DIRECTORIO DE PROFESORES DEL CURSO " PASCAL "

DEL 28 DE ENERO AL 2 DE FEBRERO, UNIVERSIDAD VERACRUZANA
VERACRUZ, VER.

1. M. EN C. JOSE RICARDO CIRIA MERCE (COORDINADOR)
SUBDIRECTOR DE NUEVOS PROYECTOS
COORDINACION DE LA ADMINISTRACION ESCOLAR
U N A M
EDIFICIO IIMAS PLANTA BAJA CUBICULO "F"
MEXICO, D.F.
550 50 46

2. ING. JUAN ALEJANDRO JIMENEZ GARCIA
JEFE DEL CENTRO DE CALCULO
FACULTAD DE INGENIERIA
U N A M
MEXICO, D.F.
550 57 34

3. ING. SALVADOR MEDINA MORAN
JEFE DE DISEÑO
SUBDIRECCION DE NUEVOS PROYECTOS
COORDINACION DE LA ADMINISTRACION ESCOLAR
U N A M
EDIFICIO IIMAS P.B. CUBICULO P
MEXICO, D.F.
550 50 46 y 45

4. ING. LUIS MIGUEL MURGUIA MARIN
JEFE DE DESARROLLO
SUBDIRECCION DE NUEVOS PROYECTOS
COORDINACION DE LA ADMINISTRACION ESCOLAR
EDIFICIO IIMAS P.B. CUBICULO P
U N A M
MEXICO, D.F.
550 50 46 y 45.

FEBRERO 82.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
FACULTAD DE INGENIERIA
DIVISION DE EDUCACION CONTINUA

P A S C A L

L U N E S

PROFESORES :
M.C.JOSE RICARDO CIRIA M.
ING.SALVADOR MEDINA MORAN

INTRODUCCION A LA COMPUTADORA DIGITAL.
CONCEPTO DE PROGRAMA
DIAGRAMAS DE SINTAXIS.
DECLARACION DE UN PROGRAMA EN PASCAL.
PROGRAMA SENCILLO Y EXPLICACION DE CADA UNO DE SUS ELEMENTOS.
HISTORIA,Y COMENTARIOS ACERCA DEL LENGUAJE PASCAL.
BREVE DESCRIPCION DEL SISTEMA
INTRODUCCION AL EDITOR DEL SISTEMA

PRIMERA PRACTICA

USO DEL EDITOR.
COPIA,COMPILACION Y EJECUCION DE UNO O MAS PROGRAMAS CORTOS

M A R T E S

PROFESORES :
ING.SALVADOR MEDINA MORAN
M.C.JOSE RICARDO CIRIA M.

IDENTIFICADORES.
TIPOS PREDEFINIDOS DE DATOS.
CONSTANTES.
OPERADORES ARITMETICOS.
ASIGNACION.
INTRODUCCION A E/S : READ Y WRITELN.
ENTRADA SALIDA. READ,READLN,WRITE,WRITELN.
FUNCIONES PREDEFINIDAS.
INTRODUCCION A LA PROGRAMACION ESTRUCTURADA.
FIGURAS LOGICAS.

SEGUNDA PRACTICA

DESARROLLO DE UN PROGRAMA A PARTIR DE SU PSEUDOCODIGO.

M I E R C O L E S

PROFESORES :
ING.SALVADOR MEDINA MORAN
M.C.JOSE RICARDO CIRIA M.
ING.LUIS MIGUEL MURGUIA MARIN

OPERADORES LOGICOS.
INTRODUCCION A DECISION E ITERACION : IF-THEN-ELSE ; CASE ; WHILE.
CONTINUA DECISION E ITERACION : REPEAT. FOR TO.
ARREGLOS DE UNA DIMENSION DE TIPOS PREDEFINIDOS.
ARREGLOS DE CARACTERES EMPACADOS.

TERCERA PRACTICA

COMPILACION Y EJECUCION DE UN PROGRAMA DESARROLLADO EN CLASE.

J U E V E S

PROFESORES :
ING.LUIS MIGUEL MURGUIA MARIN
M.C.ALEJANDRO JIMENEZ GARCIA

SUBRANGOS.
DECLARACION DE ARREGLOS DE UNA DIMENSION Y UTILIZACION DE
SUBRANGOS COMO SUBINDICES.
ARREGLOS DE N DIMENSIONES.

SUBRUTINAS Y FUNCIONES.
PASO DE PARAMETROS POR VALOR Y POR NOMBRE.

CUARTA PRACTICA

COMPILACION Y EJECUCION DE PROGRAMAS UTILIZANDO LOS CONCEPTOS
ANTERIORMENTE EXPUESTOS.

V I E R N E S

PROFESORES :
ING.LUIS MIGUEL MURGUIA MARIN
M.C.ALEJANDRO JIMENEZ GARCIA.

ESCALARES.
CONJUNTOS.
EJEMPLOS VARIOS

QUINTA PRACTICA

COMPILACION Y EJECUCION DE EJEMPLOS DESARROLLADOS.

S A B A D O

PROFESORES :
ING.LUIS MIGUEL MURGUIA MARIN
M.C.ALEJANDRO JIMENEZ GARCIA.

COMPILACION Y EJECUCION DE EJEMPLOS DESARROLLADOS.

EVALUACION DEL PERSONAL DOCENTE

(1)

CURSO: "PASCAL"

FECHA: 28 DE ENERO AL 2 DE FEBRERO

	CONFERENCISTA	DOMINIO DEL TEMA	EFICIENCIA EN EL USO DE AYUDAS AUDIOVISUALES	MANTENIMIENTO DEL INTERES. (COMUNICACION CON LOS ASISTENTES, AMENIDAD, FACILIDAD DE EXPRESION).	PUNTUALIDAD	
1.	M.EN C. JOSE R. CIRIA MERCE					
2.	ING. SALVADOR MEDINA MORAN					
3.	ING. JUAN ALEJANDRO JIMENEZ GARCIA					
4.	ING. LUIS MIGUEL MURGUIA MARIN					
5.						
6.						
7.						
8.						
9.						
	ESCALA DE EVALUACION: 1 a 10					

EVALUACION DE LA ENSEÑANZA

SU EVALUACION SINCERA NOS AYUDARA A MEJORAR LOS PROGRAMAS POSTERIORES QUE DISEÑAREMOS PARA USTED.

	TEMA	ORGANIZACION Y DESARROLLO DEL TEMA	GRADO DE PROFUNDIDAD LOGRADO EN EL TEMA	GRADO DE ACTUALIZACION LOGRADO EN EL TEMA	UTILIDAD PRACTICA DEL TEMA	
	INTRODUCCION A LA COMPUTADORA DIGITAL					
	CONCEPTO DE PROGRAMA					
	DIAGRAMAS DE SINTAXIS					
	DECLARACION DE UN PROGRAMA EN "PASCAL"					
	PROGRAMA SENCILLO Y EXPLICACION DE CADA UNO DE LOS ELEMENTOS.					
	HIST. Y COMENTARIOS ACERCA DEL LENGUAJE					
	BREVE DESCRIPCION DEL SIST. VAX 11/780					
	PRIMERA PRACTICA					
	IDENTIFICADORES					
	INTRODUCCION AL EDIT. DEL SIST.VAX 11/780					

ESCALA DE EVALUACION: 1 a 10

EVALUACION DE LA ENSEÑANZA

3

SU EVALUACION SINCERA NOS AYUDARA A MEJORAR LOS PROGRAMAS POSTERIORES QUE DISEÑAREMOS PARA USTED.

TEMA	ORGANIZACION Y DESARROLLO DEL TEMA	GRADO DE PROFUNDIDAD LOGRADO EN EL TEMA	GRADO DE ACTUALIZACION LOGRADO EN EL TEMA	UTILIDAD PRACTICA DEL TEMA	
TIPOS PREDEFINIDOS DE DATOS					
CONSTANTES					
OPERADORES ARITMETICOS					
ASIGNACION					
INTRODUCCION A E/S READ Y WRITELN					
ENTRADA Y SALIDA.READ,READLN,WRITE,WRITELN					
FUNCIONES PREDEFINIDAS					
INTRODUC. A LA PROGRAMACION ESTREUCTURADA					
FIGURAS LOGICAS					
SEGUNDA PRACTICA DESARROLLO DE UN PROGRAMA					

ESCALA DE EVALUACION : 1 a 10

EVALUACION DE LA ENSEÑANZA

SU EVALUACION SINCERA NOS AYUDARA A MEJORAR LOS PROGRAMAS POSTERIORES QUE DISEÑAREMOS PARA USTED.

TEMA	ORGANIZACION Y DESARROLLO DEL TEMA	GRADO DE PROFUNDIDAD LOGRADO EN EL TEMA	GRADO DE ACTUALIZACION LOGRADO EN EL TEMA	UTILIDAD PRACTICA DEL TEMA	
OPERADORES LOGICOS					
INTRODUCCION A DESICION O INTERACCION..					
ARREGLOS DE UNA DIMENSION DE TIPOS PREDEFI.					
ARREGLOS DE CARACTERES EMPACADOS					
TERCERA PRACTICA COMPILACION Y EJECUCION DE UN PROGRAMA					
SUBRANGOS					
DECLARACION DE ARREGLOS DE UNA DIMENSION					
ARREGLOS DE N. DIMENSIONES					
SUBRUTINAS Y FUNCIONES					
PASO DE PARAMETROS POR VALOR Y POR NOMBRE					
ESCALA DE EVALUACION: 1 a 10					

EVALUACION DE LA ENSEÑANZA

SU EVALUACION SINCERA NOS AYUDARA A MEJORAR LOS PROGRAMAS POSTERIORES QUE DISEÑAREMOS PARA USTED.

TEMA	ORGANIZACION Y DESARROLLO DEL TEMA	GRADO DE PROFUNDIDAD LOGRADO EN EL TEMA	GRADO DE ACTUALIZACION LOGRADO EN EL TEMA	UTILIDAD PRACTICA DEL TEMA	
CUARTA PRACTICA COMPILACION Y EJECUCION DE PROGRAMAS...					
ESCALERES					
CONJUNTOS					
EJEMPLOS VARIOS					
QUINTA PRACTICA COMPILACION Y EJECUCION DE EJEMPLOS					
ESCALA DE EVALUACION: 1 a 10					

EVALUACION DEL CURSO

	CONCEPTO	EVALUACION
1.	APLICACION INMEDIATA DE LOS CONCEPTOS EXPUESTOS	
2.	CLARIDAD CON QUE SE EXPUSIERON LOS TEMAS	
3.	GRADO DE ACTUALIZACION LOGRADO CON EL CURSO	
4.	CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
5.	CONTINUIDAD EN LOS TEMAS DEL CURSO	
6.	CALIDAD DE LAS NOTAS DEL CURSO	
7.	GRADO DE MOTIVACION LOGRADO CON EL CURSO	

ESCALA DE EVALUACION DE 1 A 10

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

CURSO: "PASCAL" DEL 28 DE ENERO AL 2 DE FEBRERO
VERACRUZ, VER.

A P U N T E S

M. EN C. RICARDO CIRIA MERCE
ING. ALEJANDRO JIMENEZ GARCIA
ING. SALVADOR MEDINA MORAN
ING. LUIS M. MURGUIA MARIN

FEBRERO 1985

pascal

División de Educación Continua
Facultad de Ingeniería
Universidad Nacional Autónoma de México

Estas notas son una guía de los temas presentados en este curso. En ningún momento pretenden ser un libro de texto. Su objetivo es sustituir a los apuntes o notas que se pudieran tomar durante la exposición de los temas, en la idea de que el alumno no distraiga su atención de ellos para efectuar anotaciones. Incluyen una amplia bibliografía a la cual sugerimos se acuda para efectos de consulta o referencia.

Muy atentamente

Ricardo Ciria Merce
Alejandro Jiménez García
Salvador Medina Morán
Luis Miguel Murguía Marín
Antonio Pérez Ayala

CAPITULO 1 INTRODUCCION

1.1 DESARROLLO HISTORICO DE DISPOSITIVOS AUTOMATICOS DE CALCULO 1-1
1.2 CONCEPTO DE COMPUTADORA 1-8
1.3 BREVE HISTORIA DEL LENGUAJE PASCAL. 1-19

CAPITULO 2 INTRODUCCION AL LENGUAJE PASCAL

2.1 GENERALIDADES Y VENTAJAS. 2-1
2.2 MODULARIDAD Y ESTRUCTURACION DE PROGRAMAS. 2-3
2.3 COMPARACION CON OTROS LENGUAJES. 2-4

CAPITULO 3 ESTRUCTURA DE UN PROGRAMA EN PASCAL.

3.1 DIAGRAMAS DE SINTAXIS. 3-2
3.2 DECLARACION DE UN PROGRAMA EN PASCAL. 3-4

CAPITULO 4 TIPOS DE DATOS

4.1 IDENTIFICADORES 4-1
4.2 CONSTANTES. 4-1
4.3 ALMACENAMIENTO DE DATOS 4-2
4.3.1 TIPO INTEGER 4-2
4.3.2 TIPO REAL 4-2
4.3.3 TIPO BOOLEAN 4-2
4.3.4 TIPO CHAR 4-3
4.4 OPERADORES ARITMETICOS 4-3
4.5 ASIGNACION 4-3
4.6 EXPRESION ARITMETICA 4-3

CAPITULO 5 INSTRUCCIONES DE ENTRADA Y SALIDA

5.1 INTRODUCCION 5-1
5.2 READ 5-2
5.3 WRITE 5-4
5.4 READLN 5-6
5.5 WRITELN 5-8
5.6 CONSIDERACIONES SOBRE LAS INSTRUCCIONES DE ENTRADA Y SALIDA. 5-9
5.7 EOLN Y EOF. 5-13
5.8 MISCELANEAS SOBRE LAS INSTRUCCIONES DE ENTRADA Y SALIDA 5-15
5.9 CONSIDERACIONES FINALES. 5-17

CAPITULO 6 DECISION E ITERACION

6.1 OPERADORES LOGICOS 6-1
6.2 INSTRUCCION IF 6-2
6.3 INSTRUCCION REPEAT 6-4

6.4 INSTRUCCION WHILE 6-6
6.5 INSTRUCCION FOR 6-7

CAPITULO 7 ARREGLOS

7.1 DIAGRAMA DE SINTAXIS. 7-3
7.2 ARREGLOS MULTIDIMENSIONALES. 7-4
7.3 ARREGLOS EMPACADOS. 7-7
7.4 ARREGLOS BOOLEANOS 7-8

CAPITULO 8 FUNCIONES Y PROCEDIMIENTOS

8.1 FUNCIONES PREDEFINIDAS 8-2
8.2 FUNCIONES DEFINIDAS POR EL USUARIO. 8-5
8.3 PROCEDIMIENTOS 8-10
8.4 PASO DE PARAMETROS POR REFERENCIA 8-14

CAPITULO 9 TIPOS NO PREDEFINIDOS DE DATOS.

9.1 ESCALARES. 9-2
9.2 SUBRANGOS. 9-8
9.3 CONJUNTOS 9-11

APENDICE A DIAGRAMAS DE SINTAXIS

APENDICE B INTRODUCCION AL SISTEMA VAX-11/780

B.1 COMO ENTRAR AL SISTEMA. B-1
B.2 COMO SALIR DEL SISTEMA. B-2
B.3 COMO VER LOS ARCHIVOS ALMACENADOS EN LA CLAVE. . . B-2
B.4 COMO BORRAR UN ARCHIVO. B-2
B.5 COMO SACAR UN LISTADO DE UN PROGRAMA. B-2
B.6 COMO CREAR UN NUEVO ARCHIVO Y COMO HACERLE MODIFICACIONES B-2
B.7 COMO COMPILAR, LIGAR Y CORRER UN PROGRAMA. B-3
B.8 COMANDOS DE EDICION. B-3
B.9 COMO MANDAR IMPRIMIR A PAPEL LOS RESULTADOS DE UN PROGRAMA. B-4

APENDICE C PROGRAMACION ESTRUCTURADA

C.1 TEOREMA DE LA ESTRUCTURA C-2

APENDICE D PRACTICAS

D.1 PRIMERA PRACTICA D-2
D.2 SEGUNDA PRACTICA D-7
D.3 TERCERA PRACTICA D-8
D.4 CUARTA PRACTICA D-9
D.5 QUINTA PRACTICA D-10

APENDICE E EJEMPLOS

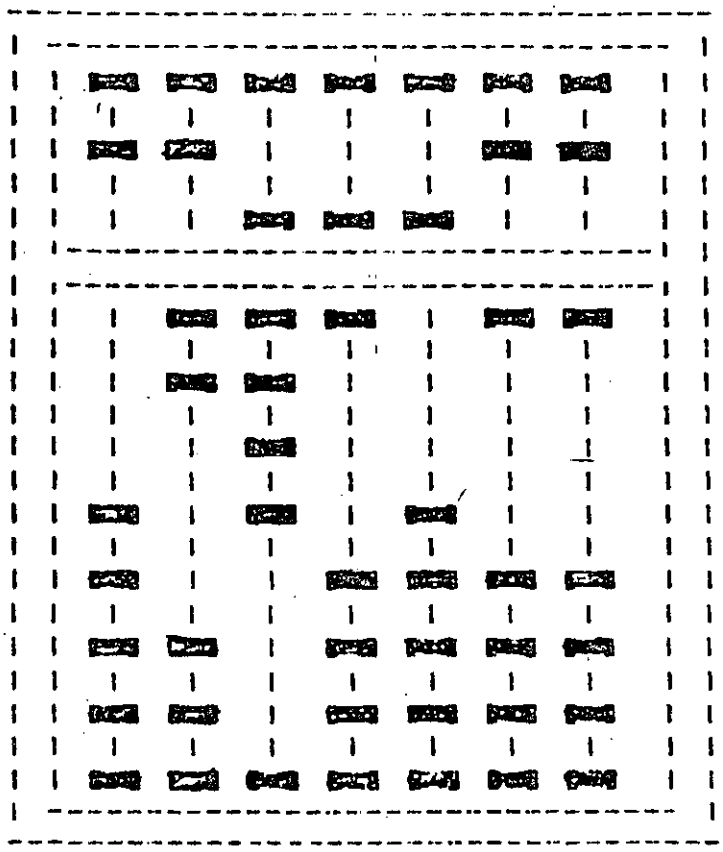
APENDICE F BIBLIOGRAFIA

CAPITULO 1

INTRODUCCION

1.1 DESARROLLO HISTORICO DE DISPOSITIVOS AUTOMATICOS DE CALCULO

2500 años A.C.- Una de las primeras herramientas mecánicas de cálculo fué el ábaco, del cual se encuentran versiones primitivas en el Medio Oriente.



0 2 9 6 5 1 1

A B A C O

El ábaco es aun muy popular en algunos círculos; de hecho, un operador de ábaco bien entrenado puede sumar columnas de números con mayor rapidez que muchos operadores de calculadoras electrónicas.

1614 - Se publican las estructuras de NAPIER como una herramienta para multiplicar, las cuales fueron desarrolladas por un noble escocés llamado JOHN NAPIER. Una familia completa de estructuras de Napier consta de 9 hileras, una por cada uno de los dísitos del 1 al 9. Cada hilera es, en esencia, una columna de una tabla de mutiplicación.

1		3	1.8	4	384 * 6 = 2304
/2		/6	1/6	/8	
/3		/9	2/4	1/2	
/4		1/2	3/2	1/6	
/5		1/5	4/0	2/0	
/6	X	1/8	4/8	2/4	= 2304
/7		2/1	5/6	2/8	
/8		2/4	6/4	3/2	
/9		2/7	7/2	3/6	

1633 - El clérigo inglés, WILLIAM OUGHTRED, inventó un dispositivo para calcular basado en los logaritmos de Napier, que denominó "círculos de proporción" que llegaría a ser conocido como la regla de cálculo. Rápida, portátil y barata, fué muy popular entre los científicos y los ingenieros hasta hace muy poco tiempo.

1642 - A la edad de 19 años el filósofo y matemático BLAISE PASCAL desarrolló una calculadora de ruedas giratorias, predecesora de la popular calculadora de escritorio, construida precisamente para ayudar a su padre, quién era cobrador de impuestos en el pueblo de Roven. Su funcionamiento es similar al de un odómetro de automóvil. Solo podía sumar y restar, e indirectamente multiplicar, (mediante sumas sucesivas) y dividir (mediante restas sucesivas).

1822 - CHARLES BABBAGE (1792 - 1871), matemático e ingeniero inglés, profesor de matemáticas en la Universidad de Cambridge, construyó el modelo funcional de una máquina para calcular tablas denominada "máquinas de diferencias".

1833 - BABBAGE concibe la idea de mejorar sustancialmente la máquina de diferencias, la que bautizó como la 'máquina analítica', más general que la máquina de diferencias, la máquina analítica podía ser 'programada' para evaluar un amplio intervalo de funciones diferentes.

A pesar de que el diseño estuvo definitivamente completo, nunca llegó a construirse, en gran parte debido a que la tecnología de la época no estaba lo bastante avanzada. Tuvo que pasar casi un siglo, para que ideas similares a éstas fueran puestas en práctica. BABBAGE es considerado por muchos como el padre de la computadora actual.

1835 - ADA AUGUSTA, condesa de Lovelace, conoce a CHARLES BABBAGE y empieza a trabajar en el proyecto de la máquina analítica.

Nació en 1815, hija del poeta inglés Lord Byron, se casó con el conde de Lovelace. En 1842 tradujo del inglés al italiano una primera descripción de la máquina analítica, añadiendo muchas notas por su cuenta. Se refirió a 'ciclos de operación', al repetido uso de las tarjetas en estructuras del tipo de subrutinas y se refirió también a la computación no numérica y a la manipulación simbólica.

Observó que la máquina analítica no 'originaba nada' y que solo podía hacer 'aquello que uno sabía como ordenarle que realizara'. Una de sus notas fué una descripción detallada para calcular los números de Bernoulli con la máquina analítica que para muchos fué el primer 'programa'. Murió de cáncer en 1852, casi un siglo antes de que apareciera la primera computadora de programas almacenados.

Muchos autores le reconocen el honor de haber sido la primera programadora y la llaman la madre de las computadoras actuales.

1880 - HERMAN HOLLERITH, experto en estadística, fué comisionado por el Census Bureau de los Estados Unidos de América para desarrollar un proceso que agilizara el censo de 1880. Hollerith propuso que los datos fueran perforados en tarjetas y tabulados automáticamente con ayuda de máquinas diseñadas especialmente.

Las tarjetas perforadas habían sido usadas en 1745 por el francés Joseph Marie Jacquard, que diseñó un método para utilizar los agujeros en una tarjeta, para controlar la selección de los hilos en los diseños de los tejidos en los telares. Con este nuevo procedimiento, los datos del censo de 1890 fueron procesados totalmente en menos de 3 años; cuando en el censo de 1880 se llevaron 8 años para procesar la información, calcularon que en el de 1890 tardarían más de 10 años.

En 1896 Hollerith organizó la Tabulating Machine Company, que fué adquirida después por la International Business Machine Corporation (IBM).

Las tarjetas utilizadas por Hollerith son de 80 columnas con perforaciones rectangulares.

1930.- GEORGE STIBITZ de la Bell Telephone Laboratories, a pedido del ejército de los Estados Unidos desarrolla el proyecto de 5 computadoras de gran escala a las que se denominó Computadoras de relays de Bell debido a que utilizaban relays electromecánicos como componentes operacionales básicos.

Estas máquinas probaron que podían realizar operaciones las 24 horas del día y durante 7 días a la semana, con muy pocos errores y muy poco tiempo perdido debido a fallas.

También en esta época, en el edificio de física del Iowa State College, John Vincent Atanasoff con ayuda de Clifford Berry, comenzó a formular los principios de la primera calculadora electrónica automática. Una vez terminado este prototipo a principios de la década de los 40's, fué capaz de resolver con un alto grado de exactitud ecuaciones simultáneas de hasta 29 incógnitas.

Entre los principios importantes que se incorporaron a ésta máquina están el uso de la base 2 o binaria, en vez de la base estándar 10 o decimal y la incorporación de una memoria de máquina resenerable.

Atanasoff y su trabajo no fueron reconocidos sino hasta 1973. Un juez de distrito de Minneapolis, falló un juicio de violación de patente por 200 millones de dólares en el que estaban envueltas dos compañías de computadoras.

En Europa en este decenio también se trabajó mucho en ésta area. El alemán Konrad Zuse se dedicó al diseño y construcción de máquinas computadoras. Su primer intento fué la Z1, construida en la sala del departamento de sus padres en Berlín. Su trabajo fué destruido en la Segunda Guerra Mundial.

Inglaterra desarrolló el sistema 'Coloso', Una computadora construida por la inteligencia británica que entró en servicio en diciembre de 1943. La información sobre las computadoras 'Coloso' y su uso permanece aun hoy clasificada como secreta.

1944 - HOWARD AIKEN terminó un calculador gigantesco con ayuda de IBM denominado MARK I. Fué la predecesora inmediata de las computadoras automáticas. Aiken estaba familiarizado con el trabajo de Charles Babbage y reconoció su influencia en sus trabajos. El MARK I se denominó posteriormente "el sueño de Babbage hecho realidad".

1946 - JOHN W. MAUCHLY y J. PRESPER ECKERT de la Escuela de Ingeniería Electrónica Moore de la Universidad de Pennsylvania, financiados por el Laboratorio de investigaciones balísticas de Maryland construyeron la primera computadora a gran escala completamente electrónica. El ENIAC (calculador e integrador numérico electrónico). Mauchly fué influenciado por el trabajo de Atanasoff en Iowa.

Estaba compuesto de cerca de 18000 tubos de vacío (bulbos), 70000 resistencias y 10000 capacitores que consumían 150 Kilowats de potencia. Ocupaba un espacio superior a 15000 pies cuadrados (4572 m2, 67.5 m. de lado) y pesaba alrededor de 30 toneladas.

Muchos predijeron que el ENIAC nunca trabajaría debido a la baja confiabilidad de los tubos de vacío, sin embargo fué terminado con éxito y durante su tiempo de vida trabajó 80223 horas (casi 10 años). Podía ejecutar 5000 sumas o restas ó 300 multiplicaciones por segundo ! con lo cual se transformó en el más rápido de sus contemporáneos. Su mayor defecto fué su limitada memoria. Opacado por las nuevas máquinas que fueron apareciendo, el ENIAC fué retirado del servicio en 1955.

La programación de estas computadoras era un trabajo arduo y pesado que requería estar totalmente familiarizado con la máquina, con los detalles de su operación, una gran cantidad de ingenio y mucha paciencia.

Los programas del ENIAC se cargaban y cambiaban modificando los alambrados de sus componentes; tarea de uno o dos días. John Von Neuman, consultor del proyecto ENIAC, fué el primero en proponer el concepto de programa almacenado. Sugirió que las instrucciones podían almacenarse en la computadora junto con los datos.

Esta idea, conocida ahora como el concepto de Von Neuman, aumentó la flexibilidad y aplicabilidad de la computadora en dos sentidos:

a) Las instrucciones podían cambiarse sin tener que volver a alambrar manualmente las conexiones (y por lo tanto más rápidamente).

b) Las instrucciones serían almacenadas como números y la computadora podría procesarlas como si fuesen datos, haciendo posible la alteración de ellos en su secuencia y la modificación automática de ellas.

Von Neuman, el grupo de la ENIAC y H.H. Goldstine, iniciaron la construcción de una computadora de programas almacenados, el EDVAC, pero su terminación se vió retrasada hasta 1952. Fué el prototipo de las computadoras en serie.

1949 - En mayo sale la primera computadora que podía almacenar un programa digital. El EDSAC (Electronic Delay Storage Calculator) fué construido en la Universidad de Cambridge en Inglaterra bajo la dirección de M.V. Wilkes.

1951 - Mauchly y Eckert pasaron del proyecto ENIAC a la construcción de una máquina denominada "Computadora Automática Universal" o UNIVAC que fué la primera computadora digital producida comercialmente. Sus características principales son:

a) Era capaz de alcanzar una alta velocidad debido a que utilizaba diodos de cristal en lugar de bulbos, anticipando la era del estado sólido.

b) Podía leer información.

c) Podía efectuar operaciones y escribir información de salida en forma simultánea.

d) Tenía un complicado sistema de cinta magnética.

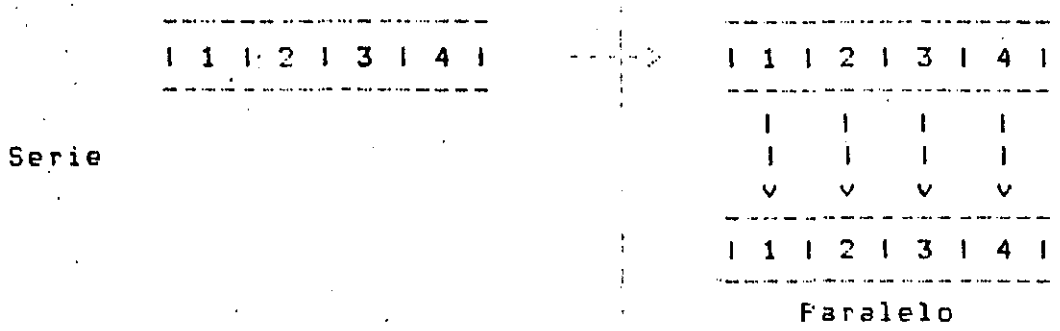
e) Sus dispositivos de entrada y salida eran relativamente veloces.

La primera UNIVAC fué instalada en el U.S. Census Bureau. La primera instalación comercial se efectuó poco después en la nueva planta de la General Electric en Kentucky.

1952 - En el Instituto de Estudios Avanzados de la Universidad de Princeton bajo la supervisión de John Von Neuman sale la computadora IAS; también se termina la EDVAC.

Estas máquinas introdujeron los diseños básicos para dos importantes tipos de computadoras: en serie y en paralelo (IAS). Las diferencias entre los dos tipos estriba en los métodos de mover información de una parte de la máquina a otra y de la manera en que se lleva a cabo la suma.

La máquina paralela mueve todos los dígitos que forman un número al mismo tiempo y la máquina en serie mueve un dígito a la vez.



Cuando se usa el concepto de 'en paralelo' para la suma, todos los correspondientes pares de dígitos se suman simultáneamente. En una máquina en serie los pares de dígitos se suman un par a la vez, casi de la misma manera que en la aritmética manual.

1953 - IBM instala su primera computadora, la IBM 701.

1954 - IBM instala su primera IBM 650, máquina de tamaño mediano, pensaba vender menos de 50 y vendieron más de un millar.

1956 - IBM saca a la venta la IBM 704, científica, de gran escala y mucho éxito.

1959 - IBM lanza la IBM 7090, de las primeras máquinas transistorizadas.

1960 - Burroughs lanza la B5500, es la primera computadora que trabaja con el principio de stack. Es la primera computadora con Hardware orientado, también la primera computadora que se construye para un lenguaje en especial. Es la primera en cuya totalidad el Software está programado en lenguaje de alto nivel (ALGOL).

1961 - IBM lanza la serie IBM/360 con objeto de estandarizar el equipo de IBM. Tuvo un tremendo impacto en la totalidad de la industria de las computadoras, miles de IBM/360 se han instalado en todo el mundo, con lo que se estableció la posición dominante de IBM en el mercado.

Se instala la primera minicomputadora; la PDP1 de la Digital Equipment Corporation. Fué instalada en el Instituto Tecnológico de Massachusets.

1969 - IBM lanza la IBM/1130, computadora científica mediana de gran popularidad.

1970 - Se construye el primer microprocesador de marca Intel que sale inmediatamente al mercado.

1973 - Sale al mercado la CRAY I de la CRAY Corporation, primera computadora que puede realizar más de un millón de operaciones en un segundo.

1978 - Sale al mercado la primera supermini, con 32 bits en la palabra.

1981 - Científicos Japoneses anuncian las computadoras de la 5a. generación.

1984 - CRAY CORPORATION lanza al mercado, en el mes de marzo la CRAY II, es la primera computadora que puede desarrollar más de 200 millones de operaciones por segundo.

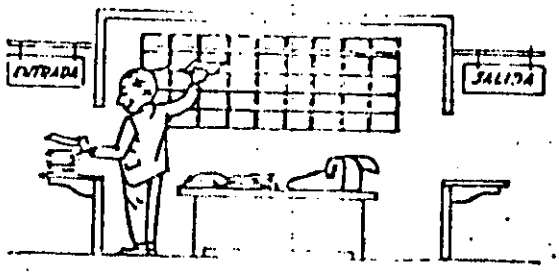
1.2 CONCEPTO DE COMPUTADORA

El objeto de esta breve reseña sobre las computadoras electrónicas y sus múltiples aplicaciones al servicio del hombre, es transmitir al lector una completa visión de conjunto, mediante un lenguaje sencillo que permita comprender conceptualmente los temas tratados, sin necesidad de conocimientos previos en la materia.

Esperamos que estas páginas, muy simples en apariencia pero con profundo contenido, permitan, a quienes las lean, ingresar al maravilloso mundo de las máquinas automáticas.

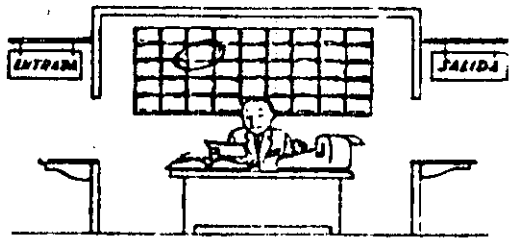
Este señor se llama Control. Trabaja en una pequeña habitación. Tiene a su disposición una máquina de calcular que suma, resta, multiplica y divide. Tiene también el señor Control un archivo parecido al casillero que existe en los trenes para clasificación postal. Hay,

además, en la habitación, dos ventanillas identificadas con sendos carteles: 'Entrada' y 'Salida'. El señor Control tiene un manual, que le indica cómo debe desenvolverse con estos elementos, si alguien le pide que haga un trabajo.



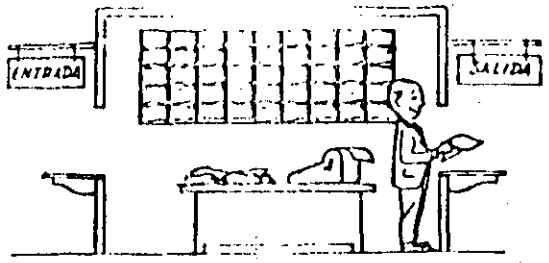
Una persona quiere saber el resultado de un complicado cálculo. Para ello, escribe ordenada, precisa y detalladamente, cada una de las operaciones que, en conjunto, integran ese cálculo, anota cada instrucción elemental en una hoja de papel y coloca todas las hojas en orden en la ventanilla 'Entrada'.

El señor Control, al ver las hojas, lee en su manual que debe tomar esas hojas con instrucciones, una por una, y colocarlas correlativamente en su archivo. Y así lo hace.

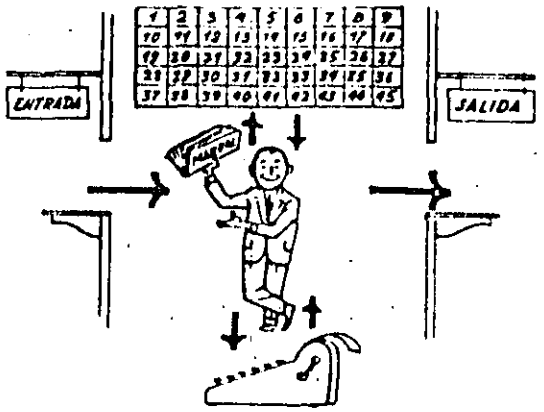


Una vez ubicadas todas las instrucciones en el archivo, el señor Control consulta nuevamente el manual. Allí se le indica que, a continuación, debe tomar la instrucción de la casilla 1 y ejecutarla, luego la de la casilla 2 y ejecutarla, y así sucesivamente hasta ejecutar la última instrucción. Algunas instrucciones indicarán que hay que sumar una cantidad a otra (instrucciones aritméticas); otras, que el señor Control debe ir a la ventanilla 'Entrada' para buscar algún dato que intervenga en el cálculo (instrucciones de 'entrada/salida'), dato que la persona que le formuló el problema habrá colocado ya en dicha ventanilla, en otra hoja de papel. Finalmente, otras instrucciones indicarán que debe elegirse una de entre dos alternativas (instrucciones lógicas): por ejemplo, supongamos que una parte del cálculo desde la instrucción que está en la casilla 5 del archivo hasta la que está en la casilla 9 debe ejecutarse 15 veces porque el cálculo así lo exige. En tal caso, la instrucción que está en la casilla 10 indicará que, si los pasos 5 a 9 se han ejecutado menos de 15 veces, se debe volver al paso 5. Cuando se hayan realizado las 15 repeticiones y

no antes, el señor Control seguirá con la instrucción de la casilla 11.



Después de ejecutar todas las instrucciones del archivo, haciendo con la máquina de calcular las operaciones en ellas indicadas, el señor Control entrega, a través de la ventanilla 'Salida', los resultados obtenidos ... y se sienta a esperar un nuevo trabajo.



