

Lógica de Programación. Nivel IV

agosto, 2019



Objetivos del nivel

- Validar datos
- Sacar provecho de los arreglos
- Aprender a hacer ordenamiento
- Aprender a trabajar con arreglos paralelos
- Aprender a trabajar con matrices
- Entender la Recursividad

Prerrequisitos del nivel

- Lógica de Programación Nivel III

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADI F1). Para obtener más información sobre este u otros cursos visite nuestro sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños. Copyright 2019. Todos los derechos reservados.



Contenido del nivel

Capítulo 1. Depuración de Programas

- 1.1.- Errores y Depuración.
- 1.2.- Depuración en Pseint.

Capítulo 2. Validaciones 1

- 2.1.- Principios Sobre Validaciones.
- 2.2.- Esquema de un Ciclo de Validación.
- 2.3.- Validación de Números.

Capítulo 3. Validaciones 2

- 3.1.- Validación de Caracteres.
- 3.2.- Validación de Cadenas.

Capítulo 4. Validaciones 3

- 4.1.- Conversiones y Validaciones.
- 4.2.- Conversión de Tipos de Datos.
- 4.3.- Validación de Número Entero.

Capítulo 5. Buenas Prácticas de Programación

- 5.1.- Generalidades Sobre Buenas Prácticas.
- 5.2.- Fácil Lectura y Entendimiento Del Código.
- 5.3.- Uso Adecuado de Las Variables.
- 5.4.- Prácticas Relacionadas Con la Modularidad.
- 5.5.- Prácticas Relacionadas Con Los Bucles.

Capítulo 6. Ordenamiento de Arreglos

- 6.1.- Introducción al Ordenamiento de Arreglos.
- 6.2.- Generalidades Sobre Algoritmos de Ordenamiento.
- 6.3.- Algoritmo Burbuja.

Capítulo 7. Arreglos Paralelos

- 7.1.- ¿Qué son los Arreglos Paralelos?.
- 7.2.- Carga de Arreglos Paralelos.
- 7.3.- Mostrar Valores en Arreglos Paralelos.

Capítulo 8. Uso de Arreglos Paralelos

- 8.1.- Mostrar Información de Forma Selectiva.
- 8.2.- Promediar Valores y Mostrado Selectivo.

Capítulo 9. Matrices

- 9.1.- ¿Qué es una Matriz y Cómo se declara?.
- 9.2.- Acceso a Elementos de Una Matriz.
- 9.3.- Recorriendo Una Matriz.
- 9.4.- Cargar y Mostrar Una Matriz.

Capítulo 10. Operaciones Con Matrices 1

- 10.1.- Generalidades Sobre Las Operaciones Con Matrices.
- 10.2.- Contadores Con Matrices.
- 10.3.- Acumuladores Con Matrices.

Capítulo 11. Operaciones Con Matrices 2

- 11.1.- Promedios Con Matrices.
- 11.2.- Porcentajes Con Matrices.

Capítulo 12. Operaciones Con Arreglos y Matrices 1

- 12.1.- Combinando Arreglos y Matrices.

Capítulo 13. Operaciones Con Arreglos y Matrices 2

- 13.1.- Matrices y Determinación de Mayores y Menores.

13.2.- Determinación de Mayores y Menores.

Capítulo 14. Recursividad

14.1.- Principios Sobre la Recursividad.

14.2.- Ejemplo de Función Factorial.



Capítulo 1. DEPURACIÓN DE PROGRAMAS

1.1.- Errores y Depuración

Cuando se programa se pueden presentar 3 tipos de errores:

De Sintaxis	<ul style="list-style-type: none">• Se dan cuando no se cumple con las reglas del lenguaje al conformar las instrucciones.• Éstos errores son indicados por el verificador sintáctico.
Lógicos	<ul style="list-style-type: none">• Ocurren cuando las operaciones aritméticas o lógicas no son correctas.• Este tipo de errores son difíciles de detectar, debido a que no son mostrados por el verificador sintáctico.
De Ejecución	<ul style="list-style-type: none">• Ocurren durante la ejecución del programa.• Pueden ser causados al ingresar un valor incompatible con el tipo que le fue asociado a una variable en su definición.• Pueden ser consecuencia de errores lógicos.

La DEPURACIÓN, consiste en recorrer un algoritmo en búsqueda de errores, en procura de resolverlos de la manera de forma que esto tome el menor tiempo posible.

La depuración de los programas es posible gracias a la existencia de herramientas, llamadas debuggers en inglés, que permiten realizar las siguientes tareas:

- ejecutar el código línea a línea
- examinar qué valores van tomando las variables a través de la ejecución
- detener la ejecución en un punto particular (break point)

El uso de los depuradores toma protagonismo para detectar y corregir errores lógicos, que pueden provocar que un programa no genere lo que se espera de él.

En general, la creación de programas usando algún lenguaje de programación se realiza usando algún entorno de desarrollo integrado conocido bajo el acrónimo IDE (aplicación informática que proporciona servicios integrales para facilitarle al programador el desarrollo de software) y todas ellas incluyen depuradores de código.

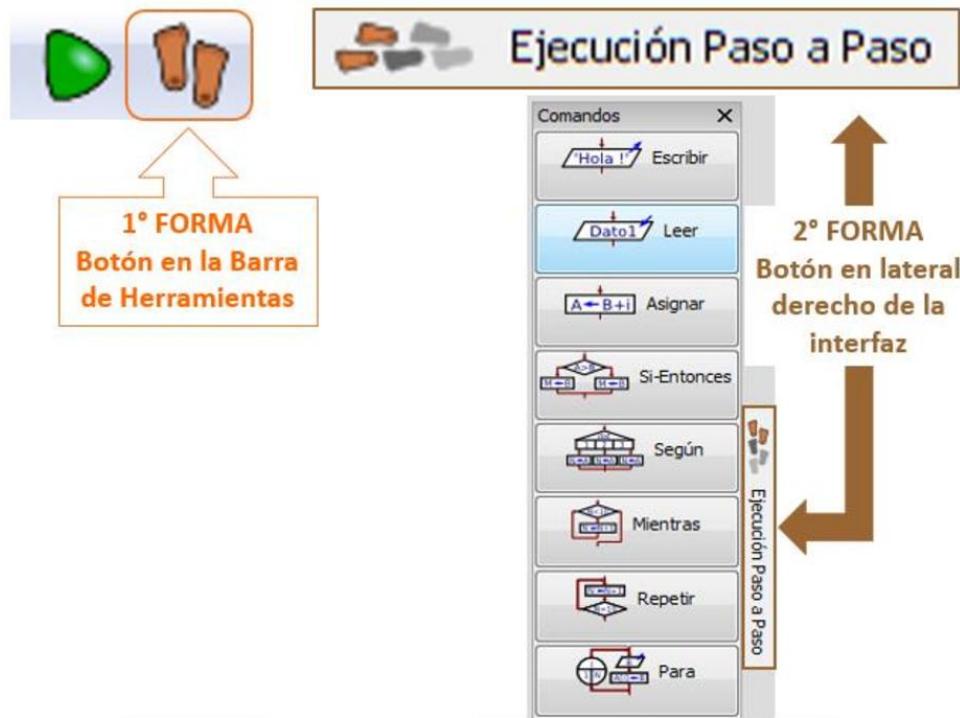
1.2.- Depuración en PseInt

En PseInt la herramienta para depurar los algoritmos se conoce bajo el nombre de "Ejecución Paso a Paso", y como todas las herramientas en su tipo, permite examinar que pasa al ejecutarse cada instrucción del código, para facilitar la detección y corrección de errores. Hay 2 formas de activar la depuración:

1º) Hacer clic en el botón ubicado en la barra de herramientas que tiene la imagen de los 2 pies.

2º) Hacer clic en el botón mostrado en el lateral derecho de la interfaz.





Al iniciar la Ejecución Paso a Paso, ocurren 3 eventos:

(1) Aparece una flecha al lado del número de línea apuntando a sentencia en ejecución, al mismo tiempo que esta es sombreada en verde. Por omisión, la ejecución proseguirá automáticamente a la siguiente instrucción (modo continuo) a menos que el usuario decida pausar la ejecución.

(2) Al lado derecho de la interfaz se despliega el Panel "Paso a Paso", donde hay ligeras diferencias en los botones dependiendo de cuál botón se presionó para iniciar la depuración. Ellas se ven a continuación.

(3) En la barra de estado se muestra un mensaje indicando que se está ejecutando "paso a paso"

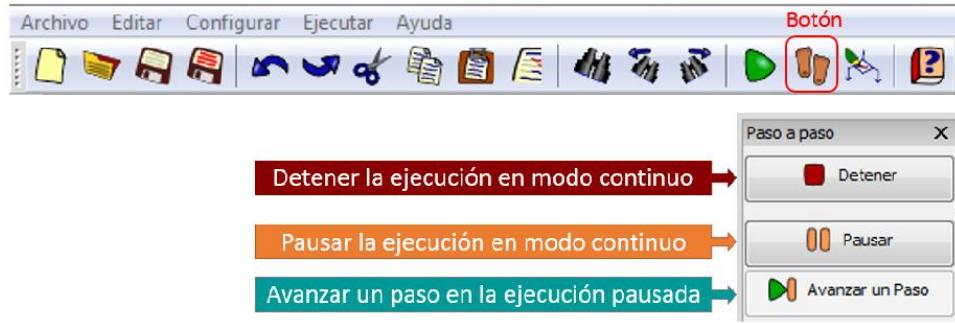
```

1 Algoritmo Decanato
2 // BLOQUE DECLARATIVO
3 Definir nomb_part,area_curso Como Caracter
4 Definir costo_curso Como Real
5 Definir cuota_inic Como Real
6 Definir cuota_mens Como Real
7 Definir balance_rest Como Real
8
9 // BLOQUE DE LECTURA
10 mostrar "Ingrese 1"
11 Esperar 2 segundos
12 mostrar "¿Nombre de"
13 leer nomb_part
14 mostrar "¿Área del"
15 leer area_curso
16 mostrar "¿Costo de"
17 leer costo_curso
18
19 //BLOQUE DE PROCESO
20 cuota_inic=costo_curso*0.20
21 balance_rest=costo_curso-cuota_inic
22 cuota_mens=balance_rest/24
23 cuota_mens=costo_curso*0.8/24
24
25 // BLOQUE DE SALIDAS
26 mostrar "Presione cualquier tecla para ver las salidas"
27 Esperar Tecla
28 Limpiar Pantalla
29 mostrar "Nombre del participante: " nomb_part
30 mostrar "Área del curso: " area_curso
31 mostrar "Costo total del curso: " costo_curso " Bs."
32

```

B pseudocódigo está siendo ejecutado paso a paso.

Si la depuración se inicia como consecuencia de haber pulsado el botón ubicado en la Barra de Herramientas, los botones que aparecen al tope de panel y su propósito se aprecian en la siguiente imagen:

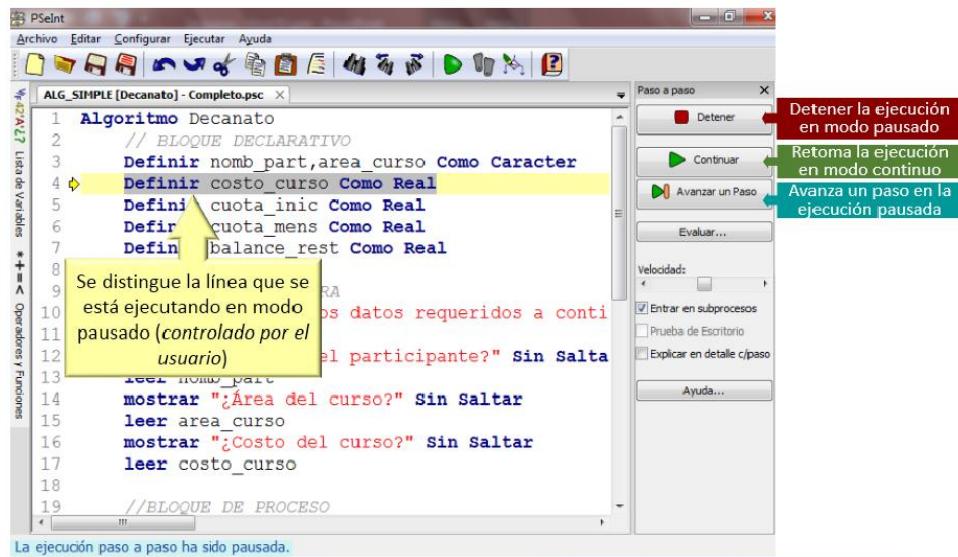


Si la depuración se inicia como consecuencia de haber pulsado el botón ubicado en el botón derecho de la interfaz, los botones que aparecen al tope de panel y su propósito se destacan en la imagen. Cabe destacar que los botones de la imagen anterior, sustituyen a los mostrados una vez se haya iniciado la depuración.

