Universidad Politécnica de la Región Ribereña

Algoritmos

```
VARIABLES
   Edad, cmayor, cmenor, inc son enteras
INICIO
   Cmayor=0, cmenor=0
   PARA inc=1 HASTA 10 HACER
      LEER edad
      SI edad >= 18 ENTONCES
         Cmayor = cmayor + 1
      SINO
         Cmenor = cmenor + 1
      FIN SI
   FIN PARA
   ESCRIBIR "Total de personas mayores:", cmayor
   ESCRIBIR "Total de personas menores:", cmenor
FIN
```

***OBJETIVO**. Al término de la materia el estudiante será capaz de dar solución a problemas cotidianos mediante el uso de algoritmos utilizando estructuras secuenciales simples, dobles o múltiples, así como el uso de estructuras repetitivas y tipos de datos estructurados como los arreglos.*

Curso

Nivel Básico – Avanzado

Duración: 75 Hrs

Edición del documento:
M. en A. Héctor Flores Gutiérrez

UPR

INDICE

Algoritmos.

Métodos de Solución de Problemas

Casi inconscientemente, los humanos efectuamos cotidianamente una serie de pasos, procedimientos o acciones que nos permiten alcanzar un resultado o resolver un problema.

Esta serie de pasos, procedimientos o acciones, comenzamos a aplicarlas muy temprano en la mañana cuando, por ejemplo, decidimos tomar un baño.

Posteriormente cuando pensamos en desayunar también seguimos una serie de pasos que nos permiten alcanzar un resultado específico: tomar el desayuno.

La historia se repite innumerables veces durante el día. Continuamente seguimos una serie de pasos o conjunto de acciones que nos permiten alcanzar un resultado

Estamos en realidad aplicando un algoritmo para resolver un problema.

"Formalmente definimos un algoritmo como un conjunto de pasos, procedimientos o acciones que nos permiten alcanzar un resultado o resolver un problema"

Muchas veces aplicamos el algoritmo de manera inadvertida, inconscientemente o automáticamente. Esto generalmente se produce cuando el problema que tenemos en frente lo hemos resulto con anterioridad un gran número de veces.

Por otra parte, existe una gran cantidad de problemas que requieren de un análisis profundo y de un pensamiento flexible y estructurado para su solución.

Ejemplo: Construya un algoritmo para preparar "Pechugas de pollo en salsa de elote y chile poblano".

Ingredientes (para 6 personas)
3 pechugas deshuesadas, sin piel y partidas a la mitad
1 diente de ajo
4 gramos de pimienta negra
Sal
6 cucharadas de aceite
5 chiles poblanos asados y limpios
½ taza de leche
¼ taza de crema ligera
1 lata de crema de elote

Algoritmo (Preparación).

- Muela el ajo, la pimienta y un poco de sal y únteselo a las pechugas
- Caliente el aceite y dore las pechugas
- Licue los chiles con la leche y la crema, y mézclelos con la crema de elote.
- En una fuente coloque las pechugas y báñelas con la mezcla anterior.
- Cubra el platón con papel aluminio y hornee a 200° C, durante 15 minutos.

Conceptos Fundamentales

Dentro de los conceptos fundamentales, trataremos algunos que son necesarios para la construcción de algoritmos, diagramas de flujo y programas. Primero analizaremos los tipos de datos.

Tipos de datos

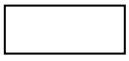
Los datos a procesar por una computadora pueden clasificarse en:

- Simples
- Estructurados

La principal característica de los datos simples es que ocupan sólo una casilla de memoria (fig. 1A). Por lo tanto, una variable simple hace referencia a un único valor a la vez. Dentro de este grupo de datos se encuentran: enteros, reales, caracteres, booleanos, enumerados y subrangos.

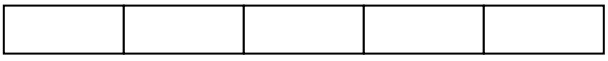
Los datos estructurados se caracterizan por el hecho de que con un nombre (identificador de variable estructurada) se hace referencia a un grupo de casillas de memoria (fig. 1B). Es decir, un dato estructurado tiene varios componentes. Cada uno de los componentes puede ser a su vez un dato simple o estructurado. Dentro de este grupo de datos se encuentran: arreglos, cadena de caracteres, registros y conjuntos.

Identificador



1A

identificador



1B

Datos Numéricos

Dentro de los tipos de datos numéricos encontramos los enteros y los reales. Los enteros son números que pueden estar precedidos del signo + o -, y que no tienen parte decimal. Por ejemplo:

128 528 714 530 6235 14780

Los reales son números que pueden estar precedidos del signo + o -, y que tienen una parte decimal. Por ejemplo:

7.5 128.0 -37.865 129.7 16000.50 -15.0

Datos Alfanuméricos

Dentro de este tipo de datos encontramos los de tipo carácter (simple) y de cadena de caracteres (estructurado). Son datos cuyo contenidos pueden ser letras del abecedario (a, b, c,…, z), dígitos (0, 1, 2,…, 9) o símbolos especiales (#, $, ^, *, %, /, !, +, -, …, etc.). Debemos remarcar que aunque este tipo de datos pueden contener números, no pueden ser utilizados para realizar operaciones aritméticas.
Un tipo de dato carácter contiene un solo carácter, y se escribe entre apóstrofes. Por ejemplo:

'a' 'B' '$' '9' '-' '#' 'f'

Un dato tipo cadena de caracteres contiene un conjunto de caracteres, y se escribe entre comillas. La longitud de una cadena depende de los lenguajes de programación, aunque normalmente se acepta una longitud máxima de 255.

"abcde" "$9#7" "Carlos Gómez""Rosario" "754-27-22"

Datos Lógicos

Dentro de este tipo de datos encontramos los booleanos. Son datos que sólo pueden tomar dos valores: verdadero (true) o falso (false).

Identificadores, Constantes y Variables

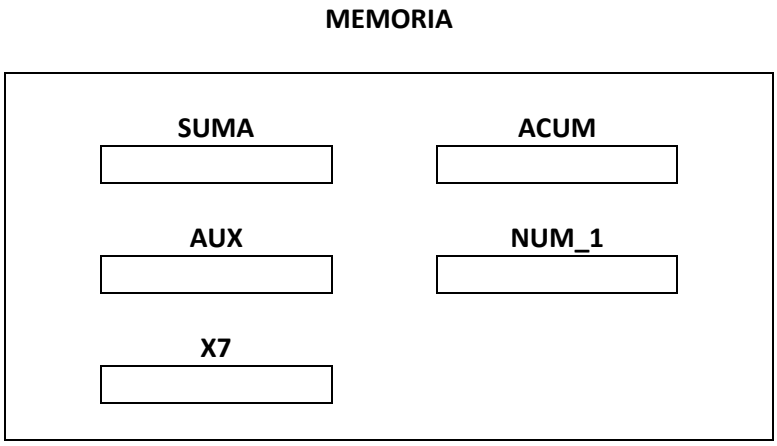
Identificadores

Los datos a procesar por una computadora, ya sean simples o estructurados, deben almacenarse en casillas o celdas de memoria para su posterior utilización. Estas casillas o celdas de memoria (constantes o variables) tienen un nombre que permite su identificación.

Llamaremos identificador al nombre que se les da a las casillas de memoria. Un identificador se forma de acuerdo a ciertas reglas:

- El primer carácter que forma un identificador debe ser una letra (a, b, c, …,z)
- Los demás caracteres pueden ser letras (a, b, c, …,z), dígitos (0, 1, 2, …, 9) o el siguiente símbolo especial: _
- La longitud del identificador es igual a 7 en la mayoría de los lenguajes de programación.

En la siguiente figura podemos observar ejemplos de identificadores.

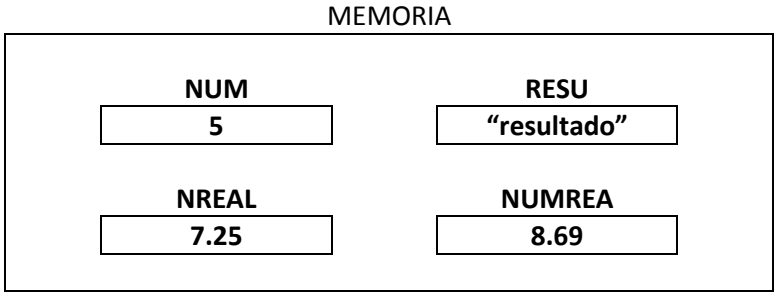


Casillas de memoria con los nombres de identificadores

Constantes

Las constantes son datos que no cambian durante la ejecución de un programa. Para nombrar las constantes utilizamos los identificadores que mencionamos anteriormente. Existen tipo de constantes como tipos de datos, por lo tanto, puede haber constantes de tipo entero, real, carácter, cadena de caracteres, etc.

Observe en la figura que se muestra en la parte de abajo, la constante NUM es de tipo entero, NREAL y NUMREA son de tipo real, y RESU de tipo cadena de caracteres. Estas constantes no cambiarán su valor durante la ejecución del programa. Es muy importante que los nombres de las constantes sean representativos de la función que tienen las mismas en el programa.

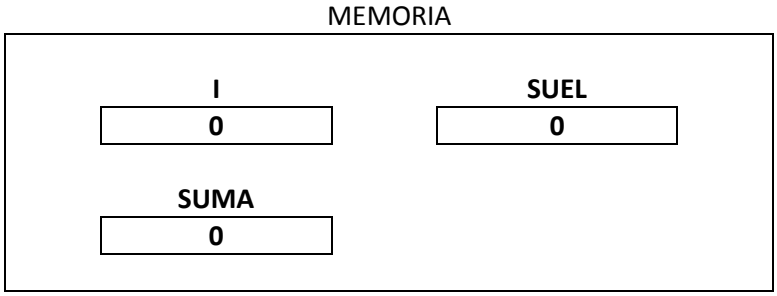


Constantes representadas en memoria

Variables

Las variables son objetos que pueden cambiar su valor durante la ejecución de un programa. Para nombrar las variables utilizaremos los identificadores. Al igual que las constantes, pueden existir tipos de variables como tipos de datos.

En la siguiente figura, la variable I es de tipo entero, tiene un valor inicial de cero y cambiará su valor durante la ejecución del programa. Las variables SUEL y SUMA son de tipo real, están inicializadas con el valor de cero, y al igual que la variable I, seguramente cambiarán su valor durante la ejecución del programa.



Variables representadas en memoria

Debemos remarcar que los nombres de las variables deben ser representativos de la función que cumplen en el programa.

Operaciones Aritméticas.

Para poder realizar operaciones aritméticas necesitamos de operadores aritméticos. Estos operadores nos permitirán realizar operaciones aritméticas entre operandos: números, constantes o variables. El resultado de una operación aritmética será un número.

En la siguiente tabla presentamos los operadores aritméticos, la operación que pueden realizar, un ejemplo de su uso y el resultado.

Tabla de Operadores Aritméticos			
Operador Aritmético	**Operación**	**Ejemplo**	**Resultado**
**	Potencia	4 ** 3	64
*	Multiplicación	8.25 * 7	57.75
/	División	15 / 4	3.75
+	Suma	125.8 + 62.50	188.28
-	Resta	65.30 – 32.33	32.97
mod	Módulo(residuo)	15 mod 2	1
div	División entera	17 div 3	5

Al evaluar expresiones que contienen operadores aritméticos debemos respetar la jerarquía en el orden de aplicación. Es decir, si tenemos en una expresión más de un operador, debemos aplicar primero el operador de mayor jerarquía, resolver esa operación, y así sucesivamente. Es importante señalar que el operador () es un operador asociativo que tiene la prioridad más alta en cualquier lenguaje de programación. En la siguiente tabla se presenta la jerarquía de los operadores.

Jerarquía de los Operadores Aritméticos		
Operador	**Jerarquía**	**Operación**
**	(mayor)	Potencia
*, /, mod, div	↓	Multiplicación, división, módulo, división entera
+, -	(menor)	Suma, resta

Las reglas para resolver una expresión aritmética son las siguientes:

1. Si una expresión contiene subexpresiones entre paréntesis, éstas se evalúan primero; respetando claro está la jerarquía de los operadores aritméticos en esta subexpresión. Si las subexpresiones se encuentran anidadas por paréntesis, primero se evalúan las subexpresiones que se encuentran en el último nivel de anidamiento.
2. Los operadores aritméticos se aplican teniendo en cuenta la jerarquía y de izquierda a derecha

A continuación en el siguiente ejemplo presentamos varios casos y la forma de resolver los mismos.

