Programación

UNIDAD 1
INTRODUCCIÓN A LA PROGRAMACIÓN

José L. Berenguel

Tabla de Contenidos

1. Introducción

¿Qué es programar?

- Dar una serie de instrucciones al ordenador para indicarle lo que tiene que hacer (resolver un problema).

¿Qué es un programa?

- Una secuencia ordenada de instrucciones que un ordenador es capaz de ejecutar.

1. Introducción

- **Y entonces, ¿cómo creo programas?**

1) Es fundamental **comprender el problema** que queremos resolver al detalle, sin dudas y sin que haya ambigüedades o posibles interpretaciones.

Ejemplo: Ve al super a comprar leche.

Ejercicio: Rescribe la frase anterior para que no haya ambigüedades.

2) Analizar los datos que son necesarios para obtener el resultado (**datos de entrada**) y sus características.

Ejemplo: Calcula el factorial de x.

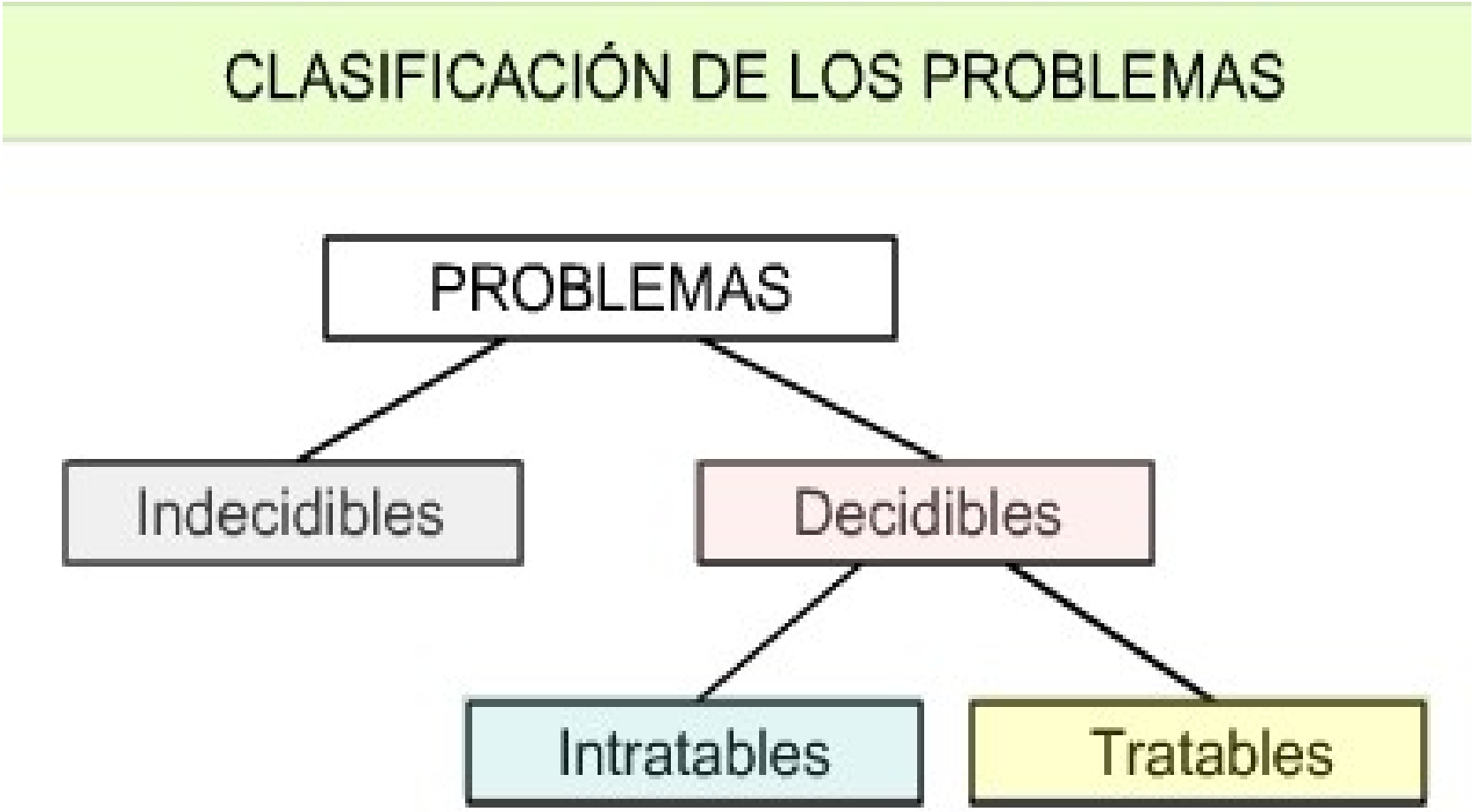
Ejercicio: ¿Qué valores puede tomar x?

1. Introducción

- **¿Cómo comprender el problema que debo resolver?**
  - Depende del tipo de problema al que nos enfrentamos.
    - **Problemas grandes y complejos**: hay que descomponerlo en problemas más sencillos que seamos capaces de manejar: divide y venceras.
    - **Problemas pequeños y sencillos**: analizando los datos que requiere y creando diagramas o modelos visuales que nos ayuden a comprobar su correcto funcionamiento.
  - Utilizando **modelos de desarrollo de software**.
    - Existen muchos y diversos. Se estudian en el módulo Entornos de Desarrollo.

COMPRENDER EL PROBLEMA ES EL PASO MÁS IMPORTANTE PARA PROGRAMAR

1.1. Clasificación de problemas



1.1. Clasificación de problemas

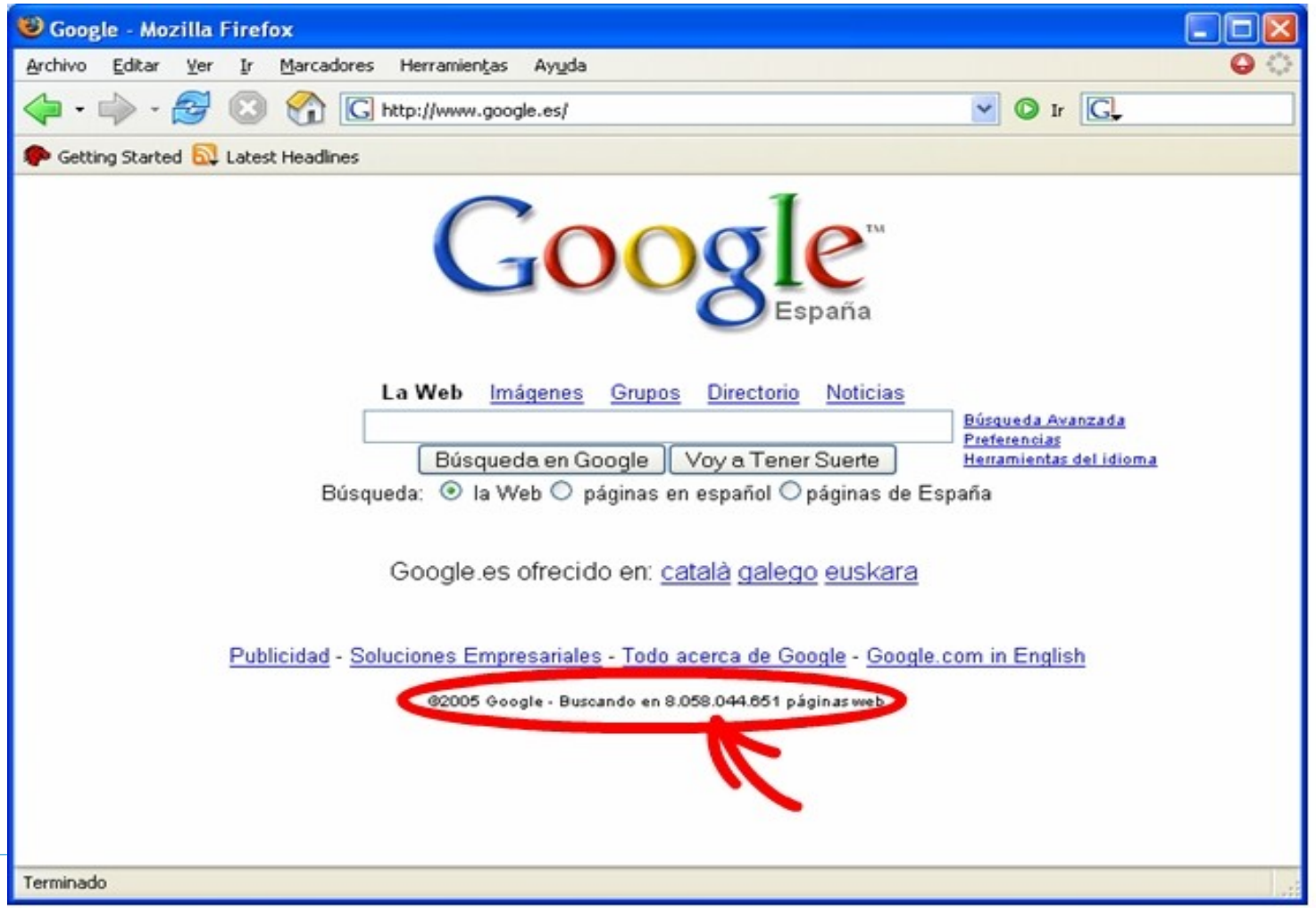
- Es necesario diseñar algoritmos eficientes.

¿Cómo Multiplicar 25 x100?

25 +25 +25 +25 . . . . . . +25 2500	100 +100 . . . . . . +100 2500	100 x 25 500 200 2500	$25 \times 10^2 = 2500$
Suma 25 cien veces	Suma 100 veinticinco veces	Multiplicación típica	Multiplicar por una potencia de 10. Simplemente añadimos dos ceros

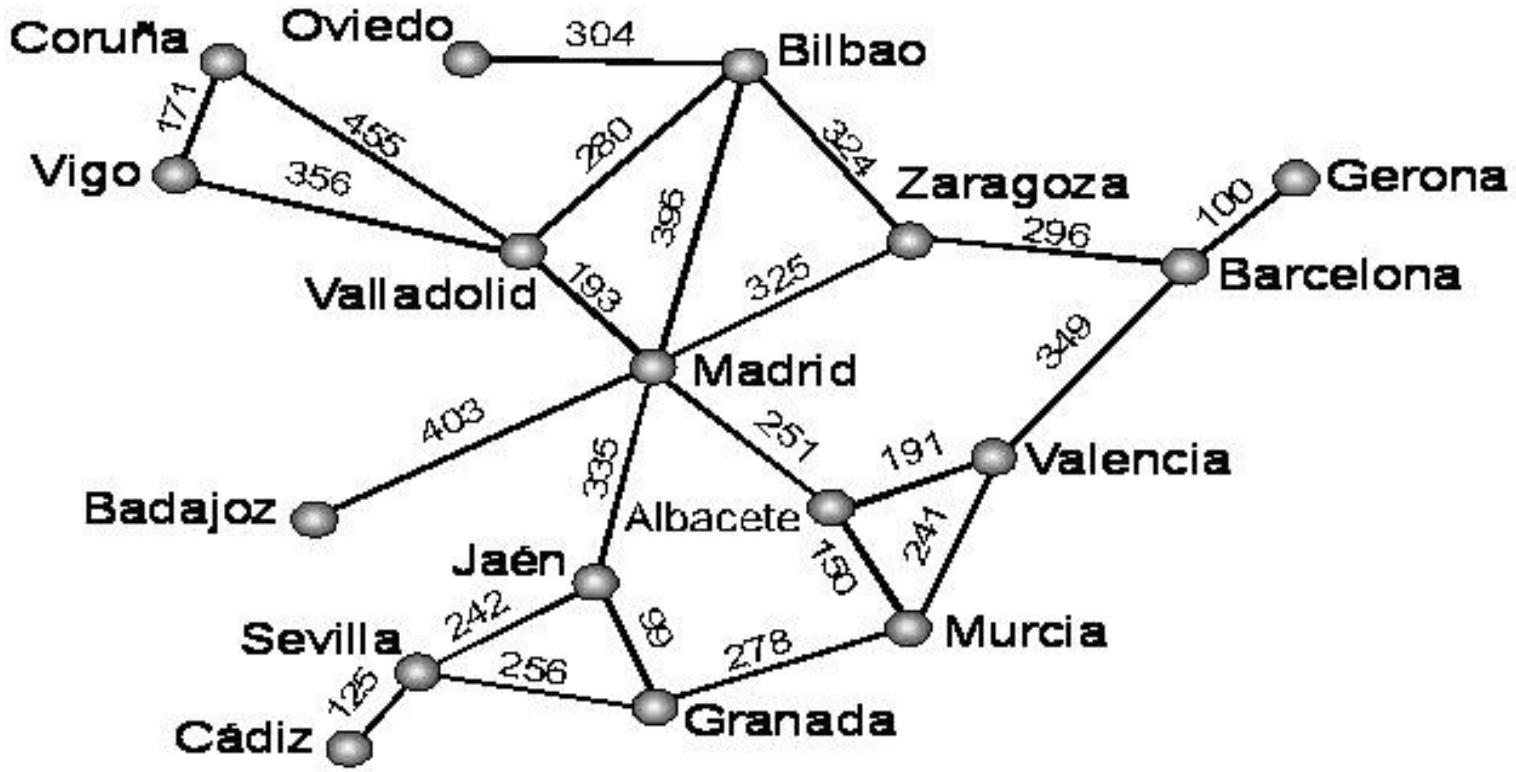
1.1. Clasificación de problemas

- Ejemplos de algoritmos eficientes.

