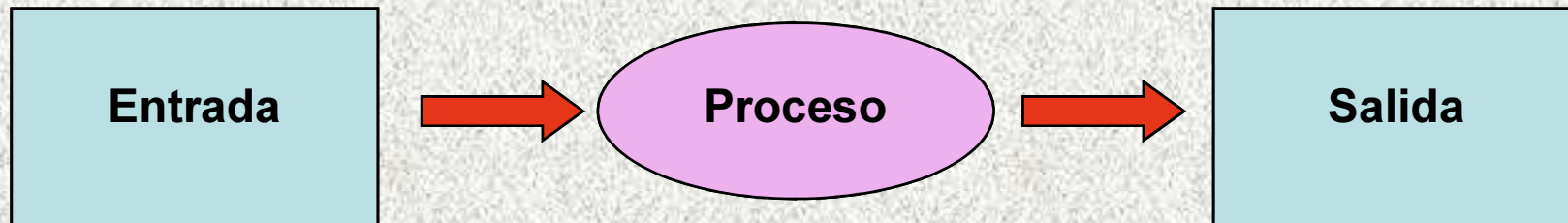


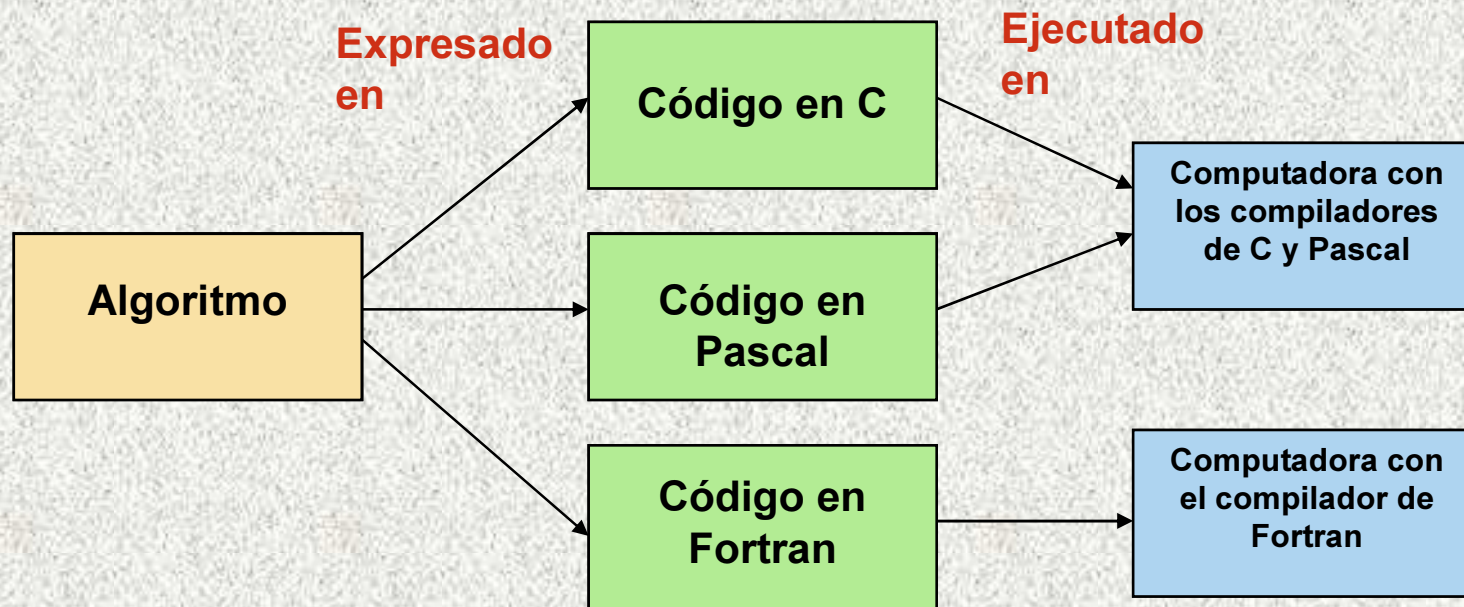
# ALGORITMO

- Deriva de la traducción al latín de la palabra árabe **alkhowarizmi**, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.
- Es un procedimiento para resolver un problema específico. Describe un conjunto finito y ordenado de pasos, reglas o instrucciones bien definidas para producir el resultado esperado a partir de las entradas dadas.
- Permite automatizar una operación.
- La definición de un algoritmo debe contener una **entrada** de datos, una descripción del **proceso** que se aplicará a los mismos y una **salida** con los resultados.



# PROGRAMA

- Es la implementación o expresión de un **algoritmo** en un determinado **lenguaje de programación** siguiendo las reglas establecidas por el lenguaje elegido.
- Está formado por un conjunto de órdenes (**instrucciones** o **sentencias**) que el procesador debe ejecutar.
- Los algoritmos son independientes tanto del lenguaje de programación en el que se expresan como del hardware en que se ejecutan los programas