Caso a.

$$\underline{7 + 5} - 6$$
$$\underline{12 - 6}$$
$$6$$

Caso b.

$$9 + \underline{7 * 8} - 36 / 5$$
$$9 + 56 - \underline{36 / 5}$$
$$\underline{9 + 56} - 7.2$$
$$\underline{65 - 7.2}$$
$$57.8$$

Caso c.

$$7 * \underline{5 ** 3} / 4 \text{ div } 3$$
$$\underline{7 * 125} / 4 \text{ div } 3$$
$$\underline{875 / 4} \text{ div } 3$$
$$\underline{218.75 \text{ div } 3}$$
$$72$$

Caso d.

7 * 8 (160 mod 3 ** 3) div 5 * 13 – 28
7 * 8 * (160 mod 27) div 5 * 13 – 28
7 * 8 * 25 div 5 * 13 – 28
56 * 25 div 5 * 13 – 28
1400 div 5 * 13 – 25
280 * 13 – 25
3640 – 28
3612

Caso e.

15 / 2 * (7 + (68 – 15 * 33 + (45 ** 2 / 16) / 3) / 15) + 19
15 / 2 * (7 + (68 – 15 * 33 + (2025 / 16) / 3) / 15) + 19
15 / 2 * (7 + (68 – 15 * 33 + 126.5625 / 3) / 15) + 19
15 / 2 * (7 + (68 – 495 + 126.5625 / 3) / 15) + 19
15 / 2 * (7 + (68 – 495 + 42.1875) / 15) + 19
15 / 2 * (7 + (-427 + 42.1875) / 15) + 19
15 / 2 * (7 + (-)384.8125 / 15) + 19
15 / 2 * (7 + (-)25.6541) + 19
15 / 2 * (-)18.6541 + 19
7.5 * (-)18.6541 + 19
-139.9062 + 19
-120.9062

Ejemplos de operaciones aritméticas.

(2+3)*4-17 div 2+1 5*4-17 div 2+1 20-17 div 2 +1 20-8+1 12+1 13	**((3 + 2)**2 – 15) / 2 * 5** (5**2 - 15) / 2 * 5 (25 - 15) / 2 * 5 10 / 2 * 5 5 * 5 25
2 * 1 + .05 * 4 – 1 / 5 * 10 2 + .05 * 4 – 1/5 * 10 2 + 0.2 - 0.2 * 10 2.2 – 2 0.2	**2**(1 – 1) / (1 * 10) / (1 / 10)** 2**0 / (1 * 10) / (1 / 10) 1 / 10 / 0.1 0.1/0.1 1
1 – 4 + 10 + 3 – 5 + 5 -3 + 10 + 3 – 5 + 5 7 + 3 – 5 + 5 10 – 5 + 5 5 + 5 10	**-3 * 1 + 2 * 4 – 1 / 2 * 10** -3 + 2 * 4 – 1 / 2 * 10 -3 + 8 – 1 / 2 * 10 -3 + 8 – 0.5 * 10 -3 + 8 – 5 5 – 5 0
2 * 4^2 – 6 * 4 + 12 2 * 16 – 6 * 4 + 12 32 – 6 * 4 + 12 32 – 24 + 12 20	**(2 * 4)**2 – 6 * 4 + 12** 8**2 – 6 * 4 + 12 64 – 24 + 12 40 + 12 52
(4(2 * 1) – 6 * (10 / 10)) / 2** (4**2 – 6 * (10 / 10)) / 2 (16 – 6 * (10 / 10)) / 2 (16 – 6 *1) / 2 (16 - 6) / 2 10 / 2 5	**-1 + (10**2 – 4 * 1 * 4)**2 / 2 / 1** -1 + (100 – 4 * 1 * 4)**2 / 2 / 1 -1 + (100 - 16)**2 / 2 / 1 -1 + (84)**2/ 2 / 1 -1 + 7056 / 2 / 1 -1 + 3528 3527
(3 + (2 + 1 / 2 * 1^2) – (1 – (1 + 3^3 + 5 / 1) / 2)) (3 + (2 + 0.5 * 1) – (1 – (1 + 27 + 5 / 1) / 2)) (3 + 2.5 – (1 – (1 + 27 + 5) / 2)) (3 + 2.5 – (1 – 33 / 2)) (3 + 2.5 – (1 - 16.5)) 3 + 2.5 + 15.5 21	**(3^(5 – 1) / 2 * 1 – (1 + (2 - 1)^2)) * 2 – 2** (3^4 / 2 * 1 – (1 + 1^2)) * 2 – 2 (81 / 2 * 1 – (1 + 1)) * 2 – 2 (81 / 2 * 1 - 2) * 2 – 2 (40.5 - 2) * 2 – 2 38.5 * 2 – 2 77 – 2 75

Expresiones Lógicas

Las expresiones lógicas o booleanas, llamadas así en honor del matemático George Boole, están constituidas por números, constantes o variables y operadores lógicos o relacionales. El valor que pueden tener estas expresiones es el de verdadero o falso. Se utilizan frecuentemente en las estructuras selectivas (dependiendo del resultado de la evaluación se toma por un determinado camino alternativo) y en las estructuras repetitivas (dependiendo del resultado de la evaluación se continúa con el ciclo o se interrumpe al mismo)

Operadores Relacionales

Los operadores relacionales son operadores que permiten comparar dos operandos. Los operandos pueden ser números, alfanuméricos, constantes o variables. Las constantes o variables, a su vez, pueden ser de tipo entero, real, carácter o cadena de caracteres. El resultado de una expresión con operadores relaciones es verdadero o falso.

En la siguiente tabla presentamos los operadores relacionales, la operación que pueden realizar, un ejemplo de su uso y el resultado.

Operadores Relacionales			
Operador	**Operación**	**Ejemplo**	**Resultado**
=	Igual que	“hola” = “lola”	FALSO
< >	Diferente a	‘a’ <> ‘b’	VERDADERO
<	Menor que	7 < 15	VERDADERO
>	Mayor que	22 > 11	VERDADERO
<=	Menor o igual que	15 <= 22	VERDADERO
>=	Mayor o igual que	35 >= 20	VERDADERO

En el siguiente ejemplo presentamos varios casos de expresiones lógicas con operadores relacionales y la forma de resolver las mismas.

Caso a.

A = 5
B = 16

(A ** 2) > (B * 2)
25 > (B * 2)
25 > 32
FALSO

Caso b.

X = 6
B = 7.8

(X * 5 + B ** 3 / 4) <= (X ** 3 div B)

(X * 5 + 474.552 / 4) <= (X ** 3 div B)

(30 + 474.552 / 4) <= (X ** 3 div B)

(30 + 118.638) <= (X ** 3 div B)

148.638 <= (X ** 3 div B)

148.638 <= (216 div B)

148.638 <= 27

FALSO

Caso c.

((1580 mod 6 * 2 ** 7) > (7 + 8 * 3 ** 4)) > ((15 * 2) = (60 * 2 / 4))
((1580 mod 6 * 128) > (7 + 8 * 3 ** 4)) > ((15 * 2) = (60 * 2 / 4))
((2* 128) > (7 + 8 * 3 ** 4)) > ((15 * 2) = (60 * 2 / 4))
(256 > (7 + 8 * 3 ** 4)) > ((15 * 2) = (60 * 2 / 4))
(256 > (7 + 8 * 81)) > ((15 * 2) = (60 * 2 / 4))
(256 > (7 + 648)) > ((15 * 2) = (60 * 2 / 4))
(256 > 655) > ((15 * 2) = (60 * 2 / 4))
FALSO > ((15 * 2) = (60 * 2 / 4))
FALSO > (30 = (60 * 2 / 4))
FALSO > (30 = (120 / 4))
FALSO > (30 = 30)
FALSO > VERDADERO
FALSO

NOTA: Cuando se utilizan los operadores de relación con operandos lógicos, ***falso*** es menor que ***verdadero***.

Operadores Lógicos

Los operadores lógicos son operadores que permiten formular condiciones complejas a partir de condiciones simples. Los operadores lógicos son de conjunción (y), disyunción (o) y negación (no). En la siguiente tabla presentamos el operador lógico, la expresión lógica y significado de dicha expresión, teniendo encuentra la jerarquía correspondiente.

Operadores Lógicos	
Operador Lógico	**Jerarquía**
NO, NOT	Mayor
Y, AND	
O, OR	Menor

Las siguientes tablas son las tablas de verdad de los operadores lógicos

A	NOT A
FALSO	VERDADERO
VERDADERO	FALSO

A	B	A AND B
FALSO	FALSO	FALSO
FALSO	VERDADERO	FALSO
VERDADERO	FALSO	FALSO
VERDADERO	VERDADERO	VERDADERO

En una evaluación del tipo (Y o AND), habiendo un FALSO, el resultado que producirá será un FALSO.

A	B	A OR B
FALSO	FALSO	FALSO
FALSO	VERDADERO	VERDADERO
VERDADERO	FALSO	VERDADERO
VERDADERO	VERDADERO	VERDADERO

En una operación del tipo (O u OR), habiendo un VERDADERO, el resultado que producirá será un VERDADERO

Por último, la siguiente tabla es la jerarquía correspondiente de todos los operadores (aritméticos, relacionales y lógicos).

Jerarquía de los Operadores	
Operadores	**Jerarquía**
( )	Mayor
**	
*, /, div, mod	
+, -	
=, <>, <, >, <=, >=	
NOT	
AND	Menor
OR	

A continuación presentamos varios casos y la forma de resolver los mismos.

Caso 1.

NOT(15 >= 7 ** 2) OR (43 – 8 * 2 DIV 4 <> 3 * 2 DIV 2)
NOT(15 >= 49) OR (43 – 8 * 2 DIV 4 <> 3 * 2 DIV 2)
NOT FALSO OR (43 – 16 DIV 4 <> 3 * 2 DIV 2)
NOT FALSO OR (43 – 4 <> 3 * 2 DIV 2)
NOT FALSO OR (43 – 4 <> 6 DIV 2)
NOT FALSO OR (43 – 4 <> 3)
NOT FALSO OR (39 <> 3)
NOT FALSO OR VERDADERO
VERDADERO OR VERDADERO
VERDADERO

Caso 2.

(15 >= 7 * 3 ** 2 AND 8 > 3 AND 15 > 6) OR NOT(7 * 3 < 5 + 12 * 2 DIV 3 ** 2)
(15 >= 7 * 9 AND 8 > 3 AND 15 > 6) OR NOT(7 * 3 < 5 + 12 * 2 DIV 3 ** 2)
(15 >= 63 AND 8 > 3 AND 15 > 6) OR NOT(7 * 3 < 5 + 12 * 2 DIV 3 ** 2)
(FALSO AND VERDADERO AND VERDADERO) OR NOT(7 * 3 < 5 + 12 * 2 DIV 3 ** 2)
(FALSO AND VERDADERO) OR NOT(7 * 3 < 5 + 12 * 2 DIV 3 ** 2)
FALSO OR NOT(7 * 3 < 5 + 12 * 2 DIV 3 ** 2)
FALSO OR NOT(21 < 5 + 24 DIV 9)
FALSO OR NOT(21 < 5 + 2)
FALSO OR NOT(21 < 7)
FALSO OR NOT FALSO
FALSO OR VERDADERO
VERDADERO

Caso 3.

NOT((7 * 3 DIV 2 * 4) > (15 / 2 * 6 >= 15 * 2 / 17 = 15))
NOT((21 DIV 2 * 4) > (15 / 2 * 6 >= 15 * 2 / 17 = 15))
NOT((10 * 4) > (15 / 2 * 6 >= 15 * 2 / 17 = 15))
NOT(40 > (15 / 2 * 6 >= 15 * 2 / 17 = 15))
NOT(40 > (7.5* 6 >= 15 * 2 / 17 = 15))
NOT(40 > (45 >= 30 / 17 = 15))
NOT(40 > (45 >= 1.75 = 15))
NOT(40 > (VERDADERO = 15))
ERROR

NOTA: No se puede realizar la comparación entre un valor lógico y un numérico, utilizando un operador relacional.

Ejemplos de operaciones con todos los operadores.

