

# Programación

## UD 1: Introducción a la Programación

# Introducción a la Programación

---

- 1.- Algoritmos y Programas
- 2.- Lenguajes de Programación. Tipos
- 3.- Entornos de desarrollo
- 4.- Representación de los algoritmos
  - 4.1.- Pseudocódigo
  - 4.2.- Diagramas de Flujo

---

# 1.- Algoritmos y programas

---

# 1.- Algoritmos y programas

---

## Definiciones:

**Algoritmo:** Secuencia finita de reglas o instrucciones que especifican un conjunto de operaciones, que al ser ejecutadas por un agente ejecutor (máquina real o abstracta), resuelve cualquier problema de un tipo determinado en un tiempo finito.

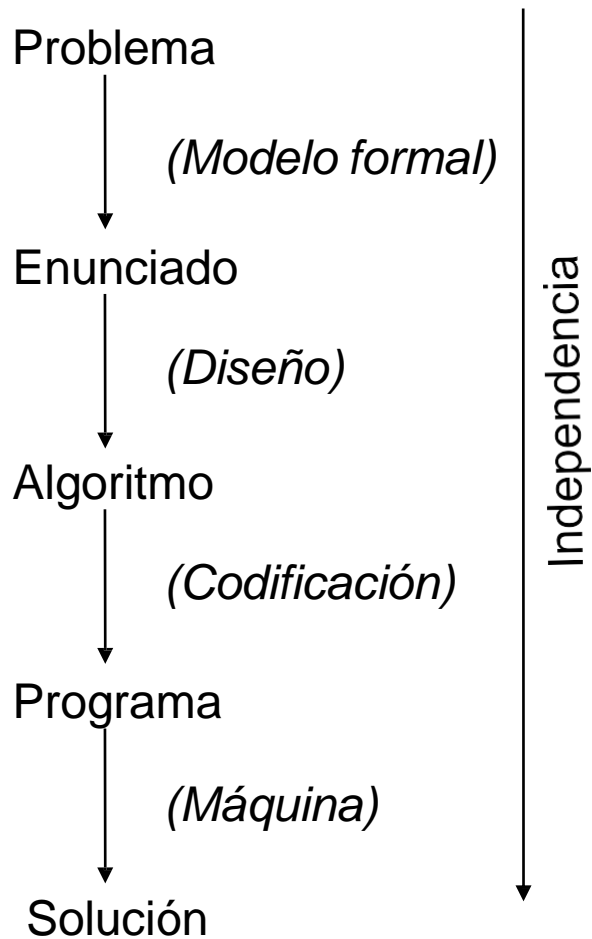
**Programa informático:** Conjunto de instrucciones que implementan un algoritmo. Una vez ejecutadas, las instrucciones realizarán una o varias tareas en un ordenador.

**Programación:** Es el proceso por el cual una persona desarrolla un programa valiéndose de una herramienta que le permita escribir el código (el cual puede estar en uno o varios lenguajes, tales como C++, Java y Python entre otros) y de otra que sea capaz de “traducirlo” a lo que se conoce como lenguaje de máquina, el cual puede ser entendido por un microprocesador.

## PROGRAMACIÓN = ALGORITMOS + ESTRUCTURAS DE DATOS

# 1.- Algoritmos y programas

---



- Dado un problema intentaremos encontrar un modelo formal que nos permita representarlo como un enunciado.
- Mediante una técnica de diseño realizaremos un algoritmo que resuelva el problema
- Mediante un lenguaje de programación realizaremos el programa.
- Una vez ejecutado el programa por un agente ejecutor obtendremos un resultado que nos dará la solución del problema.

# 1.- Algoritmos y programas

---

Algoritmos:

**Datos y variables:**

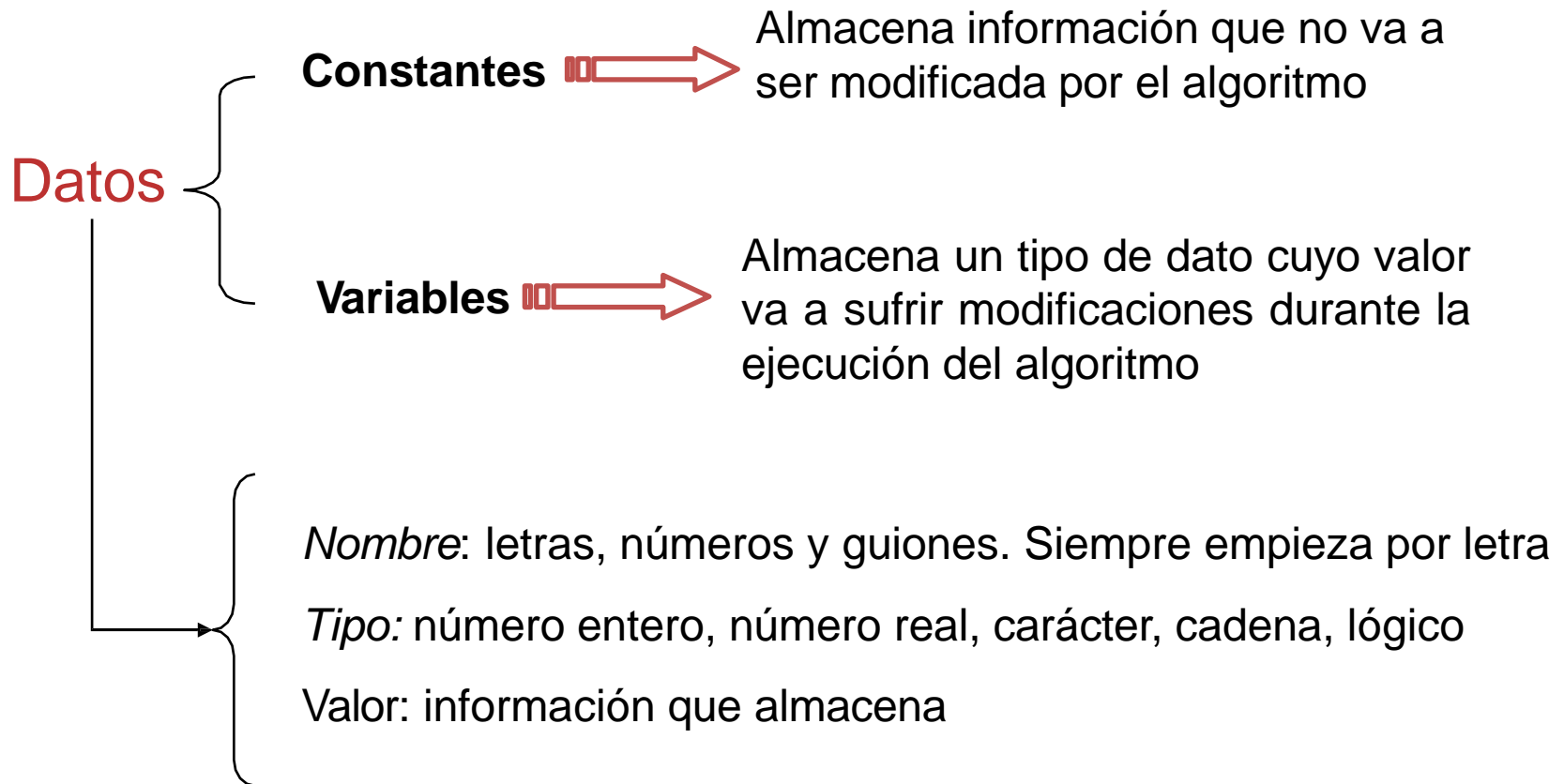
ALGORITMO = Técnica para resolver problemas a través de una serie de pasos intermedios hasta llegar a resultado.