Obsérvese que la actuación del señor Control es puramente mecánica: sólo sigue las indicaciones de su manual y cumple de acuerdo con ellas, las instrucciones que recibe a través de la ventanilla 'Entrada'. Toma decisiones, pero solamente cuando se le señalan las alternativas que existen y con qué criterio debe elegir una de ellas. El señor Control puede resolvernos cualquier problema, por complicado que éste sea. Pero para ello debemos indicarle paso a paso, en la forma más elemental y detallada, todo lo que debe hacer para resolverlo, sin olvidarnos absolutamente nada porque, en ese caso, el señor Control no sabría continuar por sí mismo. Haga el lector la prueba de formular un problema cualquiera de modo tal que una persona que no conozca nada acerca de ese problema, pueda resolverlo sin necesidad de hacer consultas. Verá que es una experiencia interesantísima.

El esquema que acabamos de representar mediante el señor Control y sus elementos de trabajo, corresponde exactamente al esquema de funcionamiento de una computadora electrónica.

A continuación presentaremos una breve descripción de los elementos de la computadora que corresponden a los elementos de trabajo del señor Control.

Las unidades de Entrada (representadas por la ventanilla 'Entrada'): Son en la computadora, dispositivos capaces de leer información (Instrucciones o Datos) con el objeto de procesarla. Existen una gran variedad de elementos de entrada, entre los cuales tenemos:

Tarjetas de Cartulina y Cintas de Papel: Que son perforadas de manera que cada perforación representa un número, una letra o un símbolo especial de acuerdo con un código predeterminado.

Cintas magnéticas: Conocidas como 'memorias externas' tienen la ventaja de permitir almacenar la información en forma más concentrada (a razón de 800 a 6250 caracteres por pulgada de longitud) y de ser más veloces, ya que pueden enviar o recibir información a la unidad de Control a velocidades que van de 10,000 a 680,000 caracteres por segundo. Pueden llegar a tener hasta 2400 ft. de longitud.

Disco Magnético: También conocidos como 'Memoria externa', en general tienen un diámetro aproximado de 30 cm. y pueden grabar hasta 400,000,000 de letras, números, y caracteres especiales, formando palabras, cifras, o registros completos. Se pueden grabar o leer a razón de 77,000 a 312,000 caracteres por segundo y su tiempo de acceso a un registro alcanza un promedio de 30 milisegundos.

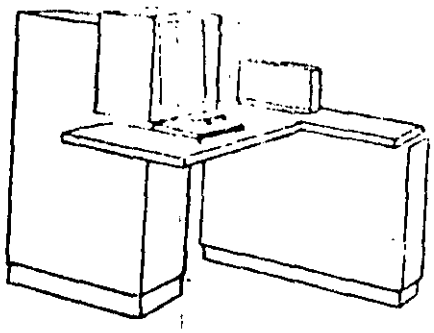
Una diferencia importante entre las cintas y los discos es la siguiente:

En las cintas los registros se graban o leen secuencialmente.

En los discos se tiene 'libre acceso' a un registro cualquiera, en forma inmediata, pues cada registro se localiza por su posición física dentro del disco.

Lectora Optica de Caracteres Impresos: Puede leer un documento impreso por una máquina de escribir, o por una máquina de contabilidad o por la impresora de una computadora, a una velocidad de 30,000 caracteres por minuto.

Unidad de Representación Visual: Esta unidad de entrada/salida sirve para hacer consultas a la computadora, por medio de un teclado de máquina de escribir, y obtener la respuesta reflejada en una pequeña pantalla de televisión. La imagen está formada por hasta 24 renglones de hasta 80 caracteres (letras, números, o signos especiales) cada uno.



Vemos aquí otra Unidad de Representación Visual, más evolucionada que la anterior, la comunicación hombre-máquina puede establecerse en ella por medio de gráficas, es decir que la entrada y la salida de datos se hace por medio de imágenes.

Cuenta esta unidad para ello con un dispositivo con forma de lápiz, que tiene en su punta una celda fotoeléctrica. Un delgado haz de luz parte en determinado momento de un punto de la pantalla y la recorre en forma de zig-zag. Si se apoya el "lápiz" en cualquier posición de la pantalla, su celda fotoeléctrica detectará en algún momento el haz de luz.

Por el tiempo transcurrido desde que el haz de luz comenzó su "barrido" hasta que fue detectado, la computadora determina en qué punto de la pantalla se encuentra apoyado el "lápiz".

Como el barrido dura una fracción de segundo y se realizan muchos barridos por segundo, se puede "escribir" con el "lápiz" sobre la pantalla y el dibujo "ingresa" en la memoria de la computadora como una sucesión de puntos codificados.

La pantalla está imaginariamente dividida en 1,040,576 puntos, de manera que los trazos que se obtienen son prácticamente continuos. Pueden dibujarse así curvas, estructuras, letras, números y cualquier tipo de gráfica, y esa información ingresa automáticamente a la computadora.

Por otra parte, los resultados obtenidos por la computadora son representados en la pantalla también como curvas, letras, etc., bajo control del programa almacenado en la memoria.

Lectora Optica de Manuscritos: Salvo algunas pequeñas restricciones en cuanto al formato de los caracteres, esta unidad puede "leer" documentos escritos por cualquier persona y con cualquier ejemplo a una velocidad aproximada de 30,000 caracteres por minuto.

El registrador/analizador Fotográfico es una unidad de Entrada/Salida de datos que realiza las siguientes funciones.

1. Registra los resultados de la computadora sobre microfotografías, mediante un tubo de rayos catódicos, que inciden sobre una película fotográfica, y cuyo haz electrónico actúa gobernado por el Programa Almacenado. La película se revela automáticamente dentro de la unidad y 48 segundos después está lista para ser proyectada.

2. Proyecta sobre una pantalla translúcida las microfotografías registradas.

3. Analiza imágenes reproducidas en negativo sobre película transparente, las digitaliza y las transmite a la Unidad Central de Procesamiento.

La película utilizada tiene 30.5 milímetros de ancho y 120 metros de longitud. la Entrada o Salida de imágenes puede consistir en letras, números, símbolos, dibujos, gráficos, mapas, curvas, etc. En una microfotografía de 30.5 mm x 30.5 mm pueden registrarse hasta 30,600 letras y números, o hasta 16,777,216 puntos correspondientes a imágenes.

La velocidad de Registro/Análisis es de 40,000 letras, números y símbolos por segundo, o su equivalente si se trata de imágenes.

Máquina de Escribir (Teletipo).

Las unidades de almacenamiento o memorias (Representadas por el archivo del señor Control) permiten registrar las instrucciones y los datos para resolver un problema; entre éstas se tienen:

Los Anillos Magnetizantes: Estos pueden magnetizarse en un sentido o en otro 'recordando' así un 1 o un 0 respectivamente. Con 8 de éstos anillos se forma una posición de memoria, en la cual puede registrarse una letra, un dígito o un carácter especial, según las distintas combinaciones de anillos 'En 1' y 'En 0', de acuerdo a un código predeterminado.

Las Memorias de flip-flops

Las Cintas Magnéticas

Los Discos Magnéticos

El dispositivo aritmético (representado por la máquina de calcular) que realiza las cuatro operaciones aritméticas.

Las unidades de salida (representadas por la ventanilla 'Salida') que pueden ser:

Impresoras

Máquinas de Escribir (Teletipos)

Grabadoras de Cintas Magnéticas

Grabadoras de Discos Magnéticos

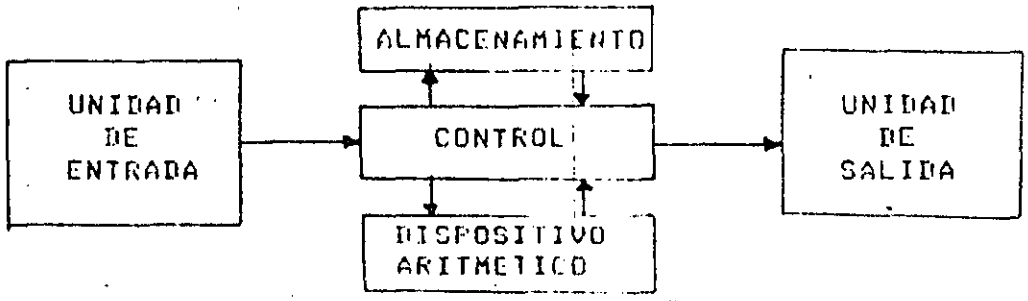
Unidad de Representación Visual

Registrador Analizador Fotográfico

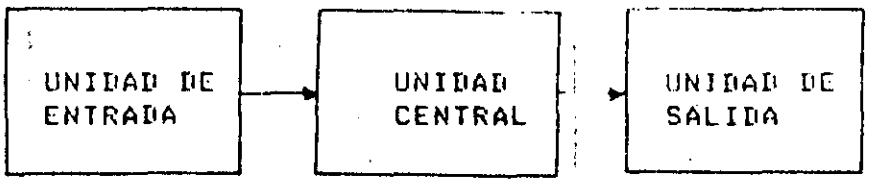
Unidad de Respuesta Oral con la cual la computadora puede hablar en todo el sentido de la palabra. Contiene una cinta magnetofónica en la cual un locutor ha grabado un diccionario de una gran variedad de palabras, en cualquier idioma.

Finalmente, un **dispositivo electrónico de control** (representado por el señor Control) ayudado de un **programa especial o sistema operativo** (representado por el manual del señor Control), sobierna todas las unidades que componen la computadora.

Habiendo descrito las partes que componen la computadora podemos mostrar el siguiente esquema que la representa:



O en forma más resumida:



Siendo:

UNIDAD CENTRAL
- ALMACENAMIENTO
- CONTROL
- DISPOSITIVO ARITMETICO

Hemos hablado hasta este momento de la computadora electrónica desde el punto de vista conceptual. Durante las dos últimas décadas se han producido avances tecnológicos tan extraordinarios en materia de electrónica que la computadora ha sufrido enormes transformaciones. Veremos ahora cómo se ha ido modificando la idea orisinal hasta llegar a los más modernos sistemas de procesamiento de datos.

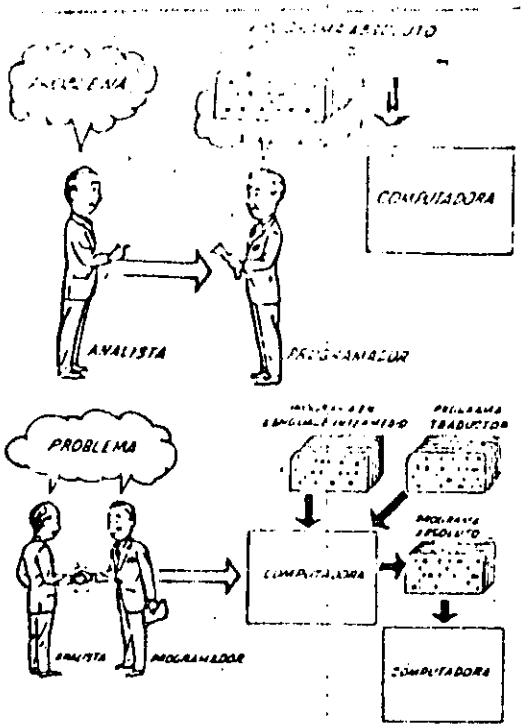
Las Primeras computadoras tenian circuitos con válvulas de vacio. Los tiempos de operación se median en ellas en milisegundos (milésimas de segundo). Cuando aparecieron los transistores, el diseño de los circuitos se mejoró notablemente y la duración de las operaciones en las computadoras que utilizaban esta 'Tecnología de Estado Sólido' se midió en microsegundos (millonésimas de segundo).

El hecho de que las nuevas máquinas fueran miles de veces más rápidas que las anteriores, trajo aparejada la creación de unidades de entrada, salida y memoria de forma mucho más veloces.

La invención de un nuevo tipo de transistor ('chip') provocó una verdadera revolución en los circuitos electrónicos y sus procesos de fabricación. El nuevo elemento es tan pequeño que en un dedal de costura caben más de 50,000 chips. Debido a su tamaño, se les denomina circuitos microminiaturizados o microcircuitos. Los tiempos de operación se miden ahora en nanosegundos (milmillonésimas de segundo). Ha nacido en esta forma la tercera generación de computadoras, y las altas velocidades alcanzadas posibilitaron un nuevo enfoque en el diseño de los sistemas de procesamiento de datos.

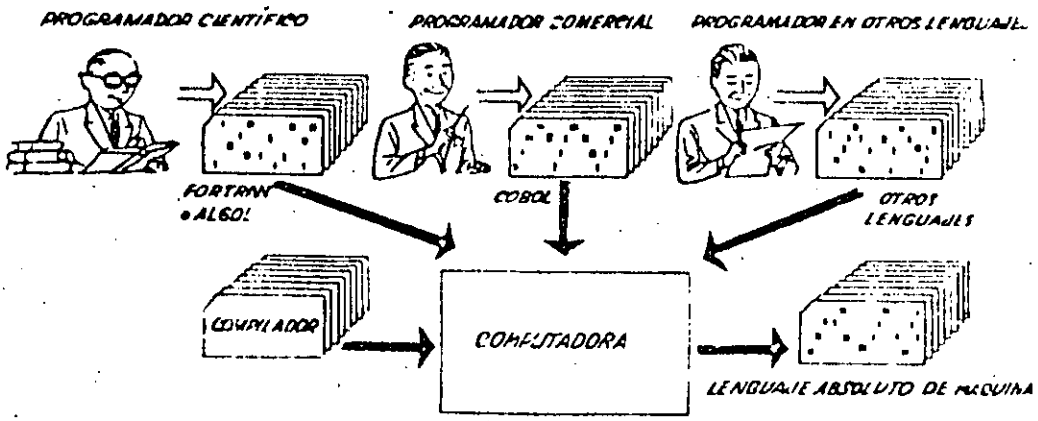
Enunciaremos brevemente los adelantos que esta tercera generación ha introducido con respecto a la tecnología anterior:

- La computadora se autogobierna y trabaja sin detenerse, pasando de un trabajo a otro sin demora alguna.
- El Operador interviene sólo cuando algún problema excepcional ocurre. La comunicación entre hombre y máquina se realiza sólo sobre la base de 'Informes por Excepción'.
- Si ocurre una falla en los circuitos o en la parte electromecánica la máquina realiza un autodiagnóstico e indica cual es la anomalía.
- La velocidad de Entrada-Proceso-Salida se ha incrementado extraordinariamente.
- Todas las operaciones del sistema se realizan en forma simultánea.
- Los lenguajes de programación han revolucionado de manera notable.
- El autocontrol y la autoverificación de operaciones han alcanzado niveles insospechados.
- Pueden realizarse, con máximo rendimiento, varios trabajos distintos simultáneamente. (Multiproceso).



Hasta ahora hemos visto muchas unidades que, en distintas combinaciones, confisuran computadoras electrónicas para las más variadas aplicaciones. Ahora nos detendremos para analizar el manejo de dichos sistemas.

El programa de Instrucciones almacenado en la Unidad Central de Procesamiento, consta de una secuencia de órdenes y comandos, expresados sesún una codificación especial denominada "Lensuaje de Máquina". Las primeras computadoras se "programaban" en este complejo lensuaje. Había entonces una enorme diferencia entre nuestro idioma y aquél sesún el cuál debíamos comunicarnos con la máquina. Esto oblisaba a un sran esfuerzo común entre el analista que conocía el problema, y el programador que conocía la computadora, pues ambos hablaban del mismo proceso en distintos lensuajes.



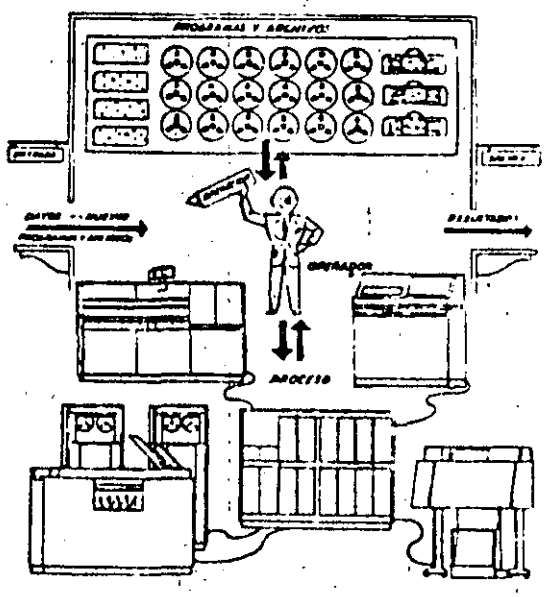
Se crearon, para solucionar el problema, lenguajes intermedios cada vez más parecidos a nuestro idioma. Es decir que cada nuevo lenguaje intermedio se acercaba más al problema y se alejaba más de la máquina. Para cada uno de estos lenguajes se creó un programa traductor llamado 'Compilador' o 'Ensamblador' que tenía la misión de traducir el lenguaje intermedio al de máquina. Ahora, el analista y el programador 'hablan un mismo idioma': ambos conocen el problema y la solución.

Pero la computadora sesuía desarrollándose, y pronto los lenguajes intermedios fueron insuficientes para formular intrincados problemas científicos o comerciales. Nacieron, entonces, lenguajes especializados: dos de ellos, el FORTRAN y el ALGOL, permiten programar problemas científicos-técnicos utilizando una notación casi idéntica a la notación matemática común. El COBOL es un lenguaje comercial cuyas sentencias configuran oraciones y frases en forma tal que una persona que no sabe qué es una computadora, que puede leer un programa y entender perfectamente qué es lo que hará la máquina cuando lo tenga almacenado.

Cada uno de estos lenguajes tiene un programa Compilador para cada tipo distinto de computadora capaz de procesarlo. Esto significa que un programador que sabe FORTRAN, por ejemplo, puede programar una computadora aún sin conocerla. Es decir que estos tres lenguajes constituyen un 'Esperanto' de las máquinas.

La tercera generación de computadoras permitió abordar complejos problemas que incluían, entre otros, aspectos comerciales y científicos.

Hemos llegado así a que la computadora nos 'entienda', en lugar de que se limite a recibir órdenes en su idioma.



1.3 BREVE HISTORIA DEL LENGUAJE PASCAL.

El lenguaje Pascal marca una etapa en el desarrollo de los lenguajes de programación de computadoras, ya que es el primer lenguaje que engloba de una manera coherente los conceptos de programación estructurada que habían sido definidos por Edsger Dijkstra y C.A.R. Hoare.

El lenguaje de programación Pascal, es el resultado del esfuerzo de desarrollar en los últimos años, iniciado dentro del Working Group 2.1 of IFIP, un nuevo lenguaje que fuera el sucesor del ALGOL 60. Los primeros esfuerzos para desarrollar el nuevo lenguaje se concretan en 1965 con el ALGOL W, el cual puede considerarse el predecesor directo del Pascal.

Una primera versión del lenguaje Pascal fue realizada en 1968. El nuevo lenguaje seguía, en cuanto a su espíritu, la línea de los lenguajes ALGOL 68 y el ya mencionado ALGOL W.

Después de un importante impulso en los años 1968-1970, aparece en el año 1970 el primer compilador operativo, el cual se publica en 1971. El Pascal tal como lo conocemos, fue desarrollado por Niklaus Wirth en la Eidgenossische Technische Hochschule de Zurich (Confederación de Escuelas Técnicas Superiores de Zurich). El gran interés suscitado por las publicaciones escritas por Wirth ('The Programming Language Pascal' y 'The Design of a Pascal Compiler'), motivó el desarrollo de otros compiladores y como consecuencia la consolidación del lenguaje Pascal. Dos años de experiencia en la utilización del lenguaje dieron lugar a que en el año de 1973 apareciera la publicación de un 'Revised Report' y una definición y representación de un lenguaje en términos del conjunto de caracteres ISO (International Standard Organization).

Pascal es ahora ampliamente aceptado como un útil lenguaje que puede ser eficientemente implementado, y como una excelente herramienta de enseñanza.

CAPITULO 2

INTRODUCCION AL LENGUAJE PASCAL

2.1 GENERALIDADES Y VENTAJAS.

Hay dos razones principales para programar en un lenguaje de alto nivel como Pascal en lugar de usar un lenguaje de bajo nivel como lo es el ensamblador. Primero, es más fácil escribir programas en un lenguaje de alto nivel. Segundo, y ésto es lo más importante, es más fácil leer y entender un programa escrito en un lenguaje de alto nivel. Los programadores profesionales gastan mucho tiempo revisando programas escritos por ellos mismos o por otros. Si no se entienden totalmente estos programas, sus modificaciones durante la revisión podrían incluso arruinar los programas. Por tanto, es tan importante aprender a leer y a entender programas como lo es aprender a escribirlos.

Por otro lado, al comienzo de la vida de una instalación, se desarrollan programas para aplicaciones nuevas, que parecen que van a ser definitivos e inalterables; pero poco a poco la dura realidad se hace sentir en forma de modificaciones y reformas que van socavando la mejor (o no tanto) estructura inicial de los programas. Si un programa es incapaz de ser modificado, entonces su inflexibilidad hace que caiga en desuso y se abandone al cabo de algún tiempo. Bajo este punto de vista se fue viendo, poco a poco, la importancia de la actividad del mantenimiento de un programa, es decir, su adaptación continua a los cambios tales como los siguientes:

a) Aumento de volumen de datos o registros.

b) Cambios en la organización de los archivos.

c) Paso de los archivos a bases de datos.

d) Cambios de legislación que afectan a nóminas y seguros sociales, por ejemplo.

e) Cambios estructurales y/o departamentales de la empresa, que motivan modificaciones en el tipo y formato de los informes de salida.

f) Cambios en el propio equipo de procesamiento de datos.

Están lejos los tiempos que se empleaba el concepto de "programa definitivo", en los que un programa compilado se usaba rutinariamente, llegándose a olvidar (e incluso extraviar) el programa fuente original.

Las modificaciones hay que esperarlas y, si no desearlas, por lo menos ser capaz de vivir con ellas; de lo contrario, se puede llegar a situaciones límite en las que con el paso del tiempo pueden necesitarse gran parte del personal de un equipo de procesamiento de datos, para el mantenimiento de los programas hechos en el pasado; naturalmente ésto es una nueva situación límite, que al ir acercándose motiva que se tomen medidas correctoras.

Quizás el primero que captó este cambio de mentalidad fue G. Weinberg que con su libro clásico: "The psychology of computer programming" empezó a hacer énfasis en el aspecto humano de la programación. A continuación se resumen algunos puntos que menciona:

a) Es importante el leer programas. Weinberg indica que aunque haya otros tipos de lectura más amena, se puede aprender mucho de dicha lectura.

b) Como consecuencia de lo anterior un programa no debe escribirse con la creencia de que sólo lo va a leer un compilador, sino también alguna persona (incluso uno mismo), ya sea con el fin altruista de aprender, o con el más prosaico de tener que modificar o adaptar el programa a una nueva necesidad.

c) Entonces, insiste Weinberg en evitar las sentencias complejas (por ejemplo, afirma que cinco niveles de paréntesis anidados parece ser una especie de límite humano que no es conveniente sobrepasar).

d) Con lo que venimos diciendo de la actividad del mantenimiento de programas, es esencial que un programa se pueda leer y comprender con facilidad con el fin de que sea fácil el introducir cambios, sin tener que comenzar un nuevo calvario depuratorio de igual o mayor importancia que el que se realizó al poner a punto el programa original.

Asimismo, es difícil trabajar en el campo de la computación por largo tiempo sin toparse con el hecho de que hay dos escuelas de metodologías de programación, y por tanto, dos clases de programadores. Dichas escuelas son:

a) La computación científica y

b) el procesamiento administrativo y comercial de datos.

Los programadores que trabajan para algunos de los campos mencionados, tienden a pensar que los problemas del otro campo carecen de interés, son triviales e inútiles.

Este problema es tan profundo, que hay muchos lenguajes, y aun computadoras, diseñadas para resolver problemas científicos o administrativos, pero no ambos. Sin embarso, ésto tiene cierta justificación: la programación administrativa requiere un eficiente acceso a grandes cantidades de información con relativamente pocos cálculos; mientras que la programación científica frecuentemente involucra una pequeña o mediana cantidad de datos, pero una intensa explotación de los recursos de cálculo. Estos requerimientos hacen que haya diferencias aun en la representación numérica interna, pues programas con enfoque comercial requieren un rango pequeño de valores, pero una buena precisión, mientras que los programas con enfoque científico requieren un mayor rango de valores y también de una buena precisión.

Esta dicotomía tiende a ocultar el hecho de que los problemas básicos de programación son actualmente muy similares. Además, los más recientes lenguajes de programación tratan de tender un puente para salvar dichos problemas, ésto es, proporcionan tanto "características" científicas como administrativas. Precisamente, uno de estos lenguajes es el Pascal, el cual, con relativamente un número pequeño de construcciones básicas, puede usarse efectivamente para resolver problemas tanto científicos como administrativos.

2.2 MODULARIDAD Y ESTRUCTURACION DE PROGRAMAS.

Insistiendo sobre lo dicho en la sección anterior, si un programa debe ser claro, intelisible y corto, para que una persona lo comprenda y sea capaz de introducirle mejoras, todo ello de una forma fácil y con pocas o ninguna repercuciones, a continuación se darán algunas características que deberá poseer dicho programa:

a) Ser secuencial, es decir, que nuestra atención no tenga que dispersarse continuamente al intentar atrapar la lógica del programador en cuestión. Dijkstra afirma que las bifurcaciones son una de las causas principales de errores de un programa y además lo hacen difícil de seguir o leer por una persona.

b) Ser Estructurado, es decir, que se pueda escribir un programa empleando sólo las siguientes estructuras básicas: PROCESO SECUENCIAL, IF-THEN-ELSE, y la instrucción de iteración WHILE.

c) Ser corto.

d) Ser fragmentado: como los programas de la vida real tienen un tamaño considerable, es importante el conseguir dividirlos en fragmentos, que en nuestro caso Pascal lo realiza a través de las funciones (FUNCTIONS) y subrutinas (PROCEDURES). Lo anterior no es fácil de hacer pero con dedicación y ejercicio se adquiere soltura en dividir un programa en segmentos que realicen su misión de una forma lo más independiente posible del resto del programa, facilitando así su lectura y modificación en su caso.

Las características del lenguaje Pascal permiten hacer uso de las anteriores características de "Programación Estructurada".

Asimismo, en la actualidad hay una multitud de lenguajes incompatibles que causan el problema de la transportabilidad, es decir, un programa en cierto lenguaje compilado y ejecutado en cierta máquina, puede no funcionar en otra, aun teniendo el compilador del mismo lenguaje. Un programa en Pascal en su forma estándar, puede correr en cualquier computadora que tenga el compilador del lenguaje Pascal.

2.3 COMPARACION CON OTROS LENGUAJES.

El Pascal es un lenguaje muy logrado y de una difusión extraordinaria, que puede causar la falsa impresión de ser un hecho aislado en los lenguajes de programación, cuando en realidad es una etapa en un hecho evolutivo continuo como es el de ir buscando lenguajes de programación cada vez más adecuados al usuario. Dicho movimiento evolutivo es lento por varias causas obvias:

a) No hay medidas cuantitativas para apreciar la mejor o peor calidad de un lenguaje.

b) Hay varios objetivos de diseño distintos para los lenguajes de programación, por ejemplo: sencillez, complejidad, metas particulares, etc. El número tan elevado de los lenguajes de programación existentes es un indicio de esta evolución con objetivos dispersos y confusos.

c) Hay objetivos de diseño que sí son concretos e importantes como la facilidad de compilación y uso.

En esta sección se tratará de relacionar y comparar el Pascal con otros lenguajes tales como PL/1 y ALGOL, para lo cual se da una relación de puntos característicos del Pascal:

a) La declaración de variables es obligatoria.

b) Hay palabras reservadas que no pueden emplearse como identificadores (algunos ejemplos de palabras reservadas son: BEGIN, END, IF, TO, DOWNTO, etc.).

c) Es un lenguaje de formato libre: una instrucción puede ponerse en cualquier parte de la línea y separarse de la instrucción siguiente con un punto y coma.

d) Los tipos básicos son: entero, real, booleano y caracter (INTEGER, REAL, BOOLEAN Y CHAR). Otras estructuras son: arreglos (ARRAY), registros (RECORD) -que son las estructuras de información de PL/1 y COBOL)- los conjuntos y archivos (SETS Y FILES). Todas estas estructuras se pueden combinar entre sí.

e) Los arreglos (ARRAY) tienen un tamaño fijo, decidido al compilar el programa. Se han simplificado los arreglos dinámicos de PL/1 y ALGOL.

f) La estructura básica de un programa Pascal con su BEGIN, END y el punto final, es similar a la de ALGOL.

g) Además de la instrucción condicional (IF-THEN-ELSE), existe la instrucción CASE, como en el caso de ALGOL.

h) La instrucción de iteración FOR es más sencilla que en ALGOL, sólo admite incrementos unitarios positivos o negativos.

i) Los parámetros de las funciones y subrutinas (PROCEDURES) pueden ser del tipo referencia (nombre) o valor.

j) Hay algunas otras características del ALGOL que se han suprimido del Pascal, como son:

- Las instrucciones de asignación múltiple.

- Las variables tipo OWN.

Estas y algunas otras características del ALGOL hacen bastante más complejo el problema de su compilación.

k) La estructura de bloques con su mecanismo de variables locales y globales es una característica común de ALGOL y PL/1 con la que también cuenta Pascal.

l) La transferencia de control se realiza con la instrucción GOTO. Las etiquetas Pascal son enteros sin signo que hay que declarar.

Como se habrá observado, las características del Pascal están lejos de ser nuevas (sería difícil que lo fueran), pero lo importante es su elección cuidadosa pensando fundamentalmente en:

- El usuario, dándole un lenguaje estructurado, poderoso y de baja complejidad.

- La compilación del lenguaje, eliminando algunos aspectos de ventaja dudosa para el usuario y que hacen que la compilación del lenguaje sea más compleja.

CAPITULO 3

ESTRUCTURA DE UN PROGRAMA EN PASCAL.

Un programa es simplemente una secuencia de instrucciones, como lo son una receta de cocina, una partitura musical, un instructivo de tejido, etc. Los programas en este sentido existen mucho antes de que fueran inventadas las computadoras; los programas para computadora pueden ser mucho más grandes y más complejos que los otros tipos de programas, por lo tanto, escribir un programa para computadora requiere mucho cuidado y gran precisión

La computadora ejecuta las instrucciones secuencialmente (esto es, ejecuta las instrucciones de una en una y consecutivamente) a menos que se le indique otra cosa.

Todos los programas en Pascal contienen dos tipos diferentes de símbolos que son:

1. Los símbolos que pertenecen al lenguaje, ésto es, que tienen un significado especial y definido, como pueden ser un solo caracter o parejas de caracteres tales como:

   + - / * : , ; ( ) :=

   o demás caracteres, conocidos como palabras reservadas, las cuales solo pueden ser utilizadas como lo indican las reglas del lenguaje; a continuación se da un listado de todas aquellas palabras que son consideradas como palabras reservadas en Pascal.

AND	ARRAY	BEGIN
CASE	DIV	DO
DOWNTO	ELSE	END
FILE	FOR	FUNCTION
GOTO	IF	IN
LABEL	MOD	NIL
NOT	OF	OR
PACKED	PROCEDURE	PROGRAM
RECORD	REPEAT	SET
THEN	TO	TYPE
UNTIL	VAR	WHILE
WITH		

2. Los demás símbolos no pertenecen al lenguaje y son conocidos como "identificadores", que están compuestos por letras y dígitos, el primer caracter debe ser siempre una letra, ejemplos:

carlos	identificador	pi	mayo1983
dato	cliente	saldo	numerodefactura

Todos estos son identificadores válidos y están impresos con letras minúsculas para diferenciarlos de las palabras reservadas.

A continuación se dan unos ejemplos de identificadores no válidos en Pascal explicando el motivo por el que son inválidos:

pi3.1416 - Tiene un punto, que no es considerado como letra ni como número.

17mayo1983 - Comienza con número y siempre debe de empezar con letra.

1dato - Misma razón que el anterior

num de fact - Contiene espacios en blanco que tampoco son considerados como letras.

3.1 DIAGRAMAS DE SINTAXIS.

Las reglas para la construcción de los identificadores, se pueden observar claramente en el siguiente diagrama de sintaxis:

```
                          ------
identificador - -------> | LETRA |  ------------------------------>
                          ------   |                              |
                                   |    ----------                |
                                   |   | LETRA  |<--|
                                   |    ----------                |
                                   |    ----------                |
                                   |<--| DIGITO |<--|
                                        ----------
```

en donde para construir correctamente un identificador solamente es necesario seguir las flechas. Los marcos rectangulares significan que se está haciendo referencia a otro diagrama de sintaxis, por lo tanto en LETRA hay que buscar el diagrama de sintaxis que se llame letra y que es:

```
letra -- ---------> A  ---------------------------->
         |-------> B     ---|
         |-------> C  -------|
         |                   |
         .                   .
         .                   .
         .                   .
         |                   |
          ------> Z  ------
```

```
digito ------------> 0  ---------------------------->
         |-------> 1  -------|
         |-------> 2  -------|
         |                   |
         .                   .
         .                   .
         .                   .
         |                   |
          ------> 9  -------
```

Estos diagramas de sintaxis indican qué caracteres son reconocidos como letra o como dígito. En los diagramas de sintaxis que se incluyen en los apéndices, los círculos y los marcos con extremos redondeados indican que deben ser escritos en el programa exactamente como aparecen en el marco o círculo. Siguiendo estos diagramas vemos que para formar un identificador lo primero que nos encontramos es una letra, después podemos seguir con otra letra o con un dígito y así sucesivamente hasta que deseemos salir del diagrama.

3.2 DECLARACION DE UN PROGRAMA EN PASCAL.

Esto es conocido como una estructura de bloque. El diagrama de sintaxis de un diagrama de Pascal se representa de la siguiente manera:

```
                 -----------   -----------------      -----------------
programa --->| PROGRAM |-->| identificador |-->(-->| identificador |-->  ) --
                 -----------   -----------------      -----------------
                                                   |                  |
                                                    -------  ,  <--------

-----------------------------------------------------------------------------
|
|                                  -----------
---------------->  ;  ---------->| bloque |      -->  .  ------------->
                                  -----------
```

CAPITULO 4

TIPOS DE DATOS

4.1 IDENTIFICADORES

Están compuestos por letras y dígitos y el primer caracter debe ser una letra. Los siguientes son ejemplos de identificadores válidos:

CASA PRESION1 PRESION2

La longitud máxima de un identificador es de 8 caracteres (entre letras y números).

4.2 CONSTANTES.

Es frecuente que algunos valores utilizados en un programa sean conocidos previamente (ej. PI = 3.1415926535, E = 2.7182818). A éstos valores se les llama constantes y su nombre y valor se proporciona en la sección de constantes, dentro de las declaraciones, por ejemplo:

```
CONST
        TAMHOJA = 45;
        PI = 3.141592;
```

también es válido declarar una constante en función de otra ya declarada.

```
CONST
        ALFA = 7512.5;
        BETA = -ALFA;
```

4.3 ALMACENAMIENTO DE DATOS

Los datos, dentro de un programa, pueden estar contenidos en constantes o variables. La diferencia entre ambos es que los segundos pueden cambiar de valor a lo largo de la ejecución del programa, mientras que los primeros no.

En Pascal hay cuatro TIPOS PREDEFINIDOS de datos:

INTEGER, REAL, BOOLEAN y CHAR.

Cada variable que se utilice en un programa, habrá que declararla previamente, por ejemplo:

```
VAR
        X, INDICE, I, J, K : INTEGER;
        GAMA, PRESION      : REAL ;
        INDICADOR          : BOOLEAN;
        MARCA, MARCAFIN    : CHAR;
```

4.3.1 TIPO INTEGER

Los enteros son considerados en Pascal en la forma usual, es decir números naturales positivos y negativos como 534, 1984, -12, etc., es decir con parte fraccionaria nula. El cero es un entero.

4.3.2 TIPO REAL

Son cantidades con parte fraccionaria y su representación interna es diferente a la de las cantidades enteras: ej.: 53.4, 1984.0, -12.25, etc.

4.3.3 TIPO BOOLEAN

Las variables booleanas solo pueden tener dos valores: falso ó verdadero (FALSE o TRUE) y pueden ser, por ejemplo, el resultado de una comparación: Ej.: A<B sólo puede ser falso, ó verdadero, dependiendo de los valores de A y B.

4.3.4 TIPO CHAR

El valor de una variable CHAR, es un caracter (letra, digito, caracter especial, etc.), Ej: A, Z, 1, *, etc.

4.4 OPERADORES ARITMETICOS

Es posible combinar variables y constantes, mediante operadores.

Existen operadores para cada uno de los tipos estándar:

ENTEROS	+	-	*	DIV	MOD
REALES	+	-	*	/	
BOOLEAN	AND	OR	NOT		

4.5 ASIGNACION

Para dar o cambiar el valor a las diferentes variables de un programa, se utiliza el símbolo compuesto ':=', llamado símbolo de asignación. Así si deseamos que una variable previamente declarada, llamada A, tenga el valor de 5.7, indicaremos:

A:=5.7

del lado izquierdo del símbolo de asignación SIEMPRE aparecerá el nombre de una variable; del lado derecho aparecerá una 'expresión aritmética'.