5 – 2 > 4 AND NOT 0.5 = 1 / 2
3 > 4 AND NOT 0.5 = 0.5
FALSO AND NOT VERDADERO
FALSO AND FALSO
FALSO

3 * 5**2 > 84 OR 2 * 3 * 1 <= 10
75 > 84 OR 6 <= 10
FALSO OR VERDADERO
VERDADERO

NOT (2 * 1 = 4 / 2 OR (3 - 5) * 10 > 4) = (NOT 1 / 2 = 2 * 1) AND NOT 4 < (3 – 5) * 10
NOT (2 = 2 OR -2 * 10 > 4) = (NOT 0.5 = 2) AND NOT 4 < -2 * 10
NOT VERDADERO OR -20 > 4 = NOT FALSO AND NOT 4 < -20
FALSO OR FALSO = VERDADERO AND NOT FALSO
FALSO = VERDADERO AND VERDADERO
FALSO = VERDADERO
FALSO

NOT (1 > 1 * 4) AND 5 < (2 **(1 + 1) * 1 / 4)
NOT (1 > 4) AND 5 < (2**2 * 1 / 4)
NOT FALSO AND 5 < (4 * 1 / 4)
VERDADERO AND 5 < 1
VERDADERO AND FALSO
FALSO

Bloque de Asignación

Un bloque de asignación se utiliza para asignar valores o expresiones a una variable. La asignación es una operación destructiva. Esto significa que si la variable tenía asignado un valor, éste se destruye, conservando ahora el nuevo valor. El formato de la asignación es el siguiente:

Variable ← expresión o valor

Donde: expresión puede ser aritmética o lógica, o una constante o variable. Observemos a continuación el siguiente ejemplo:

Supongamos que las variables I, ACUM y J son de tipo entero, REA y SUM de tipo real, CAR de tipo carácter y BAND de tipo booleano. Consideremos también que tenemos que realizar las siguientes asignaciones:

1. I ← 0
2. I ← I + 1
3. ACUM ← 0
4. J ← 5 ** 2 DIV 3
5. CAR ← ‘a’
6. ACUM ← J DIV I
7. REA ← ACUM / 3
8. VAND ← (8 > 5) AND (15 < 2 ** 3)
9. SUM ← ACUM * 5 / J ** 2
10. I ← I * 3
11. REA ← REA / 5
12. BAND ← BAND O (I = J)
13. I ← REA
14. CAR ← J

En la siguiente tabla podemos observar los valores que van tomando las variables en memoria.

MEMORIA

Número de Asignación	I	J	ACUM	REA	SUM	CAR	BAND
1	~~0~~						
2	~~1~~						
3			~~0~~				
4		8					
5						~~'a'~~	
6			8				
7				~~2.66~~			
8							~~FALSO~~
9					0.625		
10	~~3~~						
11				0.532			
12							FALSO
13	Error						
14						Error	

Observe que en la asignación número 13 a la variable I se le asigna una variable de tipo real, por lo que se produce un error. Lo mismo ocurre en la asignación número 14, a la variable tipo carácter CAR se le asigna una variable de tipo entero, por lo que también se produce un error.

Variables Contadores.- Un contador es una variable (casi siempre de tipo entero) cuyo valor se incrementa o decrementa según se requiera. Es habitual llamar a esta variable "cont" (de contador) o "i" (de índice).

El contador suele usarse de este modo:
Primero, se inicializa antes de su uso. Es decir, se le da un valor inicial. Por ejemplo:
cont ← 0

Segundo, se modifica a lo largo del algoritmo
cont ← cont + 1

Esto quiere decir que el valor de la variable "cont" se incrementa en una unidad y es asignado de nuevo a la variable contador. Es decir, si cont valía 0 antes de esta instrucción, cont valdrá 1 después de su ejecución.

Otra forma típica del contador es:
cont ← cont - 1

En este caso, la variable se decrementa en una unidad; si cont valía 5 antes de la instrucción, tendremos que cont valdrá 4 después de su ejecución.

El incremento o decremento no tiene por qué ser de una unidad. La cantidad que haya que incrementar o decrementar vendrá dada por la naturaleza del problema.

Tercero, se utiliza en la condición de salida de un bucle. Normalmente, se compara con el valor máximo (o mínimo) que debe alcanzar el contador para dejar de repetir las instrucciones del bucle.

Variables Acumuladoras.- Las variables acumuladoras tienen la misión de almacenar resultados sucesivos, es decir, de acumular resultados, de ahí su nombre.

Las variables acumuladoras también deben ser inicializadas. Si llamamos "acum" a un acumulador, escribiremos antes de su uso algo como esto:
acum ← 0

Por supuesto, el valor inicial puede cambiar, dependiendo de la naturaleza del problema. Más tarde, en el cuerpo del algoritmo, la forma en la que nos la solemos encontrar es:
Acum ← acum + N

Siendo N otra variable.

Variables Conmutadoras.- Un conmutador (o interruptor) es una variable que sólo puede tomar *dos valores*. Pueden ser, por tanto, de tipo booleano, aunque también pueden usarse variables enteras o de tipo carácter.

La variable conmutador recibirá uno de los dos valores posibles antes de su uso. Cuando se usa un conmutador para controlar un ciclo, dentro del cuerpo del ciclo, debe cambiarse ese valor bajo ciertas condiciones. Utilizando el conmutador en la condición de salida del ciclo, puede controlarse el número de repeticiones.

Algoritmo 01. Dado un conjunto de datos de entrada(A,B,C,D), invierte el orden de los mismos cuando los imprime

```
VARIABLES
    A,B,C y D Son variables de tipo entero
INICIO
    Leer A,B,C,D
    Escribir D,C,B,A
FIN
```

Algoritmo 02. Dado los datos enteros A y B, escriba el resultado de la siguiente expresión:

$$\frac{(A + B)^2}{3}$$

Datos:

A, B: Variables de tipo entero
RESUL: Variable real

VARIABLES
A,B son enteras
RESUL es real
INICIO
Leer A,B
Hacer RESUL ← (A + B)**2/3
Escribir RESUL
FIN

Prueba de escritorio

NÚMERO DE CORRIDA	DATOS		RESULTADO
	A	B	RESUL
1	5	6	40.33
2	7	10	96.33
3	0	3	3.00
4	12	2	65.33
5	14	-5	27.00

Algoritmo 03. Dadas 3 calificaciones enteras, calcule el promedio de ellas.

Estructura básica de un algoritmo e instrucciones.

Un algoritmo está compuesto de la siguiente estructura:

Teoría sobre el problema a resolver
VARIABLES
En esta sección de declaran las variables a utilizar y su tipo de dato
INICIO *Marca el inicio para la solución del problema*
En esta sección se desarrolla el problema a resolver utilizando las instrucciones que se requieran tales como: Leer, Escribir. Tomas de decisiones simples o múltiples y ciclos
FIN *Marca el inicio para la solución del problema*

Algoritmo 04. Dados 2 valores enteros A y B proporcionados por el usuario, escribir los resultados de las operaciones básicas. (suma, resta, mult y división).

```
VARIABLES
    A, B, suma, resta, mult son enteros
    división es real
INICIO
    Leer A, B
    Suma = A + B
    Resta = A – B
    Mult = A * B
    División = A / B
    Escribir suma, resta, mult, division
FIN
```

Algoritmo 05. Convertir grados Fahrenheit a Celsius. Plantear la fórmula.

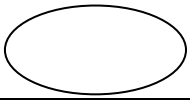
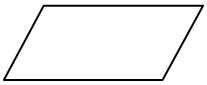
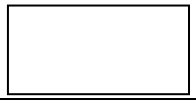
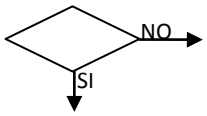
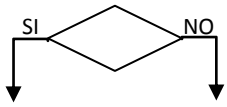
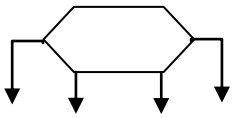
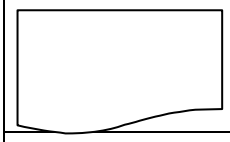
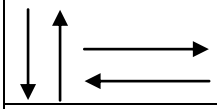
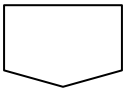
```
VARIABLES
    Fahrenheit, Celsius son reales
INICIO
    Leer Fahrenheit
    Celsius = (Fahrenheit - 32) * 5 / 9
    Escribir Celsius
FIN
```

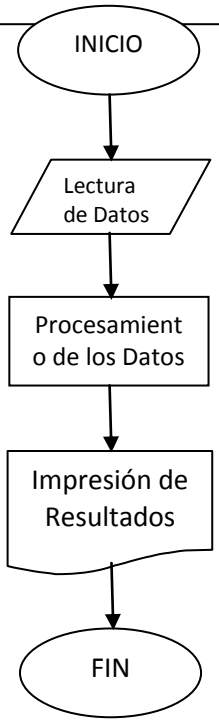
Algoritmo 06. Convertir grados Celsius a Fahrenheit. Plantear la fórmula.

```
VARIABLES
    Fahrenheit, Celsius son reales
INICIO
    Leer Celsius
    Fahrenheit = Celsius * 9 / 5 + 32
    Escribir Fahrenheit
FIN
```

Diagramas de Flujo

Un diagrama de flujo representa la esquematización gráfica de un algoritmo. En realidad muestra gráficamente los pasos o procesos a seguir para alcanzar la solución de un problema. Su correcta construcción es sumamente importante porque a partir del mismo se escribe un programa en algún lenguaje de programación. Si el diagrama de flujo está completo y correcto, el paso del mismo a un lenguaje de programación es relativamente simple y directo.

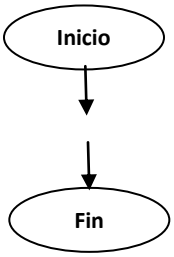
Símbolos utilizados en los Diagramas de Flujo	
Representación del símbolo	**Explicación del símbolo**
	Símbolo utilizado para marcar el inicio y el fin del diagrama de flujo.
	Símbolo utilizado para introducir los datos de entrada. Expresa lectura.
	Símbolo utilizado para representar un proceso. En su interior se expresan asignaciones, operaciones aritméticas, cambios de valor de las celdas en memoria, etc.
	Símbolo utilizado para representar una decisión. En su interior se almacena una condición, y dependiendo del resultado de la evaluación de la misma se sigue por una de las ramas o caminos alternativos. Este símbolo se utiliza en la estructura selectiva *si entonces* y en las estructuras repetitivas *repetir* y *mientras*.
	Símbolo utilizado para representar la estructura selectiva doble *si entonces/sino*. En su interior se almacena una condición. Si el resultado es verdadero se continúa por el camino de la izquierda, y si es falso por el camino de la derecha.
	Símbolo utilizado para representar una decisión múltiple. En su interior se almacena un selector, y dependiendo del valor de dicho selector se sigue por una de las ramas o caminos alternativos. Este símbolo se utiliza en la estructura selectiva *si múltiple*.
	Símbolo utilizado para representar la impresión de un resultado. Expresa *escritura*.
	Símbolos utilizados para expresar la dirección del flujo del diagrama.
	Símbolo utilizado para expresar conexión dentro de una misma página
	Símbolo utilizado para expresar conexión entre páginas diferentes.
	Símbolo utilizado para expresar un módulo de un problema. En realidad expresa que para continuar con el flujo normal del diagrama debemos primero resolver el sub problema que enuncia en su interior



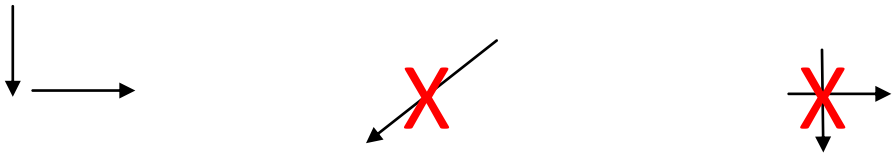
Nota: *Se debe observar que estas fases se presentan en la mayoría de los diagramas de flujo, aunque a veces en orden diferente o repitiendo alguna(s) de ellas. También es frecuente tener que realizar toma de decisiones y repetir una serie de pasos un número determinado o no de veces.*

Conjunto de reglas que permiten la construcción de diagramas de flujo.

1. Todo diagrama de flujo debe tener un inicio y un fin.

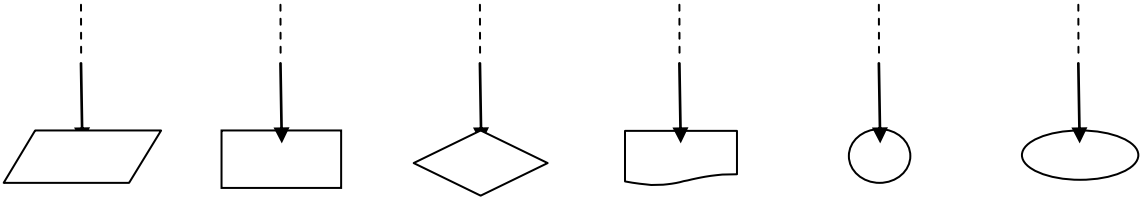


2. Las líneas utilizadas para indicar la dirección del flujo del diagrama deben ser rectas, verticales y horizontales.



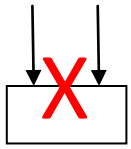
No deben ser inclinadas Tampoco debemos cruzarlas

3. Todas las líneas utilizadas para indicar la dirección del flujo del diagrama deben estar conectadas. La conexión puede ser a un símbolo que exprese lectura, proceso, decisión, impresión, conexión o fin de diagrama.