Pero siempre vamos a manejar distintos tipos de datos en un algoritmo ... tiempo, euros, cantidad de productos ...

Y necesitaremos almacenar los resultados de los cálculos intermedios de cada algoritmo.

# 1.- Algoritmos y programas

---



# 1.- Algoritmos y programas

---

## Ejemplo de algoritmo:

## Reparación de un ordenador

- Me traen un ordenador estropeado
- Empiezo a contar el tiempo.
- Compruebo el pc y detecto las averias.
- Cambio las piezas estropeadas y ... funciona.
- Anoto las piezas cambiadas.
- Le pregunto al dueño si paga con tarjeta o en efectivo. Con tarjeta se recarga un 2%.
- Cobro al dueño del pc por el tiempo trabajado por horas y las piezas cambiadas.
- Anoto sus datos de cliente en base datos. Fin



# 1.- Algoritmos y programas

---

## Elementos usados

He almacenado el tiempo de reparación. Variable numérica entera.

- El precio por hora es constante.
- El precio de cada pieza no es exacto en euros, tiene céntimos. Variable numérica real.
- Pago con tarjeta. Verdadero o falso. Variable lógica.
- Almaceno la suma total en variable. ¿Tipo?
- Almaceno los datos de cliente en variable tipo cadena de caracteres.

# 1.- Algoritmos y programas

---

## Tipos de datos

**Enteros:** Son los números enteros. Como horas exactas o euros sin céntimos

**Reales:** Son los números con decimales. Como precios de productos.

**Lógicos:** Tienen dos valores Verdadero o Falso.

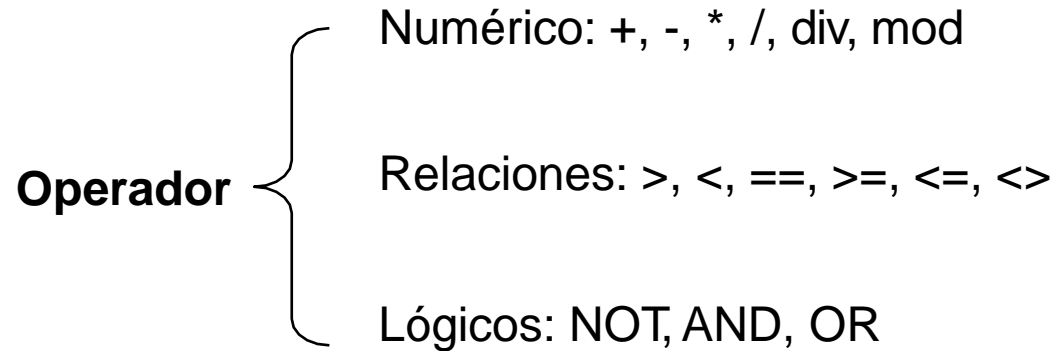
**Carácter:** Son las letras del alfabeto.

**Cadena de caracteres:** Son un conjunto de caracteres como el nombre y apellidos de una persona.

# 1.- Algoritmos y programas

---

## Instrucciones



**Operando:** es una variable, una constante, etc.. un elemento que tiene un valor.

**Expresión:** constante o variable, es un conjunto de operadores y operandos.

Ejemplo:  $x = 12 + 3 * 4$

# 1.- Algoritmos y programas

---

## Características de los algoritmos

Un algoritmo debe ser:

- ✓ **Preciso**, debe indicar el orden de realización de cada paso.
- ✓ **Definido**, si se sigue un algoritmo dos veces se debe obtener el mismo resultado cada vez.
- ✓ **Finito**, debe terminar en un número finito de pasos

---

## 2.- Lenguajes de Programación. Tipos

---

## 2.- Lenguajes de Programación

---

**Definición:** “Un lenguaje de programación es un lenguaje formal que especifica una serie de instrucciones para que una computadora produzca diversas clases de datos. Los lenguajes de programación pueden usarse para crear programas que pongan en práctica algoritmos específicos que controlen el comportamiento físico y lógico de una computadora.” ([Wikipedia](#))

