Estructuras Repetitivas

Programación Lógica

UNIVERSIDAD SIGLO 21 | MIEMBRO DE LA RED ILUMNO

1. Estructuras repetitivas

En el diseño de algoritmos, hay una gran variedad de situaciones que requieren que una o más instrucciones se repitan varias veces. Las instrucciones iterativas o de repetición permiten ejecutar una secuencia de instrucciones más de una vez.

Un bucle es una sección de código que se repite. Es decir que cuando se termina de ejecutar la última instrucción del conjunto, el flujo de control retorna a la primera sentencia y comienza una nueva repetición. Se denomina iteración al hecho de repetir la ejecución de una secuencia de acciones.

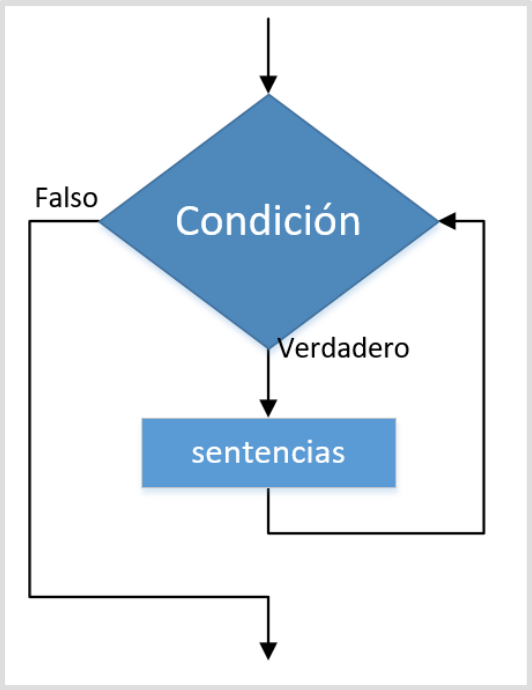
1.1. Estructura *mientras*

Se ejecuta un conjunto de sentencias mientras el resultado de una expresión (condición) sea verdadera. En la estructura *mientras*, se evalúa primero la condición y si esta es verdadera, entonces se ejecutan las sentencias definidas en el bucle. Al finalizar la iteración, se vuelve a evaluar la condición para volver a ejecutar una nueva iteración hasta que la condición dé como resultado un valor *falso*.

Pseudocódigo:

mientras *<condición>* ***hacer***
    sentencia 1
    sentencia 2
    .
    .
    sentencia n
fin-mientras

Figura 1: Diagrama de flujo estructura *mientras*



Fuente: elaboración propia.

En una estructura *mientras*, lo primero que se ejecuta es la evaluación de la condición y si esta es falsa la primera vez que se ingresa a esta estructura, las sentencias asociadas al bucle nunca se ejecutarán. Por ejemplo:

```
dato1 ← 5
dato2 ← 1
mientras dato2 > dato1 hacer
    leer(t)
    mostrar(t)
    dato2 ← dato2 + 1
fin-mientras
```

Las sentencias definidas en el bucle de la estructura *mientras* nunca se ejecutarán, ya que se ejecutan si *dato2* es mayor que *dato1*, pero en este caso *dato1* contiene el entero 5 y *dato2* contiene el entero 1, por lo tanto, *dato2* no es mayor que *dato1*.

Un bucle de una estructura repetitiva que nunca deja de ejecutarse se denomina bucle infinito. Un ejemplo es el siguiente:

```
dato1 ← 5
```

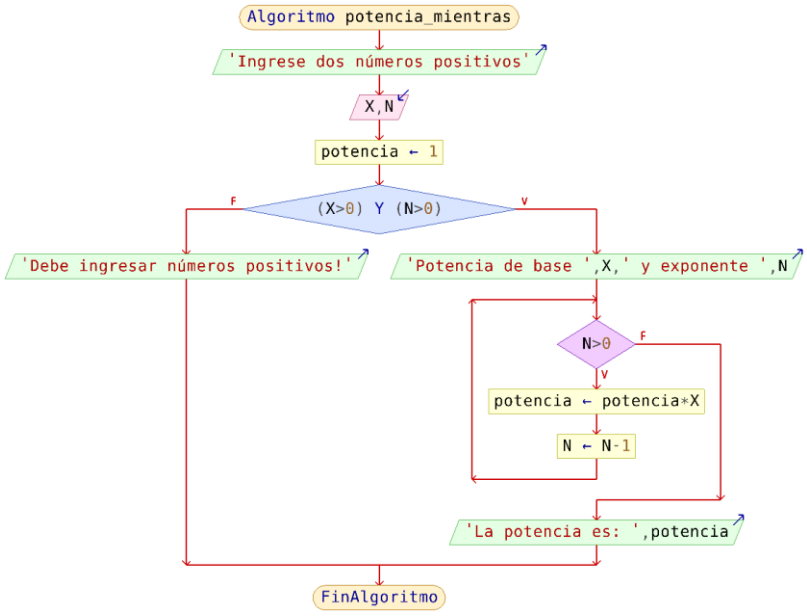
```
dato2 ← 1
mientras dato1 > dato2 hacer
    leer(t)
    mostrar(t)
fin-mientras
```

