

Lógica de Programación. Nivel III

Mayo, 2018



Objetivos del nivel

- Aprender a usar instrucciones repetitivas (ciclos)
- Entender el uso de contadores y acumuladores
- Aprender a manejar arreglos

Prerrequisitos del nivel

- Lógica de Programación Nivel II

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADIF1). Para obtener más información sobre este u otros cursos visite nuestra 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 2018. Todos los derechos reservados.

ACADEMIA DE SOFTWARE



Contenido del nivel

Capítulo 1. Estructuras Selectivas. Parte 6

- 1.1.- Según Sea.
- 1.2.- Otro.
- 1.3.- Según Sea Vs. si Anidado.

Capítulo 2. Contadores y Acumuladores

- 2.1.- Concepto de Contador.
- 2.2.- Contador General y Específico.
- 2.3.- Concepto de Acumulador.
- 2.4.- Acumulador General y Específico.

Capítulo 3. Estructuras Repetitivas. Parte 3

- 3.1.- Repetir-hasta.
- 3.2.- ¿Desea Continuar?.

Capítulo 4. Algoritmos Con Menús

- 4.1.- Menú y Opciones.
- 4.2.- Incorporación Del Según.
- 4.3.- Incorporando el Ciclo.

Capítulo 5. Promedios y Porcentajes

- 5.1.- Promedio.
- 5.2.- Promedios Específicos.
- 5.3.- Porcentajes.

Capítulo 6. Mayores y Menores

- 6.1.- Conceptos.
- 6.2.- Determinación de Mayores.
- 6.3.- Determinación de Menores.

Capítulo 7. Estructuras Repetitivas. Parte 4

- 7.1.- Estructura Mientras.
- 7.2.- Condiciones Compuestas.
- 7.3.- Comparación Con Otros Ciclos.

Capítulo 8. Estructuras Repetitivas Dobles. Parte 1

- 8.1.- Estructura Repetitiva Anidada.
- 8.2.- Combinación de Ciclos Anidados.
- 8.3.- Entradas y Salidas.

Capítulo 9. Estructuras Repetitivas Dobles. Parte 2

- 9.1.- Acumuladores y Contadores.
- 9.2.- Promedios.

Capítulo 10. Arreglos. Parte 1

- 10.1.- Concepto.
- 10.2.- Declaración de un Arreglo.
- 10.3.- Índice de Los Arreglos.

Capítulo 11. Arreglos. Parte 2

- 11.1.- Lectura de Arreglos.
- 11.2.- Mostrar Arreglos.

Capítulo 12. Arreglos Paralelos

- 12.1.- Concepto.
- 12.2.- Lectura de Datos.
- 12.3.- Mostrar Datos.

Capítulo 13. Aprovechando los Arreglos

- 13.1.- Promedio.
- 13.2.- Determinar Mayores.
- 13.3.- Determinar Menores.

Capítulo 14. Búsqueda en Arreglos

- 14.1.- Buscar un Elemento único.
- 14.2.- Buscar un Elemento Repetido.

Capítulo 15. Ordenamiento

- 15.1.- Introducción.
- 15.2.- Algoritmo Burbuja.



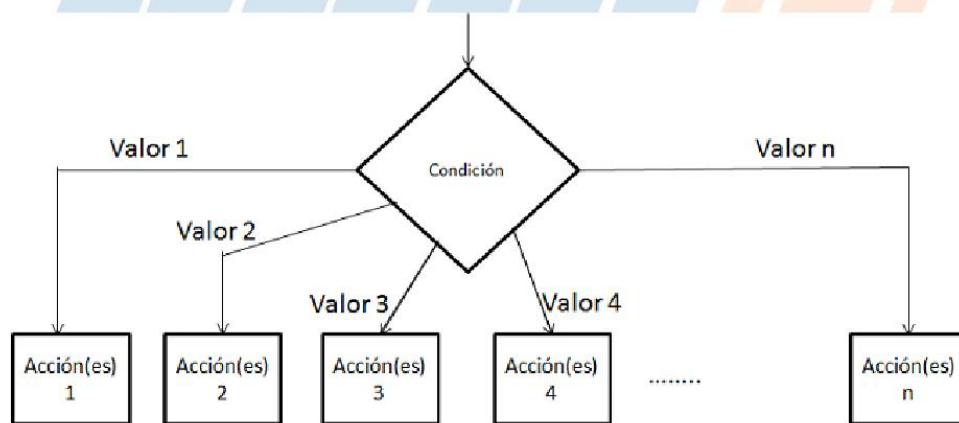
Capítulo 1. ESTRUCTURAS SELECTIVAS. PARTE 6

1.1.- Según Sea

En la estructura selectiva de alternativa múltiple se evalúa una condición o expresión que puede tomar “n” valores distintos. Según sea el valor de la expresión de un instante dado, se ejecutarán las acciones correspondientes a ese valor. En realidad equivale a un conjunto de condicionales anidadas. En muchos lenguajes de programación se conocen como sentencias Case o Switch. Es una sentencia de la forma:

```
Según sea <expresión>
    <valor1>: <acciones 1>
    <valor2>: <acciones 2>
    [<otro>: <acciones n> ]
fin segun
```

El siguiente es el Diagrama de flujo de a estructura selectiva Segun Sea:

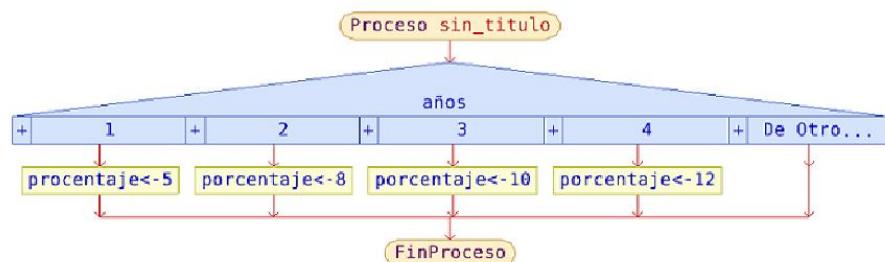


Dependiendo de los años de antiguedad de un vehículo se le pagará una prima a su dueño. Si el vehículo tiene 1 año de antiguedad se le paga 5%, si tiene 2 años se le paga 8%, si tiene 3 años 10%, si tiene 4 años 12%. El pseudocódigo de este problema es el siguiente:

segun sea años

- 1: porc = 5
- 2: porc = 8
- 3: porc =10
- 4: porc =12

fin segun



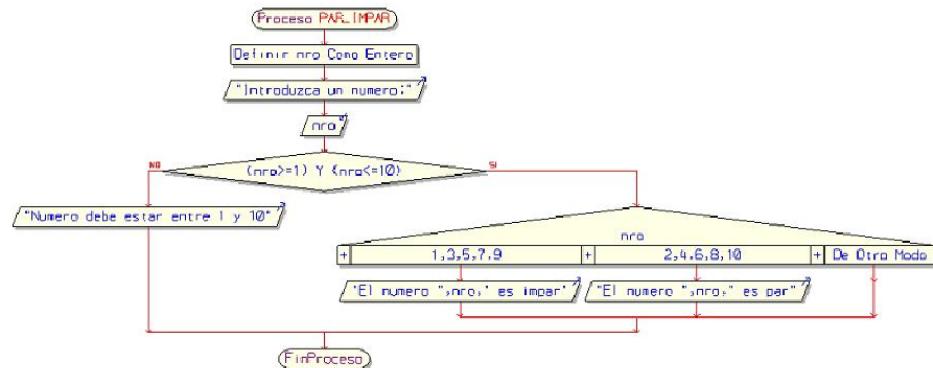
Ahora vea el pseudocódigo del siguiente ejemplo:

Se desea leer por teclado un número entre 1 y 10 y se desea visualizar si el número es par o impar

```

1  Proceso PAR_IMPAR
2
3      definir nro Como Entero
4
5      mostrar "Introduzca un numero:"
6      leer nro
7
8      si (nro>=1) y (nro<=10) Entonces
9          Segun nro Hacer
10             1,3,5,7,9:
11                 escribir "El numero " nro " es impar"
12             2,4,6,8,10:
13                 escribir "El numero " nro " es par"
14             Fin Segun
15         sino
16             escribir "Numero debe estar entre 1 y 10"
17
18     FinSi
19
20
21  FinProceso
  
```

Flujograma del ejemplo anterior



1.2.- Otro

Las acciones asociadas a la opción 'Otro:' se ejecutan cuando la expresión no toma ninguno de los valores que aparecen antes. En algunos lenguajes de programación se le conoce como alternativa 'Otherwise', 'Else' o 'Default'.

El valor con el que se compara la expresión es dependiente de los lenguajes de programación. Por lo general, se espera un tipo de dato determinístico y con valores secuenciales, tales como los enteros y caracteres. En general, ese valor puede ser un valor constante, un rango de valores determinístico o incluso otra condición. La sentencia 'según sea' (case) siempre equivale a una anidación de condicionales, pero lo contrario no es cierto.

