

# Ministerio de Producción

## Secretaría de Industria y Servicios

### Subsecretaría de Servicios Tecnológicos y Productivos

y

# Ministerio de Educación y Deportes

A través del

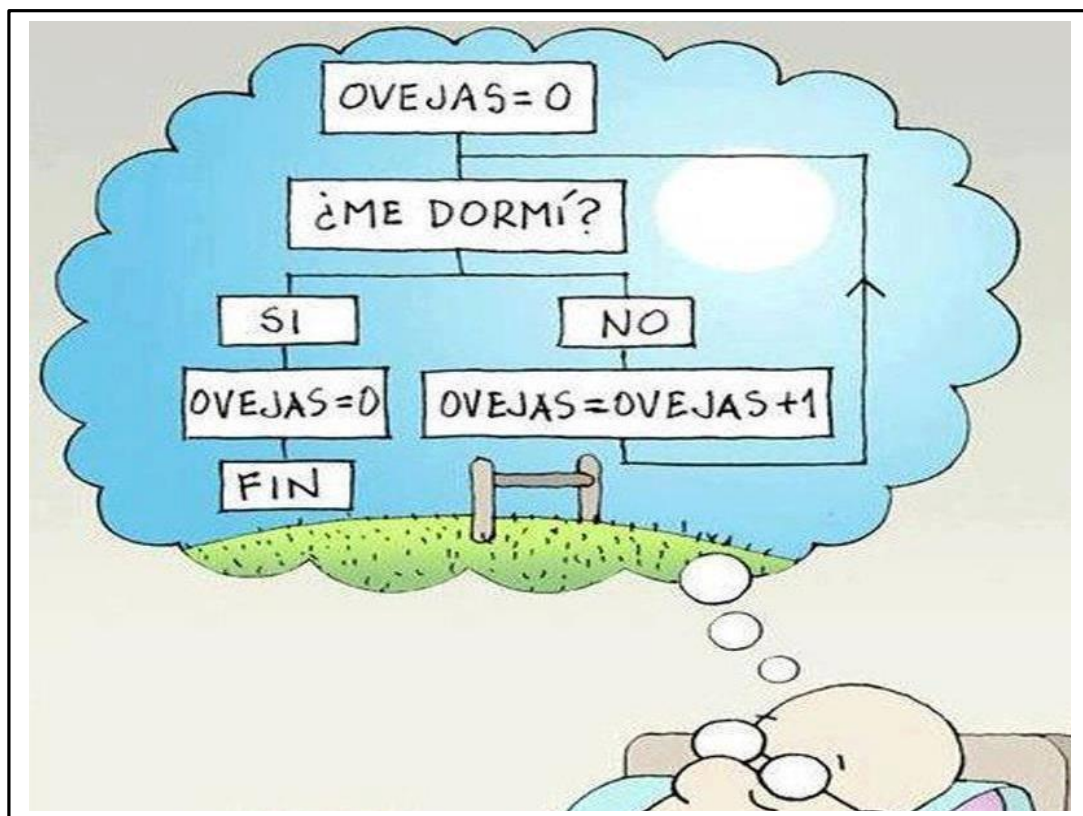
**inet** | Instituto Nacional de  
Educación Tecnológica



## Analistas del Conocimiento

### Dimensión Programador

# Guía de Ejercicios Prácticos para el Módulo Técnicas de Programación



## Tabla de Contenido

<b>INTRODUCCIÓN .....</b>	<b>4</b>
<b>REFERENCIAS DE LOS TIPOS DE DATOS .....</b>	<b>5</b>
<b>ENUNCIADOS DE LOS EJERCICIOS A DESARROLLAR EN LA GUÍA .....</b>	<b>6</b>
<b>EJERCICIOS ESTRUCTURAS DE CONTROL .....</b>	<b>6</b>
EJERCICIO 1 – SECUENCIAL .....	6
EJERCICIO 2 – ALTERNATIVA SIMPLE .....	6
EJERCICIO 3 – ALTERNATIVA DOBLE .....	6
EJERCICIO 4 – ALTERNATIVA DOBLE .....	6
EJERCICIO 5 – ALTERNATIVA MÚLTIPLE .....	6
EJERCICIO 6 – REPETITIVA MIENTRAS (WHILE) .....	7
EJERCICIO 7 – REPETITIVA HACER MIENTRAS (DO WHILE) .....	7
EJERCICIO 8 – REPETITIVA PARA (FOR) .....	7
<b>EJERCICIOS SOBRE ESTRUCTURAS DE DATOS .....</b>	<b>7</b>
EJERCICIO 9 – ARREGLO BOOLEANO .....	7
EJERCICIO 10 – DOS ARREGLOS .....	7
EJERCICIO 11 – PILA .....	8
<b>EJERCICIOS ALGORITMOS FUNDAMENTALES .....</b>	<b>8</b>
EJERCICIO 12 – ORDENAMIENTO POR INSERCIÓN .....	8
EJERCICIO 13 – ORDENAMIENTO DE LA BURBUJA .....	8
EJERCICIO 14 – ORDENAMIENTO POR SELECCIÓN .....	8
EJERCICIO 15 – BÚSQUEDA SECUENCIAL .....	9
<b>SOLUCIONES PROPUESTAS .....</b>	<b>10</b>
<b>EJERCICIOS ESTRUCTURAS DE CONTROL .....</b>	<b>10</b>
EJERCICIO 1 – SECUENCIAL .....	10
EJERCICIO 2 – ALTERNATIVA SIMPLE .....	12
EJERCICIO 3 – ALTERNATIVA DOBLE .....	15
EJERCICIO 4 – ALTERNATIVA DOBLE .....	18
EJERCICIO 5 – ALTERNATIVA MÚLTIPLE .....	21
EJERCICIO 6 – REPETITIVA MIENTRAS (WHILE) .....	23
EJERCICIO 7 – REPETITIVA HACER MIENTRAS (DO WHILE) .....	26
EJERCICIO 8 – REPETITIVA PARA (FOR) .....	29
<b>EJERCICIOS SOBRE ESTRUCTURAS DE DATOS .....</b>	<b>31</b>
EJERCICIO 9 – ARREGLO BOOLEANO .....	31
EJERCICIO 10 – DOS ARREGLOS .....	34
EJERCICIO 11 – PILA .....	37
<b>EJERCICIOS ALGORITMOS FUNDAMENTALES .....</b>	<b>41</b>
EJERCICIO 12 – ORDENAMIENTO POR INSERCIÓN .....	41
EJERCICIO 13 – ORDENAMIENTO DE LA BURBUJA .....	43
EJERCICIO 15 – BÚSQUEDA SECUENCIAL .....	47
<b>FUENTES DE INFORMACIÓN .....</b>	<b>50</b>

## Introducción

La guía práctica del Módulo Técnicas de Programación incluye ejercicios correspondientes vinculados a los contenidos desarrollados en el apunte teórico del módulo. El objetivo de esta guía es brindar una herramienta de apoyo, que facilite el desarrollo de los temas y posibilite aplicar los conocimientos adquiridos mostrando casos prácticos y su resolución propuesta.

Como primer paso, es necesario introducir una técnica utilizada para validar la resolución de problemas con algoritmos, de uso frecuente en el ámbito informático, denominada: **Pruebas de Escritorio**.

Una *prueba de escritorio* se utiliza para validar utilizando datos reales como ejemplo, un algoritmo definido y así comprobar si se obtiene el resultado deseado. El proceso para realizar una prueba de escritorio consiste en hacer seguimiento de un algoritmo recorriendo sus líneas secuencialmente, simulando el funcionamiento de la computadora. A medida que se van recorriendo las líneas se anotan en una tabla auxiliar los valores que van tomando las variables.

Para poder realizar una prueba de escritorio, es necesario como primera medida identificar cuáles son las variables de entrada, cuáles son las variables auxiliares y cuáles son las variables de salida. Una vez identificadas las variables, se debe distinguir el valor que toma cada una de ellas, a medida que se realizan las operaciones del algoritmo, utilizando para ello una tabla.

A continuación, se muestra un ejemplo sencillo para clarificar el concepto, suponiendo que tenemos el siguiente problema:






*“Se desea diseñar un algoritmo que, de acuerdo a la altura de una persona, le permita entrar a un juego en un parque de diversiones. En este caso, para poder subir a la montaña rusa, si la persona mide 1.30 mts. o más, puede ingresar en caso contrario no puede.”*

Algoritmo a Probar:	Prueba de Escritorio	
INICIO validarAltura DECIMAL alturaPermitida = 1.30 SI (alturaPersona >= alturaPermitida) ENTONCES: “Puede ingresar a la montaña rusa” SINO: “No puede ingresar a la montaña rusa” FIN_SI FIN	Altura Persona	Salida
	1.50	“Puede ingresar a la montaña rusa”
	1.20	“No puede ingresar a la montaña rusa”
	1.30	“Puede ingresar a la montaña rusa”
	1.00	“No puede ingresar a la montaña rusa”

A lo largo de toda la guía se desarrollarán diferentes ejercicios, cada uno de ellos tiene un enunciado que describe el problema, su resolución propuesta y la prueba de escritorio para validarlo.

Las tablas utilizadas para mostrar las pruebas de escritorio varían de acuerdo a la complejidad del ejercicio.

## Referencias de Símbolos del Diagrama de Flujo

Nombre del Símbolo	Representación del Símbolo	Función
<b>Inicio/Final</b>		Representa el comienzo o la finalización de un algoritmo.
<b>Entrada</b>		Representa la entrada de datos por medio de un dispositivo de entrada, en general representa entrada de datos por teclado
<b>Proceso</b>		Representa una operación como puede ser una operación matemática (suma, resta, multiplicación, etc), una declaración de una variable o una asignación de una variable a un valor, etc.
<b>Decisión</b>		Representa una condición: realiza comparaciones entre distintos valores. Al evaluar dicha condición puede dar como resultado dos posibles valores: verdadero o falso, en función de que si esa condición se cumple o no.
<b>Salida</b>		Representa la salida de datos o resultados. Sirve para representar los mensajes que el sistema le muestra al usuario por pantalla.

## Referencias de los Tipos de Datos

Tipo de Datos en Pseudocódigo	Explicación	Ejemplo	Correspondencia en Java
<b>Texto</b>	Representa un carácter o una cadena de caracteres. Su valor debe ir encerrado entre comillas dobles. Si se desea concatenar texto con variables (que pueden ser de tipo: entero, decimal o booleano), se utiliza el signo más (+), de la siguiente manera: Entero suma = 20 + 50; "El valor de la suma es: " + suma	"Hoy es viernes", "b", "El precio del producto es \$20"	String
<b>Entero</b>	Representa un número entero.	10,29,30	int
<b>Decimal</b>	Representa un número decimal.	21.3, 43.9, 23.564	float o double
<b>Booleano</b>	Define una bandera que puede tomar dos posibles valores: Verdadero o Falso.	Verdadero, falso	boolean

## Enunciados de los Ejercicios a Desarrollar en la Guía

---

### Ejercicios Estructuras de Control

#### Ejercicio 1 – Secuencial

Escribir un algoritmo que permita realizar una suma de dos números enteros. El usuario deberá ingresar primero un número, luego el siguiente número, y el sistema arrojará el resultado correspondiente.

---

#### Ejercicio 2 – Alternativa Simple

Escribir un algoritmo que permita loguearse (registrarse) a un sistema, ingresando un nombre de usuario y la contraseña adecuada. Considerar que tanto el usuario como la contraseña están formados sólo por letras. El sistema deberá validar que el usuario y la contraseña sean correctas, comparándolas con lo que el sistema tiene registrado para ese usuario.

*\*\*Aclaración, en los sistemas reales, el inicio de sesión es mucho más complejo que lo que se muestra a continuación. Se ha simplificado el proceso, abstrayendo la validación a una función denominada `esValido()` que resuelve la verificación del usuario y su contraseña.*

---

#### Ejercicio 3 – Alternativa Doble

Escribir el algoritmo que, a partir de la cantidad de bancos de un aula y la cantidad de alumnos inscriptos para un curso, permita determinar si alcanzan los bancos existentes. De no ser así, informar además cuantos bancos sería necesario agregar. El usuario deberá ingresar por teclado tanto la cantidad de bancos que tiene el aula, como la cantidad de alumnos inscriptos para el curso.

---

#### Ejercicio 4 – Alternativa Doble

Diseñar un algoritmo que permita aplicar un descuento del 10% al monto total de una compra si la forma de pago empleada es de contado. El usuario deberá ingresar el monto de la compra realizada y la forma de pago utilizada. Si es contado, deberá aplicar el descuento, sino se deberá mostrar un mensaje informando que para dicha forma de pago no tiene descuento.

---

#### Ejercicio 5 – Alternativa Múltiple

Diseñar un algoritmo que devuelva el nombre del mes, a partir del número de mes, ingresado por teclado, por el usuario. Si el usuario ingresa un número de mes que no exista, se deberá mostrar un mensaje indicando que el número ingresado no es correcto.

---

### Ejercicio 6 – Repetitiva Mientras (While)

Diseñar un algoritmo que muestre por pantalla la tabla de multiplicación del número que ingrese el usuario. Para definir hasta que número desea que muestre la tabla de multiplicación el usuario también deberá ingresar este valor. La tabla de multiplicación a mostrar debe empezar en la multiplicación por 1.

