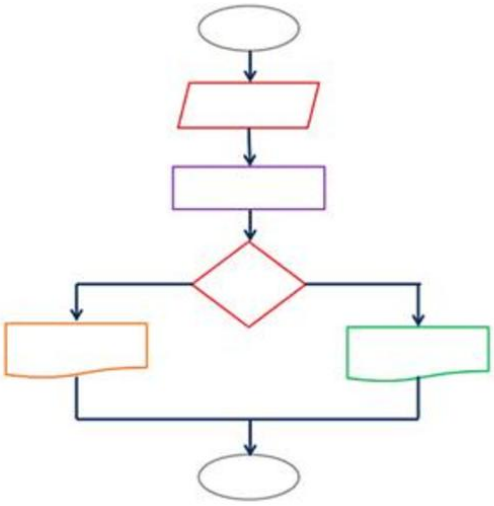
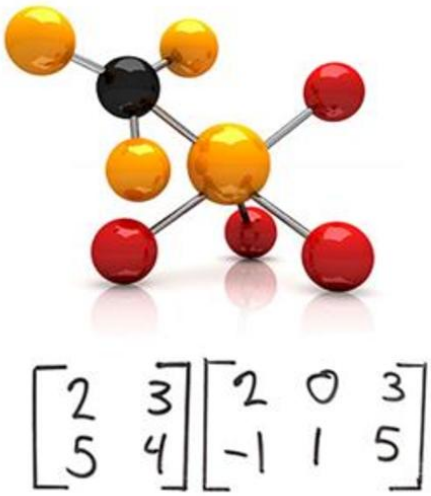




Lógica de Programación. Nivel II

junio, 2019

F1
ACADEMIA DE SOFTWARE

Objetivos del nivel

- Entender el uso de las instrucciones de control en los algoritmos.
- Aprender a usar instrucciones condicionales.
- Aprender a usar instrucciones repetitivas (ciclos).
- Familiarizarse con los arreglos.

Prerrequisitos del nivel

- Lógica de Programación Nivel I

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.

ACADEMIA DE SOFTWARE

F1
ACADEMIA DE SOFTWARE

Contenido del nivel

Capítulo 1. Expresiones Lógicas

1.1.- ¿Qué es una Expresión Lógica?.
1.2.- Operadores Relacionales.
1.3.- Construcción de Expresiones Lógicas.

Capítulo 2. Estructuras Selectivas 1

2.1.- Control de Flujo y Expresiones Lógicas
2.2.- ¿Qué es una Estructura Selectiva?.
2.2.- Estructura Selectiva Simple.

Capítulo 3. Estructuras Repetitivas 1: Definiciones

3.1.- Conceptos Básicos Sobre Ciclos.
3.2.- Control de Ciclos.
3.3.- Tipos de Ciclos.
3.4.- Ciclos Infinitos.

Capítulo 4. Estructuras Repetitivas 2: Para

4.1.- Estructura Repetitiva "para".
4.2.- Número de Iteraciones Variable.
4.3.- Variando el Incremento o Decremento.

Capítulo 5. Estructuras Selectivas 2: Selectiva Doble

5.1.- Estructura Selectiva Doble.
5.2.- Aplicaciones de la Selectiva Doble.

Capítulo 6. Estructuras Selectivas 3: Anid. de Selectivas

6.1.- Anidación de Condicionales.
6.2.- Anidación de Condicionales Por el Entonces.
6.3.- Anidación de Condicionales Por el Sino.

ACADEMIA DE SOFTWARE

Capítulo 7. Estructuras Selectivas 4: Aplic. de Anidamientos

7.1.- Comparación de Variables.
7.2.- Evaluación de Intervalos Contiguos.

Capítulo 8. Operaciones Lógicas y Selectivas 1

8.1.- Operadores Lógicos.
8.2.- Expresión Lógica Compuesta.
8.3.- Evaluación de Intervalos Discontiguos.

Capítulo 9. Operaciones Lógicas y Selectivas 2

9.1.- ¿ Qué es Una Tabla de Verdad?.
9.2.- Tabla de la Conjunción.
9.3.- Tabla de la Disyunción.
9.4.- Priorización de Evaluación en Expresiones Lógicas.

Capítulo 10. Operaciones Lógicas y Selectivas 3

10.1.- Empleo de Variables Lógicas.
10.2.- Tabla de la Negación.
10.3.- Uso de la Negación.

ACADEMIA DE SOFTWARE

Capítulo 11. Estructuras Selectivas 5: Alternativa Múltiple

11.1.- Estructura Según.
11.2.- Opción Otro.
11.3.- Según Vs. Anid. de Selectivas.

Capítulo 12. Estructuras Repetitivas 3: Repetir

12.1.- Ciclo Repetir.
12.2.- Repetir...Hasta.
12.3.- Repetir...mientras.
12.4.- Repetir-Hasta vs Repetir-Mientras.

Capítulo 13. Estructuras Repetitivas 4: Mientras

13.1.- Estructura Mientras.
13.2.- Interrumpiendo Con Condiciones Compuestas.
13.3.- Comparación Con Otros Ciclos.

Capítulo 14. Arreglos 1: Definiciones y Acceso

14.1.- Concepto.
14.2.- Declaración de un Arreglo.
14.3.- Índice de Los Arreglos.

Capítulo 15. Arreglos 2: Lectura y Mostrado

15.1.- Visitar Las Posiciones Del Arreglo.
15.2.- Lectura de Arreglos.
15.3.- Mostrar Arreglos.

ACADEMIA DE SOFTWARE

Capítulo 1. EXPRESIONES LÓGICAS

1.1.- ¿Qué es una Expresión Lógica?

Una EXPRESIÓN LÓGICA es aquella que sólo puede tomar uno de 2 valores: verdadero o falso (nótese que dice "o", es decir, no puede ser verdadero y falso a la vez). Dicho valor resulta de evaluar dicha expresión, que se infiere a partir de interrogantes explícitas o no, planteadas en los problemas. Los siguientes son ejemplos de cuestionamientos, que se pueden traducir a expresiones lógicas:

- ¿ Tengo dinero ?
- ¿ Eres mayor de edad ?
- ¿ Esta haciendo frío ?
- ¿ Es de noche ?
- ¿ Eres más alto que yo ?

Computacionalmente ante todas estas preguntas sólo son válidos los 2 posibles resultados referidos, y en cada caso, la respuesta siempre dependerá de "algo".

En los problemas que se van a intentar resolver con programas de computadora, habrá diversidad de circunstancias en las cuales los algoritmos (antecesores a los programas) deben hacer una cosa u otra según convenga, para lo cual se necesitará evaluar si algo es verdad o no.

ACADEMIA DE SOFTWARE

Por ello, es requerido identificar las expresiones lógicas que deberán estar involucradas en la resolución a dichos problemas.

1.2.- Operadores Relacionales

Las expresiones lógicas en los algoritmos de computadoras deben expresarse en términos de variables comparándose con otras variables o valores, lo cual permite determinar la "relación" que existe entre los elementos que se están comparando. Para ello, se usan los Operadores Relacionales:

< Menor que
<= Menor o igual que

> Mayor que
>= Mayor o igual que
= Igualdad [en C: ==]
<> Diferente [en C: !=]

Los elementos que se comparan deben ser del mismo tipo o de tipos de datos compatibles, es decir, no se puede comparar números con alfanuméricos, caracteres con booleanos, etc.

En las expresiones lógicas, el operador relacional se coloca entre los 2 operandos (los elementos que se desean comparar entre ellos). Estos operandos pueden ser valores fijos o variables o combinación de ambos. Aquí se plantea un ejemplo con 2 operandos fijos:

5 > 7

El resultado de esta expresión siempre es falso, debido a que el número 5 no es mayor que el número 7. Otra forma de escribir esta expresión es la siguiente:

7 < 5

De igual forma, esta expresión es falsa. Es una expresión equivalente a la anterior.

Cabe destacar que rara vez se comparan 2 valores fijos, debido a que el resultado siempre será el mismo.

ACADEMIA DE SOFTWARE

Para evaluar una expresión donde los operandos son expresiones aritméticas o matemáticas:

1) Se resuelve el primer operando y se sustituye por su valor.
2) Se resuelve el segundo operando y se sustituye por su valor.
3) Se aplica el operador relacional y se devuelve el valor booleano correspondiente.

Por ejemplo:

1er operando 2do operando
((5* 4) + 1 - (5 ^ 2)) < (2 – 1)
-4 < 1

La expresión resulta en Verdadero.

Siempre que se tenga que, en algún operando hay alguna expresión, está debe ser resuelta antes de hacer la comparación (aplicar el operador relacional).

Normalmente se utilizan variables en las expresiones lógicas. El resultado de la expresión dependerá del valor de las variables. Por ejemplo, si se tienen las variables “x” y "y", asignándole el valor 2 a "x" y a la variable “y” se le asigna el valor 4. Cada respuesta debe ser VERDADERO o FALSO, según corresponda:

1. x > 10 :_____
2. y < 0 :_____
3. 3 >= x :_____
4. x = y :_____
5. x <> y :_____
6. y > x :_____
7. x > =y :_____
8. 3*x > y :_____
9. x+y < 5 :_____
10. y-2 = x :_____

1.3.- Construcción de Expresiones Lógicas

ACADEMIA DE SOFTWARE

En los casos donde es necesario realizar distintas acciones con base en "preguntas", ante cada una de ellas debe determinarse la(s) variable(s) involucrada(s) y/o el(los) valor(es) contra el(los) cual(es) se planteará la comparación, usando los operadores relacionales que correspondan.

A partir de las interrogantes ejemplificadas anteriormente, se pueden estructurar las expresiones lógicas a evaluar. Por ejemplo, véanse las variables implicadas en las siguientes preguntas:

- ¿ Eres mayor de edad ?, tiene implícito la evaluación de la variable "edad",
- ¿ Está haciendo frío ?, tiene implícito la variable "temperatura", etc.

Expresión en lenguaje natural	Expresión lógica
- ¿ Tengo dinero ?	dinero > 0

- ¿ Eres mayor de edad ?	edad > 17 [o edad>=18]
- ¿ Esta haciendo frío ?	temperatura < 20
- ¿ Es de noche ?	luminosidad <= 1
- ¿ Eres más alto que yo ?	mi_estatura < tu_estatura

En algunos casos puede que no sea tan fácil traducir la interrogante en lenguaje natural cómo expresión lógica. Por ejemplo, ante la pregunta: ¿ Está haciendo frío ? hay otra interrogante a resolver: ¿ A qué temperatura se considera que hay frío ?, la respuesta puede ser relativa porque para algunas personas "frío" puede ser debajo de 16 grados, pero para otros puede ser debajo de los 20 grados. La pregunta ¿ Es mayor de edad ? tendría implícito también la pregunta "¿a qué edad se es mayor de edad?". En algunos países la mayoría de edad se cumple a los 21, en otros a los 18.

Los siguientes son ejemplos de expresiones que se pueden llevar a expresiones lógicas:

- ¿ se acabaron las entradas ?
- ¿ aprobé la materia ?
- ¿ mi hermano es más viejo que yo ?
- ¿ eres de la tercera edad ?
- ¿ estoy libre de deudas ?
- ¿ ya comenzó la película ?
- ¿ este año es bisiesto ?

ACADEMIA DE SOFTWARE

Capítulo 2. ESTRUCTURAS SELECTIVAS 1

2.1.- Control de Flujo

El orden de ejecución de las sentencias es lo que se conoce por el flujo del programa, y se puede variar a voluntad utilizando las sentencias de control de flujo, con las que es muy fácil alterarlo para adecuarlo a las necesidades del programador. Esto es lo que hace potentes a los lenguajes de programación.

Básicamente, existen tres tipos de sentencias, secuencial, condicional e iterativa, y utilizándolas apropiadamente se puede escribir cualquier programa. Las sentencias secuenciales son las que se ejecutan una a continuación de la otra. Y son las que se han venido usando desde el 1er nivel del curso. Las sentencias condicionales e iterativas sirven para manejar la posibilidad de cambiar el control de flujo, y se verán a partir de ahora. En este tipo de instrucciones es necesario utilizar expresiones lógicas.

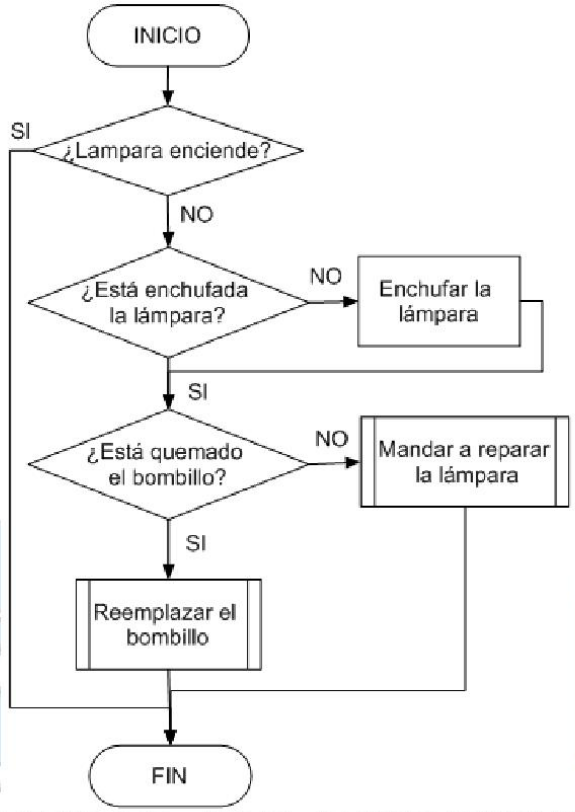
2.2.- ¿Qué es una Estructura Selectiva?

Antes de ver formalmente que es una Estructura Selectiva es importante recordar que en el nivel 1, se aprendió que los rombos se utilizan para establecer puntos de decisión. Usando lenguaje coloquial, dentro de ellos se anotaron interrogantes, y dependiendo de la respuesta a ellas se ejecutaban una(s) acción(es) u otra(s). Esto se aprecia en el flujograma que delinea las acciones implicadas al encender una lámpara.

