Capítulo 4

4. ESTRUCTURAS DE REPETICIÓN

Hasta ahora todo lo visto permite construir pseudocódigos que se ejecutan tan sólo una vez, es decir, se ejecutan una a una las instrucciones de una manera secuencial hasta llegar a la ejecución de la instrucción **Fin**, sin embargo existen problemas que requieren repetir grupos de instrucciones, por ejemplo, programas que constantemente están requiriendo el ingreso de una cierta cantidad de datos.

Suponga que una compañía tiene 30 empleados y se desea calcular el aumento del sueldo de cada uno de ellos, se debe construir un programa en el cual se pueda ingresar el sueldo de cada uno de los empleados. Con las instrucciones hasta ahora conocidas se produciría un programa muy extenso, la solución es emplear estructuras de repetición.

Las estructuras de repetición y las estructuras de selección pertenecen a las instrucciones de bifurcación y aunque ambas usan una condición para determinar la acción a ejecutar, las estructuras de repetición se diferencian de las de selección porque permiten que ciertas instrucciones se ejecuten más de una vez hasta que se satisfaga la condición. Las estructuras de repetición también son conocidas como ciclos o bucles.

4.1 ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA

La estructura de repetición Mientras Haga tiene el siguiente formato general en el pseudocódigo, ver Ilustración 34:

Mientras (**condición**) **haga**

Instrucción 1
Instrucción 2
:
Modificador de Condición
Instrucción N
Fin_mientras

Ilustración 34. Formato general pseudocódigo para la estructura de repetición Mientras Haga.

La estructura de repetición Mientras Haga contiene tres palabras reservadas: **Mientras**, **haga** y **Fin_mientras**. Comienza con la palabra **Mientras** y termina con **Fin_mientras**. Además, contiene un cuerpo de instrucciones entre ambas palabras, y una **condición** dentro de paréntesis. La **condición** al ser evaluada debe siempre generar un valor de tipo booleano: **verdadero** o **falso**. Dentro del cuerpo de instrucciones debe contener una en particular, el Modificador de Condición, esta instrucción se encarga de cambiar el contenido de la variable que se encuentra en la **condición**, esto permite que en algún momento se termine la ejecución del ciclo. La ausencia o mala construcción del Modificador de Condición puede ocasionar ciclos infinitos – ciclos que nunca terminan – y por tanto, se pierde el control de la ejecución del programa.

La **condición** se puede construir de dos formas:

- Usando los operadores relacionales y/o operadores lógicos en una expresión que al ser resuelta genera un valor booleano
- O usando una variable de tipo booleano.

En ambos casos, las variables empleadas en la construcción de la **condición** deben estar declaradas y contener un valor antes de ser evaluadas.

Cuando la **condición** genera un resultado **verdadero** automáticamente se ejecuta de una manera secuencial el cuerpo de instrucciones, incluida la instrucción llamada Modificador de Condición; al ejecutarse la última instrucción -Instrucción N- la ejecución del programa se devuelve a evaluar de nuevo la **condición**, si ésta genera un resultado **verdadero**, ingresa nuevamente a la estructura de repetición y ejecuta todo el cuerpo de instrucciones, incluido el Modificador de Condición; este proceso se repite cada vez que la **condición** genere un resultado verdadero como se muestra en la Figura 4.2.

Cada ingreso al cuerpo de instrucciones de toda estructura de repetición se le conoce como Iteración, la cantidad de iteraciones que se lleve a cabo en un programa dependerá del límite puesto en la **condición**. Con cada iteración el contenido de la variable que se encuentra en la **condición** va cambiando debido a la ejecución de la instrucción Modificador de Condición hasta que alcance, sobrepase o sea igual al límite, cuando esto ocurre se termina la ejecución de la estructura de repetición.

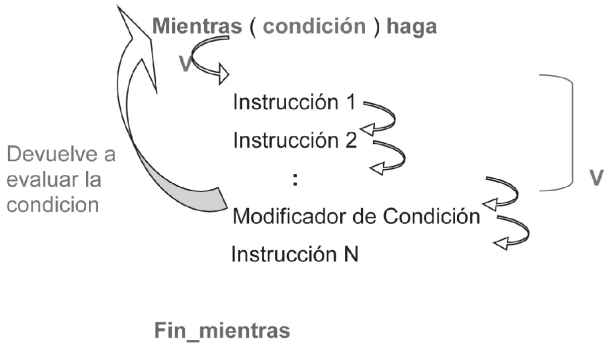


Figura 4.2 Funcionamiento de la estructura de repetición Mientras Haga cuando la **condición** da **verdadero**.

Cuando la **condición** genera un resultado **falso**, la ejecución del programa salta a la palabra reservada **Fin_mientras** cerrando la estructura de repetición y luego la ejecución continua con las instrucciones siguientes, en la figura 4.3 se muestra este comportamiento.

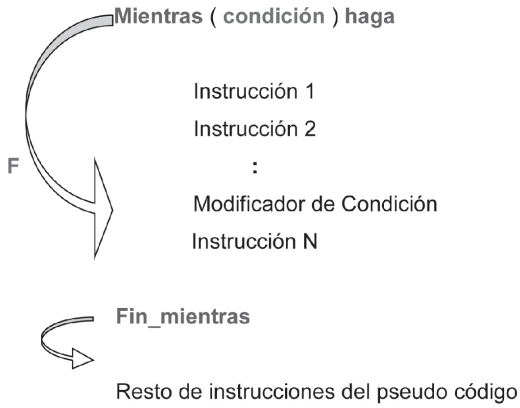


Figura 4.3 Funcionamiento de la estructura de repetición Mientras Haga cuando la **condición** da **falso**.

La siguiente figura muestra el formato general en el diagrama de flujo para de la estructura de repetición Mientras Haga.

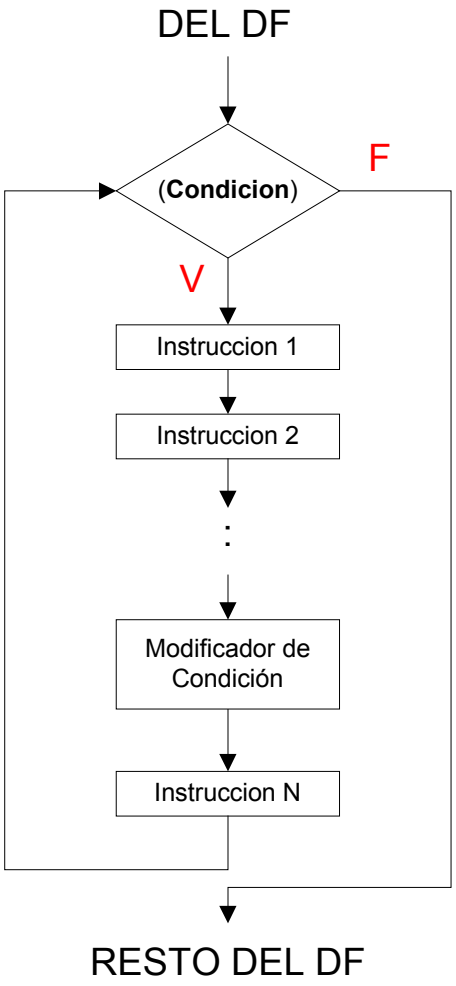


Figura 4.4 Formato general diagrama de flujo para la estructura de repetición Mientras Haga.

En la Figura 4.4 no se aprecia las palabras reservadas **Mientras**, **haga** y **Fin_mientras**, éstas no tienen representación en el diagrama de flujo. Como se observa en el diagrama de flujo, en el camino del **falso** no existe instrucción alguna y llega directamente para continuar con la ejecución del resto de instrucciones – RESTO DEL DF –.

4.2 ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS

La estructura de repetición Haga Mientrastiene el siguiente formato general en el pseudocódigo:

Haga

Instrucción 1
Instrucción 2
:
Modificador de Condición
Instrucción N

Mientras (condición)

Figura 4.5 Formato general pseudocódigo para la estructura de repetición Haga Mientras.

La estructura de repetición Haga Mientrascontiene dos palabras reservadas: **Haga** y **Mientras**. Comienza con la palabra **Haga** y termina con **Mientras** y dentro de éstas se encuentra el cuerpo de instrucciones el cual incluye el Modificador de Condición. Además, contiene una **condición** dentro de paréntesis. La **condición** al ser evaluada debe siempre generar un valor de tipo booleano: **verdadero** o **falso**. En la **condición** se deben usar los operadores relacionales y/o lógicos, y sus variables deben estar declaradas y contener algún dato.

Al ejecutarse la palabra reservada **Haga**, el programa interpreta que se está ingresando a una estructura de repetición, luego se sigue con la ejecución de cada una de las instrucciones que hacen parte del cuerpo, incluida el Modificador de Condición, al ejecutarse ésta se modifica el contenido de la variable que se encuentra en la **condición**. Después de la ejecución de la instrucción N, se evalúa la **condición**, si ésta genera un resultado **verdadero**, la ejecución del programa salta automáticamente a la primera instrucción que se encuentra después de la palabra reservada **Haga** y este proceso se repite toda vez que la **condición** genere un resultado **verdadero**.

En la estructura de repetición Haga Mientrasse evalúa la **condición** al final, por lo que todas las instrucciones que hacen parte de la estructura se ejecutan antes de la condición, es decir, primero se ejecuta y luego se pregunta. Esa es la principal diferencia con la estructura de repetición Mientras Haga.

La siguiente figura muestra el funcionamiento de la estructura de repetición Haga Mientrascuando la **condición** genera un resultado **verdadero**.

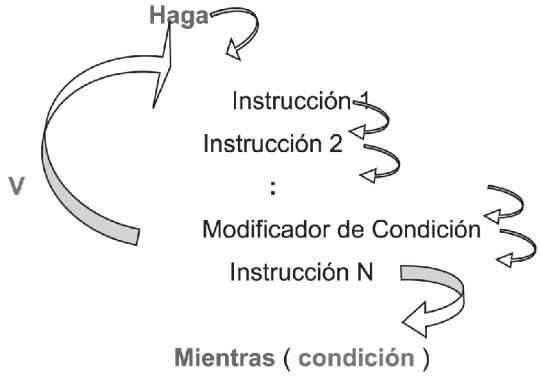


Figura 4.6 Funcionamiento de la estructura de repetición Haga Mientrascuando la **condición** da **verdadero**.

Cuando la **condición** genera **falso** se termina la ejecución de la estructura y se continúa con la ejecución del resto de instrucciones del pseudocódigo. La siguiente figura muestra este comportamiento.

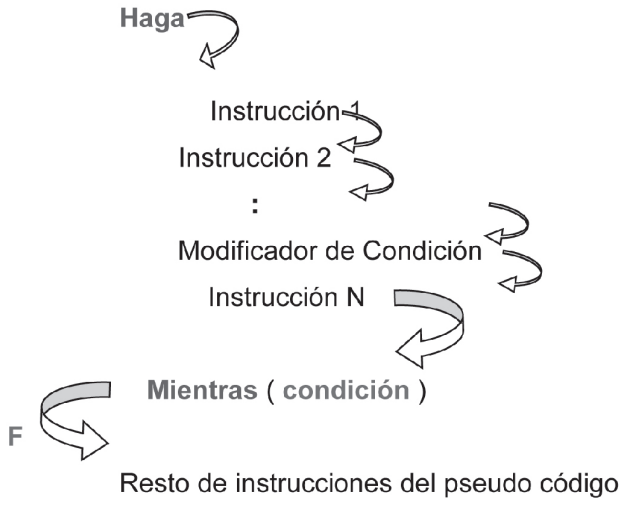


Figura 4.7 Funcionamiento de la estructura de repetición Haga Mientrascuando la **condición** da **falso**.

La siguiente figura muestra el formato general del diagrama de flujo para la estructura de repetición Haga Mientras.

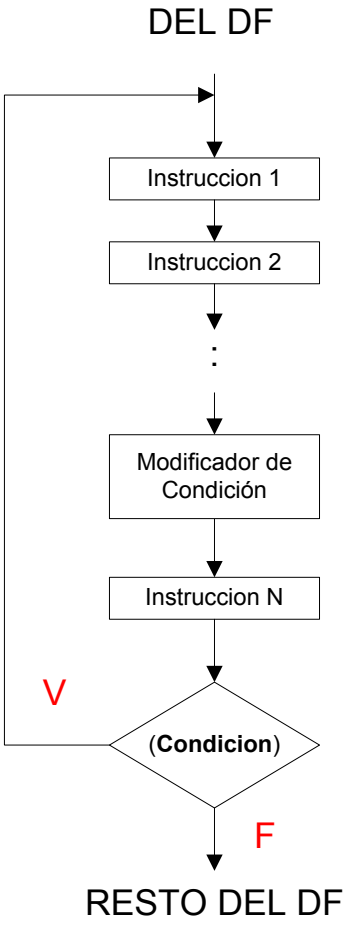


Figura 4.8 Formato general diagrama de flujo para la estructura de repetición Haga Mientras.

En la figura 4.8 no se aprecia las palabras reservadas **Haga** y **Mientras**, éstas no tienen representación en el diagrama de flujo. Como se observa el camino del **falso** va directamente a continuar con la ejecución del resto de instrucciones – RESTO DEL DF –.