### Ejemplo. Hola mundo en c++

```
#include <iostream>

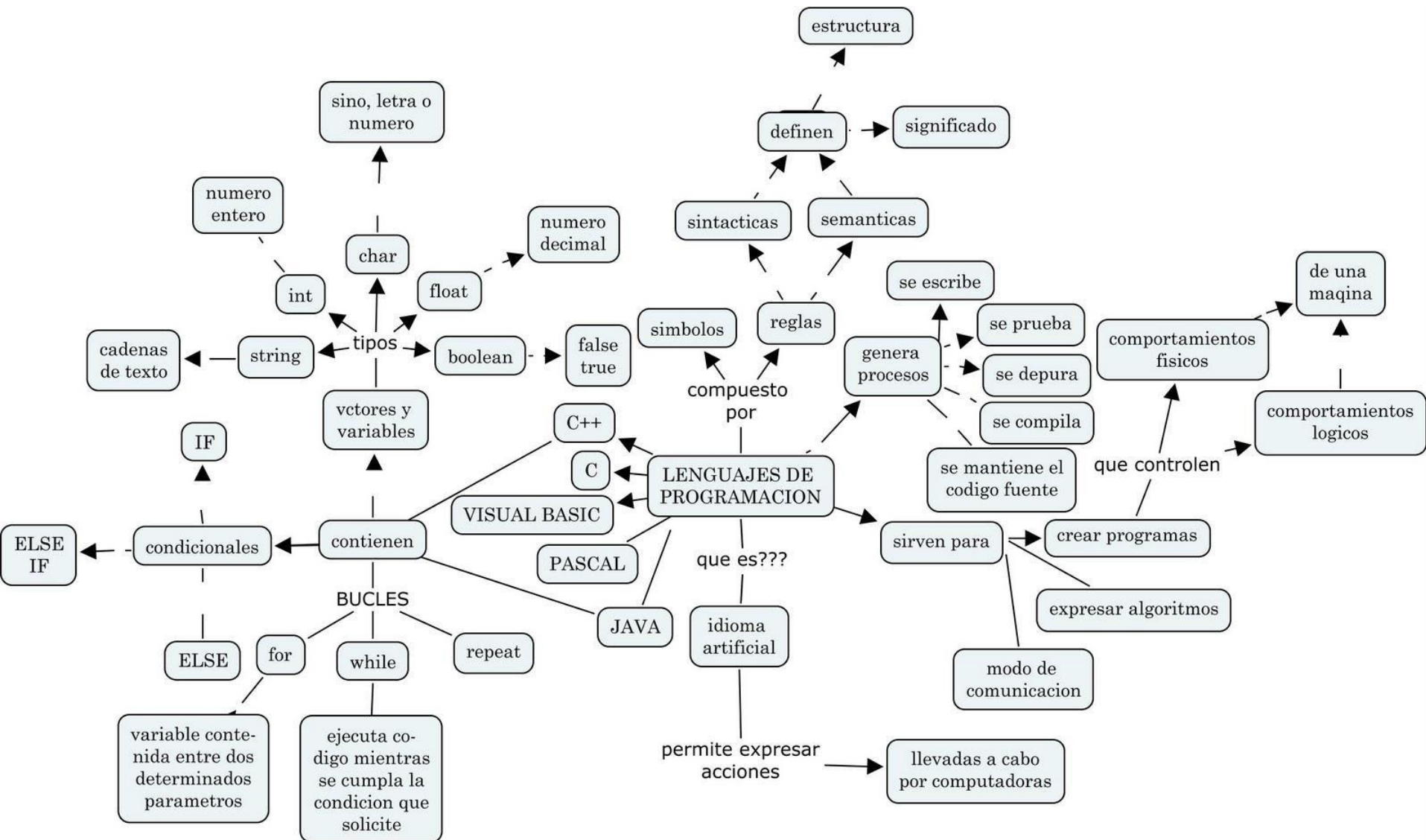
using namespace std;

int main() {
    cout << "Hola Mundo" << endl;
    return 0;
}
```