ACADEMIA DE SOFTWARE

Este ejemplo es relevante, porque permite ver el impacto de responder a interrogantes cruciales que pueden alterar el flujo de ejecución.

Computacionalmente, todas las interrogantes deberán ser traducidas a expresiones lógicas.



ACADEMIA DE SOFTWARE

Hasta ahora se han desarrollado algoritmos "secuenciales", donde todas las instrucciones que conforman un algoritmo son ejecutadas siempre, sin importar lo que suceda con los datos durante la ejecución del programa.

Una ESTRUCTURA SELECTIVA, es un tipo de instrucción que se utiliza para tomar decisiones lógicas y manejar la posibilidad de proseguir la ejecución por un camino alternativo. Por lo cual, son sentencias de control de flujo, porque permiten que el algoritmo realice alguna(s) instrucción(es) sólo bajo algunas circunstancias, así pues, éstas estarán condicionadas.

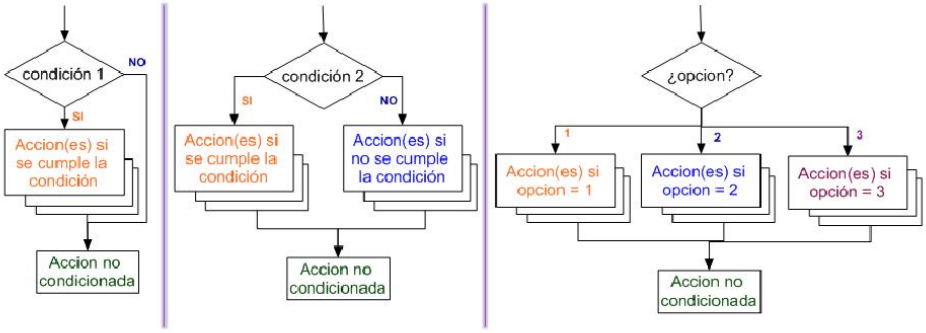
Asimismo estas selectivas son denominadas estructuras... condicionales, de decisión, de toma de decisiones o de alternativas y se clasifican en 3 tipos:

- simples

- dobles
- de alternativa múltiple.

Apréciese en los 3 posibles flujogramas de las selectivas, como posteriormente a la evaluación de una condición (expresión lógica) el control de flujo del programa pudiera direccionarse a la ejecución de un grupo de acciones o a otro.

Las estructuras condicionales deberán ser conformadas y utilizadas en una diversa gama de situaciones, destacando que es determinante detectar cuál(es) instrucción(es) deben estar condicionadas y por cuál condición.



2.3.- Estructura Selectiva Simple

La Estructura Selectiva Simple es la estructura condicional elemental. Presenta la forma:

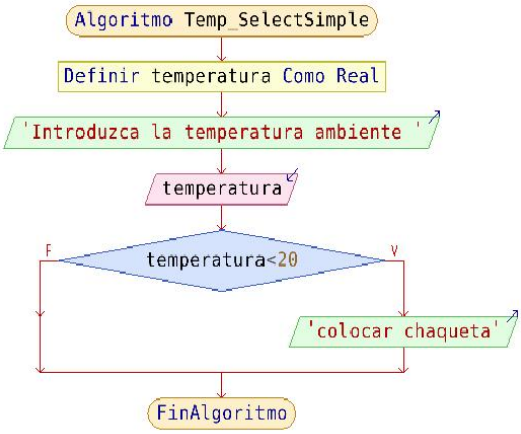
```
Si <condicion> Entonces
    <Acciones>
Fin Si
```

En esta estructura condicional, si la condición es Verdadera, se ejecuta el conjunto de acciones asociadas al "entonces", si el resultado es Falso no se ejecuta ninguna acción. Por ej., indicar que se use una chaqueta si se tiene frío (temperatura menor a 20 grados).

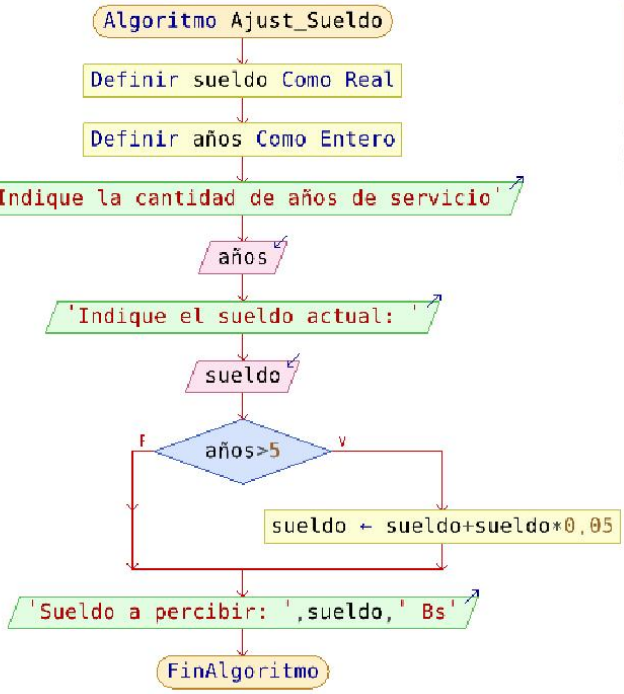
ACADEMIA DE SOFTWARE

```
Si temperatura < 20  entonces
     mostrar "colocar chaqueta"
Fin Si
```

En este caso, si se cumple la condición se mostrará el mensaje, en caso contrario no aparecerá nada.



En el siguiente flujograma se aprecia la implementación de la Selectiva Simple en PseInt, para calcular un aumento de 5% al sueldo a un trabajador solamente cuando tiene más de 5 años en la empresa:



El algoritmo textual de este flujograma se muestra a continuación. Es importante identificar cuales son las acciones secuenciales y cual es la acción condicionada.

```
Algoritmo Ajust_Sueldo
    Definir sueldo Como Real
    Definir años como Entero

    Mostrar "Indique la cantidad de años de servicio"
    Leer años
    Mostrar "Indique el sueldo actual: "
    Leer sueldo

    Si años > 5 Entonces
        sueldo = sueldo + sueldo * 0.05
    FinSi

    Mostrar "Sueldo a percibir: " sueldo " Bs"
FinAlgoritmo
```

Véase como se valoran los aspectos determinantes cuando en el ejemplo anterior:

1) ¿Cuál es la condición que debe evaluarse en la selectiva?: La condición correcta es: años > 5. Si por ejemplo, se cambiara el operador por >=, se produciría un error lógico, ya que si el empleado tiene 5 años también se la calcularía el aumento, lo cual no cumple con el objetivo planteado.

2) ¿Cuáles instrucciones están condicionadas y cuáles no?... ya que colocar una instrucción dentro del "si" cuando debe estar afuera o viceversa acarrearía un error lógico. En ejemplo, las declaraciones de variables, entradas y salida se hacen siempre, debido a que no están condicionadas por nada, en particular es destacable que, el mensaje de salida se presentará independientemente de si el trabajador tiene más de 5 años en el empleo o no.

Capítulo 3. ESTRUCTURAS REPETITIVAS 1: DEFINICIONES

3.1.- Conceptos Básicos Sobre Ciclos

Ya se ha comenzado a ver como incluir cambios en el control de flujo usando estructuras selectivas simples, de manera que en esos algoritmos algunas instrucciones se ejecutan dependiendo de condiciones.

Por otro lado, hay problemas cuya resolución amerita repetir pasos específicos. Para ello, deben usarse sentencias o estructuras iterativas.

Una ESTRUCTURA ITERATIVA es aquella instrucción que se coloca en el código para lograr que ciertas instrucciones se ejecuten un número finito de veces. En estas estructuras también llamadas de ciclo o bucle, cada repetición se denomina ITERACIÓN y coloquialmente, también es conocida como "vuelta" o "pasada".

En la vida diaria, por ejemplo, existen situaciones que frecuentemente se resuelven realizando una determinada secuencia de pasos que puede repetirse varias veces, tales como:

- Las operaciones que realiza para llamar por teléfono, mientras no se logre la comunicación.
- El proceso que sigue para comer, mientras no se termine la comida.

ACADEMIA DE SOFTWARE

Otro ejemplo cotidiano donde se evidencia el uso de ciclos, es la corrección que hace un profesor de un examen de selección simple aplicado a una sección, ya que para ello el profesor tendría que:

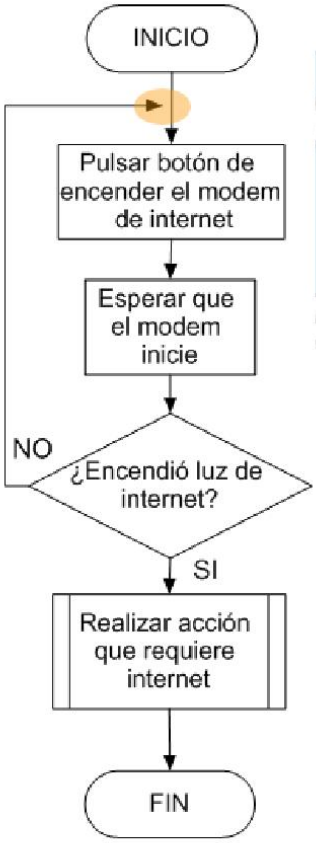
1) Ver la respuesta de la primera pregunta y compararla con la respuesta correcta, anotar la nota en la pregunta. Este paso se va a repetir tantas veces, como preguntas tenga el examen.
2) Totalizar la nota del examen sumando la nota de cada pregunta,
3) Ver la cédula,
4) Cotejar el nombre del alumno,
5) Asignar la nota final al examen,
6) Anotar la nota en lista de publicación de notas.

Adicionalmente, este algoritmo lo debe repetir con cada examen presentado.

¿Dónde están los ciclos?

Un ejemplo donde se use un ciclo usando una computadora, sería un caso en cual se desee conocer el promedio de notas obtenido por c/u de los N alumnos de una sección, repitiendo las acciones de leer 3 notas, promediarlas y mostrar el promedio resultante.

En los flujogramas con ciclos, se aprecia al menos una línea que sube y apunta a algún punto anterior en el diagrama, lo cual denotará que el control del flujo retorna a ese punto y por lo tanto se repetirán las instrucciones a partir de él.



3.2.- Control de Ciclos

Todo bucle implica la evaluación de una condición implícita o explícita, que es la que va a determinar si debe ejecutarse el bucle y hasta cuándo. Por esto, una vez se identifique que es necesario usar ciclos, se deben determinar 2 cosas:

1) qué instrucciones van a repetirse en el ciclo y,
2) qué condición va a mantener el ciclo en ejecución (dicho de otra forma, como se controlará el ciclo). Esto último es muy importante, ya que dicha condición deberá evaluarse al principio o al final de la sentencia repetitiva (depende de la sentencia iterativa que se emplee).

Los ciclos pueden romperse (o controlarse) de 2 formas:

- cuando se haya cumplido el número de iteraciones que se esperaba dar.
- cuando ocurra algo específico, por ejemplo, que una variable tome un valor.

Por lo anterior, cada problema puede ser muy particular. La forma de control del ciclo o de los ciclos (en caso de que haya varios) debe determinarse al analizar el problema.

3.3.- Tipos de Ciclos

ACADEMIA DE SOFTWARE

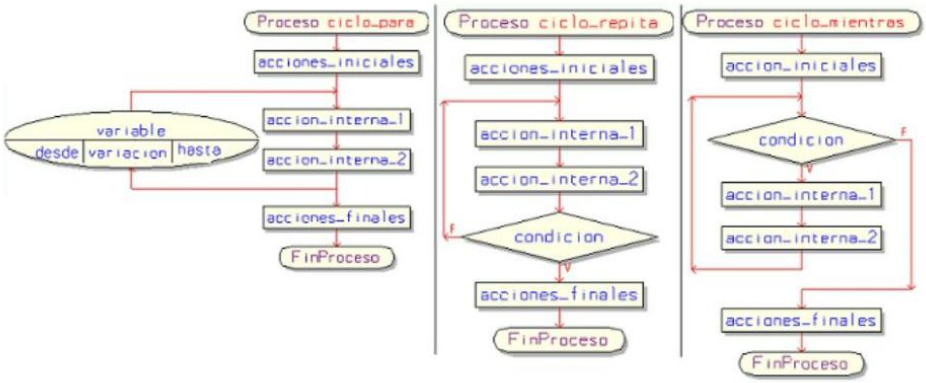
Los lenguajes de programación tradicionalmente implementan 3 tipos de ciclos, los cuales se usan dependiendo de la situación y del problema a resolver. Los tipos de ciclos son:

- PARA o DESDE (for): se utiliza generalmente cuando la cantidad de iteraciones es predecible (se conoce antes de iniciar el ciclo).

- REPETIR (repeat, do...until, do..while): se utiliza cuando la cantidad de iteraciones es impredecible.

- MIENTRAS (while): también se utiliza cuando la cantidad de iteraciones es impredecible, pero a diferencia del anterior, se utiliza en situaciones más particulares; es el caso, cuando es necesario manejar la posibilidad de que nunca se entre al ciclo.

Cada tipo de ciclo se representa de manera diferente dentro de un flujograma:



3.4.- Ciclos Infinitos

El establecimiento de una condición incorrecta en un ciclo podría hacer que éste nunca se rompiera, generando así, lo que se llama en programación un "ciclo infinito", que virtualmente tardaría un tiempo infinito ejecutándose. Esto nunca debería ocurrir en situaciones tradicionales, puesto que en la generalidad de los casos los algoritmos deben ejecutarse en un tiempo finito.

ACADEMIA DE SOFTWARE

Aun así, existen situaciones en las cuales los algoritmos deben ejecutar un "ciclo infinito controlado", por ejemplo: los sistemas que funcionan recibiendo "eventos" o mensajes del usuario de forma asíncrona (sistemas de chat, de ventanas, etc.). Este tipo de ciclo se romperán por acciones externas al programa, por ej.: una acción realizada por el usuario del sistema.

Este tipo de ciclo no serán abarcados en este curso.

