

**GUÍA DE TRABAJO # 4**  
**ESTRUCTURAS DE REPETICIÓN**

### 1. IDENTIFICACIÓN DE LA ASIGNATURA

<b>Asignatura:</b>	Lógica de Programación y Laboratorio						
<b>Código:</b>	000506001		<b>Área:</b>	Ciencias Básicas Tecnología			
<b>Plan de Estudios:</b>	2019		<b>Semestre:</b>	2			
<b>Créditos: 5</b>	<b>TPS: 6</b>	<b>TIS: 9</b>	<b>TPT: 96</b>	<b>TIT: 144</b>			
<b>Trabajo Independiente:</b>	<b>Trabajo Presencial:</b>						
Trabajo Teórico	X	Trabajo Práctico	X	Trabajo Teórico		Trabajo Práctico	X

### 2. TEMA Y COMPETENCIA:

<b>Contenido Temático:</b>	Variables contadoras, acumuladoras y centinelas Estructura repetitiva Para Estructura repetitiva Mientras Estructura repetitiva Haga-Mientras
<b>Competencia por Desarrollar:</b>	Reconoce las estructuras repetitivas: mientras y para con sus variaciones.
<b>Resultado de aprendizaje:</b>	Diseña algoritmos que usan las estructuras repetitivas para solucionar un problema para luego ser codificados en un lenguaje de programación de alto nivel.

### 3. RECURSOS REQUERIDOS

- Equipo de cómputo con conexión a Internet
- Procesador de textos como Word o Notepad++
- NetBeans u otro IDE para Java

### 4. MARCO TEÓRICO

Las estructuras de repetición, conocidas como bucles o ciclos, nos permiten ejecutar repetidas veces un bloque de código, esto mientras que la condición (o expresión lógica) que está definida en la estructura se cumpla, es decir, sea verdadera. Los ciclos nos ayudan a optimizar el tiempo empleado en escribir el código, esto porque nos evitan escribir el mismo bloque de instrucciones repetidas veces, mejorando así la legibilidad del programa.

Veamos un ejemplo bastante simple. Piense en el algoritmo que nos permita escribir en la pantalla 10 veces la frase “¡Hola Mundo!”. La solución que se nos viene a la mente sería usar 10 veces la instrucción **Escribir**, así:

```
//Escribimos ¡Hola Mundo! 10 veces en la pantalla
Escribir("¡Hola Mundo!")
```

Si analizamos esta solución, que es bastante simple, vemos se usa la misma instrucción 10 veces, repetidamente. Si el problema cambia y ahora queremos escribir en la pantalla la frase “¡Hola Mundo!” 1000 veces, ¿cuál sería la solución?

Siguiendo la misma línea de la solución anterior, deberíamos escribir la misma instrucción 1000 veces, pero ¿cuánto tiempo vamos a tardar escribiendo ese algoritmo? ¡Sí! ¡Mucho tiempo!

Una solución que nos permite optimizar nuestro tiempo consiste en usar una estructura repetitiva. Por ejemplo, una solución usando una estructura **Para** en el problema de mostrar 1000 veces en la pantalla “¡Hola Mundo!” podría ser la siguiente:

```
//Declaramos una variable que nos permite contar cuantas veces se ha ejecutado el ciclo
Entero: i=1

//Usamos la estructura Para para repetir la instrucción 1000 veces
Para (i=1; i<=1000; i=i+1) Haga
    Escribir("¡Hola Mundo!") //Mostramos un mensaje al usuario
Fin_Para
```

Cuando se analizan las líneas de código anterior, vemos que hay una estructura repetitiva (**Para-Haga**) que se encarga de repetir lo que está en su cuerpo (la instrucción **Escribir**). Como toda estructura repetitiva, hay una variable que controla la ejecución del ciclo, que es la variable **i**. Esta variable determina cuántas veces se repite el ciclo con base en una condición de control (**i<=1000**).

En general, las estructuras de repetición funcionan de una manera muy similar. **Para-Haga**, **Mientras-Haga** y **Haga-Mientras** son las estructuras de repetición básicas, pero antes de entrar en el detalle de estas estructuras vamos a describir tres tipos de variables que se usan en problemas con ciclos. Estas variables se denominan **contadoras**, **acumuladoras** o **centinelas**.

## 4.1 VARIABLES CONTADORAS, ACUMULADORAS Y CENTINELAS

Hay ciertas variables que, sin importar la naturaleza del algoritmo, se usan para contar, acumular o vigilar si se reúne una condición en la ejecución de nuestro algoritmo. A esas variables se les llaman **contadoras**, **acumuladoras** y **centinelas**, respectivamente.