Ahora veamos el ejemplo anterior, agregando la opción OTRO:

Dependiendo de los años de antigüedad de un vehículo se le pagará una prima a su dueño. Si el vehículo tiene 1 año de antigüedad se le paga 5%, si tiene 2 años se le paga 8%, si tiene 3 años 10%, si tiene 4 años 12% y si tiene cinco años o más la prima es de 15%. El pseudocódigo de este problema es el siguiente:

según sea años

1: porc = 5

2: porc = 8

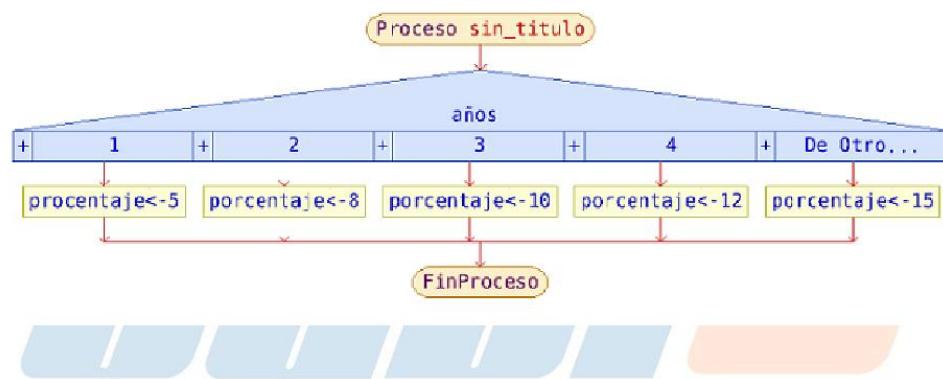
3: porc =10

4: porc =12

otro:

porc = 15

fin según



1.3.- Según Sea Vs. si Anidado

Véase como el ejemplo donde dependiendo del número que se ingrese indica un día de la semana, se puede hacer en ambas estructuras:

```
Segun nroDia  Hacer
    1: escribir "Lunes"
    2: escribir "Martes"
    3: escribir "Miercoles"
    4: escribir "Jueves"
    5: escribir "Viernes"
    6,7: escribir "Fin se semana"
De otro modo:
    escribir "error, la semana tiene 7 dias"
FinSegun

si nroDia=1 Entonces
    escribir "Lunes"
sino
    si nroDia=2 entonces
        escribir "Martes"
    Sino
        si nroDia=3 entonces
            escribir "Miercoles"
        Sino
            si nroDia=4 entonces
                escribir "Jueves"
            Sino
                si nroDia=5 entonces
                    escribir "Viernes"
                Sino
                    si nroDia=6 o nroDia=7 entonces
                        escribir "Fin de Semana"
                    Sino
                        Escribir "Error, la semana tiene 7 dias"
                FinSi
            FinSi
        FinSi
    FinSi
FinSi
```



ACADEMIA DE SOFTWARE

Capítulo 2. CONTADORES Y ACUMULADORES

2.1.- Concepto de Contador

Un CONTADOR es una variable numérica entera, cuyo valor se va incrementando de uno en uno. El objetivo principal de un contador, es como su nombre lo indica, llevar la cuenta de la ocurrencia de algo durante la ejecución del algoritmo.

Por ejemplo: Si se van a leer datos de alumnos como nombre, sexo, edad, podemos tener los siguientes contadores:

- contador de mujeres.
- contador de hombres.
- contador de personas mayores de 25 años.
- contador de mujeres menores de edad, etc.

Cada contador siempre debe inicializarse en cero (0) antes de iniciar el ciclo repetitivo dentro del cual éste se debe incrementar. La inicialización tiene el propósito de asegurarse que su valor al final sea correcto. La sintaxis general de un contador es la siguiente:

ACADEMIA DE SOFTWARE

Un contador puede incrementarse o no según el caso. Ej:

```
contador1 = 0      // Inicialización del contador1
contador2 = 0      // Inicialización del contador2
Para i = 1 hasta 10
    contador1 = contador1 + 1          // Se incrementa siempre
    Si < condición >
        Entonces contador2 = contador2 + 1 // No se incrementa siempre
    FinSi
FinPara
```

Mostrar 'El contador 1 dió: ' contador1

Mostrar 'El contador 2 dió: ' contador2

2.2.- Contador General y Específico

Un CONTADOR GENERAL se incrementa en todas las iteraciones del ciclo repetitivo, su variación no depende de ninguna circunstancia. En el ejemplo se aprecia la inicialización, incremento y despliegue de la variable contador:

```

1 Algoritmo Contar_Vueltas
2   Definir i Como Entero
3   Definir contador Como Entero
4
5   contador=0 // se inicializa
6
7   Para i=1 hasta 10 con paso 2 Hacer
8     Mostrar i
9     // se actualiza
10    contador=contador+1
11  FinPara
12  // se utiliza
13  Mostrar "El ciclo dió " contador " vueltas"
14 FinAlgoritmo

```

```

*** Ejecución Iniciada.
1
3
5
7
9
El ciclo dió 5 vueltas
*** Ejecución Finalizada

```

Un CONTADOR ESPECÍFICO es aquél que se incrementa sólo cuando se cumple alguna condición. Esto implica, que se debe evaluar una condición previamente a la actualización del contador.

El siguiente es un ejemplo de 2 contadores específicos:

- positivos : se incrementará cuando un valor sea positivo
- negativos: otro cuando el número sea negativo

Premisa: los números ingresados deben ser distintos a cero(0).

```

1 Algoritmo Pos_Y_Neg
2   Definir num,i Como Entero
3   Definir positivos,negativos Como Entero
4
5   positivos=0
6   negativos=0
7   Para i=1 hasta 10 Hacer
8     Mostrar "Ingrese un número entero:"
9     Leer num
10    Si num>0
11      entonces positivos=positivos+1
12      sino negativos=negativos+1
13    FinSi
14  FinPara
15  Mostrar "Positivos: " positivos
16  Mostrar "Negativos: " negativos
17 FinAlgoritmo

```

En este otro ejemplo se leen 10 números y se determina cuántos de ellos son pares y cuántos son impares:

```

1 Algoritmo Par_Y_Impar
2   Definir num,i Como Entero
3   Definir pares,impares Como Entero
4
5   pares=0
6   impares=0
7   Para i=1 hasta 10 Hacer
8     Mostrar "Ingrese un número entero:"
9     Leer num
10    Si (num mod 2) = 0
11      entonces pares=pares+1
12      sino impares=impares+1
13    FinSi
14  FinPara
15  Mostrar "Pares: " pares
16  Mostrar "Impares: " impares
17 FinAlgoritmo

```

2.3.- Concepto de Acumulador

Un ACUMULADOR es una variable numérica entera o real, cuyo valor se va incrementando en un valor variable. Por ejemplo, si se van a leer datos de alumnos como nombre, sexo y nota, podemos tener los siguientes acumuladores:

- acumulador de notas de las mujeres y
- acumulador de notas de los hombres.

La sintaxis general de un acumulador es la siguiente:

acumulador = acumulador + variable

Donde "variable" contiene el valor que se necesita acumular. Normalmente esta variable es un dato de entrada o de salida que se calcula o se lee dentro de un ciclo. Los acumuladores se pueden determinar independientemente del tipo de ciclo.

Los acumuladores al igual que los contadores, siempre deben inicializarse en 0 (cero) antes de iniciar el ciclo repetitivo, para asegurar que su valor sea correcto y deben acumularse dentro del ciclo. Típicamente el valor resultante en un acumulador se usa se utiliza fuera del ciclo.

Un acumulador puede incrementarse o no según el caso. Ej:

```
acumulador1 = 0      // Inicialización del contador1
acumulador2 = 0      // Inicialización del contador2

Para i = 1 hasta 10
    // Se incrementa siempre
    acumulador1 = acumulador1 + variable

    Si < condición > Entonces
        // No se incrementa siempre
        acumulador2 = acumulador2 + variable  FinSi
    FinPara
    Mostrar 'El acumulador 1 dió: ' acumulador1
    Mostrar 'El acumulador 2 dió: ' acumulador2
```

2.4.- Acumulador General y Específico

Un ACUMULADOR GENERAL se incrementa en todas las iteraciones del ciclo repetitivo, su variación no depende de ninguna circunstancia. En el ejemplo se usa un ciclo "para" y se muestra cómo acumular el sueldo de 100 empleados:

```

1 Algoritmo Acumular_Sueldos
2   Definir i Como Entero
3   Definir acum_sueldos,sueldo como Entero
4   Definir nombre Como Caracter
5
6   acum_sueldos=0
7   Para i=1 hasta 100 Hacer
8     Limpiar Pantalla
9     Mostrar "Ingrese el nombre y el sueldo del " i "º empleado"
10    Leer nombre,sueldo
11    Mostrar "El sueldo de " nombre " incrementará el acumulado en : " sueldo
12    Esperar 2 Segundos
13    acum_sueldos=acum_sueldos+sueldo
14  FinPara
15  Mostrar "El acumulado de todos los sueldos dió: " acum_sueldos
16 FinAlgoritmo

```

Un ACUMULADOR ESPECÍFICO es aquél que se incrementa sólo cuando se cumple alguna condición. Esto implica, que se debe evaluar una condición previamente a la actualización del mismo.

```

1 Algoritmo Acumular_Sueldos
2   Definir i Como Entero
3   Definir acum_sueldos_hom,acum_sueldos_muj,sueldo como Real
4   Definir nombre,sexo Como Caracter
5
6   acum_sueldos_hom=0
7   acum_sueldos_muj=0
8   Para i=1 hasta 100 Hacer
9     Limpiar Pantalla
10    Mostrar "Ingrese el nombre, sexo (F/M), y el sueldo del " i "º empleado"
11    Leer nombre,sexo,sueldo
12    Segun Mayusculas(Sexo)
13      "F": acum_sueldos_muj=acum_sueldos_muj+sueldo
14      "M": acum_sueldos_hom=acum_sueldos_hom+sueldo
15    FinSegun
16    Mostrar "El sueldo de " nombre " incrementará el acumulador respectivo en: " sueldo
17    Esperar 2 Segundos
18  FinPara
19  Mostrar "El acumulado de sueldos de las mujeres dió: " acum_sueldos_muj
20  Mostrar "El acumulado de sueldos de los hombres dió: " acum_sueldos_hom
21 FinAlgoritmo

```

Capítulo 3. ESTRUCTURAS REPETITIVAS. PARTE 3

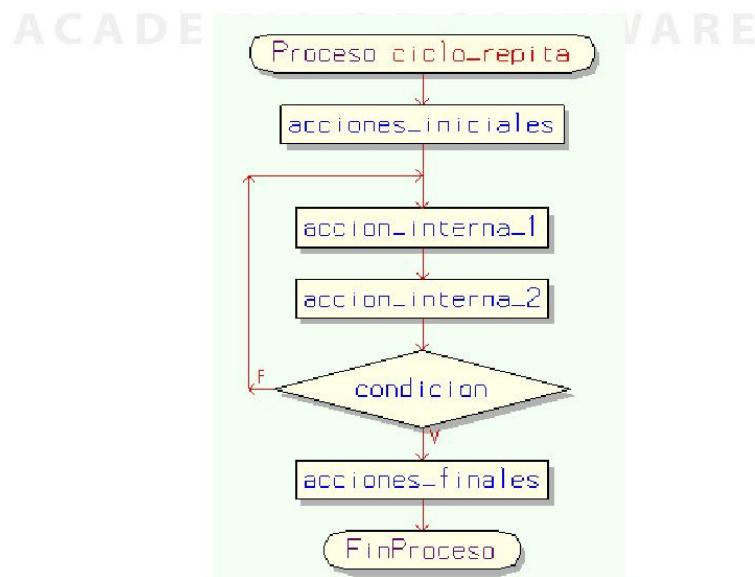
3.1.- Repetir-hasta

La estructura iterativa "Repetir-Hasta" se utiliza cuando no se conoce de antemano la cantidad de iteraciones que ejecutara un ciclo. En vez de utilizar un contador, se utiliza una expresión lógica (como las expresiones usadas en las condicionales "si").

