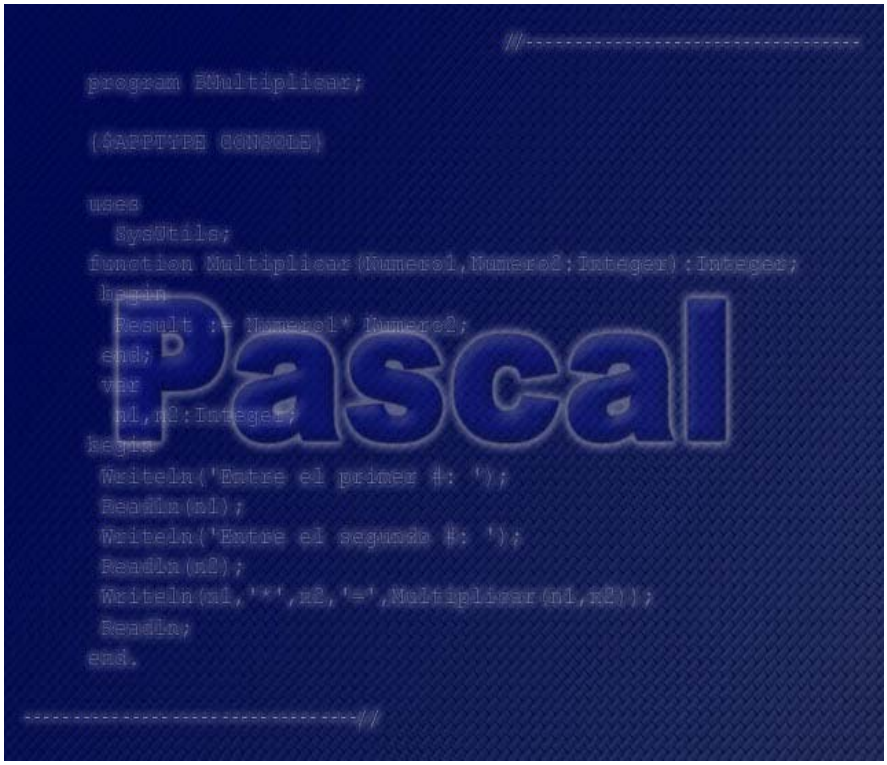
APRENDA PASCAL



AUTORES:

Prof. Rodiert Zayas Alzamora.

Prof. Reinier Escobar Santiesteban.

“AÑO 49 DE LA REVOLUCIÓN”

Índice

Prefacio 1

Capítulo 1 Introducción a la programación en Borland Pascal..... 2
Introducción al Borland Pascal..... 3
Estructura de un programa en pascal..... 3
Palabras reservadas..... 4
Identificadores..... 4
Tipos de datos..... 5
Variables y Constantes..... 6
Operadores aritméticos , lógicos y de relación..... 8
Funciones estándar de Pascal..... 11
Tipos de errores..... 12
Convertir expresiones y fórmulas matemáticas en expresiones Pascal..... 12
Resumen..... 13
Ejercicios de autoevaluación..... 15
Respuestas de ejercicios de autoevaluación..... 17
Ejercicios propuestos..... 18

Capítulo 2 Estructuras básicas de control..... 22
Estructura de control lineal..... 23
Instrucciones de escritura, lectura y asignación..... 23
Solución de problemas lineales..... 25
Estructura de control condicional. Simple, doble y múltiples..... 29
Solución de problemas condicionales..... 33
Generación de números aleatorios..... 37
Estructura de control iterativa. For, While y Repeat..... 39
Solución de problemas iterativos..... 43
Rastreo de programas..... 46
Resumen..... 48
Ejercicios de autoevaluación..... 51
Respuestas de ejercicios de autoevaluación..... 53
Ejercicios propuestos..... 55

Capítulo 3 Estructuras de datos..... 60
Datos estructurados..... 61
Arreglos de una dimensión..... 61
Operaciones con arreglos..... 63
Arreglos de dos dimensiones..... 74
Resumen..... 77
Ejercicios de autoevaluación..... 78
Respuestas de ejercicios de autoevaluación..... 79
Ejercicios propuestos..... 80

Capítulo 4 Registros y Ficheros..... **82**
Registros..... **83**
Definición de tipos de registros..... **83**
Arreglos de registros..... **85**
Ficheros..... **87**
Constructor del tipo de dato file..... **87**
Operaciones con ficheros..... **88**
Resumen..... **92**
Ejercicios de autoevaluación..... **93**
Respuestas de ejercicios de autoevaluación..... **95**
Ejercicios propuestos..... **96**

Capítulo 5 Modularidad..... **98**
Principios de modularidad. Ventajas..... **99**
Procedimientos. Estructura de un procedimiento..... **99**
Paso de parámetro por valor y por referencia..... **102**
Ejemplos de procedimientos creados por el programador..... **104**
Funciones. Estructura de una función..... **105**
Ejemplos de funciones creadas por el programador..... **106**
Recursividad. Funciones recursivas..... **109**
Resumen..... **111**
Ejercicios de autoevaluación..... **112**
Respuestas de ejercicios de autoevaluación..... **114**
Ejercicios propuestos..... **116**

Bibliografía..... **118**

Prefacio

Este material ha sido creado por la gran necesidad de contar con un libro de texto de programación en Pascal para nivel medio, o sea, para alumnos con poca o ninguna experiencia en programación, pero que también ofreciera un tratamiento riguroso a la teoría “trabajo con algoritmos” que se requiere para programar con mayor eficiencia; que estuviera al nivel de las exigencias actuales de los Institutos Politécnicos de Informática (IPI), en especial para los alumnos del primer año.

Pascal desde su creación, se convirtió en una de las herramientas más utilizadas para la enseñanza de la programación, ya que con este se forman los principios básicos de la programación estructurada.

Este libro cuenta con una gran recopilación de ejemplos, ejercicios y programas de diversos campos, para motivar al estudiante a resolver problemas relacionados con la vida real. Todos estos programas y ejemplos aquí presente fueron diseñados para ser ejecutados en Borland Pascal 7.0, pero en cualquier otro compilador de Pascal se pueden utilizar. Cuenta con ilustraciones de algoritmos y programas completos, utilizando una herramienta tan poderosa el mundo de la programación como es la enseñanza mediante ejemplos.

Espero que les sea de mucha utilidad, ya que teniendo la lógica del trabajo en un lenguaje no visual como este, no presentarán ningún problema para cuando lleguen a un lenguaje visual.

Cualquier error que se encuentren en este texto por mínimo que sea, les agradeceríamos (los autores) sus correcciones, críticas, comentarios y sugerencias para mejorar el producto. Por favor dirija toda su correspondencia a cualquiera de estas direcciones electrónicas:

rzayasa@ispij.rimed.cu
rescobar@ipiij.rimed.cu

Cualquier información, ayuda, recomendación o crítica será bien recibida.

Los autores.

Capítulo 1

Introducción a la programación en Borland Pascal

Objetivos

- Familiarizarse con el Borland Pascal y su estructura para programar.
- Identificar los componentes básicos Pascal, tales como, palabras reservadas, identificadores, así como los tipos de datos fundamentales.
- Declarar variables y definir constantes.
- Conocer los operadores aritméticos, lógicos y de relación.
- Familiarizarse con las funciones básicas del lenguaje.
- Convertir expresiones matemáticas en expresiones Pascal.

Pensamientos

◘ Nada es más valioso que el día de hoy.
Goethe.

◘ Después de escalar una montaña muy alta
descubrimos que hay muchas otras montañas por escalar.
Nelson Mandela

◘ Mucha gente persigue lo que posee y está
esclavizada por las mismas cosas que desea conseguir.
Anwar el Sadat

◘ No hay caminos para la paz, la paz es el camino.
Mahatma Gandhi.

◘ Estar en paz consigo mismo es el medio más seguro
de comenzar a estarlo con los demás.
Fray Luis de León

Sinopsis

1.1 Introducción al Pascal.
1.2 Estructura de un programa en Pascal.
1.3 Palabras reservadas.
1.4 Identificadores.
1.5 Tipos de datos.
1.6 Variables y Constantes.
1.7 Operadores aritméticos , lógicos y de relación.
1.8 Funciones estándar de Pascal.
1.9 Tipos de errores.
1.10 Convertir expresiones y fórmulas matemáticas en expresiones Pascal.
1.11 Resumen.
1.12 Ejercicios de autoevaluación.
1.13 Respuestas de ejercicios de autoevaluación.
1.14 Ejercicios propuestos.

1.1 Introducción al Pascal.

Pascal es un lenguaje de programación de alto nivel, diseñado a principios de la década de 1970 por el profesor Niklaus Wirth como un elemento de aprendizaje, se denominó así en honor a Blaise Pascal, matemático y filósofo francés del siglo XVII que creó la primera calculadora. Pascal, se trata de un lenguaje estructurado, disponible en numerosas versiones, tanto interpretado como compilado. En este libro se trabaja bajo la base del Borland Pascal 7.0.

De Pascal se derivan diversos lenguajes, entre ellos, el conocidísimo Object Pascal, una versión avanzada orientada a objetos.

La aceptación y el uso de Pascal se incrementó considerablemente en 1984, cuando Borland International (actualmente Inprise Corp.) introdujo Turbo Pascal. Se trataba de un compilador de Pascal, de alta velocidad y bajo costo, para sistemas MS-DOS, entre otros; incluía un entorno de desarrollo IDE, en vez de las tradicionales herramientas en línea de comandos.

En los últimos años su popularidad ha decrecido, aunque sigue comercializándose, y se utiliza muy a menudo en medios educativos. El producto comercial actual más conocido es Borland Delphi, un entorno de programación visual, basado en Turbo Pascal para Windows y que usa las extensiones de Object Pascal. Existen también versiones para desarrollos bajo Linux.

1.2 Estructura de un programa en pascal.

El lenguaje utilizado en Borland Pascal 7.0 (BP7) es estructurado, lo que significa que cada programa requiere una forma específica de escritura para que sea entendido por el compilador. Recuerde que un programa no es más que un conjunto de instrucciones que se le da al ordenador para resolver un problema determinado. En la “Figura 1.1” aparece las partes o módulos del Lenguaje de Programación (LP) BP7.

Cabecera
Declaraciones
Sentencias

Figura 1.1 Partes de un programa en BP7.

Cabecera:
En esta sección de programa solo lleva el nombre del programa.

Ejemplo:

```
Program Nombre;
```

Declaraciones:
Aquí se le indica al compilador todos los indentificadores y unidades que utilizará nuestro programa.

Sentencias:
Aquí se escriben todo el código de instrucciones que realizará nuestro programa. Todas las instrucciones van escritas entre las reservadas begin y end.

La estructura del programa quedaría de la siguiente forma:

Pascal	Pseudocódigo
PROGRAM Nombre; { Cabecera } { Declaraciones } **BEGIN** { Sentencias } **END.**	**Algoritmo** Nombre { Cabecera } { Declaraciones } **Inicio** { Sentencias } **Fin**

Las palabras BEGIN y END representan inicio y fin del programa. Puedes ver la relación entre ambas estructuras.

1.3 Palabras reservadas.

Todo LP cuenta con una serie de palabras reservadas que son las que le permite al programador poder comunicarse con el mismo. Estas palabras reservadas “Figura 1.2” son propias de cada lenguaje, esto, entre otras cosas, lleva consigo a establecer una diferencia marcada entre un lenguaje y otro aunque existen palabras que son muy comunes en todos los lenguajes.

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

Figura 1.2 Palabras reservadas de Pascal.

1.4 Identificadores.

Los "identificadores" son palabras que usaremos para referirnos a una variable, una constante, al nombre del programa, el nombre de una función o de un procedimiento, etc.

Estos nombres o "identificadores" serán combinaciones de letras y/ o números, junto con algunos pocos símbolos especiales, como el guión bajo (_).

Deben seguir las siguientes características:

- No podrán empezar con un número.
- Puede tener cualquier carácter alfabético A..Z, sin Ñ.
- No puede tener tilde en ningún momento.
- No puede tener espacios intermedios.
- No pueden ser palabras reservadas del LP.
- No pueden existir dos identificadores con el mismo nombre.

En la Figura 1.3 se muestran algunos identificadores válidos y otros no válidos.

Identificadores válidos:	**Identificadores no válidos**:
Programa_1	1_Programa
Mi_Programa	Mi Programa
Suma	Sum@
Tarea_2	2 Tarea
Area	Área
Numeros_Primos	#_Primos
ParImpar	Par o Impar
Positivo_Negativo	Positivo_Negativo**.**
AreaTriangulo	ÁreaTriángulo
ArregloN	Array

Figura 1.3 Ejemplo de identificadores válidos (Izquierda) y no válidos (Derecha).

Nota: Pascal no diferencia las minúsculas de las mayúsculas. Ejemplo los identificadores *NUMEROS* y *numeros* son completamente iguales.

1.5 Tipos de datos.

A la hora de guardar los datos en la computadora, estos deben ser numéricos, lógicos, cadenas de caracteres, etc.. Lo que quiere decir que deben ser almacenados con un tipo de dato. En programación, el tipo de dato (data type en Inglés) especifica el rango de valores que puede contener una variable o una constante, y se almacena esa información en la memoria de la computadora. El Pascal puede adquirir muchos tipos diferentes de datos. Entre ellos están los datos simples y los datos estructurados, estos últimos serán objeto de estudio en el "Capítulo 3 Estructuras de datos". Vamos a ver por ahora los datos simples "Figura 1.4".

Los dato de tipo simple, no está compuesto de otras estructuras, que no sean los bits, y que por tanto su representación sobre el ordenador es de forma directa. Los datos simples son elementos individuales(números, caracteres, etc.) que se asocian a identificadores únicos en correspondencia uno a uno.

Pseudocódigo	**Borland Pascal**	**Descripción**
Enteros	Integer	Número entero comprendido entre -32768 hasta 32767.
Real	Real	Es un número real.
Texto	String	Cadena de caracteres, empleado para almacenar y representar mensajes de más de una letra (hasta 255).
Lógico	Boolean	Es una variable lógica, que puede valer *TRUE* (verdadero) o *FALSE* (falso).

Figura 1.4 Tipos de datos estándar de Pascal.

1.6 Variables y Constantes.

Variables:
Una variable equivale a la clásica incógnita "x" que todos hemos usado en las matemáticas, que puede ser cualquier número. Ahora nuestras variables podrán tener cualquier valor (no sólo un número: también podremos guardar textos, fichas sobre personas o libros, etc.) y nombres más largos (y que expliquen mejor su contenido).
Una variable es una posición de memoria a la que nosotros asignamos un nombre y en la que podremos almacenar datos que pueden ser modificados a lo largo de todo el programa.

En Pascal tenemos que declarar las variables que vamos a usar. Esto puede parecer incómodo, pero en la práctica ayuda a conseguir programas legibles y más fáciles de corregir o ampliar. Además, evita los errores que puedan surgir al emplear variables incorrectas.

Para declarar una **variable** "Figura 1.5", tenemos que emplear la palabra clave **var**, luego escribimos el nombre que vamos a dar a la variable, y del tipo de datos que va a almacenar esa variable.

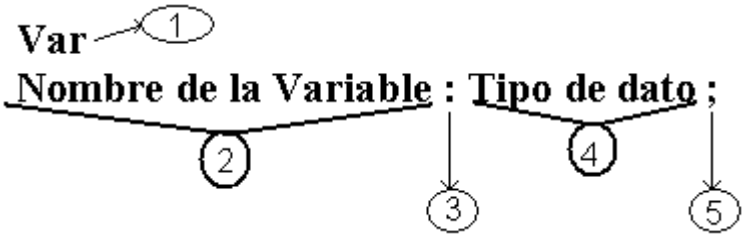


Figura 1.5 Estructura general para declarar variables.

1.- Palabra clave para declarar variables.
2.- Aquí se pone el nombre de la variable a declarar. Los nombres de las variables siguen las reglas que ya habíamos mencionado para los identificadores en general, o sea, el nombre de una variable es un indentificador.
3.- Los dos puntos es un separador e indica el tipo de dato que tendrá la variable a declarar.
4.- Especifica al ordenador qué tipo de valor va a almacenar la variable(números reales, números enteros, cadenas de textos, etc..).
5.- Fin de la declaración.

Ejemplo:
Se tienen las siguientes situaciones:

- Tenemos necesidad de almacenar el nombre y la edad de una persona.

  Respuesta:
  Var
  Nombre: String;
  Edad: Integer;

- Queremos almacenar el valor de las precipitaciones caída de un día.

Respuesta:
Var
  Prec: Integer;

- Almacenar si un estudiante es de la Unión de Jóvenes Comunista (UJC) o no lo es.

Respuesta:
Var
  UJC: Boolean;

Si deseas declarar varias variables de un mismo tipo puedes separarlas con una coma y luego decir el tipo de dato. Ejemplo:

Var
Variable1, Variable2, Variable3: Integer;

En este caso tenemos tres variables que van a almacenar números enteros.

Nota:
En una variable, el valor que queda almacenado en ella es el último que se le asigna. Para entender mejor esto analice el siguiente ejemplo:

♣ Dadas las variables A, B y C diga qué valor queda almacenado en cada una de ellas:
A = 5
B = 8
C = 10
A = B + C
B = A + C
C = A + B

Respuesta:
El valor que queda almacenado en A es 18, en B es 28 y en C es 46.

Al que no le dio así que observe el análisis que les muestro a continuación:

- En las tres primeras líneas A, B y C comienzan con 5, 8 y 10 respectivamente.
- En la línea (A = B + C) se le asigna un nuevo valor a la variable “A”, diciéndole que almacena la suma de las variables B y C, es decir 8 + 10, quedando almacenado en A el número 18.
- En la línea (B = A + C) se le asigna un nuevo valor a la variable “B”, diciéndole que almacena la suma de las variables A y C, es decir 18 + 10, quedando almacenado en B el número 28.
- En la línea (C = A + B) se le asigna un nuevo valor a la variable “C”, diciéndole que almacena la suma de las variables A y B, es decir 18 + 28, quedando almacenado en B el número 46.

Al no asignarle más ningún valor a las variables A, B y C, los valores que quedan almacenados son 18, 28 y 46 respectivamente.

Constantes:
Cuando desarrollamos un programa, nos podemos encontrar con que hay variables que realmente "no varían" a lo largo de la ejecución del programa, sino que su valor es constante. Una constante en programación es similar al valor de la constante Π que todos conocemos, cuyo valor no cambia en ningún momento.
Entonces podemos definir que una constante es una posición de memoria a la que nosotros asignamos un nombre y un valor que no puede ser modificado a lo largo de todo el programa.

En Pascal hay una manera especial de definir las constantes "Figura 1.6", que es con la palabra reservada **const**.

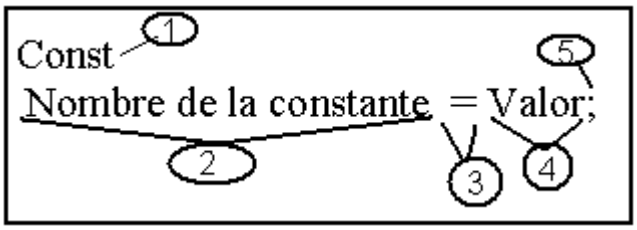


Figura 1.6 Estructura general para definir constantes.

1.- Palabra reservada para definir constantes.
2.- Aquí se pone el nombre de la constante a definir. Los nombres de las constantes siguen las reglas que ya habíamos mencionado para los identificadores en general.
3.- Operador (=) que indica asignación (Solo en estos casos).
4.- El valor que va a tener la constante.
5.- Representa el fin de la definición de la constante.

Ejemplo:
Se tienen las siguientes situaciones:

- Se quiere definir una constante que almacene el valor de Π.
  Respuesta:
  Const
  Pi = 3.14;
- Definir una constante que almacene el descuento de un trabajador que es de $100 mensuales.
  Respuesta:
  Const
  Descuento = 100;

1.7 Operadores aritméticos , lógicos y de relación.

Pascal tiene una serie de operadores que les permitirá desarrollar los cálculos necesarios un tu aplicación. En la Figura 1.7 aparecen los operadores aritméticos que utiliza Pascal. Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales.
Los operadores relación "Figura 1.9" al igual que los operadores de lógicos "Figura 1.10" también juegan un papel fundamental en la elaboración de un programa en Pascal.

Operador	Descripción	Ejemplo	Y	Z	X
+	Adición	X:= Y + Z	5	2	7
-	Sustracción	X:= Y - Z	5	2	3
*	Multiplicación	X:= Y * Z	5	2	10
/	División real	X:= Y / Z	5	2	2.5
div	División entera	X:= Y div Z	5	2	2
mod	Resto de una división	X:= Y mod Z	5	2	1

Figura 1.7 Operadores aritméticos y ejemplo de su utilización.

Puedes observar que en la columna **Ejemplo** de la Figura 1.7 se utilizó el operador **:=** que es el de asignación.

En los primeros tres ejemplo espero que no tengan ningún tipo de duda ya que a la variable **X** se le asigna un valor, el de la Suma, la Resta y la Multiplicación de las variables **Y** y **Z** respectivamente.

Los ejemplos 4 y 5 tienen una diferencia significativa y es que en el caso del **/** es una división real y por lo tanto el valor de la variable **X** tiene que ser Real, y el **div** realiza una división entera y con esto me refiero a que al dividir el toma el resultado de la división y elimina los lugares después de la coma, dejando así solo el valor entero del resultado, en este caso **X** tiene que ser declara como Integer.

El operador **mod** devuelve el resto o residuo de una división de enteros.

Analice el ejemplo que se muestra en la Figura 1.7 y observe la explicación en la Figura 1.8.

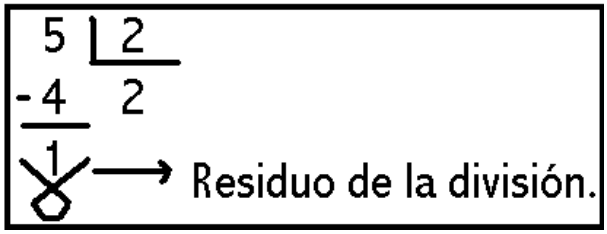


Figura 1.8 Ejemplo del valor devuelto al usar el operador mod.

Operador	Descripción
=	Igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Distinto de

Figura 1.9 Operadores de relación en Pascal.

Ejemplo del uso de operadores de relación:

♣ Dadas las variables A y B, diga cuando la expresión puede ser evaluada como verdadera y cuando como falsa. Ejemplifique.

a) A = B b) A > B c) A < B
d) A >= B e) A <= B f) A <> B

Repuestas:

a) Verdadera si A y B son iguales, ejemplo A = 8 y B = 8.
   Falsa si A y B son diferentes, ejemplo A = 5 y B = 8.
b) Verdadera si A es mayor que B, ejemplo A = 10 y B = 8.
   Falsa si A es menor o igual que B, ejemplo A = 3 y B = 5.
c) Verdadera si A es menor que B, ejemplo A = 10 y B = 18.
   Falsa si A es mayor o igual que B, ejemplo A = 6 y B = 6.
d) Verdadera si A es mayor o igual que B, ejemplo A = 10 y B = 10.
   Falsa si A es menor que B, ejemplo A = 1 y B = 6.
e) Verdadera si A es menor o igual que B, ejemplo A = 10 y B = 10.
   Falsa si A es mayor que B, ejemplo A = 12 y B = 6.
f) Verdadera si A es diferente a B, ejemplo A = 5 y B = 8.
   Falsa si A igual que B, ejemplo A = 1 y B = 1.

En los operadores de relación espero que no halla ninguna duda, así que prestaré toda mi atención en los operadores lógicos.

Operador	Descripción	Significado
or	**o**	La expresión es verdadera (True) si al menos un operando es verdadero y falsa (False) si todos los operando son falsos.
and	**y**	La expresión es falsa si al menos un operando es falso y verdadera si todos los operando son verdaderos.
xor	**o exclusivo**	La expresión es verdadera si solo un operando es verdadero y falsa en cualquier otro caso.
not	**negación**	Este operador se usa como prefijo para negar un operando booleano.

Figura 1.10 Operadores lógicos en Pascal.

Or: En español significa “o”. Con este operador el valor de la expresión es verdadero si al menos una de las preguntas es verdadera de lo contrario es falsa, o como ya estudiaste en las primeras unidades cuando estamos ante una disyunción Figura 1.11 a.

And: En español significa “y”. Con este operador el valor de la expresión es verdadero si todas las proposiciones son verdaderas, de lo contrario es falsa, o como se dice cuando estamos ante una conjunción Figura 1.11 b.

Xor: En español significa “o exclusivo”, a diferencia del “**or”,** este solo es verdadero cuando existe un solo valor verdadero entre varias proposiciones o preguntas lógicas, como quieras llamarlas.

Not: En español significa “no” o “negación”, y se utiliza para negar el valor de una expresión Figura 1.11 c.

p	q	p v q
V	V	V
V	F	V
F	V	V
F	F	F

a

p	q	p ^ q
V	V	V
V	F	F
F	V	F
F	F	F

b

p	- p
V	F
V	F
F	V
F	V

c

Figura 1.11 Tablas de la verdad. a)Disyunción. b) Conjunción. c) Negación.

Nota:
Para evitar ambigüedades en el orden de realización de las operaciones lógicas, los operandos boléanos deben ir encerrados entre paréntesis, aunque en ocasiones no necesariamente los lleve.

Ejemplo de uso de los operadores lógicos:
♣ Dadas las variables A y B, diga si la expresión es evaluada como verdadera o como falsa si conocemos que A = -2 y B = 120. Explique.

a) (A > 0) or (B > 100) b) (A <= 0) and (B > 100)
c) (A = -2) xor (B = 120) d) not(A <> B)

Respuestas:
a) Verdadera. Porque el segundo operando es verdadero.
b) Verdadera. Porque los dos operando son verdaderos.
c) Falsa. Porque uno y solo de los operando tiene que ser verdadero.
d) Falsa. Porque el valor del operando es verdadero pero el operador not niega su valor y por tanto el valor resultante es falso.

1.8 Funciones estándar de Pascal.

Pascal al igual que todo lenguaje de programación cuenta con una serie de funciones que hacen posible un funcionamiento adecuado del mismo sin tener que crear nuevamente conjuntos de códigos para que realicen una determinada tarea. En la Figura 1.12 se muestran algunas de las funciones estándar, las que estudiarás su uso en otros capítulos.

Función	Efecto
abs(x)	Calcula el valor absoluto de x
chr(x)	Determina el carácter representado por x en el código ASCII.
cos(x)	Calcula el coseno de x en radianes.
sen(x)	Calcula el seno de x en radianes.
odd(x)	Determina si x es impar.
round(x)	Redondea el valor de x al entero más próximo.
sqr(x)	Calcula el cuadrado de x.
sqrt(x)	Calcula la raíz cuadrada de x (x>0).
trunc(x)	Trunca x (suprime la parte decimal).
copy	Retorna una subcadena de una cadena dada.
pos	Localiza una determinada cadena dentro de otra.
lenght	Devuelve el tamaño de una cadena.
concat	Une las cadenas pasadas como parámetro.

Figura 1.12 Funciones estándar de Pascal.

Existen otras funciones estándar que se mostrarán en su debido tiempo.

1.9 Tipos de errores.

Cuando terminas de escribir el código de tu programa este necesita ser compilado para poder visualizar tu creación y así ver si cumple con las expectativas deseadas. En BP7 la compilación se realiza cuando se ejecuta el mismo, presionando la combinación de teclas Ctrl. + F9.

El compilador es un programa que traduce un Lenguaje de Alto Nivel como C o Pascal a un programa de lenguaje de máquina. Un compilador traduce todo un programa a lenguaje de máquina, y simultáneamente verifica la sintaxis y muestra mensajes de error o de advertencia, según sea apropiado. Por tanto, el programa sólo se puede ejecutar después que este proceso se haya completado, en lo cual difiere con un intérprete, que traduce una línea a la vez e inmediatamente la ejecuta.

Errores sintácticos o gramaticales:
Varios de los errores más usuales que surgen de la compilación suelen ser variables o constantes incorrectamente declaradas, referencias a variables no declaradas, uso incorrecto de los puntos y comas, entre otros que ya chocarás con ellos en el transcurso del aprendizaje.

Errores de ejecución:
Este tipo de error se ocasionan cuando utilizas tipos de datos incompatibles en un argumentos de una función estándar usada, una división por 0, intentar cambiar el valor a una constante, uso de variables cuyo tipo no corresponda con el tipo de dato que se le asigna, etc.