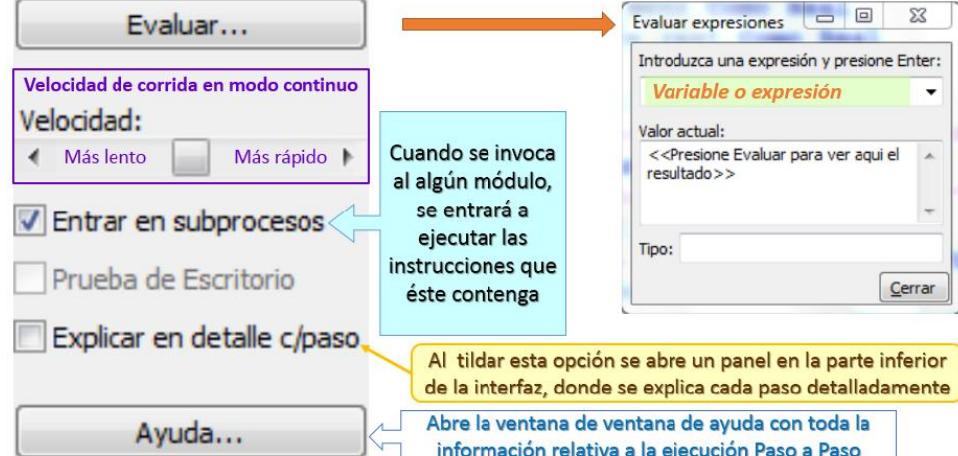


Si se desea observar con mayor minuciosidad lo que está ocurriendo al ejecutarse cada línea, debe pulsarse el botón "Pausar", acto seguido se detendrá la ejecución en modo continuo, y dependerá del usuario proseguir con ella. Note los cambios en la interfaz:

- al destacar la línea
- los botones en el panel "Paso a Paso"
- mensaje de estado, indicado que la ejecución está pausada



La siguiente porción de panel "Paso a Paso" estará presente indistintamente de la manera como se haya iniciado la depuración.



Capítulo 2. VALIDACIONES 1

2.1.- Principios Sobre Validaciones

En informática, la validación de datos consiste en el uso de técnicas para asegurarse de que los datos captados por una aplicación sean lo más fieles a los esperados, esto es, que sean claros, correctos y útiles para su procesamiento y la consecuente generación de los resultados. Cabe destacar que, dichos datos pueden ser suministrados a otros programas, aplicaciones y servicios.

Para validar los datos se utilizan rutinas, a menudo llamadas "reglas de validación", "restricciones de validación" o "rutinas de comprobación", que comprueban que los datos que se introducen en el sistema cumplen los requisitos.

Por ejemplo, los sistemas de gestión de bases de datos permiten la implementación de algunos métodos de validación útiles, pero ello escapa al alcance de este curso.

Aplicando Lógica de Programación, en particular, creando funciones se pueden implementar validaciones, lo cual se desarrollará en éste capítulo y los 2 subsiguientes.

La importancia de la validación de datos radica en que:

1) Comprueba que los datos están aptos para el propósito deseado, lo cual los hará de uso razonable y dará certidumbre sobre los mismos antes de su procesamiento.

2) No validar correctamente o cometer omisiones en la validación de datos puede conducir a:

- Tener datos inútiles, generará información inútil.
- Ocurrencia de errores lógicos.

- Obtención de resultados no esperados, lo cual creará dudas al respecto de la confiabilidad del programa o sistema.
- Dar lugar a agujeros de seguridad.

En cuanto a la validación deben tenerse presente siempre las siguientes reglas:

- Dondequiera que se le vaya requerir el ingreso de un dato al usuario que debe tomar valores que cumpla con uno o varios criterios, ese dato debe validarse.
- Debe usarse siempre el tipo de datos más flexible según el dato a captar.
- Evitar el ingreso de datos absurdos, tales como una edad negativa, por encima de los 120 años o teniendo como contenido una cadena de caracteres.
- Siempre se le debe informar al usuario lo más específicamente posible, usando lenguaje simple, cual fue el error cometido al ingresar un dato incorrecto.
- Tener siempre presente que el usuario podría intencionalmente ingresar datos no válidos para evaluar la confiabilidad del programa.

2.2.- Esquema de un Ciclo de Validación

Para validar datos en un programa se puede usar el ciclo "Mientras" o el ciclo "Repetir" en cualquiera de sus variantes, sin embargo, éste último es el más intuitivo. Recuérdese que la condición evaluada en el Repetir-Hasta es opuesta a la evaluada en el Repetir-Mientras.



Una opción al usar Repetir-Mientras, es mantener corriendo el ciclo siempre que se cumpla la negación de la condición que se plantearía en el Repetir-Hasta.



Al implementar el ciclo Repetir deben tenerse en cuenta las siguientes consideraciones:

- La condición de ruptura debe evaluar la validez del dato.
- Las acciones internas usan mensaje y dato, las cuales deberían ser parámetros de la función validadora.
- La lectura del dato que se pretende validar siempre debe estar dentro del ciclo, porque él debe ser leído todas las veces que sea necesario hasta que su valor sea válido.

2.3.- Validación de Números

A continuación, se presentan 2 ejemplos de validación de números, donde se adopta como premisa que el usuario no ingresará valores diferentes a números.

En el siguiente algoritmo, la función realiza la lectura validada de números hasta que sea ingresado un número positivo:

```
Funcion numPos=Func_LeerPos(Mensaje)
    Limpiar Pantalla
    Repetir
        Mostrar Mensaje Sin Saltar
        Leer num
        Hasta Que (num>0)
        numPos=num
    FinFuncion

Algoritmo Validar_NumPositivo
    Mostrar "Ingrese números hasta ingresar un número positivo"
    Mensaje="Ingrese un número positivo"
    numPos=Func_LeerPos(Mensaje)
FinAlgoritmo
```

En el siguiente algoritmo se lee un número dentro de un rango, para lo cual se implementa un módulo tradicional para leer de forma validada los topes del rango y una función para leer el número hasta que se encuentre dentro del rango de forma validada:

```
Subproceso Leer_Topes(topeInf Por Referencia,topeSup Por Referencia)
    Repetir
        topeInf=Func_LeerPos("Ingrese el tope inferior del rango")
        topeSup=Func_LeerPos("Ingrese el tope superior del rango")
        Si topeInf>topeSup
            entonces
                Mostrar "El valor mínimo supera al máximo, reingréselos"
            FinSi
        Mientras Que ~(topeInf<=TopeSup)
    FinSubProceso

    Funcion num=Func_LeerNumEnRango(Mensaje,topeInf,topeSup)
        Limpiar Pantalla
        Repetir
            Mostrar Mensaje topeInf " a " topeSup Sin Saltar
            Leer num
        Mientras Que ~(num>=topeInf Y num<=topeSup)
    FinFuncion

    Algoritmo Validar_Nums
        Leer_Topes(tInf,tSup)
        num=Func_LeerNumEnRango("Ingrese número en el rango de ",tInf,tSup)
    FinAlgoritmo
```



Capítulo 3. VALIDACIONES 2

3.1.- Validación de Caracteres

Así como se puede implementar funciones para validar números, también se pueden implementar funciones para validar cadenas. Es frecuente usar funciones para validar este tipo de dato, cuando de su validez depende la determinación de los otros datos y/o la progresión exitosa en la ejecución del programa, ejemplos de ello son:

- Validar que usuario introduzca una talla válida, para luego determinar el precio unitario.
- Validar el género de una persona, de lo cual depende determinar si se incrementa un contador u otro.
- Validar la respuesta de un usuario, para controlar la ruptura de un ciclo.

La versión más simple de una función, para validar una cadena maneja los valores válidos como literales de texto. Esto se exemplifica en la función implementada en el algoritmo para leer el género de una persona.

```

Funcion genero=Func_LeerGenero(Mensaje)
    Luminpiar Pantalla
    Repetir
        Mostrar Mensaje Sin Saltar
        Leer genero
        genero=Mayusculas(genero)
    Hasta Que (genero=="F" O genero=="M")
FinFuncion

Algoritmo Validar_genero
    genero=Func_LeerGenero("Ingrese el género de la persona (F/M)")
FinAlgoritmo

```

Diagrama de flujo que muestra la ejecución de la función Func_LeerGenero. Se indica que el resultado es 'Literales de Texto'.

La validación de caracteres cuya respuesta es dicotómica se puede implementar con una función reusable como la que se ejemplifica, donde se le solicita al usuario que ingrese los 2 caracteres válidos:

```

Subproceso Leer_Letras_Validas(letral1 Por Referencia,letra2 Por Referencia)
    Mostrar "Ingrese las 2 letras válidas:"
    Leer letral1,letra2
    letral1=Mayusculas(letral1)
    letra2=Mayusculas(letra2)
FinSubProceso

Funcion letra=Func_LeerLetra(Mensaje,letral1,letra2)
    Limpiar Pantalla
    Repetir
        Mostrar Mensaje
        Leer letra
        letra=Mayusculas(letra)
    Hasta Que (letra=letral1 O letra=letra2)
FinFuncion
    FinFuncion → Valores variables

Algoritmo Validar_letras
    Leer_Letras_Validas(letral1,letra2)
    Mensaje="Ingrese letras (sólo "+letral1+ " y " +letra2+" son válidas)"
    letra=Func_LeerLetra(Mensaje,letral1,letra2)
FinAlgoritmo

```

3.2.- Validación de Cadenas

Lo implementado en la validación de caracteres se puede extrapolar para validar cadenas y para explicarlo, a continuación, se muestran las partes de un algoritmo modular con reusabilidad, cuyo objetivo es leer una cadena, validando que ésta se encuentre entre un conjunto de cadenas válidas que serán ingresadas por el usuario. El cuerpo principal es el siguiente:

```

Algoritmo Validar_Cadenas_Con_Arreglo
    // Validación de la cantidad de cadenas y dimensionamiento del arreglo
    Cant_Cadenas=Func_LeerPos("Ingrese la cantidad de cadenas válidas")
    Dimension ArrCadenas[Cant_Cadenas]
    // Validación de la longitud máxima de las cadenas a almacenar
    LongMaxCad=Func_LeerPos("Ingrese la longitud máxima de una cadena")
    // Cargar arreglo con las cadenas válidas
    Cargar_Letras_Validas(Cant_Cadenas,LongMaxCad,ArrCadenas)
    // Lectura de una cadena de forma validada
    cade=Func_LeerCad(Cant_Cadenas,ArrCadenas)
FinAlgoritmo

```

La primera función que es necesario usar es la que se vió en el capítulo anterior para asegurarse que el usuario ingrese un número positivo, lo cual debe cumplirse tanto para la cantidad de cadenas (Cant_Cadenas), como para la longitud máxima de la cadena (LongMaxCad). Por ello, Func_LeerPos se invoca 2 veces en el cuerpo principal.

```
Funcion numPos=Func_LeerPos(Mensaje)
    Repetir
        |   Mostrar Mensaje Sin Saltar
        |   Ler num
        |   Hasta Que (num>0)
        |   numPos=num
    FinFuncion
```

El siguiente módulo tiene como objetivo cargar un arreglo con las cadenas válidas, para lo cual debe asegurarse que ninguna cadena se repita.

```
Subproceso Cargar_Letras_Validas(CantCad,LongMaxCad,ArrCad Por Referencia)
    i=1 // Inicialización de la variable indice para el posicionamiento en el arreglo
    Repetir
        |   Repetir
        |       |   Mostrar "Ingrese la " i ""° cadena:" Sin Saltar
        |       |   Ler cad
        |       |   cad=Mayusculas(cad) // Conversión de la cadena a Mayúsculas
        |       |   LongCad=Longitud(cad) // Cálculo de la longitud de la cadena
        |       |   cadVacia=(cad=="") // Asignación de valor lógico de cad="" a cadVacia
        |       |   Si cadVacia
        |           |       |   entonces
        |               |       |       |   Mostrar "La cadena debe tener al menos 1 carácter"
        |               |       |   sino Si LongCad>LongMaxCad
        |                   |       |       |   entonces
        |                       |       |       |       |   Mostrar "La cadena ingresada excede la longitud máxima"
        |                   |       |   FinSi
        |   FinSi
        |   Hasta Que (LongCad<=LongMaxCad y ~cadVacia) // Se rompe cuando la longitud sea válida
        |   // Una vez que la longitud de la cadena es válida se verifica si existe, de no
        |   // existir se incluirá la cadena en el arreglo
        |   cadExiste=Func_BuscCad(CantCad,ArrCad,cad)
        |   Si ~cadExiste
        |       |   entonces
        |           |       |   ArrCad[i]=cad // Guardado en el arreglo de la cadena
        |           |       |   i=i+1 // Incremento de la variable indice
        |           |       |   sino Mostrar "La cadena " cad " ya está registrada"
        |   FinSi
    Mientras Que i<=CantCad // Se itera mientras no se haya alcanzado el tamaño del arreglo
FinSubProceso
```

