una empresa de programación no deberá probar sus propios programas.

Aquí el argumento es similar al anterior.

- Inspeccionar concienzudamente el resultado de cada prueba.

Un importante número de errores, son errores expuestos previamente por casos de prueba que escaparon a la detección debido a fallas en la inspección cuidadosa de los resultados de esos casos de prueba.

- Los casos de prueba pueden ser escritos tanto para condiciones de entrada inválidas e inesperadas como para condiciones válidas y esperadas.

Cuando se prueba un programa hay una tendencia natural a concentrarse en las condiciones de entrada válidas y esperadas a costa de ignorar las inválidas e inesperadas.

- Examinar un programa para comprobar que no hace lo que se supone que debe hacer es solo la mitad del problema.

La otra mitad consiste en ver si el programa hace lo que no se supone que debe hacer.

Esto implica que los programas deben ser examinados con respecto a efectos laterales indeseados.

- Evite los casos de prueba desechables a menos que el programa sea verdaderamente desecahble.

Cada vez que debemos probar el programa los casos de prueba deben ser reinventados.

- No planear un esfuerzo de prueba con la suposición tácita de que no se encontrarán errores.

Esto es una mala interpretación de la definición de prueba, es decir, la suposición de que la prueba es el proceso de mostrar de que el programa funciona correctamente.

- La probabilidad de encontrar errores adicionales en una sección del programa es proporcional al número de errores ya encontrados en esa misma sección.
- Las pruebas constituyen una tarea altamente creativa y con un desafío intelectual.

DOCUMENTACION

La documentación tiene un papel importante en todo proyecto de software, pero en la práctica se le presta poca atención, ya sea por que no se la genera o por que no se la utiliza. Sin embargo su necesidad es evidente. Lo importante no es documentar, lo importante es disponer de la información necesaria cuando hace falta.

La documentación es el nexo entre las distintas actividades casi simultáneas que se realizan y es la que permite ejercer el control efectivo del proyecto.

Resulta muy difícil modificar con rapidez el procesamiento de una aplicación cuando no se tiene la menor idea de donde ocurre el proceso del sistema.

Para solucionar este problema, la dirección del proyecto debe tener el coraje de imponer normas de documentación y exigir que la gente se ajuste a ellas.

La documentación persigue objetivos internos y externos. No solo documentamos para comunicarnos entre los miembros del grupo de trabajo, sino también para comunicarnos con el usuario del programa de las decisiones del equipo sobre el producto.

Objetivos Internos

Se refiere a la necesidad de transmitir información de un grupo a otro, por ejemplo del diseñador a los especialistas en bases de datos. Se documenta con el propósito de:

- Comprender con claridad los compromisos que se asumen, las políticas que se definen y los riesgos que se corren.
- Comunicar información referida a los productos de un sector del área a otro.

Objetivos Externos

Se persigue la comunicación con el usuario, por lo tanto se debe lograr que entienda la documentación para que pueda ejercer el control.

Los objetivos son:

- Controlar las alternativas que sufre el proyecto y prevenir con anticipación situaciones indeseables.
- Comunicar a los usuarios las decisiones de diseño referidos a los productos.

¿Que Documentar?

Se documenta todo, la documentación no es una fase del proyecto, es un subproducto.

En la medida en que se debe comunicar algo, se crea un documento. Básicamente todas las decisiones chicas o grandes se documentan.

Existen tantos documentos de sistemas y programas como centros informáticos hay. Esto debido a la falta de normas estándares para documentación.

Todos ellos pueden clasificarse en seis tipos genéricos:

- Diseño conceptual.
- Especificación funcional.
- Especificación del sistema.
- Especificación del programa.
- Manual de operaciones.
- Manual de usuario.

Nos referiremos solamente a los tres últimos.

Existen tres grupos de personas que necesitan conocer la documentación del programa: programadores, operadores y usuarios.

Los requerimientos necesarios para cada uno de ellos son diferentes. , en función de las misiones de cada grupo:

- Programadores: manual de mantenimiento del programa.
- Operadores: manual del operador.
- Usuario: manual del usuario.

ESPECIFICACIONES DEL PROGRAMA: cada programa del sistema debe tener un documento correspondiente de especificación. Si las especificaciones del programa se preparan correctamente, un programador sin previo conocimiento tendría que poder discernir rápidamente como y donde hacer cambios.

Las especificaciones del programa son:

- Nombre del programa.
- Proceso al que pertenece.
- Sistema al que pertenece.
- Lenguaje de programación.
- Descripción de entradas/salidas.
- Descripción de la función general del programa con indicación de las tareas a realizar y del algoritmo de resolución.
- Diseños de impresión y arreglos de pantallas.
- Estructuras de los archivos.