4.6 EXPRESION ARITMETICA

Una expresión aritmética es un conjunto de constantes y/o variables agrupadas (si es necesario) por operadores aritméticos. Por ejemplo

5.7 + 8.256

es una expresión aritmética que combina dos constantes con un operador.

Supongamos que las variables A, B, C, D y E ya han sido declaradas y son de tipo entero:

```
A:=3;
B:=2;
C:=4;
D:=A+B+C+7;
E:=C-(2*A);
A:=A+1;
```

después de la ejecución de estas seis instrucciones, las variables contendrán los siguientes valores:

```
A : 4
B : 2
C : 4
D : 16
E : -2
```

CAPITULO 5

INSTRUCCIONES DE ENTRADA Y SALIDA

5.1 INTRODUCCION

Una de las mayores ventajas de los programas es el poder operar con datos distintos en cada ejecución de los mismos. Mediante las instrucciones de entrada y salida un programa deja de ser inerte, para convertirse en un medio de comunicación con el exterior. La entrada reune todos los datos que se le suministran a un programa y la salida se refiere a todos los resultados que suministra el programa. La estructura más sencilla de un programa es la que se muestra en la figura 4.1

Fig. 4.1 Entrada y Salida de un Programa

En esta figura no se pretende reflejar ninguna relación temporal: un programa puede leer muchas veces o no, al principio o al final de su ejecución.

Por otra parte, tanto los números como otros datos, requieren una representación interna en la computadora, y esta representación interna es diferente de la representación externa. En particular, la representación binaria se emplea para representar números internamente, puesto que, en una computadora, ésta es más eficiente para el cálculo aritmético, que la representación decimal. En consecuencia, cuando un número o letra se lee, debe convertirse de caracteres a una representación interna (binaria) y, cuando un caracter se escribe la

conversión es inversa, es decir, se pasa de su representación interna (binaria) a caracteres (representación externa).

Estas operaciones de conversión son ejecutadas automáticamente, en Pascal, por las instrucciones READ, READLN, WRITE y WRITELN que serán tratadas en el presente capítulo, para que la conversión mencionada sea transparente al usuario.

Ejemplo:

Supongamos que tenemos declaradas las variables:

```
largo, ancho, alto, area:  INTEGER
```

Ahora, escribimos unas instrucciones para calcular el área de un paralelepípedo determinado:

```
largo := 10;
ancho := 5;
alto := 6;
area := 2*(largo * ancho + largo * alto + ancho * alto);
```

La utilidad de este fragmento de programa es muy pequeña, dada su inflexibilidad, que se deriva de las tres instrucciones de asignación fijas. Si queremos que este programa sirva para calcular áreas de diversos paralelepípedos, tendremos que cambiar las tres instrucciones de asignación que preceden a la fórmula de cálculo del área; ésto se puede hacer con una instrucción de lectura. También interesa que los resultados de un programa sean conocidos por nosotros; la forma de conseguirlo es con una instrucción de impresión a la salida.

5.2 READ

Mediante la instrucción READ, el lenguaje Pascal permite la lectura de un dato y su asignación a una variable del programa.

Ejemplo (continuación)

Para darle flexibilidad a nuestro programa del ejemplo anterior, sustituiremos las tres instrucciones de asignación por las tres instrucciones de lectura (READ) siguientes:

```
READ(largo);
READ(ancho);
READ(alto);
area :=2* (largo*ancho+largo*alto+ancho*alto)
```

Si utilizamos como entrada los datos siguientes:

```
10 5 3
```

Equivale a ejecutar las asignaciones:

```
largo := 10;
ancho := 5;
alto := 3;
```

con la posibilidad de cambiarse, modificando sólo los datos de lectura.

La instrucción READ(Variable entera) comienza la exploración de caracteres de entrada, ignorando los blancos; el siguiente número entero que se encuentre, lo asignará a la Variable entera.

Se pueden leer varias variables con una instrucción READ, colocándolas en forma de lista entre paréntesis. Entonces, los datos de entrada respectivos deben de ir separados, entre sí, por uno o más blancos. En el ejemplo anterior se pueden fundir los tres READ'S quedando:

```
READ (largo,ancho,alto)
```

Teniendo el mismo efecto antes explicado.

Asimismo, la instrucción READ puede leer datos tipo INTEGER, REAL o CHAR. Para datos de tipo INTEGER y REAL ignorará los blancos y marcará un error si encuentra otro caracter que no sea blanco o dígito.

5.3 WRITE

El papel de una instrucción de salida es obtener resultados del programa en forma legible, bien sobre papel de impresora, pantalla o cualquier otro dispositivo de salida. En Pascal, este cometido se realiza mediante la instrucción WRITE. Así, mediante la instrucción WRITE, el valor de una variable o expresión se imprimirá o representará en la salida estándar de la computadora

Ejemplo (continuación):

Ahora hacemos que se imprima el resultado del cálculo del área del paralelepípedo con la instrucción:

```
WRITE(area)
```

el segmento de programa quedará:

```
READ(largo, ancho, alto);
area :=2*(largo*ancho+largo*alto+ancho*alto);
WRITE(area);
```

Si deseáramos escribir también los datos leídos, es decir, el largo,ancho y alto, la instrucción quedaría:

```
WRITE(largo,ancho,alto,area)
```

y los valores de cada una de las variables saldrían impresos sobre una línea.

Como podemos apreciar, al igual que en el caso de la instrucción READ, la instrucción WRITE puede tener cualquier número de parámetros. Sin embargo, cada parámetro de una instrucción WRITE puede ser una "expresión" y no necesariamente una variable simple, y el valor de esta expresión, una vez evaluada, saldrá escrita en la salida estándar de la computadora.

Ejemplo:

Si suponemos que 'm' y 'n' son variables enteras con valores asignados, la instrucción:

```
WRITE (m,n,m+n,m-n,m*n);
```

Imprimirá los valores de m,n, la suma, la diferencia y el producto respectivamente.

La forma de salida hasta aquí expuesta, nos permite únicamente escribir los valores numéricos de las variables. Si deseáramos que cualquiera de los valores impresos estuviera acompañado de un texto, es necesario que dicho texto, encerrado entre apóstrofes, aparezca como parámetro en la instrucción WRITE, o bien, el texto debe ser declarado como una constante de tipo caracter. En la salida se reproduce dicho texto sin apóstrofes.

Ejemplo:

La instrucción:

```
WRITE ('el area es = ',area);
```

producirá la siguiente salida:

```
el area es =        50475
```

Un resultado equivalente se obtendría mediante la siguiente declaración:

```
CONST
     texto = 'el area es = ';
     ------------------------
     ------------------------
     ------------------------
     WRITE(texto, area);
```

Hemos supuesto que la variable área tiene el valor 50475. Si el texto a escribir tiene algún caracter apóstrofe, éste deberá escribirse duplicado.

En el ejemplo anterior se supone que la salida estándar de cualquier número entero ocupa doce posiciones. Esta es la razón por la que después del signo igual (=) aparecen varios blancos o espacios (siete).

Si conociéramos a priori que el valor de área no excede de cinco cifras, la instrucción anterior podría escribirse:

```
WRITE ('el area es = ',area:5)
```

El sufijo '5' de la variable 'area:5' especifica que el valor neto se escribirá en un campo de 5 espacios. La salida sería ahora:

```
el area es = 50475
```

Ejemplo:

La siguiente instrucción:

```
WRITE ('largo = ',largo:5, 'ancho = ',ancho:5,' area =',area:6);
```

producirá la salida:

```
largo =xxxxx ancho =xxxxx area =xxxxxx
```

5.4 READLN

Podemos considerar que la entrada de datos consiste en varias líneas (si la entrada de datos fuera sobre tarjeta perforada, entonces cada tarjeta de datos sería una línea).

En el caso de utilizar la instrucción READ, se pasará automáticamente a una nueva línea de entrada de datos, si no hay más datos en la línea actual.

Para algunos propósitos se desea forzar a que la entrada de datos sea una línea cada vez. La instrucción que se utilizará para este caso es READLN.

Después de que una instrucción READLN haya sido ejecutada, la parte no leída de la línea actual de entrada de datos es saltada, u omitida, y la siguiente entrada será tomada de la siguiente línea.

Ejemplo:

Consideremos las instrucciones:

```
READLN (nmr);
READLN (par);
```

Donde nmr y par son variables enteras.

Si la entrada de datos es:

```
50 num1
8 num2
```

entonces los valores 50 y 8 respectivamente serán asignados a las dos variables; los textos de ambas líneas son ignorados.

Si las instrucciones READLN fueran reemplazadas por las instrucciones READ, la segunda instrucción READ hubiera tomado como valor entero el texto num1, produciéndose un error.

En la entrada de datos, se leen éstos, siempre de izquierda a derecha, dentro de cada línea, y desde una línea.

Como resumen de lo expuesto hasta ahora, vamos a describir las distintas formas generales de las instrucciones de entrada. La instrucción de entrada tiene las cuatro formas siguientes:

```
READ (input,V1,V2,...,Vn);
READ (V1,V2,...,Vn);
READLN (input,V1,V2,...,Vn);
READLN (V1,V2,...,Vn);
```

donde las Vi representan las variables de entrada, e 'input' el archivo o fuente de los datos de entrada.

5.5 WRITELN

Se habrá observado hasta aquí que los valores numéricos de salida se escriben sobre una única línea. Si deseáramos escribir sobre varias líneas, utilizaremos WRITELN en lugar de WRITE. WRITELN es idéntico a la instrucción WRITE, pero después de imprimir el valor de sus parámetros, escribe un carriage-return al archivo de salida.

Utilizando nuestro ejemplo de área de un paralelepípedo, si escribiéramos las siguientes instrucciones:

```
WRITELN ('largo=',largo:5);
WRITELN ('ancho=',ancho:5);
WRITELN ('ancho=',alto:5);
WRITELN ('area=',area:6);
```

la salida que producirá es:

```
largo=xxxxx
ancho=xxxxx
alto=xxxxx
area=xxxxxx
```

La utilización de un WRITELN sin parámetros, producirá una línea en blanco.

Resumiendo, la instrucción de salida tiene las cuatro formas siguientes:

Así pues, si tuviéramos las siguientes asignaciones:

```
r:=3
h:=4
PI:=3.1416
```

siendo r,h,PI variables reales, el resultado de ejecutar la instrucción WRITE (r,h,PI) sería:

```
3.00000E+00 4.00000E+00 3.1416E+00
```

Esta salida, según hemos visto, podríamos controlarla si escribiéramos, por ejemplo:

```
WRITELN (r:14,h:20,PI:20)
```

colocándose entonces los resultados en campos cuya longitud sería, respectivamente, de 14,20 y 20 posiciones. Podemos incluso, como ya hemos visto, obtener una salida en notación decimal (convencional) especificando, el número de decimales que deseamos obtener después del punto decimal, y la longitud del punto. Así:

```
WRITELN (r:8:1, h:12:1, PI:12:4)
```

dará una salida:

```
3.0          4.0       3.1416
```

Se recomienda ser consistente en los formatos de salida, a menos de que se tenga una buena razón para no hacerlo. Los valores del formato pueden ser definidos en la sección de declaración de constantes, por ejemplo:

```
CONST
     precision = 6;
     campo     =16;
VAR
   resultado : real;
              .
              .
              .
WRITELN (resultado: precision: campo);
```

Si ésto se hace, es muy simple adaptar el programa para otra computadora u otra necesidad.

Otra propiedad importante de las instrucciones WRITE y WRITELN es la siguiente:

Supongamos que tenemos la siguiente instrucción: WRITE(número:campo), si la variable 'número' es de tipo entero y contiene más caracteres que los especificados por campo, esta variable 'número' se imprimirá en un campo más amplio. Además, hay que tener en cuenta que la instrucción WRITE justifica el valor a la derecha.

Asimismo, las instrucciones READ, READLN, WRITE y WRITELN pueden utilizarse para entrada y salida de caracteres, siempre y cuando se tensan presentes varios aspectos:

Si queremos asignar valores numéricos y no numéricos (caracteres) a variables numéricas (tipo REAL o INTEGER) y a variables tipo CHAR respectivamente, a través de una instrucción READ, ésto es: leer datos mezclados (números y caracteres), debemos tener en cuenta las siguientes reglas:

a) Para variables numéricas, no se tienen en cuenta los blancos a la izquierda; cuando se encuentra el primer valor numérico, continúa con el siguiente si es numérico y así sucesivamente, hasta encontrar un blanco o una letra.

b) Para variables tipo CHAR, todos los caracteres son válidos, así, el caracter blanco (espacio en blanco), que se ha utilizado para separar valores numéricos en la lectura, para el caso de variables tipo CHAR, y sobre todo para cadenas de caracteres (PACKED ARRAY), es un caracter que se trata igual que los demás.

Supongamos, por ejemplo, que el valor que deseamos leer es:

7m45t

mediante la instrucción:

READ (X,S1,Y,S2)

siendo X e Y variables enteras (INTEGER) y S1,S2 variables tipo caracter (CHAR). Después de la lectura, se tendrá:

X=7 ; S1=m ; Y=45 ; S2=t

Si el dato hubiera sido:

543p 79 t

el resultado de la lectura sería:

X=543 ; S1=p ; Y=79 ; S2=blanco

La instrucción WRITE (ch:campo), en la cual, campo es una expresión entera y ch es una variable de tipo CHAR, escribirá campo-1 blancos y después el caracter ch.

Un uso particular de la instrucción WRITE es:

WRITE (' ':campo)

escribirá: 'campo' blancos (tantos blancos como los que especifique campo)

También pueden escribirse valores booleanos mediante las instrucciones WRITE y WRITELN. Por ejemplo, la instrucción:

```
WRITE (bandera)
```

donde la variable bandera es de tipo booleano (BOOLEAN), imprimirá: TRUE o FALSE dependiendo del valor de la variable bandera.

Por otro lado, es importante hacer notar que las instrucciones READ o READLN no aceptan argumentos booleanos.

5.7 EOLN Y EOF.

En la mayoría de las aplicaciones, los archivos requieren cierta clase de subestructura. Por ejemplo, un libro, aunque puede considerarse como una secuencia única de caracteres, se subdivide en capítulos y párrafos. El objeto de esta subestructura es proporcionar algunos puntos explícitos de referencia, algunas coordenadas para facilitar la orientación en la larga secuencia de información.

Los archivos cuyos componentes son de tipo CHAR juegan un papel especialmente importante en el cálculo y proceso de datos: constituyen el elemento de contacto entre las computadoras y los usuarios humanos. Tanto la "entrada" legible proporcionada por los programadores, como la "salida" legible que representa los resultados calculados, están constituidos por secuencias de caracteres.

La comunicación entre un proceso en el computador y su autor humano se establece, finalmente, mediante un elemento puente (interfaz) que puede representarse por dos archivos de textos. Uno de ellos contiene la "entrada (input)" al proceso, y el otro los resultados procesados llamados "salida (output)".

El archivo "input", es pues, sólo para lectura y el archivo "output" sólo para escritura.

Así pues, los archivos de texto en el mundo real, tal como los programas y datos, no son meras cadenas de caracteres, sino que están estructurados de varias maneras, convencionalmente en líneas y páginas. Hay procedimientos estándar y funciones en Pascal que permiten a los programas generar archivos con esta estructura, y también reconocer tal estructura en el archivo de texto de entrada (input).

```
WHILE NOT EOLN(INPUT) DO
     GET(INPUT);
(*ENDWHILE*)
GET(INPUT);
```

considerando que la instrucción GET toma un caracter de la línea actual.

De esta manera, la siguiente llamada a READ obtendrá el primer caracter de la siguiente línea, a menos que el final del archivo haya sido alcanzado.

Pascal también proporciona una función estándar de tipo booleano llamada "EOF" (End of File), la cual tomará el valor de verdadero cuando el proceso haya alcanzado el fin del archivo.

Cualquier intento de leer cualquier cosa del archivo después que EOF ha llegado a ser verdadero, causará un error de ejecución, de esta manera, si se está leyendo iterativamente, debemos verificar si no hemos llegado al final del archivo.

Ejemplo:

```
READ(INPUT,DATO);
WHILE NOT EOF DO
      BEGIN
      READLN(INPUT);
      READ(INPUT,DATO);
      END
(*ENDWHILE*)
```

En este fragmento de programa leeremos iterativamente DATO hasta que "cerremos" el archivo de entrada, con lo cual terminará nuestro proceso de lectura.

5.8 MISCELANEAS SOBRE LAS INSTRUCCIONES DE ENTRADA Y SALIDA .

Los tipos definidos por el usuario, como se verá más adelante, son de uso exclusivamente interno. Los valores de dichos tipos pueden ser asignados, utilizados y probados dentro del programa, pero estos valores no pueden representarse por medio de las instrucciones WRITE o WRITELN. Igualmente, tampoco pueden ser asignados estos valores desde un medio externo por medio de las instrucciones READ o READLN. Así pues, no es correcto utilizar una variable declarada como TYPE en instrucciones de escritura y lectura. Asimismo, no es posible en Pascal estándar leer o

escribir valores escalares directamente, pero podemos usar:

```
WRITE (ORD(escalar));
```

La cual escribirá el ORD (función ordinal) del escalar en cuestión.

Respecto a las variables tipo PACKED ARRAY OF CHAR cabe hacer notar lo siguiente:

Supongamos que tenemos la siguiente declaración:

```
TYPE
    STRING4=PACKED ARRAY[1..4] OF CHAR;
VAR
    STR : STRING4;
```

Mediante la instrucción de asignación podemos escribir:

```
STR:='ABCD';
```

Es decir, asignamos los cuatro caracteres a la variable STR; esta operación no se puede realizar mediante la instrucción READ(STR), y únicamente será posible realizar la asignación mediante la siguiente iteración:

```
FOR I:=1 TO 4 DO
    READ(STR[I])
(*ENDFOR*);
```

Y leer de la entrada ABCD.

La salida de valores tipo PACKED ARRAY OF CHAR (strings) es relativamente fácil, ya que las variables pueden incluirse en la lista de parámetros de las instrucciones WRITE o WRITELN, y la salida no se ajusta a un tamaño fijo, en lugar de ésto, el tamaño del campo es igual al número de caracteres a escribir. Esto es, una variable tipo CHAR se imprimirá en columna y en el caso de un string, el tamaño del campo es igual a la longitud del string. Se hace notar que un string puede aparecer en una instrucción WRITE o WRITELN y que su valor será impreso sin tener que utilizar una iteración.

5.9 CONSIDERACIONES FINALES.

Según todo lo que hemos visto a lo largo del presente capítulo, no es difícil escribir programas interactivos en Pascal, siempre y cuando se sigan algunas reglas simples. La dificultad reside en el hecho de que la instrucción READLN no lee al final de la línea actual, sino que lee el primer caracter de la línea siguiente. Consideremos el siguiente fragmento de programa:

```
WRITELN ('Dame el primer numero ');
READLN (Primero);
WRITELN ('Dame el segundo numero ');
READLN (Segundo);
```

Este programa imprimirá 'Dame el primer numero' y leerá el valor de Primero. La instrucción READLN no regresa hasta que el primer caracter de la siguiente línea haya sido leído, y así, el programa preguntará por Segundo, antes de que haya impreso 'Dame el segundo numero'. Desafortunadamente, este problema ha sido tratado de diferente manera en diferentes implementaciones. Daremos algunas sugerencias que puedan ser útiles:

Póngase un READLN inmediatamente antes de leer una línea de texto (otra que no sea la primer línea) de una terminal, y procure no usar READLN con parámetros. Asegúrese de procesar una línea completa de entrada antes de enviar cualquier mensaje a la salida.

Tómese en cuenta que la instrucción WRITELN es usada para terminar la línea actual de salida y empezar una nueva.

CAPITULO 6

DECISION E ITERACION

6.1 OPERADORES LOGICOS

Existen los siguientes operadores lógicos o de relación, aplicables a los tipos INTEGER, REAL, BOOLEAN y CHAR:

=	igual	A = B	A igual a B
<>	diferente	A <> B	A diferente de B
<	menor	A < B	A menor que B
<=	menor o igual	A <= B	A menor o igual a B
>=	mayor o igual	A >= B	A mayor o igual a B
>	mayor	A > B	A mayor que B

El resultado de efectuar operaciones lógicas será siempre FALSO O VERDADERO.

Por ejemplo, si A:=5 y B:=7 :

```
A = B  ---------- falso
A < B  ---------- verdadero
A <> B ---------- verdadero
A <= B ---------- verdadero
A >= B ---------- falso
```

Es posible combinar valores lógicos, mediante operadores lógicos:

A OR B	Verdadero si A o B son verdaderos.
A AND B	Verdadero si A y B son verdaderos.
NOT A	Verdadero si A es falso, Falso si A es verdadero.

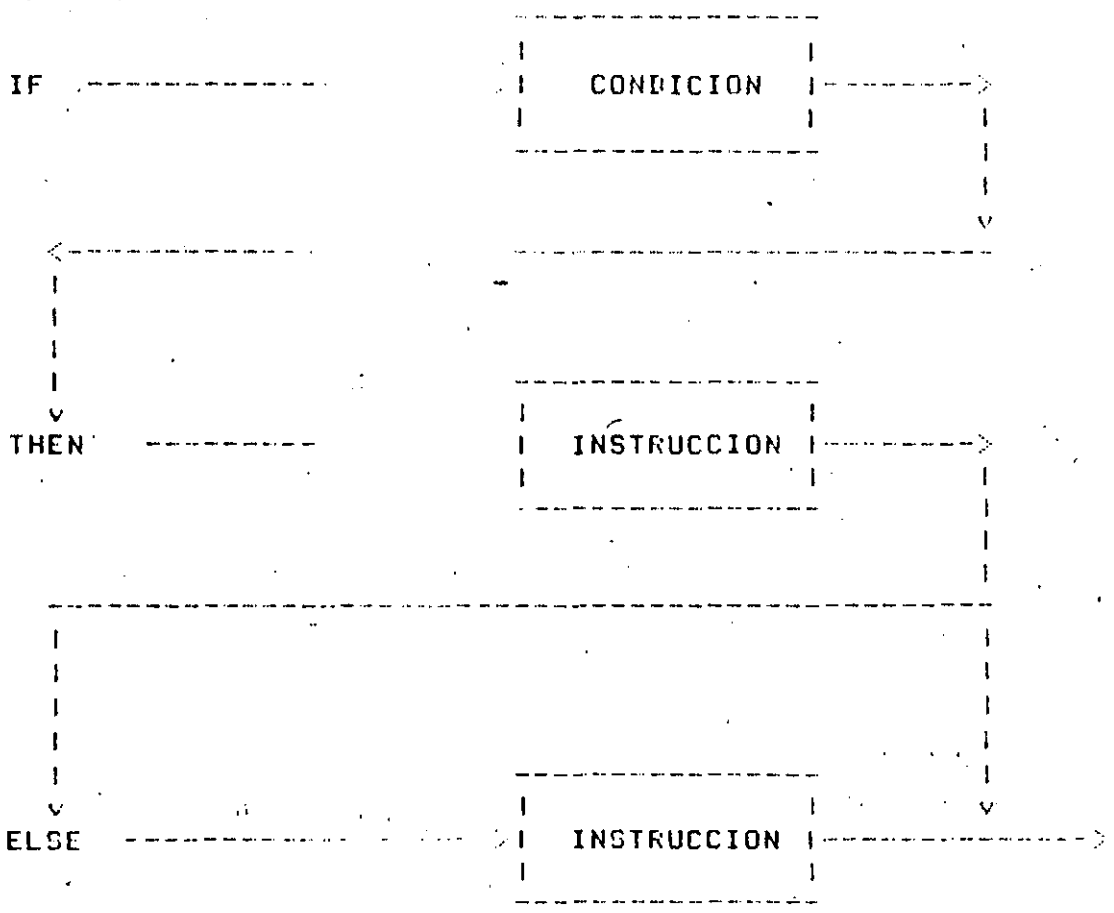
Ejemplos: Para A:=5, B:=7, C:=1 y D:=0

(A < B) OR (C = D)	Verdadero, ya que A es menor que B, aunque C no es igual a D.
(A < B) AND (C = D)	Falso, ya que A es menor que B, pero C no es igual a D.
NOT (A < B)	Falso, ya que A es menor que B.

6.2 INSTRUCCION IF

Esta instrucción nos permite seleccionar de entre dos alternativas, resultantes de una operación lógica.

DIAGRAMA DE SINTAXIS.



Ejemplo

```
IF A>B THEN
    A:=0
ELSE
    B:=1
(*ENDIF*);
```

En esta instrucción, si el valor de A es mayor al de B, entonces se efectuará la asignación:

A:=0

de lo contrario (ELSE), se efectuará la asignación:

```
N:=1
```

Es válido omitir la parte 'ELSE' de la instrucción IF, como se muestra en el siguiente ejemplo:

```
IF A=0  THEN
    X:=X + 1
(*ENDIF*);
```

Como hemos visto, si el valor de 'condición' es verdadero la instrucción que sigue al THEN es ejecutada y si el valor de 'condición' es falso, la instrucción que sigue al 'ELSE' (si existe) es ejecutada.

Las instrucciones que siguen a THEN y ELSE pueden ser compuestas, de tal forma que lo siguiente sería válido:

```
IF condición  THEN
   BEGIN
   instrucciones
   END
ELSE
   BEGIN
   instrucciones
   END
(*ENDIF*);
```

6.3 INSTRUCCION REPEAT

La instrucción REPEAT nos permite ejecutar una o más instrucciones hasta que cierta condición se cumpla. La forma general de esta instrucción es:

```
REPEAT
    instrucciones
UNTIL condición;
```

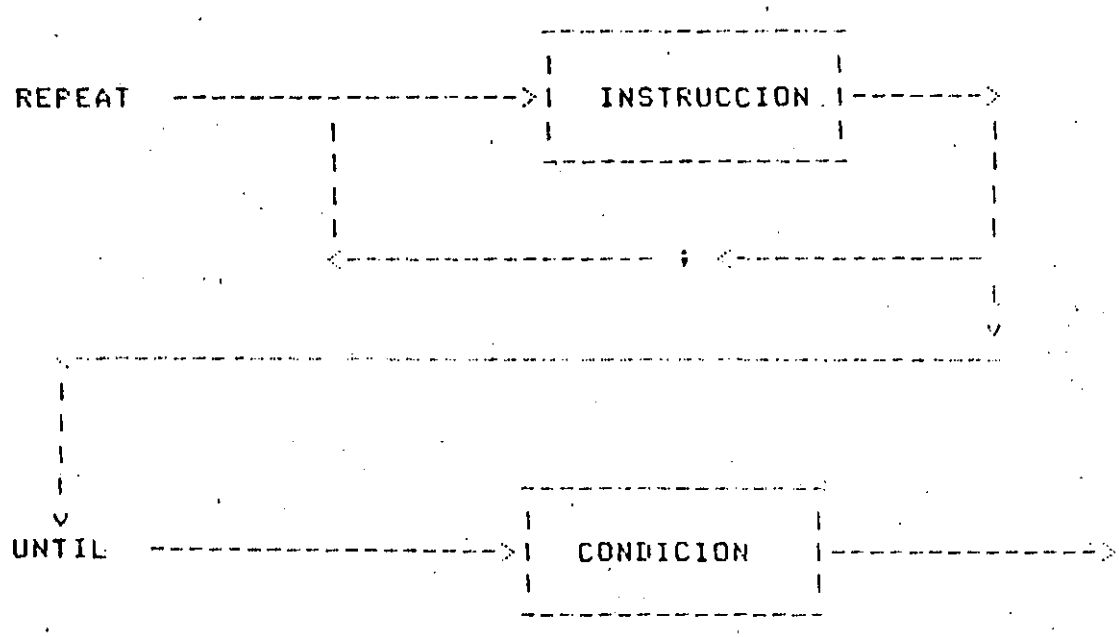
Esta instrucción resulta particularmente útil cuando es conocido que para obtener cierto resultado tenemos que ejecutar una o varias instrucciones al menos una vez, por ejemplo, suponsamos que deseamos conocer cuantas veces hay que sumarle a la cantidad 76.3012, el número 0.0037, para que el primero sea igual o mayor a 77, entonces,

```
CONT:=0;
A:=76.3012;
B:=0.0037;
REPEAT
   A:=A+B;
   CONT:=CONT+1;
UNTIL A>=77;
```

Este fragmento de programa entregaría en la variable CONT (CONTADOR), el número buscado.

Habrá que tener siempre precaución de que al menos una de las instrucciones dentro del REPEAT (es decir después de la palabra REPEAT y antes de la palabra UNTIL), hagan que la condición especificada se cumpla, de lo contrario obtendríamos una iteración infinita.

Se presenta a continuación el diagrama de sintaxis de ésta instrucción:



6.4 INSTRUCCION WHILE

Otra forma de ejecutar repetidamente una instrucción es mediante la instrucción WHILE, cuyo funcionamiento es simlar a REPEAT, pero la condición que detiene la iteración es evaluada antes de ejecutar la instrucción. La forma general es la siguiente.

```
WHILE condición DO
    instrucción
(*ENDWHILE*);
```

Por ejemplo:

```
A:=1;
B:=7.36;
WHILE B > A DO
    BEGIN
    B:=B-1.236;
    A:=A+.003;
    END
(*ENDWHILE*);
```

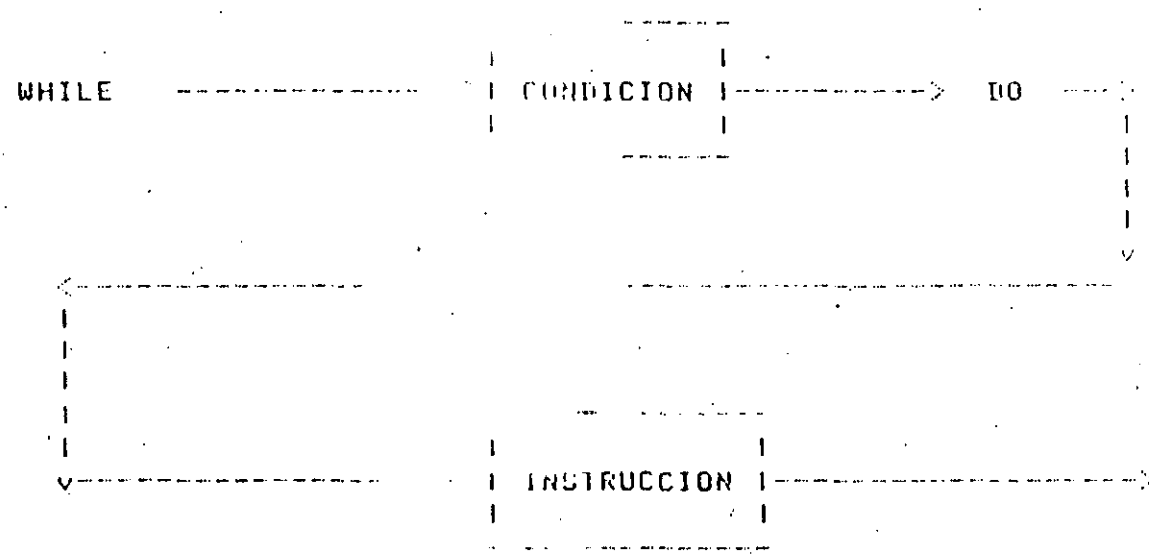
En este ejemplo, mientras B sea mayor a A, las dos asignaciones de la instrucción compuesta que siguen al WHILE serán ejecutadas.

Es importante resaltar los siguientes puntos:

- Al igual que REPEAT, dentro de la iteración será necesario hacer que en algún momento la iteración se detenga (para evitar un iteración infinita).

- Como se mencionó, la condición es evaluada antes, así que puede darse el caso de que la instrucción dentro del WHILE nunca se ejecute, a diferencia de REPEAT en donde al menos una vez se pasa por la o las instrucciones dentro de éste.

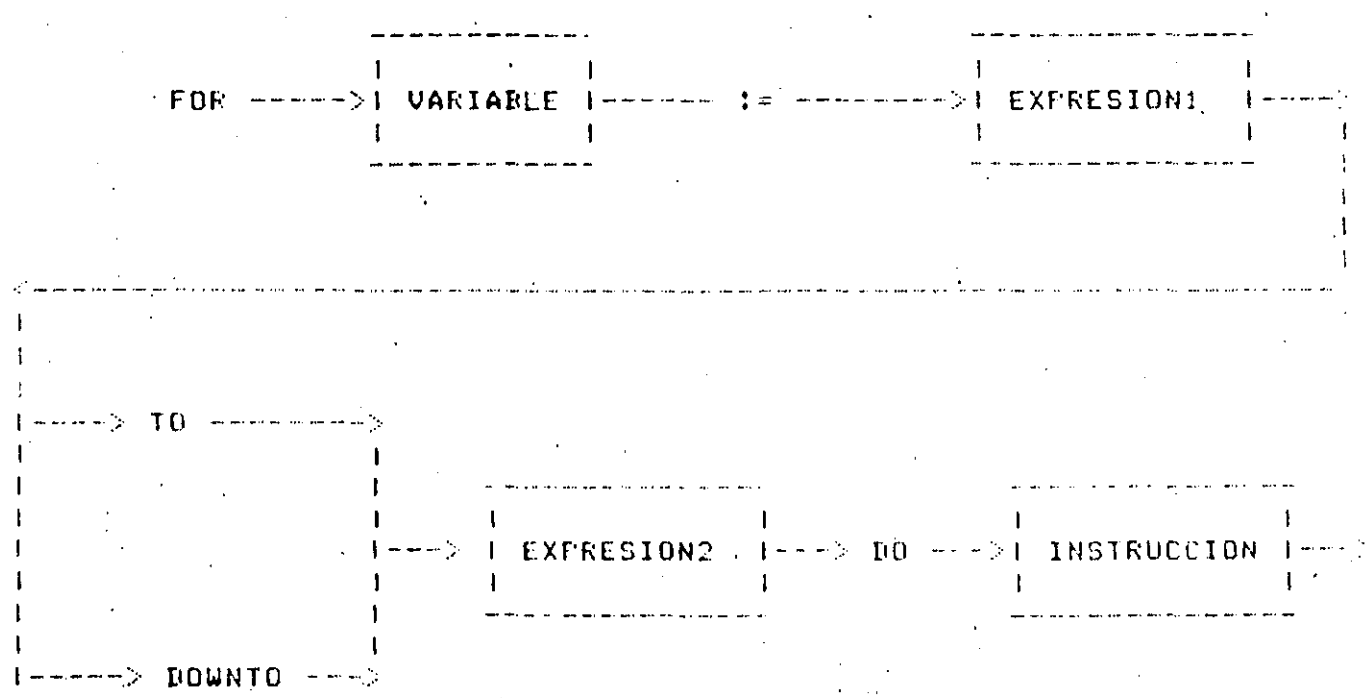
El diagrama de sintaxis correspondiente es el siguiente:



6.5 INSTRUCCION FOR

En muchas ocasiones es necesario repetir la ejecución de una o más instrucciones y de ninguna de éstas depende el número de veces que hay que hacerlo (por ejemplo, si es conocido que tenemos que leer exactamente diez datos, habrá que ejecutar diez veces una instrucción de entrada). Se cuenta en Pascal con la instrucción FOR, que permite hacerlo fácilmente.

El diagrama de sintáxis es el siguiente:



Esta instrucción funciona de la siguiente forma:

- Se efectúa la asignación

variable:= expresión1

-Si la expresión1 es mayor a la expresión2, la instrucción no se ejecuta, y el flujo del programa continúa a la siguiente instrucción después del FOR.

-Si la expresión1 es menor o igual a la expresión2, la instrucción se ejecuta y después de hacerlo, el valor de la expresión1 será:

expresión1 := SUCC (expresión1)

es decir forma el 'siguiente' valor (en el caso de indicar 'DOWNTO', en lugar de 'TO', se tomará el 'anterior' valor, en lugar del 'siguiente').

-Si después de este 'incremento', el valor de expresión1 sigue siendo menor o igual al de expresión2, se vuelve a 'incrementar' o 'decrementar' a expresión1 y se ejecuta nuevamente la instrucción. Esto se repite hasta que la última condición señalada se deje de cumplir.

Los tipos de expresión1 y expresión2, deben ser tales que la función SUCC se encuentre definida para ellos.

Por ejemplo: sean I y J variables enteras:

```
J:=10;
FOR I:=1 TO J DO
    WRITELN (I)
(*ENDFOR*);
```

el flujo del programa pasará exactamente J veces por la instrucción WRITELN, en este caso J vale 10.