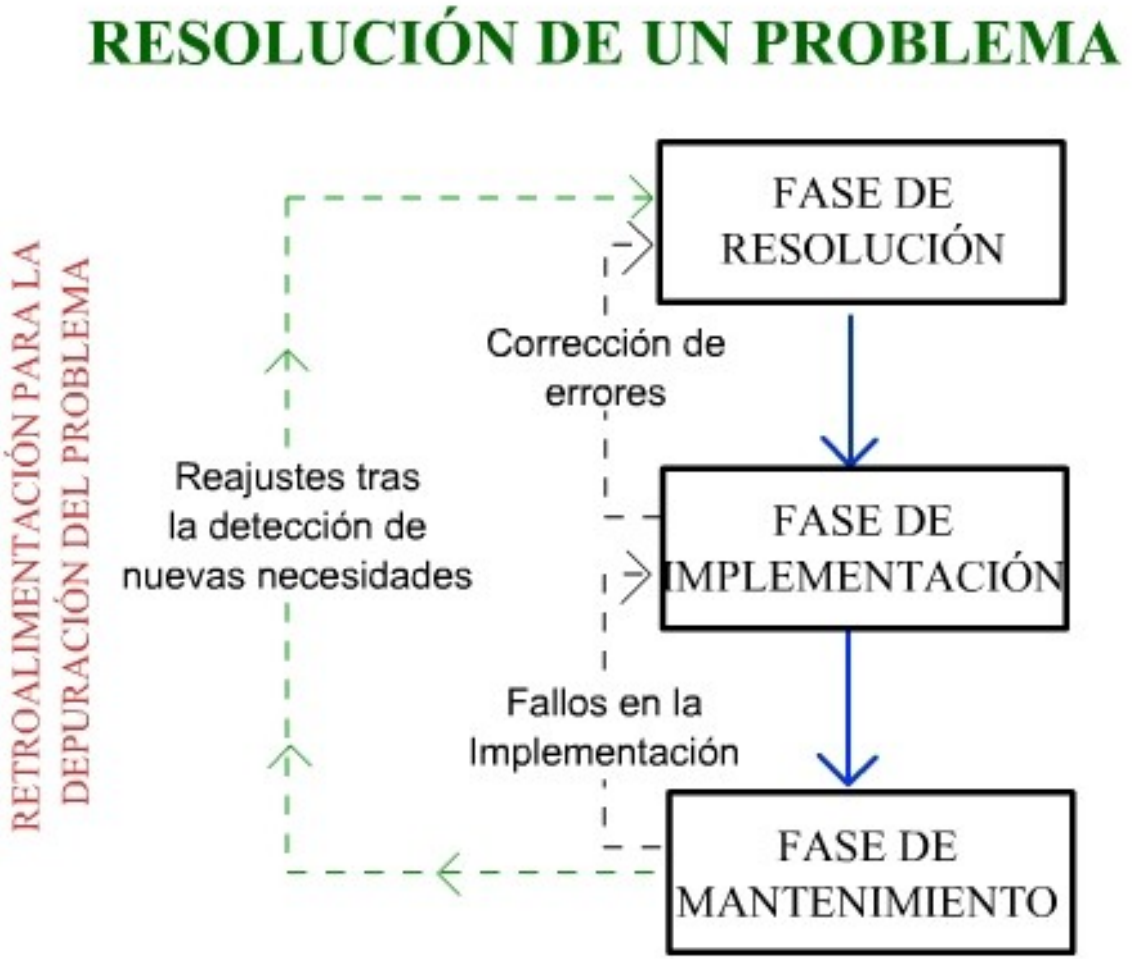


1.1. Clasificación de problemas

- Ejemplos de algoritmos eficientes.



1.2. Fases para resolver un problema



1.2. Fases para resolver un problema

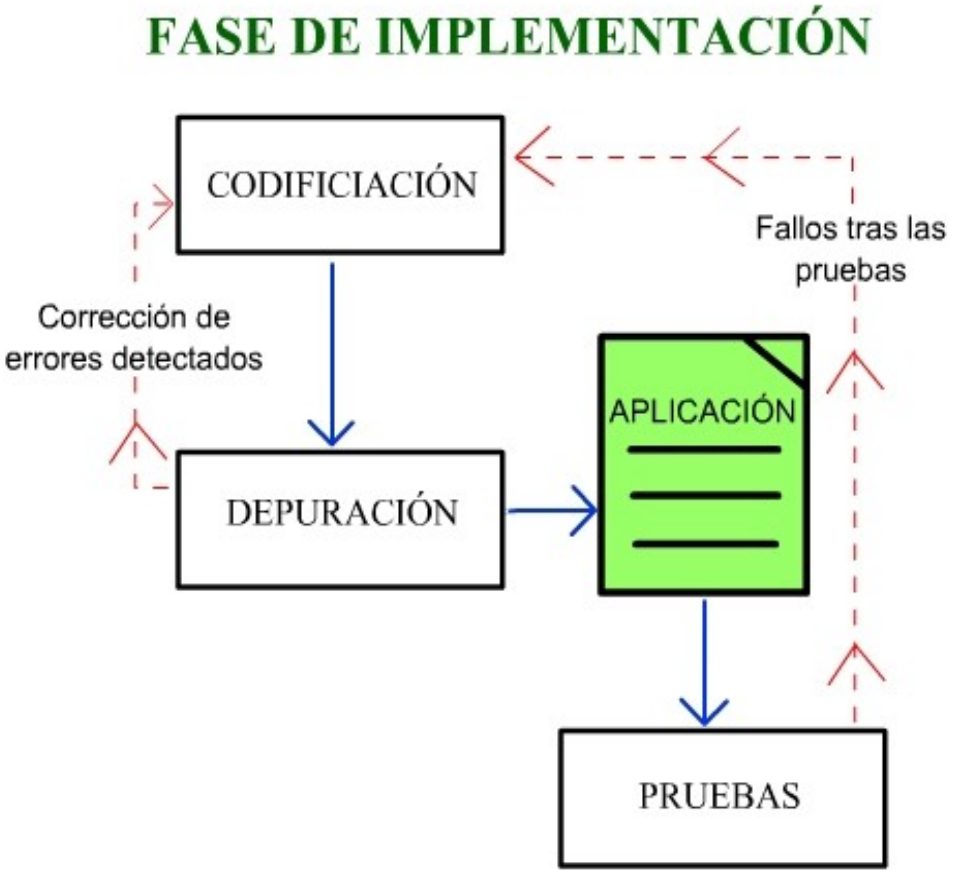
- **Fase de resolución**: El resultado de esta fase será un **algoritmo verificado**.

1.2. Fases para resolver un problema

- **Análisis del problema**:
  - Debemos estudiar el problema hasta comprenderlo.
  - Obtendremos las especificaciones del problema.
- **Diseño del problema**:
  - Describe la solución del problema: diagramas de diseño UML, diagramas de flujo, pseudocódigo...
  - Los problemas complejos debemos dividirlos en problemas más sencillos: **diseño descendente**.
- **Verificación manual del algoritmo:**
  - Probar el algoritmo con un conjunto de datos que incluya todos los casos posibles.

1.2. Fases para resolver un problema

- **Fase de Implementación**



1.2. Fases para resolver un problema

Codificación

- Se traduce el algoritmo diseñado en la fase anterior en el lenguaje de programación que usemos.

```
nombre ←  “Juan”
Si (nombre = “Pepe”)
     Escribir ( “El nombre del empleado es ”, nombre)
Caso Contrario
     Escribir (“No lo conozco, pero se llama ” , nombre)
Fin-Si
```

```
String nombre = “Juan” ;
if (nombre.equals(“Pepe”)){
     System.out.println(“El nombre del empleado es ”+ nombre) ;
}else {
     System.out.println(“No lo conozco, pero se llama ” + nombre) ;
}
```

1.2. Fases para la resolución de un problema

- **Codificación:** Tras la codificación, debemos traducir el lenguaje de alto nivel a código máquina: compilar el programa.
  - **Compiladores**. Realiza un análisis de todo el código mostrando un listado completo de errores. El proceso de traducción se realiza sólo una vez.
  - **Intérpretes**. Las instrucciones se analizan una a una. El código objeto no se guarda por lo que hay que realizar la traducción cada vez que el programa se utilice.

1.2. Fases para la resolución de un problema

- **Depuración:** Consiste en corregir los errores encontrados tras la fase de compilación y pruebas.
- El proceso de depuración incluye:
  - Compilar el programa.
    - Analizar el listado de errores.
    - Ejecutar el conjunto de prueba.
  - Editar y corregir el texto del programa.
  - Volver al paso inicial.

1.2. Fases para la resolución de un problema

- **Fase de explotación y mantenimiento:**
  - **Ajustes**. Tras implantar la aplicación, suele ser necesario realizar unos pequeños ajustes.
  - **Mejoras**. Se pueden mejorar diversos aspectos de la aplicación: rapidez, sencillez, etc.
  - **Adaptaciones**. Con el paso del tiempo el programa hay que adaptarlo a nuevos requirimientos, no previstos inicialmente.

2. Definición de algoritmo

- Proviene de **Mohammed Al-Khowarizmi,** matemático persa del siglo IX.
- Un **algoritmo** es:
  - Un método para resolver un problema mediante una serie finita y ordenada de pasos bien definidos.
- El **programador** es una persona que resuelve problemas mediante el diseño de algoritmos.

2. Definición de algoritmo

- **Características de un algoritmo**:
  - Preciso, sin ambigüedades.
  - Bien definido. Para un mismo dato de entrada siempre obtiene el mismo resultado.
  - Finito.
  - Independiente del lenguaje de programación.
  - No son únicos.

3. Diseño de algoritmos

- Existen diversas maneras de diseñar un algoritmo:
  - **Lenguaje natural**. Se utiliza una descripción textual para explicar cómo resolver el problema. Pueden ser adecuados para problemas muy sencillos pero no es recomendable en otro tipo de problemas.
  - **Diagramas de flujo, flujogramas u ordinogramas**. Representan de forma gráfica la secuencia lógica de pasos que sigue el programa.
  - **Pseudocódigo**. Utiliza una estructura similar a un lenguaje de programación pero no tan estricto, y más cercano al lenguaje natural.

3.1. Diagramas de flujo

- Características de los diagramas de flujo:
  - Son visuales.
  - Adecuados para diagramas no muy grandes.
  - Los símbolos están conectados por líneas.
  - Siempre hay una caja de inicio y otra de fin.

3.1. Diagramas de flujo

- Símbolos de los diagramas de flujo:

Instrucción de asignación

Instrucción de entrada o de salida

Inicio o Fin del algoritmo

Instrucción de control

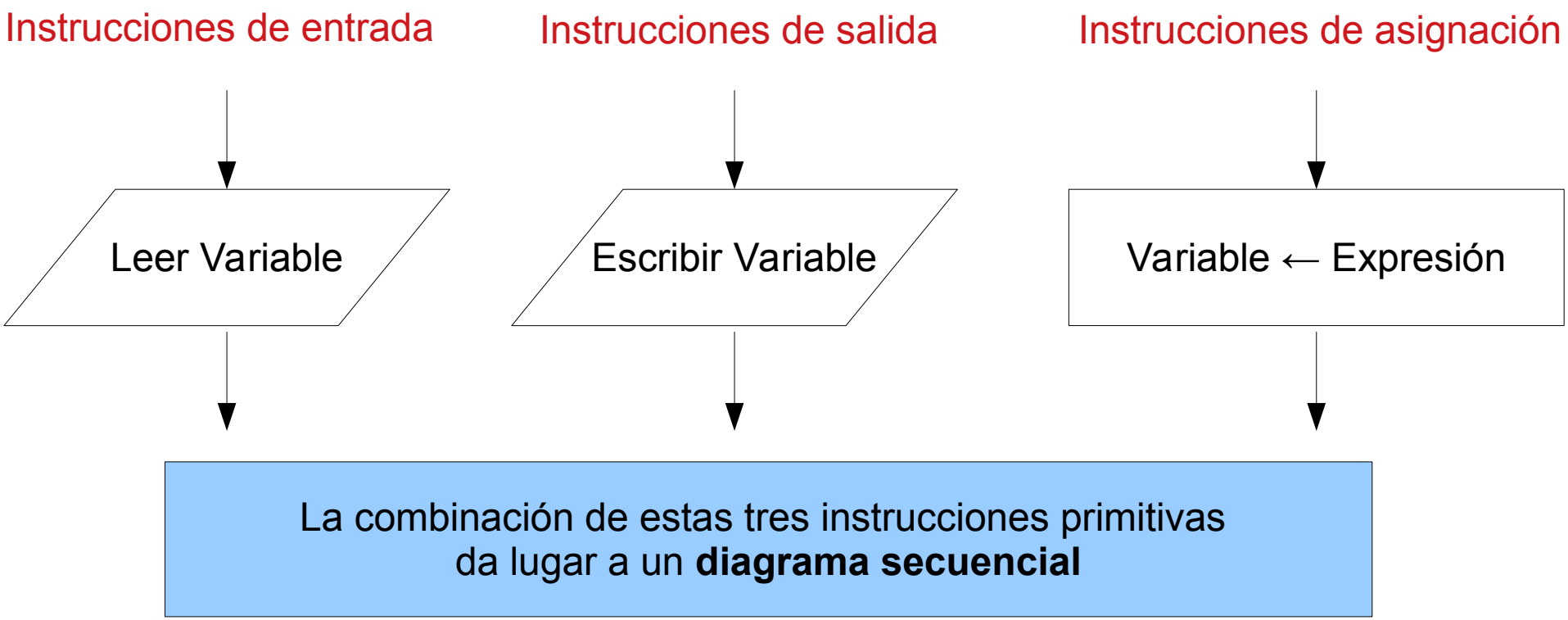
Llamada a un subprograma

Indica el orden de las acciones del algoritmo

Conector de reagrupamiento de una instrucción de control

3.1. Diagramas de flujo

- **Instrucciones primitivas**:



3.1. Diagramas de flujo

PROBLEMA EJEMPLO

Calcula el área de un triángulo

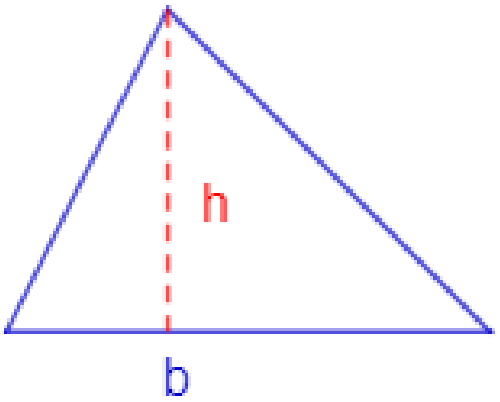
3.1. Diagramas de flujo

- Vamos a aplicar lo aprendido para resolver este problema.
  - Paso 1. Análisis del problema:
    - ¿Cómo se calcula el área de un triángulo?
    - ¿Qué datos necesito? ¿Qué valores pueden tomar esos datos?
    - ¿Qué resultado se mostrará?
    - ¿Cuándo acaba el algoritmo?