Las secciones que debe agregar el programador son:

- El algoritmo de resolución en forma de diagrama de flujo o tablas de decisión.
- Diagrama de estructuras o diagramas de bloques.
- Flow de pantallas.
- Listado del programa fuente.
- Diseño de los casos de pruebas.
- Manual de mantenimiento con las restricciones y limitaciones e información de operaciones (orden de E/S, tiempos de respuestas, descripción de los mensajes de error, etc.).

El programador describe el funcionamiento y uso del programa en una documentación técnica y de usuario.

La documentación de un programa debe ser de dos tipos: documentación para personal informático y documentación para usuario.

La primera se describió anteriormente.

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

Se divide en:

- Documentación interna.
- Documentación externa.

Documentación Interna: esta documentación cubre los aspectos del programa relativos a la sintaxis del

lenguaje. Esta documentación esta contenida en los comentarios encerrados entre llaves o bien paréntesis o asteriscos.

Algunos tópicos a considerar son:

- Cabecera del programa (nombre del programador, fecha de la versión actual, breve descripción de la función del programa).
- Nombres significativos para describir identificadores.
- Comentarios relativos a la función del programa como un todo, así como los módulos que comprenden el programa.
- Claridad de estilo y formato (una sentencia por línea, líneas en blanco para separa módulos, procedimientos, funciones, unidades, etc.).
- Comentarios significativos.

Documentación Externa: documentación ajena al programa fuente, que se suele incluir en un manual que acompaña al programa.

La documentación externa debe incluir:

- Listado actual del programa fuente, mapas de memoria, referencias cruzadas, etc.
- Especificación del programa: documento que define el propósito y modo de funcionamiento del programa.
- Especificaciones de fórmulas complejas.
- Especificación de los datos a procesar: archivos externos incluyendo el formato de las estructuras delos registros, campos, etc.
- Formato de pantallas utilizados para interactuar con los usuarios.
- Cualquier indicación especial que pueda servir a los programadores que deben mantener el programa.

MANUAL DE OPERACIONES: los manuales de operaciones son los menos genéricos de los documentos del sistema, dado que su formato y contenido varían según la estructura organizacional del departamento de operaciones, el hardware instalado, el software del sistema y la naturaleza de la aplicación (en línea, batch, etc.).

Las secciones que deberían considerarse son:

- Una descripción narrativa de cada trabajo.
- Instrucciones para preparación.
- Procedimiento de rearranque/recuperación.
- Distribución de la salida.

MANUAL DE USUARIO: debe redactarse de modo de que todos los que interactuen con el sistema puedan entender fácilmente el papel que les toca desempeñar en la operación.

La regla de oro en la preparación de este documento es no deje colgado al usuario.

La documentación para el usuario debe incluir una descripción general de las tareas que realiza el programa, así como una descripción detalla de todas las instrucciones que sean necesarias para su instalación, puesta en marcha y funcionamiento; así como consejos, recomendaciones de uso, explicación de los mensajes de error y modo de solucionarlos, entrada y salida de datos, menú de opciones, etc.

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

- Ordenes necesarias para cargar el programa en memoria, desde el almacenamiento secundario (en disco) y arrancar su funcionamiento.
- Nombres de los archivos externos a los que accede el programa.
- Formato de todos los mensajes de error o informes.
- Opciones en el funcionamiento del programa.
- Descripción detallada de la función realizada por el programa.
- Descripción detallada, preferiblemente con ejemplos, de cualquier salida producida por el programa.

¿Quién Documenta?

El *documentador* o *documentalista* tiene a su cargo la recolección, organización y difusión de todos los documentos que se producen en el equipo.

Los documentalistas son los reciben y evalúan consultas sobre campos, programas y estructuras, de hecho, quienes mantienen actualizada la información, garantizando su uso.

EXCESOS: hay dos síntomas distintos y opuestos con relación a la documentación:

- Papiromanía: en la que el foco de atención se centra en los papeles que se producen y no en lo que se produce en los papeles.
- Papirofobia: en la que el incompetente respalda su incapacidad de discernir entre algo bien hecho o mal hecho, evitando tener que decidir hasta que es demasiado tarde, omitiendo todos los documentos.

La documentación es esencial para un proyecto. Pero solo lo esencial es documentación, el resto es exceso.

MANTENIMIENTO

La actividad más cara en la vida del software es el mantenimiento.