CAPITULO 7

ARREGLOS

Un arreglo es una colección ordenada de variables que son del mismo tipo. Por ejemplo, una línea de texto puede ser representada como un arreglo de caracteres, un vector puede ser representado como un arreglo de números reales. Como una página de libro contiene renglones de texto, una página puede ser representada como un arreglo de líneas de texto, y el libro a su vez puede ser representado como un arreglo de páginas. El tipo de un arreglo se declara en función de:

a) El tipo del índice, y

b) El tipo de los componentes.

ejemplo:

```
TYPE
    direccion = (x,y,z);
    vector = ARRAY [direccion] OF REAL;
VAR
    s,t: direccion;
    u,v: vector;
```

Se declara el tipo vector. El tipo del índice del tipo vector es 'direccion' y el tipo de los componentes de tipo vector es REAL. De la manera usual se pueden declarar variables del tipo vector, como se muestra al declarar las variables u,v en el ejemplo anterior. Las variables que sean del tipo vector tienen tres componentes, cada uno correspondiente a los tres valores del tipo 'direccion'. Los tres componentes del vector u son:

u[x], u[y], u[z]

La notación matemática convencional de los vectores usa subíndices para denotar cada elemento del vector. Los componentes del vector u se denotarán de la siguiente manera:

$$u = u_x \quad u_y \quad u_z$$

Por esta razón, a los valores de los tipos de los índices se les conoce como subíndices. Cada uno de los componentes del vector u es una variable del tipo real y pueden ser usados en cualquier contexto donde se permite el manejo de las variables de tipo real. Ejemplo: La normal de un vector (también conocida como producto punto) es igual a la suma de los cuadrados de cada uno de los componentes del vector, y puede ser calculada de cualquiera de las siguientes formas:

a) normal:=(u[x]*u[x])+(u[y]*u[y])+(u[z]*u[z])

b) normal:= SQR(u[x])+SQR(u[y])+SQR(u[z])

c)

```
BEGIN
normal:= 0;
FOR i:= x TO z DO
    normal:=normal+SQR(u[i]);
(*ENDFOR*)
END
```

El valor de los componentes de un arreglo puede ser asignado a otro arreglo (siempre y cuando sea del mismo tipo), mediante una simple instrucción de asignación.

Ejemplo: La instrucción

u:=v

en donde u y v son del tipo vector, es equivalente al conjunto de instrucciones de asignación

```
u[x]:=v[x]
u[y]:=v[y]
u[z]:=v[z]
```

Un arreglo puede ser también un valor o una variable de un parámetro de un 'procedure' o de una 'function'. Por ejemplo: El producto punto de dos vectores es calculado por la función:

```
FUNCTION  prodpunto(  u,v: vector ): REAL
    VAR
         ppun:REAL;
         s:direccion;
    BEGIN
         ppun:=0;
         FOR s:=x TO z DO
             ppun:=ppuntu[x]*v[s];
         (*ENDFOR*)
         prodpunto:=ppun;
    END
(*ENDFUNCTION*);
```

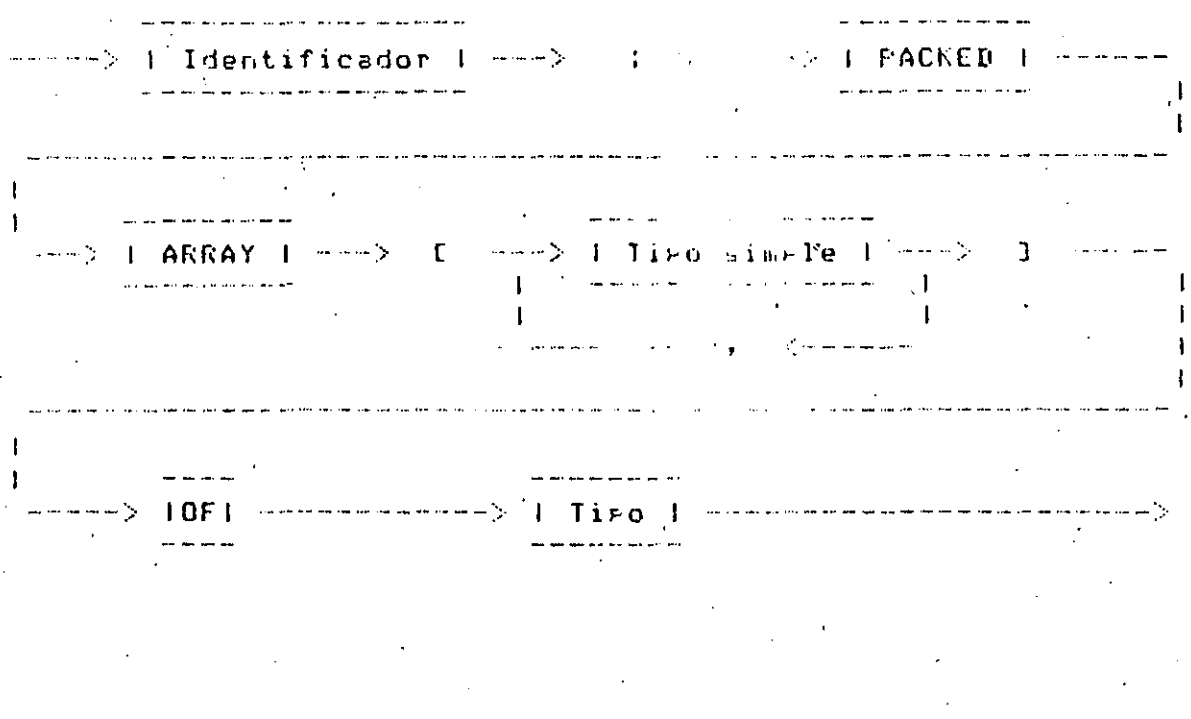
El resultado del producto cruz de los vectores es un tercer vector. El valor de una función no puede ser un arreglo, por lo tanto no se puede escribir una función que realice el producto cruz. Por otro lado, se puede escribir un procedimiento que evalúe el producto cruz de los vectores u,v y el resultado sea el vector w.

```
PROCEDURE  prodcruz(  u,v:vector;
                      VAR w:vector);
    BEGIN
    w[x]:=u[y]*v[z] - u[z]*v[y];
    w[y]:=u[z]*v[x] - u[x]*v[z];
    w[z]:=u[x]*v[y] - u[y]*v[x];
    END;
(*ENDPROCEDURE*)
```

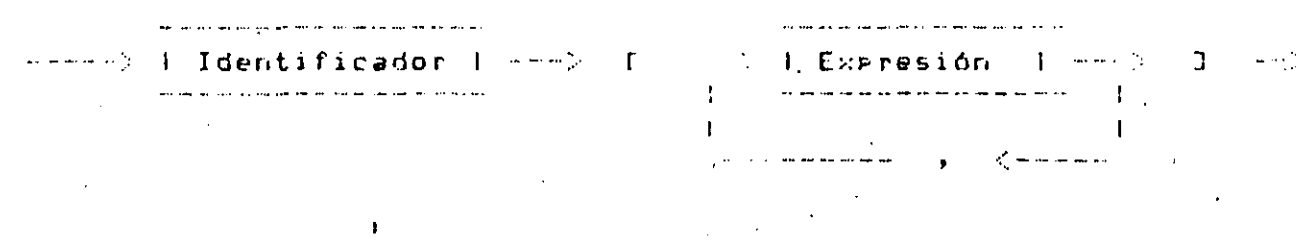
7.1 DIAGRAMA DE SINTAXIS.

La sintaxis de la declaración de una variable cuyo tipo sea ARRAY se muestra en el siguiente diagrama de sintaxis. El tipo del índice debe ser un tipo escalar o un subrango. Debe recordarse que REAL no es un escalar, por lo tanto, no puede declararse un arreglo donde el tipo del índice sea REAL. El tipo del componente puede ser de cualquier tipo, incluyendo los tipos estructurados. Un elemento o componente de un arreglo tiene las mismas propiedades que una variable de su tipo de base, y es por lo tanto una variedad de factor.

Declaración de arreglos:



Declaración de los componentes de un arreglo:



7.2 ARREGLOS MULTIDIMENSIONALES.

El tipo base de un arreglo puede ser él mismo un arreglo. En estas declaraciones, el tipo base de matriz es 'columna'.

```
CONST
     limsuperior=10;
TYPE
     subindices    = 1..limsuperior;
     columna       = ARRAY [subindices] OF REAL;
     matriz        = ARRAY [subindices] OF columna;
```

La declaración de 'columna' se puede incorporar a la declaración de 'matriz', y quedaría de la siguiente forma:

```
TYPE
    matriz= ARRAY [subindices] OF ARRAY [subindices] OF REAL;
```

Existe otra forma más conveniente de representar esta expresión:

```
TYPE
    matriz= ARRAY [subindices,subindices] OF REAL;
```

Declaremos algunas variables para ejemplificar algunos casos:

```
VAR
    a,b,c : matriz;
    r,s,t : subindices;
```

La columna s de la matriz c se denotaría de la siguiente forma:

c[s]

El elemento o componente t de la matriz c es una variable real que puede ser denotada como:

c[s] [t]

o

c[s,t]

El arreglo c es llamado arreglo bidimensional o de dos dimensiones, porque podemos imaginar que está guardado en memoria de la siguiente manera:

```
FOR r:=1 TO limsuperior DO
    FOR s:=1 TO limsuperior DO
        BEGIN
        c[r,s]:=0;
        FOR t:=1 TO limsuperior DO
            c[r,s]:=c[r,s]+a[r,t]*b[t,s];
        (*ENDFOR t*)
        END
    (*ENDFOR s*)
(*ENDFOR r*)
```

El componente c[r,s] es accesado (2 * limsuperior) veces en la instrucción FOR más interna, si se trabaja con un buen compilador, ésto no tiene importancia, pero si se trabaja con un compilador sencillo, se puede evitar ésto, haciendo referencia a él, una sola vez de la siguiente forma:

```
FOR r:=1 TO limsuperior DO
    FOR s:=1 TO limsuperior DO
        BEGIN
        sum :=0;
        FOR t:=1 TO limsuperior DO
            sum := sum+a[r,t]*b[t,s];
        (*ENDFOR t*)
            c[r,s]:=sum;
         END
    (*ENDFOR s*)
(*ENDFOR r*)
```

7.3 ARREGLOS EMPACADOS.

Los componentes de un arreglo son almacenados en memoria en palabras consecutivas, ésta es una manera eficiente para almacenar elementos enteros y reales de un arreglo. Pero no siempre es una manera eficiente para almacenar variables de otros tipos, porque puede darse el caso de que haya espacio de memoria malgastado. La cantidad de memoria malgastada puede ser reducida empacando los componentes de un arreglo dentro de una palabra de memoria. El compilador realizará este trabajo al declarar el arreglo 'PACKED'. Por ejemplo, el arreglo 'muchoscaracteres' declarado de la siguiente manera:

```
VAR
    muchoscaracteres : ARRAY [1..1000] OF CHAR;
```

Es equivalente a:

```
conjuntox:= conjuntox + indice;
conjuntox:= conjuntox + indice;
```

Y la expresión booleana:

```
arreglo[indice];
```

Es equivalente a:

```
indice IN conjuntox;
```

Los operadores <= y => para preguntar si un conjunto contiene a otro conjunto, no pueden ser aplicados a arreglos booleanos, por supuesto. Las operaciones en un conjunto seran más rápidas que las operaciones correspondientes en un arreglo booleano y debe procurarse usar conjuntos en lugar de arreglos booleanos, siempre que sea posible. En muchos compiladores - Pascal, un arreglo booleano puede contener más elementos o componentes que un conjunto. Cuando se necesita usar un conjunto muy grande de elementos, digamos de 10,000 o 100,000 componentes, probablemente el programa sea más fácil de escribir (y de leer) si se utiliza un arreglo booleano de una dimensión que un arreglo de conjuntos. Una considerable cantidad de espacio de memoria puede salvarse (con el incremento del tiempo de ejecución) empacando el arreglo.

El algoritmo clásico para enumerar los números primos es "La criba de Eratostenes". Supongamos que se desea encontrar los números primos menores que 10. Debemos empezar a escribir los números desde el 2 hasta el 10:

2 3 4 5 6 7 8 9 10

Luego quitamos el número más pequeño y lo tomamos como número primo, y quitamos (dejando caer por la coladera de la criba) todos sus múltiplos. Después de este primer paso tenemos el número 2 como número primo y la criba contiene únicamente números impares:

3 5 7 9

Después del segundo paso tenemos el 3 como número primo y quedan únicamente el 5 y el 7 en la criba. El proceso termina cuando la criba está vacía.

Declararemos:

```
CONST
    maximo = 100000;
VAR
    criba: PACKED ARRAY [2..maximo] OF BOOLEAN;
```

Inicialmente, se pondrán en 'verdadero' todos los componentes del arreglo booleano que representa a la criba indicando que todos los números están presentes. Como vayamos quitando números, pondremos el elemento correspondiente del arreglo en 'falso'. El programa consiste de dos 'loops' anidados, uno para encontrar los números primos y el otro para quitar todos sus múltiplos.

```
PROGRAM numerosprimos (INPUT, OUTPUT);
    CONST
        primerprimo=2;
        maximo=100000;
    VAR
        criba:PACKED ARRAY [primerprimo..maximo] OF BOOLEAN;
        rango,factor,multiplo,izin: 0..maximo;
    BEGIN
        READ(rango);
        FOR factor:=primerprimo TO rango DO
            criba[factor]:=true
        (*ENDFOR*);
        izin:= rango-primerprimo+1;
        factor:=primerprimo-1;
        REPEAT
            factor:=factor+1;
            IF criba[factor] THEN
                BEGIN                    (* vector es primo *)
                WRITELN(factor);
                multiplo:=1;
                WHILE factor*multiplo <= rango DO
                    BEGIN
                    IF criba[factor*multiplo] THEN
                       BEGIN                (* quitamos multiplo*)
                       criba[factor*multiplo]:=FALSE;
                       izin:=izin-1;
                       END
                    (*NOELSE*)
                    (*ENDIF*);
                    multiplo:=multiplo+1;
                    END
                (*ENDWHILE*);
                END
            (*NOELSE*)
            (*ENDIF*);
        UNTIL izin = 0;
    END (*numeros primos*)
(*ENDPROGRAM*).
```

Resultado de la corrida del programa:

Entrada:

50

Salida:

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
```

CAPITULO 8

FUNCIONES Y PROCEDIMIENTOS

Es frecuente encontrar ocasiones, dentro de un programa de computadora, en el que se necesita obtener un resultado de acuerdo a una cierta fórmula ya establecida, por lo que se escribe la fórmula en instrucciones del lenguaje de computadora, las cuales pudieran ser muy numerosas, y sucede que más adelante se requiere obtener un resultado semejante utilizando la misma fórmula, pero trabajando con datos diferentes.

También ocurre que una vez definido un cierto grupo de instrucciones encargadas de realizar algún proceso en particular dentro del programa, se observa que este mismo proceso es necesario repetirlo en alguna otra sección del programa.

Ante la repetición continua de estas situaciones, es obvio preguntarse si sería posible escribir las instrucciones necesarias para efectuar estos procesos una sola vez y utilizarlas luego repetidamente tantas veces como sea necesario. La respuesta es sí, éste es precisamente el objetivo de las funciones y de los procedimientos, ya que ayudan a minimizar el trabajo del programador y permiten que la labor de un cierto programador sea aprovechada por muchos otros.

Los procedimientos del lenguaje Pascal son grupos de instrucciones que se encargan de realizar un cierto proceso en particular, generalmente bien delimitado de otros procesos, y que el programador define no solo para evitarse trabajo en el caso de que fuera necesario repetir ese proceso en varias ocasiones, sino porque al dividir la labor de un progrma en segmentos pequeños y bien definidos el programador puede darse cuenta más rapidamente si ha cometido algún error o ha olvidado algún detalle, al visualizar y comprender por completo el funcionamiento de un proceso que es relativamente pequeño.

Las funciones son grupos de instrucciones que se encargan de evaluar lguna fórmula en particular, siguiendo el método definido por el rogramador, y entregar un resultado de acuerdo a un cierto dato de entrada en base al cual se evaluará la fórmula correspondiente.

Además de las funciones que un programador puede definir para sus propios cálculos, es común que un lenguaje de aplicación científica como es Pascal tenga incluidas las funciones matemáticas y de otro tipo que se utilizan más frecuentemente por los programadores, como son raíz cuadrada, logaritmo, etc.

De esta manera, a las funciones que ya trae definido el lenguaje se les llama funciones predefinidas, mientras que las que especifica el programador serán funciones definidas por el usuario.

8.1 FUNCIONES PREDEFINIDAS

Para utilizar una función, basta con poner el nombre de la función y adelante de él y encerrado entre paréntesis, el argumento de la función, o sea, la cantidad sobre la cual va a operar la función. Tanto el argumento con el que va a operar la función como el valor que regresa la función, son expresiones aritméticas y por tanto se regirán por las mismas reglas vistas anteriormente para expresiones aritméticas. (Capítulo 4)

Por ejemplo, si SQRT es el nombre de la función que calcula la raíz cuadrada de un número y SIN es el nombre de la que obtiene el seno trigonométrico, tendríamos que:

SQRT(25)	representa el valor 5.0 (raíz cuadrada de 25)
SQRT(A)	es la raíz cuadrada de la variable A
SIN(X+Y)	es el seno de X+Y
SQRT(SIN(A+B)*SIN(A-B))	es la raíz cuadrada del producto del seno de A+B multiplicado por el seno de A-B

Puesto que existen diferentes tipos de datos y el manejo de los mismos varía de un tipo a otro, al utilizar las funciones predefinidas es necesario observar los estándares en cuanto a los tipos de datos con que operan las funciones, es decir, de qué tipo debe ser la cantidad sobre la cual opera la función, y de qué tipo es el valor que proporciona la función.

La siguiente tabla muestra las funciones predefinidas con que cuenta el lenguaje Pascal mostrando el nombre de la función, el tipo de la función, es decir, el tipo del valor que entrega la función como resultado, el tipo de su argumento, o sea, el tipo del valor con el cual va a trabajar la función, y una breve descripción del cálculo que realiza la función; las funciones están agrupadas de acuerdo al tipo de cálculo que realizan.

NOMBRE FUNCION	TIPO DE FUNCION	TIPO DE ARGUMENTO	DESCRIPCION DE LA FUNCION
*******	FUNCIONES MATEMATICAS		*******
SIN	REAL	REAL ó INTEGER	Seno trisonométrico del argumento.
COS	REAL	REAL ó INTEGER	Coseno trisonométrico del argumento.
ARCTAN	REAL	REAL ó INTEGER	Arco tangente del argumento.
LN	REAL	REAL ó INTEGER	Logaritmo natural del argumento.
EXP	REAL	REAL ó INTEGER	Exponencial (antilogaritmo natural) del argumento.
SQRT	REAL	REAL ó INTEGER	Raiz cuadrada del argumento.
SQR	Igual que argumento	REAL ó INTEGER	Argumento elevado al cuadrado.
ABS	Igual que argumento	REAL ó INTEGER	Valor absoluto del argumento.
TRUNC	INTEGER	REAL	Trunca (elimina) la parte fraccionaria del argumento.
ROUND	INTEGER	REAL	Redondea al entero más cercano al argumento.
*******	FUNCIONES DE TIPO BOOLEAN		*******
ODD	BOOLEAN	INTEGER	Informa si el argumento es impar.
EOLN	BOOLEAN	FILE	Informa si se terminó una línea de datos de entrada.
EOF	BOOLEAN	FILE	Informa si se terminó un archivo de datos de entrada.

******* FUNCIONES SOBRE ARGUMENTOS ENUMERADOS *******

PRED	Igual que argumento	Cualquiera menos REAL	Predecesor del argumento, valor inmediato anterior.
SUCC	Igual que argumento	Cualquiera menos REAL	Sucesor del argumento, valor inmediato posterior.
ORD	INTEGER	CHAR ó BOOLEAN	Ordinal del argumento, posición del argumento.
CHR	CHAR	INTEGER	Caracter correspondiente al ordinal dado por el argumento.

Las funciones mostradas en la tabla anterior son las funciones predefinidas estándar de Pascal, o sea, las funciones que presumiblemente tiene cualquier compilador Pascal que exista; sin embargo, es costumbre que los compiladores particulares de tal o cual marca de computadoras tengan una serie de funciones adicionales a las estándar que realizan algunas operaciones interesantes, por lo que siempre convendrá revisar el manual del lenguaje Pascal a que tengamos acceso para conocer todas las funciones predefinidas con que cuenta.

Las funciones SIN, COS, ARCTAN, LN, EXP y SQRT pueden aplicarse a argumentos de tipo REAL ó INTEGER pero el valor que proporcionan siempre es de tipo REAL, por ejemplo;

```
EXP(1.0)    vale: 2.718281
EXP(1)      vale: 2.718281

SQRT(25.0)  vale: 5.0
SQRT(25)    vale: 5.0
```

Las funciones SQR y ABS pueden aplicarse a argumentos de tipo REAL ó INTEGER y regresan un valor del mismo tipo que el argumento, por ejemplo:

```
SQR(5)      vale: 25
SQR(5.0)    vale: 25.0

ABS(-7)     vale: 7
ABS(-1.0)   vale: 1.0
```

Las funciones TRUNC y ROUND sirven para convertir un valor de tipo REAL a otro de tipo INTEGER; la función TRUNC simplemente desecha la parte fraccionaria del argumento:

```
TRUNC(3.25) vale: 3
TRUNC(3.75) vale: 3
```

Mientras que la función ROUND regresa el valor entero más cercano al argumento, por ejemplo:

```
ROUND(3.25) vale: 3
ROUND(3.75) vale: 4
```

Las funciones ODD, EOLN y EOF regresan un valor de tipo BOOLEAN el cual puede probarse en una instrucción IF o WHILE ó manipularse de cualquier otra forma. La función ODD informa si el argumento es impar o no, el argumento debe ser de tipo INTEGER, por ejemplo:

```
I:=-6;      ODD(I)      vale: FALSE
I:=17;      ODD(I)      vale: TRUE
```

Las funciones EOLN y EOF se describen detalladamente en el capitulo 5.7 correspondiente a entrada y salida.

Las funciones PRED, SUCC, ORD y CHR se describen con detalle en el capitulo 9.1 correspondiente a tipos escalares de datos.

8.2 FUNCIONES DEFINIDAS POR EL USUARIO.

A pesar de que las funciones predefinidas que proporciona el lenguaje Pascal son de una gran ayuda, hay muchos otros cálculos que los programadores utilizan con frecuencia y que no se proporcionan como funciones predefinidas de Pascal.

Como ésto ocurre muy a menudo, el lenguaje Pascal nos permite definir nuestras propias funciones las cuales, una vez definidas, podremos utilizar en forma completamente similar a las funciones predefinidas.

Para definir una función debemos especificar varias cosas como son: qué nombre va a tener la función, qué tipo de valor va a regresar y de qué tipo debe ser su argumento (ó argumentos, si es que tiene más de uno).

Esto se especifica de la siguiente manera: se coloca la palabra reservada FUNCTION seguida por el nombre que tendrá la nueva función seguido por un paréntesis izquierdo, el cual encerrará el nombre del parámetro formal, el cual a su vez debe de ir seguido por un caracter de dos puntos (:) y el tipo del parámetro formal; finalmente se cierra el paréntesis derecho correspondiente y adelante de él se coloca otro caracter dos puntos y el tipo del valor que regresará la función.

El parámetro formal es la variable dentro de la función que representará el valor del argumento cuando la función sea utilizada; dicho de otra forma, si estamos utilizando la función, hablamos de proporcionarle un argumento para que trabaje, mientras que si estamos definiendo la función, hablamos de definir un parámetro formal para saber que manipulación hacerle al argumento correspondiente.

Como ya se indicó previamente, la definición del programa principal comienza con la instrucción PROGRAM descrita en un capítulo anterior, después sigue la declaración de constantes, la de tipos (si es que hay), la de variables, y al final un bloque BEGIN el cual encierra las instrucciones que se ejecutarán y que termina en una instrucción END.

La definición de una función es enteramente similar a la de un programa principal, la única diferencia es que comienza con una instrucción FUNCTION en vez de un PROGRAM; ahora bien, adicionalmente dentro del bloque que encierra las instrucciones correspondientes a la función debe utilizarse la variable definida como parámetro formal de la función y se debe asignar un valor al nombre de la función; este valor será el que entregue la función, como resultado, al programa llamador.

Por ejemplo, supongamos que queremos definir una nueva función llamada INVERSO la cual se utilizará con un argumento de tipo REAL y regresará un valor también de tipo REAL; el valor que regrese la función será el inverso del valor proporcionado como argumento. La definición de esta función se haría de la siguiente manera:

```
FUNCTION INVERSO ( ARG : REAL ): REAL ;
   BEGIN
   INVERSO:=1.0/ARG
   END
(*ENDFUNCTION*);
```

El parámetro formal está indicado por la variable llamada ARG, la cual se está declarando de tipo REAL; esta variable contendrá el valor de entrada proporcionado al utilizar esta función y es una variable local al bloque BEGIN que constituye la función, por lo que no afectará en nada a ninguna variable externa a este bloque.

La definición de una función (ó de las funciones, si es que se define más de una) deberá hacerse después de la declaración de variables del programa principal y antes del BEGIN correspondiente al bloque del programa principal.

Una vez definida una función podrá utilizarse en forma completamente similar a la de las funciones predefinidas de Pascal, por ejemplo:

```
INVERSO(4.0)                 representa al valor 0.25 (ó 1/4.0).
INVERSO(A)                   es el inverso de la variable A.
INVERSO(SQRT(X)+SQRT(Y))     es el inverso de la suma de la raiz
                             cuadrada de X mas la raiz cuadrada de Y.
```

Al utilizar una función definida por el usuario (así como una predefinida) debemos proporcionar a la función un argumento del mismo tipo que el del parámetro formal definido en la función, de lo contrario ocurrirá un error; sin embargo, cuando el parámetro formal es de tipo REAL se puede proporcionar un argumento de cualquiera de los tipos REAL ó INTEGER sin ningún problema; por ejemplo, suponiendo que I, J y K son variables de tipo INTEGER, tendríamos que:

```
INVERSO(2)                   es el valor 0.5 (ó 1/2).
INVERSO(I)                   es el inverso de la variable I.
INVERSO(SQR(J) DIV SQR(K))   es el inverso de la división
                             del cuadrado de la variable J
                             entre el cuadrado de la variable K.
```

Por supuesto, una función definida por el usuario puede tener más de un argumento, en cuyo caso bastará con declarar los correspondientes parámetros formales y utilizarlos en forma adecuada dentro de la función; por ejemplo, supongamos que dentro de un cierto programa de manejo de triángulos, constantemente necesitamos el valor de la hipotenusa cuando conocemos el valor de los dos catetos corespondientes; para hacer este cálculo bastará con definir la siguiente función:

```
FUNCTION HIPOTENUSA ( LADOA, LADOB : REAL ) : REAL;
    BEGIN
    HIPOTENUSA:=SQRT(SQR(LADOA)+SQR(LADOB));
    END
(*ENDFUNCTION*);
```

Al utilizar esta función habrá que separar los dos argumentos correspondientes por una coma de la siguiente manera:

```
HIPOTENUSA(CAT1,CAT2)          o
HIPOTENUSA(SQRT(A-B*C),X*X-SIN(1.0-LN(X/Y)))      etc...
```

De igual forma que un programa principal, una función puede tener cualquier definición de constantes, de tipos de variables ó de variables que sea necesario; estas declaraciones sólo regirán dentro del bloque BEGIN-END correspondiente a la función.

Un primer uso de lo anterior es cuando una función utiliza una cierta constante que no requiere las demás funciones ó el programa principal, por lo que no hay razón para declarar esa constante en forma global a la función, ó bien, cuando la función requiere de un almacenamiento auxiliar para efectuar su cálculo, es conveniente que ese almacenamiento temporal se haga en una variable local a la función con el fin de evitar que inadvertidamente la función modifique un valor global que no debería modificar.

Una función puede tener, inclusive, declaraciones de funciones internas a ella, que sólo serán conocidas por la declaración "anidada" de funciones; deberá ir en su lugar correspondiente, es decir, después de la declaración de variables de la función externa y antes del BEGIN correspondiente al bloque de la función externa.

Un ejemplo de uso de una definición "anidada" de funciones es cuando la labor que efectúa una función es muy complicada y requiere a su vez de la ayuda de otra función para facilitar su trabajo, ahora bien, si la función auxiliar no se utiliza en ninguna parte mas que en la función complicada, no tiene caso que esta función auxiliar sea declarada en forma global; y esto podría aún ocasionar errores si de casualidad la función auxiliar se llamara igual que alguna otra cantidad global.

Una diferencia importante entre las funciones predefinidas y las definidas por el usuario, es que mientras las primeras requieren que el argumento sea un sólo valor de un tipo predefinido de Pascal, como se vió en el subcapítulo 8.1 anterior, las funciones definidas por el usuario pueden tener argumentos que sean de cualquier tipo, ya sea predefinido o bien definido por el usuario.

Esto implica que los argumentos de una función definida por el usuario pueden ser de cualquier tipo definido por el usuario (escalares, subrangos, conjuntos, registros, arreglos, etc.). Para utilizar esta característica del lenguaje Pascal bastará con declarar del tipo adecuado al parámetro formal de la función y utilizarlo correctamente de acuerdo con las reglas existentes para el manejo del mismo.

Por ejemplo, supongamos que deseamos definir una función que sume todos los elementos de un vector de valores de tipo REAL y que nos proporcione la suma como resultado, además del arreglo de valores tendremos que indicarle a esta función cuántos elementos deseamos que sume. Las declaraciones globales del arreglo y la definición de la función podrían ser como sigue:

```
CONST
    MAXIMO = 20;
TYPE
    SUBINDICE = 1..MAXIMO;
VAR
    VECTOR : ARRAY [SUBINDICE] OF REAL;
    LIMITE : SUBINDICE;
    SUMA : REAL;
FUNCTION SUMAVEC ( ARGVEC : ARRAY [SUBINDICE] OF REAL;
                   ARGLIM : SUBINDICE ) : REAL;
    VAR
        AUX : REAL;
        I : SUBINDICE;
    BEGIN
    AUX:=0;
    FOR I:=1 TO ARGLIM DO
        AUX:=AUX+ARGVEC[I]
    (*ENDFOR*);
    SUMAVEC:=AUX;
    END
(*ENDFUNCTION*);
```

Una vez definida la función anterior podría utilizarse como sigue:

```
LIMITE:=14;
SUMA:=SUMAVEC(VECTOR,LIMITE);
```

A pesar de que el argumento pasado a una función, y por consiguiente el correspondiente parámetro formal, puede ser de cualquier tipo, el valor que regresa la función sólo puede ser de un tipo elemental de datos, o sea, debe ser un sólo valor de cualquiera de los tipos predefinidos de Pascal o bien, un sólo valor de cualquier tipo definido por el usuario, es decir, no puede regresar ningún conjunto de datos como podría ser un arreglo ó un registro, pero sí podría regresar un

sólo elemento de un arreglo o de un registro.

Es importante recordar que los parámetros formales de las funciones son variables locales a ellas, por lo tanto, la ejecución de una función, de acuerdo a lo visto hasta este punto, no puede modificar el valor de su correspondiente argumento.

Como en ocasiones es deseable que la ejecución de una función modifique el argumento correspondiente, ya que ésto puede minimizar el trabajo de programación, el lenguaje Pascal cuenta con los mecanismos para realizar tal operación. Sobre estos mecanismos tratará el subcapítulo 8.4 que veremos más adelante.

8.3 PROCEDIMIENTOS

Los procedimientos son similares a las funciones en el sentido de que también son grupos de instrucciones que tienen sus propias declaraciones de constantes, variables, etc. y se declaran en forma enteramente similar a las funciones, sólo que la instrucción inicial debe ser PROCEDURE en vez de FUNCTION. Puesto que las funciones y procedimientos son pequeños programas definidos dentro del programa principal, se acostumbra llamarlos "subprogramas".

La diferencia fundamental entre ambos tipos de subprogramas radica en que mientras las funciones siempre entregan un valor como resultado, y por esta razón hay que utilizarlas como expresiones aritméticas ó de otro tipo, los procedimientos nunca entregan un valor en forma explícita y por esta razón hay que utilizarlos como si fueran alguna instrucción de Pascal. Debido a esta diferencia, los subprogramas de función generalmente se definen con uno o más parámetros formales, ya que casi siempre que utilizamos una función le debemos proporcionar el argumento con el cual trabajará, mientras que en el caso de los procedimientos, al definirle o no parámetros formales depende fundamentalmente del tipo de proceso que efectuará.

Las funciones y los procedimientos están diseñados para diferentes aplicaciones; las funciones se utilizan cuando deseamos obtener un resultado aritmético ó de otro tipo a partir de ciertos datos, mientras que los procedimientos se utilizan cuando deseamos ejecutar varias veces un cierto proceso o bien, como se explicará más adelante, cuando deseamos dividir un proceso grande en varios procesos pequeños.

Los procedimientos también pueden entregar un valor como las funciones, pero nunca lo hacen de forma explícita sino a través de variables globales o bien, como veremos en el siguiente subcapítulo, modificando el valor de sus argumentos.

Supongamos que durante la generación de un reporte frecuentemente necesitamos imprimir varias veces un mismo caracter dado, lo cual está definido mediante las variables NUMVECES y CARACTER; la variable NUMVECES es de tipo INTEGER e indica el número de veces que se repetirá el caracter, mientras que la variable CARACTER es de tipo CHAR y contiene el caracter a repetir; el siguiente procedimiento llamado REPITECAR efectuaría ese proceso.