### Ejemplo. Hola mundo en Java

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hola mundo");
    }
}
```

# 2.- Lenguajes de Programación



## 2.- Lenguajes de Programación

---





# 2.- Lenguajes de Programación

---

## Ejercicio:

- ✓ Otras definiciones de Lenguaje de Programación
- ✓ Lenguajes de 1era, 2nda, 3era y 4ta generación (¿5ta?)
- ✓ Clasificación de los lenguajes de programación según:
  - La proximidad del lenguaje a la máquina (Alto nivel Vs Bajo nivel)
  - En función del paradigma de programación (Imperativos Vs Declarativos)
  - La traducción al código máquina (Interpretados Vs Compilados)
  - Según su funcionalidad

---

## 3.- Entornos de Desarrollo

---

# 3.- Entornos de Desarrollo

---

**Definición:** “Un entorno de desarrollo integrado, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.” ([Wikipedia](#))

Los IDEs pueden estar dedicados a un lenguaje de programación específico o servir para distintos lenguajes aunque hoy en día suelen ser multilenguaje.

Existen IDEs multiplataforma, es decir, se pueden ejecutar sobre distintos SO y arquitecturas. Normalmente desarrollados en JAVA.

# 3.- Entornos de Desarrollo

---

Un IDE, consta al menos de los siguientes elementos:

- ✓ Un editor de texto o código. Actualmente con sintaxis coloreada, predicción de texto y navegación por el código
- ✓ Un compilador y/o intérprete.
- ✓ Un depurador de errores. (Breakpoints, ejecución paso a paso, visualización de variables, pila, etc...)
- ✓ Opcionalmente. Funciones para la construcción de interfaces gráficas (GUI)
- ✓ Opcionalmente. Algún sistema de control de versiones.
- ✓ Opcionalmente. Herramientas de generación de pruebas y documentación de código.
- ✓ Etc, etc...

# 3.- Entornos de Desarrollo

Algunos de los IDE más utilizados son:

	Windows	Linux	Java
