Estructuras Condicionales

Programación Lógica

UNIVERSIDAD SIGLO 21 | MIEMBRO DE LA RED ILUMNO

1. Estructuras condicionales

Las estructuras secuenciales ejecutan una sentencia a continuación de otra de acuerdo con el orden especificado en el algoritmo. Muchas veces, es deseable ejecutar una acción si se cumple una condición. Por ejemplo, si el divisor de una operación de división es positivo, entonces se debe calcular el cociente, de lo contrario, informar que el número divisor debe ser positivo. Las estructuras de control condicionales son utilizadas para ejecutar un conjunto de sentencias si se cumple una condición. Esto permite modificar el orden de ejecución de las sentencias según el resultado de una expresión que se evalúa. El tipo de dato del resultado de la condición debe ser de tipo lógico.

Las estructuras condicionales se clasifican en: simples, dobles y múltiples.

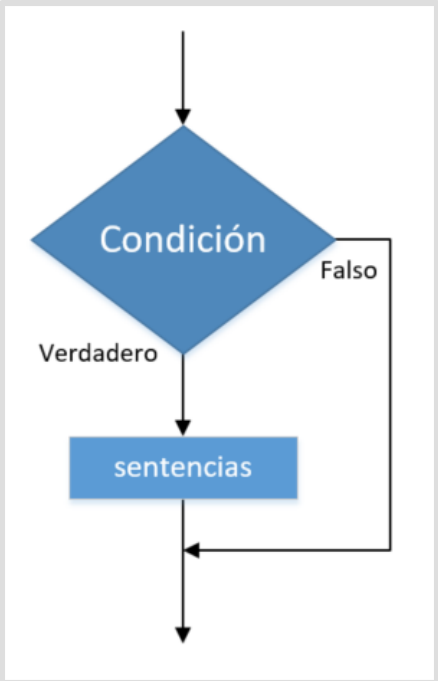
1.1. Condicional simple

Evalúa una expresión lógica (condición). Si esta es verdadera, ejecuta las sentencias especificadas a continuación de la palabra reservada *entonces*; si es falsa, no realiza ninguna acción y continúa con el flujo del algoritmo.

Pseudocódigo:

si *<condición>* ***entonces***
    sentencia 1
    sentencia 2
    .
    .
    sentencia N
fin-si

Figura 1: Diagrama de flujo condicional simple



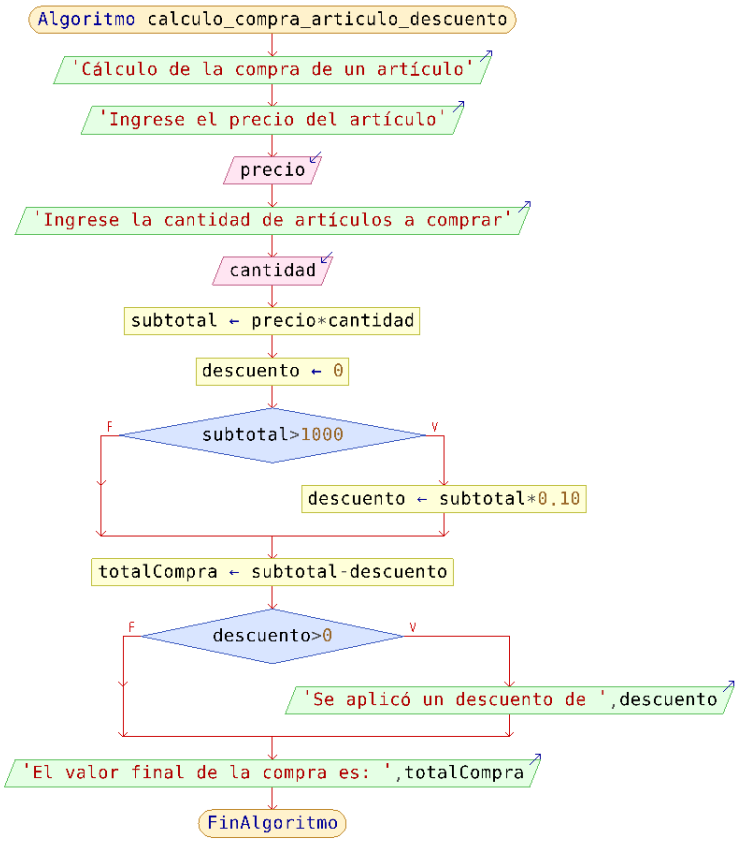
Fuente: elaboración propia.