Capítulo 4. ESTRUCTURAS REPETITIVAS 2: PARA

4.1.- Estructura Repetitiva "para"

La estructura repetitiva DESDE o PARA (llamada "for" en la mayoría de los lenguajes de programación) es usada cuando se conoce el número exacto de veces que se debe ejecutar el ciclo.

Este ciclo se usa para llevar conteos, para lo cual usa una variable denominada variable índice, a la que se le asigna un valor inicial (VI), se le indica el valor final (VF) y opcionalmente se le asocia la cantidad en la que variará.

En cada iteración del bucle, la variable índice se incrementa o decrementa de manera automática, en un valor constante, lo cual la constituye en una operación implícita. Dado lo anterior, el programador no se tiene que ocupar de actualizarla colocando una instrucción dentro del bucle. Así, la ejecución del ciclo continuará hasta que se alcance o se sobrepase el valor que se ha establecido como final, momento en el cual se rompe el ciclo.

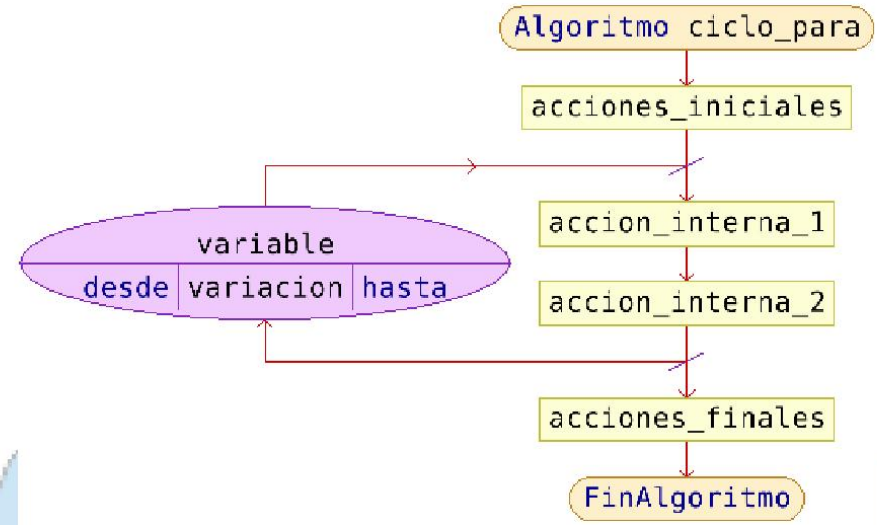
Por omisión, la variación que experimentará la variable índice es de 1 en 1.

La estructura general del ciclo "para" es la siguiente:

ACADEMIA DE SOFTWARE

```
acciones_iniciales

para variable=ValorInicial hasta ValorFinal

        accion_interna_1
        accion_interna_2

fin para

acciones_finales
```

El flujograma general del ciclo “para” es el siguiente:



En PseInt la estructura del ciclo "Para" es la siguiente:

```
instrucciones iniciales

para variable=VI hasta VF con paso X

    accion_interna_1
    accion_interna_2

fin para

instrucciones finales
```

ACADEMIA DE SOFTWARE

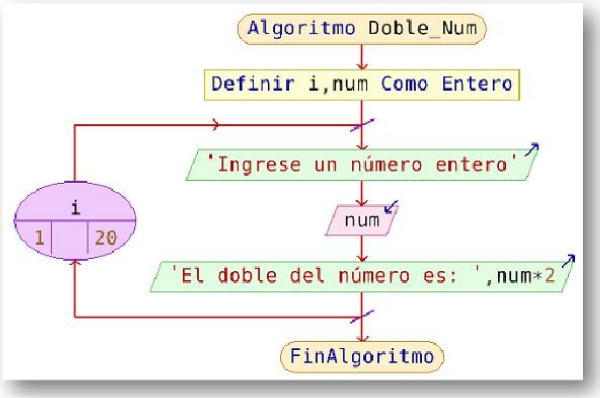
Donde VI es el valor inicial de la variable y VF es el valor final. La sección [con paso] es opcional, allí se establece en cuantas unidades se va ir modificando la variable que controla el ciclo, si se omite, la variable se irá aumentando o decrementando de 1 en 1 en cada iteración, según la relación entre VI y VF.

Generalmente, este el ciclo PARA se usa para "contar" de forma progresiva (Up-To). Pero también se puede usar para hacerlo de forma regresiva (Down-To):

	Progresiva	Regresiva
Relación entre los valores	VI <= VF	VI >= VF
Variación produce…en la variable índice	Incrementos	Decrementos
Signo del valor de la variación	Ninguno (signo positivo no se escribe antes del número)	- (signo negativo antecediendo al número)

Por ejemplo, si quiere leer 20 números enteros y mostrar su doble. En este caso el valor inicial es 1 y el valor final es 20, y la variable que controla el ciclo aumentara de 1 en 1 (comportamiento por omisión).

```
1 Algoritmo Doble_Num
2     definir i,num Como Entero
3     Para i=1 hasta 20 Hacer
4         Mostrar "Ingrese un número entero " Sin Saltar
5         Leer num
6         Mostrar "El doble del número es: " num*2
7     FinPara
8 FinAlgoritmo
```



El valor inicial no necesariamente tiene que ser 1 o 0, puede ser cualquier valor, que generalmente es menor que el valor final. El siguiente es un ejemplo:

```
1  Algoritmo Doble_Num
2      definir i,num Como Entero
3
4      Para i=50 hasta 75 Hacer
5          Mostrar "Ingrese un número entero " Sin Saltar
6          Leer num
7          Mostrar "El doble del número es: " num*2
8      FinPara
9
10 FinAlgoritmo
```

4.2.- Número de Iteraciones Variable

Los ciclos Up-To generalmente se usan para procesar lotes de datos, y la cantidad de veces que se va a iterar se le puede requerir al usuario para flexibilizar el algoritmo. De esta manera, el valor inicial debe ser 1 y el valor final dejaría de ser un valor fijo y dependería de una entrada, lo cual mantiene la predecibilidad de esta estructura iterativa. Véase el siguiente ejemplo:

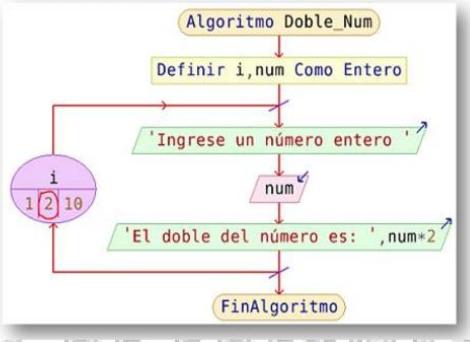
ACADEMIA DE SOFTWARE

```
1  Algoritmo Doble_Num
2      Definir i,num,n Como Entero
3
4      Mostrar "Ingrese cuántos números va a procesar" Sin Saltar
5      Leer N
6
7      Para i=1 hasta N Hacer
8          Mostrar "Ingrese un número entero " Sin Saltar
9          Leer num
10         Mostrar "El doble del número es: " num*2
11     FinPara
12
13 FinAlgoritmo
```

4.3.- Variando el Incremento o Decremento

El incremento de la variable típicamente se hace de 1 en 1, pero hay ocasiones donde se necesita que varíe en otros valores (2 en 2, 5 en 5, etc.). A continuación, se muestra un ejemplo de cómo cambiar el incremento de la variable que controla el ciclo usando la opción "con paso":

```
1  Algoritmo Doble_Num
2      Definir i,num Como Entero
3
4      // Este ciclo se repetirá 5 veces porque
5      // la variable i, se incrementará de 2 en 2
6      Para i=1 hasta 10 con paso 2 Hacer
7          Mostrar "Ingrese un número entero " Sin Saltar
8          Leer num
9          Mostrar "El doble del número es: " num*2
10     FinPara
11
12 FinAlgoritmo
```



ACADEMIA DE SOFTWARE

Con esta variación es posible realizar ciclos que se ejecuten en decremento (Down-To), donde el valor de la variable que controla el ciclo va desde un valor inicial mayor que el valor final. El decremento puede realizarse 1 en 1 (por omisión) o con otros valores, por ejemplo 2 en 2:

```
Algoritmo Cta_Regresiva
    Definir i Como Entero

    Para i=10 hasta 1 con paso -2 Hacer
        Mostrar i
    FinPara

FinAlgoritmo
```

```
PSeInt - Ejecutando proceso CTA_REGRESIVA
*** Ejecución Iniciac
10
8
6
4
2
*** Ejecución Finali:
```

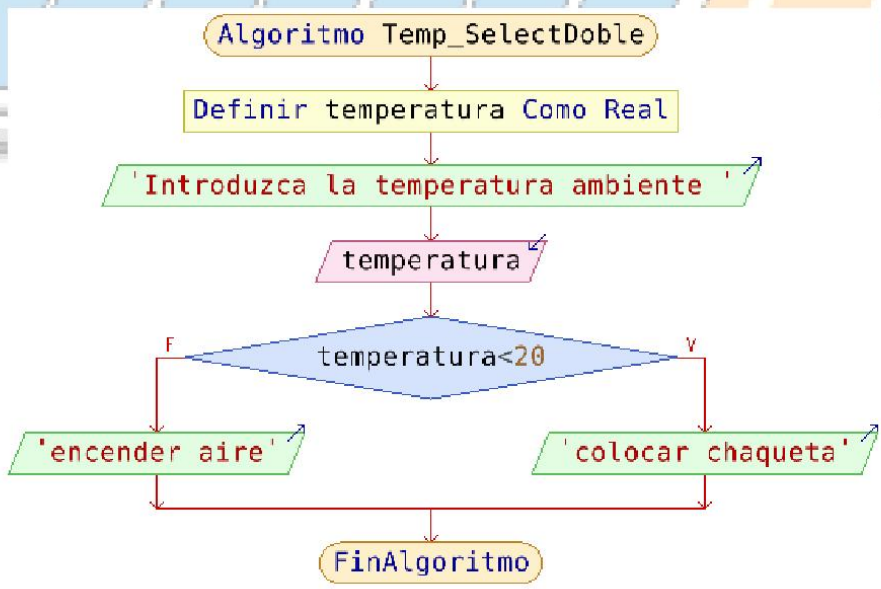
Capítulo 5. ESTRUCTURAS SELECTIVAS 2: SELECTIVA DOBLE

5.1.- Estructura Selectiva Doble

La estructura selectiva doble es una condicional que evalúa una condición, en caso de ser Verdadera, se ejecutan el conjunto de acciones asociadas al "entonces", de ser es Falsa se ejecuta el conjunto de acciones asociadas a la parte del "sino". En el ejemplo de la temperatura, se puede hacer algo en caso de que no haya frío:

```
Si temperatura < 20  entonces
        mostrar "colocar chaqueta"
sino
        mostrar "encender aire"
fin Si
```

Las 2 instrucciones de salida nunca podrán ejecutarse a la vez, o se ejecuta una o la otra.



El siguiente es un ejemplo completo de cómo usar una selectiva doble para calcular un monto de descuento, en función del monto al que se le aplicará. El planteamiento base es: Dado un monto calcular el descuento considerando que si el monto ingresado es mayor de 30000, el descuento a aplicar es 10% y si es igual o menor a dicho monto el descuento es 2%:

```
1  Algoritmo CalcDscto_Mto
2      Definir monto,descuento Como Real
3
4      Mostrar "Ingrese un monto"
5      Leer monto
6      Si monto > 30000 Entonces
7          descuento = monto * 10/100
8      Sino
9          descuento = monto * 2/100
10     FinSi
11
12     Mostrar "Descuento: " descuento " Bs."
13 FinAlgoritmo
```

ACADEMIA DE SOFTWARE

Es importante destacar que la instrucción selectiva doble no tiene una única forma de plantearse, debido a que se puede usar una expresión lógica diferente, que también permita lograr el objetivo. Por ejemplo, si se necesita saber si alguien es mayor o menor de edad para mostrar el mensaje que aplique, la selectiva se puede plantear de 2 formas equivalentes:

```
Algoritmo Msj_MayMenEdad_v1
    Definir edad Como Entero

    Mostrar "Introduzca la edad"
    Leer edad
    Mostrar "La persona es " Sin Saltar

    Si edad < 18
        Entonces Mostrar "menor de edad"
        Sino Mostrar "mayor de edad"
    FinSi

FinAlgoritmo
```

```
Algoritmo Msj_MayMenEdad_v2
    Definir edad Como Entero

    Mostrar "Introduzca la edad"
    Leer edad
    Mostrar "La persona es " Sin Saltar

    Si edad >= 18
        Entonces Mostrar "mayor de edad"
        Sino Mostrar "menor de edad"
    FinSi

FinAlgoritmo
```

Se puede notar que lo único que cambió entre un algoritmo y otro fue lo siguiente:

- En el 2do algoritmo se colocó la condición contraria o complementaria, a la planteada en el 1ero.
- La instrucción que está por el "entonces" en el 1ero se coloca en el "sino" del 2do y viceversa.