4.3 ESTRUCTURA DE REPETICIÓN REPITA HASTA

La estructura de repetición Repita Hasta tiene el siguiente formato general en el pseudocódigo:

Repita

Instrucción 1
Instrucción 2
:
Modificador de Condición
Instrucción N

Hasta (**condición**)

Figura 4.9 Formato general pseudocódigo para la Estructura de Repetición Repita Hasta.

La estructura de repetición Repita Hasta contiene dos palabras reservadas: **Repita** y **Hasta**. Comienza con la palabra **Repita** y termina con **Hasta** y dentro de éstas se encuentra el cuerpo de instrucciones el cual incluye el Modificador de Condición. Además, contiene una **condición** dentro de paréntesis. La **condición** al ser evaluada debe siempre generar un valor de tipo booleano: **verdadero** o **falso**. En la **condición** se deben usar los operadores relacionales y/o lógicos, y sus variables deben estar declaradas y contener algún dato.

Igual que en la estructura de repetición Repita Hasta, la estructura de repetición Repita Hasta ejecuta las instrucciones asociadas a la estructura por lo menos una vez y luego pregunta. Sin embargo, en una estructura de repetición Repita Hasta se repite la ejecución del cuerpo de instrucciones, cuando la **condición** genera **falso** como resultado.

El Modificador de Condición cumple las mismas funciones en esta estructura de repetición como en las anteriores.

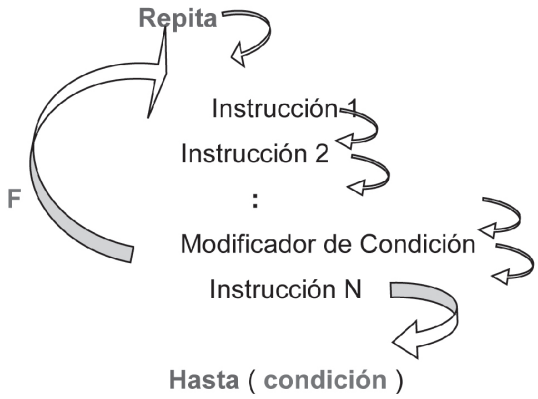


Figura 4.10 Funcionamiento de la estructura de repetición Repita Hasta cuando la **condición** da **falso**.

Cuando la **condición** genera un resultado **verdadero**, termina la ejecución de la estructura de repetición Repita Hasta y se continua con la ejecución del resto de instrucciones del pseudocódigo, como muestra la Figura 4.11.

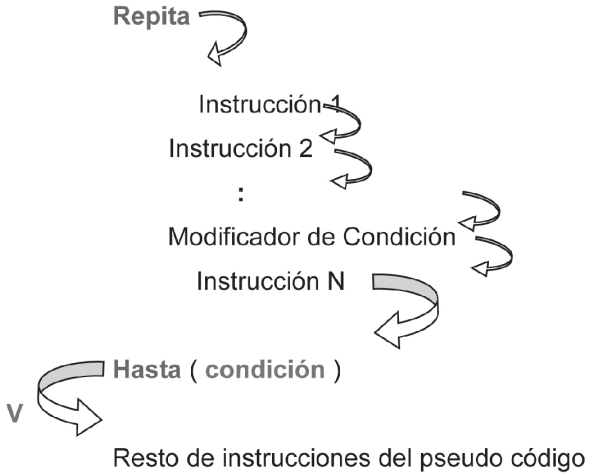


Figura 4.11 Funcionamiento de la estructura de repetición Repita Hasta cuando la **condición** da **verdadero**.

En la figura 4.12 se puede apreciar el formato general del diagrama de flujo para la estructura de repetición Repita Hasta.

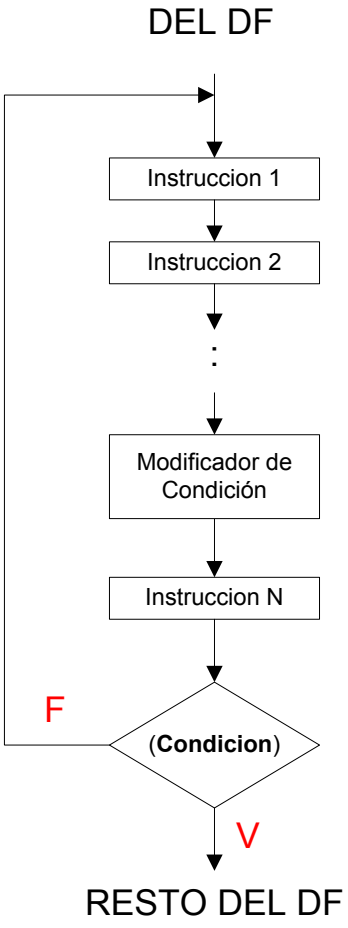


Figura 4.12 Formato general diagrama de flujo para la estructura de repetición Repita Hasta.

Note que la única diferencia entre las figuras 4.8 y 4.12 es la ubicación de las letras **V** y **F**. Igual que en la Figura 4.8, la Figura 4.12 no posee las palabras reservadas **Repita** y **Hasta**.

4.4 ESTRUCTURA DE REPETICIÓN PARA

La Estructura de Repetición Para tiene el siguiente formato general en el pseudocódigo:

Para V ← Vi hasta Vf (INC o DEC) haga

Instrucción 1
Instrucción 2
:
Instrucción N

Fin_para

Figura 4.13 Formato general pseudocódigo para la Estructura de Repetición Para.

La estructura de repetición Para contiene las siguientes palabras reservadas: **Para**, **hasta**, **haga** y **Fin_para**. Comienza con la palabra **Para** y termina con **Fin_para** y dentro de éstas se encuentra el cuerpo de instrucciones. Está conformada por una Inicialización (**V ← Vi**), una variable final **Vf** con la cual **V** comparara su valor, lo que correspondería a la **condición**, y dos palabras reservadas - **INC**, **DEC** - que permiten el incremento o decremento de la variable **V**.

La estructura de repetición Para es la única estructura de repetición que no contiene entre sus instrucciones la llamada Modificador de Condición, ésta es reemplazada por las palabras reservadas **INC** – incremento – o **DEC** – decremento – las cuales se colocan directamente dentro del ciclo Para.

En la Figura 4.13, **Vi** y **Vf** significan Valor Inicial y Valor Final respectivamente y son variables que contienen valores enteros por tanto, la estructura de repetición Para es utilizada únicamente para aquellos problemas donde **V** desea alcanzar el valor de **Vf** y superarlo. Las variables **Vi**, **V** y **Vf** deben estar declaradas y **Vi** y **Vf** deben contener valores antes de ejecutar la estructura de repetición. Se puede utilizar la estructura de repetición Para sin las variables **Vi** y **Vf**, simplemente colocando los valores numéricos respectivos, si este es el caso no se necesita declarar las dos variables.

Cuando se ingresa por primera vez a la estructura de repetición Para y sólo en esta única ocasión, a **V** se le asigna el valor de **Vi** y este valor se compara con el que contiene **Vf** de la siguiente manera:

1. Cuando se utiliza el operador menor o igual que (≤) para comparar a **V** con **Vf**, es decir se realiza **V ≤ Vf,** si el resultado es **verdadero**, se ingresa a la estructura de repetición y se ejecuta el cuerpo de instrucciones asociado a ésta. Después de la última instrucción se incrementa el valor de **V**, para lograr esto se usa la palabra reservada **INC,** si se desea incrementar de uno en uno se coloca **INC** 1 – la ausencia de esta palabra también puede ser interpretada como un incremento de uno en uno –, en forma general, el número que acompaña la palabra **INC** indica el valor del incremento, **INC** 1 se puede representar como:

INC 1 es lo mismo que V ← V + 1

Una vez se incrementa a **V** se compara su nuevo valor con el de **Vf** es decir, **V ≤ Vf**, si el resultado es **verdadero** se ejecuta nuevamente el cuerpo de instrucciones, luego se ejecuta el **INC** y, después se compara el nuevo valor de **V** con **Vf**, este proceso se repite cada vez que **V ≤ Vf** genere un resultado **verdadero**. En la Figura 4.14 se aprecia este comportamiento.

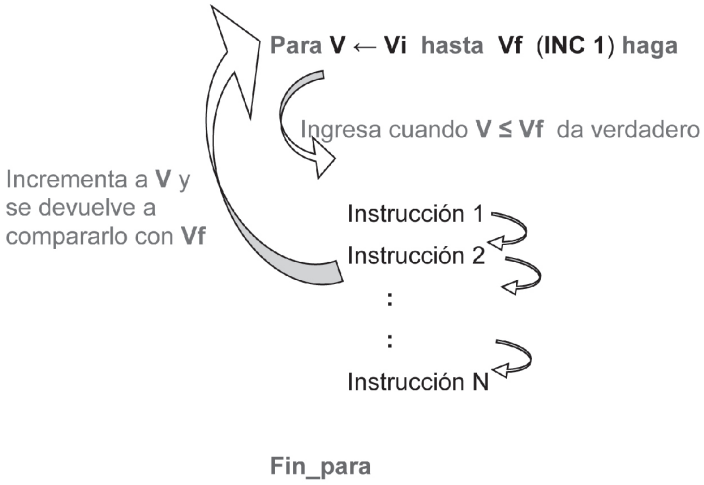


Figura 4.14 Funcionamiento de la estructura de repetición Para usando **INC** cuando **V ≤ Vf** da **verdadero**.

Cuando **V** supera el valor de **Vf**, la comparación **V ≤ Vf** genera un resultado **falso**, la ejecución del programa salta automáticamente a **Fin_para** y lo ejecuta cerrando el ciclo Para y se continua con la ejecución del resto de instrucciones del pseudocódigo, como muestra la Figura 4.15.

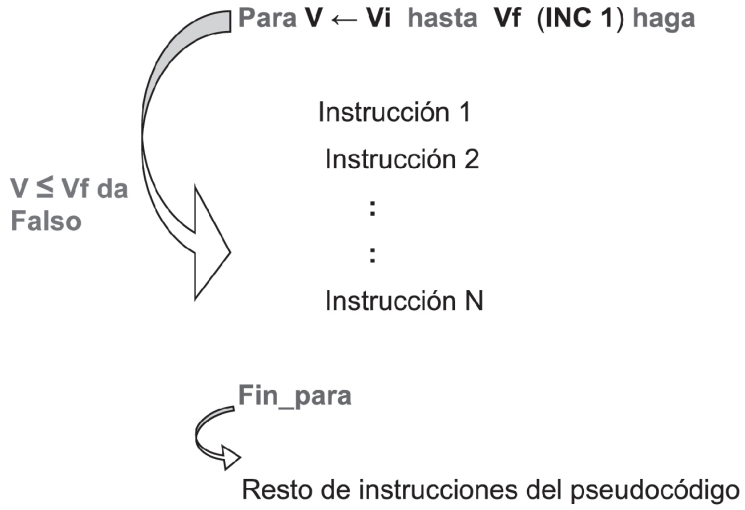


Figura 4.15 Funcionamiento de la estructura de repetición Para usando **INC** cuando **V ≤ Vf** da **falso**.

Si **Vi** es menor que **Vf** es recomendable nunca usar el decremento, **DEC**, ya que esto provocará que el valor de **V** disminuya en vez de aumentar por tanto, la **ERP** nunca terminaría su ejecución.

2. Cuando se utiliza el operador mayor o igual que (≥) para comparar a **V** con **Vf**, es decir se realiza **V ≥ Vf,** si el resultado es **verdadero**, se ingresa a la estructura de repetición y se ejecuta el cuerpo de instrucciones asociado a ésta, después de ejecutar la última instrucción se decrementa el valor de **V**, para lograr esto se usa la palabra reservada **DEC** si se desea decrementar de uno en uno se coloca **DEC** 1, en forma general, el número que acompaña la palabra **DEC** indica el valor del decremento, **DEC** 1 se puede representar como:

DEC 1 es lo mismo que V ← V – 1

Una vez se decrementa a **V** se compara su nuevo valor con el de **Vf** es decir, **V ≥ Vf**, si el resultado es verdadero se ejecuta nuevamente el cuerpo de instrucciones de la **ER**, luego se ejecuta el **DEC** y, después se compara el nuevo valor de **V** con **Vf**, este proceso se repite cada vez que **V ≥ Vf** genere un resultado verdadero. En la Figura 4.16 se aprecia este comportamiento.

