

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

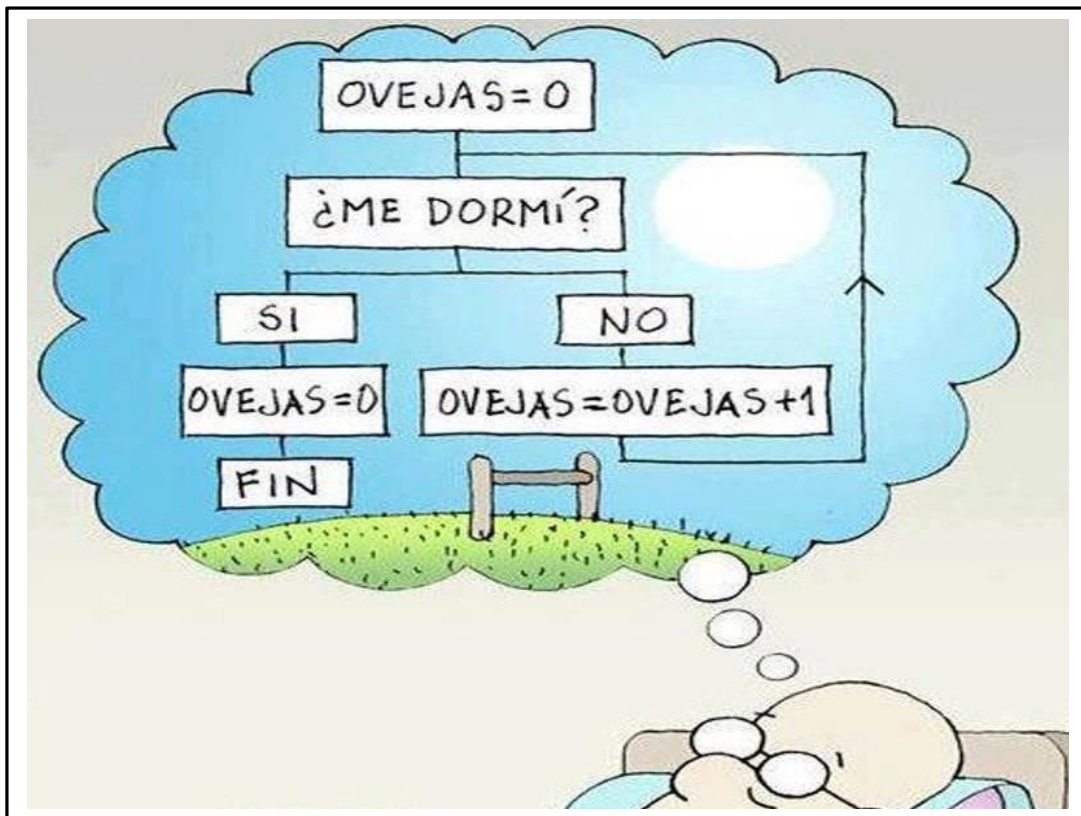


Tabla de Contenido

INTRODUCCIÓN.....	4
ENUNCIADOS DE LOS EJERCICIOS A DESARROLLAR EN LA GUÍA.....	5
EJERCICIOS ESTRUCTURAS DE CONTROL.....	5
EJERCICIO 1 – SECUENCIAL	5
EJERCICIO 2 – ALTERNATIVA SIMPLE.....	5
EJERCICIO 3 – ALTERNATIVA DOBLE	5
EJERCICIO 4 – ALTERNATIVA DOBLE	5
EJERCICIO 5 – ALTERNATIVA MÚLTIPLE	5
EJERCICIO 6 – REPETITIVA WHILE	5
EJERCICIO 7 – REPETITIVA DO WHILE	6
EJERCICIO 8 – REPETITIVA FOR.....	6
EJERCICIOS SOBRE ESTRUCTURAS DE DATOS.....	6
EJERCICIO 9 – ARRAY BOOLEANO	6
EJERCICIO 10 – DOS ARRAYS	6
EJERCICIO 11 – PILA	7
EJERCICIOS ALGORITMOS FUNDAMENTALES.....	7
EJERCICIO 12 – ORDENAMIENTO POR INSERCIÓN	7
EJERCICIO 13 – ORDENAMIENTO DE LA BURBUJA	7
EJERCICIO 14 – ORDENAMIENTO POR SELECCIÓN.....	7
EJERCICIO 15 – BÚSQUEDA SECUENCIAL	7
SOLUCIONES PROPUESTAS.....	9
EJERCICIOS ESTRUCTURAS DE CONTROL.....	9
EJERCICIO 1 – SECUENCIAL	9
EJERCICIO 2 – ALTERNATIVA SIMPLE.....	11
EJERCICIO 3 – ALTERNATIVA DOBLE	13
EJERCICIO 4 – ALTERNATIVA DOBLE	15
EJERCICIO 5 – ALTERNATIVA MÚLTIPLE	17
EJERCICIO 6 – REPETITIVA WHILE	19
EJERCICIO 7 – REPETITIVA DO WHILE	21
EJERCICIO 8 – REPETITIVA FOR.....	23
EJERCICIOS SOBRE ESTRUCTURAS DE DATOS.....	25
EJERCICIO 9 – ARRAY BOOLEANO	25
EJERCICIO 10 – DOS ARRAYS	27
EJERCICIO 11 – PILA	30
EJERCICIOS ALGORITMOS FUNDAMENTALES.....	33
EJERCICIO 12 – ORDENAMIENTO POR INSERCIÓN	33
EJERCICIO 13 – ORDENAMIENTO DE LA BURBUJA	35
EJERCICIO 15 – BÚSQUEDA SECUENCIAL	39
FUENTES DE INFORMACIÓN	42

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 subirse 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 FLOAT 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.

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.

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.

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.

Ejercicio 6 – Repetitiva 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.

Ejercicio 7 – Repetitiva Do While

Diseñar un algoritmo que muestre por pantalla la tabla de multiplicación del número que ingrese el usuario. Para definir hasta qué número desea que muestre la tabla de multiplicación el usuario también deberá ingresar este valor.

Ejercicio 8 – Repetitiva 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 – Array 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(true) implica que está ocupada y si es falso(false) la butaca está desocupada.

Ejercicio 10 – Dos Arrays

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.

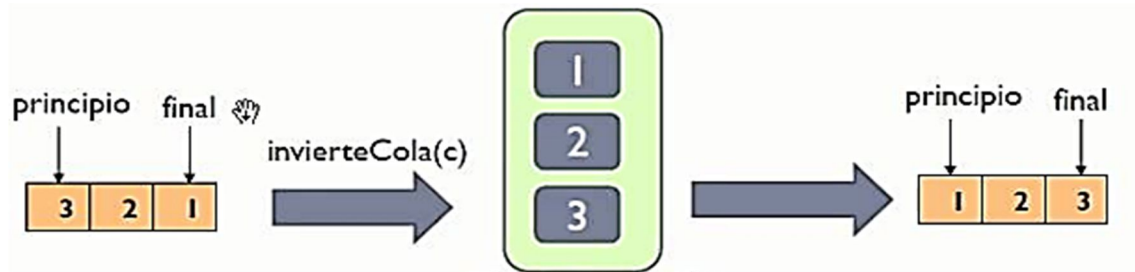
Listas necesarias para resolver el problema:

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

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:

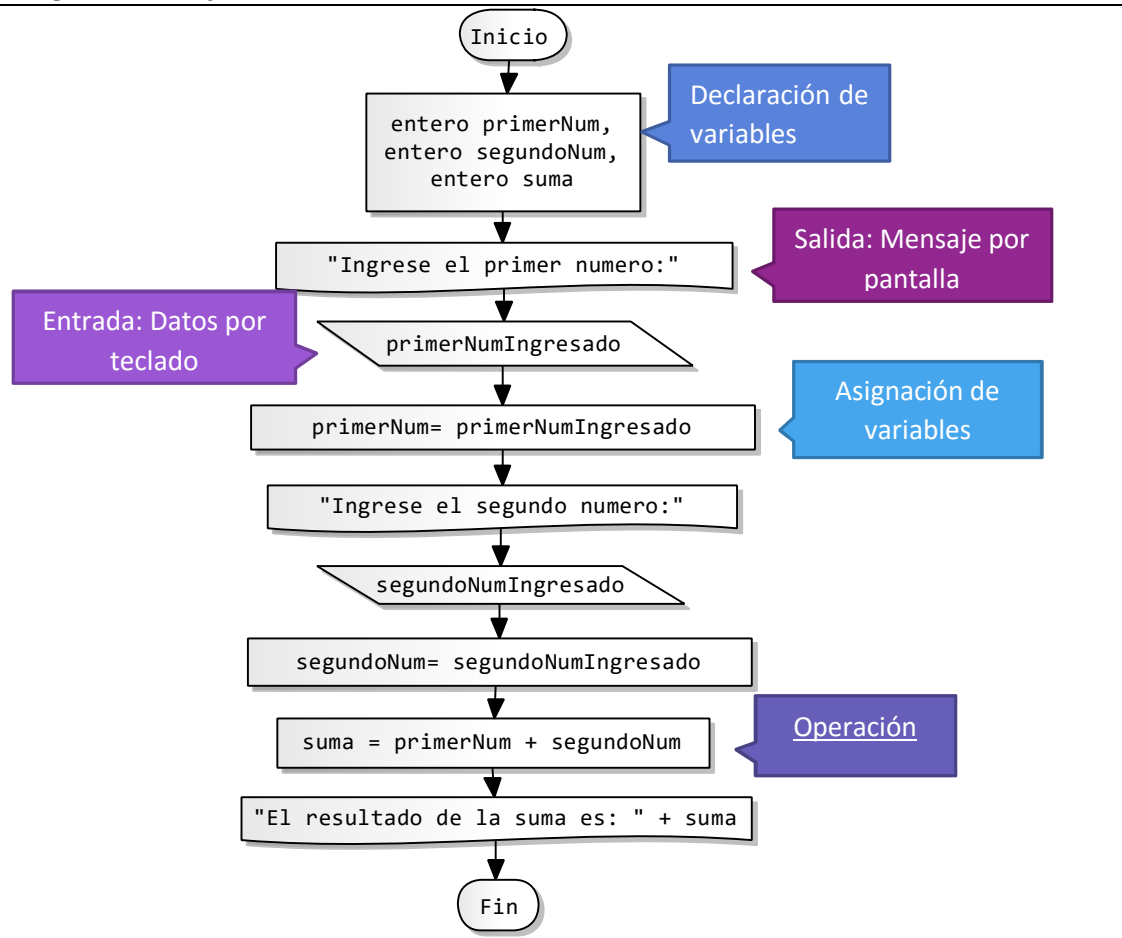
Escribir un algoritmo que permita realizar una suma de dos números enteros. El usuario deberá ingresar primero un número, luego el siguiente número, y el sistema arrojará el resultado correspondiente.

Pseudocódigo

```

INICIO
int primerNum;
int segundoNum;
int 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	int	primerNumIngresado
Entrada	Int	segundoNumIngresado
Auxiliar	Int	primerNum
Auxiliar	Int	segundoNum
Salida	int	Suma

Ejecución de Pruebas

N° Prueba	Entrada		Asignación		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

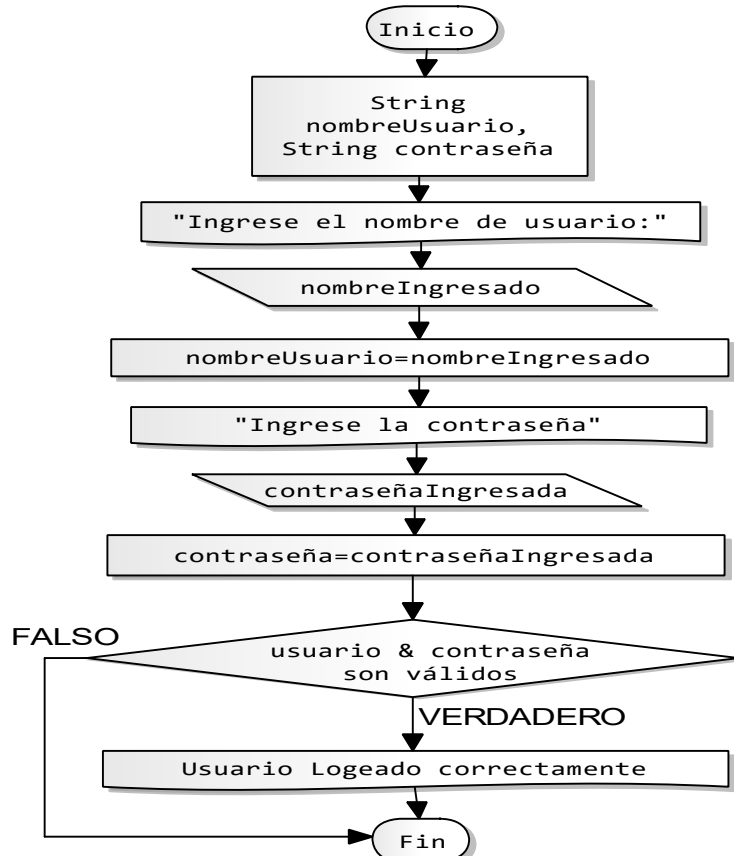
Escribir un algoritmo que permita loguearse (registrarse) a un sistema, ingresando un nombre de usuario y la contraseña adecuada. Considerar que tanto el usuario como la contraseña están formados sólo por letras. El sistema deberá validar que el usuario y la contraseña sean correctas, comparándolas con 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
String nombreUsuario;
String 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;
IF(esValido(usuario) && esValido(contraseña))
THEN
IMPRIMIR: "Usuario logeado con éxito".
END IF
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	String	nombreIngresado
Entrada	String	contraseñaIngresada
Auxiliar	String	nombreUsuario
Auxiliar	String	contraseña
Salida	Mensaje	"Usuario Logeado correctamente"

Ejecución de Pruebas

N° Prueba	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

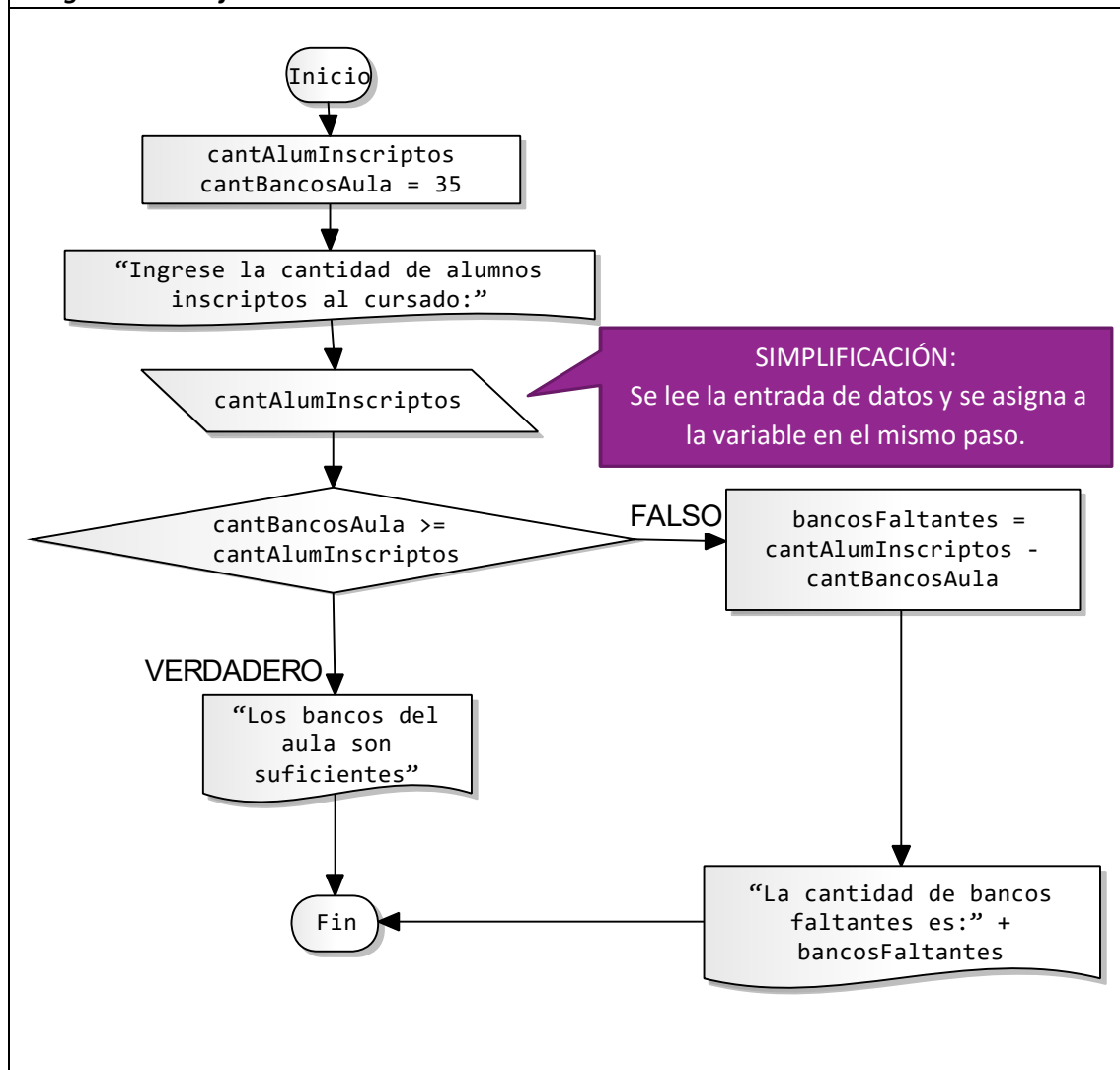
Enunciado

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.

Pseudocódigo

```

INICIO
int cantBancosAula = 35
int cantAlumInscriptos;
int bancosFaltantes;
IMPRIMIR: "Ingrese la cantidad de alumnos inscriptos al cursado:"
TOMAR Y ASIGNAR: cantAlumInscriptos;
IF (cantBancosAula >= cantAlumInscriptos)
    THEN: IMPRIMIR: "Los bancos del aula son suficientes".
ELSE
    bancosFaltantes = cantAlumInscriptos - cantBancosAula
    IMPRIMIR: "La cantidad de bancos faltantes es:" + bancosFaltantes.
END IF
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	int	cantAlumInscriptos
Auxiliar	Int	cantBancosAula
Salida	Int	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:" + 22
2	45	38 >= 45: NO	45 - 38 = 7	"La cantidad de bancos faltantes es:" + 22
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 int.	
5	38	38 >= 38: SI		"Los bancos del aula son suficientes".