Error lógico:
Este es error más difícil de divisar de todos, ya que al compilarse el programa, no se emiten mensajes de error ya que la estructura de programa y la implementación de la escritura está correcta, pero no ofrece el resultado esperado. En estos casos el programador debe hacer un chequeo de su programa sentencia a sentencia y que cada una de estas se correspondan con el algoritmo diseñado.

Mientras más horas se le dedique a la programación más habilidades se logran y disminuyen las posibilidades errores sintácticos y de ejecución al igual que los errores lógicos.

1.10 Convertir expresiones y fórmulas matemáticas en expresiones Pascal.

Este tema va a estar destinado a la adquisición de habilidades en la traducción de expresiones algebraicas y lógicas en expresiones del lenguaje de programación Pascal. Esto sin duda, favorecerá en gran medida a adelantar en los próximos capítulos cuando se empiece la programación.
Teniendo la base de estas conversiones les ayudará a disminuir los errores de programación en la ejecución de sus programas.

Aquí aparecerán una serie de ejemplos de conversión de expresiones matemáticas en expresiones Pascal.

Representa las siguientes expresiones en una expresión Pascal:

- Suma de tres números diferentes.

R/: **Suma:= N1 + N2 + N3;**

- x= 3y + 5

R/: **x:= 3 * y + 5;**

- $y = \frac{a + b}{z - x}$

R/: **y:= (a + b) / (z - x);**

- Promedio de cuatro notas de un alumno.

R/: **Prom:= (Nota1 + Nota2 + Nota3 + Nota4) / 4;**

- Área del triángulo.

R/: **Area:= A * B / 2;**

- Cuadrado de un número aumentado en 5.

R/: **C:= Sqr(Num) + 5;** ó **C:= Num * Num + 5;**

- Sucesor y antecesor de un número.

R/: **Sucesor:= Num + 1;**
Antecesor:= Num – 1;

- Una persona del sexo masculino y de edad menor que 25.

R/: **(Sexo = 'M') and (Edad < 25);**

- Número positivo o que sea par.

R/: **(Num > 0) or (Num mod 2 = 0);**

1.11 Resumen.

- Pascal fue diseñado para enseñar programación estructurada en contextos académicos.
- La programación estructurada es un método disciplinado de escribir programas que resulten claros, demostrablemente correctos y fáciles de modificar.
- De Pascal derivan diversos lenguajes de programación como Ada y Object Pascal.
- Un programa en Pascal está dividido en tres partes o zonas: Cabecera, Declaraciones y Sentencias.

- Los identificadores son combinaciones de letras y/ o números, junto con algunos pocos símbolos especiales. Deben seguir las siguientes características:
    - No podrán empezar con un número.
    - Puede tener cualquier carácter alfabético A..Z, sin Ñ ni acentos.
    - No puede tener acentuación.
    - No puede tener espacios intermedios.
    - No pueden ser palabras reservas del LP.
    - No pueden existir dos identificadores con el mismo nombre.
- El tipo de dato (data type en Inglés) especifica el rango de valores que puede contener una variable o una constante, y se almacena esa información en la memoria de la computadora.
-

Pseudocódigo	Borland Pascal	Descripción
Enteros	Integer	Número entero comprendido entre -32768 hasta 32767.
Real	Real	Es un número real.
Texto	String	Cadena de caracteres, empleado para almacenar y representar mensajes de más de una letra (hasta 255).
Lógico	Boolean	Es una variable lógica, que puede valer *TRUE* (verdadero) o *FALSE* (falso).

Figura 1.4 Tipos de datos estándar de Pascal.

- Una variable es una posición de memoria a la que nosotros asignamos un nombre y en la que podremos almacenar datos que pueden ser modificados a lo largo de todo el programa.
- En una variable, el valor que queda finalmente almacenado en ella es el último valor que se le asigna.
- Una constante es una posición de memoria a la que nosotros asignamos un nombre y un valor que no puede ser modificado a lo largo de todo el programa.
- Operadores aritméticos.

Operador	Descripción
+	**Adición**
-	**Sustracción**
*****	**Multiplicación**
/	**División real**
div	**División entera**
mod	**Resto de una división**

- El operador de asignación que utiliza Pascal es **:=**.
- Operadores de relación en Pascal.

Operador	Descripción
=	**Igual a**
>	**Mayor que**
<	**Menor que**
>=	**Mayor o igual que**
<=	**Menor o igual que**
<>	**Distinto de**

- Or: En español significa “o”. Con este operador el valor de la expresión es verdadero (True) si al menos una de las proposiciones es verdadera de lo contrario se evalúa como falsa (False).
- And: En español significa “y”. Con este operador el valor de la expresión es verdadero si todas las proposiciones son verdaderas, de lo contrario es falsa.
- Xor: En español significa “o exclusivo”, a diferencia del “**or”,** este solo es verdadero cuando existe un solo valor verdadero entre varias proposiciones o preguntas lógicas, como quieras llamarlas.
- Not: En español significa “no” o “negación”, y se utiliza para negar el valor de una expresión.
- Para evitar ambigüedades en el orden de realización de las operaciones lógicas, los operandos boléanos deben ir encerrados entre paréntesis.
- Un compilador traduce todo un programa a lenguaje de máquina, y simultáneamente verifica la sintaxis y muestra mensajes de error o de advertencia, según sea apropiado. Por tanto, el programa sólo se puede ejecutar después que este proceso se haya completado.
- Un intérprete es un traductor incluido en un lenguaje de programación, el cual convierte el código fuente de un programa de alto nivel (línea por línea) en enunciados del lenguaje de máquina.
- La diferencia de un compilador y un intérprete radica en que compilador debe traducir el programa completo antes de que pueda comenzar la ejecución, un intérprete traduce y ejecuta cada línea a la vez. El resultado final de una compilación es un ejecutable.
- Los tipos de errores son:
- Errores sintácticos o gramaticales.
- Errores de ejecución.
- Errores lógicos.

1.12 Ejercicios de autoevaluación.

1.1 Llene los espacios en blanco de cada una de los siguientes incisos:

a) Pascal es un _____ de alto nivel, diseñado a principios de la década de 1970 por el profesor _____.
b) Se denominó Pascal en honor a _____, matemático y filósofo francés del siglo XVII que creó la primera calculadora.
c) Las partes de un programa en BP7 son _____, _____ y _____.

1.2 Coloque verdadero o falso en cada una de las siguientes proposiciones en correspondencia con las características que debe seguir un identificador:

a) _____ Podrán empezar con un número.
b) _____ Pueden tener espacios intermedios.
c) _____ Pueden tener tilde en algunos momentos.
d) _____ No pueden ser palabras reservadas del LP.
e) _____ Pueden existir dos identificadores con el mismo nombre.
f) _____ Puede tener cualquier carácter alfabético A..Z, sin Ñ.

1.3 Diga cuáles de los identificadores siguientes pueden son válidos y cuáles no. Para los no válidos (si es que existe alguno)explique por qué no puede serlo:

a) Programa_1
b) Begin
c) @rea
d) 1_Programa
e) triángulo
f) MOD
g) CUADRADO

1.4 Son completamente iguales los identificadores “EJERCICIOS” y “ejercicios”.

1.5 Los tipos de datos se clasifican en _____ y _____.

1.6 Para declarar variables se utiliza la palabra reservada _____ y para definir una constante _____.

1.7 Dadas las variables A, B, C y D diga qué valor queda almacenado en cada una de ellas al finalizar estas líneas:

A = 2
B = Sqr(A)
C = 10
A = B + C
B = A div 2
C = A + B
D = A

1.8 Linealice las siguientes expresiones matemáticas en expresiones Pascal.

a)

$$x = \frac{a + b + c}{a + \frac{b}{c}}$$

b)

$$x = a + b + \frac{c}{d} + \frac{\frac{a}{b - c}}{\frac{a}{b + c}}$$

1.9 En BP7 la compilación se realiza cuando se ejecuta el mismo, presionando la combinación de teclas _____.

1.10 Enlace la columna A con la B según corresponda:

Columna A	Columna B
1.- Error sintáctico o gramatical.	____ Error que se produce cuando utilizas tipos de datos incompatibles en un argumentos de una función determinada, una división por 0, cuando se intenta cambiar el valor a una constante, etc.
2.- Error lógico.	____ Cuando se hace referencias a variables no declaradas. Variables o constantes incorrectamente declaradas, uso incorrecto de los puntos y comas.
3.- Error de ejecución.	____ Error más difícil de distinguir de todos**.** La compilación del programa no emite ningún mensaje de error pero no ofrece el resultado esperado.

1.13 Respuestas de ejercicios de autoevaluación.

1.1
a) Lenguaje de Programación, Niklaus Wirth.
b) Blaise Pascal.
c) Cabecera, Declaraciones y Sentencias.

1.2
a) Falso. Nunca puede un identificador comenzar con número.
b) Falso. No se puede dejar espacios intermedios.
c) Falso. En ningún momento pueden tener tilde.
d) Verdadero.
e) Falso. No pueden existir dos identificadores con nombres iguales.
f) Verdadero.

1.3
a) Válido.
b) No válido. Begin es una palabra reservada de Pascal.
c) No válido. No se puede utilizar símbolos como @, $, !, ¿, ?, entre otros.
d) No válido. No puede comenzar con número.
e) No válido. No se pueden tildar en ningún momento.
f) No válido. Mod es una palabra reservada de Pascal.
g) Válido.

1.4 Si.
1.5 Simples, Estructurados.
1.6 Var, Const.
1.7 A = 14, B = 7, C = 21 y D = 14.

1.8
a) x:= (a + b + c) / (a + b / c);
b) x:= a + b + c / d + (a / (b - c)) / (a / (b + c));

1.9 Ctrl. + F9.
1.10 3, 1, 2.

1.16 Ejercicios propuestos.

1.11 Elabore la estructura de un programa con nombre Calculadora.

1.12 Diga cuáles de los identificadores siguientes son válidos y cuáles no. Para los no válidos (si es que existe alguno) proponga un válido sin que pierda su significado:

Área	IMAGENES	Suma_$	Calle Martí	Calculo
Diccionario	Triangulos=	VALIDARDATOS	EntrarNotas	A_TrianG
ProyectoSOFT	Contar C	A	Array	Juego

1.13 A partir de la definición de variable cite ejemplos donde tengas que hacer uso de las mismas.

1.14 A partir de la definición de constante cite ejemplos donde tengas que hacer uso de estas para ganar en eficiencia y claridad en tu prorama.

1.15 Declare las variables que serán necesarias para resolver las siguientes situaciones:

a) Para almacenar el nombre de una persona.
b) Almacenar el nombre, la edad y el sexo de una persona.
c) Para almacenar un número real.
d) Para almacenar la suma de dos números enteros.
e) Variable que determina si un estudiante pertenece a la UJC o no; y si un trabajador es del PCC o no.
f) Promediar las estaturas de dos estudiantes.
g) Variable que determina si un número es par o no.

1.16 Definas las siguientes constantes:

a) Representa el valor de □.
b) Diez es el valor máximo de una lista.
c) Constante que muestre "*El resultado final es:* ".

1.17 Los siguientes incisos deberán resolverse utilizando las reglas de la aritmética.

a) a = 10 b = 20 c = 10 a = a + 15 b = b + 12 c = a * c ¿Qué valores quedan en las variables a, b y c?	**b)** a = 1 b = 2 c = 3 a = a + 2 b = a + 2 + b c = a + 2 + c a = b / 2 b = b div 2 c = c / 2 ¿Qué valores quedan en las variables a, b y c?
c) a ← 8 b ← 7 c ← 5 d ← 8 a ← a + b – c + d b ← a + b – c + d c ← a + b – c + d d ← a + b – c + d a ← a + b – c + d b ← a + b – c + d c ← a + b – c + d d ← a + b – c + d ¿Qué valores quedan almacenados en las variables a, b c y d?	**d)** a ← 8 b ← 2 c ← 18 d ← 18 a ← a + b b ← a - b c ← a * b d ← a **mod** b a ← a - b b ← Sqr(a) + b c ← a - b d ← a + Sqr(b) ¿Qué valores quedan almacenados en las variables a, b, c y d?

1.18 Linealice las siguientes expresiones (no se olvide que linealizar significa escribir una expresión algebraica en una sola línea). En cada uno de los siguientes incisos escribir el orden en que el computador realizaría las operaciones.

A)	$x = \sqrt{a} + \frac{b}{c+a}$	D)	$x = \frac{a + b + \frac{a}{b}}{\sqrt{5c}}$
B)	$x = \frac{a + b + c}{a^2 + \frac{3b}{c}}$	E)	$x = \frac{a + b + \frac{c}{d * a}}{a + b * \frac{c}{d}}$
C)	$x = a + \frac{a + \frac{a + b}{c + d}}{a + \frac{a}{b}}$	F)	$x = \sqrt{\frac{a}{+ b}} + \frac{5 a^3 \frac{b}{a}}{a -}$

1.19 Complete la siguiente tabla:

Operadores Aritméticos	
Operador	**Descripción**
	Adición
-	
	Multiplicación
/	
div	
	Resto de una división

1.20 Diga el orden de precedencia de los operadores aritmético.

1.21 ¿Cuál es el operador de asignación que utiliza Pascal?

1.22 Diga qué valor queda en A después de leer las siguientes expresiones por separadas.

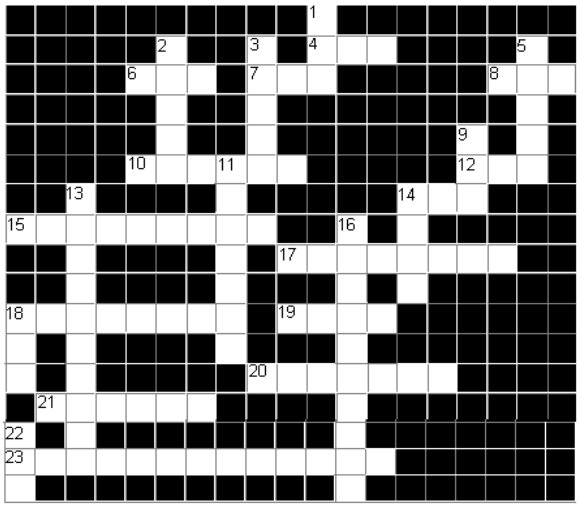
a) A:= 12 + 40 div 2;
b) A:= 10 mod 2;
c) A:= 100 – 60 / (25 + 5) * 40;

1.23 Complete la tabla siguiente:

Operadores de Relación	
Operador	**Descripción**
	Igual a
>	
	Menor que
	Mayor o igual que
<=	
	Distinto de

1.24 Complete el siguiente crucigrama:

1) Operador lógico que la expresión es verdadera cuando todas las proposiciones son verdaderas.
2) Palabra reservada para definir una constante.
3) Palabra reservada para indicar el inicio de un bloque.
4) Operador lógico que expresa negación.
5) Función que redondea el valor de x al entero más próximo.
6) Operador aritmético que devuelve el resto de una división.
7) Palabra reservada para indicar el fin de un bloque.
8) Función que localiza una determinada cadena dentro de otra.
9) Operador lógico que es verdadero cuando existe un solo valor verdadero entre varias expresiones.
10) Tipo de dato que almacena una cadena de caracteres.
11) Tipo de dato que almacena números enteros.
12) Función que determina si un número es impar.
13) Traductor incluido en un lenguaje de programación, el cual convierte el código fuente en enunciados del lenguaje de máquina.
14) (Horizontal) Función que haya el cuadrado de un número.
(Vertical) Función que haya la raíz cuadrada de un número.
15) Identificador al que le asignamos un valor que no puede ser modificado a lo largo de todo el programa.
16) Programa que traduce todo un programa a lenguaje de máquina, y simultáneamente verifica la sintaxis.



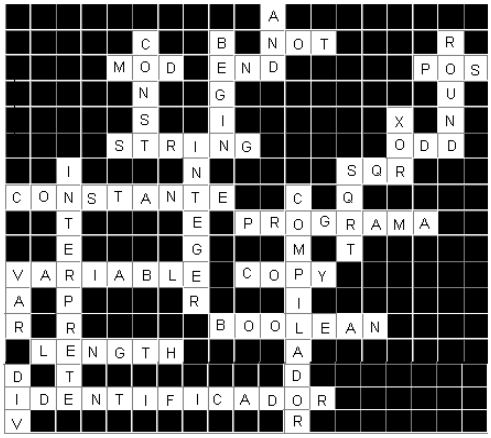
17) Secuencia de instrucciones que puede ejecutar una computadora. Sinónimo de software.
18) (Horizontal) Palabra reservada para declarar variables.
(Vertical) Identificador al que le asignamos un valor que puede ser modificado a lo largo de todo el programa.
19) Función que devuelve una subcadena de una cadena dada.
20) Tipo de dato para almacenar valores verdaderos o falsos.
21) Función que devuelve el tamaño de una cadena.
22) Operador aritmético utilizado para dividir número enteros y devuelve un entero.
23) Se utiliza para referirnos a una variable, una constante, al nombre del programa, el nombre de una función o de un procedimiento, etc.

1.25 Escriba en una expresión pascal las siguientes expresiones:

a) Un número menor o igual a otro número.
b) Un texto cualquiera es igual al texto "*FIDEL*".
c) Número de carné de identidad distinto de otro.
d) Representar un número positivo.
e) Representar un número impar.
f) Representar un número par.
g) Representar un número negativo.

1.26 Escriba en una expresión pascal las siguientes expresiones:

a) Un número mayor que 10 y menor o igual que 50.
b) Número comprendido entre 100 y 200.
c) Un texto distinto de “NO” e igual a “SI”.
d) Un número N mayor que el sucesor de un número K o menor que el sucesor del propio número K.
e) Un número distinto de 0 y que sea menor o igual que 5.



Respuesta del crucigrama

Capítulo 2

Estructuras básicas de control

Objetivos

- Escribir programas simples de computación en Pascal.
- Utilizar instrucciones simples de entrada, salida y asignación.
- Codificar enunciados donde se utilice los diferentes tipos de toma de decisiones.
- Generar números aleatorios.
- Utilizar las estructuras iterativas For, While y Repeat.
- Rastrear programas.

Pensamientos

◘ Dominad vuestros apetitos, queridos, y habréis conquistado la naturaleza humana.

Charles Dickens.

◘ La meta en si no es lo importante. La forma como se hace el viaje es lo que le da sentido a la vida.

María Esther Erosa.

◘ El mejor profeta del futuro es el pasado.

Lord Byron.

◘ Muchísimas personas sobrestiman lo que no son y subestiman lo que son.

Malcom Forbes.

◘ La sabiduría es esa cualidad que nos refrena de meternos en situaciones donde es necesario actuar con sabiduría.

Doug Larson.

Sinopsis

2.1 Estructura de control lineal.
2.2 Instrucciones de escritura, lectura y asignación.
2.3 Solución de problemas lineales.
2.4 Estructura de control condicional. Simple, doble y múltiples.
2.5 Solución de problemas condicionales.
2.6 Generación de números aleatorios.
2.6 Estructura de control iterativa. For, While y Repeat.
2.7 Solución de problemas iterativos.
2.8 Rastreo de programas.
2.9 Resumen.
2.10 Ejercicios de autoevaluación.
2.11 Respuestas de ejercicios de autoevaluación.
2.12 Ejercicios propuestos.

2.1 Estructura de control lineal.

En el capítulo anterior se dieron a conocer los elementos esenciales en Pascal como son: los identificadores, los tipos de datos, la forma de escribir las expresiones, la estructura de un programa general, entre otros aspectos de vital importancia en el mundo de la programación. Pero también conocemos de estudios anteriores que existen diferentes estructuras de control (Lineal, Condicional y Cíclica).

En este sección estudiarás en que consiste la estructura de control lineal y en la siguiente sección a cómo utilizarlas.

La estructura de control lineal está compuesta solamente por:

- Instrucciones de escritura.
- Instrucciones de asignación.
- Instrucciones de lectura.

Para la creación de un programa es necesario conocer cómo se introducen los datos, cómo se operan estos y cómo se imprimen los resultados. Es de gran importancia el conocimiento de cómo trabaja esta estructura ya que las demás estructuras también trabajan con esta.

2.2 Instrucciones de escritura, lectura y asignación.

En esta sección se abordarán las de instrucciones de escritura, lectura y asignación. Confeccionaremos programas sencillos en el lenguaje de programación BP7 y se mostrarán algunas aplicaciones de ejemplos que les servirán para profundizar en cuanto a la estética del programa a elaborar.

Instrucciones de escrituras (WRITE y WRITELN):

La instrucción WRITE realiza una impresión de datos en la pantalla. Estos datos (que son datos de salida) pueden ser cadenas de caracteres, números o variables. Las cadenas de caracteres deben ir encerradas entre apóstrofo (‘’).

Cuando se desea imprimir datos que son más de uno deben ir separados por comas.

Ejemplo del uso de WRITE:

WRITE(‘Hola Mundo.’);

- Se imprime en pantalla:

Hola Mundo.

WRITE(‘Mi nota en la prueba de LTP fue: ’ , NOTA);

- Se imprime en pantalla suponiendo que NOTA sea igual a 100:

Mi nota en la prueba de LTP fue: 100

La instrucción WRITELN es muy parecida a WRITE, la diferencia es que escribe un indicador de fin de línea después del último dato. Esto hace que la próxima instrucción que ejecute lo haga en la línea siguiente ya sea de escritura o de lectura. La filosofía de trabajo de esta es igual a la anterior y con la práctica vas a entender cuando es más factible utilizar una cuando otra.

En ocasiones deseamos mostrar un número real de la manera que solo aparezcan dos lugares después de la coma. Para esto analice el siguiente ejemplo, donde PROM es una variable real que almacena el valor 4.89367929.

Ejemplo:

Write('El promedio de las notas del grupo es de: ', PROM:5:2);

Aquí se muestra el texto, después de la coma la variable. El **:5** representa el ancho de separación entre el texto y la variable. El **:2** representa la cantidad de lugares decimales después del punto. Esto solo es posible si la variable es real.

Primer programa

Nuestro primer programa es conocido Hola Mundo. Para ganar en organización siempre que vallas a realizar un programa cree una carpeta con el nombre del programa que vas a elaborar con el objetivo de guardar en esta nuestro programa.

Este programa consiste en mostrar en pantalla el texto Hola Mundo. Figura 2.1.

PROGRAM Hola_Mundo; **BEGIN** Write('Hola Mundo.'); Readln; **END**.	Algoritmo Hola Mundo Inicio Mostrar "Hola Mundo." Fin

Figura 2.1 Programa y Algoritmo Hola Mundo.

En la figura puedes apreciar cómo queda elaborado nuestro primer programa, recuerde que puedes ver el resultado de lo escrito compilando en programa presionando Ctrl. + F9. En el programa aparece la instrucción Readln que se utiliza para que realice una espera, simula una entrada de datos.

En figura 2.2 se muestran dos ejemplos de programas para que puedas observar la diferencia que existe entre "Write" y "Writeln".

PROGRAM Uso_Write; **BEGIN** Write('Linea 1.'); Write('Linea 2.'); Readln; **END**.	**PROGRAM** Uso_Writeln; **BegIn** Writeln('Linea 1.'); Writeln('Linea 2.'); Readln; **END**.

Figura 2.2 Diferencia entre Write y Writeln.

Instrucciones de lectura (READ y READLN):

La instrucción read lee un dato de entrada y luego se lo asigna a una variable.

Ejemplo:

read (variable);

Si es más de un variable de entrada, estas van separadas por comas. Las variables de tipo booleanas no pueden incluirse en la lista de variables de entrada.

Ejemplo de una sentencia read:

read(a, b,c);

Esta sentencia hace que se lean 3 datos de entrada y sean asignados a las variables **a, b, c** respectivamente.

La instrucción Readln es parecida a la read, se usa para leer los datos de entrada y luego asignarlos a variables. La diferencia está en que la instrucción Readln comienza leyendo los datos en una línea después.
Con el tiempo verás cuando utilizas Read y cuando Readln;
En la Figura 2.3 aparece un ejemplo de programa y su algoritmo que solicita al usuario su nombre y luego le da la bienvenida a Pascal.

PROGRAM Bienvenido; **Var** Nombre: **String**; **BEGIN** Write('Introduzca su nombre: '); Readln(Nombre); Writeln('Bienvenido a Pascal ',Nombre); Readln; **END**.	Algoritmo Bienvenido Var Nombre: Texto; Inicio Mostrar "Introduzca su nombre: " Leer Nombre Mostrar "Bienvenido a Pascal ",Nombre Fin

Figura 2.3 Programa y algoritmo Bienvenido.

En el programa siguiente (Figura 2.4), le pide al usuario que entre dos nombre y muestra un mensaje donde concatena cadenas.

PROGRAM Concatenar; **Var** Nom1, Nom2: **String**; **Const** Mens = ' estudia con '; **BEGIN** Write('Entre un nombre: '); Readln(Nom1); Write('Entre otro nombre: '); Readln(Nom2); Writeln(Nom,Mens,Nom2); Readln; **END**.	Algoritmo Concatenar Var Nom1, Nom2: Texto; Const Mens =" estudia con "; Inicio Mostrar "Entre un nombre: " Leer Nom1 Mostrar "Entre otro nombre: " Leer Nom2 Mostrar Nom1, Mens, Nom2 Fin

Figura 2.4 Programa y algoritmo Concatenar.

2.3 Solución de problemas lineales.

En este epígrafe se mostrarán problemas resueltos mediante algoritmos y su paso al BP7. También se mostrarán en algunas soluciones aspectos nuevos que te servirán para hacer un programa más eficiente.

1) Confeccione un programa que pida al usuario dos números y muestre su suma. Solución 1:

PROGRAM Suma; Var N1,N2,S: Integer; **BEGIN** Write('Entre dos números: '); Readln(N1,N2); S:= N1 + N2; Writeln('La suma es: ', S); Readln; **END**.	Algoritmo Suma Var N1, N2, S: Entero; Inicio Mostrar "Entre dos números: " Leer N1, N2 S := N1 + N2 Mostrar "La suma es: ", S Fin

Otra forma de resolver este problema:

PROGRAM Suma; Var N1,N2: Integer; **BEGIN** Write('Entre dos números: '); Readln(N1,N2); Writeln('La suma es: ', N1 + N2); Readln; **END**.	Algoritmo Suma Var N1, N2: Entero; Inicio Mostrar "Entre dos números: " Leer N1, N2 Mostrar "La suma es: ", N1 + N2 Fin

Explicación:

En la primera solución se almacena la suma de los números N1 y N2 en la variable S para posteriormente mostrarla.

En la segunda solución la suma de N1 y N2 se realiza en el momento en que se muestra, esto reduce el código y no se necesita declarar ninguna variable para almacenar esta suma.

Nota: Recuerde colocar un ponto y coma al final de cada línea.

2) Confeccione un programa que pida al usuario cuatro números y este muestre la suma del primero con el último, el cuadrado del segundo y la raíz cuadrada del tercero.

Nota: El algoritmo de este se los dejo a ustedes.

Solución

```
PROGRAM Numero;
Uses CRT;
Var A,B,C,D : Integer;
BEGIN
  CLRSCR;          { Limpia la pantalla }
  Write('Entre cuatro números: ');
  Readln(A, B, C, D);  { Almacena cuatro números }
  Writeln('Suma del primero y el último: ', A + D);{Muestro la suma de A y D}
  Writeln('Cuadrado de segundo: ', Sqr(B)); {Muestro el cuadrado de B }
  Writeln('Raíz cuadrada del tercero: ', Sqrt(C)); {Raíz cuadrada de C}
  Readln; { Simula la entrada de datos }
END.
```

En este programa aparecen aspectos que nunca han visto. Primero, la línea **Uses CRT;** declara y habilita para su uso a la unidad **CRT**. Las unidades son módulos independientes del programa que, aunque no son ejecutables por si mismos, pueden ser utilizados por el programa principal sin necesidad de reescribir el código que contienen. Para la utilización de estos "subprogramas" es necesaria su declaración.

La palabra reservada USES cumple el propósito de declarar las unidades.

La instrucción **CLRSCR** se utiliza para borrar la pantalla completa y sitúa el cursor en la esquina superior izquierda, siempre que se valla a utilizar esta instrucción debe ser declarada primeramente la unidad CRT. De ahora en adelante siempre la usaremos para ganar en estética.

Las Llave **"{"** y **"}"** le indica al compilador que está en presencia de un comentario. Todo el texto que se escriba entre llaves no será interpretado por el compilador, esto se utiliza para ganar en claridad en cuanto al significado de cada línea de código.

3) Escribir un programa en Pascal que calcule el área de un rectángulo.