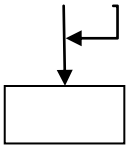


4. El diagrama de flujo debe ser construido de arriba hacia abajo y de izquierda a derecha
5. La notación utilizada en el diagrama de flujo debe ser independiente del lenguaje de programación. La solución presentada en el diagrama puede escribirse posteriormente y fácilmente en diferentes lenguajes de programación
6. Es conveniente cuando realizamos una tarea compleja poner comentarios que expresen o ayuden a entender lo que hicimos.
7. Si el diagrama de flujo requiriera más de una hoja para su construcción, debemos utilizar los conectores adecuados y enumerar las páginas convenientemente.

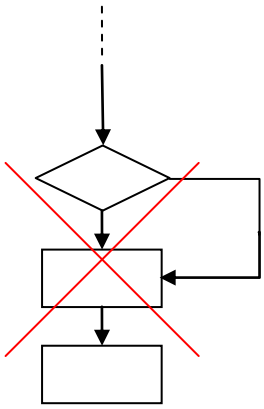
8. No puede llegar más de una línea a un símbolo.



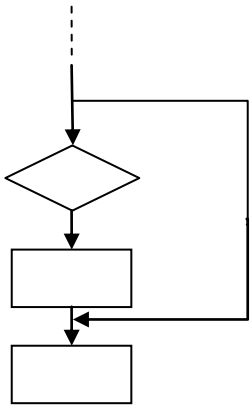
No válido



Válido



No válido



Válido

Construcción de Diagramas de Flujo con Base en Algoritmos Secuenciales

Algoritmo 01. Suponga que un individuo desea invertir su capital en un banco y desea saber cuánto dinero ganara después de un mes si el banco paga a razón de 2% mensual

Algoritmo 02. Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

Algoritmo 03. Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.

Algoritmo 04. Un alumno desea saber cuál será su calificación final en la materia de Algoritmos. Dicha calificación se compone de los siguientes porcentajes:

55% del promedio de sus tres calificaciones parciales.
30% de la calificación del examen final.
15% de la calificación de un trabajo final.

Algoritmo 05. Dada una cantidad en pesos, obtener la equivalencia en dólares, asumiendo que la unidad cambiaría es un dato desconocido.

Algoritmo 06. Calcular el nuevo salario de un obrero si obtuvo un incremento del 25% sobre su salario anterior.

Algoritmo 07. En un hospital existen tres áreas: Ginecología, Pediatría, traumatología. El presupuesto anual del hospital se reparte conforme a la siguiente tabla:

Área Porcentaje del presupuesto
Ginecología 40%
Traumatología 30%
Pediatría 30%

Obtener la cantidad de dinero que recibirá cada área, para cualquier monto presupuestal.

Algoritmo 08. El dueño de una tienda compra un artículo a un precio determinado. Obtener el precio en que lo debe vender para obtener una ganancia del 30%

Algoritmo 09. Todos los lunes, miércoles y viernes, una persona corre la misma ruta y cronometra los tiempos obtenidos. Determinar el tiempo promedio que la persona tarda en recorrer la ruta en una semana cualquiera.

Algoritmo 10. Tres personas deciden invertir su dinero para fundar una empresa. Cada una de ellas invierte una cantidad distinta. Obtener el porcentaje que cada quien invierte con respecto a la cantidad total invertida.

Algoritmo 11. Desarrolle un algoritmo tal que dados los 3 lados de un triángulo, pueda determinar su área. Esta la calculamos aplicando la siguiente fórmula:

Área = $\sqrt{S * (S - L1) * (S - L2) * (S - L3)}$

S = (L1 + L2 + L3) / 2

Dónde: Elevar n número a 0.5 es equivalente a la raíz cuadrada.

Algoritmo 12. Calcule e imprima el número de segundos que hay en un determinado número de días.

Algoritmo 13. Dado como datos el radio y la altura de un cilindro, calcule e imprima el área y su volumen.

Dónde:

Volumen = π * radio^2 * altura
Área = 2 * π * radio * altura

Algoritmo 14. Teniendo una distancia de 417 kms. De Ciudad Victoria a Miguel Alemán. Calcule el tiempo de viaje en automóvil (Horas y minutos), suponiendo una misma velocidad desde que sale, hasta el arribo.

Algoritmo 15. Una gasolinera despacha gasolina y la bomba surtidora registra la compra en galones, pero el precio de la gasolina está fijado en 8.20 el litro. Construya un algoritmo que calcule y escriba cuanto hay que cobrarle al cliente si este consume "*n*" galones.

Dónde:

1 Galón = 3.785 lts.

Algoritmo 16. Construya un algoritmo que calcule la distancia entre dos puntos, dado como datos las coordenadas de los puntos P1 y P2.

Dónde:

X1, Y1 representan las coordenadas del punto P1 en el eje de las X, Y
X2, Y2 representan las coordenadas del punto P2 en el eje de las X, Y.

Consideraciones:

Para calcular la distancia "D" entre dos puntos dados P1 y P2 aplicamos la siguiente fórmula:

$$D = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

Estructuras Algorítmicas Selectivas

Introducción

Las estructuras lógicas selectivas se encuentran en la solución algorítmica de casi todo tipo de problemas. Las utilizamos cuando en el desarrollo de la solución de un problema debemos tomar una decisión, para establecer un proceso o señalar un camino alternativo a seguir.

Esta toma de decisión se basa en la evaluación de una o más condiciones que nos señalarán como alternativa o consecuencia, la rama a seguir.

Hay situaciones en que la toma de decisiones se realiza en cascada. Es decir se toma una decisión, se marca la rama correspondiente a seguir, se vuelve a tomar otra decisión y así sucesivamente. Por lo que para alcanzar la solución de un problema o sub problema debemos aplicar prácticamente un árbol de decisión.

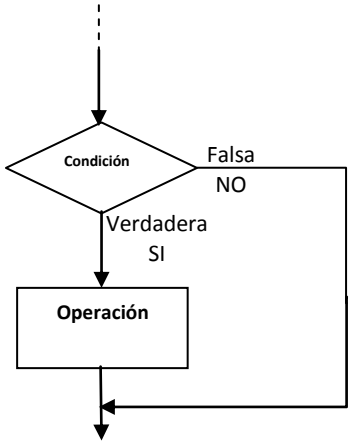
Las estructuras algorítmicas selectivas que se utilizan para la toma de decisiones lógicas las podemos clasificar de la siguiente forma:

1	SI ENTONCES	Estructura selectiva simple
2	SI ENTONCES / SINO	Estructura selectiva doble
3	SI MULTIPLE	Estructura selectiva múltiple

Estructura Selectiva Simple si entonces

La estructura selectiva *si entonces* permite que el flujo del algoritmo o diagrama siga por un camino específico si se cumple una condición o conjunto de condiciones. Si al evaluar la condición o condiciones el resultado es verdadero, entonces se ejecuta la operación u operaciones en su parte verdadera, luego se continúa con su secuencia normal.

```
Si condición entonces
        Hacer operación
Fin si
```



Donde:

Condición expresa la condición o conjunto de condiciones a evaluar.

Operación expresa la operación o conjunto de operaciones que se van a realizar si la condición resulta verdadera.

Ejercicios con Algoritmos de Estructura Selectiva Simple

Algoritmo 01. Dado como dato la calificación de un alumno en un examen, escriba "Aprobado" en caso de que esa calificación sea mayor a 8. Hacer también su diagrama de flujo y prueba de escritorio.

```
VARIABLES
    calif es real
INICIO
    Leer calif
    Si calif > 8 entonces
        Escribir "Aprobado"
    Fin si
FIN
```

Algoritmo 02. Desarrolle un algoritmo que determine si un número es par y que escriba dicho número junto con el letrero "***n*** es un número par"

Algoritmo 03. Dado como dato el sueldo de un trabajador, aplíquele un aumento del 15% si su sueldo es inferior a $1,000.00. Escriba en este caso el nuevo sueldo del trabajador. Haga el diagrama correspondiente y su prueba de escritorio.

Algoritmo 04. Hacer el algoritmo para escribir un programa que indique si un número ingresado por el teclado es positivo.

Algoritmo 05. Escriba un algoritmo que con base en tres valores enteros (val1, val2, val3), determine cuál de ellos es el mayor.

Algoritmo 06. Para que un alumno de la Politécnica pague $200 de inscripción necesita sacar un promedio de 9 o más. Con base en sus calificaciones, determine si alcanza este promedio y de ser así escriba "El alumno tiene beca"

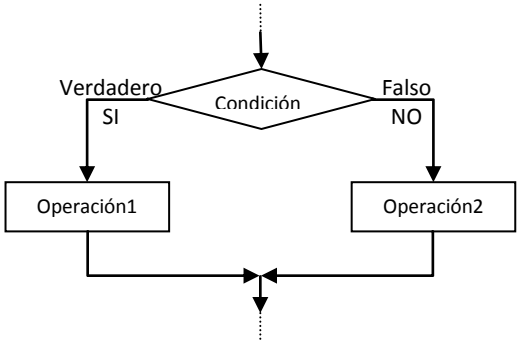
Algoritmo 07. El pasaje de Reynosa a Cd. Victoria cuesta $375.00 pesos, pero la compañía de autobuses hace descuentos de 60% tercer edad, 50% estudiantes, 35% menores de edad y 0% clientes regulares. Escriba un algoritmo que aplique al precio del boleto el descuento correspondiente según el tipo de persona que va a viajar.

Algoritmo 08. La compañía Marinela lanza la promoción de 2 x 1 y medio en todos sus productos. Desarrolle un algoritmo para el cobro de estos productos, por ejemplo, si el cliente lleva 5, cobrar 4 dentro de la promoción y uno con precio normal. Escriba el total a pagar.

Algoritmo 09. Tomando como base el algoritmo 08, controle el error que se haría en el cálculo del total a pagar si la cantidad entrante fuera un número negativo.

Estructuras Selectiva Doble si entonces / sino

La estructura selectiva si entonces / sino permite que el flujo del diagrama se bifurque por dos ramas diferentes en el punto de la toma de decisión(es). Si al evaluar la condición (o condiciones) el resultado es verdadero, entonces se sigue por un camino específico y se ejecuta(n) cierta(s) operación(es). Por otra parte, si el resultado es falso entonces se sigue por otro camino y se ejecuta(n) otra(s) operación(es). En ambos casos, luego de ejecutarse la(s) operación(es) indicada(s), se continúa con la secuencia normal del diagrama.

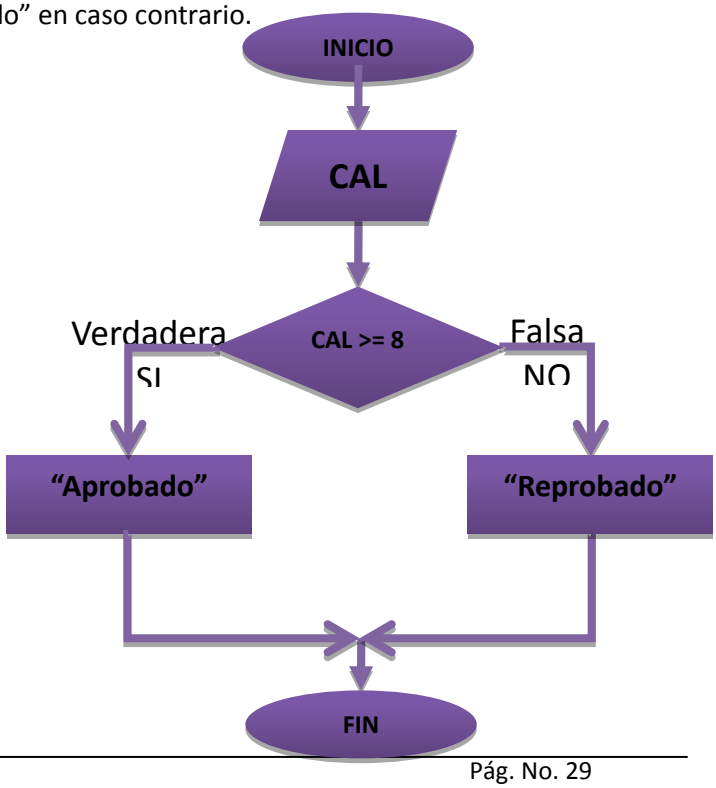


El diagrama de flujo en lenguaje algorítmico se representa de esta forma.