Las sentencias definidas en el bucle de la estructura *mientras* se ejecutarán infinitamente, ya que estas se ejecutan si *dato1* es mayor que *dato2*. En este caso, *dato1* contiene el entero 5 y *dato2* contiene el entero 1, pero ambas variables nunca se modifican, la primera es siempre mayor que la segunda y resulta siempre la condición como *verdadera*.

Ejercicios de ejemplo

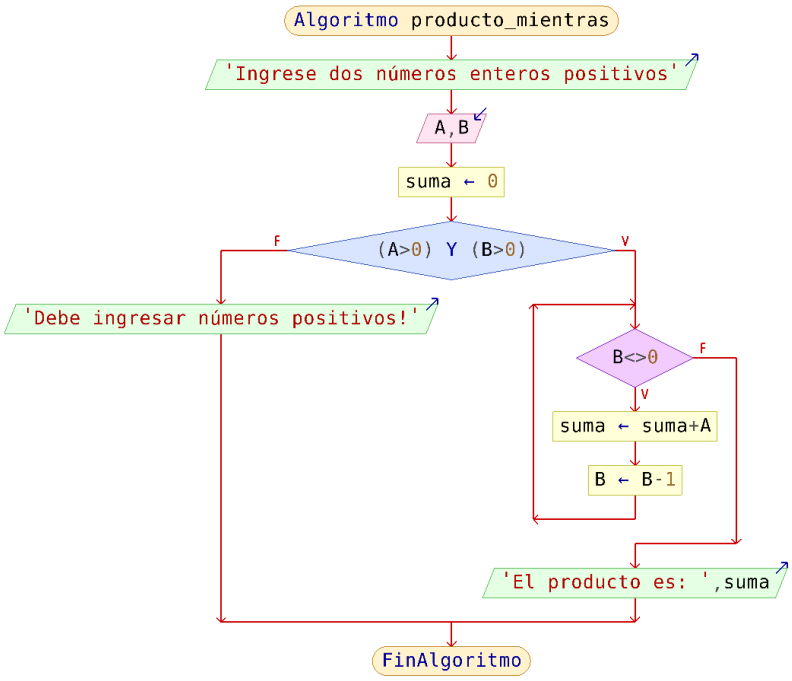
1) Desarrollar un algoritmo que permita ingresar dos números enteros positivos (*X* y *N*) y se muestre por pantalla la *N*-ésima potencia de *X*.

```
algoritmo potencia_mientras
var
    entero: X, N, potencia
inicio
    mostrar("Ingrese dos números positivos")
    leer(X,N)
    potencia ← 1
    si (X > 0) y (N > 0) entonces
            mostrar("Potencia de base ", X, " y exponente ", N)
            mientras N > 0 hacer
                potencia ← potencia * X
                N ← N-1
            fin-mientras
            mostrar("La potencia es: ", potencia)
    si-no
            mostrar("Debe ingresar números positivos!")
    fin-si
fin
```



2) Desarrollar un programa que permita ingresar dos números enteros positivos y se muestre por pantalla el producto de ambos aplicando el algoritmo de sumas sucesivas.

```
algoritmo producto_mientras
var
    entero: A, B, suma
inicio
    mostrar("Ingrese dos números enteros positivos")
    leer(A,B)
    suma ← 0
    si (A > 0) y (B > 0) entonces
        mientras B <> 0 hacer
            suma ← suma + A
            B ← B-1
        fin-mientras
        mostrar("El producto es: ", suma)
    si-no
        mostrar("Debe ingresar números positivos!")
    fin-si
fin
```