Solución:

Para calcular el área de un rectángulo tenemos que conocer su fórmula:

Área del rectángulo = lado1 * lado2

Algoritmo Área del rectángulo
Var Lado1, Lado2, Area : Enteros;
Inicio
Leer Lado1,Lado2
Area = Lado1 * Lado2
Mostrar Area
Fin

```
PROGRAM Area_Rectangulo;
        USES CRT;
        VAR lado1,lado2,area: INTEGER;
BEGIN
 ClrScr; { Limpiamos la pantalla }
{Este programa nos va a servir para calcular el área de un rectángulo}
{Entran los valores necesarios}
 Writeln('Para calcular el Área de un Rectángulo.');
 Write('Entre el valor del lado1: '); Readln(lado1);
 Write('Entre el valor del lado2: '); Readln(lado2);
 area:=lado1* lado2; {Calculamos el área}
 Writeln('El área de este rectángulo es: ', area); {Lo mostramos en pantalla}
 Readln;
END.
```

4) Escribir un programa que calcule la longitud y el área de una circunferencia.

Solución:

Para calcular la longitud de la circunferencia tenemos que:

longitud de la circunferencia = 2 * PI * radio

Para calcular el área de la circunferencia tenemos que:

área de la circunferencia = PI * radio^2

Sabemos que PI es una constante de valor 3.14 por lo tanto solo necesitamos el valor del radio para resolver este problema.

Algoritmo Área y Longitud de una Circunferencia
Var radio, longitud, area: Real;
Const PI = 3.14;
Inicio
Leer radio;
Area = 2* PI * radio
Longitud = PI * radio * radio
Mostrar Area, longitud
Fin

```
PROGRAM Area_y_Long_Circunf;
  USES CRT;
 VAR radio,longitud,area:REAL;
  Const PI = 3.14;
BEGIN
 ClrScr; { Limpiamos la pantalla }
{Este programa nos calcula el área y la longitud de una circunferencia}
{Entra el valor necesario}
 Writeln('Para calcular el área y la longitud de una circunferencia.');
 Write('Entre el valor del radio: '); Readln(radio);
 longitud:=2*PI*radio;
 area:=PI*SQR(radio);
{Se muestran los resultados}
 Writeln ('LONGITUD DE LA CIRCUNFERENCIA: ', longitud:5:2);
 Writeln ('ÁREA DE LA CIRCUNFERENCIA: ', area:5:2);
 Readln;
END.
```

5) Escribir un programa en Pascal que calcule el número de horas, minutos y segundos que hay en N segundos.

Solución:

- 1 hora tiene 3600 segundos, por lo tanto, si dividimos la cantidad de segundos entre 3600 tendremos la cantidad de horas.
- 1 minuto tiene 60 segundo, por lo tanto, si dividimos la cantidad de segundos entre 60 tendremos la cantidad de minutos.

Nota: El algoritmo se los dejo para que lo realicen.

```
PROGRAM Hora_Minutos;
      USES CRT;
Var horas, minutos, segundos, n: INTEGER;
BEGIN
ClrScr; { Limpiamos la pantalla }
{Calcula el número de horas, minutos y segundos que hay en N segundos.}
 Writeln('Para calcular el número de horas, minutos y segundos.');
 Write('Entre la cantidad de segundos: '); Readln(n);
 horas:= n div 3600;
 minutos:= (n mod 3600) div 60;
 segundos:= (n mod 3600) – minutos * 60;
 WRITELN ('En ',n,' segundos hay: ');
 WRITE (horas,' hora con ',minutos,' minutos y ',segundos,' segundos.');
 Readln;
END.
```

2.4 Estructura de control condicional. Simple, doble y múltiples.

En diversas ocasiones tenemos que comparar datos para poder controlar nuestro programa, esto se resuelve mediante las estructuras de control condicional. En este tema vamos a ver cómo podemos evaluar condiciones desde Pascal.

Condicional Simple:

Existen tres tipos (Simples, Dobles y Múltiples). La primera construcción que trataremos es la estructura selectiva simple (Figura 2. 5), que expresa bastante bien lo que podemos hacer con ella.

Pascal	Pseudocódigo

If Condición Then
  Acción verdadera;
	Si Condición Entonces
	  Acción verdadera
	Fin_Si

Figura 2.5 Estructura de condicional simple.

La orden que está después del Then se ejecutará si y solo si la condición planteada entre el If y el Then es “Verdadera”, o sea, si se cumple lo que se pregunta. Después de la palabra Then no se pone punto y coma, ya que la expresión continúa y este se coloca al final de la expresión. En caso de que se deseen poner más de una orden después del Then, se deben escribir entre las palabras reservadas begin y end**;**(Figura 2.6). Esto se debe a que las "sentencias u órdenes" pueden ser simples o una compuestas. Las sentencias compuestas se forman agrupando varias simples entre begin y end.

Pascal	Pseudocódigo

If Condición Then
begin
  Acción1;
  Acción2;
  AcciónN;
end;
	Si Condición Entonces
	  Acción1
	  Acción2
	  AcciónN
	Fin_Si

Figura 2.6 Estructura de condicional simple compuesta por varias órdenes verdaderas.

Ejemplo:

Observe el programa de la figura 2.7. Aquí se lee un número y a continuación se pregunta si es mayor que cero. En caso que se entre un 8 el programa imprime en la pantalla:

```
El 8 es positivo.
Su Cuadrado es: 64
...Programa Terminado...
```

Pero en caso que se entre un 0 el programa imprime en pantalla:

```
...Programa Terminado...
```

Ahora pruebe usted con un número negativo y responda qué resultado imprime en pantalla y por qué.

```
Program Positivo;
Var  num: Integer;
Begin
readln(num);
if num>0 then
 begin
  writeln('El ',num,' es positivo.');
  writeln('Su Cuadrado es: ',sqr(num));
 end;
  writeln('...Programa Terminado...')
 Readln;
End.
```

Figura 2.7 Ejemplo del uso de la estructura condicional simple.

En esta tipo de estructura si la condición planteada no es verdadera no se ejecuta la o las acciones verdaderas y el programa continúa con las demás acciones.

Condicional Doble:

En la construcción de una estructura condicional doble (Figura 2.8), podemos ver que si se cumple la condición se ejecutan un grupo acciones y si no las otro grupo. La estructura selectiva simple es muy fácil de usar y muy común pero en ocasiones tenemos necesidad de decirle al programa para si no se cumple una condición que nos realice una determinada tarea y si se cumple nos haga otra. Con la simple es posible hacerlo, pero se vuelve un programa muy largo, más complejo y a la vez mucho menos eficiente.

Pascal	Pseudocódigo

If Condición Then
begin
  Acción_Verd 1;
  Acción_Verd 2;
  Acción_Verd N;
end
Else
begin
  Acción_Falsa 1;
  Acción_Falsa 2;
  Acción_Falsa N;
end;
	Si Condición Entonces
	  Acción_Verd 1
	  Acción_Verd 2
	  Acción_Verd N
	Sino
	  Acción_Falsa 1
	  Acción_Falsa 2
	  Acción_Falsa N
	Fin_Si

Figura 2.8 Estructura condicional doble.

Ejemplo:

Observe el programa de la figura 2.9. Aquí se lee un número y a continuación se pregunta si es mayor que cero. En caso que se entre un 5 la expresión se evalúa como TRUE y el programa imprime en la pantalla:

```
El 5 es positivo.
Su Cuadrado es: 25
...Programa Terminado...
```

En caso que se entre un -9 la expresión se evalúa como FALSE y el programa imprime en pantalla:

```
El -9 es negativo.
Su Inverso es: 9
...Programa Terminado...
```

Ahora pruebe usted con el número cero y responda qué resultado imprime en pantalla y diga si este programa es completamente correcto.

```
Program Positivo_Negativo;
Var  num: Integer;
Begin
readln(num);
if num>0 then
 begin
  writeln('El ',num,' es positivo');
  writeln('Su Cuadrado es: ',sqr(num));
 end
else
 begin
  writeln('El ',num,' es negativo');
  writeln('Su Inverso es: ',num*(-1));
 end;
  writeln('...Programa Terminado...')
 Readln;
End.
```

Figura 2.9 Ejemplo del uso de la estructura condicional doble.

Condicional Múltiple:

En determinados momentos una sola condición no permite al programador satisfacer sus necesidades. Para estos casos, en BP7 al igual que mochos LP, es posible anidar las estructuras condicionales, colocando estructuras (If ... Then ... else) dentro de las estructuras (If ...then .... else). Por ejemplo el programa de la figura 2.10 imprime "Excelente" para cuando la nota de un estudiante dado es 5, "Muy Bien" para cuando sea 4, "Bien" para cuando sea 3, "Suspenso" para cuando sea 2 y "Fuera de Rango" para cualquier valor que sea distinto a esto valores mencionados.

Fíjense que en el programa no se coloca el punto y coma hasta que se termina la expresión.

Estructura (Case .. of):

Hay otro tipo de estructura múltiple llamada Case .. of. En la figura 2.11 aparece un ejemplo que resuelve el mismo problema de la figura 2.10 lo que se hace utilizando esta nueva estructura.

Aquí se puede observar un programa más simple que el anterior, pero tenga en cuenta que esta estructura se debe utilizar solo en caso necesarios ya que trabaja solo con valores enteros o lógicos.

Si se desea dar más de una orden para un valor verdadero, debe ir encerrado entre begin y end, como aparece en la figura 2.12.

Pascal	Pseudocódigo

Program Calificación_1;
Uses CRT;
Var Nota: Integer;
Begin
  Readln(Nota);
  If Nota = 5 Then
  Writeln(‘Excelente.’)
  Else if Nota = 4 Then
  Writeln(‘Muy Bien.’)
  Else if Nota = 3 Then
  Writeln(‘Bien’)
  Else if Nota = 2 Then
  Writeln(‘Suspenso.’)
  Else
  Writeln(‘Fuera de Rango.’);
Readln;
End.
	Algoritmo Calificación 1
	Var Nota: Entero;
	Inicio
	  Leer Nota
	  Si Nota = 5 Entonces
	  Mostrar “Excelente.”
	  Sino Nota = 4 Entonces
	  Mostrar “Muy Bien.”
	  Sino Nota = 3 Entonces
	  Mostrar “Bien.”
	  Sino Nota = 2 Entonces
	  Mostrar “Suspenso.”
	  Sino
	  Mostrar “Fuera de Rango.”
	  Fin_Si
	Fin

Figura 2.10 Ejemplo del uso de la estructura condicional múltiple utilizando IF ...THEN ...ELSE.

Pascal	Pseudocódigo

Program Calificación_2;
Uses CRT;
Var Nota: Integer;
Begin
  Readln(Nota);
  Case Nota of
  5: Writeln(‘Excelente.’);
  4: Writeln(‘Muy Bien.’);
  3: Writeln(‘Bien’);
  2: Writeln(‘Suspenso.’);
  Else
  Writeln(‘Fuera de Rango.’);
  end;
Readln;
End.
	Algoritmo Calificación 2
	Var Nota: Entero;
	Inicio
	  Leer Nota
	  Caso Nota
	  5: Mostrar “Excelente.”
	  4: Mostrar “Muy Bien.”
	  3: Mostrar “Bien.”
	  2: Mostrar “Suspenso.”
	  Sino
	  Mostrar “Fuera de Rango.”
	  Fin_Caso
	Fin

Figura 2.11 Ejemplo del uso de la estructura condicional múltiple utilizando Case .. of.

```
Caso A of
3:
begin
 Acción1;
 Acción2;
 AcciónN;
end;
```

Figura 2.12 Estructura Case que ejecuta varias órdenes si la condición se evalúa como verdadera, en este caso cuando A sea igual 3.

2.5 Solución de problemas condicionales.

1.- Escribir un programa que determine si un número introducido es par o es impar.
Solución:
Conociendo que todo número par es múltiplo de 2, llegamos a la conclusión que PAR = Num mod 2 = 0. Si esto es verdadero entonces Num es Par y si no es Impar.

Algoritmo Par_Impar1 Var Num: Entero; Inicio Leer Num Si Num mod 2 = 0 Then Mostrar “Número Par.” Sino Mostrar “Número Impar.” Fin_Si Fin	Algoritmo Par_Impar2 Var Num: Entero; Par: Boolean; Inicio Leer Num Par:= Num mod 2 = 0 Si Par = True Then Mostrar “Número Par.” Sino Mostrar “Número Impar.” Fin_Si Fin

Program Par_Impar1;
Uses CRT;
Var Num: Integer;
Begin
ClrScr; { Limpiamos la pantalla }
{Clasifica un número en Par o Impar}
Write(‘Para clasificar un número en ‘);
Writeln(‘Par o Impar.’);
Write(‘Entre un número: ’);
Readln(Num);
If Num mod 2 = 0 Then
Writeln(‘Número PAR.’)
Else
Writeln(‘Número IMPAR.’);
Readln;
End.
	Program Par_Impar2;
	Uses CRT;
	Var
	Num: Integer;
	Par: Boolean;
	Begin
	ClrScr; { Limpiamos la pantalla }
	{Clasifica un número en Par o Impar}
	Write(‘Para clasificar un número en ‘);
	Writeln(‘Par o Impar.’);
	Write(‘Entre un número: ’);
	Readln(Num);
	Par:= Num mod 2 = 0;
	If Par = True Then
	Writeln(‘Número PAR.’)
	Else
	Writeln(‘Número IMPAR.’);
	Readln;
	End.

2.- Elabore el programa que resuelva el siguiente problema.
Se conoce el peso en libras de un atleta de boxeo, se sabe que oscila entre 124 lb. y 140 lb. se desea precisar si el atleta está en la división de los 57, 60 ó 63,5 Kg.

Nota: División - 57 Kg. Desde 124 hasta 129 lb.
División - 60 Kg. Hasta 137 lb.
División – 63,5Kg Hasta 145 lb.

```
Program Boxeadores1;
Uses CRT;
Var Peso: Real;
Begin
ClrScr; { Limpiamos la pantalla }
{División de un boxeador}
 Write('Para clasificar al boxeador ');
 Write('Entre su peso en libras: ');
 Readln(Peso);
 If (Peso >= 124) and (Peso <= 129) Then
  Begin
   TextColor(1);
   Writeln('DIVISIÓN 57 KG.');
 end
 Else If Peso <= 137 Then
  Begin
   TextColor(2);
   Writeln('DIVISIÓN 60 KG.');
  end
 Else If Peso <= 140 Then
  Begin
   TextColor(3);
   Writeln('DIVISIÓN 63.5 KG.');
  end
 Else
  Begin
   TextColor(60);
   Writeln('NO PUEDE BOXEAR');
  end;
Readln;
End.
```

En el programa se encontraron con la instrucción TextColor, esta se usa para seleccionar el color del texto en la pantalla. En la figura 2.13 aparece las constantes definidas en BP7 para los colores. Para hacer uso de esta instrucción se debe llamar primer a la unidad CRT.

Constante	Valor	Color	Constante	Valor	Color
Black	0	Negro	Dark gray	8	Gris oscuro
Blue	1	Azul	Light Blue	9	Azul claro
Green	2	Verde	Light Green	10	Verde claro
Cyan	3	Cyan	Light Cyan	11	Cyan claro
Red	4	Rojo	Light Red	12	Rojo claro
Magenta	5	Magenta	Light Magenta	13	Magenta claro
Brown	6	Marr□n	Yellow	14	Amarillo
Light Gray	7	Gris claro	White	15	Blanco

Figura 2.13 Constantes de colores definidas en BP7.

Nota: Pruebe con valores numéricos mayores de 16 para que vea su función.
Nota: Mediante la instrucción TextBackGround selecciona el color del fondo de la pantalla. Los valores que puede utilizar son del 0 al 7, también es posible utilizar las constantes predefinidas para los colores.

3.- Clasifique la nota final de matemática de un estudiante en cuanto a:

0..59 -------- Suspenso
60..75------- Regular
76..89------- Bien
90..95------- Muy bien
96..100----- Excelente
Cualquier otra nota será una nota no válida.

```
program Clasificar_Nota;
Uses
  CRT;
 Var
  N: Integer;
Begin
ClrScr;
{Clasifica la nota de un estudiante}
 Writeln(‘Para clasificar su nota en matemática‘);
 Write('Entre su nota: '); Readln(N);
 Case N of
  0..59:
          begin
           Textcolor(Red);
            Writeln('Suspenso.');
          end;
  60..75:Writeln('Regular.');
  76..89:Writeln('Bien.');
  90..95:Writeln('Muy bien.');
  96..100:Writeln('Excelente.');
 else
     TextColor(20);
      Writeln(‘Nota no Válida.');
 end;
 Readln;
end.
```

4.- Conociendo la longitud de tres segmentos de rectas, determinar si con ellos se puede formar un triángulo.

Solución:
Para dar respuesta a este problema se debe conocer la desigualdad triangular que plantea (entre otras palabras) que para poder formar un triángulo, la suma de dos lados tiene que ser mayor que el tercero.

Program Formar_Triang;
Uses
CRT;
Var
L1, L2, L3: Integer;
Const
Se_Puede = ‘Se puede formar un Triángulo.’;
No_Puede =‘No se puede formar un Triángulo.’;
Begin
ClrScr;
{Para saber si se puede formar un triángulo o no teniendo tres segmentos}
Write(‘Para saber si se puede ‘);
Writeln(‘formar un triángulo.‘);
Writeln;
Write('Entre el valor del Lado1: ');
Readln(L1);
Write('Entre el valor del Lado2: ');
Readln(L2);
Write('Entre el valor del Lado3: ');
Readln(L3);
If ((L1+L2)>L3) and ((L2+L3)>L1) and ((L1+L3)>L2) Then
Writeln(Se_Puede)
Else
Writeln(No_Puede);
Readln;
End.
	Algoritmo Formar un triángulo
	Var L1, L2, L3 : Entero;
	Inicio
	Leer L1, L2 ,L3
	Si ((L1 + L2)> L3) y ((L2 + L3)> L1) y ((L1 + L3)> L2) Entonces
	Mostrar “Se puede formar.”
	Sino
	Mostrar “No se puede formar.”
	Fin_si
	Fin

5.- Escriba un programa que pregunte al usuario en qué sentido se desea realizar la conversión de Radianes a Grados o de Grados a Radianes, y luego pida el valor necesario.

Para convertir de Grados a Radianes:

$$VR=(\Pi*VG)/180$$

Para convertir de Radianes a Grados:

$$VG=(180*VR)/\Pi$$

```
Program Convertir;
Uses
  CRT;
 Var
  VR, VG: Real;
  C: Integer;
 Const
  PI = 3.14;
Begin
ClrScr;
 Write('Programa para convertir de Radianes a Grados o de Grados a Radianes.');
 Writeln('Entre 0, para convertir de Radianes a Grados.');
 Writeln('Entre 1, para convertir de Grados a Radianes.');
 Write('Entre 1 ó 2, según la conversión: ');Readln(C); Writeln; {Espacio}
   If C = 1 Then
    begin
       Write('Entre el valor del Grado: ');Readln(VG);
       Writeln('Valor en Radianes: ', PI * VG / 180:5:2);
    end
   Else If C = 0 Then
    begin
      Write('Entre los Radianes: ');Readln(VR);
       Writeln('Valor en Grados: ', 180 * VR / PI:5:2);
    end
   Else
     Writeln('Conversión no válida.');
 Readln;
End.
```

2.6 Generación de números aleatorios.

Existe en la vida cotidiana un elemento llamado "suerte", que a todo tipo de persona le gustaría tener. Este elemento se ve mucho en juegos, donde se tenga que hacer uso de dados, lanzar una moneda, escoger un carta al azar, entre muchas otras, etcétera.

En un programa Pascal este elemento puede ser introducido mediante una función llamada RANDOM. Considere el siguiente enunciado:

A:= Random(MAX);

La función Random genera un número entero entre 0 y MAX –1. El valor de MAX debe ser un entero no mayor a 32767, que es el valor máximo de un entero.

El rango de valores producidos directamente por Random, a menudo son distintos de lo que requiere para una aplicación específica. Por ejemplo, un programa que simule lanzar una moneda al aire pudiera requerir solo de 0 para "caras" y 1 para "escudo". Un programa de juegos de dados, que simule la tirada de un dado de seis caras, requeriría de enteros aleatorios del rango de 1 a 6.

El programa de la figura 2.14, simula la tirada de un dado dos veces.

```
Program Aleatorio;
Uses CRT;
Var Dado: Integer;
Begin
 ClrScr;
 Write('Presione Enter para lanzar un dado.');Readln;
 Dado:= Random(6) + 1;
 Writeln('Primer tiro: '.Dado);
 Write('Presione Enter para lanzar otro dado.');Readln;
 Dado:= Random(6) + 1;
 Writeln('Segundo tiro: ',Dado);
 Readln;
End.
```

Figura 2.14 Utilización de la función Random.

En el programa "Aleatorio", se generan dos números aleatorios de 1 a 6 en líneas diferentes mediante (Dado:= Random(6) + 1;). Si nos limitamos a Random(6) solamente, se generará un número aleatorio de 0 a 5, por lo que con Random(6)+1 garantizamos que el resultado general sea un número comprendido entre 1 y 6 que son los números reales de cada cara de un dado. Pero cuando ejecutamos este programa varias veces nos percatamos que siempre genera los mismos números, y realmente esto no es lo que se quiere. Este problema se resuelve mediante el procedimiento Randomize. En la figura 2.15 reparamos este problema generado en el programa anterior. Siempre que se utilice la función Random es necesario colocar antes el procedimiento Randomize.

```
Program Aleatorio;
Uses CRT;
Var Dado: Integer;
Begin
 ClrScr;
 Randomize;
 Write('Presione Enter para lanzar un dado.');Readln;
 Dado:= Random(6) + 1;
 Writeln('Primer tiro: '.Dado);
 Write('Presione Enter para lanzar otro dado.');Readln;
 Dado:= Random(6) + 1;
 Writeln('Segundo tiro: ',Dado);
 Readln;
End.
```

Figura 2.15 Utilización de la Randomize y la función Random.

2.7 Estructura de control iterativa. For, While y Repeat.

Supongamos que nos piden que realicemos un dibujo sencillo que se despliegue en la pantalla. Nuestro objetivo es mostrar un cuadrado de 5 filas por 5 columnas trazado con cualquier carácter. En la figura 2. 16 se muestra la solución de este programa utilizando los elementos de programación que ya conocemos.

```
Program Dibujo;
Uses CRT;
Begin
 ClrScr;
  Writeln('* * * * *');
  Writeln('* * * * *');
  Writeln('* * * * *');
  Writeln('* * * * *');
  Writeln('* * * * *');
 Readln;
End.
```

Figura 2.16 Dibujo de un cuadrado de 5 x 5.

Básicamente este programa cumple con el objetivo. Pero... ¿No perciben que algo anda mal? Aquí se repitió cinco veces la misma instrucción. ¿Qué sucedería si nos pidieran una figura más grande? Tendríamos que agregar más líneas de código. Y si en vez de una figura , se les ocurría pedirnos que listáramos una lista de números del uno al millón, el código sería inmenso. También imagínense lo que tardaríamos en escribirlo.
Sin embargo tenemos un recurso para acudir en cuanto surjan estos problemas. Son las estructuras de iteración. Una iteración consiste en una repetición de un bloque de sentencias un número determinando de veces o hasta que se cumpla una condición.

De esta forma el código puede simplificarse notablemente. Antes de entrar en los detalles de la implementación en BP7, veremos en la Figura 2.17 cómo podemos realizar el programa dibujo en pseudocódigo utilizando una estructura iterativa de tres formas:

Algoritmo Dibujo1	Algoritmo Dibujo2	Algoritmo Dibujo3
Var I:Entero;	Var I:Entero;	Var I:Entero;
Inicio	Inicio	Inicio
Para I:=1 a 5 hacer	I := 1	I := 0
Mostrar ("*****")	Mientras I<=5 hacer	Repetir
Fin_Para	Mostrar ("*****")	Mostrar ("*****")
Fin	I:= I + 1	I:= I + 1
	Fin_Mientras	Hasta I = 5
	Fin	Fin

Figura 2.17 Diferentes algoritmos para elaborar el programa Dibujo.

BP7 cuenta con tres tipos de **Bucles** (estructuras iterativas). Estos son:

- **For**.
- **While**.
- **Repeat**.

Nota:
Todo lenguaje de programación cuenta con estructuras de control repetitivas.

Elementos comunes de los ciclos.
Aunque existan tres tipos de bucles, estos tienen elementos que les son comunes.
Todo bucle tiene:

- Un punto de inicio.
- Un punto de terminación.
- Un cuerpo, que contiene la o las acciones que se quieren repetir.
- Una prueba de condición que determina cuando debe concluir el ciclo.

Nota:
1.- Si el cuerpo contiene varias líneas, deberá estar bloqueado con instrucciones **Begin** y **End** (con la excepción del ciclo **Repeat**).
2.- El punto de terminación del ciclo puede ser la palabra clave **end** (en el caso de los ciclos **For** y **While**), o bien, **Until** (en caso del **Repeat**).

For:
Este es el bucle más usado. Toma dos parámetros: los valores inicial y final. Puede contar en forma ascendente (**To**) y en forma descendente (**Downto**) Figura 2.18.

Ciclo **For** Ascendente	Ciclo **For** Descendente

For valor_inicial **to** valor_final **do**
begin
Acciones o acción a repetir;
end;
	For valor_inicial **Downto** valor_final **do**
	begin
	Acciones o acción a repetir;
	end;

Figura 2.18 Estructura general del bucle For.

Ejemplo del bucle "For":

Conteo ascendente	Conteo descendente

Program For_Ascendente;
Uses CRT;
Var I: Integer;
Begin
For I:=0 **to** 9 **do**
begin
Write(' ',I);
end;
Readln;
End.
	Program For_Descendente;
	Uses CRT;
	Var I: Integer;
	Begin
	For I:=9 **downto** 0 **do**
	begin
	Write(' ',I);
	end;
	Readln;
	End.

Explicación del conteo ascendente:
Aquí le asignamos a la variable "**I**" el valor 0 que va a ser el valor inicial del ciclo. La palabra **to** indica que "**I**" se va a aumenta de uno en uno hasta llegar al valor final que en este caso es 9. Lo que está entre "**begin**" y "**end**;", va a mostrar el valor de "I" en cada *iteración* o repetición del bucle.

Aquí no era necesario las palabras begin y end porque se utiliza una solo línea de código a repetir.
En la figura 2.21(A) aparece el programa Dibujo resuelto de una manera más eficiente utilizando el bucle FOR.

Explicación del conteo descendente:
Aquí le asignamos a la variable "**I**" el valor 9 que va a ser el valor inicial del bucle. La palabra **Downto** indica que "**I**" se va a decrementar de uno en uno hasta llegar al valor final que en este caso es 0.

While:
Este ciclo difiere del **For** en que contiene una condición de prueba que se verifica al inicio de cada iteración. En tanto la condición de prueba sea verdadera (**True**), el ciclo continúa operando. Figura 2.19.

```
While < condición > do
 begin
  Acciones o acción a repetir;
 end;
```

Figura 2.19 Estructura del bucle While.

Ejemplo del bucle While:

```
Program Bucle_While;
Uses CRT;
Var I: Integer;
Begin
 I:=0;
 While I < 10 do
  Begin
   Write(' ',I);
   I:= I + 1;
  end;
 Readln;
End.
```

Este programa hace lo mismo que el programa "For_Ascendente", lo que en este caso lo realizamos con un ciclo While. Fíjense, iniciamos la variable "I" con valor 0. Al comenzar el ciclo hay una condición que mientras esta sea verdadera, se van repetir las acciones que están entre las palabras "begin" y "end;". Referente a la línea **"I:= I + 1"**, lo que hace es aumentar la variable "I" de uno en uno en cada iteración del ciclo, ya que este no lo hace por si solo como el For.
En la figura 2.21(B) aparece el programa Dibujo resuelto de una manera más eficiente utilizando el bucle While.
Nota:
Si no haces este incremento en cada repetición del ciclo, este será infinito y tendrás un programa bloqueado y hay que tener cuidado con esto. Recuerden que los ciclos tienen que ser finitos y siempre hay que buscar la manera de que la condición del ciclo sea falsa para que termine de operar.

Bueno con lo aprendido hasta ahora, usted está en condiciones de elaborar un programa que realice la misma operación que el programa “For_Descendente” pero utilizando el ciclo While.

Repeat:
Este ciclo es casi idéntico al While. La diferencia es que en este, la expresión condicional se verifica al final del ciclo, esto hace que las acciones que están en el cuerpo del bucle se ejecutan al menos una vez. Figura 2.20.

```
Repeat
  Acción o acciones a repetir;
Until < condición >;
```

Figura 2.20 Estructura del bucle Repeat.