---

### Ejercicio 7 – Repetitiva Hacer Mientras (Do While)

Diseñar un algoritmo que muestre por pantalla la tabla de multiplicación del número que ingrese el usuario. Para definir hasta que número desea que muestre la tabla de multiplicación el usuario también deberá ingresar este valor. La tabla de multiplicación a mostrar debe empezar en la multiplicación por 1.

---

### Ejercicio 8 – Repetitiva Para (For)

Diseñar un algoritmo que realice el promedio de 4 números. Los números podrán ser decimales y serán ingresados por pantalla por el usuario.

---

## Ejercicios sobre Estructuras de Datos

### Ejercicio 9 – Arreglo Booleano

Diseñar un algoritmo que recorra las butacas de una sala de cine y determine cuántas butacas desocupadas hay en la sala. Suponga que inicialmente tiene un array (arreglo) con valores booleanos que si es verdadero(verdadero) implica que está ocupada y si es falso(falso) la butaca está desocupada. Tenga en cuenta que el array deberá ser creado e inicializado al principio del algoritmo.

---

### Ejercicio 10 – Dos Arreglos

Una escuela tiene un total de 3 aulas con la siguiente capacidad:

Identificador Aula	Cantidad de Bancos del Aula
Azul	40
Verde	35
Amarillo	30

Sabiendo la cantidad de bancos de cada aula, el usuario deberá ingresar la cantidad de alumnos inscriptos para cursar tercer grado y el sistema deberá determinar qué aula es la indicada para la cantidad ingresada. La escuela ya sabe que la máxima capacidad de sus aulas es de 40 alumnos, por lo tanto, la cantidad de alumnos inscriptos que ingresa el usuario siempre será un número menor o igual a 40.

Listas necesarias para resolver el problema:

40	35	30
0	1	2

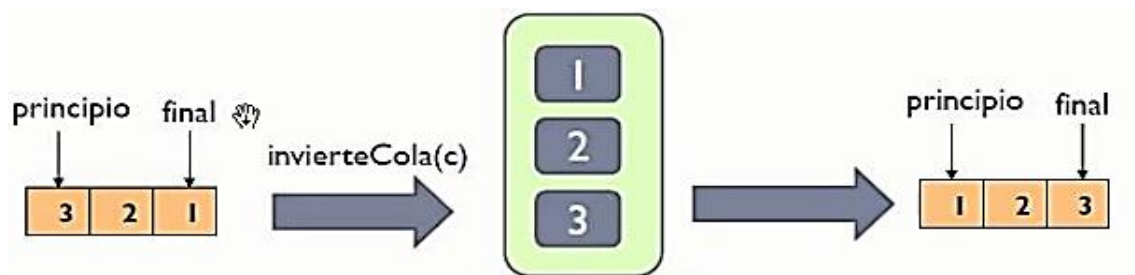
  

Azul	Verde	Amarillo
0	1	2

### Ejercicio 11 – Pila

#### Enunciado:

Diseñar un algoritmo que a partir de una pila inicial de tres elementos devuelva una pila invertida. La pila contiene números enteros como se muestra en la figura. Al comienzo la pila está vacía, se deben apilar los siguientes elementos: 1,2,3 y luego invertir su orden.



## Ejercicios Algoritmos Fundamentales

### Ejercicio 12 – Ordenamiento por Inserción

<https://www.youtube.com/watch?v=5kVQ8kf52K4>

Escribir el pseudocódigo y las pruebas de escritorio para realizar el ordenamiento de un vector con 5 números enteros. El usuario ingresa los números que él desea, cree un vector para guardar temporalmente dichos datos y luego realice el ordenamiento del mismo por inserción

### Ejercicio 13 – Ordenamiento de la Burbuja

<https://www.youtube.com/watch?v=L3d48etbseY>

Escribir el pseudocódigo y las pruebas de escritorio para realizar el ordenamiento de un vector con 5 números enteros. El usuario ingresa los números que él desea, cree un vector para guardar temporalmente dichos datos y luego realice el ordenamiento del mismo a través del método de la burbuja.

### Ejercicio 14 – Ordenamiento por Selección

<https://www.youtube.com/watch?v=I0YwcUJB3vo>

Escribir el pseudocódigo y las pruebas de escritorio para realizar el ordenamiento de un vector con 5 números enteros. El usuario ingresa los números que él desea, cree un vector para guardar



temporalmente dichos datos y luego realice el ordenamiento del mismo a través del método de la burbuja.

---

#### Ejercicio 15 – Búsqueda Secuencial

Escribir el pseudocódigo y las pruebas de escritorio para realizar la búsqueda del nombre de un cliente en un vector que contiene 5 clientes en total. El cliente a buscar será ingresado por pantalla por el usuario. El algoritmo deberá devolver, en caso de que ese nombre exista, la posición en donde se encuentra dicho cliente dentro del vector.

---

## Soluciones Propuestas

### Ejercicios Estructuras de Control

#### Ejercicio 1 – Secuencial

##### Enunciado:

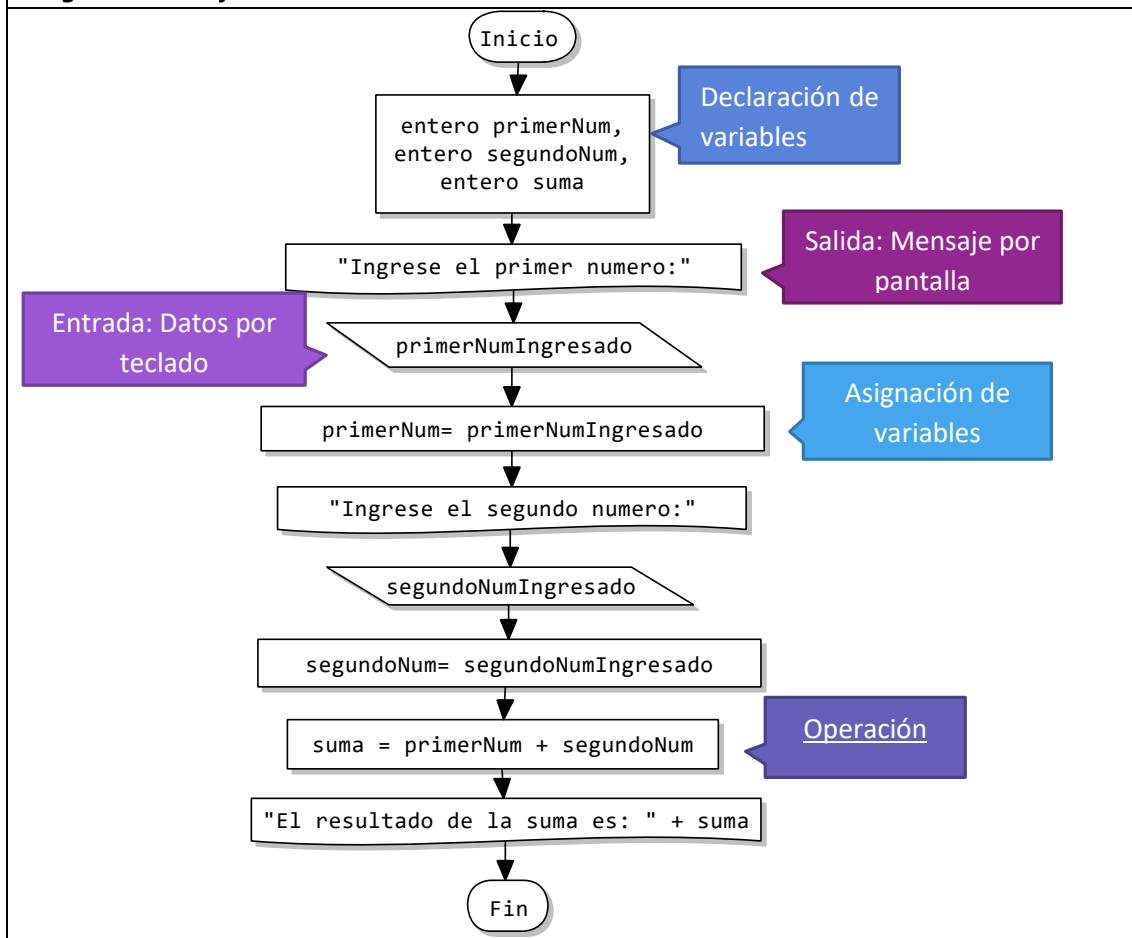
Escribir un algoritmo que permita realizar una suma de dos números enteros. El usuario deberá ingresar primero un número, luego el siguiente número, y el sistema arrojará el resultado correspondiente.

##### Pseudocódigo

```

INICIO
entero primerNum
entero segundoNum
entero suma
IMPRIMIR: "Ingrese el primer número:"
TOMAR: primerNumIngresado
PrimerNum=primerNumIngresado
IMPRIMIR: "Ingrese el segundo número:"
TOMAR: segundoNumIngresado
SegundoNum=segundoNumIngresado
Suma=primerNum+segundoNum
IMPRIMIR: "El resultado de la suma es:" + suma
FIN
  
```

##### Diagrama de Flujo



**Prueba de Escritorio***Identificación de nombres de variables, con su tipo de variable y tipo de dato.*

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Entrada	Entero	primerNumIngresado
Entrada	Entero	segundoNumIngresado
Auxiliar	Entero	primerNum
Auxiliar	Entero	segundoNum
Salida	Entero	Suma

**Ejecución de Pruebas**

N° Prueba	Entrada		Asignación		Operación	Salida
	Primer Num Ingresado	Segundo Num Ingresado	primer Num	segundo Num	suma	Mensaje
1	20	30	20	30	20 + 30=50	"El resultado de la suma es:" + 50
2	15	150	15	150	15 + 150 =165	"El resultado de la suma es:" + 165
3	130	300	130	300	130 + 300=430	"El resultado de la suma es:" + 430

## Ejercicio 2 – Alternativa Simple

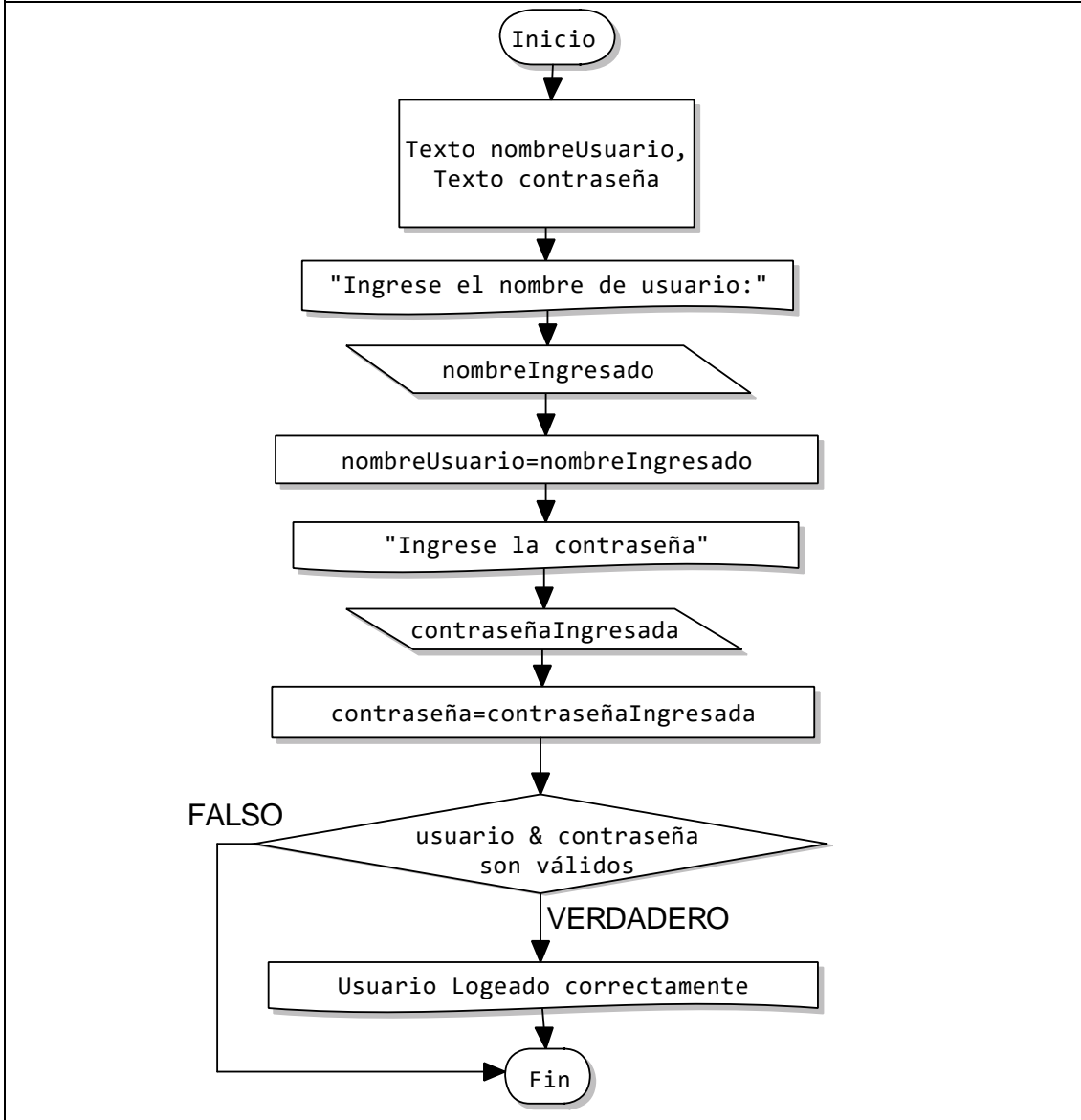
**Enunciado**

Escribir un algoritmo que permita loguearse (registrarse) a un sistema, ingresando un nombre de usuario y la contraseña adecuada. Considerar que tanto el usuario como la contraseña están formados sólo por letras. El sistema deberá validar que el usuario y la contraseña sean correctas, comparándolas con lo que el sistema tiene registrado para ese usuario.