Si bien los costos globales de mantenimiento han bajado año tras año, lo cierto es que ocupa por lo menos el 50% de los recursos en cualquier organización.

Esto lo convierte en la etapa más importante de la vida de un producto de software.

Solo el 23% de las tareas corresponden a nuevos desarrollos, el 27% son cambios mayores a alguna aplicación y el 50% restante son cambios menores.

Mantenimiento es la actividad por la cuál se repara, perfecciona o extiende un producto de software.

Esto significa que son tres actividades bajo un mismo nombre: extensivo, ajuste y corrección.

El mantenimiento correctivo comienza tras la detección de errores en el proceso de verificación.

La tarea básica es la depuración del código: rara vez los errores detectados alcanzan proporciones catastróficas que obliguen a reveer el diseño.

Los procesos que tienen lugar durante la depuración son:

- Reproducir el error.
- Diagnosticar la causa.
- Corregirla.
- Verificar la corrección realizada.

En la depuración, el profesional examina un suceso a la luz de su repetibilidad (si un error no es repetible es casi imposible diagnosticar).

La reproductibilidad del error permite realizar experiencias que contribuyen al diagnóstico.

Luego se debe ubicar la porción del código responsable del error, la mejor manera es recorrer el código hacia atrás, partiendo del punto de la impresión del error, y pasando sentencia por sentencia hacia atrás.

Esta técnica se conoce como backtracking, es aplicable solo en los programas con una buena estructuración.

Ubicado el código responsable se procede a corregirlo. Esta tarea merece la mayor atención ya que en la práctica se la subestima. Debe tenerse cuidado, ya que con la corrección del error podemos incorporar otro nuevo.

Un estudio estadístico muestra de que uno de cada dos correcciones introduce un nuevo error.

La verificación de que la corrección efectivamente eliminó el error se debe realizar con el rigor de la prueba de programas nuevos.

La alteración de un sistema implica riesgos grandes.

Una modificación es siempre una corrección: para las nuevas especificaciones, el comportamiento anterior es un error.

Pero ese no es el único efecto a modificar.

Recordemos que el producto de software es mucho más que programas: hay una documentación que mantener y a la que se le dedicó tiempo y dinero.

Si al primer uso que se le da en mantenimiento, esa documentación queda desactualizada, el esfuerzo invertido se pierde. Las herramientas de documentación están al servicio para que ello no ocurra.

A veces ocurre que los programadores descubren tardíamente que las rutinas que han desarrollado independientemente coinciden en un 90%, duplicando innecesariamente esfuerzos y recursos.

Este es un caso de incomunicación dentro del equipo.

Cada modificación descarta un código probado y lo reemplaza por un código nuevo y de calidad desconocida.

Para disminuir esa sensación de incertidumbre y los costos de producción y mantenimiento, una posibilidad es la de generar software en paquetes que puedan ser usados, sacándolos de la biblioteca permitiendo así la *reusabilidad* de los módulos.

Debemos nombrar lo que se conoce como mantenimiento preventivo: no se produce por la detección de un error, sino por la sospecha de que ocurrirá.

La ventaja de modificar antes de lo estrictamente necesario, radica en que es más barato hacer que mantener, sobre todo cuando el producto es conocido, además del efecto colateral positivo de la generación de la documentación actualizada previamente inexistente.

BIBLIOGRAFIA

- Informática: Presente y Futuro.

Donald H. Sanders.

Mc. Graw Hill.

1985. México.

- Metodología de la Programación.

Luis Joyanes Aguilar.

Mc. Graw Hill.

1987. España.

- Estructura interna de la PC.

Gastón Hillar.

H.A.S.A.

1998. Argentina.

- La PC por dentro.

M.C. Guinzburg.

Biblioteca Tecnica Superior.

1998. Argentina.

- Programación en Turbo/Borland Pascal 7.0

Luis Joyanes Aguilar.

Mc. Graw Hill.

1998. España.

- Guía para DOS de Peter Norton, DOS 6.2

Peter Norton.

- Enciclopedia de Microcomputación

- El Gran Libro del Siglo

Autores varios.

BLUME.

1998. Argentina.

- Diccionario Enciclopédico Ilustrado.

Visor Enciclopedias Audiovisuales.

1997. Argentina.

- Revista Solo Programadores

Año III, N° 40

España. Articulo sobre Sistemas Distribuidos

- Apuntes de clase.