Si *condición* entonces
    Hacer operación 1
Sino
    Hacer operación 2
Fin si

Algoritmo 01. Dado como dato la calificación de un alumno en un examen, escriba “aprobado” si su calificación es mayor o igual que 8 y “reprobado” en caso contrario.

VARIABLES
    calif es real
INICIO
    Leer calif
    Si calif >= 8 Entonces
        Escribir “Aprobado”
    Sino
        Escribir “Reprobado”
    Fin si
FIN



Ejercicios con algoritmos de estructura selectiva doble.

Algoritmo 01. Construya un algoritmo dado un número entero positivo, determine y escriba si este número es par o impar.

Algoritmo 02. Con base en la edad proporcionada, determine y escriba si la persona es mayor o menor de edad.

Algoritmo 03. Dado un número entero positivo, verifique y escriba si se encuentra en el rango de 0 a 20 ó es mayor que 20.

Algoritmo 04. Construya un algoritmo que determine y escriba dado un número entero positivo, si este es menor, mayor o igual que cero.

Algoritmo 05. Dados 3 números enteros positivo, determine y escriba cual es el mayor.

Algoritmo 06. Dado como dato el sueldo de un trabajador, aplique un aumento del 15% si su sueldo es inferior a $1000.00 y 12% en caso contrario. Escriba el nuevo sueldo.

Algoritmo 07. Hacer un algoritmo que calcule el pago que hacen un grupo de personas para ver una película teniendo en cuenta que si el grupo es menor de 8 personas el pago es de 45 pesos por persona y para grupos de 8 personas o más el pago es 30 pesos por persona.

Algoritmo 08. Determine el elemento mayor de 3 números enteros.

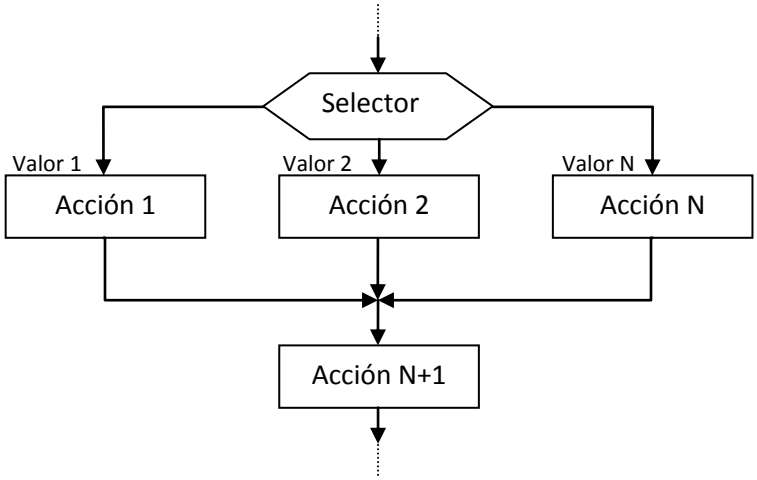
Algoritmo 09. En una tienda se efectúa un descuento a los clientes dependiendo del monto de la compra. El descuento se lleva a cabo con base en el siguiente criterio:

1. Si el monto es menor que $500.00 no hay descuento
2. Si el monto está comprendido entre $501 y $1,000, hay un 5% de descuento
3. Si el monto está comprendido entre $1,001 y $7,000, hay un 10% de descuento
4. Si el monto está comprendido entre $7,001 y $15,000, hay un 15% de descuento
5. Si el monto es mayor a $15,000, recibe un 20% de descuento

Construya un algoritmo tal que dado el monto de la compra de un cliente, determine y escriba lo que debe pagar.

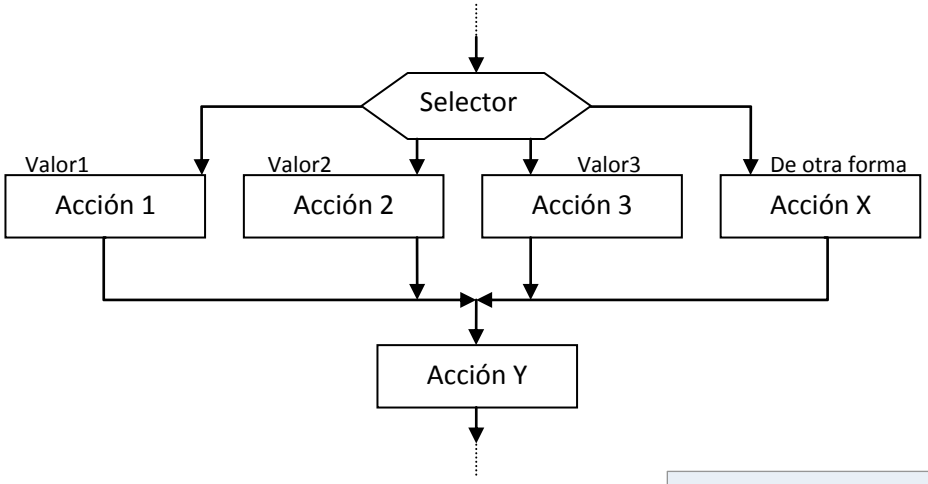
Estructuras Selectiva Múltiple Si-múltiple

La estructura selectiva si múltiple permite que el flujo del diagrama se bifurque por varias ramas en el punto de la toma de decisión(es), esto en función del valor que tome el selector. Así si el selector toma el valor 1 se ejecutará la acción 1, si toma el valor 2 se ejecutará la acción 2, si toma el valor N se realizará la acción N, y si toma un valor distinto de los valores comprendidos entre 1 y N, continuará con el flujo normal del diagrama realizándose la acción N+1.



.
.
Si ***selector*** Igual
    Valor 1: Hacer acción 1
    Valor 2: Hacer acción 2
    .
    .
    .
    Valor N: Hacer acción N
Fin si múltiple
Hacer acción N+1
.
.

La estructura selectiva múltiple es muy flexible, lo que permite aplicarse de diferentes formas.



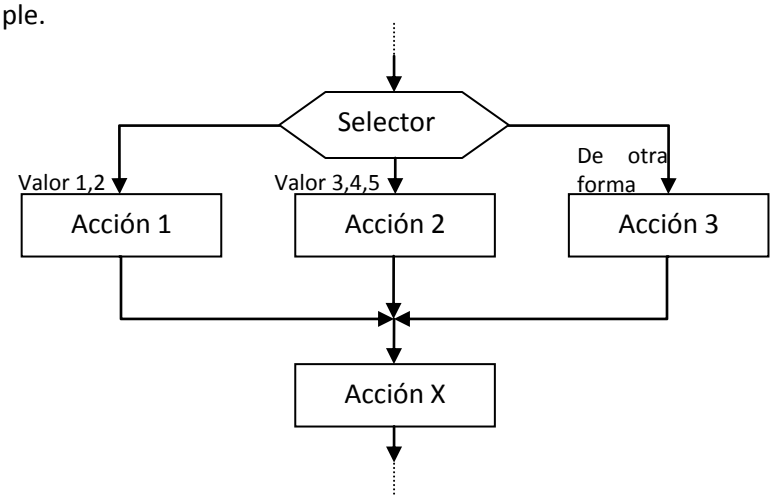
.
.
Si ***selector*** Igual
Valor 1: Hacer acción 1
Valor 2: Hacer acción 2
Valor 2: Hacer acción 3
De otra forma: Hacer acción X
Fin si múltiple
Hacer acción Y
.
.

Observe que si el selector toma el valor 1 se ejecuta la acción 1, si toma el valor 2 se realiza la acción 2, si toma el valor 3 se realiza la acción 3, y si toma cualquier otro valor se realiza la acción X. Luego cuando se continúa con el flujo normal del diagrama se realiza la acción Y

Observa que si el selector toma el valor 1 se ejecuta la acción 1, si toma el valor 2 se realiza la acción 2, si toma el valor 3 se realiza la acción 3, y si toma cualquier otro valor se realiza la acción X. Luego cuando se continúa con el flujo normal del diagrama se realiza la acción Y.

Observa la siguiente forma del uso del si-múltiple.

.
.
.
Si ***selector*** Igual
Valor 1,2: Hacer acción 1
Valor 3,4,5: Hacer acción 2
De otra forma: Hacer acción X
Fin si múltiple
Hacer acción X
.
.
.



Observa que si el selector toma el valor 1 ó 2 se realiza la acción 1, si el selector toma el valor 3, 4 ó 5 se realiza la acción 2, y si el selector toma cualquier otro valor se realiza la acción 3. Luego cuando se continúa con el flujo normal del diagrama se realiza la acción X.

Ejercicios con algoritmos de estructura selectiva *múltiple*.

Algoritmo 01. Desarrolle un algoritmo que muestre las cuatro operaciones aritméticas y permita elegir una opción y realice la operación con 2 datos enteros dados como entrada. Escriba el resultado y realice el diagrama de flujo y prueba de escritorio.

Algoritmo 02. Dados como datos la categoría y el sueldo de un trabajador, calcule el aumento correspondiente teniendo en cuenta la siguiente tabla. Imprimir la categoría del trabajador y el nuevo sueldo.

Incrementos	
Categoría	Aumento
1	15%
2	10%
3	8%
4	7%

Algoritmo 03. Desarrolle un algoritmo que muestre las opciones de cálculo de área de un círculo, rectángulo y circunferencia. Escriba cuál área fue calculada y su resultado.

Algoritmo 04. Construya un algoritmo que escriba la fecha del día en el formato "Hoy es Martes 15 de Febrero de 2011", dado el número de día de la semana, el día del mes y el año.

Algoritmo 05. Desarrolle un algoritmo que dada una calificación escriba los siguientes letreros

10	Felicidades
9	Muy Bien
8	Sigue Adelante
7	Puedes Mejorar
6 ó menor	Lo siento, No Aprobaste

Algoritmo 06. La COMAPA tiene su tarifa de cobro de servicio distribuida en 5 zonas, la cual obviamente tiene una variación en el precio del consumo por m^3 y se desglosa de la siguiente manera:

Zona No.	**Ubicación**	**Precio x m^3**
1	Centro	0.28 ¢
2	Norte	0.30 ¢
3	Este	0.28 ¢
4	Sur	0.25 ¢
5	Oeste	0.25 ¢

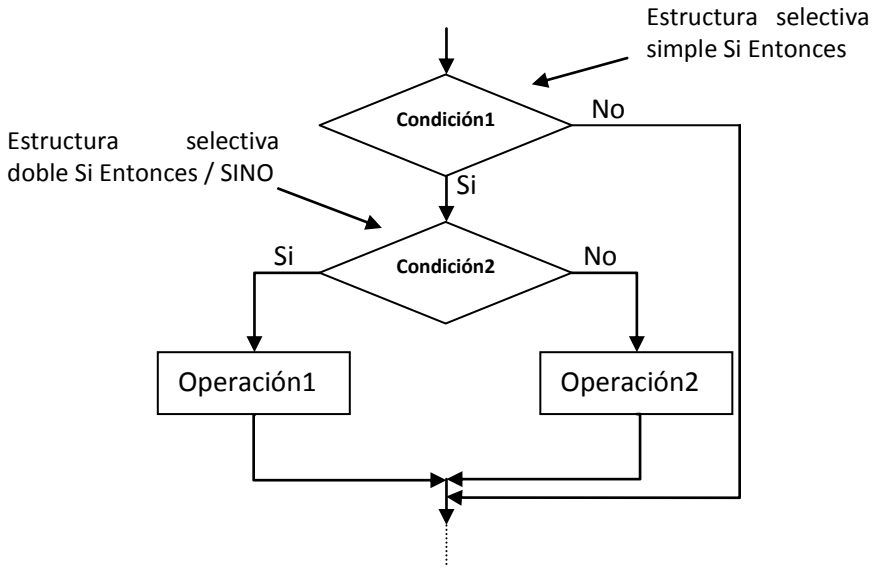
Desarrolle un algoritmo que calcule el total a pagar por consumo de un mes y escriba lo siguiente:

Zona No.:_____
Ubicación: _____
Consumo m^3:_____
Total a pagar: $_____

Estructuras Selectivas Anidadas (en cascada)

En numerosos casos en el desarrollo de la solución de problemas, encontramos que luego de tomar una decisión y marcar el camino correspondiente a seguir, es necesario tomar otra decisión. Luego de evaluar las condiciones, se señala nuevamente la rama correspondiente a seguir y nuevamente podemos tener que tomar otra decisión. El proceso puede repetirse numerosas veces.

En el siguiente ejemplo tenemos una estructura selectiva SI ENTONCES que contiene dentro de ella otra estructura selectiva SI ENTONCES / SINO.



.
.

```
Si condición1 Entonces
    Si condición2 Entonces
        Hacer operación1
    Sino
        Hacer operación2
    Fin Si
Fin Si
```

.
.