*\*\*Aclaración, en los sistemas reales, el inicio de sesión es mucho más complejo que lo que se muestra a continuación. Se ha simplificado el proceso, abstrayendo la validación a una función denominada esValido() que resuelve la verificación del usuario y su contraseña.*

**Pseudocódigo**

```
INICIO
Texto nombreUsuario
Texto contraseña //Suponiendo que la contraseña es sólo de caracteres.
IMPRIMIR: "Ingrese el nombre de usuario:"
TOMAR: nombreIngresado
nombreUsuario=nombreIngresado;
IMPRIMIR: "Ingrese la contraseña"
TOMAR: contraseñaIngresada
contraseña=contraseñaIngresada
SI(esValido(usuario) && esValido(contraseña))
ENTONCES
IMPRIMIR: "Usuario logeado con éxito".
FIN SI
FIN
```

**Diagrama de Flujo****Prueba de Escritorio**

*Identificación de nombres de variables, con su tipo de variable y tipo de dato.*

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Entrada	Texto	nombreIngresado
Entrada	Texto	contraseñaIngresada
Auxiliar	Texto	nombreUsuario
Auxiliar	Texto	contraseña
Salida	Mensaje	"Usuario Logeado correctamente"

**Ejecución de Pruebas**

N° Prue ba	Entrada		Asignación		Condición	Salida
	nombre Ingresado	contraseña Ingresada	nombre Usuario	contraseña	Usuario & contraseña son validos	Mensaje
1	Juan	Pokemon	Juan	Pokemon	Verdadero	“Usuario Logeado correctamente”
2	Julieta	Pikachu	Julieta	Pikachu	Verdadero	“Usuario Logeado correctamente”
3	Andrea	NoSoyFanDe Pokemon	Andrea	NoSoyFanDe Pokemon	Falso	

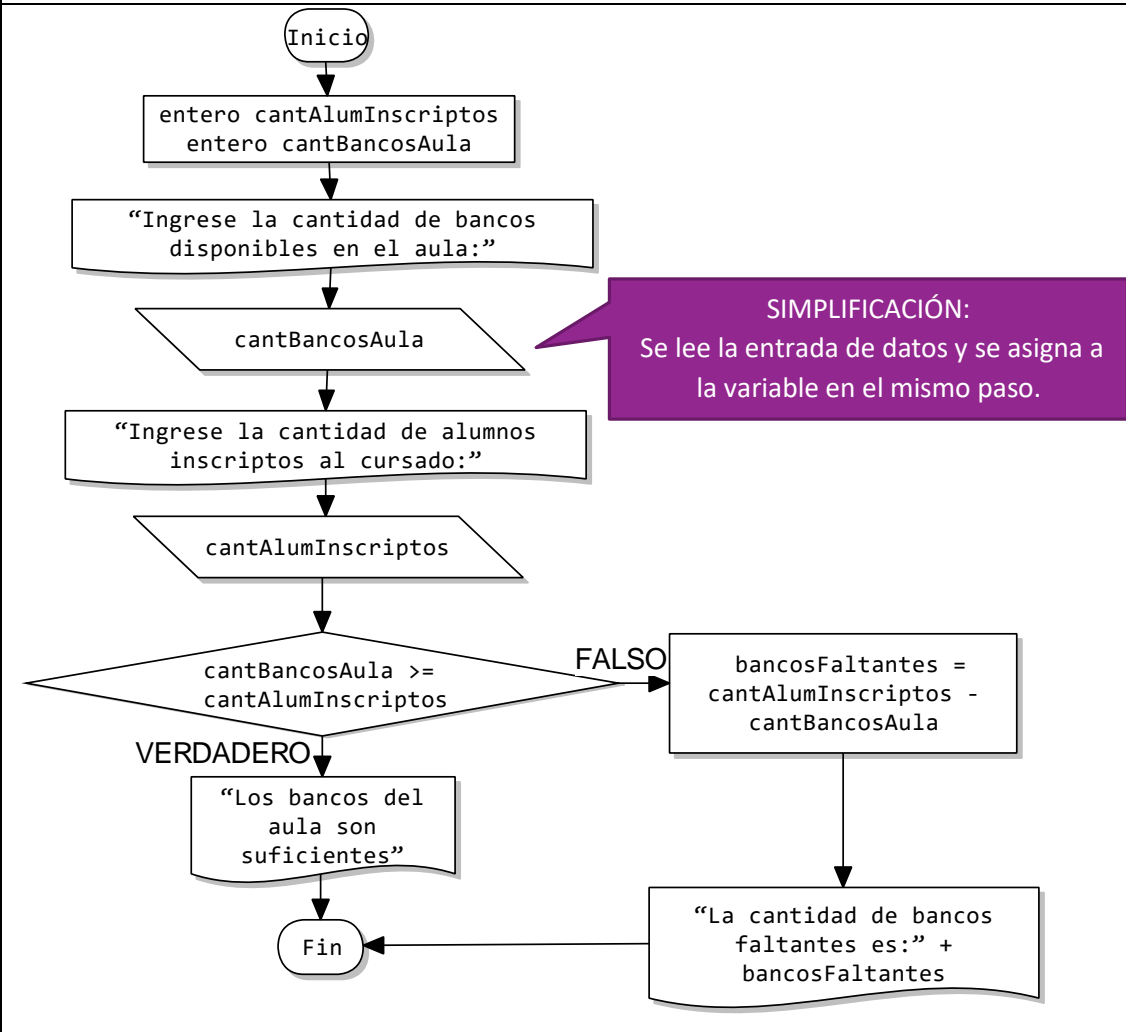
## Ejercicio 3 – Alternativa Doble

**Enunciado**

Escribir el algoritmo que, a partir de la cantidad de bancos de un aula y la cantidad de alumnos inscriptos para un curso, permita determinar si alcanzan los bancos existentes. De no ser así, informar además cuantos bancos sería necesario agregar. El usuario deberá ingresar por teclado tanto la cantidad de bancos que tiene el aula, como la cantidad de alumnos inscriptos para el curso.

**Pseudocódigo**

```
INICIO
entero cantBancosAula
entero cantAlumInscriptos
entero bancosFaltantes
IMPRIMIR: "Ingrese la cantidad de bancos disponibles en el aula:"
TOMAR Y ASIGNAR: cantBancosAula;
IMPRIMIR: "Ingrese la cantidad de alumnos inscriptos al cursado:"
TOMAR Y ASIGNAR: cantAlumInscriptos;
SI (cantBancosAula >= cantAlumInscriptos)
    ENTONCES: IMPRIMIR: "Los bancos del aula son suficientes".
SINO
    bancosFaltantes = cantAlumInscriptos - cantBancosAula
    IMPRIMIR: "La cantidad de bancos faltantes es:" + bancosFaltantes.
FIN SI
FIN
```

**Diagrama de Flujo****Prueba de Escritorio**

**Identificación de nombres de variables, con su tipo de variable y tipo de dato.**

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Entrada	Entero	cantAlumInscriptos
Entrada	Entero	cantBancosAula
Salida	Entero	bancosFaltantes



**Ejecución de Pruebas**

N° Prueba	Entrada	Bloque de decisión	Salida	
	cantAlum Inscriptos	cantBancosAula >= cantAlumInscriptos	BancosFaltantes	Mensaje
1	50	38 >= 50: NO	50 - 38 = 12	"La cantidad de bancos faltantes es:" + 12
2	45	35 >= 40: NO	40 - 35 = 5	"La cantidad de bancos faltantes es:" + 5
3	35	38 >= 35: SI		"Los bancos del aula son suficientes".
4	veinte	35 >= veinte	Error de tipo de dato. Esperando un valor de tipo Entero.	
5	38	30 >= 30: SI		"Los bancos del aula son suficientes".

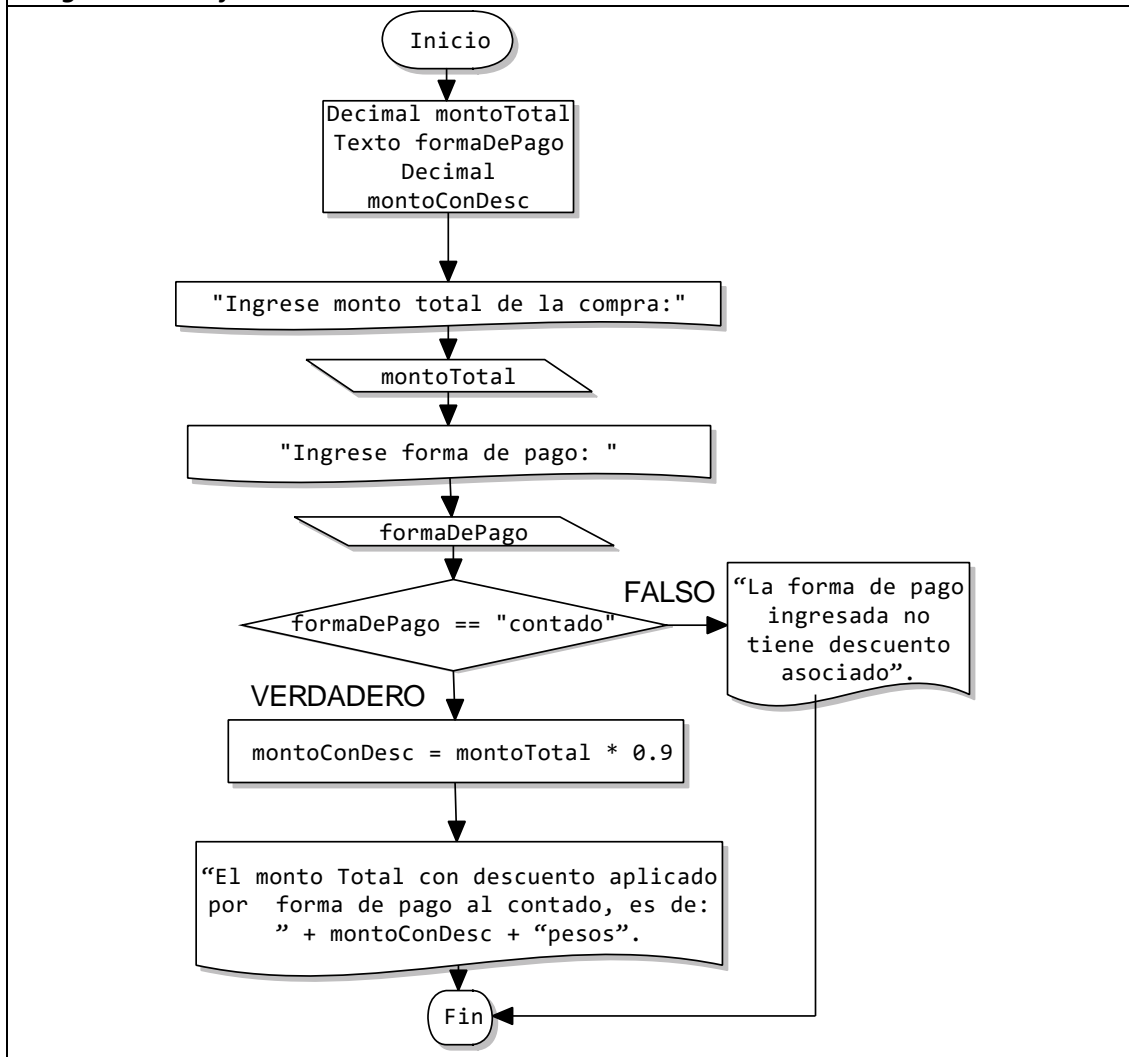
## Ejercicio 4 – Alternativa Doble

**Enunciado**

Diseñar un algoritmo que permita aplicar un descuento del 10% al monto total de una compra si la forma de pago empleada es de contado. El usuario deberá ingresar el monto de la compra realizada y la forma de pago utilizada. Si es contado, deberá aplicar el descuento, sino se deberá mostrar un mensaje informando que para dicha forma de pago no tiene descuento.

**Pseudocódigo**

```
INICIO
Decimal montoTotal
Texto formaDePago
Decimal montoConDesc
IMPRIMIR: "Ingrese monto total de la compra"
TOMAR Y ASIGNAR montoTotal;
IMPRIMIR: "Ingrese forma de pago"
TOMAR Y ASIGNAR formaDePago;
SI (formaDePago == "contado")
    montoConDesc = montoTotal * 0.9
    IMPRIMIR: "El monto Total con descuento aplicado por forma de pago al
contado, es de:" + montoConDesc + "pesos".
SINO
    IMPRIMIR: "La forma de pago ingresada no tiene descuento asociado".
FIN SI.
FIN
```

**Diagrama de Flujo****Prueba de Escritorio**

*Identificación de nombres de variables, con su tipo de variable y tipo de dato.*

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Entrada	Decimal	MontoTotal
Entrada	Texto	formaDePago
Salida	Decimal	montoConDesc

**Ejecución de Pruebas**

N° Prueba	Entrada		Bloque de Decisión	Operación	Salida
	Monto Total	Forma DePago	FormaDePago == "contado"	Monto ConDesc	Mensaje
1	1320	"tarjeta de crédito"	"tarjeta de crédito" == "contado"		"La forma de pago ingresada no tiene descuento asociado".