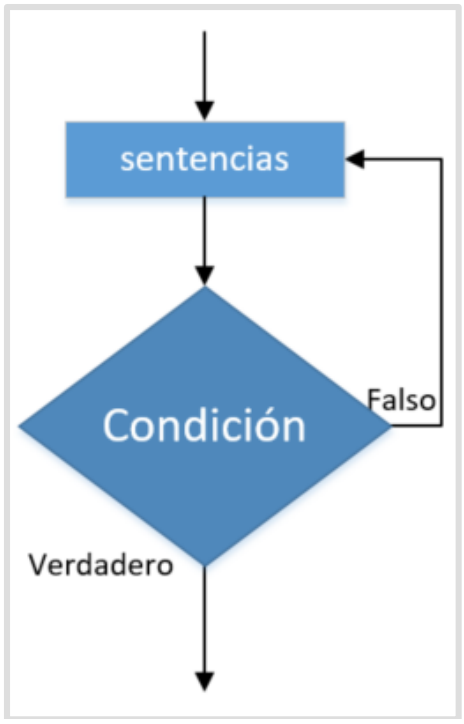
1.2. Estructura *repetir-hasta*

Esta estructura repetitiva se utiliza cuando se desea que se ejecute una iteración al menos una vez antes de comprobar la condición de repetición. Es decir, primero se ejecuta el bucle y luego se comprueba la condición para reproducir una nueva iteración o no del bucle. Esta estructura ejecuta el bucle de repetición mientras el resultado de la condición que se evalúa sea falso.

Pseudocódigo:

repetir *<condición>*
    sentencia 1
    sentencia 2
    .
    .
    sentencia n
hasta-que *<condición>*

Figura 2: Diagrama de flujo estructura *repetir-hasta*



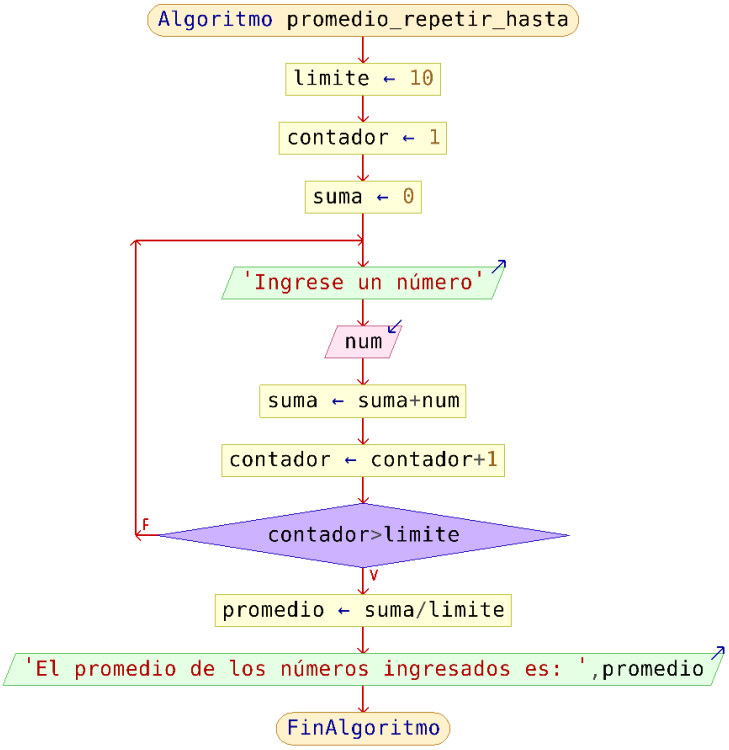
Fuente: elaboración propia.

Ejercicios de ejemplo

1) Desarrollar un algoritmo que permita ingresar 10 números enteros por teclado y que muestre por pantalla el promedio.

```
algoritmo promedio_repetir_hasta
var
    entero: limite, contador, suma, num
    real: promedio
inicio
    limite ← 10
    contador ← 1
    suma ← 0
    repetir
        mostrar("Ingrese un número")
        leer(num)
        suma ← suma + num
        contador ← contador + 1
    hasta-que contador > limite
    promedio ← suma / limite
    mostrar("El promedio de los números ingresados es: ", promedio)
```