Ejercicios con algoritmos de estructura selectiva *en cascada*.

Algoritmo 01. Dados como dato 3 números enteros, identifique cuál es el mayor. Considere que los números pueden ser iguales. Desarrolle el algoritmo correspondiente.

Posibles casos:

Para que:	Tiene que ser:	y
A sea mayor	A > B	A > C
B sea mayor	B > A	B > C
C sea mayor	C > A	C > B
A y B o B y A sean mayores	A = B	> C
A y C o C y A sean mayores	A = C	> B
B y C o C y B sean mayores	B = C	> A
A, B y C sean iguales	A = B	= C

```
VARIABLES
    A,B,C son enteros
INICIO
    LEER A,B,C
    Si A > B Entonces
        Si A > C Entonces
            ESCRIBIR "A es el mayor"
        Sino
            Si A = C Entonces
                ESCRIBIR "A y C son los mayores"
            Sino
                ESCRIBIR "C es el mayor"
            Fin Si
        Fin Si
    Sino
        Si A = B Entonces
            Si A > C Entonces
                ESCRIBIR "A y B son los mayores"
            Sino
                Si A = C Entonces
                    ESCRIBIR "A,B y C son iguales"
                Sino
                    ESCRIBIR "C es el mayor"
                Fin Si
            Fin Si
        Sino
            Si B > C Entonces
                ESCRIBIR "B es mayor"
            Sino
                Si B = C Entonces
                    ESCRIBIR "B y C son mayores"
                Sino
                    ESCRIBIR "C es el mayor"
                Fin Si
            Fin Si
        Fin Si
    Fin Si
FIN
```

Algoritmo 02. Dados los datos A, B, C que representan números enteros diferentes, construya un algoritmo para escribir estos números en forma descendente.

Algoritmo 03. Construya un algoritmo de flujo tal que dado como dato una temperatura en grados Fahrenheit, determine el deporte que es apropiado practicar a esa temperatura, teniendo en cuenta la siguiente tabla:

DEPORTE	TEMPERATURA
Natación	>30
Tenis	>20 y <=30
Golf	>0 y <=20
Esquí	<=0

Estructuras Algorítmicas Repetitivas

Es muy común encontrar en la práctica algoritmos cuyas operaciones se deben ejecutar un número repetido de veces. Si bien las instrucciones son las mismas, los datos sobre los que se opera varían. El conjunto de instrucciones que se ejecuta repetidamente se llama ciclo.

Todo ciclo debe terminar de ejecutarse luego de un número finito de veces, por lo que es necesario en cada iteración del mismo, evaluar las condiciones necesarias para decidir si se debe seguir ejecutando o si debe detenerse. En todo ciclo, siempre debe existir una condición de parada o fin de ciclo.

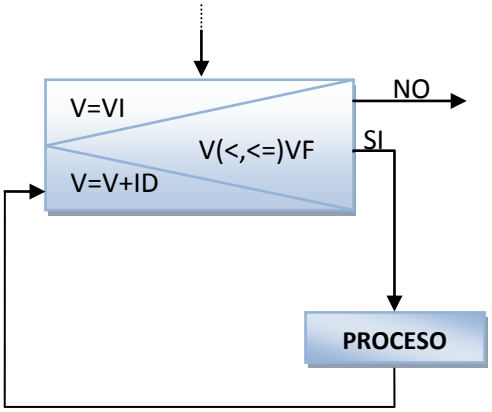
En algunos algoritmos podemos establecer a priori que el ciclo se repetirá un número definido de veces. Es decir, el número de repeticiones no dependerá de las proposiciones dentro del ciclo. Llamaremos ***PARA*** a la estructura algorítmica repetitiva que se ejecuta un número definido de veces.

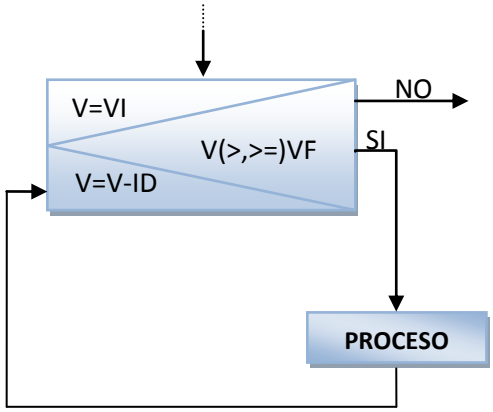
Por otra parte, en algunos algoritmos no podemos establecer a priori el número de veces que ha de ejecutarse el ciclo, sino que este número dependerá de las proposiciones dentro del mismo. Llamaremos ***MIENTRAS*** a la estructura algorítmica repetitiva que se ejecuta mientras la condición evaluada resulta verdadera

Estructura repetitiva o ciclo: *PARA* (FOR)

El ciclo ***PARA*** conocido comúnmente como **FOR**, es la estructura algorítmica adecuada para utilizar en un ciclo que se ejecutará un número definido de veces. Este tipo de estructura está presente en todos los lenguajes de programación, ya sean estructurados u orientados a objetos. Por ejemplo cuando necesitamos calcular la nómina total de la empresa, tenemos que sumar los sueldos de los N empleados de la misma. Cuando necesitamos obtener el promedio de calificaciones de un curso, debemos sumar las N calificaciones de los alumnos y dividir esa suma entre N. es decir, sabemos de antemano cuántas veces tenemos que repetir una determinada operación, acción o tarea. El número de repeticiones no depende de las proposiciones dentro del ciclo. El número de veces se obtiene del planteamiento del problema o de una lectura que indica que el número de iteraciones se debe realizar para N ocurrencias.

Diagrama de flujo de la estructura algorítmica ***PARA*** (**FOR**)





.
.
.

PARA V *desde* VI *hasta* VF HACER
    .
    .
    .
    {proceso}
    .
    .
    .
    V = V + ID
FIN PARA

Dónde:
- V Es la variable de control del ciclo
- VI Es el valor inicial
- VF Es el valor inicial
- ID Es el incremento o decremento, según sea la estructura repetir ascendente o descendente.

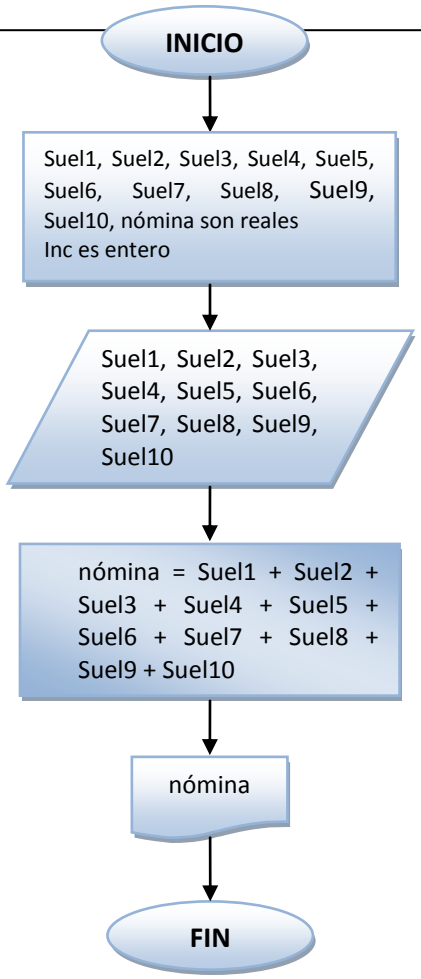
V (contador del ciclo, generalmente representado por las letras I, J, K, V) toma un valor inicial y se compara con VF (Valor Final). El ciclo se ejecuta mientras V es menor, menor igual, mayor o mayor igual al valor de V. El valor de V se incrementa o decrementa en cada iteración. Cuando V supera el valor de V. entonces el ciclo se detiene

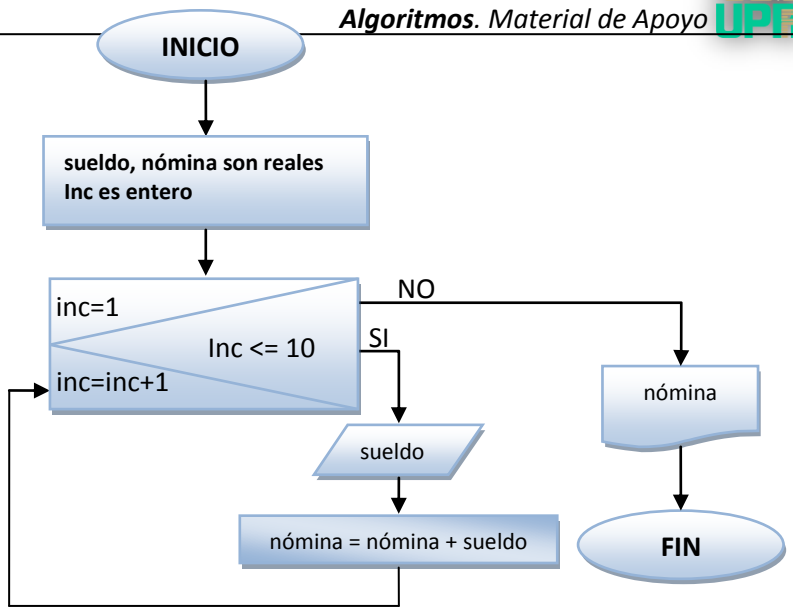
Algoritmos que calcula y escribe la nómina de 10 empleados.

```
Algoritmo si uso de ciclos
VARIABLES
    Suel1, Suel2, Suel3, Suel4, Suel5,
    Suel6, Suel7, Suel8, Suel9, Suel10,
    nómina son reales
INICIO
    LEER Suel1, Suel2, Suel3, Suel4,
    Suel5, Suel6, Suel7, Suel8, Suel9,
    Suel10

    nómina = Suel1 + Suel2 + Suel3 +
    Suel4 + Suel5 + Suel6 + Suel7 +
    Suel8 + Suel9 + Suel10
    ESCRIBIR nómina
FIN
```

```
Algoritmo usando ciclo PARA
VARIABLES
    sueldo, nómina son reales
    Inc es entero
INICIO
    PARA inc=1 hasta 10 hacer
        LEER sueldo
        nómina = nómina + sueldo
    FIN PARA
    ESCRIBIR nómina
FIN
```





Inc: Es una variable de tipo entero que representa la variable de control del ciclo. Contabiliza el número de veces que ha de repetirse una determinada acción. El contador toma un valor inicial (generalmente 0 ó 1) y se incrementa en la mayoría de los casos en una unidad en cada vuelta del ciclo

nómina: Es una variable de tipo real que representa un acumulador. Este se utiliza cuando debemos obtener el total acumulado de un conjunto de cantidades. Generalmente se inicializa en cero.

Es importante saber cómo trabajan los contadores y los acumuladores ya que en el uso de ciclos su utilización será casi obligada.

Cómo funcionan los contadores?

Un contador es una variable de tipo entero que se inicializa por lo general en 1 ó 0 dependiendo del problema y por default se incrementa en 1 cada vez que se ejecuta la sentencia donde se encuentra ubicado. La sintaxis es:

Var_cont = Var_cont + 1

Por ejemplo: suponiendo la variable cont inicializada en 0

cont = 0 /*aquí se inicializa la variable con valor igual a 0*/

cont = cont + 1 /*En esta parte la variable **cont =** a su valor (**0**) **+ 1**, por lo que **cont** ahora vale 1*/

Si repetimos este último paso, **cont** ahora valdrá 2 debido a que su valor actual es 1. Cómo es que sucede esto?. Es simple, la variable misma se involucra en la operación y de esta forma su valor forma parte de la fórmula, en la cual incluye su valor actual a quien se le sumará 1 y se lo asignará ella misma incrementándose.

Esto es en el caso simple de un contador en 1. En los casos de incrementos en 2 ó más, se hace una pequeña modificación en el número que incrementa de la siguiente manera:

Cont = cont + 2

Esto hace que ahora los incrementos sean de 2 cada vez que se ejecute.

Y cómo funcionan los acumuladores?

La diferencia de los acumuladores con los contadores, radica en que la variable que marca el incremento puede ser cualquier valor, pero la estructura es la misma, por ejemplo: La suma de calificaciones en un acumulador llamado **suma** inicializado en **0**.

Suma = 0

Suma = suma + calif

Si observa en esta operación, la variable al igual que en los contadores, se involucra en la fórmula para poder acumular así su ***valor actual + el nuevo valor a sumarse***. Esta es la diferencia entre el uso de contadores y acumuladores.

En resumen, un contador cada vez que se ejecuta suma un valor constante a su valor actual y un acumulador, cada vez que se ejecuta, suma el valor que tenga la variable que esta después del signo + y lo acumula en ella.