<b>2</b>	400	"tarjeta de débito"	"tarjeta de débito"=="contado"		"La forma de pago ingresada no tiene descuento asociado".
<b>3</b>	1320	"contado"	"contado"=="contado"	$1320 * 0.9 = 1188$	"El monto Total con descuento aplicado por forma de pago al contado, es de:" + 1188 + "pesos".
<b>4</b>	400	"contado"	"contado"=="contado"	$400 * 0.9 = 360$	"El monto Total con descuento aplicado por forma de pago al contado, es de:" + 360 + "pesos".

## Ejercicio 5 – Alternativa Múltiple

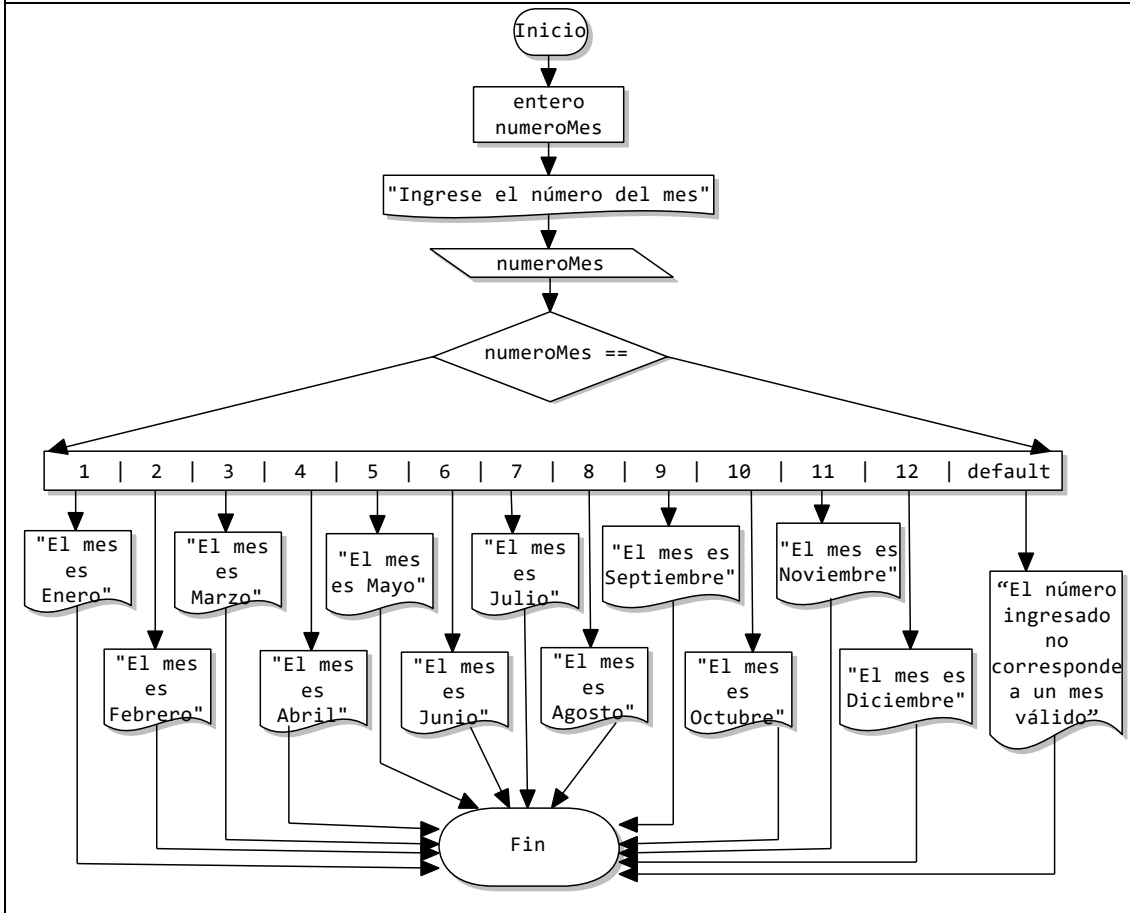
**Enunciado**

Diseñar un algoritmo que devuelva el nombre del mes, a partir del número de mes, ingresado por teclado, por el usuario.

**Pseudocódigo**

```

INICIO
Entero numeroMes;
IMPRIMIR: "Ingrese el número del mes"
TOMAR Y ASIGNAR numeroMes;
Según (numeroMes)
    caso (1): "El mes es Enero"
        salir
    caso (2): "El mes es Febrero"
        salir
    caso (3): "El mes es Marzo"
        salir
    caso (4): "El mes es Abril"
        salir
    caso (5): "El mes es Mayo"
        salir
    caso (6): "El mes es Junio"
        salir
    caso (7): "El mes es Julio"
        salir
    caso (8): "El mes es Agosto"
        salir
    caso (9): "El mes es Septiembre"
        salir
    caso (10): "El mes es Octubre"
        salir
    caso (11): "El mes es Noviembre"
        salir
    caso (12): "El mes es Diciembre"
        salir
    Defecto: "El número ingresado no corresponde a un mes válido"
//El defecto o default, sirve para el caso en que la variable numeroMes
no corresponda con ninguna de las opciones contempladas (en cada uno de
los casos, en inglés se utiliza la palabra case). De esta manera el switch
(así se denomina en inglés a esta estructura de control) entrará por el
defecto y mostrará el mensaje correspondiente.
FIN
  
```

**Diagrama de Flujo****Prueba de Escritorio**

*Identificación de nombres de variables, con su tipo de variable y tipo de dato.*

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Entrada	Entero	NumeroMes
Salida	Texto	Mensaje

**Ejecución de Pruebas**

N° Prueba	Entrada numeroMes	Bloque de Decisión según(numeroMes), caso	Salida Mensaje
1	11	Según(11), caso 11	"El mes es Noviembre";
2	8	Según(8), caso 8	"El mes es Agosto";
3	12	Según(12), caso 12	"El mes es Diciembre";
4	4	Según(4), caso 4	"El mes es Abril"

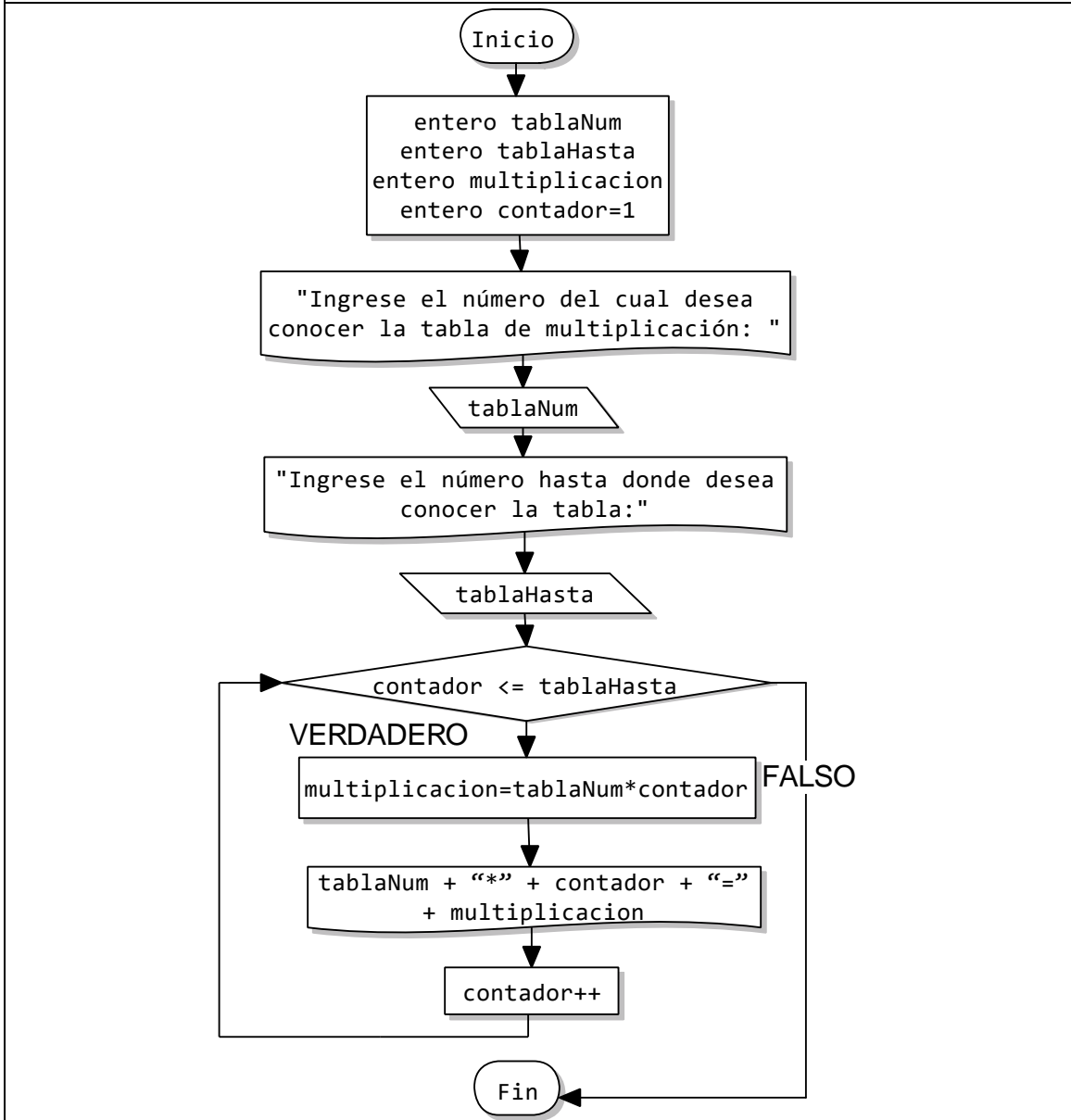
## Ejercicio 6 – Repetitiva Mientras (While)

**Enunciado**

Diseñar un algoritmo que muestre por pantalla la tabla de multiplicación del número que ingrese el usuario. Para definir hasta que número desea que muestre la tabla de multiplicación el usuario también deberá ingresar este valor. La tabla de multiplicación a mostrar debe empezar en la multiplicación por 1.

**Pseudocódigo**

```
INICIO
entero tablaNum;
entero tablaHasta;
entero contador=1;
entero multiplicación
IMPRIMIR: "Ingrese el número del cual desea conocer la tabla de
multiplicación:"
TOMAR Y ASIGNAR tablaNum;
IMPRIMIR: "Ingrese el número hasta donde desea conocer la tabla:"
TOMAR Y ASIGNAR tablaHasta;
MIENTRAS(contador <=tablaHasta)
multiplicacion=tablaNum*contador;
IMPRIMIR: tablaNum + "*" + contador + "=" + multiplicacion
Contador++
FIN MIENTRAS
FIN
```

**Diagrama de Flujo****Prueba de Escritorio**

*Identificación de nombres de variables, con su tipo de variable y tipo de dato.*

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Entrada	Entero	tablaNum
Entrada	Entero	tablaHasta
Auxiliar	Entero	Contador (c)
Salida	Entero	Multiplicación (m)



**Ejecución de Pruebas**

N° Prueba	Entrada		Auxiliar	Bucle Mientras/ While	Operaciones		Salida
	tabla Num	tabla Hasta	contador (c)	while(Contador<= TablaHasta)	m= tablaNum* contador	c++	Mensaje
1.1	8	3	1	1<=3:si	m=8*1=8	2	8*1=8
1.2	8	3	2	2<=3:si	m=8*2=16	3	8*2=16
1.3	8	3	3	3<=3:si	m=8*3=24	4	8*3=24
1.4	8	3	4	4<=3:no ->Fin			
2.1	4	5	1	1<=5:si	m=4*1=4	2	4*1=4
2.2	4	5	2	2<=5:si	m=4*2=8	3	4*2=8
2.3	4	5	3	3<=5:si	m=4*3=12	4	4*3=12
2.4	4	5	4	4<=5:no	m=4*4=16	5	4*4=16
2.5	4	5	5	5<=5:si	m=4*5=20	6	4*4=20
2.6	4	5	6	6<=5:no ->Fin			

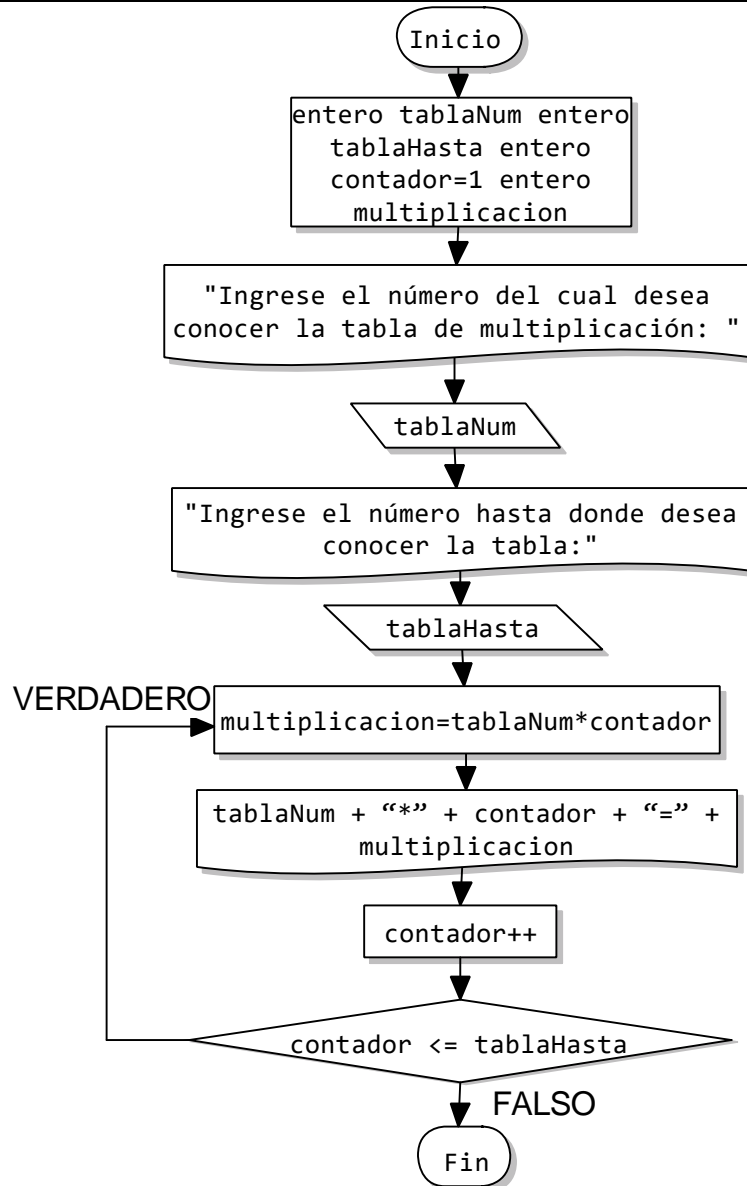
## Ejercicio 7 – Repetitiva Hacer Mientras (Do While)

**Enunciado**

Diseñar un algoritmo que muestre por pantalla la tabla de multiplicación del número que ingrese el usuario. Para definir hasta que número desea que muestre la tabla de multiplicación el usuario también deberá ingresar este valor. La tabla de multiplicación a mostrar debe empezar en la multiplicación por 1.