Ejercicios de ejemplo

1) Implementar un algoritmo que calcule y muestre por pantalla el valor final de la compra de un artículo. El algoritmo debe permitir el ingreso del precio del artículo y la cantidad de artículos. Si el subtotal es superior a 1000, entonces se debe aplicar un descuento del 10 %.

```
algoritmo calculo_compra_articulo_descuento
var
    entero: cantidad
    real: precio, descuento, subtotal, totalCompra
inicio
    mostrar("Cálculo de la compra de un artículo")

    mostrar("Ingrese el precio del artículo")
    leer(precio)
    mostrar("Ingrese la cantidad de artículos a comprar")
    leer(cantidad)

    subtotal ← precio * cantidad
    descuento ← 0
```

```
    si subtotal > 1000 entonces
        descuento ← subtotal * 0.10
    fin-si

    totalCompra ← subtotal - descuento

    si descuento > 0 entonces
        mostrar("Se aplicó un descuento de ", descuento)
    fin-si

    mostrar("El valor final de la compra es: ", totalCompra)
fin
```



1.2. Condicional doble

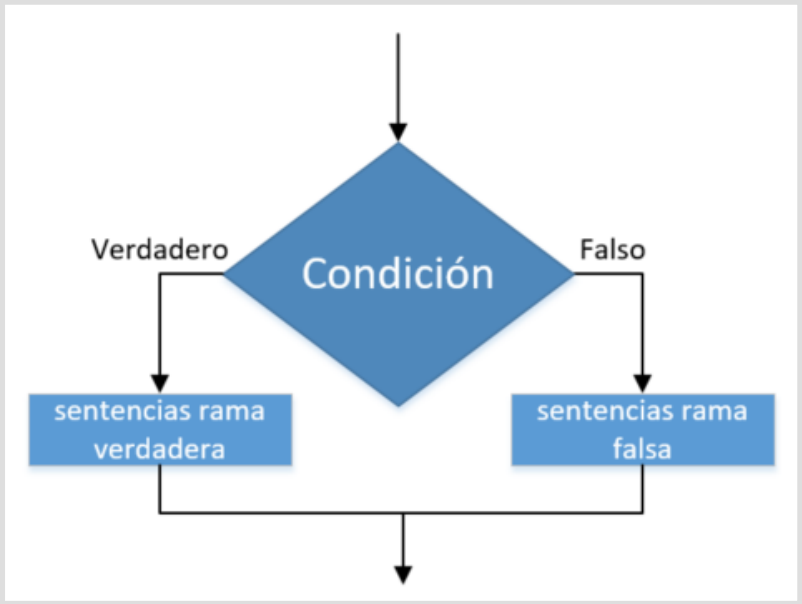
Evalúa una expresión lógica (condición). Si esta es verdadera, ejecuta las sentencias que se encuentran a continuación de la palabra reservada *entonces* (rama verdadera), pero si la condición resulta ser falsa, entonces

se ejecutan las sentencias especificadas a continuación de la palabra reservada *si-no* (rama falsa).

Pseudocódigo:

si *<condición>* ***entonces***
    sentencia 1
    sentencia 2
    .
    .
    sentencia N
si-no
    sentencia 1
    sentencia 2
    .
    .
    sentencia N
fin-si

