



# Tema 1

---

## Introducción a la programación

1



## Introducción. **Conceptos previos**

- **Informática (RAE):**

- Conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de computadoras electrónicas.

- **Computadora:**

- máquina capaz de aceptar unos *datos* de entrada, efectuar con ellos operaciones lógicas y aritméticas, y proporcionar la información resultante a través de un medio de salida.

2

## 1.1. Arquitectura Básica

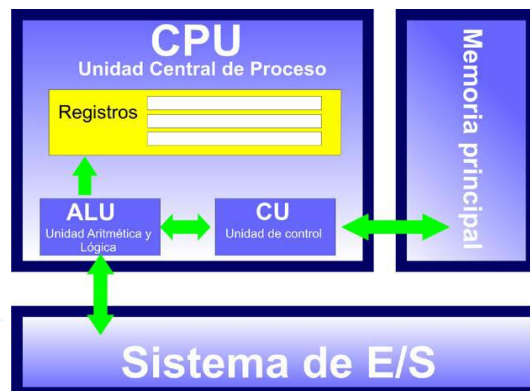
☒ Basada en la arquitectura inventada por Von Neumann en 1945.

☒ Idea: “Crear una máquina capaz de ejecutar instrucciones leídas de memoria y ejecutadas en la CPU comunicándose con el exterior a través de la Unidad de E/S”



3

## ARQUITECTURA DE VON NEUMANN



4



## ARQUITECTURA DE VON NEUMANN

- Memoria Principal: espacio de almacenamiento temporal dividido en celdas de igual tamaño destinado a almacenar instrucciones y datos.
- Unidad Central de Proceso: Se encarga de la ejecución de las instrucciones almacenadas en la memoria.
- Unidad Aritmética: encargada de realizar las operaciones aritméticas (sumas, restas, ...) y lógicas (and, or, not, ...).
- Unidad de Control: Su función es decodificar las instrucciones del programa en ejecución y generar todas las señales necesarias para que puedan ser ejecutadas.
- Unidad de E/S: permite la comunicación de la CPU y la memoria con el exterior: impresora, monitor, teclado, ...
- Registros: almacén temporal que se usa durante la ejecución de las instrucciones.

5



## 1.2. TIPOS DE DISPOSITIVOS DE E/S

- Dispositivos de Entrada: permiten al usuario introducir la información en el sistema. Los datos leídos se almacenan en memoria. Ejemplos: teclado, ratón, escanner, webcam ...
- Dispositivos de Salida: son los encargados de mostrar los resultados obtenidos por la computadora al usuario. Ejemplos: monitor, impresora, ...
- Dispositivos de Entrada/Salida : permiten tanto la entrada de información en la computadora como la salida de la misma. Ejemplos: tarjeta de red, módem, etc.

6



## 2. Sistemas Operativos. Tipos

. **Definición:** Un Sistema Operativo es un programa o conjunto de programas que actúa como intermediario entre el usuario y el hardware del ordenador, ocultando su complejidad mediante una interfaz sencilla de utilizar.

. **Objetivos:**

- ♦ Hacer cómoda la utilización de la computadora.
- ♦ Utilizar recursos de la computadora de forma eficiente.

7



## Funciones del SO

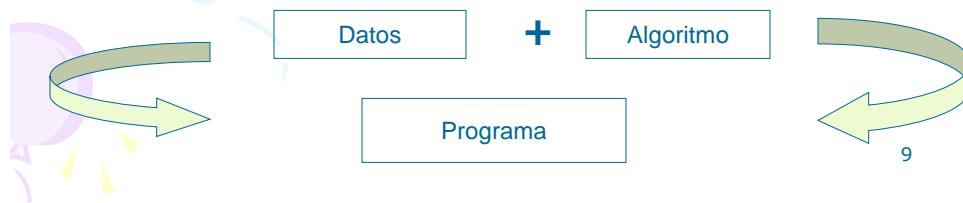
- . Gestión de procesos
- . Gestión de memoria
- . Gestión de dispositivos de E/S
- . Gestión del sistema de ficheros
- . Gestión de la red
- . Protección

Gran parte de estas funciones las realiza un componente importante del SO llamado kernel que es la parte del SO residente en memoria.

8

## El ordenador y los algoritmos (1)

- Un ordenador es una máquina que ejecuta **algoritmos**
- **Algoritmo (RAE)**: Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.
- La ejecución o procesamiento de un algoritmo supone la transformación de una información de salida o resultados.