5.2.- Aplicaciones de la Selectiva Doble

Una de las aplicaciones más comunes de la selectiva doble, es la de determinar si un número es positivo o negativo. Para esto hay que determinar si un número es mayor que cero o menor que cero, partiendo de la premisa que el número es distinto de cero. Véase el siguiente algoritmo...¿que pasará si se ingresa 0?

```
1  Algoritmo Positivo_Negativo
2      Definir nro Como Entero
3
4      Mostrar "Ingrese un número distinto de 0"
5      Leer nro
6      Si nro > 0 Entonces
7          Mostrar "El número es positivo"
8      Sino
9          Mostrar "El número es negativo"
10     FinSi
11 FinAlgoritmo
```

Otra aplicación muy frecuente de la selectiva doble es la de determinar si un número es par o impar, lo cual implica determinar si ese número es divisible entre 2 o no. Para esto se utiliza el operador "resto de la división", que en PseInt es el "%" o el mod. Si el resto de la división de un número entre 2 es cero, significa que el número es par, de lo contrario, el número es impar. El siguiente es el algoritmo para determinar si un número es par o impar:

```
1  Algoritmo ParOImpar
2      Definir nro Como Entero
3
4      Mostrar "Ingrese un número"
5      Leer nro
6      Si (nro % 2) = 0 Entonces
7          Mostrar "El número es par"
8      Sino
9          Mostrar "El número es impar"
10     FinSi
11 FinAlgoritmo
```

Otra aplicación de este tipo de estructura es combinarlas con funciones, cuando se necesita hacer comparaciones de cadenas leídas por el teclado. En el ejemplo, se necesita determinar si un usuario introduce el valor "si"... En este caso, el usuario puede escribir "SI", "Si", "sI" o "si", lo cual haría muy complicado hacer la comparación, debido a que se tendría que comparar con cada una de las posibilidades que el usuario puede introducir. Usando la función mayúsculas o minúsculas se simplifica este problema, debido a que se convierte la cadena introducida por el usuario a un formato y luego se compara con el valor convertido:

ACADEMIA DE SOFTWARE

```
1  Algoritmo Eval_Respuesta
2      Definir respuesta como Caracter
3
4      Mostrar "¿Ud. es mayor de edad (Si/No)?"
5      Leer respuesta
6
7      Si Minusculas(respuesta) = "si" Entonces
8          Mostrar "Ud. respondió que si"
9      Sino
10         Mostrar "Ud. no respondió que si"
11     FinSi
12 FinAlgoritmo
```

Capítulo 6. ESTRUCTURAS SELECTIVAS 3: ANID. DE SELECTIVAS

6.1.- Anidación de Condicionales

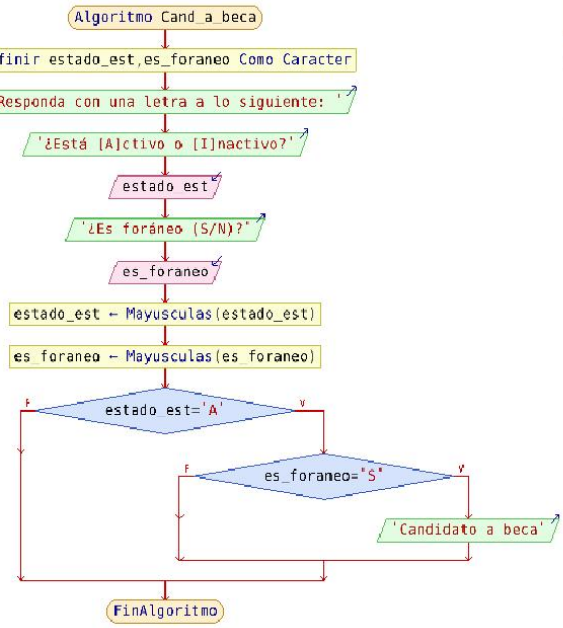
En muchas situaciones, se necesita plantear una nueva estructura condicional, después de verificar que la evaluación de una condición sea falsa o verdadera.

En estos casos, se recomienda utilizar la anidación o el anidamiento de sentencias condicionales, lo cual se refiere a la inclusión de estructuras selectivas dentro de otras.

6.2.- Anidación de Condicionales Por el Entonces

Las sentencias condicionales se pueden anidar por el ENTONCES. La ventaja de anidar sentencias condicionales así, es que la condicional que se plantee por el entonces solo se ejecutará al cumplirse la condición que acaba de ser evaluada.

Obsérvese este flujograma, donde se implementa este tipo de anidamiento. A través de él, se verifica si un estudiante está activo y es foráneo, para luego mostrar un mensaje:



Obsérvese el algoritmo correspondiente al flujograma anterior, donde se aprecia que deben cumplirse las 2 condiciones evaluadas (1 en cada selectiva) para que se muestre el mensaje:

```
Algoritmo Cand_a_beca
    Definir estado_est,es_foraneo como caracter
    Mostrar "Responda con una letra a lo siguiente: "
    Mostrar "¿Está [A]ctivo o [I]nactivo?" Sin Saltar
    Leer estado_est
    Mostrar "¿Es foráneo (S/N)?" Sin Saltar
    Leer es_foraneo
    estado_est=Mayusculas(estado_est)
    es_foraneo=Mayusculas(es_foraneo)
    Si estado_est = "A"
        entonces
            Si es_foraneo = "S"
                entonces Mostrar "Candidato a beca"
            FinSi
    FinSi
FinAlgoritmo
```

La sintaxis general de las estructuras condicionales anidadas por el ENTONCES es la siguiente:

ACADEMIA DE SOFTWARE

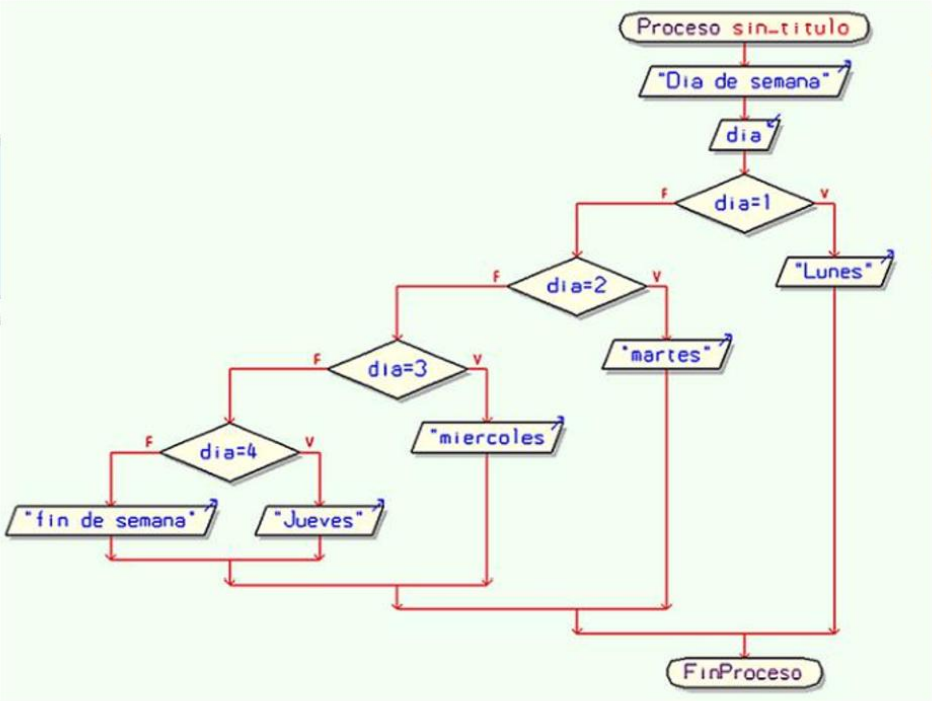
```
si <condicion> entonces
    . . .
    si <condicion> entonces
        . . .
        si <condicion> entonces
            . . .
            si <condicion> entonces
                . . .
            fin si

        fin si
    fin si
fin si
```

Cada una de las estructuras selectivas simples también podrían tener un SINO.

6.3.- Anidación de Condicionales Por el Sino

Las sentencias condicionales también se pueden anidar por el SINO. La ventaja de anidar sentencias condicionales así, es que cuando una se cumple no hay por qué evaluar las condiciones que están debajo o que le siguen.

En el presente flujograma el propósito es mostrar un mensaje dependiendo del número de día que sea ingresado, con base en la premisa de que el número de día sea válido, es decir, que esté entre 1 y 7.



El algoritmo textual del flujograma anterior se muestra a continuación...

```
1  Proceso sin_titulo
2      Escribir "Dia de semana"
3      Leer dia
4      Si dia=1 Entonces
5          Escribir "Lunes"
6      Sino
7          Si dia=2 Entonces
8              Escribir "martes"
9          Sino
10             Si dia=3 Entonces
11                 Escribir "miercoles"
12             Sino
13                 Si dia=4 Entonces
14                     Escribir "Jueves"
15                 Sino
16                     Escribir "fin de semana"
17                 FinSi
18             FinSi
19         FinSi
20     FinSi
21 FinProceso
```

La sintaxis general de las estructuras condicionales anidadas por el SINO es la siguiente:

ACADEMIA DE SOFTWARE

si <condicion> entonces
. . .
sino si <condicion> entonces
. . .
sino
si <condicion> entonces
. . .
sino si <condicion> entonces
. . .
fin si

fin si
finsi
fin si

Capítulo 7. ESTRUCTURAS SELECTIVAS 4: APLIC. DE ANIDAMIENTOS

7.1.- Comparación de Variables

Un ejemplo típico de uso de condicionales anidadas por el SINO es el de comparar 2 variables "var1" y "var2" para saber si son iguales o cual de las de 2 es la mayor.

Las 3 posibilidades pudiera ser necesario manejar son:
- que "var1" sea mayor que "var2"
- que "var2" sea mayor que "var1"
- que "var1" y "var2" sean iguales.

El siguiente es un ejemplo donde se comparan 2 variables para determinar cual es la relación entre ellas y mostrar el mensaje que aplica en cada caso:

```
Algoritmo Comparar_Edades
    Definir edad1,edad2 Como Entero

    Mostrar "Ingrese las edades a comparar "
    Mostrar "Edad 1" Sin Saltar
    Leer edad1
    Mostrar "Edad 2" Sin Saltar
    Leer edad2

    Si edad1>edad2
        Entonces Mostrar "La mayor edad es Edad 1"
        Sino Si edad1<edad2
                Entonces Mostrar "La mayor edad es Edad 2"
                Sino Mostrar "Edad 1 y Edad 2 son iguales"
            FinSi
    FinSi
FinAlgoritmo
```

7.2.- Evaluación de Intervalos Contiguos

Otra aplicación típica de los "si" anidados es la evaluación de intervalos contiguos. Esto se refiere, a verificar si un valor está en algún intervalo de varios intervalos que está uno

a continuación del otro. Por ejemplo, dada la edad de una persona se necesita saber cuál es su estado:

- Un bebé tiene una edad comprendida entre 0 y 2 años
- Un niño tiene edad comprendida entre 3 y 10 años
- Un adolescente tiene edad entre 11 y 17 años
- Un joven tiene edad entre 18 y 25 años
- Un adulto tiene edad entre 26 y 54 años
- Un adulto mayor tiene edad entre 55 y 74 años
- Un anciano tiene edad mayor a 74 años

En un caso como éste, una edad sólo puede estar en alguno de los intervalos definidos.

El algoritmo textual que resuelve el problema planteado es el siguiente:

```
Algoritmo Mensaje_segun_Edad
    Definir edad Como Entero
    Mostrar "Ingrese la edad de la persona"
    Leer edad
    Si edad>=0
       entonces
        Si edad<=2
           entonces mostrar "Es un bebé"
           sino Si edad<=10
                   entonces mostrar "Es un niño"
                   sino Si edad<=17
                           entonces mostrar "Es un adolescente"
                           sino Si edad<=25
                                   entonces mostrar "Es un joven"
                                   sino Si edad<=54
                                           entonces mostrar "Es un adulto"
                                           sino Si edad<=74
                                                   entonces mostrar "Es un adulto mayor"
                                                   sino mostrar "Es un anciano"
                                                FinSi
                                           FinSi
                                   FinSi
                           FinSi
                   FinSi
        FinSi
    FinSi
FinAlgoritmo
```

Cuando se evalúan intervalos contiguos, deben tenerse presente las siguientes reglas:

1) Siempre se evalúa 1 sola variable. Es decir, una sola variable es la que debe ser comparada con distintos valores a través de distintas selectivas dobles.

2) Se anidan las selectivas por el SINO y siempre se debe comparar con un solo valor en cada selectiva.

3) Las condiciones que se deben colocar en cada selectiva deben plantearse en cierto orden:

   - Ascendente => ir comparando con los topes superiores de c/rango. En el caso que se acaba de ver: comenzar a evaluar la edad a ver si es igual o menor a 2.

   - Descendente => ir comparando con los topes inferiores de c/rango. En el caso que se acaba de ver: comenzar
   - a evaluar la edad a ver si es mayor o igual a 74.

Esta regla, naturalmente aplica al valor contra el cual se comparará la variable en cada nueva selectiva planteada por el SINO.

ACADEMIA DE SOFTWARE

Capítulo 8. OPERACIONES LÓGICAS Y SELECTIVAS 1

8.1.- Operadores Lógicos

Cuando se estructuran expresiones lógicas, existen algunos casos en donde se requiere evaluar más de una condición al mismo tiempo, donde se evalúen varias variables o donde una variable se compara con varios valores.

Para crear ese tipo de expresiones lógicas, se emplean los Operadores Lógicos, siendo los más usados los siguientes:

Operación	Término	¿Qué implica?
Conjunción	**Y** *and*	Cumplimiento de todas condiciones entre las que media el operador
Disyunción	**O** *or*	Cumplimiento de alguna de las condiciones entre las que media el operador
Negación	**No** *not*	No cumplimiento de lo evaluado

8.2.- Expresión Lógica Compuesta

ACADEMIA DE SOFTWARE

Una Expresión Lógica Compuesta es una expresión donde se colocan 2 más condiciones simples unidas por operadores lógicos.

En muchas ocasiones, es necesario plantear este tipo de expresiones previo a la realización de algún procesamiento de información por parte del computador.

Por lo anterior, será necesario plantear en una misma expresión lógica la evaluación de más de una condición y/o relación, para lo cual podría ser necesario combinar operadores relacionales y operadores lógicos.

Cuando se evalúen expresiones lógicas compuestas, se evaluará primero el resultado de las expresiones lógicas simples y luego se aplicarán los operadores lógicos, aplicando la regla de asociatividad por la izquierda. Esto sin dejar de lado, que siempre se podrá dar prioridad a ciertas expresiones usando paréntesis.