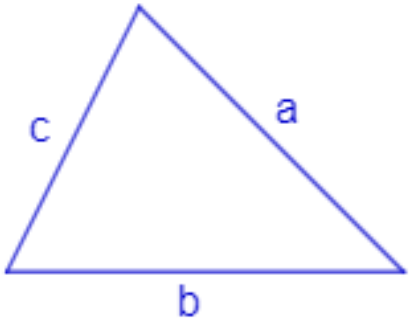
3.1. Diagramas de flujo

- Vamos a aplicar lo aprendido para resolver este problema.
  - Paso 1. Análisis del problema: Solución
    - ¿Cómo se calcula el área de un triángulo? Ambiguo

$$A = \frac{b \cdot h}{2}$$



$$A = \sqrt{s(s-a)(s-b)(s-c)}$$



$$s = \frac{a + b + c}{2}$$

3.1. Diagramas de flujo

- Vamos a aplicar lo aprendido para resolver este problema.
  - Paso 1. Análisis del problema: Solución
    - ¿Qué datos necesito? ¿Qué valores pueden tomar esos datos?

      La base y la altura del triángulo o los 3 lados

      Los valores deben ser números reales mayores que 0.
    - ¿Qué resultado se mostrará?

      El área del triángulo.
    - ¿Cuándo acaba el algoritmo?

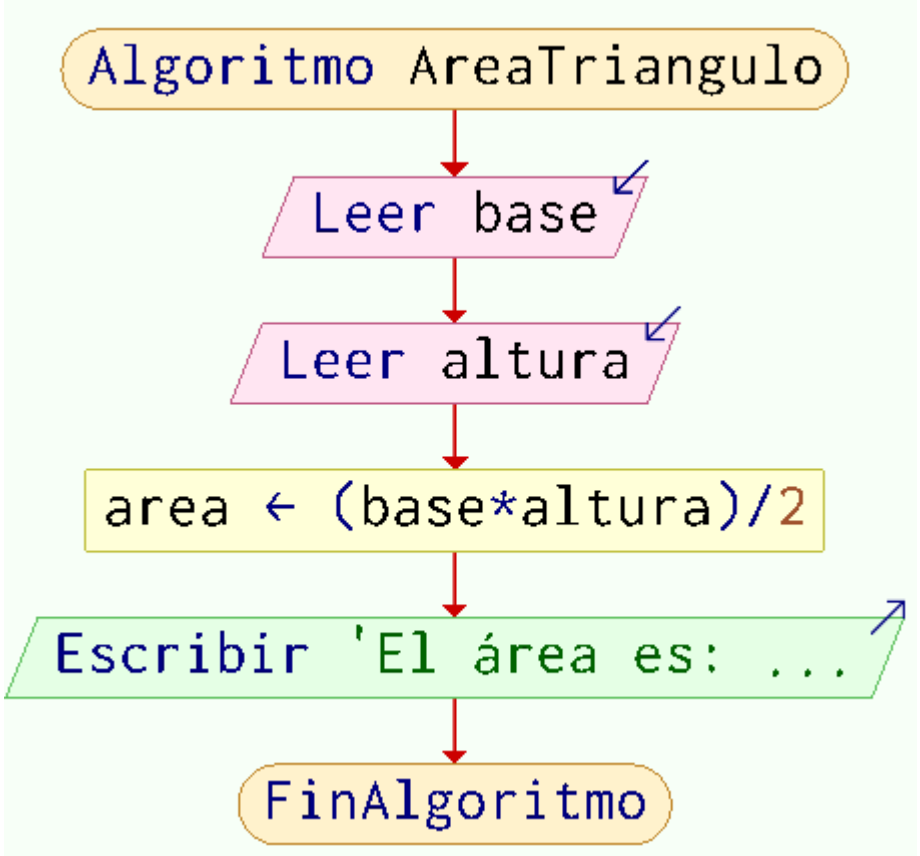
      Después de mostrar el resultado al usuario.

3.1. Diagramas de flujo

- Vamos a aplicar lo aprendido para resolver este problema.
  - Paso 2. Describir los pasos del algoritmo mediante diagrama de flujo.
    - Recuerda los símbolos de incio y fin.
    - Utiliza el símbolo de entrada para leer los datos necesarios.
    - Utiliza el símbolo de proceso para realizar cálculos y obtener el valor del área.
    - Utiliza el símbolo de salida para mostrar el resultado.

3.1. Diagramas de flujo

- Posible solución del Paso 2:



3.1. Diagramas de flujo

- Vamos a aplicar lo aprendido para resolver este problema.
  - Paso 3. Verificación del algoritmo: usar un conjunto de datos amplio de valores válidos y no válidos para evaluar la salida del algoritmo.
    - Base = 10, Altura = 5
    - Base = 5, altura = -5
    - Base = 0, altura = 3
    - Base = 5.5, altura = 8.6
    - Base = -3.4, altura = -5,6

3.1. Diagramas de flujo

- Vamos a aplicar lo aprendido para resolver este problema.
  - Paso 3. Verificación del algoritmo: usar un conjunto de datos amplio de valores válidos y no válidos para evaluar la salida del algoritmo.
    - Base = 10, Altura = 5 Solución: area = 25
    - Base = 5, altura = -5 Solución: area = No existe
    - Base = 0, altura = 3 Solución: area = No existe
    - Base = 5.5, altura = 8.6 Solución: area = 23.65
    - Base = -3.4, altura = -5,6 Solución: area = No existe

3.1. Diagramas de flujo

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

1) **Cambio de signo**. Diseña un algoritmo que pida un número y muestre por pantalla el número por pantalla cambiado de signo.

2) **Media aritmética**. Diseña un algoritmo que pida dos números y calcule la media de ambos.

3) **Conversor euros-pesetas**. Diseña un algoritmo que pida la cantidad de euros y calcule cuántas pesetas son.

4) **Volumen de un cono**. Diseña un algoritmo que calcule el volumen de un cono.

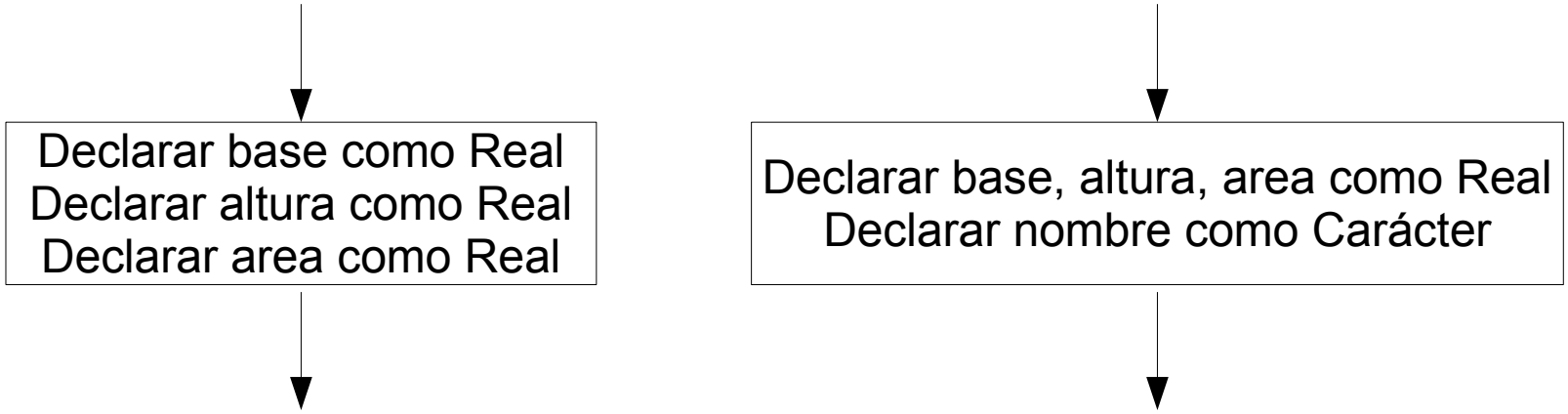
3.2. Variables y tipos de datos

- Concepto y definición de **variable**.
  - Es un dato que se aloja en la memoria del ordenador.
  - El nombre o identificador que damos a la variable está asociado a la dirección de memoria.
  - Los datos pueden ser de distintos tipos. En nuestros diagramas usaremos solo los siguientes:
    - **Carácter**. Para almacenar cadenas de caracteres, se encierran entre comillas simples o dobles. Ejemplo: ”Juan”, ’IES Zaidín-Vergeles’.
    - **Entero**. Para almacenar números sin decimales. Ejemplo: 0, -3, 143
    - **Real**. Para almacenar números con decimales. Ejemplo: -3.8, 0.44
    - **Lógico**. Para almacenar los valores verdadero o falso.

Como norma general no podremos hacer operaciones con variables de distinto tipo

3.2. Variables y tipos de datos

- **Declarar variables** en diagramas de flujo:
  - Se declaran al principo del diagrama, antes de leer, escribir u operar con la variable.
  - Se pueden declarar individualmente, o separadas por comas para variables del mismo tipo.



Ejercicio: Modifica los diagramas realizados anteriormente para añadir las declaraciones de variables necesarias.

3.2. Variables y tipos de datos

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

5) **Operaciones aritméticas**. Escribe un programa que sume, reste, multiplique y divida dos números introducidos por teclado.

6) **Salario semanal**. Escribe un programa que calcule el salario semanal de un empleado según las horas trabajadas, a razón de 12€ la hora.

3.2. Variables y tipos de datos

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

7) **Calcular nota**. Realiza un programa que calcule la nota que hace falta sacar en el segundo examen de Programación para obtener la media deseada, teniendo en cuenta que el primer examen cuenta el 40% y el segundo el 60%

Ejemplos:

```
Introduce la nota del primer examen: 7
¿Qué nota quieres sacar en el trimestre? 8.5
Para tener un 8.5 en el trimestre necesitas sacar un 9.5 en el segundo examen.
```

```
Introduce la nota del primer examen: 8
¿Qué nota quieres sacar en el trimestre? 7
Para tener un 7 en el trimestre necesitas sacar un 6.33 en el segundo examen.
```

4. Operadores y expresiones

- Una **expresión** es una sentencia que se puede evaluar dando como resultado un valor de un tipo de dato determinado (Caracter, Entero, Real o Lógico).
- Los **operadores** permiten realizar una determinada operación sobre valores, variables u otras expresiones.
- Tipos:
  - **Operadores aritméticos**.
  - **Operadores de comparación o relacionales**.
  - **Operadores lógicos**.

4.1. Operadores aritméticos

OPERADOR	NOMBRE	EJEMPLO	DESCRIPCIÓN
+	Suma	a+b+c 3+4+edad	Realiza la suma de los operandos
-	Resta	b-c numero-5	Resta los operandos
*	Multiplicación	a*b 3*4	Multiplica los operandos
/	División	a/b numero/10	Divide los operandos. Si son enteros el resultado es la división entera
% o mod	Módulo o resto	a%b b%2	Obtiene el resto de la división entera de los operandos

El operador % se emplea para saber si un número es divisible por otro.

Ejemplo: Un número es par si es divisible por 2: a % 2 == 0

Ejemplo: Un número es múltiplo de 5: a % 5 == 0

4.2. Operadores de comparación

- Comparan los valores de la expresión e indican si el resultado es verdadero o falso.

OPERADOR	NOMBRE	EJEMPLO	DESCRIPCIÓN
==	igual	a == b	a es igual a b
!=	distinto	a != b	a es distinto de b
<	menor que	a < b	a es menor que b
>	mayor que	a > b	a es mayor que b
<=	menor o igual que	a < = b	a es menor o igual que b
>=	mayor o igual que	a > = b	a es mayor o igual que b

Ejemplos:

nota >= 5
edad < 18
nombre == ”Jose”

5+8 < 3*4
nota+3 == 10
(3+4)*(5+6)+edad > 18

Las expresiones se pueden componer utilizando valores literales o variables
Se pueden utilizar paréntesis para indicar la precedencia de las operaciones

4.3. Operadores lógicos

- Los operadores lógicos se emplean para determinar la veracidad o falsedad de un conjunto de expresiones de comparación.

OPERADOR	NOMBRE	EJEMPLO	DEVUELVE VERDADERO CUANDO...
&&	y	(7 > 2) && (2 < 4)	las dos condiciones son verdaderas
||	o	(7 > 2) || (2 < 4)	al menos una de las condiciones es verdadera
!	no	!(7 > 2)	la condición es falsa

Ejemplo. Determinar si un número no está entre 1 y 100

n < 1 || n > 100

Ejemplo. Determinar si un número está entre 1 y 100

n >= 1 && n <= 100

4.3. Operadores lógicos

- Tabla de verdad de los operadores lógicos.

A	B	A && B	A || B	!A	!B
V	V	V	V	F	F
V	F	F	V	F	V
F	V	F	V	V	F
F	F	F	F	V	V

- Los operadores lógicos se pueden utilizar para formar expresiones más complejas (cada letra puede ser a su vez una expresión). Ejemplo:

A > B && C > D && E < F || G==H

Ejercicio: Sustituye las letras del ejemplo anterior por otras expresiones aritméticas, lógicas o de comparación para formar una expresión más compleja y evalúa su resultado.