En la estructura repetitiva "Repetir-Hasta", la condición del ciclo se evalúa después de realizar la primera iteración del bucle. Este bucle se repite mientras la condición evaluada al final se mantenga Falsa. Al cumplirse la condición y volverse Verdadera, se sale del bucle. De allí su nombre "Repetir Hasta" que se cumpla la condición.

Como la condición se evalúa al final, se pasa al menos una vez por el bucle. Así pues, cuando un bucle se tenga que ejecutar como mínimo una vez, se puede usar una estructura "Repetir-Hasta". La condición debe dejar de cumplirse en algún momento, de lo contrario se producirá en un ciclo infinito, lo cual sería un error lógico en la mayoría de los casos.

El flujograma general del ciclo "Repetir ... Hasta" se muestra a continuación:

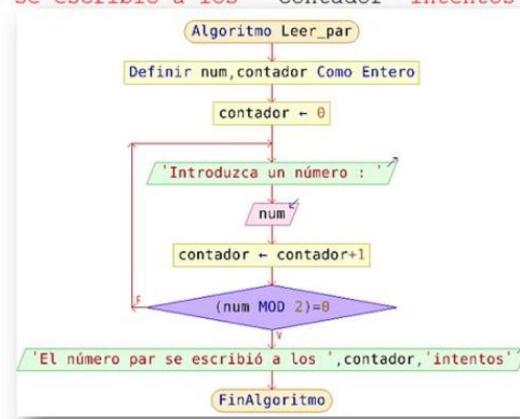


A continuación se muestra un ejemplo, donde el usuario tendrá que escribir números hasta que escriba un número par:

```

1 Algoritmo Leer_par
2   Definir num,contador como entero
3
4   contador=0
5
6   Repetir
7     Mostrar "Introduzca un número : " Sin Saltar
8     Leer num
9     contador=contador+1
10    Hasta Que (num mod 2) = 0
11
12   Mostrar "El número par se escribió a los ", contador, "intentos"
13 FinAlgoritmo

```



3.2.- ¿Desea Continuar?

Una de las formas más comunes de romper un ciclo es preguntándole al usuario si desea continuar procesando elementos. El ciclo "Repetir..Hasta" es uno de los más frecuentemente usados para este tipo de situaciones, debido a que se asume que por lo menos una vez se procesarán los datos y al final se le pregunta al usuario si desea procesar otra vez, y así sucesivamente. Para esto debemos usar una variable alfanumérica. El siguiente es un ejemplo:

```
1 Algoritmo Leer_par
2     Definir num,contador como entero
3     Definir respuesta Como Caracter
4
5     contador=0
6
7     Repetir
8         Mostrar "Introduzca un número : " Sin Saltar
9         Leer num
10        Mostrar "La mitad de " num " es: " num/2
11        contador=contador+1
12        Mostrar "¿Desea procesar otro número (S/N)?" Sin Saltar
13        Leer respuesta
14        Hasta Que Mayusculas(respuesta) = "N"
15
16        Mostrar "Se procesaron " contador "números"
17 FinAlgoritmo
```



Capítulo 4. ALGORITMOS CON MENÚS

4.1.- Menú y Opciones

En informática, un menú es una serie de opciones que el usuario puede elegir para realizar determinadas tareas.

En su forma más simple, los menús se presentan en forma de lista, el cual consta de unas salidas donde se muestran las opciones al usuario:

```
mostrar "MENÚ DE OPCIONES"
mostrar "1 - Ingresar Datos"
mostrar "2 - Realizar Cálculos"
mostrar "3 - Mostrar resultados"
mostrar "4 - Salir"
mostrar "Ingrese una opción..."
```

*** Ejecución Iniciada. ***
MENÚ DE OPCIONES
 1 - Ingresar Datos
 2 - Realizar Cálculos
 3 - Mostrar resultados
 4 - Salir
 Ingrese una opción...

Después de presentar el menú, debe requerírsela al usuario la opción de su preferencia, la cual se usará para determinar lo que ocurrirá a continuación:

```
mostrar "MENÚ DE OPCIONES"
mostrar "1 - Ingresar Datos"
mostrar "2 - Realizar Cálculos"
mostrar "3 - Mostrar resultados"
mostrar "4 - Salir"
mostrar "Ingrese una opción..."
```

*** Ejecución Iniciada. ***
MENÚ DE OPCIONES
 1 - Ingresar Datos
 2 - Realizar Cálculos
 3 - Mostrar resultados
 4 - Salir
 Ingrese una opción...
 >

4.2.- Incorporación Del Según

Una vez leída la opción de menú, se debe dirigir el control de flujo usando una estructura de alternativa de selectiva múltiple (SEGÚN). Por cada opción que implica

ejecutar acciones, se le debe pasar el control de ejecución a un módulo correspondiente.

Nótese que en el caso que sea ingresado una opción no válida, se le informa al usuario mediante un mensaje.

```
Segun opc_menu hacer
    1: Leer_Datos
    2: Realizar_Cálculos
    3: Mostrar_Salidas
    4: mostrar "Ud. escogió salir del programa"
    De Otro Modo:
        mostrar "Opción incorrecta"
FinSegun
```

4.3.- Incorporando el Ciclo

Luego de ejecutada la acción asociada, se debe presentar nuevamente el menú para que el usuario pueda seleccionar una nueva opción. El ciclo que típicamente se usa a esos efectos es el "Repetir..Hasta", el cual estará controlado por el valor que tome la variable evaluada en el Según. El bucle se rompe cuando el usuario selecciona la opción para salir.

La estructura completa queda de la siguiente manera:

```
Algoritmo Modulos_Y_Menu
    definir opc_menu Como Entero

    repetir
        mostrar "MENÚ DE OPCIONES"
        mostrar "1 - Ingresar Datos"
        mostrar "2 - Realizar Cálculos"
        mostrar "3 - Mostrar resultados"
        mostrar "4 - Salir"
        mostrar "Ingrese una opción..."
        Leer opc_menu
        Segun opc_menu hacer
            1: Leer_Datos
            2: Realizar_Cálculos
            3: Mostrar_Salidas
            4: mostrar "Ud. escogió salir del programa"
            De Otro Modo:
                mostrar "Opción incorrecta"
        FinSegun
    Hasta Que opc_menu = 4
FinAlgoritmo
```



ACADEMIA DE SOFTWARE

Capítulo 5. PROMEDIOS Y PORCENTAJES

5.1.- Promedio

Los promedios son cálculos muy frecuentemente solicitados en los problemas donde hay ciclos repetitivos (de cualquier tipo). Son muy frecuentes las preguntas como: calcule el promedio de notas de una sección, calcule el promedio de montos pagados por los clientes, etc.

Los promedios se calculan de forma general con la siguiente fórmula:

$$\text{promedio} = \text{acumuladorAlgo} / \text{contadorAlgo}$$

Tanto el acumulador como el contador se van actualizando dentro del ciclo, pero el promedio se calcula luego del fin del ciclo.

El siguiente es un ejemplo de un promedio, donde se solicitan 10 números al usuario y se imprimen su promedio al finalizar el ciclo. En este caso, la división se hace entre "10" porque es el número de datos procesados, que se conoce de antemano:

```
1 Proceso Promediar_Nums
2   Definir i Como Entero
3   Definir num,suma,promedio Como Real
4
5   Para i=1 hasta 10 hacer
6     Mostrar "Ingrese el " i "º número" Sin Saltar
7     Leer num
8
9     suma= suma + num // incremento del acumulador
10    FinPara
11
12  promedio= suma / 10 // cálculo del promedio
13  Mostrar "El promedio de los 10 números fue: " promedio
14 FinProceso
```

Si el número de vueltas del ciclo es variable, por ejemplo, porque el ciclo se rompe cuando el usuario responde que no desea continuar o selecciona la opción "salir" (en el caso de los menús), se debe usar una variable para dividir al acumulador. Por ejemplo:

```
1  Proceso sin_titulo
2      definir acum_sueldos,sueldo,promedio como real
3      definir i como numero
4      definir nombre Como Caracter
5
6      acum_sueldos = 0
7      contador = 0
8      repetir
9          Mostrar "Ingrese el nombre :"
10         Leer nombre
11         Mostrar "Ingrese el sueldo :"
12         Leer sueldo
13
14         contador = contador + 1
15         acum_sueldos=acum_sueldos + sueldo
16         mostrar "Desea procesar otro numero ?"
17         leer respuesta
18         hasta que Mayusculas(respuesta) == "NO"
19         promedio = acum_sueldos / contador
20
21         Mostrar "el promedio de los sueldos es :" promedio
22 FinProceso
```

5.2.- Promedios Específicos

Al igual que con los contadores específicos, en ocasiones se necesita calcular el promedio de un conjunto dado de valores. Aquí se debe tener especial cuidado debido a que es posible que el contador nunca se incremente, lo que podría generar una división entre 0. Por ejemplo, se necesitan calcular el promedio de precios de los pantalones tipo 1:

```
1 Proceso sin_titulo
2     definir acum_precio_tipo1,precio,promedio como real
3     definir tipo como numero
4
5     acum_precio_tipo1 = 0
6     contador = 0
7     repetir
8         Mostrar "Ingrese el tipo de pantalon :"
9         Leer tipo
10        Mostrar "Ingrese el precio :"
11        Leer precio
12
13        si tipo==1 Entonces
14            contador = contador + 1
15            acum_precio_tipo1=acum_precio_tipo1 + precio
16        FinSi
17
18        mostrar "Desea procesar otro pantalon ?"
19        leer respuesta
20        hasta que Mayusculas(respuesta) == "NO"
21        // se calcula el promedio
22        si contador>0 entonces
23            promedio = acum_precio_tipo1 / contador
24        Sino
25            promedio=0
26        fin si
27
28        Mostrar "el promedio de los sueldos es :" promedio
29 FinProceso
```

5.3.- Porcentajes

Los porcentajes son similares a los promedios. Se utilizan para determinar la relación que hay entre un contador específico y un contador general. Por ejemplo, calcular el porcentaje de aprobados, el porcentaje de clientes que compró más de 3 artículos, el porcentaje de hombres, etc. La fórmula general de los porcentajes es:

$$\text{porcentaje} = \text{contadorEspecifico} / \text{contadorGeneral} * 100$$

De igual forma que con los promedios, los contadores se actualizan dentro del ciclo repetitivo y se calcula luego de terminado el ciclo. En caso de que el "contadorEspecifico" sea 0, el porcentaje resultará 0 (0 entre cualquier número resulta

0), pero si el "contadorGeneral" es 0, no se debe realizar la división, porque a que ocurriría un error de ejecución debido a que la división entre 0 no existe.