Apréciese los siguientes ejemplos de expresiones lógicas compuestas usando el operador Y:

- que el computador muestre la boleta de un alumno, si éste estudia la carrera de medicina y su promedio de calificaciones es mayor de 70

  (carrera="medicina") Y (prom_nota>70)

- para entrar a un parque infantil el niño debe entre 2 y 8 años de edad:

  (edad >= 2) Y (edad <= 8)

Obsérvese la selectiva, para asignar un costo de entrada a un evento de 15000 Bs, sólo si una persona es menor de edad y es estudiante:

```
Si (edad<18) Y (es_estudiante=verdadero) Entonces
        Costo_Entrada=15000
FinSi
```

Ahora véase ejemplos de expresiones lógicas compuestas usando el operador O:

- que el computador verifique si se ingresó una A o una B

  (letra_ingresada="A") O (letra_ingresada="B")

- que una persona tenga menos de 15 años o más de 30 años

  (edad<15) O (edad>30)

Véase la selectiva en un caso donde están exentos de pagar la entrada a un espectáculo, los niños menores de 8 años y los adultos mayores de 65 años:

```
Si (edad < 8)  O  (edad > 65) Entonces
        Costo_Entrada=0
  FinSi
```

8.3.- Evaluación de Intervalos Discontiguos

Otra aplicación del Anidamiento de Selectivas es, usarlo para hacer evaluaciones dentro de intervalos discontiguos. Esto se refiere, a evaluar si un valor está en alguno de distintos intervalos que no están uno a continuación del otro. Así pues:

- Si se decidió evaluar los rangos de manera ascendente, el tope superior de un intervalo, no es el tope inferior del siguiente intervalo. Ejemplo:
  >10 y <=20
  >=25 y <40

- Si se decidió evaluar los rangos de manera descendente, el tope inferior de un intervalo, no es el tope superior del siguiente intervalo. Ejemplo:
  <40 y <=25
  <=20 y <10

Para lograr el objetivo de identificar dentro de cual rango está el valor a examinar, es necesario usa Expresiones Lógicas Compuestas.

Cabe destacar, que la comparación de las variables con los distintos rangos, no necesariamente deben estar ordenados a través de las selectivas anidadas. Esto se aprecia en el siguiente ejemplo: si se necesita evaluar si un valor ingresado está en algunos de estos rangos discontiguos:

ACADEMIA DE SOFTWARE

>5 y <10
<0 y >-10
>10 y <=50

```
Algoritmo intervalos_discontiguos
    Definir val como Entero
    Mostrar "Ingrese una valor" Sin Saltar
    Leer val
    Mostrar "El valor ingresado " Sin Saltar
    Si val>5 y val<10
        entonces Mostrar "es mayor a 5 y menor a 10"
        sino Si val<0 y val>(-10)
                entonces Mostrar "es negativo y mayor que -10"
                sino si val>10 y val<=50
                        entonces Mostrar "es mayor a 10 y menor o igual a 50"
                        sino Mostrar "no está en los rangos evaluados"
                    FinSi
            FinSi
    FinSi
FinAlgoritmo
```

Capítulo 9. OPERACIONES LÓGICAS Y SELECTIVAS 2

9.1.- ¿ Qué es Una Tabla de Verdad?

El resultado de la evaluación de expresiones lógicas con más de una condición se basa en las tablas de verdad.

Una Tabla de Verdad, o tabla de valores de verdad, es una tabla que despliega el valor de verdad de una proposición (expresión lógica) compuesta, para cada combinación de valores de verdad que pueda tomar sus componentes.

Considérese 2 proposiciones: A y B. Cada una puede tomar uno de dos valores de verdad: V (verdadero) o F (falso). Por lo tanto, los valores de verdad de A y B pueden combinarse de 4 maneras distintas:

- Ambas son verdaderas,
- A es verdadera y B falsa,
- A es falsa y B verdadera, o
- Ambas son falsas.

Esto puede expresarse con una tabla simple:

ACADEMIA DE SOFTWARE

A	B
V	V
V	F
F	V
F	F

En general, una tabla de verdad debe tener 2^n filas, donde n es la cantidad de proposiciones simples que están planteadas y cada fila representa cada una de las combinaciones de valores de verdad de las proposiciones.

Por ejemplo, la siguiente tabla tiene 3 proposiciones simples (A, B y C) y por lo tanto debe tener 8 filas (2^3).

A continuación, se verá como al evaluar 2 proposiciones con la mediación de los 3 operadores lógicos vistos, se obtiene los distintos valores de verdad según el operador empleado.

A	B	C
V	V	V
V	V	F
V	F	V
V	F	F
F	V	V
F	V	F
F	F	V
F	F	F

9.2.- Tabla de la Conjunción

La conjunción (Y) es un operador lógico que opera sobre dos valores de verdad, uno por cada una de dos proposiciones, devolviendo el valor de verdad verdadero cuando ambas proposiciones son verdaderas, y falso en cualquier otro caso.

Una conjunción es un enunciado con dos o más elementos requeridos. En PseInt el operador de conjunción puede ser la letra Y o el &. La tabla de verdad de la conjunción es la siguiente:

A	B	A y B
V	V	V
V	F	F
F	V	F
F	F	F

Véase las condiciones a evaluar en el caso donde para entrar a un parque infantil el niño debe tener de 3 a 7 años.

	A	B	A y B
	(edad>2)	(edad<8)	(edad>2) y (edad<8)
Edad=1	F	V	F
	(edad>2)	(edad<8)	(edad>2) y (edad<8)
Edad=3	V	V	V
	(edad>2)	(edad<8)	(edad>2) y (edad<8)
Edad=15	V	F	F
	(edad>2)	(edad<8)	(edad>2) y (edad<8)
Edad=2	F	V	F
	(edad>2)	(edad<8)	(edad>2) y (edad<8)
Edad=8	V	F	F

Véase las condiciones que deben cumplirse para saber si un participante CADI aprobó y es mayor de edad:

	A	B	A y B
	nota>=80	edad>=18	(nota>=80) y (edad>=18)
nota=80 / edad=40	V	V	V
	nota>=80	edad>=18	(nota>=80) y (edad>=18)
nota=85 / edad=17	V	F	F
	nota>=80	edad>=18	(nota>=80) y (edad>=18)
nota=60 / edad=29	F	V	F
	nota>=80	edad>=18	(nota>=80) y (edad>=18)
nota=50 / edad=15	F	F	F

9.3.- Tabla de la Disyunción

La disyunción (O) es un operador lógico que opera sobre dos valores de verdad, devolviendo el valor de verdad verdadero cuando una de las proposiciones es verdadera, o cuando ambas lo son, y falso cuando ambas son falsas.

Una disyunción es un enunciado con dos o más elementos optativos. En PseInt el operador de conjunción puede ser la letra O o el | (Este símbolo se escribe combinando Alt+124). La tabla de verdad de la disyunción es la siguiente:

A	B	A o B
V	V	V
V	F	V
F	V	V
F	F	F

Véase el ejemplo en el que el boleto para entrar un espectáculo es gratis para los niños menores de 8 años y los adultos mayores de 65 años

	A	B	A o B
	(edad<8)	(edad>65)	(edad<8) o (edad>65)
Edad=6	V	F	V
	(edad<8)	(edad>65)	(edad<8) o (edad>65)
Edad=9	F	F	F
	(edad<8)	(edad>65)	(edad<8) o (edad>65)
Edad=66	F	V	V
	(edad<8)	(edad>65)	(edad<8) o (edad>65)
Edad=8	F	F	F
	(edad<8)	(edad>65)	(edad<8) o (edad>65)
Edad=65	F	F	F

Véase las condiciones que deben cumplirse para saber si un estudiante debe ir a prueba sustitutiva, al tener una nota máxima o un promedio menor o igual a 10 (en la escala de 20):

	A	B	A o B
	nota_max<=10	promedio<=10	(nota_max<=10) o (promedio<=10)
nota_max=9 / promedio=10	V	V	V
	nota_max<=10	promedio<=10	(nota_max<=10) o (promedio<=10)
nota_max=8 / promedio=15	V	F	V
	nota_max<=10	promedio<=10	(nota_max<=10) o (promedio<=10)
nota_max=15 / promedio=9	F	V	V
	nota_max<=10	promedio<=10	(nota_max<=10) o (promedio<=10)
nota_max=19 / promedio=18	F	F	F

9.4.- Priorización de Evaluación en Expresiones Lógicas

Si se tienen varias condiciones en una expresión lógica compuesta la evaluación de las mismas se hará aplicando la regla de asociatividad por la izquierda.

Por ejemplo, si una persona está casada y es mayor de edad o es mayor de 50 años, la condición a evaluar sería la siguiente:

esCasado=verdadero Y esMayorDeEdad=verdadero O edad>50

Primero se evaluará si son verdaderas las variables lógicas esCasado Y esMayorDeEdad, y al tener ese resultado, éste evaluará con el operador de disyunción con lo que resulte de verificar si la edad es mayor a 50.

En algunos casos, es necesario usar paréntesis para agrupar las condiciones lógicas que desean evaluarse antes que otras cuando se están usando operadores lógicos.

Cabe destacar que, a mayor complejidad de la expresión lógica, más determinante será el uso de dichos símbolos.

Por ejemplo, si se quisiera hacer más específica la verificación condicional para la entrada preferencial a las personas de la 3ra edad: mujeres a partir de 55 y hombres a partir de 60, la condición a evaluar sería:

(genero="F" Y edad>=55) O (genero="M" Y edad>=60)

En esta condición se evalúan 2 datos de una persona, primero se evalúa si el género es femenino y su edad es mayor o igual a 55, luego se evalúa si el género es masculino y la edad mayor o igual a 60, y luego se evalúa la disyunción entre los resultados.

En el caso de verificar si la edad de una persona está comprendida entre 1 y 20, o entre 40 y 70, la condición sería:

(edad>=1 Y edad<=20) O (edad>=40 Y edad<=70)

En esta condición, primero se evalúa si la edad está en el primer rango, luego se evalúa si está en el segundo rango y por último se evalúa la disyunción entre los 2 resultados.
Si se desea evaluar si el tipo de boleto ingresado es igual a G (general) o A (abonado), y además que la persona pague usando tarjeta de débito o de crédito, la condición sería:

(tipo_entrada="G" O tipo_entrada="A") Y (medio_pago="TD" O medio_pago="TC")

En esta condición, primero se evalúa el tipo de entrada, luego el medio de pago y por último se evalúa la conjunción entre los 2 resultados.

En este ejemplo se aprecia la diferencia entre usar los paréntesis y no hacerlo.

```
Algoritmo Prec_Parentesis
    esEst=Verdadero // La persona es estudiante
    genero="f" // La persona es de genero femenino
    edad=25

    Mostrar edad>=30 y esEst=Verdadero o genero="f"
    Mostrar (edad>=30 y esEst=Verdadero) o genero="f"
    Mostrar edad>=30 y (esEst=Verdadero o genero="f")
FinAlgoritmo
```

```
PSeInt - Ejecutando
*** Ejecu
VERDADERO
VERDADERO
FALSO
*** Ejecu
```

Capítulo 10. OPERACIONES LÓGICAS Y SELECTIVAS 3

10.1.- Empleo de Variables Lógicas

Adicional a lo que se ha visto, las variables lógicas se pueden usar de varias maneras, por ejemplo, se puede asignar a una variable el resultado de una expresión lógica para después evaluarla. Obsérvese el siguiente ejemplo:

```
Algoritmo Manip_Var_Logicas1
    // Variable para registrar si los números son diferentes
    Definir Num1_DifA_Num2 como Logico
    // Variable para registrar si el 1° número es menor que el 2°
    Definir Num1_MenQ_Num2 como Logico

    Mostrar "Ingrese 2 números para compararlos"
    Leer Num1,Num2
    Num1_DifA_Num2=(Num1<>Num2)
    Num1_MenQ_Num2=(Num1<Num2)
    Si Num1_DifA_Num2 = Verdadero
        Entonces Mostrar Num1 " es diferente a " Num2
            Si Num1_MenQ_Num2 = Verdadero
                Entonces Mostrar Num1 " es menor que " Num2
                Sino Mostrar Num1 " es mayor que " Num2
            FinSi
        Sino Mostrar "Los números ingresados son iguales"
    FinSi
FinAlgoritmo
```

Para evaluar el valor de una variable lógica no es necesario compararla con "Verdadero" o "Falso".

Siempre y cuando el identificador sea significativo, se puede evaluar el valor de la variable colocando únicamente su nombre, en la instrucción donde se está evaluando la condición, al hacerlo se evaluará si la variable es verdadera.

En los siguientes ejemplos se muestra el valor de verdad de varias expresiones lógicas donde se combinan los operadores lógicos:

```
Algoritmo Manip_Var_Logicas3
    Definir nombre como Caracter
    Definir esCasado como Logico

    esCasado=falso
    esEstudiante=verdadero
    nombre="Martha"
    edad=45
    genero="F"

    Mostrar "Es " esCasado " que " nombre " está casada"
    Mostrar edad >=18
    Mostrar ~(edad>=18) // equivalente a edad<18
    Mostrar esEstudiante y genero="F" o genero="M" y edad<18
    Mostrar esEstudiante y (genero="F" o genero="M") y edad<18
    Mostrar ~(nombre="Mario") // equivalente a nombre<>Mario
    Mostrar ~(edad>=18 y genero<>"M")
    Mostrar ~(esCasado y esAdulto o n="Martha")
    Mostrar esCasado y (~esAdulto o n="Martha")
FinAlgoritmo
```

```
PSeInt - Ejecutando proceso MANIP_VAR_LOGICAS3
*** Ejecución Iniciada. ***
Es FALSO que Martha está casada
VERDADERO
FALSO
VERDADERO
FALSO
VERDADERO
FALSO
VERDADERO
FALSO
*** Ejecución Finalizada. ***
```

Como se puede evaluar una variable lógica sin comparar con verdadero o falso, se simplifica el ejemplo de comparación de los números:

```
Algoritmo Manip_Var_Logicas1
    // Variable para registrar si los números son diferentes
    Definir Num1_DifA_Num2 como Logico
    // Variable para registrar si el 1° número es menor que el 2°
    Definir Num1_MenQ_Num2 como Logico

    Mostrar "Ingrese 2 números para compararlos"
    Leer Num1,Num2
    Num1_DifA_Num2=(Num1<>Num2)
    Num1_MenQ_Num2=(Num1<Num2)
    Si Num1_DifA_Num2              // no se colocó =Verdadero
        Entonces Mostrar Num1 " es diferente a " Num2
            Si Num1_MenQ_Num2      // no se colocó =Verdadero
                Entonces Mostrar Num1 " es menor que " Num2
                Sino Mostrar Num1 " es mayor que " Num2
            FinSi
        Sino Mostrar "Los números ingresados son iguales"
    FinSi
FinAlgoritmo
```

10.2.- Tabla de la Negación

La negación (No) es un operador lógico que opera sobre un solo valor de verdad, devolviendo el valor de verdad falso cuando la proposición es verdadera, y verdadero cuando la proposición es falsa.