**Pseudocódigo**

```
INICIO
entero tablaNum;
entero tablaHasta;
entero contador=1;
entero multiplicación;
IMPRIMIR: "Ingrese el número del cual desea conocer la tabla de
multiplicación:"
TOMAR Y ASIGNAR tablaNum
IMPRIMIR: "Ingrese el número hasta donde desea conocer la tabla:"
TOMAR Y ASIGNAR tablaHasta
HACER
multiplicación=tablaNum*contador
IMPRIMIR: tablaNum + "*" + contador + "=" + multiplicación}
Contador++;
MIENTRAS(contador <=tablaHasta)
FIN HACER
FIN
```

**Diagrama de Flujo****Prueba de Escritorio****Identificación de variables de entrada, tipos de variables y tipo de datos**

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Entrada	Entero	tablaNum
Entrada	Entero	tablaHasta
Auxiliar	Entero	Contador (c)
Salida	Entero	Multiplicación (m)

**Ejecución de Pruebas**

N° Prueba	Entrada		Auxiliar	Operaciones del bloque Hacer Mientras / Do While			
				Do			While
	tabla Num	tabla Hasta	contador (c)	m= tablaNum* contador	c++	Mensaje	(Contador<= TablaHasta)
1.1	8	3	1	m=8*1=8	2	8*1=8	1<=3:si
1.2	8	3	2	m=8*2=16	3	8*2=16	2<=3:si
1.3	8	3	3	m=8*3=24	4	8*3=24	3<=3:si
1.4	8	3	4				4<=3:no ->Fin
2.1	4	5	1	m=4*1=4	2	4*1=4	1<=5:si
2.2	4	5	2	m=4*2=8	3	4*2=8	2<=5:si
2.3	4	5	3	m=4*3=12	4	4*3=12	3<=5:si
2.4	4	5	4	m=4*4=16	5	4*4=16	4<=5:no
2.5	4	5	5	m=4*5=20	6	4*4=20	5<=5:si
2.6	4	5	6				6<=5:no ->Fin

## Ejercicio 8 – Repetitiva Para (For)

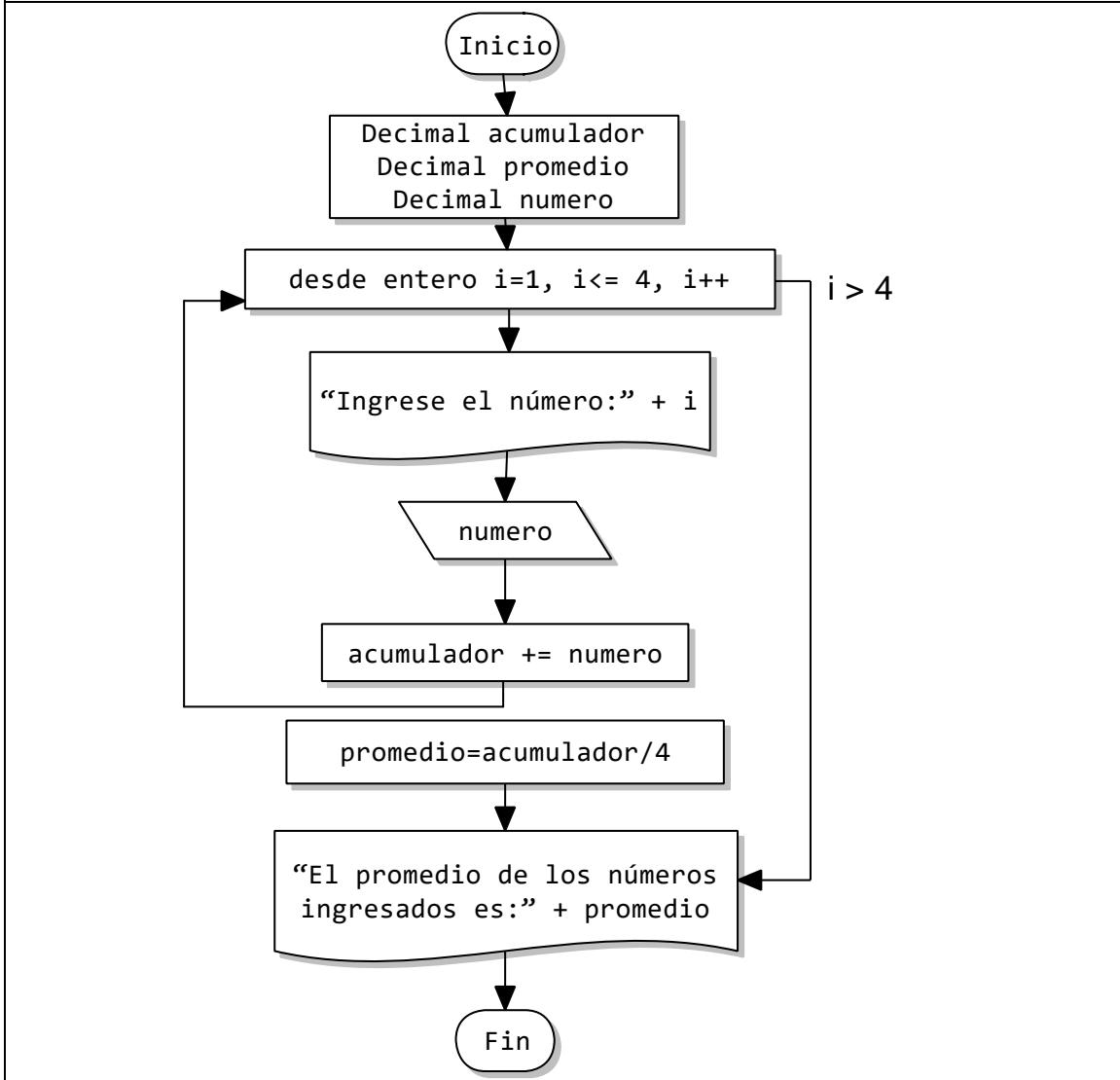
**Enunciado**

Diseñar un algoritmo que realice el promedio de 4 números. Los números podrán ser decimales y serán ingresados por pantalla por el usuario.

**Pseudocódigo**

```

INICIO
Decimal acumulador
Decimal promedio
Decimal numero
PARA(entero i=1, i<= 4, i++)
    IMPRIMIR: "Ingrese el número:" + i
    TOMAR numero
    acumulador += numero
FIN PARA
promedio=acumulador/4
IMPRIMIR: "El promedio de los números ingresados es:" + promedio
FIN
  
```

**Diagrama de Flujo**

**Prueba de Escritorio****Identificación de nombres de variables, con su tipo de variable y tipo de dato.**

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Auxiliar	Decimal	acumulador (a)
Salida	Decimal	promedio(p)
Salida	Texto	mensaje

**Ejecución de Pruebas**

N° Prueba	Entrada	Ciclo For/Para	Auxiliar	Salida	
	Numero	i<=4	acumulador a+=numero	promedio p=a/4	Mensaje
Prueba 1	20.6	1<=4: si	a=20.6		El promedio de los números ingresados es: 22.7
	11.4	2<=4: si	a=32		
	8	3<=4: si	a=40		
	50.8	4<=4: si	a=90.8		
		5<=4: no, Fin For		P=90.8/4=22.7	
Prueba 2	28	1<=4: si	a=28		El promedio de los números ingresados es: 127.5
	100.40	2<=4: si	a=128.40		
	80.90	3<=4: si	a=209.3		
	300.70	4<=4: si	a=510		
		5<=4: no, Fin For		P=510/4=127.5	

## Ejercicios sobre Estructuras de Datos

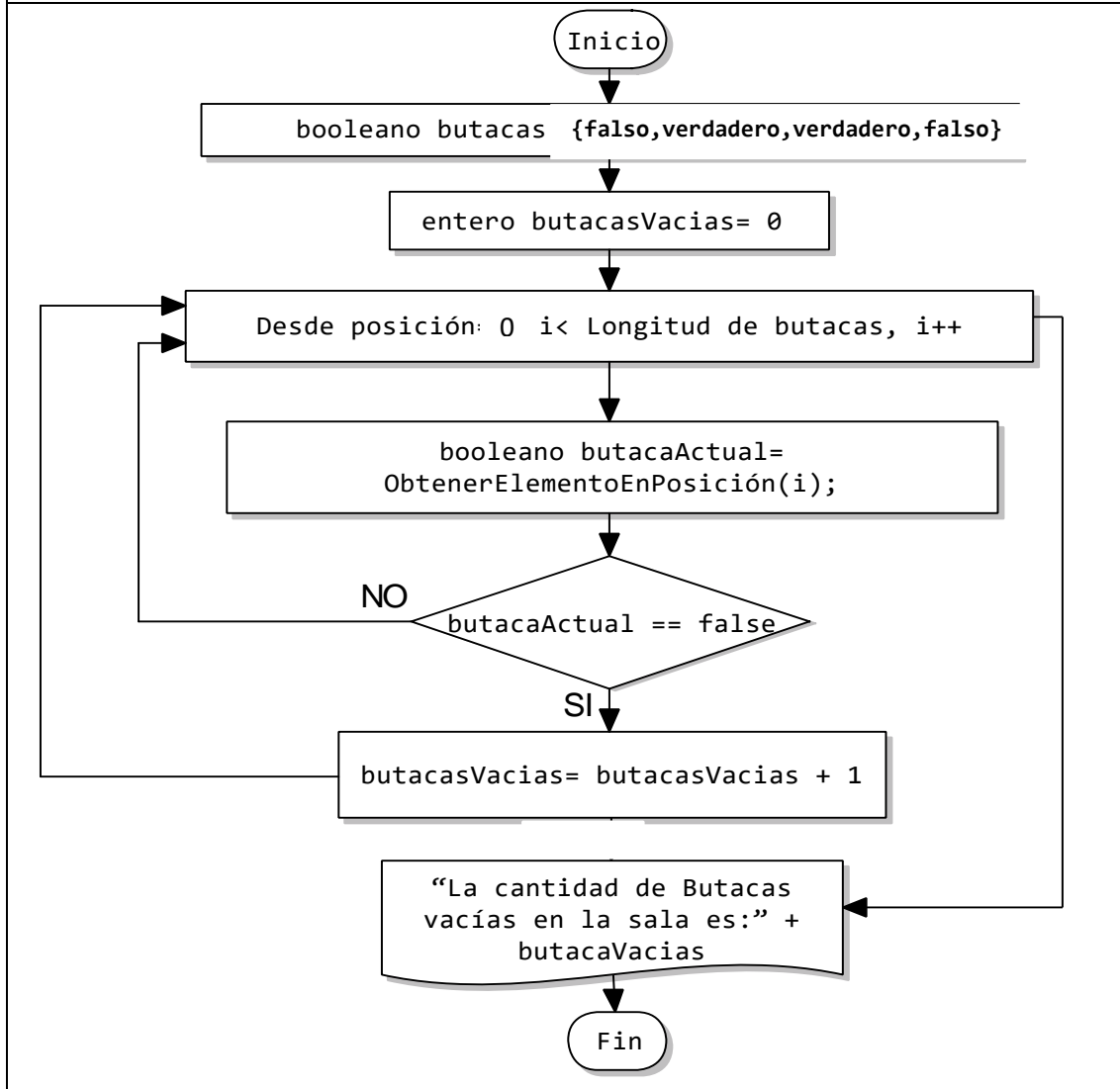
### Ejercicio 9 – Arreglo Booleano

**Enunciado:**

Diseñar un algoritmo que recorra las butacas de una sala de cine y determine cuántas butacas desocupadas hay en la sala. Suponga que inicialmente tiene un array (arreglo) con valores booleanos que si es verdadero(verdadero) implica que está ocupada y si es falso(falso) la butaca está desocupada. Tenga en cuenta que el array deberá ser creado e inicializado al principio del algoritmo.