```
PROCEDURE REPITECAR;
    VAR
        I : INTEGER;
    BEGIN
    FOR I:=1 TO NUMVECES DO
        WRITE(CARACTER);
    (*ENDFOR*)
    END
(*ENDPROCEDURE*);
```

La declaración del procedimiento es completamente similar a la de una función, sólo que en este caso en particular el procedimiento no tiene parámetros formales, por lo que no tiene los paréntesis que los encierran; de igual manera, puesto que un procedimiento no entrega un valor como una función, no lleva el caracter de dos puntos y el tipo del valor que regresa, como en el caso de las funciones.

El procedimiento anterior opera basándose en los valores de las variables NUMVECES y CARACTER las cuales presumiblemente son variables globales al procedimiento, ya que de no ser así, el compilador marcaría el error correspondiente a "Identificador no declarado".

Para utilizar este procedimiento en el programa llamador habría que asignarle valores a las variables NUMVECES y CARACTER y después ejecutar el procedimiento, lo cual se hace simplemente poniendo su nombre, por ejemplo:

```
NUMVECES := 25;
CARACTER := '*';
REPITECAR;
```

Las tres instrucciones anteriores ocasionarían que se mandaran 25 asteriscos a la línea de impresión correspondiente. Como puede verse fácilmente, el utilizar repetidamente un procedimiento que trabaja en base a variables globales puede ocasionar algunos problemas, ya que hay que estar asignando cada vez los valores correspondientes a las variables necesarias, además de que no se aprecia claramente la relación que hay entre las variables globales y el procedimiento; cuando ésto ocurre es más conveniente declarar parámetros formales para el

procedimiento de la manera usual. Por ejemplo:

```
PROCEDURE REPITECAR ( NUMVECES : INTEGER; CARACTER : CHAR );
    VAR
        I : INTEGER;
    BEGIN
    FOR I:=1 TO NUMVECES DO
        WRITE(CARACTER);
    END
(*ENDPROCEDURE*);
```

De esta manera la utilización del procedimiento REPITECAR es más sencilla y comprensible, ya que para mandar imprimir 47 veces el caracter de igual (=), después dejar 13 espacios en blanco y finalmente repetir otros 30 asteriscos bastaría lo siguiente:

```
REPITECAR ( 47, '=' );
REPITECAR ( 13, ' ' );
REPITECAR ( 30, '*' );
```

Tan importante, y más aún, que la facilidad de repetir en forma sencilla un cierto proceso al definirlo en un procedimiento y luego utilizarlo repetidamente, es el hecho de que los procedimientos nos ayudan a hacer más comprensible y claro nuestro programa al permitirnos dividirlo en secciones relativamente pequeñas y fáciles de entender.

En efecto, cuando estamos desarrollando un programa de tamaño mediano o grande, normalmente el programa en su totalidad se compone de una serie de pequeñas tareas, algunas veces relacionadas entre sí, algunas veces independientes una de otra. Al estar encargados del desarrollo y pruebas del programa, o bien de una modificación posterior del mismo, generalmente no necesitamos tener a la vista la totalidad del programa sino solamente aquellas secciones de él que se relacionan directa o indirectamente con la parte del programa que estemos probando ó desarrollando en un momento dado.

De esta manera, es muy conveniente dividir todo el programa en procedimientos encargados de una tarea específica aunque muchos de estos procedimientos no sean utilizados en el programa mas que una sola vez, ya que al hacerlo así nos daremos cuenta de inmediato qué es lo que tiene que realizar un procedimiento en particular y podremos detectar más facilmente los posibles errores al concentrarnos exclusivamente en un proceso pequeño.

Además, haciendo esta división en una forma ordenada obtendremos otras ventajas, por ejemplo: si dos o más procedimientos se relacionan entre sí por medio de ciertos datos, los cuales no tienen nada que ver con algunos otros procedimientos; es conveniente declarar esos datos como parámetros locales solamente de los procedimientos que los utilizan. Por otro lado, si existen otros datos que utilizan la mayoría de los procedimientos, es conveniente que se declaren como variables globales a todos ellos.

Esto nos permite detectar de inmediato cuáles son los datos de entrada y de salida de un procedimiento en particular y cuales son generales a todo el programa con lo que sabríamos con sólo revisar la definición de los procedimientos cuales de ellos habría que modificar y cuáles no en caso de que se alteraran los datos con que opera el programa.

Como regla general, un procedimiento no debe ser más grande que lo que podamos revisar de una sola vez, es decir, una hoja impresa por la computadora ó el número de líneas mostradas en una pantalla de una terminal; si un procedimiento es más grande que esto siempre será posible dividirlo en dos o más procedimientos más pequeños que hagan la misma labor.

Al escoger los nombres de los procedimientos, así como en general el nombre de cualquier identificador que definamos en un programa, será de una gran ayuda si el nombre se escoge de tal manera que describa el proceso que realiza el procedimiento, o bien, el uso que tiene el identificador.

Por ejemplo, lo siguiente puede ser una sección de un programa encargado de invertir matrices:

```
LEEDIMENSIONES;
LEEMATRIZ;
IMPRIMEMATRIZ (ENTRADA);
SACADETERMINANTE;
IF DETERMINANTE > 0 THEN
    BEGIN
    INVIERTEMATRIZ;
    IMPRIMEMATRIZ (INVERSA);
ELSE
    IMPRIMEMENSAJE;
(*ENDIF*)
```

El proceso que efectúa este programa es claro para cualquiera que lo revise, como igualmente son claros los procesos que deben ejecutar cada uno de los procedimientos en que está dividido el programa; en caso de que ocurra algún error, es fácil detectar en cual procedimiento ocurrió

y bastará con concentrar nuestra atención en ese procedimiento en particular sin importarnos los demás, ya que ningún procedimiento se relaciona directamente con otro.

Finalmente, al tener el programa dividido de esta manera podremos encargar la programación de los procedimientos a diferentes programadores sin tener que explicar a cada uno qué es lo que realiza el programa; bastará con indicarles lo que tiene que hacer su procedimiento en particular, cuáles son las variables con que deben trabajar y en que variables deben dejar su resultado. O bien, podremos contar con una serie de procedimientos específicos ya escritos y simplemente tomar uno o varios que realicen los procesos que queramos y con leves modificaciones incluirlos en nuestro programa.

8.4 PASO DE PARAMETROS POR REFERENCIA

La forma de pasar un argumento a un subprograma vista hasta este momento implica que el parámetro formal del subprograma es una variable local a él y, por lo tanto, el subprograma no podrá modificar el valor del argumento correspondiente. Por lo tanto, al utilizar un subprograma el argumento podrá ser cualquier expresión aritmética o de otro tipo, ya que lo que se "pasa" al parámetro formal correspondiente del subprograma simplemente es el valor que tenga la expresión al momento de hacer la llamada al subprograma. Debido a ésto ese tipo de paso de parámetros se llama "paso de parámetros por valor".

Existe otro tipo de paso de parámetros en el cual el parámetro formal del subprograma se "conecta" con el argumento correspondiente permitiendo que toda manipulación que se haga al parámetro formal se vea reflejada en el argumento, como por ejemplo, alterar su valor. En este caso, el argumento no podrá ser una expresión, sino que forzosamente deberá ser una variable, ya que si se alterara el valor del parámetro formal habría que alterar igualmente el valor del argumento, y tal cosa no sería posible si el argumento no fuera simplemente una variable.

Al utilizar esta facilidad del lenguaje Pascal, el parámetro formal hace referencia al argumento como si fuera el mismo argumento el que se está manipulando; por lo tanto, este nuevo tipo de paso de parámetros se llama "paso de parámetros por referencia".

El subprograma es el que indica cuáles parámetros se pasan por valor y cuales por referencia, la distinción consiste en incluir la palabra reservada VAR en la declaración de los parámetros que serán pasados por referencia. De esta manera, el parámetro formal efectivamente representará una variable externa al subprograma.

Por ejemplo, supongamos que queremos hacer un procedimiento que intercambie el valor de sus dos argumentos como podría ser el siguiente:

```
PROCEDURE CAMBIA (VAR ARG1, ARG2 : REAL );
      VAR
          AUX : REAL;
      BEGIN
          AUX:=ARG1;
          ARG1:=ARG2;
          ARG2:=AUX;
      END;
(*endprocedure*);
```

Como se ve, la única diferencia con lo antes visto es la utilización de la palabra reservada VAR al declarar los parámetros formales del procedimiento; por supuesto, si esa palabra no se incluyera, el subprograma no funcionaría en forma adecuada ya que los valores de los argumentos al regresar del procedimiento no se verían afectados en lo más mínimo.

Para utilizar este procedimiento, podría hacerse como sigue:

```
CAMBIA ( A, B);
CAMBIA ( LADOX, LADOY );
```

Pero cualquiera de los siguientes ejemplos serían incorrectos:

```
CAMBIA (A, 4.5 );
CAMBIA (X, SIN(Y)+2.5 );
```

Ya que en el caso de que los parámetros sean declarados por referencia, los argumentos correspondientes necesariamente deberán ser simplemente nombres de variables.

El paso de parámetros por referencia es útil cuando se desea que el subprograma entregue más de un resultado, ya que al poder hacer referencia a variables externas a él, podrá entregar cualquier número de resultados a través de las variables que sean utilizadas como argumentos.

Supongamos que deseamos hacer un subprograma que obtenga la suma de dos matrices proporcionándole como datos las dos matrices y la dimensión de ellas, veamos el siguiente ejemplo:

```
CONST
   LIMITE = 20;
TYPE
   SUBINDICE = 1..LIMITE;
   MATRIZ = ARRAY [SUBINDICE,SUBINDICE] OF REAL;
PROCEDURE SUMAMAT ( VAR A     : MATRIZ;
                        B, C  : MATRIZ;
                        N, M  : SUBINDICE );
   VAR
       I, J : SUBINDICE;
   BEGIN
   FOR I:=1 TO N DO
       FOR J:=1 TO M DO
           A[I,J]:=B[I,J]+C[I,J]
       (*ENDFOR*)
   (*ENDFOR*);
   END
(*ENDPROCEDURE*);
```

El procedimiento SUMAMAT tiene cinco parámetros formales, tres arreglos que son la matriz del resultado y las dos matrices de datos, y dos subrangos que indican la dimensión de las matrices; puesto que solamente la primer matriz será la generada por el subprograma, es la única que tiene la declaración VAR de paso de parámetros por referencia, las demás no la tienen ya que no la necesitan y además con ello se evita el riesgo de que por error el subprograma modifique los valores de parámetros que no debiera.

La utilización del procedimiento se hace en la forma usual, por ejemplo:

```
VAR
    MATSUMA,MAT1,MAT2,RESUL,DATOX,DATOY : MATRIZ;
    DIM1,DIM2 : SUBINDICE;
SUMAMAT ( MATSUMA, MAT1, MAT2, DIM1, DIM2 );
SUMAMAT ( RESUL, DATOX, DATOY, 10, 8 );
```

Nótese que, puesto que solamente el parámetro formal de la primer matriz fué declarado por referencia, los demás argumentos pueden pasarse por valor, como en el último ejemplo. Por supuesto, todos los argumentos que representen matrices deberán ser simplemente el nombre de la matriz, ya que en el lenguaje Pascal no se permite ninguna expresión aritmética o de otro tipo que involucre un arreglo completo.

CAPITULO 9

TIPOS NO PREDEFINIDOS DE DATOS.

En el capítulo de tipos predefinidos de datos se vieron los tipos de datos que proporciona el lenguaje Pascal al programador (INTEGER, REAL, CHAR y BOOLEAN) para que éste resuelva sus problemas; sin embargo, algunas veces estos tipos de datos no cubren todos los requerimientos de un programador y no porque sean insuficientes, ya que dos tipos de cantidades numéricas, un tipo de caracteres y un tipo lógico son más que suficientes para resolver casi cualquier problema de computación, sino porque en repetidas ocasiones el programador necesita definir algunas otras características de sus datos que no son evidentes por el solo hecho de declarar un dato INTEGER o REAL, por ejemplo: qué rango de valores pueden tomar sus datos, cómo está relacionado un dato con otro, siguen sus datos un orden numérico natural (1,2,3, etc) o su programa requiere que sigan un orden distinto, etc.

Además de lo anterior, un programador puede desear que los valores que toman los datos tengan un sentido más claro para él; por ejemplo, supongamos que en un programa hay una variable UNIDADES tal que si vale 1 indica que las unidades están en centímetros y si vale 2 indica que están en pulgadas; por qué no hacer ésto más evidente para el programador y lograr que a la variable UNIDADES se le pueda asignar el valor 'CENTIMETROS' o 'PULGADAS' con el fin de evitarle la labor adicional al programador de tener que acordarse que el 1 es tal cosa y el 2 tal otra.

Para suplir estos requerimientos, el lenguaje Pascal cuenta con la posibilidad de que el programador defina sus propios tipos de datos y las propiedades de los mismos; por supuesto, esta operación no crea nada nuevo en el lenguaje, solo ocasiona que el manejo que antes hacía el programador para realizar en forma adecuada las operaciones anteriormente expuestas lo efectúe el lenguaje Pascal en forma automática.

Los nuevos 'tipos' de datos que puede definir el programador son cuatro y se llaman: escalares, subrangos, conjuntos (SET) y registros (RECORD); este capítulo trata solo de los tres primeros, ya que el tipo registro (RECORD) y el otro tipo predefinido de Pascal que es apuntador (POINTER) y que generalmente se usa junto con RECORD, están fuera del

alcance de este curso.

Como ya se vió anteriormente, para declarar las variables hay que poner el nombre de la variable seguida de su tipo; por lo tanto, si queramos declarar una variable de un tipo no predefinido por Pascal, previamente a la declaración de la variable tendremos que definir el nuevo tipo. Esto se hace en una sección de declaración de tipos que se coloca antes de la de declaración de variables y que comienza con la palabra reservada TYPE seguida por el nombre del nuevo tipo, un signo igual (=) y las características del nuevo tipo.

Esta sección de declaración de tipos ocasiona que se defina un nuevo tipo de datos y se le dé el nombre especificado por el programador, una vez hecho ésto podrá declararse una variable utilizando el nombre del nuevo tipo de datos o utilizarse en cualquier lugar en el que se pueda utilizar un tipo predefinido de datos de Pascal, por ejemplo:

```
TYPE
     NUEVO = <descripcion del nuevo tipo>;
VAR
     VARNUEVA : NUEVO;
```

9.1 ESCALARES.

Sucede a menudo que una variable toma únicamente un cierto número de valores perfectamente definidos dentro del rango de valores que la variable podría tomar; por ejemplo, suponsamos que se deseo utilizar un dato llamado DENOMINACION el cual representa los valores de las diversas denominaciones de monedas y billetes que existen, entonces ese dato sólo tomaría los valores siguientes (considerando que no exista la moneda fraccionaria o centavos):

1 5 10 20 50 100 500 1000 2000 5000 10000

Si quisiéramos definir una variable que va a contener el valor de una determinada moneda o billete, no sería adecuado declarar esa variable de tipo INTEGER o REAL, ya que entonces podría tomar cualquier valor que no represente una denominación existente; en este caso es más conveniente declarar un nuevo 'tipo' de variable que sólo pueda tomar los valores que nos interesan.

Puesto que esos valores van a constituir un tipo de datos que previamente no existía en Pascal, debemos definir perfectamente cuáles son esos valores para su adecuado proceso por parte del lenguaje Pascal, y qué es lo que representa cada uno de ellos para nuestra propia

conveniencia; ésto se hace por medio de identificadores que se encierran entre paréntesis y se separan por comas de la siguiente manera:

```
TYPE
      DENOMINACION = (UNO,CINCO,DIEZ,VEINTE,CINCUENTA,
                      CIEN,QUINIENTOS,MIL,DOSMIL,
                      CINCOMIL,DIEZMIL);
```

Estos valores son palabras que sólo tienen significado para nosotros y que no afectan al lenguaje Pascal de ninguna forma, son simplemente una facilidad que brinda el lenguaje para que podamos manejar cantidades con nombres más familiares para nosotros; veamos otro ejemplo:

```
TYPE
   DIA    = (DOMINGO,LUNES,MARTES,MIERCOLES,
               JUEVES,VIERNES,SABADO);
   PLANETA = (MERCURIO,VENUS,TIERRA,MARTE,JUPITER,
              SATURNO,URANO,PLUTON,NEPTUNO);
```

La declaración de un tipo escalar define cuales valores componen ese tipo, por lo tanto, a una variable de un tipo dado solamente podrán asignársele valores del mismo tipo (ésto es cierto para todos los tipos de Pascal, aún los predefinidos, con la única excepción de que a una variable de tipo REAL podrá asignársele un valor de tipo INTEGER); cualquier intento de asignar un valor de un tipo a una variable de otro ocasionará un error.

Un valor escalar (identificador o palabra) definido como componente de un tipo no puede ser también componente de otro, por ejemplo, una vez declarado el tipo DENOMINACION anterior no podrá utilizarse ninguno de sus valores (UNO,CINCO,etc.) para definir un nuevo tipo. Cada valor es exclusivamente de un tipo dado (ésto también se cumple con los tipos predefinidos de Pascal).

Además de definir los valores que lo componen, la declaración de un tipo también especifica el "orden natural" de esos valores, o sea, cual es el primer valor del tipo, cual es el último y cual es la posición relativa de todos los valores intermedios.

Por ejemplo, considerando la anterior declaración del tipo DENOMINACION podriamos tener lo siguiente:

```
VAR
    VALOR1, VALOR2 : DENOMINACION;
```

Lo anterior declararía las variables VALOR1 y VALOR2 del tipo DENOMINACION, entonces sería posible efectuar las siguientes asignaciones:

```
VALOR1 := VEINTE;
VALOR2 := MIL;
```

Además de la asignación de valores, las únicas operaciones que podemos efectuar con cantidades de tipos escalares son los operadores lógicos de relación (mayor que, igual a, etc); de acuerdo a lo anterior, las siguientes relaciones serían ciertas:

```
VALOR1 < VALOR2
VALOR1 > DIEZ
VALOR2 < CINCOMIL
```

Siempre que puede indicarse exactamente cual es el valor siguiente a un valor dado de cualquier tipo, se dice que el tipo es enumerado; todos los tipos de Pascal son enumerados, excepto REAL. Cuando se tienen valores de un tipo enumerado se pueden utilizar las funciones SUCC, PRED y ORD.

La función SUCC regresa el sucesor de un valor dado, o sea, el valor siguiente a un valor dado de acuerdo al orden natural del tipo del valor; por ejemplo:

```
SUCC(4)   regresa  5
SUCC(-3)  es igual a  -2
```

El último valor que compone un tipo dado no tiene sucesor. De acuerdo al tipo DENOMINACION anteriormente definido, tenemos:

```
SUCC(UNO)      es igual a  CINCO
SUCC(VALOR1)   vale  CINCUENTA
SUCC(DIEZMIL)  no existe (marcaría error)
```

La función PRED regresa el predecesor del valor dado, es decir, el valor inmediato anterior al valor dado; el primer valor de cualquier tipo no tiene predecesor. Por ejemplo:

```
PRED (8)        vale 7
PRED (-3)       regresa un -4
PRED (VALOR2)   es igual a QUINIENTOS
PRED (DIEZMIL)  vale CINCOMIL
PRED (UNO)      no está definido (marcaría error)
```

La función ORD proporciona el ordinal de un valor dado, o sea, la posición relativa de un valor dado dentro de los valores que componen un cierto tipo; el primer valor de cualquier tipo tiene un ordinal igual a cero, el siguiente tiene como ordinal 1 y así sucesivamente hasta que el último valor tiene un ordinal igual a N-1 en donde N es el número de valores que componen ese tipo.

A pesar de que el tipo INTEGER es enumerado, no es posible indicar con precisión cual es su primer valor, ya que el mismo depende de factores que están fuera del lenguaje Pascal, como el tipo de máquina que se esté usando, por lo tanto, la función ORD no puede aplicarse a valores del tipo INTEGER.

Para la definición anterior de DENOMINACION tenemos que:

```
ORD(UNO)          es igual a 0
ORD(VALOR1)       vale 3
ORD(QUINIENTOS)   regresa un 6
```

Un tipo escalar de datos también puede ser utilizado para declarar la dimensión de un arreglo, por ejemplo:

```
VAR
   COBRO : ARRAY [DENOMINACION] OF REAL;
```

Declararía un vector llamado COBRO que está formado por 11 elementos, cada uno de tipo REAL; el primer elemento es el subíndice UNO, el segundo elemento es el subíndice CINCO, el tercero el DIEZ, etc. Por supuesto, para poder tener acceso a los elementos de este vector por medio de una variable, la variable tendría que ser declarada del mismo tipo que la dimensión del arreglo, o sea, DENOMINACION. De acuerdo a los ejemplos anteriores, las variables VALOR1 y VALOR2 podrían servir de subíndices de este vector en la forma usual.

Además de las funciones antes indicadas, las variables de un tipo escalar pueden utilizarse junto con las instrucciones FOR y CASE de la siguiente manera:

Una instrucción FOR asignará a la variable índice el valor inicial indicado y pasará por cada uno de los valores componentes del tipo hasta llegar al valor final indicado en la instrucción FOR; por supuesto, la variable índice, el valor inicial y el valor final deben ser todos del mismo tipo, veamos:

```
FOR VALOR1 := UNO TO CIEN DO
```

Variará el valor de la variable VALOR1 haciéndolo igual a UNO, CINCO, DIEZ, VEINTE, CINCUENTA y CIEN.

```
VALOR2 := MIL;
FOR VALOR1 := VALOR2 TO DIEZMIL DO
```

Asignará a la variable VALOR1 los valores de MIL, DOSMIL, CINCOMIL y DIEZMIL secuencialmente.

En una instrucción CASE puede probarse el valor de una variable de tipo escalar como selectora del caso y utilizarse los valores de ese tipo como etiquetas del caso; veamos un ejemplo.

Supongamos que tenemos una pequeña tienda que solamente vende cuatro artículos: leche, refrescos, ron del país y cognac importado; los precios de cada artículo son: $65, $36, $950 y $4500 respectivamente; la leche no paga impuesto, los refrescos pagan el 6 % de impuesto, el ron del país paga 15 % y el cognac importado 20 %.

Las siguientes secciones de un programa ilustran el uso de los conceptos mencionados:

```
TYPE
     ARTICULO = (LECHE,REFRESCO,RON,COGNAC);
VAR
     QUECOMPRO                          : ARTICULO;
     CANTIDAD,PRECIO,PORCENTAJE,
     SUBTOTAL,IMPUESTO,TOTAL            : REAL;
. . .
. . .
CASE QUECOMPRO OF
     LECHE:
         BEGIN
         PRECIO:=65;
         PORCENTAJE:=0.00
         END
     REFRESCO:
         BEGIN
         PRECIO:=36;
         PORCENTAJE:=0.06
         END
     RON:
         BEGIN
         PRECIO:=950;
         PORCENTAJE:=0.15
         END
     COGNAC:
         BEGIN
         PRECIO:=4500;
         PORCENTAJE:=0.20
         END
END
SUBTOTAL:=CANTIDAD*PRECIO;
IMPUESTO:=SUBTOTAL*PORCENTAJE
TOTAL:=SUBTOTAL+IMPUESTO;
```

Es muy importante recordar que las palabras que forman los valores de un tipo escalar no tienen absolutamente ningún significado para el lenguaje; el significado se lo damos nosotros al usarlas en forma adecuada. Dicho de otra forma, el anterior segmento de programa funcionaría de manera exactamente igual si nosotros declaráramos a la variable QUECOMPRO de tipo INTEGER y entonces siguiéramos un estándar definido por nosotros mismos, por ejemplo: si QUECOMPRO vale 1, se trata de leche, si QUECOMPRO vale 2 es refresco, etc., y entonces en vez de usar las palabras LECHE, REFRESCO, etc. usaríamos las cantidades 1, 2, etc.

Por medio de los tipos escalares el lenguaje Pascal nos brinda la facilidad de incluir nuestro estándar en el nombre mismo de los valores y del tipo de datos que estamos definiendo, pero a cambio de ello nos restringe el uso de esas cantidades a unas cuantas operaciones solamente.

Los escalares son muy útiles al ayudar al programador a comprender que es lo que está haciendo su programa, por lo que apoyan grandemente en la detección de errores y el mantenimiento de programas; además, la restricción en el uso de los tipos escalares siempre es posible eliminar transformando el valor de una cantidad de un tipo escalar a una de tipo INTEGER por medio de la función ORD y una vez teniendo este valor INTEGER, cualquier manipulación sobre él es posible.

9.2 SUBRANGOS.

En la mayoría de los programas de aplicación se utilizan en numerosas ocasiones, variables que pueden tomar todos los valores de un tipo de datos dado, pero limitadas a un cierto intervalo (ó subrango) del rango total de valores de ese tipo.

Un ejemplo muy claro de esto son las variables que se utilizan como subíndices para tener acceso a los elementos de un arreglo; por ejemplo, si se tiene definido un vector con elementos numerados desde el 1 hasta el 20, no sería conveniente, y hasta podría ocasionar errores, que la variable que va a servir de subíndice de ese vector fuera declarada de tipo INTEGER, ya que en este caso podría tomar cualquier valor que no estuviera en el intervalo de 1 a 20.

El lenguaje Pascal nos brinda la facilidad de definir un tipo de datos que se componga de un subrango de valores de otro tipo, de tal forma que el lenguaje revise de manera automática los valores asignados a una variable de tipo subrango y detecte cualquier intento de asignarle un valor que esté fuera de su rango.

Todos los tipos de datos subrango están referidos a un tipo de datos definido con anterioridad el cual se llama el tipo base del subrango. Por ejemplo, para definir un tipo de datos subrango del tipo base INTEGER y llamado SUBINDICE el cual sólo podrá tomar los valores INTEGER del 1 al 20 inclusive, se usaría la siguiente declaración de tipo:

```
TYPE
    SUBINDICE = 1..20;
```

La sintaxis para la declaración del subrango consiste en poner el valor inicial del subrango, poner dos veces un punto y finalmente el valor final del subrango; estos dos valores deben pertenecer a un mismo tipo de datos, el cual es el tipo base del subrango y además el valor inicial debe estar primero que el valor final de acuerdo al orden natural de valores del tipo base del subrango.

Una vez definido este tipo de datos se puede utilizar en cualquier lugar en el que se pueda utilizar un tipo predefinido de Pascal, pero los subrangos son más comunmente utilizados para declarar la dimensión de un arreglo y para declarar las variables que servirán de subíndices para tener acceso a los elementos de ese arreglo, por ejemplo:

```
VAR
    VECTOR : ARRAY [SUBINDICE] OF REAL;
    I,J    : SUBINDICE;
```

La mayoría de las veces que se utilizan variables INTEGER en un programa, es para usarlas de una u otra forma como subíndices, por lo que los subrangos son grandemente utilizados. más aún, cuando no se utiliza una variable INTEGER como subíndice, casi siempre se tiene una definición del rango de valores que la variable tomará en un programa e inclusive se ponen pruebas para revisar el valor de esa variable a fin de detectar que no se haya salido del rango establecido. La idea del tipo subrango es quitarle toda esa labor al programador y pasársela al lenguaje Pascal y tan es así, que las variables de tipo INTEGER sólo deberían ser utilizadas cuando el programador no tiene la más mínima idea de cuáles son los valores que podrá tomar una variable, lo cual ocurre en muy raras ocasiones.

Los subrangos siempre deben definirse a partir de un tipo base que sea enumerado, por lo que no es posible definir subrangos del tipo REAL. Por otro lado, el tipo predefinido CHAR sí es enumerado, por lo que es posible definir subrangos de CHAR los cuales pueden facilitar en mucho la programación, ya que pueden ser utilizados, por ejemplo, como dimensión de vectores los cuales tendrán como subíndices los caracteres mismos, por los que ciertos programas de manejo de textos y caracteres se pueden facilitar.

Sin embargo, hay que tener presente que los componentes del subrango dependen del orden natural de valores del tipo CHAR, y éste a su vez depende del código interno que maneje la computadora anfitrión, por lo que un mismo subrango de caracteres podría no tener los mismos elementos ni estar en el mismo orden si se transportara el programa de una computadora a otra.

Como los tipos escalares de datos vistos anteriormente también son enumerados, también pueden ser utilizados para declarar subrangos de ellos. Por ejemplo, a partir de la declaración del tipo escalar DENOMINACION vista anteriormente y que es la siguiente:

```
TYPE
    DENOMINACION = (UNO,CINCO,DIEZ,VEINTE,CINCUENTA,
                    CIEN,QUINIENTOS,MIL,DOSMIL,
                    CINCOMIL,DIEZMIL);
```

Podrían declararse los siguientes subrangos:

```
TYPE
    MONEDA  = UNO..CINCUENTA;
    BILLETE = CINCUENTA..DIEZMIL;
```

El tipo subrango MONEDA tiene como componentes los valores UNO, CINCO, DIEZ, VEINTE y CINCUENTA, mientras que el tipo subrango BILLETE está formado por los valores CINCUENTA, CIEN, QUINIENTOS, MIL, DOSMIL, CINCOMIL y DIEZMIL; como vemos, es completamente válido que se definan dos o mas subrangos que se traslapen, es decir, que compartan uno o más valores del tipo base.

En los dos subrangos declarados anteriormente, el tipo base es DENOMINACION y debe estar definido antes de la declaración de los subrangos; en seguida se muestran otros ejemplos tomando como base los tipos escalares DIA y PLANETA mostrados en la descripción antes vista de tipos escalares.

```
TYPE
    DIALABORABLE     = LUNES..VIERNES;
    INTERIORPLANETA  = MERCURIO..MARTE;
    EXTERIORPLANETA  = JUPITER..NEPTUNO;
```

Las operaciones con variables de tipo subrango son las mismas que las del correspondiente tipo base, es decir, si el tipo base es INTEGER, a la variable de subrango se le podrán aplicar todas las operaciones definidas para enteras y podrá combinarse con otras cantidades que de una u otra forma pertenezcan al tipo INTEGER, ya porque sean declaradas INTEGER directamente, o porque sean subrangos de INTEGER.

En el caso de que el tipo base sea un escalar, sólo se podrán aplicar los operandos definidos para escalares (operadores lógicos de relación) combinando las cantidades de tipo subrango con otras que de una u otra forma sean del mismo tipo base escalar.

Por supuesto, no importa de que tipo base sea el subrango, cualquier intento de asignarle un valor que esté fuera del rango de valores definido para ese tipo provocará un error en la ejecución del programa.

9.3 CONJUNTOS

El lenguaje Pascal tiene la facilidad de definir un tipo de datos que sea un conjunto de elementos al cual se le pueden aplicar ciertos operadores bien definidos y que funciona de manera enteramente similar a la que dictan las reglas del álgebra de conjuntos.

Tratando de explicar esas reglas del álgebra de conjuntos en términos del lenguaje de programación Pascal, tendríamos los siguientes conceptos:

Un tipo de datos SET (conjunto) es uno tal que las variables que sean declaradas de ese tipo son capaces de contener uno, varios, todos o ninguno de los valores del tipo base al cual está asociado el conjunto, pero que nunca pueden contener dos veces un mismo valor dado. Para declarar un tipo SET es necesario indicar de que tipo van a ser los valores que puedan contener las variables del tipo SET; por ejemplo, podríamos tener un tipo de datos SET llamado NUMEROS que sea un conjunto de valores de tipo REAL, por ejemplo:

```
TYPE
    NUMEROS = SET OF REAL;
```

Una vez declarado este nuevo tipo de datos, podríamos declarar variables de ese tipo, por ejemplo:

```
VAR
    CONJ1 : NUMEROS;
```

La variable CONJ1 es de tipo NUMEROS, el tipo NUMEROS es un SET (conjunto) de valores tipo REAL; de esta manera, la variable CONJ1 es capaz de contener un valor de tipo REAL, o varios, o todos, o ninguno de los valores de tipo REAL; cada uno de estos estados de la variable CONJ1 representa sus valores; cada valor de tipo REAL que esté contenido en la variable CONJ1 se dice que es miembro del conjunto contenido en CONJ1.

Es como si la variable CONJ1 fuera un vector que tuviera "casilleros" (elementos) suficientes para acomodar a todos y cada uno de los valores de tipo REAL que existen en el lenguaje; de esta manera, si todos los casilleros estuvieran vacíos, se diría que la variable CONJ1 contiene un conjunto vacío y ese sería el valor de la variable en ese momento, si la variable tuviera ocupados los casilleros correspondientes a los valores 1.5, 17.38 y 24.0 se diría que es un conjunto de tres miembros y que los miembros son las tres cantidades de tipo REAL 1.5, 17.38 y 24.0, y ese estado de la variable (el contener precisamente a esos tres valores) es

otro valor de la variable de tipo SLI; después podría contener a otros 7 valores diferentes y sería otro valor de la variable SET (ó otro conjunto distinto), etc.

Puesto que la variable CONJ1 es un conjunto de valores de tipo REAL, puede contener a todos los valores de tipo REAL que existen; por lo tanto, cualquier valor de tipo REAL que se pueda definir en cualquier momento en el lenguaje o bien está contenido en el conjunto (su casillero correspondiente está ocupado) o no lo está (su casillero está vacío) y no hay más posibilidades.

Una vez que una variable de tipo SET contiene un cierto elemento de su tipo base (cuando un casillero está ocupado) no puede contenerlo otra vez (el casillero ya está ocupado por ese valor), lo más que puede hacer es contener a más valores además de los que ya tenía (ocupar más casilleros vacíos) o contener a menos valores de los que tenía (desocupar casilleros ya ocupados) o contener a un conjunto diferente de valores (ocupar algunos casilleros y desocupar otros); cada conjunto diferente de valores (cada combinación diferente de casilleros vacíos y llenos) representa un valor diferente de la variable de tipo SET.

Obviamente el número de valores que puede contener una variable de tipo SET es mucho mayor que el número de valores que contiene su tipo base de datos; para ser exactos es 2 elevado a la N en donde N es el número de valores que tiene el tipo base del conjunto.

La declaración de un tipo SET requiere siempre que se especifique el tipo de valores que va a contener el conjunto, es decir, el tipo base del conjunto; este tipo base puede ser cualquiera de los tipos predefinidos de Pascal (INTEGER, REAL, CHAR y BOOLEAN) o cualquier tipo escalar definido por el usuario.

Veamos como se definiría un SET de valores escalares y las operaciones que pueden aplicarse a cualquier tipo de variable de tipo SET:

```
TYPE
    INGREDIENTES = (LECHE,HARINA,HUEVO,AZUCAR,
                    MANTEQUILLA,FRUTA);
    POSTRE       = SET OF INGREDIENTES;
VAR
    LICUADO,
    HOTCAKE,
    PASTEL,
    SOPA         : POSTRE;
```

INGREDIENTES es un tipo escalar cuyos valores son LECHE, HARINA, etc.; POSTRE es un tipo SET (conjunto) de los valores que componen el tipo INGREDIENTES; finalmente, LICUADO, HOTCAKE, PASTEL y SOPA son variables de tipo POSTRE, o sea, conjuntos de los valores que componen el tipo escalar INGREDIENTES.

Para asignar un valor a cualquier variable de un tipo SET necesitamos una forma de escribir un conjunto, esa forma es encerrando los miembros del conjunto entre paréntesis cuadrados y separándolos entre sí por medio de comas:

```
LICUADO := [LECHE,AZUCAR,FRUTA];
HOTCAKE := [LECHE,HARINA,HUEVO];
PASTEL  := [LECHE,HARINA,HUEVO,AZUCAR,MANTEQUILLA];
```

Dos paréntesis cuadrados que no encierran ningún miembro constituyen un conjunto vacío, por ejemplo:

```
SOPA := [ ];
```

Si los miembros de un conjunto son valores que estan en el mismo orden natural con que fueron declarados en su correspondiente tipo base escalar, entonces se puede utilizar la misma notación que en los subrangos:

```
HOTCAKE := [LECHE..HUEVO];
PASTEL  := [LECHE..MANTEQUILLA];
```

Los operadores lógicos de relación pueden utilizarse con conjuntos; los operadores = (igualdad) y <> (desigualdad) preguntan si dos conjuntos tienen o no exactamente los mismos miembros:

```
HOTCAKE = [LECHE,HARINA,HUEVO]   es verdadero
PASTEL  = LICUADO    es falso
SOPA    <> LICUADO    es verdadero
```

El operador <= (esta contenido en) pregunta si todos los miembros del conjunto de la izquierda también los contiene el conjunto de la derecha (aunque este último contenga además otros miembros), por ejemplo:

```
HOTCAKE <= PASTEL      es verdadero
LICUADO <= PASTEL      es falso
```

El operador >= (contiene) es similar al operador anterior:

```
[LECHE..FRUTA] >= LICUADO      es verdadero
SOPA >= HOTCAKE                es falso
```

El operador '+' aplicado a conjuntos indica la unión de conjuntos y da como resultado un conjunto tal que contiene a todos los miembros de ambos conjuntos:

```
HOTCAKE+LICUADO = [LECHE,HARINA,AZUCAR,HUEVO,FRUTA]
```

El operador '*' indica la intersección de conjuntos y da como resultado un conjunto tal que contiene solamente aquellos miembros que estén contenidos en los dos conjuntos:

```
HOTCAKE*LICUADO = [LECHE]
```

Finalmente, la palabra reservada IN (pertenencia) pregunta si un determinado valor del mismo tipo que el tipo base del conjunto está contenido en un conjunto dado, por ejemplo:

```
FRUTA IN LICUADO       es verdadero
HARINA IN [LECHE..HUEVO]   es verdadero
MANTEQUILLA IN HOTCAKE     es falso
```

Hay que notar que las operaciones sobre conjuntos anteriormente descritas nos permiten formar conjuntos, formar la unión e intersección de conjuntos, preguntar si dos conjuntos son iguales, si un conjunto está contenido en otro o si un miembro dado está contenido en un conjunto, pero no nos permiten extraer los miembros de un conjunto una vez que ha sido formado.

Dicho de otro modo, no hay manera de conocer en forma directa los miembros que tiene un conjunto en un momento dado; para hacer esta operación es necesario generar todos los valores del tipo base del conjunto y para cada uno de ellos preguntar si está o no contenido en el conjunto, sólo de esta manera es posible conocer lo que tiene almacenado un conjunto.

Los conjuntos son una herramienta poderosa del lenguaje Pascal, por medio de ellos es posible evitar un trabajo de programación que de otra forma sería mayor; por ejemplo, en varias aplicaciones de manejo de caracteres es necesaio saber si un determinado caracter posiblemente leído de una tarjeta de datos es un dígito o una letra.

Para hacer está pregunta sin utilizar conjuntos sería necesario escribir:

```
IF CARACTER >= '0' AND CARACTER <= '9' THEN
    (*CARACTER ES UN DIGITO*)

IF CARACTER >= 'A' AND CARACTER <= 'Z' THEN
    (*CARACTER ES UNA LETRA*)
```

Utilizando conjuntos, además de hacer estas preguntas en forma más sencilla, también es mucho más evidente la idea de la pregunta cuando se hacen revisiones posteriores del programa, veamos:

```
TYPE
    CARACTERES = SET OF CHAR;
VAR
    NUMEROS : CARACTERES;
    LETRAS  : CARACTERES;

LETRAS  := ['A'..'Z'];
NUMEROS := ['0'..'9'];

IF CARACTER IN NUMEROS THEN
    (*CARACTER ES UN DIGITO*)