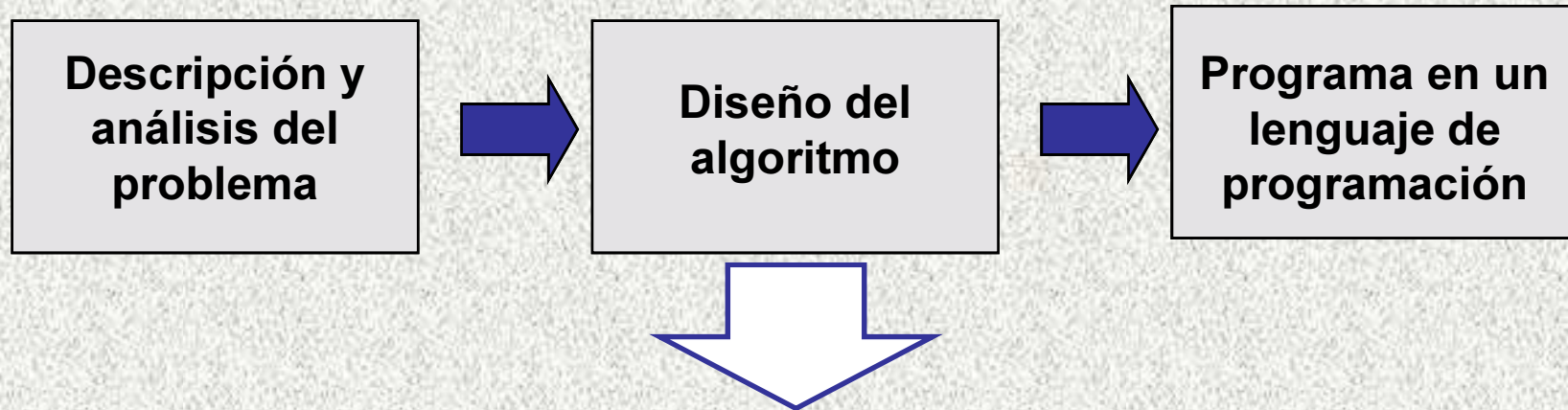


# PASOS PARA LA SOLUCIÓN DE UN PROBLEMA

1. **Definición del problema:** enunciado en forma clara y precisa.
2. **Análisis del problema:**
  - ❖ Datos de entrada.
  - ❖ Información que se desea producir (salida).
  - ❖ Métodos y fórmulas que se necesitan para procesar los datos.
3. **Diseño del algoritmo.**
4. **Codificación:** escribir el programa en un lenguaje de alto nivel (código fuente).
5. **Prueba y depuración:** identificar y eliminar errores (los más comunes son los sintácticos y lógicos). Verificar con datos conocidos (entrada y salida).
6. **Documentación:** guía o comunicación escrita (enunciados, funciones, dibujos, diagramas). Ayuda a comprender el programa para facilitar futuras modificaciones.
7. **Mantenimiento:** actualizaciones o nuevas versiones.



# ALGORITMO: CARACTERÍSTICAS



- Debe tener un **inicio** y un **fin**.
- Debe ser **definido**, formado por etapas bien definidas y concretas. Debe producir los mismos resultados (especificaciones) para las mismas condiciones de entrada.
- Debe ser **preciso**, es decir, cada instrucción debe indicar de forma inequívoca que operación se tiene que realizar (no debe permitir dobles interpretaciones), el orden en que se realizan y cuales son los datos de entrada y de salida.
- Debe ser **finito** en tamaño (número limitado de pasos) y tiempo de ejecución.
- Debe ser general, soportar la mayoría de las variantes que se puedan presentar en el problema.




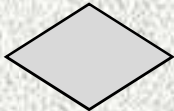



# ALGORITMO: HERRAMIENTAS de DISEÑO

Se busca realizar una especificación y representación gráfica del algoritmo utilizando estructuras de control definidas.

- **Pseudocódigo**: expresión en lenguaje natural, el programador no se preocupa por las reglas de un lenguaje específico. Es un paso intermedio entre la solución del problema y su codificación en un lenguaje de programación elegido. No hay estándares. Permite definir las estructuras de datos y las operaciones que se realizarán. Fácil de modificar.
- **Diagramas de flujo u organigramas (*flowchart*)**: representación gráfica (utiliza símbolos conectados por flechas para operaciones específicas) que permite detallar el flujo (orden en que se ejecutan las de instrucciones). No se declaran las variables. Para modificar hay que rehacer el dibujo.
- **Diagramas de Nassi-Schneiderman (N-S)**: conocido también como diagrama de **Chapin**. Es una técnica de representación gráfica que combina las anteriores. Un algoritmo es representado con un rectángulo dividido en franjas o bandas horizontales, cada una representa una acción a realizar.

# ALGORITMO: DIAGRAMAS → SIMBOLOGIA

## ➤ Flujo

Símbolo	Significado
	Indica el inicio y fin del diagrama.
	Proceso o acción a realizar (asignación de un valor en memoria y/o ejecución de una operación).
	Indica una entrada o salida de datos.
	Representa una comparación de valores (decisión lógica) con dos posibles caminos a seguir.
	Estructura repetitiva, se repiten las instrucciones un número determinado de veces ( <b>iteraciones</b> ).
	Se usa como conector entre dos partes del diagrama.
	Indican el sentido o secuencia en que se ejecutan las operaciones.



# ALGORITMO: DIAGRAMAS → SIMBOLOGIA

## ➤ Nassi-Schneiderman (N-S)

Inicio
<acción 1>
<acción 2>
.....
<acción N>
Fin

Mientras (condición)	
	<acción 1>
	.....
	<acción N>

Hasta (condición)	
	<acción 1>
	.....
	<acción N>

condición	
si	no
<acción 1a>	<acción 1b>
.....	.....
<acción N>	<acción M>

# ALGORITMO: Ejemplo 1 (pseudocódigo)

- Algoritmo para escribir una carta:

1. Tomar una hoja de papel.
2. Tomar un lápiz.
3. Sentarse en una silla.
4. Apoyar la hoja de papel en una mesa.
5. Escribir la fecha.
6. Escribir a quien va dirigida.
7. Escribir el contenido de la carta.
8. Firmar la carta.
9. Dejar el lápiz en el lugar en el que lo encontramos.
10. Tomar un sobre.
11. Doblar la carta de tal manera de que quepa en el sobre.
12. Guardar la carta en el sobre.
13. Completar el sobre.



# ALGORITMO: Ejemplo 2 (pseudocódigo + N-S)

## ▪ Definición del problema

Calcular el área y el perímetro de un rectángulo.

## ▪ Análisis del problema:

Datos entrada: longitud de la base (b)      longitud de la altura (a)

Datos salida:      área (ar)      perímetro (pe)

Procesos:       $ar = b * a$        $pe = 2 * (b + a)$

## ▪ Algoritmo

1. Inicio
2. Leer base y altura (b, a)
3.  $ar \leftarrow b * a$
4.  $pe \leftarrow 2 * (b + a)$
5. Escribir "Área del rectángulo: ", ar
6. Escribir "Perímetro del rectángulo: ", pe
7. Fin

Inicio
real: b, a, ar, pe
Leer b, a
$ar = b * a$
$pe = 2 * (b + a)$
Escribir "área: ", ar
Escribir "perímetro: ", pe
Fin

# ALGORITMO: Ejemplo 3

- **Definición del problema**

Calcular los valores medios de la edad y el peso de un grupo de estudiantes.

- **Análisis del problema:**

Datos entrada:      Número de estudiantes (NE).

Datos auxiliares:    Contador (N)  
SumaEdad (SE)  
SumaPeso (SP).

Datos salida:        Edad promedio (EP)  
Peso promedio (PP).