## El ordenador y los algoritmos (2)

- Una tarea ejecutable por un ordenador da lugar a un problema algorítmico
- El algoritmo es la solución a ese problema (**software**)
- Cuando un algoritmo es ejecutado por los circuitos de un ordenador (**hardware**) se origina un **proceso** que genera resultados a partir de los datos.
- Un **proceso** se caracteriza por una sucesión de estados de determinadas magnitudes que están almacenadas en la memoria del ordenador, y que llamaremos **variables**.



# características de los algoritmos

- **Un algoritmo debe resolver el problema para el que fue formulado.** Lógicamente no sirve un algoritmo que no resuelve ese problema. En el caso de los programadores, a veces crean algoritmos que resuelven problemas diferentes al planteado.
- **Los algoritmos son independientes del ordenador.** Los algoritmos se escriben para poder ser utilizados en cualquier máquina.
- **Los algoritmos deben de ser precisos.** Los resultados de los cálculos deben de ser exactos, de manera rigurosa. No es válido un algoritmo que sólo aproxime la solución.
- **Los algoritmos deben de ser finitos.** Deben de finalizar en algún momento. No es un algoritmo válido aquel que produce situaciones en las que el algoritmo no termina.
- **Los algoritmos deben de poder repetirse.** Deben de permitir su ejecución las veces que haga falta. No son válidos los que tras ejecutarse una vez, ya no pueden volver a hacerlo por la razón que sea.

11



## Algoritmos. Datos formales

- Los algoritmos se construyen utilizando elementos simples para que el lenguaje se parezca más al de las computadoras
- Los datos de los algoritmos son:
  - **Números** (10, 25, 5.32)
  - **Textos** ("Hola mundo")
  - **Lógicos** (VERDADERO, FALSO, true, false)
  - Datos compuestos (registros, listas)

12



## Algoritmos. Datos formales

- Los algoritmos pueden tener **expresiones**, las cuales se componen de:
  - **Variables** (*x, resta, saldo*)
  - **Operadores** (+, -, \*, >, <, Y(AND))
- Las instrucciones de los algoritmos pueden ser
  - **Primitivas** (*entrada, salida, asignación*)
  - **De control** (*bucle, condición*)

13



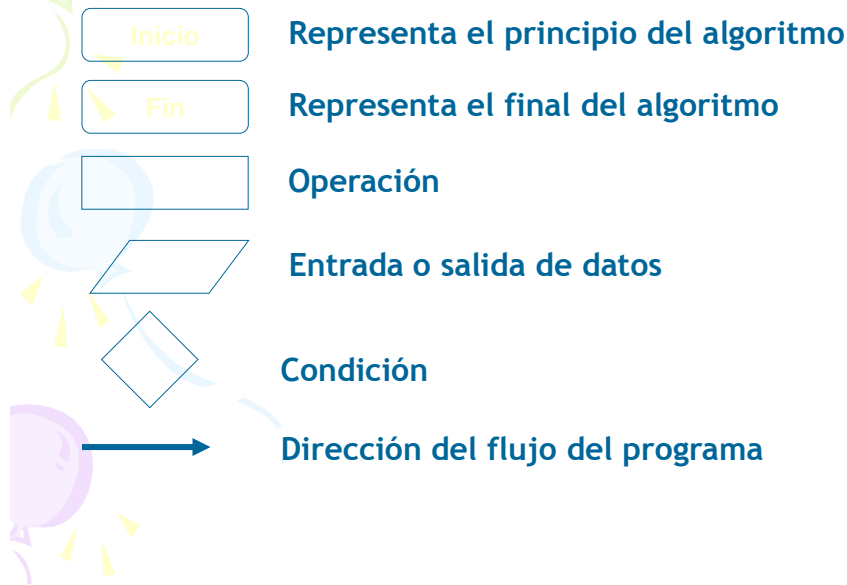
## Representación de algoritmos

- Existen códigos especiales que sirven para representar algoritmos
- La razón de su uso es normalizar la escritura de algoritmos
- Hay **diagramas de flujo** y **pseudocódigo**
- El primero trabaja con representaciones gráficas, el segundo con un lenguaje especial

14




# Diagramas de flujo



15



# Pseudocódigo

- 
- Es otra forma de representar algoritmos
  - Se asemeja más a los lenguajes de programación de alto nivel



16





## Pseudocódigo. Instrucciones (1)

- **ESCRIBIR.** Muestra algo en el dispositivo de salida (la pantalla normalmente)
- **LEER** *variable*. Lee un dato (del teclado) y lo almacena en esa variable
- **←**. Asignar valor
- Comparaciones:  $<$   $>$   $\neq$   $\leq$   $\geq$