- **Variables Contadoras:** como su nombre lo indica, es una variable cuyo trabajo es contar, parece obvio, ¿no?. Las variables contadoras se caracterizan por ser de tipo **Entero** y porque su valor incrementa (o disminuye) en un valor constante. La forma de una variable contadora puede ser, entre otras, las siguientes:

```

//Declaramos tres variables acumuladoras e iniciamos su valor en cero
Entero: i=0, j=0, k=0

//De esta manera hacemos que la variable incremente su valor en 1. Es decir, cuenta de 1 en 1
k++

//Este es otro ejemplo de una variable contadora que aumenta su valor de 1 en 1
i = i + 1

//Otro caso puede ser el siguiente, en el cual la variable cuenta de 2 en 2
j += 2

```

Note que para actualizar el valor de una variable contadora se pueden usar tres operadores diferentes:

- El operador de incremento unitario (`++`): el cual toma una variable y la incrementa en 1. La versión para decrementar una variable en 1, es el operador de decremento (`--`).
- El operador suma (`+`), el cual nos permite sumar un valor específico a la variable. Para que se considere un incremento, el resultado de la suma debe almacenarse en la misma variable a la que se está sumando el valor específico, así: `i=i+1`. Note que esta expresión nos indica que en la variable `i` se almacena el valor que tiene la variable `i` sumado con 1.
- El operador de incremento con asignación directa (`+=`), el cual nos permite tomar el valor de una variable, incrementarlo y almacenar el resultado en la misma variable. La versión del decremento de este operador es (`-=`).

- **Variables Acumuladoras:** son variables cuyo propósito es ir almacenando valores numéricos obtenidos de una suma continua de *valores que NO son constantes*. La forma de estas variables es la siguiente:

```

//Declaramos una variable acumuladora (x) y otra auxiliar (y)
Real: x=5, y

Escribir("Ingrese un número: ")
Leer(y)

// En este caso a la variable x se le suma el valor de y, el cual es un valor que no es constante
x = x + y

// En este caso usamos el operador de incremento con asignación
x += y

// En este otro caso se acumulan las multiplicaciones
x = x * y

```

- **Variables Centinelas:** estas variables suelen ser de tipo lógico y se caracterizan porque cambian su estado (o valor) cuando ocurre una situación determinada que se está monitoreando. A los centinelas también se les conoce como banderas y se usan para controlar ciclos en los que de antemano no se conoce cuántas veces se va a repetir. Veamos el siguiente ejemplo, el cual verifica si un número ingresado es negativo para cambiar el estado de la variable centinela.

```

//Declaramos una variable centinela y una variable para almacenar un número entero
Lógico: bandera = verdadero
Entero: y

//La variable centinela cambiará su valor cuando el número ingresado sea negativo
Escribir("Ingrese un número: ")
Leer(y)

Si (y < 0) Entonces
    bandera = falso
Fin_Si

```

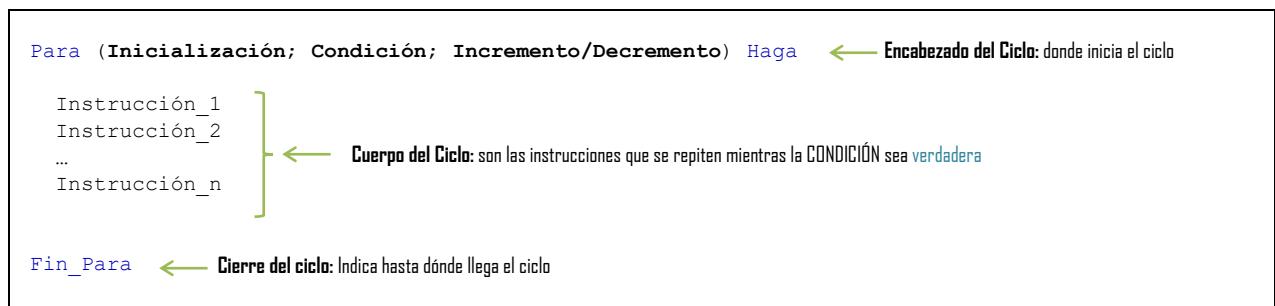
En este ejemplo vemos que la variable centinela (llamada `bandera`) cambia su valor cuando se reúne una condición específica, que en este caso es que el número ingresado sea negativo: `Si (y < 0)`.