Para asegurarse que no se dupliquen las cadenas en el arreglo, es necesario hacer búsqueda de elementos únicos en el arreglo donde están guardada las cadenas (esto se vio en el capítulo 14 del nivel 3), lo cual se implementó en la siguiente función:

```
Funcion cadExiste=Func_BuscCad(CantCad,ArrCad,cadBuscada)
    cadExiste=Falso
    i=1
    Repetir
        Si ArrCad[i]=cadBuscada
            entonces cadExiste=Verdadero
            sino i=i+1
        FinSi
        Hasta Que cadExiste o i>CantCad
    FinFuncion
```

Una vez se ha cargado el arreglo con las cadenas válidas, se puede leer una cadena de forma validada, y esto se implementa en la siguiente función:

```
Funcion letra=Func LeerCad(CantCad,ArrCad)
    Luminar Pantalla
    Repetir
        Mostrar "Ingrese una cadena válida entre las siguientes " Sin Saltar
        Mostrar Carac Validos(CantCad,ArrCad)
        Leer cad
        cad=Mayusculas(cad)
        CadExiste=Func_BuscCad(CantCad,ArrCad,cad)
    Hasta Que CadExiste
FinFuncion
```

La función anterior, utiliza el siguiente módulo para mostrar las cadenas válidas, con el objeto de hacer lo más informativo posible el mensaje que le indica al usuario que es lo que debe ingresar:

```
Subproceso Mostrar_Carac_Validos(CantCad,ArrCad)
    Para i=1 hasta CantCad
        Mostrar ArrCad[i] Sin Saltar
        Si i<CantCad
            entonces Mostrar "/" Sin Saltar
        FinSi
    FinPara
FinSubProceso
```



Capítulo 4. VALIDACIONES 3

4.1.- Conversiones y Validaciones

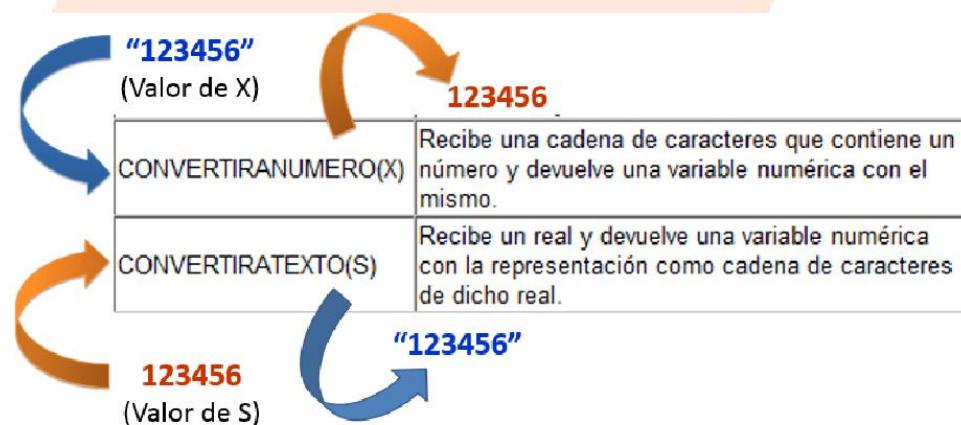
Lo visto hasta ahora, en cuanto a las validaciones se fundamenta en la premisa de que el usuario siempre ingresará un dato que es compatible con el tipo de dato asociado a la variable donde se almacena el valor leído. Sin embargo, no se puede contar con que eso será siempre así.

Por lo anterior, para blindar la validación se perfila como indispensable realizar un análisis más exhaustivo del dato ingresado por el usuario.

Es así que, para hacer la validación infalible, en el caso de números, será necesario manejar la posibilidad de que el usuario ingrese un dato, cuyo tipo sea incompatible con el que fue asociado a la variable en cuestión. A los efectos de lograr el propósito se debe hacer uso de las funciones de conversión.

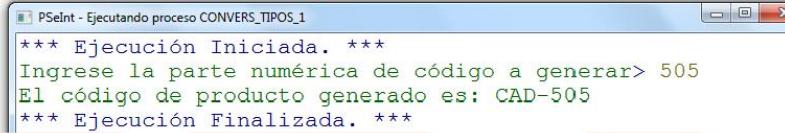
4.2.- Conversión de Tipos de Datos

Para hacer conversiones PseInt implementa las 2 funciones destacadas en la imagen:



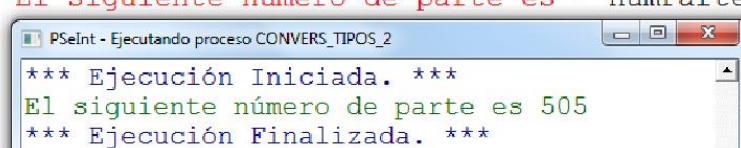
Antes de ver la utilización de ellas a efectos de validar datos, es importante entender cómo operan las mismas. En el siguiente ejemplo, se utiliza la función "ConvertirATexto" para generar un código de producto partiendo de la premisa de que el usuario ingresará un número de parte de tipo numérico entero:

```
Algoritmo Convers_Tipos_1
    Definir prefijoFab,codProd como Caracter
    Definir numParte como Entero
    prefijoFab="CAD-"
    Mostrar "Ingrese la parte numérica de código a generar" Sin Saltar
    Leer numParte
    // Generación del código de producto
    codProd=prefijoFab+ConvertirATexto(numParte)
    Mostrar "El código de producto generado es: " codProd
FinAlgoritmo
```



Siguiendo en la tónica del número de parte, en el siguiente ejemplo se extrae del último código de producto asignado, los últimos 3 caracteres que representan el número de parte, para convertirlo usando la función "ConvertirANumero", para usar ese valor numérico para incrementarlo y generar el siguiente número de parte:

```
Algoritmo Convers_Tipos_2
    Definir numParteTxt,codProd como Caracter
    Definir numParte como Entero
    // Generación del siguiente número de parte
    codProd="CAD-504"
    numParteTxt=Subcadena(codProd,5,7)
    numParte=ConvertirANumero(numParteTxt)
    numParte=numParte+1
    mostrar "El siguiente número de parte es " numParte
FinAlgoritmo
```



4.3.- Validación de Número Entero

En el algoritmo cuyas partes se refieren a continuación, se implementa la validación de una edad, que se lee como carácter, para manejar la posibilidad de que si el usuario ingresara letras cuando se le requiera la edad, no se genere un error de ejecución. El cuerpo principal del algoritmo sería el siguiente:

```
Algoritmo Leer_Edad_Validada
    Definir edad como Entero

        edad = Func_Validar_Entero("Introduzca la edad:")
        Mostrar "La edad es: ",edad
    FinAlgoritmo
```

Esta es la función invocada desde el cuerpo principal, la cual está diseñada para leer valores hasta que el valor ingresado sea entero.

Obsérvese que se examinan uno a uno los caracteres de la cadena leída hasta que se encuentra un carácter no numérico.

```

Funcion valor_valido <- Func_Validar_Entero(Mensaje)
  Definir cad_caracteres,carac_i Como Caracter
  Definir caracterEsNumero como Logico
  Definir i,Long_cadena Como Entero

  Repetir
    Mostrar Mensaje Sin Saltar
    Leer cad_caracteres
    Long_cadena = Longitud(cad_caracteres)
    i=1
    Repetir
      carac_i = Subcadena(cad_caracteres,i,i)
      caracterEsNumero = Func_Validar_Caracter_Numerico(carac_i)
      Si caracterEsNumero // Aplica al último carácter examinado
        entonces i = i + 1
      FinSi
      Mientras Que (i<=Long_cadena y caracterEsNumero)
      si ~caracterEsNumero entonces
        Mostrar ";Valor no válido!... Intente de nuevo"
        Mostrar ""
      FinSi
      Hasta Que caracterEsNumero
    valor_valido = ConvertirANumero(cad_caracteres)
  Fin Funcion

```

Desde Func_Validar_Entero se invoca a esta función, que verifica si un determinado carácter es un dígito numérico.

```

Funcion es_numero <- Func_Validar_Caracter_Numerico(caract)
  Definir caracteres_numericos,carac_num como caracter
  Definir Long_carac_nums,i Como Entero

  caracteres_numericos = "0123456789"
  Long_carac_nums = Longitud(caracteres_numericos)
  i = 1
  Repetir
    carac_num = Subcadena(caracteres_numericos,i,i)
    es_numero = (caract = carac_num)
    i = i + 1
  Hasta Que (es_numero o i > Long_carac_nums)
FinFuncion

```

Capítulo 5. BUENAS PRÁCTICAS DE PROGRAMACIÓN

5.1.- Generalidades Sobre Buenas Prácticas

En general, la MANTENIBILIDAD es la propiedad de un sistema que representa la cantidad de esfuerzo (hr/recurso humano) requerida para conservar su funcionamiento normal o para restituirlo una vez se ha presentado un evento de falla.

Un sistema es Altamente Mantenible cuando el esfuerzo asociado a darle mantenimiento o a realizar restitución es bajo.

Los sistemas de Baja Mantenibilidad o poco mantenibles requieren de grandes esfuerzos para sostenerse o restituirse.

La Alta Mantenibilidad de los programas informáticos se puede lograr adoptando buenas prácticas de programación que se presentan en este capítulo.

Cuando se detecta la necesidad de desarrollar una aplicación informática, puede existir un número potencialmente infinito de programas que satisfagan los mismos requisitos. Sin embargo, no todos estos programas comparten los mismos atributos de calidad, ya que:

- no todos son igualmente eficientes, tanto en consumo de procesador como de memoria; como tiempo de ejecución.
- no todos son igual de legibles.
- no todos son igual de fáciles de modificar.
- no todos son igual de fáciles de probar y verificar que funcionan correctamente.

Por lo anterior, se recomienda el apego a las buenas prácticas de programación que se indican a continuación.

Las siguientes son prácticas de carácter general:

No usar caracteres propios del castellano en la codificación de programas, ya que las letras con símbolos (ácento, virgulilla o diéresis), al no ser caracteres básicos ASCII, pueden sufrir alteraciones cuando un archivo de código se abre en computadoras diferentes, incluso si ambas contienen el mismo sistema operativo.

Se debe colocar una sentencia o instrucción por línea.

Debe procurarse mantener la longitud de la línea en 80 caracteres máximo. Si debe aplicar un salto de línea para dar continuidad a una instrucción, se debe aplicar la indentación en la(s) línea(s) de continuación (esto último no es posible en PseInt).

No dejar espacios inmediatamente dentro de paréntesis, corchetes o llaves.

Deben evitarse los archivos de muchas líneas. En el caso de Java, por ej. No debería superar las 2000 líneas, ya que son incómodos de manejar.

Se debe evitar al máximo asignar literales dentro del código, por lo cual se recomienda leer todo dato que pueda hacer que el código sea más flexible y mantenible.

Se debe saber depurar sin usar depuradores, esto significa, valerse de sentencias que sirven para mostrar información por consola. Los depuradores de código son unas excelentes herramientas, sin embargo, suelen fallar cuando se trata de integrar una aplicación con un sistema mantenido por terceros, cuyo código fuente no está disponible. Además, no funcionan normalmente cuando se trabaja con simuladores de tecnologías novedosas, como, por ejemplo, aquellos diseñados para dispositivos móviles.

5.2.- Fácil Lectura y Entendimiento Del Código

Hay varios aspectos a cuidar para facilitar la lectura del código:

- Indentación del código.
- Dejar espacios en blanco.
- Dejar líneas en blanco.
- No usar espacios en blanco en algunos casos.
- Estandarizar lo referente a las sentencias de asignación.