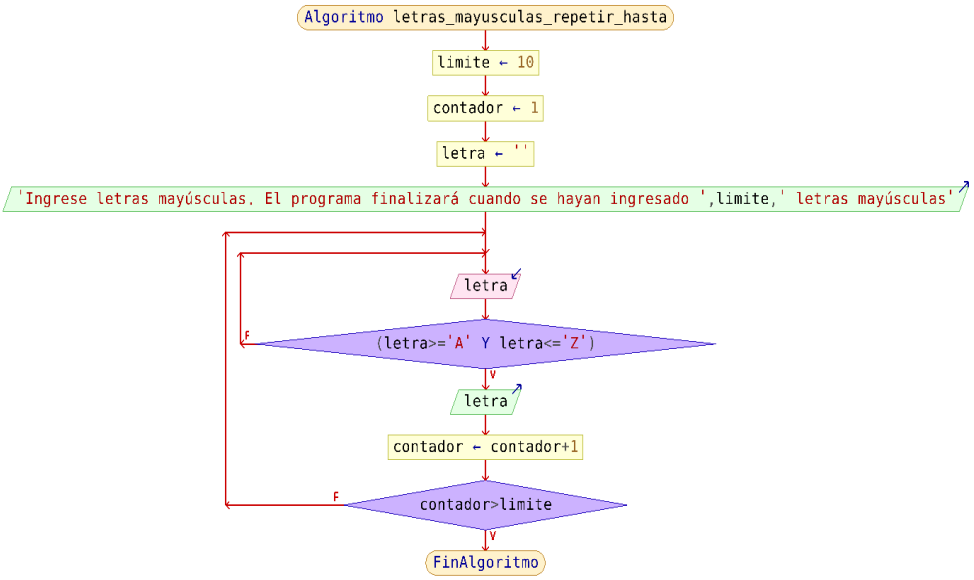
fin



2) Desarrollar un algoritmo que permita ingresar letras mayúsculas y que las muestre por pantalla. El programa debe finalizar cuando se hayan ingresado 10 letras mayúsculas.

algoritmo letras_mayusculas_repetir_hasta
var
    entero: limite, contador
    caracter: letra
inicio
    limite ← 10
    contador ← 1
    letra ← ''
    mostrar("Ingrese letras mayúsculas. El programa finalizará cuando se hayan ingresado ", limite, " letras mayúsculas")
    repetir
        repetir
            leer(letra)
        hasta-que (letra >= 'A' Y letra <= 'Z')

mostrar(letra)
contador ← contador + 1
hasta-que contador > limite
fin



1.3. Estructura *para/desde*

En muchas ocasiones se conoce el número de veces que se desea que un bucle se repita. Una estructura de control muy comúnmente usada en estos casos es el ciclo *para* o *desde*. Se utiliza una variable como índice que se incrementa o decrementa en forma automática al finalizar cada ciclo.

Para ejecutar el bucle, se evalúa una condición que debe ser verdadera. La condición más comúnmente usada es que el valor del índice sea menor o igual a un valor final. Si esto se cumple, entonces se ejecuta el bucle. Al finalizar, se incrementa o decrementa el índice y se vuelve a evaluar la condición. Cuando la condición es falsa, se ignora el bucle y se continúa con el resto del algoritmo.

Pseudocódigo:

desde *v* ← v_i ***hasta*** v_f ***<inc/dec> hacer***
sentencia 1
sentencia 2
.

.
sentencia n
fin-desde

La variable *v* es una variable entera que debe ser declarada. Los valores v_i y v_f son los valores inicial y final que tomará la variable *v*. En el formato de esta estructura, hay que elegir entre incrementar o decrementar la variable de control *v*, ya que no pueden ocurrir al mismo tiempo ambas opciones. Si el incremento es por el valor 1, entonces se puede obviar la sentencia *inc 1* en la estructura. Se pueden utilizar indistintamente las palabras reservadas *desde/fin-desde* y *para/fin-para*.

desde *v* ← *1* ***hasta*** *5* ***hacer***
<sentencias>
fin-desde

En este caso, la variable *v* se inicializa en 1 y, luego de ejecutar todas las sentencias que están comprendidas dentro de la estructura (bucle), automáticamente se incrementa en 1. Después de ejecutar varias iteraciones, cuando el valor de la variable *v* alcanza el valor 6, no se realizan más iteraciones y se prosigue con la próxima instrucción que está a continuación de la sentencia *fin-desde*. El siguiente ejemplo tiene el mismo efecto:

desde *v* ← *5* ***hasta*** *9* ***hacer***
<sentencias>
fin-desde

La variable de control *v* se inicializa en 5, el bucle se repite 5 veces hasta que *v* es igual a 10. En este punto, la condición de que *v* sea menor o igual a 9 es falsa y, en consecuencia, se finaliza la estructura *desde*.

desde *v* ← *1* ***hasta*** *5* ***inc*** *2* ***hacer***
<sentencias>
fin-desde

En este caso, la variable *v* se inicializa con el valor 1 y con la ejecución de cada bucle se incrementa en 2 unidades. El bucle se repite 3 veces para los valores 1, 3 y 5 de la variable *v*, y finaliza cuando esta tiene un valor mayor a 5.