Por ejemplo, si necesita calcular el porcentaje de aprobados de una sección, se debe primero hacer un contador de todos los alumnos aprobados, el cual se va incrementando dentro del ciclo y luego se divide entre la cantidad de alumnos. El código de este ejemplo es el siguiente:

```
1 Proceso sin_titulo
2     definir contaprobados, i, nota como numero
3     definir porcaprobados Como Real
4
5     contaprobados=0
6     para i=1 hasta 20 hacer
7         mostrar "ingrese la nota "
8         leer nota
9         si nota > 10 entonces
10            contaprobados= contaprobados + 1
11        FinSi
12    fin para
13    porcaprobados = contaprobados / 20 * 100
14    mostrar "el porcentaje de aprobados es " porcaprobados
15 FinProceso
```

ACADEMIA DE SOFTWARE

En ocasiones se necesitan calcular porcentajes de contadores específicos. Por ejemplo, el porcentaje de aprobados de los hombres. En este caso, existen 2 contadores específicos. Se debe tener la precaución de no hacer una división entre 0:

```
1 Proceso sin_titulo
2     definir conthombres,contahombres, i, nota como numero
3     Definir sexo Como Caracter
4     definir porc_hombres_aprobados como real
5
6     conthombres=0
7     contahombres=0
8     para i=1 hasta 20 hacer
9         mostrar "ingrese el sexo "
10        leer sexo
11        mostrar "ingrese la nota "
12        leer nota
13        si sexo=="M" entonces // masculino
14            conthombres=conthombres+1
15            si nota > 10 entonces
16                contahombres= contahombres + 1
17            FinSi
18        fin si
19    fin para
20    // se verifica si el conthombres es mayor a 0
21    si conthombres >0 entonces
22        porc_hombres_aprobados = contahombres / conthombres * 100
23    sino
24        porc_hombres_aprobados =0
25    fin si
26    mostrar "el porcentaje de hombres que aprobaron es " porc_hombres_aprobados
27 FinProceso
```



ACADEMIA DE SOFTWARE

Capítulo 6. MAYORES Y MENORES

6.1.- Conceptos

Determinar el mayor valor de los elementos procesados en un ciclo es también una de las salidas más comunes que se solicitan en este tipo de ejercicios. Existen 3 variantes de este problema:

- 1) mostrar el mayor valor de algo.
- 2) mostrar quien tuvo el mayor valor de algo (implica el anterior. Por ejemplo, el nombre del alumno que obtuvo la mayor nota).
- 3) mostrar el mayor valor y quien lo obtuvo.

En los 2 últimos casos existe la posibilidad de que haya empates, es decir, que varios elementos obtengan el mismo valor mayor. En este punto, solo se podrá obtener el valor del primero de ellos. Para resolver este caso, se necesitan arreglos (que se verá más adelante).

Determinar el menor valor de un conjunto de valores es un problema muy similar al de determinar el mayor valor. También existen 3 variantes del problema:

- 1) mostrar el menor valor de algo.
- 2) mostrar quien tuvo el menor valor de algo (implica el anterior. Por ejemplo, el nombre del alumno que obtuvo la menor nota).
- 3) mostrar el menor valor y quien lo obtuvo.

En los 2 últimos casos existe la posibilidad de que haya empates, es decir, que varios "elementos" (ej: alumnos) obtengan el mismo valor menor. En este punto, solo se podrá obtener el valor del primero de ellos. Para resolver este caso, se necesitan arreglos (que se verá más adelante).

Las eventualidades importantes en este tipo de algoritmos, son las siguientes:

- Se debe partir del hecho de que una variable leída que es objeto de comparación solo puede tomar un valor a la vez.
- Los datos introducidos por el usuario en cada iteración del ciclo, no necesariamente se introducirán ordenados.
- Se debe usar una variable referencial contra la cual se comparará mediante el operador relacional que aplique

Por lo tanto, cada vez que el ciclo da una iteración, la variable en cuestión toma un nuevo valor, lo que hace que se pierda el valor anterior. Esto conlleva, que al final del ciclo sólo se tenga el último valor que la variable tomó. Este tipo de problemas ocurre con cualquier tipo de ciclos y con cualquier tipo de variables.

6.2.- Determinación de Mayores

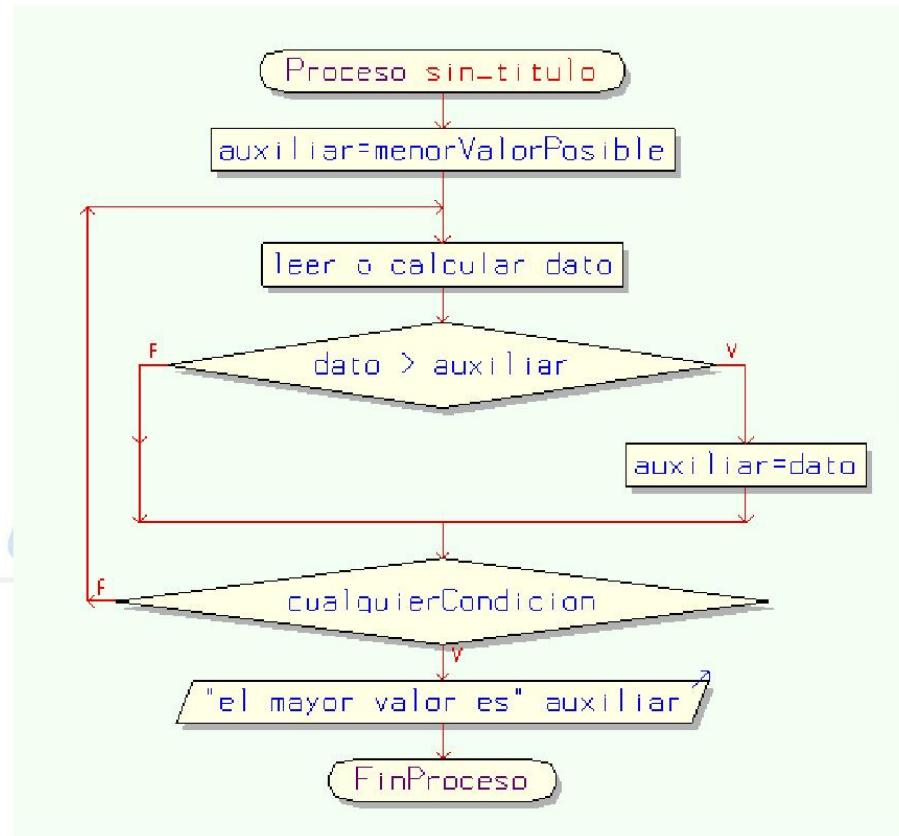
Para calcular el mayor valor de los elementos procesados en un ciclo, se debe tener una variable "auxiliar" (no tiene que recibir ese nombre necesariamente), que va a ser comparada dentro del ciclo con el dato del cual se está buscando el mayor valor (por ejemplo, la edad de un grupo de trabajadores), y que almacenará el valor mayor encontrado hasta ese momento.

Esta variable "auxiliar" debe ser inicializada antes del ciclo con un valor muy bajo con respecto a los valores que puede tomar variable de la cual se va a determinar el mayor valor. Por ejemplo, si se necesita determinar la mayor edad de un grupo de personas, la variable "auxiliar" debe inicializarse en 0, porque nadie tendrá una edad menor a 0.

También debe considerarse que si se desea determinar el mayor valor de una variable que puede tener valores negativos, no se puede inicializar en 0. Por ejemplo, se necesita determinar la mayor temperatura de una ciudad que en ocasiones tiene temperaturas bajo cero, se debe inicializar en la mínima temperatura posible que se pueda alcanzar, por ejemplo, -30.

Para determinar el mayor valor, la variable "auxiliar" se debe comparar dentro del ciclo con el dato en cuestión, luego de leerlo o de calcularlo. Esto implica una comparación, si es verdadera, se actualiza el valor de la variable "auxiliar" con el valor que está siendo mayor en ese momento.