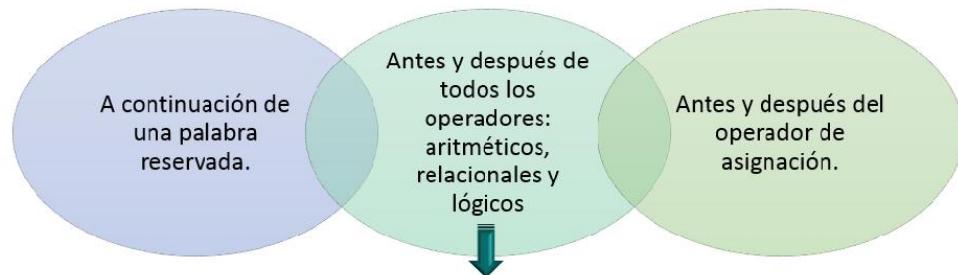
El cuidado sobre la indentación es uno de los aspectos claves:

Usar la indentación estándar de cuatro (4) espacios, que corresponde a un tabulador, para hacer sangrado a la derecha tantas veces como sea necesario

Las sentencias contenidas dentro de una instrucción de control (selectivas o iterativas) deben estar indentadas un nivel más adentro que ésta

Utilizar los tabuladores para la indentación típica. Cualquier espacio en blanco luego del nivel de indentación deben ser espacios reales, de modo que el formateo será razonable independientemente de la cantidad de espacios a los que un tabulador sea equivalente

Es importante dejar espacios en blanco en estos casos:



La única excepción a esta regla es cuando se usan operadores unarios, que actúan sobre un solo operando:

- El signo de menos (-) antepuesto para indicar que ese operando es negativo
- El signo de más (+) colocado 2 veces después de una variable para incrementarla en 1, por ej.: j++
- El signo de menos (-) colocado 2 veces después de una variable para decrementarla en 1, por ej.: k--

Para romper con la monotonía visual de ver una línea de código a continuación de otra se recomienda dejar líneas en blanco en los siguientes casos:

Antes de un comentario de una línea o de bloque.

Antes de un bloque de código que tenga un propósito particular.

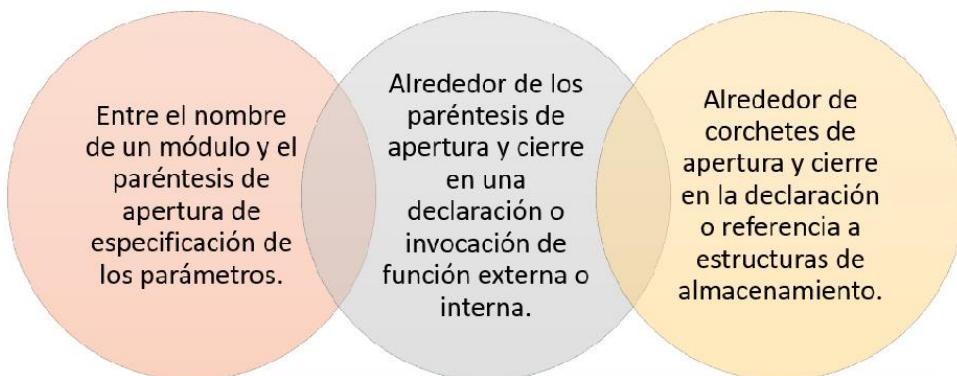
Para separar "párrafos" de líneas de código relacionados,

- Ej.: en un programa secuencial, una línea en blanco entre secciones del bloque de proceso.

Entre la definición de variables locales en un módulo y su primera sentencia.

Entre la definición de un módulo y la del módulo inmediato inferior.

No se debe usar espacios en blanco en los siguientes casos:



En cuanto a las sentencias de asignación debe velarse por:



Además, para facilitar el entendimiento del código se recomienda usar comentarios:

Es indispensable documentar las porciones de código donde se hacen cálculos o determinaciones que ameritan explicación.

En programación secuencial se recomienda documentar el propósito de cada bloque de código.

A continuación de una sentencia se pueden hacer un comentario breve.

En el caso de comentar varias sentencias consecutivas, los comentarios deben ser movidos a la derecha lo suficientemente lejos para separarlos de ellas.

Si más de un comentario corto aparece en el mismo trozo de código, deben ser indentados con la misma profundidad.

Para que sea fácil e intuitivo entender el propósito de todas las variables, los módulos y los parámetros, se debe cumplir con las siguientes recomendaciones al identificarlos:

- * Cada identificador debe ser mnemotécnico, de manera que incluso un lector casual al verlo infiera su uso, es decir, que sea significativo.
- * No se debe abusar de las abreviaciones, porque se pudieran dar ambigüedades al intentar interpretar su propósito.
- * Se recomienda adoptar un tipo de nomenclatura:

Upper Case

- Todas las letras del identificador se encuentran en mayúsculas.
- Ej.: EJEMPLODENOMENCLATURA

Camel Case

- Los identificadores tiene mayúsculas y minúsculas, tiene dos (2) variantes:
 - Upper Camel Case: la primera letra de cada una de las palabras es mayúscula. Ej.: EjemploDeNomenclatura
 - Lower Camel Case: igual que la anterior con la excepción de que la primera letra es minúscula. Ej.: ejemploDeNomenclatura.

Snake Case

- Cada una de las palabras, se separa por un guión bajo (_). Ej.: ejemplo_de_nomenclatura

5.3.- Uso Adecuado de Las Variables

Las siguientes prácticas están relacionadas con el uso apropiado de las variables:

Los identificadores de variables no deben ser muy largos.

Se debe evitar variables identificadas con una sola letra, excepto en variables temporales de uso breve.

Se debe utilizar siempre una variable para un único propósito.

Todas las variables deben declararse, además se recomienda colocar solamente una declaración por línea con su correspondiente comentario.

Se debe evitar el uso de variables globales, ya que, a cualquier variable globalmente accesible, pudiera ser modificada fácilmente en cualquier parte del programa. Lo cual pudiera desencadenar una serie de fallos en cadena, cuya resolución tomaría un tiempo valioso.

En cuanto a las declaraciones... éstas deberían plantearse antes de cualquier otra sentencia. En programación secuencial las declaraciones deben situarse al comienzo del cuerpo principal y en programación modular al comienzo de los módulos.

Hay dos (2) excepciones con respecto la regla previa:

- Declaración de las variables dentro de una estructura como un bucle. En algunos lenguajes de programación como Java, la declaración de la variable índice se coloca dentro de la sentencia que marca el inicio del bucle. Por ej: `for (int i = 0; ...)`
- Si se tiene que leer el tamaño de una estructura de almacenamiento, antes de dimensionarla la declaración debe hacerse después de la lectura.

Es recomendable inicializar todas las variables en donde resulte conveniente a efectos de lograr el objetivo del programa:



5.4.- Prácticas Relacionadas Con la Modularidad

Para aplicar de la mejor manera posible la modularidad y facilitar la ubicación de los módulos y su entendimiento deben seguirse los siguientes lineamientos:

- 1) Establecer una nomenclatura para nombrar los módulos y funciones externas.
- 2) La especificación de los parámetros formales y actuales, debe colocarse dentro de un juego de paréntesis que no esté separado por espacios en blanco del nombre del módulo.
- 3) Los módulos deben tener especificados sólo los parámetros que requiera, indistintamente que sean parámetros de entrada y/o de salida.

- 4) Se deben evitar módulos con contenido superior a 24 líneas de código "reales" (es decir, sin contar las líneas en blanco que se agregan para mayor legibilidad y los comentarios).
- 5) Todos los datos que no deben salir de un módulo, que no sean parámetros de entrada deben ser manejados como variables locales.
- 6) En general, si dos o más trozos de código pueden aparecer juntos en un solo módulo o separados en varios sub-módulos, la opción recomendada siempre es separarlos, salvo justificación irrefutable.
- 7) Los módulos deben ser "singulares en propósito", es decir, un módulo debería realizar una sola cosa (cohesión), y el nombre del mismo debería reflejar esto en forma precisa. Si un módulo realiza más de una cosa que se relacionen de alguna manera, asegúrese que esto esté reflejado en su nombre.
- 8) Se recomienda evitar tener 10 parámetros o más en los módulos, si no existe una causa plenamente justificada para ello. De esta manera, se asegura que los módulos no se encuentran sobrecargados de funcionalidad, favoreciendo el mantenimiento eficiente del código.
- 9) Evitar copiar y pegar trozos quasi idénticos de código a lo largo de un programa, ya que eso provocaría una cantidad de líneas de código innecesarias. De darse ese caso, es altamente probable que la mejor opción sea crear un módulo reusable.
- 10) Cuando se va a construir un módulo reusable tanto su nombre como sus parámetros deben ser significativos, pero de carácter general.
- 11) Cuando se van a crear una función externa, se debe asegurar que, dentro de ella, se le asigne a la variable de retorno el valor que corresponda, a los efectos de cumplir con el objetivo de la función.

5.5.- Prácticas Relacionadas Con Los Bucles

Para hacer uso de los bucles de manera óptima, se recomienda:

Optar por usar un bucle PARA c/vez que sea posible. Las ventajas de ese bucle es que reúne el control del bucle en un solo lugar, y permite que se use una variable de control (variable índice) que no es accesible desde fuera del mismo y es modificada automáticamente, por lo cual nunca se debe modificar dentro del bucle, porque se alteraría el comportamiento del mismo.

Optar por un bucle Mientras cuando se necesite manejar la posibilidad de que nunca se entre al mismo.

Optar por el bucle Repetir cuando es necesario que se realice al menos una iteración.

En los bucles cuya ruptura depende de la especificación de una condición explícita (Repetir y Mientras), se debe asegurar que en el cuerpo del bucle se realiza los cómputos y/o determinaciones que converjan hacia tal condición, de otra manera se caería en un ciclo infinito.

Se debe procurar hacer bucles que sean lo suficientemente cortos como para ver todas sus acciones internas a la vez. Esto es especialmente importante si el cuerpo del bucle es complejo. Si el código del bucle es mayor a las 15 líneas, se debe considerar la posibilidad de reestructurar el código, a través de estrategias como la modularización.



Capítulo 6. ORDENAMIENTO DE ARREGLOS

6.1.- Introducción al Ordenamiento de Arreglos

Los valores en un arreglo están de forma implícita organizados secuencialmente (uno después del otro), pero no necesariamente están ordenados.

Al cargar los datos en un arreglo (leer por el teclado) generalmente se cargan de forma desordenada. En muchos casos se puede sacar provecho del arreglo sin ordenarlo.

Sin embargo, hay situaciones que ameritan ordenar los elementos de un arreglo, es el caso cuando es solicitado mostrar un listado ordenado de valores.

Los criterios de orden, que se pueden aplicar en el ordenamiento de los elementos de un arreglo, son los que se han manejado con anterioridad:

- Ascendente. Ej.: Listado de notas ordenados de menor a mayor.
- Descendente. Ej.: Listado de notas ordenados de mayor a menor.

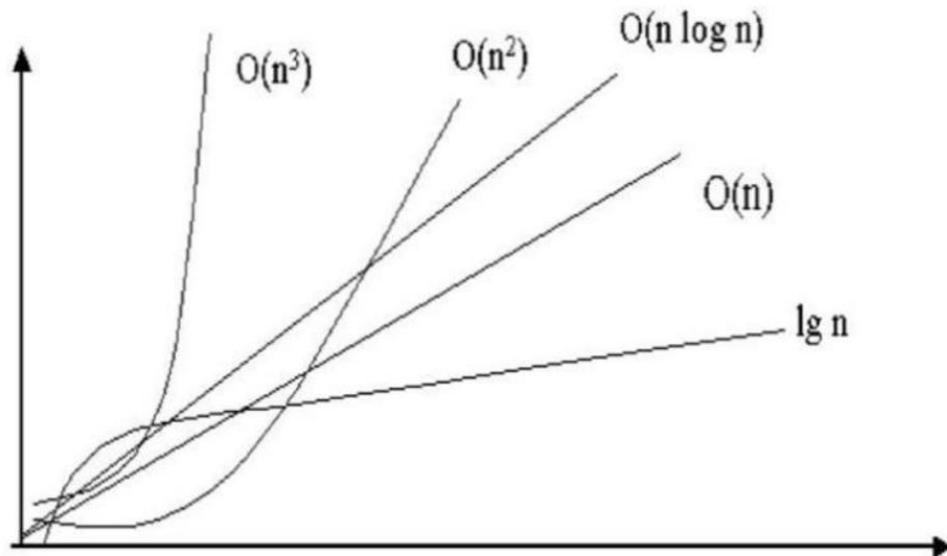
6.2.- Generalidades Sobre Algoritmos de Ordenamiento

Existen varios algoritmos de ordenamiento ya predefinidos para ordenar los elementos de un arreglo. A continuación, se muestra una tabla con los algoritmos que son estables, es decir, los que son efectivos en completar el ordenamiento.