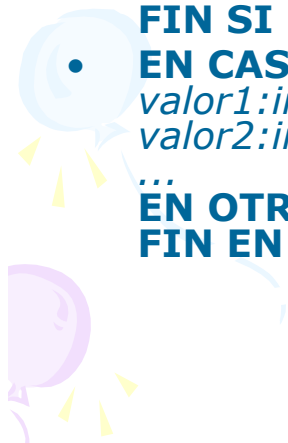


17



## Pseudocódigo. Instrucciones (2)


- **SI** *condición* **ENTONCES** *instrucciones*  
**[SINO** *instrucciones*  
**FIN SI**
- **EN CASO DE** *variable*  
*valor1:instrucciones*  
*valor2:instrucciones*  
...  
**EN OTRO CASO:** *instrucciones*  
**FIN EN CASO DE**



18



## Pseudocódigo. Instrucciones (3)

- **MIENTRAS** *condición* **HACER**  
*instrucciones*  
**FIN MIENTRAS**
  - **REPETIR**  
*instrucciones*  
**HASTA** *condición*
  - **PARA** *variable* **DE** *valorInicial* **A**  
*valorFinal* [**INCREMENTO** *valor*]  
*instrucciones*  
**FIN PARA**
- 

19



## Lenguajes de programación. Tipos (I)

### Clasificación cronológica



• **Lenguajes de primera generación:** A principio de los 50 se usaba el lenguaje máquina y el ensamblador.

• **Lenguajes de segunda generación:** a finales de los 50 principios de los 60 se amplía el uso de los lenguajes para las aplicaciones científico-militares y aparecen las aplicaciones de gestión. Fortran, Cobol y Algol.

• **Lenguajes de tercera generación:** nacen con la programación estructurada y presentan potentes posibilidades de estructuración de datos y procedimientos. Pascal, C, etc.

• **Lenguajes de cuarta generación:** combinan características procedurales y no procedurales. SQL, OASIS, lenguajes de generación de informes, etc.

20




## Lenguajes de programación. Tipos (II)

### Según su proximidad al lenguaje máquina

**Lenguajes de bajo nivel:** la única ventaja de estos lenguajes es que ocupan poco espacio en memoria y se ejecutan con mayor rapidez.

- ♦ **Lenguaje máquina:** cada instrucción se codifica como una secuencia de 1's y 0's, lo que dificulta el desarrollo de programas complejos.
- ♦ **Lenguaje ensamblador:** cada instrucción en ensamblador equivale a una instrucción en lenguaje máquina. Utiliza palabras mnemotécnicas en lugar de cadenas de bits.

21



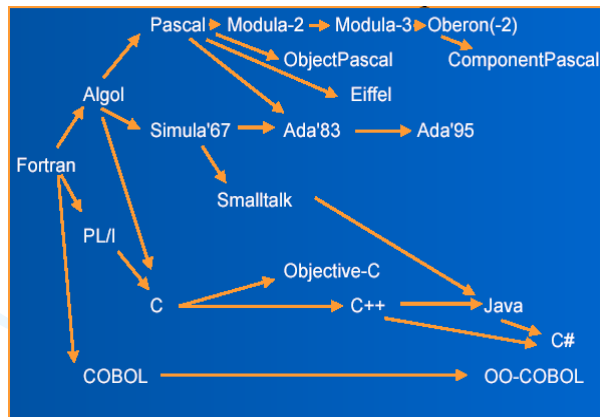
## Lenguajes de programación. Tipos (III)

**Lenguajes de alto nivel:** surgen con posterioridad a los anteriores, se caracterizan por:

- ♦ **Independencia de la máquina:** se puede usar el mismo programa en diferentes equipos con la única condición de disponer de un programa traductor o compilador.
- ♦ **Aproximarse al lenguaje natural:** se sustituyen los mnemotécnicos por sentencias if...then...else, entre otras.
- ♦ **Incluir rutinas de uso frecuente:** disponen de una serie de librerías que se pueden utilizar siempre que se quiera sin necesidad de programarlas cada vez. Funciones matemáticas, manejo de cadenas, etc.

22

# Lenguajes de alto nivel



23

## Lenguajes de programación. Tipos

### Según funcionalidad

■ **Lenguajes para la educación:** Logo, Pilot, Eiffel, ...