Ejercicio 4 – Alternativa Doble

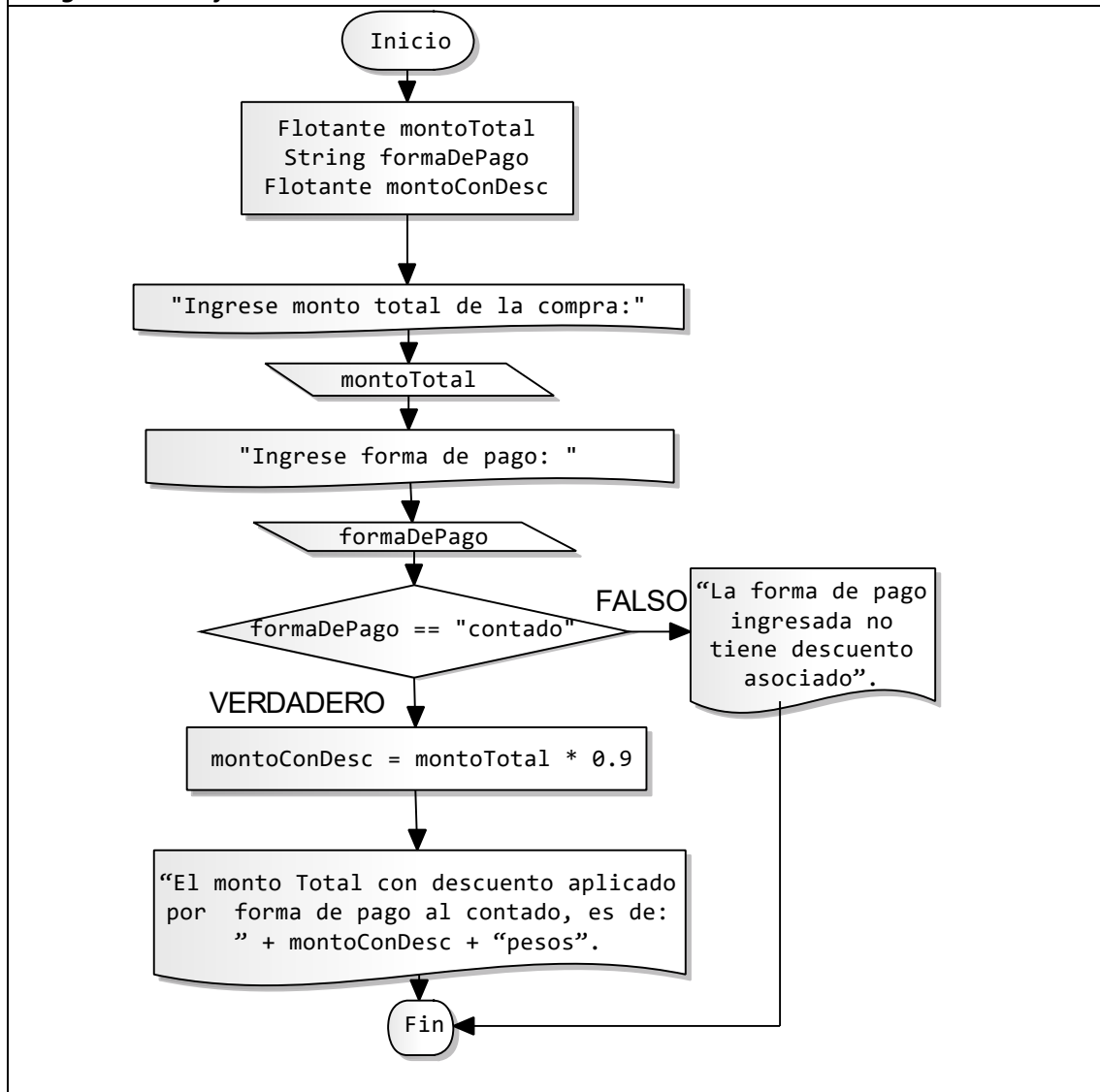
Enunciado

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.

Pseudocódigo

```

INICIO
Float montoTotal
String formaDePago
Float montoConDesc
IMPRIMIR: "Ingrese monto total de la compra"
TOMAR Y ASIGNAR montoTotal;
IMPRIMIR: "Ingrese forma de pago"
TOMAR Y ASIGNAR formaDePago;
IF (formaDePago == "contado")
    montoConDesc = montoTotal * 0.9
    IMPRIMIR: "El monto Total con descuento aplicado por forma de pago al
    contado, es de:" + montoConDesc + "pesos".
ELSE
    IMPRIMIR: "La forma de pago ingresada no tiene descuento asociado".
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	Float	montoTotal
Entrada	String	formaDePago
Salida	Float	montoConDesc

Ejecución de Pruebas

N° Prueba	Entrada		Bloque de Decisión	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".
2	400	"tarjeta de débito"	"tarjeta de débito"==" contado"		"La forma de pago ingresada no tiene descuento asociado".
3	1320	"contado"	"contado"==" contado"	1320 * 0.9= 1188	"El monto Total con descuento aplicado por forma de pago al contado, es de:" + 1188 + "pesos".
4	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

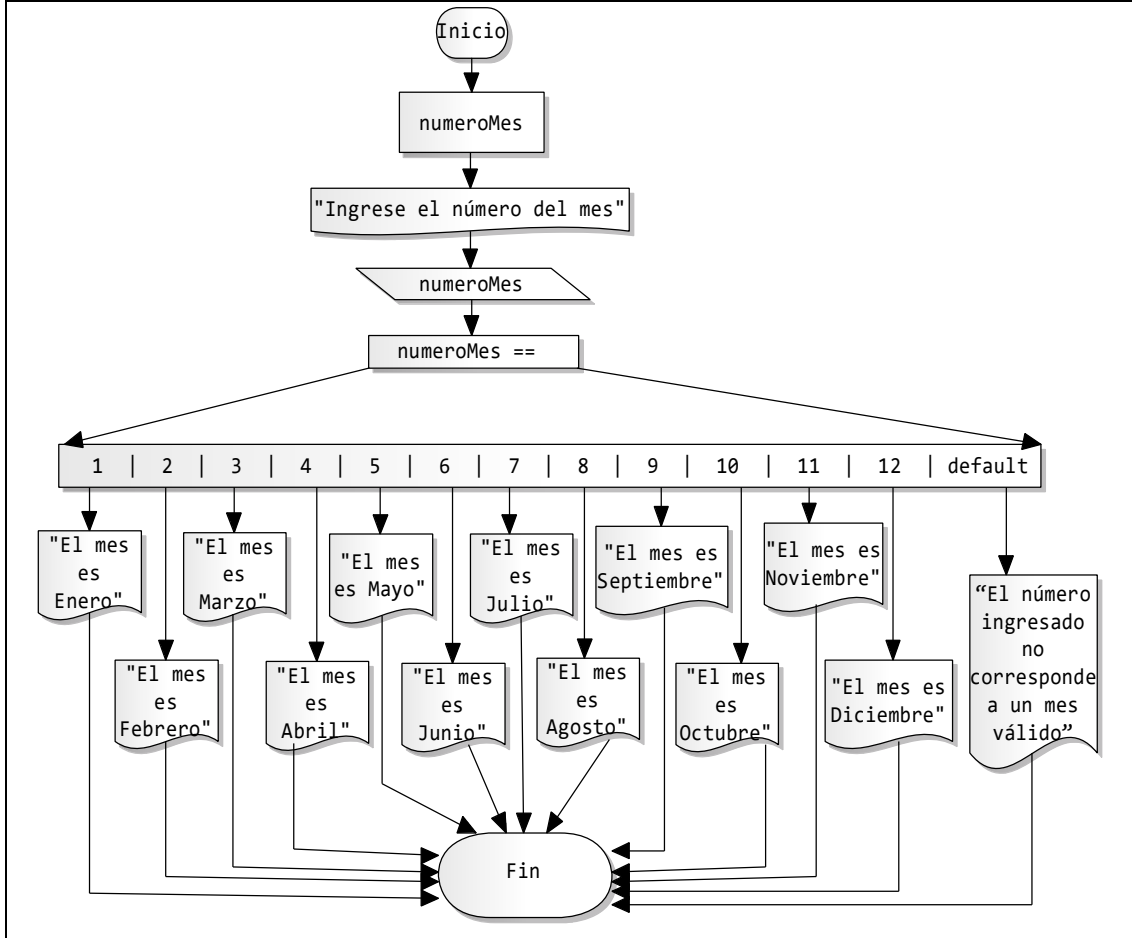
Enunciado

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

Pseudocódigo