IF CARACTER IN LETRAS THEN
    (*CARACTER ES UNA LETRA*)
```

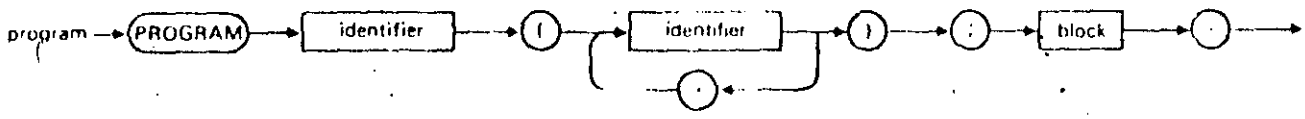
Por supuesto, y dependiendo de la aplicación, las variables de tipo SET pueden tener un uso muy grande y aliviar en mucho el trabajo del programador.

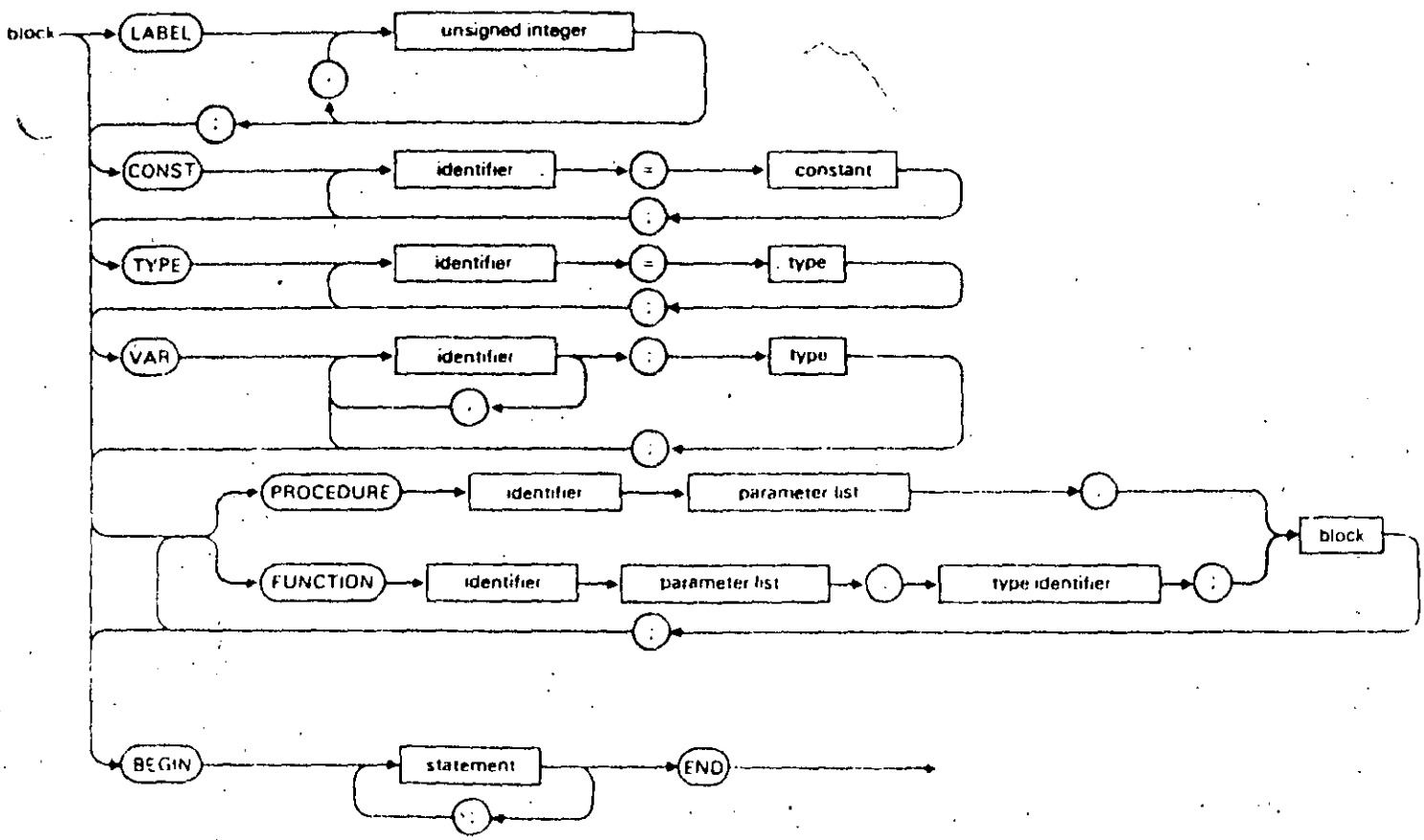
APENDICE A

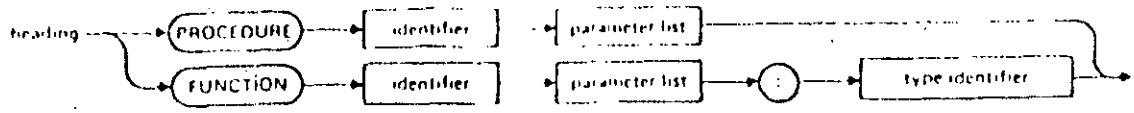
DIAGRAMAS DE SINTAXIS

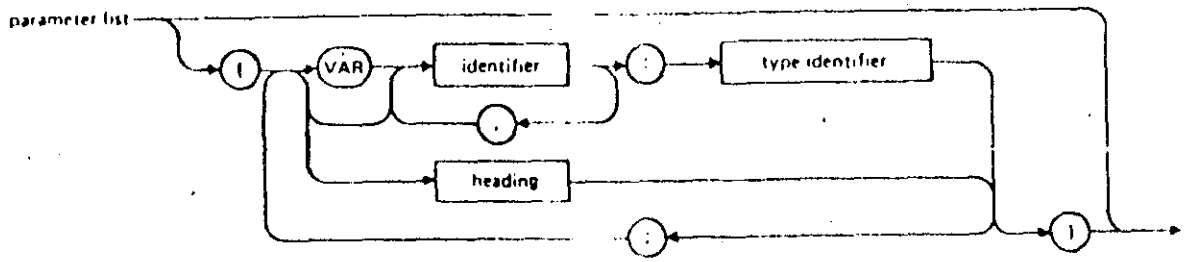
APENDICE A

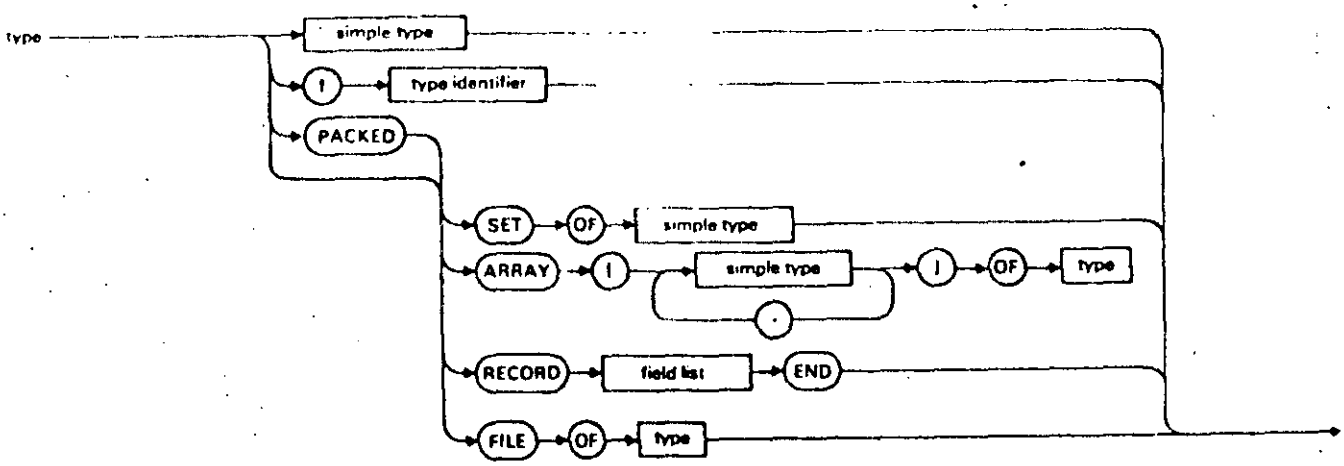
DIAGRAMAS DE SINTAXIS

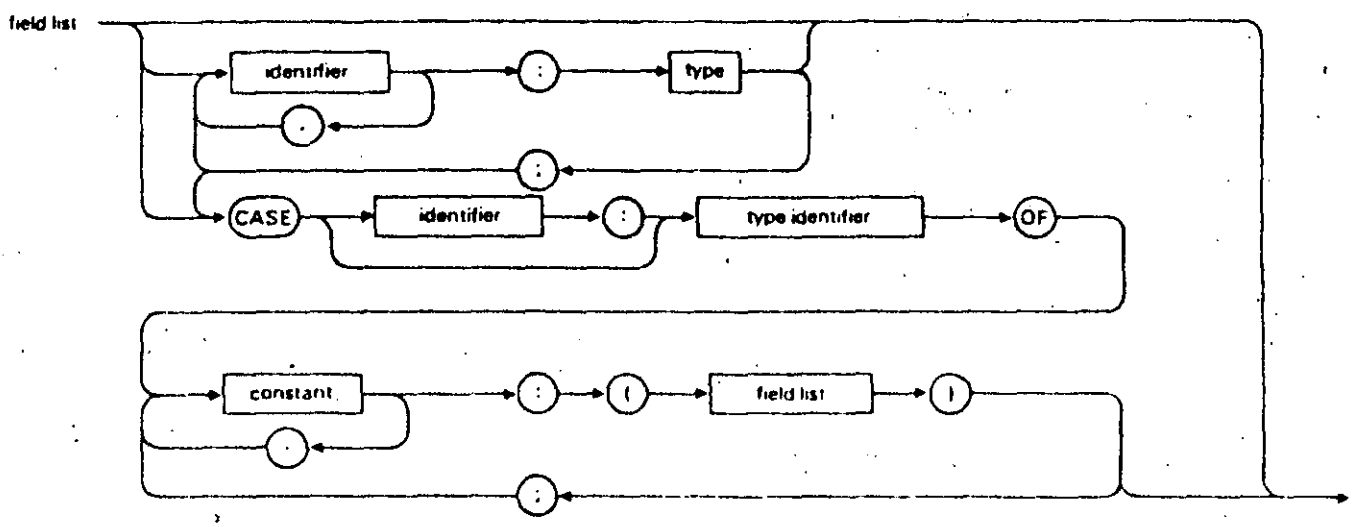


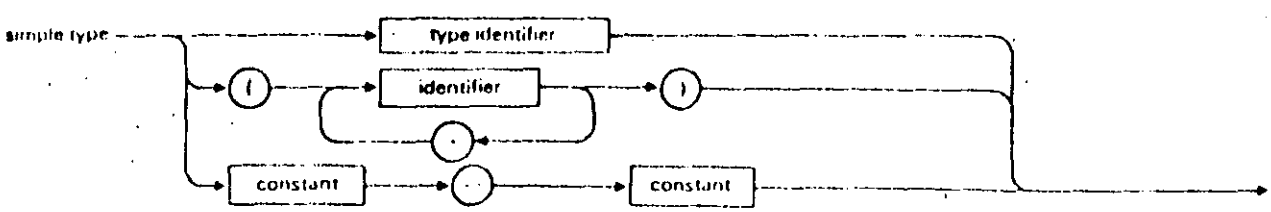


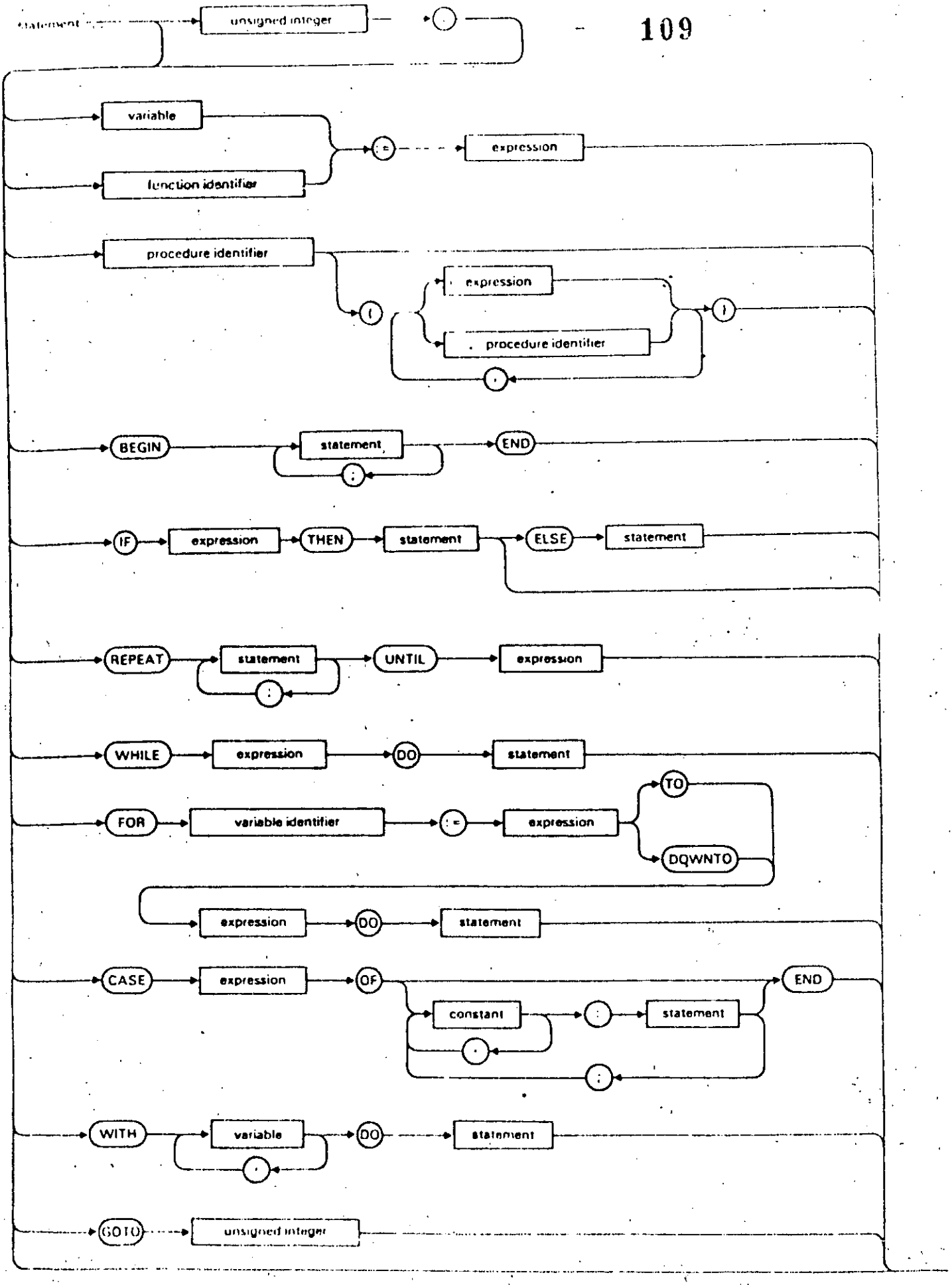

heading
PROCEDURE
identifier
parameter list
FUNCTION
identifier
parameter list
:
type identifier

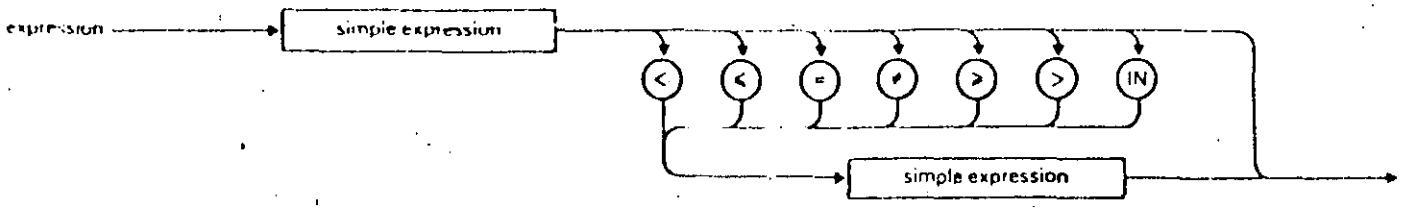

parameter list
(
VAR
identifier
,
:
type identifier
heading
;
)

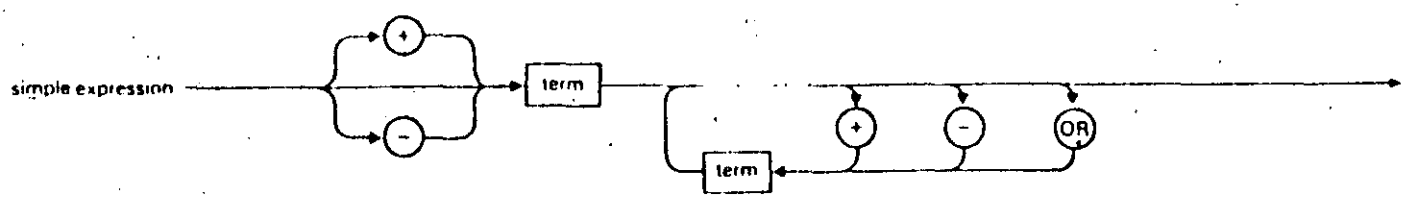

type
simple type
(
type identifier
PACKED
SET
OF
simple type
ARRAY
(
simple type
,
)
OF
type
RECORD
field list
END
FILE
OF
type

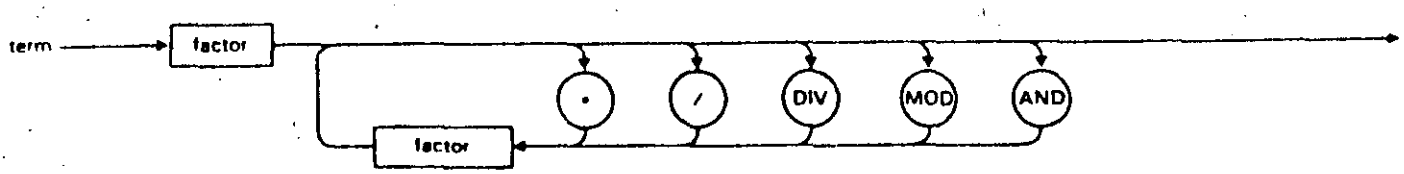

field list
identifier
,
:
type
;
CASE
identifier
:
type identifier
OF
constant
,
:
(
field list
)
;

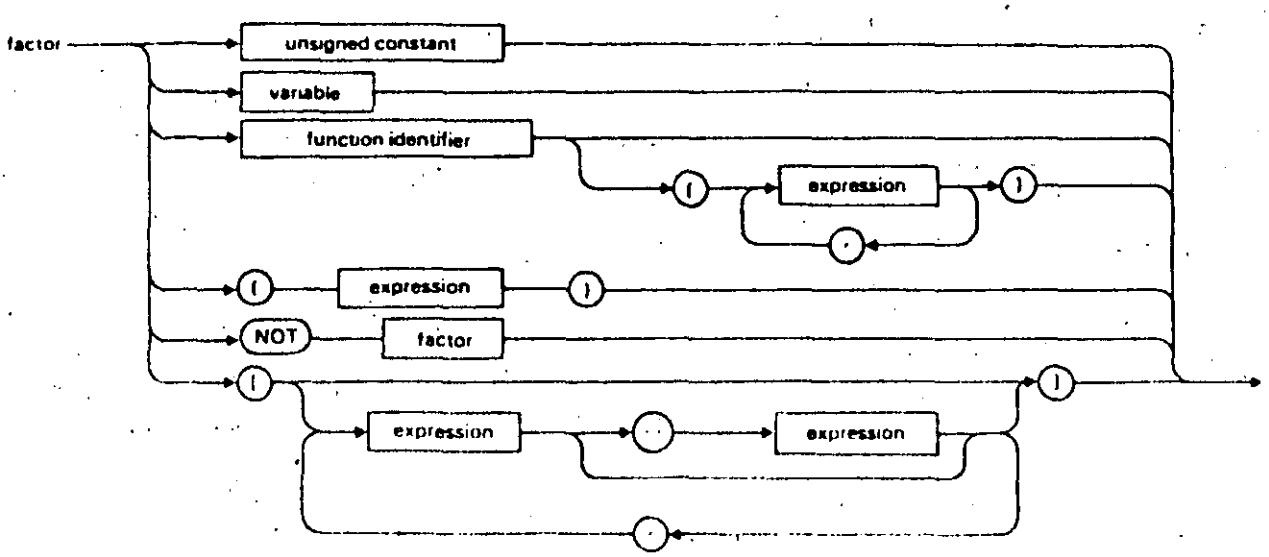

simple type
type identifier
(
identifier
,
)
constant
..
constant

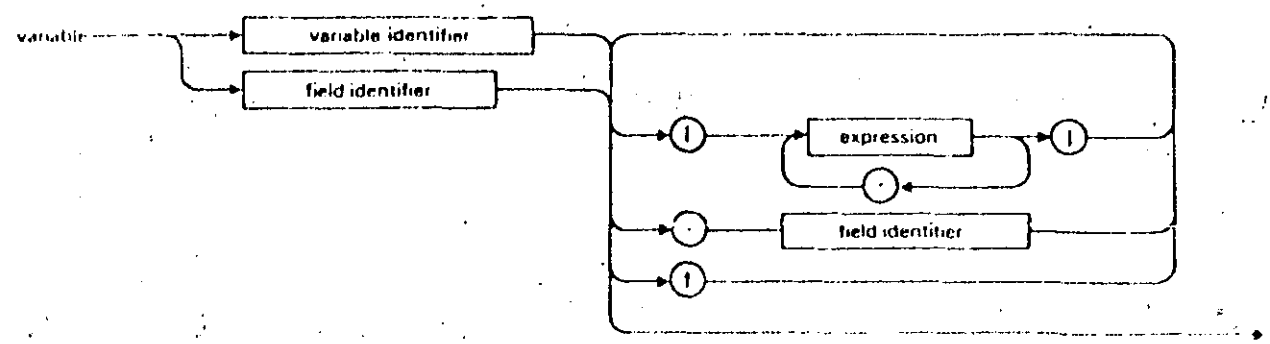

statement
unsigned integer
variable
function identifier
:=
expression
procedure identifier
(
expression
procedure identifier
)
BEGIN
statement
END
IF
expression
THEN
statement
ELSE
statement
REPEAT
statement
UNTIL
expression
WHILE
expression
DO
statement
FOR
variable identifier
:=
expression
TO
DOWNTO
expression
DO
statement
CASE
expression
OF
constant
statement
END
WITH
variable
DO
statement
GOTO
unsigned integer

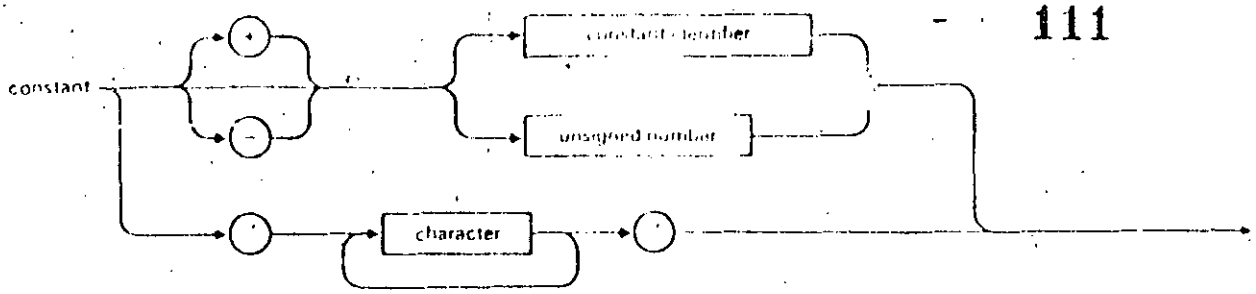

expression
simple expression
<
<
=
≠
>
>
IN
simple expression

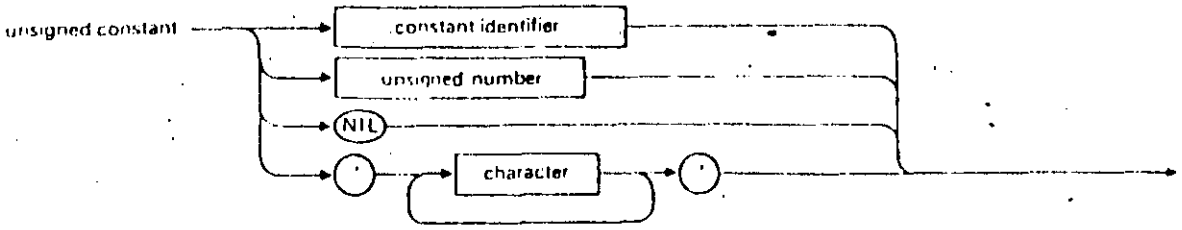

simple expression
+
-
term
+
-
OR
term

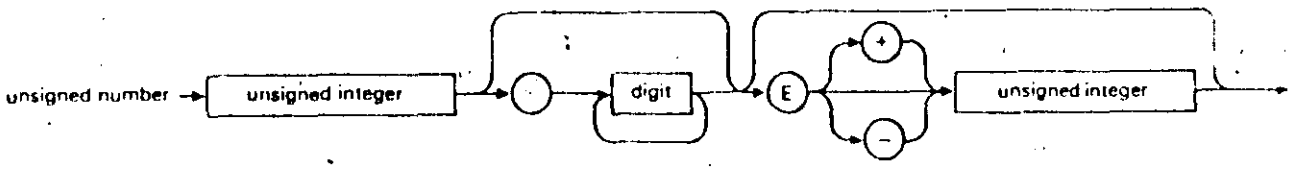

term
factor
*
/
DIV
MOD
AND
factor

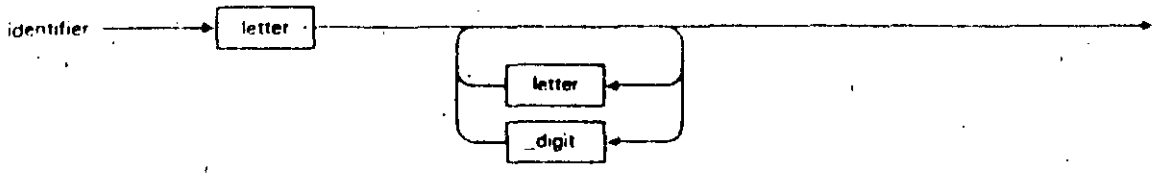

factor
unsigned constant
variable
function identifier
(
expression
)
,
(
expression
)
NOT
factor
(
expression
..
expression
)
,


variable
variable identifier
field identifier
(
expression
)
,
.
field identifier
↑

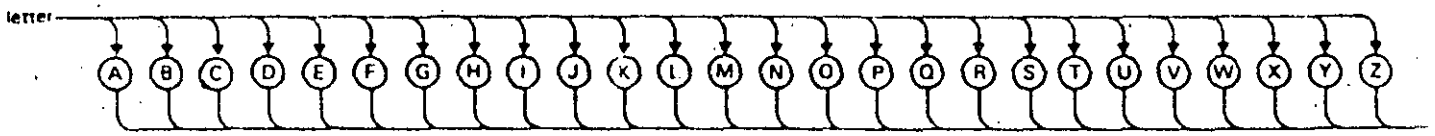

constant
+
-
constant identifier
unsigned number
'
character
'

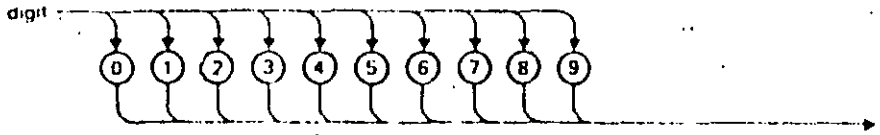

unsigned constant
constant identifier
unsigned number
NIL
'
character
'


unsigned number
unsigned integer
.
digit
E
+
-
unsigned integer


identifier
letter
letter
_digit

unsigned integer
digit


letter
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z


digit
0 1 2 3 4 5 6 7 8 9

APENDICE B

INTRODUCCION AL SISTEMA VAX-11/780

A continuación presentamos un breve resúmen de los comandos necesarios para editar, compilar y correr un programa.

B.1 COMO ENTRAR AL SISTEMA.

Oprima la tecla < RETURN >

El sistema responderá:

Que tengas una agradable sesion

Username:

Responda con el nombre de la clave que le fue asignada.

A continuación el sistema pedirá:

Password:

Teclee la clave secreta que le fue asignada a su clave.

N O T A : Al teclear el PASSWORD éste NO aparecerá en la pantalla.

El sistema verificará la validez de su clave y, en caso de aceptarla, responderá:

Bienvenidos al Sistema VAX/VMS V3.3 (C e c a f i)

8-SEP-1984 11:15:08.29

BUENAS TARDES

B.2 COMO SALIR DEL SISTEMA.

Para poder salir de sesión, bastará con dar el comando

$ LOG

B.3 COMO VER LOS ARCHIVOS ALMACENADOS EN LA CLAVE.

Teclee el comando

$ DIRECTORY

Este comando nos permitirá ver los archivos almacenados en la clave. Estos aparecerán en orden alfabético y de izquierda a derecha.

B.4 COMO BORRAR UN ARCHIVO.

Para poder borrar un archivo que en un momento dado ya no interese, teclee el comando:

$ DELETE < nombre de archivo >

B.5 COMO SACAR UN LISTADO DE UN PROGRAMA.

Para poder imprimir un programa fuente o un archivo de resultados teclee el comando:

$ PRINT < nombre de archivo

B.6 COMO CREAR UN NUEVO ARCHIVO Y COMO HACERLE MODIFICACIONES

A UNO YA EXISTENTE.

Para poder crear o modificar un archivo, deberá dar el comando:

$ EDIT < nombre de archivo >

Si el archivo no existe, lo creará como nuevo; si el archivo existe, lo 'traerá' como archivo de trabajo y lo podremos modificar.

B.7 COMO COMPILAR, LIGAR Y CORRER UN PROGRAMA.

Para poder ejecutar un programa, es necesario primero traducirlo a un lenguaje que entienda la máquina (lenguaje binario), posteriormente "pegarle" algunas rutinas del sistema y luego "cargarlo" a memoria para que se ejecute.

Esto lo lograremos con el comando:

$ COLIGO

Al dar este comando, el sistema responderá:

PROGRAMA:

A lo cual el usuario indicará el nombre del programa que desea compilar, ligar y correr.

Además el sistema pide el lenguaje en el que está escrito el programa, en nuestro caso PASCAL.

LENGUAJE: PASCAL

B.8 COMANDOS DE EDICION.

Para poder modificar un programa es necesario conocer algunos comandos del editor.

Para invocar al editor se debe dar el comando:

$ EDIT

Con esto el sistema responderá:

*

[EOB]

El asterisco (*) le informa al usuario que está en modo de edición.

[EOB] es una marca que indica el final del archivo.

Para poder insertar un programa es necesario dar el comando:

* INSERT

Y a continuación teclear nuestro programa.

Cuando terminemos, se deberá teclear simultáneamente la tecla CTRL y la Z (CTRL/Z)

Para poder ver lo que hemos insertado se hace con el comando:

* TYPE WHOLE

Para poder cambiar una letra, palabra o palabras de una línea, teclee el comando:

* SUBSTITUTE/TEXTO VIEJO/TEXTO NUEVO/rango de líneas

Para poder borrar una línea, teclee el comando:

* DELETE rango de líneas

Para poder guardar en disco el programa tecleado, teclee el comando:

* EXIT

B.9 COMO MANDAR IMPRIMIR A PAPEL LOS RESULTADOS DE UN PROGRAMA.

Supongamos que el programa se llama PRUEBA.PAS

Realice la siguiente secuencia:

```
$ ASIGNA
$ RUN PRUEBA
$ DEASIGNA
$ PRINT PRUEBA.PAS, RES.LIS
```

APENDICE C

PROGRAMACION ESTRUCTURADA

La última década ha visto aparecer un nuevo método de programación: ese método es conocido como la programación estructurada.

La programación estructurada no es otra cosa que un método de construcción de programas en el cual el rigor y la estructura reemplazan a la programación intuitiva y desorganizada.

La estructura de un programa está determinada por las construcciones que se han usado para dirigir el flujo de control.

El flujo de control en un programa es el orden en el cual se deben ejecutar las instrucciones del programa. E. Yourdon en su libro "Diseño Estructurado" da una definición formal de lo que es un programa de computadora.

Un programa puede ser definido como: "Una precisa y ordenada secuencia de instrucciones y agregados de instrucciones los cuales en total, definen, describen, dirigen o caracterizan la realización de alguna tarea".

Es importante recordar que mientras se está leyendo el listado del programa de arriba hacia abajo, la ejecución del mismo se puede llevar de una manera muy diferente.

Uno de los objetivos de la programación estructurada es que trata que el flujo de control se realice en forma tal que la secuencia de ejecución sea muy similar a la secuencia de lectura del programa. Esto impone al programador una disciplina rigurosa en términos de las estructuras que puede utilizar y además, en la forma en que éstas pueden ser usadas de acuerdo con la letra de la ley, cualquier programa escrito que utiliza exclusivamente tales estructuras es, por definición, un programa estructurado. Por desgracia, los malos programas pueden escribirse utilizando cualquier técnica. Es mucho más importante estar de acuerdo no con la letra, sino con el espíritu de la ley, si se trata

de estructurar un programa.

C.1 TEOREMA DE LA ESTRUCTURA

Cualquier problema susceptible de ser representado por un programa de computadora, se puede resolver usando las siguientes reglas:

1. Utilizar solo las figuras lógicas básicas:

   -SECUENCIA

   -IF THEN ELSE

   -DO WHILE

2. Es permisible anidar unas dentro de otras.

3. Es permisible la combinación de ellas.

4. Tiene solo una entrada y una sola salida o terminación.

5. No debe romper la secuencia de ejecución.

6. Debe ser claro para su legibilidad.

En la actualidad existen dos figuras lógicas que se considera se han integrado a las figuras lógicas básicas y son:

-REPEAT UNTIL

-CASE OF

A los lenguajes de programación que soportan estas estructuras se les conoce como lenguajes estructurados, siendo el lenguaje Pascal uno de estos lenguajes junto con Algol, Fortran 77, C y PL/1.

SECUENCIA: Es una instrucción o conjunto de instrucciones que no modifican o afectan el flujo de control de la ejecución del programa.

El diagrama que representa la figura lógica SECUENCIA es un rectángulo.