**Pseudocódigo**

```
INICIO
Booleano butacas[] = {falso,verdadero,verdadero,falso}
entero butacasVacías =0 //Contador que guarda la cantidad de butacas
vacías.
PARA (entero i=0, i< butacas.lenght(), i++)
Booleano butacaActual= Obtener(butacas, i);
SI (butacaActual == falso)
    butacasVacías++; // suma 1 al valor de la variable butacasVacías.
FIN SI
FIN PARA
IMPRIMIR: "La cantidad de Butacas vacías en la sala es:" + butacasVacías
FIN
```

**Diagrama de Flujo****Prueba de Escritorio**

Identificación de nombre de variables, con su tipo de variable y tipo de datos

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Auxiliar	Array Booleano	Butacas
Auxiliar	Booleano	ButacaActual (ba)
Auxiliar y de Salida	entero	ButacasVacías (bv)



**Ejecución de Pruebas**

N° Prueba	Auxiliar	Ciclo For / Para		Bloque de Decisión	Auxiliar contador	Salida
	butacas	l<longitud butacas	i, ba	ba==falso	bv	Mensaje
Prueba 1	[falso, falso, verdadero, verdadero]	0<4: si	0, falso	falso == falso	Bv=0+1=1	"La cantidad de Butacas vacías en la sala es:" + 2
		1<4: si	1, falso	falso == falso	Bv=1+1=2	
		2<4: si	2, verdadero	verdadero == falso	Bv=2	
		3<4: si	3, verdadero	verdadero == falso	Bv=2	
		4<4: no, Fin For				
Prueba 2	[verdadero, verdadero, verdadero, falso]	0<4: si	0, verdadero	verdadero == falso	Bv=0	"La cantidad de Butacas vacías en la sala es:" + 1
		1<4: si	1, verdadero	verdadero == falso	Bv=0	
		2<4: si	2, verdadero	verdadero == falso	Bv=0	
		3<4: si	3, falso	falso == falso	Bv=0+1=1	
		4<4: no, Fin For				

## Ejercicio 10 – Dos Arreglos

Una escuela tiene un total de 3 aulas con la siguiente capacidad:

Identificador Aula	Cantidad de Bancos del Aula
Azul	40
Verde	35
Amarillo	30

Sabiendo la cantidad de bancos de cada aula, el usuario deberá ingresar la cantidad de alumnos inscriptos para cursar tercer grado y el sistema deberá determinar qué aula es la indicada para la cantidad ingresada. La escuela ya sabe que la máxima capacidad de sus aulas es de 40 alumnos, por lo tanto, la cantidad de alumnos inscriptos que ingresa el usuario siempre será un número menor o igual a 40.

Listas necesarias para resolver el problema:

Azul	Verde	Amarillo	40	35	30
0	1	2	0	1	2

**Pseudocódigo**

INICIO

Texto listaColoresAulas[] = ["Azul", "Verde", "Amarillo"]

entero listaCapacAulas[] = [40, 35, 30]

entero cantAlumIns

IMPRIMIR: "Ingrese la cantidad de alumnos inscriptos al cursado:"

TOMAR Y ASIGNAR: cantAlumIns

entero capacidadAulaAux = Obtener(listaCapacAulas, 0) //Inicialización.

PARA (entero i=1, i< listaCapacAulas.lenght(), i++)

entero capacidadAulaActual = Obtener (listaCapacAulas, i);

SI (capacidadAulaActual >= cantAlumIns && capacidadAulaActual < capacidadAulaAux)

capacidadAulaAux = capacidadAulaActual;

entero indiceAulaAux = i;

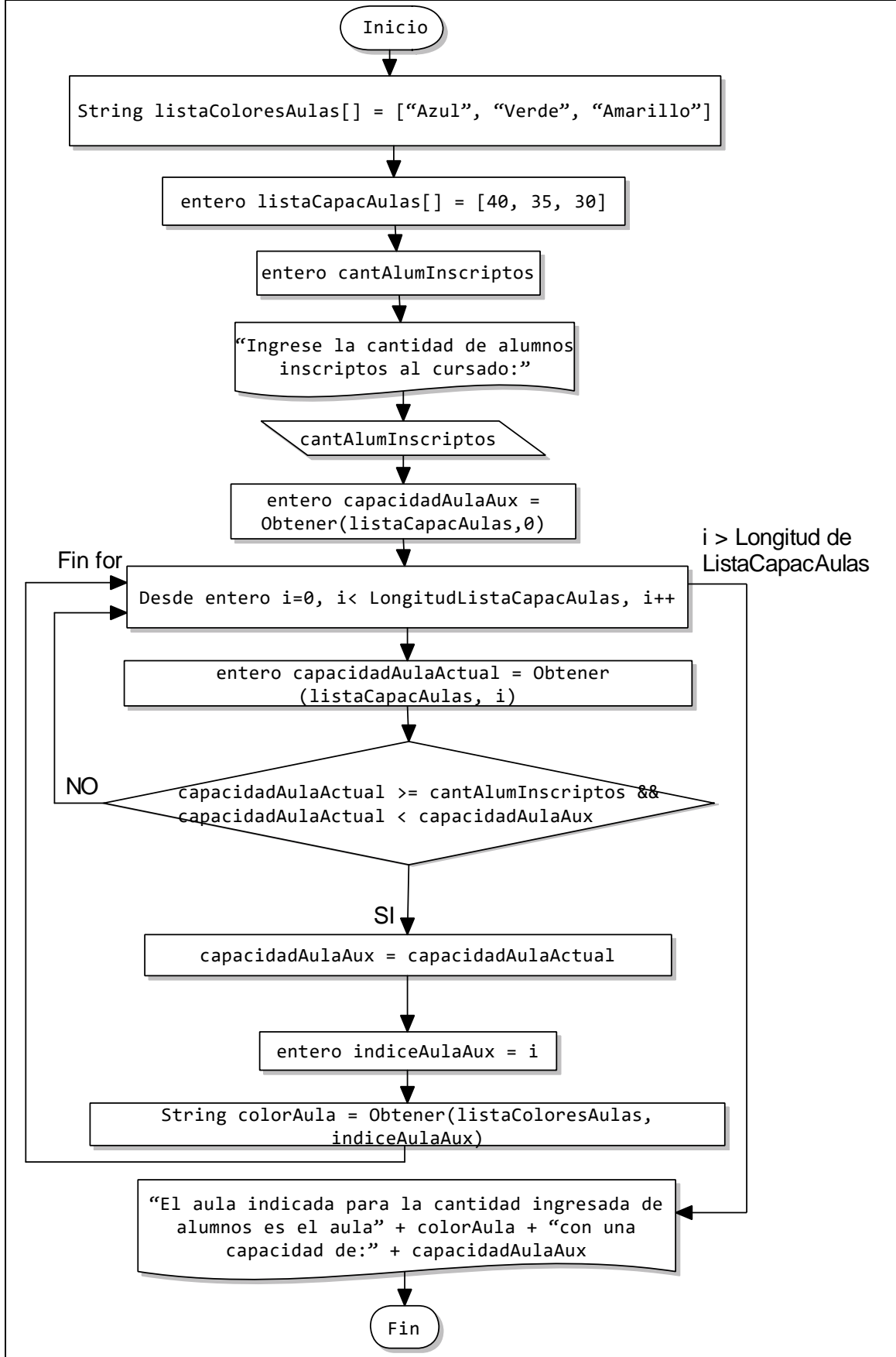
FIN SI

Fin PARA

Texto colorAula = Obtener(listaColoresAulas, indiceAulaAux);

IMPRIMIR: "El aula indicada para la cantidad ingresada de alumnos es el aula" + colorAula + "con una capacidad de:" + capacidadAulaAux;

FIN

**Diagrama de Flujo**

**Prueba de Escritorio**

Identificación de nombre de variables, con su tipo de variable y tipo de datos

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Auxiliar	Array Entero	listaCapacAulas= [40, 35, 30]
Auxiliar	Entero	listaColoresAulas= ["Azul", "Verde", "Amarillo"]
Auxiliar y de Salida	Entero	capacidadAulaAux
Auxiliar	Entero	capacidadAulaActual
Auxiliar	Entero	indiceAulaAux
Salida	Texto	colorAula
Entrada	Entero	cantAlumIns

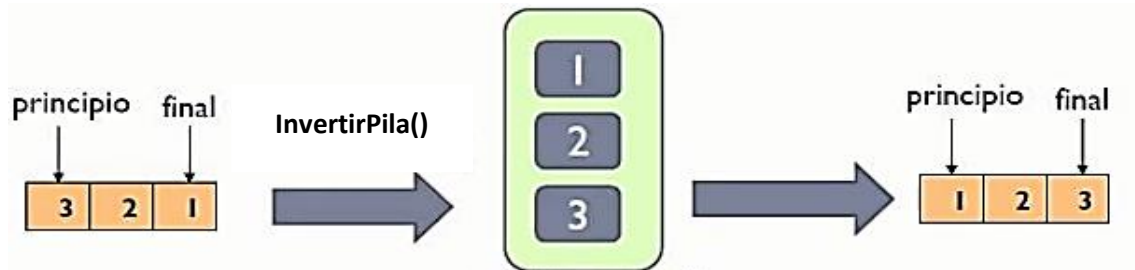
**Ejecución de Pruebas**

N° de Prueba	Entrada	Auxiliares	Array: Ciclo For		Bloque de decisión	Auxiliares	Salida
	cantAlumIns	Capacidad AulaAux	i	Capacidad AulaActual	capacidadAulaActual >= cantAlumIns && capacidadAulaActual < capacidadAulaAux	indiceAulaAux	colorAula
1. a	30	40	1	35	¿35 >= 30 y 35 < 40? -> SI; <b>capacidadAulaAux=35</b>	1	
1. b		35	2	30	¿30 >= 30 y 30 < 35? -> SI; <b>capacidadAulaAux = 30</b>	2	Amarillo
2. a	35	40	1	35	¿35 >= 35 y 35 < 40? -> SI; <b>capacidadAulaAux= 35</b>	1	
2. b		35	2	30	¿30 >= 35 y 30 < 35? -> NO	1	Verde

## Ejercicio 11 – Pila

**Enunciado:**

Diseñar un algoritmo que a partir de una pila inicial de tres elementos devuelva una pila invertida de dichos elementos. La pila inicial se encuentra vacía, usted deberá apilar los elementos y mostrar la pila original. Luego invertir los elementos, y mostrar la nueva pila invertida.

**Pseudocódigo****Alternativa 1)**

```

INICIO
Pila de datos enteros pilaOriginal;
Pila de datos enteros pilaInvertida;
For (i=1, i<=3, i++)
    apilar (pilaOriginal, i);
Fin For
IMPRIMIR "Pila Inicial:" + pilaOriginal
For (i=1, i<=3, i++)
    Apilar (pilaInvertida, Desapilar(pilaOriginal))
Fin For
FIN