Nombre traducido	Nombre original	Complejidad
Ordenamiento de burbuja	Bubblesort	$O(n^2)$
Ordenamiento de burbuja bidireccional	Cocktail sort	$O(n^2)$
Ordenamiento por inserción	Insertion sort	$O(n^2)$ ("en el peor de los casos")
Ordenamiento por casilleros	Bucket sort	$O(n)$
Ordenamiento por cuentas	Counting sort	$O(n+k)$
Ordenamiento por mezcla	Merge sort	$O(n \log n)$
Ordenamiento con árbol binario	Binary tree sort	$O(n \log n)$
	Pigeonhole sort	$O(n+k)$
Ordenamiento Radix	Radix sort	$O(nk)$
	Distribution sort	$O(n^3)$ versión recursiva
	Gnome sort	$O(n^2)$

Todos los algoritmos computacionales tienen un grado de complejidad, que perfila la eficiencia del mismo en el uso de los recursos a través del tiempo. Esa complejidad se cataloga como el orden (O) de algoritmo, el cual se expresa generalmente en términos de la cantidad de datos procesados por el programa, denominada n , que puede ser el tamaño dado o estimado.

En esta gráfica se aprecia cómo cambian los órdenes de complejidad de la mayoría de los algoritmos de ordenamiento:



6.3.- Algoritmo Burbuja

El algoritmo de ordenamiento más sencillo es el denominado "Burbuja", donde los valores se van comparando en pares y se intercambian de ser necesario. En el siguiente ejemplo del algoritmo burbuja se ordenan los elementos de menor a mayor (ascendente):

```

Algoritmo Burbuja_Ascendente
    Dimension Arreglo[tamanio]
    // Carga del arreglo
    Para i=1 hasta tamanio-1
        Para j=1 hasta tamanio-1
            Si Arreglo[j]>Arreglo[j+1]
                Entonces
                    aux=Arreglo[j]
                    Arreglo[j]=Arreglo[j+1]
                    Arreglo[j+1]=aux
                FinSi
            FinPara
        FinPara
    FinAlgoritmo

```

Este algoritmo "Burbuja" ordena los elementos del arreglo de forma descendente:

```
Algoritmo Burbuja_Descendente
    Dimension Arreglo[tamano]
    // Carga del arreglo
    Para i=1 hasta tamano-1
        Para j=1 hasta tamano-1
            Si Arreglo[j]<Arreglo[j+1]
                Entonces
                    aux=Arreglo[j]
                    Arreglo[j]=Arreglo[j+1]
                    Arreglo[j+1]=aux
                FinSi
            FinPara
        FinPara
    FinAlgoritmo
```

Capítulo 7. ARREGLOS PARALELOS

7.1.- ¿Qué son los Arreglos Paralelos?

En programación, se habla de "Arreglos Paralelos" cuando se tienen 2 o más arreglos con datos relacionados entre sí, los cuales poseen el mismo tamaño.

La relación que hay entre ellos es que en la posición que se esté visitando en cada arreglo se guarda un dato de cada entidad. Ejemplos de entidades son: alumno, cliente, carro, sucursal, etc.

La cantidad de arreglos paralelos necesarios en pro de la resolución de un problema a través de un algoritmo no tiene un máximo.

Por ejemplo, si se necesita almacenar los nombres y las notas de 7 estudiantes, se necesitan 2 arreglos de 7 posiciones: el 1ero de tipo alfanumérico para los nombres y el 2do de tipo real para las notas.

Obsérvese que en la posición 1 de cada arreglo están los datos del 1er estudiante, en la posición 2 de cada arreglo están los datos del 2do estudiante, y así sucesivamente.

Nombres						
Alberto José	Ana María	Pedro Juan	Maria José	Víctor Daniel	José Juan	Ana Julia
1	2	3	4	5	6	7

Notas						
12.5	16.8	9.3	19.1	11.7	18.3	8.5
1	2	3	4	5	6	7

Extendiendo el ejemplo anterior, se incorporan 2 nuevos arreglos para guardar las edades y los sexos de los 7 estudiantes:

Nombres

Alberto José	Ana María	Pedro Juan	María José	Víctor Daniel	José Juan	Ana Julia
1	2	3	4	5	6	7

Notas

12.5	16.8	9.3	19.1	11.7	18.3	8.5
1	2	3	4	5	6	7

Edades

22	19	21	20	25	23	20
1	2	3	4	5	6	7

Sexos

M	F	M	F	M	M	F
1	2	3	4	5	6	7

Tomando como base el ejemplo anterior, es evidente que los datos de cada estudiante se tendrán en la posición que se esté visitando, por ejemplo:

En la primera posición de c/arreglo se tiene la información del 1er estudiante:

Nombre: Alberto José

Nota: 12.5

Edad: 22 años

Sexo: Masculino ("M")

En la segunda posición de c/arreglo se tiene la información del 2do estudiante:

Nombre: Ana María

Nota: 16.8

Edad: 19 años

Sexo: Femenino ("F")

... y así sucesivamente.

Como se explicó con anterioridad, el tipo de ciclo más usado para trabajar con arreglos es el ciclo "Para", pero también ese puede usar los ciclos "Repetir" o "Mientras", teniendo la previsión de manejar la variable índice de forma apropiada. Esta aseveración también aplica a la manipulación de Arreglos Paralelos.

7.2.- Carga de Arreglos Paralelos

Cuando se manejan arreglos paralelos, todos deben cargarse paralelamente para respetar la relación que hay entre las posiciones en cada arreglo, es decir, todas las posiciones 1 deben cargarse en la 1era iteración del ciclo, todas las posiciones 2 deben cargarse en la 2da iteración del ciclo, y así sucesivamente hasta llenar todas las posiciones de los arreglos en paralelo. Por ejemplo, en el caso de usar solo 2 arreglos (nombres y notas) para registrar la información de los estudiantes, se emplea el siguiente algoritmo:

```
1 Algoritmo Carg_Nomb_Y_Notas|
2     Dimension nombres[7]
3     Dimension notas[7]
4
5     Para i=1 Hasta 7
6         Mostrar "Ingrese los datos del " i "º estudiante"
7         Mostrar "Nombre: " Sin Saltar
8         Leer nombres[i]
9         Mostrar "Nota: " Sin Saltar
10        Leer notas[i]
11    FinPara
12 FinAlgoritmo
```

Si se trabaja de forma modular, se trasladaría el ciclo al módulo que quedaría según se muestra. Nótese que ambos arreglos deben ser declarados antes de invocar al módulo y deben ser parámetros por referencia del módulo que registrará los datos en ellos.

```

1 SubProceso Cargar_Arreglos(nomb por Referencia,not por Referencia)
2   Para i=1 hasta 7
3     Mostrar "Ingrese los datos del " i "º estudiante"
4     Mostrar "Nombre: " Sin Saltar
5     Leer nomb[i]
6     Mostrar "Nota: " Sin Saltar
7     Leer not[i]
8   FinPara
9 FinSubProceso
10
11 Algoritmo Carg_Nomb_Y_Notas
12   Dimension nombres[7]
13   Dimension notas[7]
14
15   Cargar_Arreglos(nombres,notas)
16 FinAlgoritmo

```

Extiendo el ejemplo para registrar los datos de los estudiantes con los arreglos de Edades y Sexos, el algoritmo para cargar los 4 arreglos sería el que se muestra. Obsérvese que se han añadido las sentencias para cargar dichos arreglos.

```

1 Algoritmo Carg_Arreglos
2   Dimension nombres[7],notas[7],edades[7],sexos[7]
3
4   Para i=1 hasta 7
5     Mostrar "Ingrese los datos del " i "º estudiante"
6     Mostrar "Nombre: " Sin Saltar
7     Leer nombres[i]
8     Mostrar "Nota: " Sin Saltar
9     Leer notas[i]
10    Mostrar "Edad: " Sin Saltar
11    Leer edades[i]
12    Mostrar "Sexo: " Sin Saltar
13    Leer sexos[i]
14  FinPara
15 FinAlgoritmo

```

7.3.- Mostrar Valores en Arreglos Paralelos

Para mostrar el contenido de arreglos paralelos de forma modular, siguiendo con el ejemplo de los nombres y notas de los alumnos, en el presente

algoritmo se aprecia la similitud con el módulo para cargar los arreglos, en cuanto a acceder los datos que están en la posición visitada en cada iteración con el objeto de mostrarlos:

```
1 SubProceso Mostrar_Arreglos(nomb,not)
2   Para i=1 hasta 7
3     Mostrar "Datos del " i "º estudiante"
4     Mostrar "Nombre: " nomb[i]
5     Mostrar "Nota: " not[i]
6   FinPara
7 FinSubProceso
8
9 Algoritmo Most_Nomb_Y_Notas
10  Dimension nombres[7]
11  Dimension notas[7]
12
13  Mostrar_Arreglos(nombres,notas)
14 FinAlgoritmo
```

Capítulo 8. USO DE ARREGLOS PARALELOS

8.1.- Mostrar Información de Forma Selectiva

Con arreglos paralelos se puede desplegar listados de información que están basados en la evaluación de una condición. Por ejemplo, con los arreglos paralelos de nombres y notas, se puede imprimir un listado de los alumnos que obtuvieron la mayor nota:

```
Mostrar "Los alumnos que obtuvieron la mayor nota fueron: "
Para i=1 hasta 7
  Si notas[i]=mayornota
    entonces Mostrar nombres[i]
  FinSi
FinPara
Mostrar "La mayor nota la obtuvieron" cont "alumnos"
```

Gracias al uso de arreglos paralelos, también se puede determinar a quién pertenece el valor mayor, por ejemplo, la mayor nota. En este algoritmo, de haber empates, se guardará sólo el 1er nombre que corresponde con tener la mayor nota:

```
// inicialmente se asume que en la 1era posición está el mayor
mayornota=notas[1]
quien=nombres[1]
para i=2 hasta 15 Hacer
  Si notas[i] > mayornota entonces
    mayornota=notas[i]
    quien=nombres[i]
  FinSi
FinPara
Mostrar "La mayor nota fue: " mayornota
mostrar "y la obtuvo: " quien
```

Usando los mismos arreglos paralelos nombres y notas, el siguiente sería el algoritmo que permitiría saber quién es el primer estudiante que obtuvo la menor nota:

```
// inicialmente se asume que en la 1era posición está el menor
menornota=notas[1]
quien=nombres[1]
para i=2 hasta 15 Hacer
    Si notas[i] < menornota entonces
        menornota=notas[i]
        quien=nombres[i]
    FinSi
FinPara
Mostrar "La menor nota fue: " menornota
Mostrar "y la obtuvo: " quien
```

8.2.- Promediar Valores y Mostrado Selectivo

Siguiendo con el ejemplo del empleo de los arreglos de nombres y notas de 7 estudiantes se puede mostrar cuantos estudiantes tienen nota superior al promedio. Éste sería el cuerpo principal del algoritmo:

```
Algoritmo ListarMayProm
    Dimension nombres[7],notas[7]
    Definir prom_not como real

    Cargar_Arreglos(nombres,notas)
    prom_not=Func_Prom_Notas(notas)
    Mostrar_NotMayProm(nombres,notas,prom_not)
FinAlgoritmo
```

Lo primero que debe hacerse al cargar los arreglos paralelos y el módulo para hacerlo es el que se muestra. Nótese que se invoca a una función para leer una nota válida en el rango de 1 a 100 ptos.

```

Subproceso Cargar_Arreglos(nombres Por Referencia,notas Por Referencia)
    Definir i como Entero
    Definir nota como Real

    Para i=1 hasta 7
        Mostrar "Ingrese los datos del " i "º estudiante "
        Mostrar "Nombre: " Sin Saltar
        Leer nombres[i]
        nota=Func_LeerNota("Nota: ",1,100)
        notas[i]=nota
        Mostrar ""
    FinPara
FinSubProceso

```

La presente función se invoca en el módulo anterior para asegurarse que se lea una nota válida en el rango comprendido entre NotaMin y NotaMax, cuyos valores recibidos por parámetros en el presente algoritmo son 1 y 100 respectivamente.

```

Funcion nota=Func_LeerNota(Mensaje,NotaMin,NotaMax)
Repetir
    Mostrar Mensaje Sin Saltar
    Leer not
    Mientras Que ~(not>=NotaMin Y not<=NotaMax)
        nota=not
    FinFuncion

```

La siguiente función invocada desde el Cuerpo Principal, es la que determina el promedio de notas, recorriendo para ello el arreglo notas:

```

Funcion prom_notas=Func_Prom_Notas(notas)
    Definir acum_notas como real
    Definir nota como Real

    acum_notas=0
    Para cada nota de notas
        acum_notas=acum_notas+nota
    FinPara
    prom_notas=acum_notas/7
FinFuncion

```

El último módulo invocado, presenta el promedio determinado para hacer más informativa la salida y luego genera un listado con la información de los estudiantes para los cuales se cumple que su nota es mayor al promedio. Es destacable que se está accediendo a los 2 arreglos a efectos de generar las salidas:

```
Subproceso Mostrar_NotMayProm(nombres,notas,prom_not)
    Limpiar Pantalla
    Mostrar "El promedio de notas fue: " prom_not " ptos."
    Para i=1 hasta 7
        Si notas[i]>prom_not
            Mostrar nombres[i] " obtuvo " notas[i] Sin Saltar
            Mostrar " ptos, por lo cual supera la nota promedio"
        FinSi
    FinPara
FinSubProceso
```

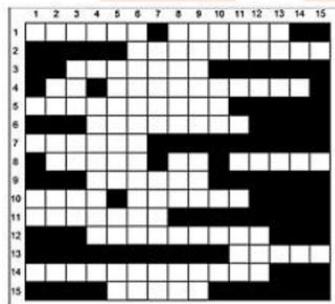
Capítulo 9. MATRICES

9.1.- ¿Qué es una Matriz y Cómo se declara?