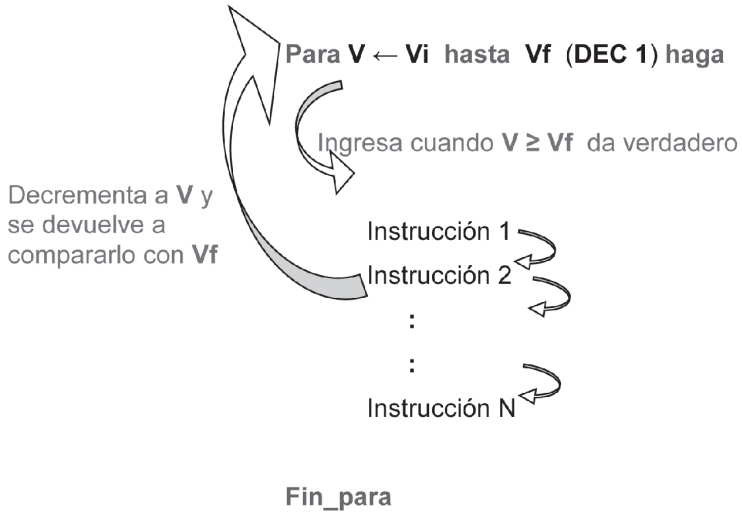


Figura 4.16 Funcionamiento de la estructura de repetición Para usando **DEC** cuando **V ≥ Vf** da **verdadero**.

Cuando **V** es inferior al valor de **Vf**, la comparación **V ≥ Vf** genera un resultado **falso**, la ejecución del programa salta automáticamente a **Fin_para** y lo ejecuta cerrando el ciclo Para y se continua con la ejecución del resto de instrucciones del pseudocódigo, como muestra la Figura 4.17.

Si **Vi** es mayor que **Vf** no se debe usar el incremento, **INC**, ya que esto provocará que el valor de **V** aumente en vez de disminuir por tanto, la estructura de repetición Para nunca terminaría su ejecución.

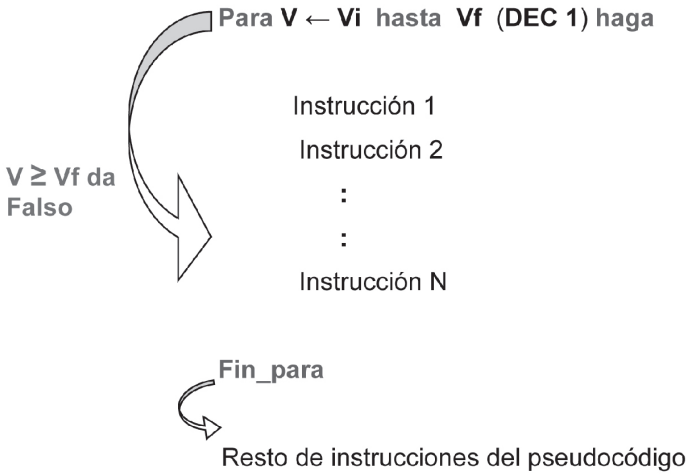


Figura 4.17 Funcionamiento de la estructura de repetición Para usando **DEC** cuando **V ≥ Vf** da **falso**.

La siguiente figura muestra el formato general del diagrama de flujo para la Estructura de Repetición Para.

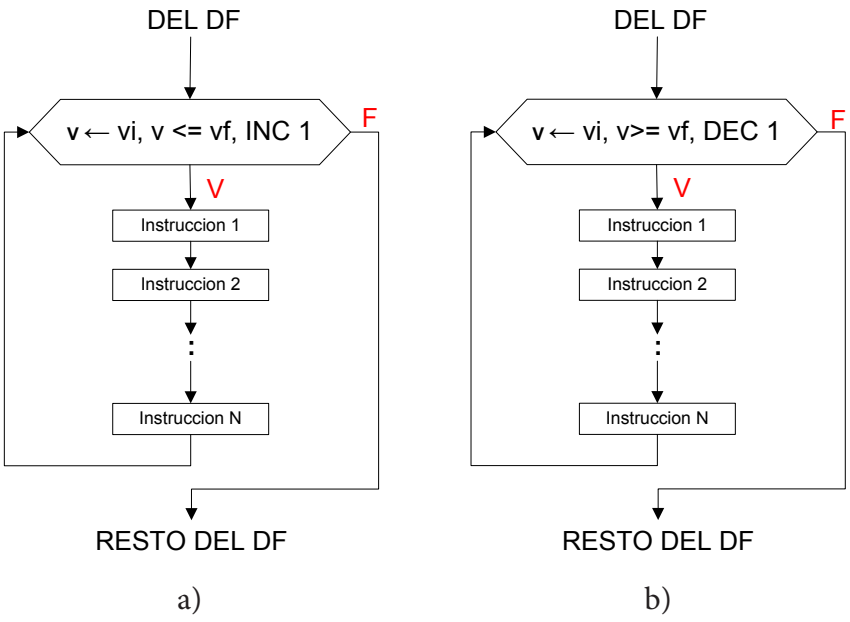


Figura 4.18 Formato general diagrama de flujo para la estructura de repetición Para. a) Con **V <= Vf**, b) Con **V >= Vf**

4.5 CONSTRUCCIÓN DE ESTRUCTURAS DE REPETICIÓN

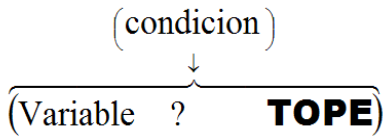
Si se necesita realizar una operación o cálculo más de una vez, se debe utilizar una estructura de repetición. Aquello que se debe realizar más de una vez representa el cuerpo de instrucciones de la estructura de repetición.

Conocidas las notas finales de un curso de 30 estudiantes, construya un programa que encuentre el promedio de notas del salón.

Para calcular el promedio del salón es necesario conocer todas las notas y para ello es necesario pedir la nota de cada estudiante, este proceso se debe hacer 30 veces por tanto, se necesita una estructura de repetición.

A continuación se presenta una guía para ayudar al lector a comprender cómo construir la **condición**.

En la **condición** debe existir una variable que contendrá el dato a comparar con el valor al cual se desea llegar. El valor deseado o el valor al cual se desea llegar son conocidos como **Tope**. En la estructura de repetición Para el valor de la variable depende de la palabra reservada **INC** o **DEC**. Por tanto, hasta el momento se conoce que la **condición** está conformada de la siguiente forma:



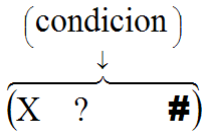
La **condición** debe generar uno de dos valores posibles **verdadero** o **falso** por lo que el símbolo de interrogación indica que en esa posición se debe colocar un operador relacional y/o lógico.

Cuando el Tope es de tipo **Numérico** existen dos casos:

1. Cuando un número en el enunciado representa la cantidad de veces que hay que realizar la misma operación. En este caso se debe construir una instrucción llamada **Contador**, que normalmente cuenta de uno en uno:

$$\mathbf{X \leftarrow X + 1}$$

La variable que aparece en el contador es la misma variable que se coloca en la **condición** entonces, hasta ahora la **condición** tiene la siguiente forma:



Donde el símbolo de numeral representa un Tope numérico. La variable **X** debe estar declarada y contener un dato. Asignar a **X** un valor de arranque antes de ejecutarse el contador se conoce como Inicialización y tiene generalmente la forma:

$$\mathbf{X \leftarrow 0} \quad \text{o} \quad \mathbf{X \leftarrow 1}$$

La Inicialización se coloca antes de la estructura de repetición por lo que los formatos del pseudocódigo para las estructuras de repetición Mientras Haga, Haga Mientras y Repetir Hasta quedan como se muestra a continuación:

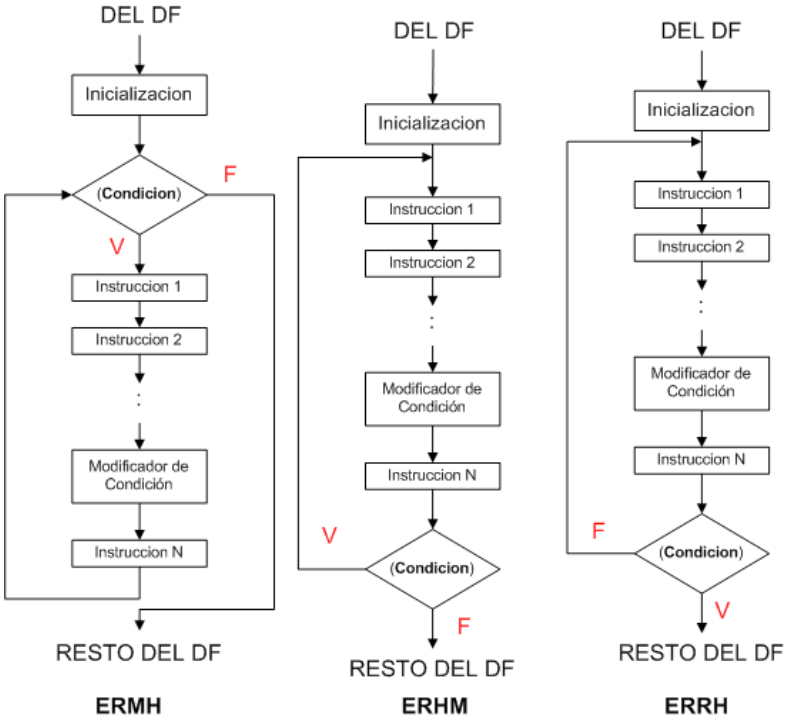


Figura 4.19 Formatos de las estructuras de repetición Mientras Haga, Haga Mientras y Repetir Hasta con Inicialización.

Hasta el momento se tiene para cualquier estructura de repetición excepto la Para, lo siguiente:

Mientras Haga	**Haga Mientras**	**Repita Hasta**
Inicialización	Inicialización	Inicialización
Mientras (***Condición***) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
Modificador de Condición	Modificador de Condición	Modificador de Condición
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (***Condición***)	**Hasta** (***Condición***)

Figura 4.20 Formatos de las Mientras Haga, Haga Mientras y Repetir Hasta con Inicialización en el pseudocódigo.

Reemplazando lo hasta ahora visto, se tiene

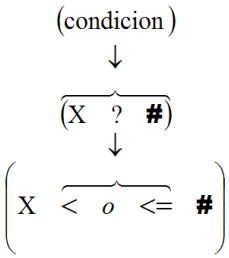
Mientras Haga	**Haga Mientras**	**Repita Hasta**
X ← 0	**X ← 0**	**X ← 0**
Mientras (**X ? TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
X ← X + 1	**X ← X + 1**	**X ← X + 1**
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (**X ? TOPE**)	**Hasta** (**X ? TOPE**)

Figura 4.21 Formatos de las Mientras Haga, Haga Mientras y Repetir Hasta con la instrucción de Inicialización y del Contador en el pseudocódigo.

El Tope en este caso, representa la cantidad de veces que se debe ejecutar la estructura de repetición, por lo que, el Tope es el valor al cual **X** desea llegar y debe ser inferior a Tope.

Basándose en lo anterior, el operador que reemplazaría al símbolo de interrogación dentro de la condición es un operador relacional: el menor que - **<** - o,

el menor o igual que - ≤ -. El operador puesto en la **condición** debe permitir que se pueda ingresar a la estructura de repetición, a ***condición*** se convierte en:



Supóngase que se desea mostrar en pantalla cinco veces la frase "hola", el trozo de pseudocódigo que realiza esto es el siguiente:

	Prueba de escritorio		
X ← 0	X	**Visualización**	**Comentarios**
Mientras (X < 5) haga	0	hola	Cumple; ejecuta Escribir; incrementa a X
Escribir ("hola")	1	hola	Cumple; ejecuta Escribir; incrementa a X
X ← X + 1	2	hola	Cumple; ejecuta Escribir; incrementa a X
Fin_mientras	3	hola	Cumple; ejecuta Escribir; incrementa a X
	4	hola	Cumple; ejecuta Escribir; incrementa a X
	5		No cumple, X tiene el mismo valor del **Tope**

En el ejemplo anterior, usando el operador menor que, la estructura de repetición se ejecutó 5 veces, lo estipulado en el **TOPE**. Cuando la variable **X** tiene el valor de 5 la **condición** no se cumple y se termina la ejecución de la estructura de repetición Mientras Haga.

Cambiando a menor o igual se tiene:

	Prueba de escritorio		
X ← 0	X	**Visualización**	**Comentarios**
Mientras (X <= 5) haga	0	hola	Cumple; ejecuta Escribir; incrementa a X
Escribir ("hola")	1	hola	Cumple; ejecuta Escribir; incrementa a X
X ← X + 1	2	hola	Cumple; ejecuta Escribir; incrementa a X
Fin_mientras	3	hola	Cumple; ejecuta Escribir; incrementa a X
	4	hola	Cumple; ejecuta Escribir; incrementa a X
	5	hola	Cumple; ejecuta Escribir; incrementa a X
	6		No cumple, X supera el valor del **TOPE**

Resolviendo el mismo ejemplo pero usando el operador menor o igual que, se visualizó seis veces la palabra *hola* no cumpliendo esto con el enunciado, por tanto, el operador menor o igual que, no satisface la solución del problema con una Inicialización de **X** en 0. Sin embargo, al cambiar la Inicialización a 1 y dejando el operador menor o igual que, se tiene:

	Prueba de escritorio		
X ← 1	X	**Visualización**	**Comentarios**
Mientras (X <= 5) haga	1	hola	Cumple; ejecuta Escribir; incrementa a X
Escribir ("hola")	2	hola	Cumple; ejecuta Escribir; incrementa a X
X ← X + 1	3	hola	Cumple; ejecuta Escribir; incrementa a X
Fin_mientras	4	hola	Cumple; ejecuta Escribir; incrementa a X
	5	hola	Cumple; ejecuta Escribir; incrementa a X
	6		No cumple, X supera el valor del **TOPE**

Con el ejemplo anterior se concluye lo siguiente para estructura de repetición Mientras Haga, estructura de repetición Haga Mientrasy estructura de repetición Repetir Hasta, si se usa:

- **X ← 0** se utiliza el operador es **<**
- **X ← 1** se utiliza el operador es **<=**

Reemplazando lo hasta ahora visto, se tiene

Mientras Haga	**Haga Mientras**	**Repetir Hasta**
X ← 0	**X ← 0**	**X ← 0**
Mientras (**X < TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
X ← X + 1	**X ← X + 1**	**X ← X + 1**
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (**X < TOPE**)	**Hasta** (**X >= TOPE**)

Figura 4.22 Formatos de las estructura de repetición Mientras Haga, Haga Mientras y Repetir Hasta cuando el Tope es numérico, con Inicialización, Contador y condición.

