GUÍA DE TRABAJO - ESTRUCTURAS REPETITIVAS

1. IDENTIFICACIÓN DE LA ASIGNATURA

Asignatura:	Lógica y Representación I			
Código:	2554208		**Tipo de Curso:**	Obligatorio
Plan de Estudios:	5		**Semestre:**	1
Créditos: 3	**TPS: 6**	**TIS: 3**	**TPT: 64**	**TIT: 32**

2. OBJETIVO

Unidad:	Estructuras Repetitivas
Objetivo de la asignatura:	Desarrollar en el estudiante la habilidad para plantear soluciones lógicas a problemas computacionales que deriven en la implementación de programas en un lenguaje de programación
Resultados de aprendizaje abordados:	▪ RAA2: Emplear las estructuras de control de flujo en el diseño de algoritmos y programas que requieren la toma de decisiones y la repetición de bloques de código ▪ RAA4: Escribir algoritmos y programas que solucionen problemas computacionales
Contenido temático:	▪ Variables contadoras, acumuladoras y centinelas ▪ Estructura Mientras-Haga ▪ Estructura Para-Haga

3. RECURSOS REQUERIDOS

- Lector de archivos PDF
- Bloc de notas como Notepad++

4. MARCO TEÓRICO

Las estructuras de repetición, también conocidas como bucles o ciclos, nos permiten ejecutar múltiples veces un bloque de código, esto mientras que una condición (o expresión lógica) 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!")
Escribir("¡Hola Mundo!")
Escribir("¡Hola Mundo!")
Escribir("¡Hola Mundo!")
Escribir("¡Hola Mundo!")
Escribir("¡Hola Mundo!")
Escribir("¡Hola Mundo!")
```

```
Escribir("¡Hola Mundo!")
Escribir("¡Hola Mundo!")
Escribir("¡Hola Mundo!")
```

Si analizamos esta solución, que es bastante simple, vemos que se usa la misma instrucción 10 veces. 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-Haga` en el problema de mostrar 1000 veces en la pantalla "¡Hola Mundo!" podría ser la siguiente:

```
Inicio

# 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 hasta 1000 incremento 1 Haga
  Escribir("¡Hola Mundo!") # Se muestra el mensaje en la pantalla
Fin_Para

Fin
```

Cuando se analizan las líneas de código anterior, vemos que hay una estructura repetitiva (`Para-Haga`) que se encarga de repetir muchas veces un conjunto de instrucciones (aquí 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 (`hasta 1000`).

En general, las estructuras de repetición funcionan de una manera muy similar. `Para-Haga` y `Mientras-Haga` son las estructuras de repetición básicas, pero antes de entrar en el detalle 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 su ejecución. 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. Las variables contadoras se caracterizan por ser de tipo **`Entero`**, se suelen inicializar en 0 y su valor incrementa (o disminuye) en un valor constante. La forma de una variable contadora puede ser, entre otras, las siguientes:

```
Inicio

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

# Así hacemos que la variable incremente su valor en 1. Es decir, cuenta de 1 en 1
k +=1

# 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

Fin
```

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

- El operador suma (**+**) , el cual nos permite tomar la variable y sumar a esta un valor específico. En este caso el resultado se debe almacenar en la misma variable, por ejemplo: `i=i+1`. Note que esta expresión nos indica que en la variable `i` se almacena el valor que tiene esa variable sumado con `1`.
- El operador de incremento con asignación directa (**+=**) nos permite tomar el valor de una variable, incrementarlo y almacenar el resultado directamente en la misma variable.

- **Variables Acumuladoras:** una variable acumuladora es una variable utilizada para acumular o sumar valores numéricos de manera continua. Estas variables acumulan valores que *NO son constantes*. La forma de estas variables es la siguiente:

```
Inicio

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

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

# 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

Fin
```

- **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.

```
Inicio

# 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

Fin
```

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-Haga

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-Haga` tiene tres partes en su encabezado: la ***inicialización*** de la variable controladora del ciclo, el ***valor de finalización*** que define el límite ciclo y el ***incremento*** o ***decremento*** de la variable controladora. Además, el ciclo tiene un cuerpo, que contiene las instrucciones que se van a repetir, y el cierre del ciclo, que denota hasta dónde llega su cuerpo.

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

```
Para inicialización hasta Valor_final+1 incremento Val_incremento Haga  ← Encabezado del Ciclo

  Instrucción_1
  Instrucción_2
  …                                  ← Cuerpo del Ciclo: son las instrucciones que se repiten hasta que la variable de control llegue al valor_final
  Instrucción_n

Fin_Para   ← Cierre del ciclo: Indica hasta dónde llega el ciclo
```

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

Inicialización: asigna el valor inicial de 1 a la variable de control

Valor_final: define el número de iteraciones del ciclo, que en este caso es 1000. Note que el se debe sumar 1 porque no incluye el último dato.

Incremento: aumenta de 1 en 1 la variable de control del ciclo

```
Para      i=1          hasta          1001        incremento        1      Haga

  Escribir("¡Hola Mundo!")   ← Cuerpo del Ciclo: son las instrucciones que se repiten

Fin_Para        ← Cierre del ciclo: Indica hasta dónde llega el ciclo
```

- 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 el ***valor_final*** (`1001`), el cual especifica que el ciclo se repetirá hasta que la variable `i` llegue a `1000` (uno valor antes de `1001`). Por último, está el incremento (`1`), el cual señala que al final de cada iteración la variable `i` incrementará 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 ***variable sea menor al valor_final***.
- 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 muchos números en la 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-Haga` es la siguiente:

```
Inicio

# 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 hasta 29 incremento 1 Haga

  Escribir( x ) # Como x cambia en cada iteración, aquí se muestran los números en pantalla

Fin_Para

Fin
```

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 el valor final nos indica que el ciclo se va a repetir hasta que la variable `x` llegue al valor 28, 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.

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 y a partir de ese número 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:

```
Inicio

# Variables
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 hasta mayor+1 incremento 1 Haga

  suma = suma + i # Como i cambia en cada iteración, aquí estaríamos 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 )

Fin
```

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 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 cual 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 llegar 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?

Importante: La estructura **Para-Haga** 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
- El ***valor de finalización*** de la variable de control
- El ***valor 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-Haga`, 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            ← Cuerpo del Ciclo: son las instrucciones que se repiten mientras la Condición sea verdadera
   …
   Instrucción_n
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-Haga`, 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 dentro del ciclo. 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.

Siguiendo lo anterior, la variable de control tiene que inicializarse antes del 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.