Ejemplo del ciclo "Repeat" :

```
program Ciclo_Repeat;
Uses CRT;
 Var I: Integer;
begin
 I:=0;
 Repeat
  Write(' ',I);
  I:=I+1;
 Until I=10;
  Readln;
end.
```

Este programa hace lo mismo que el programa “For_Ascendente”, lo que en este caso lo realizamos con un ciclo Repeat. Fíjense, iniciamos la variable “I” con valor 0, la condición del ciclo está al final, lo que quiere decir esto que cuando utilizamos este ciclo, el cuerpo de este se va a ejecutar al menos una vez. Este ciclo se lee así “Repite (Repeat) estas acciones hasta (Until) que la condición sea verdadera”.
En la figura 2.21(C) aparece el programa Dibujo resuelto de una manera más eficiente utilizando el bucle Repeat.

Bueno con lo aprendido hasta ahora del bucle Repeat, usted está en condiciones de elaborar un programa que realice la misma operación que el programa “For_Descendente” pero utilizando el bucle Repeat.

A	B	C

Program Dibujo1;
Uses CRT;
Var I: Integer;
Begin
For I:=1 To 5 do
Writeln ("*****");
Readln;
End.
	Program Dibujo2;
	Uses CRT;
	Var I: Integer;
	Begin
	I := 1;
	While I <=5 do
	begin
	Mostrar ("*****");
	I:= I + 1;
	end;
	Readln;
	End.
		Program Dibujo3;
		Uses CRT;
		Var I: Integer;
		Begin
		I := 0;
		Repeat
		Writeln ("*****");
		I:= I + 1;
		Until I = 5;
		Readln;
		End.

Figura 2.21 Programa Dibujo resuelto mediante los tres tipos de bucles.

Funciones Inc y Dec.

Hasta ahora hemos utilizado el incremento de variables en un valor constante, y me refiero a la línea **"I:= I + 1;",** o sea, que la variable "I" se iba aumentar de uno en uno en cada iteración de ciclo. Pascal, al igual que otros lenguajes de programación, cuenta con funciones que nos permite hacer esto de una manera más fácil. La función INC, incrementa una variable dada en un número específico. Ver la figura 2.22 (A).

La función DEC, Decrementa de una variable dada en un número específico. Ver la figura 2.22 (B).

A	B
Inc(I); ------ Igual que **I:= I + 1;** **Inc**(I,10); ------ Igual que **I:= I+10;**	**Dec**(I); ------ Igual que **I:= I-1;** **Dec**(I,10);------ Igual que **I:= I-10;**

Figura 2.22 Ejemplo del uso de las funciones INC y DEC.

♠ Prueba cada una de estas funciones en los programas anteriores para que veas que es más cómodo de trabajar utilizando estas funciones.

2.8 Solución de problemas iterativos.

Los problemas se pueden resolver de muchas formas, aquí se muestra un vía que puede resultar fácil de entender. En cada problema se utiliza un bucle para que usted elabore su propio programa utilizando cualquiera de los otros estudiados en este epígrafe.

1.- Escribir un programa que visualice en pantalla los números pares entre 1 y 25.

Solución:

Program Pares_M25;
Uses crt;
Var num:integer;
Begin
clrscr;
num:=2;
Repeat
Writeln (num);
num:= num + 2;
Until num= 26;
Readln;
End.
	Algoritmo Pares Menores que 25
	Var num: entero;
	Inicio
	num:=2
	Repetir
	Mostrar num
	num:= num + 2
	Hasta num = 26
	Fin

2.- Escribir un programa en Pascal que visualice en pantalla los números múltiplos de 5 comprendidos entre 1 y 100.
Solución:

Program Mult_25;
Uses crt;
Var num:integer;
Begin
Clrscr;
num:= 5;
While num <= 100 do
begin
writeln (num);
Inc(num,5);
end;
Readln;
end.
	Algoritmo Multíplos 25;
	Var num: Entero;
	Inicio
	num:= 5
	Mientras num <= 100 hacer
	Mostrar num
	num:= num + 5
	Fin_Mientras
	Fin

3.- Escriba un programa que calcule el aporte total de las MTT de un grupo de 10 estudiantes.
Solución:

Program Aporte_MTT;
Uses crt;
Var I: Integer;
Aprt, AT: Real;
Begin
Clrscr;
AT:= 0;
For I:= 1 to 10 do
begin
Write (‘Entre el aporte ’,I,’: ’);
Readln(Aprt);
AT:= AT + Aprt;
end;
Writeln(‘Aporte del grupo: ’,AT:0:2);
Readln;
end.
	Algoritmo Aporte MTT
	Var I: Entero;
	Aprt, AT: Real;
	Inicio
	AT:= 0
	Para I:= 1 a 10 hacer
	Mostrar “Entre el aporte “,I,”: ”
	Leer Aprt
	AT:= AT + Aprt
	Fin_Para
	Mostrar “Aporte del grupo: ”, AT
	Fin

4.- Confeccione un programa para leer las notas de un grupo de N estudiantes. Agréguele la posibilidad de promediar las notas del grupo.

Solución:

Aquí necesitamos conocer la cantidad de estudiantes y almacenar este valor en la variable N, luego pedimos las N notas mediante un bucle y las vamos sumando, al final dividimos la suma entre N para obtener el promedio.

Program Promedio;
Uses
CRT;
Var
Prom, Sum,Nota: Real;
I, N: Integer;
Begin
ClrScr;
Write('Programa para promediar las notas de un estudiante.');
Write('Entre la cantidad de notas: ');
Readln(N); Writeln; {Espacio}
Sum:=0; {El sumador comienza con 0}
For I:=1 to N do
begin
Write('Entre la nota ',I,': ');
Readln(Nota);
Sum:= Sum + Nota;{Almacena la suma}
end;
Prom:= Sum / N;
Writeln('El promedio es: ',Prom:0:2);
Readln;
End.
	Algoritmo Promedio
	Var
	Prom, Sum,Nota: Real;
	I, N: Entero;
	Inicio
	Mostrar "Entre la cantidad de notas:"
	Leer N
	Sum:=0
	Para I:=1 a N hacer
	Mostrar "Entre la nota ", I, ": "
	Leer Nota
	Sum:= Sum + Nota
	Fin_Para
	Prom:= Sum / N
	Mostrar "El promedio es: ", Prom
	Fin

5.- Escribir un programa que realice la pregunta ¿Desea continuar S/N? y que no deje de hacerla hasta que el usuario teclee N.

Solución:

Program Salida;
Uses crt;
Var Respuesta: String;
Begin
Clrscr;
Repeat
Writeln;
Writeln ('¿Desea continuar: s/n?');
Readln (Respuesta);
Until Respuesta = 'n';
End.
	Algoritmo Salida
	Var Respuesta: Texto;
	Inicio
	Repetir
	Mostrar "¿Desea continuar: s/n?"
	Leer Respuesta
	Hasta Respuesta ="n"
	Fin

6.- Escribir un programa que detecte si un número es primo o no.

Solución:

Un número es primo si sólo es divisible por sí mismo y por la unidad (1).

Ejemplo: 2,3,4,7,11,13,17 son números primos.

9 no es número primo, porque es divisible por 1, 9, 3.

El algoritmo para resolver este problema está en dividir sucesivamente el número estudiado por 2,3,4, etc., hasta el propio número.
Los números primos son enteros mayores que 1 sin divisores enteros positivos, exceptuando el 1 y ellos mismos. Todos los primos son impares, excepto el 2.

```
Program Primo;
 Uses crt;
 Var I, N:Integer;
  P: Boolean;
  Respuesta: String;
Begin
 Clrscr;
 Repeat
  P:=True;
  I:=2;
  Writeln;
  Writeln('Para determinar si un número es primo o no.');
  Write(Entre un número: ); Readln(N);
  While (I < N) and (P=True) do
    begin
     If N mod I = 0 Then P:=False;
     Inc(I);
    end;
   If (P=True) and (N<>1) Then
     Writeln(N, ' es un número primo.')
   else
     Writeln(N, ' no es un número primo.');
 Writeln ('¿Desea continuar: s/n?'); Readln (Respuesta);
Until Respuesta = 'n';
End.
```

2.9 Rastreo de programas.

El rastreo consiste en realizar un recorrido por el programa paso a paso e ir anotando la operación que realiza cada línea que se ejecuta. Esta operación se realiza para comprobar si su programa está correctamente elaborado, claro que si se siguen los pasos lógicos para resolver un problema mediante un lenguaje de programación esto se debe realizar en cuanto se desarrolle el algoritmo. En BP7 se puede hacer esto presionando la tecla F8 o en el menú RUN escoger la opción Step Over.
Esto se hace dando como datos valores reales, para ver si se tienen los resultados esperados, en caso de que no de cómo se esperaba entonces se busca en qué línea está el error lógico. En la figura 2.23 aparece el rastreo del programa Aporte_MTT.

AT	I	I:=1 a 10		Mostrar AT
		Leer Aprt	AT:= AT + Aprt	
0	1	0.50	0.50	-
-	2	10	10.50	-
-	3	20	30.50	-
-	4	5	35.50	-
-	5	2.40	37.90	-
-	6	3.05	40.95	-
-	7	10	50.95	-
-	8	0	50.95	-
-	9	20	70.95	-
-	10	30	100.95	-
-	11	-	-	100.95

Figura 2.23 Rastreo del programa Aporte_MTT.

Esta es una habilidad que debe dominar un programador de punta a cabo, ya que conociendo con claridad que sucederá cuando se ejecuta cada una de la órdenes siempre se obtendrán los resultados esperados. Ahora solo cada de parte de usted la ejercitación de esta operación.

Instrucciones Delay, Sound y NoSound:

A partir de las estructuras cíclicas conocidas y con las instrucciones Sound, NoSound y Delay, podemos crear sonidos para darle un ambiente a nuestro programa más agradable. Al igual que la instrucción TextColor, estas antes de ser utilizadas se tiene que llamar la unidad CRT.

Delay: Detiene la ejecución del programa durante un tiempo determinado. Este tiempo se da en milisegundos. El intervalo válido es desde 0 hasta 65535, la precisión del retardo depende de la precisión del reloj interno de la computadora. En la figura 2.24 hay un ejemplo donde el programa espera aproximadamente 5 segundos y luego ejecuta la próxima orden.

```
Program Espera;
Uses crt;
Begin
 Writeln('Retardo de aproximadamente 5 segundos');
 Delay(5000);
 Writeln('Fin del retardo.');
 Readln;
End.
```

Figura 2.24 Ejemplo de la instrucción Delay.

Sound: Genera un sonido en la bocina de la computadora a una frecuencia determinada por el valor pasado como parámetro. Para detener el sonido es necesario ejecutar la instrucción NoSound. La figura 2.25 presenta un programa donde se pone a disposición de ustedes ejemplos de sonidos combinando estas instrucciones con estructuras cíclicas.

```
Program Sonidos;
Uses CRT;
Var I,J: Integer;
Begin
 CLRSCR;
 Write('Presione Enter para escuchar un sonido.');Readln;
 for i:=1 to 1 do
  begin
   for j:=1 to 3 do
    begin
     sound(200);     Delay(200);
     sound(200);     Delay(200);
     NoSound;
    end;
  sound(300);delay(200);sound(250);delay(180);sound(200);delay(500);
  NoSound;
 end;
 Write('Presione Enter para escuchar un sonido ganador.');Readln;
for i:=1 to 2 do
 begin
  sound(600);  delay(150);
  sound(480);  delay(150);
  nosound;
 end;
 sound(600); delay(500);
 NoSound;
 Write('Presione Enter para escuchar un sonido perdedor.');Readln;
 for i:=1 to 2 do
 begin
  sound(500);  delay(150);
  sound(280);  delay(150);
  nosound;
  sound(150);  delay(500);
  nosound;
 end;
 Readln;
End.
```

Figura 2.25 Ejemplo de sonidos.

De ahora en adelante puede crear sus propios sonidos para sus programas.

2.10 Resumen.

- Las estructuras de controles son Simples, Condicionales y Cíclicas.
- La estructura de control lineal está compuesta solamente por:
  - Instrucciones de escritura.
  - Instrucciones de asignación.
  - Instrucciones de lectura.

- Las instrucciones de escrituras son:
  - **Write**: Realiza una impresión de datos en la pantalla y coloca el cursor al final de la línea.
  - **Writeln**: Realiza una impresión de datos en la pantalla y coloca el cursor en la línea siguiente.
- Antes de finalizar el programa se debe colocar la instrucción Readln para que el programa realice un espera, esto garantiza la visualización de los resultados.
- Las instrucciones de lecturas son:
  - **Read**: Lee un dato de entrada y luego se lo asigna a una variable.
  - **Readln**: Parecida a la read, la diferencia está en que la instrucción Readln comienza leyendo los datos una línea después.
- Recuerde poner punto y coma al final de cada orden.
- La palabra reservada USES tiene el propósito de declarar las unidades.
- La unidad CRT tiene un conjunto de instrucciones que facilitan el trabajo a la hora de programar y no tener que crear instrucciones que ya existen.
- La instrucción **CLRSCR** borra la pantalla completa y sitúa el cursor en la esquina superior izquierda.
- Siempre que utilices la instrucción CLRSCR debe declarar primeramente la unidad CRT.
- Los comentarios se escriben entre llaves **"{"** y **"}"**.
- Las estructuras condicionales se clasifican en tres tipos:
  - Simples.
  - Dobles.
  - Múltiples.
- Estructura de condicional simple.

Pascal	**Pseudocódigo**

If Condición Then
  Acción verdadera;
	Si Condición Entonces
	  Acción verdadera
	Fin_Si

- Estructura condicional doble.

Pascal	**Pseudocódigo**

If Condición Then
begin
  Acción_Verd 1;
  Acción_Verd 2;
  Acción_Verd N;
end
Else
begin
  Acción_Falsa 1;
  Acción_Falsa 2;
  Acción_Falsa N;
end;
	Si Condición Entonces
	  Acción_Verd 1
	  Acción_Verd 2
	  Acción_Verd N
	Sino
	  Acción_Falsa 1
	  Acción_Falsa 2
	  Acción_Falsa N
	Fin_Si

- La estructura Case..Of, maneja una serie de decisiones en las cuales una variable o expresión particular se prueba para cada uno de los valores que puede asumir, y se toman diferentes acciones.

- **TextColor**: Instrucción que se usa para seleccionar el color del texto en la pantalla.
- **TextBackGround**: Instrucción que permite seleccionar el color del fondo de la pantalla.
- **Delay**: Detiene la ejecución del programa durante un tiempo especificado en milisegundos.
- **Sound**: Genera un sonido en la bocina de la computadora a una frecuencia determinada.
- **NoSound**: Desactiva el sonido iniciado con la instrucción Sound.
- **Random**: Instrucción que genera un número aleatorio de un rango numérico dado como parámetro.
- **Randomize**: Instrucción que prepara a Random para que la selección de los número no sean siempre la misma secuencia cada ves que se inicie el programa.
- Un ciclo o bucle es una repetición de un bloque de sentencias un número determinando de veces o hasta que se cumpla una condición.
- Tipos de bucles de BP7:
  - **For**.
  - **While**.
  - **Repeat**.
- Elementos comunes de los bucles:
  - Un punto de inicio.
  - Un punto de terminación.
  - Un cuerpo, que contiene la o las acciones que se quieren repetir.
  - Una prueba de condición que determina cuando debe concluir el ciclo.
- Si el cuerpo del bucle contiene varias líneas, deberá estar bloqueado con instrucciones **Begin** y **End** (con la excepción del ciclo **Repeat**).
- El punto de terminación del bucle puede ser la palabra clave **end** (en el caso de los bucles **For** y **While**), o bien, **Until** (en caso del **Repeat**).
- Estructura del bucle For:

Ciclo **For** Ascendente	Ciclo **For** Descendente

For valor_inicial **to** valor_final **do**
begin
Acciones o acción a repetir;
end;
	For valor_inicial **Downto** valor_final **do**
	begin
	Acciones o acción a repetir;
	end;

- Estructura del bucle While:

```
While < condición > do
 begin
  Acciones o acción a repetir;
 end;
```

- Estructura del bucle Repeat:

```
Repeat
  Acción o acciones a repetir;
Until < condición >;
```

- **Rastreo**: Consiste en realizar un recorrido por el programa paso a paso para comprobar si está correctamente elaborado En BP7 se puede hacer esto presionando la tecla **F8** o en el menú **RUN** escoger la opción **Step Over**.

2.11 Ejercicios de autoevaluación.

1.1 Llene cada uno de los siguiente espacios vacíos:

a) Las estructuras de control se clasifican en estructuras _____ , _____ y _____.
b) La estructura de control lineal está compuesta solamente por instrucciones de _____ , _____ y _____.
c) La instrucción _____ realiza una impresión de datos en la pantalla y _____ realiza la misma acción pero coloca el cursor en la línea siguiente.
d) La instrucción _____ es parecida a la Read, se usa para leer los datos de entrada y luego asignarlos a variables.
e) La instrucción _____ se usa para seleccionar el color del texto en la pantalla y _____ para seleccionar el color del fondo de la pantalla.
f) _____ se utiliza para generar un número aleatorio.
g) Para utilizar instrucciones como CLRSCR y TextColor hay que declarar la unidad _____ .
h) Se llama _____ al recorrido que se hace por el programa paso a paso para ver la operación que realiza cada línea de código.
i) ___ se utiliza para comenzar un comentario y ___ para terminarlo.

1.2 Diga que imprimiría en pantalla las siguientes líneas de código:

a)
```
Write('** Estoy aprendiendo PASCAL **');
```
b)
```
Write('Mi nombre es: ');
Write('Rodiert.');
```
c)
```
Writeln('Mi nombre es: ');
Writeln('Reinier.');
```
d)
```
Writeln;
```

1.3 Escriba un enunciado para cumplir con cada uno de los siguientes incisos:

a) Declare un programa con el nombre Promedio.
b) Declare las variables X, Y, Z y P que sean de tipo entero.
c) Indíquele al usuario que escriba tres números.
d) Lea tres enteros y almacénelos en las variables X, Y y Z.
e) Calcule el promedio de las variables X, Y y Z y asígnaselo a la variable P.
f) Imprima en pantalla "El promedio es: " seguido de la variable P.
g) Realice una espera de cuatro segundos.

1.4 Corrija los errores (si es que existe alguno) de cada uno de los siguientes enunciados:

a) Write 'Hola Mundo';
b) Realn A);
c) If N = 10
   Writeln(número igual a 10)
d) Suma = N1 + N2;

e) For I = 1 do 20 to
f) While I<31 do I:= I+1;

1.5 Diga cuáles de las siguientes afirmaciones pertenecen a los elementos comunes de los ciclos:

a) ___ Un prueba lógica que se verifica siempre al inicio.
b) ___ Un cuerpo, que contiene la o las acciones que se quieren repetir.
c) ___ Una prueba de condición que determina cuando debe concluir el ciclo.
d) ___ Un cuerpo se repite infinitamente.
e) ___ Un punto de terminación.
f) ___ Un punto de inicio.
g) ___ No poseen punto de terminación.

1.6 Escriba un programa utilizando estructuras secuenciales que muestre en pantalla una figura de color rojo igual a la siguiente:

```
**********
**********
**********
**********
```

1.7 Escriba un programa utilizando el bucle For que muestre en pantalla una figura de color azul igual a la que sigue:

```
*
**
***
****
*****
******
```

1.8 Utilice las técnicas apropiadas para garantizar la salida para cada uno de los siguientes incisos. No puedes hacer ningún cambio a excepción de inserción de begin y end. Pudiera ser posible que no se requiera de ninguna modificación.

```
If (Y = 10) and (Z = 7) Then
  Writeln('@@@@@@')
Else
  Writeln('######');
  Writeln('$$$$$$');
  Writeln('&&&&&&');
```

a) Suponiendo que Z =7 y Y =10. Se produce la siguiente salida:

```
@@@@@@
$$$$$$
&&&&&&
```

b) Suponiendo que Z =7 y Y =10. Se produce la siguiente salida:

```
@@@@@@
```

c) Suponiendo que Z =7 y Y =10. Se produce la siguiente salida:

```
@@@@@@
&&&&&&
```

d) Suponiendo que Z =7 y Y =7. Se produce la siguiente salida:

```
######
$$$$$$
&&&&&&
```

1.9 ¿Qué es lo que escribe el siguiente programa?

```
Program Ejercicio;
Var X, Y, Total: Integer;
Begin
 X:= 1;
 Total:= 0;
 While X<= 10 do
 begin
  Y:= X * X;
  Total:= Total + Y;
  INC(X);
 end;
 Writeln('El total es ',Total);
 Writeln('El valor final de X es: ', X);
 Readln;
End.
```

2.12 Respuestas de ejercicios de autoevaluación.

1.1

a) Lineales o secuenciales, Condicionales y Cíclicas.
b) Escritura, Asignación y Lectura.
c) Write y Writeln.
d) Readln.
e) TextColor y TextBackGround.
f) Random.
g) CRT.
h) Rastreo.
i) "**{** " y "**}** "

1.2

a)
** Estoy aprendiendo PASCAL **

b)
Mi nombre es: Rodiert.

c)
Mi nombre es:
Reinier.

d) Aparece la línea en blanco.

1.3

a)

```
Program Promedio;

        Begin

End.
```

b) Var X, Y, Z: **Integer**;
c) **Write**('Entre tres números: ');
d) Readln(X, Y, Z);
e) P:= (X + Y + Z) DIV 3;
f) Writeln('El promedio es: ', P);

g) Delay(4000);

1.4

a) Write (‘Hola Mundo’);
b) Readln (A);
c) If N = 10 Then
   Writeln(‘número igual a 10’);
d) Suma := N1 + N2;
e) For I:= 1 to 20 do
f) No tiene errores.

1.5

a) **No**. b) **Si**. c)**Si**. d) **No**. e) **Si**. f) **Si**. g) **No**.

1.6

```
Program Figura1;
Uses CRT;
Begin
 TextColor(Red);
 Writeln('**********');
 Writeln('**********');
 Writeln('**********');
 Writeln('**********');
 Readln;
End.
```

1.7

```
Program Figura2;
Uses CRT;
Var I, J: Integer;
Begin
 TextColor(Blue);
 For I:=1 to 6 do
  begin
   For J:=1 to I do Write('*');
   Writeln;
  end;
Readln;
End.
```

1.8

a) No requiere modificación.
b)

```
If (Y = 10) and (Z = 7) Then
  Writeln('@@@@@@')
Else
 begin
  Writeln('######');
  Writeln('$$$$$$');
  Writeln('&&&&&&');
 end;
```

c)

```
If (Y = 10) and (Z = 7) Then
  Writeln('@@@@@@')
Else
 begin
  Writeln('######');
  Writeln('$$$$$$');
 end;
 Writeln('&&&&&&');
```

d) No requiere modificación.

1.9

```
El total es 385
El valor final de X es: 11
```

2.13 Ejercicios propuestos.

1.10 Realice un programa donde pida al usuario dos números y muestre la suma, resta, multiplicación y división de los dos.
1.11 Elabore un programa que calcule el área de un rectángulo.
1.12 Elabore un programa que calcule la longitud y el área de una circunferencia.
1.13 Elabore un programa que calcule el área de un triángulo.
1.14 Queremos construir un programa que pida los datos necesarios para calcular y mostrar por pantalla el volumen de una cisterna.
1.15 Confeccione un programa para calcular el volumen de un cilindro circular recto.

$$VC = \pi R^2 h$$

1.16 Confeccione un programa para calcular la distancia recorrida por un automóvil.

$$S = VT$$

1.17 A un trabajador se le entrega un artículo electrodoméstico para pagarlo por descuento. Confecciones un programa que calcule la cantidad de meses demorará en pagar el artículo, si sabemos que mensualmente le descontarán el 15% de su salario.
1.18 Elabore un programa que pide al usuario un número y de este muestre:

- El cuadrado.
- La raíz cuadrada.
- El sucesor.
- El antecesor.
- El opuesto.
- El inverso.

1.19 Escribir un programa que calcule la velocidad de un proyectil que recorre **E** kilómetros en **T** minutos. Expresar el resultado en metros /segundo.

Velocidad = espacio / tiempo

1.20 Escribir un programa que teniendo los datos necesarios evalúe la siguiente expresión:

a) (a+7 *c) / (b+2-a)+2 *b
b) (a + 5) * 3 / b – 1
c) $\left(-b+\sqrt{\left(b^2-4*a*c\right)}/\left(2*a\right)\right)$

1.21 Escribir un programa que detecte si un número introducido desde le teclado es positivo o negativo. Si es cero mostrar un mensaje referente al caso.
1.22 Elaborar un programa que pida dos números y este devuelva cuál es el mayor y cuál el menor. De ser iguales mostrar un mensaje referente al caso.
1.23 Dado tres números determinar el mayor y el menor o si son iguales.
1.24 Dado tres números cualquiera determinar:

a) Si se han introducido en orden creciente.
b) Si se han introducido en orden decreciente.
c) Si se han introducido desordenadamente.
d) Si son iguales.

1.25 Crear un programa que pida dos números A y B y muestre si A es múltiplo de B o no.

1.26 Dado un número entero cualquiera entrado por el usuario, determinar si el mismo es par, impar o si es neutro.

1.27 Confeccione un programa que dado un número del 1 a 7 escriba el correspondiente nombre del día de la semana. Si se entra cualquier otro número el programa emitirá un mensaje de error.

1.28 Escribir un programa que lea desde teclado el importe bruto de una factura y determine el importe neto según los siguientes criterios.

· Importe bruto menor de 2 00 -> sin descuento

· Importe bruto mayor o igual de 2 00 -> 15% de descuento

1.29 Conociendo la longitud de tres segmentos de rectas, determinar si con ellos se puede formar un triángulo. Si esto es posible clasifíquelos en equiláteros, isósceles o escaleno.

1.30 Dado tres números cualesquiera:

a) Si los tres son iguales calcular su suma
b) Si solo dos de ellos son iguales calcular la media de los tres.
c) Si son diferentes los tres determinar cuál es el mayor y cuál el menor.

1.31 Construir un programa para evaluar la función:

F(x)= 0 si x< -1

F(x)= 0,45 * x + 2 si -1 <= x <0

F(x)= x * 1/3 + 6 si x>= 0

1.32 Se conoce el peso en libras de un atleta de boxeo, se sabe que oscila entre 124 lb. y 140 lb. se desea precisar si el atleta está en la división de los 57, 60 ó 63,5 Kg.

Nota: División - 57 Kg. Desde 124 hasta 129 lb.
División - 60 Kg. Hasta 137 lb.
División – 63,5Kg hasta 145 lb.

1.33 Escriba un programa que pregunte en que sentido se desea realizar la conversión de Radianes a Grados o de Grados a Radianes, y luego pida el valor necesario.

Para convertir de Grados a Radianes:

VR =(π* VG)/180

Para convertir de Radianes a Grados:

VG =(180* VR)/ π

1.34 Elabora un programa que pida el nombre de usuario y una contraseña. Si el nombre de usuario y la contraseña coinciden el programa mostrará un mensaje de bienvenida y en caso contrario un mensaje de error.

Nota: El nombre del usuario es: ***batalla***
La contraseña es: ***ideas***

1.35 Elabore un programa que muestre los números sucesivos del 0 al 100 utilizando cualquiera de los tres tipos de bucles y la función INC.

1.36 Elabore un `programa que muestre los números del 0 al 100 con saltos de 10. Ejemplo:

0 10 20 30 40 50 60 70 80 90 100

1.37 Modifique el programa del ejercicio 1.36 para que muestre cada número en línea diferentes.

1.38 Modifique el programa del ejercicio 1.37 para que muestre el siguiente resultado mediante un conteo descendente con la función DEC:

```
100
90
80
70
60
50
40
30
20
10
0
```

1.39 Cree un programa que pida al usuario un número entero y devuelva su tabla de multiplicación del 0 al 10. Ejemplo:

```
Entre un número: 6
0 * 6 = 0
1 * 6 = 6
2 * 6 = 12
3 * 6 = 18
4 * 6 = 24
5 * 6 = 30
6 * 6 = 36
7 * 6 = 42
8 * 6 = 48
9 * 6 = 54
10 * 6 = 60
```

1.40 Modifique el programa 1.39 para que cicle mientras que el usuario quiera seguir calculando. El programa muestra la tabla del número introducido y pregunta si desea introducir otro número, en caso afirmativo el programa se vuelve a ejecutar y si no se cierra.