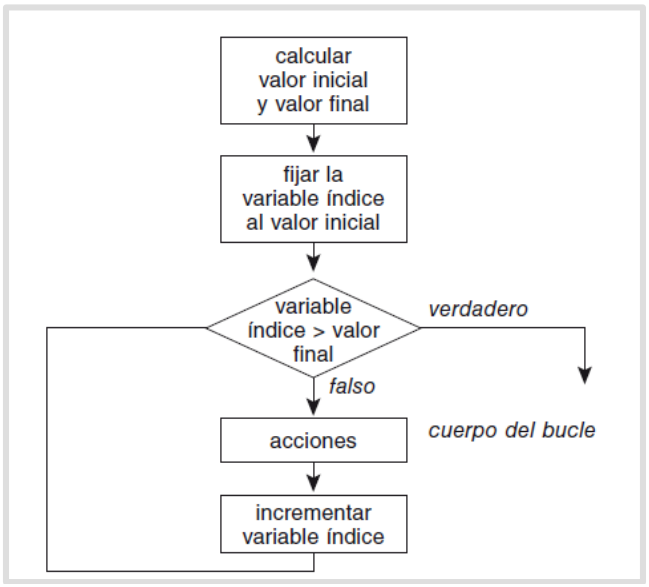
También se puede aplicar la misma lógica de repetición, pero decrementando la variable índice, como en el siguiente ejemplo:

desde *v* ← *10* ***hasta*** *2* ***dec*** *2* ***hacer***

<sentencias>
fin-desde

En este escenario, la variable *v* se inicializa con el valor 10 y con la ejecución de cada bucle se decrementa en 2 unidades. Cuando *v* es menor que 2, entonces se finaliza la estructura repetitiva. El bucle se repite 5 veces para los valores 10, 8, 6, 4 y 2 de la variable *v*, y finaliza cuando esta tiene un valor menor a 2.

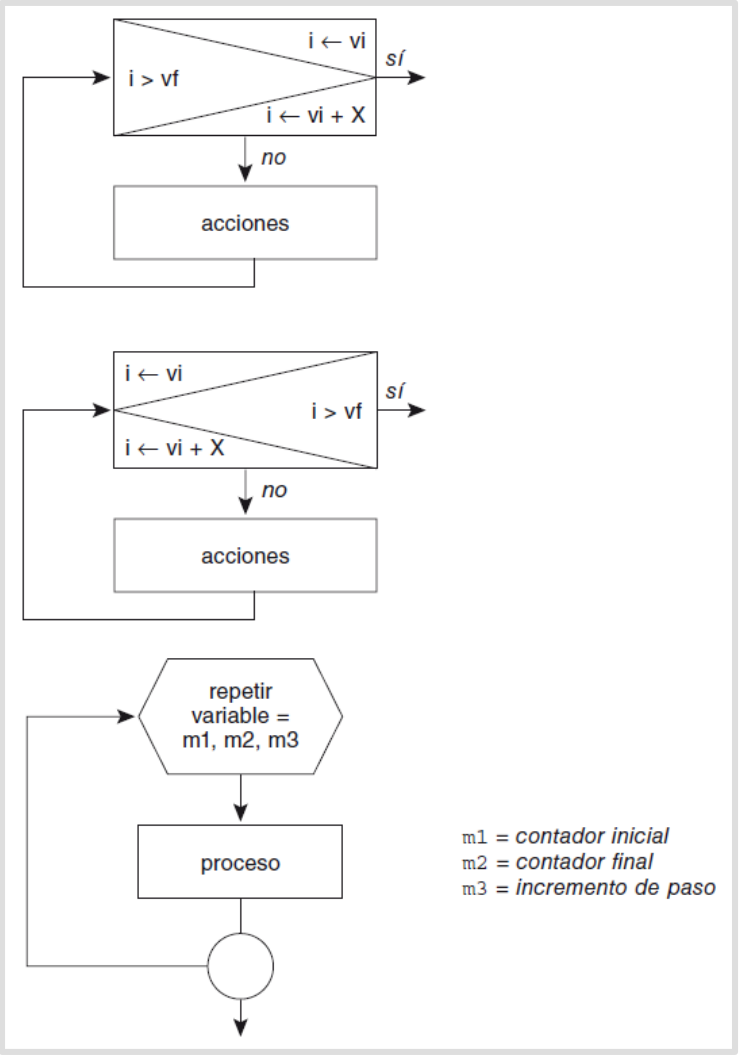
Figura 3: Diagrama de flujo estructura *desde/para*