```
        |
        |
        |
        V
  -------------
  | SECUENCIA |
  -------------
        |
        |
        |
        V
```

EJEMPLO:

```
READ (variable, X);
WRITELN ('Este es simplemente un comentario');
a:=10;
x:=cos(pi);
```

La ejecución de estas instrucciones se realiza en forma secuencial, ésto es, primero se ejecuta la instrucción READ, luego WRITELN y así secuencialmente hasta ejecutar la última asignación, por lo tanto se considera como una secuencia o conjunto de secuencias.

IF-THEN-ELSE: Es una instrucción que toma una decisión en base a una condición, el resultado de esta condición solo puede ser 'verdadero' o 'falso', al poder tener solamente estos dos valores diferentes, se le conoce al resultado, como resultado booleano en honor al francés George Boole que escribió las reglas del álgebra que llevan su nombre.

El formato es el siguiente:

```
IF <condición> THEN
   SECUENCIA 1 <--- {resultó verdadera la condición}
ELSE
   SECUENCIA 2 <--- {resultó falsa la condición}
ENDIF
```

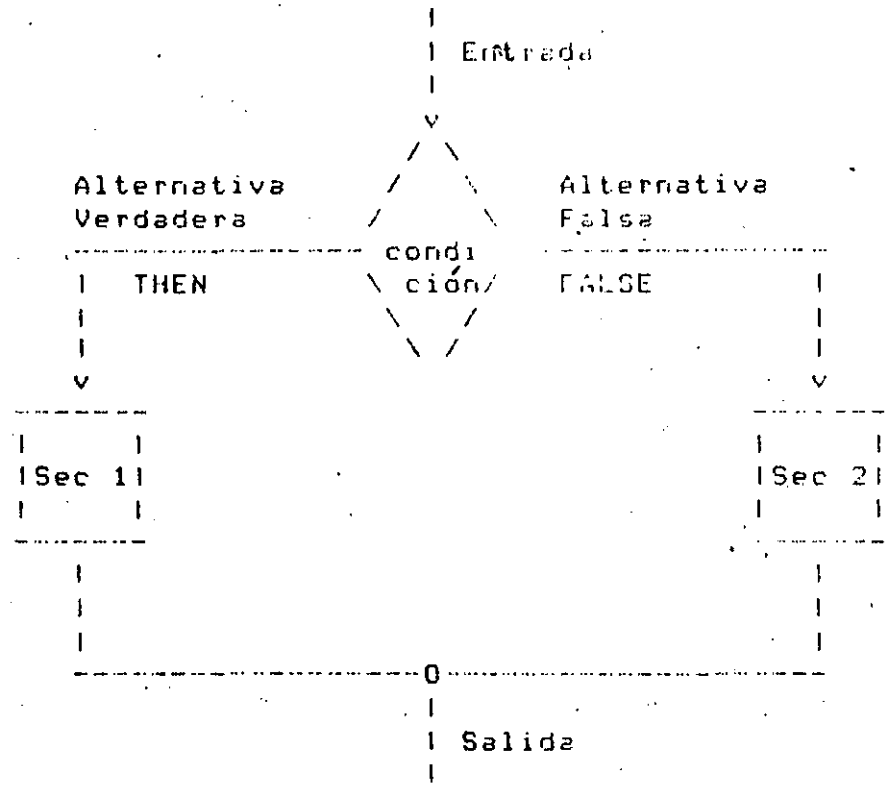
Algunos ejemplos de la condición pueden ser:

```
4 <= 7   verdadero
4 > 7    falso
4 = 7    falso
4 <> 7   verdadero
```

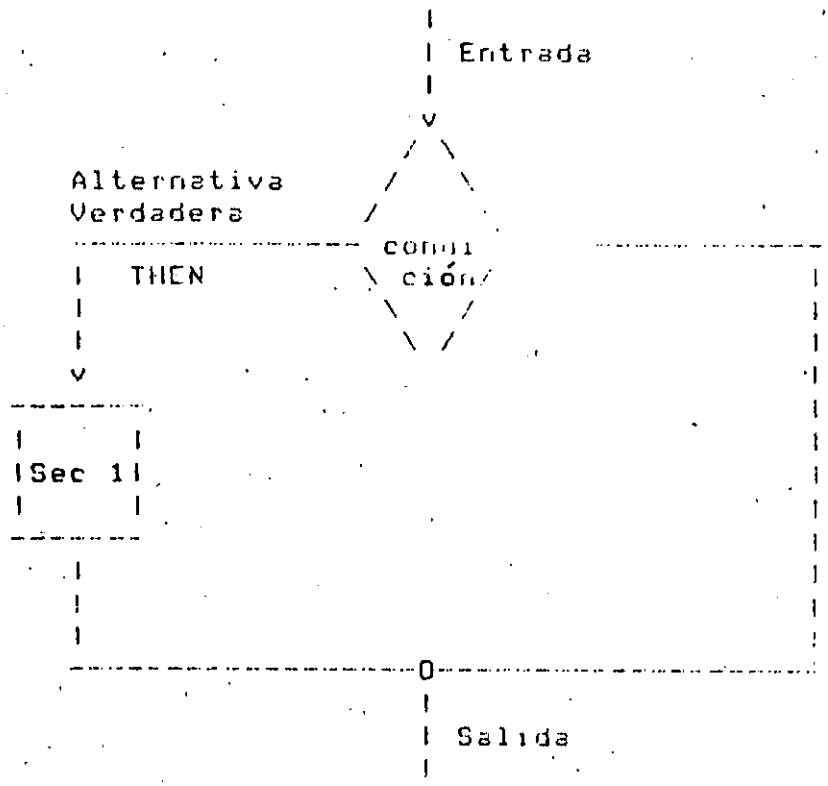
Dependiendo del resultado de la evaluación de la condición, se ejecutará la instrucción o conjunto de instrucciones agrupadas en la secuencia 1 (solamente si el resultado es verdadero) o la instrucción o conjunto de instrucciones de la secuencia 2 (si y solo si el resultado fué falso).

Una posible variante de esta estructura de control sería que no hubiera una acción específica para cuando la condición tuviera un valor de falso, esta variante es conocida como IF-THEN.

REPRESENTACION EN DIAGRAMA DE FLUJO DE LA FIGURA LOGICA IF THEN ELSE



REPRESENTACION EN DIAGRAMA DE FLUJO DE LA FIGURA LOGICA IF THEN



Un ejemplo que muestre claramente como se ejecutaría un programa con 2 instrucciones IF sería el siguiente:

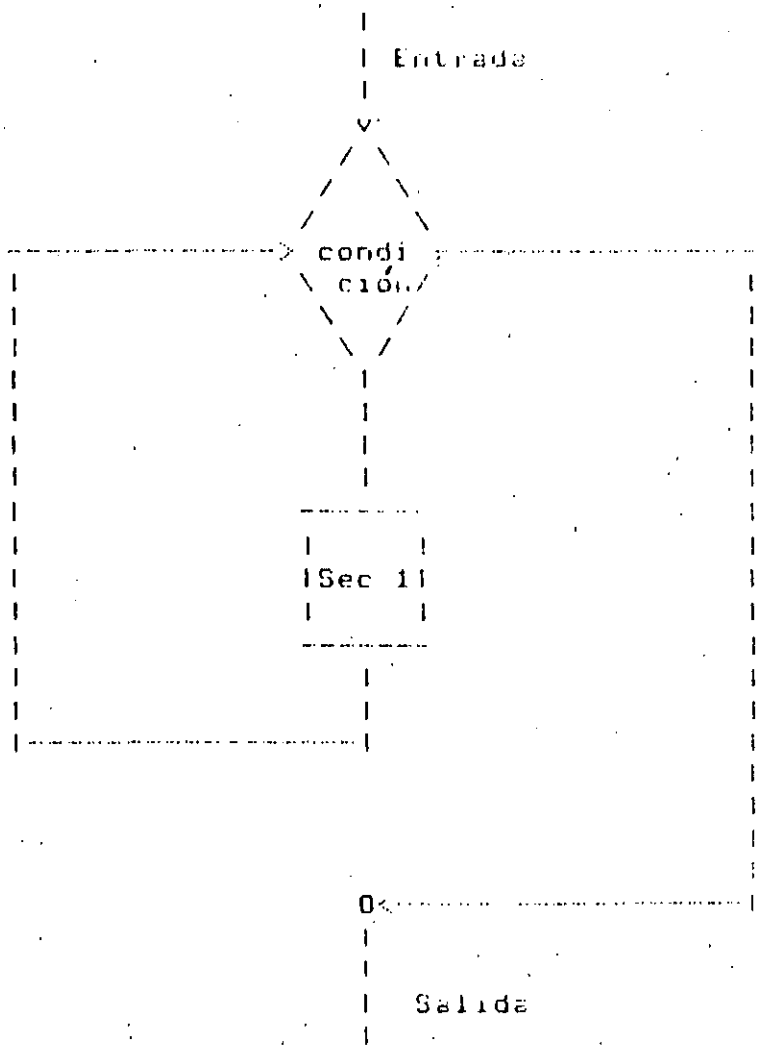
```
SECUENCIA #0
IF condición A THEN
   SECUENCIA #1
ELSE
   SECUENCIA #2
ENDIF
SECUENCIA #3
IF condición B THEN
   SECUENCIA #4
NOELSE
ENDIF
SECUENCIA #5
```

CONDICION A	VERDADERO	VERDADERO	FALSO	FALSO
CONDICION B	VERDADERO	FALSO	VERDADERO	FALSO
	SECUENCIA 0	SECUENCIA 0	SECUENCIA 0	SECUENCIA 0
	SECUENCIA 1	SECUENCIA 1	SECUENCIA 2	SECUENCIA 2
	SECUENCIA 3	SECUENCIA 3	SECUENCIA 3	SECUENCIA 3
	SECUENCIA 4	SECUENCIA 5	SECUENCIA 4	SECUENCIA 5
	SECUENCIA 5		SECUENCIA 5	

Tabla que muestra el flujo del programa dependiendo de los valores de las condiciones.

DO-WHILE: Es una instrucción de repetición o iteración condicional, ésto es, una instrucción o conjunto de instrucciones se ejecutará repetidamente hasta que la condición deje de cumplirse, cuando la instrucción o conjunto de instrucciones no afecte la condición, estas instrucciones se estarán ejecutando indefinidamente, a ésto se le conoce como entrar en un Loop infinito. Antes de ejecutar las instrucciones pregunta si se cumple la condición.

DIAGRAMA DE FLUJO DE LA FIGURA LOGICA DO-WHILE



Cabe hacer notar que si al llegar al DO-WHILE la condición no se cumple, no se ejecutará ni una sola vez la secuencia 1.

APENDICE B

PRACTICAS

P R A C T I C A S

D.1 PRIMERA PRACTICA

Objetivo:

Familiarizar al estudiante con el equipo VAX 11/780, mediante la creación, compilación y ejecución de uno o mas programas.

No se pretende que el estudiante comprenda exactamente el significado de cada instrucción de los programas, los cuales deberán ser "copiados" literalmente, para interactuar con el editor del sistema y posteriormente compilarlos y ejecutarlos.

EXPLICACION DEL JUEGO DE LOS CERILLOS

El programa 'Juego de cerillos' es un programa (de entretenimiento) que simula tener inteligencia, de la forma más elemental, a este tipo de programas se les conoce como programas de inteligencia artificial, en donde llega a haber programas tan complejos como los que juegan ajedrez, en un nivel muy alto de competencia.

La idea de este juego es competir con el programa y vencerlo, observando las siguientes reglas: Juega la computadora contra una persona.

a) Empieza el juego con 21 cerillos.

b) El programa pregunta quien retira primero cerillos, si será ella misma o el jugador.

c) Se retirarán cerillos alternativamente.

d) Se pueden retirar uno o varios cerillos siendo el máximo 5 cerillos los que se pueden retirar.

e) El que retire el último cerillo pierde.

f) Los cerillos se representan como '!'.

Que tenga suerte !

```
PROGRAM JUEGO( INPUT,OUTPUT );
CONST
     TIROMAX = 5 ;
     TIROMIN = 1 ;
     LIMINF  = 0 ;
     LIMSUP  = 8 ;

TYPE
     RANGE   = TIROMIN..TIROMAX;
VAR

     TIRO                        : RANGE   ;
     TIROVALIDO,HAYCERILLOS      : BOOLEAN;
     IND,NUMCERILLOS             : INTEGER;
     RESPUESTA                   : ARRAY[0..0] OF CHAR;
     PRIMOS                      : ARRAY[ LIMINF..LIMSUP ] OF INTEGER;

(*    ----> PROCEDIMIENTO  DE  INICIALIZACION <----    *)

PROCEDURE INICIALIZACION ;
     VAR
          I:INTEGER;
     BEGIN

     FOR I:=0 TO 8 DO
          CASE I OF
               0: PRIMOS[I]:=01;
               1: PRIMOS[I]:=02;
               2: PRIMOS[I]:=03;
               3: PRIMOS[I]:=05;
               4: PRIMOS[I]:=07;
               5: PRIMOS[I]:=11;
               6: PRIMOS[I]:=13;
               7: PRIMOS[I]:=17;
               8: PRIMOS[I]:=19;
          END
     (*ENDFOR*) ;

     IND:=8;
     NUMCERILLOS:=21;
     HAYCERILLOS:=TRUE;
     END
(*ENDPROC*);

(* PROCEDIMIENTO QUE DESPLIEGA CERILLOS EN PANTALLA *)

PROCEDURE DESPLIEGAPANTALLA;
VAR

     IX      ,
     CONT    : INTEGER ;

     BEGIN
     WRITELN;WRITELN;WRITELN;
     CONT:=1;

     FOR IX:=1 TO NUMCERILLOS DO
          BEGIN
          WRITE('I':5);
          CONT:=CONT+1;
```

```
        IF   [illegible]
             [illegible]
             [illegible]
             WRITELN;WRITELN
             END
        (*ENDIF*);

        END
    (*ENDFOR*);

    WRITELN;

    END
(*ENDPROC*);

(* --> FUNCION QUE DETERMINA CUANTOS CERILLOS    *)
(*      DEBE DE QUITAR LA COMPUTADORA             *)

FUNCTION FUNC( PARAMETRO:INTEGER ):INTEGER;
    BEGIN

    WHILE PARAMETRO <=  PRIMOSE IND 3 DO
        IND:= IND-1
    (*ENDWHILE*);

    FUNC:=PRIMOSE IND 3;
    END
(*ENDFUNC*) ;

(* ----> PROGRAMA PRINCIPAL <---- *)

BEGIN
INICIALIZACION;
DESPLIEGAPANTALLA;
WRITELN('* QUIERES QUE TIRE PRIMERO LA COMPUTADORA?(S,N) *');
READ(RESPUESTA[0]);

IF   (RESPUESTA[0] = 'S') THEN
    BEGIN
    WRITELN('* OK, PRIMERO TIRA LA COMPUTADORA *') ;
    TIRO := NUMCERILLOS - FUNC ( NUMCERILLOS )      ;
    NUMCERILLOS  :=  NUMCERILLOS - TIRO             ;
    WRITE('* QUITE ',TIRO:2,' CERILLOS, QUEDAN : ');
    WRITE(NUMCERILLOS:2,'  *')
    END
ELSE
    WRITELN('* BUENO, PRIMERO TIRAS TU *')
(*ENDIF*) ;

WHILE HAYCERILLOS DO
    BEGIN
    TIROVALIDO:=FALSE;

    WHILE NOT TIROVALIDO DO
        BEGIN
        DESPLIEGAPANTALLA;
        WRITELN;    WRITELN;
        WRITE('* HAY ',NUMCERILLOS:2,' CERILLOS,');
        WRITE(' CUANTOS QUITAS ?: ');
        READ(TIRO);

        IF   ((TIRO>0) AND (TIRO<6) AND (TIRO<=NUMCERILLOS)) THEN
             BEGIN
             TIROVALIDO:=TRUE;
             NUMCERILLOS:=NUMCERILLOS-TIRO;
```

```
        WRITE('* QUITASTE ',TIRO:5,' CERILLOS,QUEDAN    : ');
        WRITE(NUMCERILLOS:2);
        END
    (*NOELSE*)
    (*ENDIF*) ;

    END
  (*ENDWHILE*);

  IF  NUMCERILLOS > 6 THEN
      BEGIN
      TIRO:=NUMCERILLOS-FUNC(NUMCERILLOS);
      NUMCERILLOS:=NUMCERILLOS-TIRO;
      WRITELN;
      WRITE('* QUITE ',TIRO:2,' CERILLOS, QUEDAN        : ');
      WRITE(NUMCERILLOS:2,' *');
      END
  ELSE IF NUMCERILLOS > 1 THEN
      BEGIN
      TIRO:=NUMCERILLOS-1;
      NUMCERILLOS:=NUMCERILLOS-TIRO;
      WRITELN;
      WRITE('* QUITE ',TIRO:2,' CERILLOS, QUEDAN        : ');
      WRITE(NUMCERILLOS:2,' *');
      END
  ELSE IF NUMCERILLOS=1 THEN
      BEGIN
      WRITELN;
      WRITELN('* perdi, quite el ultimo cerillo');
      HAYCERILLOS:=FALSE;
      END
  ELSE
      BEGIN
      WRITELN;WRITELN;WRITELN;
      WRITELN('* !!!! G A N E E E E, JA  JA  JA');
      WRITELN('* CUANDO QUIERAS JUGAMOS OTRO ....');
      HAYCERILLOS:=FALSE;
      END
  (*ENDIF*);

  END
(*ENDWHILE*);

END.
$
```

```
     !     !     !
     !     !     !
     !     !     !
     !     !     !
     !     !     !
     !     !     !
     !     !     !

* QUIERES QUE TIRE PRIMERO LA COMPUTADORA?(S,N) *

S
* OK, PRIMERO TIRA LA COMPUTADORA *
  QUITE  2 CERILLOS, QUEDAN : 19  *

     !     !     !
     !     !     !
     !     !     !
     !     !     !
     !     !     !
     !     !     !
     !

* HAY 19 CERILLOS, CUANTOS QUITAS ?:  5

* QUITASTE      5 CERILLOS,QUEDAN    : 14
* QUITE  1 CERILLOS, QUEDAN          : 13 *

     !     !     !
     !     !     !
     !     !     !
     !     !     !
```

```
* HAY 13 CERILLOS, CUANTOS QUITAS ?:
                                        5

* QUITASTE      5 CERILLOS,QUEDAN    :  0
* QUITE  1 CERILLOS, QUEDAN          :  7 *

    !     !     !

    !     !     !

    !

* HAY  7 CERILLOS, CUANTOS QUITAS ?:   4

* QUITASTE      4 CERILLOS,QUEDAN    :  3
* QUITE  2 CERILLOS, QUEDAN          :  1 *

    !

* HAY  1 CERILLOS, CUANTOS QUITAS ?:   1

* QUITASTE      1 CERILLOS,QUEDAN    :  0

* !!!! G A N E E E E, JA  JA  JA
* CUANDO QUIERAS JUGAMOS OTRO ....
$
```

CALCULO DEL BIORRITMO

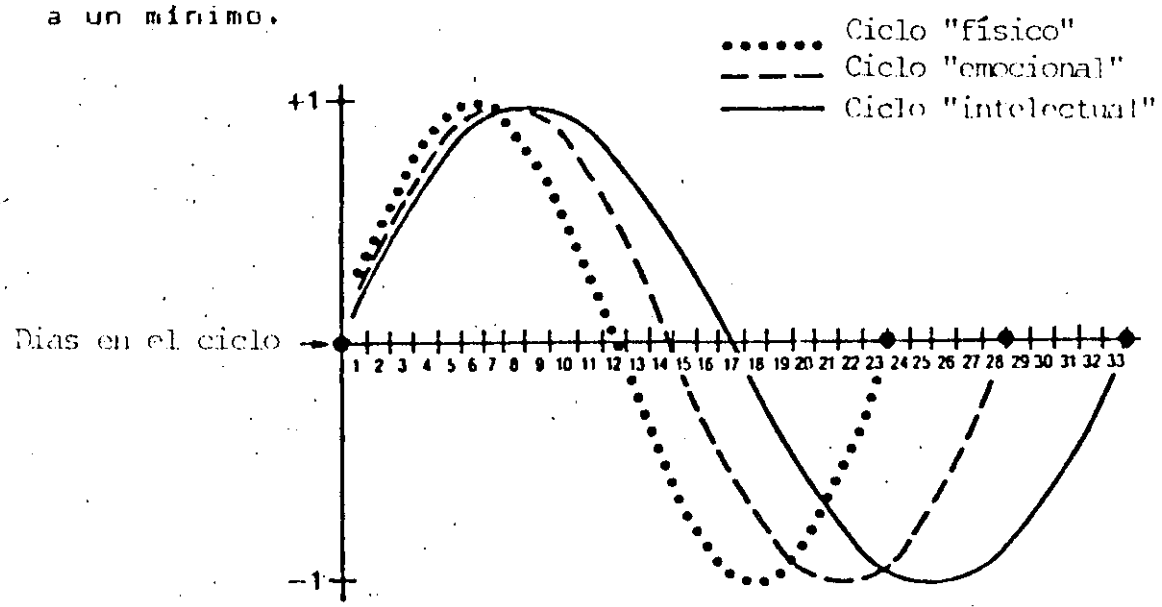
El biorritmo es una teoría que establece que existen tres ciclos en la vida de cada persona, mismos que se inician el día en que se nace:

1. El ciclo físico, con duración de 23 días.

2. El ciclo emocional, con duración de 28 días.

3. El ciclo intelectual, con duración de 33 días.

En la primera mitad de cada ciclo los niveles son crecientes, hasta llegar a un máximo, a partir del cual, empiezan a decrecer hasta llegar a un mínimo.



Las amplitudes de este ciclo de biorritmo, pueden ser expresados como un valor entre -1 y 1, utilizando la siguiente ecuación:

$$\text{amplitud} = \frac{\text{seno}(360 * \text{número de días desde el nacimiento})}{\text{número de días en el ciclo}}$$

El programa que aquí se presenta solicita como datos la fecha de nacimiento y la fecha de la cual se desea conocer la amplitud de los tres ciclos; entrega como salida estas amplitudes con valores entre -100 y 100.

```
PROGRAM BIORRITMO(INPUT,OUTPUT);
     VAR NUM,A,M,D2,M2,A2,N,D,F,E,I : INTEGER;

FUNCTION BR(CICLO:INTEGER):INTEGER;
     BEGIN
     BR:=TRUNC(100*SIN(6.2832*(NUM/CICLO)));
     END;
(*END BR*)

FUNCTION DIAJ(DD,MM,AA:INTEGER):INTEGER;
     BEGIN
     IF MM <= 2 THEN
          BEGIN
          A:=AA-1;
          M:=MM-13;
          END
     ELSE
          BEGIN
          A:=AA;
          M:=MM+1;
          END;
     (*END IF*)
     DIAJ:=TRUNC(365.25*A)+TRUNC(30.001*M)+DD+1720982;
END;
(*END DIAJ*)

PROCEDURE LEE;
     BEGIN
     WRITELN('DE QUE FECHA DESEAS CONOCER TU BIORRITMO? DD MM AA
     WRITELN('(PARA TERMINAR 0 0 0)');
     READ(D2,M2,A2);
     END;
(*END LEE*)

(**** PROGRAMA PRINCIPAL ****)

BEGIN
WRITELN('DAME LA FECHA DE TU NACIMIENTO DD MM AAAA');
READ(D2,M2,A2);
N:=DIAJ(D2,M2,A2);
LEE;
WHILE D2 <> 0 DO
     BEGIN
     D:=DIAJ(D2,M2,A2);
     NUM:=D-N;
     F:=BR(23);
     E:=BR(28);
     I:=BR(33);
     WRITELN('ESCALA              :-100-MINIMO , 100-MAXIMO');
     WRITELN('NIVEL FISICO        :',F);
     WRITELN('NIVEL EMOCIONAL     :',E);
     WRITELN('NIVEL INTELECTUAL :',I);
     WRITELN;
     LEE;
     END;
(*END WHILE*)
END.
```

```
DAME LA FECHA DE TU NACIMIENTO DD MM AAAA
11 08 1960
DE QUE FECHA DESEAS CONOCER TU BIORRITMO? DD MM AAAA
(PARA TERMINAR 0 0 0)
16 08 1984
ESCALA              :-100=MINIMO , 100=MAXIMO
NIVEL FISICO        :          31
NIVEL EMOCIONAL     :          99
NIVEL INTELECTUAL   :         -97

DE QUE FECHA DESEAS CONOCER TU BIORRITMO? DD MM AAAA
(PARA TERMINAR 0 0 0)
17 08 1984
ESCALA              :-100=MINIMO , 100=MAXIMO
NIVEL FISICO        :          62
NIVEL EMOCIONAL     :          97
NIVEL INTELECTUAL   :         -90

DE QUE FECHA DESEAS CONOCER TU BIORRITMO? DD MM AAAA
(PARA TERMINAR 0 0 0)
18 08 1984
ESCALA              :-100=MINIMO , 100=MAXIMO
NIVEL FISICO        :          39
NIVEL EMOCIONAL     :          89
NIVEL INTELECTUAL   :         -81

DE QUE FECHA DESEAS CONOCER TU BIORRITMO? DD MM AAAA
(PARA TERMINAR 0 0 0)
19 08 1984
ESCALA              :-100=MINIMO , 100=MAXIMO
NIVEL FISICO        :          13
NIVEL EMOCIONAL     :          77
NIVEL INTELECTUAL   :         -68

DE QUE FECHA DESEAS CONOCER TU BIORRITMO? DD MM AAAA
(PARA TERMINAR 0 0 0)
20 08 1984
ESCALA              :-100=MINIMO , 100=MAXIMO
NIVEL FISICO        :         -14
NIVEL EMOCIONAL     :          61
NIVEL INTELECTUAL   :         -53

DE QUE FECHA DESEAS CONOCER TU BIORRITMO? DD MM AAAA
(PARA TERMINAR 0 0 0)
31 08 1984
ESCALA              :-100=MINIMO , 100=MAXIMO
NIVEL FISICO        :           0
NIVEL EMOCIONAL     :         -97
NIVEL INTELECTUAL   :          99

DE QUE FECHA DESEAS CONOCER TU BIORRITMO? DD MM AAAA
(PARA TERMINAR 0 0 0)

0 0 0
```

DIAS DE LA SEMANA

Este programa permite conocer, mediante la aplicación de un conocido procedimiento, el día de la semana que corresponde a una fecha determinada. El procedimiento es el siguiente:

Dada una fecha DD MM AAAA :

- Si MM es 1 ó 2, entonces hacer

  AA=AA-1

  MM=MM+2

- Sea A la parte entera del siguiente cociente

$$\frac{AA}{100}$$

- Sea b la parte entera del siguiente cociente

$$\frac{A}{4}$$

- Sea B=2 - A + b

- Sea C la parte entera del siguiente producto

  365.25 X AA

- Sea D la parte entera del siguiente producto

  30.6001 X (MM + 1)

- Sea JD = B+ C + D + DD + 1720994.5 .

- Sea

$$X = \frac{JD + 1.5}{7}$$

- Sea d la parte entera de X

- Redondear al entero más próximo el resultado de la siguiente operación.

$$Z = (X - b) X 7,$$

después sea Z el valor resultante

- El valor de Z indicará el día de la semana, según la siguiente tabla:

Z	DIA
0	domingo
1	lunes
2	martes
3	miercoles
4	jueves
5	viernes
6	sabado

NOTA: Este procedimiento sólo es válido para fechas posteriores al 16 de octubre de 1582.

```
PROGRAM DIAS(INPUT,OUTPUT);
VAR DD,MM,AA ,Z      : INTEGER ;
A,B,C,D,JD,X         : REAL   ;

BEGIN
WRITELN('DAME LA FECHA DD MM AAAA');
READ(DD,MM,AA);
WHILE DD <> 00 DO
     BEGIN
     IF MM < 3 THEN
          BEGIN
          AA:= AA - 1;
          MM:= MM + 12 ;
          END ;
     (*ENDIF*)
     A:=TRUNC( AA/100 );
     B:=2 - A + TRUNC( A/4 );
     C:=TRUNC( 365.25 * AA);
     D:=TRUNC( 30.6001 * ( MM+1 ) );
     JD:=B + C + D + DD + 1720994.5 ;
     X:=( JD + 1.5 ) / 7 ;
     Z:=ROUND(( X-TRUNC( X ) ) * 7) ;
     CASE  Z OF
          0: WRITELN('DOMINGO');
          1: WRITELN('LUNES');
          2: WRITELN('MARTES');
          3: WRITELN('MIERCOLES');
          4: WRITELN('JUEVES');
          5: WRITELN('VIERNES');
          6: WRITELN('SABADO');
     END;
     WRITELN;
     WRITELN('DAME LA FECHA DD MM AA');
     READ(DD,MM,AA);
     END;
(*END WHILE*)
END.
```

```
 RUN CURS01
DAME LA FECHA DD MM AAAA
6 3 1959
VIERNES

DAME LA FECHA DD MM AA
11 8 1960
JUEVES

DAME LA FECHA DD MM AA
16 8 1984
JUEVES

DAME LA FECHA DD MM AA
14 7 1952
LUNES

DAME LA FECHA DD MM AA
18 9 1955
DOMINGO

DAME LA FECHA DD MM AA
7 9 1984
VIERNES

DAME LA FECHA DD MM AA
9 10 1984
MARTES

DAME LA FECHA DD MM AA
6 10 1984
SABADO

DAME LA FECHA DD MM AA
00 00 0000
$
```

D.2 SEGUNDA PRACTICA

Objetivo:

Presentar un ejemplo práctico donde se muestre el uso de las instrucciones vistas hasta este momento: identificadores, tipos predefinidos de datos, asignación, instrucciones de entrada y salida, constantes, etc. Asimismo, el programa aquí presentado se desarrolló en base a las reglas de la programación estructurada usando para ello las figuras lógicas recientemente vistas (estas figuras se verán con más detalle en las siguientes sesiones).

Breve descripción del programa:

El presente programa sirve para convertir grados centígrados o Celsius a grados Fahrenheit y visceversa (Fahrenheit a Celsius). En primer lugar, el programa pide (o presunta) que tipo de conversión se va a llevar a cabo (°C ->°F o°F ->°C), a continuación se proporciona al programa la cifra a convertir y el programa entrega como salida no sólo la conversión solicitada, sino el equivalente a Kelvin y Rankin (escalas de temperatura absoluta).

Pseudocódigo de conversión de grados.

```
WHILE  Haya datos DO
     SEQ: Seleccione tipo de conversión ( C ->  F ó   F ->  C)
     SEQ: Lee la cifra a convertir
     IF   tipo-conversión = 'F' THEN
          SEQ:  F <-  ( C * 1.8) + 32
     ELSE
          SEQ:  C <- ( F - 32) / 1.8
     ENDIF;
     SEQ: KELVIN <-  C + 273;
     SEQ: RANKIN <-  F + 460;
     IF Tipo-conversión = 'F' THEN
          SEQ: Imprime 'grados CELSIUS a FAHRENHEIT = '  F
     ELSE
          SEQ: Imprime 'GRADOS FAHRENHEIT a CELSIUS = '  C
     ENDIF;
     SEQ: Imprime KELVIN;
     SEQ: Imprime RANKIN;
ENDWHILE;
```

```
PROGRAM CONVIERTE (INPUT,OUTPUT);

CONST
    CAMPO     = 5 ;
    PRECISION= 2 ;

VAR
    GRADOS,
    CELSIUS,
    FAHRENHEIT,
    KELVIN,
    RANKIN       : REAL;
    CONVERSION : CHAR;

BEGIN
WRITELN(OUTPUT,'SELECCIONE SU CONVERSION DE ACUERDO A',
        ' LA SIGUIENTE CONVENCION :');
WRITELN(OUTPUT,'GRADOS CELSIUS (CENTIGRADOS) A FAHRENHEIT :',
        ' TECLEE UNA F');
WRITELN(OUTPUT,'GRADOS FAHRENHEIT A CELSIUS : TECLEE UNA C');
WRITELN(OUTPUT,'SELECCIONE SU CONVERSION [''F'' O ''C'']');
READ(INPUT,CONVERSION);
WHILE (CONVERSION = 'F') OR (CONVERSION = 'C') DO
    BEGIN
    WRITELN(OUTPUT,'TECLEE LA CIFRA QUE DESEE CONVERTIR: ');
    READLN;
    READ(INPUT,GRADOS);
    IF CONVERSION = 'F' THEN
        BEGIN
        IF (GRADOS < -273) THEN
            WRITELN(OUTPUT,'ESA TEMPERATURA NO ES FACTIBLE DE',
                    ' SER ALCANZADA')
        ELSE
            BEGIN
            CELSIUS:=GRADOS;
            FAHRENHEIT:=(CELSIUS*1.8)+32;
            END
        (*ENDIF*)
        END
    ELSE
        BEGIN
        IF (GRADOS < -460) THEN
            WRITELN(OUTPUT,'ESA TEMPERATURA NO ES FACTIBLE DE',
                    ' SER ALCANZADA')
        ELSE
            BEGIN
            FAHRENHEIT:=GRADOS;
            CELSIUS:=(FAHRENHEIT-32)/1.8;
            END
        (*ENDIF*)
        END
    (*ENDIF*);
    KELVIN:=CELSIUS+273;
    RANKIN:=FAHRENHEIT+460;
    IF CONVERSION = 'F' THEN
        BEGIN
        WRITELN(OUTPUT,GRADOS:CAMPO:PRECISION,' GRADOS CELSIUS',
                ' SON : ');
```

```
        BEGIN
        WRITELN(OUTPUT,GRADOS:CAMPO:PRECISION,' GRADOS',
                  ' FAHRENHEIT SON : ');
        WRITELN(OUTPUT,CELSIUS:CAMPO:PRECISION,' GRADOS',
                  ' CELSIUS');
        END
    (*ENDIF*);
    WRITELN(OUTPUT,KELVIN:CAMPO:PRECISION,' KELVIN');
    WRITELN(OUTPUT,RANKIN:CAMPO:PRECISION,' RANKIN');
    WRITELN;
    WRITELN(OUTPUT,'SELECCIONE SU CONVERSION ['''F''' O '''C''']');
    READLN;
    READ(INPUT,CONVERSION);
    END
END.
$
```

```
 RUN CONVIERTE
SELECCIONE SU CONVERSION DE ACUERDO A LA SIGUIENTE CONVENCION
GRADOS CELSIUS (CENTIGRADOS) A FAHRENHEIT : TECLEE UNA F
GRADOS FAHRENHEIT A CELSIUS : TECLEE UNA C
SELECCIONE SU CONVERSION ['F' O 'C']
F
TECLEE LA CIFRA QUE DESEE CONVERTIR:
0
 0.00 GRADOS CELSIUS SON :
32.00 GRADOS FAHRENHEIT
273.00 KELVIN
492.00 RANKIN

SELECCIONE SU CONVERSION ['F' O 'C']
C
TECLEE LA CIFRA QUE DESEE CONVERTIR:
212
212.00 GRADOS FAHRENHEIT SON :
100.00 GRADOS CELSIUS
373.00 KELVIN
672.00 RANKIN

SELECCIONE SU CONVERSION ['F' O 'C']
F
TECLEE LA CIFRA QUE DESEE CONVERTIR:
-273
-273.00 GRADOS CELSIUS SON :
-459.40 GRADOS FAHRENHEIT
 0.00 KELVIN
 0.60 RANKIN

SELECCIONE SU CONVERSION ['F' O 'C']
C
TECLEE LA CIFRA QUE DESEE CONVERTIR:
-460
-460.00 GRADOS FAHRENHEIT SON :
-273.33 GRADOS CELSIUS
-0.33 KELVIN
 0.00 RANKIN

SELECCIONE SU CONVERSION ['F' O 'C']
-280
```

D.3 TERCERA PRACTICA

Objetivo:

Observar, mediante un programa pequeño e interesante, el manejo de los elementos de un arreglo, particularmente de un arreglo de caracteres.

Descripción del programa:

Existen unos curiosos textos que tienen la propiedad de significar lo mismo cuando se leen de izquierda a derecha que cuando se leen de derecha a izquierda, por supuesto, sin considerar puntuación ni espacios en blanco sino únicamente letras, por ejemplo, la palabra RADAR o el texto ANITA LAVA LA TINA; tales textos se llaman palindromas.

El programa de la presente práctica permite detectar cuáles textos son palindromas y cuáles no; esto se hace almacenando los caracteres del texto en un arreglo, pero solamente aquellos que son letras, para después comparar uno a uno los caracteres almacenados al principio con los almacenados al final a fin de detectar si el texto forma un palindroma.

Cada palindroma debe terminarse con un punto y coma, esto permite que se puedan revisar palindromas muy largos que ocupen más de una línea de entrada.

B.3 TERCERA PRACTICA

Objetivo:

Observar, mediante un programa pequeño e interesante, el manejo de los elementos de un arreglo, particularmente de un arreglo de caracteres.

Descripción del programa:

Existen unos curiosos textos que tienen la propiedad de significar lo mismo cuando se leen de izquierda a derecha que cuando se leen de derecha a izquierda, por supuesto, sin considerar puntuación ni espacios en blanco sino únicamente letras, por ejemplo, la palabra RADAR o el texto ANITA LAVA LA TINA; tales textos se llaman palíndromos.

El programa de la presente práctica permite detectar cuáles textos son palíndromos y cuáles no; ésto se hace almacenando los caracteres del texto en un arreglo, pero solamente aquellos que son letras, para después comparar uno a uno los caracteres almacenados al principio con los almacenados al final a fin de detectar si el texto forma un palíndromo.

Cada palíndromo debe terminarse con un punto y coma, esto permite que se puedan revisar palíndromos muy largos que ocupen más de una línea de entrada.

```
PROGRAM PALINDROMOS (INPUT,OUTPUT);

  INST
     INICIAL = 1;
     BUFSIZE = 1300;
VAR
     BUFFER   : PACKED ARRAY [INICIAL..BUFSIZE] OF CHAR;
     BYTE     : CHAR;
     IZQ,DER  : INTEGER;

BEGIN
READ(INPUT,BYTE);
WHILE NOT EOF DO
     BEGIN
     DER:=INICIAL;
     WHILE BYTE <> ';' DO
          BEGIN
          IF EOLN THEN
               WRITELN
          ELSE
               BEGIN
               WRITE(OUTPUT,BYTE);
               IF ('A' <= BYTE) AND (BYTE <='Z') THEN
                    BEGIN
                    BUFFER[DER] := BYTE;
                    DER:= SUCC(DER);
                    END
               (*NOELSE*)
               (*ENDIF*)
               END;
          (*ENDIF*)
          READ(INPUT,BYTE)
          END;
     (*ENDWHILE*)
     WRITELN;
     WRITELN;
     IZQ:=INICIAL;
     DER:=PRED(DER);
     WHILE (BUFFER[IZQ] = BUFFER[DER]) AND (IZQ < DER) DO
          BEGIN
          IZQ:=SUCC(IZQ);
          DER:=PRED(DER);
          END;
     (*ENDWHILE*)
     IF IZQ < DER THEN
          WRITELN(OUTPUT,'NO ES PALINDROMO')
     ELSE
          WRITELN(OUTPUT,'SI ES PALINDROMO');
     (*ENDIF*)
     WRITELN;
     READ(INPUT,BYTE);
     END
(*ENDWHILE*)

END.
```

ANA LOS ANAGRAMAS AMARGAN A SOLANA

SI ES PALINDROMO

A SU RALO VELLO LO LLEVO LA RUSA

SI ES PALINDROMO

SARA A LA RUSA RASURALA A RAS

SI ES PALINDROMO

ANITA LAVA LA TINA

SI ES PALINDROMO

ESTO NO ES UN PALINDROMO

NO ES PALINDROMO

B.4 CUARTA PRACTICA

Objetivo:

Confirmar mediante un ejemplo de aplicación práctica, los temas recientemente expuestos relacionados con subrutinas, funciones y paso de parámetros.

El programa acepta como entrada una cantidad numérica y entrega como salida esta cantidad escrita en palabras. Se hace uso de una subrutina que efectúa esta tarea para cantidades de tres cifras y, el programa principal "acomoda" el resultado de esta rutina en MILES, CIENMILES, etc.

D.5 QUINTA PRACTICA

Objetivos:

Mostrar mediante un ejemplo de aplicación el uso de diversos temas tratados a lo largo del curso como son:

a) Uso de valores constantes.

b) Tipos definidos por el usuario (escalares).

c) Subrangos.

d) Funciones, subrutinas y paso de parámetros.

Asimismo, se hace especial énfasis en la modularidad y estructuración del programa. También se hace uso intensivo de las figuras lógicas SEQUENCE, IF THEN ELSE Y WHILE DO.

Breve descripción del funcionamiento del programa:

El presente programa de ejemplo consta de tres fases principales:

1. Creación e impresión de un máximo de 200 números enteros aleatorios (random). El máximo de números a generar es dado como dato de entrada al programa.

2. Ordenamiento e impresión de los números previamente generados.

3. Búsqueda de algún (o algunos) número(s) en particular dentro de la lista ordenada de números.

Para esta última fase, el programa pregunta iterativamente si se quiere buscar algún número, a lo cual se le debe proporcionar la letra 'S' (Si) o bien, si se le responde con cualquier otro caracter, se tomará como si la respuesta hubiese sido negativa y el programa terminará. Si la respuesta fue 'S' (Si), se preguntará cuál es el número que se quiere buscar, a lo cual se contestará con el número requerido, si dicho número se encuentra en la lista

ordenada, el programa dará como salida la posición donde encuentra, de no existir dicho número en la lista, el programa indicará que no fue encontrado y preguntará si se quiere buscar algún otro número.