## 4.2 ESTRUCTURA DE REPETICIÓN PARA

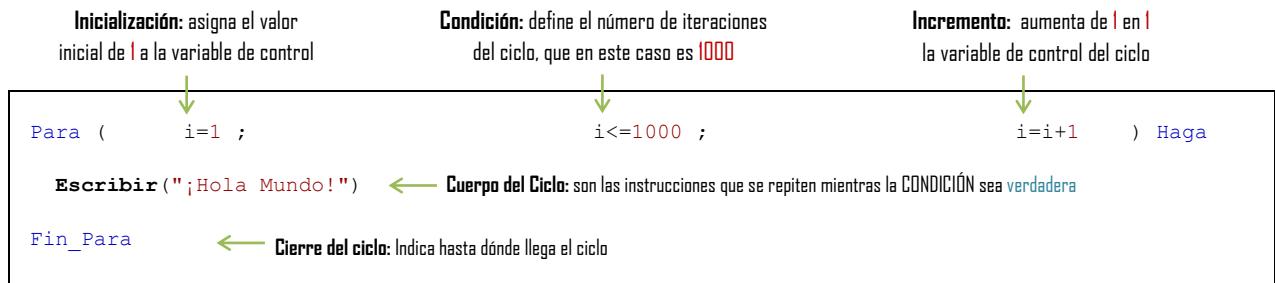
Este tipo de estructura de control es muy práctica cuando de antemano se conoce cuántas veces es necesario repetir un bloque de instrucciones. Esto se debe a que su funcionamiento se centra en contar las veces que se ha repetido el bloque de instrucciones, hasta llegar a un tope definido por una condición.

En sus diferentes versiones un ciclo `Para` tiene tres partes en su encabezado: la **inicialización** de la variable controladora del ciclo, la **condición** que controla el ciclo y el **incremento o decremento** de la variable controladora. Cada parte del encabezado está separada de la otra por un punto y coma (;). Además, el ciclo tiene un cuerpo, que contiene las instrucciones que se van a repetir, y el cierre del ciclo, que denota hasta donde llega el cuerpo de este.

La forma general de un ciclo `Para` es la siguiente:



Retomando el ejemplo que muestra 1000 veces en la pantalla la frase “¡Hola Mundo!”, analicemos cada una de las partes de la estructura:



- El **encabezado** del ciclo está compuesto por tres partes: primero está la **inicialización** (`i=1`), en la que se indica que la variable `i` inicia con el valor de uno. Después está la **condición** (`i<=1000`), la cual especifica que el ciclo se repetirá mientras la variable `i` sea menor o igual a 1000. Por último, está el incremento (`i=i+1`), el cual señala que al final de cada iteración la variable `i` aumentará su valor de uno en uno.
- Luego está el **cuerpo** del ciclo, el cual contiene las instrucciones que se repetirán una y otra vez, mientras que la **condición** en el encabezado sea **verdadera**.
- Por último, la instrucción `Fin_Para` indica el punto dónde finaliza el ciclo. En el proceso de ejecución de las instrucciones en el cuerpo, cuando se llega este punto se incrementa automáticamente la variable de control, de acuerdo con la instrucción definida para ello en el encabezado del ciclo y se evalúa la condición del encabezado para determinar si el ciclo se repite o no.

Veamos algunos ejemplos en los que podemos usar esta estructura de repetición.



### Ejemplo: Mostrando números en pantalla

Desarrolle un algoritmo que muestre en la pantalla los números enteros del 15 al 28.

Si analizamos este problema, su solución requiere un ciclo que repita 13 veces la instrucción **Escribir**. Pero ¿qué debemos a escribir en la pantalla?

Como indica el enunciado, se deben escribir los números del 15 al 28, es decir: 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 y 28. Estos números tienen la particularidad que inician en 15, aumentan de 1 en 1 y terminan en 28.

Con base en esto, una posible solución usando una estructura **Para** es la siguiente:

```
//Declaramos una variable para controlar el ciclo
Entero: x

//Ahora usamos el ciclo para ir del 15 al 28, esto lo hacemos con la variable de control x
Para (x = 15; x <= 28; x++) Haga

  Escribir( x ) //Como x cambia en cada iteración, aquí se mostrarían los números en pantalla

Fin_Para
```

Revisando el encabezado de la solución anterior, puedes observar que la variable de control inicia en 15 (que es el primer número que debemos mostrar), luego la condición nos indica que el ciclo se va a repetir mientras la variable **x** sea menor o igual a 10, lo que garantiza que el cuerpo del ciclo se ejecuta cuando **x** toma los valores de 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 y 28. En el último incremento, la variable de control toma el valor de 29, pero en ese caso la condición se hace falsa y por tanto termina la ejecución del ciclo.



### Ejemplo: Sumando números

Desarrolle un algoritmo que muestre en la pantalla la suma de los números enteros que están entre dos números ingresados por el usuario.

De acuerdo con el enunciado, para solucionar este problema se deben solicitar dos números al usuario. Después se debe identificar cuál de los números es el menor, a partir del cual empezaremos a sumar hasta llegar al mayor, que es el tope. Así, esta solución requiere que usemos un condicional para identificar el número menor y mayor y después un ciclo para ir sumando los números uno a uno desde el menor hasta el mayor.