```

**Alternativa 2)**

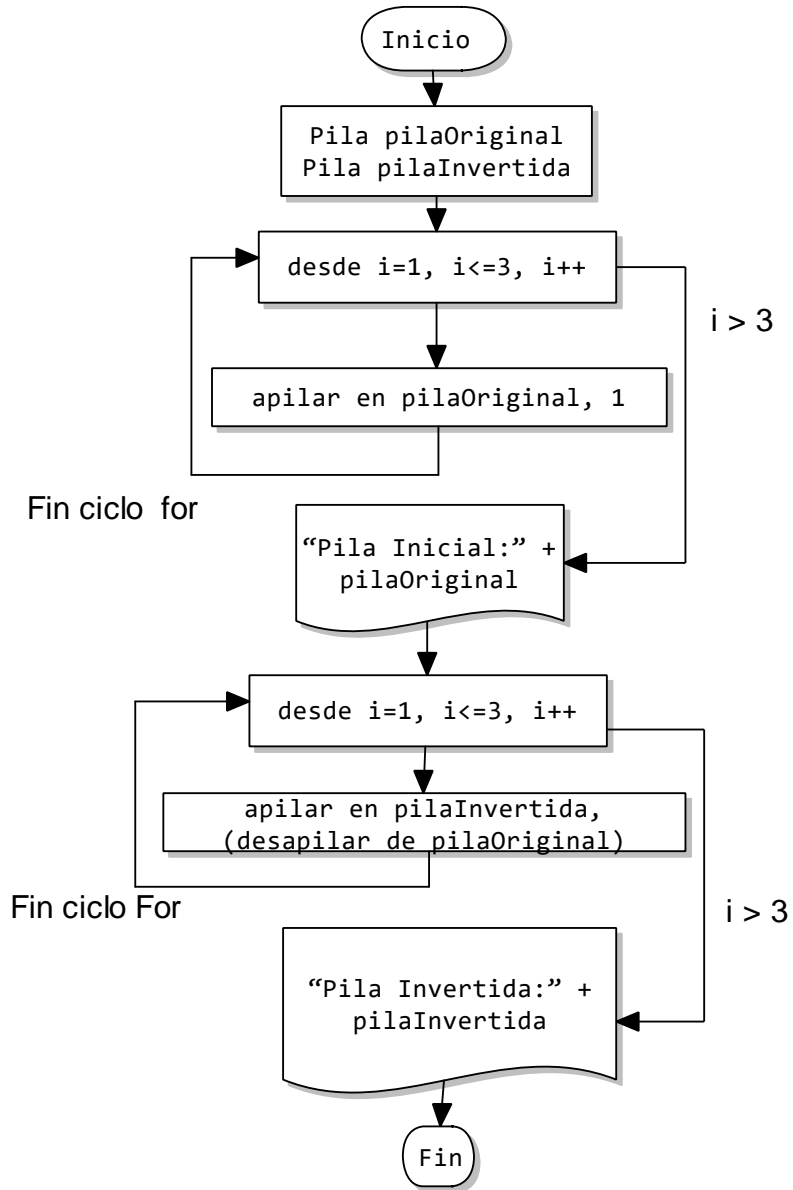
```

INICIO
Pila de datos enteros pilaOriginal;
Pila de datos enteros pilaInvertida;
apilar (pilaOriginal, 1);
apilar (pilaOriginal, 2);
apilar (pilaOriginal, 3);
IMPRIMIR "Pila Inicial:" + pilaOriginal
entero principio= desapilar(pilaEnteros);//principio=3
apilar (pilaInvertida,principio);//el elemento "principio" pasa a ser "final"
entero medio= desapilar(pilaEnteros);//medio=2
apilar(pilaInvertida, medio);//el elemento "medio" seguirá siendo "medio"
entero final = desapilar(pilaEnteros);//final=1
apilar(pilaInvertida, final);//el elemento "final" pasa a ser "principio"
IMPRIMIR: "Pila Invertida:" + pilaInvertida
FIN

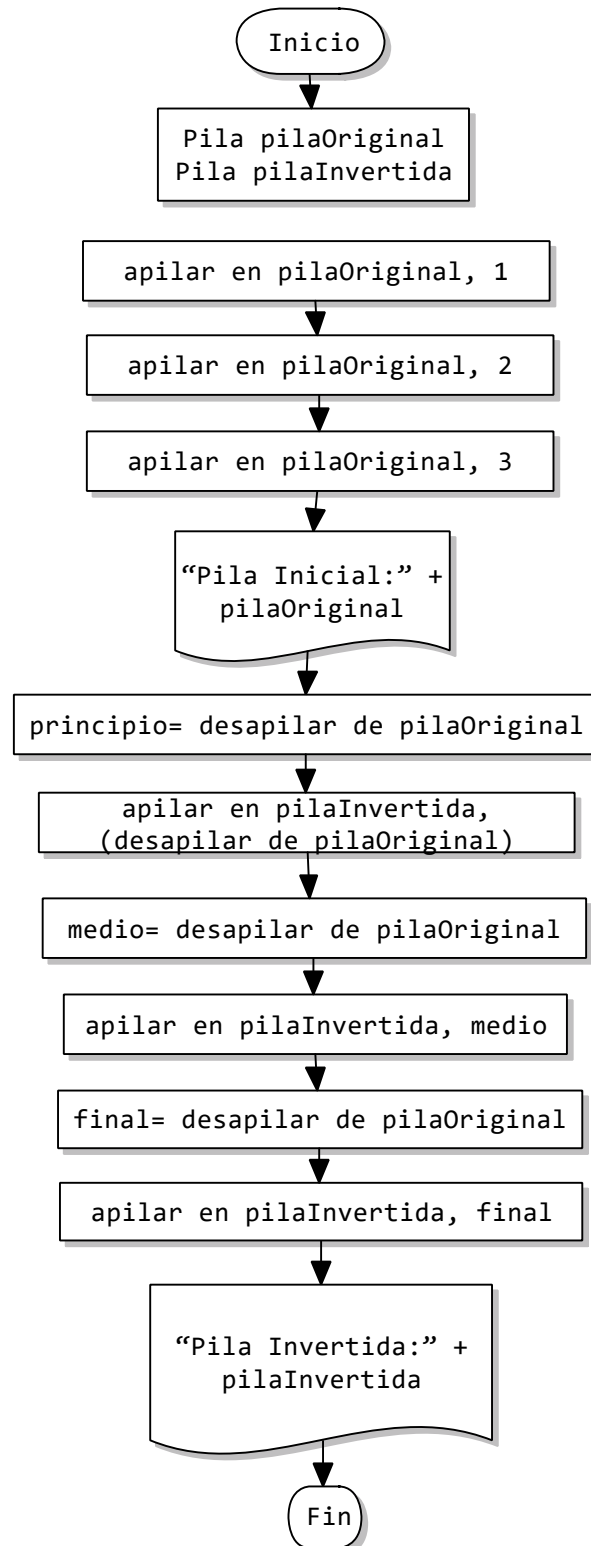
```

**Diagrama de Flujo**

**Alternativa 1)**



Alternativa 2)



**Prueba de Escritorio para alternativa 1)**

Identificación de nombre de variables, con su tipo de variable y tipo de datos

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Auxiliar	Pila de Enteros	PilaOriginal
Salida	Pila de Enteros	PilaInvertida

**Ejecución de Pruebas**

N° Prueba	1er Ciclo For			2do Ciclo For		
	i	Apilar en Pila Original	Pila Original	i	Apilar en Pila Invertida	Salida Pila Invertida
Prueba 1	i=1	Apilar(pilaOriginal, 1)	[3] [2] [1]	i=1	Apilar(pilaInvertida, Desapilar(pilaOriginal))	[1] [2] [3]
	i=2	Apilar(pilaOriginal, 2)		i=2	Apilar(pilaInvertida, Desapilar(pilaOriginal))	
	i=3	Apilar(pilaOriginal, 3)		i=3	Apilar(pilaInvertida, Desapilar(pilaOriginal))	

**Prueba de Escritorio para alternativa 2)**

Identificación de nombre de variables, con su tipo de variable y tipo de datos

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Auxiliar	Pila de Enteros	PilaOriginal
Auxiliar	entero	principio
Auxiliar	entero	medio
Auxiliar	entero	final
Salida	Pila de Enteros	PilaInvertida

**Ejecución de Pruebas**

Id.	Operaciones				Salida
	Apilar en Pila Original	Pila Original	Desapilar de Pila Original	Apilar en Pila Invertida	Pila Invertida
Prueba 1	Apilar1	[3]	Principio=11	Apilar(Principio)	[3]
	Apilar2	[2]	Medio=10	Apilar(Medio)	[2]
	Apilar3	[1]	Final=9	Apilar(Final)	[1]



## Ejercicios Algoritmos Fundamentales

### Ejercicio 12 – Ordenamiento por Inserción

#### Enunciado:

<https://www.youtube.com/watch?v=5kVQ8kf52K4>

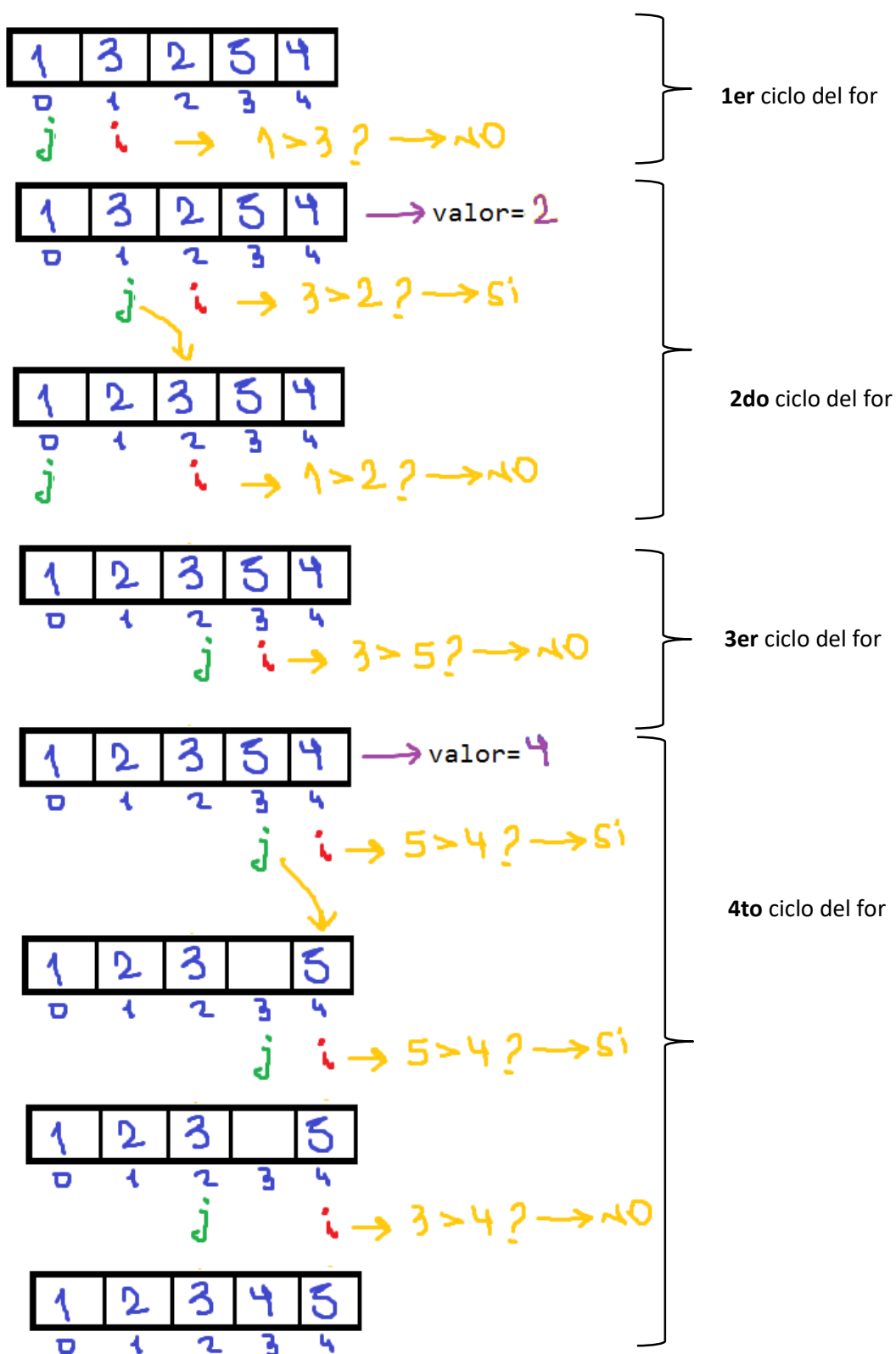
Escribir el pseudocódigo y las pruebas de escritorio para realizar el ordenamiento de un vector con 5 números enteros. El usuario ingresa los números que él desea, cree un vector para guardar temporalmente dichos datos y luego realice el ordenamiento del mismo por inserción

#### Pseudocódigo

```

INICIO
Lista entero listaNum;
//Bucle para la toma de datos
FOR (i=1, i<=5, i++)
    IMPRIMIR: "Ingrese un número";
    ENTERO numero;
    TOMAR Y ASIGNAR: numero;
    listaNum.insertar(i, numero); //Se inserta el elemento: numero en la
    posición i de la lista: listaNum
FIN FOR
IMPRIMIR: "La lista formada es:" + listaNum;
FOR (entero i = 1; i < longitud(listaNum); i++);
    entero valor = listaNum[i]
    entero j = i-1
    MIENTRAS (j >= 0 && listaNum[j] > valor)
        HACER:
            listaNum[j+1] = listaNum[j]
            j--
    FIN_MIENTRAS
    listaNum[j+1] = valor
FIN FOR
IMPRIMIR: "La lista ordenada es:" + listaNum;
FIN
  
```

## Prueba de Escritorio



## Ejercicio 13 – Ordenamiento de la Burbuja

**Enunciado:**

<https://www.youtube.com/watch?v=L3d48etbseY>

Escribir el pseudocódigo y las pruebas de escritorio para realizar el ordenamiento de un vector con 5 números enteros. El usuario ingresa los números que él desea, cree un vector para guardar temporalmente dichos datos y luego realice el ordenamiento del mismo a través del método de la burbuja.