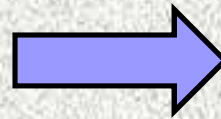
Procesos:             $EP = SE/NE$   
 $PP = SP/NE$ .

# ALGORITMO: Ejemplo 3 (continuación)

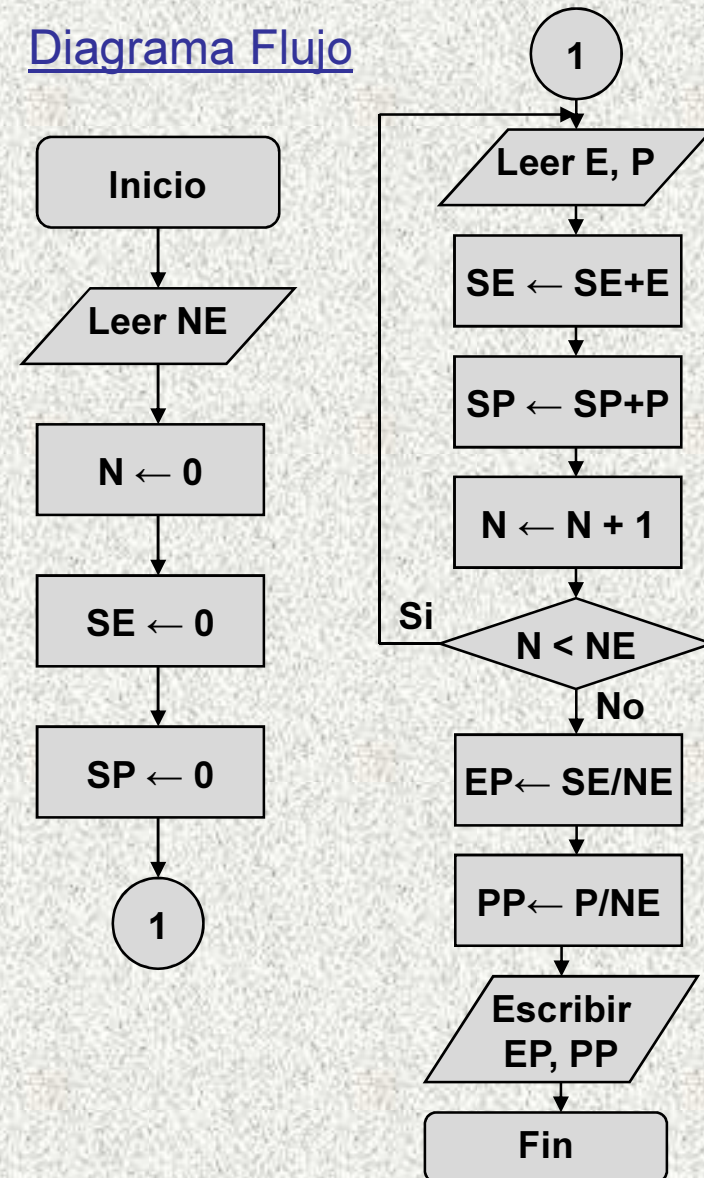
## ■ Algoritmo

### Pseudocódigo

1. Inicio
2. Leer cantidad estudiantes (NE)
3. Inicializar  $N \leftarrow 0$ ,  $SE \leftarrow 0$ ,  $SP \leftarrow 0$
4. MIENTRAS  $N < NE$ 
  - 4.1 Leer la edad y el peso del estudiante (E,P)
  - 4.2  $SE = SE + E$
  - 4.3  $SP = SP + P$
  - 4.4 Incrementar el contador:  $N \leftarrow N + 1$
- FIN-MIENTRAS
5. Calcular la media de las edades  $EP = SE/NE$
6. Calcular la media de los pesos  $PP = SP/NE$
7. Escribir "La edad promedio es: ", EP
8. Escribir "El peso promedio es: ", PP
9. Fin



### Diagrama Flujo





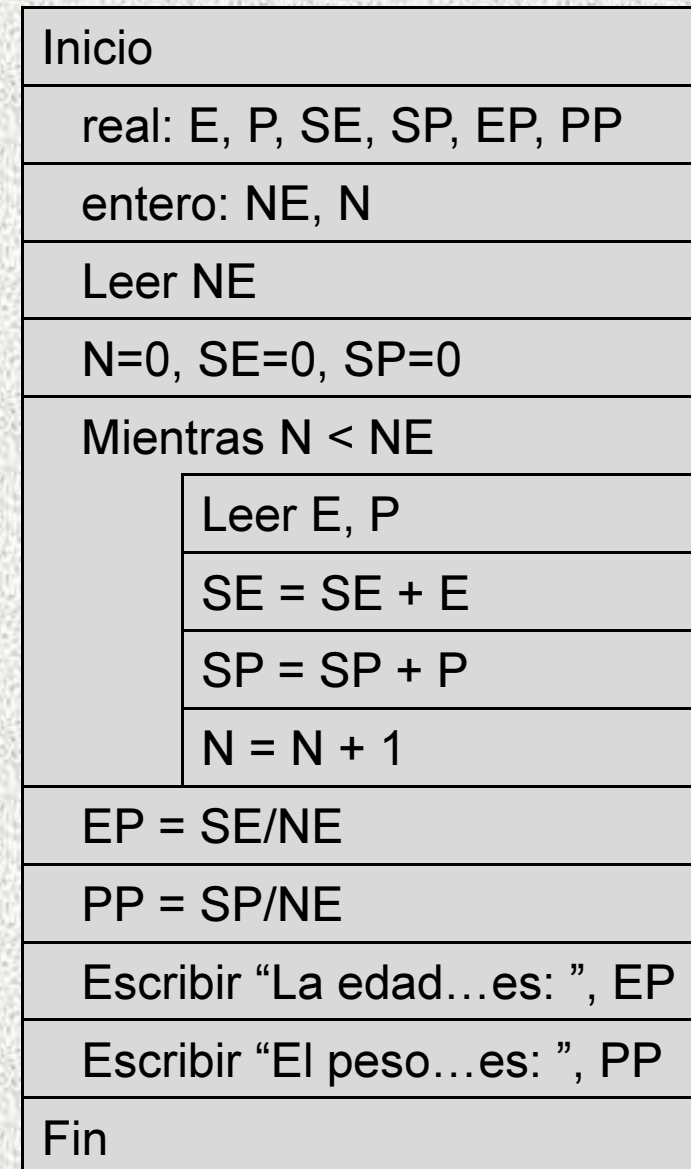
## ALGORITMO: Ejemplo 3 (continuación)