Diego Cajal

Javier Macias

En 1981 IBM (la mayor fabricante de computadoras del mundo) presento la PC (Personal Computer), vendiendo ese año 25 mil unidades; tres años mas tarde alcanzo los tres millones. En el interior de la PC se hallaba un microprocesador de la Intel Corporation de Santa Clara, California (EE.UU.), y un sistema operativo (programa que facilita el funcionamiento de otros programas): el MS−DOS de Microsoft, con sede en Seattle.

Sistema Experto: brinda información relevante sobre ciertos temas, realizar diagnósticos, pronósticos, simular situaciones reales, etc., sobre la base de un almacenamiento de conocimientos previos, introducidos por un experto en el tema.

Distintos circuitos de un computador se comunican entre sí mediante un conjunto de conductores (cables o líneas) que interconectan electrónicamente las patas de los chips que contienen dichos circuitos.

Un **bus** de un computador es una estructura de interconexión para la comunicación selectiva entre dos o más módulos de un computador, a fin de poder transmitir información entre dos módulos por vez.

La palabra **registro** en general indica un circuito que puede almacenar temporalmente datos o instrucciones. Los registros de la CPU sirven para guardar información relacionada con la instrucción en curso de ejecución, y con las próximas instrucciones a ejecutar.

Las computadoras no utilizan el sistema de numeración decimal, sino el sistema de numeración en base dos, denominado sistema binario. En el sistema binario cada dígito representa solamente dos posibles valores: 0 ó 1 y se conoce como **BIT** (Binary Digit − dígito binario). Representa la cantidad de información más pequeña que una computadora puede manipular.

Máquina Virtual: el término se utiliza cuando se trata de una red, consiste en ocultar al usuario la existencia de una red de ordenadores, mediante la ilusión de que se trabaja en una máquina única y muy grande.

A los fines de que resulte fácil simbolizar números en otros sistemas numéricos, los símbolos 0 y 1 existen en todos los sistemas, con igual significado que en el decimal. Si otros sistemas usan algunos o todos los símbolos decimales restantes (del 2 al 9), se ha acordado que su significado es el mismo que en decimal. Así 7 representa siete unidades, ya sea en decimal, octal o hexadecimal.

En lo que sigue usaremos los subíndices 10, 2, 16 y 8 cuando sea necesario distinguir entre números *decimales, binarios hexadecimales y octales*, respectivamente

CODIGO ASCII (léase asqui): siglas de American Standard Code for Information Interchange, es un código binario ampliamente usado para la transmisión de información, y para codificar los caracteres de un teclado, así como los que debe imprimir una impresora o mostrar una pantalla.

PSEUDOCODIGO: *Pseudo* o *Seudo* significa falso, imitación y *código* se refiere a las instrucciones escritas en un lenguaje de programación; Pseudocódigo no es realmente un código sino una imitación y una versión abreviada de instrucciones reales de computadoras. Es una técnica para expresar en un lenguaje natural la lógica de un programa, es decir su flujo de control. Ejemplo: DO (hacer), IF−THEN−ELSE (si−entonces−sino), etc.

NOTA: En algunos entornos de programación las funciones de usuario y de programador suelen ser las mismas, en consecuencia, la documentación del programa se puede concretar a especificaciones del programa y al manual del usuario solamente.

F.M.A. / U.C.S.E. − D.A.S.S.

Informática

Página 15 de 106

_____

E

S

CPU

E/S

MEMORIA PRINCIPAL es RAM + ROM

ALU

MP

UC

Reemplazar cada dígito Hexadecimal por el cuarteto binario equivalente.

- Separar a partir de la derecha el número binario en cuartetos, pudiéndose completar con ceros a la izquierda, de ser necesario.
- Reemplazar cada cuarteto por el dígito Hexadecimal equivalente.
- Escribir sobre cada posición binaria su peso decimal.
- Sumar los pesos de las posiciones cuyos bits valen uno (o más en otra base).

Operaciones sobre datos

INICIO

FIN

Entrada de datos

Salida de datos

Proceso

A

B

N

Prueba

A

Prueba

A

B

C

B

M

Prueba

De la

Condición

Prueba

De la

Condición

N

INICIO

FIN

Lectura de M, N, P

Suma=M+N+P

Impresión

De

M, N, P y Suma

Impresión

De

M, N, P y Suma

Suma=M+N+P

Lectura de M, N, P

INICIO

FIN

NOT(FF)

INICIO

FIN

N=0

N=B

B=INT(N/2)*2

N

El número ingresado es CERO

El número ingresado es PAR

El número ingresado es IMPAR

C2

C2

C4

C1

C4

C1

C3

R5

R4

R1

R3

R6

R2

R7

R5

C3

C4

C1

C2