Así sucesivamente hasta procesar todos los valores (hasta terminar el ciclo). El hecho de que la variable auxiliar haya sido inicializada con un valor muy bajo, asegura que en la primera iteración la condición será verdadera. En forma general usando un ciclo "Repetir" el algoritmo es el siguiente:



El siguiente es un ejemplo donde se va a leer la nota de 15 alumnos y necesita determinar cuál fue la mayor nota:

```
1 Proceso sin_titulo
2     definir mayornota, i, nota como entero
3
4     mayornota = 0
5     para i=1 hasta 15 hacer
6         mostrar "ingrese la nota "
7         leer nota
8
9         si nota > mayornota entonces
10            mayornota = nota
11        FinSi
12    fin para
13    mostrar "la mayor nota fue " mayornota
14 FinProceso
```

Si se necesita mostrar algo que identifique al mayor valor (por ejemplo, un nombre), se debe utilizar una variable adicional. El siguiente es una extensión del ejemplo anterior, donde se mostrará el nombre del alumno (el 1ero para el que se cumple) que obtuvo la mayor nota:

```
1 Proceso sin_titulo
2     definir mayornota, i, nota como entero
3     definir nombre, nombre_del_mayor Como Caracter
4
5     mayornota = 0
6     para i=1 hasta 5 hacer
7         mostrar "ingrese el nombre "
8         leer nombre
9         mostrar "ingrese la nota "
10        leer nota
11
12        si nota > mayornota entonces
13            mayornota = nota
14            nombre_del_mayor=nombre
15        FinSi
16    fin para
17    mostrar "El alumno con la mayor nota fue " nombre_del_mayor
18 FinProceso
```

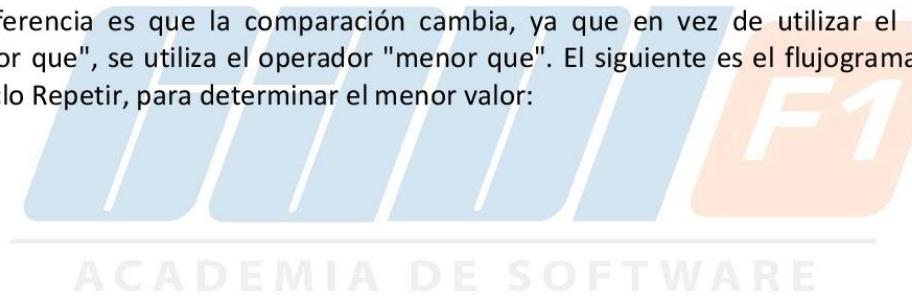
6.3.- Determinación de Menores

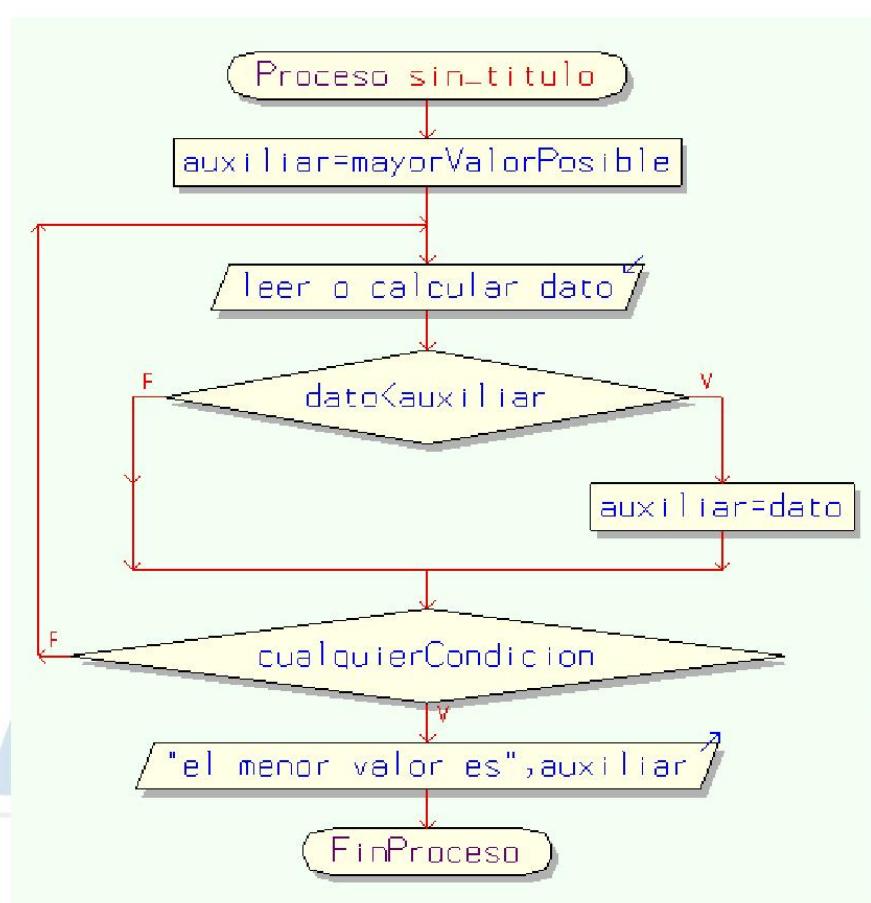
Para determinar el menor valor de un conjunto de valores, en la inicialización, a diferencia de cuando se determina el mayor, se debe inicializar la variable "auxiliar" en un valor muy alto (el más alto que pueda recibir la variable en cuestión).

Por ejemplo, si se requiere calcular la nota más baja de todos los alumnos, donde la nota puede tener valores enteros entre 0 y 100, la variable debe inicializarse en 101, debido a que es posible que la nota más baja sea 100 pts (en el caso de que todos los alumnos obtuvieron 100 pts) y porque nadie podrá obtener una nota más alta de 100 pts (asumiendo que los datos de entrada son válidos).

Para la identificación del menor valor es básicamente el mismo algoritmo que en la determinación del mayor valor.

La diferencia es que la comparación cambia, ya que en vez de utilizar el operador "mayor que", se utiliza el operador "menor que". El siguiente es el flujoograma, usando un ciclo Repetir, para determinar el menor valor:





El siguiente es un ejemplo donde se va a leer la nota de 15 alumnos y necesita determinar cuál fue la menor nota:

```
1 Proceso sin_titulo
2     definir menornota, i, nota como entero
3
4     menornota = 101
5     para i=1 hasta 15 hacer
6         mostrar "ingrese la nota "
7         leer nota
8
9         si nota < menornota entonces
10            menornota = nota
11        FinSi
12    fin para
13    mostrar "la menor nota fue " menornota
14 FinProceso
```

Si se necesita mostrar algo que identifique al menor valor (por ejemplo, un nombre), se debe utilizar una variable adicional. El siguiente es una extensión del ejemplo anterior, donde se mostrará el nombre del 1er alumno que obtuvo la menor nota:

```
1 Proceso sin_titulo
2     definir menornota, i, nota como entero
3     definir nombre, nombre_del_menor Como Caracter
4
5     menornota = 101
6     para i=1 hasta 15 hacer
7         mostrar "ingrese el nombre "
8         leer nombre
9         mostrar "ingrese la nota "
10        leer nota
11
12        si nota < menornota entonces
13            menornota = nota
14            nombre_del_menor=nombre
15        FinSi
16    fin para
17    mostrar "El alumno con la menor nota fue " nombre_del_menor
18 FinProceso
```

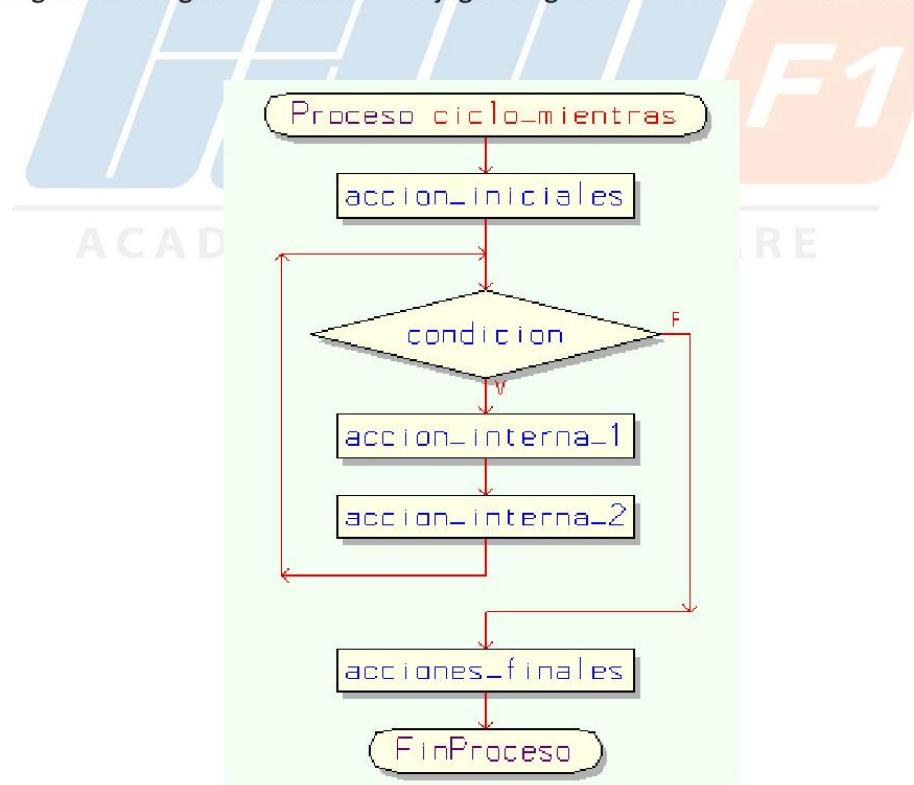
Capítulo 7. ESTRUCTURAS REPETITIVAS. PARTE 4

7.1.- Estructura Mientras

En la estructura repetitiva Mientras-Hacer, la condición de entrada al ciclo se evalúa antes de realizar cualquier iteración del bucle. Si la condición no se cumple, el ciclo no se ejecuta y el programa continúa con la secuencia de acciones siguientes al ciclo. Si la condición es verdadera comienzan a ejecutarse las acciones internas.

Después de la última acción interna, se repite el proceso de evaluación de la condición, si la condición es verdadera de nuevo, entonces se da una nueva iteración. Este proceso continúa hasta que la condición sea falsa (momento en el que se "rompe" el bucle) y la ejecución prosiga con la sentencia siguiente después del bucle.

. En la siguiente imagen se muestra el flujograma general del uso del ciclo "mientras":



Al evaluarse la condición, la 1era vez antes de entrar al bucle, si la condición es falsa no se entrará al bucle, en esta circunstancia el bucle nunca se ejecuta. Por lo tanto, este bucle debe usarse cuando exista la posibilidad de que el bucle nunca deba ejecutarse.