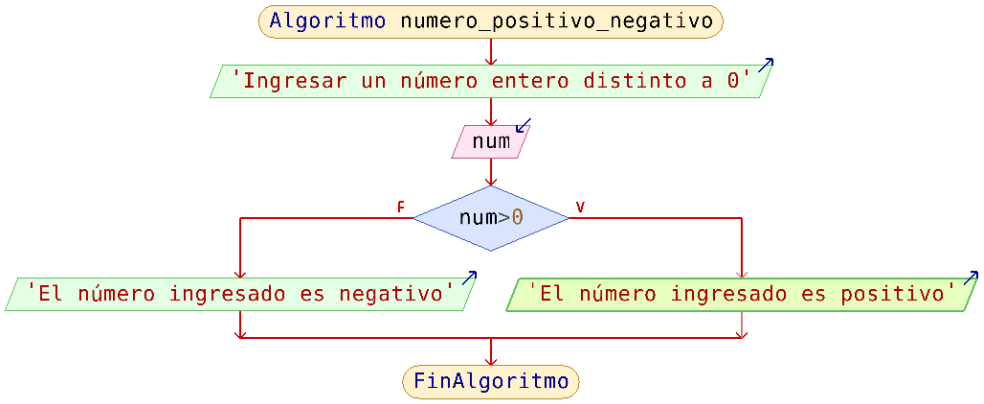
Figura 2: Diagrama de flujo condicional doble



Fuente: elaboración propia.

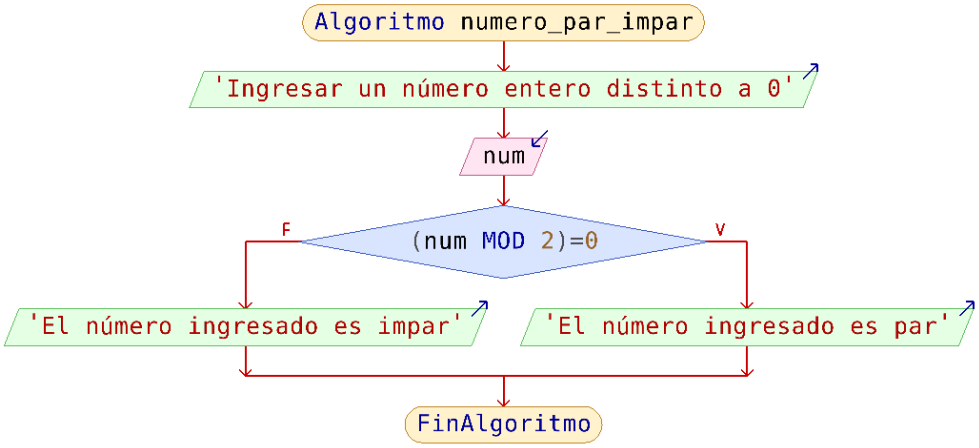
Ejercicios de ejemplo

1) Desarrollar un algoritmo que permita determinar si un número ingresado por teclado es positivo o negativo.



2) Desarrollar un algoritmo que permita determinar si un número ingresado por teclado es par o impar.

```
algoritmo numero_par_impar
var
    entero: num
inicio
    mostrar("Ingresar un número entero distinto a 0")
    leer(num)
    si (num mod 2) = 0 entonces
        mostrar("El número ingresado es par")
    si-no
        mostrar("El número ingresado es impar")
    fin-si
fin
```



1.3. Condicional múltiple

La estructura condicional múltiple es utilizada cuando se debe elegir entre más de dos alternativas de acuerdo con el valor de una variable. Evalúa una expresión (condición) que puede resultar en *N* valores distintos. Según el resultado de la expresión, se ejecutarán las sentencias definidas para el valor que se ha obtenido.

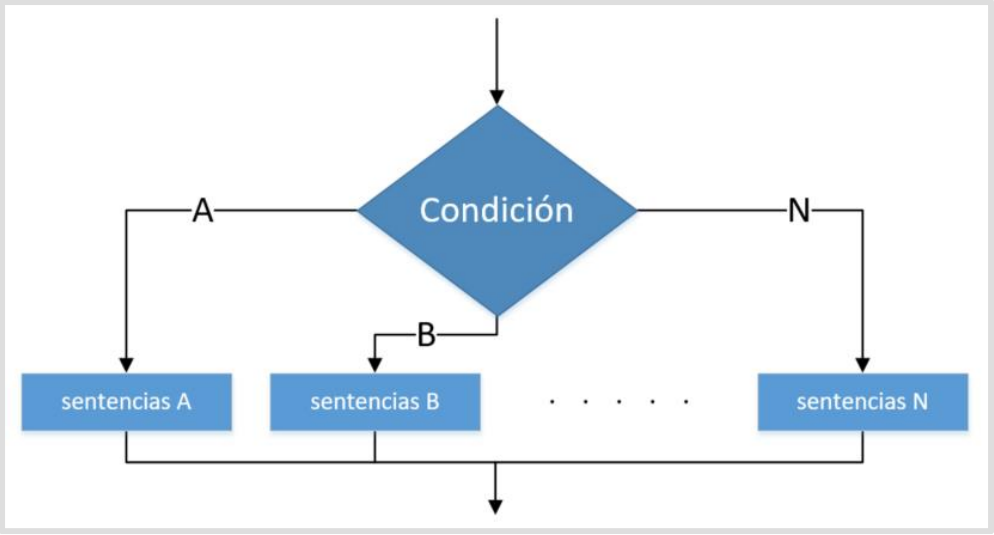
Pseudocódigo:

segun-sea *<expresión>* ***hacer***
    caso *A:* *sentencia A1*
        sentencia A2
        .
        .
        sentencia An
    caso *B:* *sentencia B1*
        sentencia B2
        .
        .
        sentencia Bn
    .
    .
    caso *N:* *sentencia N1*
        sentencia N2
        .
        .
        sentencia Nn
    si-no:
        sentencia X1

sentencia X2
.
.
sentencia Xn
fin-segun

Esta expresión requiere que la variable que se evalúa sea de tipo entero o carácter. La aplicación de la sentencia *si-no* es opcional y debe ser utilizada cuando necesitamos ejecutar un conjunto de acciones en el caso de que el resultado de la variable no sea igual a ninguno de los valores especificados en las sentencias *caso*.

Figura 3: Diagrama de flujo condicional múltiple



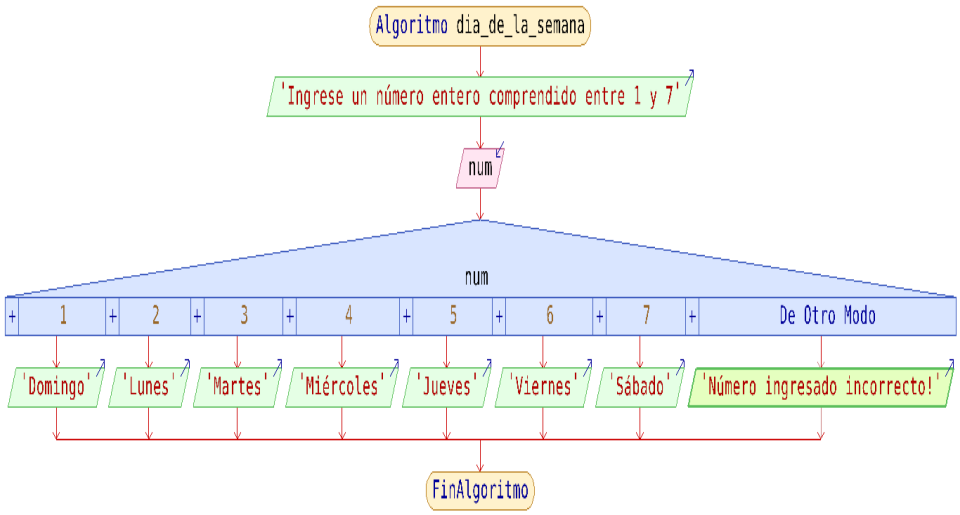
Fuente: elaboración propia.

Ejercicios de ejemplo

1) Desarrollar un algoritmo que permita ingresar un número entero comprendido entre 1 y 7, y que muestre por pantalla el día de la semana al que se corresponde.

```
algoritmo dia_de_la_semana
var
    entero: num
inicio
    mostrar("Ingrese un número entero comprendido entre 1 y 7")
    leer(num)
    segun-sea num hacer
```