Fuente: Joyanes Aguilar, 2008, p. 172.

Existen distintas alternativas para representar el diagrama de flujo de la estructura repetitiva *desde*. Algunos autores simplifican la representación de esta estructura utilizando algún símbolo propio como se indica en la Figura 8.

Figura 4: Diagrama de flujo estructura *desde/para*



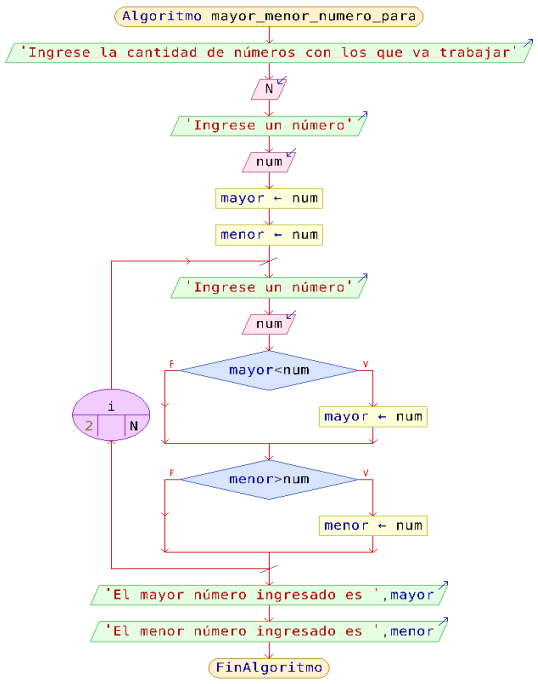
Fuente: Joyanes Aguilar, 2008, p. 173.

Ejercicios de ejemplo

1) Desarrollar un algoritmo que permita ingresar *N* números por teclado y que muestre por pantalla el mayor y menor de ellos.

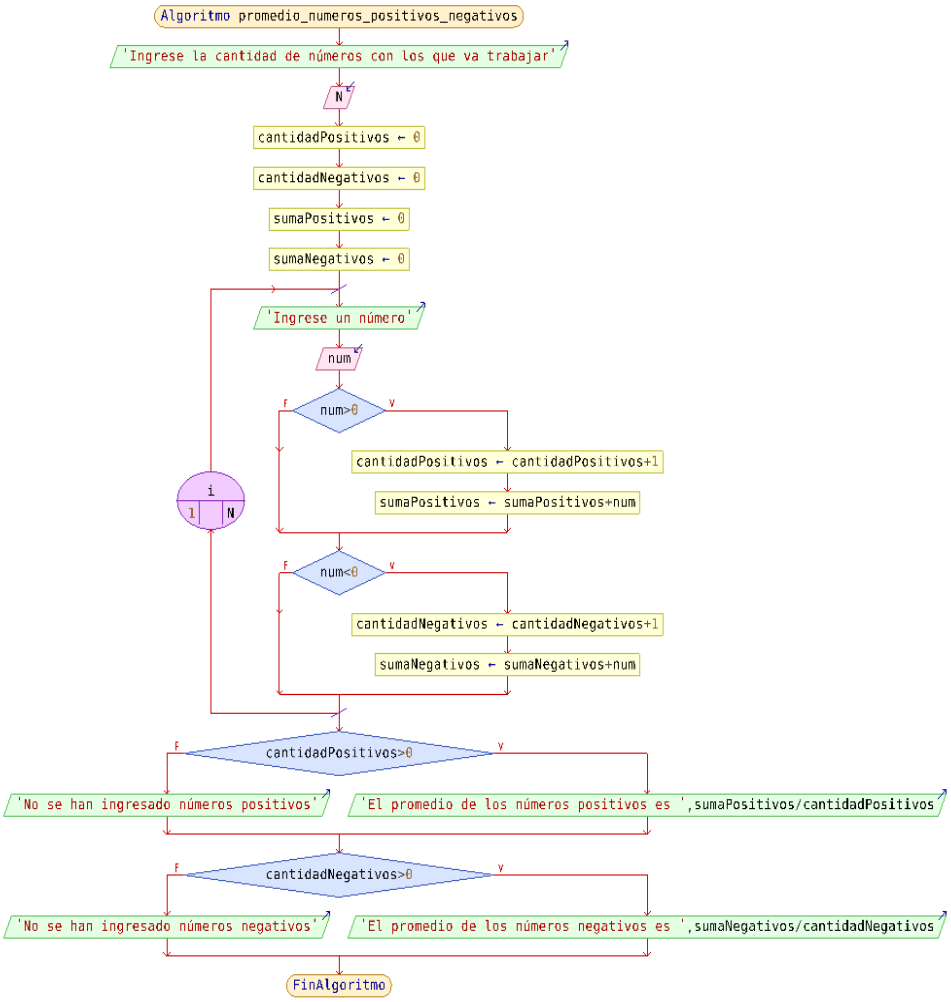
```
algoritmo mayor_menor_numero_para
var
    entero: N, num, mayor, menor, i
inicio
    mostrar("Ingrese la cantidad de números con los que va trabajar")
    leer(N)

    mostrar("Ingrese un número")
```

```
    leer(num)
    mayor ←num
    menor ←num

    para i←2 hasta N hacer
        mostrar("Ingrese un número")
        leer(num)
        si mayor < num entonces
            mayor ←num
        fin-si
        si menor > num entonces
            menor ←num
        fin-si
    fin-para

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



2) Desarrollar un algoritmo que permita ingresar *N* números enteros por teclado y que muestre por pantalla los promedios de los números positivos y negativos.