En PseInt el operador de conjunción puede ser la sílaba NO o el ~ (Este símbolo se escribe combinando Alt+126).

La tabla de verdad de la negación es la siguiente:

A	No A
V	F
F	V

En este ejemplo se aprecia lo que ocurre con la negación de la expresión lógica (edad>=20)

ACADEMIA DE SOFTWARE

	A edad >= 20	No A ~ (edad >= 20)
edad = 25	V	F

	edad >= 20	~ (edad >= 20)
edad = 17	F	V

Ahora véase el ejemplo de la negación la negación de la expresión lógica (nota<50)

	A	No A
	nota < 50	~ (nota < 50)
nota = 46	V	F

	nota < 50	~ (nota < 50)
nota = 75	F	V

10.3.- Uso de la Negación

En la programación, la negación es el operador que sirve para evaluar el valor contrario de una variable lógica o una condición.

Así pues, para evaluar:

- la posibilidad de que una variable lógica haya tomado un valor contrario al que tiene o
- el incumplimiento de una condición

Bastará con anteponerle el símbolo de la negación a la variable o a la condición.

En 2 selectivas del algoritmo mostrado, se están evaluando 2 variables lógicas, antecedidas por el operador de negación. Nótese que el primer mensaje no se muestra porque esMayorDeEdad es verdadero.

```
Algoritmo Manip_Var_Logicas2

    Definir esMayorDeEdad,esEstudiante como Logico

    esMayorDeEdad=Verdadero
    esEstudiante=Falso

    Si ~esMayorDeEdad // equivale a evaluar si esMayorDeEdad=Falso
        entonces Mostrar "La persona es menor de edad"
    FinSi
    Si ~esEstudiante // equivale a evaluar si esEstudiante=Falso
        entonces Mostrar "La persona no es estudiante"
    FinSi
FinAlgoritmo
```

```
PSeInt - Ejecutando proceso MANIP_VAR_LOGICAS2
*** Ejecución Iniciada. ***
La persona no es estudiante
*** Ejecución Finalizada. ***
```

Aquí se aprecian varios ejemplos de evaluación de expresiones lógicas donde también están implicados los operadores de negación, conjunción y disyunción:

```
Algoritmo Manip_Var_Logicas3
    Definir nomb,genero como caracter
    Definir esCasado,esMayorDeEdad como logico
    Definir edad como entero

    esCasado=falso
    esMayorDeEdad=verdadero
    nomb="Martha"
    edad=45
    genero="F"

    Mostrar ~(edad>=18) // ¿edad <18?
    Mostrar ~(edad>=18 y edad<=30) // ¿edad no está entre 18 y 30?
    Mostrar ~(nombre="Mario") // ¿nombre<>Mario?
    Mostrar ~(edad>=18 y genero ="M")
    mostrar ~(esCasado y esMayorDeEdad o nomb="Martha")
    mostrar ~esCasado y (esMayorDeEdad o nomb="Martha")
FinAlgoritmo
```

```
PSeInt - Ejecutan
*** Ejecu
FALSO
VERDADERO
VERDADERO
VERDADERO
FALSO
VERDADERO
*** Ejecu
```

Capítulo 11. ESTRUCTURAS SELECTIVAS 5: ALTERNATIVA MÚLTIPLE

11.1.- Estructura Según

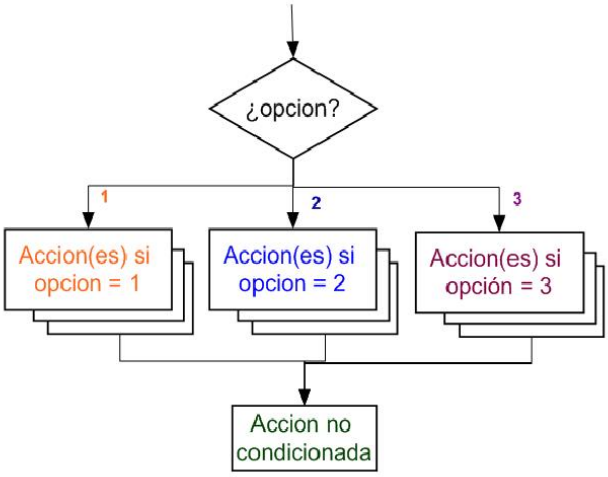
En la estructura selectiva de alternativa múltiple se evalúa una variable que puede tomar "n" valores distintos.

Según sea el valor de la variable en un instante dado, se ejecutarán las acciones asociadas a ese valor. En muchos lenguajes de programación se conoce a este tipo de selectiva como sentencias Case o Switch.

En pseudocódigo la sentencia se estructura de la siguiente manera:

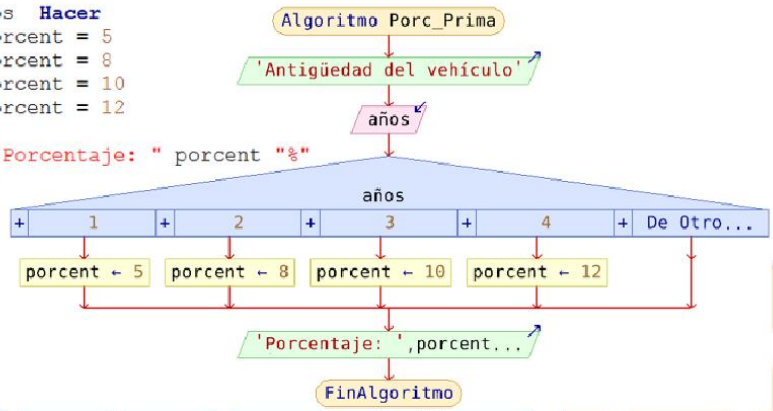
```
Según <variable>
    <valor1>:  <acciones 1>
    <valor2>:  <acciones 2>
      .
      .
    [<otro>:  <acciones por omisión> ]
fin segun
```

El siguiente es el flujograma general de la estructura selectiva de alternativa múltiple:



Véase este ejemplo... 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%:

```
Algoritmo Porc_Prima
    // Definición de variables
    Escribir "Antigüedad del vehículo"
    Leer años
    Segun años  Hacer
        1: porcent = 5
        2: porcent = 8
        3: porcent = 10
        4: porcent = 12
    FinSegun
    Mostrar "Porcentaje: " porcent "%"
FinAlgoritmo
```



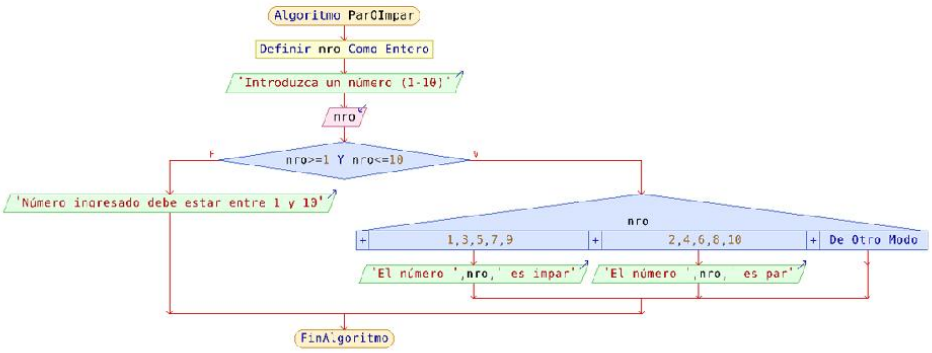
Ahora vea el pseudocódigo del siguiente ejemplo:

Se desea leer por teclado un número entre 1 y 10, con el objeto de visualizar si el número es par o impar.

ACADEMIA DE SOFTWARE

```
Algoritmo ParOImpar

    Definir nro Como Entero

    Mostrar "Introduzca un número (1-10)" Sin Saltar
    Leer nro

    Si nro>=1 y nro<=10
        Entonces
            Segun nro Hacer
                1,3,5,7,9: Mostrar "El número " nro " es impar"
                2,4,6,8,10: Mostrar "El número " nro " es par"
            FinSegun
        Sino Mostrar "Número ingresado debe estar entre 1 y 10"
    FinSi

FinAlgoritmo
```

Apreciése en el flujograma del ejemplo anterior, el anidamiento del Según dentro del entonces de la primera selectica.



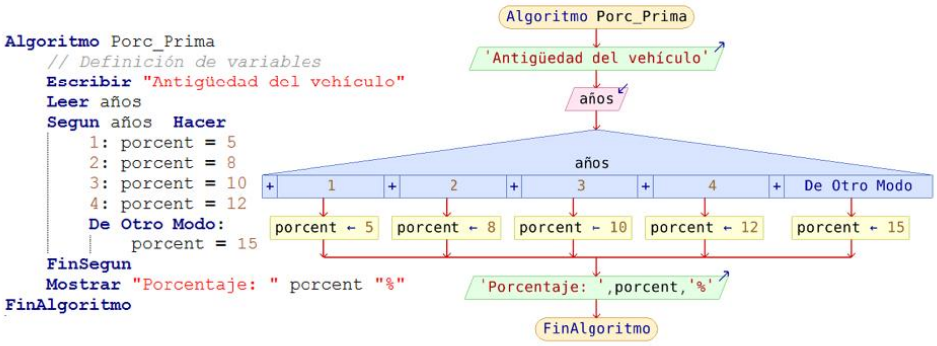
11.2.- Opción Otro

Esta estructura tiene una opción llamada "Otro:". Las acciones asociadas a esta opción 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'.

ACADEMIA DE SOFTWARE

El valor con el que se compara la variable es dependiente de los lenguajes de programación. Por lo general, se espera un tipo de dato determinístico (se conoce con certeza), tales como los enteros y caracteres. En general, esos valores pueden ser fijos, pueden ser variables o un conjunto de valores determinístico (se usa la coma para separar un valor de otro) o incluso otra condición (esto último no es implementable en PseInt).

Volviendo al ejemplo del porcentaje de prima, y agregando la opción OTRO, el enunciado varía, porque se le añade al final lo indicado después de los puntos suspensivos: Dependiendo de los años de antigüedad de un vehículo se le pagará una prima a su dueño... y si tiene cinco años o más la prima es de 15%. El algoritmo que resuelve esta situación es el siguiente:



11.3.- Según Vs. Anid. de Selectivas

La estructura "Según" equivale a un anidamiento de condicionales por el Sino, pero lo contrario siempre es cierto. Véase este ejemplo donde dependiendo del número que se ingrese se muestra el nombre del día de la semana o un mensaje de error:

```
Algoritmo Nomb_Dia_Semana
    Segun nroDia Hacer
        1: Escribir "Lunes"
        2: Escribir "Martes"
        3: Escribir "Miércoles"
        4: Escribir "Jueves"
        5: Escribir "Viernes"
        6,7: Escribir "Fin de semana"
        De Otro Modo:
            Escribir " ¡Error!..."
            Escribir "La semana tiene 7 dias"
    FinSegun
FinAlgoritmo
```

```
Algoritmo Nomb_Dia_Semana
  Si nroDia=1
    entonces Escribir "Lunes"
    sino si nroDia=2
        entonces Escribir "Martes"
        sino si nroDia=3
            entonces Escribir "Miércoles"
            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!..."
                        Escribir "La semana tiene 7 dias"
                      FinSi
                    FinSi
                FinSi
            FinSi
        FinSi
  FinSi
FinAlgoritmo
```

Capítulo 12. ESTRUCTURAS REPETITIVAS 3: REPETIR

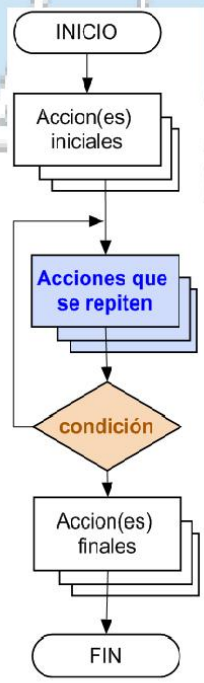
12.1.- Ciclo Repetir

La estructura iterativa REPETIR se utiliza cuando no se conoce de antemano la cantidad de iteraciones que ejecutará el ciclo.

En este ciclo se evalúa una expresión lógica (como las expresiones usadas en las condicionales "si").

Obsérvese en el flujograma general que en este tipo de iterativa, la condición que controla el ciclo se evalúa después de realizar la 1era iteración del bucle. Por lo tanto, se repiten las acciones dentro del bucle al menos una vez.

En este flujograma no están identificados los flujos de salida del rombo porque este tipo de iterativa tiene 2 variantes y el comportamiento de una es opuesto al de la otra.