Cuando se usa una estructura de repetición Repetir Hasta si se inicializa con 0 se debe usar el operador mayor o igual que (véase la Figura 4.22), y el símbolo > cuando se inicializa con 1, esto porque se busca que al menos se genere la repetición de la misma una vez.

Cuando se usa la estructura de repetición Para debe tenerse en cuenta lo siguiente: el valor de la Inicialización debe guardarse en la variable **Vi**. El Tope se coloca donde va la variable **Vf**. **INC** o **DEC** cumplirán con la función del Contador. La siguiente figura muestra la aplicación del Tope numérico en la estructura de repetición Para.

```
Para X ← Vi hasta Vf (INC) haga
  Instrucción 1
  Instrucción 2
      :
      :
  Instrucción N
Fin_para
```

a)

```
Para X ← 1 hasta TOPE (INC) haga
  Instrucción 1
  Instrucción 2
      :
      :
  Instrucción N
Fin_para
```

b)

Figura 4.23 Formato de la estructura de repetición Para con Tope numérico. a) Usando las variables Vi y Vf. b) Usando los valores inicial y final.

Normalmente el valor inicial, **Vi**, es 1 pero puede cambiar dependiendo de las características del problema. Se pueden mezclar los casos a) y b) para construir la estructura de repetición Para.

Cuando el número en el enunciado representa la terminación del programa. Por ejemplo, un juego de computadora que termina cuando el usuario presiona la tecla "5", en este caso el programa se ejecuta muchas veces y sólo termina su ejecución cuando se presiona el número 5.

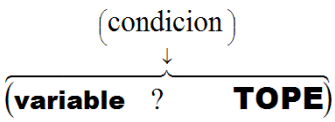
En este caso NO se usa Contador, y el Modificador de Condición depende del uso de dos instrucciones como se muestra a continuación:

Modificador de Condición →
- **Escribir** ("aquí se coloca un texto que pide se ingrese un número para terminar o continuar ")
- **Leer** (***variable***)

Figura 4.24 Instrucciones que conforman el Modificador de Condición cuando el Tope es numérico.

Como se observa en la Figura 4.24, en la instrucción **Escribir** se debe colocar un texto que informe al usuario la necesidad de ingresar un número particular para terminar o continuar con la ejecución del programa. Inmediatamente el usuario ingresa el número, éste es capturado por la instrucción **Leer** y es almacenado en **variable**, ésta es exactamente la misma que aparece en la **condición**. El usuario es quien decide la continuación o no de la ejecución de la estructura de repetición.

La ***condición*** tiene la forma



Es necesario que se satisfagan dos cosas: primero, conocer el operador que reemplazará al símbolo de interrogación y, segundo, que la **variable** tenga un dato almacenado.

Como la estructura de repetición Mientras Haga evalúa la **condición** al comienzo, en **variable** se puede almacenar un dato de dos maneras posibles:

- Inicializándolo con un valor que permita que se ejecute la instrucción de repetición al menos una vez, es decir, debe ser diferente al Tope. Cuando se ejecuta las instrucciones que conforman el Modificador de Condición, se le da el "poder" al usuario para que él decida sobre la continuación de la ejecución de la estructura de repetición.

```
Mientras Haga

Inicialización
Mientras ( variable ? TOPE) haga
  Instrucción 1
  Instrucción 2
       :
       :
  Escribir ("numero a pedir")
  Leer (variable)
  Instrucción N
Fin_Mientras
```

Figura 4.25 Formatos de las estructuras de repetición Mientras Haga, Haga Mientras y Repetir Hasta con las instrucciones que conforman el Modificador de Condición.

- Arrancando por la decisión del Usuario. En este caso se deben colocar las instrucciones del Modificador de Condición también antes de la estructura de repetición. Si el usuario desea que se ejecute la estructura de repetición, él debe ingresar un dato diferente Tope.

```
Mientras Haga

 Escribir ("numero a pedir")
 Leer (variable)
Mientras ( variable <> TOPE) haga
 Instrucción 1
 Instrucción 2
      :
      :
 Escribir ("numero a pedir")
 Leer (variable)
 Instrucción N
Fin_Mientras
```

Figura 4. 27 Formatos de la estructura de repetición Mientras Haga cuando el Tope es numérico, con las instrucciones que conforman el Modificador de Condición, la **condición** construida y el arranque por decisión del Usuario.

Las estructura de repetición Haga Mientras y Repetir Hasta no necesitan Inicialización ya que estas estructuras evalúan la **condición** al final, y cuando esto ocurre **variable** ya contiene un valor que ha sido suministrado por el usuario.

Mientras Haga	**Haga Mientras**	**Repetir Hasta**
variable ← numero diferente a TOPE		
Mientras (*variable* <> **TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
Escribir ("numero a pedir")	**Escribir** ("numero a pedir")	**Escribir** ("numero a pedir")
Leer (*variable*)	**Leer** (*variable*)	**Leer** (*variable*)
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (*variable* <> **TOPE**)	**Hasta** (*variable* = **TOPE**)

Figura 4.26 Formatos de las estructuras de repetición MIENTRAS HAGA, HAGA MIENTRAS y REPETIR HASTA cuando el Tope es numérico

Cuando el Tope es de tipo **carácter**, **cadena** o **booleano**, como en el caso de "Digite SALIR para terminar", el usuario es quien decide la terminación del programa. Esta categoría no usa Contador y la construcción del Modificador de Condición depende del uso de dos instrucciones como se muestra a continuación:

Modificador de Condición →
- **Escribir** ("aquí se coloca un texto que pide se ingrese un carácter, cadena o booleano para terminar o continuar ")
- **Leer** (***variable***)

Figura 4.28 Instrucciones que conforman el Modificador de Condición cuando el Tope es NO Numérico.

El funcionamiento de este Modificador de Condición es similar al que se vio anteriormente y sólo se diferencia de éste por que el usuario debe ingresar un **carácter**, **cadena** o **booleano** para la terminación de la estructura de repetición.

Igual que en el caso de Tope numérico, la estructura de repetición Mientras Haga debe inicializar la variable de la **condición** antes de arrancar el ciclo. Puede ser iniciada por el programador o por el Usuario. Las estructuras de repetición Haga Mientras y Repetir Hasta no necesitan de Inicialización ya que cuando se evalúa la variable en la **condición** ya posee un valor digitado por el Usuario. A continuación se ilustran los dos casos:

Mientras Haga	**Haga Mientras**	**Repetir Hasta**
variable ← carácter, cadena o booleano		
Mientras (***variable*** <> **TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
Escribir ("cadena, carácter o booleano a pedir")	**Escribir** ("cadena, carácter o boolenao a pedir")	**Escribir** ("cadena, carácter o boolenao a pedir")
Leer (***variable***)	**Leer** (***variable***)	**Leer** (***variable***)
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (***variable*** <> **TOPE**)	**Hasta** (***variable*** = **TOPE**)

Figura 4.29 Formatos de las estructura de repetición Mientras Haga, Haga Mientras y Repetir Hasta cuando el Tope es NO numérico, con las instrucciones que conforman el Modificador de condición, la **condición** construida y la Inicialización para Mientras Haga**.**

Mientras Haga	**Haga Mientras**	**Repetir Hasta**
Escribir ("cadena, carácter o booleano a pedir")		
Leer (***variable***)		
Mientras (***variable*** **<> TOPE**) **haga**	**Haga**	**Repita**
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
:	:	:
:	:	:
Escribir ("cadena, carácter o booleano a pedir")	**Escribir** ("cadena, carácter o booleano a pedir")	**Escribir** ("cadena, carácter o booleano a pedir")
Leer (***variable***)	**Leer** (***variable***)	**Leer** (***variable***)
Instrucción N	Instrucción N	Instrucción N
Fin_Mientras	**Mientras** (***variable*** **<> TOPE**)	**Hasta** (***variable*** **= TOPE**)

Figura 4.30 Formatos de las estructura de repetición Mientras Haga, Haga Mientras y Repetir Hasta cuando el Tope es NO numérico, con las instrucciones que conforman el Modificador de condición, la **condición** construida y el Arranque por decisión del Usuario para Mientras Haga**.**

En las figuras 4.29 y 4.30 se puede apreciar que las estructura de repetición Haga Mientras y Repetir Hasta no tienen diferencias entre sí y éstas se diferencian de las figuras 4.27 y 4.26 en que en las primeras solicitan del usuario un **carácter**, **cadena** o **booleano** para la terminación de la estructura de repetición mientras que en las segundas solicitan un número.

A continuación se explicarán los conceptos vistos hasta aquí en el libro, a través de un ejemplo. Escriba un pseudocódigo tal, que dados N números enteros, determine cuántos de ellos son pares y cuántos impares.

Primero se determina si es necesario usar alguna estructura de selección. Para determinar si un número es par, éste se debe comparar por medio de alguna operación empleando el número dos, por ejemplo:

Num **MOD** 2 = 0,

donde Num representa el número ingresado por Usuario. Cuando esta igualdad se cumple, indica que el valor que contiene Num es par. Cuando

Num **MOD** 2 < > 0

se cumple, indica que el valor que contiene Num es impar.

La condición sería: **Si** (Num MOD 2 <> 0) **entonces**

Las operaciones no generan ningún error aritmético sin embargo, hay un caso crítico, la cantidad de números debe ser positiva, la condición para este caso es:

Si (N > 0) **entonces**

donde N representa la cantidad de números.

Ahora se revisará el uso de estructuras de repetición. El enunciado dice ".. dados N números enteros", esto significa primero, que el usuario es quien ingresará la cantidad de datos y el valor de los mismos, y segundo, que a cada dato ingresado se le debe determinar si es par o impar y este análisis se debe hacer N veces, por tanto, se concluye que es necesario el uso de una estructura de repetición.

Primero se identifica el Tope, según el enunciado se ingresarán N números enteros, con base en esto se concluye que el Tope es numérico. A cada número ingresado se le debe determinar si es par o impar esto indica que el proceso de comparación se debe realizar N veces por tanto, en este ejemplo N es el Tope.

Según lo anterior se construye el *Contador*

$$\mathbf{X \leftarrow X + 1}$$

La **condición** y la Inicialización queda:

X ← 0
Mientras (X < N) **haga**

Como en la **condición** existe la variable N, es necesario capturar este valor antes de que se ejecute ésta y para ello se usa las siguientes instrucciones:

Escribir ("Ingrese la cantidad de números")
Leer (N)

Dentro de la estructura de repetición Mientras Haga se debe capturar un número por vez y luego analizarlo para determinar si es par o impar.

Ahora se construye el Algoritmo, y una vez construido éste, se elabora el pseudocódigo y el Diagrama de Flujo respectivo.

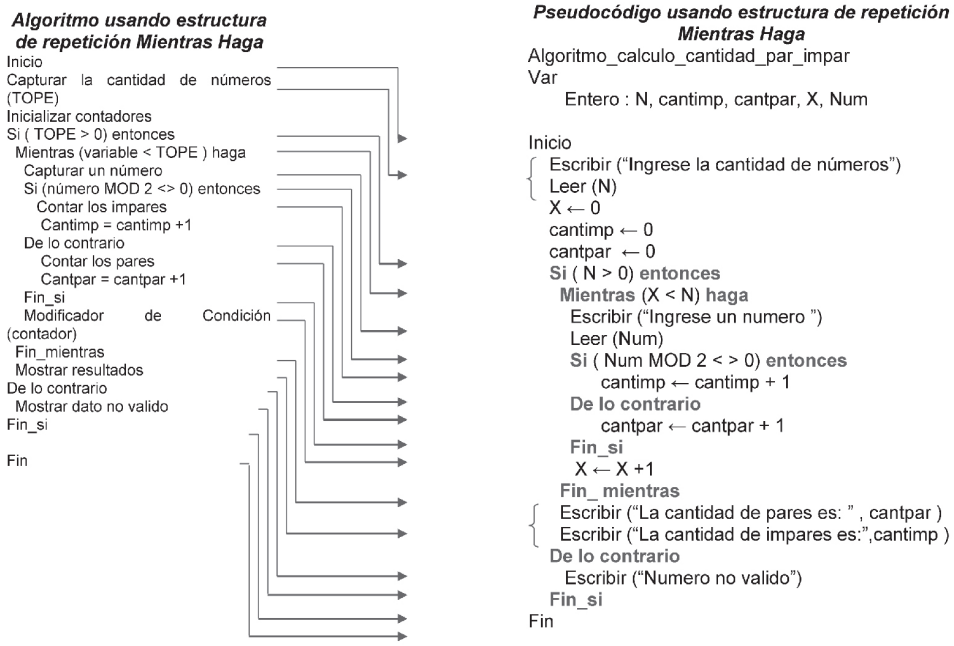


Figura 4.31 Relación entre el algoritmo y el pseudocódigo del ejemplo.