```
algoritmo promedio_numeros_positivos_negativos
var
    entero: N, cantidadPositivos, cantidadNegativos, sumaPositivos,
sumaNegativos, i
    real: num
inicio
    mostrar("Ingrese la cantidad de números con los que va trabajar")
    leer(N)

    cantidadPositivos ← 0
    cantidadNegativos ← 0
    sumaPositivos ← 0
    sumaNegativos ← 0
    para i←1 hasta N hacer
        mostrar("Ingrese un número")
        leer(num)

        si num > 0 entonces
            cantidadPositivos ← cantidadPositivos + 1
            sumaPositivos ← sumaPositivos + num
        fin-si

        si num < 0 entonces
            cantidadNegativos ← cantidadNegativos + 1
            sumaNegativos ← sumaNegativos + num
        fin-si

    fin-para

    si cantidadPositivos > 0 entonces
        mostrar("El promedio de los números positivos es ",
        sumaPositivos/cantidadPositivos)
    si-no
        mostrar("No se han ingresado números positivos")
    fin-si

    si cantidadNegativos > 0 entonces
        mostrar("El promedio de los números negativos es ",
        sumaNegativos/cantidadNegativos)
    si-no
        mostrar("No se han ingresado números negativos")
    fin-si
fin
```



1.4. Equivalencia entre estructuras repetitivas

Las estructuras repetitivas hasta aquí vistas son equivalentes unas con otras. Dependiendo del problema para solucionar, es posible que alguna de las estructuras de repetición se adapte más fácilmente que otra, pero siempre es posible reemplazarla realizando algunas variaciones sencillas. Por ejemplo, si se deseara acumular en una variable la suma de los 10 primeros números enteros positivos que ingresa un usuario desde el teclado, podríamos tener los siguientes algoritmos:

a) Usando la estructura *mientras*:

```
……
suma ← 0
contador ← 1
mientras contador <= 10 hacer
    leer(dato)
    suma ← suma + dato
    contador ← contador + 1
fin-mientras
……
```

b) Usando la estructura *repetir-hasta*:

```
……
suma ← 0
contador ← 1
repetir
    leer(dato)
    suma ← suma + dato
    contador ← contador + 1
hasta-que contador > 10
……
```

c) Usando la estructura *desde/para*:

```
……
suma ← 0
para contador ← 1 hasta 10 hacer
    leer(dato)
    suma ← suma + dato
fin-para
……
```

Referencias

Joyanes Aguilar, L. (2008). *Fundamentos de programación. Algoritmos, estructura de datos y objetos*. Madrid, ES: McGraw-Hill.