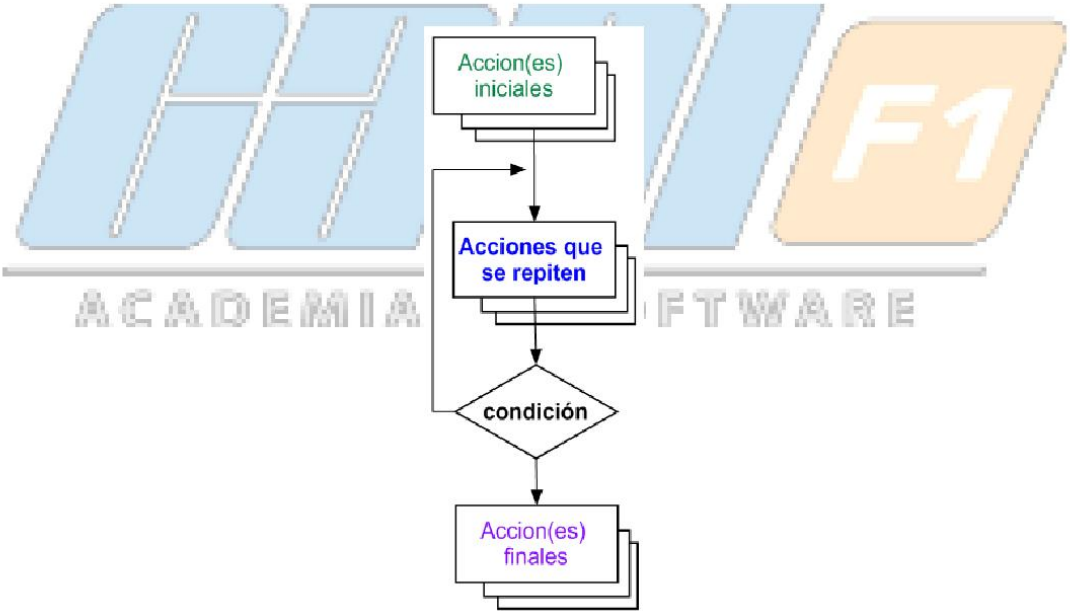
El ciclo "Repetir" es la apropiada cuando se requiera que un bucle se ejecute como mínimo una vez.

Cabe destacar que, la condición que controla el ciclo debe cambiar de estado en algún momento, de lo contrario se producirá en un ciclo infinito, lo cual sería un error lógico en la mayoría de los casos.

Como se refirió este tipo de iterativas tiene 2 variantes:

- Repetir-Hasta
- Repetir-Mientras

A continuación, se verá como se comporta cada una de ellas de forma conceptual y ejemplificada.



12.2.- Repetir...Hasta

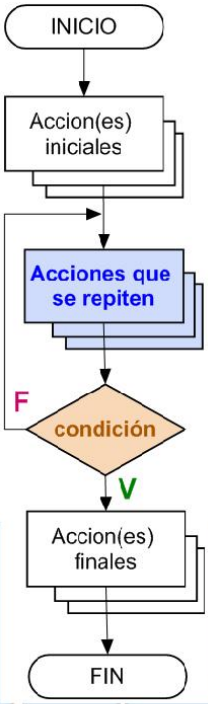
El bucle Repetir-Hasta (repeat...until) se repite siempre y cuando la condición evaluada al final sea Falsa.

En este ciclo al cumplirse la condición y volverse Verdadera, se rompe el bucle. De allí, su nombre “Repetir...Hasta” que se cumpla la condición.

La estructura general del ciclo "Repetir-Hasta" es la siguiente:

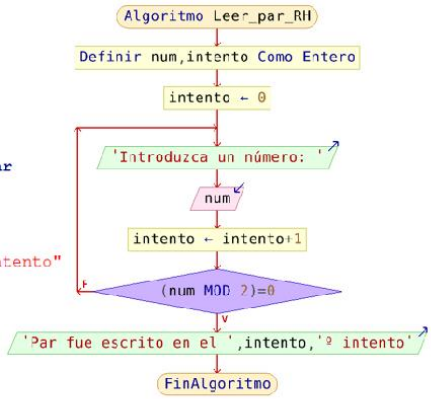
```
acciones_iniciales

Repetir

    accion_interna_1
    accion_interna_2

Hasta que <condicion>

acciones_finales
```

El flujograma general del ciclo “Repetir-Hasta” es el siguiente:



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

```
Algoritmo Leer_par_RH
    Definir num,intento Como Entero

    intento=0

    Repetir
        Mostrar "Introduzca un número: " Sin Saltar
        Leer num
        intento=intento+1
    Hasta Que (num mod 2) = 0

    Mostrar "Par fue escrito en el " intento "º intento"
FinAlgoritmo
```

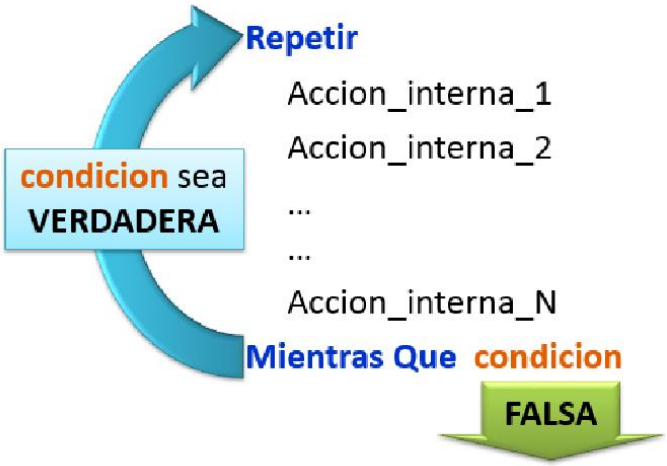


En el siguiente ejemplo se procesan números para mostrar su mitad. Nótese que la condición de ruptura es que el usuario responda usando una letra si desea interrumpir el procesamiento:

```
Algoritmo Calc_MitadNum_RH
    Definir num,nums_proc Como Entero
    Definir respuesta Como Caracter

    nums_proc = 0

    Repetir
        Escribir 'Introduzca un número: ' Sin Saltar
        Leer num
        Mostrar "La mitad de " num " es: " num/2
        nums_proc = nums_proc+1
        Mostrar "Si no desea procesar más números presione n ó N"
        Leer respuesta
    Hasta Que Mayusculas(respuesta)="N"

    Escribir "Se procesaron " num_proc " números"
FinAlgoritmo
```

12.3.- Repetir...mientras

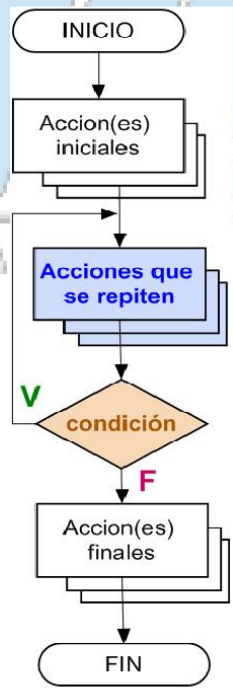
El bucle Repetir-Mientras (do...while) se repite siempre y cuando la condición evaluada al final sea Verdadera. En este ciclo al dejar de cumplirse la condición y volverse Falsa, se rompe el bucle. De allí su nombre "Repetir...Mientras" que se cumpla la condición.



La estructura general al usar el ciclo Repetir-Mientras es la siguiente:

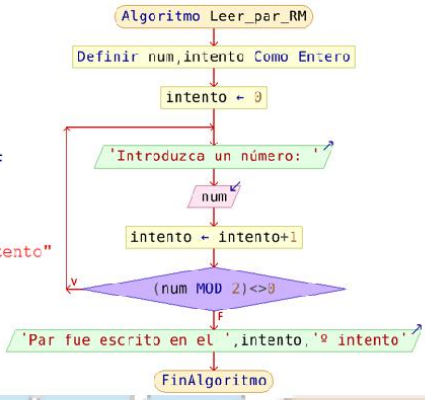
```
acciones_iniciales

Repetir

    accion_interna_1
    accion_interna_2

Mientras que <condicion>

acciones_finales
```

El flujograma general del ciclo "Repetir-Mientras" se muestra a continuación:



A continuación, se muestra un ejemplo, donde el usuario tendrá que escribir números mientras el número ingresado no sea un par.

```
Algoritmo Leer_par_RM
    Definir num,intento Como Entero

    intento=0

    Repetir
        Mostrar "Introduzca un número: " Sin Saltar
        Leer num
        intento=intento+1
    Mientras Que (num mod 2) <> 0

    Mostrar "Par fue escrito en el " intento "º intento"
FinAlgoritmo
```

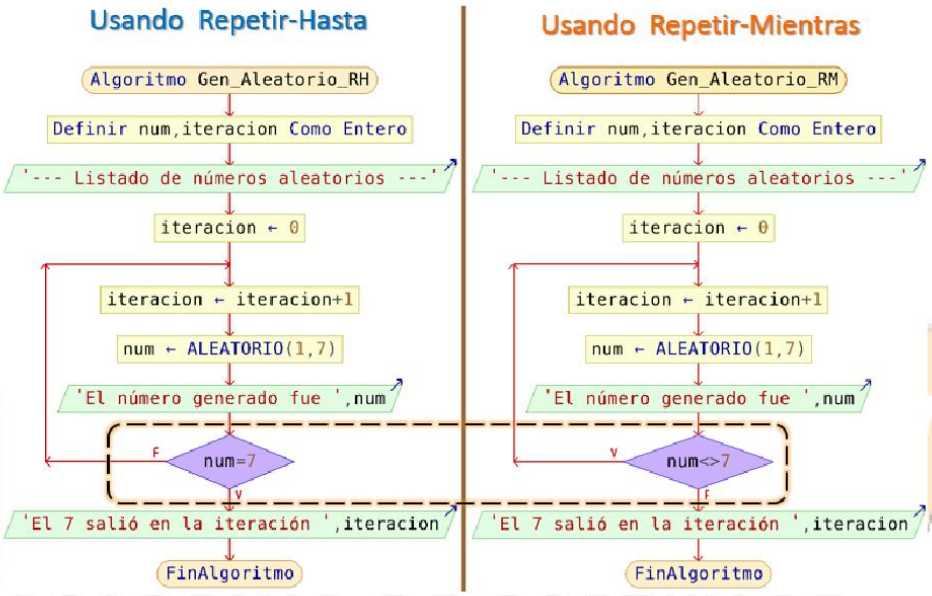


Para controlar el ciclo hasta que el usuario de una respuesta particular, también se puede usar el el "Repetir...Mientras". El siguiente algoritmo se basa en el ejemplo visto con anterioridad, el cual repite las acciones internas mientras la respuesta sea distinta del caracter evaluado:

```
Algoritmo Calc_MitadNum_RM
    Definir num,nums_proc Como Entero
    Definir respuesta Como Caracter

    nums_proc = 0

    Repetir
        Escribir 'Introduzca un número: ' Sin Saltar
        Leer num
        Mostrar "La mitad de " num " es: " num/2
        nums_proc = nums_proc+1
        Mostrar "Si no desea procesar más números presione n ó N"
        Leer respuesta
    Mientras Que Mayusculas(respuesta)<>"N"

    Escribir "Se procesaron " num_proc " números"
FinAlgoritmo
```

12.4.- Repetir-Hasta vs Repetir-Mientras

Los dos flujogramas representados logran el mismo objetivo: generar aleatorios hasta que sea generado el número 7. Nótese que las condiciones validadas son opuestas entre sí:



ACADEMIA DE SOFTWARE

Capítulo 13. ESTRUCTURAS REPETITIVAS 4: MIENTRAS

13.1.- Estructura Mientras

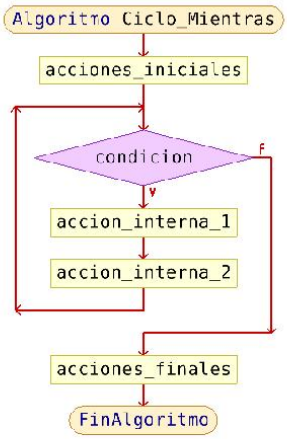
En la estructura repetitiva Mientras-Hacer (while), la condición de entrada al ciclo se evalúa antes de realizar cualquier iteración del bucle. De no cumplirse la condición, el ciclo no se ejecuta y el programa continúa con la secuencia de acciones siguientes al ciclo. Sólo si la condición se cumple 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.

Al evaluarse la condición, antes de entrar al bucle, si la condición es falsa no se iterará (bucle nunca se ejecuta). Por lo tanto, esta iterativa debe usarse cuando exista la posibilidad de que no se produzca ninguna iteración.

El ciclo "Mientras-Hacer" tal vez sea el ciclo menos utilizado, debido a que su empleo imperioso no se da en muchos algoritmos, ya que en la mayoría de los algoritmos se necesita dar por lo menos una iteración. En la siguiente imagen se muestra el flujograma general de este ciclo:

ACADEMIA DE SOFTWARE

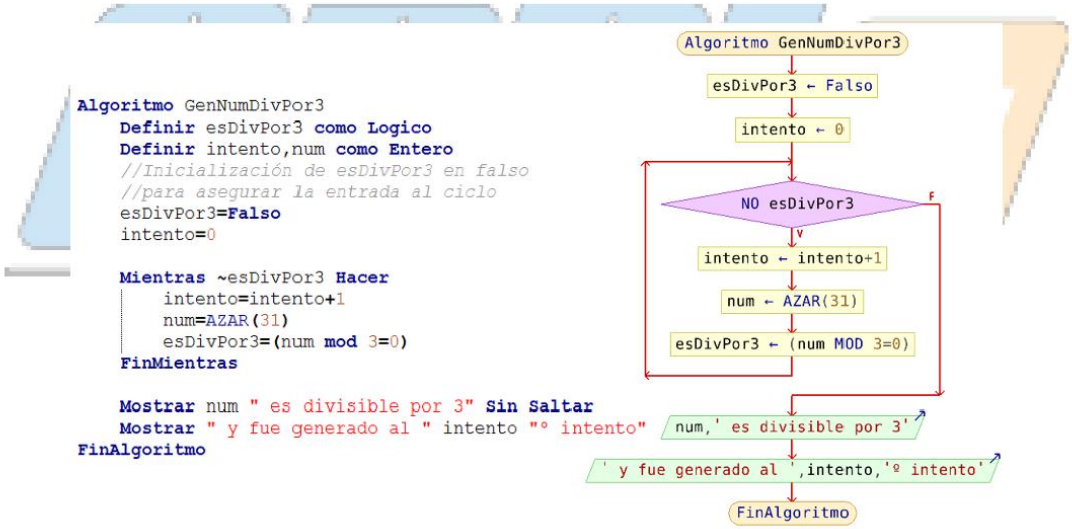


Este ciclo es usado comúnmente para examinar información en archivos (recorriéndolos mientras no se haya llegado al final del archivo), pero no se ejemplificará en este curso debido a que está fuera del alcance del mismo. En los siguientes algoritmos el objetivo es generar aleatorios con miras a obtener un número divisible entre 3.