1.41 Partiendo del ejercicio 1.34. Modifique este programa para que cicle la escritura del nombre de usuario y su contraseña mientras esta sea incorrecta. Si el usuario se equivoca cuatro veces entonces el programa emitirá un mensaje diciendo “Usted no puede entrar” y el programa se sierra automáticamente después de esperar unos tres segundos.

1.42 Hacer un programa que permita determinar la cantidad de días en los que las precipitaciones caídas estuvieron por encima de 44 milímetros. Conociendo que la cantidad de precipitaciones caídas por días se encuentran expresadas en un dato cuyo final esta dado por -1.

1.43 En una lista se tiene el aporte hecho por N estudiante a las MTT. Calcular el aporte realizado por los estudiantes, cuántos estudiantes han aportado más de $10.00 y cuál fue el mayor aporte realizado.

1.44 En un trabajo voluntario en la caña part6iciparon N brigadas. Se conoció la cantidad de caballerías que limpió cada brigada y se quiere hacer un programa que calcule:

a) El total de caballerías que limpiaron en total.
b) Cuántas brigadas limpiaron más de 10 caballerías.
c) Si todas las brigadas limpiaron más de 10 caballerías clasificar el trabajo voluntario en “TRABAJO VOLUNTARIO PRODUCTIVO”

1.45 Dadas las edades de un grupo de N alumnos elabore un programa que determine la edad promedio del grupo, la mayor y la menor edad.

1.46 Calcule los múltiplos de un número A, mientras los mismos no excedan de 100.

1.47 Utilizando las técnicas aprendidas en este capítulo, escriba un programa que calcule los cuadrados y los cubos de los números del 1 al 10. El programa imprimirá la siguiente tabla:

Número	Cuadrado	Cubo
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

1.48 Escribir un programa que calcule el factorial de un número. Por ejemplo:

- Factorial de 3 = 24 (porque 1*2*3*4=24)
- Factorial de 6 = 720 (porque 1*2*3*4*5*6=720)

1.49 Elabore un programa que muestre todos los números primos que hay entre NI y NF. Agregue la posibilidad de contar la cantidad que hay. Para realizar este ejercicio puedes ver el ejercicio 6 del tema 2.8 en este capítulo.

1.50 Realice un programa que pida al usuario una palabra y este determine si es palíndromo o no.

Palíndromo: Palabra o frase que se lee igual de izquierda a derecha, que de derecha a izquierda.

Ejemplos de palabras palíndromo:

reinier, ala, ana, sallas, oso, anilina.

1.51 Realice un programa que pida al usuario un número y este determine si es capicúa o no.

Capicúa: Un número que es igual leído de izquierda a derecha que de derecha a izquierda, como 1331, 121,333,898,183381,etc.

1.52 En pascal mediante la función SQR podemos obtener el cuadrado de un número cualquiera, pero ¿qué pasa si queremos calcular 4^5 ó 6^7? Elabore un programa que le pida al usuario dos números, tome como base al primero y como exponente al segundo. El programa mostrará en pantalla la potencia que representa esos números.

1.53 Elabore un programa que pida al usuario un número entero positivo y devuelva su equivalente en el sistema numérico binario.

1.54 Elabore un programa que pida al usuario un número entero positivo y devuelva su equivalente en el sistema numérico binario, octal y hexadecimal. Ejemplo de programa para el número 20.

Entre un número entero positivo: 20			
Decimal	**Binario**	**Octal**	**Hexadecimal**
20	10100	24	14

1.55 La computadora está jugando un papel creciente en la educación. Realice un programa que ayudaría a un alumno de una escuela primaria a ejercitar su habilidades a la hora de multiplicar. El programa deberá general dos números aleatorios de 0 a 10. A continuación deberá escribir una pregunta que le indique al pionero que debe multiplicar los dos números y escribir la respuesta.

Su programa verifica la respuesta. Si es correcta, imprime de manera aleatoria una de las respuestas correctas que aparecen al final y a continuación solicita otra multiplicación. Si la respuesta es incorrecta, imprimirá de manera aleatoria una de las respuestas incorrecta que aparecen al final y a continuación permitirá que el alumno vuelva a intentar responder la misma pregunta, hasta que al final conteste correctamente.

- -

Respuestas correctas:

- Muy bien.
- Excelente.
- Buen trabajo.
- ¡OH! Sigue así.

- -

Respuestas incorrectas:

- No. Por favor intente de nuevo.
- Incorrecto. Intente una vez más
- No se rinda
- No. siga intentando

- -

1.56 Elabore un programa que simule el juego “Adivine el Número”. Su programa escoge un número al azar en el rango del 1 al 1000. El jugador tendrá 10 intentos para adivinar el número. Siempre que el jugador escribe un número incorrecto el programa responde de las siguientes formas:

- En caso que el número generado sea mayor que el número introducido:

Respuesta: “El número a adivinar es mayor.”

- En caso que el número generado sea menor que el número introducido:

Respuesta: “El número a adivinar es menor.”

El jugador gana cuando adivine el número generado por el programa y aparecerán los dos números parpadeando en pantalla. Si el jugador consume todos los intentos sin haber adivinado el número, el programa mostrará el número generado en pantalla parpadeando y un mensaje indicando que perdió.

Cuando termine el juego independientemente del resultado el programa debe preguntar si se desea adivinar otro número. Si la respuesta es si, el programa debe ejecutarse nuevamente y si es no, saldrá de inmediato.

Capítulo 3

Estructuras de datos

Objetivos

- Caracterizar las estructuras de datos.
- Declarar arreglos unidimensionales y bidimensionales.
- Hacer referencia a los elementos de un arreglo.
- Realizar operaciones básicas con arreglos(Recorrido, Búsqueda, Inserción, Eliminación y Ordenamiento).

Pensamientos

◘ El saber no se obtiene por azar. Hay que buscarlo con afán y alimentarlo con diligencia.

Abigail Adams.

◘ Dos pueden lograr lo que resulta imposible para uno.

Proverbio Árabe.

◘ No podemos ayudar a otros a subir una montaña si acercarnos a la cima nosotros mismos.

Norman Schwarzkopf.

◘ "Errar es humano", sobre todo cuando uno es el que comete el error.

J. Soares.

◘ Un hombre es rico en proporción a las cosas que puede tener por sí mismo.

Henry D. Thoureau.

Sinopsis

3.1 Datos estructurados.
3.2 Arreglos de una dimensión.
3.3 Operaciones con arreglos.
3.4 Arreglos de dos dimensiones.
3.5 Resumen.
3.6 Ejercicios de autoevaluación.
3.7 Respuestas de ejercicios de autoevaluación.
3.8 Ejercicios propuestos.

3.1 Datos estructurados.

Si tuviéramos la necesidad de hacer un programa que hallara el promedio alcanzado en una asignatura determinada de un grupo de N estudiantes, lo resolveríamos con las herramientas de programación que hasta ahora se han estudiado, pero si además del promedio se necesitara obtener un listado ordenado con los nombres y las notas alcanzadas por los estudiantes para conformar un escalafón, ya las cosas se enredarían porque se utilizarían un gran número de variables para la obtención de este fin, esto realmente no es conveniente por el gran trabajo que esto requiere, en Pascal existe un tipo de dato que permite la facilidad del trabajo con listas de datos por lo que resolvería este problema de manera sencilla.

Los datos pueden ser simples o estructurados, los simples como se estudió en temas anteriores pueden almacenar un único valor en cada variable y los compuestos o estructurados están formados por tipos de datos simples. El estudio de estas estructuras comprenden un análisis de cómo se combinan las estructuras simples para formar las compuestas. En este tema se trabajará con una de las estructuras de datos más utilizadas a la hora de programar, los Arreglos.
Un arreglo (**Array**) se define como una colección finita, homogénea y ordenada de elementos.

Finita: Todo **Array** tiene un límite, es decir, debe determinarse cual será el número, máximo de elementos que formarán parte del **Array**.
Homogéneo: Todos los elementos de un **Array** son del mismo tipo(todos enteros, todos boleanos, etcétera), pero nunca una combinación de distintos tipos.
Ordenada: Se puede determinar cual es el primer elemento, el segundo elemento, el tercero, ...y el enésimo elemento.

La cantidad de elementos del arreglo se específica cuando se crea éste, en la zona de declaraciones del programa, definiendo el número de dimensiones o número de índices del mismo y los límites máximo y mínimo que cada uno de ellos puede tomar, que llamaremos rango. Según sea este número, diferenciaremos los siguientes tipos de arreglos:

- Unidimensionales. (Vectores)
- Bidimensionales. (Matrices)
- Multidimensionales.

3.2 Arreglos de una dimensión.

Un arreglo unidimensional puede ser considerado como una lista(es decir, una columna) de datos del mismo tipo, a los que colectivamente nos referiremos mediante un nombre.

Un arreglo llamado Notas que tenga espacio para almacenar cinco notas(5, 2, 5, 2, 4) se pueden ver forma abstracta en la figura 3.1.

Notas

5	2	3	2	4

Figura 3.1 Ejemplo de arreglo de una dimensión expresado de forma abstracta.

Para declarar un arreglo de un dimensión Figura 3.2, se realiza en la zona de declaraciones, al igual que para declarar las variables se coloca Var, luego el nombre con que se identificará el arreglo, la palabra reservada Array y dentro de corchetes el rango de elementos que tendrá dicho arreglo seguido del tipo de dato.

```
Var
  Nombre : Array [Valor_Inicial..Valor_Final] of Tipo de dato;
```

Figura 3.2 Estructura general de un arreglo unidimensional.

El ejemplo que se pone a continuación define la variable **Grupo** como un arreglo que contiene 10 datos de tipo Integer.

```
Var
Grupo : Array [1..10] of Integer;
```

Podemos imaginarnos el arreglo Grupo como una fila de diez elementos del tipo que hemos especificado (Integer). El arreglo Grupo quedaría así:

Grupo										
Número de orden.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Elementos.										

Para acceder a un elemento cualquiera de un arreglo se hace mediante el nombre del arreglo y el número del elemento a través el subíndice del mismo.
Ejemplo:
En el programa de la Figura 3.3 se declara un Arreglo llamado *Paises* y se asigna en el momento de su declaración los elementos que lo componen.

```
Program Array_Paises;
Uses
  CRT;
 Const  LI=1;  LS=8;
 Var
  Paises: ARRAY [LI..LS] of String =( 'Cuba','Venezuela','Bolivia','México'
                                      ,'Panamá','Chile','Ecuador', 'Hondura');
Begin
 CLRSCR;
  Writeln (Paises[1]); { Muestra en pantalla Cuba }
  Writeln (Paises[7]); { Muestra en pantalla Ecuador }
 Readln;
End.
```

Figura 3.3 Ejemplo de cómo acceder a un elemento de un arreglo.

Para asignar un valor a un elemento de un arreglo, podemos hacerlo de la siguiente forma:

Nombre[6] **:=** 'Antonio';

Ahora en el elemento 6 del arreglo *Nombre* contiene el valor *Antonio.*

3.3 Operaciones con arreglos.

Las operaciones que se suelen utilizar habitualmente sobre una estructura lineal son las siguientes:

- Recorrido. Procesamiento de cada elemento de la lista.
- Búsqueda. Búsqueda de la posición ocupada por un elemento con un determinado valor.
- Inserción. Adición de un nuevo elemento a la lista.
- Borrado. Eliminación de un elemento de la lista.
- Ordenación. Organizar los elementos de la lista de acuerdo con algún tipo de orden.
- Mezcla. Combinar dos listas en una.

Cuando se habla de lista en esta sección, nos referimos a un arreglo de una dimensión o unidimensional. Vamos a ver cada una de estas operaciones:

Recorrido:

La operación de recorrido consiste en visitar todos los elementos del arreglo por lo menos una vez.

Los algoritmos de la Figura 3.4 recorren el arreglo *Paises* y van mostrando cada uno de sus elementos. Este arreglo cuenta con un límite inferior *LI* y un límite superior *LS*, los elementos que lo compone son todos de texto. La variable *I* es la que va recorriendo el arreglo.

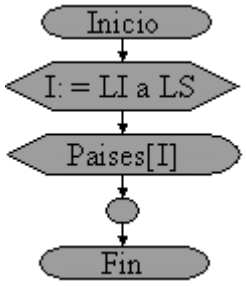
Pseudocódigo	Diagrama de flujo
Inicio Para I:= LI hasta LS hacer Escribir Paises[I] Fin_Para Fin	

Figura 3.4 Algoritmos recorrido del arreglo Paises.

En la Figura 3.5 se presenta el programa que realiza el recorrido del arreglo *Paises* partiendo de los algoritmos de la Figura 3.4.

```
Program Recorrido;
Uses
  CRT;
 Const  LI=1;  LS=8;
 Var
  Paises: ARRAY [LI..LS] of String =( 'Cuba','Venezuela','Bolivia','México'
                                      ,'Panamá','Chile','Ecuador', 'Hondura');
  I: Integer;
Begin
 CLRSCR;
 For I:=LI to LS do
  Writeln (Paises[I]); { Muestra en pantalla el contenido del elemento I }
 Readln;
End.
```

Figura 3.5 Programa Recorrido.

A continuación se muestra en la Figura 3.6 un programa que rellena y muestra un arreglo de 10 elementos con números aleatorios comprendidos entre 1 y 100.

```
Program Rellena_Muestra;
Uses
  CRT;
 Var Arr_num: ARRAY [1..10] of INTEGER;
 Var  i: INTEGER;
Begin
 Randomize;
   For i:=1 to 10 do
   begin
    Arr_num[i]:=Random(100)+1;
    Writeln(Arr_num[i]);
   end;
 Readln;
End.
```

Figura 3.6 Programa que rellena un arreglo con números aleatorios y los muestra.

Búsqueda:

En ciertas ocasiones es necesario saber si un elemento cualquiera está o no en un arreglo. Los algoritmos de la Figura 3.7 realizan una búsqueda, para entender estos algoritmos debemos tener claro que *Dato* es el nombre del arreglo, *N* es la cantidad de elementos que lo componen, *i* es la variable que controla en ciclo y *Elemento* es el elemento buscado. Estos algoritmos muestran la posición del elemento buscado las veces que esté.

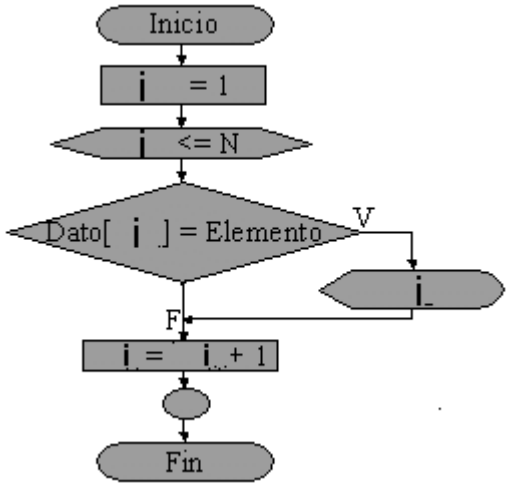
Pseudocódigo	Diagrama de flujo
Inicio i = 1 Mientras i <= N hacer Si Dato[i] = Elemento Entonces Escribir i Fin_Si i = i +1 Fin_Mientras Fin	

Figura 3.7 Algoritmos Búsqueda.

Estos algoritmos se utilizan para saber la posición que ocupa un elemento determinado, donde asumimos que dicho elemento siempre aparece en el arreglo. En la práctica nos percatamos que no siempre el elemento que se busca aparece en el arreglo, el algoritmo anterior no es eficiente ya que el programa no realiza ninguna acción si el elemento buscado no está en lista, es decir, en el arreglo. El programa de la Figura 3.8 se encarga de validar estos pequeños detalles que en el mundo de la programación se le conoce como tratamiento de errores.

```
Program Buscar;
Uses
  CRT;
  Var  Dato: Array [1..10] of Integer =(34,21,33,41,34,60,74,87,90,10);
   N,I,Lug, Elemento: INTEGER;
   Hay:Boolean;
Begin
 Hay:=False;
 Lug:=1;
 Write ('Entre el número a buscar: ');Readln(Elemento);
   While Lug < 10 do
    begin
     If Dato[Lug]=Elemento Then
      begin
       Hay:=True;
       Writeln('En la posición ',Lug);
      end;
     Lug:=Lug+1;
    end;
   If Hay = False Then Writeln('El elemento buscado no esta en la lista');
 Readln;
End.
```

Figura 3.8 Programa que realiza una búsqueda.

En la Figura 3.9 se muestra un programa que después de asignar los números (23, 45, 68, 99, 10, 15 y 4) a al arreglo NUM, determina las posiciones del en las que se encuentran el valor máximo y el valor mínimo, pero que además muestre cuáles son estos valores.

```
Program Mayor_Menor;
Uses
  CRT;
 CONST NUM: ARRAY [1..7] of INTEGER=(23,45,68,99,10,15,4);
 VAR i, mayor, menor, posi_mayor, posi_menor: INTEGER;
Begin
{ Buscamos el mayor }
 Mayor := NUM[1];
  For i:=2 to 7 do
   begin
    If NUM[i] >= mayor then
     begin
      Mayor := NUM[i];
      posi_mayor :=i;
     end
   end;
  Writeln ('El número mayor es: ',mayor:3);
  Writeln ('Su posición es:     ',posi_mayor:3);
  Writeln ('');
 { Buscamos el menor }
 Menor := NUM[1];
   For i:=2 to 7 do
   begin
    If NUM[i] <= menor then
     begin
      Menor := NUM[i];
      posi_menor := i;
     end
   end;
 Writeln ('El numero menor es: ',menor:3);
 Writeln ('Su posición es:     ',posi_menor:3);
 Readln;
End.
```

Figura 3.9 Programa que muestra el valor mayor y el menor de un arreglo de números, además la posición de cada uno de ellos.

Inserción:

A la hora de realizar una inserción de un elemento en un arreglo equivale a una simple asignación indicándole la posición en la que se quiere insertar. Por ejemplo, si queremos insertar en un arreglo llamado *Notas*, el número 100 en la posición 5 se haría de la siguiente forma:

Notas[5]:= 100;

La inserción se complica cuando tenemos un arreglo de 10 elementos (por citar un ejemplo), de ellos solo están ocupados 6 y queremos insertar el número 100 en la posición 3, de forma tal que los elementos que están en la posiciones 3, 4, 5 y 6 se trasladen una posición más arriba. Observe el ejemplo en la Figura 3.10.

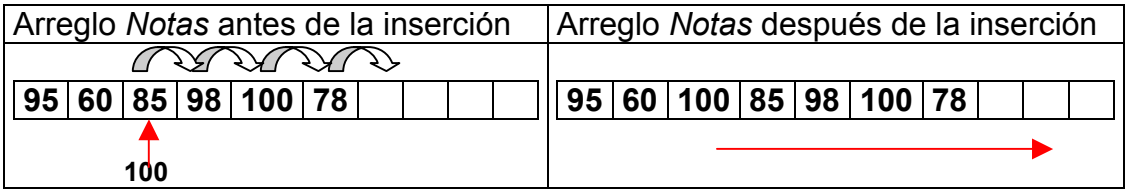


Figura 3.10 Ejemplo de inserción en un arreglo.

Como ves, luego de insertar el 100 en la posición Notas[3], los elementos se corrieron un elemento hacia la derecha. El algoritmo de la figura 3.11 les permite insertar un elemento en una posición determinada, donde *Notas* es el nombre del arreglo, *N* es la cantidad de elementos no vacíos que lo componen (6 en el ejemplo de la figura 3.10), *K* es la posición en la que queremos insertar el elemento, i es un controlador y *Elemento* es el nuevo dato que queremos insertar.

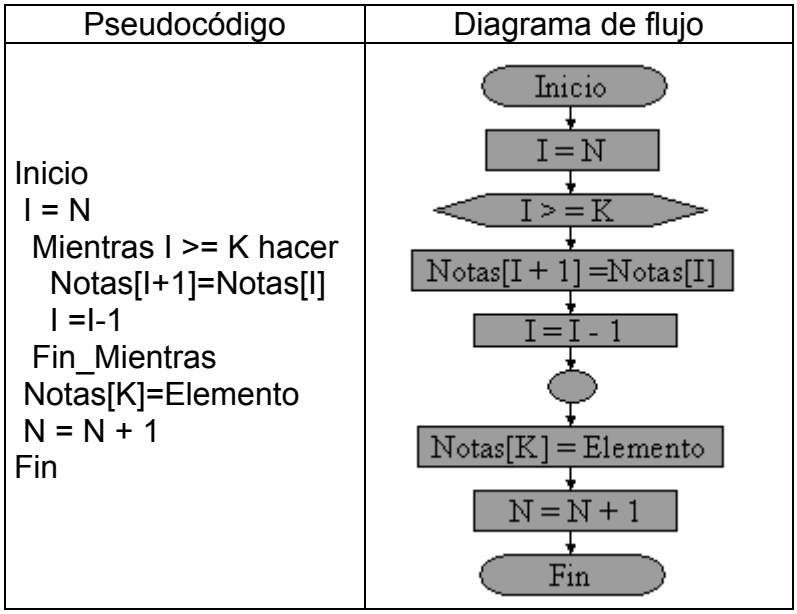


Figura 3.11 Algoritmos Inserción.

En el programa de la Figura 3.12 se muestra la operación de inserción mediante un arreglo conformado por nombres de personalidades de nuestra revolución. Cuando arranca el programa le asignamos siete nombres al arreglo *Personalidades* para que luego el usuario introduzca el nombre de alguna otra personalidad en la posición que desee. En el programa se explica que acción realiza cada operación.

```
Program Array_Insert;
Uses
  CRT;
var Personalidades: ARRAY [1..8] of String;
 N,I,K:Integer;
 Elemento: String;
Begin
 Personalidades [1]:='Fidel Castro';
 Personalidades [2]:='Raúl Castro';
 Personalidades [3]:='Camilo Cienfuegos';
 Personalidades [4]:='Ernesto Guevara';
 Personalidades [5]:='Juan Almeida';
 Personalidades [6]:='Celia Sánchez';
 Personalidades [7]:='Magali Montané';
 I:=1;
 N:=0;
 While Personalidades [I]<> '' do { Para tener la cantidad de }
  begin                              { elementos no vacíos del arreglo. }
   N:=N+1;
   I:=I+1;
  end;
 For I:=1 to n do {Para mostrar los elementos del arreglo.}
  Writeln(LA[I]);
  Writeln('');
  Writeln('Para insertar un nombre de una personalidad al arreglo.');
  Write('Entre el nombre de la personalidad: ');
  Readln(Elemento); {Almacena el nombre.}
  Writeln('');
  Write('Entre la posición: ');Readln(K); {Almaceno la posición.}
  I:= N;
 While I >= K do {Mientras queden elementos por mover.}
  Begin
 {Traslado un elemento a la derecha.}
   Personalidades [I + 1]:= Personalidades [I];
   I:=I-1;
  end;
  Personalidades [K]:=Elemento; { Asigno el nombre. }
  N:= N + 1;                         { Actualizo la cantidad de elementos.}
  For I:=1 to n do { Para mostrar los elementos del arreglo.}
  Writeln(Personalidades [I]);
 Readln;
End.
```

Figura 3.12 Programa inserción de un elemento.

Borrado:
Al igual que en la inserción, a la hora de realizar el borrado de algún elemento en un arreglo implica que todos los elementos que le siguen a la derecha tienen que trasladarse un lugar a la izquierda. Esto que se explicó anteriormente hay que hacerlo por una de las características de los arreglos, me refiero al que son estructuras de datos secuenciales, o sea, que no pueden haber espacios intermedios en blanco. Observe la Figura 3.13.

Figura 3.13 Ejemplo de eliminación en un arreglo

Como ves, luego de eliminar el 60 en la posición Notas[2], los elementos se corrieron una posición hacia la izquierda. El algoritmo de la Figura 3.14 les permite eliminar un elemento en una posición determinada. Donde *Notas* es el nombre del arreglo, *N* es la cantidad de elementos no vacíos que lo componen(7 en el ejemplo anterior), *K* es la posición del elemento que queremos eliminar, *I* es un controlador.

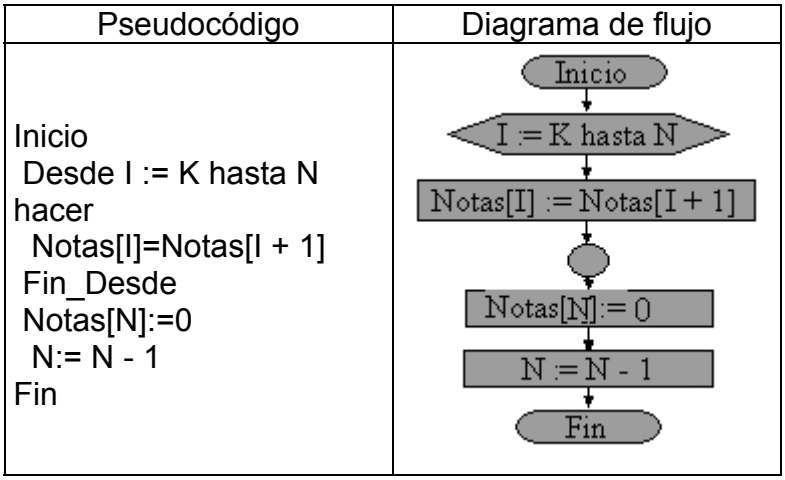


Figura 3.14 Algoritmo eliminación.

El ejemplo de la Figura 3.15 utilizamos una lista de 7 nombres que están almacenados en un arreglo. Aquí se le muestra al usuario la lista y la opción de eliminar el elemento que desee mediante su posición.

En este caso utilizamos un arreglo llamado LA que puede almacenar hasta 8 nombres. Mediante K eliminamos el elemento perteneciente a dicho valor.

```
Program Array_Eliminar;
Uses
  CRT;
var LA: ARRAY [1..8] of String;
 N,I,K:Integer;
Begin
 CLRSCR;
 LA[1]:='Rodiert';
 LA[2]:='Reinier';
 LA[3]:='Fernando';
 LA[4]:='Lázaro';
 LA[5]:='Juan';
 LA[6]:='Yilian';
 LA[7]:='Bruno';
 I:=1;
 N:=0;
 While LA[I]<> '' do
  begin
   N:=N+1;
   I:=I+1;
  end;
 For I:=1 to n do Writeln(I,'.- ',LA[I]);
 Writeln('');
 Writeln('Para borrar un elemento de la lista');
 Write('Posición del elemento a borrar: ');Readln(K);
 For I:=K to N do  LA[I]:= LA[I + 1];
 N:=N-1;
 Writeln('');
 For I:=1 to n do  Writeln(I,'.- ',LA[I]);
 Readln;
End.
```

Figura 3.15 Programa eliminación.

Ordenación:

Cuando trabajamos con lista, ya sea tanto de números como de textos, debemos conocer cómo ordenar dichas listas para cuando nos haga falta realizar esta operación tan utilizada en estos tipos de datos (estructurados).

La ordenación puede ser de dos formas (Ascendente y Descendente). La primera consiste en ordenar todos los elementos de menor a mayor Figura 3.16 A, mientras que la segunda ordena los elementos de mayor a menor Figura 3.16 B.

A	B
1	5
2	4
3	3
4	2
5	1

Figura 3.16 Formas de ordenar. A - Ascendente.
B - Descendente.

El programa de la figura 3.17 realiza una ordenación de un arreglo para mostrarlo en forma de escalafón, o sea, de forma descendente. Este programa pide al usuario 10 nombres con su nota en base a 100 y al final muestra un escalafón de ese grupo de estudiantes.

```
Program Ordenar;
Uses CRT;
Var    Notas : Array [1..10] of Integer;
       Nombres: Array [1..10] of String;
 i, j, AuxInt:Integer;
AuxStr:String;
Begin
CLRSCR;
Writeln;
For i:=1 to 10 do {Almacena los nombre y las notas entradas}
begin
 Write('Entre el nombre del alumno ',i,': ');
 Readln(Nombres[i]);
 Write('Entre la nota de ', Nombres[i],': ');
 Readln(Notas[i]);
end;                    {fin de la entrada de datos}
For I:=1 to 10 do              {comienzo a ordenar}
begin
 For j:=1 to 10-I do
 begin
  If Notas[j]<Notas[j+1]Then
  begin                            {intercambio los valores}
   AuxInt:= Notas[j];
   AuxStr:= Nombres[j];
   Notas[j]:=Notas[j +1];
   Nombres[j]:= Nombres[j +1];
   Notas[j +1]:= AuxInt;
   Nombres[j +1]:=AuxStr;
  end;   {fin del intercambio}
 end;
end;            {fin del ordenamiento}
{se muestran los resultados del ordenamiento}
ClrScr;
Writeln;
Writeln('*** Escalafón del grupo ***');
For i:=1 to 10 do
 begin
  J:=i+1;
  Gotoxy(3,j*2);  Write(I,'.- ',Nombres[i]);
  Gotoxy(20,j*2); Write(Notas[i]);
 end;
Readln;
End.
```

Figura 3.17 Programa ordenación.