Código abierto	<ul style="list-style-type: none"><li>-<b>DevC++</b>. IDE completo para utilizar MinGW (Minimalist GNU for Windows)</li><li>-<b>Visual-MinGW</b>. Diseñado para utilizar MinGW</li></ul>	<ul style="list-style-type: none"><li>-<b>Emacs, Vim</b>. Editores de textos tradicionales de Unix, muy engorrosos.</li><li>-<b>Anjuta</b>. C/C++, incorpora las herramientas GNU gcc, make, gdb, entre otros</li><li>-<b>Kdevelop</b>. C/C++, Fortran, Pascal, Perl... Permite desarrollo de interfaces gráficas.</li></ul>	<ul style="list-style-type: none"><li>-<b>Eclipse</b>. IDE independiente de la plataforma. Extensible mediante módulos. Da soporte por defecto para Java ampliable a otros lenguajes. Recomendado por Google para el desarrollo para Android</li><li>- <b>Netbeans</b>. Idem eclipse</li></ul>
Propietarios	<ul style="list-style-type: none"><li>-<b>Visual Studio</b>. El IDE más popular de Microsoft. Admite C#, C++ y Visual Basic</li><li>-<b>C++ Builder. Delphi</b>. RAD multiplataforma de Embarcadero basados en ObjectPascal y C++</li></ul>	<ul style="list-style-type: none"><li>-<b>Code Forge</b>. Admite mas de 30 lenguajes.</li><li>- <b>Maguma Workbench</b></li></ul>	<ul style="list-style-type: none"><li>-<b>Jbuilder</b>. El más popular de los IDE comerciales para Java. Producto de Embarcadero compañía que cuenta también con Delphi y C++ Builder.</li><li>- <b>AIDE</b>. Android para Android</li></ul>

# 3.- Entornos de Desarrollo

---

## IDE para 1ºDAW

La última versión de Eclipse



Get **Eclipse IDE 2020-06**

Install your favorite desktop IDE packages.

[Download 64 bit](#)

[Download Packages](#) | [Need Help?](#)

---

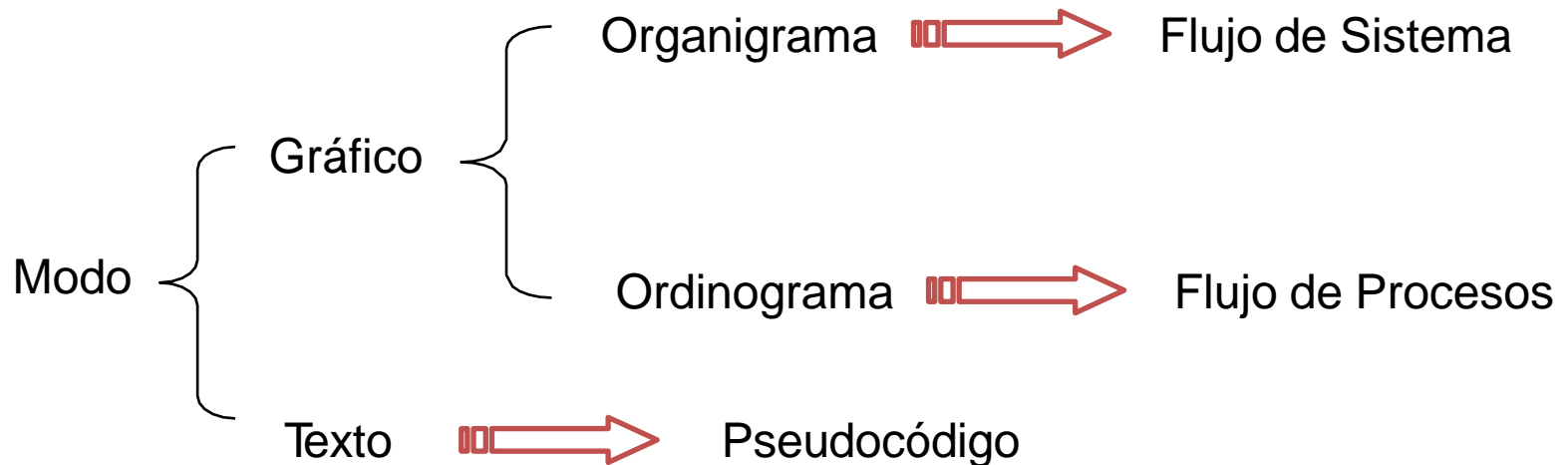
## 4.- Representación de algoritmos

---

# 4.- Representación de algoritmos

---

Los algoritmos se pueden representar de las siguientes formas:



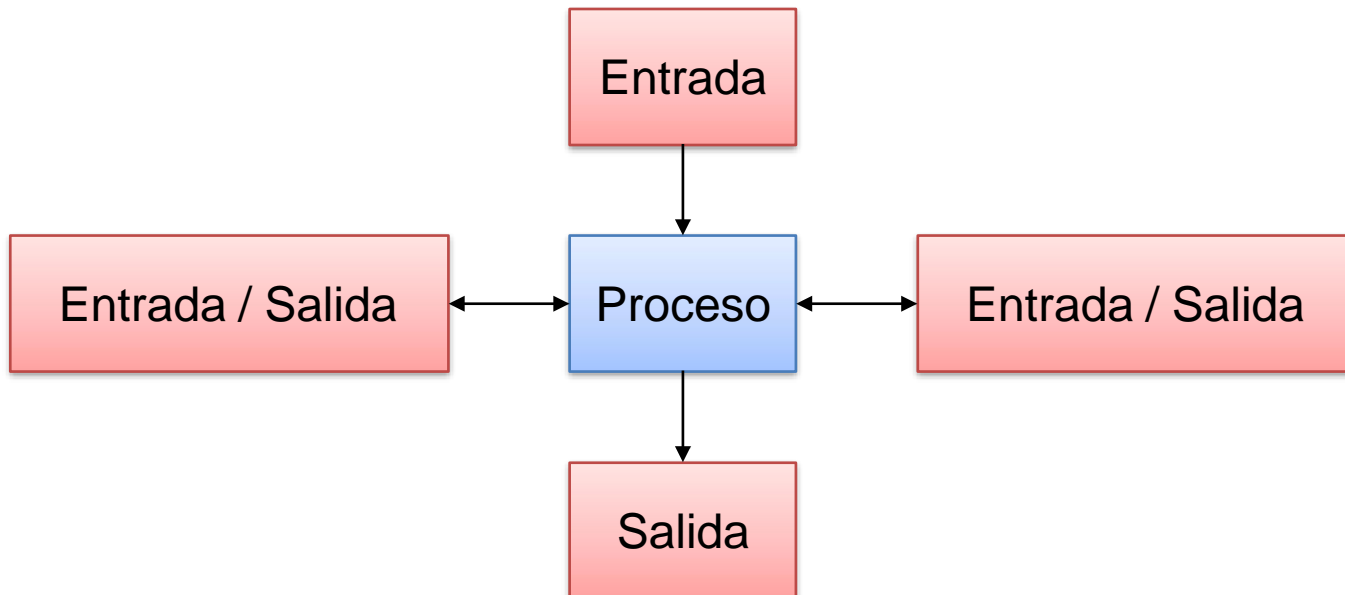


# 4.- Representación de algoritmos

---

## Organigrama:

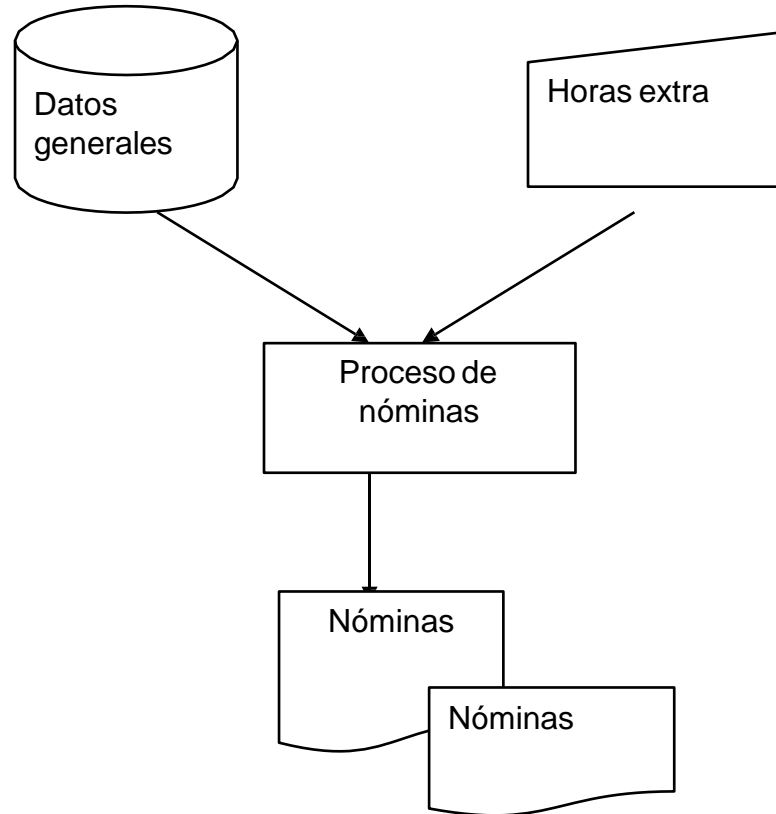
Representa el flujo de datos y soportes que intervienen en un sistema



# 4.- Representación de algoritmos

---

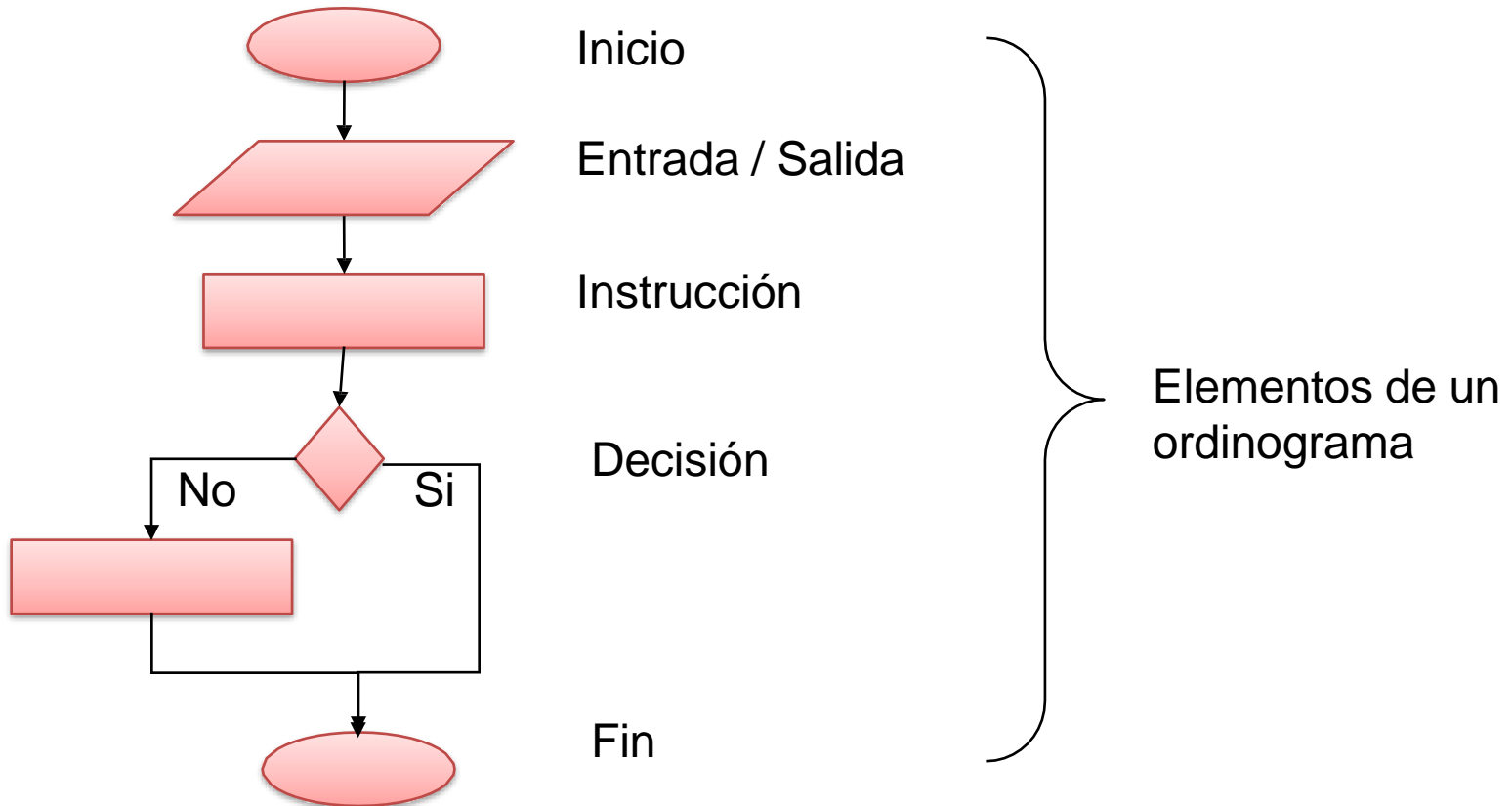
## Organigrama (ejemplo):



# 4.- Representación de algoritmos

## Ordinograma (Diagrama de flujo):

Representa el flujo de datos de un proceso



# 4.- Representación de algoritmos

---

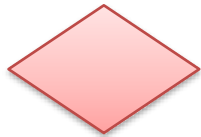