■ **De propósito general:** concebidos no para una aplicación concreta, sino para ser eficaces en diversos campos. Pascal, C

■ **Lenguajes de inteligencia artificial:** creados para la emulación de programas que emulan comportamientos inteligentes. Lisp, Prolog.

■ **Orientados a la gestión:** las aplicaciones de gestión se caracterizan por tener la necesidad de manejar grandes volúmenes de información de forma fiable. Cobol, Clipper, PL/SQL, ...

■ **Científicos:** se caracterizan por realizar pocas operaciones de E/S y por el contrario una gran cantidad de cálculos complejos. Fortran, Apl, ...

■ **Orientados a Internet:** son lenguajes transversales a la máquina y al sistema operativo. PHP, Java.

■ **Orientados a tiempo real:** pensados para dar respuesta a exigentes requisitos temporales.

24

# Lenguajes de programación. Tipos

## Según traducción a código máquina

**Interpretados:** la máxima es “sentencia traducida, sentencia ejecutada”. Toma un programa fuente y lo va traduciendo y ejecutando simultáneamente.

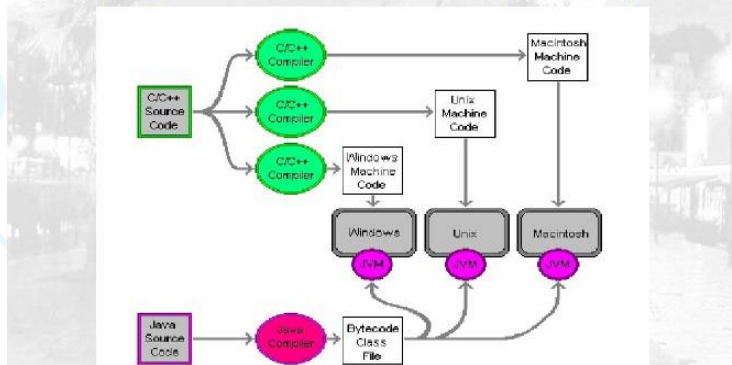
- ♦ **Ventajas:** posee una sintaxis más elaborada que la de los ensambladores. Además la velocidad de traducción depende fundamentalmente de la sintaxis del lenguaje. Cuanto más simple, más rápido será el análisis de cada sentencia.
- ♦ **Inconvenientes:** una sentencia que tenga que ejecutarse varias veces tendrá que traducirse también varias veces.

La forma de evitar estos retardos consiste en adoptar una estrategia diferente a la traducción-ejecución. Consiste en traducir el programa completo y después ejecutarlo. Así surgen los compiladores.

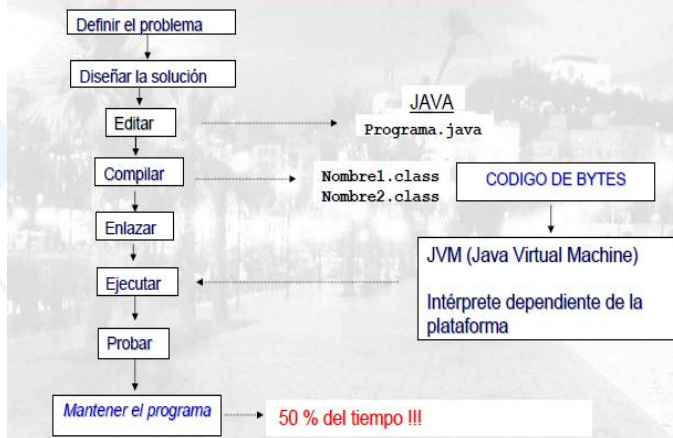
**Compilados:** un compilador es un traductor que genera un programa objeto (programa escrito en un lenguaje de bajo nivel) a partir de un programa fuente (programa en lenguaje de alto nivel). 25

## Compiladores e Interpretes: El caso de Java

*Todo programa Java es compilado y después interpretado*



## Fases de construcción de un programa



27

## ¿Cómo debe ser un programa?

- Correcto
  - ¿Quién quiere un programa incorrecto?, pero...*
  - ¿Existe la perfección?*
- Eficiente
  - Uso eficaz de los recursos (memoria, disco, CPU ...)*
- Fácilmente modificable
  - Los clientes cambian.*
  - Los problemas evolucionan.*
  - La comprensión del problema y de la solución mejoran.*