En este programa luego de haber introducido los datos necesarios se comienza el ordenamiento. En la primera iteración del primer bucle For del ordenamiento, garantizamos que quede el valor mínimo en el último elemento de los arreglos Nombres y Notas. En la segunda iteración garantizamos que el valor mínimo de los restante elementos quede en el penúltimo y así sucesivamente hasta que todos los elementos de los arreglos queden ordenados de mayor a menor. Tenga en cuenta que la comparación se realiza a partir del arreglo Notas, ya que lo que se quiere lograr es un escalafón.
A la hora de mostrar los resultados se utilizó la instrucción Gotoxy, coloca el cursor en las coordenadas especificadas por X y Y, donde X representa la columna partiendo de izquierda a derecha y Y representa la fila partiendo de arriba hacia abajo. Ver ejemplo de la Figura 3.18.

```
Program posicion;
uses crt;
Begin
 ClrScr;
 Gotoxy(10,10);
 Write('Fila 10, Columna 10');
 Gotoxy(20, 20);
 Write('Fila 20,Columna 20');
  Readln;
End.
```

Figura 3.18 Ejemplo del uso del Gotoxy.

Mezcla:
Esta operación consiste en hacer una combinación entre dos a más arreglos teniendo en cuenta el objetivo que se quiera alcanzar. Muchas veces tenemos una lista de números y queremos almacenar hacer otra lista donde aparezcan todos los números que cumplan con una determinada condición. En la figura 3.19 se presenta un programa que resuelve la siguiente situación problémica: "Escribir un programa en BP7 que rellene un arreglo con cinco números enteros consecutivos y haga una copia de ese arreglo en otro".

```
Program Copia;
Uses crt;
var arr_num1,arr_num2:array [5..10] of integer;
 i: Integer;
Begin
  Clrscr;
    For i:=5 to 10 do
     begin
       arr_num1[i]:=i;
       arr_num2[i]:=arr_num1[i];
     end;
    For i:=5 to 10 do
      Writeln (arr_num2[i]);
  Readln;
End.
```

Figura 3.19 Ejemplo de programa que copia un arreglo en otro.

Independientemente que en este ejemplo se muestra cómo es posible asignarle un elemento determinado de un arreglo a un elemento determinado de otro arreglo, no satisface mi necesidad con respecto a lo que se puede hacer con esta operación. La figura 3.20 cuenta con un programa ejemplo donde a partir de una lista de 10 números introducidos por el usuario, estos se almacenan en un arreglo y luego almacenamos en otro los números que cumplan con la condición de ser pares.

```
Program Mezcla;
Uses crt;
var arr_num1, arr_num2:array [1..10] of Integer;
I,Cont: Integer;
Begin
Clrscr;
Cont:=1;
Gotoxy(2,2);
For i:=1 to 10 do    { Entran y se almacenan los números en arr_num1}
begin
Write('Entre un número: ');
Readln(arr_num1[i]);
gotoxy(2,2);
clreol;
end;                 { fin de la entrada}
For i:=1 to 10 do
begin                { Rellenar arr_num2 con los números pares de arr_num1}
If arr_num1[i] mod 2 = 0 Then
begin
arr_num2[Cont]:= arr_num1[i];
Inc(Cont);
end;
end;                 { fin del relleno}
Writeln('ARREGLO 1          ARREGLO 2');
for i:=1 to 10 do    {contenido del arreglo 1}
begin
 gotoxy(5,i+2);
 write(arr_num1[i]);
end;
for i:=1 to cont-1 do {contenido del arreglo 2}
begin
 gotoxy(24,i+2);
 write(arr_num2[i]);
end;
Readln;
End.
```

Figura 3.20 Programa que utiliza el contenido de un arreglo para formar otro.

En este programa se utiliza una instrucción nueva, CLREOL. Este borra todos los caracteres de la línea actual desde la posición del cursor hasta el final de la línea. Ver el programa de la figura 3.21.

```
Program Borrar_Linea;
uses crt;
var x,y : integer;
Begin
for x := 1 to 24 do
for y := 1 to 80 do
write('#');
gotoxy(15,15);
clreol;
Readln;
End.
```

Figura 3.21 Ejemplo del uso de la instrucción CLREOL.

Como pueden apreciar en este programa se rellena la pantalla con el símbolo de # y luego se borra la línea 15 a partir de la posición 15.

3.4 Arreglos de dos dimensiones.

La concepción de arreglo no se restringe sólo a los arreglos de una dimensión, como se había planteado al principio del capítulo también pueden definirse arreglos multidimensionales, que son muy utilizados en aplicaciones de BP7. Por ejemplo, un arreglo de estos es muy útil para describir un objeto que tenga físicamente dos dimensiones, tal es el caso de un mapa, un juego de damas o de ajedrez. Es útil para un programa para almacén de departamento que tenga 20 sucursales, cada una de las cuales vende 30 artículos.

En este capítulo se observó que un arreglo unidimensional puede entenderse como una lista (que se mostraba como una columna de datos). Un arreglo multidimensional (Matriz) tiene una filosofía de trabajo bastante parecida a la de los arreglos unidimensionales.

Una matriz se pudiera imaginar como una tabla de datos, la cual está compuesta por filas y columnas, ver figura 3.22.

	1	2	3	4	5	6	7	8	9	10
1										
2	ss								jj	
3				xx						
4										
5							yy			
6		pp								
7										
8										zz
9								ii		
10			qq							

Figura 3.22 Tabla de 10x10 que representa una matriz.

Partiendo del ejemplo de la figura 3.22, podemos definir este arreglo de la siguiente forma:

```
var
Tabla : Array [1..10,1..10] of String;
```

Así, la primera dimensión (el primer índice) podría referirse al número de la fila, la segunda dimensión (el segundo índice) al número de columnas. De esta forma, con Tabla[3,4] nos estaríamos refiriendo al elemento que he marcado con "xx" y con Tabla[5,7] nos referiríamos al elemento marcado con "yy".

Con los conocimientos alcanzados en este capítulo podemos hacer de manera sencilla un recorrido de este tipo de arreglo. En la figura 3.23 se muestra un programa que asigna en el momento de su declaración una serie de números a una matriz con datos fijos (declarada como una constante) y posteriormente la muestra haciendo un recorrido por cada uno de sus elementos.

```
Program Recorrido;
Uses CRT;
 const arr_num: array [1..2,1..6] of integer=( (3,6,9,7,8,0),(1,4,3,2,7,0) );
 var i, j: integer;
Begin
 CLRSCR;
 Writeln;
 for i:=1 to 2 do
  begin
   for j:=1 to 6 do
     write (arr_num[i, j]:3);
   Writeln;
  end;
 Readln;
End.
```

Figura 3.23 Recorrido de una matriz.

A continuación se resolverán dos problemas donde se tenga que aplicar por necesidad arreglos de dos dimensiones.

Ejemplo 1:

Confeccione un programa que pida al usuario el nombre de tres estudiantes y sus notas en las asignaturas de Matemática, Historia y Español y a continuación muestre el promedio de cada alumno de la siguiente forma:

Nombre	Matemática	Historia	Español	Promedio
Nombre1	100	90	90	93.3
Nombre2	80	80	80	80.0
Nombre3	70	100	96	88.6

<u>Solución</u>:

Como pueden apreciar aquí hay datos que son del mismo tipo. Estos datos están organizados por tres filas y cuatro columnas, por lo que se puede utilizar una matriz para darle solución a este problema.

```
Program Promedio;
Uses CRT;
var Notas: array [1..3,1..4] of real;
Nombres: array[1..3] of String;
i, j: integer;
suma:real;
Begin
for i:=1 to 3 do
begin
CLRSCR;
Writeln;
suma:=0;
Write('Entre el nombre ',i,': '); Readln(Nombres[i]);
for j:=1 to 3 do
begin
Writeln;
Case j of
1: begin
    Write('Nota de Matemática: ');Readln(Notas[i, j]);
   end;
2: begin
    Write('Nota de Historia: ');Readln(Notas[i, j]);
   end;
3: begin
    Write('Nota de Español: ');Readln(Notas[i, j]);
   end
end;
suma:= suma + Notas[i, j];     { Variable sumadora }
end;
Notas[i,4]:=suma / 3;              { Calcula el promedio }
end;
Clrscr;
Writeln;
Writeln('Nombre     Matemática   Historia     Español      Promedio');
For I:= 1 to 3 do
begin
gotoxy(1,i+2);
Write(Nombres[i]);
For j:=1 to 4 do
begin
gotoxy(j*13,i+2);
Write(Notas[I, j]:0:1);
end;
Writeln;
end;
Readln;
End.
```

Ejemplo 2:
Escribir un programa que determine la posición de la siguiente matriz en la que se encuentra el valor máximo.

23	45	68
34	99	12
25	78	89

Solución:

```
Program Max_Valor;
      uses crt;
      const arr_num:array[1..3,1..3] of integer=( (23,45,68),
                                                  (34,99,12),
                                                  (25,78,89) );
      var i,j,val_max,pos_max_i,pos_max_j:integer;
Begin
  Clrscr;
  Writeln;
  val_max:=arr_num[1,1];
    for i:=1 to 3 do
     begin
       for j:=1 to 3 do
        begin
          if arr_num[i,j] > val_max then
           begin
             val_max:=arr_num[i,j];
             pos_max_i:= i;
             pos_max_j:= j;
           end;
        end;
     end;
  Writeln('Valor máximo: ', val_max:3, ' Posición: ', pos_max_i:3, pos_max_j:3);
  Readln;
End.
```

3.5 Resumen.

- Un arreglo se define como una colección finita, homogénea y ordenada de elementos.
- Características de los arreglos:

Finita: Tiene un límite, eso indica que debe determinarse cual será el número, máximo de elementos que formarán parte del arreglo.
Homogéneo: Todos los elementos de un arreglo son del mismo tipo (todos enteros, todos boleanos, etcétera), pero nunca una combinación de ellos.
Ordenada: Se puede determinar cuál es el primer elemento, el segundo elemento, el tercero, ...y el enésimo elemento.

- Según la cantidad de dimensiones los arreglos se clasifican en:

- Unidimensionales. (Vectores)
- Bidimensionales. (Matrices)
- Multidimensionales.

- Estructura general de un arreglo unidimensional.

```
Var
  Nombre : Array [Valor_Inicial..Valor_Final] of Tipo de dato;
```

- Para acceder a un elemento cualquiera de un arreglo se hace mediante el nombre del arreglo y el número del elemento a través el subíndice del mismo.
- Los arreglos se definen en la zona de declaraciones del programa.
- Se puede definir un arreglo que almacene valores constantes, esto es posible mediante la palabra reservada const, ejemplo:

```
Const
  Arr_Const: array [1..5] of integer= ( 3,6,9,7,8 );
```

- Operaciones con arreglos:
- Recorrido. Procesamiento de cada elemento de la lista.
- Búsqueda. Búsqueda de la posición ocupada por un elemento con un determinado valor.
- Inserción. Adición de un nuevo elemento a la lista.
- Borrado. Eliminación de un elemento de la lista.
- Ordenación. Organizar los elementos de la lista de acuerdo con algún tipo de orden.
- Mezcla. Combinar dos listas en una.
- **Gotoxy:** Coloca el cursor en las coordenadas especificadas por X y Y, donde X representa la columna partiendo de izquierda a derecha y Y representa la fila partiendo de arriba hacia abajo.
- **Clreol:** Borra todos los caracteres de la línea actual desde la posición del cursor hasta el final de la línea.
- Estructura para definir un arreglo de dos dimensiones:

```
                                Segunda dimensión
Var
Nombre : Array [V_Ini..V_F, V_Ini..V_F] of Tipo de dato;
                Primera dimensión
```

3.6 Ejercicios de autoevaluación.

1.1 Diga cuáles de las siguientes proposiciones cumplen la con definición de arreglo:

a) ___ Arreglo: Conjunto de elementos entrelazados que permite almacenar datos de diferentes tipos.

b)___ Arreglo: Lista fija, semejante y no ordenada de elementos.

c) ___ Arreglo: Colección finita, homogénea y ordenada de elementos.

1.2 Complete las oraciones siguientes:

a) Los arreglos se clasifican en _____ , _____ y _____ por la cantidad de dimensiones que cuentan.

b) Todo arreglo tiene que ser _____, ordenado y _____ para poder definirse como tal.

1.3 Enlace la columna A con la B según corresponda referente a las características de los arreglos:

Columna A	Columna B
1.- Ordenada.	___ Tiene un límite, eso indica que debe determinarse cual será el número, máximo de elementos que formarán parte del arreglo.
2.- Homogéneo.	___ Todos los elementos de un arreglo son del mismo tipo de dato, pero nunca una combinación de ellos.
3.- Finita.	___ Se puede determinar cuál es el primer elemento, el segundo elemento, el tercero, ...y el enésimo elemento.

1.4 Enlace la columna A con la B según corresponda referente a las operaciones con arreglos:

Columna A	Columna B
1.- Mezcla.	___ Búsqueda de la posición ocupada por un elemento con un determinado valor.
2.- Ordenación.	___ Procesamiento de cada elemento de la lista.
3.- Búsqueda.	___ Combinar dos listas en una.
4.- Borrado.	___ Adición de un nuevo elemento a la lista.
5.- Inserción.	___ Organizar los elementos de la lista de acuerdo con algún tipo de orden.
6.- Recorrido.	___ Eliminación de un elemento de la lista.

1.5 Declare un arreglo que reserve espacio para almacenar cada una de las siguientes situaciones:

a) Los nombres de los estudiantes de un grupo de 25 alumnos.
b) Las precipitaciones caídas en el mes de mayo.
c) Los nacimientos ocurridos en los años de 1959 al 2000.
d) Almacenar qué estudiante de los 25 de un grupo es o no de la UJC.

1.6 ¿Qué hace la instrucción GOTOXY?

1.7 ¿Qué hace la instrucción CLREOL?

3.7 Respuestas de ejercicios de autoevaluación.

1.1 a) No. b) No. c) Si.

1.2

a) Unidimensionales o Vectores, Bidimensionales o Matrices y Multidimensionales.
b) Finito, Homogéneo.

1.3 El orden es 3, 2 y 1.

1.4 El orden es 3, 6, 1, 5, 2 y 4.

1.5

a) Var Nombres : Array[1..25] of String;
b) Var Mayo : Array[1..31] of Real;
c) Var Nacim : Array[1959..2000] of String;
d) Var UJC : Array[1..25] of Boolean;

1.6 Coloca el cursor en las coordenadas especificadas.

1.7 Borra todos los caracteres de la línea actual desde la posición del cursor hasta el final de la línea.

1.8 Ejercicios propuestos.

1.9 Escribir un programa que lea diez números, los guarde en un arreglo y a continuación los imprima en pantalla por su orden de entrada.

1.10 Escribir un programa que lea cinco números, los guarde en un arreglo y almacene en la última posición la suma de los elementos que componen el arreglo, a continuación los imprima en pantalla de la siguiente forma:

1	2	3	4	5	6
100	40	70	62	50	322

1.11 Escribir un programa que lea diez números, los guarde en un arreglo y a continuación los imprima en pantalla en orden inverso.

1.12 Escribir un programa que lea diez números, los guarde en un arreglo y a continuación los ordene e imprima en de menor a mayor.

1.13 Escribir un programa que lea tres números y los guarde en un vector. A continuación los ordenará y guardará los valores ordenados en otro vector. Finalmente sacará ambas listas de números por la pantalla.

1.14 Escribir un programa que solicite cinco números, los almacene en un array y luego calcule la media aritmética de esos números.

1.15 Escribir un programa Pascal que tras asignar los números, -2, 5, 8, -9, 10, 15 y –4 a un array calcule, independientemente, la suma de los elementos positivos y negativos.

1.16 Escribir un programa en Pascal que tras entrar siete números y almacenarlos en un array, determine las posiciones del array en las que se encuentran el máximo y el mínimo valor.

1.17 Escribir un programa que almacene en un array los números primos comprendidos entre 1 y 100.

1.18 Escribir un programa que genera la inversa de una palabra. La palabra original y la invertida deben almacenarse en array independientes. La palabra debe tener como máximo 15 caracteres.

1.19 El modo de un arreglo de números es el número que más se repite en el arreglo. Si varios números están repetidos con igual frecuencia máxima, no existe modo al igual que si ningún número se repite. Confeccione un programa que pida diez números y muestre el modo de estos si es que lo tiene y en caso contrario muestre un mensaje.

1.20 Realizar un programa que haga una lectura de 10 números enteros, los almacene en un arreglo y calcule el factorial a cada uno de los números leídos almacenándolos en otro arreglo y a continuación los muestre.

1.21 Confeccione un programa que solicite los nombres y los dos apellidos de los estudiantes de un grupo con capacidad para veinte alumnos. El programa ofrecerá el listado ordenado alfabéticamente del grupo.

1.22 En una lista se tiene los nombres y el aporte hecho a las MTT por los 20 estudiantes de un grupo. Calcular el aporte realizado por los estudiantes, mostrar los estudiantes que han aportado más de $10.00 y el nombre del estudiante del mayor aporte realizado.

1.23 Escribir un programa que llene una matriz de 5 por 5 de valores aleatorios. A continuación el programa debe imprimirla por pantalla.

1.24 Escribir un programa que pida un número entre 1 y 20 e imprima por pantalla su tabla de sumar. Por ejemplo si el número fuera el 3 la tabla debería ser:

+	0	1	2	3
0	0	1	2	3
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6

1.25 Escribir un programa que determine la posición de la siguiente matriz en la que se encuentra el valor máximo y el valor mínimo.

15	55	67
31	89	13
28	74	99

1.26 Escribir un programa que sume, independientemente, los elementos positivos y negativos de la siguiente matriz:

-12	23	32
45	-56	-10
25	78	89

1.27 Escribir un programa que sume los elementos de cada una de las filas y de las columnas de la siguiente matriz; el resultado de cada suma se almacenará en la última posición de la fila o columna correspondiente. Además la suma total de todos los elementos de la matriz se almacenará en el elemento de la esquina inferior derecha de la matriz:

1	7	0
5	6	0
6	4	0
7	3	0
1	0	0

Capítulo 4

Registros y Ficheros

Objetivos

- Caracterizar datos estructurados de tipo registro, definirlo, referenciarlo e implementarlo en programas donde sea necesario su utilización.
- Definir arreglos de registros.
- Caracterizar datos estructurados de tipo fichero, definirlo, referenciarlo e implementarlo en programas donde sea necesario su utilización..
- Utilizar operaciones con ficheros.

Pensamientos

◘ No trates de convertirte en un hombre de éxito,
procura ser un hombre de valores.
Albert Einstein.

◘ Confíe en sí mismo. Usted sabe más de lo que cree.
Benjamín Spock.

◘ No se es superior porque se tenga inteligencia,
sino porque se sabe emplearla.
Marcela Farías H.

◘ La mejor inversión de una moneda es una llamada
telefónica al hombre que se necesite en el momento justo.
IlkA Chase.

◘ Es mejor consultar las cosas con la almohada a tiempo
que perder el sueño por su culpa después.
Baltasar Garcián.

Sinopsis

4.1 Registros.
4.2 Definición de tipos de registros.
4.3 Arreglos de registros.
4.4 Ficheros.
4.5 Constructor del tipo de dato file.
4.6 Operaciones con ficheros.
4.7 Resumen.
4.8 Ejercicios de autoevaluación.
4.9 Respuestas de ejercicios de autoevaluación.
4.10 Ejercicios propuestos.

4.1 Registros.

Cuando se resuelven problemas mediante programas, el programador en muchas ocasiones suele encontrarse en situaciones en las que se quiere representar en un tipo de dato una información que posee estructura interna y son heterogénea, es decir, los campos en los que se subdivide el dato no son todos del mismo tipo. Para estos casos BP7, como otros lenguajes de programación actuales, proporciona los datos tipo registro (records).
Un registro es un tipo de dato definido por el programador en el que puede especificar su estructura interna.

4.2 Definición de tipos de registros.

En esta estructura el programador da nombre al nuevo tipo de dato y a cada uno de los campos que lo componen, y especifica el tipo de dato que puede ocupar cada uno de los campos. En la figura 4.1 aparece una declaración de este tipo de estructura.

```
Type Nombre = Record
  NombreCampo1 : TipoDato1;
  NombreCampo2 : TipoDato2;
  NombreCampoN : TipoDatoN;
End;
```

Figura 4.1 Estructura para definir un registro.

En la figura 4.1, nombre es el identificador del registro, NombreCampo1, NombreCampo2 y NombreCampoN son los campos que conforman al registro.
TipoDato1, TipoDato2 y TipoDatoN son la especificación del tipo de dato que se va a almacenar en cada uno de los campos. Esta especificación puede pertenecer a un tipo de dato estándar del BP7 o cualquiera que se halla definido en el programa.
Por ejemplo, hay casos en los que puede ser conveniente definir un tipo de variable para almacenar la información de una persona. En la figura 4.2 hay una posible definición.

```
Type Persona = Record
  Nombre : String[15];
  Apellidos : String[30];
  Sexo : String[1];
  Edad: Integer;
  Estatura: Real;
  Peso: Real;
  Ujc: Boolean;
End;
 Var
   Pers : Persona;
```

Figura 4.2 Ejemplo de un registro llamado Persona.

Como se puede apreciar, luego de crear nuestro registro, tenemos que declarar una variable con el tipo que acabamos de establecer.

Para referirnos a un campo de un registro se especifica mediante el nombre de la variable el nombre del campo unidos por un punto. La figura 4.3 muestra cómo podría continuar el ejemplo anterior asignándole datos a los campos del registro.

```
Pers.Nombre:='Rodiert';
Pers.Apellidos:='Zayas Alzamora';
Pers.Sexo:='M';
Pers.Edad:=24;
Pers.Estatura:=1.70;
Pers.Peso:=68.50;
Pers.Ujc:=True;
```

Figura 4.3 Ejemplo para asignar valor a un registro.

Si queremos mostrar los campos Nombre y Apellidos, por citar un ejemplo, solo se hace mediante las siguientes líneas:

```
Writeln('Nombre: ',Pers.Nombre);
Writeln('Apellidos: ',Pers.Apellidos);
```

Además el BP7 cuenta con una estructura llamada With que permite acceder de manera más cómoda a los valores de los campos de los registros. En la figura 4.4 está la sintaxis de la estructura With.

```
With Nombre Do
  begin
   Campo1;
   Campo2;
   CampoN;
  end;
```

Figura 4.4 Sintaxis de la estructura With.

donde nombre es la especificación de la variable (o las variables separadas por comas) del tipo Record y a continuación se hace referencia al nombre de los campos a los que queremos acceder. Por ejemplo, las instrucciones de la figura 4.3, se podrán haber escrito del siguiente modo:

```
With Pers Do
Begin
  Nombre:='Rodiert';
  Apellidos:='Zayas Alzamora';
  Sexo:='M';
  Edad:=24;
  Estatura:=1.70;
  Peso:=68.50;
  Ujc:=True;
End;
```

La elección del uso de la estructura With se suele hacer en cada caso particular según la legibilidad que se requiera en el programa.

Con lo que tenemos hasta ahora, podemos hacer un programa ejemplo donde se aparezcan los datos generales o parciales de un estudiante con su nota final del curso. En la figura 4.5 aparece este programa.

```
Program Ejemplo;
Uses CRT;
Type Fecha = Record   { Registro para la fecha de nacimiento }
mes : 1..12;
dia : 1..31;
ano : Integer ;
End;
Alumno = Record          { Registro Alumno }
nombre : String;
apellidos : String;
nacimiento : Fecha;      { Campo del tipo registro fecha }
nota : Real;
End;
Var Est: Alumno;            { Variable del tipo registro Alumno }
Begin
ClrScr;     { Damos los datos al campo }
Write('Nombre : ') ; Readln(Est.nombre);
Write('Apellidos : '); Readln(Est.apellidos);
Write('Año de nacimiento : '); Readln(Est.nacimiento.ano);
Write('Mes de nacimiento: ');  Readln(Est.nacimiento.mes);
Write('Día de nacimiento: ');   Readln(Est.nacimiento.dia);
Write('Nota final del curso: '); Readln(Est.nota);
Write('Presione ENTER para ver los resultados almacenados');Readln;
Writeln;
Writeln('Nombre : ',Est.nombre);
Writeln('Apellidos : ',Est.apellidos);
Writeln('Fecha de nacimiento : ', Est.nacimiento.dia, ' - ',
                Est.nacimiento.mes, ' - ',Est.nacimiento.ano);
Writeln('Nota final del curso: ', Est.nota:0:2);
Readln;
End.
```

Figura 4.5 Programa ejemplo para el uso de registros.

4.3 Arreglos de registros.

En el ejemplo de la figura 4.5 se puede aplicar para el caso de un solo alumno, pero existe la posibilidad de introducir una base de datos que contuviera información de los alumnos de un grupo, donde lo más lógico es ordenar los alumnos en una lista. Para ello, se puede crear una estructura en forma de arreglo cuyos elementos sean los registros definidos para almacenar la información de los alumnos. Al programa de la figura 4.5 se le puede modificar algunas líneas y quedaría como se ve en la figura 4.6 y ya quedaría para un grupo de veinte alumnos.

```
Program Ejemplo;
Uses CRT;
Type Fecha = Record   { Registro para la fecha de nacimiento }
mes : 1..12;
dia : 1..31;
ano : Integer ;
End; Alumno = Record          { Registro Alumno }
nombre : String;
apellidos : String;
nacimiento : Fecha;           { Campo del tipo registro fecha }
nota : Real;
End;
Var  Listado: Array[1..20] of Alumno;    { Arreglo de tipo registro }
i: Integer;
Begin
ClrScr;      { Damos los datos al campo }
For i:=1 to 20 do
begin
Writeln;
Write('Nombre ',i,': ') ;
Readln(Listado[i].nombre);
Write('Apellidos: ');
Readln(Listado[i].apellidos);
Write('Año de nacimiento: ');
Readln(Listado[i].nacimiento.ano);
Write('Mes de nacimiento: ');
Readln(Listado[i].nacimiento.mes);
Write('Día de nacimiento: ');
Readln(Listado[i].nacimiento.dia);
Write('Nota final del curso: ');
Readln(Listado[i].nota);
ClrScr;
end;
Writeln;
Write('Presione ENTER para ver los resultados almacenados');
Readln;
Writeln;
For i:=1 to 20 do
begin
Writeln(i,' Nombre: ', Listado[i].nombre);
Writeln('Apellidos: ', Listado[i].apellidos);
Writeln('Fecha de nacimiento: ', Listado[i].nacimiento.dia, ' - ',
   Listado[i].nacimiento.mes, ' - ', Listado[i].nacimiento.ano);
Writeln('Nota final del curso: ', Listado[i].nota:0:2);
Writeln;
end;
Readln;
End.
```

Figura 4.6 Arreglo de registros.

Posteriormente de estudiar todos estos tipos de datos definidos por el usuario debe estar mucho más claro el sentido de los conceptos tipo de dato y variable.
Mediante las estructuras de registros y los arreglos el BP7 ofrece la posibilidad de definir tipos de datos muy próximos a los utilizados en el lenguaje natural y que son una organización precisa de tipos de datos más sencillos, ya que se parecen más a la realidad.

4.4 Ficheros.

Un fichero al igual que un arreglo, es otro tipo de dato estructurado. Constituye una colección de datos todos del mismo tipo. La diferencia principal de este y un arreglo, es que en un fichero puede almacenarse en un dispositivo de almacenamiento masivo de información, es decir que nos permite almacenar información permanentemente y acceder a ella cuando sea necesario. Muchas aplicaciones requieren de esta importante estructura.
Los tipos de almacenamientos que vimos hasta ahora, pierden su valor cuando se cierra la aplicación. En general hay dos tipos de ficheros que pueden ser generados: ficheros permanentes y ficheros temporales. Los permanentes son los que se mantienen en un dispositivo de almacenamiento de memoria auxiliar y por tanto se pueden guardar después de completar la ejecución del programa. El contenido de estos ficheros pueden ser recuperados o modificados en cualquier momento, bien por el programa que creó al fichero o por cualquier otro programa que haga referencia a este. Estos ficheros se denominan en BP7 ficheros externos.
Los ficheros temporales, son los que hemos vistos en capítulos anteriores, ya que son almacenados en la memoria principal de la computadora y el valor que se almacena se pierde tan pronto como se termina la ejecución del programa que creó el mismo, por ende, estos ficheros son menos útiles y menos comunes, se denominan ficheros internos.