```
INICIO
Int numeroMes;
IMPRIMIR: "Ingrese el número del mes"
TOMAR Y ASIGNAR numeroMes;
Switch (numeroMes)
    case (1): "El mes es Enero";
    Break;
    case (2): "El mes es Febrero";
    Break;
    case (3): "El mes es Marzo";
    Break;
    case (4): "El mes es Abril";
    Break;
    case (5): "El mes es Mayo";
    Break;
    case (6): "El mes es Junio";
    Break;
    case (7): "El mes es Julio";
    Break;
    case (8): "El mes es Agosto";
    Break;
    case (9): "El mes es Septiembre";
    Break;
    case (10): "El mes es Octubre";
    Break;
    case (11): "El mes es Noviembre";
    Break;
    case (12): "El mes es Diciembre";
    Break;
    Default: "El número ingresado no corresponde a un mes válido"
//El default, es un valor por defecto. En caso de que la variable numeroMes
no corresponda con ninguna de las opciones contempladas(case) entonces el
switch entrará por el default 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	int	numeroMes
Salida	String	Mensaje

Ejecución de Pruebas

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

Ejercicio 6 – Repetitiva While

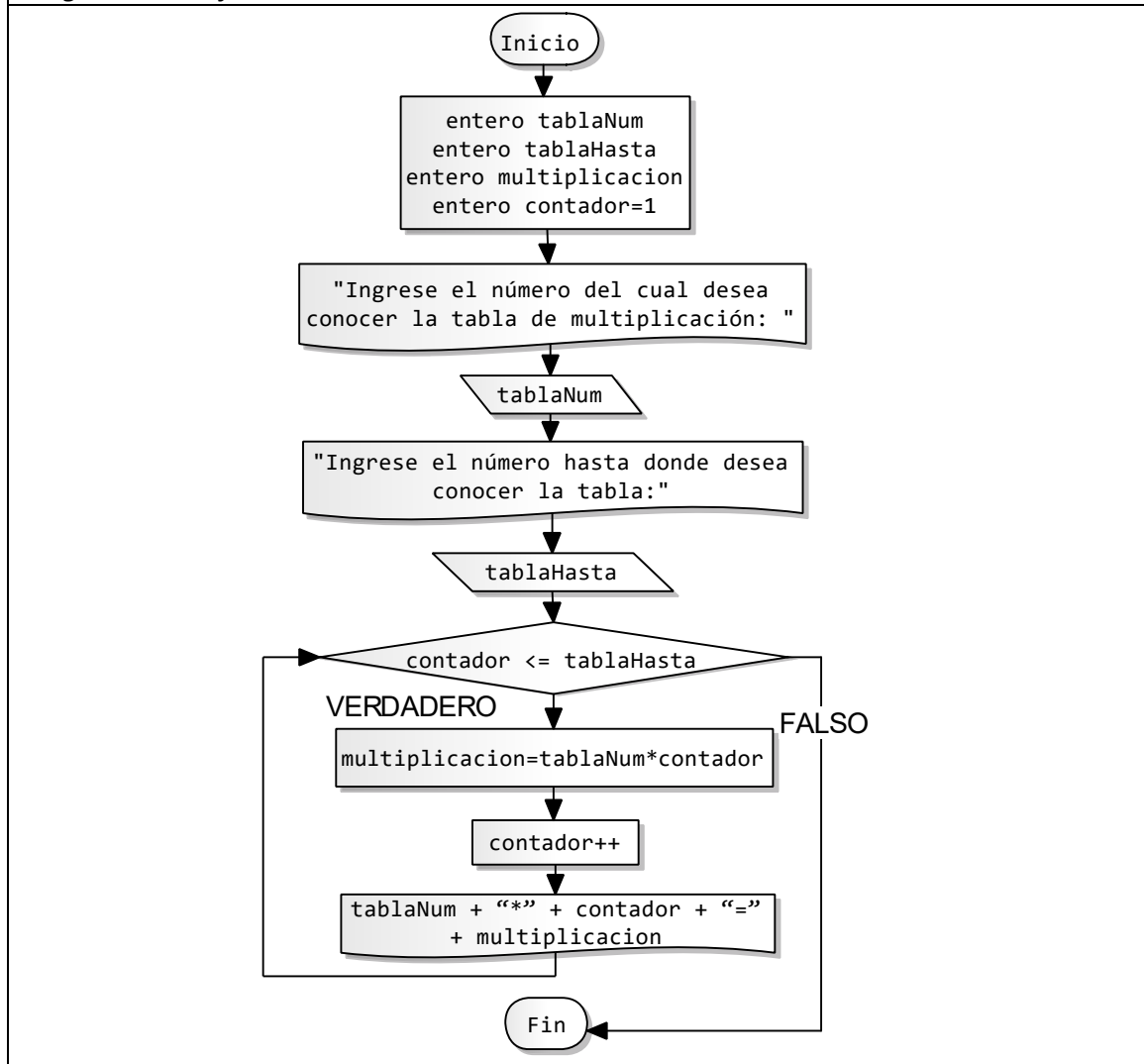
Enunciado

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.

Pseudocódigo

```

INICIO
int tablaNum;
int tablaHasta;
int contador=1;
int 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;
While(contador <=tablaHasta)
{multiplicación=tablaNum*contador;
Contador++;
IMPRIMIR: tablaNum + "*" + contador + "=" + multiplicacion}
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	int	tablaNum
Entrada	int	tablaHasta
Auxiliar	int	Contador (c)
Salida	int	Multiplicación (m)

Ejecución de Pruebas

N° Prueba	Entrada		Auxiliar	Bucle 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 Do While

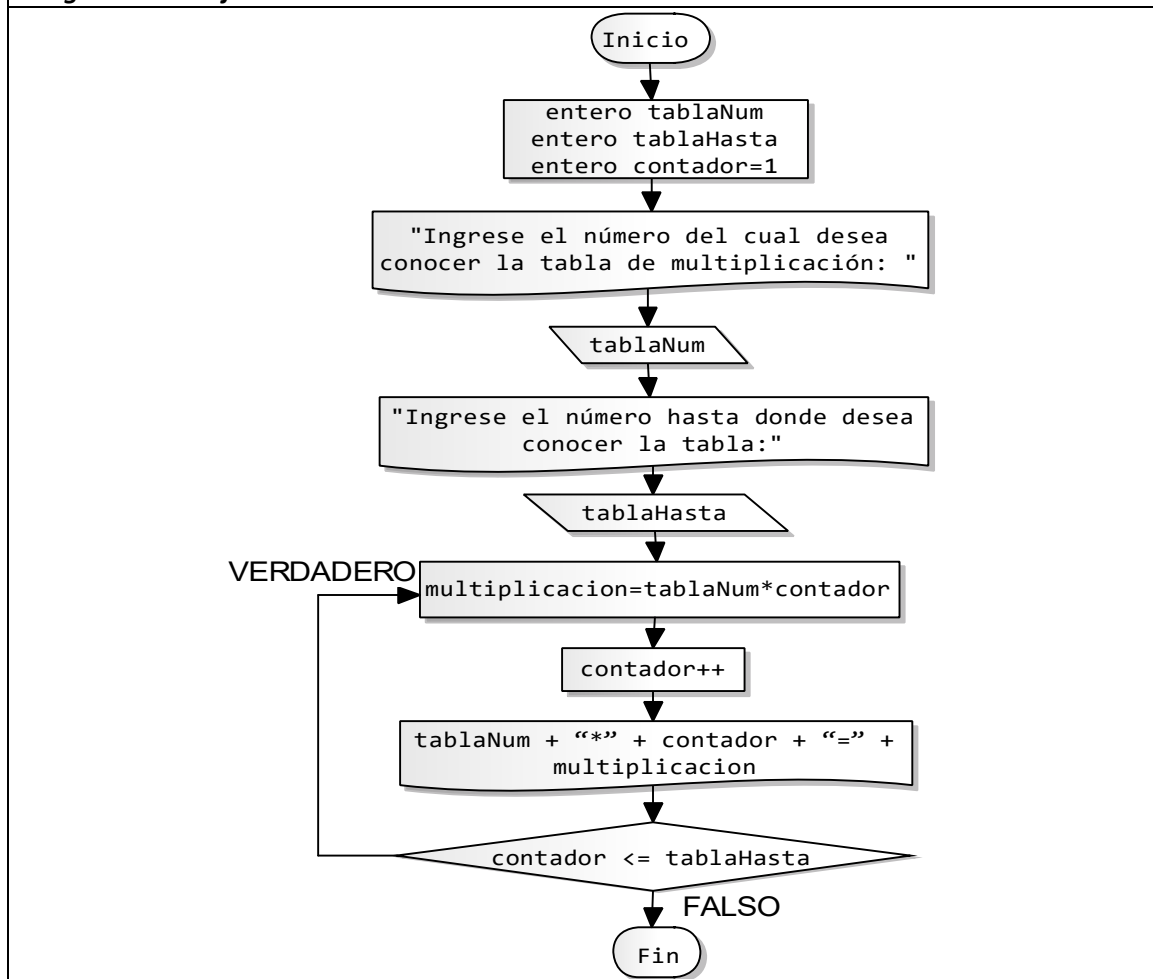
Enunciado

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.

Pseudocódigo