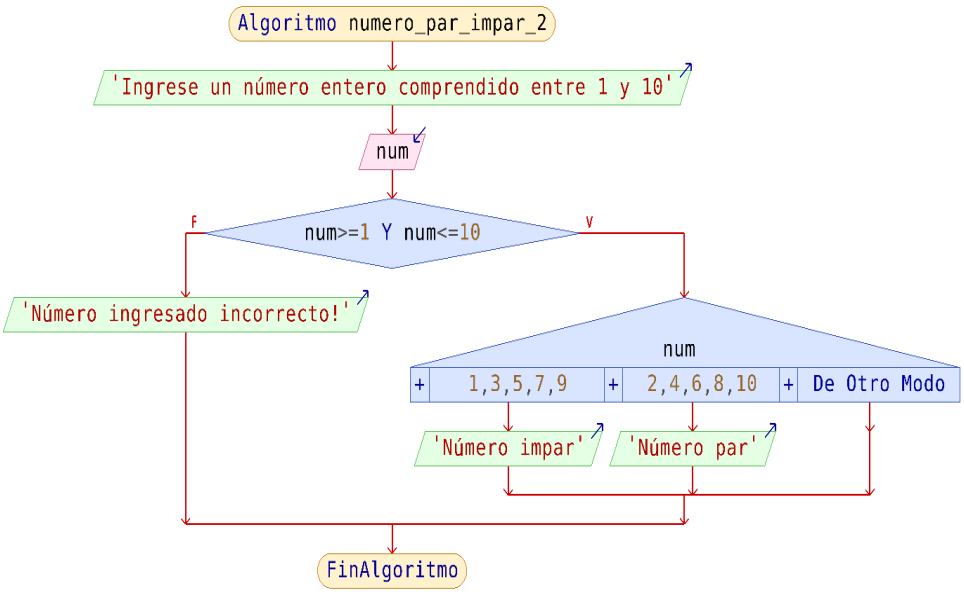
```
            caso 1:
                mostrar("Domingo")
            caso 2:
                mostrar("Lunes")
            caso 3:
                mostrar("Martes")
            caso 4:
                mostrar("Miércoles")
            caso 5:
                mostrar("Jueves")
            caso 6:
                mostrar("Viernes")
            caso 7:
                mostrar("Sábado")
            si-no:
                mostrar("Número ingresado incorrecto!")
    fin-segun
fin
```



2) Desarrollar un algoritmo que permita ingresar un número entero comprendido entre 1 y 10, y que muestre por pantalla si el número es par o impar.

```
algoritmo numero_par_impar_2
var
    entero: num
```

```
inicio
    mostrar("Ingrese un número entero comprendido entre 1 y 10")
    leer(num)

    si num >= 1 y num <= 10 entonces
        segun-sea num hacer
            caso 1, 3, 5, 7, 9:
                mostrar("Número impar")
            caso 2, 4, 6, 8, 10:
                mostrar("Número par")
        fin-segun
    si-no
        mostrar("Número ingresado incorrecto!")
    fin-si
fin
```

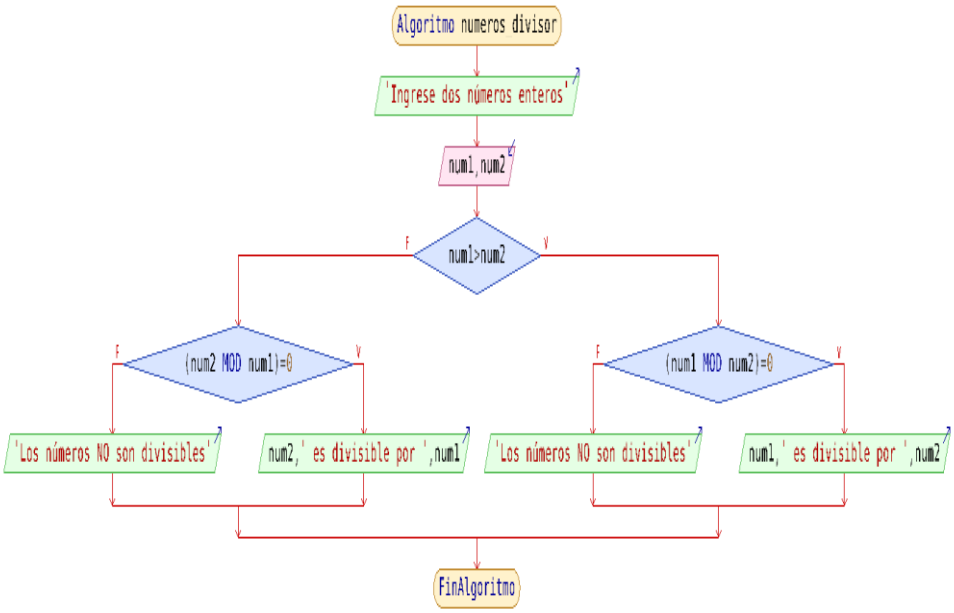


1.4. Ejercicios de aplicación

1) Implementar un algoritmo que permita ingresar dos números enteros por teclado y que determine si son divisibles.