Algoritmo 01. Calcule la suma de los números del 1 al 10 y escríbala. Hacer la prueba de escritorio y su diagrama de flujo.
Algoritmo 02. Desarrolle un algoritmo que pida un número del 1 al 10 y escriba su tabla de multiplicar.

Algoritmo 03. Desarrolle un algoritmo que escriba las 10 tablas de multiplicar.

Algoritmo 04. Crear un algoritmo que escriba los números del 1 al 10

Algoritmo 05. Desarrolle un algoritmo que lea 3 calificaciones y calcule el promedio.

Algoritmo 06. Desarrolle un algoritmo que lea 3 calificaciones y calcule el promedio de N alumnos y escriba al final el promedio general.

Algoritmo 07. Con base en la edad, determine de 10 personas, cuántas son mayores de edad y cuantas son menores y escriba los resultados

Algoritmo 08. Desarrolle un algoritmo que lea 10 números enteros y escriba cuantos fueron negativos, cuantos positivos y cuantos cero.

Algoritmo 09. Crear un algoritmo que eleve un número a la N potencia

Algoritmo 10. Desarrolle un algoritmo que de 10 números nos diga cuantos fueron pares y cuantos impares.

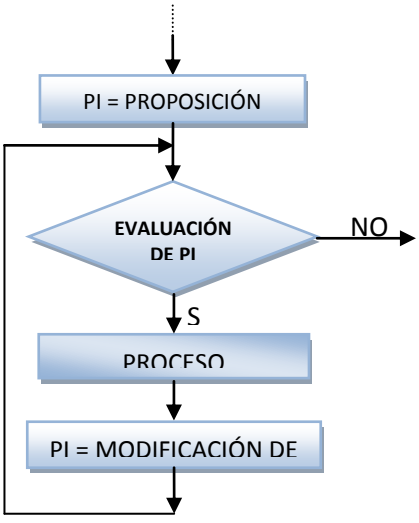
Algoritmo 11. Desarrolle un algoritmo que capture la venta de N productos, que con base en el precio, calcule y escriba el total a pagar.

La Estructura Repetitiva Mientras (While)

La estructura repetitiva Mientras conocida como while, es la estructura adecuada para utilizar en un ciclo cuando no sabemos el número de veces que éste se ha de repetir. Dicho número depende de las proposiciones dentro del ciclo. Ejemplos en la vida cotidiana encontramos muchos. Por ejemplo, supongamos que tenemos que obtener el total de una serie de gastos, pero no sabemos exactamente cuántos son; o cuando tenemos que sacar el promedio de calificaciones de un examen, pero no sabemos precisamente cuántos alumnos lo aplicaron. Tenemos que sumar las calificaciones e ir contando el número de alumnos, esto con el fin de poder obtener posteriormente el promedio. El ciclo se repite mientras tengamos calificaciones de alumnos.

En la estructura mientras se distinguen dos partes:

- Ciclo: conjunto de instrucciones que se ejecutarán repetidamente.
- Condición de terminación: la evaluación de esta condición permite decidir cuándo finalizará la ejecución del ciclo. La condición se evalúa al inicio del mismo.



El diagrama de flujo de la estructura algorítmica ***mientras*** es el siguiente:

Dónde:

PI: La proposición inicial, debe tener un valor verdadero inicialmente. Si el valor de PI es falso, entonces el ciclo no se ejecuta.

Debe existir también un enunciado dentro del ciclo que afecte la condición para evitar que el ciclo se ejecute indefinidamente.

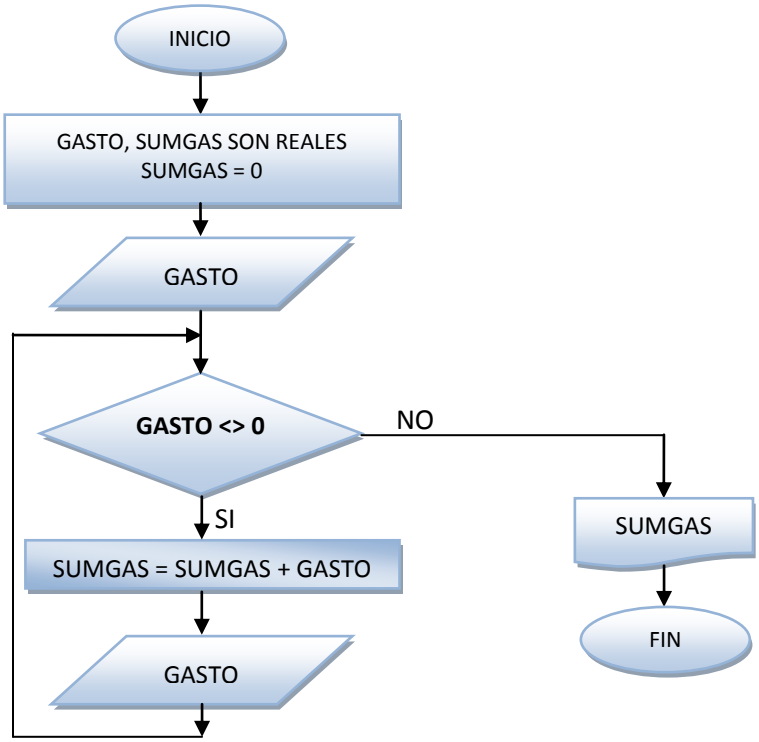
En lenguaje algorítmico la estructura ***mientras*** la expresamos de esta forma:

Algoritmo 01. Supongamos que debemos obtener la suma de los gastos que hicimos en nuestro último viaje, pero no sabemos exactamente cuántos fueron. Los datos son expresados en forma:

Datos: Gasto1, Gasto2,…, 0

Dónde:

Gastoi es una variable de tipo real que representa el gasto número i.



SUMGAS: Es una variable de tipo real. Es un acumulador. Acumula los datos efectuados.

GASTO: es una variable de tipo real. Su valor en la primera lectura debe ser verdadero, es decir distinto de 0. Su valor se modifica en cada vuelta del ciclo. Cuando gasto tome el valor de 0, entonces el ciclo se detendrá.

Algoritmo 02. Genere un algoritmo que escriba los términos de la siguiente serie:

2, 5, 7, 10, 12, 15, 17,…, 60

Para lograr esto, utilice una bandera para aplicar diferentes incrementos

Algoritmo 03. Hacer un algoritmo que realice el cobro de n productos en una tienda. Escribir el total a pagar. Por cada producto que se cobre deberá pedir cantidad y precio y calcular total de los artículos y así con cada uno de los demás hasta terminar.

Ejercicios con ciclos PARA (FOR)

Algoritmo 04. Calcule la suma de los números del 1 al 10 y escríbala. Hacer la prueba de escritorio y su diagrama de flujo.

Algoritmo 05. Desarrolle un algoritmo que pida un número del 1 al 10 y escriba su tabla de multiplicar.

Algoritmo 06. Desarrolle un algoritmo que escriba las 10 tablas de multiplicar.

Algoritmo 07. Crear un algoritmo que escriba los números del 1 al 10

Algoritmo 08. Desarrolle un algoritmo que lea 3 calificaciones y calcule el promedio.

Algoritmo 09. Desarrolle un algoritmo que lea 3 calificaciones y calcule el promedio de N alumnos y escriba al final el promedio general.

Algoritmo 10. Con base en la edad, determine de 10 personas, cuántas son mayores de edad y cuantas son menores y escriba los resultados

Algoritmo 11. Desarrolle un algoritmo que lea 10 números enteros y escriba cuantos fueron negativos, cuantos positivos y cuantos cero.

Algoritmo 12. Crear un algoritmo que eleve un número a la N potencia

Algoritmo 13. Desarrolle un algoritmo que de 10 números nos diga cuantos fueron pares y cuantos impares.

Algoritmo 14. Desarrolle un algoritmo que capture la venta de N productos, que con base en el precio, calcule y escriba el total a pagar.

Ejercicios con ciclos MIENTRAS (WHILE)

Algoritmo 15. Genere un algoritmo que escriba los términos de la siguiente serie:

2, 5, 7, 10, 12, 15, 17,…, 60

Para lograr esto, utilice una bandera para aplicar diferentes incrementos

Algoritmo 16. Hacer un algoritmo que realice el cobro de n productos en una tienda. Escribir el total a pagar. Por cada producto que se cobre deberá pedir cantidad y precio y calcular total de los artículos y así con cada uno de los demás hasta terminar.

Algoritmo 17. Desarrolle un algoritmo para la conjetura de ULAM. Esto es lo siguiente:

- Comience con cualquier entero positivo
- Si es par, divídalo entre 2; si es impar, multiplíquelo por 3 y agréguele 1
- Obtenga enteros sucesivamente repitiendo el proceso

Al final, obtendrá el número 1, independientemente del entero inicial. Por ejemplo, cuando el entero inicial es 26, la secuencia será: 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Algoritmo 18. Escriba los primeros 10 números de la serie de Fibonacci.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

El siguiente número se calcula sumando los dos que están antes de él.
El 2 se calcula sumando los dos delante de él (1+1)
El 21 se calcula sumando los dos delante de él (8+13)
La regla es $x_n = {}_{xn-1} + x_{n-2}$

Utilice un contador para controlar el ciclo.

Algoritmo 19. Desarrolle un algoritmo que determine si un número entero positivo y mayor que 1 es primo o no. Un número primo es divisible entre sí mismo y la unidad, el 1 queda descartado como número primo (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,…).

Algoritmo 20. Escriba un algoritmo que Lea n cantidad de números enteros positivos y cuando sea un negativo termine. Deberá escribir estos números con la condición de que si se lee el mismo número en forma consecutiva, solo escriba uno.

Algoritmo 21. Desarrolle un algoritmo que lea 5 números enteros, determine cuál es el mayor y escriba su tabla de multiplicar.

Algoritmo 22. Escriba un algoritmo que lea un número, determine si es par o impar. Si es par que escriba todos los pares hasta el cero. Si es impar, que escriba todos los impares hasta el 1.

Estructuras de Datos: Arreglos

Con frecuencia podemos encontrar problemas cuya solución es muy difícil de implementar si utilizamos tipos simples de datos. Por otra parte, podemos encontrar una buena solución al problema utilizando tipos estructurados de datos.

Veamos los siguientes ejemplos:

Algoritmo ejemplo 01. Se tienen los sueldos de un grupo de 70 empleados de una empresa y necesitamos saber cuántos de estos empleados tienen un sueldo superior al promedio del grupo.

Algoritmos utilizando tipos de datos simples.

Como primera opción podemos definir una variable para cada sueldo, esto representaría un algoritmo larguísimo, ya que después de obtener el promedio necesitaríamos verificar cada uno de los 70 sueldos contra el promedio obtenido y sería establecer una selectiva para cada uno de ellos.

Como segunda opción, podemos utilizar ciclos para minimizar la cantidad de variables.

```
VARIABLES
    Acum, prom, sueldo son reales
    Cont, i son enteras
INICIO
    Acum = 0
    Cont = 0
    PARA i = 1 HASTA 70 HACER
        LEER sueldo
        Acum = acum + sueldo
    FIN PARA
    Prom = acum / 70
    PARA i = 1 HASTA 70 HACER
        LEER sueldo
        SI sueldo > prom ENTONCES
            Cont = cont + 1
        FIN SI
    FIN PARA
    ESCRIBIR cont
FIN
```

En este segundo caso tendríamos que leer 2 veces el sueldo de cada uno de los trabajadores, lo cual no es muy práctico.

Las dos soluciones son muy representativas de los inconvenientes a los que debemos enfrentarnos al tratar de resolver el problema utilizando tipos simples de datos.

Puede observarse entonces, que ninguna de las dos soluciones resulta práctica y eficiente. Es necesario un nuevo tipo de datos que permita tratar estos problemas de una manera más adecuada. Los tipos de datos estructurados que ayudan a resolver problemas como éste son los arreglos.

Arreglos Unidimensionales

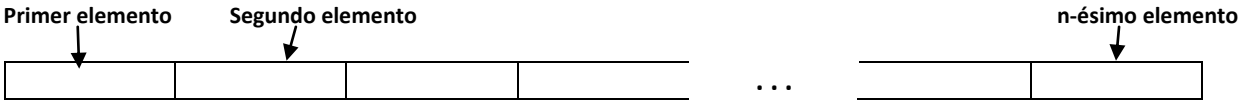
Un arreglo se define como una colección finita, homogénea y ordenada de elementos.

Finita. Todo arreglo tiene un límite, es decir se debe determinar cuál será el número máximo de elementos que podrán formar parte del arreglo.

Homogénea. Todos los elementos de un arreglo son del mismo tipo (todos enteros, todos reales, etc., pero nunca una combinación de distintos tipos).