```

INICIO
int tablaNum;
int tablaHasta;
int contador=1;
int 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;
Do{multiplicación=tablaNum*contador;
  Contador++;
  IMPRIMIR: tablaNum + "*" + contador + "=" + multiplicacion}
While(contador <=tablaHasta)
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	int	tablaNum
Entrada	int	tablaHasta
Auxiliar	int	Contador (c)
Salida	int	Multiplicación (m)

Ejecución de Pruebas

N° Prueba	Entrada		Auxiliar	Operaciones del bloque 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 For

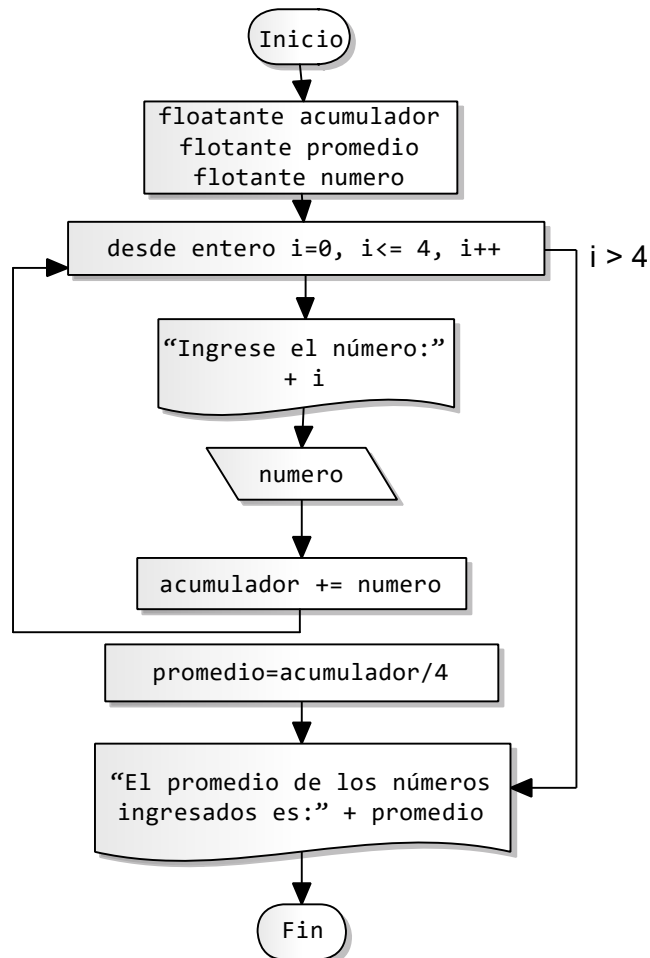
Enunciado

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
float acumulador;
float promedio;
float numero;
For (int i=0, i<= 4, i++)
    IMPRIMIR: "Ingrese el número:" + i
    TOMAR numero;
    acumulador += numero;
Fin For
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	float	acumulador (a)
Salida	float	promedio(p)
Salida	string	mensaje

Ejecución de Pruebas

Id.	Entrada	Ciclo For	Auxiliar	Salida	
	numero	i<longitud butacas	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 – Array Booleano

Enunciado:

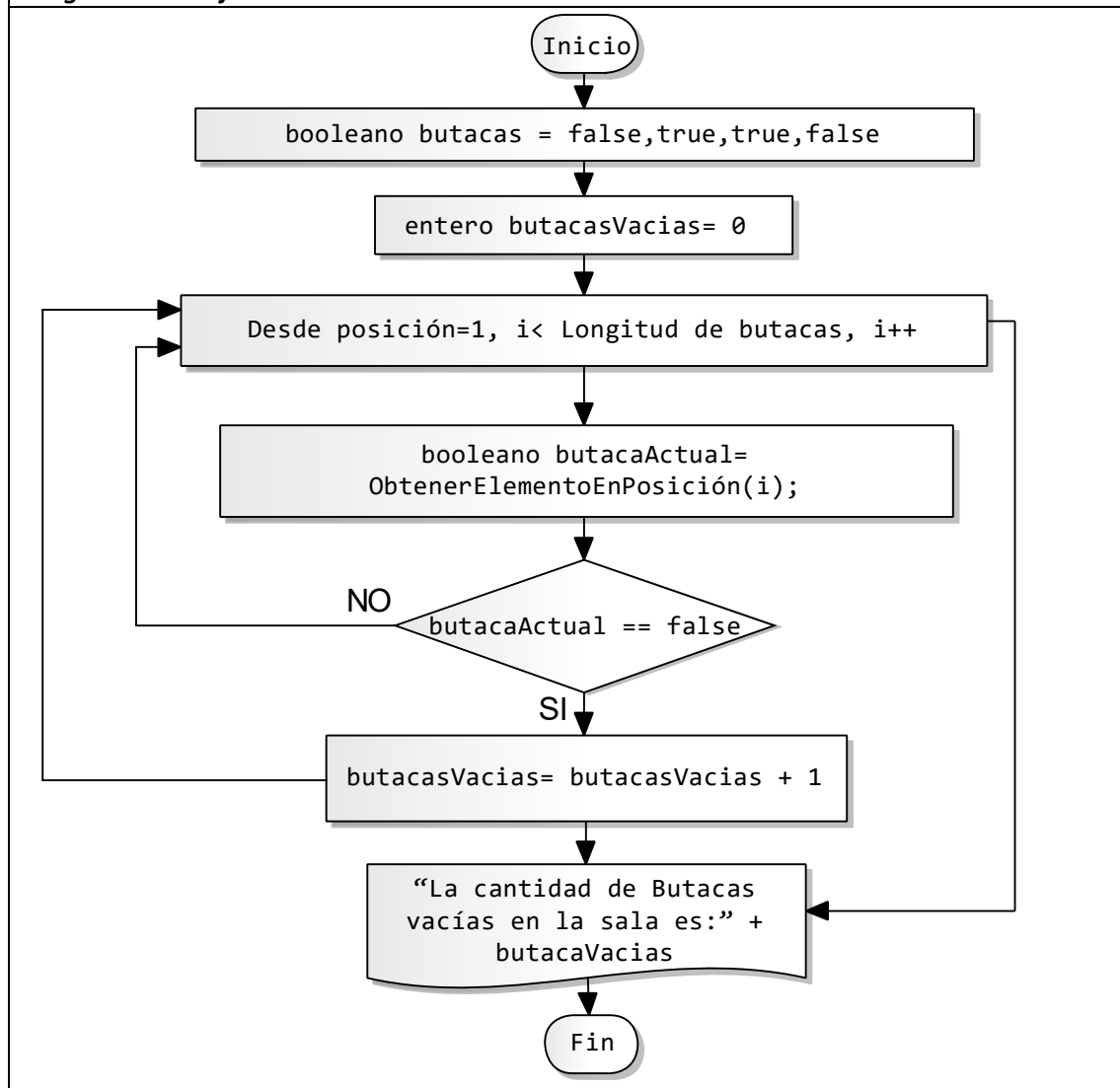
Diseñar un algoritmo que recorra las butacas de una sala de cine y determine cuántas butacas desocupadas hay en la sala. Suponga que inicialmente tiene un array (arreglo) con valores booleanos que si es verdadero(true) implica que está ocupada y si es falso(false) la butaca está desocupada.