Una posible solución a este problema es la siguiente:

```
//Se requieren 2 variables para los números, el mayor, el menor, un acumulador y la controladora
Entero: num1, num2, mayor, menor, suma=0, i

//Pedimos al usuario los dos números
Escribir("Ingrese dos números enteros: ")
Leer(num1, num2)
```

```

//Determinamos cual es el número menor y cual el mayor
Si (num1 < num2) Entonces
    menor = num1
    mayor = num2
Sino
    menor = num2
    mayor = num1
Fin_Si

//Usamos un ciclo para ir sumando los números del menor al mayor
Para (i = menor; i <= mayor; i=i+1) Haga

    suma = suma + i //Como i cambia en cada iteración, aquí estariamos sumando los números

Fin_Para

//Al terminar la ejecución del ciclo mostramos el resultado en la pantalla
Escribir("La suma de los números entre ", menor, " y ", mayor, " es:" suma )

```

En la solución anterior `num1` y `num2` son las variables en las que se almacenan los números que se piden al usuario. Hay dos variables auxiliares que son `menor` y `mayor`, las cuales se usan para almacenar en ellas los números menor y mayor, respectivamente. La variable `i` es la variable que se usa para controlar el ciclo y la variable `suma` es un acumulador que debe ser inicializado en cero (que es el elemento neutro de la suma).

Después de que el usuario ingresa los números, estos se comparan para determinar cuál es el menor y el mayor. Esto se hace utilizando un **condicional compuesto**. Posteriormente, está el encabezado del ciclo en el que la variable `i` empieza en el número menor e irá aumentando, de uno en uno, hasta sobrepasar al número mayor. En ese momento se deja de ejecutar el ciclo. En el cuerpo del ciclo tenemos que la variable `suma` va acumulando, en cada iteración, el valor de la variable `i`. Finalmente, después del ciclo se muestra el valor de la variable `suma`.

Analice lo siguiente, ¿qué pasaría si la instrucción `Escribir` estuviera en el cuerpo del ciclo?

**Para Recordar:**

- El ciclo `Para` se usa en los problemas en los que se conoce cuántas veces se van a repetir las instrucciones.
- El encabezado de un ciclo `Para` tiene tres partes:
  - La **inicialización** de la variable de control
  - La **condición** que mientras sea verdadera hará que se repita el ciclo
  - La instrucción de **incremento o decremento** de la variable de control



### 4.3 ESTRUCTURA DE REPETICIÓN MIENTRAS-HAGA

La estructura `Mientras-Haga` es una generalización de la estructura `Para`, y por tanto la podemos usar para crear ciclos en problemas en los que de antemano se conoce cuántas iteraciones se van a realizar y también en problemas en los que no se conoce cuántas iteraciones se van a realizar. Esta es la forma general de un ciclo `Mientras-Haga`:

```

Mientras (Condición) Haga ← Encabezado del Ciclo: donde inicia el ciclo y se establece la condición de continuación
    Instrucción_1
    Instrucción_2
    ...
    Instrucción_n } ← Cuerpo del Ciclo: son las instrucciones que se repiten mientras la Condición sea verdadera
Fin_Mientras ← Cierre del ciclo: Indica hasta dónde llega el ciclo

```

La estructura **Mientras-Haga** está compuesta por tres partes:

- El **Encabezado** (**Mientras-Haga**) que define la **condición** que determina si el ciclo se repite o no.
- El **Cuerpo** especifica cuáles son las instrucciones que se estarán repitiendo y que se ejecutarán siempre que la condición en el encabezado sea **verdadera**.
- El **Cierre** del ciclo (**Fin\_Mientras**), que indica hasta dónde llega el ciclo.

Al igual que el ciclo **Para**, la estructura **Mientras-Haga** se repetirá siempre que la **condición** del encabezado sea **verdadera**; por tanto, cuando esta **Condición** se hace **falsa**, el ciclo dejará de repetirse. Ahora, debe tener presente que esta estructura no especifica en dónde y cómo se hace la inicialización de la variable de control, y tampoco define cómo y en dónde cambia esa variable. Esto es importante porque si la variable de control no cambia su valor en la ejecución tendremos un ciclo que se repetirá “infinitas” veces, lo cual constituye un serio error del programador.

Es por esto que la variable de control tiene que inicializarse antes de iniciar el ciclo y también debe garantizarse que esa variable cambie en el cuerpo del ciclo, esto para evitar que el ciclo nunca se ejecute o que se ejecute infinitas veces.

Volamos al problema de mostrar en la pantalla 1000 veces la frase “¡Hola Mundo!”. Como vimos, la solución a este problema consiste en contar el número de veces que se ha mostrado la frase “¡Hola Mundo!”. Es decir, debemos usar una variable contadora para controlar el ciclo. Una posible solución con la estructura **Mientras-Haga** es la siguiente.

```
//Declaramos la variable contadora y que controla el ciclo
Entero: i

// Inicializamos la variable de control antes del ciclo
i=1

//El ciclo se debe repetir mientras el contador sea menor o igual a 1000
Mientras (i <= 1000) Haga

    Escribir("¡Hola Mundo!")

    //No se nos puede olvidar actualizar la variable de control en el cuerpo del ciclo
    i += 1

Fin_Mientras
```

En esta solución, **i** es la variable que controla el ciclo y es una variable contadora. Ahora preste atención en que su inicialización está antes de iniciar el ciclo, en donde le asignamos el valor de 1. Sin embargo, pregúntese: ¿Qué pasaría si la variable **i** no está inicializada antes de empezar el ciclo?