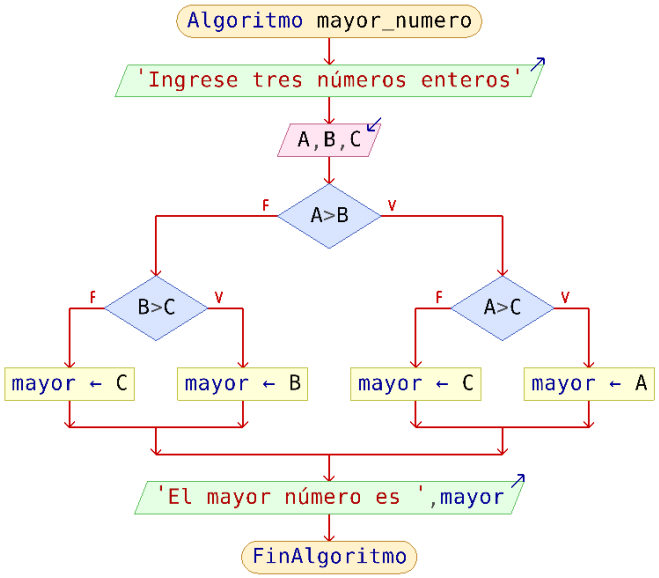
```
algoritmo numeros_divisor
var
    entero: num1, num2
```

```
inicio
    mostrar(“Ingrese dos números enteros”)
    leer(num1, num2)

    si num1 > num2 entonces
        si (num1 mod num2) = 0 entonces
            mostrar(num1, " es divisible por ", num2)
        si-no
            mostrar("Los números NO son divisibles")
        fin-si
    si-no
        si (num2 mod num1) = 0 entonces
            mostrar(num2, " es divisible por ", num1)
        si-no
            mostrar("Los números NO son divisibles")
        fin-si
    fin-si
fin
```



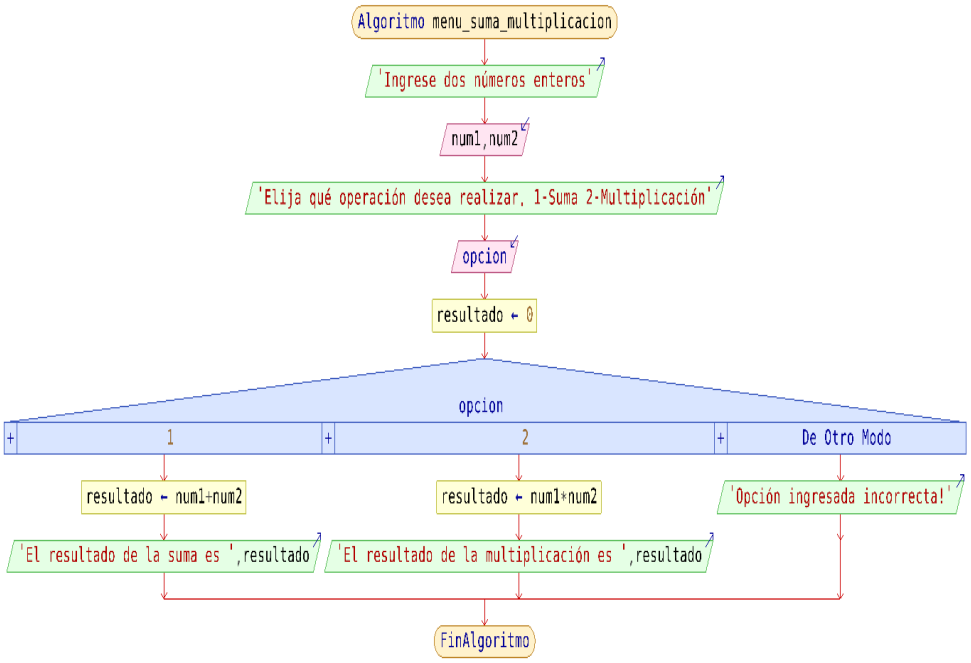
2) Implementar un algoritmo que permita ingresar tres números enteros por teclado y que determine cuál es el mayor de ellos.

```
algoritmo mayor_numero
var
    entero: A, B, C, mayor
inicio
    mostrar("Ingrese tres números enteros")
    leer(A, B, C)

    si A > B entonces
        si A > C entonces
            mayor ← A
        si-no
            mayor ← C
        fin-si
    si-no
        si B > C entonces
            mayor ← B
        si-no
            mayor ← C
        fin-si
    fin-si

    mostrar("El mayor número es ", mayor)
fin
```