El ciclo "Mientras-Hacer" tal vez sea el ciclo que en menos circunstancias se use, debido a que su necesidad imperiosa no se da en muchos algoritmos, esto porque en la mayoría de los algoritmos se necesita dar por lo menos una iteración

Existen algoritmos donde el uso del ciclo MIENTRAS es muy común, pero no se utilizarán en este curso debido a que implican el manejo de archivos. En el siguiente algoritmo se busca determinar el valor de X que hace que al evaluar una función el resultado sea 0 (se le llama la raíz de una función). Cuando se consigue el valor de X que hace cero la función, se detiene el ciclo:

```
5 Proceso sin_titulo
6     definir x Como Entero
7     definir f Como Caracter
8     definir resultado Como Real
9
10    x=0
11    mostrar "Introduzca la funcion:"
12    leer f
13
14    resultado=evaluar_funcion(f,x)
15    mostrar "El resultado de la evaluacion es " resultado
16    Mientras resultado<>0 hacer
17        x=x+1
18        resultado=evaluar_funcion(f,x)
19        mostrar "El resultado de la evaluacion es " resultado
20    FinMientras
21
22    mostrar "La raiz de la funcion es " x
23
24 FinProceso
```

7.2.- Condiciones Compuestas

En ocasiones es necesario formular una condición compuesta para determinar cuándo se debe romper el ciclo. En el ejemplo anterior, se busca la raíz de la función, pero es posible que el ciclo haga muchas iteraciones y se tarde demasiado en encontrar la raíz, o

simplemente no la consiga. Se necesita usar un contador que permita contar los intentos. Si se alcanza ciertos intentos, se detiene el ciclo:

```

5 Proceso sin_titulo
6     definir x,intentos Como Entero
7     definir f Como Caracter
8     definir resultado Como Real
9
10    intentos=0
11    x=0
12    mostrar "Introduzca la funcion:"
13    leer f
14    resultado=evaluar_funcion(f,x)
15    mostrar "El resultado de la evaluacion es " resultado
16    Mientras resultado<>0 & intentos<20 hacer
17        x=x+1
18        intentos=intentos+1
19        resultado=evaluar_funcion(f,x)
20        mostrar "El resultado de la evaluacion es " resultado
21    FinMientras
22    mostrar "La raiz de la funcion es " x
23 FinProceso

```

7.3.- Comparación Con Otros Ciclos

Cualquier ciclo PARA se puede escribir como un ciclo MIENTRAS, pero no cualquier ciclo MIENTRAS se puede escribir como un ciclo PARA. El siguiente es un ejemplo del uso del ciclo MIENTRAS que típicamente se podría hacer como un ciclo PARA:

```

1 Proceso Sumar_Num
2     Definir N,num,i,suma Como Entero
3
4     Mostrar "¿Cuántos números desea sumar?"
5     Leer N
6
7     i=1
8     Suma=0
9     Mientras i <= N
10        Mostrar "Ingrese el " i ".º número" Sin Saltar
11        Leer num
12        suma=suma+num
13        i=i+1
14    FinMientras
15    Mostrar "La suma resultante es: " suma
16 FinProceso

```

Para convertir un ciclo "repita" en un ciclo "mientras", se debe forzar a que la condición inicial se cumpla. Para esto, la variable involucrada en la condición debe inicializarse con un valor apropiado según sea el caso. En el capítulo del ciclo "repita" se mostró un ejemplo un algoritmo que lee varios números hasta que se introduce un número par. El equivalente con el ciclo "mientras" sería el siguiente:

```
1 Proceso Leer_par
2     Definir num,contador como entero
3
4     num=1 // inicializado un valor impar para asegurar que se entre al ciclo
5     contador=0
6     // el ciclo se romperá cuando sea ingresado un número par
7     Mientras (num mod 2) <> 0 Hacer
8         Mostrar "Introduzca un número : " Sin Saltar
9         Leer num
10        contador=contador+1
11    FinMientras
12
13    Mostrar "El número par se escribió a los " contador "intentos"
14 FinProceso
```



Capítulo 8. ESTRUCTURAS REPETITIVAS DOBLES. PARTE 1

8.1.- Estructura Repetitiva Anidada

Al igual que se pueden colocar unas expresiones dentro de las otras, los bucles pueden estar unos dentro de otros. Al anidar bucles hay que tener en cuenta que el bucle interno funciona como una sentencia más en el bloque del bucle externo, por lo tanto, en cada iteración del bucle externo se van a ejecutar todas las iteraciones del bucle interno.

Los bucles deben estar bien formados y ser sintácticamente correctos para que pueda haber un anidamiento válido, en otras palabras, nunca pueden cruzarse las sentencias de los bucles. Si se tiene un ciclo "Desde", también llamado ciclo "Para" que internamente posee un ciclo "Mientras-Hacer", es necesario finalizar las sentencias o instrucciones del ciclo más interno, en este caso "Mientras-Hacer" antes de colocar la culminación del ciclo externo "Desde".

Si el bucle externo se repite N veces y el interno se repite M veces y por cada iteración del externo se repite el interno, entonces el número total de iteraciones será el producto de M x N.

Los bucles que se anidan pueden ser de igual o distinto tipo.

La anidación de bucles puede ser de 2, 3, 4 y hasta más ciclos. Pueden ser del mismo tipo o de distintos tipos, por ejemplo: un ciclo "Desde" con un "Desde" interno, o un ciclo "Desde" con "Mientras-Hacer" interno (o viceversa), en fin, cualquier combinación es válida.

El siguiente ejemplo de un anidamiento de bucles que sirve para la lectura de los nombres de las asignaturas, en un centro donde hay 8 profesores y cada uno tiene 5 asignaturas:

```

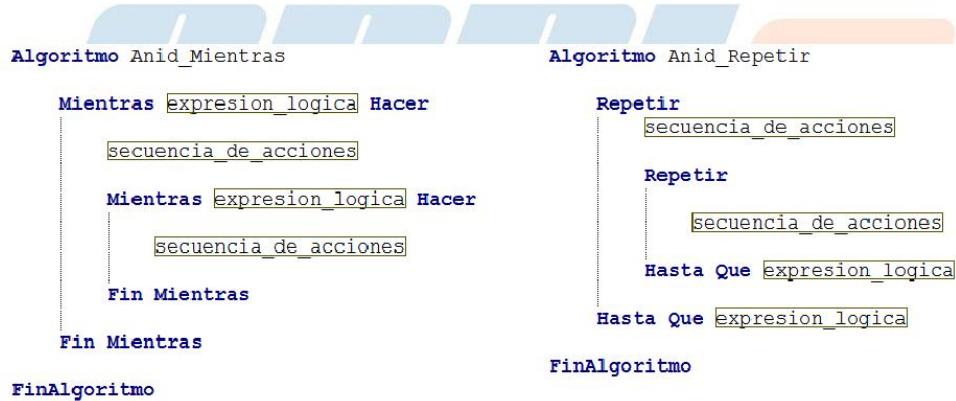
Para i = 1 hasta 8
  Para k = 1 hasta 5
    Escribir "Profesor" i "introduzca su asignatura N°" k
    Leer asignatura
  FinPara
FinPara

```

8.2.- Combinación de Ciclos Anidados

Los ciclos se pueden anidar con cualquier combinación. Los siguientes son ejemplos de cómo hacer ciclos anidados similares:

- "Mientras-Hacer" externo y "Mientras-Hacer" interno.
- "Repetir-Hasta" externo y "Repetir-Hasta" interno.



Igualmente, se puede anidar un "Para", dentro de otro.

```
Algoritmo Anid_Para
// La variable numérica de un PARA debe ser diferente a la del otro
    Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso Hacer
        secuencia_de_acciones
        Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso Hacer
            secuencia_de_acciones
        Fin Para
    Fin Para

FinAlgoritmo
```

El siguiente ejemplo muestra como el anidamiento de dos (2) ciclos genera en pantalla las tablas de multiplicar desde 1 hasta 10:

```
1 Algoritmo Tabla_Multip
2     Definir i,j Como Entero
3
4     Para i=1 hasta 10 Hacer
5         Mostrar "TABLA del #" i;
6
7         Para j=0 hasta 10 Hacer
8
9             Escribir i " * " j " = " i*j
10            FinPara
11
12        FinPara
13 FinAlgoritmo
```

Ejemplo: combinando ciclos "Mientras" y "Para"

```
1 Algoritmo Tabla_Multip
2     Definir i,j Como Entero
3
4     i=1
5     Mientras i<=10 Hacer
6         Mostrar "TABLA del #" i;
7
8         Para j=0 hasta 10 Hacer
9
10            Escribir i " * " j " = " i*j
11        FinPara
12
13        i=i+1
14
15    FinMientras
16 FinAlgoritmo
```

8.3.- Entradas y Salidas

Es importante identificar en la anidación de ciclos donde se va a pedir datos de el usuario y donde se van a mostrar las salidas. En el siguiente ejemplo, se leen entradas antes del ciclo grande, adentro del ciclo grande y adentro del ciclo pequeño. También se muestran salidas adentro del ciclo grande y después del ciclo grande:

```
escribir "profesor: ";
leer profesor;

Escribir "cantidad de alumnos ";
leer cant;

Para i<-1 Hasta cant Hacer
    Limpiar Pantalla
    Escribir "nombre del alumno ";
    leer alumno;

    Escribir "notas a procesar ";
    leer notas;

    para j <- 1 hasta notas Hacer
        escribir "nota ",j;
        leer not;

    FinPara

    escribir "Al alumno ", alumno," se le procesaron ",notas," materias."
    Esperar Tecla
Fin Para

escribir "El profesor ",profesor," proceso ",cant," alumnos.";
```



ACADEMIA DE SOFTWARE

Capítulo 9. ESTRUCTURAS REPETITIVAS DOBLES. PARTE 2

9.1.- Acumuladores y Contadores

Al momento de calcular acumuladores y contadores en algoritmos con ciclos anidados, se debe tener mucho cuidado de donde se deben inicializar y donde se deben actualizar los mismos. Cometer una equivocación en este sentido, generará un error lógico y por consecuencia un resultado diferente al esperado. En el siguiente ejemplo se muestra cómo contar la cantidad de notas en las que todos los alumnos obtuvieron 100 puntos. También, cuantos ceros obtuvo cada alumno:

```
1  Proceso sin_titulo
2      escribir "profesor: ";
3      leer profesor;
4      Escribir "cantidad de alumnos ";
5      leer cant;
6
7      cont_100_seccion=0
8      Para i<-1 Hasta cant Hacer
9          Escribir "nombre del alumno ";
10         leer alumno;
11         Escribir "notas a procesar ";
12         leer notas;
13
14         cont_0=0
15         para j <- 1 hasta notas Hacer
16             escribir "nota ",j;
17             leer not;
18             si not == 100 entonces
19                 cont_100_seccion=cont_100_seccion+1
20             sino
21                 si not == 0 Entonces
22                     cont_0=cont_0+1
23                 Finsi
24             FinSi
25         FinPara
26         escribir "Al alumno ", alumno," se le procesaron ",notas
27         escribir "El alumno obtuvo 0 en " cont_0 " notas"
28     Fin Para
29
30     escribir "El profesor ",profesor," proceso ",cant," alumnos.";
31     escribir "En la sección se obtuvieron 100 pts " cont_100_seccion
32 FinProceso
```

9.2.- Promedios

También es común calcular promedios y porcentajes en este tipo de algoritmos. Es fundamental acumular las variables que correspondan y calcular el promedio en el lugar apropiado. El siguiente ejemplo muestra cómo calcular el promedio de notas de cada alumno: el acumulador se inicializa antes del ciclo pequeño (línea 9), se acumula adentro del ciclo pequeño (línea 13) y se calcula afuera del ciclo pequeño (línea 22).