Pseudocódigo

```

INICIO
boolean butacas[] = [false,true,true,false]
int butacasVacias =0 //Contador que guarda la cantidad de butacas vacías.
For (int i=0, i< butacas.lenght(), i++)
boolean butacaActual= Obtener(listaCapacAulas, i);
if (butacaActual == false)
    butacasVacias++; // suma 1 al valor que tiene almacenado la variable
butacasVacias.
IMPRIMIR: "La cantidad de Butacas vacías en la sala es:" + butacaVacias
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 Boolean	butacas
Auxiliar	Boolean	ButacaActual (ba)
Auxiliar y de Salida	Int	ButacasVacías (bv)

Ejecución de Pruebas

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

Ejercicio 10 – Dos Arrays

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

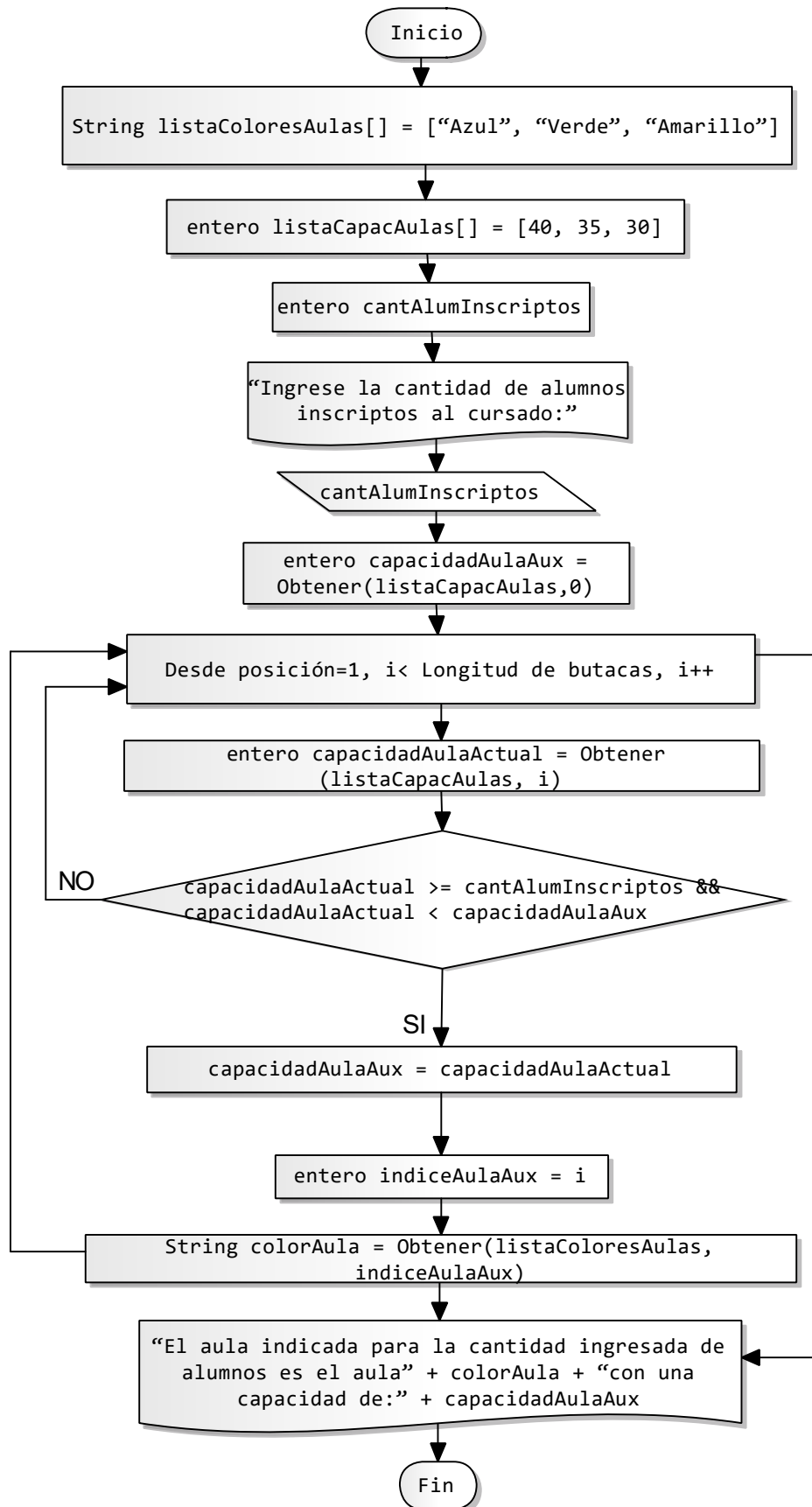
Pseudocódigo

```

INICIO
String listaColoresAulas[] = ["Azul", "Verde", "Amarillo"]
int listaCapacAulas[] = [40, 35, 30]
int cantAlumIns;
IMPRIMIR: "Ingrese la cantidad de alumnos inscriptos al cursado:"
TOMAR Y ASIGNAR: cantAlumIns;
int capacidadAulaAux = Obtener(listaCapacAulas,0);

For (i=1, i< listaCapacAulas.lenght(), i++)
    int capacidadAulaActual = Obtener (listaCapacAulas, i);
    if(capacidadAulaActual >= cantAlumIns && capacidadAulaActual <
capacidadAulaAux)
        capacidadAulaAux = capacidadAulaActual;
        int indiceAulaAux = i;
Fin FOR
String 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 Int	listaCapacAulas= [40, 35, 30]
Auxiliar	Int	listaColoresAulas= ["Azul", "Verde", "Amarillo"]
Auxiliar y de Salida	Int	capacidadAulaAux
Auxiliar	Int	capacidadAulaActual
Auxiliar	Int	indiceAulaAux
Salida	String	colorAula
Entrada	Int	cantAlumIns

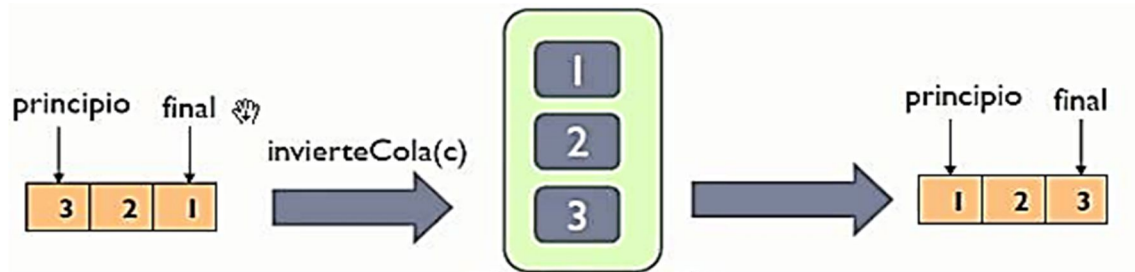
Ejecución de Pruebas

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

Ejercicio 11 – Pila

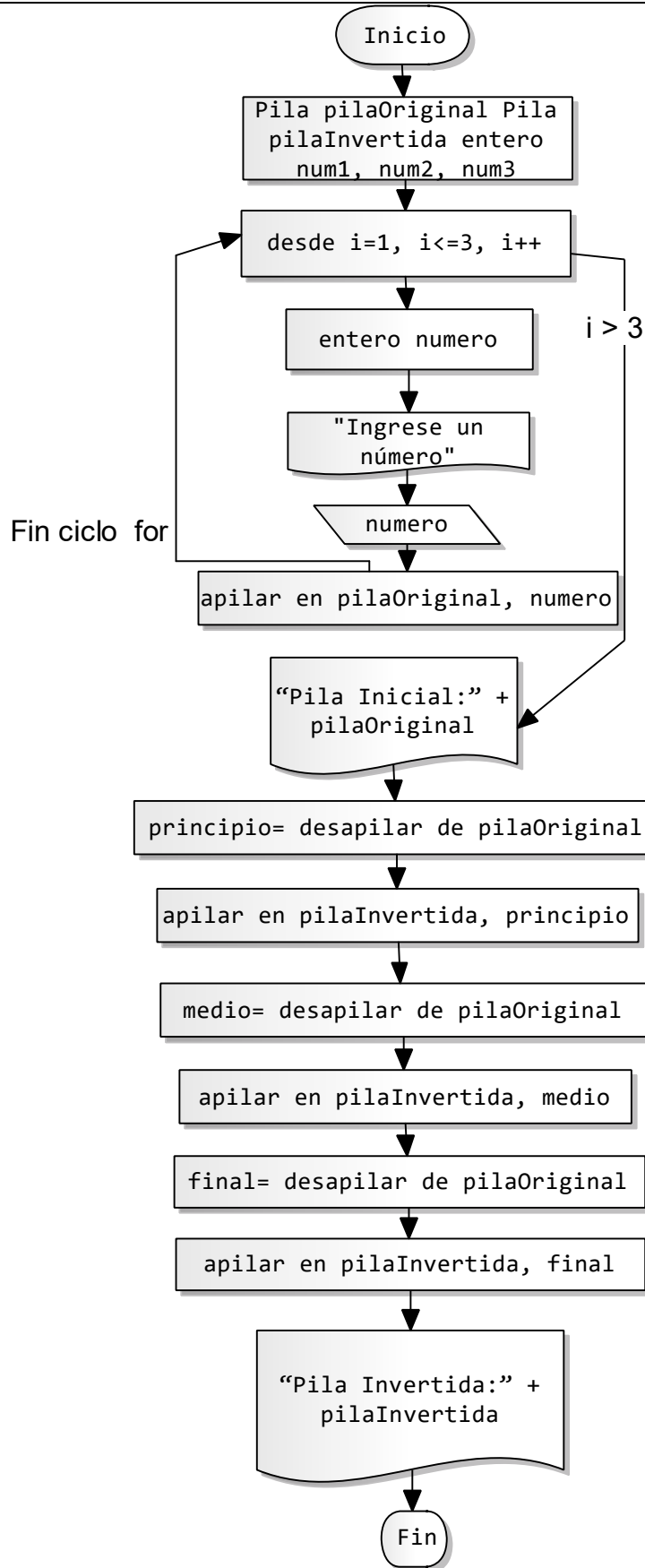
Enunciado:

Diseñar un algoritmo que a partir de una pila inicial de tres elementos devuelva una pila invertida. 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.