En los primeros 2 ejemplos se implementa el ciclo Mientras evaluando si el número generado es divisible entre 3, que es la condición de ruptura del ciclo.

En el 3er ejemplo se añade otra condición, para controlar la ruptura del ciclo.

En esta 1era versión del algoritmo, a la variable lógica se le asigna una valor que asegure el cumplimiento de la condición la 1era vez para forzar la entrada al ciclo.



Cuando se usa el Mientras es posible que no se entre al ciclo.

En esta 2da versión, puede que no se entre al ciclo porque el primer número generado ya es divisible por 3, esto se ejemplifica en la ejecución.

```
Algoritmo GenNumDivPor3
    Definir esDivPor3 como Logico
    Definir intento,num como Entero

    intento=1
    num=AZAR(31) // generación del 1er número
    esDivPor3=(num mod 3 = 0)
    Mostrar "Es " esDivPor3 " que " num " divisible por 3"
    Mientras ~esDivPor3 Hacer
        intento=intento+1
        num=AZAR(31) //generación de nums. sucesivos
        esDivPor3=(num mod 3=0)
    FinMientras

    Si intento>1
       entonces
        Mostrar num " es divisible por 3" Sin Saltar
        Mostrar " y fue generado al " intento "º intento"
     sino
        Mostrar "¡No se entró al ciclo!"
    FinSi

FinAlgoritmo
```

```
PSeInt - Ejecutando proceso GENNUMDIVPOR3
*** Ejecución Iniciada. ***
Es VERDADERO que 27 divisible por 3
¡No se entró al ciclo!
*** Ejecución Finalizada. ***
```

ACADEMIA DE SOFTWARE

13.2.- Interrumpiendo Con Condiciones Compuestas

En ocasiones es necesario formular una condición compuesta para determinar cuándo se debe romper el ciclo. Variando el ejemplo anterior, para manejar la posibilidad que después de cierto número de intentos (5) no se genere un número divisible entre 3, se condiciona la ejecución también con ese dato y si al cabo de ellos, no se obtiene lo deseado se detiene el ciclo:

```
Algoritmo GenNumDivPor3
    Definir esDivPor3 como Logico
    Definir intento,num como Entero

    esDivPor3=Falso
    intento=0

    Mientras (~esDivPor3 y intento<=5) Hacer
        intento=intento+1
        num=AZAR(31)
        esDivPor3=(num mod 3=0)
    FinMientras

    Si esDivPor3
        entonces
            Mostrar num " es divisible por 3 " Sin Saltar
            Mostrar "y fue generado al " intento "° intento"
        sino
            Mostrar "Después de 5 intentos " Sin Saltar
            Mostrar "no se generó un número divisible por 3"
    FinSi
FinAlgoritmo
```

13.3.- Comparación Con Otros Ciclos

Un ciclo PARA simple se puede escribir como un ciclo MIENTRAS, pero no cualquier ciclo MIENTRAS se puede escribir como un ciclo PARA, por ej. cuando se evalúa más de una condición. El siguiente es un ejemplo del uso del ciclo MIENTRAS que típicamente se podría hacer como un ciclo PARA:

```
Algoritmo Msj_ParOImpar
    Definir i,N,num como Entero
    Mostrar "¿Cuántos números desea evaluar?"
    Leer N

    Para i=1 hasta N
        Mostrar "Ingrese el " i "° número"
        Leer num
        Si num mod 2=0
            entonces Mostrar num " es par"
            sino Mostrar num " es impar"
        FinSi

    FinPara
FinAlgoritmo
```

```
Algoritmo Msj_ParOImpar
    Definir i,N,num como Entero
    Mostrar "¿Cuántos números desea evaluar?"
    Leer N
    i=1
    Mientras i <= N
        Mostrar "Ingrese el " i "° número"
        Leer num
        Si num mod 2=0
            entonces Mostrar num " es par"
            sino Mostrar num " es impar"
        FinSi
        i=i+1 //Incremento de la variable índice
        // La i se puede modificar en un valor<>1
        // para emular la opción con paso
    FinMientras
FinAlgoritmo
```

En el capítulo del ciclo "Repetir" se vió un ejemplo de un algoritmo que está diseñado para leer números hasta que se introduce un número par.

A continuación, se plantea la comparación de las 2 variantes de ciclo "Repetir" con el algoritmo equivalente usando el ciclo "Mientras".

Para convertir un ciclo "Repetir..." en un ciclo "Mientras", se debe forzar que la condición evaluada en la iterativa se cumpla. Para ello, la variable involucrada en la condición debe inicializarse con un valor apropiado según sea el caso.

Véase la comparación del "Repetir...Hasta" con el "Mientras".

Nótese que la condición evaluada en el "Repetir...Hasta" es distinta a la planteada en el "Mientras"

```
Algoritmo Leer_par_RH
    Definir num,intento Como Entero
    intento=0
    Repetir
        Mostrar "Introduzca un número: " Sin Saltar
        Leer num
        intento=intento+1
    Hasta Que (num mod 2) = 0
    Mostrar "Par fue escrito en el " intento "° intento"
FinAlgoritmo
```

```
Algoritmo Leer_par_Mient
    Definir num,intento Como Entero
    intento=0
    num=1 //impar para forzar la entrada al ciclo
    Mientras (num mod 2) <> 0
        Mostrar "Introduzca un número: " Sin Saltar
        Leer num
        intento=intento+1
    FinMientras
    Mostrar "Par escrito en el " intento "° intento"
FinAlgoritmo
```

Véase la comparación del "Repetir...Mientras" con el "Mientras".

Nótese que la condición evaluada en ambos casos es la misma.

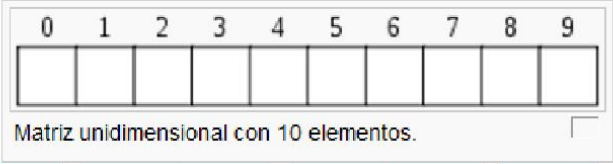
```
Algoritmo Leer_par_RM
    Definir num,intento Como Entero
    intento=0
    Repetir
        Mostrar "Introduzca un número: " Sin Saltar
        Leer num
        intento=intento+1
    Mientras Que (num mod 2) <> 0
    Mostrar "Par escrito en el " intento "º intento"
FinAlgoritmo
```

```
Algoritmo Leer_par_Mient
    Definir num,intento Como Entero
    intento=0
    num=1 //impar para forzar la entrada al ciclo
    Mientras (num mod 2) <> 0
        Mostrar "Introduzca un número: " Sin Saltar
        Leer num
        intento=intento+1
    FinMientras
    Mostrar "Par escrito en el " intento "º intento"
FinAlgoritmo
```

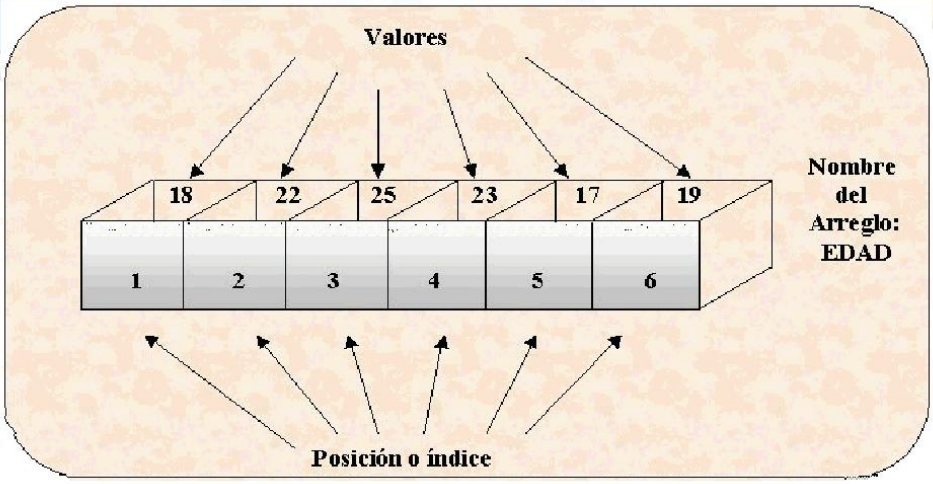
Capítulo 14. ARREGLOS 1: DEFINICIONES Y ACCESO

14.1.- Concepto

Un ARREGLO o vector es una estructura de almacenamiento unidimensional que tiene varias posiciones identificadas, donde regularmente se almacena un solo tipo de dato. Dichas posiciones se encuentran ubicados 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, donde se necesita preservar para un procesamiento posterior los distintos valores que va tomando una variable.

Los siguientes son ejemplos de problemas, que serían imposibles de resolver sin usar arreglos:

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

14.2.- 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 en todos es común que al declararlos se debe indicar el número de elementos que tendrá el arreglo. En pseudocódigo de PseInt se usa la siguiente notación:

```
Dimension nombre_arreglo[n]
```

ACADEMIA DE SOFTWARE

Donde:

- Dimension: 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 que se denominan vectores estáticos.

14.3.- Índice de Los Arreglos

Un arreglo se compone de un determinado número de elementos. Cada elemento es referenciado por la posición que ocupa dentro del vector. Dichas posiciones son accedidas por su índice, el cual de ser 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 tiene n componentes, la última tendrá como índice el valor n-1 (siendo n el tamaño del arreglo). En lenguaje C y sus derivados, 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 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:

ACADEMIA DE SOFTWARE

```
Algoritmo Acc_Arr_Edades
    Definir num Como Entero

    Dimension edades[50]  // Declaración del arreglo

    // Asignación de un valor constante en la 1era posición del arreglo
    edades[1] = 25

    num = 50
    // Asignación del valor de una variable en la 10ma posición del arreglo
    edades[10] = num

    // Evaluación del contenido de la posición 10 del arreglo
    Si edades[10] = num
        entonces Mostrar "En la posición 10 está almacenado el valor " num
    FinSi
FinAlgoritmo
```

Capítulo 15. ARREGLOS 2: LECTURA Y MOSTRADO

15.1.- Visitar Las Posiciones Del Arreglo

Antes de ver como guardar información en el arreglo o mostrar información contenida en él, se debe saber cómo visitar cada elemento del arreglo.

Para examinar un arreglo muchas veces es preciso recorrerlos. Esto se realiza por medio de las estructuras de control repetitivas, referidas como ciclos. Generalmente se utiliza el ciclo "Para", pero se puede usar también los ciclos "Mientras" o "Repetir".

Para visitar las N posiciones de un arreglo, con bastante frecuencia se utiliza el ciclo PARA:

```
Para i=1 hasta N hacer
....

// acceso a la posición i del arreglo
Fin para
```

También se puede usar una variante de dicho ciclo llamada PARA CADA (foreach). En ella, después de esas 2 palabras reservadas se coloca un identificador (referido al dato a acceder), luego la palabra reservada DE y otro identificador que es el nombre del arreglo a recorrer:

```
Para Cada <elemento> De <Arreglo> Hacer
...
// acceso a la posición actual del arreglo
Fin para
```

Este ciclo realizará tantas iteraciones como elementos contenga el arreglo, y en c/u de ellos el primer identificador servirá para referirse al elemento visitado del arreglo en cuestión.

El siguiente pseudocódigo muestra 2 algoritmos típicos para recorrer un arreglo usando los ciclos "para" y "para cada" con el objeto de aplicar una operación a cada elemento, en este caso, llenar las posiciones con valores de 2 en 2:

```
Algoritmo Arr_Pares_v1

    Definir i Como Entero
    Dimension arr_par[50]

    Para i=1 hasta 50 Hacer
        arr_par[i]=i*2
    FinPara

FinAlgoritmo
```

```
Algoritmo Arr_Pares_v2

    Definir i Como Entero
    Dimension arr_par[50]
    i=1
    Para cada num de arr_par
        num=i*2
        i=i+1
    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

15.2.- 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 que cambia de valor en cada iteración, se guarda en una posición de un arreglo. El siguiente ejemplo muestra como guardar la nota de 20 alumnos usando un ciclo PARA:

```
Algoritmo LectArr_Notas_Para

    Dimension Notas[20] //Dimensionamiento del arreglo

    // *** Instrucciones para cargar del arreglo
    Para i=1 hasta 20 hacer
        Mostrar "Ingrese la " i "° nota" Sin Saltar
        leer Notas[i]
        // en cada iteración en la posición i
        // se guardará la nota leída
    FinPara
FinAlgoritmo
```

Como se acaba de referir, otra forma de cargar los arreglos es usar la instrucción iterativa PARA CADA. Véase una forma equivalente para cargar un arreglo con 20 notas:

```
Algoritmo LectArr_Notas_PC
    Definir nota como real
    Dimension Notas[20]
    Mostrar "Ingrese una a una las 20 notas a guardar"
    Para cada nota de Notas hacer
        Leer nota
        // en cada iteración se leerá una dato en nota
        // y se guardará en la posición del arreglo Notas
        // que se esté visitando
    FinPara
FinAlgoritmo
```

15.3.- 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:

ACADEMIA DE SOFTWARE

```
Algoritmo MostArr_Notas_Para
    Definir i Como Entero
    Dimension Notas[20]
    // *** Instrucciones para cargar del arreglo
    Para i=1 hasta 20 hacer
        Mostrar i "° nota: " Notas[i]
        // en cada iteración se mostrará lo
        // que se encuentra en Notas[i]
    FinPara
FinAlgoritmo
```

También se puede mostrar el arreglo usando la iterativa PARA CADA (foreach), donde se comienza a recorrer el arreglo desde la primera posición, y en cada iteración se va mostrando el contenido de la posición visitada.

```
Algoritmo MostArr_Notas_PC
    Definir nota como real
    Dimension Notas[20]
    // *** Instrucciones para cargar del arreglo
    Mostrar "Las 20 notas ingresadas fueron: "
    Para cada nota de Notas
        Mostrar nota
        // en cada iteración se mostrará la nota almacenada
        // en la posición que se esté visitando
    FinPara
FinAlgoritmo
```

ACADEMIA DE SOFTWARE