## Ordinograma. Elementos



**Terminal.** Representa el inicio y fin de un programa



**Proceso.** Acciones del programa



**Decisión.** Indica operaciones lógicas o de comparación, así como expresiones



**Entrada/Salida.** Nos permite introducir datos en un periférico, así como mostrarlos

# 4.- Representación de algoritmos

---

## Ordinograma. Elementos



**Salida.** Usado para mostrar datos por impresora.



**Conector.** Se coloca al principio y fin de un pedazo de programa, enlaza dos partes de un programa.



**Linea de flujo.** Indica la dirección de ejecución del algoritmo.



**Subprograma.** Usado para realizar una llamada a un subprograma o subrutina.

# 4.- Representación de algoritmos

---

## Diagrama de flujo

### Características

- Inicio y Fin
- Arriba abajo, izquierda derecha
- Un símbolo por acción
- A todos los procesos debe llegar una flecha y salir otra.
- Simetría en los elementos

# 4.- Representación de algoritmos

---

## Pseudocódigo

- Lenguaje natural
- Permite escribir las instrucciones que conducen a la resolución de problema utilizando estructuras básicas de programación
- Reglas
  - Cada instrucción en una línea
  - Conjunto de palabras reservadas en minúsculas: si, entonces, fsi, mientras, fmientras, etc ...
  - Referencia a módulos entre <NOMBRE-MODULO>
  - Código indentado

# 4.- Representación de algoritmos

## Pseudocódigo. Estructura

Programa: NOMBRE correspondiente al programa  
Entorno:  
    Declaración de las estructuras de datos en general.  
Algoritmo:  
    Secuencia de instrucciones que forman el programa.  
Fin del programa.

Ejemplo



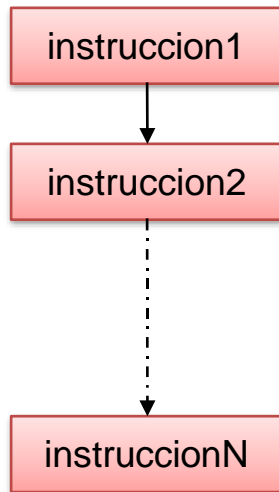
**Programa:** ARRANCA\_COCHE  
**Entorno:**  
**Algoritmo:**  
    Pisar embrague con pie izquierdo  
    Poner punto muerto  
    Dar a llave de contacto  
    Pisar embrague  
    Meter la marcha primera  
    Quitar el freno de mano  
    Levantar el pie del embrague  
**Fin del programa**



# 4.- Representación de algoritmos

---

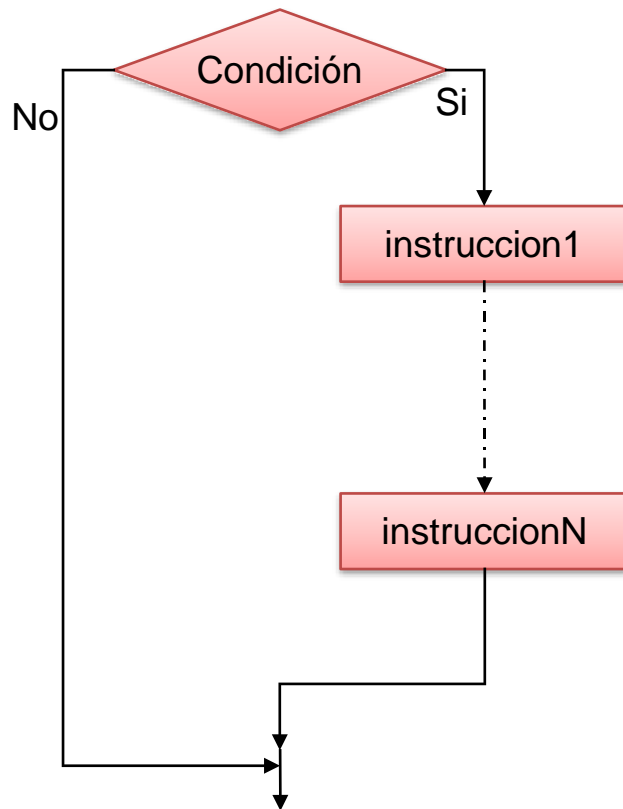
## Estructura secuencial