El encabezado del ciclo nos indica que las instrucciones se repetirán mientras la condición (**i<=1000**) sea **verdadera**. Es decir, ¿para qué valor de **i** dejará de repetirse el ciclo? Ahora, en el cuerpo del ciclo tenemos dos instrucciones, la instrucción **Escribir** que estará mostrando la frase que queremos en pantalla y la instrucción que incrementa y cambia el valor de la variable contadora **i**. Veamos otro ejemplo.



#### Ejemplo: Promediando números pares

Diseñe un algoritmo que pida al usuario un número indefinido de números enteros y al final muestre la suma de los pares ingresados. El programa debe dejar de pedir números cuando se ingresa el cero o un número negativo.

Al analizar el problema encontramos que no se sabe cuántos números se deben pedir, de ahí que no podamos usar una estructura de repetición Para. El enunciado nos indica que se dejaran de pedir números cuando se ingrese un cero o un negativo, es decir que esta es la condición que para la ejecución del ciclo. Por otro lado, en el cuerpo debemos pedir un número, evaluar si este es par y en dicho caso, sumarlo. Considerando estos elementos, una posible solución es la siguiente:

```
/* Declaramos la variable que controla el ciclo, que es el numero que ingresa el usuario además de un acumulador en el que almacenaremos la suma de los pares */
Entero: num, suma=0

/* Inicializamos la variable de control con un valor contrario a lo que indica la condición de parada */
num = 1

/* El ciclo se debe repetir mientras el numero ingresado sea positivo (contrario a la condición de parada) */
Mientras (num > 0) Haga

    Escribir("Ingrese un número positivo, o cero para terminar: ")
    Leer(num)

    //Se verifica si el número es positivo y par, en dicho caso lo sumamos al acumulador
    Si (num > 0 AND num MOD 2 == 0) Entonces
        suma += num
    Fin_Si

    Fin_Mientras

//Mostramos el resultado de la suma de los pares ingresados
Escribir("La suma de los números pares ingresados es: ", suma)
```

En esta solución solo se requieren dos variables: `num` y `suma`. La primera la usamos para almacenar los números que ingresará el usuario y que, además, nos permitirá controlar el ciclo **Mientras-Haga**. Como la variable de control siempre debe ser inicializada antes del ciclo, entonces le asignamos un valor que haga que la condición del ciclo sea **verdadera**. En este caso, se deben seguir pidiendo números cuando el número ingresado es positivo, por lo tanto, inicializamos la variable `num` con un número positivo.

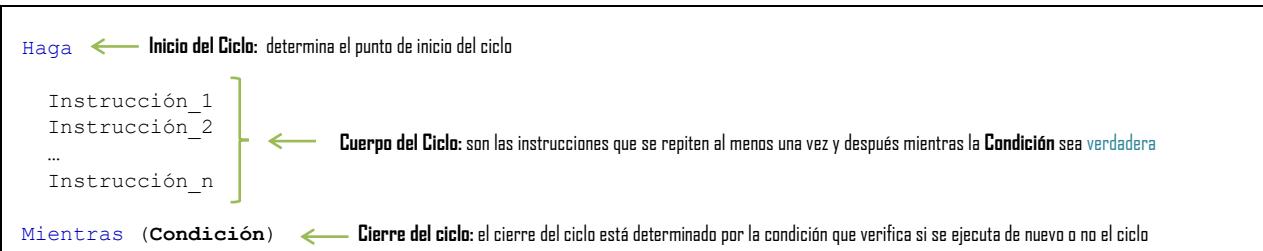
Después iniciamos la estructura de repetición, cuya condición debe ser verdadera mientras que el numero ingresado sea positivo, ya que el ciclo debe parar cuando ocurra lo contrario (que se ingrese un cero o un valor negativo). En este caso, la condición que nos permite continuar con la ejecución del ciclo cuando se ingrese un número positivo es `num > 0`. Como debemos pedir el número muchas veces, la instrucción de escritura y lectura para pedir el número al usuario debe estar en el cuerpo del ciclo, ya que, si ponemos esa instrucción antes del ciclo, solo se pedirá el número una sola vez, lo que es un error.

Una vez el usuario ha ingresado el número (que capturamos con la instrucción `Leer`), debemos verificar si dicho número cumple con la condición para acumularlo, es decir, que sea mayor a cero y que sea par. Esto es lo que hacemos con un condicional en el cuerpo del ciclo. Si el número cumple con la condición de ser positivo y par, lo acumulamos en la variable `suma`. El ciclo terminará cuando se ingrese un número negativo, y después del ciclo es que mostramos el resultado de la suma. Analice: ¿qué pasaría si la instrucción que escribe el resultado está en dentro del condicional, después de hacer la suma?