Ordenada. Se puede determinar cuál es el primer elemento, el segundo, el tercero,… y el n-ésimo elemento.

Un arreglo puede representarse gráficamente como se muestra en la siguiente figura:



Arreglo

Un arreglo tiene la característica de que puede almacenar a N elementos del mismo tipo y además permite el acceso a cada uno de estos elementos. Así, se distinguen dos partes en los arreglos.

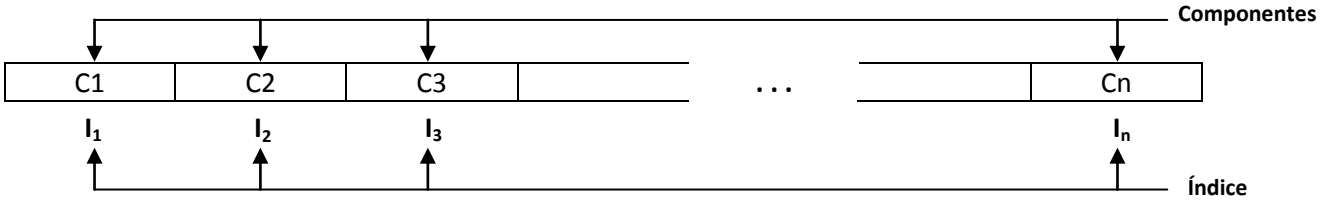
- Los componentes
- Los índices

Los componentes hacen referencia a los elementos que componen o forman el arreglo. Es decir, son los valores que se almacenan en cada una de sus casillas.

Los índices, por otra parte, son los que permiten accesar a los componentes del arreglo en forma individual. Para hacer referencia a un componente de un arreglo se necesita:

- El nombre del arreglo
- El índice del elemento

En la siguiente figura representamos un arreglo y se indican sus componentes y sus índices



Declaración de arreglos

La declaración de un arreglo se hace de la siguiente manera:

Var_arreglo = ARREGLO[*líminf* . . *límsup*] DE **TIPO**

Con los valores *líminf* y *límsup* se declara el número de elementos que tendrá el arreglo. El número total de elementos que tendrá el arreglo viene dado por *límsup*.

Con **TIPO** se declara el tipo de datos para todos los elementos del arreglo. El tipo de los elementos no tiene que ser necesariamente el mismo que el de los índices.

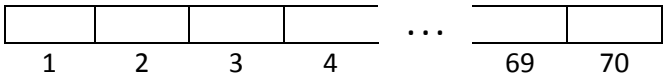
Observaciones:

a) El tipo de índice debe ser entero.
b) El tipo de los componentes puede ser cualquier tipo (entero, real, cadena de caracteres, registro, arreglo, etc.).
c) Se utilizan los corchetes "[]" para indicar el índice de un arreglo. Entre los [] se debe escribir un valor ordinal, puede ser una variable, una constante o una expresión tan compleja como se quiera, pero que dé como resultado un valor ordinal).

Por ejemplo: Sea myarray un arreglo de 70 elementos enteros. Su declaración queda como se muestra a continuación:

Myarray = ARREGLO[1..70] DE enteros

Myarray



Cada elemento del arreglo myarray será un número entero y se podrá accesar por medio de un índice que será un valor comprendido entre 1 y 70.

Así por ejemplo:

Myarray[1] hace referencia al elemento de la posición 1
Myarray[2] hace referencia al elemento de la posición 2
…
Myarray[70] hace referencia al elemento de la posición 70

Operaciones con arreglos

Las operaciones más comunes en los arreglos son las siguientes:

- Lectura/Escritura
- Asignación
- Actualización
  - Inserción
  - Eliminación
  - Modificación
- Ordenación
- Búsqueda

Como los arreglos son datos estructurados muchas de estas operaciones no pueden llevarse a cabo de manera global, sino que se debe trabajar sobre cada elemento.

Lectura.

El proceso de lectura de un arreglo consiste en leer y asignar un valor a cada uno de sus elementos. Supóngase que se desea leer todos los elementos del arreglo myarray en forma consecutiva. Podría hacerse de la siguiente manera:

```
LEER myarray[1]
LEER myarray[2]
…
LEER myarray[70]
```

De esta forma no resulta práctico, por lo tanto se usará un ciclo para leer todos los elementos del arreglo.

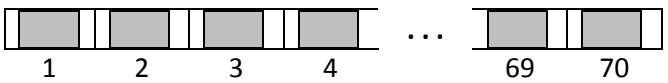
```
PARA i = 1 HASTA 70 HACER
    LEER myarray[i]
FIN PARA
```

Al variar el valor de ***i***, cada elemento leído se asigna al correspondiente componente del arreglo según la posición indicada por ***i***.

Al finalizar el ciclo de lectura se tendrá asignado un valor a cada uno de los componentes del arreglo myarray.

Puede suceder que no se necesite leer todos los componentes, sino solamente algunos de ellos. Supóngase por ejemplo que deben leerse los elementos con índices comprendidos entre el 1 y el 30. El ciclo necesario es el siguiente:

```
PARA i=1 HASTA 30 HACER
    LEER myarray[i]
FIN PARA
```



Escritura

El caso de la escritura es similar al de lectura. Se debe escribir el valor de cada uno de los componentes. Supóngase que desea escribir los primeros N componentes del arreglo myarray en forma consecutiva. Los pasos a seguir son los siguientes:

```
LEER n
PARA i = 1 HASTA n HACER
    ESCRIBIR myarray[i]
FIN PARA
```

Al variar el valor de ***i*** se escribe el elemento de myarray correspondiente a la posición indicada por ***i***.

Asignación

En general no es posible asignar directamente un valor a todo el arreglo, sino que se debe asignar el valor deseado a cada componente.

```
Myarray[1] = 120
Myarray[3] = myarray[1]/4
```

En los dos casos se asigna un valor a una determinada casilla del arreglo (en el primero a la señalada por el índice 1 y en el segundo a la indicada por el índice 3)

En el siguiente caso se asigna el 0 a todas las casillas del arreglo, con lo que éste queda de la siguiente manera:

```
PARA i = 1 HASTA 70 HACER
    Myarray = 0
FIN PARA
```

myarray

0	0	0	0	...	0	0
1	2	3	4		69	70

En algunos lenguajes, por otra parte, es posible asignar una variable tipo arreglo a otra exactamente del mismo tipo.

```
PARA i = 1 HASTA 70 HACER
    ARR1[i] = ARR[i]
FIN PARA
```

Arreglos Multidimensionales

El término dimensión representa la cantidad de índices utilizados para referenciar un elemento particular en un arreglo. Todos los arreglos vistos hasta el momento eran unidimensionales y se ha requerido solamente un índice para acceder a un elemento. Para averiguar cuántas dimensiones tiene un arreglo basta mirar su declaración. Si solamente hay un rango entre corchetes el arreglo es unidimensional. Dos rangos, es un arreglo bidimensional, y así sucesivamente. Los arreglos de más de una dimensión se denominan arreglos multidimensionales. Generalmente, el número máximo de dimensiones con el que se trabaja es de tres.

Arreglos Bidimensionales

Se declaran los arreglos bidimensionales especificando el número de renglones y el número de columnas, junto con el tipo de los componentes.

Var_arreglo = ARREGLO[*líminfR . . límsupR, líminfC . . límsupC*] DE **TIPO**

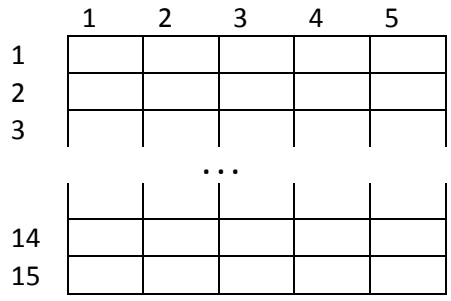
Con *líminfR . . límsupR* se declara cuántos renglones tendrá el arreglo y con *líminfC . . límsupC* se declara cuántas columnas tendrá el arreglo. Con TIPO se declara el tipo de datos de todos los componentes del arreglo.

El número total de elementos (NTE) está determinado por la expresión

NTE = límsupR * límsupC

Ejemplo de un arreglo bidimensional (MATRIZ)

Sea MATRIZ un arreglo bidimensional de números reales con índices enteros. Su representación queda de la siguiente manera:



MATRIZ = ARREGLO[1..15, 1..5]DE reales

- NTE = 15 * 5 = 75
- Cada elemento de MATRIZ será un número real. Para hacer referencia a cada uno de ellos se usarán dos índices y el nombre de la variable tipo arreglo. MATRIZ[i,j]

Así por ejemplo:

MATRIZ[2,4] hace referencia al elemento del renglón 2 y la columna 4.
MATRIZ[9,2] hace referencia al elemento del renglón 9 y la columna 2.
…
MATRIZ[15,5] hace referencia al elemento del renglón 15 y la columna 5.

Operaciones con arreglos bidimensionales

Las operaciones que pueden hacerse con los arreglos bidimensionales son las siguientes:

- Lectura/Escritura
- Asignación
- Actualización:
  - Inserción
  - Eliminación
  - Modificación
- Ordenación
- Búsqueda

Lectura

Cuando se introdujo la lectura en arreglos unidimensionales se dijo que se iban asignando valores a cada uno de los componentes. Lo mismo sucede con los arreglos bidimensionales. Sin embargo, como sus elementos deben referenciarse con dos índices, se deben utilizar dos ciclos para lograr la lectura de elementos consecutivos.

Supóngase que se desea leer todos los elementos del arreglo bidimensional MATRIZ. Los pasos a seguir son los siguientes:

```
VARIABLES
    MATRIZ=ARREGLO[1..15,1..5]DE reales
    i, j son enteros
INICIO
    PARA i=1 HASTA 15 HACER
        PARA i=1 HASTA 5 HACER
            LEER MATRIZ[i,j]
        FIN PARA
    FIN PARA
FIN
```

Al variar los índices de i y j se lee un elemento de MATRIZ, según la posición indicada por los índices i y j.

Para i= 1 y j=1, se lee el elemento del renglón 1 y columna 1
i= 1 y j=2, se lee el elemento del renglón 1 y columna 2
. . .
i= 15 y j=5, se lee el elemento del renglón 15 y columna 5

Escritura

La escritura de un arreglo bidimensional también se lleva a cabo elemento por elemento. Supóngase que se quiere escribir todos los componentes del arreglo MATRIZ. Los pasos a seguir son los que se muestran a continuación.

```
VARIABLES
    MATRIZ=ARREGLO[1..15,1..5]DE reales
    i, j son enteros
INICIO
    /*Proceso de lectura de un arreglo bidimensional*/
    PARA i=1 HASTA 15 HACER
        PARA i=1 HASTA 5 HACER
            LEER MATRIZ[i,j]
        FIN PARA
    FIN PARA

    /* Proceso de escritura de un arreglo bidimensional */
    PARA i=1 HASTA 15 HACER
        PARA i=1 HASTA 5 HACER
            ESCRIBIR MATRIZ[i,j]
        FIN PARA
    FIN PARA
FIN
```

Asignación

La asignación de valores a un arreglo bidimensional puede realizarse de dos maneras diferentes, según el número de componentes involucrados.

1. A todos los elementos del arreglo: en este caso se necesitarán dos ciclos para recorrer todo el arreglo.

```
VARIABLES
    MATRIZ=ARREGLO[1..15,1..5]DE reales
    i, j son enteros
INICIO
    PARA i=1 HASTA 15 HACER
        PARA i=1 HASTA 5 HACER
            MATRIZ[i,j] = 0
        FIN PARA
    FIN PARA
FIN
```

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
...					
13	0	0	0	0	0
14	0	0	0	0	0
15	0	0	0	0	0

2. A un elemento particular del arreglo: en este caso la asignación es directa. Por ejemplo para asignar el valor 12 al elemento del renglón 13 y columna 4, hacemos:

MATRIZ[13,4] = 12

El arreglo queda de la siguiente manera:

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
			...		
13	0	0	0	12	0
14	0	0	0	0	0
15	0	0	0	0	0

Se considera conveniente aclarar que las operaciones de lectura, escritura y asignación a todos los elementos de un arreglo bidimensional pueden hacerse por renglones o por columnas.

Bibliografías

Metodología de la programación
Algoritmo, diagramas de flujo y programas
3ª Edición
Autor: Osvaldo Cairó
Editorial: Alfaomega

Manual de Borland C++ 4.0
Autor: Chris H. Pappas / William H. Murray, III
Editorial: Osborne McGraw-Hill

Nota: Se toma material de las anteriores bibliografías y se hacen algunos cambios en edición, simbología y sintaxis para una más sencilla comprensión y uso de los conceptos y símbolos.