```
instruccion1;  
instruccion2;  
instruccion3;  
....  
instruccionN;
```

# 4.- Representación de algoritmos

## Estructura Condicional Simple



**si** condición **entonces**

instruccion1;

instruccion2;

instruccion3;

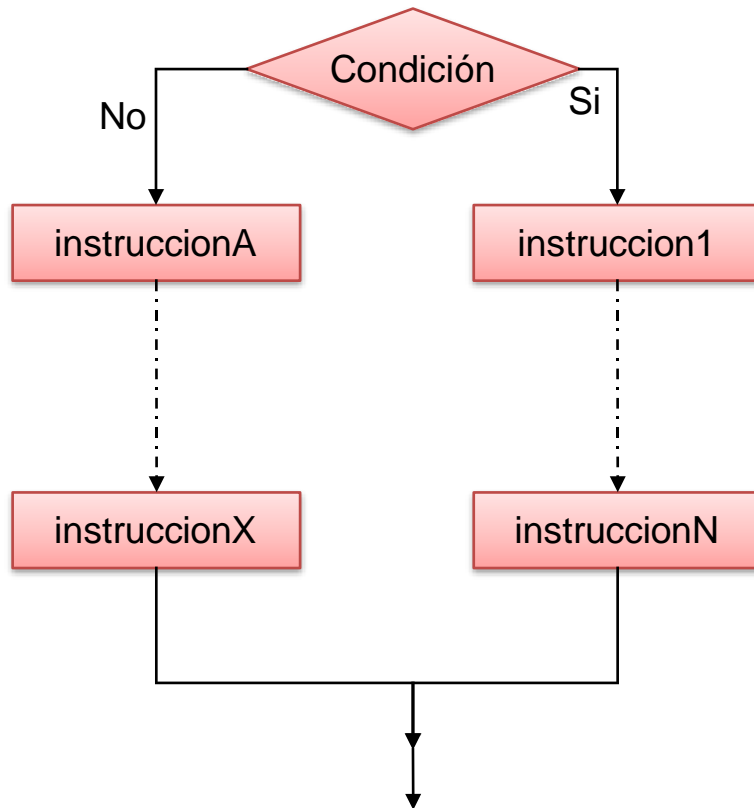
....

instruccionN;

**fsi**

# 4.- Representación de algoritmos

## Estructura Condicional Compuesta



**si** condición **entonces**

instruccion1;

....

instruccionN;

**sino**

instruccionA;

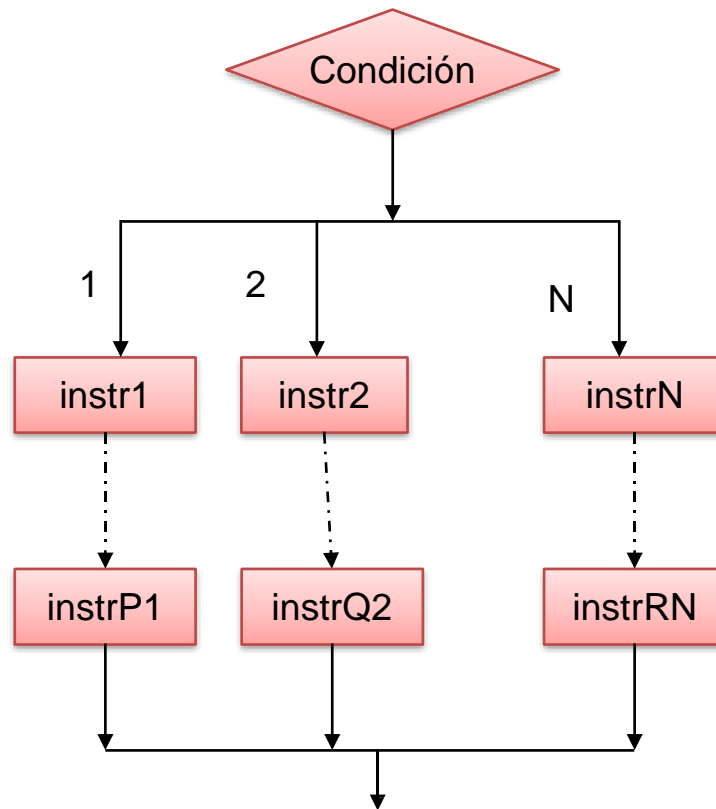
....

instruccionX;

**fsi**

# 4.- Representación de algoritmos

## Estructura Condicional Múltiple



**caso** condición **de**

**1:** instr1;

....

instrP1;

**2:** instr2;

....

instrQ2;

....

**N:** instrN;

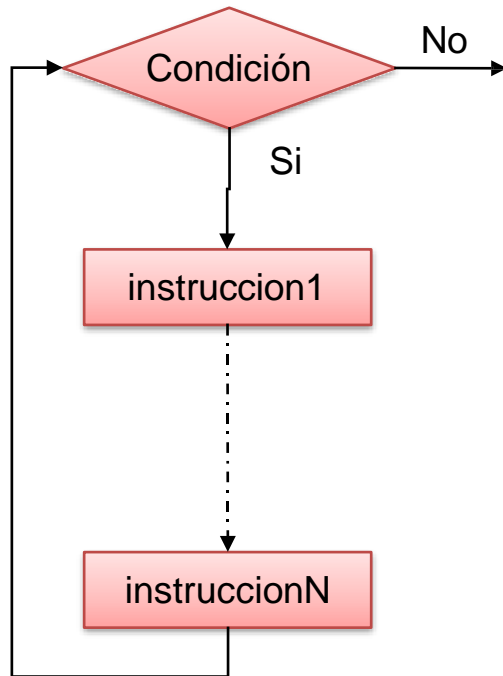
....

instrRN;

**fcaso**

# 4.- Representación de algoritmos

## Estructura repetitiva Mientras



**mientras** condición **hacer**

instruccion1;

instruccion2;

instruccion3;

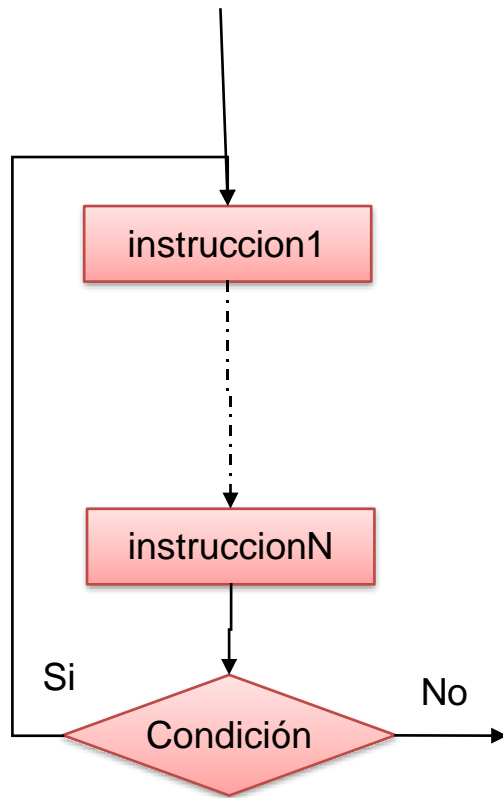
....

instruccionN;

**fmientras**

# 4.- Representación de algoritmos

## Estructura repetitiva Repite... mientras



**repetir**

instruccion1;

instruccion2;

instruccion3;

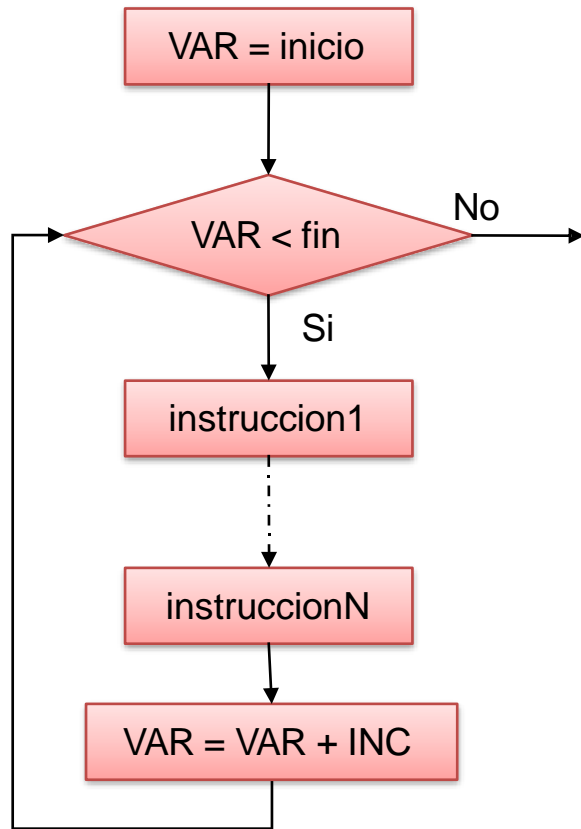
....

instruccionN;

**mientras** condición;

# 4.- Representación de algoritmos

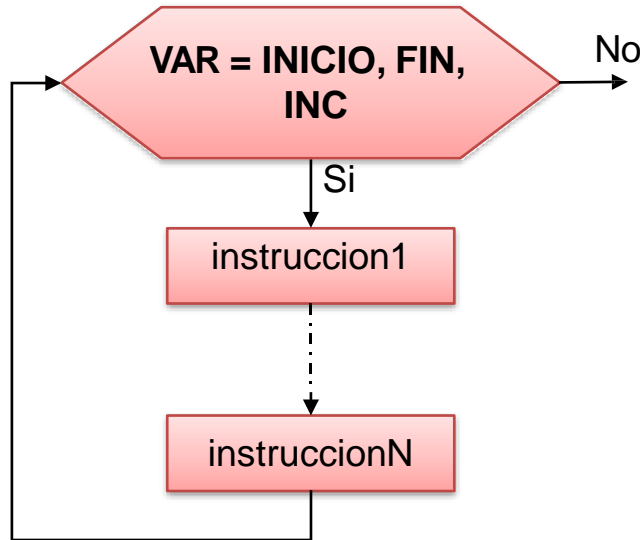
## Estructura repetitiva Para (1)



**para** VAR **desde** inicio **hasta** fin  
**[ con incremento inc ] hacer**  
    instruccion1;  
    instruccion2;  
    instruccion3;  
    ...  
    instruccionN;  
**fpara**

# 4.- Representación de algoritmos

## Estructura repetitiva Para (2)



**para** VAR **desde** inicio **hasta** fin  
**[ con incremento inc ] hacer**  
instruccion1;  