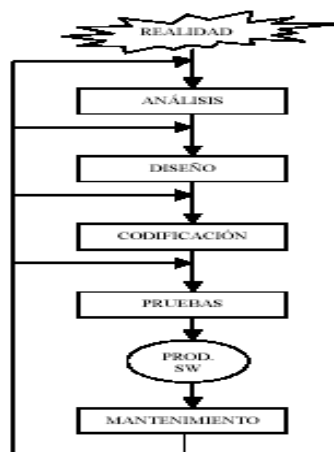
28

# CICLO DE VIDA

- Las distintas fases del ciclo de vida de un programa se realizan secuencialmente, en cada una de las etapas se genera una *documentación* que servirá para iniciar la siguiente. A este proceso se le conoce como *ciclo de vida clásico o en cascada*.
- En la elaboración de aplicaciones existen unas fases a las que se denominan Ciclo de vida, ya que una vez que llegamos a la última fase puede ser que sea necesario comenzar el ciclo de nuevo en otra fase anterior.

29

## Modelo Clásico o en cascada



30



## ANÁLISIS

Estudio de la situación y requisitos existentes. Se establece la viabilidad del proyecto. Existen diversas técnicas para realizar el análisis estructurado:

- ✓ Diagramas de flujos de datos
- ✓ Modelos de datos
- ✓ Diccionarios de datos
- ✓ Definición de las interfaces de usuario

31



## DISEÑO

Se establece una solución óptima con los suficientes detalles para la realización física de la aplicación.

Se pueden establecer las siguientes etapas:

- Diseño externo
- Diseño de datos
- Diseño modular
- Diseño procedimental

Al final de esta fase se obtiene el **cuaderno de carga**.

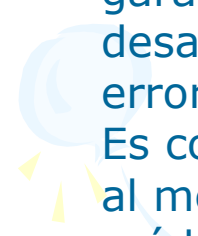
32







## **CODIFICACIÓN Y PRUEBAS**



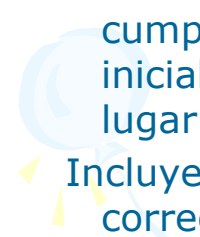
El objetivo de estas pruebas es garantizar que el sistema ha sido desarrollado correctamente, sin errores de diseño y/o programación. Es conveniente que sean planteadas al menos tanto a nivel de cada módulo (aislado del resto), como de integración del sistema.



33



## **EXPLOTACIÓN**



Esta etapa tiene como objetivo la verificación de que el sistema desarrollado cumple con los requisitos expresados inicialmente por el cliente y que han dado lugar al presente proyecto.



▶ Incluye: Formación de los usuarios (para la correcta utilización del sistema), y documentación necesaria para la explotación del sistema y el manual de uso o guía de la aplicación.

34



# MANTENIMIENTO

“DEPURAR, CORREGIR, MEJORAR Y ADAPTAR la aplicación si no responde exactamente a la solución buscada”.

Clases de mantenimiento:

- ✓ Correctivo
- ✓ Adaptativo
- ✓ Perfectivo

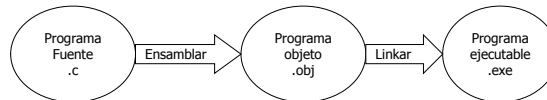


35



## Utilidades para la programación

- **Ensamblador/Linker:** un ensamblador traduce un programa en lenguaje ensamblador a código máquina. Cada instrucción en ensamblador genera una instrucción en código máquina. En el proceso de linkado se obtiene el programa ejecutable a partir del código máquina obtenido al ensamblar.



- Por lo tanto, un linker es un programa que toma los ficheros de código objeto, la información de todos los recursos necesarios (bibliotecas, librerías del lenguaje, etc), y enlaza el código objeto con su(s) biblioteca con lo que finalmente produce un fichero ejecutable (un fichero .exe).



36



## Utilidades para la programación (II)

- **Intérprete:** toma un programa fuente escrito en un lenguaje de alto nivel y lo va traduciendo y ejecutando simultáneamente.
  - **Compilador:** es un traductor que genera un programa objeto a partir de un programa fuente escrito en un lenguaje de alto nivel.
  - **Depuradores:** es una herramienta que permite depurar o limpiar los errores de un programa fuente. Normalmente esta herramienta está integrada en el entorno de programación del lenguaje.
  - **RAD:** abreviatura de Rapid Application Development. Se suele usar para referirnos a IDEs (Herramientas de Desarrollo Integrado) como Delphi, Foxpro, Eclipse, Visual Basic, etc.
  - **CASE:** Computer Aided SW Engineering, persigue proporcionar un conjunto de herramientas, bien integradas, que enlacen y automaticen todas las fases del ciclo de vida del SW.
- 