4.4. Ejercicios (I)

- Evalúa las siguientes expresiones, siendo a=1, b=2, c=3, d=4:
  - a + b – c + d
  - a * b / c
  - 1 + a * b % c
  - a + d % b – c
  - d + c / b – a

4.4. Ejercicios (II)

- Evalúa las siguientes expresiones, siendo a=1, b=2, c=3:
  - a < c
  - b <= c
  - c <= a
  - a > b
  - b >= c
  - (a + b) >= c
  - (a + b) == c
  - a != b
  - (a+b) != c

4.4. Ejercicios (III)

- Evalúa las siguientes expresiones, siendo a=5, b=7, c=17:
  - c / b == 2
  - c % b <= a % b
  - b + c / a != c - a
  - (b < c) && (c == 7)
  - (c + 1 – b == 0) || (b == 5)

4.4. Ejercicios (IV)

- Escribe las expresiones para cada una de las fórmulas siguientes:

$$x = \frac{a + \frac{b}{c}}{a} + d$$

$$x = \frac{a + \frac{b}{c}}{\frac{a}{b} + c}$$

$$x = a \frac{a + b + \frac{c}{d * a}}{a + b * \frac{c}{d}}$$

4.4. Ejercicios (V)

- Escribe las expresiones que representen cada uno de los enunciados siguientes:
  - El promedio de tres calificaciones, c1, c2 y c3, mayor que 0 y menor o igual a 10.
  - El número n debe ser par y mayor que 10 o impar y menor que 5.
  - El doble de la edad de una persona debe ser mayor que 100 y su nacionalidad debe ser española.
  - El valor de la variable k debe ser igual a 0, igual a 3 o el doble de la misma debe ser mayor que 9.

4.4. Ejercicios (VI)

- Determina si los siguientes pares de expresiones son equivalentes (para los mismos valores de entrada deben dar el mismo resultado):

```
! (A == B)                    A != B
! ((A == B) || (A == C))      (A != B) && (A != C)
! ((A == B) && (C > D))       (A != B) || (C <= D)

!A && B                       B && !A
!A || B                       B || !A
!(A && B)                     A || B
A && B || C                   A && (B || C)
(A && B || C)                 !(A || B && C)
```

5. Estructuras de control de flujo

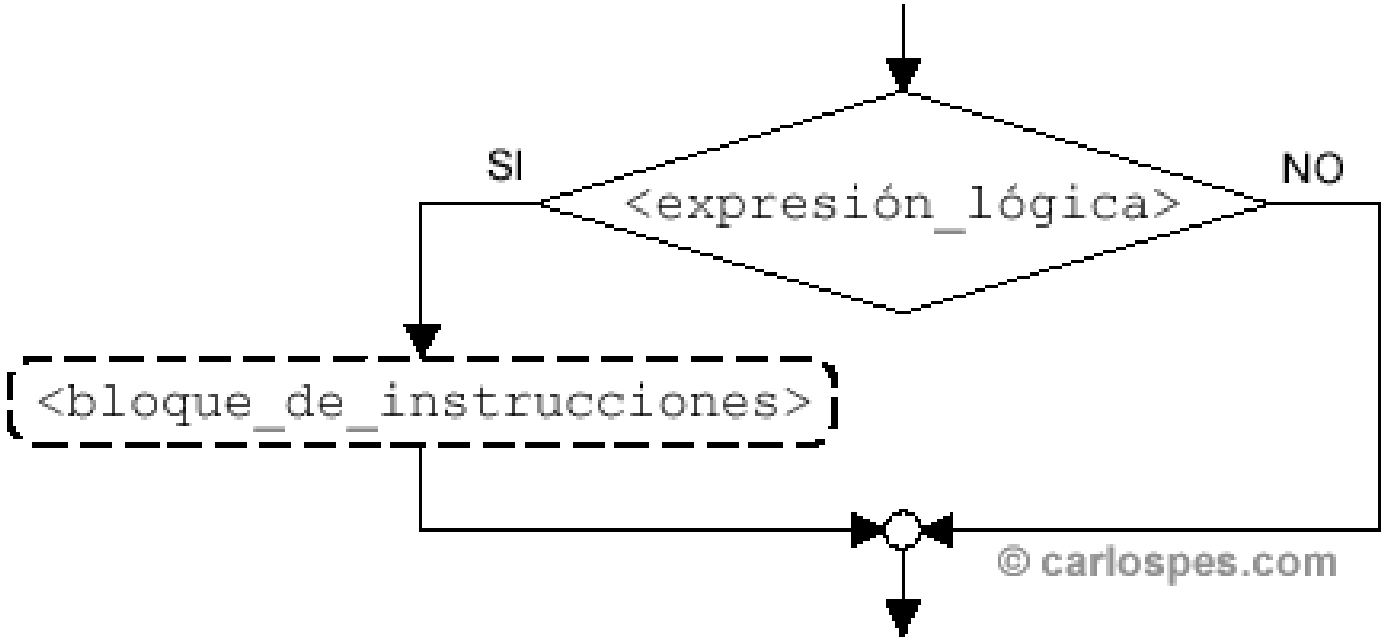
- Se puede demostrar matemáticamente que un algoritmo puede describirse usando solo 3 estructuras de control de flujo:
  - **Secuencial**. Las sentencias se ejecutan una detrás de la otra. Ya las hemos estudiado
  - **Condicional**. Se ejecutan o no, sentencias en función del valor de verdad de una condición. Operadores condicionales o lógicos
  - **Cíclica**. Se ejecutan repetidamente unas sentencias mientras se cumpla una determinada condición. Operadores condicionales o lógicos

5. Estructuras de control de flujo

- **Instrucciones condicionales**. Permiten bifurcar el código del programa en función de determinadas condiciones. Tipos:
  - Condicional simple.
  - Condicional doble.
  - Condiciónal múltiple.
- **Instrucciones cíclicas** o repetitivas. Permiten repetir bloques de código. Tipos:
  - **Mientras**. Se ejecuta mientras se cumple una condición. Primero pregunta y después se ejecuta.
  - **Hacer-Mientras**. Se ejecuta mientras se cumple una condición. Primero se ejecuta y después pregunta.
  - **Para**. Se ejecuta un número determinado de iteraciones.

5.1. Condicional simple

- Se evalúa si se cumple la condición, si es cierta se ejecuta el bloque de instrucciones, si no se continúa por el flujo normal del programa.

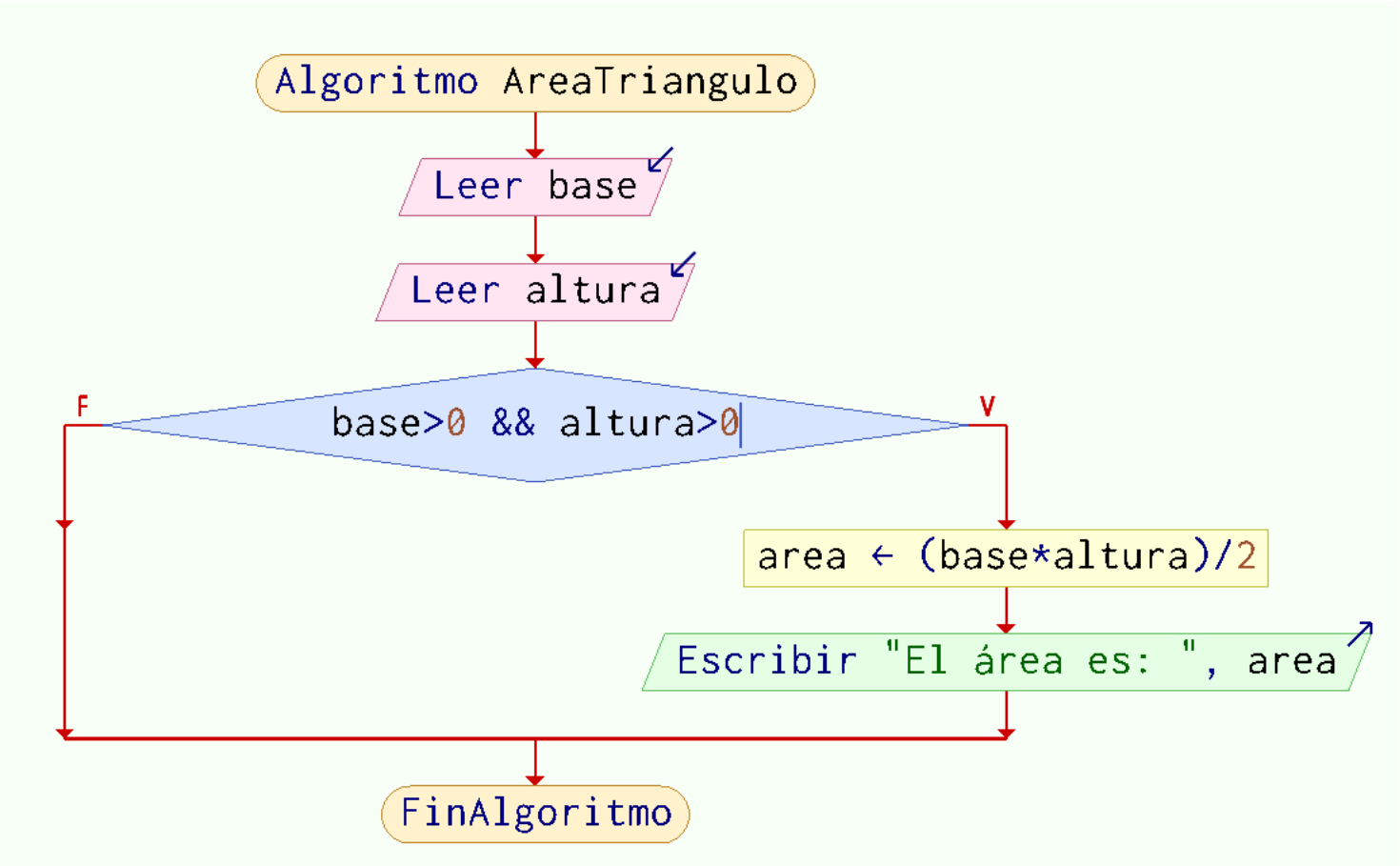


5.1. Condicional simple

Ejercicio: Utilizando un condicional simple, modifica el algorimo que calcula el área de un triángulo para que solo se realicen operaciones si los datos de entrada son válidos