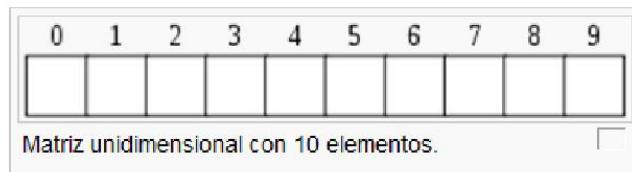
```
1  acum_seccion =0
2  cont_100_seccion=0
3  Para i<-1 Hasta cant Hacer
4      Escribir "nombre del alumno ";
5      leer alumno;
6      Escribir "notas a procesar ";
7      leer notas;
8      cont_0=0
9      acum_alum=0
10     para j <- 1 hasta notas Hacer
11         escribir "nota ",j;
12         leer not;
13         acum_alum=acum_alum + not
14         si not == 100 entonces
15             cont_100_seccion=cont_100_seccion+1
16         sino
17             si not == 0 Entonces
18                 cont_0=cont_0+1
19                 FinSi
20             FinSi
21         FinPara
22         prom_alum=acum_alum/notas
23         acum_seccion=acum_seccion + prom_alum
24         escribir "El promedio del alumno fue de " prom_alum
25         escribir "El alumno obtuvo 0 en " cont_0 " notas"
26     Fin Para
27     prom_seccion=acum_seccion / cant
28     escribir "El promedio de la sección fue de " prom_seccion " pts ";
29     escribir "En la sección se obtuvieron 100 pts " cont_100_seccion " veces "
30
```

Se calcula también el promedio de notas de la sección: el acumulador se inicializa antes del ciclo grande (línea 1), se acumula luego del ciclo pequeño (línea 23) y se calcula luego del ciclo grande (línea 27).

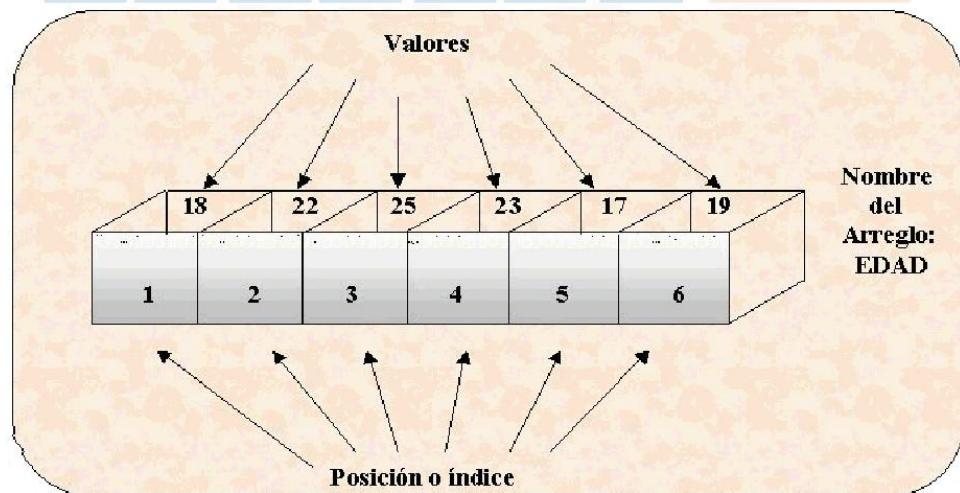
Capítulo 10. ARREGLOS. PARTE 1

10.1.- Concepto

Un arreglo o vector es una estructura de almacenamiento unidimensional que tiene varias posiciones donde se almacena un solo tipo de dato. Dichas posiciones se encuentran ubicadas en forma consecutiva en la memoria RAM



Aquí se aprecia un ejemplo de un arreglo que contiene las edades de 6 personas:



Los arreglos son necesarios para resolver cierto tipo de problemas, los cuales, sin emplearlos serían imposibles de resolver. Algunos ejemplos de estos problemas son:

- determinar si un valor de entrada ya ha sido ingresado por el usuario (búsqueda).
- imprimir al usuario los valores introducidos por él, ordenados según un criterio.
- determinar cuántos y cuáles valores cumplen con ser el mayor o menor.
- indicar cuales valores introducidos por el usuario están por debajo o por encima del promedio.

10.2.- Declaración de un Arreglo

Un arreglo es una variable y por lo tanto se debe declarar antes de usarlo. La declaración de los arreglos o vectores varía mucho según el lenguaje de programación, pero todos tienen en común que al declararlos se debe indicar el número de elementos que tendrá el arreglo. En pseudocódigo se usa la siguiente notación:

Dimension nombre_arreglo[n]

Donde:

- Dimensión: es la palabra reservada que indica que la variable es un arreglo.
- nombre_arreglo: es el nombre del arreglo.
- n: cantidad de elementos que almacenará el arreglo (tamaño del arreglo).

En PseInt el tipo de dato de los elementos del arreglo se establece por autodefinición cuando se ingresa el primer valor al arreglo.

Lo habitual es que un arreglo tenga una cantidad fija de memoria asignada, aunque dependiendo del tipo de arreglo o vector y del lenguaje de programación un vector podría tener una cantidad variable de datos. En este caso, se les denomina vectores dinámicos, en oposición, a los vectores con una cantidad fija de memoria asignada se los denomina vectores estáticos.

10.3.- Índice de Los Arreglos

Todo arreglo se compone de un determinado número de elementos. Cada elemento es referenciado por la posición que ocupa dentro del vector. Dichas posiciones son

llamadas ÍNDICE, si el índice es numérico siempre es correlativo. Existen 3 formas de indexar los elementos en un arreglo:

- Indexación base-cero (0): en este modo el primer elemento del vector será la componente cero (0) del mismo, es decir, tendrá el índice 0. En consecuencia, si el vector tienen n componentes la última tendrá como índice el valor n-1 (siendo n el tamaño del arreglo). En lenguaje C se usa este modo de indexación.
- Indexación base-uno (1): en esta forma de indexación el primer elemento del vector tiene el índice 1 y el último tiene el índice "n" (en un arreglo de n posiciones).
- Indexación base-n(n): este es un modo versátil de indexación en la que el índice del primer elemento puede ser elegido libremente. En algunos lenguajes de programación se permiten índices negativos e incluso un escalar, como cadenas de caracteres.

La forma de acceder a los elementos del arreglo es directa; esto significa que el elemento deseado es obtenido a partir de su índice y no hay que ir buscándolo posición por posición. Los siguientes son ejemplos de acceso a algunos elementos de un arreglo:

```
1 Algoritmo Acc_Arr_Edades
2   Definir num como entero
3
4   Dimension edades[50] // Declaración del arreglo
5
6   edades[1] = 25 // Asignación de un valor constante a la 1era posición del arreglo
7
8   num = 50
9   edades[10] = num // Asignación de un valor variable a la 10ma posición del arreglo
10
11  Si edades[30] = num
12    |   entonces Mostrar "En la posición 30 está almacenado el valor " num
13  FinSi
14 FinAlgoritmo
```

Para trabajar con arreglos muchas veces es preciso recorrerlos, es decir, visitar cada elemento del arreglo. Esto se realiza por medio de ciclos. Generalmente se utiliza el ciclo "para", pero se puede usar también los ciclos "mientras" o "repita". El siguiente

pseudocódigo muestra un algoritmo típico para recorrer un arreglo usando el ciclo "para" y aplicar una operación a cada elemento, en este caso, llenar las posiciones con valores de 2 en 2:

Algoritmo ejemplo

Definir i como entero
Dimension arreglo[50]

Para i=1 hasta 50 haga
 arreglo[i] = i * 2
FinPara

FinAlgoritmo

1	2	3	4	5	6	7	8	9	10	11	12	13	14	48	49	50
2	4	6	8	10	12	14	16	18	20	22	24	26	28	96	98	100

ACADEMIA DE SOFTWARE

Capítulo 11. ARREGLOS. PARTE 2

11.1.- Lectura de Arreglos

Es muy común que los valores leídos por el teclado sean almacenados en un arreglo. A esto le llamamos "lectura o carga de arreglos". Es un proceso análogo al que se hace con variables tradicionales, solo que ahora la información en vez de guardarse en una variable, se guarda en una posición de un arreglo (que finalmente también es una variable). Para ello se necesita realizar un ciclo, generalmente se usa el ciclo "para". El siguiente ejemplo muestra como leer la nota de 20 alumnos:

```
1 Algoritmo LectArr_Notas
2
3 Dimension notas[20] // Dimensionamiento del arreglo
4 definir i como entero
5
6 Para i=1 hasta 20 hacer
7   mostrar "Introduzca la nota:"
8   leer notas[i]
9   // En cada iteración en la posición i se guardará la nota leída
10 FinPara
11
12
13 FinAlgoritmo
```

ACADEMIA DE SOFTWARE

11.2.- Mostrar Arreglos

Es muy común también mostrar al usuario los valores almacenados en un arreglo. Para ello igualmente se necesita un ciclo, que permita visitar cada posición del arreglo. En el siguiente ejemplo se muestra un arreglo de 20 valores que representan las notas de 20 alumnos:

```
1 Algoritmo MostArr_Notas
2   Dimension notas[20]
3   // *** Instrucciones para cargar del arreglo
4   Para i=1 hasta 20 hacer
5     Mostrar notas[i]
6     // en cada iteración se mostrará lo que está en notas[i]
7   FinPara
8 FinAlgoritmo
```

Capítulo 12. ARREGLOS PARALELOS

12.1.- Concepto

En programación, se denomina arreglos paralelos cuando existen 2 o más arreglos relacionados entre sí, los cuales poseen el mismo tamaño. Por ejemplo, si se necesita almacenar las notas y los nombres de 7 alumnos, se necesitan 2 arreglos: uno de tipo real para las notas y otro de tipo alfanumérico para los nombres. Ambos arreglos deben ser de 7 posiciones.