```
PROGRAM ORDYBUSQ(INPUT,OUTPUT);

CONST
     DIMMAX = 200;

TYPE
     INDICE = 1..DIMMAX;
     VECTOR = ARRAY[INDICE] OF INTEGER;

VAR
     NUMEROS : VECTOR;
     BUSCADO,
     MAXIMO ,
     IX      : INDICE;
     SEMILLA : INTEGER;
     RESP    : CHAR  ;

PROCEDURE IMPRIMENUMS(VAR NUMEROS : VECTOR);
     VAR
          I,J,K : INDICE;
     BEGIN
     WRITELN;
     K:=1;
     FOR I:=1 TO MAXIMO DO
          BEGIN
          WRITE(OUTPUT,NUMEROS[I]:4);
          K:=SUCC(K);
          IF (K DIV 6) > 0 THEN
               BEGIN
               WRITELN;
               K:=1;
               END
          (*NOELSE*)
          (*ENDIF*)
          END
     (*ENDFOR*)
     END;
(*ENDPROC*)

FUNCTION RANDOM(VAR SEMILLA : INTEGER) : REAL;
     BEGIN
     RANDOM:=SEMILLA/65535;
     SEMILLA:=(25173*SEMILLA+13849) MOD 65536;
     END;
(*ENDFUNC*)

PROCEDURE GENERANUMS(VAR NUMEROS:VECTOR; VAR SEMILLA:INTEGER);
     VAR
          NUMERO : INTEGER;
          ALEATORIO : REAL;
     BEGIN
     FOR IX:= 1 TO MAXIMO DO
          BEGIN
          ALEATORIO:=RANDOM(SEMILLA);
          NUMERO:=TRUNC(ALEATORIO*1000)+1;
          NUMEROS[IX]:=NUMERO
          END
     (*ENDFOR*)
     END;
(*ENDPROC*)

PROCEDURE ORDENANUMS(VAR NUMEROS:VECTOR);
     VAR
```

```
        TEMPORAL: INTEGER;
    BEGIN
    FOR I:=2 TO MAXIMO DO
        BEGIN
        FOR J:=MAXIMO DOWNTO I DO
            IF NUMEROS[J-1] > NUMEROS[J] THEN
                BEGIN
                TEMPORAL:=NUMEROS[J-1];
                NUMEROS[J-1]:=NUMEROS[J];
                NUMEROS[J]:=TEMPORAL
                END
            (*NOELSE*)
            (*ENDIF*)
        (*ENDFOR*)
        END
    (*ENDFOR*)
    END;
(*ENDPROC*)

PROCEDURE DISECCION(BUSCADO:INTEGER);
    VAR
        I,J,K : INDICE;
    BEGIN
    I:=1;
    J:=MAXIMO;
    K:=(I+J) DIV 2;
    REPEAT
        K:=(I+J) DIV 2;
        IF BUSCADO > NUMEROS[K] THEN
            I:=K+1
        ELSE
            J:=K-1;
        (*ENDIF*)
    UNTIL (NUMEROS[K] = BUSCADO) OR (I > J);
    IF NUMEROS[K] = BUSCADO THEN
        WRITELN(OUTPUT,'EL NUMERO ',BUSCADO:4,
                ' ESTA EN LA POSICION : ',K:2)
    ELSE
        WRITELN(OUTPUT,'EL NUMERO ',BUSCADO:4,
                ' NO ESTA EN LA LISTA');
    (*ENDIF*)
    END;
(*ENDPROC*)

(*****************************)
(*                           *)
(*   PROGRAMA PRINCIPAL      *)
(*                           *)
(*****************************)

BEGIN
WRITELN(OUTPUT,'TECLEE LA SEMILLA PARA GENERAR LOS NUMEROS');
READ(INPUT,SEMILLA);
WRITELN(OUTPUT,'CUANTOS NUMEROS QUIERES GENERAR? [MAXIMO 200]');
READ(INPUT,MAXIMO);
GENERANUMS(NUMEROS,SEMILLA);
WRITELN; WRITELN;
WRITELN(OUTPUT,'LOS ',MAXIMO:4,' NUMEROS GENERADOS SON:');
WRITELN;
IMPRIMENUMS(NUMEROS);
ORDENANUMS(NUMEROS);
WRITELN; WRITELN;
WRITELN(OUTPUT,'LOS NUMEROS ORDENADOS SON:');
WRITELN;
```

```
WRITELN; WRITELN;
WRITELN(OUTPUT,'QUIERES BUSCAR ALGUN NUMERO? [S O N]');
READLN;
READ(INPUT,RESP);
WHILE RESP = 'S' DO
     BEGIN
     WRITELN;
     WRITELN(OUTPUT,'QUE NUMERO QUIERES BUSCAR?');
     READLN;
     READ(INPUT,BUSCADO);
     BISECCION(BUSCADO);
     WRITELN;
     WRITELN(OUTPUT,'QUIERES BUSCAR OTRO NUMERO? [S O N]');
     READLN;
     READ(INPUT,RESP);
     END
(*ENDWHILE*)
END.
```

```
 RUN ORDYBUSQ
TECLEE LA SEMILLA PARA GENERAR LOS NUMEROS
45
CUANTOS NUMEROS QUIERES GENERAR? [MAXIMO 200]
200

LOS  200 NUMEROS GENERADOS SON:

   1 497 221 905 369
 317 411 910 456 814
 626 489 934  97 161
 878 727 526 565 561
   4 709 634 273 109
 507 415 997 953  29
 205 968 217 633 550
 670 155 930 248 839
 188 778 346 457 987
 288 139 761 474 818
 176 233 822 978 253
 608 452 628 417 871
  36 876 964 256 496
  82 443 248   4 855
 648 581 982  55 628
 238 101 791 680 455
 204 423 350 717 702
 560 434 756 778  61
 981 558 182  35 286
 358 838 843 897 638
 916  93 611 135 839
 645 474 831 279 607
 783 963 157 341  85
  13 645 454 178 717
 166 806 107  58 391
 508 170 271 241 917
 914 841 890 643 136
 501 758 546 881 237
 510 736 448 283  92
 571 384 122 651 661
 484 489 617 865 796
 370 662 292 617 997
 708 303 494 856 927
 506 223 681 740 761
 852 455 821 177 677
 415  89 113 245  43
 718 180 353  75 130
 235  48 494 995 389
 626 536 237   2 636
 684  26 681 873 672
```

LOS NUMEROS ORDENADOS SON:

1	2	4	4	13
26	29	35	36	43
48	55	58	61	75
82	85	89	92	93
97	101	107	113	122
130	135	136	139	155
157	161	166	170	176
177	178	180	182	188
189	205	217	221	223
233	235	237	237	238
241	245	248	248	253
256	271	273	279	283
284	286	288	292	303
317	341	346	350	353
358	369	370	384	389
391	411	415	415	417
423	434	443	448	452
454	455	455	456	457
474	474	484	489	489
494	494	496	497	501
506	507	508	518	526
536	546	550	558	560
561	565	571	581	607
608	611	617	617	626
626	628	628	633	634
636	638	643	645	645
648	651	661	662	670
672	677	680	681	681
684	702	708	709	717
717	718	727	736	740
756	758	761	761	778
778	783	791	796	806
814	818	821	822	831
838	839	839	841	843
852	855	856	865	871
873	876	878	881	890
897	905	910	914	916
917	927	930	934	953
963	964	968	978	981
982	987	995	997	997

```
QUIERES BUSCAR ALGUN NUMERO? [S O N]
S

QUE NUMERO QUIERES BUSCAR?
13
EL NUMERO   13 ESTA EN LA POSICION :  5

QUIERES BUSCAR OTRO NUMERO? [S O N]
S

QUE NUMERO QUIERES BUSCAR?
561
EL NUMERO  561 ESTA EN LA POSICION : 111

QUIERES BUSCAR OTRO NUMERO? [S O N]
S

QUE NUMERO QUIERES BUSCAR?
1000
EL NUMERO 1000 NO ESTA EN LA LISTA

QUIERES BUSCAR OTRO NUMERO? [S O N]
N
$
```

SALTO DEL CABALLO.

Dado un tablero de nxn, con n campos, un caballo que se puede mover dentro del tablero de acuerdo a las reglas del ajedrez, es colocado en un campo del tablero de coordenadas X0,Y0. El problema consiste en encontrar un recorrido que cubre todo el tablero con n -1 movimientos visitando cada campo exactamente una vez.

El tablero se representa por una matriz llamada H, que es de tamaño nxn. Se utiliza la siguiente convención para determinar si un campo ha sido visitado o no:

H [X,Y] = 0 : El campo <X,Y> no ha sido visitado aún.

H [X,Y] = i : El campo <X,Y> fué visitado en el movimiento número i, donde: 1 <= i <= n .

En este programa el tamaño del tablero es de 5 x 5 que está definido por la variable N, el caballo comienza su recorrido en el campo definido por la primera columna del primer renglón de la matriz H, este campo se indica por las variables X0 y Y0. El número de campos del tablero está definido por la variable NS0.

El movimiento se realiza al principio de la instrucción REPEAT.

El programa no recibe ninsun dato de entrada.

```
PROGRAM SALTODELCABALLO( OUTPUT );

  CONST

        N  = 5;
        X0 = 1;
        Y0 = 1;
        NSQ=25;

  TYPE

        INDICES = 1..N;

  VAR

        Q: BOOLEAN;
        I,J: INDICES ;
        S: SET OF INDICES;
        A,B:ARRAY[1..8] OF INTEGER;
        H:ARRAY[INDICES,INDICES] OF INTEGER;

  PROCEDURE INTENTACON( I:INTEGER ;
                        X,Y:INDICES;
                        VAR Q:BOOLEAN);
     VAR

        K,U,V: INTEGER;
        Q1:BOOLEAN;

     BEGIN
        K:=0;

        REPEAT
           K:=SUCC(K);
           Q1:=FALSE ;
           U:=X+A[K] ;
           V:=Y+B[K] ;

           IF  (U IN S) AND (V IN S) THEN
               IF  H[U,V]=0 THEN
                   BEGIN
                   H[U,V]:=I;

                   IF  I<NSQ THEN
                       BEGIN
                       INTENTACON(I+1,U,V,Q1);
                       IF  NOT Q1 THEN
                           H[U,V]:=0
                       (*ENDIF*)
                       END
                   ELSE
                       Q1:=TRUE
                   (*ENDIF*)

                   END
               (*NOELSE*)
               (*ENDIF*)
           (*NOELSE*)
           (*ENDIF*)

        UNTIL Q1 OR (K=8);
        Q:=Q1
        END
```

```
(*                                                          *)
(*        PROGRAMA      PRINCIPAL                           *)
(*                                                          *)

    BEGIN

        S := [1,2,3,4,5];
        A[1]:=  2;      B[1]:=  1;
        A[2]:=  1;      B[2]:=  2;
        A[3]:= -1;      B[3]:=  2;
        A[4]:= -2;      B[4]:=  1;
        A[5]:= -2;      B[5]:= -1;
        A[6]:= -1;      B[6]:= -2;
        A[7]:=  1;      B[7]:= -2;
        A[8]:=  2;      B[8]:= -1;

        FOR I:=1 TO N DO
            FOR J:=1 TO N DO
                H[I,J]:= 0          (* SE INICIALIZA EL     *)
            (*ENDFOR*)              (* TABLERO CON CEROS.   *)
        (*ENDFOR*);

        H[XO,YO]:=1;                (* POSICION DE INICIO   *)
        INTENTACON(2,XO,YO,Q);      (* DE MOV. DEL CABALLO  *)

        IF  Q THEN
            FOR I:=1 TO N DO
                BEGIN
                FOR J:=1 TO N DO
                    WRITE( H[I,J]:5 )
                (*ENDFOR*);
                WRITELN
                END
            (*ENDFOR*)
        ELSE
            WRITELN('* SIN SOLUCION *')
        (*ENDIF*) ;

    END

(*ENDPROGRAM*).
```

1	6	15	10	21
14	9	20	5	16
19	2	7	22	11
8	13	24	17	4
25	18	3	12	23

$

APENDICE I

EJEMPLOS

E J E M P L O S

CALCULO DE LA FECHA DEL DOMINGO DE RESURRECCION.

La fecha del domingo de resurrección es una fecha móvil, generalmente es el primer domingo, cuarenta días posteriores a la primer luna nueva después del 21 de marzo. Dicha fecha puede encontrarse por métodos analíticos o por tablas. Hay varios métodos analíticos legados por varios matemáticos a través de los siglos. Aquí describiremos un método aparecido por primera vez en 1876 en el Butcher's Ecclesiastical Calendar, y es válido para todos los años del calendario Gregoriano, es decir, de 1583 en adelante.

El método aquí representado hace uso repetidamente de la división de un número entre otro, tomando la parte entera de la división y tratando separadamente el residuo. Los pasos a seguir son:

METODO	PARTE ENTERA	RESIDUO
1. Divídase el año dado entre 19	-	A
2. Divídase el año dado entre 100	B	C
3. Divídase B entre 4	D	E
4. Divídase (B+8) entre 25	F	-
5. Divídase (B-F+1) entre 3	G	-
6. Divídase (19A+B-D-G+15) entre 30	-	H
7. Divídase C entre 4	I	K
8. Divídase (32+2E+2I-H-K) entre 7	-	L
9. Divídase (A+11H+22L) entre 451	M	-
10. Divídase (H+L-7M+114) entre 31	N	P

11. El día del mes es que cae la fecha del Domingo de Resurrección es P+1.

El número del mes es N (si N=3 es Marzo y si N=4 es Abril)

```
PROGRAM RESURRECCION (INPUT,OUTPUT);

VAR
    A,B,C,D,E,
    F,G,H,I,K,
    L,M,N,P,
    ANIO      : INTEGER;

BEGIN
WRITE(OUTPUT,'DAME EL ANIO EN QUE DESEAS CONOCER LA PASCUA : ');
READ(INPUT,ANIO);
A:=ANIO MOD 19;
B:=ANIO DIV 100;
C:=ANIO MOD 100;
D:=B DIV 4;
E:=B MOD 4;
F:=(B+8) DIV 25;
G:=(B-F+1) DIV 3;
H:=(19*A+B-D-G+15) MOD 30;
I:=C DIV 4;
K:=C MOD 4;
L:=(32+2*E+2*I-H-K) MOD 7;
M:=(A+11*H+22*L) DIV 451;
N:=(H+L-7*M+114) DIV 31;
P:=(H+L-7*M+114) MOD 31;
P:=P+1;
WRITELN(OUTPUT,'EL DOMINGO DE RESURRECCION DEL ANIO ',ANIO:4,
        ' ES: ');
WRITELN(OUTPUT,'MES : ',N:2,'  DIA : ',P:3);
END.
```

```
DAME EL ANIO EN QUE DESEAS CONOCER LA PASCUA : 1984
EL DOMINGO DE RESURRECCION DEL ANIO 1984 ES:
MES :  4  DIA :  22
$

  RUN PASCUA
DAME EL ANIO EN QUE DESEAS CONOCER LA PASCUA : 1600
EL DOMINGO DE RESURRECCION DEL ANIO 1600 ES:
MES :  4  DIA :  21
$

  RUN PASCUA
DAME EL ANIO EN QUE DESEAS CONOCER LA PASCUA : 1810
EL DOMINGO DE RESURRECCION DEL ANIO 1810 ES:
MES :  4  DIA :  22
$

  RUN PASCUA
DAME EL ANIO EN QUE DESEAS CONOCER LA PASCUA : 1750
EL DOMINGO DE RESURRECCION DEL ANIO 1750 ES:
MES :  3  DIA :  27
$
```

TRIANGULOS

```
PROGRAM TRIANGULO (INPUT,OUTPUT);

  A,B,C,AUX : REAL;

BEGIN
WRITE('TECLEE LAS CANTIDADES A REVISAR: ');
READLN(INPUT,A,B,C);
WRITELN;
WRITELN;
WHILE (A<>0) OR (B<>0) OR (C<>0) DO
     BEGIN
     AUX:=(A+B+C)/2;
     IF ((AUX-A)*(AUX-B)*(AUX-C)) <= 0 THEN
          WRITELN(OUTPUT,'NO FORMAN UN TRIANGULO')
     ELSE
          IF (A=B) AND (B=C) THEN
               WRITELN(OUTPUT,'FORMAN UN TRIANGULO EQUILATERO')
          ELSE
               BEGIN
               IF (A=B) OR (B=C) OR (C=A) THEN
                    WRITELN(OUTPUT,'FORMAN UN TRIANGULO ISOSCELES')
               ELSE
                    WRITELN(OUTPUT,'FORMAN UN TRIANGULO ESCALENO')
               (*ENDIF*);
               WRITELN;
               A:=SQR(A);
               B:=SQR(B);
               C:=SQR(C);
               IF (A+B = C) OR (B+C = A) OR (C+A = B) THEN
                    WRITELN(OUTPUT,'TRIANGULO RECTANGULO')
               ELSE
                    WRITELN(OUTPUT,'NO ES TRIANGULO RECTANGULO')
               (*ENDIF*)
               END
     (*ENDIF*);
     WRITELN;
     WRITELN;
     WRITE(OUTPUT,'TECLEE LAS TRES CANTIDADES A REVISAR: ');
     READLN(INPUT,A,B,C);
     WRITELN;
     WRITELN;
     END
(*ENDWHILE*);
END.
```

```
 RUN TRIANGS
TECLEE LAS CANTIDADES A REVISAR: 3 4 5

FORMAN UN TRIANGULO ESCALENO

TRIANGULO RECTANGULO

TECLEE LAS TRES CANTIDADES A REVISAR: 2 2 2

FORMAN UN TRIANGULO EQUILATERO

TECLEE LAS TRES CANTIDADES A REVISAR: 0 0 9

NO FORMAN UN TRIANGULO

TECLEE LAS TRES CANTIDADES A REVISAR: 5 6 8

FORMAN UN TRIANGULO ESCALENO

NO ES TRIANGULO RECTANGULO

TECLEE LAS TRES CANTIDADES A REVISAR: 0 0 0

$
```

TIENDA DE DEPARTAMENTOS

Supongamos que se tiene una cadena de tiendas del tipo de supermercado, y cada tienda a su vez está compuesta por 'n' departamentos distintos (ferretería, aparatos eléctricos, blancos, ropa de caballeros, ropa de damas, etc.). Se quiere desarrollar un algoritmo que nos permita conocer el monto de las ventas tanto por tienda como por departamento, es decir, poder saber cuánto vende cada tienda contabilizando el total de ventas en sus diferentes departamentos, así como poder saber cuánto venden cada uno de los departamentos en todas las tiendas (por ejemplo, cuánto vende el departamento de ferretería en todas las tiendas).

Como primer paso, podemos construir una 'tabla' que contenga la siguiente información:

	TIENDA 1	TIENDA	TIENDA N
Departamento 1	venta (1,1)	venta (1,2)	venta (1,n)
Departamento 2	venta (2,1)	venta (2,2)	venta (2,n)
.	.	.	.
.	.	.	.
.	.	.	.
Departamento N	venta (n,1)	venta (n,2)	venta (n,n)

Llamaremos columnas a los datos cuyo encabezado es Tienda 1, Tienda 2,...,Tienda N, es decir a los datos que representan las ventas por tienda en los diferentes departamentos, y llamaremos renglones a los datos que representan las ventas por departamento en las distintas tiendas. Por último, cada dato que aparece dentro del cuerpo de la tabla se leerá de la siguiente manera:

Venta (1,1) representa el total de ventas del departamento 1 en la tienda 1.

Venta (1,2) representa el total de ventas del departamento 1 en la tienda 2.

Venta (1,N) representa el total de ventas del departamento 1 en en la tienda N.

De la misma forma:

Venta (2,1) representa el total de ventas del departamento 2 en la tienda 1.

Venta (2,N) representa el total de ventas del departamento 2 en la tienda N.

Venta (N,N) representa el total de ventas del departamento N en la tienda N.

La 'Tabla' así construida es un arreglo de 2 dimensiones conocido como matriz, donde cualquier elemento dentro de este arreglo puede localizarse fácilmente por su posición es decir, por medio de su 'renglón' y 'columna'.

Ahora bien, la suma de los elementos de cada renglón nos dará el total de ventas de un departamento en las diversas tiendas, y la suma de las columnas nos dará el total de ventas por tienda. Por último, si sumamos el total de ventas por tienda encontraremos el total de ventas de todas las tiendas (de toda nuestra cadena), este último resultado puede también obtenerse sumando el total de ventas por departamento. Expresando gráficamente lo antes expuesto tendremos:

	Tienda 1	Tienda 2	Tienda N	Total de Ventas por Depto.
Depto 1	Venta(1,1)	Venta(1,2)	Venta(1,N)	Total de Ventas Depto 1
Depto 2	Venta(2,1)	Venta(2,2)	Venta(2,N)	Total de Ventas Depto 2
Depto N	Venta(N,1)	Venta(N,2)	Venta(N,N)	Total de Ventas Depto N
Total de ventas por tienda	Total de Ventas Tienda 1	Total de Ventas Tienda 2	Total de Ventas Tienda N	Total de Ventas en la Cadena de Tiendas

Pseudocódigo Tienda de Departamentos:

```
SEQ: Leer Número de Tiendas (Máximo 10)
SEQ: Leer Número de Departamentos por tienda (Máximo 10)
SEQ: Leer Elementos de la Matriz (Venta (1,1), Venta (1,2)...etc.)
SEQ: Calcula total de Ventas por Departamento e Imprime totales
SEQ: Calcula total de Ventas por Tienda e Imprime totales
SEQ: Calcula total de Ventas de la Cadena de tiendas e Imprimelo.
```

```
PROGRAM TIENDADEPTOS (INPUT,OUTPUT);

CONST
     MAXIMO    = 11;
     CAMPO     =  8;
     PRECISION =  2;

TYPE
     DIMENSION = 1..MAXIMO;

VAR
     MATRIZ : ARRAY [DIMENSION,DIMENSION] OF REAL;
     CONTADOR   ,
     NUMTIENDAS,
     NUMDEPTOS  ,
     RENGLON    ,
     COLUMNA    : DIMENSION;
     TOTDEPTOS  ,
     TOTTIENDAS,
     TOTAL      ,
     DATO       : REAL;

BEGIN
WRITELN(OUTPUT,'TECLEE EL NUMERO MAXIMO DE TIENDAS ',
          '[MAXIMO 10]');
READ(INPUT,NUMTIENDAS);
WRITELN(OUTPUT,'TECLEE EL NUMERO DE DEPARTAMENTOS POR TIENDA'
          '[MAXIMO 10]');
READ(INPUT,NUMDEPTOS);

(* INICIALIZACION DE MATRIZ Y CONTADORES *)

FOR RENGLON:=1 TO NUMDEPTOS DO
     FOR COLUMNA:=1 TO NUMTIENDAS DO
          MATRIZ[RENGLON,COLUMNA]:=0;
     (*ENDFOR*)
(*ENDFOR*)
CONTADOR:=0;
TOTDEPTOS:=0;
TOTTIENDAS:=0;
TOTAL:=0;

(* LECTURA DE DATOS *)

WRITELN(OUTPUT,'TECLEE LOS DATOS DE LAS VENTAS');
FOR RENGLON:=1 TO NUMDEPTOS DO
     FOR COLUMNA:=1 TO NUMTIENDAS DO
          BEGIN
          READ(INPUT,DATO);
          MATRIZ[RENGLON,COLUMNA]:=DATO;
          END
     (*ENDFOR*)
(*ENDFOR*);

   CALCULO DEL TOTAL DE VENTAS POR DEPARTAMENTO *)

FOR RENGLON:=1 TO NUMDEPTOS DO
     BEGIN
     FOR COLUMNA:=1 TO NUMTIENDAS DO
          TOTDEPTOS:=TOTDEPTOS+MATRIZ[RENGLON,COLUMNA];
     (*ENDFOR*)
```

```
    MATRIZ[RENGLON,MAXIMO]:=TOTDEPTOS;
    WRITELN(OUTPUT,'TOTAL DEPARTAMENTO ',CONTADOR:3,': ',
            TOTDEPTOS:CAMPO:PRECISION);
    WRITELN;
    TOTDEPTOS:=0;
    END
(*ENDFOR*);

(* CALCULO DEL TOTAL DE VENTAS POR TIENDA *)

CONTADOR:=0;
FOR COLUMNA:=1 TO NUMTIENDAS DO
    BEGIN
    FOR RENGLON:=1 TO NUMDEPTOS DO
        TOTTIENDAS:=TOTTIENDAS+MATRIZ[RENGLON,COLUMNA];
    (*ENDFOR*)
    CONTADOR:=SUCC(CONTADOR);
    MATRIZ[MAXIMO,COLUMNA]:=TOTTIENDAS;
    WRITELN(OUTPUT,'TOTAL TIENDA ',CONTADOR:3,': ',
            TOTTIENDAS:CAMPO:PRECISION);
    WRITELN;
    TOTTIENDAS:=0;
    END
(*ENDFOR*);

(* CALCULO DE TOTALES *)

FOR RENGLON:=1 TO NUMTIENDAS DO
    TOTAL:=TOTAL+MATRIZ[RENGLON,MAXIMO]
(*ENDFOR*);
WRITELN(OUTPUT,'EL TOTAL DE VENTAS DE LA CADENA DE TIENDAS',
        ' ES : ',TOTAL:CAMPO:PRECISION);
WRITELN;
END.
$
```

```
TECLEE EL NUMERO MAXIMO DE TIENDAS [MAXIMO 10]

4
TECLEE EL NUMERO DE DEPARTAMENTOS POR TIENDA[MAXIMO 10]

3
TECLEE LOS DATOS DE LAS VENTAS

1962
324.5
17.90
1984.85
23452.10
7855
98324
92354.25
12.50
9823
75535.75
11932
TOTAL DEPARTAMENTO   1:  4289.25

TOTAL DEPARTAMENTO   2: 221985.36

TOTAL DEPARTAMENTO   3: 97303.25

TOTAL TIENDA   1: 25426.60

TOTAL TIENDA   2: 18002.50

TOTAL TIENDA   3: 173877.66

TOTAL TIENDA   4: 106271.10

EL TOTAL DE VENTAS DE LA CADENA DE TIENDAS ES : 323577.88

?
```

PROBLEMA DE LAS 8 REINAS

El problema de las 8 reinas consiste en acomodar, en un tablero de ajedrez a 8 damas o reinas, sin que ninguna amenace a otra. Existen más de 90 soluciones diferentes y una de ellas es la que se muestra a continuación:

	0	1	2	3	4	5	6	7
0						D		
1				D				
2							D	
3	D							
4			D					
5					D			
6		D						
7								D

Como es conocido, las reinas amenazan la columna y renglón en que se encuentran, así como diagonalmente.

Existen varias formas de resolver este problema y, la que aquí se presenta, es una que proporciona soluciones parciales, es decir, no obtienen todos las soluciones, sino algunas de ellas.

El programa funciona de la siguiente forma:

- Se pregunta al usuario un valor de 'semilla' para iniciar la generación de números pseudoaleatorios (entre 0 y 7).

- Dado que en cada columna sólo puede haber una reina, cada solución puede verse como una permutación de 8 elementos, tomados de 8 en 8. Por ejemplo, la solución mostrada anteriormente, se puede visualizar como:

3,6,4,1,5,0,2,7

que comprenden a los renglones, y su posición corresponde a las columnas, es decir, 3 en la columna 0, 6 en la columna 1, 4 en la 2, etc. La posible solución se guarda en el subíndice 8 del arreglo 'vector', que es un arreglo de conjuntos.

- La condición primera para añadir un elemento a este conjunto es que no exista ya, lo que asegura que las damas no se amenacen horizontalmente (la amenaza vertical queda eliminada automáticamente, ya que la posición del elemento indica la columna que le corresponde).

- La segunda condición es verificar si existe alguna amenaza diagonalmente, lo cual se lleva a cabo mediante el procedimiento 'condicion', quien, al agregarse un nuevo elemento al conjunto de la solución, llena a su vez los conjuntos (del 1 al 7 en el arreglo 'vector') que contienen las casillas amenazadas diagonalmente con la inserción de una nueva reina. Veamos un ejemplo:

Considerando la solución presentada, lo primero que ocurrió fue la inclusión de un 3 en el conjunto V[8]; ésto hizo que el conjunto V[1] contuviera un 2 y un 4, que son las casillas amenazadas por 3 en la columna 1. De la misma forma el conjunto V[2] contiene ahora un 1 y un 5, que son las casillas amenazadas diagonalmente por 3 en la columna 2; el conjunto V[3] contiene un 0 y un 6 y el conjunto V[4] contendrá un 7.

Cada vez que se propone un nuevo elemento para la solución y éste cumple con las condiciones de amenaza, éstos se ven incrementados para la restricción de entradas a futuros elementos.

Como la búsqueda de posiciones es aleatoria, en muchos casos se encuentran configuraciones parciales que nunca llevarán a una solución, por lo cual el programa hace hasta 50 intentos de agregar un nuevo elemento, si después de ésto no se ha avanzado, la permutación se desecha y se reinicia el procedimiento desde un principio.

```
PROGRAM REINAS (INPUT,OUTPUT);

  TE
    POSICION = 0..7;
    PERMUTACION = SET OF POSICION;
    VECTOR    = ARRAY[0..8] OF PERMUTACION;

VAR
    SEMILLA,
    A       ,
    N       ,
    SAL     ,
    COL     ,
    L       ,
    P1      ,
    P2      ,
    C       ,
    K       ,
    CONT    ,
    J       : INTEGER;
    DAMA    : POSICION;
    V       : VECTOR ;

FUNCTION RANDOM(VAR SEED:INTEGER):INTEGER;
    BEGIN
    RANDOM:=SEED;
    SEED:=(25173*SEED+13849) MOD 32767;
END
  RANDOM*);

PROCEDURE CONDICION;
    BEGIN
    WRITE(OUTPUT,' ',DAMA:2);
    V[8]:=V[8]+[DAMA];
    C:=0;
    K:=SUCC(K);
    L:=SUCC(L);
    FOR COL:=K TO 7 DO
        BEGIN
        C:=SUCC(C);
        P1:=DAMA+C;
        P2:=DAMA-C;
        IF P1 IN V[0] THEN
            V[COL]:=V[COL]+[P1]
        (*NOELSE*)
        (*ENDIF*);
        IF P2 IN V[0] THEN
            V[COL]:=V[COL]+[P2];
        (*NOELSE*)
        (*ENDIF*);
        END;
    (*ENDFOR*)
END
  CONDICION*);

(*********************************)
(*                               *)
(*      PROGRAMA PRINCIPAL       *)
(*                               *)
(*********************************)
```

```
VE0]:=[0,1,2,3,4,5,6,7];
WRITE(OUTPUT,'SEMILLA : (PARA TERMINAR : 0) ');
A:=0;
READ(INPUT,SEMILLA);
W  LE (SEMILLA > 0) DO
     BEGIN
     N:=0;
     SAL:=0;
     CONT:=0;
     FOR J:=1 TO 8 DO V[J]:=L];
     DAMA:=RANDOM(SEMILLA) MOD 8;
     C:=0;
     K:=0;
     L:=0;
     A:=SUCC(A);
     WRITE(A,'.-');
     CONDICION;
     REPEAT
          DAMA:=RANDOM(SEMILLA) MOD 8;
          WHILE (DAMA IN V[8]) OR (DAMA IN V[L]) AND (SAL=0) DO
               BEGIN
               CONT:=SUCC(CONT);
               IF (CONT >= 50) THEN
                    SAL:=1
               (*NOELSE*)
               (*ENDIF*);
               DAMA:=RANDOM(SEMILLA) MOD 8;
               END
          (*ENDWHILE*);
          IF SAL = 0 THEN
               CONDICION
          ELSE
               WRITELN(OUTPUT,'....NO!');
          (*ENDIF*)
     UNTIL (V[8]=[0..7]) OR (SAL=1);
     IF SAL = 0 THEN
          BEGIN
          WRITELN(OUTPUT,'...PERMUTACION VALIDA');
          WRITELN;
          WRITE(OUTPUT,'OTRA SEMILLA : ');
          READLN(SEMILLA);
          A:=0;
          END
     ELSE
          SEMILLA:=SUCC(SEMILLA);
     (*ENDIF*);
END
(*ENDWHILE*)
END.
```

```
RUN REINAS

SEMILLA : (PARA TERMINAR : 0)102
          1.-  6  2  0  7  1  4....NO!
          2.-  1  7  2  0  3....NO!
          3.-  3  7  0  2  5  1  6  4...PERMUTACION VALIDA

OTRA SEMILLA : 180
          1.-  4  7  0  3  5  2....NO!
          2.-  1  5  0  2  6....NO!
          3.-  1  4  0  7  3....NO!
          4.-  5  1  6  0  2  4  7  3...PERMUTACION VALIDA

OTRA SEMILLA : 160
          1.-  0  4  6  1  5  7....NO!
          2.-  1  7  2  6  3  0  4....NO!
          3.-  2  5  3  0  4....NO!
          4.-  4  6  0  2  5  1....NO!
          5.-  1  3  6  2  7  5  0....NO!
          6.-  5  2  0  3  7  4  1....NO!
          7.-  4  7  1  3  6  2....NO!
          8.-  6  4  0  7  5  2....NO!
          9.-  4  7  5  0  6  1....NO!
         10.-  3  6  4  2  5....NO!
         11.-  0  7  4  6  1....NO!
         12.-  1  7  0  6  3  5....NO!
         13.-  7  2  0  3  6  4....NO!
         14.-  4  1  3  0  2  7  5....NO!
         15.-  4  0  3  6  2  5  1....NO!
         16.-  2  5  7  1  4  0....NO!
         17.-  7  0  6  3  5....NO!
         18.-  1  6  2  0  7  4....NO!
         19.-  7  5  0  2  6....NO!
         20.-  2  5  1  4  0  3  6....NO!
         21.-  1  7  0  6  3  5....NO!
         22.-  4  0  7  3  6  2....NO!
         23.-  7  1  4  6  0  3  5....NO!
         24.-  0  2  7  5  1....NO!
         25.-  1  4  7  5  0  2  6....NO!
         26.-  4  7  3  0  2  5  1  6...PERMUTACION VALIDA

OTRA SEMILLA : 1
          1.-  1  7  2  6  3  0  4....NO!
          2.-  0  4  6  1  3  7....NO!
          3.-  7  3  1  6  2  5....NO!
          4.-  3  7  4  1  5  2  6....NO!
          5.-  7  3  0  2  5  1  6  4...PERMUTACION VALIDA

OTRA SEMILLA : 0
$
```

APENDICE F

BIBLIOGRAFIA

BIBLIOGRAFIA

Bibliografía recomendada sobre Pascal.

1) Atkinson, Laurence — Pascal Programming; John Wiley & Sons; E.U.A., 1981

2) Borgerson, Mark — A Basic Programmer's Guide to Pascal; John Wiley & Sons; E.U.A., 1982

3) Bowles, Kenneth — Microcomputer, Problem Solving using Pascal; 1977

4) Cherry, W. George — Pascal Programming Structures, and Introduction to Systematic Programming; Reston Publishing Company (A Prentice-Hall Company), E.U.A., 1980

5) Graham, Roger — Practical Pascal for Microcomputers; John Wiley & Sons; E.U.A., 1983

6) Grogono, Peter — Programming in Pascal; Adison Wesley.

7) Grogono, Peter — Programming in Pascal with Pascal/1000; Adison Wesley.

8) Katzan, Harry Jr. — Invitation to Pascal; Petrocelli.

9) Kieburtz, Richard B. — Structure Programming and Problem Solving with Pascal; 1978

10) Mc. Glynn, Daniel — Fundamentals of Microcomputers Programming, Including Pascal; John Wiley & Sons, 1983

11) Morales, Lozano A. y Sanchis, Llorca T.J. — Programación con el lenguaje PASCAL; Edit. Paraninfo; España, 1980

12) Moore, Lawrie — Foundations of Programming with Pascal. 1980

13) Ríos Sánchez J. Ramón y Bajar R. Victoria — Lenguaje Pascal; Edit Limusa, México

14) Schneider, G. Michael — An Introduction to Programming and Problem Solving with Pascal; 1978

15) Tremblay, Jean Paul — Structure Pascal; 1980

16) Welsh, Jim — Introduction to Pascal; 1979

17) Wilson, J.R. — A Practical Introduction to Pascal; 1979

18) Wirth, Nicklaus — Systematic Programming and Introduction, 1973

19) Wirth, Nicklaus y Jensen, Kathleen — Pascal: User Manual and Report; 2ed, Springer-Verlag

Bibliografía recomendada sobre algunos otros temas expuestos en el curso:

20) Alagic, Suad — The Design of Well Structure and Correct Programs; 1978

21) Burton, E. Philip — A Dictionary of Minicomputing and Microcomputing; John Wiley & Sons; E.U.A., 1982

22) Hill, J.D. y Meek, B.L. — Programming Language Standardisation; John Wiley & Sons, 1982

23) IEEE — American National Standard Pascal Computer Programming Language; John Wiley & Sons; E.U.A., 1983

24) Yourdon, Edward — Techniques of Program Structure and Design; Prentice Hall; E.U.A., 1975