## 4.4 ESTRUCTURA DE REPETICIÓN HAGA-MIENTRAS

Otra estructura repetitiva es el ciclo **Haga-Mientras**. Esta estructura se diferencia del ciclo Mientras-Haga en que las instrucciones en el cuerpo del ciclo siempre se van a ejecutar por lo menos una vez, esto porque la condición se evalúa al

final del ciclo, a diferencia de la estructura [Mientras-Haga](#), en la cual la condición se evalúa al inicio del ciclo y antes de ejecutar el cuerpo. Esta es la forma general de un ciclo [Haga-Mientras](#):



La estructura [Haga-Mientras](#) está compuesta por tres partes:

- El **Inicio** ([Haga](#)) que determina el punto de partida de las instrucciones que se repetirán.
- El **Cuerpo** que contiene las instrucciones que se estarán repitiendo y que se ejecutarán siempre que la condición en el cierre del ciclo sea [verdadera](#).
- El **Cierre** del ciclo, que contiene la condición de parada, indica hasta dónde llega el ciclo.

Revisemos de nuevo el problema de mostrar en la pantalla 1000 veces la frase “¡Hola Mundo!”. Una posible solución con la estructura [Haga-Mientras](#) es la siguiente.

```
//Declaramos la variable contadora y que controla el ciclo
Entero: i

// Inicializamos la variable de control antes del ciclo
i=1

//Aquí inicia el ciclo que contiene las instrucciones a repetir
Haga

    Escribir("¡Hola Mundo!")

    //No se nos puede olvidar actualizar la variable de control en el cuerpo del ciclo
    i += 1

Mientras (i <= 1000) //El ciclo termina con la condición
```

Igual que en la estructura anterior, la variable `i` es la que controla el ciclo contando el número de veces que se impreso la frase “¡Hola Mundo!”. Observe que esta variable se debe inicializar antes de empezar el ciclo. No obstante, pregúntese: ¿Qué pasaría si la variable `i` se inicializa dentro del ciclo, después de la instrucción [Haga](#)? Luego están las instrucciones del cuerpo del ciclo que muestran el mensaje en la pantalla y que actualizan el valor de la variable controladora. Finalmente, el cierre del ciclo contiene la condición que indica que el ciclo se repetirá mientras que `i<=1000`.

La estructura [Haga-Mientras](#) suele utilizarse, entre otros, para la validación de datos de entrada. Veamos un ejemplo.



#### Ejemplo: validación de datos

Diseñe un algoritmo que solicite al usuario un número entre 1 y 100 (incluidos). Si el número está fuera de rango se debe mostrar un error y pedir el número de nuevo, de lo contrario se indicará que el número está en el rango correcto y finalizará el programa.

Al analizar el problema encontramos que no es posible de antemano determinar cuántos números erróneos se ingresarán antes de ingresar un número correcto. Es por esta razón que no podemos usar una estructura de repetición [Para](#). El

enunciado nos indica que se dejará de pedir el número cuando se ingrese uno que esté entre 1 y 100, es decir que el ciclo se debe repetir mientras que el número que ingresa esté por fuera de ese rango. Considerando esto, una posible solución es la siguiente:

```
/* Declaramos la variable que controla el ciclo, que es el número que ingresa el usuario */
Entero: num

/* En este caso no es necesario inicializar la variable de control ya que vamos a pedir su valor
en el propio ciclo*/
Haga

Escribir("Ingrese un número en el rango [1, 100]: ")
Leer(num)

//Se verifica si el número está fuera del rango para mostrar un error
Si (num < 0 OR num > 100) Entonces
    Escribir("Error, el número ingresado no está en el rango [1, 100]. Ingréselo nuevamente!")
Sino
    Escribir("Perfecto! El número ingresado está en el rango solicitado!")
Fin_Si

Mientras (num < 0 OR num > 100)
```

En esta solución hay algunas consideraciones:

- Observe que la variable que controla el ciclo es la variable `num`. La identificamos porque esta variable es la que se usa en la condición del ciclo. Si se es cuidadoso con la revisión, debe notar que `num` no se inicializó antes del ciclo y esto es porque, a diferencia del ciclo `Mientras-Haga`, la condición está al final y, por tanto, la variable no debe tener un valor al inicio del ciclo.
- Otro elemento importante aquí es la condición del ciclo. Note que en este ejemplo la condición es `(num<0 OR num>100)`, lo que indica que el ciclo se va a repetir mientras que el número que se ingresa en `num` sea menor a cero o mayor a 100, los cuales son los valores fuera del rango que generan el error y que deben hacer que se vuelva a pedir el número.