### Pseudocódigo

1. Inicio
2. Leer cantidad estudiantes (NE)
3. Inicializar  $N \leftarrow 0$ ,  $SE \leftarrow 0$ ,  $SP \leftarrow 0$
4. MIENTRAS  $N < NE$ 
  - 4.1 Leer la edad y el peso del estudiante (E,P)
  - 4.2  $SE = SE + E$
  - 4.3  $SP = SP + P$
  - 4.4 Incrementar el contador:  $N \leftarrow N + 1$
- FIN-MIENTRAS
5. Calcular la media de las edades  $EP = SE/NE$
6. Calcular la media de los pesos  $PP = SP/NE$
7. Escribir "La edad promedio es: ", EP
8. Escribir "El peso promedio es: ", PP
9. Fin



### Diagrama N-S



# PROGRAMACION ESTRUCTURADA

Desarrollada en los años sesenta por Edsger W. Dijkstra (1930-2002), se basa en el *teorema de la estructura* (Böhm y Jacopini, 1966): *todo programa puede escribirse utilizando únicamente tres estructuras básicas para el control de la programación.*

La programación estructurada es un conjunto de técnicas que incorporan:

- Estructuras de control básico: **secuencia**, **selección** e **iteración** (repetición).
- Diseño descendente (Top-Down): descomposición en problemas más simples.
- Programación Modular: descomposición del programa en módulos independientes.
- Estructuras de datos: conjunto o colección de valores.

Un programa estructurado se compone de **funciones**, **módulos** y/o **subrutinas**, cada una con una sola entrada y una sola salida. En ejecución no tiene partes por las cuales nunca pasa, ni tiene ciclos infinitos.

**Objetivo**: desarrollar programas fáciles de escribir, verificar, leer (por el usuario) y mantener (modificar).

## **P. E.: VENTAJAS**

- Los programas son más fáciles de entender.
- Reducción del esfuerzo en las pruebas.
- Programas más sencillos y más rápidos.
- Facilidad para la transformación y conversión de un lenguaje a otro.
- Reducción de los costos de mantenimiento.
- Aumento en la productividad del programador.

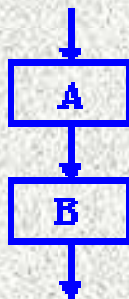


## P. E.: ESTRUCTURAS DE CONTROL

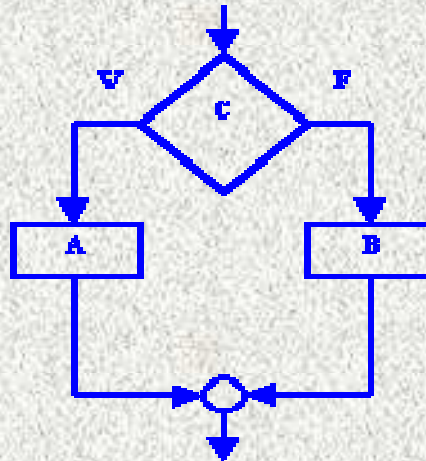
1. **Secuencia:** Sucesión simple de dos o mas instrucciones, una después de la otra, en el mismo orden en que aparecen; sin cambios de ruta.
2. **Selección:** Estructura SI-VERDADERO-FALSO (básica), plantea la selección entre dos alternativas, evaluando una condición. Equivale a la instrucción **if** de todos los lenguajes de programación.
3. **Iteración:** Estructura HACER-MIENTRAS (básica), ejecución repetida de una o mas instrucciones mientras que se cumple una determinada condición.