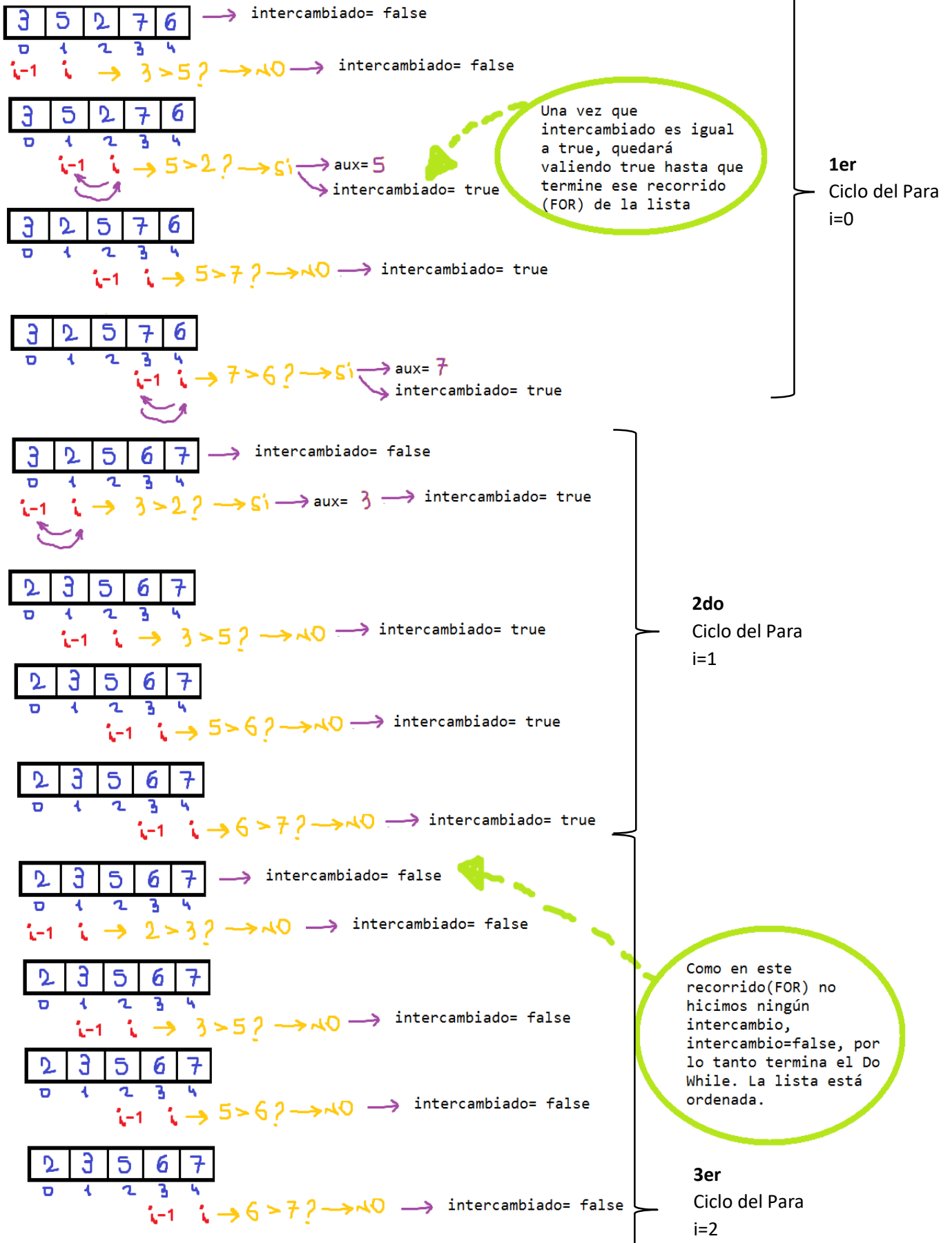
**Pseudocódigo**

```

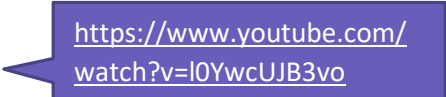
INICIO
Lista entero listaNum;
PARA (i=1, i<=5, i++)
    IMPRIMIR: "Ingrese un número";
    entero numero;
    TOMAR Y ASIGNAR: numero;
    listaNum.insertar(i, numero);
FIN PARA
IMPRIMIR: "La lista formada es:" + listaNum;
HACER:
    Booleano enteroercambiado = falso
    PARA (entero i = 1; i < n; i++)
        //si este par no está ordenado
        SI (listaNum[i-1] > listaNum[i])
            //los intercambiamos y recordamos que algo ha cambiado
            ENTERO aux = listaNum[i-1]
            listaNum[i-1] = listaNum[i]
            listaNum[i] = aux
            intercambiado = verdadero
        FIN_SI
    FIN_PARA
MIENTRAS:(intercambiado == verdadero)
IMPRIMIR: "La lista ordenada es:" + listaNum;
FIN

```

**Prueba de Escritorio**



## Ejercicio 14 – Ordenamiento por Selección


<https://www.youtube.com/watch?v=I0YwcUJB3vo>
**Enunciado:**

Escribir el pseudocódigo y las pruebas de escritorio para realizar el ordenamiento de un vector con 5 números enteros. El usuario ingresa los números que él desea, cree un vector para guardar temporalmente dichos datos y luego realice el ordenamiento del mismo a través del método de la burbuja.

**Pseudocódigo**

```

INICIO
Lista entero listaNum;
entero n;
PARA (entero i=1, i<=5, i++)
    IMPRIMIR: "Ingrese un número";
    entero numero;
    TOMAR Y ASIGNAR: numero;
    listaNum.insertar(i, numero);
FIN PARA
IMPRIMIR: "La lista formada es:" + listaNum;
n = longitud(listaNum)
PARA (entero i = 1; i < n - 1; i++)
    entero minimo = i
    PARA (entero j = i+1; j < n; j++) // si este par no está ordenado
        SI (listaNum[j] < listaNum[minimo])
            minimo = j
        FIN SI
    FIN PARA
    SI (minimo != i) //intercambiamos el actual con el mínimo encontrado
        entero aux = listaNum[minimo]
        listaNum[minimo] = listaNum[j]
        listaNum[j] = aux
    FIN SI
FIN PARA
IMPRIMIR: "La lista ordenada es:" + listaNum;
FIN

```

**Prueba de Escritorio**

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 0$   
 $i \rightarrow j \rightarrow 5 < 3 ? \rightarrow \text{NO}$

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 0$   
 $i \rightarrow j \rightarrow 4 < 3 ? \rightarrow \text{NO}$

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 0$   
 $i \rightarrow j \rightarrow 7 < 3 ? \rightarrow \text{NO}$

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 0$   
 $i \rightarrow j \rightarrow 6 < 3 ? \rightarrow \text{NO}$

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 1$   
 $i \rightarrow j \rightarrow 4 < 5 ? \rightarrow \text{Si} \rightarrow \text{aux} = 5$   
 $\text{minimo} = 2$

3	4	5	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 2$   
 $i \rightarrow j \rightarrow 7 < 5 ? \rightarrow \text{NO}$

3	4	5	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 2$   
 $i \rightarrow j \rightarrow 6 < 5 ? \rightarrow \text{NO}$

3	4	5	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 2$   
 $i \rightarrow j \rightarrow 7 < 5 ? \rightarrow \text{NO}$

3	4	5	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 2$   
 $i \rightarrow j \rightarrow 6 < 5 ? \rightarrow \text{NO}$

3	4	5	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 3$   
 $i \rightarrow j \rightarrow 6 < 7 ? \rightarrow \text{Si} \rightarrow \text{aux} = 7$   
 $\text{minimo} = 4$

3	4	5	6	7
0	1	2	3	4

1er ciclo del Para  
i=0

2do ciclo del Para  
i=1

3er ciclo del Para  
i=2

4to ciclo del Para  
i=3

## Ejercicio 15 – Búsqueda Secuencial

**Enunciado:**

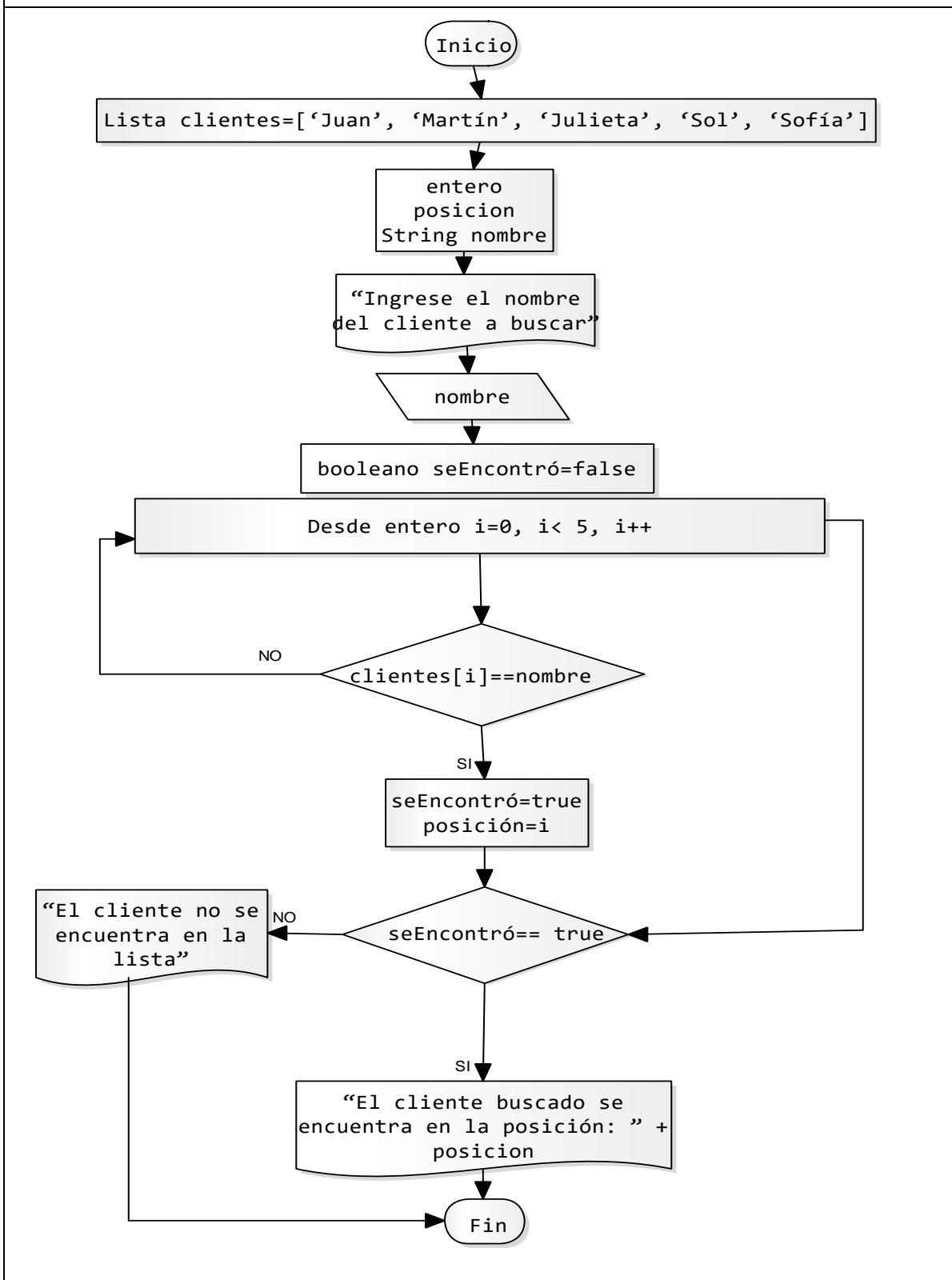
Escribir el pseudocódigo y las pruebas de escritorio para realizar la búsqueda del nombre de un cliente en un vector que contiene 5 clientes en total. El cliente a buscar será ingresado por pantalla por el usuario. El algoritmo deberá devolver, en caso de que ese nombre exista, la posición en donde se encuentra dicho cliente dentro del vector.

**Pseudocódigo**

```

INICIO
Lista clientes=['Juan', 'Martín', 'Julieta', 'Sol', 'Sofía']
entero posicion
Texto nombre
IMPRIMIR: "Ingrese el nombre del cliente a buscar"
    TOMAR Y ASIGNAR: nombre
BOOLEANO seEncontró= falso
// recorremos la lista, revisando cada elemento de la misma, para ver
// si es el cliente ingresado
PARA (entero i = 0; i < 5 - 1; i++)
// comparamos el cliente de la posición actual con la variable nombre.
    SI (clientes[i] == nombre)
        SeEncontró = verdadero
        posicion=i
        break; //Una vez encontrado el cliente, se deja de recorrer la lista. El
            break corta el ciclo PARA.
    FIN_SI
// si nunca se cumple clientes[i], entonces la variable que indica si se
// encontró o no el cliente: seEncontró, quedará valiendo falso.
FIN_PARA
SI (seEncontró == verdadero)
    IMPRIMIR: "El cliente buscado se encuentra en la posición: " + posicion
SINO
    IMPRIMIR: "El cliente no se encuentra en la lista"
FIN SI
FIN
  
```

**Diagrama de Flujo**





**Prueba de Escritorio**

Identificación de variables de entrada, tipos de variables y tipo de datos

TIPO VARIABLE	TIPO DE DATO	NOMBRE
Entrada	Texto	nombre
Auxiliar	Lista Texto	clientes
Auxiliar	Booleano	SeEncontró
Salida	Entero	posición(p)

**Ejecución de Pruebas**

N° de Prueba	Entrada	Auxiliares	Array: Ciclo PARA		Auxiliares		Salida
	nombre	clientes:	i	clientes[i] == nombre	SeEncontró	p	
1	"Sol"	['Juan' 'Martín' 'Julieta' 'Sol' 'Sofía']	0	'Juan'== 'Sol' -> falso	Falso		El cliente buscado se encuentra en la posición: 3
			1	'Martín'== 'Sol' -> falso	Falso		
			2	'Julieta'== 'Sol' -> falso	Falso		
			3	'Sol'== 'Sol' -> verdadero	Verdadero	3	
			4	'Sofía'== 'Sol' -> falso	Verdadero		
1	"Pedro"	['Juan' 'Martín' 'Julieta' 'Sol' 'Sofía']	0	'Juan'== 'Pedro' -> falso	Falso		El cliente no se encuentra en la lista
			1	'Martín'== 'Pedro' -> falso	Falso		
			2	'Julieta'== 'Pedro' -> falso	Falso		
			3	'Sol'== 'Pedro' -> falso	Falso		
			4	'Sofía'== 'Pedro' -> falso	Falso		

## Fuentes de Información

---

- **Frittelli, Valerio** – “Algoritmos y Estructuras de Datos” 1ra Edición (Editorial Científica Universitaria Año 2001)
- **Sommerville, Ian** - “INGENIERÍA DE SOFTWARE” 9na Edición (Editorial Addison-Wesley Año 2011).