Pseudocódigo usando estructura de repetición

Mientras Haga

```
Algoritmo_calculo_cantidad_par_impar
Var
    Entero : N, cantimp, cantpar, X, Num
Inicio
  Escribir ("Ingrese la cantidad de números")
  Leer (N)
  X ← 0
  cantimp ← 0
  cantpar ← 0
  Si ( N > 0) entonces
   Mientras (X < N) haga
    Escribir ("Ingrese un numero ")
    Leer (Num)
    Si ( Num MOD 2 < > 0) entonces
        cantimp ← cantimp + 1
    De lo contrario
        cantpar ← cantpar + 1
    Fin_si
    X ← X +1
   Fin_ mientras
   Escribir ("La cantidad de pares es: " , cantpar )
   Escribir ("La cantidad de impares es:",cantimp )
  De lo contrario
    Escribir ("Numero no valido")
  Fin_si
Fin
```

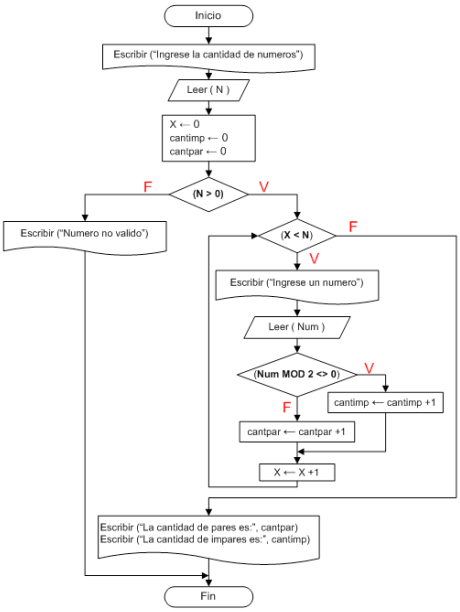


Figura 4.32 Relación entre el diagrama de flujo y pseudocódigo del ejemplo.

El mismo ejemplo se puede resolver utilizando las otras Estructuras de Repetición.

Pseudocódigo usando Haga Mientras

```
Algoritmo_calculo_cantidad_par_impar
Var
    Entero : N, cantimp, cantpar, X, Num
Inicio
Escribir ("Ingrese la cantidad de números")
Leer (N)
X ← 0
cantimp ← 0
cantpar ← 0
Si ( N > 0) entonces
  Haga
    Escribir ("Ingrese un numero ")
    Leer (Num)
    Si ( Num MOD 2 < > 0) entonces
      cantimp ← cantimp + 1
    De lo contrario
      cantpar ← cantpar + 1
    Fin_si
    X ← X +1
  Mientras (X < N)
Escribir ("La cantidad de pares es:" , cantpar )
Escribir("La cantidad de impares es:",cantimp )
De lo contrario
  Escribir ("Numero no valido")
  Fin_si
Fin
```

Pseudocódigo usando Repetir Hasta

```
Algoritmo_calculo_cantidad_par_impar
Var
    Entero : N, cantimp, cantpar, X, Num
Inicio
Escribir ("Ingrese la cantidad de números")
Leer (N)
X ← 0
cantimp ← 0
cantpar ← 0
Si ( N > 0) entonces
  Repita
    Escribir ("Ingrese un numero ")
    Leer (Num)
    Si ( Num MOD 2 < > 0) entonces
      cantimp ← cantimp + 1
    De lo contrario
      cantpar ← cantpar + 1
    Fin_si
    X ← X +1
  Hasta (X >= N)
Escribir ("La cantidad de pares es:" , cantpar )
Escribir("La cantidad de impares es:",cantimp )
De lo contrario
  Escribir ("Numero no valido")
  Fin_si
Fin
```

Pseudocódigo usando Para

```
Algoritmo_calculo_cantidad_par_impar
Var
    Entero : N, cantimp, cantpar, X, Num
Inicio
Escribir ("Ingrese la cantidad de números")
Leer (N)
cantimp ← 0
cantpar ← 0
Si ( N > 0) entonces
  Para X← 1 hasta N INC 1 haga
    Escribir ("Ingrese un numero ")
    Leer (Num)
    Si ( Num MOD 2 < > 0) entonces
      cantimp ← cantimp + 1
    De lo contrario
      cantpar ← cantpar + 1
    Fin_si
  Fin_Para
  Escribir ("La cantidad de pares es:" , cantpar )
  Escribir("La cantidad de impares es:",cantimp )
De lo contrario
  Escribir ("Numero no valido")
Fin_si
Fin
```

Figura 4.33 Solución del ejemplo utilizando estructura de repetición Haga Mientras, Repetir Hasta y Para.

4.6 CASOS ESPECIALES

Siempre que se construya una estructura de repetición es importante tener en cuenta el control de la misma, recuérdese que para ello se debe incluir el Modificador de Condición dentro del cuerpo de la estructura sin embargo, en el desarrollo de un pseudocódigo o programa en general, el programador puede construir una estructura de repetición que se ejecute infinitamente o por el contrario, que nunca se ejecute, el primer caso se conoce como Ciclo Infinito y el segundo como Ciclo Nulo.

Normalmente un Ciclo Infinito puede ocurrir por alguno de los siguientes errores:

a. Cuando el programador no tuvo en cuenta en la construcción del pseudocódigo la creación del Modificador de Condición perdiendo el control total del programa en la ejecución de la estructura de repetición, el siguiente trozo de pseudocódigo muestra esto:

```
X ← 1
Mientras (X < 5) haga
  Escribir ("Hola")
Fin_mientras
```

```
X ← 1
Haga
  Escribir ("Hola")
Mientras (X < 5)
```

```
X ← 1
Repita
  Escribir ("Hola")
Hasta (X >= 5)
```

```
X ← 1
Mientras (X<>5) haga
  Escribir ("Hola")
Fin_mientras
```

```
X ← 1
Haga
  Escribir ("Hola")
Mientras (X <> 5)
```

```
X ← 1
Repita
  Escribir ("Hola")
Hasta (X=5)
```

```
X ← "n"
Mientras (X<>"S" O X<>"s") haga
  Escribir ("Hola")
Fin_mientras
```

```
X ← "n"
Haga
  Escribir ("Hola")
Mientras (X<>"S" O X<>"s")
```

```
X ← "n"
Repita
  Escribir ("Hola")
Hasta (X = "S" O X = "s")
```

Figura 4.34 Ejemplos de Ciclo Infinito por ausencia del Modificador de Condición para las diferentes estructuras de repetición

b. Cuando el programador no construye correctamente el Modificador de Condición, es decir, no tuvo en cuenta el operador aritmético correcto (+ o –) para el Tope numérico o, no incluyó en la instrucción Leer la variable que se encuentra en la condición para el Tope.

```
X ← 6                          X ← 5                          X ← 1
Mientras (X > 5) haga          Haga                           Repita
 Escribir ("Hola")              Escribir ("Hola")              Escribir ("Hola")
 X ← X + 1                      X ← X – 1                      X ← X – 1
Fin_mientras                   Mientras (X < 5)               Hasta (X >= 5)
```

a)

```
X ← 1                          X ← 1                          X ← 1
Mientras (X <> 5) haga         Haga                           Repita
 Escribir ("Hola")              Escribir ("Hola")              Escribir ("Hola")
 Escribir ("Ingrese 5 para      Escribir ("Ingrese 5 para      Escribir("Ingrese 5 para
terminar")                     terminar")                     terminar")
 Leer (Num)                     Leer (Num)                     Leer (Num)
Fin_mientras                   Mientras (X <> 5)              Hasta (X = 5)
```

b)

```
X ← "n"                        X ← "n"                        X ← "n"
Mientras (X<>"S" O             Haga                           Repita
X<>"s") haga                    Escribir ("Hola")              Escribir ("Hola")
 Escribir ("Hola")              Escribir ("Ingrese S para      Escribir("Ingrese S para ter-
 Escribir ("Ingrese S para      terminar")                     minar")
terminar")                      Leer (Res)                     Leer (Res)
 Leer (Res)                    Mientras (X<>"S" O             Hasta (X = "S" O X = "s")
Fin_mientras                   X<>"s")
```

c)

Figura 4.35 Ejemplo de Ciclo Infinito cuando no se construye correctamente el Modificador de Condición para las diferentes estructuras de repetición.

c. Cuando el programador no incluye el Modificador de Condición dentro de la estructura de repetición pero si en el programa.

```
X ← 10
Mientras (X > 5) haga
  Escribir ("Hola")
Fin_mientras
X ← X – 1
```

```
X ← 1
Haga
  Escribir ("Hola")
Mientras (X < 5)
  X ← X + 1
```

```
X ← 1
Repita
  Escribir ("Hola")
Hasta (X >= 5)
X ← X + 1
```

a)

```
X ← 1
Mientras (X <> 5) haga
 Escribir ("Hola")
Fin_mientras
 Escribir ("Ingrese 5 para terminar")
 Leer (X)
```

```
X ← 1
Haga
 Escribir ("Hola")
Mientras (X <> 5)
 Escribir ("Ingrese 5 para terminar")
 Leer (X)
```

```
X ← 1
Repita
 Escribir ("Hola")
Hasta (X = 5)
  Escribir("Ingrese 5 para terminar")
 Leer (X)
```

b)

```
X ← "n"
Mientras (X<>"S" O X<>"s") haga
 Escribir ("Hola")
Fin_mientras
 Escribir ("Ingrese S para terminar")
 Leer (X)
```

```
X ← "n"
Haga
 Escribir ("Hola")
Mientras (X<>"S" O X<>"s")
  Escribir ("Ingrese S para terminar")
 Leer (X)
```

```
X ← "n"
Repita
 Escribir ("Hola")
Hasta (X = "S" O X = "s")
 Escribir("Ingrese S para terminar")
 Leer (X)
```

c)

Figura 4.39 Ejemplo de Ciclo Infinito cuando el Modificador de Condición está por fuera de la estructura de repetición**.**

Un Ciclo Nulo ocurre en aquellas estructuras donde se pregunta antes de ingresar a la estructura de repetición y sucede cuando no se cumple la condición, provocando que la estructura nunca se ejecute. En estructura de repetición Haga Mientras y Repetir Hasta por la forma en que se construyen no se puede dar el Ciclo Nulo ya que en estas se ingresa primero y se pregunta después.

Otra manera de Ciclo Nulo ocurre cuando se utiliza el arranque por decisión del Usuario, y éste digita un valor que hace que no se ingrese a la estructura de repetición.

```
X ← 10
Mientras (X < 5) haga
  Escribir ("Hola")
  X ← X + 1
Fin_mientras
```
a)

```
Escribir ("Ingrese 5 para terminar")
Leer (X)
Mientras (X <> 5) haga
  Escribir ("Hola")
  Escribir ("Ingrese 5 para terminar")
  Leer (X)
Fin_mientras
```
b)

```
Escribir ("Ingrese S para terminar")
Leer (X)
Mientras (X <> "S" O X <> "s" ) haga
  Escribir ("Hola")
  Escribir ("Ingrese S para terminar")
  Leer (X)
Fin_mientras
```
c)

Figura 4.40 Ejemplos de Ciclo Nulo

4.7 EJERCICIOS

4.7.1 Ejercicios con Respuesta

1. Construya un pseudocódigo tal que lea un número entero N, muestre la cantidad de términos y el resultado de la siguiente serie:

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots\pm\frac{1}{N}$$

2. Construya un pseudocódigo tal que encuentre y muestre todos los enteros positivos, comenzando desde el cero, que satisfacen la siguiente expresión:

$$P^3+Q^4-2*P^2 \quad \langle \quad 680$$

4.7.2 Ejercicios sin Respuesta

3. Construya un pseudocódigo que eleve un número **X** a una potencia dada, usando sucesivas multiplicaciones.

4. Construya un pseudocódigo que calcule el factorial de un número.

5. Construya un pseudocódigo que dado un número N, indique si se trata de un número primo o o no.

6. Dada una secuencia de N números, construya un pseudocódigo que muestre por pantalla el resultado de la Sumatoria de los números.

7. Usted acaba de compra el Chimbis, dentro del cual se venden 10 variedades de productos. Decide por estrategia de mercadeo colocar en la entrada los productos con precios más bajos, así que construye un pseudocódigo que le ayude a ordenar ascendentemente por precio todos los productos que hay en bodega y los muestre por pantalla.