**Para Recordar:**

- El ciclo `Mientras-Haga` y el ciclo `Haga-Mientras` se utilizan en cualquier tipo de problemas en el que se requiere repetir un conjunto de instrucciones, bien sea de manera determinada o de manera indeterminada.
- En ambos ciclos, si la **condición** es verdadera estos repetirán, es decir, el ciclo frena la ejecución cuando la condición se hace falsa.
- En ambos ciclos se debe tener cuidado de modificar la variable de control del ciclo, esto para evitar un ciclo que se ejecute infinitas veces.
- La diferencia entre ambos ciclos es que el `Haga-Mientras` ejecuta por lo menos una vez el cuerpo del ciclo, mientras que el ciclo `Mientras-Haga` puede no ejecutar el bien sea porque la condición es falsa de entrada o porque la variable de control no tiene ningún valor inicial.



## 4.5 ESTRUCTURAS DE REPETICIÓN ANIDADAS

Así como en los condicionales, en los ciclos también puede haber ciclos dentro de otros ciclos, a lo que se llama ciclos anidados. En tal caso, el ciclo más interno es el que primero se ejecuta, lo que conlleva a que este se reinicie cada vez que el ciclo externo genere una nueva pasada por su cuerpo. Veamos esto con un ejemplo: ¿cuál es el valor que se muestra al finalizar el siguiente algoritmo?

```

/* El ciclo externo lo controla la variable i y al ciclo interno lo controla la variable j */
Entero: i=1, j, suma=0

//Aqui inicial el ciclo Externo
Mientras (i < 10) Haga
    j = 10

    Haga //Aqui inicia el ciclo interno
    suma += j
    j -= 4
    Mientras (j > 0)
        i += 3
    Fin_Mientras
    Escribir("La suma es: ", suma)

```

Para solucionar este problema debemos hacer la prueba de escritorio para las tres variables involucradas que son i, j y suma:

i	j	suma
1	10	10
	6	16
	2	18
	-2	
4	10	28
	6	34
	2	36
	-2	
7	10	46
	6	52
	2	54
	-2	
10		

Observe que el ciclo externo se ejecuta tres veces, cuando i=1, i=4 e i=7. Eso quiere decir que el ciclo interno se reinicia tres veces y las instrucciones dentro de él se ejecutan 3 veces, cuando j=10, j=6 y j=2. Siguiendo la prueba de escritorio, el valor que se muestra en pantalla es 54.



### Ejemplo: promediando número primos

Diseñe un algoritmo que calcule el promedio de los números primos de dos cifras.

Para desarrollar este problema, lo primero que debemos tener presente es que los números de dos cifras son aquellos que van desde el 10 hasta el 99, por tanto, necesitamos un ciclo (el externo) que inicie en el 10 y termine en el 99. Después, debemos considerar cuando un número es primo, esto para saber cuáles de los números entre el 10 y el 99 se deben promediar. Un número es primo cuando este es divisible SOLAMENTE por la unidad y por él mismo. Esto quiere decir que un número es primo SOLAMENTE cuando tiene dos divisores: la unidad y él mismo. En este sentido, para determinar si un número es primo debemos analizar cuántos divisores tiene, entre 1 y él mismo. Esto configura el segundo ciclo. Una posible solución, no óptima, a este problema es la siguiente.

```

/* Se requieren 2 variables para controlar los ciclos, una para contar los divisores, una para
sumar los primos y otra para contar el número de primos encontrados */
Entero: i, j, divisores, suma=0, contador=0

/* Aquí inicia el ciclo externo que se mueve del 10 al 99 usando la variable i */

```

```

Para (i=10; i<=99; i++) Haga

    //Cada que i cambia su valor, el número de divisores de i se deben reiniciar a 0
    divisores = 0

    //Este es el ciclo interno, el cual cuenta los divisores de i que hay entre 1 y él mismo
    Para (j=1; j<=i; j++) Haga
        //Verificamos si j es un divisor de i, en tal caso lo contamos
        Si (i MOD j == 0) entonces
            divisores = divisores + 1
        Fin_si
    Fin_Para

    /* Evaluamos cuántos son los divisores de i, si son 2, se considera primo y por tanto lo sumamos
       y lo contamos */
    Si (divisores == 2) entonces
        suma += i
        contador++
    Fin_Si
Fin_Para

Escribir("El promedio de los números primos entre 10 y 99 es: ", (suma/contador))

```

Observe que en esta solución hay dos ciclos `Para` anidados; el primero (controlado por la variable `i`) recorre los números de dos cifras, es decir los números que inician en 10 y terminan en 99. El segundo ciclo (el interno) tiene como trabajo contar los divisores del número `i`. Es por esta razón que ese ciclo siempre empieza en 1 y termina en el valor que tiene `i`. Al terminar el ciclo interno se evalúa el número de divisores encontrados. Si `i` sólo tiene dos divisores, entonces `i` es primo y por tanto debemos sumarlo y contarla, para poder calcular el promedio más adelante. Es importante tener presente que la variable `divisores` debe reiniciar su valor a cero cada vez que `i` cambia de número, sino se produciría un error lógico y no encontraría ningún número primo, esto porque 10 tiene 4 divisores y si no se reinicia el contador de divisores, el algoritmo encontrará que el 11 tiene 6 divisores: los 4 del 10 más los 2 del 11, y así sucesivamente.