```
Inicio

# 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

Fin
```

En esta solución, `i` es la variable que controla el ciclo y es una variable contadora. Preste atención en que la inicialización de esta variable se hace antes de iniciar el ciclo, en donde le asignamos el valor de 1. Sin embargo, ¿Qué pasaría si la variable `i` no está inicializada antes de empezar el ciclo?

El encabezado del ciclo nos indica que el bloque de código del ciclo se repetirá mientras la condición (`i<=1000`) sea `verdadera`. Ahora, en el cuerpo del ciclo tenemos dos instrucciones, la instrucción `Escribir` que muestra la frase que queremos en la pantalla y la instrucción que incrementa y cambia el valor de la variable contadora `i`. Veamos otro ejemplo.

Ejemplo – Promedio de números pares

- Diseñe un algoritmo que pida al usuario números enteros y al final muestre la suma de los números pares ingresados. El programa debe finalizar cuando se ingresa el número cero o un número negativo, los cuales no deben ser considerados en el promedio.

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:

```
Inicio

""" La variable que controla el ciclo es el número que ingresa el usuario, además, creamos un
acumulador, para sumar de los números pares, y un contador, para contarlos"""
Entero num, suma=0, cont=0

# Inicializamos la variable de control con un valor que haga verdadera la condición
num = 1

# El ciclo se repete mientras que el numero ingresado sea positivo
Mientras (num > 0) Haga

  Escribir("Ingrese un número positivo. Si quiere terminar ingrese el cero: ")
  Leer(num)

  # Se verifica que el número sea positivo y par para sumarlo al acumulador y contarlo
```

```
  Si (num > 0 AND num MOD 2 == 0) Entonces
    suma += num
    cont += 1
  Fin_Si

Fin_Mientras

# Mostramos el promedio de los pares ingresados
Escribir("El promedio de los números pares ingresados es: ", (suma/cont))

Fin
```

En esta solución solo se requieren tres variables: `num`, `suma` y `cont`. 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 pedir números mientras que el número ingresado es positivo, por lo tanto, inicializamos la variable `num` con un número positivo.

Recuerde que el ciclo debe parar cuando 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?

Veamos otro ejemplo.

Ejemplo – Validación de datos

- Diseñe un algoritmo que solicite al usuario un número entre 1 y 100 (ambos 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:

```
Inicio

# Declaramos la variable que controla el ciclo, que es el número que ingresa el usuario
Entero num

# Iniciamos la variable con un valor que cumpla la condición de repetición
num = -1

Mientras (num < 0 OR num >100) 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

Fin_Mientras

Fin
```

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.
- 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 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.

4.4 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?

```
Inicio

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

# Aquí inicial el ciclo Externo
Mientras (i < 10) Haga
  j = 10

  Mientras (j > 0) # Aquí inicia el ciclo interno
    suma += j
    j -= 4
  Fin_Mientras

  i += 3
Fin_Mientras

Escribir("La suma es: ", suma)
Fin
```

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 – Promedio de 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 debemos considerar en el promedio.

Un número es primo cuando este es divisible SOLAMENTE por la unidad y por el mismo. Esto quiere decir que un número es primo SOLO cuando tiene dos divisores: la unidad y él mismo. En este sentido, para determinar si un número es primo debemos analizar cuántos divisores tiene el número. Esto configura el segundo ciclo. Una posible solución, no óptima, a este problema es la siguiente.

```
Inicio

""" 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 hasta 100 incremento 1 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 hasta i+1 incremento 1 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 divisores tiene i, si son 2, si es primo 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))

Fin
```

Observe que en esta solución hay dos ciclos `Para-Haga` 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 cuántos divisores tiene `i`. Si sólo son dos divisores, entonces `i` es primo y, por tanto, debemos sumarlo y contarlo para 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 encontrará 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.

5. PROCEDIMIENTO

Ejercicio – Ciclos

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, la lista de requerimientos y las entidades del mundo.
- Defina los requerimientos en su formato extendido
- 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
- Desarrolle la solución en seudocódigo e implemente la misma un lenguaje de programación orientado por objetos

6. BIBLIOGRAFÍA

- Herrera, A., Ebratt, R., & Capacho, J. (2016). Diseño y construcción de algoritmos. Barranquilla: Universidad del Norte.
- Aguilar, L. (2020). Fundamentos de programación: algoritmos, estructuras de datos y objetos (5a.ed.). México: Mc Graw Hill.
- Thomas Mailund. Introduction to Computational Thinking: Problem Solving, Algorithms, Data Structures, and More, Apress, 2021.
- Steven S. Skiena. The Algorithm Design Manual. Third Edition, Springer, 2020.
- Samuel Tomi Klein. Basic Concepts in Algorithms, World Scientific Publishing, 2021.

Elaborado Por:	Carlos Andrés Mera Banguero
Versión:	1.0
Fecha	Enero de 2024