4.5 Constructor del tipo de dato file.

La definición de un fichero realiza en la zona para declarar una variable, solo hay que indicarle que es de tipo fichero. La forma general de esta declaración está expresada en la figura 4.7.

```
Var
Nombre : File of tipo;
```

Figura 4.7 Estructura general para definir un fichero.

Aquí N*ombre* es un identificador que representa el nombre del fichero y tipo se refiere al tipo de datos de los componentes individuales del un fichero. Estos componentes han de ser todos del mismo tipo: simples o estructurados, pero no pueden ser datos de tipo fichero. Por lo que un fichero no puede estar incluido dentro de otro fichero.

Ejemplos de Definición de ficheros:

-El ejemplo siguiente es una definición de fichero en la que cada componente es un dato de tipo Integer:

```
Var Puntuacion: FILE of Integer;
```

-Definición de fichero en la que cada componente es un arreglo real de una dimensión:

```
TYPE Temp=array [1..100] of real;
Var
 datos: FILE of Temp;
```

-Definición de fichero donde sus componentes son registros:

```
 TYPE Color = (Blanco, Amarillo, Azul, Rozado);
        Habitacion = RECORD
                Costo: Real;
                Largo: Integer;
                Ancho: integer;
                Color_Hab: Color;
  End;
Var Casa :FILE of Habitacion;
```

-Ahora se muestra un ejemplo de definición de fichero en el que dos ficheros distintos tienen los mismos componentes de tipo registro:

```
TYPE estado =(soltero, casado, divorciado);
        Persona=RECORD
                Nombre: ARRAY¨[1..80] of char;
                Sexo: char;
                 Edad: integer;
                Estadocivil: estado;
 End;
 Usuarios = File of persona;
Var usuarioviejo, usuarionuevo: Usuarios;
```

Observar que la longitud del fichero nunca se especifica dentro de la definición de fichero. Como regla general, la longitud máxima de un fichero está determinada sólo por la capacidad física del medio en que se está almacenando.

4.6 Operaciones con ficheros.

Las operaciones que podemos hacer con ficheros son las de escritura, lectura y recorrido. Con el tipo de dato File se pueden direccionar las entradas y salidas de los programas a impresoras, dispositivos auxiliares y también a archivos.

El procedimiento Assign permite asignar a una variable del tipo File la dirección del dispositivo de entrada o salida que el programador pretende utilizar. Este procedimiento tiene dos argumentos. El primero es el nombre de la variable definida del tipo File y el segundo es un dato tipo String que contiene el nombre con el que el sistema operativo identifica el fichero que se quiere utilizar, ver figura 4.8.

```
Assign(Nom_Fichero,Var_String);
```

Figura 4.8 Estructura del procedimiento Assign.

En la figura 4.9 se presenta un programa donde el usuario entra su nombre y este se almacena en un TXT con el nombre uno en el lugar donde está guardado el programa. En el programa Escritura también aparecen dos instrucciones de tipo procedimiento: ReWrite y Close, relacionados con las tareas que se realizan en un ordenador para transferir datos.

```
Program Escritura;
Var
almacen : File of string; { Definimos el fichero }
Texto : String; { Declaramos la variable que queremos almacenar }
Begin
{1} Writeln('Entre su nombre: '); {Pedimos el nombre del usuario }
     Readln(Texto); { Lo almacenamos en Texto}
{2} Assign(almacen,'uno.txt'); { Creamos un TXT con el nombre uno }
{3} Rewrite(almacen); { Preparamos al fichero para la escritura }
{4} Write(almacen,Texto); { Escribimos en el TXT el valor de Texto }
{5} Close(almacen);   { Cerramos el canal de comunicación }
   Readln;
End.
```

Figura 4.9 Programa escritura de datos en un TXT.

ReWrite: Especifica que el dispositivo identificado con almacen se va a utilizar para salida de datos. Esta orden implica la realización de tareas que dependen del tipo de dispositivo al que nos estamos refiriendo. En el caso de archivos se ha de proceder a crear el archivo. Cuando el dispositivo se va a utilizar como entrada de datos, las inicializaciones necesarias se realizan con la instrucción Reset. También se utiliza Reset para salida de datos a archivos que ya existen.

Close: Cierra el canal de comunicación abierto anteriormente con el procedimiento Reset o Rewrite. Siempre que se acaba de transferir los datos del programa a un archivo hay que realizar esta tarea que va dirigida a liberar el archivo del control ejercido por el programa.

Nota: En todo programa que se realiza una entrada o salida de datos a un dispositivo que no es el estándar se han de incluir las ordenes correspondientes a la asignación de dispositivo, inicialización y cierre.

La lectura de los datos se realiza con la instrucción ya conocida por ustedes Read, en el programa de la figura 4.10 se muestra cómo se usa esta instrucción haciendo referencia al TXT que creamos en el programa de la figura 4.9.

Como es de suponer este programa lo que hace es mostrar en pantalla el nombre introducido por el usuario en el programa anterior. En este se utiliza la instrucción Reset ya que el fichero existe y la operación que se va a realizar es de lectura.

```
Program Lectura;
Var
almacen : File of string;
Texto:String;
Begin
 Assign(almacen,'uno.txt');
 Reset(almacen);
 Read(almacen,Texto);
 Close(almacen);
 Writeln(Texto);
 Readln;
End.
```

Figura 4.10 Programa Lectura que utiliza la instrucción Read.

Por ejemplo, en el problema esbozado en la sección anterior sobre una base de datos con información sobre los alumnos, el mantenimiento de dicha base de datos para que sea más eficiente podrá ser del modo que aparece en la figura 4.11. Aquí los datos introducidos no se pierden al cerrar el programa ya que estos son almacenados en fichero de texto llamado Archi.txt.

En este programa se utiliza la función Eof que tiene como argumento la variable del tipo File carpeta. La función Eof es del tipo Boolean que devuelve el valor True si se a alcanzado el fin del fichero al que apunta su argumento. La función se utiliza en el programa Base_Datos para recorrer todo el archivo hasta el final y después añadir el nuevo registro.

```
Program Base_Datos;
Uses CRT;
Type Fecha = Record
 mes: 1 .. 12;
 dia: 1 .. 31;
 ano: Integer;
end;
 Alumno = Record
 nombre : String;
 apellidos : String;
 nacimiento : Fecha;
 nota : Real;
 end;
Var
 alum: array[1..20] of alumno;
 Carpeta: file of alumno;
 I:Integer;
 Listado: alumno;
 nomArchi: String;
Begin
nomArchi:= 'Archi.txt';
Clrscr;
{Continúa}
```

```
for i:=1 to 20 do
  begin
    with Listado  do
      begin
        writeln;
        Write('Nombre ',i,': ') ; Readln(nombre);
        Alum[i].nombre:= nombre;
        Write('Apellidos: '); Readln(apellidos);
        Alum[i]. apellidos:= apellidos;
        with nacimiento do
          begin
            Write('Año de nacimiento: '); Readln(ano);
           Alum[i].nacimiento.ano:= ano;
           Write('Mes de nacimiento: '); Readln(mes);
           Alum[i]. nacimiento.mes:= mes;
           Write('Día de nacimiento: '); Readln(dia);
           Alum[i].nacimiento. dia:= dia;
          end;
       Write('Nota final del curso: '); Readln(nota);
       Alum[i].nota:= nota;
      end;
    Clrscr;
  end;
Assign(carpeta,nomArchi);
rewrite(carpeta);
for i:=1 to 20 do write(carpeta,alum[i]);
close(carpeta);  Writeln;
write('Presione Enter para ver los resultados ');Readln;
Clrscr;
Writeln;
reset(Carpeta);
while not eof(Carpeta) do
  begin
    read(carpeta,listado);
    with listado do
      begin
        writeln ('Nombre y Apellidos: ',nombre,'  ',apellidos);
        with nacimiento do
          begin
            writeln ('Fecha de Nac.: ',dia,' - ',mes,' - ',ano);
          end;
        writeln ('Nota final: ',nota:0:2);
      end;
    writeln;
   end;
 close(carpeta);
 readln;
End.
```

Figura 4.11 Programa para almacenar datos en un fichero y luego mostrarlos en pantalla.

4.7 Resumen.

- Un registro es una estructura de datos en la cual se puede almacenar una colección de datos cuyos tipos pueden ser distintos.
- Estructura para definir un registro.

```
Type Nombre = Record
  NombreCampo1 : TipoDato1;
  NombreCampo2 : TipoDato2;
  NombreCampoN : TipoDatoN;
End;
```

- La definición de un registro se realiza en la zona de declaraciones del programa.
- Para referirnos a un campo de un registro se especifica mediante el nombre de la variable y el nombre del campo unidos por un punto. Ejemplo:

Nombre_Registro.Nombre_Campo

- **With**: Estructura que permite acceder de manera más cómoda a los valores de los campos de los registros.
- Sintaxis de la estructura With.

```
With Nombre Do
 begin
  Campo1;
  Campo2;
  CampoN;
 end;
```

- Los arreglos de registros son estructuras cuyos elementos del arreglo estén formados por registros definidos para almacenar información.
- Para crear un arreglo de registros primero se define el registro y luego se hace el arreglo y se especifica que el tipo de dato es el registro definido. Ejemplo:

```
Type Fecha = Record
  mes : 1..12;
  dia : 1..31;
  ano : Integer ;
End;
Var  Nacimiento: Array[1..20] of Fecha;    { Arreglo de tipo registro }
```

- **Fichero**: Tipo de dato estructurado parecido a un arreglo. Es una colección de datos todos del mismo tipo que permite almacenar información permanentemente y acceder a ella cuando sea necesario.
- **File**: Estructura de datos que nos permite guardar un fichero de datos en disco para no perderlos después de ejecutar el programa.
- **Ficheros permanentes:** El contenido de estos ficheros pueden ser recuperados o modificados en cualquier momento, bien por el programa que creó al fichero o por cualquier otro programa que haga referencia a este. Estos ficheros se denominan ficheros externos.

- **Ficheros temporales:** Son los datos que son almacenados en la memoria principal de la computadora y el valor que se almacena se pierde tan pronto como se termina la ejecución del programa que creó el mismo, por ende, estos ficheros son menos útiles aunque no dejan de ser aprovechados, se denominan ficheros internos.
- Estructura general para definir un fichero.

```
Var
Nombre : File of tipo;
```

- **Assign:** Procedimiento que permite asignar a una variable del tipo File la dirección del dispositivo de entrada o salida.
- Estructura del procedimiento Assign.

```
Assign(Nom_Fichero,Var_String);
```

- **ReWrite**: Especifica que el dispositivo identificado como parámetro se va a utilizar para salida de datos. En el caso de archivos se ha de proceder a crear el archivo. Este prepara al archivo para la escritura.
- **Reset**: Instrucción que se utiliza cuando se va a realizar un entrada de datos. También se utiliza Reset para salida de datos a archivos que ya existen.
- **Close**: Cierra el canal de comunicación abierto con el procedimiento Reset o Rewrite. Siempre que se acaba de transferir los datos del programa a un archivo hay que realizar esta tarea que va dirigida a liberar el archivo del control ejercido por el programa.
- **Eof:** Función del tipo Boolean que devuelve el valor True si se a alcanzado el fin del fichero al que apunta su argumento.

4.8 Ejercicios de autoevaluación.

1.1 Defina un registro para cada una de las situaciones siguientes:
   a) Almacenar los resultados de los tres trabajos de control de la asignatura de matemática.
   b) Almacenar la cantidad de Mango, Toronjas, Naranjas y Piñas que se guarda en un almacén de frutas.
   c) Los datos de un persona ingresada en un hospital cuyos datos son: Nombre, Apellidos, Sexo, Edad, Sala y Número de Cama.

1.2 Declare una variable para cada uno de los registros definidos en el ejercicio anterior.

1.3 Partiendo de las declaraciones hechas por usted en del ejercicio 1.2 realice una asignación con un valor válido para el campo Naranjas del inciso b y para el campo Sala del inciso c del ejercicio 1.1.

1.4 Utilizando la estructura With realice una asignación a cada campo del inciso c del ejercicio 1.1.

1.5 Defina un arreglo cuyos 10 elementos estén conformado por un registro llamado libros con los campos Título y Autor.

1.6 Complete las siguientes oraciones relacionado con los ficheros:
   a) Se llama _____ a una colección de datos todos del mismo tipo que permite almacenar información permanentemente y acceder a ella cuando sea necesario.
   b) Hay dos tipos de ficheros que pueden ser generados, estos son los ficheros _____ y ficheros _____.

c) Los ficheros _____ almacenan los datos en la memoria principal de la computadora y el valor almacenado se pierde tan pronto como se termina la ejecución del programa. También se conocen como ficheros _____.

d) El contenido de los ficheros _____pueden ser recuperados o modificados en cualquier momento. Estos ficheros también se conocen como ficheros _____.

1.7 Enlace según corresponda la columna A con la B:

Columna A	Columna B
1.- **Assign**	___ Especifica que el dispositivo identificado como parámetro se va a utilizar para salida de datos.
2.- **ReWrite**	___ Cierra el canal de comunicación abierto con el procedimiento Reset o Rewrite.
3.- **Reset**	___ Procedimiento que permite asignar a una variable del tipo File la dirección del dispositivo de entrada o salida.
4.- **Close**	___ Función del tipo Boolean que devuelve el valor True si se a alcanzado el fin del fichero al que apunta su argumento.
5.- **Eof**	___ Instrucción que se utiliza cuando se va a realizar un entrada de datos.

1.8 Organice el programa siguiente teniendo en cuenta que tiene como objetivo el de almacenar en un TXT el texto REVOLUCIÓN:

```
Program Escritura;
DirAlm : File of String;
Texto: String;
Var
Begin
   Write(DirAlm,Texto);
  Texto:= ’REVOLUCIÓN’;
  Close(DirAlm);
  Rewrite(DirAlm);
  Readln;
  Assign(DirAlm,'Dato.txt');
End.
```

1.9 Organice el programa siguiente teniendo en cuenta que tiene como objetivo leer dato de un TXT ya existente y mostrarlo en pantalla:

```
Program Lectura;
 Contenedor: File of String;
 Var
 T:String;
Begin
  Writeln('El texto encontrado es: ',T);
  Reset(Contenedor);
  Readln;
  Close(Contenedor);
  Assign(Contenedor,'Dato.txt');
  Read(Contenedor,T);
End.
```

4.9 Respuestas de ejercicios de autoevaluación.

1.1

a)	b)	c)
Type Mat = Record PrimTCont : Real; SengTCont : Real; TercTCont : Real; **End**;	**Type** Frutas = Record Mango: Integer; Toronjas: Integer; Naranjas: Integer; Piñas: Integer; **End**;	**Type** Paciente = Record Nombre: String; Apellidos: String; Sexo: String[1]; Edad: Integer; Sala: String; Cama: Integer; **End**;

1.2

Inciso a	Inciso b	Inciso c
Var Asig: Mat;	Var Frut: Frutas;	Var Ingreso: Paciente;

1.3

Inciso a: Frut.Naranjas:=77;

Inciso b: Ingreso.Sala:= '3B';

1.4

```
With Paciente do
   begin
      Nombre:='Reinier';
      Apellidos:= 'Escobar Santiesteban';
      Sexo:= 'M';
      Edad:= 24;
      Sala:= '3B';
      Cama:= 5;
   end;
```

1.5

```
Type Libros record
   Titulo: String;
   Autor: String;
 end;
 Var CantLib: Array[] of Libros;
```

1.6 a) Fichero. b) Permanentes, Temporales. c) Temporales, Internos.
d) Permanentes, Externos.

1.7 Columna B: 2,4, 1, 5 y 3.

1.8

```
Program Escritura;
Var
 DirAlm : File of String;
Texto: String;
Begin
  Texto:= ’REVOLUCIÓN’;
  Assign(DirAlm,'Dato.txt');
  Rewrite(DirAlm);
  Write(DirAlm,Texto);
  Close(DirAlm);
  Readln;
End.
```

1.9

```
Program Lectura;
Var
 Contenedor: File of String;
 T:String;
Begin
 Assign(Contenedor,'Dato.txt');
 Reset(Contenedor);
 Read(Contenedor,T);
 Close(Contenedor);
 Writeln(‘El texto encontrado es: ’,T);
 Readln;
End.
```

4.10 **Ejercicios propuestos.**

1.10 Definir un dato Alumno de tipo registro que contenga: nombre, apellidos, edad y su promedio general.

1.11 Realice un programa que permita entrar los siguientes datos de un grupo n de alumnos: nombre, apellidos e índice académico. Seguidamente de entrados los datos se desea obtener un listado ordenado por índice académico de forma descendente.

1.12 Realice un programa que permita entrar los siguientes datos de un grupo 15 de alumnos: nombre, apellidos y las notas de los cuatro trabajos prácticos en la asignatura de LTP. El programa pide los datos necesarios y a continuación muestra un listado de los estudiantes que tienen al menos un trabajo práctico suspenso.

1.13 Elabore un programa que permita entrar los datos de 20 de trabajadores: nombre, apellidos, salario básico y la cantidad de ausencias en el mes.

- Mostrar cobro del mes trabajado con relación a la cantidad de ausencias en el mes (24 días laborables).
- Obtenga el total de dinero a pagar por la empresa.
- Obtenga el promedio del total a pagar con respecto a los trabajadores.
- Obtenga el porcentaje de trabajadores cuyo salario supera el promedio.

1.14 Se desea guardar en memoria un listado que contenga la información de los nueve bateadores regulares un equipo de béisbol. Los datos que se necesitan son: nombre, apellidos, número de uniforme, veces al bate y cantidad de hit conectados. Después de entrado estos datos, se desea obtener una sublista ordenada en forma descendente por el average de cada pelotero.

Fórmula para calcular el average:

Ave = Cantidad de hit / veces al bate * 1000.

1.15 Realice un programa que guarde en memoria un listado que contenga la siguiente información de los 16 equipos de béisbol cubano. Los datos que se necesitan son: nombre del equipo, juegos jugados y cantidad de juegos ganados. Después de entrado estos datos, se desea obtener una sublista ordenada en forma descendente por el average de cada equipo.

Fórmula para calcular el average:

Ave = Juegos ganados / Juegos jugados * 1000.

1.16 Confeccione un programa que almacene en un arreglo de registro los datos personales de los 10 trabajadores de un departamento como son: nombre, primer apellido, segundo apellido, edad, carne de identidad y sexo. Agregue la posibilidad de que el usuario pueda ver o no los datos almacenados.

1.17 Elabore un programa que permita entrar los siguientes datos de un grupo de alumnos y guardarlos en un fichero.

Nombre, apellidos, edad y sexo.

1.18 En la fábrica de refresco de la Isla de la Juventud, se lleva un control estricto de cada uno de sus trabajadores, se conoce el nombre completo, el número de identidad, el sexo, el teléfono en caso que lo posean y además si son del PCC, del la UJC o no pertenecen a ninguna de las dos.

a. Almacene los anteriores datos en un fichero.
b. Agregue la posibilidad de que el programa lea los datos del fichero y devuelva los datos de un trabajador dado identificado.
c. Realice un grupo de instrucciones que lea los datos del fichero y muestre en pantalla los trabajadores que pertenecen a la UJC.
d. Realice un grupo de instrucciones que lea los datos del fichero y devuelva la cantidad de trabajadores que no tienen teléfono particular.

1.19 Realice un programa donde el programa almacene en un fichero una contraseña. Al ejecutarse el programa se pide que se introduzca la contraseña para poder entrar a trabajar. Se dan cinco oportunidades para poner la contraseña correcta y si no se la sabe en estos intentos el programa se cerrará. Si se sabe la contraseña y logra entrar, el usuario tendrá la posibilidad de modificar la contraseña y esa nueva es la que quedará almacenada en el fichero.

Capítulo 5

Modularidad

Objetivos

- Caracterizar el principio de la modularidad.
- Definir e implementar módulos, a través de procedimientos con vista a desarrollar buenos hábitos de la programación.
- Pasar parámetros por valor y por referencia.
- Definir e Implementar funciones, importancia de su utilización.
- Comprender cómo escribir y utilizar funciones que se llamen a sí mismas (Recursividad).

Pensamientos

◘ La precisión en el objetivo es el punto de partida para cualquier triunfo.

Clement Stone.

◘ La perfección no existe. Pero si la buscamos, alcanzaríamos la excelencia.

Vince Lombarda.

◘ Nunca se debe perder tiempo en vanos lamentos por el pasado ni en quejas por los cambios que nos incomodan, pues el cambio es la esencia de la vida.

Anatole France.

◘ La juventud es el tiempo de estudiar la sabiduría, así como la vejez es el tiempo de practicarla.

J. J. Rousseau.

◘ Así que, en lugar de pensar en alcanzar el cielo algún día, sigo mi camino, día a día.

Emily Dickinson.

Sinopsis

5.1 Principios de modularidad. Ventajas.
5.2 Procedimientos. Estructura de un procedimiento.
5.3 Paso de parámetro por valor y por referencia.
5.4 Ejemplos de procedimientos creados por el programador.
5.5 Funciones. Estructura de una función.
5.6 Ejemplos de funciones creadas por el programador.
5.7 Recursividad. Funciones recursivas.
5.8 Resumen.
5.9 Ejercicios de autoevaluación.
5.11 Respuestas de ejercicios de autoevaluación.
5.12 Ejercicios propuestos.

5.1 Principios de modularidad. Ventajas.

Con las estructuras vistas hasta ahora del BP7 se pueden construir programas que realizan tareas muy complejas pero con escasa legibilidad por su tamaño en cuanto a líneas de código, independientemente de los comentarios que se colocan en algunas líneas para hacer más claro el programa. En la informática, al igual que en otras ciencias, es muy común, y además muy efectivo, resolver problemas mediante la modularidad.
Este proceso consiste en tomar un problema convertirlo en una serie de problemas más sencillos y resolver después cada uno de ellos. A esta estrategia se lo conoce con el nombre "Divide y Vencerás".
Desde el punto de vista de la legibilidad es deseable que un programa de ordenador pueda estructurarse siguiendo lo más posible el esquema conceptual que se ha pensado utilizar para resolver el problema.

BP7 brinda dos herramientas básicas para realizar programación descendente o Top - Down: procedimientos (Procedure) y funciones (Function). Generalmente se les conoce a estos como subprogramas. BP7 agrega además el concepto de unidad (Unit), que permite aprovechar módulos independientes ya compilados donde aparecen un conjunto de subprogramas que le facilitan al programador el no tener que hacer subprogramas que ya existen.

Un subprograma es un conjunto de sentencias de un programa que realizan una determinada tarea y que pueden ser ejecutadas desde más de un punto del programa principal, también se le conoce con el nombre de subrutina.

En este capítulo te dotarás de los conocimientos necesarios para la construcción de procedimientos y funciones, estructuras básicas en la programación modular.

5.2 Procedimientos. Estructura de un procedimiento.

En algunas ocasiones usamos procedimientos que ya vienen con el lenguaje, en la figura 5.1 se muestran algunos procedimientos predefinidos en BP7. Nosotros también podemos crear los nuestros. Un procedimiento *es una sección de código separada del programa principal que realiza una acción y no devuelve valor alguno.*

ClrEol	ClrScr	Sound	TextColor
Delay	GotoXY	NoSound	TextBackground

Figura 5.1 Ejemplos de procedimientos predefinidos.

La sintaxis con la que se especifican estos procedimientos es similar a la estructura de un programa en BP7 pero se usa la palabra reservada Procedure para indicar que se trata de un procedimiento parcial, en la figura 5.2 aparece la estructura general para definir un procedimiento.
Recuerden que para crear un procedimientos, lo debemos hacer en la zona de declaraciones.

	Partes
① ② ③ PROCEDURE Nombre ( ) ; ④ — BEGIN ⑤ — Acciones; ⑥ — END ;	1.- Palabra clave del procedimiento. 2.- Nombre del procedimiento. 3.- Lista de parámetros. 4.- Inicio del procedimiento. 5.- Cuerpo del procedimiento. 6.- Fin del procedimiento.

Figura 5.2 Partes de un procedimiento para su declaración.

A continuación se vamos a crear un procedimiento que imprima en pantalla una “**Bienvenido a la Modularidad**”.

```
PROCEDURE Titulo;
BEGIN
WriteLn (‘**Bienvenido a la Modularidad**’);
END;
```

Ya cuando se a declarado el procedimiento es posible emplearlo como una instrucción de BP7. Al uso de un procedimiento se le conoce como llamada al procedimiento. El programa de la figura 5.3 ilustra el uso o la llamada al procedimiento creado anteriormente

```
Program Procedimientos;
Uses Crt;
Procedure titulo; {Creación del procedimiento}
begin
Writeln (‘**BIENVENIDO A LA MODULARIDAD**’);
end;
Begin
 Writeln ('Programa ejemplo del uso de procedimientos');
 Writeln;
 Titulo;  {Llama al procedimiento}
 Readln;
End.
```

Figura 5.3 Creación y llamada del procedimiento Titulo.

El problema que les mostraremos a continuación será resuelto mediante la programación modular con el objetivo de que vean la organización que muestra esta forma de programar.

Problema:

Dado los nombres y las notas de un grupo de N estudiantes, confecciones un programa que muestre en pantalla los nombres de los estudiantes que tengan más de 90 puntos.

```
Program Notas;
Uses CRT;
Var
 Nomb: array[1..100] of String;
 Nota:array[1..100] of Real;
 N, j, i: Integer;
Procedure Entrada;  {Procedimiento para entrada de datos}
begin
 CLRSCR;
 Write('Entre la cantidad de estudiantes del grupo: '); Readln(N);
  For i:= 1 to N do
    begin
       Write('Nombre ',i,': '); Readln(Nomb[i]);
       Write('Nota de ',Nomb[i],': '); Readln(Nota[i]);
    end;
 CLRSCR;
end;
Procedure Salida;
begin
 Writeln('Presione ENTER para ver los resultados.'); Readln;
 CLRSCR;
 Writeln;
 Write('Buscando notas mayores que 90');
  For i:=1 to 5 do
    begin
      Write(' . ');
     Delay(1000);
   end;
 CLRSCR;
 Writeln('Resultados');
 Writeln('Estudiantes con más de 90 puntos:');
 j:=1;
 For I:=1 to N do
   begin
     if nota[i] >90 Then
       begin
         Write(j,'.- ',Nomb[i]);
        Gotoxy(15,j+2);Write('- - - ',Nota[i]:0:2);
        INC(j);
        Writeln;
      end;
   end;
 Readln;
end;
Begin
 Entrada;  }  {  Parte del programa donde se realizan }
 Salida;   }  {   las llamadas a los procedimientos. }
End.
```

5.3 Paso de parámetro por valor y por referencia.

Como se ha explicado hasta ahora, uno de los objetivos fundamentales de la programación modular es el de no repetir líneas de código, o mejor dicho, códigos que realicen la misma operación aunque con diferentes variable.
BP7 brinda la solución de estos problemas mediante la fabricación de procedimientos utilizando la lista de parámetros. Un parámetro es una variable que puede ser un número, una fecha, un nombre de archivo, o simplemente texto; en fin, cualquier cosa que personalice el funcionamiento de un programa.

Los parámetro puede ser pasados por valor o por referencia en esta sección se resolverán problemas donde se ejemplifique el uso de esto tipos de parámetros.

En el programa de la figura 5.4 se realiza un procedimiento que calcula el área de un círculo, donde se pasa como parámetros Radio y Area. El primero es un parámetro por valor ya que este es utilizado solo para el cálculo pero su valor no cambia durante el proceso, mientras que el segundo es un parámetro por referencia (o dirección como también se le conoce) porque el contenido cambia.

```
Program area_circ;
uses crt;
var radiocirc, resultado: real;
Procedure areacirculo (radio:real;var area: real);
const pi = 3.14;
begin
area := pi * sqr(radio);
end;
Begin
clrscr;
writeln;
write('introduzca el radio del círculo: '); readln(radiocirc);
writeln;
if (radiocirc > 0) then
begin
areacirculo(radiocirc, resultado);
{radiocirc se corresponde con radio y resultado con area}
gotoxy(20,5);
writeln('el área del círculo es: ',resultado:8:2);
end
else
write('no puede introducir un radio negativo.');
readln;
End.
```

Figura 5.4 Ejemplo de paso por valor y por referencia.