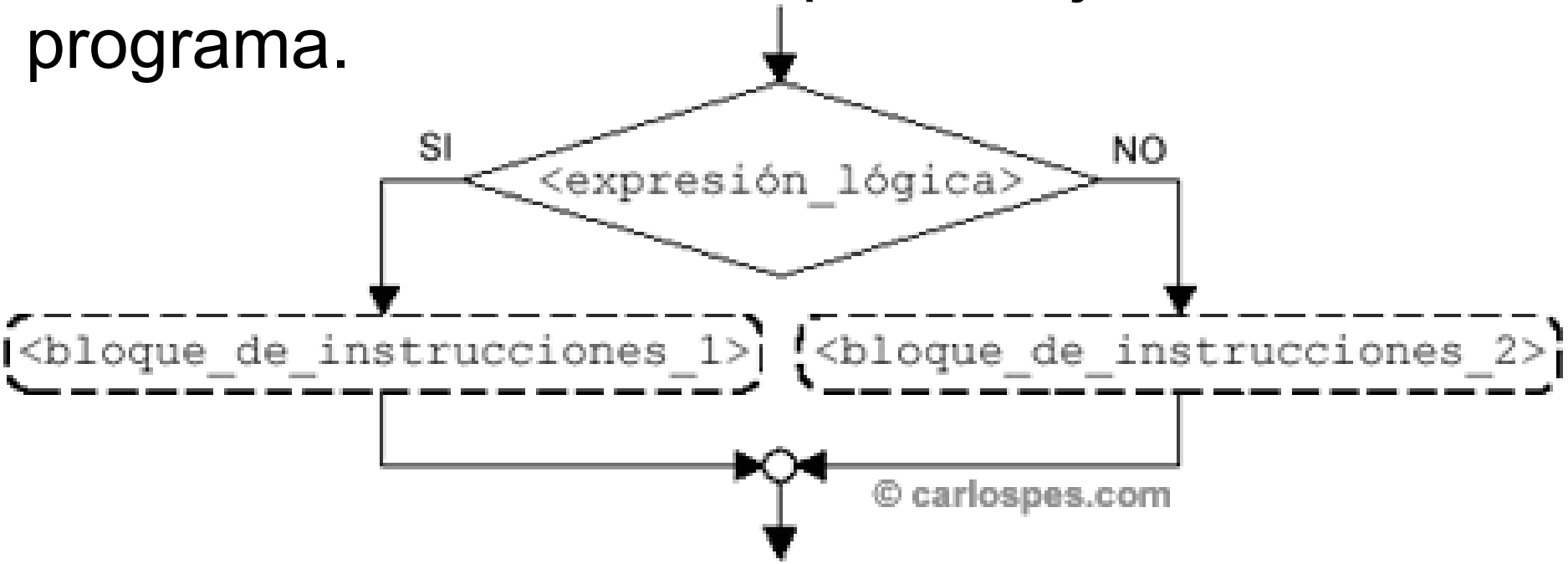
5.1. Condicional simple

Solución



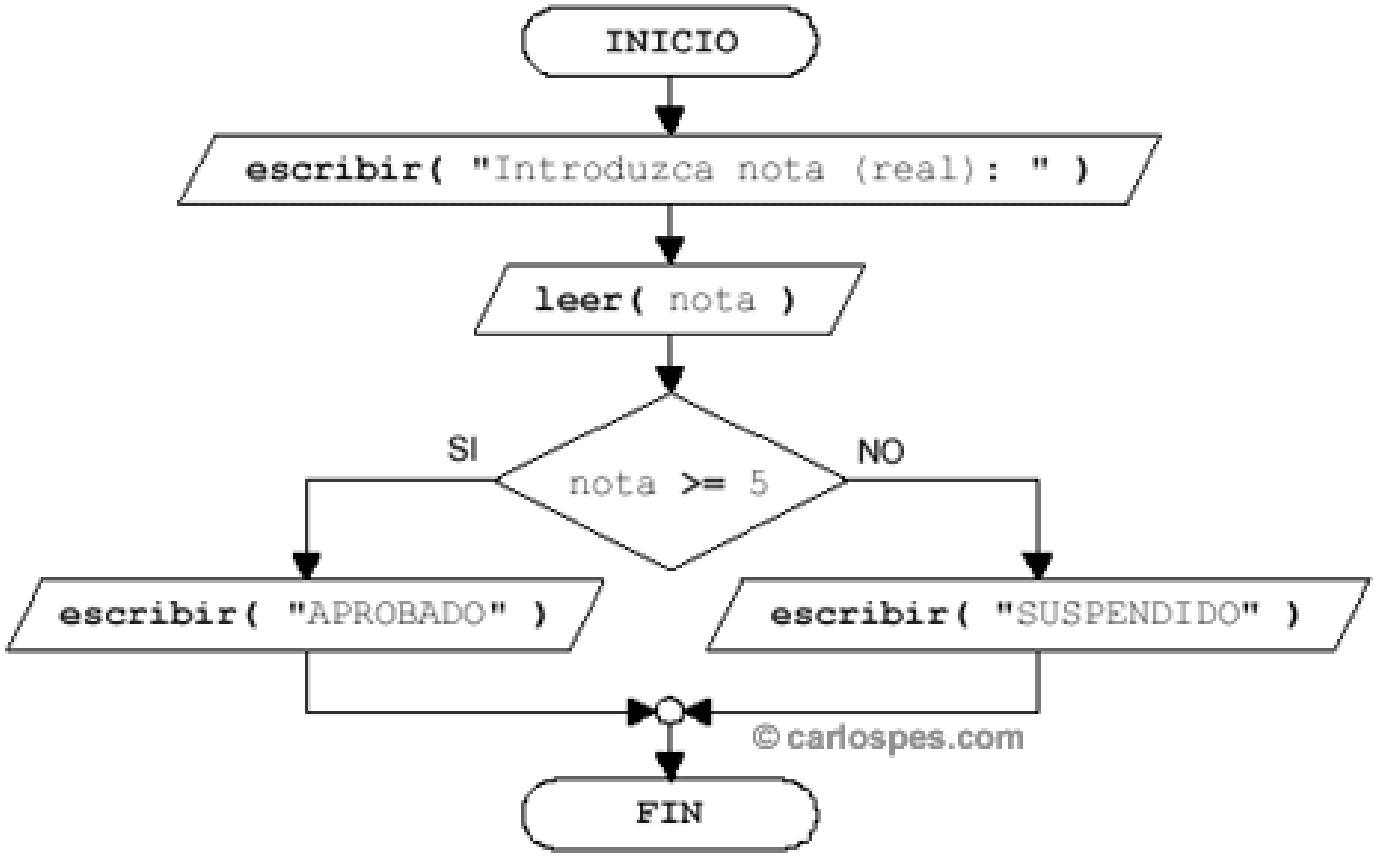
5.2. Condicional doble

- Se evalúa si se cumple la condición, si es cierta se ejecuta el bloque de instrucciones 1, si no se ejecuta el bloque de instrucciones 2. Finalmente, se continúa por el flujo normal del programa.



5.2. Condicional doble

- **Ejemplo**. Escribe un programa que pida una nota e indique si está aprobado o suspenso.

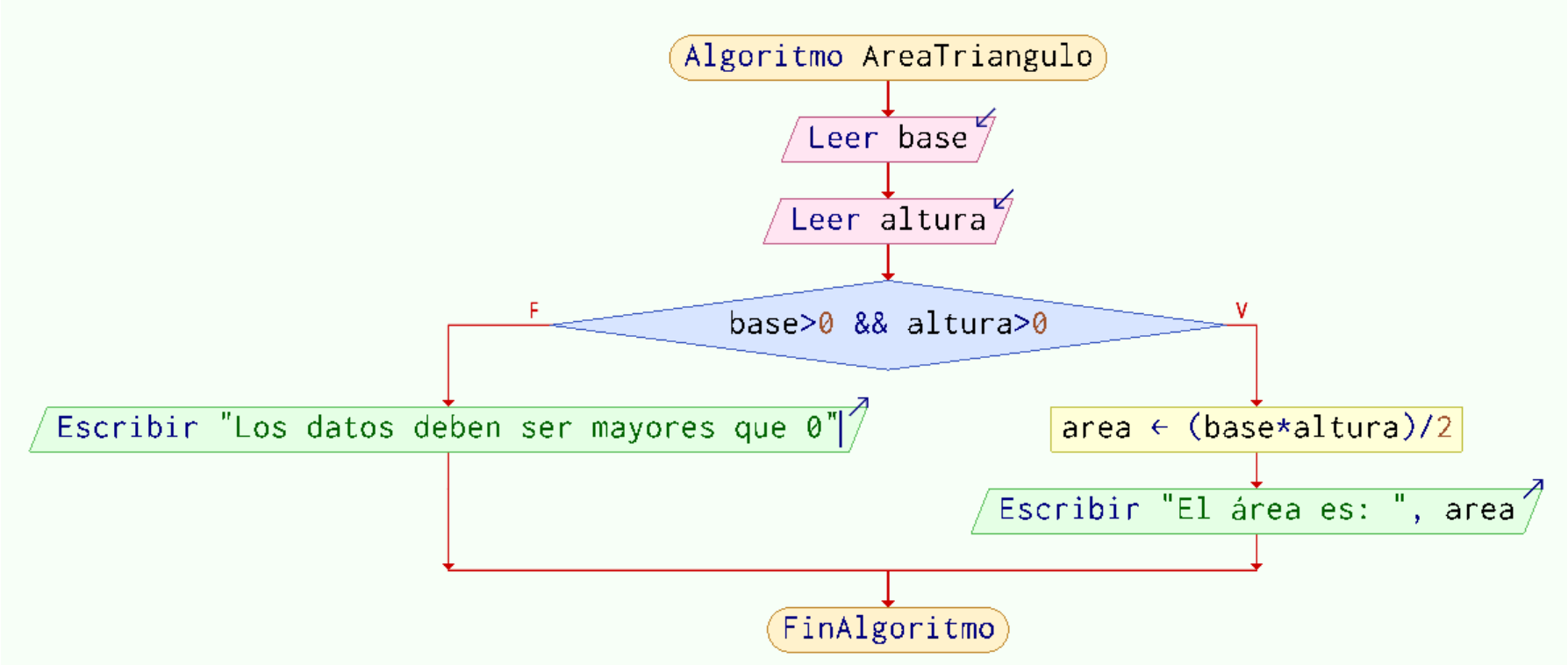


5.2. Condicional doble

Ejercicio: Utilizando un condicional doble, modifica el algorimo que calcula el área de un triángulo para que solo se realicen operaciones si los datos de entrada son válidos y en caso contrario muestre un mensaje de error al usuario

5.2. Condicional doble

Solución

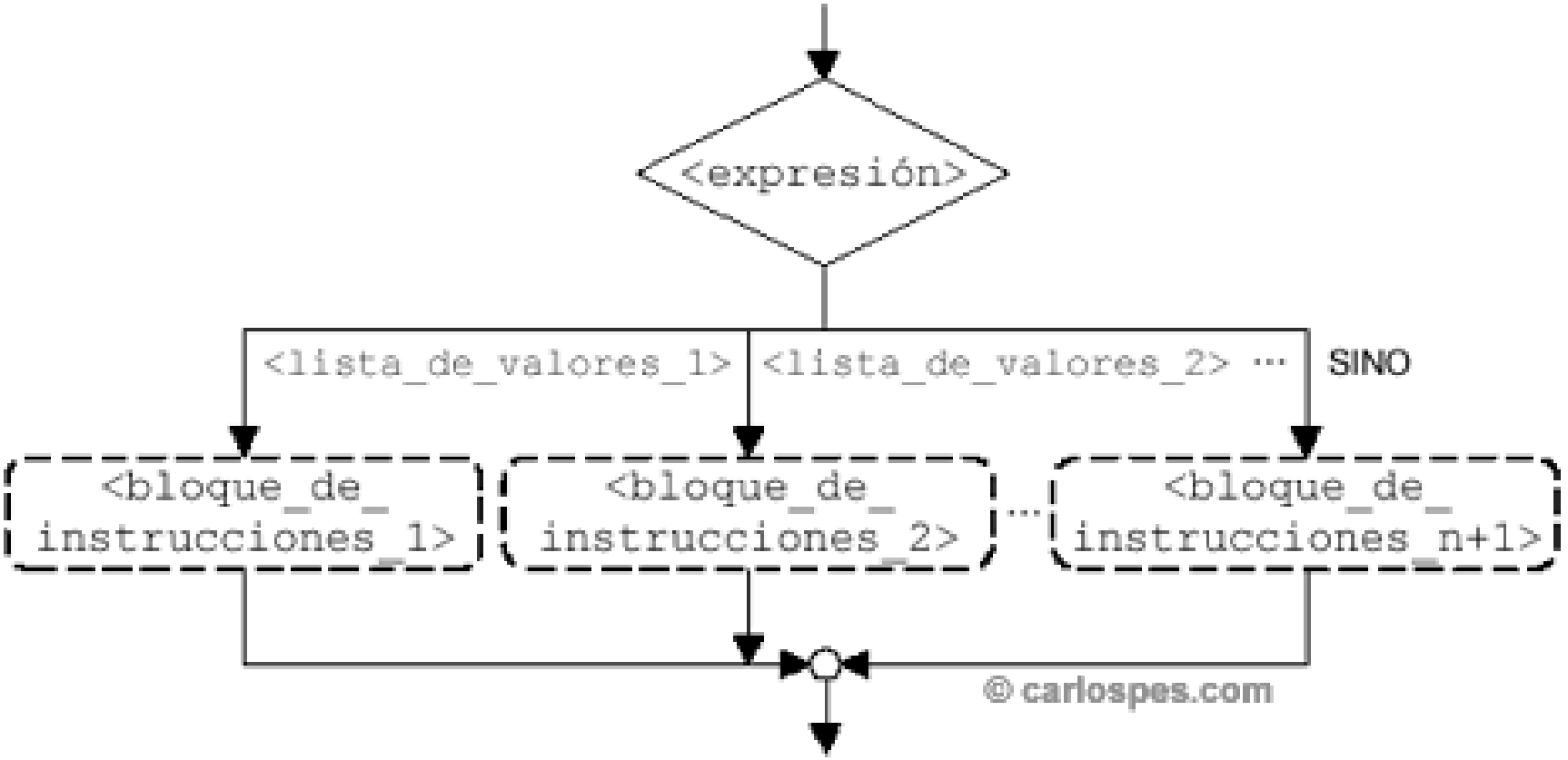


5.2. Condicional doble

Ejercicio: Modifica los diagramas anteriores para que se realicen los cálculos solo si los datos introducidos por el usuario son válidos. En caso contrario, deberá mostrarse un mensaje de error al usuario.

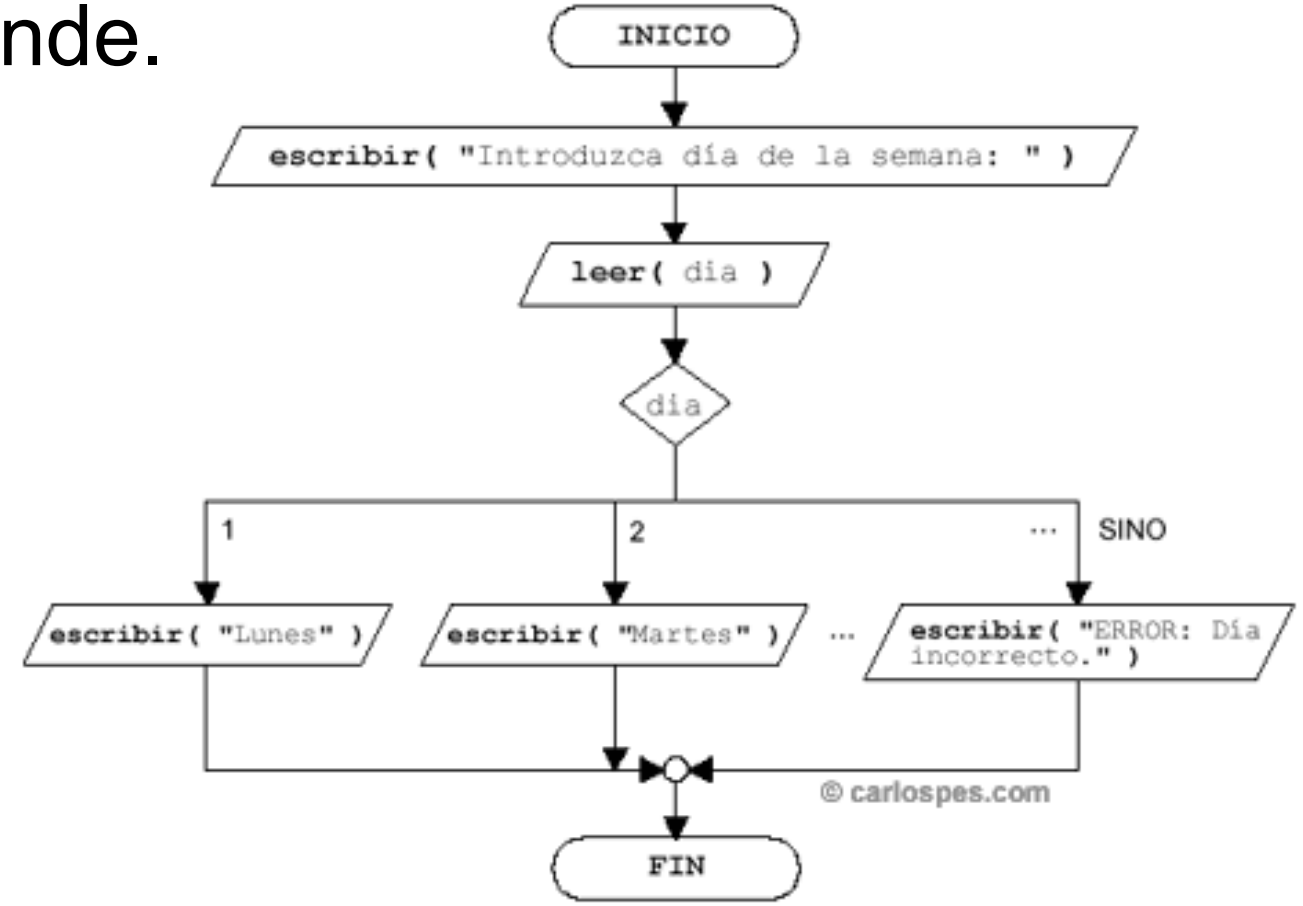
5.3. Condicional múltiple

- Se ejecuta la rama de instrucciones que coincide con la evaluación de la expresión.