#### Extiende tu conocimiento ...

Si quieras ver un video explicativo sobre lo que es un ciclo y los tipos de ciclos, te invito a que visites el siguiente enlace:

<https://www.youtube.com/watch?v=E8f1rwUfNG4>



## 5. PROCEDIMIENTO

En el Centro de Atención Telefónica “**Call Services**” se registran todas las llamadas que se atienden. De cada llamada se almacenan los datos básicos del cliente: nombre, el documento de identificación y su número telefónico. Además, de la llamada como tal se registra su tiempo en SEGUNDOS, el cual se discrimina en dos: tiempo de conversación y tiempo de documentación. Una llamada puede quedar en estado incompleto, lo que sucede cuando esta se termina de manera intempestiva. Como se desconoce el número de llamadas que serán registradas en el día, después de registrar una llamada se debe indagar al usuario si se desea o no registrar otra llamada. Desarrolle un programa que, además de permitir registrar las llamadas, muestre la siguiente información al finalizar todos los registros:

- El tiempo promedio de atención en MINUTOS entre las llamadas que tienen el estado incompleto
- El tiempo total en MINUTOS de conversación y el tiempo total en SEGUNDOS de documentación entre todas las llamadas con estado completo.
- Los datos del cliente que tuvo la llamada con la mayor cantidad de tiempo

Analice el enunciado y con base en él:

- Defina el cliente, el usuario y las entidades del mundo del problema que intervienen en la solución

- Defina los requerimientos que deben ser implementados para construir una solución al problema
- Describa los atributos y métodos que deben tener las clases que representan las entidades del mundo del problema
- Con base en la descripción anterior, bosqueje el diagrama de clases de la solución
- Implemente la solución en seudocódigo o algún lenguaje de programación orientado por objetos

## 6. RÚBRICA DE EVALUACIÓN

A continuación, se describen los elementos considerados en la evaluación y su rango de valoración.

Resultado de aprendizaje	Indicador de logro	Rango de valoración
Diseño de la solución al problema planteado que incluye la especificación del problema, especificación de requerimientos, diagrama de clases e implementación	<b>Especificación del problema:</b> identifica de manera correcta el cliente, el usuario, la lista preliminar de requerimientos y los elementos del mundo que intervienen en la solución del problema	0.0 – 0.5
	<b>Especificación de requerimientos:</b> describe correctamente cada funcionalidad del sistema e identifica las entradas y salidas específicas de estas	0.0 – 1.0
	<b>Diagrama de clases:</b> hay una correspondencia entre las clases y las entidades del mundo del problema identificadas. Además, se evidencia la identificación de los métodos que le darán solución al problema con base en los requerimientos	0.0 – 1.0
	<b>Implementación:</b> se entrega un seudocódigo o código en un lenguaje de programación que da solución al problema y es coherente con la especificación de requerimientos y el diagrama de clases	0.0 – 2.5

## 7. BIBLIOGRAFÍA

- Jorge Villalobos, Rubby Casallas, “Fundamentos de Programación –Aprendizaje Activo Basado en Casos”, Universidad de los Andes.  
Disponible en: <http://cupi2.uniandes.edu.co/images/APO1/fundamentos-de-programacion.pdf>
- Jorge Martínez, “Fundamentos de Programación en Java”, Universidad Complutense de Madrid.  
Disponible en: <https://docs.google.com/file/d/0Byy7aUI9u4fBdnZjVnZOampmTjA/edit>
- UskoKruM2010, “Tutorial Java - 10. Estructura Repetitiva (Bucle) WHILE | UskoKruM2010”. Tutorial en YouTube.  
Disponible en: <https://www.youtube.com/watch?v=EL7o6R4Pyp4>
- UskoKruM2010, “Tutorial Java - 11. Estructura Repetitiva (Bucle) DO-WHILE | UskoKruM2010”. Tutorial en YouTube.  
Disponible en: <https://www.youtube.com/watch?v=V3KmwRiB2pk>
- UskoKruM2010, “Tutorial Java - 12. Estructura Repetitiva (Bucle) FOR | UskoKruM2010”. Tutorial en YouTube.  
Disponible en: <https://www.youtube.com/watch?v=qman8My2oDw>

Elaborado Por:	Carlos Andres Mera Banguero
Versión:	1.0
Fecha	Noviembre de 2022
Aprobado Por:	Comité Curricular Departamento de Sistemas de Información Acta 02 del 7 de febrero de 2023