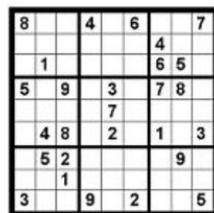
Una matriz es un arreglo bidimensional de valores organizados en filas (o renglones) y columnas, donde una fila es cada una de las líneas horizontales de la matriz y una columna es cada una de las líneas verticales. A una matriz con M filas y N columnas se le denomina matriz MxN. El tamaño de una matriz siempre se refiere con el número de filas primero y el número de columnas después. Dos matrices se dice que son iguales si tienen el mismo tamaño y los mismos valores.

Las matrices se utilizan para múltiples aplicaciones, en particular, sirven para representar los coeficientes de los sistemas de ecuaciones lineales o para representar las aplicaciones lineales.

Hay muchos ejemplos de matrices en la vida real, uno de ellos son los horarios de clase, los crucigramas, sudoku y excel, entre otros.



	Lunes	Martes	Miercoles	Jueves	Viernes
7-9					
9-11					
11-1					



	A	B	C	D	E	F
1						
2	850.00	x	77.10	=	100,000.00	
3	1700.00	x	38.55	=	100,000.00	
4	3400.00	x	19.28	=	100,000.00	
5	6800.00	x	9.64	=	100,000.00	
6	13600.00	x	4.82	=	100,000.00	
7	27200.00	x	2.41	=	100,000.00	
8	425.00	x	154.20	=	100,000.00	
9	212.50	x	308.40	=	100,000.00	
10	106.25	x	616.80	=	100,000.00	
11	53.13	x	1233.60	=	100,000.00	
12	26.56	x	2467.20	=	100,000.00	
13	13.28	x	4934.40	=	100,000.00	
14						

Las matrices se declaran como un arreglo, con la diferencia de que se definen 2 dimensiones: la cantidad de filas y la cantidad de columnas (siempre en ese orden). De forma general la definición es la siguiente:

Dimension nombre_matriz[numFilas,numColumnas]

donde:

numFilas y numColumnas son 2 números enteros.

La cantidad de filas y de columnas dependerá del problema que se desea resolver.

9.2.- Acceso a Elementos de Una Matriz

Por ejemplo, si se necesitan almacenar los montos vendidos cada día de la semana en una empresa que cuenta con 4 sucursales, se puede guardar estos montos en una matriz, de 4x7 o de 7x4.

En caso de ser 4x7, las filas representan las sucursales y las columnas los días de la semana.

En caso de ser 7x4, las filas representan los días de la semana y las columnas las sucursales.

<code>matriz[1..4 , 1..7] real</code>	<u>Días de la semana</u>	
<code>matriz[1..7 , 1..4] real</code>	<u>Sucursales</u>	

Para acceder a los elementos de una matriz se deben usar 2 índices, uno para referirse a la fila y otro para referirse a la columna, siempre en ese orden, primero la fila y luego la columna (en algunos lenguajes de programación esto puede cambiar).

Siguiendo el ejemplo de las ventas diarias de las 4 sucursales de una empresa... si se necesita registrar que el martes en la sucursal 3 el monto de las ventas fue 8.470.00, la instrucción de asignación sería:

`matriz[2,3] = 8470000`

Nótese que no se colocan los puntos, porque en programación se usa punto en números reales para marcar el inicio de los decimales. Otros ejemplos serían:

`matriz[7,2] = 6854000`

`matriz[5,4] = 0`

	1	2	3	4
1	9.125.000	1.788.500	0	6.149.500
2	4.195.000	0	8.470.000	0
3	2.455.670	0	1.304.500	0
4	0	4.860.000	1.978.450	5.896.000
5	1.045.600	0	5.320.000	0
6	0	7.144.500	0	9.781.500
7	0	6.854.000	6.187.540	0

9.3.- Recorriendo Una Matriz

Para recorrer las matrices se deben utilizar 2 ciclos, uno dentro de otro, lo cual se vio con anterioridad bajo el concepto de "Anidamiento de Ciclos".

El ciclo que se usa generalmente para trabajar con matrices es el ciclo "Para", debido a que se siempre se conoce de antemano la cantidad de filas y columnas de la matriz.

El recorrido se puede hacer por filas o por columnas. El recorrido más común se hace por filas, pero la escogencia final dependerá del programador y/o del problema.

Este es el anidamiento de ciclos para recorrer la matriz 7x4 por filas:

```
Para f = 1 hasta 7
  para c = 1 hasta 4
    // instrucciones
  Fin para
Fin para
```

Nótese que el ciclo externo corresponde a las filas.

	1	2	3	4
1	9.125.000	1.788.500	0	6.149.500
2	4.195.000	0	8.470.000	0
3	2.455.670	0	1.304.500	0
4	0	4.860.000	1.978.450	5.896.000
5	1.045.600	0	5.320.000	0
6	0	7.144.500	0	9.781.500
7	0	6.854.000	6.187.540	0

Este es el anidamiento de ciclos para recorrer la matriz 7x4 por columnas:

```

Para c = 1 hasta 4
  Para f = 1 hasta 7
    //instrucciones
  Fin para
Fin para

```

Nótese que el ciclo externo corresponde a las columnas.

	1	2	3	4
1	9.125.000	1.788.500	0	6.149.500
2	4.195.000	0	8.470.000	0
3	2.455.670	0	1.304.500	0
4	0	4.860.000	1.978.450	5.896.000
5	1.045.600	0	5.320.000	0
6	0	7.144.500	0	9.781.500
7	0	6.854.000	6.187.540	0

9.4.- Cargar y Mostrar Una Matriz

La instrucción de lectura debe estar dentro del anidamiento, que permitirá acceder todas las posiciones de la matriz, de esta forma, ésta se va recorriendo y a su vez se van almacenando los datos en ella. En este ejemplo de registran las ventas semanales de las 4 sucursales en una matriz 7x4:

Algoritmo

Dimension Vtas_Sem[7,4]

Para c=1 hasta 4

 Para f = 1 hasta 7

 Mostrar "Ingrese monto vendido el día " f " en la sucursal " c Sin Saltar

 Leer Vtas_Sem[f,c]

 Fin para

Fin para

FinAlgoritmo

Al igual que la instrucción de lectura, la instrucción para mostrar, debe estar entre dos ciclos y de este modo se hará el recorrido de toda la matriz, mostrando sus elementos.

Algoritmo

Dimension Vtas_Sem[7,4]

// Cargar de la matriz

Para f=1 hasta 7

 Mostrar "Los montos de ventas del " f "º día fueron: " Sin Saltar

 Para c=1 hasta 4

 Mostrar Vtas_Sem[f,c] "-" Sin Saltar

 Fin para

 Mostrar " " // Para que produzca un salto de línea

 Fin Para

Fin Algoritmo

Capítulo 10. OPERACIONES CON MATRICES 1

10.1.- Generalidades Sobre Las Operaciones Con Matrices

Con las matrices se pueden hacer las mismas operaciones que con los arreglos:

- contar todos los elementos,
- contar valores que cumplan con una condición,
- sumatorias (acumuladores) por fila o por columna,
- cálculo de promedios globales,
- cálculo de promedio por fila o por columna,
- determinar el valor mayor de una fila o de una columna,
- buscar un valor en toda la matriz, en una fila o en una columna en particular
- ordenar por fila o por columna o toda la matriz.

Dependiendo del caso se tendrán que emplear 2 ciclos o uno, y se deberá recorrer la matriz por fila o por columna.

10.2.- Contadores Con Matrices

Para saber cuántos elementos tiene una matriz basta con multiplicar el número de filas por el número de columnas. Por lo cual sería innecesario contar todos los elementos.

Ahora bien, resulta muy útil recorrer la matriz para realizar conteos selectivos, de manera que el contador sólo se incremente al cumplirse una condición.

Siguiendo con el ejemplo de las ventas de las sucursales, es muy relevante contar cuantas veces el monto de ventas es 0. Dado que es necesario un contador, éste es de carácter general porque no se refiere a obtener información detallada por cada sucursal. Por lo tanto, debe inicializarse fuera del anidamiento de ciclos e incrementarse dentro del ciclo interno.

```

Algoritmo Contar_Vtas0
    Dimension Vtas_Sem[7,4]
    Definir f,c,cont_0 como Entero
    // Carga de la matriz
    cont_0 = 0
    Para f=1 hasta 7
        Para c=1 hasta 4
            Si Vtas_Sem[f,c]=0
                entonces cont_0=cont_0+1
            FinSi
        FinPara
    FinPara
    Mostrar "Hubo " cont_0 " cierres de venta en 0"
FinAlgoritmo

```

Al procesar datos almacenados en matrices, muchas veces es necesario usar contadores de carácter específico. En el siguiente algoritmo, se usa un contador para contabilizar cuantos días de la semana cada sucursal no registró ventas. Nótese que el contador se inicializa justo antes de entrar al ciclo interno donde se actualiza solamente si se cumple la condición.

```

Algoritmo Contar_Vtas0_xSuc
    Dimension Vtas_Sem[7,4]
    Definir f,c,cont_0 como Entero
    // Carga de la matriz
    Para c=1 hasta 4
        cont_0_xSuc = 0
        Para f=1 hasta 7
            Si Vtas_Sem[f,c]=0
                entonces cont_0_xSuc =cont_0_xSuc+1
            FinSi
        FinPara
        Mostrar "La " c "º sucursal no " Sin Saltar
        Mostrar "registró ventas en " cont_0_xSuc " días"
    FinPara
FinAlgoritmo

```

10.3.- Acumuladores Con Matrices

Usando los datos de la matriz donde se registraron las ventas de las sucursales se pueden calcular el acumulado general, y también se pueden calcular

acumulados específicos. Este algoritmo, calcula la sumatoria de todas las ventas recorriendo la matriz por filas:

```
Algoritmo Totaliz_Vtas_Sem
    Dimension Vtas_Sem[7,4]
    Definir f,c,acum_vtas como Entero
    // Carga de la matriz
    acum_vtas=0
    Para f=1 hasta 7
        Para c=1 hasta 4
            acum_vtas=acum_vtas+Vtas_Sem[f,c]
        FinPara
        Mostrar "Total de ventas de la semana: " acum_vtas " Bs."
    FinPara
FinAlgoritmo
```

La determinación de acumuladores específicos, también pudiera ser necesario al procesar datos en una matriz. Por ej.: si se necesita calcular el monto vendido en toda la semana por una sucursal en particular, será necesario usar un acumulador específico. En este algoritmo, se le requiere al usuario cual es la sucursal (suc) para la que desea obtener el acumulado y se usa un solo ciclo para acceder posiciones de la matriz, moviéndose a través de las filas dejando fija la columna (suc):

```
Algoritmo Totaliz_Vtas_Suc
    Dimension Vtas_Sem[7,4]
    Definir f,suc,acum_vtas_suc como Entero
    // Carga de la matriz
    acum_vtas_suc=0
    Mensaje="Ingrese la sucursal (1-4) a la que desea totalizar las ventas"
    suc=Func_LeerNum(Mensaje,1,4) // Función que lee un número en rango
    Para f=1 hasta 7
        acum_vtas_suc=acum_vtas_suc+Vtas_Sem[f,suc]
    FinPara
    Mostrar "Total de ventas de la " suc Sin Saltar
    Mostrar "° sucursal: " acum_vtas_suc " Bs."
FinAlgoritmo
```