5.3. Condicional múltiple

- **Ejemplo**. Escribe un programa que pida un número y escriba el día de la semana al que corresponde.



5.4. Ejercicios condicionales

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

8) **Suma dos números**. Escribe un programa que pida dos números y calcule la suma de ellos. Mostrará en pantalla ”La suma es mayor que cero” o ”La suma NO es mayor que cero” en función del resultado.

9) **Asignatura**. Escribe un programa que pida el día de la semana y muestre qué asignatura toca a primera hora.

10) **Saludo**. Escribe un programa que pida una hora (0-23) y muestre ”Buenos días”, ”Buenas tardes” o ”buenas noches”, según la hora. Se utilizarán los tramos 6 a 12, 13 a 20 y de 21 a 5, respectivamente.

5.4. Ejercicios condicionales

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

11) **Salario semanal 2**. Ampliaremos el ejercicio anterior para considerar las horas extras. Las primeras 40h se pagan a 12 euros. Las siguientes se pagan a 16 euros.

12) **Ecuación primer grado**. Escribe un programa que resuelva una ecuación de primer grado ($ax+b = 0$).

Ejemplos:

```
Este programa resuelve ecuaciones de primer grado del tipo ax + b = 0
Por favor, introduzca el valor de a: 2
Ahora introduzca el valor de b: 1
x = -0.5
```

```
Este programa resuelve ecuaciones de primer grado del tipo ax + b = 0
Por favor, introduzca el valor de a: 0
Ahora introduzca el valor de b: 7
Esa ecuación no tiene solución real.
```

5.4. Ejercicios condicionales

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

13) **Caída**. Escribe un programa que calcule el tiempo que tardará en caer un objeto desde una altura h. Aplica la fórmula $t=\sqrt{\frac{2h}{g}}$ siendo $g = 9.81\ m/s^2$

14) **Horóscopo**. Escribe un programa que nos muestre el horóscopo a partir del día y mes de nacimiento.

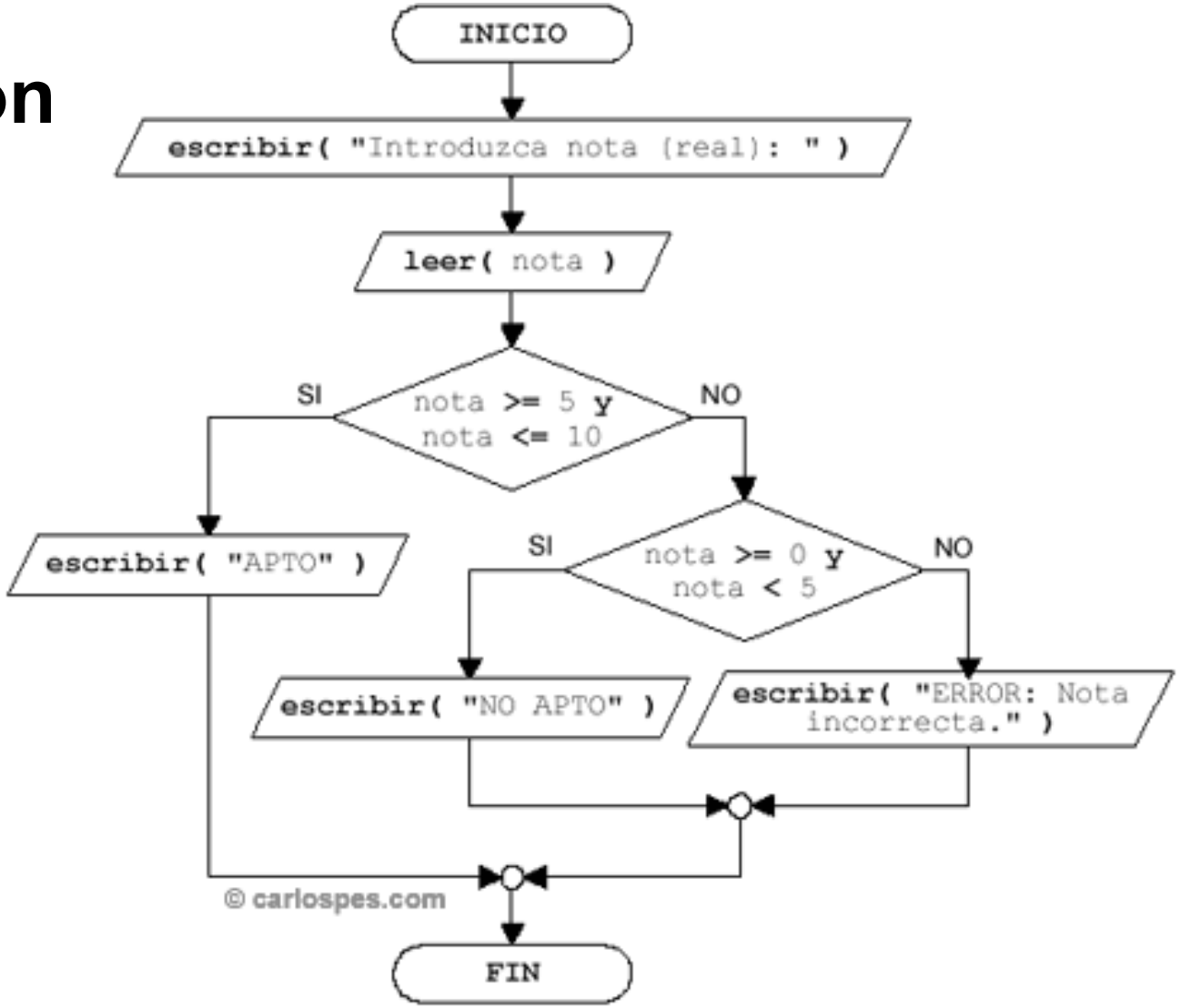
15) **Medianoche**. Escribe un programa que dada una hora determinada (horas y minutos), calcule el número de segundos que faltan para llegar a la medianoche.

5.5. Condicionales anidados

- **Ejemplo**: Diseña un algoritmo que pida una nota de un examen y muestre por pantalla:
  - ”APTO”, en caso de que la nota sea mayor o igual que 5.
  - ”NO APTO”, en caso de que la nota se mayor o igual que 0 y menor que 5.
  - ”Error, nota incorrecta”, en caso de que la nota sea menor que 0 o mayor que 10.

5.5. Condicionales anidados

- **Solución**



5.5. Condicionales anidados

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

16) **Ordenar 3**. Escribe un programa que ordene 3 números enteros introducidos por teclado.

17) **Nota Programación**. Escribe un programa que pide las notas de dos exámenes de programación. Si la media es mayor o igual a 5, el alumno estará aprobado y se mostrará la media. Si no, Se preguntará al usuario *¿Cuál ha sido el resultado de la recuperación? (apto/no apto)*. Si el resultado es apto, la nota será 5, en caso contrario se mantendrá la media anterior.

5.5. Condicionales anidados

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

18) **Piedra, papel, tijera**. Escribe un programa que implemente este juego para dos usuarios. Si alguno introduce una opción incorrecta se mostrará un mensaje de error.

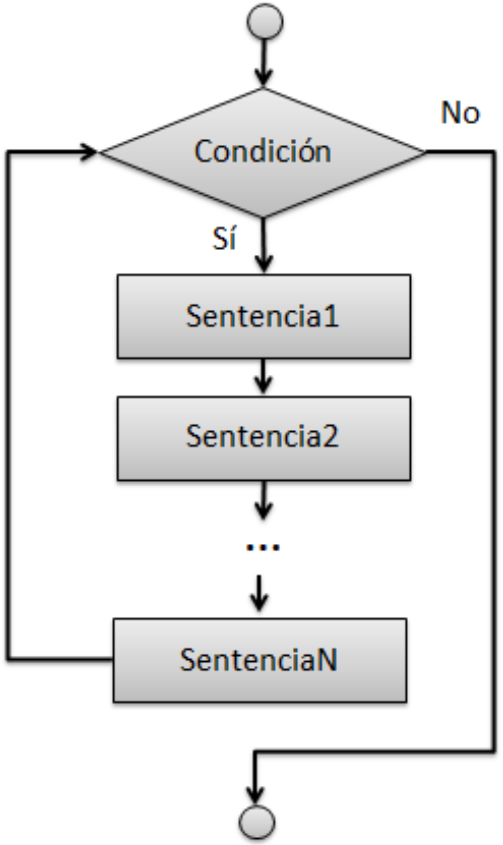
19) **Desayuno**. Escribe un programa que calcule el precio de un desayuno. Primero pregunta al usuario que ha tomado para comer: tostada, churros o donuts. Los churros valen 1,50€ y el donut 1€. En caso de tomar tostada, debe preguntar si es básica (1,20€) o especial (1,60€). Por último preguntará la bebida, zumo o café, a 1,80€ y 1,20€ respectivamente.

Recordemos...

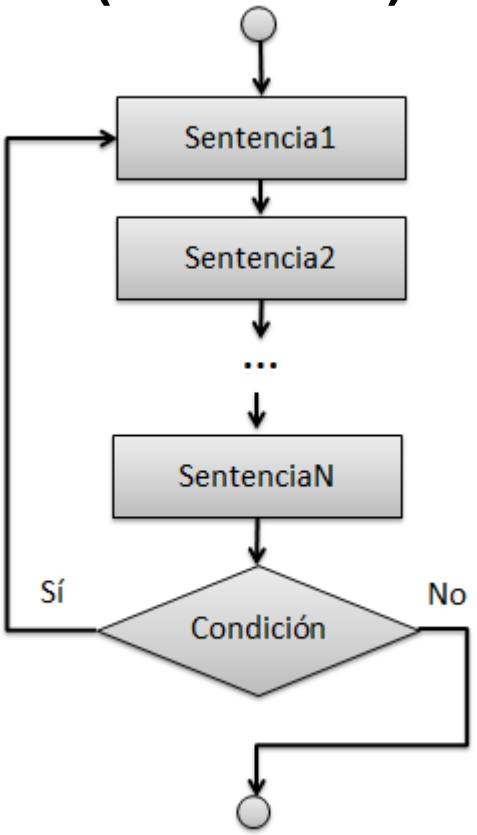
- **Instrucciones condicionales**. Permiten bifurcar el código del programa en función de determinadas condiciones. Tipos:
  - Condicional simple. Ya las hemos estudiado
  - Condicional doble. Ya las hemos estudiado
  - Condiciónal múltiple. Ya las hemos estudiado
- **Instrucciones cíclicas** o repetitivas. Permiten repetir bloques de código. Tipos:
  - **Mientras**. Se ejecuta mientras se cumple una condición. Primero pregunta y después se ejecuta.
  - **Hacer-Mientras**. Se ejecuta mientras se cumple una condición. Primero se ejecuta y después pregunta.
  - **Para**. Se ejecuta un número determinado de iteraciones.

5.6. Estructuras cíclicas

Mientras (While)

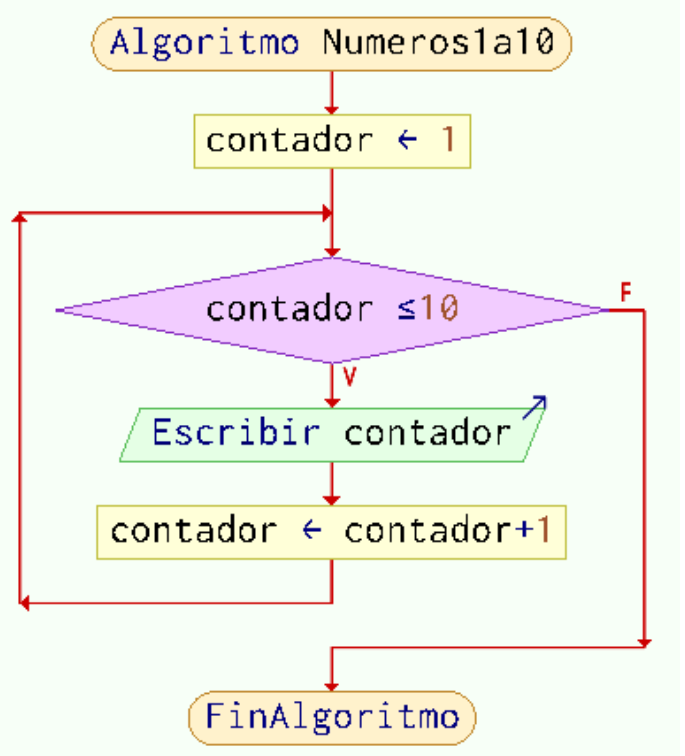


Hacer … Mientras (Do While)