4.7.3 Respuesta a los Ejercicios

1. Construya un pseudocódigo tal que lea un número entero N, muestre la cantidad de términos y el resultado de la siguiente serie:

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots\pm\frac{1}{N}$$

Note de la serie anterior que N se encuentra en el denominador y que el mismo se incrementa de uno en uno además, los números pares están precedidos por el signo menos y los impares con el más. N representa la cantidad de números y ésta es ingresada por el usuario. Nótese además que si el usuario ingresa el número cero (0), indicando cero términos de la serie, el pseudocódigo en su ejecución no debe mostrar ni valor alguno ni termino de la serie. El primer término (término 1) es el número 1.

El incremento de uno en uno ofrece la pista para la construcción de la serie y, para llevar a cabo esto, se utiliza la instrucción conocida como el *contador*, es decir:

$$X \leftarrow X + 1$$

Esta instrucción permitirá incrementar de uno en uno la variable *X* que se utilizará para la construcción de la serie. Recuérdese que para ejecutar esta instrucción X debe estar declarada y contener un dato por lo cual se debe inicializar esta variable. El *contador* cumple tres funciones en este ejemplo, primero ofrece el número que se ubicará en el denominador, segundo, informa sobre los pares para así incluir el signo respectivo en la expresión y, por último, lleva la cuenta de la cantidad de veces que se debe realizar o ejecutar la Estructura de Repetición.

Como los pares llevan el signo menos se puede incorporar este signo con base en descubrir cuando la variable X es par, para ello se hace uso de la siguiente expresión:

$$X \quad MOD \quad 2 = 0$$

La anterior ecuación indica que se debe llevar a cabo una comparación para poder descubrir cuando el contenido de X es par, es decir, si el resultado de X **MOD** 2 es igual a 0 el número es par, nótese que ser par implica que el resultado de la comparación debe ser verdadero. X **MOD** 2 = 0 es la condición de una Estructura de Selección Compuesta.

Si el usuario ingresa el número 3, el pseudocódigo construido debe mostrar la serie de la siguiente manera:

$$1 - \frac{1}{2} + \frac{1}{3}$$

Y a su vez debe visualizar el resultado de esa suma, es decir, 0.8333333

Para obtener el valor anterior, el programa construido debe restar ½ de 1 y el resultado debe ser sumado a **1/3**. En el pseudocódigo existe una instrucción que permite conservar resultados para que éstos puedan ser usados nueva-

mente con otros valores y así obtener el valor final, la misma es conocida con el nombre de *acumulador*, y se construye de la siguiente manera:

$$sum \leftarrow sum + X$$

La variable *sum* contendrá el resultado de la ejecución de la instrucción anterior, conservando así el valor que después se sumara con el contenido de X generándose un nuevo dato para *sum*. Recuérdese que en el pseudocódigo ejecutar una instrucción de asignación implica que primero se debe resolver el lado derecho de la flecha y el resultado se almacena en el lado izquierdo. Con base en esto, antes de ejecutarse el *acumulador* primero se debe asignar un valor a la variable que se encuentra al lado izquierdo de la flecha, si esto no se hace, *sum* no tendría dato alguno produciéndose un error en la ejecución de esta instrucción. El valor inicial de *sum* debe ser cero para que el mismo no afecte el resultado de la operación suma.

Nótese que el *acumulador* es muy similar al *contador*, aquella se diferencia de ésta en el uso de dos variables mientras que en el *contador* se utiliza un número como segundo operando.

Al igual que en el *contador*, en la construcción del *acumulador* se debe tener en cuenta el uso de una misma variable la cual se debe encontrar a ambos lados de la flecha.

En la solución del ejemplo 4.2 es necesario el uso de dos acumuladores, el primero para los signos positivos y, el segundo, para los negativos, quedando:

$$sum \leftarrow sum + \frac{1}{X}$$

$$sum \leftarrow sum - \frac{1}{X}$$

Si X **MOD** 2 =0 se utiliza el *acumulador* con el signo menos, de lo contrario se usa la otra expresión.

Para resolver el ejemplo primero se aplican las herramientas de la sección 4.1.5 para determinar si es necesario usar alguna **ES**, por tanto se tiene:

Herramienta #

1. No se cumple

2. Para determinar si un numero es par, éste se debe comparar por medio de la siguiente expresión

X **MOD** 2 = 0

Cuando esta igualdad se cumple, indica que el valor que contiene X es par, de lo contrario, el contenido de X es impar.
Nótese como con el uso de comparaciones (utilización del operador =) se puede resolver el problema por tanto, esta herramienta se cumple y se necesita usar una **ES**, la condición es

Si (X **MOD** 2 = 0) **entonces**

Si se cumple esta condición se debe ejecutar el acumulador con el signo menos. Para esta herramienta se puede utilizar **ESS** o **ESC.**

3. Nótese que la serie utiliza signos positivos para los pares y negativos para los impares; conocer si es par o impar depende de una única expresión, X **MOD** 2, un resultado diferente a cero indica que el numero contenido en X es impar, de lo contrario es par. Se puede utilizar una **ESM** para resolver este ejercicio quedando la misma de la siguiente manera:

Si (X **MOD** 2) **igual**

0 **:** operaciones para par
1 **:** operaciones para impar

Nótese que existen dos maneras para resolver el ejemplo 4.2, el programador podrá escoger cualquiera de ellas, en este texto se llevará a cabo la solución utilizando primero **ESS** o **ESC** y después **ESM**.

4. Operaciones Aritméticas

Las operaciones para resolver el ejercicio son:

$$sum \leftarrow sum + \frac{1}{X}$$

$$sum \leftarrow sum - \frac{1}{X}$$

Nótese que estas ecuaciones generan error aritmético cuando X contiene el valor cero, para resolver esta inquietud se puede escoger uno de dos caminos, primero, se inicializa a X en cero pero se coloca el *contador* antes de la ejecución de alguna de las instrucciones del *acumulador* o, segundo, se da un valor inicial a X de 1 y se coloca al *contador* después de la ejecución del *acumulador*.

4. Casos Críticos

Se cumple por tanto se requiere una **ES**. La cantidad de números debe ser positiva, la condición para este caso es:

Si (N > 0) **entonces**

Donde N representa la cantidad de números, información ingresada por el usuario. Esta estructura debe colocarse inmediatamente después de capturar el valor de N.

Ahora se aplican las herramientas de la sección 4.1.5 para determinar si es necesario usar alguna **ER**, por tanto se tiene:

Herramienta #

1. Nótese del enunciado del ejemplo 4.2 que según la cantidad ingresada por el usuario, se deben realizar en igual cantidad las sumas y restas, es decir, si el usuario ingresa el número 4, primero el pseudocódigo en su ejecución debe mostrar

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}$$

Pero después debe visualizar el resultado de esta expresión (sumas y restas) y la misma depende de la cantidad de elementos que el usuario desea por lo que, el calculo a realizar está en función de N, por tanto, se concluye que es necesario el uso de una **ER**.

2. Sobre la ***condición***

Primero se identifica el **TOPE**, según el enunciado se ingresará un número que representa la cantidad de términos a visualizar y sobre éstos se realizará un cálculo. La cantidad de elementos a sumar o restar depende directamente de la cantidad de términos a visualizar que a su vez es el valor ingresado por el usuario y almacenado en la variable N, entonces N representa la cantidad de veces que se debe realizar la operación, con base en esto se concluye que el **TOPE** es numérico y **caso 1**. En este ejemplo N es el **TOPE**. Según lo anterior se construye el *contador*

$$\mathbf{X \leftarrow X + 1}$$

Para un **TOPE** numérico **caso 1** se puede utilizar cualquier Estructura de Repetición. Según la figura 4.22 para la **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**, la ***condición*** y la *Inicialización* queda

X ← 0

Mientras (X < N) **haga**

Como en la ***condición*** existe la variable N, es necesario capturar este valor antes de que se ejecute ésta y para ello se usa las siguientes instrucciones

Escribir ("Ingrese la cantidad de términos de la serie a visualizar y obtener
su resultado")
Leer (N)

Después de capturar a N, se coloca la **condición** del caso crítico.
Dentro de la **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA** se debe construir la serie y después obtener el resultado de la misma.

Ahora se aplican las herramientas de la sección 4.1.6 y 4.1.6 para construir el Algoritmo, una vez construido éste, se elaborará el pseudocódigo. Primero se resolverá el ejemplo utilizando **ESS**, luego usando **ESC** y por ultimo **ESM**, se elaborará el pseudocódigo con la Estructura de Repetición **Para**.

Solución al ejemplo 4.2 usando **ESS** y **ERP**

Herr. ***ALGORITMO usando ESS***

1. Inicio

2. Capturar la cantidad de términos de la serie a visualizar (TOPE).

3. Inicializar acumulador
Si (TOPE>0) entonces
Para X = 1 hasta TOPE haga
Si (X **MOD** 2 = 0) entonces
Crear serie con negativos
sum = sum – 1/X
visualizar termino
Fin_si
Si (X **MOD** 2 <> 0) entonces
Crear serie con positivos
sum = sum + 1/X
visualizar termino
Fin_si
Fin_para
Mostrar el resultado de la serie
Fin_si
Si(TOPE <= 0) entonces
Mostrar dato no valido
Fin_si

4. Fin

Pseudocódigo usando ESS y ERP

```
Algoritmo_muestra_terminos_y_resultado_serie
var
    Decimal : sum
    Entero: X,N
Inicio
Escribir(" Ingrese la cantidad de términos de la serie a
                visualizar y obtener su resultado ")
Leer (N)
sum ← 0
Si (N >0) entonces
   Escribir ("La serie es:")
   Para X← 1 hasta N haga
     Si (X MOD 2 = 0) entonces
        sum ← sum – 1/X
        Escribir ( "– 1/", X)
     Fin_si
     Si (X MOD 2 <> 0) entonces
        sum ← sum + 1/X
        Si (X =1) entonces
           Escribir ( "1")
        Fin_si
        Si (X <>1) entonces
           Escribir ( "+ 1/", X)
        Fin_si
     Fin_si
   Fin_para
   Escribir ("El resultado de la serie es", sum)
Fin_si
Si (N<=0) entonces
   Escribir ("Dato no valido")
Fin_si
Fin
```

Nótese que en la construcción del pseudocódigo, en la visualización de los términos con signo positivo se hizo uso de dos Estructuras de Selección, esto es con el fin de mostrar el primer elemento de la serie, el número 1, sin que éste aparezca junto a los operadores + y /. La segunda instrucción Escribir se utiliza para que aparezca una sola vez la frase *La serie es:*, si esta instrucción se ubica dentro de la Estructura de Repetición, la misma aparecerá la cantidad de veces que representa el **TOPE**.

A continuación se resuelve el ejemplo 4.2 usando **ESC**.

Solución al ejemplo 4.2 usando **ESM** y **ERP**

Herr. ***ALGORITMO usando ESM***

1. Inicio

2. Capturar la cantidad de términos de la serie a visualizar (TOPE).

3. Si (TOPE>0) entonces
  Para X = 1 hasta TOPE haga
    Si (X **MOD** 2) igual
      0: Crear serie con negativos
        sum ← sum – 1/X
        visualizar termino
      1: Crear serie con positivos
        sum ← sum + 1/X
        visualizar termino
    Fin_si
  Fin_para
  Mostrar el resultado de la serie
  De lo contrario
  Mostrar dato no valido
  Fin_si

4. Fin

Pseudocódigo usando ESM y ERP

```
Algoritmo_muestra_terminos_y_resultado_serie
var
    Decimal : sum
    Entero: X,N
Inicio
Escribir(" Ingrese la cantidad de términos de la serie a
            visualizar y obtener su resultado ")
Leer (N)
sum ← 0
Si (N >0) entonces
   Escribir ("La serie es:")
   Para X← 1 hasta N haga
     Si (X MOD 2) igual

        0: sum ← sum – 1/X
           Escribir ( "– 1/", X)
        1: sum ← sum + 1/X
           Si (X ) igual
                1: Escribir ( "1")
           De lo contrario: Escribir ( "+ 1/", X)
           Fin_si
     Fin_si
   Fin_para
   Escribir ("El resultado de la serie es", sum)
De lo contrario
   Escribir ("Dato no valido")
Fin_si
Fin
```

Los operadores +, – y / que se encuentran dentro de las instrucciones Escribir sirven para se visualicen cuando éstas se ejecutan esto con el fin de construir la serie pedida.

A continuación se resuelve el ejemplo 4.2 usando **ESM**.