Capítulo 11. OPERACIONES CON MATRICES 2

11.1.- Promedios Con Matrices

Para obtener el promedio general de los valores almacenados en una matriz, deben acumularse todos los valores en una variable que se inicialice fuera del anidamiento de ciclos y se actualice dentro del ciclo interno. Finalizadas las MxN iteraciones debe calcular el promedio. Esto se ejemplifica con la matriz que se ha venido utilizando:

```
Algoritmo Prom_Gral_Vtas
    Dimension Vtas_Sem[7,4]
    Definir f,c,acum_vtas como Entero
    Definir prom_vtas como Real
    // Carga de la matriz
    acum_vtas=0
    Para f=1 hasta 7
        Para c=1 hasta 4
            |   |   acum_vtas=acum_vtas+Vtas_Sem[f,c]
        FinPara
    FinPara
    prom_vtas=acum_vtas/28 // MxN -> M filas y N columnas
    Mostrar "Promedio General de Ventas " prom_vtas " Bs."
FinAlgoritmo
```

Así como se puede acumular los valores de una fila o de una columna, se puede calcular un promedio por fila o por columna. Usando la matriz de ventas de las sucursales, si se necesitara calcular el promedio de ventas de un día en particular, se deben acumular todos los valores de la fila que corresponda al número de día ingresado por el usuario y fuera del ciclo se calcula el promedio, dividiendo el acumulado entre la cantidad de sucursales que están representadas en las columnas:

```

Algoritmo Prom_Vtas1Dia
    Dimension Vtas_Sem[7,4]
    Definir c,dia,acum_vtas_dia como Entero
    Definir prom_vtas_dia como Real
    // Carga de la matriz
    Mensaje="¿Día de la semana para el cual desea el promedio de ventas (1-7) ?"
    Dia=Func_LeerNum(Mensaje,1,7)
    acum_vtas_dia=0
    Para c=1 hasta 4
        acum_vtas_dia=acum_vtas_dia+Vtas_Sem[dia,c]
    FinPara
    prom_vtas_dia=acum_vtas_dia/4
    Mostrar "Promedio de ventas del " dia "º día: " prom_vtas_dia
FinAlgoritmo

```

11.2.- Porcentajes Con Matrices

Procesando datos de matrices también se pueden obtener porcentajes. Siguiendo con la matriz de las sucursales, se puede obtener el porcentaje de los montos de venta que son iguales a cero:

```

Algoritmo Deter_Porc_Vtas0
    Dimension Vtas_Sem[7,4]
    Definir f,c,cont_0 como Entero
    Definir porc_vtas0 como Real
    // Carga de la matriz
    cont_0 = 0
    Para f=1 hasta 7
        Para c=1 hasta 4
            Si Vtas_Sem[f,c]=0
                entonces cont_0=cont_0+1
            FinSi
        FinPara
    FinPara
    porc_vtas0=(cont_0/28)*100
    Mostrar "Porcentaje de ventas en 0: " porc_vtas0 "%"
FinAlgoritmo

```

Otra información útil que se podría determinar con respecto a las ventas de las sucursales es cuál es el porcentaje de las ventas registradas que superan al promedio general de ventas:

```

Algoritmo Deter_Porc_SupAlProm
    Dimension Vtas_Sem[7,4]
    Definir f,c,cont_SupProm como Entero
    Definir prom_vtas,porc_vtas_SupProm como Real
    // Carga de la matriz
    prom_vtas=Func_Det_PromVtas(Vtas_Sem)
    cont_SupProm = 0
    Para f=1 hasta 7
        Para c=1 hasta 4
            Si Vtas_Sem[f,c]>prom_vtas
                entonces cont_SupProm=cont_SupProm+1
            FinSi
        FinPara
    FinPara
    porc_vtas_SupProm=(cont_SupProm/28)*100
    Mostrar "El promedio general de ventas es: " prom_vtas " Bs."
    Mostrar "y el porcentaje de ventas que lo superan es: " porc_vtas_SupProm "%"
FinAlgoritmo

```

Consultando la matriz de ventas de las sucursales, y solicitándole al usuario un día específico, se puede determinar cuál es el porcentaje de las sucursales cuyas ventas superaron al promedio general de ventas:

```

Algoritmo Deter_Porc_SucSupAlProm
    Dimension Vtas_Sem[7,4]
    Definir f,c,cont_SucSupProm como Entero
    Definir prom_vtas,porc_vtas_SucSupProm como Real
    // Carga de la matriz
    Mensaje="¿Día de la semana en el cual desea contrastar con el promedio de ventas (1-7)?"
    Dia=Func_LeerNum(Mensaje,1,7)
    prom_vtas=Func_Det_PromVtas(Vtas_Sem)
    cont_SucSupProm = 0
    Para c=1 hasta 4
        Si Vtas_Sem[dia,c]>prom_vtas
            entonces cont_SucSupProm=cont_SucSupProm+1
        FinSi
    FinPara
    porc_vtas_SucSupProm=(cont_SucSupProm/4)*100
    Mostrar "El promedio general de ventas es: " prom_vtas " Bs."
    Mostrar "El porcentaje de sucursales cuyas ventas " Sin Saltar
    Mostrar "superaron al promedio es: " porc_vtas_SucSupProm "%"
FinAlgoritmo

```

Capítulo 12. OPERACIONES CON ARREGLOS Y MATRICES 1

12.1.- Combinando Arreglos y Matrices

Los arreglos y matrices se pueden usar de forma simultánea, para generar información más elaborada, que puede requerirse en problemas donde:

- se desea presentar información de forma más intuitiva o
- se enfrenta un nivel de complejidad que supera a los problemas tipo que se han visto hasta ahora.

El uso simultáneo de estas estructuras de almacenamiento, que hasta ahora se ha visto por separado, se desarrollará en este capítulo dándole continuidad al ejemplo de las ventas que se registran en 4 sucursales de una empresa, a lo largo de los 7 días de la semana.

Para ello, se parte de la premisa de que además de la matriz donde se registran los montos de venta, se manejarán 2 arreglos, uno para los nombres de las sucursales y otro para los nombres de los días de la semana:

Sucursales [4]	Metrópolis	Barquicenter	Sambil	Trinitarias
Días[7]	1	2	3	4
lunes	9.125.000	1.788.500	0	6.149.500
martes	4.195.000	0	8.470.000	0
miércoles	2.455.670	0	1.304.500	0
jueves	0	4.860.000	1.978.450	5.896.000
viernes	1.045.600	0	5.320.000	0
sábado	0	7.144.500	0	9.781.500
domingo	0	6.854.000	6.187.540	0

A continuación, se presentan el algoritmo modular de un algoritmo cuyo objetivo es mostrar por día de la semana, los nombres de las sucursales cuyas

ventas superaron al promedio general de ventas. El siguiente es el cuerpo principal:

```
Algoritmo Prom_Gral_Vtas
    Dimension Dias[7],Sucursales[4]
    Dimension Vtas_Sem[7,4]
    Definir prom_vtas como Real

    Cargar_Arreglos(Dias,Sucursales)
    Cargar_Matriz(Dias,Sucursales,Vtas_Sem)
    prom_vtas=Func_Det_PromVtas(Vtas_Sem)
    MostrarInfo_Vtas_SupProm(prom_vtas,Dias,Sucursales,Vtas_Sem)
FinAlgoritmo
```

El 1er módulo invocado carga los arreglos. Cabe destacar que los nombres de las sucursales se asignaron de forma estática, sólo para encajar con los valores mostrados en la imagen previa, pero lo recomendable es cargar ese arreglo requiriéndole al usuario los nombres de las mismas.

```
Subproceso Cargar_Arreglos(Dias Por Referencia,Sucursales por Referencia)
    // Carga del arreglo de los días de la semana
    dias[1]="lunes"
    dias[2]="martes"
    dias[3]="miércoles"
    dias[4]="jueves"
    dias[5]="viernes"
    dias[6]="sábado"
    dias[7]="domingo"
    // Carga del arreglo con el nombre de las sucursales
    sucursales[1]="Metrópolis"
    sucursales[2]="Barquicenter"
    sucursales[3]="Sambil"
    sucursales[4]="Trinitarias"
FinSubProceso
```

Este módulo es el que permite cargar la matriz con los montos de ventas, nótese la invocación a una función para hacer la validación de los valores ingresados. Obsérvese también que se están utilizando los arreglos para hacer más informativos al usuario, los mensajes que le guían en el proceso de ingresar los montos de venta:

```

Subproceso Cargar_Matriz(Dias,Sucursales,Vtas_Sem Por Referencia)
    // Definición de variables locales
    Para f=1 Hasta 7
        MensajeDia="Ingrese los montos vendidos el día "+Dias[f]
        Mostrar MensajeDia
        Para c=1 Hasta 4
            MensajeSuc="Monto vendido en la sucursal "+Sucursales[c]
            Vtas_Sem[f,c]=Func_LeerPos(MensajeSuc)
        FinPara
        Mostrar ""
    FinPara
FinSubProceso

```

Dentro del módulo de cargar la matriz, se invoca a esta función (vista con anterioridad) para asegurarse que los montos de ventas sean positivos:

```

Funcion numPos=Func_LeerPos(Mensaje)
    Repetir
        Mostrar Mensaje Sin Saltar
        Leer num
        Hasta Que num>=0
        numPos=num
    FinFuncion

```

Una vez se encuentra cargada toda la información necesaria en las estructuras de almacenamiento, se invoca a esta función desde el cuerpo principal, con el objeto de calcular el promedio general de ventas:

```

Funcion Prom_Vtas=Func_Det_PromVtas(Vtas_Sem)
    acum_vtas=0
    Para f=1 Hasta 7
        Para c=1 Hasta 4
            acum_vtas=acum_vtas+Vtas_Sem[f,c]
        FinPara
    FinPara
    prom_vtas=acum_vtas/28
FinFuncion

```

Después de determinar el promedio de ventas se invoca a este módulo desde el cuerpo principal para desplegar la información. Obsérvese que se accede a los arreglos y la matriz para generar información que es más fácil de interpretar para el usuario:

```
SubProceso MostrarInfo Vtas SupProm(prom vtas,Dias,Sucursales,Vtas_Sem)
    Mostrar "El promedio general de ventas fue: " prom_vtas " Bs."
    Mostrar ""
    Mostrar "Estas fueron las ventas que superaron al promedio... "
    Mostrar ""
    Para f=1 hasta 7
        Mostrar "El día " Dias[f] Sin Saltar
        Mostrar " las siguientes sucursales superaron el promedio: "
        Para c=1 hasta 4
            Si Vtas_Sem[f,c]>prom_vtas
                entonces
                    Mostrar "La sucursal " Sucursales[c] " vendió " Vtas_Sem[f,c] " Bs."
            FinSi
        FinPara
        Mostrar ""
    FinPara
FinSubProceso
```



Capítulo 13. OPERACIONES CON ARREGLOS Y MATRICES 2

13.1.- Matrices y Determinación de Mayores y Menores

Usando matrices donde se almacenan valores numéricos, se puede también:

- Conocer cuál es el valor mayor de todos los elementos registrados en la matriz.
- Conocer cuál es el valor menor de todos los elementos registrados en la matriz.

Una vez que se conozcan el valor mayor y menor, se puede mostrar información sobre las entidades que obtuvieron dichos valores, para lo cual es necesario almacenar los nombres de esas entidades en un arreglo.

Esto se desarrolla en este capítulo, siguiendo con el ejemplo de la matriz con las ventas realizadas en las sucursales.

En este tema se presentan 2 formas alternativas de lograr el objetivo: determinar los valores mayor y menor, para luego brindar información utilizandolos como punto de referencia.

En la 1era secuencia de bloques de código no se implementó reusabilidad y en la 2da secuencia sí.

En esencia en las 2 versiones, se utiliza el mismo módulo para determinar el elemento mayor y el menor, sólo cambian los subprocesos implicados en mostrar la información relacionada con dichos valores.