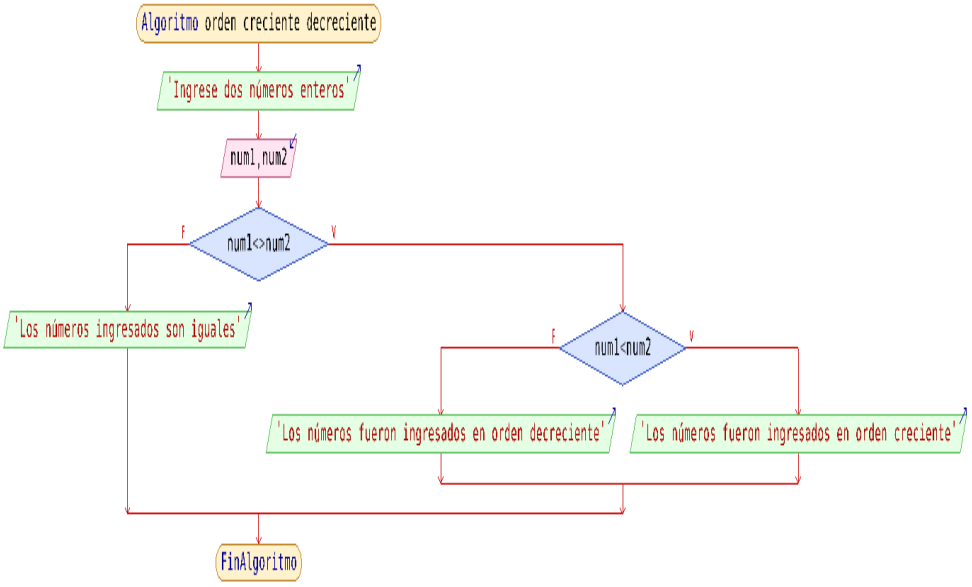
3) Implementar un algoritmo que permita ingresar dos números enteros por teclado. Luego, el usuario debe elegir si desea mostrar la suma o multiplicación de ambos números.

```
algoritmo menu_suma_multiplicacion
var
    entero: num1, num2, resultado, opcion
inicio
    mostrar(“Ingrese dos números enteros”)
    leer(num1, num2)

    mostrar("Elija qué operación desea realizar. 1-Suma 2-Multiplicación")
    leer(opcion)

    resultado ← 0
    según-sea opcion hacer
        caso 1:
            resultado ← num1 + num2
            mostrar("El resultado de la suma es ", resultado)
        caso 2:
            resultado ← num1 * num2
            mostrar("El resultado de la multiplicación es ", resultado)
        si-no:
            mostrar("Opción ingresada incorrecta!")
    fin -segun
fin
```



4) Implementar un algoritmo que permita ingresar dos números enteros por teclado y que determine si fueron ingresados en orden creciente o decreciente.

```
algoritmo orden_creciente_decreciente
var
    entero: num1, num2
inicio
    mostrar(“Ingrese dos números enteros”)
    leer(num1, num2)

    si num1 <> num2 entonces
        si num1 < num2 entonces
            mostrar("Los números fueron ingresados en orden creciente")
        si-no
            mostrar("Los números fueron ingresados en orden decreciente")
        fin-si
    si-no
        mostrar("Los números ingresados son iguales")
    fin-si
fin
```


Algoritmo orden creciente decreciente
'Ingrese dos números enteros'
num1,num2
num1<>num2
F
V
'Los números ingresados son iguales'
num1<num2
F
V
'Los números fueron ingresados en orden decreciente'
'Los números fueron ingresados en orden creciente'
FinAlgoritmo