5.6. Estructuras cíclicas

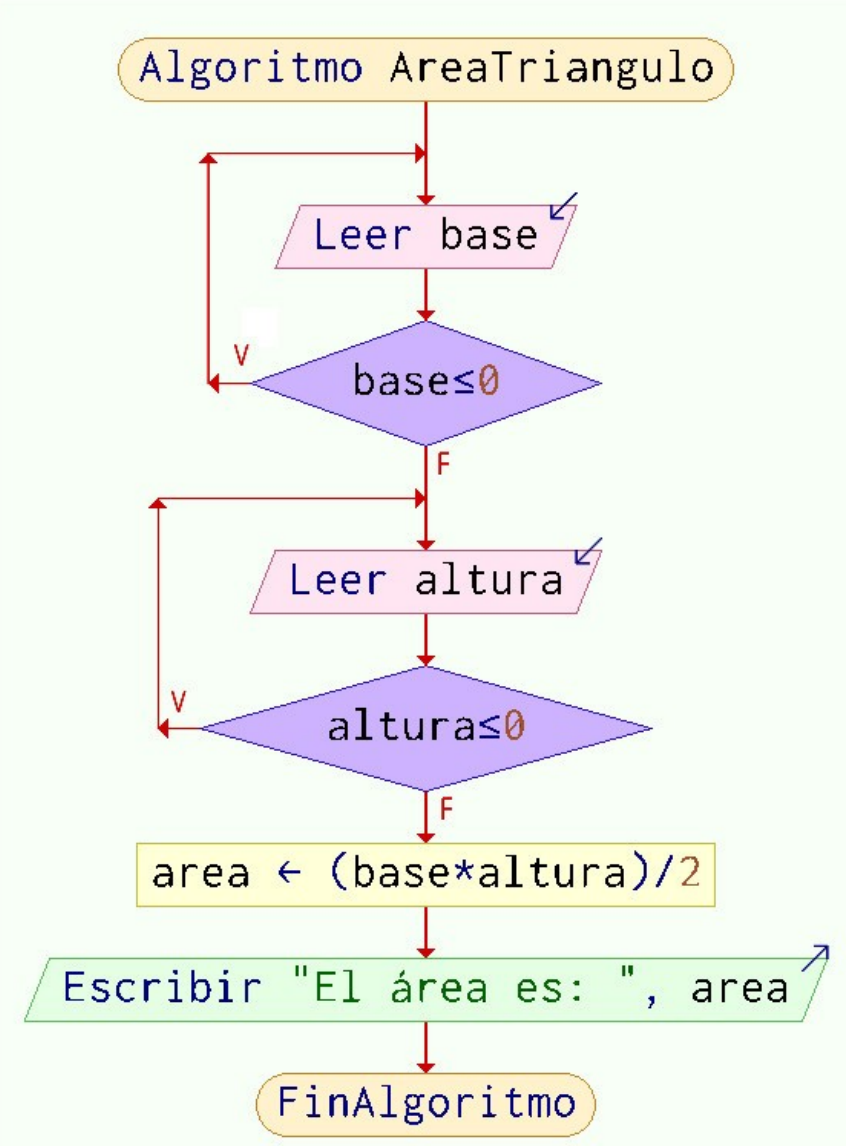
- **Ejemplo**. Escribe un programa que muestre los números del 1 al 10 por pantalla.



Ejercicio: Realiza la solución de este ejemplo con un bucle hacer-mientras

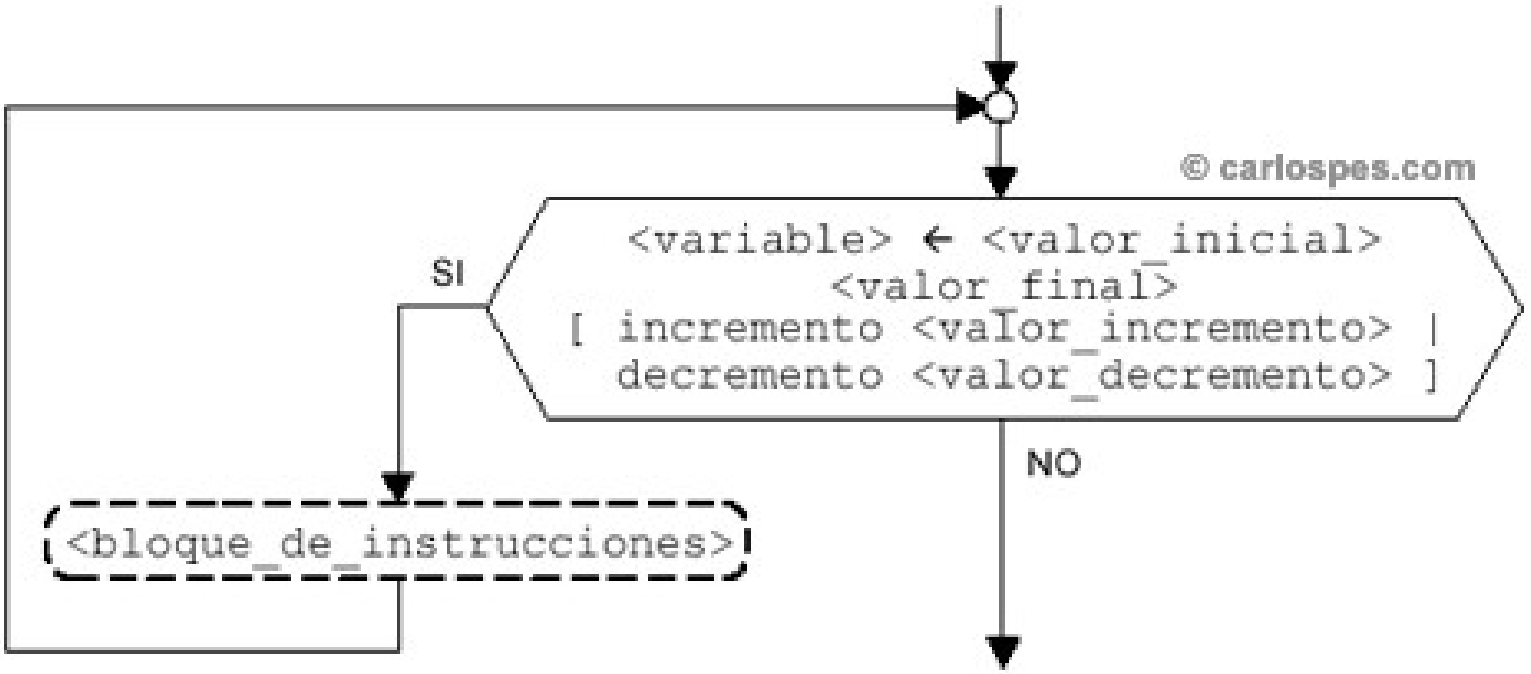
5.6. Estructuras cíclicas

- **Ejemplo**. Validar un dato de entrada con un bucle hacer-mientras.



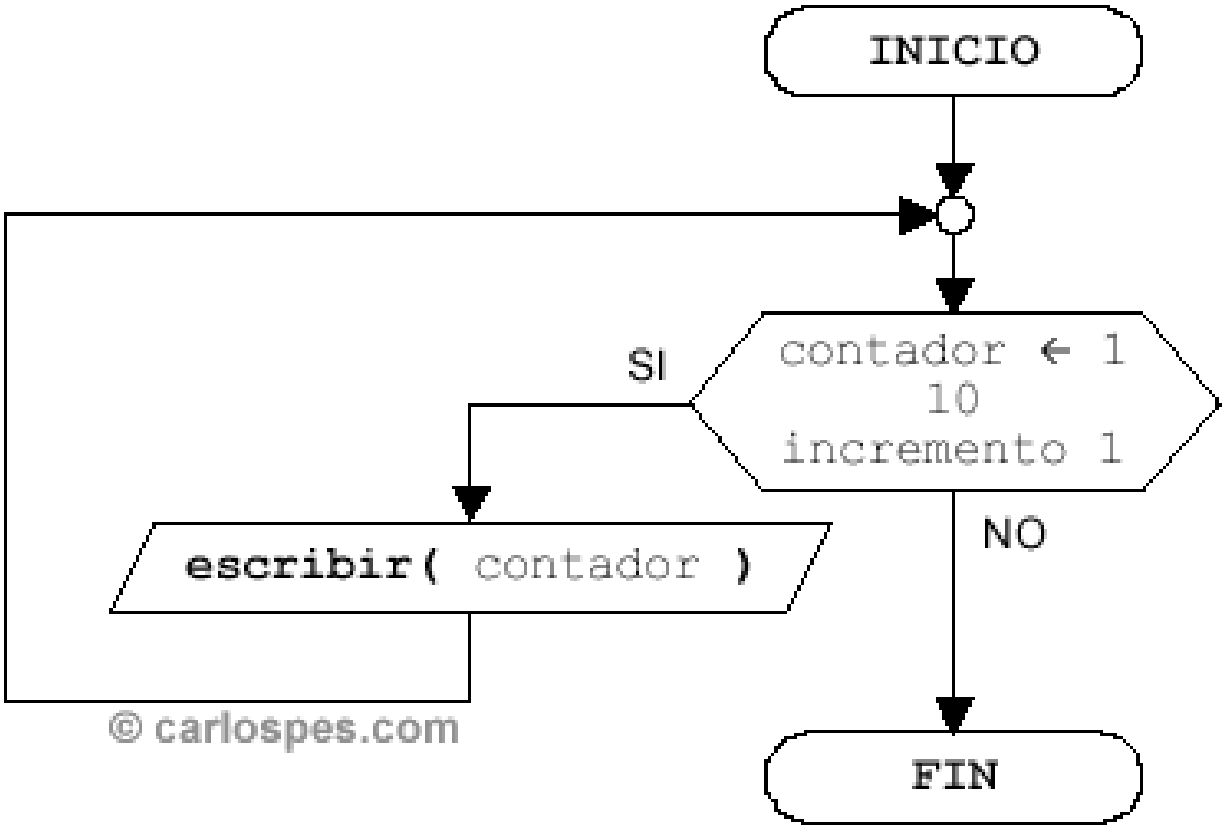
5.6. Estructuras cíclicas

Para
(For)



5.6. Estructuras cíclicas

- **Ejemplo**. Escribe un programa que muestre los números del 1 al 10 por pantalla.



5.6. Estructuras cíclicas

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

20) **Tabla de Multiplicar**. Escribe un programa que muestre la tabla de multiplicar de un número introducido por teclado.

21) **Caja fuerte**. Realiza un programa que pida un número de 4 cifras. Si no acertamos se mostrará el mensaje *"Lo siento, esa no es la combinación"* y si acertamos se nos dirá *"La caja fuerte se ha abierto"*. Disponemos de 4 intentos.

22) **Media números**. Escribe un programa que calcule la media de una serie de números positivos. El usuario indicará que ha terminado de introducir datos cuando introduzca un número negativo.

5.6. Estructuras cíclicas

- **Ejercicios**. Resuelve los siguientes problemas realizando todos los pasos estudiados.

  23) **Fibonacci**. Escribe un programa que muestre los n primeros términos de la serie de Fibonacci. El valor de n será introducido por teclado.

  24) **Positivos-negativos**. Escribe un programa que pida al usuario 10 números y muestre por pantalla cuántos son positivos y cuántos negativos.

  25) **Números**. Escribe un programa que pida números hasta que se introduzca un número negativo. Se mostrará cuántos números se han introducido, la media de los impares y el mayor de los pares. El número negativo solo se utiliza para finalizar.

6. Pseudocódigo

- Es un código intermedio entre el lenguaje natural y el lenguaje de programación.
- Evita los detalles de sintáxis de un lenguaje concreto.
- No se rige por normas estrictas.
- No son tan visuales como los diagramas de flujo.
- Son adecuados para una descripción formal de un algoritmo, independiente del lenguaje.

6. Pseudocódigo

- Estructura de un algoritmo en pseudocódigo:

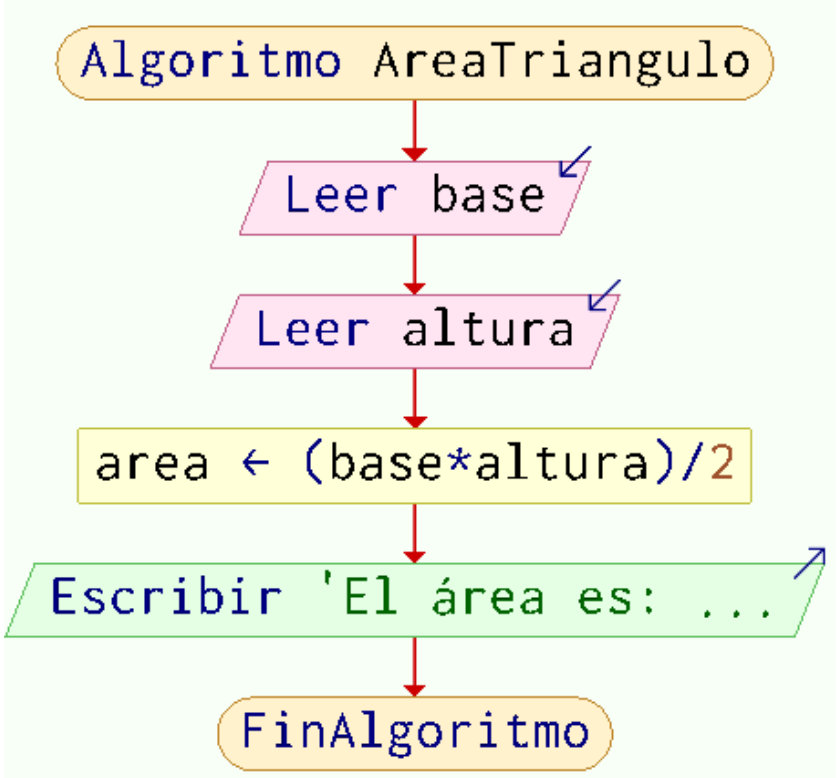
```
Algoritmo <nombre alg>
Constantes
   <nombre>: <tipo>, <nombre>: <tipo>,…
Variables
   <nombre>: <tipo>, <nombre>: <tipo>,…

Inicio
   <Instrucciones>
Fin
```

La estructura del pseudocódigo puede variar en función del autor del mismo dado su carácter flexible. Lo importante es saber identificar cada uno de los elementos estudiados en el diseño de algoritmos.