Solución al ejemplo 4.2 usando **ESM** y **ERP**

Herr. ***ALGORITMO usando ESM***

1. Inicio
2. Capturar la cantidad de términos de la serie a visualizar (TOPE).
3. Si (TOPE>0) entonces
   Para X = 1 hasta TOPE haga
   Si (X **MOD** 2) igual
   0: Crear serie con negativos
   sum ← sum – 1/X
   visualizar termino
   1: Crear serie con positivos
   sum ← sum + 1/X
   visualizar termino
   Fin_si
   Fin_para
   Mostrar el resultado de la serie
   De lo contrario
   Mostrar dato no valido
   Fin_si
4. Fin

Pseudocódigo usando ESM y ERP

```
Algoritmo_muestra_terminos_y_resultado_serie
var
    Decimal : sum
    Entero: X,N
Inicio
Escribir(" Ingrese la cantidad de términos de la serie a
            visualizar y obtener su resultado ")
Leer (N)

sum ← 0
Si (N >0) entonces
   Escribir ("La serie es:")
   Para X← 1 hasta N haga
     Si (X MOD 2) igual
        0: sum ← sum – 1/X
           Escribir ( "– 1/", X)
        1: sum ← sum + 1/X
           Si (X ) igual
                1: Escribir ( "1")
           De lo contrario: Escribir ( "+ 1/", X)
           Fin_si
     Fin_si
   Fin_para
   Escribir ("El resultado de la serie es", sum)
De lo contrario
   Escribir ("Dato no valido")
Fin_si
Fin
```

Nótese en todas las soluciones al ejemplo 4.2 que únicamente se visualiza el número 1 cuando X contiene el valor de 1 y además, el ingreso a esta zona del programa ocurre cuando X **MOD** 2 no es igual a cero, recuérdese que se evalúa al contenido de X para mostrar el 1 o el termino positivo de la serie mas no el resultado de la operación Modulo.

En la última solución del ejemplo 4.2 se puede apreciar el uso de dos **ESM**, la primera para permitir la construcción de la serie para los términos pares e impares y la segunda, para visualizar los elementos positivos de la serie, en esta última se evalúa el contenido de X que determinará cual de las dos instrucciones Escribir se debe ejecutar. Nótese como se puede utilizar una **ESM** con solo dos caminos a escoger sin embargo, las ventajas de esta estructura se desaprovechan, es decir, para dos caminos en donde se puede escoger uno de ellos es más recomendable el uso de una **ESC**, aunque no tener en cuenta esto no significa que se genere error en la ejecución del pseudocódigo. A continuación se muestran los diagramas de flujo para cada una de las soluciones presentadas al ejemplo 4.2.

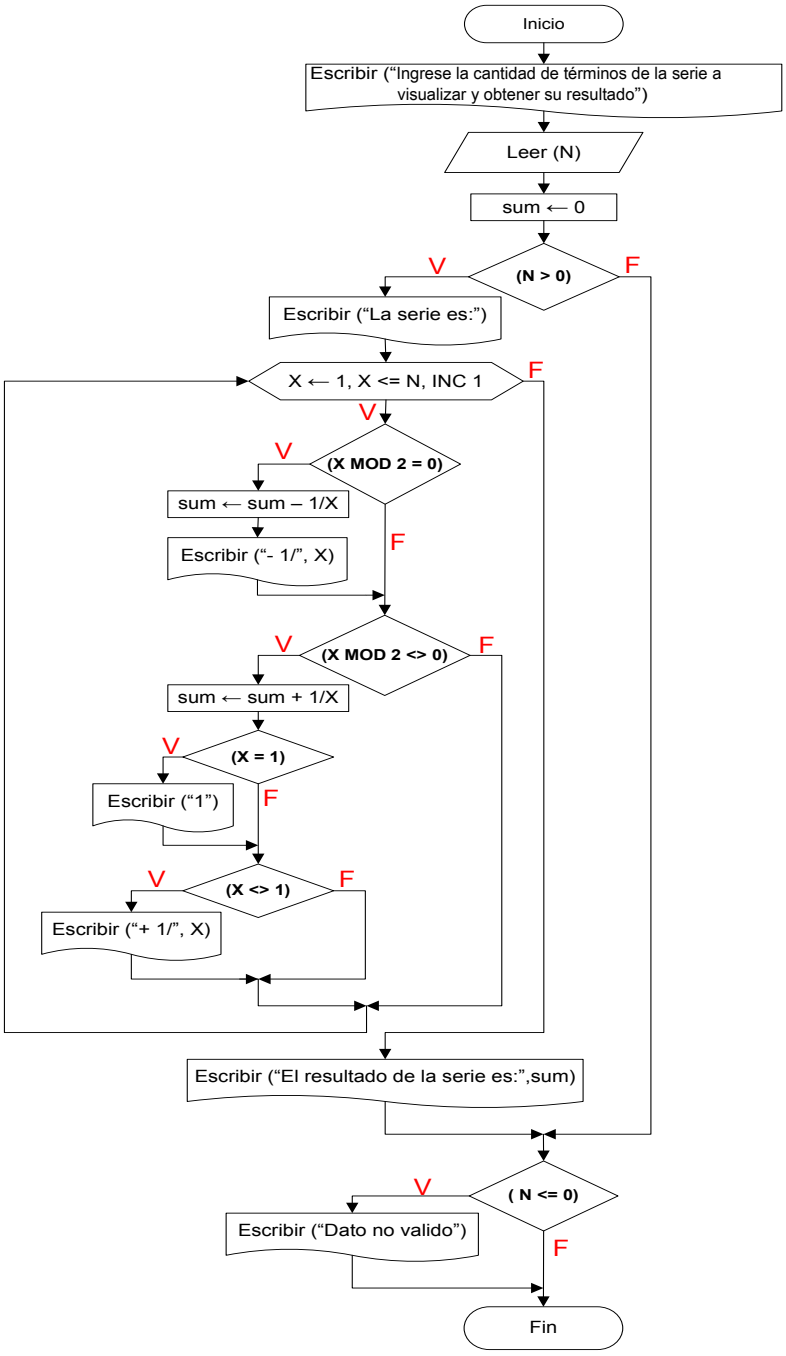


Figura 4.33 Solución del ejemplo 4.2 usando **ESS** y **ERP**.

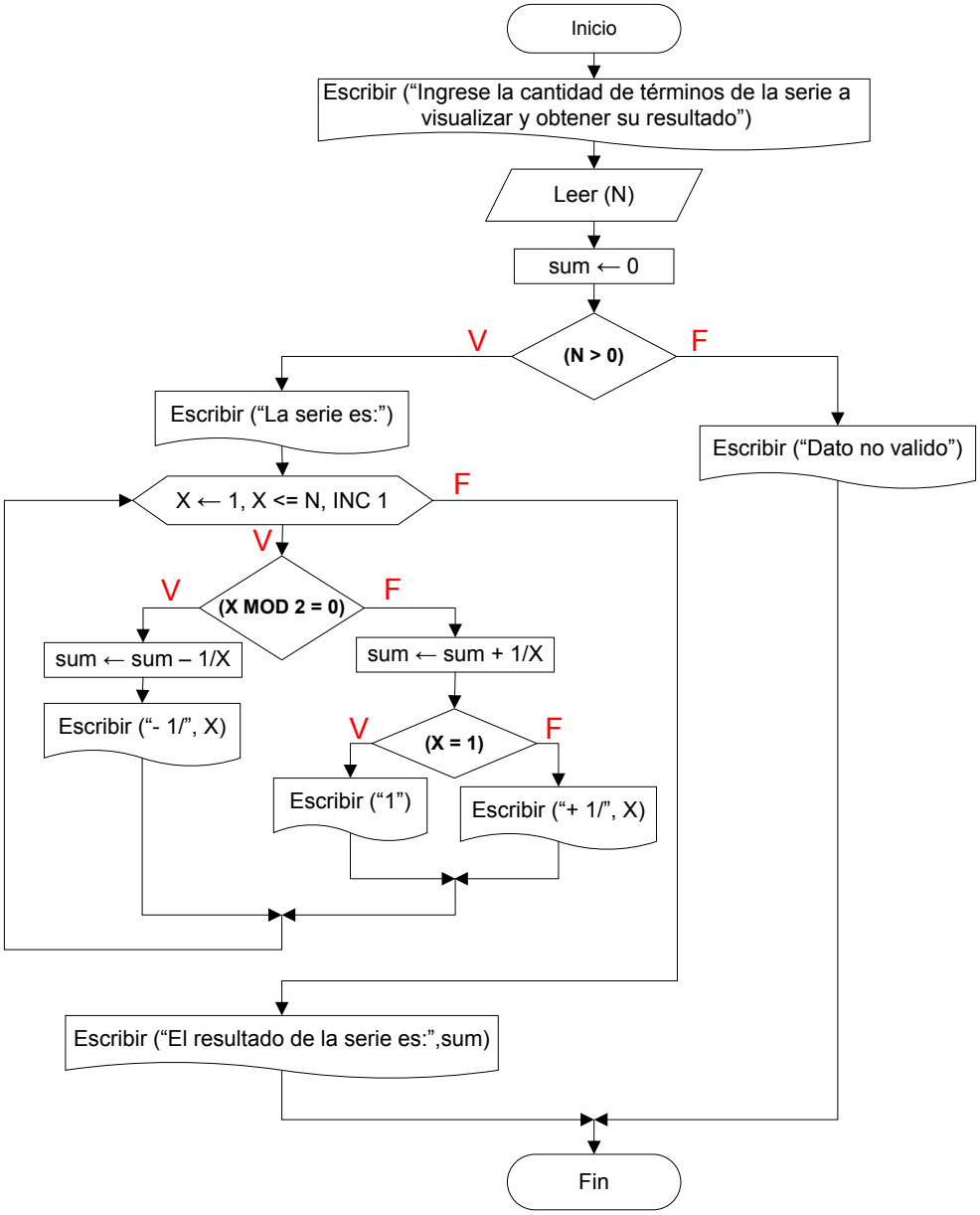


Figura 4.34 Solución del ejemplo 4.2 usando **ESC** y **ERP**.

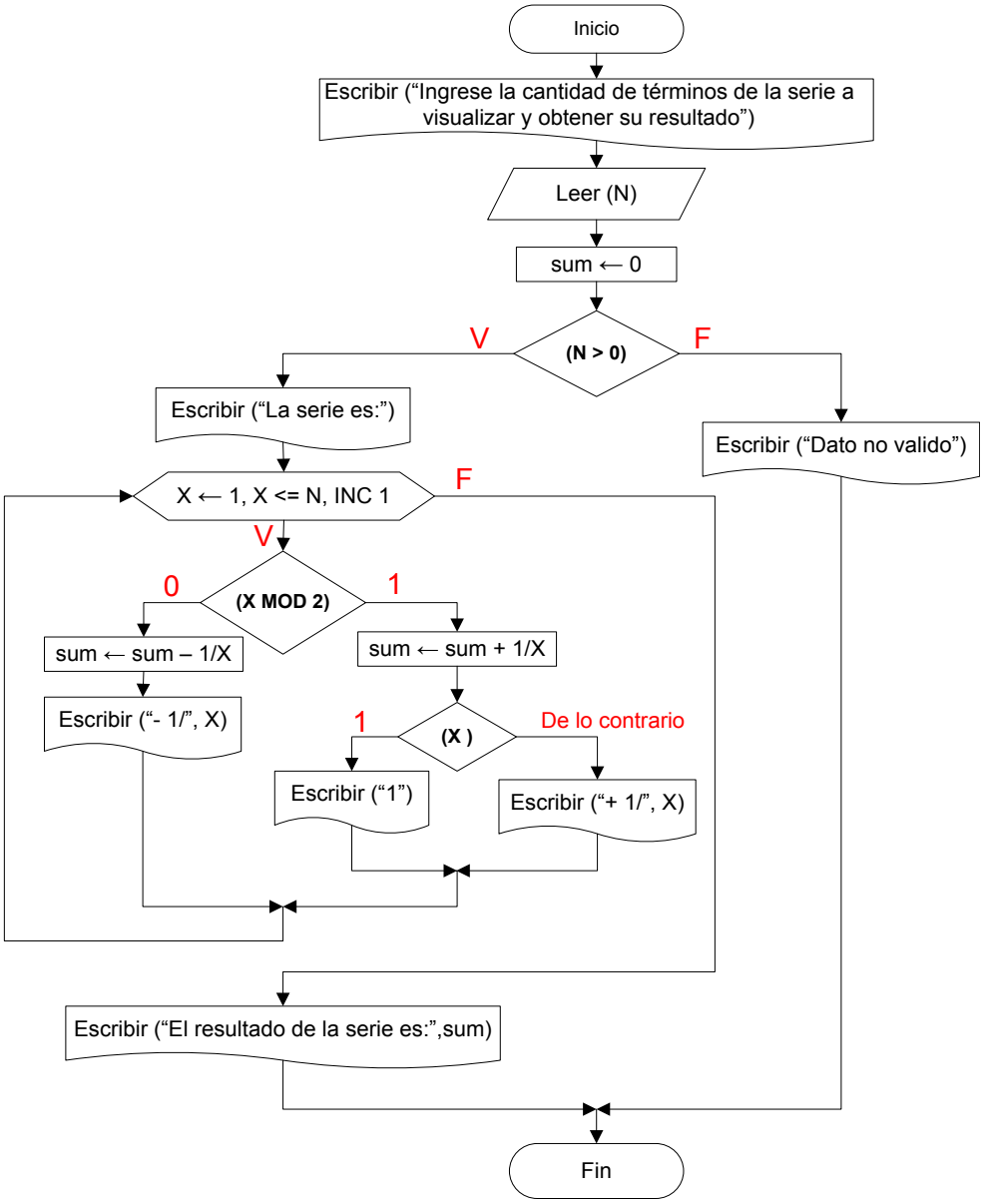


Figura 4.35 Solución del ejemplo 4.2 usando **ESM** y **ERP**.