### REPRESENTACION GRAFICA

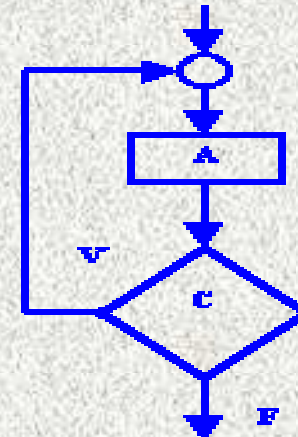
SECUENCIA



SELECCIÓN

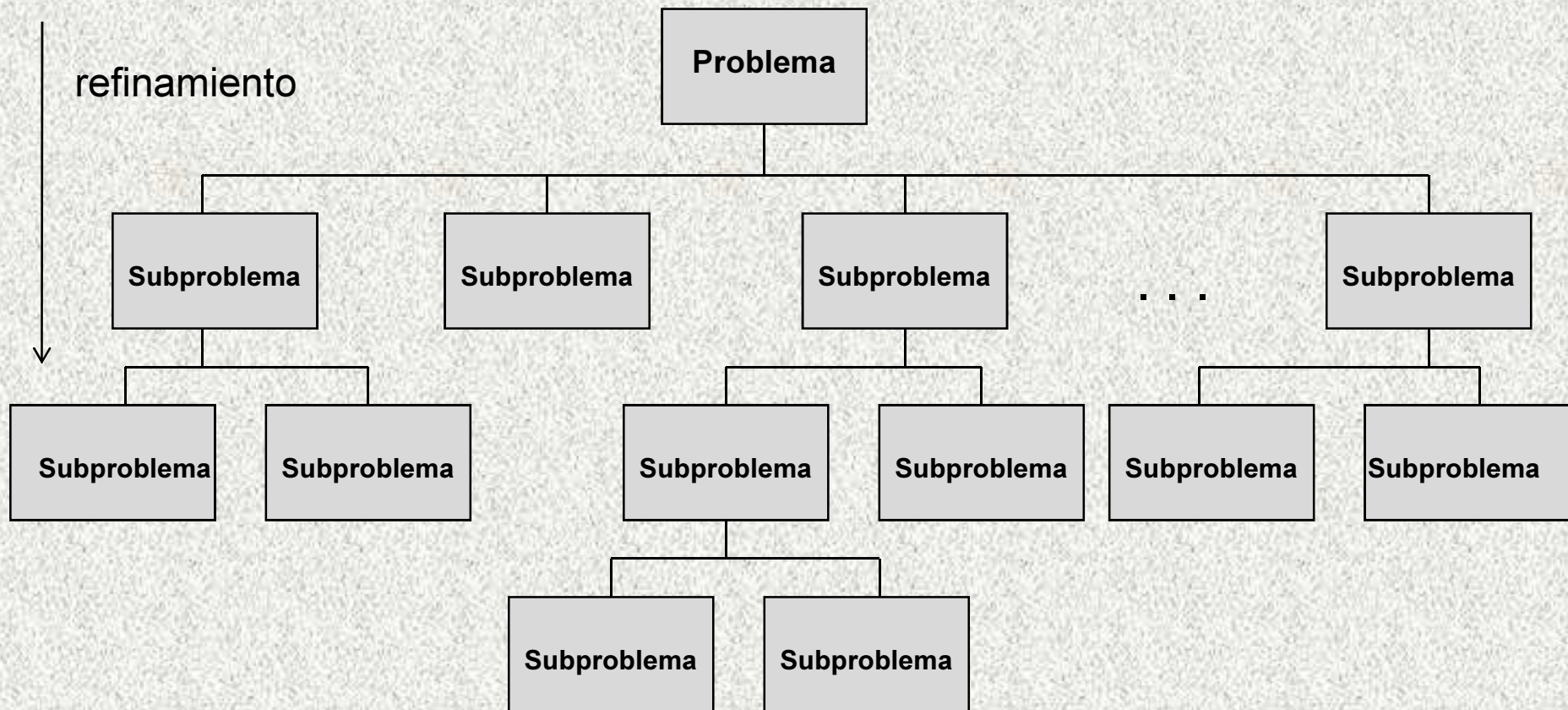


ITERACION



## P. E.: DISEÑO TOP-DOWN

1. Descomposición del problema en subproblemas más simples.
2. Resolver cada subproblema combinando las estructuras de control.
3. Ensamblar las soluciones de cada subproblema.

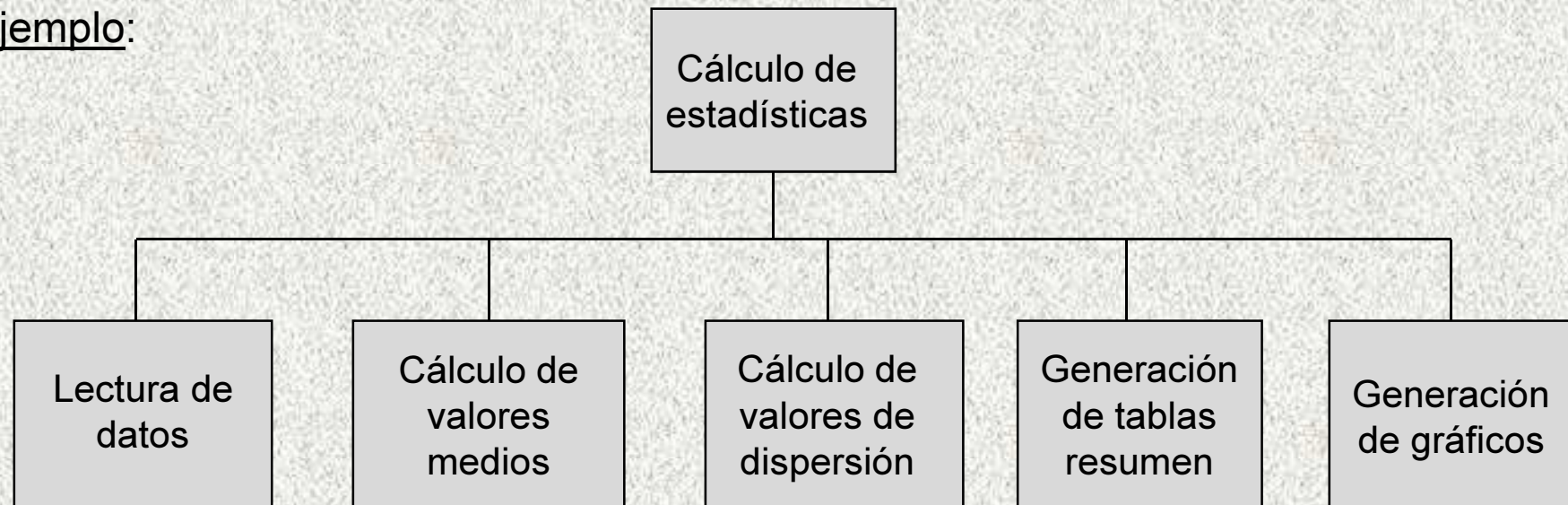


Ventajas: Reutilizar algoritmos existentes. Permite la programación modular.

## P. E.: PROGRAMACION MODULAR

Consiste en la descomposición del programa en módulos independientes, cada uno de los cuales ejecuta una única actividad o función. Cada módulo se analiza, se diseña, se codifica y se verifica por separado. El programa es una jerarquía de módulos, con uno como principal (programa principal) con función de controlador. Este transfiere el control a los módulos inmediatamente subordinados (o subprogramas), para que ejecuten sus funciones. Una vez que el módulo subordinado completó su tarea, devuelve control al módulo principal.

Ejemplo:





## P. E.: ESTRUCTURA DE DATOS

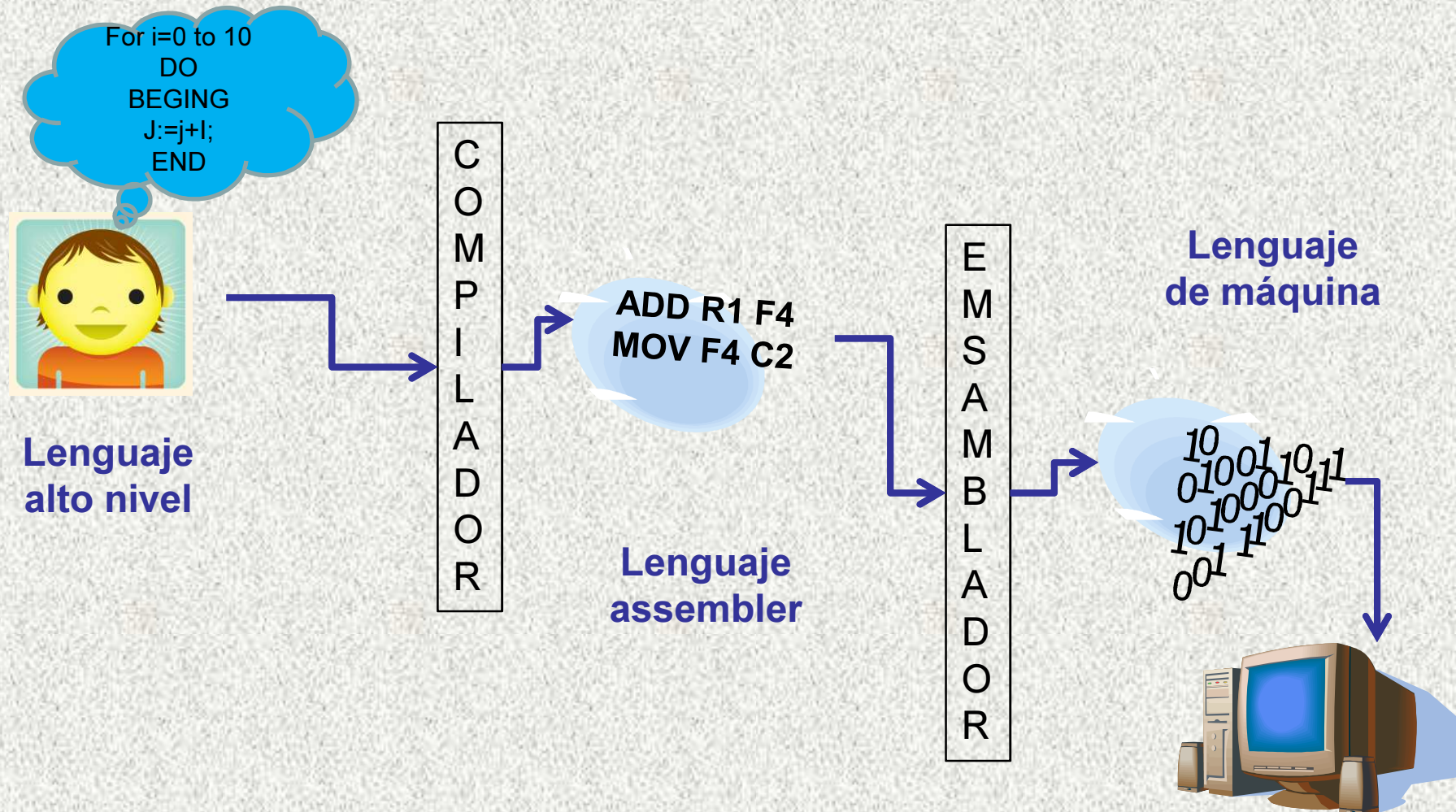
Es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manejo: **vectores**, **matrices**, listas, árboles, grafos, etc.

Cada estructura define la organización de los datos y un conjunto de operaciones que se pueden realizar sobre ellos:

- Acceder a un determinado dato para utilizar o modificar su valor.
- Incorporar un nuevo componente a la estructura.
- Eliminar un componente de la estructura.
- Encontrar un determinado valor de acuerdo a un criterio de búsqueda.
- Ordenar los elementos pertenecientes a la estructura.
- Determinar si una estructura es igual o está contenida en otra.

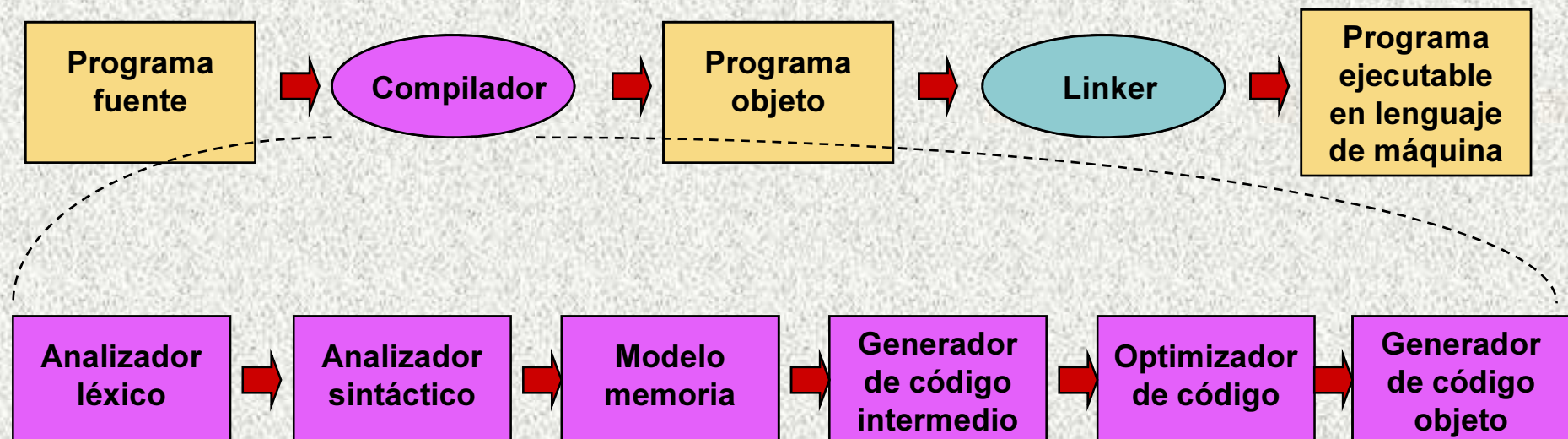
Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación. La elección de la más apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos.