**Pseudocódigo**

```

INICIO
Pila pilaOriginal;
Pila pilaInvertida;
For (i=1, i<=3, i++)
    IMPRIMIR "Ingrese un número";
    Int numero;
    TOMAR Y ASIGNAR numero;
    apilar (pilaOriginal, numero);
Fin For
IMPRIMIR "Pila Inicial:" + pilaOriginal
int principio= desapilar(pilaEnteros); //principio=3
apilar (pilaInvertida, principio); //el elemento "principio" pasa a ser "final"
int medio= desapilar(pilaEnteros); //medio=2
apilar(pilaInvertida, medio); //el elemento "medio" seguirá siendo "medio"
int final = desapilar(pilaEnteros); //final=3
apilar(pilaInvertida, final); //el elemento "final" pasa a ser "principio"
IMPRIMIR: "Pila Invertida:" + pilaInvertida
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	Pila de Enteros	PilaOriginal
Auxiliar	Int	principio
Auxiliar	Int	medio
Auxiliar	Int	final
Salida	Pila de Enteros	PilaInvertida

Ejecución de Pruebas

Id.	Ciclo For		Operaciones			Salida
	Entrada	Apilar en Pila Original	Pila Original	Desapilar de Pila Original	Apilar en Pila Invertida	Pila Invertida
Prueba 1	numero=9	Apilar9	[11]	Principio=11	Apilar(Principio)	[9]
	numero=10	Apilar10	[10]	Medio=10	Apilar(Medio)	[10]
	numero=11	Apilar11	[9]	Final=9	Apilar(Final)	[11]
Prueba 2	numero=20	Apilar20	[24]	Principio=24	Apilar(Principio)	[20]
	numero=22	Apilar22	[22]	Medio=22	Apilar(Medio)	[22]
	numero=24	Apilar24	[20]	Final=20	Apilar(Final)	[24]

Ejercicios Algoritmos Fundamentales

Ejercicio 12 – Ordenamiento por Inserción

Enunciado:

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

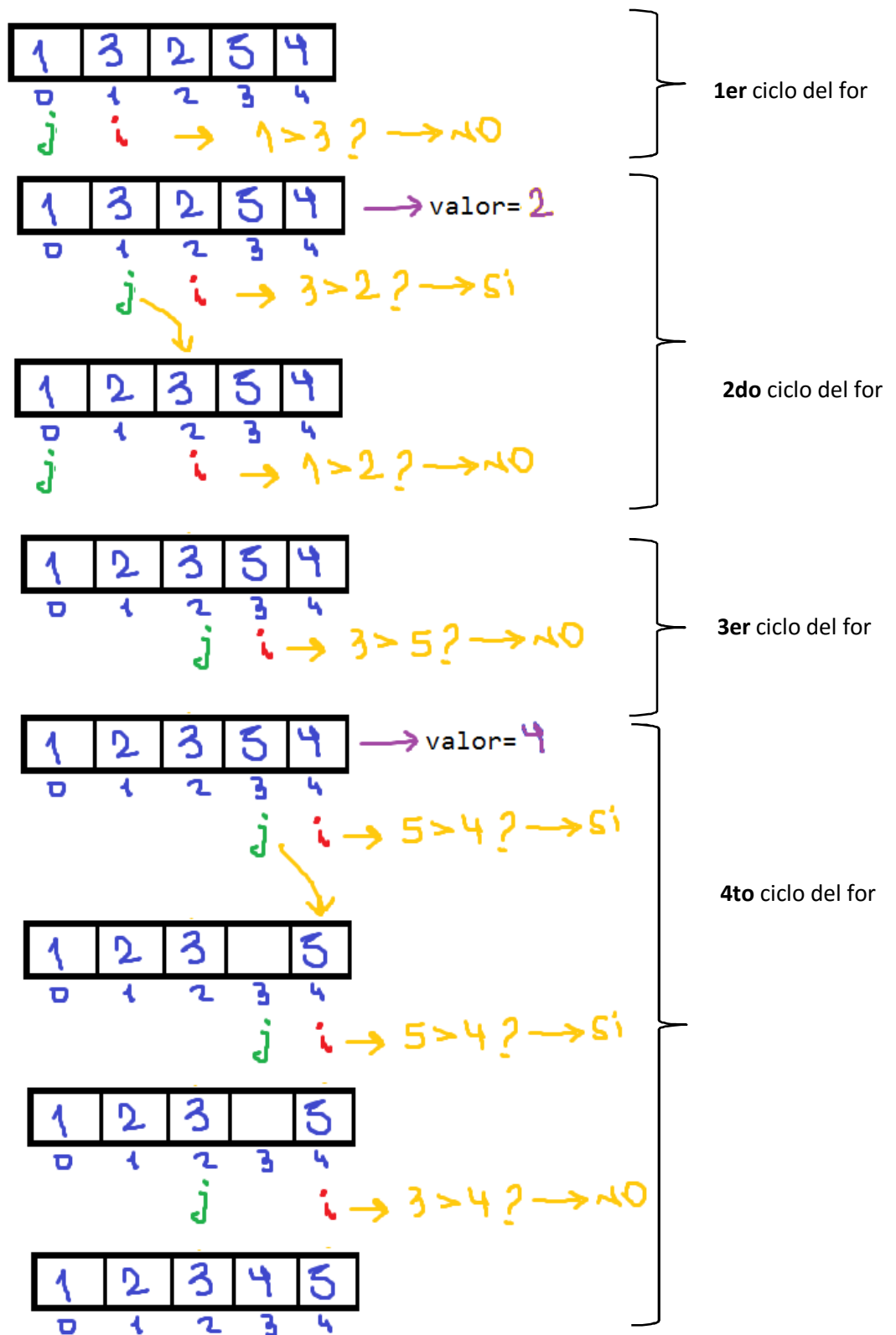
Escribir el pseudocódigo y las pruebas de escritorio para realizar el ordenamiento de un vector con 5 números enteros. El usuario ingresa los números que él desea, cree un vector para guardar temporalmente dichos datos y luego realice el ordenamiento del mismo por inserción

Pseudocódigo

```

INICIO
Lista int 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 (int i = 1; i < longitud(listaNum); i++);
    int valor = listaNum[i]
    int 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.

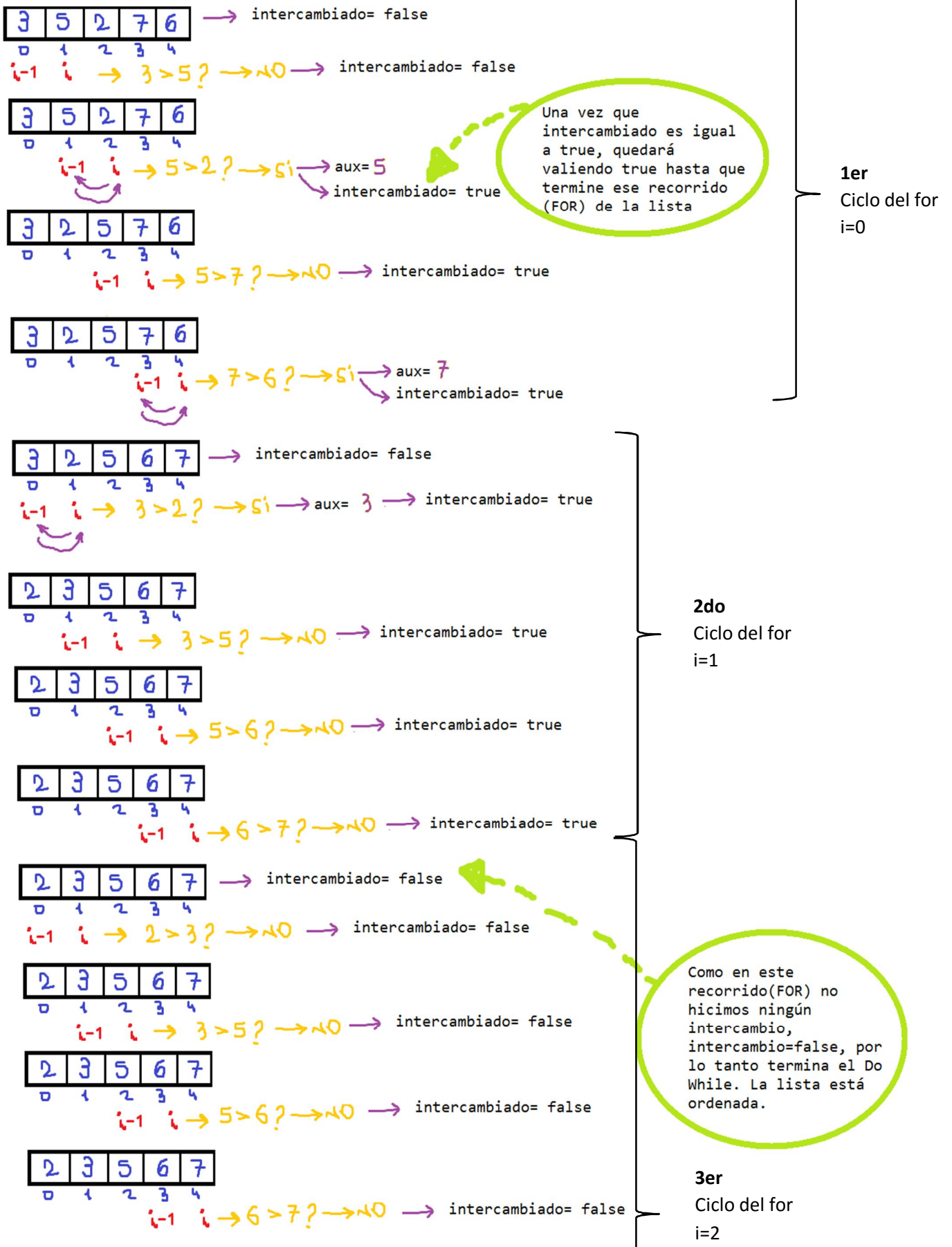
Pseudocódigo

```

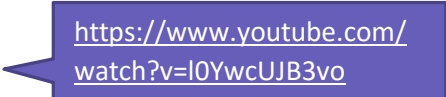
INICIO
Lista int listaNum;
FOR (i=1, i<=5, i++)
    IMPRIMIR: "Ingrese un número";
    int numero;
    TOMAR Y ASIGNAR: numero;
    listaNum.insertar(i, numero);
FIN FOR
IMPRIMIR: "La lista formada es:" + listaNum;
DO:
    boolean intercambiado = falso
    FOR (int i = 1; i < n; i++)
        //si este 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_FOR
WHILE:(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 int listaNum;
int n;
FOR (int i=1, i<=5, i++)
    IMPRIMIR: "Ingrese un número";
    int numero;
    TOMAR Y ASIGNAR: numero;
    listaNum.insertar(i, numero);
FIN FOR
IMPRIMIR: "La lista formada es:" + listaNum;
n = longitud(listaNum)
FOR (int i = 1; i < n - 1; i++)
    int minimo = i
    FOR (int j = i+1; j < n; j++) // si este par no está ordenado
        IF (listaNum[j] < listaNum[minimo])
            minimo = j
        FIN_IF
    FIN_FOR
    IF (minimo != i)//intercambiamos el actual con el mínimo encontrado
        int aux = listaNum[minimo]
        listaNum[minimo] = listaNum[j]
        listaNum[j] = aux
    FIN_IF
FIN_FOR
IMPRIMIR: "La lista ordenada es:" + listaNum;
FIN