Como pueden apreciar cuando el parámetro que se pasa al procedimiento no se indica con la palabra reservada Var este pasa por valor. Todo lo contrario ocurre cuando se quiere pasar un parámetro por referencia, este debe ser inicializado con la palabra reservada var, como aparece en la figura 5.4.

En el programa de la figura 5.5 aparece un ejemplo donde los dos parámetros que se pasan en el procedimiento Intercambio son por referencia.

```
Program Referencia;
Uses CRT;
Var X, Y: Integer;
Procedure Intercambio(Var A, B: Integer);
Var Z: Integer;
begin
Z:= A;
A:= B;
B:= Z;
end;
Procedure Entrada;
begin
ClrScr;
Writeln('Entre el valor de X: ');Readln(X);
Writeln('Entre el valor de Y: ');Readln(Y);
end;
Procedure Salida;
begin
Writeln;
Writeln('Se intercambiaron los valores de X y Y.');
Writeln('Valor de X: ',X);
Writeln('Valor de Y: ',Y);
Readln;
end;
Begin
Entrada;
Intercambio(X, Y);
Salida;
End.
```

Figura 5.5 Procedimiento Intercambio(Parámetros por referencia).

Este procedimiento que acabamos de ver (Intercambio) se puede aplicar cuando queremos organizar una lista de números como la que vimos en capítulos anteriores donde se mostraba un escalafón de un grupo de alumnos. La variable Z de este procedimiento es local y por lo tanto solo se puede utilizar en este procedimiento. Las variables X y Y son globales, estas si se pueden utilizar en cualquier parte del programa.

Nota: La lista de parámetros que se le pasan a un procedimiento deben corresponder con el tipo de dato con el se definió el parámetro en su declaración, en otras palabras, si el parámetro se definió como un entero, el

dato permisible es un número entero, de otra forma generará un error de ejecución.

5.4 Ejemplos de procedimientos creados por el programador.

Los ejemplos que aparecen a continuación deben ser llevados a un programa, ya que aquí solo se plantea la situación problémica y la posible solución mediante procedimiento. Esto les permitirá ejercitar el uso de los procedimiento en un programa.

Ejemplo 1:

Procedimiento que genere dos números aleatorios entre 1 y P para ser sumados, donde P es un parámetro por valor. N1 y N2 son parámetros por referencia. El procedimiento hace que N1 y N2 nunca sean iguales utilizando un bucle.

```
Procedure Generar(P: Integer; Var N1, N2: Integer);
begin
 Randomize;
 N1:= Random(P)+1;
 Repeat
 N2:= Random(P)+1;
 Until N1 <> N2;
 Writeln(N1,' + ',N2);
end;
```

Ejemplo 2:

Procedimiento que obtiene los factores primos de un número positivo.

```
Procedure Factores(n:integer);
  var
    i,j:integer;
    fin:boolean;
  begin
    Writeln('Los factores primos de ',n,' son: ');
    Writeln;
    For i:=n downto 1 do
      begin
        if (n mod i)=0 then
          begin
            fin:=false;
            j:=i-1;
            while (i>1) and (not fin) do
              begin
                if (i mod j)=0 then fin:=true;
                if j=1 then write(i:2);
                j:=j-1;
              end;
          end;
      end;
  end;
```

Ejemplo 3:
Procedimiento que realiza la conversión de moneda de dólares de Norteamericanos a pesos convertibles Cubanos y pida muestre la fecha para saber el día que se realizó el cambio. Recuerden que por cada dólar Norteamericanos se descuenta un 18 %.

```
Procedure convertir(Dolares: Integer);
Var
 PesosC: Real;
 D, M, A, S: Word;
begin
 PesosC:= Dolares - 0.18 * Dolares;
 Writeln( 'Cantidad de Pesos Convertibles: ',PesosC:4:2 );
 GetDate(A,M,D,S);
 Writeln('Fecha de Cambio: ',D,'-',M,'-',A);
end;
```

Aquí se utiliza la función GETDATE, esta devuelve el año, mes, día y el número de día de la semana, donde el lunes(1) y así hasta el domingo(7). Estos datos los almacena en las variables declaradas con ese objetivo, tienen que ser de tipo Word. En este ejemplo solo se muestra el día, el mes y el año porque es lo que se pide. La función GETTIME devuelve el tiempo en hora, minutos, segundos y centésimas. Pruebe esta última función, puede que le haga falta en algún programa.
Nota:
Para usar estas funciones debe ser declarada la unidad DOS en la lista Uses.

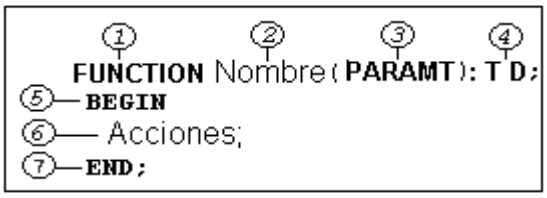
5.5 Funciones. Estructura de una función.

En el transcurso de este libro, has utilizado funciones predefinidas por el lenguaje, ejemplo de estas son las recientes GETTIME y GETDATE, otras funciones aparecen expuestas en la figura 5.6. En este epígrafe vamos a crear nuestras propias funciones, que es una de las herramienta de programación más utilizadas por los programadores. Una función no son más que una sección de código separada del programa principal que realiza una acción y devuelve un valor.

ORD	RNADOM	RAUND	SQR
LENGTH	COPY	POS	COS
SQRT	SIN	UPCASE	TRUNC

Figura 5.6 Ejemplos de funciones predefinidas del BP7.

Siempre que se va a definir una función, lo debemos hacer en la zona de declaraciones del programa. En la figura 5.7 está expresada la estructura para definir una función.



1- Palabra clave de la función.
2- Nombre de la función.
3- Lista de parámetros.
4- Tipo de devolución.
5- Inicio de la función.
6- Cuerpo de la función.
7- Fin de la función.

Figura 5.7 Estructura de una función para su declaración.

Como se puede apreciar una función es muy parecida a un procedimiento, la diferencia radica en que una función tiene que estar concebida para que siempre devuelva un resultado a partir del parámetro que se le pasa.
Vamos a ver cómo quedaría un función que sume dos números cualquiera. En la figura 5.8 se muestra la función Suma, donde se le pasa como parámetros dos números enteros y devuelve un entero.

```
Function Suma(Numero1,Numero2: Integer):Integer;
 begin
  Suma := Numero1 + Numero2;
 end;
```

Figura 5.8 Ejemplo de función que devuelve la suma de dos números.

Aquí se le asigna a la función el valor de la suma de los números entrados como parámetros. En algunas versiones de Pascal esta asignación es posible hacerla mediante la palabra Result, aquí no es posible.

Para invocar la función se procede de la misma forma que para un procedimiento. Se escribe el nombre de la función a la que se quiere invocar y entre paréntesis se colocan los parámetros que exige esta función. El programa de la figura 5.9 muestra la continuación de la figura 5.8.

```
Program Funcion_Suma;
Uses Crt;
Function Suma(Numero1,Numero2: Integer):Integer;
begin
Suma := Numero1 + Numero2;
end;
Var N1,N2:Integer;
Begin
ClrScr;
Writeln;
Write('Entre un número: ');Readln(N1);
Write('Entre otro número: ');Readln(N2);
Write('La Suma es: ',Suma(N1,N2));
readln;
End.
```

Figura 5.9 Programa que suma dos números mediante un función.

5.6 Ejemplos de funciones creadas por el programador.

Ejemplo 1:

BP7 cuenta con la función UPCASE que devuelve al carácter pasado como parámetro en mayúscula, pero y si queremos convertir toda una cadena en mayúscula. ¿Qué función nos permite hacer esto? La respuesta es ninguna. Pero con UPCASE podemos convertir toda la cadena, pasando carácter por carácter de la cadena e ir convirtiéndolo en mayúscula. Esto lo podemos hacer con un ciclo que valla desde el primer carácter hasta el último. Mediante la función LENGTH podemos saber cuál es la longitud de la cadena. Con todo esto podemos hacer un función que convierta una cadena de N caracteres.
Solución:

```
Program Funcion_Mayusc;
Uses Crt;
Function Mayuscula(T: string): string;
 var i: integer;
begin
 for i:= 1 to Length(T) do { Recorre la cadena hasta el final}
  T[I] := Upcase(T[I]);       { Va convirtiendo el caracter (i) a mayúscula}
 Mayuscula := T;            { devuelve la palabra (T) en mayúscula}
end;
var Texto:string;
Begin
 ClrScr;
 Write('Entre una palabra: '); Readln(Texto);
 Writeln('Conversión a mayúscula');
 Writeln;
 Writeln(Mayuscula(Texto));
 Readln;
End.
```

Ejemplo 2:

El programa que verán a continuación lee tres números enteros e imprime si están o no en orden numérico ascendente. En este programa se realizó una función que devuelve TRUE si dos número se ha introducido en orden ascendente y FALSE en caso contrario. Claro que si el primero es menor o igual que el segundo y el segundo es menor o igual que el tercero esto tres números tienen un orden ascendente, pues eso fue lo que se hizo aquí a la hora de llamar a la función.

Solución:

```
Program Funcion_Ordenados;
Uses Crt;
Var n1, n2, n3: integer;
Function Ordenados(i, j: Integer): Boolean;
begin
ordenados := (i <= j)
end;
Begin
ClrScr;
Writeln('*** Introduce tres números enteros ***');
Write('Primero: ');  Readln(n1);
Write('Segundo: ');Readln(n2);
Write('Tercero: ');  Readln(n3);
Writeln;
If Ordenados(n1,n2) and Ordenados(n2,n3) Then
 Writeln('Los tres números están en orden.')
Else
 Writeln('Los tres números NO están en orden');
 Readln;
End.
```

Ejemplo 3:

Muchas veces hemos escuchado decir entre los programadores: “El usuario es el ser más entrometido que existe”. Esta afirmación se debe a que en ciertos momentos se piden datos en el programa que tienen que ser numéricos, por citar un ejemplo, y el usuario entra un texto, esto produce un error fatal. El programador tiene que concebir su programa para validar esto posibles errores que en muchas ocasiones son escritos de manera inconsciente por el usuario. El programa que se muestra a continuación cuenta con una función que verifica si un caracter introducido es un número o no. Aquí se utiliza un el tipo de dato CHAR que almacena un caracter cualquiera. ReadKey realiza una lectura del caracter que se tecleó y lo almacena en la variable que se asigna, en este caso Carac.

Solución:

```
Program escaracter;
 Uses crt;
 Var carac: Char;
Function verificar (caracter: char) : Boolean;
begin
 verificar := (caracter >= '0') and (caracter <= '9');
end;
Begin
 Clrscr;
 Writeln('Introduce un caracter para ver si es numérico.');
 carac:= Readkey;  { Almaceno en carac el caracter entrado }
 Writeln(carac);      { Muestro el caracter entrado }
 If verificar (carac) then
   Writeln('El caracter introducido es numérico.')
 Else
   Writeln('El caracter introducido no es numérico.');
 Readln;
End.
```

Ejemplo 4:

Este ejemplo verifica si una palabra es palíndromo o no a través de una función. Un palíndromo es una palabra que se lee igual adelante que hacia atrás. Por ejemplo las palabras ala, ana, reinier, sayas, otto son palíndromo.

La función Pali devuelve True si la palabra pasada como parámetro es palíndromo y False en caso contrario. La función Length devuelve la cantidad de caracteres de la cadena pasada como parámetro.

Mediante el bucle For garantizamos invertir la palabra almacenada en la variable Texto y almacenar la palabra invertida en TextInv. Esta operación se realiza mediante un conteo descendente caracter por carácter.

Solución:

```
Program Palindromo;
Uses CRT;
Function Pali(Texto:String):Boolean;
var
 i: Integer;
 TextInv:string;
begin
 TextInv:='';
 For i:=Length(texto) downto 1 do
   TextInv:=TextInv + Texto[i];
 If Texto = TextInv Then
   Pali:= True
 Else
   Pali:= False;
end;
var T:string;
Begin
 ClrScr;
 Writeln('Entre una palabra para ver si es palíndromo o no.');
 Readln(T);
 If Pali(T) Then
  Writeln(T,' es palíndromo.')
 Else
  Writeln(T,' no es palíndromo.');
 Readln;
End.
```

5.7 Recursividad. Funciones recursivas.

La recursión es una herramienta de programación que suele ser una de las más difícil de entender por aquellos que son principiantes y la que más dolor de cabeza provoca a los estudiantes. Esto no es para que cojan miedo, sino para que se ajusten en sus asientos y pongan todas sus neuronas a funcionar en esta sección.

Una función recursiva es aquella que se llama así misma en el cuerpo de la función. Así planteada esta definición puede que sea un poco engorroso de entender, para esto vamos a apoyarnos en la solución de un problema donde sea necesario el uso de este tipo de función para que sea más eficiente su solución.

Veamos el siguiente problema:

Determine el factorial de un número entero positivo entrado por el teclado.

Solución:

El factorial de 4 lo podemos expresar de la siguiente forma:

4! = 1 * 2 * 3 * 4 **= 24.**

El de 5: 5! = 1 * 2 * 3 * 4 * 5 **= 120.**

De esto se puede decir en otras palabras que el factorial de N es la multiplicación sucesiva de todos los números desde 1 hasta N. Hay que tener en cuenta que el factorial de 0 es igual a 1.

En esta solución nos podemos dar cuenta que el N! es igual a N * (N - 1)!, si no entiendes lo que digo mira el ejemplo de la figura 5.10.

```
0! = 1
1! = 1
2! = 1 * 2
3! = 1 * 2 * 3
4! = 1 * 2 * 3 * 4
5! = 1 * 2 * 3 * 4 * 5
6! = 1 * 2 * 3 * 4 * 5 * 6
```

Figura 5.10 Explicación del cálculo del factorial.

Veamos cómo puede utilizarse la recursividad para evaluar el factorial de 5. Según vimos anteriormente 5! es igual a 5 * 4! Por tanto antes de poder evaluar 5!, debemos evaluar 4! Utilizando la definición, encontramos que 4! = 4 * 3! Y así repitiendo este proceso hasta llegar al caso más sencillo que es 0! = 1. Partiendo de esto tenemos que

```
(1) 5! = 5 * 4!
(2)          4! = 4 * 3!
(3)                   3! = 3 * 2!
(4)                            2! = 2 * 1!
(5)                                     1! = 1 * 0!
(6)                                              0! = 1
```

Este proceso es resuelto de atrás hacia delante, es decir comenzando por la línea (6) donde el valor es directo. Aquí se da en cada línea un resultado, el factorial del número en cuestión y este se multiplica por el otro y así hasta tener un resultado único.

Ahora cómo podemos expresar en una fórmula esto que vimos anteriormente. La función que aparece en la figura 5.11 es una función recursiva y calcula el factorial de N, teniendo en cuenta que N es un número entero positivo.

```
Function Factorial(N: Integer):Integer;
begin
  If N = 0 Then
    Factorial:= 1
  Else
    Factorial := N * Factorial(N - 1);
end;
```

Figura 5.11 Función factorial.

Como se puede apreciar, en esta función se hace un llamado a la propia función dentro del cuerpo de la misma, a esto es a lo que se le llama recursividad. Observe el rastreo de esta función en la figura 5.12 donde se pasa como parámetro el número 5.
En el rastreo se puede ver que va cambiando el parámetro de entrada en cada llamada a la función. El resultado final es 120 que es el factorial de 5.

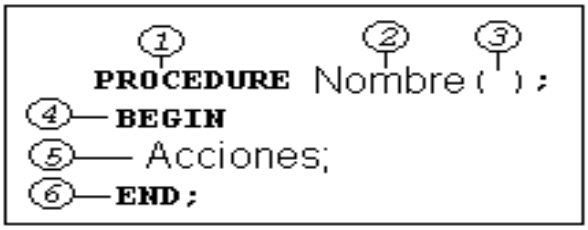
```
N = 5
Factorial =5*Factorial(4)
            N = 4
            Factorial=4 *Factorial(3)
                        N = 3
                        Factorial=3*Factorial(2)
                                    N = 2
                                    Factorial=2*Factorial(1)
                                                N = 1
                                                Factorial=1*Factorial(0)
                                                            N = 0
                                                            Factorial=1
                                                1
                                    2
                        6
            24
   120
```

Figura 5.12 Rastreo de la función factorial con parámetro 5.

Nota: Hay que tener cuidado cuando intentamos hacer una función recursiva que no tenga un valor final que ponga fin a las llamadas de la función, esto genera un programa que una ves que se llama la función esta no devuelve valor alguno y se bloquea el programa.

5.8 Resumen.

- **Procedimiento**: Sección de código separada del programa principal que realiza una acción y no devuelve valor alguno.
- Partes de un procedimiento para su declaración.



Partes
1.- Palabra clave del procedimiento. 2.- Nombre del procedimiento. 3.- Lista de parámetros. 4.- Inicio del procedimiento. 5.- Cuerpo del procedimiento. 6.- Fin del procedimiento.

- La llamada a un procedimiento se hace a través de su nombre.
- **Parámetro**: Variable que puede ser un número, una fecha, un nombre de archivo, o simplemente texto; en fin, cualquier cosa que personalice el funcionamiento de un programa.
- **Parámetro por valor**: El valor de la variable es utilizado solo para el cálculo, su valor no cambia durante el proceso.
- **Parámetro por referencia o dirección**: A esta variable se le asigna un valor durante el proceso de ejecución del procedimiento.
- Recuerde que a la hora de declarar un parámetro por referencia debe declararlo con la palabra reservada VAR.

- La lista de parámetros que se le pasan a un procedimiento deben corresponder con el tipo de dato con el se definió el parámetro en su declaración, de otra forma generará un error de ejecución.
- **Función:** Sección de código separada del programa principal que realiza una acción y devuelve un valor único.
- Estructura de una función para su declaración.

① ② ③ ④ FUNCTION Nombre(PARAMT): T D; ⑤— BEGIN ⑥— Acciones; ⑦— END;	1- Palabra clave de la función. 2- Nombre de la función. 3- Lista de parámetros. 4- Tipo de devolución. 5- Inicio de la función. 6- Cuerpo de la función. 7- Fin de la función.

- Para invocar la función se escribe el nombre de la función a la que se quiere llamar y entre paréntesis se colocan los parámetros que exige esta función.
- **UPCASE:** Función que devuelve en mayúscula al carácter pasado como parámetro.
- **LENGTH:** Función que devuelve la longitud de la cadena pasada como parámetro.
- **GETDATE**: Función que devuelve el año, mes, día y el número de día de la semana, donde el lunes(1) y así hasta el domingo(7). Estos datos los almacena en las variables declaradas con ese objetivo, tienen que ser de tipo Word.
- **GETTIME**: Función que devuelve el tiempo en hora, minutos, segundos y centésimas.
- Para utilizar las funciones GETTIME y GETDATE debe ser declarada la unidad DOS en la lista Uses.
- **Función recursiva**: Es aquella que se llama así misma en el cuerpo de la función.
- Toda función recursiva tiene que estar concebida de forma tal que tenga un límite de llamadas a ella misma para que ponga fin a la función.

5.9 Ejercicios de autoevaluación.

1.1 Complete las siguientes oraciones:

a) Se llama _____ a un subprograma que realiza una acción y devuelve un valor único y _____ al subprograma que realiza una acción y no devuelve valor alguno.
b) _____ es una función que devuelve la longitud de la cadena pasada como parámetro.
c) La función GetTime retorna el tiempo en _____, _____, _____ y _____.
d) A la variable que se pasa como parámetro se le asigna un valor durante el proceso de ejecución del procedimiento se dice que este parámetro es pasado por _____.

1.2 Corrija los errores que existen (si es que existe alguno) en cada uno de los siguientes fragmento de programas.

a) El siguiente procedimiento le pasa al parámetro S el sucesor de N. S es un parámetro por referencia.

```
Procedur (N:Integer;S:Integer);
begin
 S:= N + 1;
end;
```

b) La siguiente función retorna el antecesor de un número.

```
Function AnteS(N):String;
begin
 AnteS = N - 1;
end;
```

c) Este procedimiento realiza una espera de S segundos, teniendo en cuenta que S es una variable entera que es un parámetro del procedimiento.

```
Procedure Espera() S: Integer;
begin
 Delay(S*1000);
end;
```

d) La función siguiente retorna verdadero si el número pasado como parámetro es par y falso si es impar.

```
Function Par(Num):String;
 If Num mor 2 = 2 Then
   Par:= True
 Else
   Par:= False;
```

1.3 Diga las partes para la declaración de un procedimiento según la numeración de la columna izquierda. El columna derecha aparecen las partes en forma desordenada.

① ② ③ PROCEDURE Nombre( ); ④ BEGIN ⑤ Acciones; ⑥ END;	__ Lista de parámetros. __ Palabra clave del procedimiento. __ Inicio del procedimiento. __ Nombre del procedimiento. __ Fin del procedimiento. __ Cuerpo del procedimiento.

1.4 Diga las partes para la declaración de una función según la numeración de la columna izquierda. El columna derecha aparecen las partes en forma desordenada.

① ② ③ ④ FUNCTION Nombre(PARAMT): T D; ⑤ BEGIN ⑥ Acciones; ⑦ END;	__ Tipo de devolución. __ Nombre de la función. __ Palabra clave de la función. __ Lista de parámetros. __ Fin de la función. __ Cuerpo de la función. __ Inicio de la función.

1.5 Escribir un programa que muestre en pantalla BP7 realizado por nosotros (usar un procedimiento para cada letra).

1.6 En el ejemplo 1 del epígrafe 5.4 se creó un procedimiento que permite generar dos números de 1 a P y almacenarlos en N1 y N2. Confeccione un procedimiento parecido a este pero que genere el primer número y este sea múltiplo del segundo.

5.10 Respuestas de ejercicios de autoevaluación.

1.1

a) Función, Procedimiento.
b) Length.
c) Hora, Minutos, Segundos y Centésimas.
d) Referencia o Dirección.

1.2 Los aspectos que están en negritas son los que fueron modificados o incorporados.

a)

```
Procedure Suce(N:Integer; Var S:Integer);
begin
 S:= N + 1;
end;
```

b)

```
Function AnteS(N: Integer):Integer;
begin
 AnteS := N - 1;
end;
```

c)

```
Procedure Espera(S: Integer);
begin
 Delay(S*1000);
end;
```

d)

```
Function Par(Num: Integer):Boolean;
begin
 If Num mod 2 = 0 Then
   Par:= True
 Else
   Par:= False;
end;
```

1.3

_3_ Lista de parámetros.
_1_ Palabra clave del procedimiento.
_4_ Inicio del procedimiento.
_2_ Nombre del procedimiento.
_6_ Fin del procedimiento.
_5_ Cuerpo del procedimiento.

1.4

_4_ Tipo de devolución.
_2_ Nombre de la función.
_1_ Palabra clave de la función.
_3_ Lista de parámetros.
_7_ Fin de la función.
_6_ Cuerpo de la función.
_5_ Inicio de la función.

1.5

```
Program BP7;
uses crt;
Procedure letra_B;
begin
writeln('B B B');
writeln('B    B ');
writeln('B B B');
writeln('B    B');
writeln('B B B ');
end;
Procedure letra_P;
begin
writeln('P P P');
writeln('P    P ');
writeln('P P P');
writeln('P');
writeln('P');
end;
Procedure num_7;
begin
writeln(' 7 7 7');
writeln('    7');
writeln('   7');
writeln('  7');
writeln(' 7');
end;
Begin {empezamos el programa principal}
Clrscr;
letra_b;
writeln;
letra_p;
writeln;
num_7;
repeat until keypressed;
 {mantenemos la pantalla viendo la solución hasta que se pulse
una tecla}
end.
```

1.6

```
Procedure Generar(P: Integer; Var N1, N2: Integer);
begin
 Randomize;
 N1:= Random(P)+1;
 Repeat
 N2:= Random(P)+1;
 Until N1 mod N2 = 0;
end;
```

5.11 Ejercicios propuestos.

1.7 Realice un procedimiento llamado Inicio que reciba como parámetro un nombre y le de la bienvenida, de forma tal que si el usuario escribe Antonio el resultado en pantalla sería:

```
                    Hola Antonio.
Te damos la bienvenida a Borland Pascal 7.0
Presione ENTER para continuar.
```

1.8 Cree un procedimiento que realice una espera, de forma tal que el programa no continúe hasta que no se presione la tecla ENTRE, cualquier otra tecla que se presione no responderá con ninguna acción.

1.9 Confeccione el programa completo del ejemplo 2 del epígrafe 5.4 donde el procedimiento devuelve los factores primos de un número, este número entra por el teclado.

1.10 Confeccione un procedimiento que permita realizar la siguiente operación:

El usuario escribe una palabra, pero mientras este la está entrando se van cambiando los caracteres por *. Por ejemplo si el usuario escribe Gerardo, lo que se ve es: * * * * * * *

1.11 Escriba un programa PASCAL que calcule el máximo y el mínimo de dos números. Sin embargo, dicho programa debe apoyarse en la programación modular. Con tal fin, proceda como sigue:

a) Escriba un procedimiento Leer que lea dos números reales.

b) Escriba una función que dados dos números reales, devuelva el máximo de ellos.

c) Escriba una función que dados dos números reales, devuelva el mínimo de ellos.

d) Escriba el programa principal que lea dos números reales, obtenga el mayor y el menor de ellos, y muestre el resultado en pantalla de la siguiente forma:

```
Primer número --- Segundo número --- Mayor --- Menor
     220.59              356.85          356.85    220.59
```

1.12 Construir una función que reciba como parámetro un entero y retorne su último dígito.

1.13 Construir una función que reciba como parámetro un entero y retorne sus dos últimos dígitos.

1.14 Construir una función que reciba como parámetro un entero y retorne la cantidad de dígitos pares.

1.15 Construir una función que reciba como parámetro un entero y retorne si dicho entero es primo o no.

1.16 Construir una función que reciba como parámetro un entero y retorne la cantidad de dígitos primos.

1.17 Construir una función que reciba como parámetro un entero y retorne el primer dígito de este entero.

1.18 Construir una función que reciba como parámetro un entero y un dígito y retorne 1 si dicho entero es múltiplo de dicho dígito y 0 si no es así.

1.19 Construya una función que reciba como parámetros un texto y un caracter y retorne la cantidad de veces que está dicho caracter en el texto.

1.20 Construir una función que reciba como parámetro un entero positivo y devuelva su equivalente en el sistema binario.

1.21 Construir una función que reciba como parámetro un entero positivo y devuelva su equivalente en el sistema octal.

1.22 Construir una función que reciba como parámetro un entero positivo y devuelva su equivalente en el sistema hexadecimal.

1.23 Construir una función que reciba como parámetro un número y retorne si este puede ser un número expresado en sistema binario o no.

1.24 Partiendo de los ejercicios anteriores, construya un programa que permita realizar una conversión de un sistema numérico a otro.

Las conversiones serán de:

- Decimal a Binario.
- Decimal a Octal.
- Decimal a Hexadecimal.
- Binario a Decimal.
- Octal a Decimal.
- Hexadecimal a Decimal.

1.25 Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne la posición en la cual se encuentra el mayor de los datos del arreglo.

1.26 Construir una función que reciba como parámetro un vector de 10 posiciones enteras y retorne el promedio entero del vector.

1.27 Construir una función que reciba como parámetros una matriz 4x4 entera y un valor entero y retorne la cantidad de veces que se repite dicho valor en la matriz.

1.28 Construir una función que reciba como parámetro una matriz 4x4 entera y retorne la posición exacta en donde se encuentre almacenado el mayor número primo.

1.29 Construir una función que reciba una matriz 5x5 y retorne el valor de su moda. La moda de un conjunto de datos es el dato que más se repite.

1.30 Construir una función recursiva que permita multiplicar A * B mediante una suma.

1.31 La secuencia de Fibonacci consiste en la suma consecutiva de los número de 0 hasta el número que se quiere mostrar dicha secuencia. Por ejemplo la secuencia Fibonacci menores que 50 son:

0, 1, 1, 2, 3, 5, 8, 13, 21 y 34.

1.32 Implemente en un programa la función factorial que se creo en este capítulo.

Bibliografía Consultada

- Estructura de datos en Pascal.
- Aprenda C/C++
- Estructura de datos y otros objetos.
- Folleto “Curso básico de turbo pascal”
- Metodología de la programación. Programación en pascal. Sánchez Victor.
- Introducción a la Programación. Pascal. Vicente López.
- Folleto de Programación I para los Politécnicos de Informática.