# LENGUAJES DE PROGRAMACIÓN



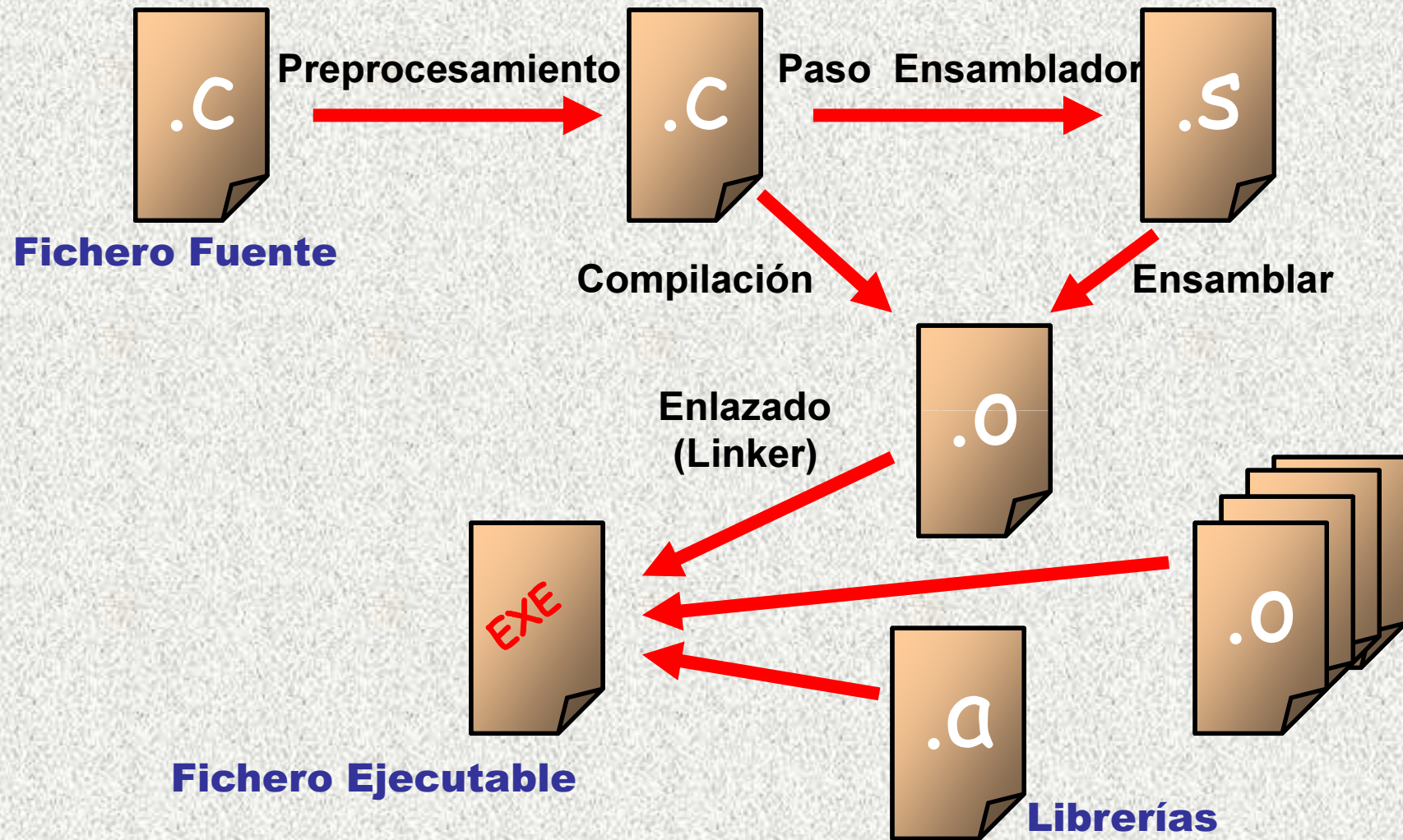
# LENGUAJES DE PROGRAMACIÓN: ALTO NIVEL

- Es un **traductor** que transforma los bloques del lenguaje en bloques binarios.
- Existen dos tipos de traductores: **compiladores** e **intérpretes**.
- Ventajas: facilidad de aprendizaje, lectura/escritura, detección y corrección de errores (**debugger**) y transformación o conversión a otro lenguaje.
- Lenguaje de programación = Léxico + Sintaxis + Gestión memoria + Excepciones
- Fases en la compilación de un programa:





# LENGUAJE ALTO NIVEL: Archivos



# LENGUAJES DE PROGRAMACIÓN (1/4)

- **FORTTRAN** ( **FOR**mula **TRAN**slation): Job Backus (1957), empleado de IBM. Fue el primer lenguaje de alto nivel. Se caracteriza por su potencia en los cálculos matemáticos pero está limitado en lo relativo al tratamiento de datos no numéricos
- **COBOL** ( **CO**mmon **B**usiness-**O**riented **L**anguage): (1959) creado por fabricantes de computadoras, usuarios y el Departamento de Defensa (USA). Actualizado en diversas oportunidades para incorporar variables locales, recursividad, reserva de memoria dinámica y características de programación estructurada y orientada a objetos. Se buscó evitar errores de redondeo en los cálculos (conversión decimal-binario). Utilizado en bancos y grandes sistemas contables.
- **ALGOL** ( **ALGO**rithmic **L**anguage): (1960) creado por un comité internacional. Estructuras de bloques anidadas: las secuencias de código y las declaraciones se pueden agrupar en bloques sin tener que estar en procedimientos separados.
  - Ámbito léxico: un bloque puede tener sus propias variables, procedimientos y funciones, invisible al código fuera de dicho bloque.

## LENGUAJES DE PROGRAMACIÓN (2/4)

- **BASIC** (**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode): (1964) John George Kemeny y Thomas Eugene Kurtz. Objetivo: facilitar la programación a estudiantes y profesores que no tuvieran formación científica. Se utilizó en casi todas las microcomputadoras a finales de los '70 y en los '80. Microsoft creó versiones MS/PC-DOS (BASICA, GW-BASIC y Quick BASIC). Aparecieron además extensiones para computadoras caseras, con capacidad para gráficos, sonido, y comandos DOS. A fines de los '80 surgieron interfaces gráficas que no eran soportadas.
- **Pascal**: Niklaus Wirth (1970). Creado para facilitar el aprendizaje de programación a sus alumnos, utilizando la programación estructurada y estructuración de datos. Excedió el ámbito académico para convertirse en una herramienta para la creación de aplicaciones de todo tipo. Es un lenguaje estructurado fuertemente tipado:
  - El código está dividido en porciones fácilmente legibles llamadas funciones o procedimientos, en oposición al antiguo estilo de programación monolítica.
  - El tipo de dato de todas las variables debe ser declarado previamente.



## LENGUAJES DE PROGRAMACIÓN (3/4)

- **C**: Brian Kernighan y Dennis M. Ritchie (1970~1972). Evolución del **B** (Laboratorios Bell). En un principio se orientó a la implementación de S.O. (Unix). Permite crear aplicaciones con un código muy eficiente (usado para software de sistemas). Dispone de las estructuras típicas de los lenguajes de alto nivel y de instrucciones que permiten un control a muy bajo nivel: los compiladores ofrecen extensiones que posibilitan mezclar código (assembler/C). Posibilita además un acceso directo a memoria o dispositivos periféricos. Es altamente transportable y muy flexible.
- **C++**: Bjarne Stroustrup (mediados '80). Extensión C (99% de compatibilidad) para incluir la **Programación Orientada a Objetos (PPO)**. Permite redefinir los operadores (**sobrecarga**), y crear nuevos tipos de datos que se comporten como tipos fundamentales. Incorpora el manejo de excepciones y plantillas (**templates**). Permite la programación multihilos (**multithreading**) y el acceso a conexiones entre computadoras (Internet).