6. Pseudocódigo

- Ejemplo:

```
Algoritmo AreaTriangulo
     Definir base, altura Como Real
     Definir area Como Real

     Escribir "Introduce la base del triángulo"
     Leer base;
     Escribir "Introduce la altura del triángulo"
     Leer altura

     area <- (base*altura) / 2
     Escribir "El área del triángulo es ", area

FinAlgoritmo
```



6. Pseudocódigo

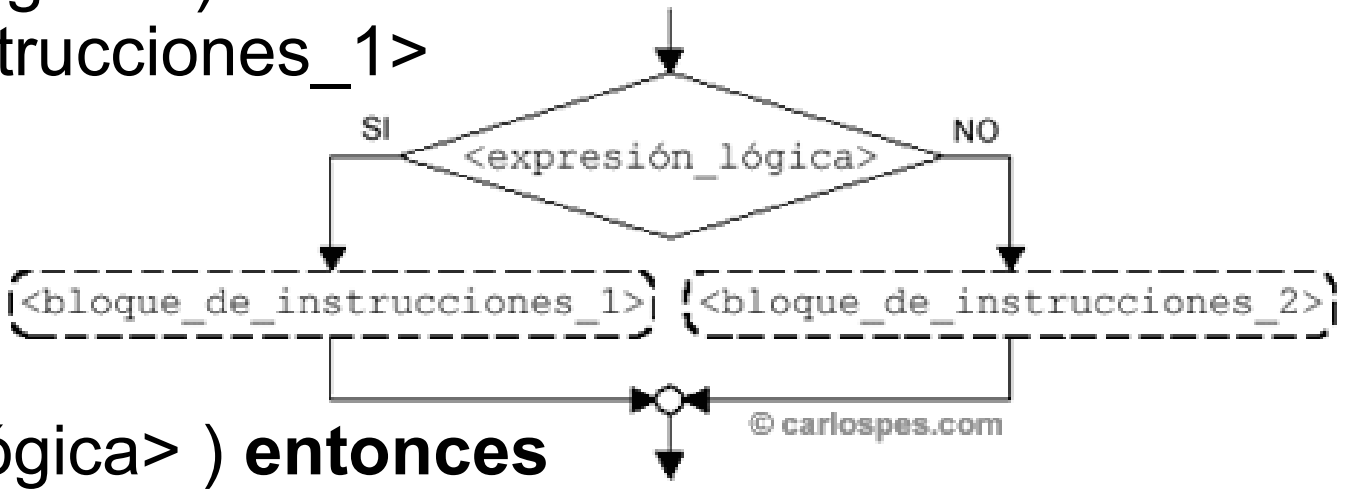
Ejercicio: Escribe el pseudocódigo de los diagramas secuenciales realizados previamente

Ejercicios 1 a 7

6. Pseudocódigo

- **Condicional simple y doble**:

Si (<expresión_lógica>) **entonces**
  <bloque_de_instrucciones_1>
FinSi



Si (<expresión_lógica>) **entonces**
  <bloque_de_instrucciones_1>
SiNo
  <bloque_de_instrucciones_2>
FinSi

6. Pseudocódigo

- Ejemplo:

```
Algoritmo AreaTriangulo
     Definir base, altura Como Real
     Definir area Como Real

     Escribir "Introduce la base del triángulo"
     Leer base;
     Escribir "Introduce la altura del triángulo"
     Leer altura;

     Si base>0 && altura>0 Entonces
          area <- (base*altura) / 2
          Escribir "El área del triángulo es ", area
     SiNo
          Escribir "ERROR: Los datos de entrada deben ser positivos"
     FinSi

FinAlgoritmo
```

6. Pseudocódigo

- **Condicional anidado**:

Si (<expresión_lógica>) **entonces**
  <bloque_de_instrucciones_1>
SiNo Si (<expresión_lógica>) **entonces**
  <bloque_de_instrucciones_2>
SiNo Si (<expresión_lógica>) **entonces**
  <bloque_de_instrucciones_3>
...
SiNo
  <bloque_de_instrucciones_N>
FinSi

6. Pseudocódigo

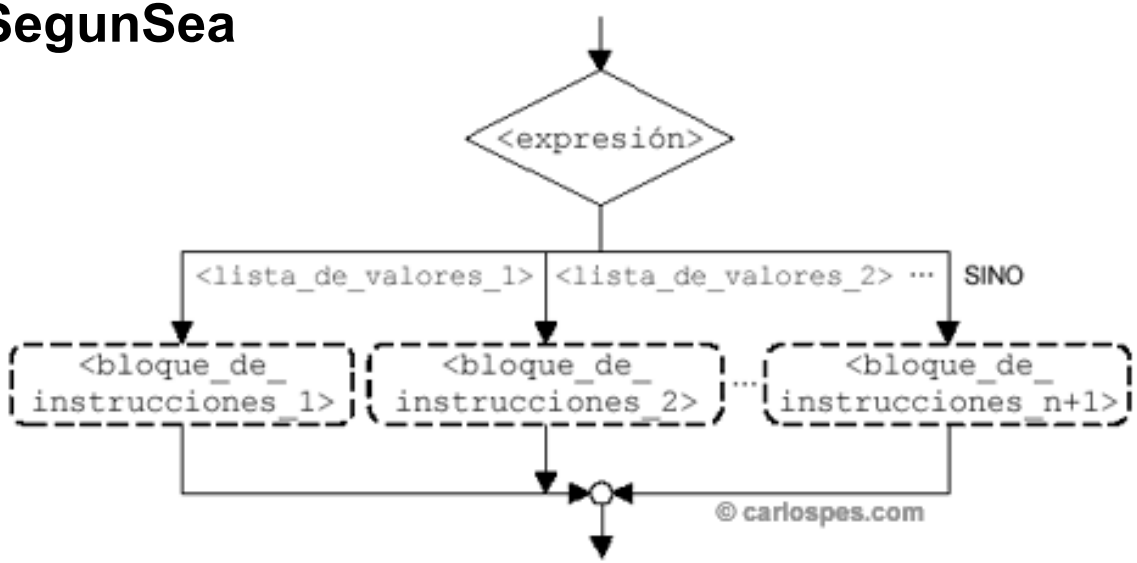
- Ejemplo:

```
Algoritmo Ordenar3
    Definir a,b,c, Como Entero

    Escribir "Introduce el primer número"
    Leer a;
    Escribir "Introduce el segundo número"
    Leer b;
    Escribir "Introduce el tercer número"
    Leer c;

    Si a>=b && a>=c && b>=c Entonces
        Escribir "Números ordenados ", a,b,c
    SiNo Si a>=b && a>=c && c>=b Entonces
        Escribir "Números ordenados ", a,c,b
    SiNo Si b>=a && b>=c && a>=c Entonces
        Escribir "Números ordenados ", b,a,c
    ...
    SiNo
        Escribir "Números ordenados ", c,b,a
    FinSi

FinAlgoritmo
```

6. Pseudocódigo

Condicional múltiple:

SegunSea(<expresión>)
  <lista_de_valores_1> : <bloque_de_instrucciones_1>
  <lista_de_valores_2> : <bloque_de_instrucciones_2>
  ...
  <lista_de_valores_n> : <bloque_de_instrucciones_n>
        [sino : <bloque_de_instrucciones_n+1>]
FinSegunSea



6. Pseudocódigo

- Ejemplo:

```
SegunSea ( dia )
         1 : escribir( "Lunes" )
         2 : escribir( "Martes" )
         3 : escribir( "Miércoles" )
         4 : escribir( "Jueves" )
         5 : escribir( "Viernes" )
         6 : escribir( "Sábado" )
         7 : escribir( "Domingo" )
         sino : escribir( "ERROR: Día incorrecto." )
FinSegunSea
```

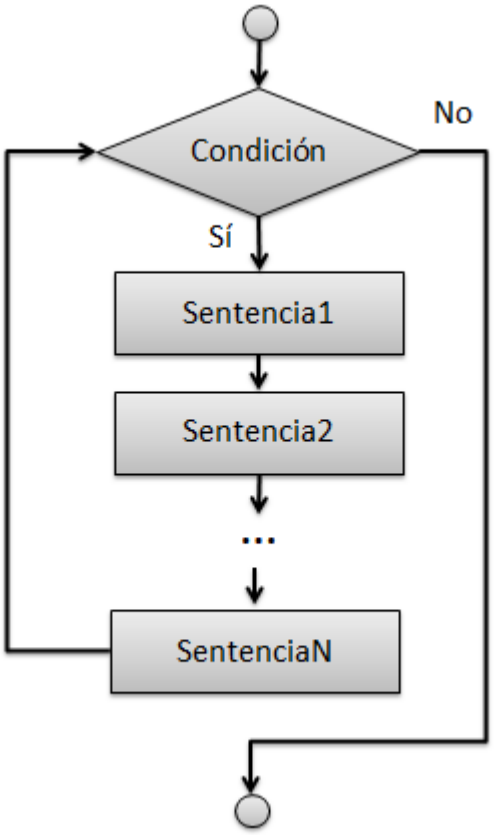
6. Pseudocódigo

Ejercicio: Escribe el pseudocódigo de los diagramas con estructuras condicionales realizados previamente

Ejercicios 8 a 19

6. Pseudocódigo

Bucle mientras (while):



Mientras (<expresión_lógica>)

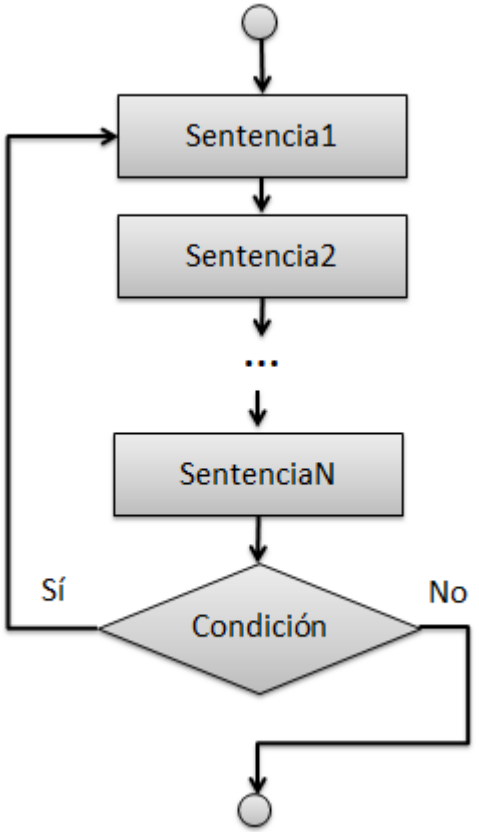
<bloque_de_instrucciones>

FinMientras

```
contador ← 1
Mientras ( contador <= 10 )
      escribir( contador )
      contador ← contador + 1
FinMientras
```

6. Pseudocódigo

Bucle hacer-mientras (do-while):



Hacer

<bloque_de_instrucciones>

Mientras (<expresión_lógica>)

```
contador ← 1
Hacer
      escribir( contador )
      contador ← contador + 1
Mientras ( contador <= 10 )
```

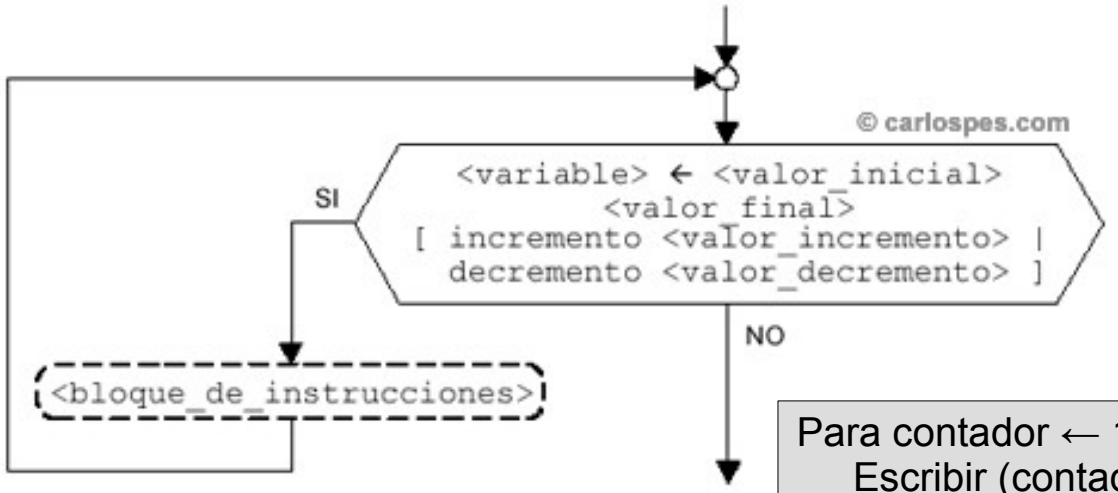
6. Pseudocódigo

Bucle Para (for):

Para <variable> ← <valor_inicial>, <condición>, incremento <valor_incremento> **hacer**

<bloque_de_instrucciones>

FinPara




```
Para contador ← 1,  contador<=10, incremento 1 hacer
    Escribir (contador)
FinPara
```

6. Pseudocódigo

ESTRUCTURA DE CONTROL	DESCRIPCIÓN
Inicio y Fin	Comienzo y final del algoritmo.
SI <condicion> ENTONCES <acciones1> SI-NO <acciones 2> FIN-SI	Ejecución condiciónal de la acción.
MIENTRAS <condición> HACER <acciones> FIN-MIENTRAS	Repite la acción mientras la condición sea verdadera.
HACER <acciones> MIENTRAS <condición>	Ejecuta las acciones mientras la condición sea verdadera.
PARA <variable ←inicio>, <condicion>, <incremento> HACER <acciones> FIN-PARA	Se repite la acción dependiendo de una variable de control que empieza tomando un valor y cada vuelta se incrementa.

6. Pseudocódigo

Ejercicio: Escribe el pseudocódigo de los diagramas con estructuras cíclicas realizados previamente

Ejercicios 20 a 25

Unidad 1. Introducción a la programación

FIN