La relación que hay entre ellos es que en la posición 1 de cada arreglo están los datos del primer alumno, en la posición 2 de cada arreglo están los datos del segundo alumno, y así sucesivamente.

Nombres						
1	2	3	4	5	6	7
jose luis	ana maria	ruben daniel	aura virginia	blanca elena	luis jose	jesus esteban

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

Así mismo puede haber 3, 4, 5 o más arreglos paralelos (los que sean necesarios). En este ejemplo se tienen 4 arreglos en paralelo para guardar información de distinta naturaleza: el 1er arreglo para los nombres, el 2do para las notas, el 3ero para las edades y el 4to para los sexos de 7 alumnos:

Nombres						
jose luis	ana maria	ruben daniel	aura virginia	blanca elena	luis jose	jesus esteban
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	21	20	23
1	2	3	4	5	6	7
Sexo						
M	F	M	F	F	M	M
1	2	3	4	5	6	7

Como se puede visualizar en el ejemplo, los datos de los alumnos se interpretan de la siguiente forma: el alumno 1 tiene nombre "jose luis", como nota 12.5, tiene 22 años de edad y es sexo "M". El alumno 2 tiene como nombre "ana maria", tiene una nota de 16.8, una edad de 19 años y es sexo "F", y así sucesivamente.

12.2.- Lectura de Datos

Para cargar los 2 primeros arreglos (nombres y notas) 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 quedaría de la siguiente manera:

```
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
```

Cuando se manejan arreglos paralelos, todos deben leerse paralelamente para respetar la relación que hay entre las posiciones en cada arreglo.

```
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
```

12.3.- Mostrar Datos

Para mostrar arreglos paralelos modularmente, siguiendo con el mismo ejemplo de los 2 primeros arreglos, es muy similar a como se implementó la lectura usando un módulo:

```
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
```

ACADEMIA DE SOFTWARE

Capítulo 13. APROVECHANDO LOS ARREGLOS

13.1.- Promedio

Para calcular el promedio de los elementos de un arreglo, se deben sumar los valores en cada posición del arreglo (usando un acumulador). Para esto es necesario visitar cada posición del arreglo con un ciclo (generalmente un "para"). El siguiente ejemplo muestra un módulo para calcular el promedio de notas de un grupo de 7 alumnos, que están almacenadas en un arreglo:

```
1 Subproceso Calcular_PromNotas (notas,promedio Por Referencia)
2     Definir acum como Real
3
4     acum=0
5     Para i=1 hasta 7 Hacer
6         acum=acum+notas[i]
7     FinPara
8     prom_notas=acum / 7
9 FinSubproceso
```

13.2.- Determinar Mayores

El algoritmo para determinar el elemento mayor en un arreglo es muy similar al algoritmo que se usó para determinar mayores en el capítulo 8. La diferencia radica, en que sin arreglos, el valor mayor debe ir determinándose en la medida en que se van leyendo y/o procesando los valores del cual necesitamos determinar el mayor.

En cambio con arreglos, los valores en cuestión, al estar almacenados en ellos, pueden verificarse en cualquier momento que se desee. Como en la mayoría de algoritmos de arreglos, éste debe recorrerse con un ciclo.

En el siguiente ejemplo se tienen cargados en un arreglo las notas de 15 alumnos y se necesita determinar cuál es la mayor nota. En el algoritmo, se asume que el mayor valor se encuentra en la primera posición del arreglo, por lo cual se empieza a recorrer a partir de la 2da posición y adentro del ciclo se compara cada elemento con el mayor. Si

alguno cumple la condición, se actualiza la variable auxiliar "mayor". Si existen varios valores iguales al mayor (empate), no importa, porque solo se desea saber cuál es el mayor valor, no cuantas veces está presente.

```
mayornota=notas[1]
para i=2 hasta 15 Hacer
    Si notas[i] > mayornota entonces
        mayornota=notas[i]
    FinSi
FinPara
Mostrar "La mayor nota fue: " mayornota
```

Al determinar el mayor valor de un arreglo, también es muy común, informar que posición ocupa el elemento mayor. Para este ejemplo de algoritmo, en el caso de haber empates (varios elementos iguales al valor mayor), se encontrará la posición del primero de los mayores.

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

Si se está trabajando con arreglos paralelos, se puede determinar a quién pertenece el valor mayor. De haber empates, se guardará sólo el 1er nombre que corresponde con el mayor. El siguiente es un ejemplo:

```
// 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
```

13.3.- Determinar Menores

De forma análoga, el algoritmo para determinar el elemento menor en un arreglo es muy similar al algoritmo que se usó para determinar menores en el capítulo 3. A sabiendas, de que si usar arreglos, el valor menor debe ir determinándose en la medida en que se van leyendo y/o procesando los valores del cual necesitamos determinar el menor.

Continuando con el ejemplo del arreglo con las notas de los 15 alumnos, a través de ese algoritmo se puede saber cuál fue la menor nota y en qué posición del arreglo está la primera nota que cumple con la condición:

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

En caso de contar con arreglos paralelos y haber almacenado los nombres, este sería el algoritmo que permitiría saber quién es el primero 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
```



Capítulo 14. BÚSQUEDA EN ARREGLOS

14.1.- Buscar un Elemento único

Buscar un elemento en un arreglo consiste en determinar si un valor se encuentra en alguna de las posiciones del arreglo. La búsqueda puede ser de un elemento único o de un elemento que puede repetirse (no único). Para buscar un elemento en un arreglo se debe hacer un ciclo y se debe verificar cada posición del arreglo con una instrucción "si", para comparar cada elemento del arreglo con el valor que se está buscando. En forma general, el algoritmo de búsqueda es el siguiente:

```
encontrado=falso  
para i=1 hasta nroelementos  
    si arreglo[i] = algoquebuscado entonces  
        encontrado=verdadero  
    fin si  
fin para
```

Cuando la búsqueda es de valores únicos, no se debe utilizar un ciclo "para", ya que en éste se sabe cuantas vueltas va a dar el algoritmo; pero en las búsquedas, no se sabe si el elemento buscado se va a encontrar en la primera posición del arreglo, en el medio o en la última posición. Es por esto, que se usa un ciclo "repetir" o "mientras". Los siguientes son ejemplos de búsqueda sin modularidad y con modularidad:

```
// buscar elementos únicos
encontrado=falso
i=1
Repetir
    si cedulas[i]=="123456"
        entonces encontrado=verdadero
        sino i=i+1
    FinSi
Hasta Que (encontrado=verdadero) o (i>7)
```

```
// buscar elementos únicos implementando un módulo
Subproceso Buscar_cedula(cedulas,cedbuscada,encontrado Por Referencia)
    encontrado=falso
    i=1
    Repetir
        si cedulas[i]=cedbuscada
            entonces encontrado=verdadero
            sino i=i+1
        FinSi
    Hasta Que (encontrado=verdadero) o (i>7)
FinSubProceso
```

14.2.- Buscar un Elemento Repetido

Con arreglos, se puede informar si hay varios elementos iguales, lo cual era imposible antes de trabajar con arreglos. Este es un caso de búsqueda de valores no únicos. Por ejemplo, en un arreglo de notas se desea saber si hay varios alumnos con la nota mayor, primero se debe determinar el valor mayor, luego, se debe buscar cuantos elementos son iguales al valor mayor. El siguiente es un ejemplo para llevar adelante dicho conteo:

```
cont=0
Para i=1 hasta 7
    Si notas[i]=mayornota
        entonces cont=cont+1
    FinSi
FinPara
Mostrar "La mayor nota la obtuvieron" cont "alumnos"
```

Con arreglos paralelos se puede desplegar listados de información que 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"
```



Capítulo 15. ORDENAMIENTO

15.1.- Introducción

Los valores en un arreglo están organizados secuencialmente, uno después del otro, pero no necesariamente deben estar ordenados. Al cargar los datos en un arreglo (leer por el teclado) generalmente se cargan de forma desordenada. No siempre es necesario ordenar los elementos de un arreglo, esto se hace cuando se solicita en un ejercicio o problema mostrar un listado ordenado de valores.

Los valores en un arreglo pueden estar ordenados ascendente (de menor a mayor) o descendente (de mayor a menor). Por ejemplo, listado de alumnos ordenados por nota de mayor a menor (descendente) o de menor a mayor (ascendente).

15.2.- Algoritmo Burbuja

Existen varios algoritmos ya predefinidos para ordenar los elementos de un arreglo, ya sea de forma ascendente o descendente. Veamos un algoritmo para ordenar llamado "burbuja", en el cual se ordenan los elementos de menor a mayor (ascendente):

```
Para i=1 hasta tamaño-1
    Para j=1 hasta tamaño-1
        Si arreglo[j] > arreglo[j+1]
            aux=arreglo[j]
            arreglo[j]=arreglo[j+1]
            arreglo[j+1]=aux
        FinSi
    FinPara
FinPara
```

En forma modular y de mayor a menor (descendente):

```
SubProceso Burbuja_Descendente (arreglo)
    Para i=1 hasta tamaño-1
        Para j=1 hasta tamaño-1
            Si arreglo[j] < arreglo[j+1] Entonces
                aux=arreglo[j]
                arreglo[j]=arreglo[j+1]
                arreglo[j+1]=aux
            FinSi
        FinPara
    FinPara
fin subprocesso
```

ACADEMIA DE SOFTWARE