## LENGUAJES DE PROGRAMACIÓN (4/4)

- **Visual Basic**: Alan Cooper (Microsoft) (1991). Dirigido por eventos. Contiene un entorno de desarrollo integrado (**IDE**) que integra editor de textos (código fuente), un debugger y un editor de interfaces gráficas (**GUI**) que facilita la programación de aplicaciones que interactúan con el usuario.
- **JAVA**: James Gosling (Sun Microsystems → Oracle) (1995). Su sintaxis deriva de C y C++, con menos utilidades de bajo nivel. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora usada. Es orientado a objetos e incluye soporte para trabajo en red.
- **.NET**: en 2001 Microsoft propuso abandonar el desarrollo basado en la **API** Win32 (Application Programming Interface) y pasar a un marco común de librerías (**framework**) independiente de la **versión** del sistema operativo: (**.NET**). Surgen las versiones de Visual Basic, C++ y C# (C Sharp). Existe un compilador para C# que genera aplicaciones compatibles con Windows, Unix, Android, etc.



# LENGUAJES DE PROGRAMACIÓN: C

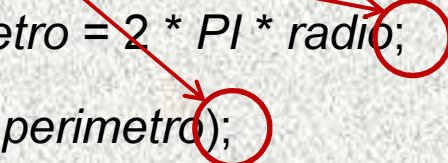
- **Características:**
  - cumple con los principios de la programación estructurada.
  - Abundancia de operadores y tipos de datos.
  - Codificación en alto nivel y bajo nivel.
  - Producción de código objeto altamente optimizado
- **Alfabeto:** pueden aparecer letras (excepto ñ y acentos), números y algunos caracteres especiales. El compilador distingue entre mayúsculas y minúsculas.
- **Léxico:** elementos básicos con los que se construyen los programas:
  - Palabras clave o reservadas: siempre en minúscula (include, define, main, etc.).
  - Separadores: espacios en blanco, saltos de línea, tabuladores.
  - Operadores: aritméticos, lógicos, de asignación, etc.
  - Identificadores: nombres de las variables y funciones definidas por el programador. No se pueden utilizar las palabras clave.
  - Constantes: valores que no cambian durante el programa (Ej.  $\pi = 3.14159$ ).



# LENGUAJE C: elementos de un programa

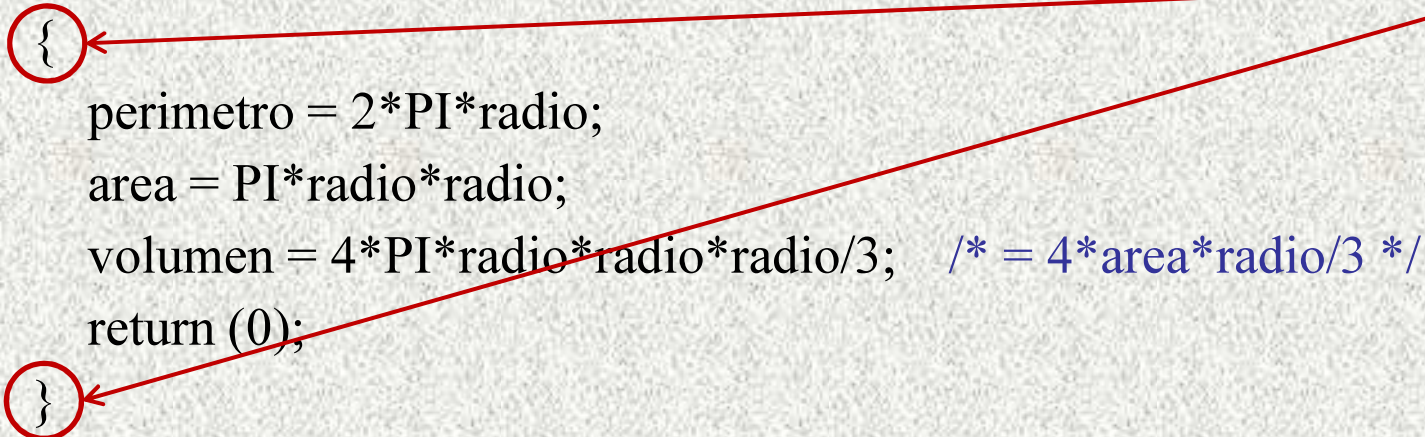
- **Datos u objetos:** información que se procesa (ej.: *radio*).
- **Expresiones:** combinación de datos mediante operadores ( $2*PI*radio$ ).
- **Instrucciones = sentencias:** acciones a realizar sobre los objetos. Todas deben acabar en ';':

```
perimetro = 2 * PI * radio;  
printf(perimetro);
```



- **Bloques:** conjunto de sentencias agrupadas. Se delimitan con llaves: { .... }

```
{  
    perimetro = 2*PI*radio;  
    area = PI*radio*radio;  
    volumen = 4*PI*radio*radio*radio/3; /* = 4*area*radio/3 */  
    return (0);  
}
```



- **Comentarios:** pueden aparecer en cualquier parte del programa. Todo texto ubicado entre los limitadores `/* esto es un comentario */`.

# LENGUAJE C: programa ejecutable

- Los prototipos de las funciones usadas por varios ficheros fuente se suelen definir en un **fichero de cabecera**, que en general tiene como extensión **.h**. Permite reutilizar código ya escrito. No se incluye la implementación de la función, sólo su declaración.
- Las funciones se agrupan en librerías, que se incluyen en el código fuente mediante la directiva (orden para el preprocesador o el compilador) **#include**.
- Programa básico:

```
#include <stdio.h>      /* archivo de cabecera, contiene la función printf */

int main()              /* función principal (la primera que se ejecuta) */
{
    /* este programa imprime un saludo en pantalla */
    printf("Hola Mundo");
    return (0);          /* valor que retorna (devuelve) la función */
}
```