```

Prueba de Escritorio

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 0$
 $i = 1, j = 2 \rightarrow 5 < 3? \rightarrow \text{NO}$

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 0$
 $i = 1, j = 3 \rightarrow 4 < 3? \rightarrow \text{NO}$

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 0$
 $i = 1, j = 4 \rightarrow 7 < 3? \rightarrow \text{NO}$

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 0$
 $i = 1, j = 5 \rightarrow 6 < 3? \rightarrow \text{NO}$

3	5	4	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 1$
 $i = 2, j = 3 \rightarrow 4 < 5? \rightarrow \text{Si} \rightarrow \text{aux} = 5, \text{minimo} = 2$

3	4	5	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 2$
 $i = 2, j = 4 \rightarrow 7 < 5? \rightarrow \text{NO}$

3	4	5	7	6
0	1	2	3	4

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

3	4	5	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 2$
 $i = 3, j = 4 \rightarrow 7 < 5? \rightarrow \text{NO}$

3	4	5	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 2$
 $i = 3, j = 5 \rightarrow 6 < 5? \rightarrow \text{NO}$

3	4	5	7	6
0	1	2	3	4

 $\rightarrow \text{minimo} = 3$
 $i = 4, j = 5 \rightarrow 6 < 7? \rightarrow \text{Si} \rightarrow \text{aux} = 7, \text{minimo} = 4$

3	4	5	6	7
0	1	2	3	4

1er ciclo del for
i=0

2do ciclo del for
i=1

3er ciclo del for
i=2

4to ciclo del for
i=3

Ejercicio 15 – Búsqueda Secuencial

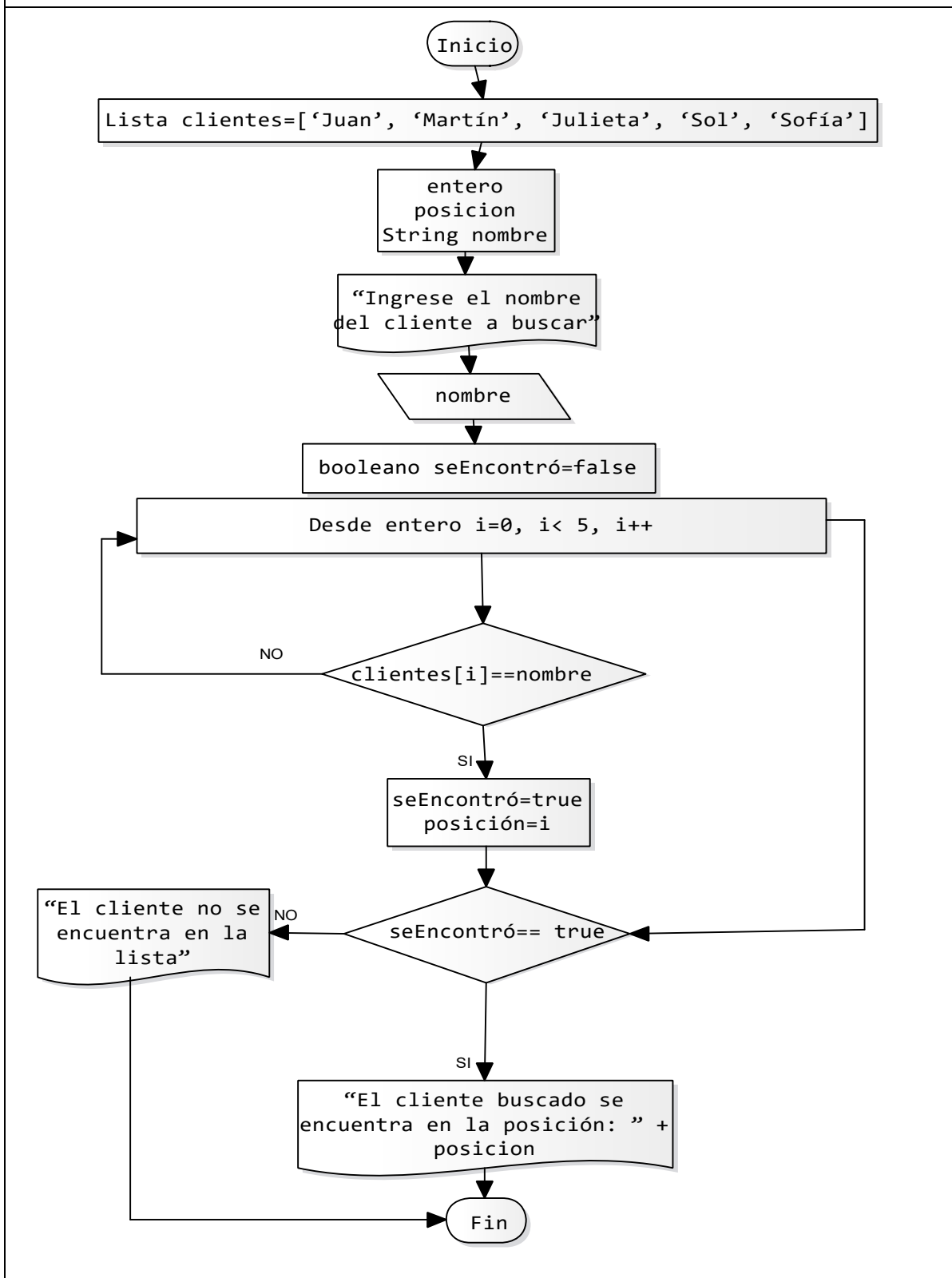
Enunciado:

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']
int posicion;
String nombre;
IMPRIMIR: "Ingrese el nombre del cliente a buscar";
    TOMAR Y ASIGNAR: nombre;
BOOLEAN seEncontró= false;
// recorremos la lista, revisando cada elemento de la misma, para ver
// si es el cliente ingresado
FOR (int i = 0; i < 5 - 1; i++)
// comparamos el cliente de la posición actual con la variable nombre.
    IF (clientes[i] == nombre)
        SeEncontró = true;
        posicion=i;
        break; //Una vez encontrado el cliente, se deja de recorrer la lista. El
            break corta el ciclo for.
    FIN_IF
// si nunca se cumple clientes[i], entonces la variable que indica si se
// encontró o no el cliente: seEncontró, quedará valiendo falso.
FIN_FOR
IF (seEncontró == true)
IMPRIMIR: "El cliente buscado se encuentra en la posición: " + posicion;
ELSE
IMPRIMIR: "El cliente no se encuentra en la lista";
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	String	nombre
Auxiliar	Lista String	clientes
Auxiliar	Boolean	SeEncontró
Salida	Int	posición(p)

Ejecución de Pruebas

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

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).