instruccion2;  
instruccion3;  
....  
instruccionN;  
**fpara**



# 4.- Representación de algoritmos

---

## Contadores, acumuladores e interruptores

Vamos a ver una serie de **variables especiales** usadas en estructuras de tipo repetitivo bien para evaluar una condición o bien de ayuda para realizar acciones.

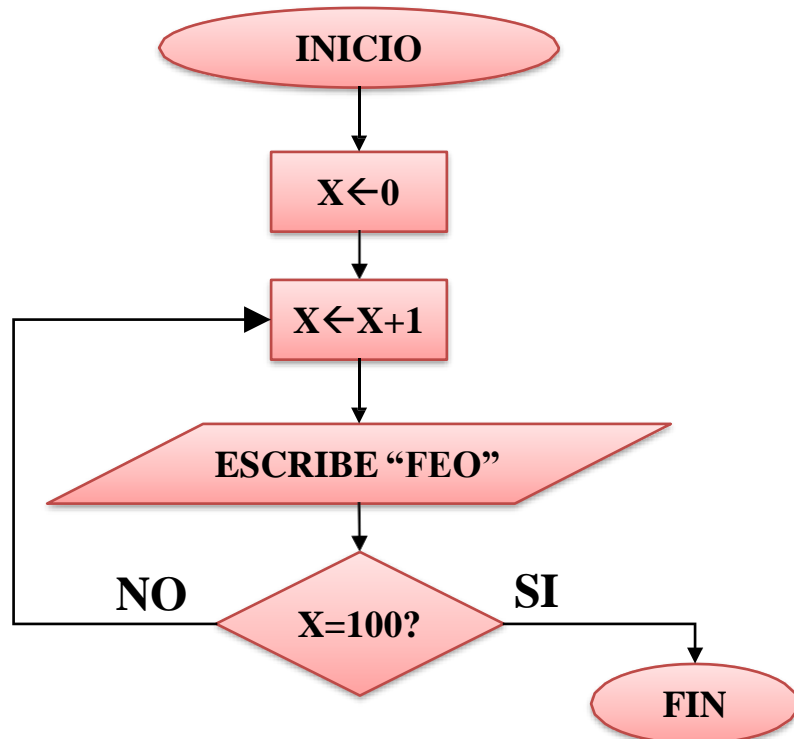
Dichas variables son las siguientes:

- Contadores
- Acumuladores
- Interruptores

# 4.- Representación de algoritmos

## Contadores

Un contador es una variable cuyo valor se incrementa o decrementa en cada iteración de un bucle, sirve para averiguar cual es el estado de un bucle.



### PSEUDOCÓDIGO:

**INICIO**

**X ← 0**

**REPETIR**

**X ← X + 1**

**ESCRIBE("FEO")**

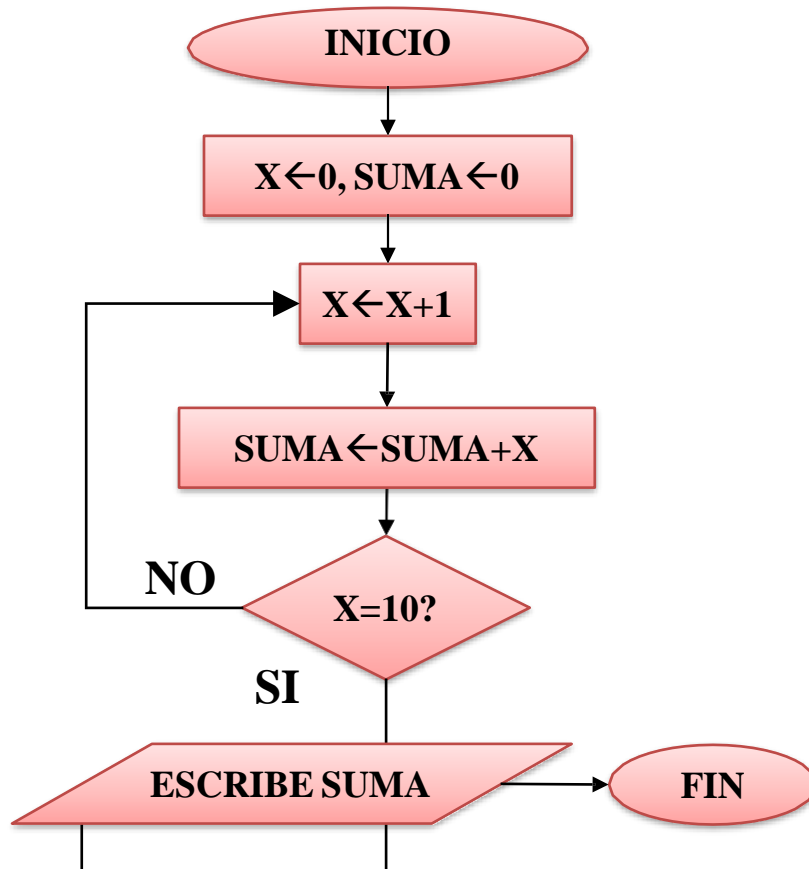
**HASTA (X=100)**

**FIN**

# 4.- Representación de algoritmos

## Acumuladores

Parecido a un contador, pero su uso es almacenar cantidades procedentes de acciones en cada iteración.



### PSEUDOCÓDIGO:

**INICIO**

**$X \leftarrow 0$**

**$\text{SUMA} \leftarrow 0$**

**REPETIR**

**$X \leftarrow X + 1$**

**$\text{SUMA} \leftarrow \text{SUMA} + X$**

**HASTA (X=10)**

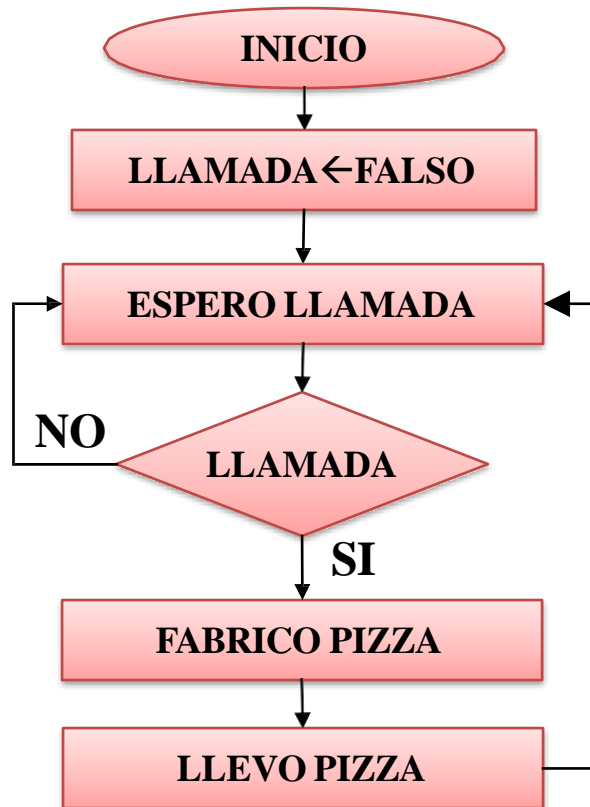
**ESCRIBE(SUMA)**

**FIN**

# 4.- Representación de algoritmos

## Interruptores

Un interruptor es una variable cuyo valor puede tomar los valores verdadero o falso. Se usa para control de bucles y estructuras selectivas.



### PSEUDOCÓDIGO :

```
INICIO
LLAMADA ← FALSO
REPETIR
    ESPERO LLAMADA
    SI LLAMADA ENTONCES
        FABRICO PIZZA
        LLEVO PIZZA
    FIN_SI
HASTA EL JUICIO FINAL
```