13.2.- Determinación de Mayores y Menores

Este módulo es común a ambas implementaciones, en él se determinan los elementos mayor y menor a través del recorrido de la matriz:

```

Subproceso Det_MayYMen(Vtas_Sem,Mto_May Por Referencia,Mto_Men Por Referencia)
    Para f=1 hasta 7
        Para c=1 hasta 4
            Si Vtas_Sem[f,c]>Mto_May
                entonces Mto_May=Vtas_Sem[f,c]
            FinSi
            Si Vtas_Sem[f,c]<Mto_Men
                entonces Mto_Men=Vtas_Sem[f,c]
            FinSi
        FinPara
    FinPara
FinSubProceso

```

Esta es la 1era versión del cuerpo principal, sin implementar reusabilidad:

```

Algoritmo May_Y_Men_Vtas
    Dimension Dias[7],Sucursales[4]
    Dimension Vtas_Sem[7,4]
    Definir prom_vtas como Real
    Definir Mto_May,Mto_Men como Real

    Mto_May=0
    Mto_Men=100000000
    Cargar_Arreglos(Dias,Sucursales)
    Cargar_Matriz(Dias,Sucursales,Vtas_Sem)
    Det_MayYMen(Vtas_Sem,Mto_May,Mto_Men)
    Mostrar ""
    MostrarInfo_May(Mto_May,Dias,Sucursales,Vtas_Sem)
    Mostrar ""
    MostrarInfo_Men(Mto_Men,Dias,Sucursales,Vtas_Sem)
    Mostrar ""
FinAlgoritmo

```

Este módulo muestra información relativa al valor mayor, y lista las sucursales que cuyas ventas se aproximan a él por un margen del 10%:

```

Subproceso MostrarInfo_May(Mto_May,Dias,Sucursales,Vtas_Sem)
    Mostrar " - - - - INFORMACIÓN RELATIVA A LA MAYOR VENTA - - - - "
    Mostrar "El monto de la mayor venta fue " Mto_May " Bs."
    CotaInfMayVta=Mto_May-(Mto_May*0.10) // Cota inferior a la mayor venta
    CotaSupMayVta=Mto_May+(Mto_May*0.10) // Cota superior a la mayor venta
    Mostrar "Rango de referencia: " CotaInfMayVta " Bs. a " CotaSupMayVta " Bs."
    Mostrar ""
    Mostrar "Información sobre las sucursales que rondaron la mayor venta: "
    Para c=1 hasta 4
        Para f=1 hasta 7
            Si Vtas_Sem[f,c]=Mto_May
                entonces
                    Mostrar "El día " Dias[f] " la sucursal " Sucursales[c] Sin Saltar
                    Mostrar " registró la mayor venta."
                sino Si (Vtas_Sem[f,c]>=CotaInfMayVta Y Vtas_Sem[f,c]<=CotaSupMayVta)
                    entonces
                        Mostrar "El día " Dias[f] " la sucursal " Sucursales[c] Sin Saltar
                        Mostrar " se aproximó a la mayor venta: " Vtas_Sem[f,c] " Bs."
                    FinSi
                FinSi
            FinPara
        FinPara
    FinSubProceso

```

Este módulo muestra información relativa al valor menor, y lista las sucursales que cuyas ventas se aproximan a él por un margen del 10%:

```

Subproceso MostrarInfo_Men(Mto_Men,Dias,Sucursales,Vtas_Sem)
    Mostrar " - - - - INFORMACIÓN RELATIVA A LA MENOR VENTA - - - - "
    Mostrar "El monto de la menor venta fue " Mto_Men " Bs."
    CotaInfMenVta=Mto_Men-(Mto_Men*0.10) // Cota inferior a la menor venta
    CotaSupMenVta=Mto_Men+(Mto_Men*0.10) // Cota superior a la menor venta
    Mostrar "Rango de referencia: " CotaInfMenVta " Bs. a " CotaSupMenVta " Bs."
    Mostrar ""
    Mostrar "Información sobre las sucursales que rondaron la menor venta: "
    Para c=1 hasta 4
        Para f=1 hasta 7
            Si Vtas_Sem[f,c]=Mto_Men
                entonces
                    Mostrar "El día " Dias[f] " la sucursal " Sucursales[c] Sin Saltar
                    Mostrar " registró la menor venta."
                sino Si (Vtas_Sem[f,c]>=CotaInfMenVta Y Vtas_Sem[f,c]<=CotaSupMenVta)
                    entonces
                        Mostrar "El día " Dias[f] " la sucursal " Sucursales[c] Sin Saltar
                        Mostrar " se aproximó a la menor venta: " Vtas_Sem[f,c] " Bs."
                    FinSi
                FinSi
            FinPara
        FinPara
    FinSubProceso

```

Este es la 2da versión del cuerpo principal, implementando reusabilidad:

```

Algoritmo May_Y_Men_Vtas_V2
    Dimension Dias[7],Sucursales[4]
    Dimension Vtas_Sem[7,4]
    Definir prom_vtas como Real
    Definir Mto_May,Mto_Men como Real

    Mto_May=0
    Mto_Men=100000000
    Cargar_Arreglos(Dias,Sucursales)
    Cargar_Matriz(Dias,Sucursales,Vtas_Sem)
    Det_MayYMen(Vtas_Sem,Mto_May,Mto_Men)
    Mostrar ""
    MostrarInfo_May_O_Menor(Mto_May,>,Dias,Sucursales,Vtas_Sem)
    Mostrar ""
    MostrarInfo_May_O_Menor(Mto_Men,<,Dias,Sucursales,Vtas_Sem)
    Mostrar ""
FinAlgoritmo

```

Este módulo es reusable y está diseñado tanto para mostrar la información relativa a los mayores como de los menores:

```

Subproceso MostrarInfo_May_O_Menor(Mto_Ref,TipoMto,Dias,Sucursales,Vtas_Sem)
    Si TipoMto=>
        entonces relacion="mayor"
        sino relacion="menor"
    FinSi
    TxtTit=Mayusculas(relacion)
    Titulo=" - - - - INFORMACIÓN RELATIVA A LA "+TxtTit+" VENTA - - - - "
    Mostrar Titulo
    MtoRefTxt=ConvertirATexto(Mto_Ref)
    SubTit="El monto de la "+relacion+" venta fue "+MtoRefTxt+ " Bs."
    Mostrar SubTit
    CotaInfVta=Mto_Ref-(Mto_Ref*0.10) // Cota inferior a la menor venta
    CotaSupVta=Mto_Ref+(Mto_Ref*0.10) // Cota superior a la menor venta
    Mostrar "Rango de referencia: " CotaInfVta " Bs. a " CotaSupVta " Bs."
    Mostrar ""
    SubTit2="Detalles sobre las sucursales que rondaron la "+relacion+ " venta: "
    Mostrar SubTit2
    Mostrar_Detalles(Mto_Ref,relacion,CotaInfVta,CotaSupVta,Dias,Sucursales,Vtas_Sem)
FinSubProceso

```

Este módulo es invocado por el módulo anterior, su objetivo es mostrar la información detallada en relación a valor menor o mayor según corresponda:

```
Subproceso Mostrar_Detalles(Mto_Ref,relacion,CotaInfVta,CotaSupVta,Dias,Sucursales,Vtas_Sem)
    Para c=1 hasta 4
        Para f=1 hasta 7
            Si Vtas_Sem[f,c]=Mto_Ref
                entonces
                    Mostrar "El día " Dias[f] " la sucursal " Sucursales[c] Sin Saltar
                    Mostrar " registró la " relacion " venta."
            sino Si (Vtas_Sem[f,c]>=CotaInfVta Y Vtas_Sem[f,c]<=CotaSupVta)
                entonces
                    Mostrar "El día " Dias[f] " la sucursal " Sucursales[c] Sin Saltar
                    Mostrar " se aproximó a la " relacion " venta: " Vtas_Sem[f,c] " Bs."
            FinSi
        FinSi
    FinPara
FinSubProceso
```



Capítulo 14. RECURSIVIDAD

14.1.- Principios Sobre la Recursividad

La RECURSIVIDAD es una técnica de programación que se utiliza para realizar una llamada a una función desde ella misma, de allí su nombre. Dicha llamada se conoce como llamada recursiva o recurrente.

Generalmente, si la primera llamada a la función se plantea sobre un problema de tamaño u orden N , cada nueva ejecución recurrente del mismo se planteará sobre problemas, de igual naturaleza que el original, pero de un tamaño menor que N .

De esta forma, al ir reduciendo progresivamente la complejidad del problema a resolver, llegará un momento en que su resolución sea más o menos trivial (o, al menos, suficientemente manejable como para resolverlo de forma no recursiva). En esa situación diremos que estamos ante un "caso base" de la recursividad.

Es frecuente que los algoritmos recurrentes sean más ineficientes en tiempo que los iterativos, aunque suelen ser mucho más breves en cantidad de líneas de código.

La mayoría de los problemas pueden resolverse de una manera directa usando métodos no recursivos. Sin embargo, otros pueden resolverse de una manera más lógica y elegante mediante la recursión.

Las propiedades de las definiciones de funciones recursivas son las siguientes:

- No debe generar una secuencia infinita de llamadas así mismo, dicho de otro modo, ha de existir al menos un caso base.
- Una función recursiva F debe definirse en términos que no impliquen a F en un argumento o grupo de argumentos.

- Debe existir una "salida" de la secuencia de llamadas recursivas. Esto es lo que se obtiene en función a través de la variable de retorno.
- Cada llamada recurrente se debería definir sobre un problema de menor complejidad (algo más fácil de resolver) del que acaba de ser resuelto con la llamada recursiva anterior.

Para ilustrar la recursividad, se utiliza como apoyo el siguiente video sobre el juego de las "Torres de Hanoi", que es un juego de lógica donde se tienen 3 torres y se tiene que mover los discos que hay en la primera torre a la tercera, siguiendo siempre algunas reglas, como que sólo se puede mover los discos de uno en uno y que no se puede poner un disco sobre otro más pequeño que él.

El video se encuentra accesible en la siguiente dirección:

<https://www.youtube.com/watch?v=qenY3AzoR1Q>

14.2.- Ejemplo de Función Factorial

Como algoritmo, el ejemplo más utilizado para entender la recursividad, por su fácil comprensión es el cálculo de números factoriales. El factorial de un número N representa el número de formas distintas de ordenar N objetos distintos (elementos sin repetición). Matemáticamente, para cualquier entero positivo N, su factorial (que se expresa como $N!$) es el producto (multiplicación) de todos los enteros menores a él:

Dado un número $n \in \mathbb{N}$, definimos...

$$n! = 1 \cdot 2 \cdot 3 \dots (n-2) \cdot (n-1) \cdot n$$

Notación de Factorial

Vista la definición apréciese los siguientes resultados de factoriales de 1 a 6:

$1! = 1$
 $2! = 2 \times 1! = 1 \times 2 = 2$
 $3! = 3 \times 2! = 1 \times 2 \times 3 = 6$
 $4! = 4 \times 3! = 1 \times 2 \times 3 \times 4 = 24$
 $5! = 5 \times 4! = 1 \times 2 \times 3 \times 5 = 120$
 $6! = 6 \times 5! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$

La única excepción a esta regla es que el factorial de 0, por definición es 1.

Ahora, repasando atentamente, se puede ver que el factorial de cada número incluye el factorial de todos los números anteriores a él.

Generalizando: Para cualquier entero N mayor a 1, se cumple que el factorial de N es igual al factorial del número anterior a N multiplicado por N. La fórmula para el cálculo es: $N! = N \times (N-1)!$

En el siguiente algoritmo se implementa la función recursiva para el cálculo del factorial de un número ingresado:

```

Funcion factorial<-Calc_Factorial(n)
    Segun n
        | 0:factorial=1
        | De otro modo:
        |   factorial=n*Calc_Factorial(n-1)
    FinSegun
FinFuncion

Algoritmo Factorial_Recursivo
    Definir Mensaje Como Caracter
    Definir n como Entero

    Mensaje="Ingrese el número para calcularle el factorial"
    n=Func LeerPos(Mensaje)
    Mostrar "El factorial de " n " es " Calc_Factorial(n)
FinAlgoritmo

```

Si se llama calc_factorial para obtener el factorial de 5 el proceso es el siguiente:

calc_factorial(5) devuelve 5 * calc_factorial(4) entra en 1º recursión
calc_factorial(4) devuelve 4 * calc_factorial(3) entra en 2º recursión
calc_factorial(3) devuelve 3 * calc_factorial(2) entra en 3º recursión
calc_factorial(2) devuelve 2 * calc_factorial(1) entra en 4º recursión
calc_factorial(1) devuelve 1 * calc_factorial(0) entra en 5º recursión
calc_factorial(0) y ...

termina la 5º recursión que devuelve 1
termina la 4º recursión que devuelve 2*1
termina la 3º recursión que devuelve 3*2*1
termina la 2º recursión que devuelve 4*3*2*1
termina la 1º recursión que devuelve 5*4*3*2*1