Note que las diferencias visuales entre las figuras 4.34 y 4.35 no son muy perceptibles, tan solo se ha cambiado las letras V y F por 0 y 1 o por 1 y De lo contrario así como también se ha creado el selector, pero el comportamiento es similar.

2. Construya un pseudocódigo tal que encuentre y muestre todos los enteros positivos, comenzando desde el cero, que satisfacen la siguiente expresión:

$$P^3 + Q^4 - 2 * P^2 \quad \langle \quad 680$$

La solución busca que se reemplacen tanto P como Q por números enteros de forma que al evaluarse la inecuación anterior la misma se cumpla y, si esto ocurre se deben mostrar estos valores. Nótese que pueden surgir muchas posibles combinaciones de forma que se satisfaga la anterior desigualdad, por ejemplo, P=Q=0 produce el mismo resultado que P=0 y Q=1, es decir, satisfacen la inecuación anterior.

Para cada valor de P y Q se debe evaluar la expresión indicando esto que el proceso de comprobación de la desigualdad se lleva a cabo muchas veces y el mismo termina cuando la inecuación no se satisface.

Nótese que para P=0 existen varios valores de Q que satisfacen la inecuación sugiriendo esto que la mejor opción de resolver el ejemplo 4.3 es darle un valor a P y a Q de arranque y crear las líneas de pseudocódigo necesarias para que cambie Q, dejando a P fijo temporalmente, hasta que la desigualdad no se satisfaga para después incrementar el valor de P y repetir el proceso de nuevo. Obsérvese que el punto de partida y de parada de la solución al ejemplo es la inecuación anterior.

Ahora se aplican las herramientas de la sección 4.1.5 para determinar si es necesario usar alguna **ES**, entonces:

Herramienta #	
1.	No se cumple
2.	En efecto se debe realizar una comparación, es decir, resolver el ejemplo implica la comprobación de la desigualdad por lo que se podría pensar en el uso de una **ES** para llevar a cabo esto, sin embargo, la inecuación se debe evaluar muchas veces y en este tipo de ejemplo en particular es más recomendable no utilizar **ES** ya que no se conoce el valor exacto de veces que se debe comprobar la inecuación.
3.	No se cumple pues el ejemplo no ofrece varios caminos a escoger.
4.	Operaciones Aritméticas Las operación para resolver el ejercicio es: $P^3 + Q^4 - 2 * P^2 \quad \langle \quad 680$ La misma no genera error aritmético para cualquier valor de P y Q, por tanto esta herramienta no se cumple.
4.	Casos Críticos Como es el programa construido que debe generar los resultados para P y Q, éstos deben ser positivos y es el programador quien crea estos valores por lo que esta herramienta no se cumple.

Ahora se aplican las herramientas de la sección 4.1.5 para determinar si es necesario usar alguna **ER**, entonces:

Herramienta #	
1.	Según el ejemplo 4.3 se debe evaluar varias veces la desigualdad, por realizar esta labor más de una vez entonces es necesario el uso de una **ER**.

2. Sobre la ***condición***

Primero se identifica el **TOPE**, según el enunciado quién determina la cantidad de veces que se debe ejecutar una **ER** es la condición generando verdadero o falso al ser evaluada, como se debe comprobar la inecuación cada vez que existe un nuevo valor para P o Q, entonces la mejor manera de detener la ejecución o no de la **ER** es colocando como condición la desigualdad, pero, esta expresión está conformada por dos variables, P y Q, las cuales deben contener un valor antes de ser evaluada por lo que inicialmente a P y Q se les debe dar un valor de arranque, para este caso es 0.

Nótese que se debe ejecutar la **ER** un numero finito de veces determinado por el cumplimiento de la desigualdad, esta cantidad es desconocida y solo podrá ser establecida en el momento de ejecución del pseudocódigo, sin embargo, el lector como ejercicio puede descubrir hasta que valor máximo para P y Q se cumple la inecuación permitiendo esto la construcción de una condición para las **ER**, pero esto implica que el programa construido solo funcione para el ejemplo en particular y, no para cualquier modificación del mismo, por tanto en este ejemplo se desarrollará una solución general y no particular (aquella en la cual se conoce el tope para P y Q).

Como se debe evaluar un número finito de veces la inecuación, la misma se considera como **TOPE** numérico **caso 1** y, se utilizan dos contadores, uno para P y el otro para Q de la siguiente manera:

P ← P + 1 **Q ← Q + 1**

Para un **TOPE** numérico **caso 1** se puede utilizar cualquier Estructura de Repetición sin embargo, para resolver este ejemplo no se puede utilizar la **ERP** ya que ésta no permite construir una condición como la inecuación lo requiere. Según la figura 4.22 para la **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**, la ***condición*** y la *Inicialización* queda

P ← 0
Q ← 0
Mientras ($P^3 + Q^4 - 2 * P^2 \quad \langle \quad 680$) **haga**

En la solución del ejemplo se utilizaran dos **ER** una para P y la otra para Q, en cada una de ellas se afectara la variable pertinente, pero las dos tendrán la misma condición.

Recuérdese que en la solución de este ejemplo no se requiere que el usuario ingrese valor alguno, éstos son construidos por la ejecución del programa creado por el programador.

Este ejemplo muestra un caso particular del **TOPE** numérico **caso 1**, en donde el usuario no ingresa dato alguno pero se requiere generar muchos valores dependiendo de la comprobación de una desigualdad.

Ahora se aplican las herramientas de la sección 4.1.6 para construir el Algoritmo, una vez construido éste, se elaborará el pseudocódigo. Primero se resolverá el ejemplo utilizando **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**, luego usando **ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS**y por último **ESTRUCTURA DE REPETICIÓN REPETIR HASTA**.

Solución al ejemplo 4.3 usando **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**

Herr. ***ALGORITMO***

1. Inicio

2. No aplica.

3. Inicializar contadores

Mientras($P^3 + Q^4 - 2P^2 \langle \quad 680$) haga

Mientras($P^3 + Q^4 - 2P^2 \langle \quad 680$) haga

Mostrar los valores de P y Q

Incrementar a Q

Q = Q+1

Fin_mientras

Incrementar a P

P = P+1

Inicializar a Q en cero

Fin_mientras

4. Fin

Pseudocódigo usando ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA

```
Algoritmo_muestra_terminos_P_y_Q
var
    Entero: P,Q
Inicio
 P ← 0
 Q ← 0
 Escribir ( "Los valores de P y Q son: ")
 Mientras (P ** 3 + Q ** 4 – 2 * P ** 2 < 680) haga
   Mientras (P ** 3 + Q ** 4 – 2 * P ** 2 < 680) haga

    Escribir ( P, Q)
    Q ← Q + 1
   Fin_mientras
   P ← P + 1
   Q ← 0
 Fin_mientras
Fin
```

Solución al ejemplo 4.3 usando **ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS**

Herr. ***ALGORITMO***

1. Inicio
2. No aplica.
3. Inicializar contadores
   Haga
     Haga
       Mostrar los valores de P y Q
       Incrementar a Q
         Q = Q+1
     Mientras($P^3 + Q^4 - 2P^2 \langle \quad 680$)
     Incrementar a P
       P = P+1
     Inicializar a Q en cero
   Mientras($P^3 + Q^4 - 2P^2 \langle \quad 680$)
4. Fin

Pseudocódigo usando ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS

```
Algoritmo_muestra_terminos_P_y_Q
var
     Entero: P,Q
Inicio
 P ← 0
 Q ← 0
 Escribir ( "Los valores de P y Q son: ")
 Haga
   Haga
     Escribir ( P, Q)
     Q ← Q + 1
   Mientras (P ** 3 + Q ** 4 – 2 * P ** 2 < 680)
   P ← P + 1
   Q ← 0
 Mientras (P ** 3 + Q ** 4 – 2 * P ** 2 < 680)
Fin
```

Nótese en ambas soluciones que se inicializa dos veces la variable Q, la primera como valor de arranque y la segunda, como reinicio del valor más bajo que puede tomar Q.

Solución al ejemplo 4.3 usando
ESTRUCTURA DE REPETICIÓN REPETIR HASTA

Herr. ***ALGORITMO***

1. Inicio

2. No aplica.

3. Inicializar contadores
Repita
Repita
Mostrar los valores de P y Q
Incrementar a Q
Q = Q+1
Hasta ($P^3 + Q^4 - 2P^2 \rangle =$ 680)
Incrementar a P
P = P+1
Inicializar a Q en cero
Hasta ($P^3 + Q^4 - 2P^2 \rangle =$ 680)

4. Fin

Pseudocódigo usando ESTRUCTURA DE REPETICIÓN REPETIR HASTA

```
Algoritmo_muestra_terminos_P_y_Q
var
    Entero: P,Q
Inicio
 P ← 0
 Q ← 0
 Escribir ( "Los valores de P y Q son: ")
 Repita
   Repita
    Escribir ( P, Q)
    Q ← Q + 1
   Hasta (P ** 3 + Q ** 4 – 2 * P ** 2 >= 680)
   P ← P + 1
   Q ← 0
 Hasta (P ** 3 + Q ** 4 – 2 * P ** 2 >= 680)
Fin
```

La primera instrucción Escribir es utilizada para que cuando se ejecute aparezca el contenido de la misma y a través de ésta se informe al usuario sobre los valores de P y Q que satisfacen la desigualdad, los cuales se muestran cuando se ejecuta la instrucción Escribir (P,Q).

Nótese que terminar de ejecutar la Estructura de Repetición más interna implica que el valor de Q sea tal que al ser reemplazado en la desigualdad en esta estructura, produce que la misma no se satisfaga, y cuando esto ocurre, se ha encontrado el valor más alto de Q para un P determinado, sin embargo, si Q no toma un nuevo valor una vez se termine la ejecución de la **ER** mas interna, ocasionará que no se cumpla la condición de la Estructura de Repetición mas externa, provocando esto que no se encuentren todos los valores de P, por

tanto, es importante inicializar en cero a Q antes de que el programa retorne a evaluar la condición mas externa.

Las soluciones del ejemplo 4.3 entregan todas las parejas de P y Q que satisfacen la desigualdad

$$P^3 + Q^4 - 2P^2 \langle \quad 680$$

Esto significa que para un P determinado, en la ejecución del programa se encontraran y visualizaran todos los que Q que junto a P satisfacen la inecuación.

La figura siguiente muestra el diagrama de flujo respectivo.

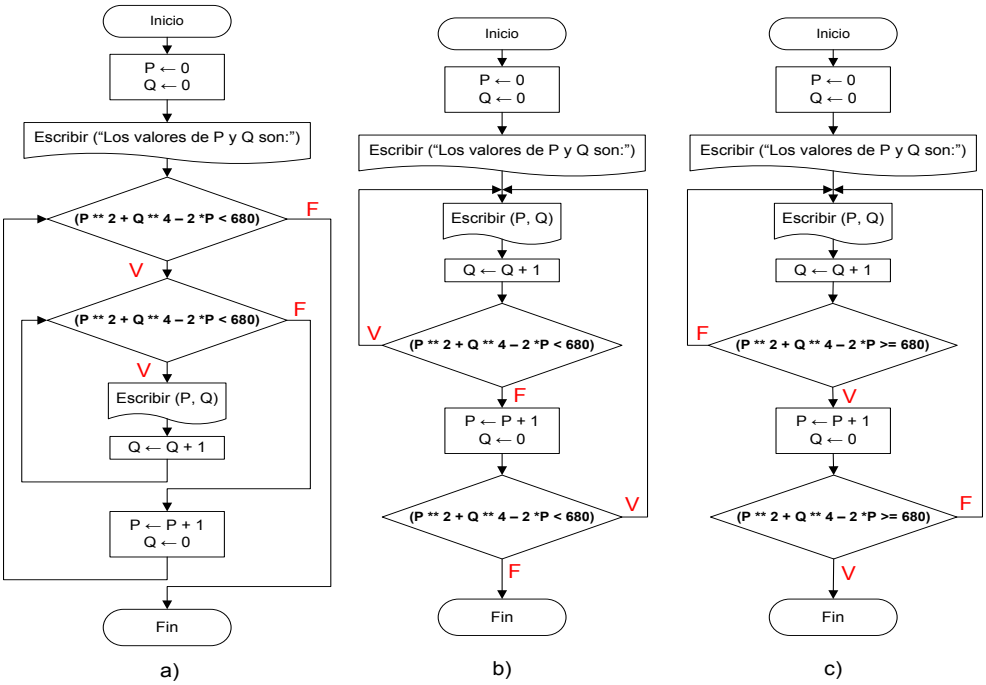


Figura 4.36 Solución del ejemplo 4.3 usando Estructuras de Repetición.
Mientras haga - **ESTRUCTURA DE REPETICIÓN MIENTRAS HAGA**
Haga Mientras - **ESTRUCTURA DE REPETICIÓN HAGA MIENTRAS**
Repita Hasta - **ESTRUCTURA DE REPETICIÓN REPETIR HASTA**