Presentación ***resumen*** del libro:

"EMPEZAR DE CERO A PROGRAMAR EN lenguaje C"

Autor: Carlos Javier Pes Rivas (correo@carlospes.com)

Capítulo 3

CICLO DE VIDA DE UN PROGRAMA

OBJETIVOS

- Saber qué es la **Ingeniería del Software**.
- Conocer los pasos que se tienen que dar para **desarrollar** un programa.
- Entender cómo el programador puede escribir unas instrucciones **comprensibles para la máquina**.
  - Hoy en día, la computadora es una herramienta indispensable en muchos ámbitos, tales como: la medicina, la astronomía, las comunicaciones, etc. Gracias a la informática, se han producido avances tecnológicos que eran impensables antes de la llamada ***"revolución de las computadoras"***. No obstante, la máquina no puede hacer *absolutamente nada* sin un software, es decir, sin un programa que le diga qué es lo que tiene que hacer. Pero, ¿cómo se hace un programa?

CONTENIDO

3.1 INTRODUCCIÓN (1/4)

- **Software a medida:**
  - Empresas (desarrolladora y cliente).
  - **EJEMPLOS:**
    - Gestión de la venta de billetes de un aeropuerto.
    - Gestión del alquiler de películas de un videoclub.
    - Gestión de los historiales médicos de los pacientes de un hospital.

- **Software no a medida:**
  - **EJEMPLOS:**
    - Procesadores de textos.
    - Juegos.
    - Enciclopedias electrónicas.

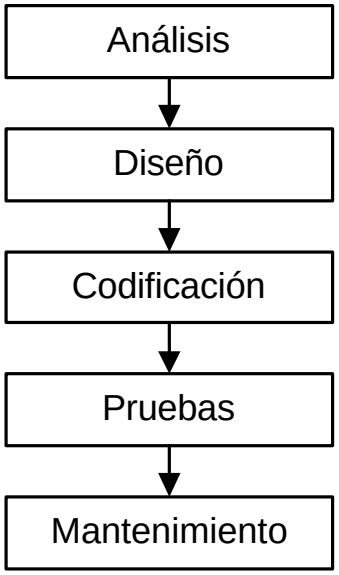
3.1 INTRODUCCIÓN (2/4)

- **Metodología de la programación:** Disciplina que alberga una serie de técnicas y conocimientos científicos relacionados con la informática.

- **Paradigmas de programación:**
  - **Programación estructurada.**
    - Aplicación del **diseño modular**.
    - Utilización, exclusivamente, de **estructuras secuenciales**, **alternativas** y **repetitivas**.
    - Empleo de **estructuras de datos** adecuadas para manipular información.
  - **Programación Orientada a Objetos.**

3.1 INTRODUCCIÓN (3/4)

- **Ingeniería del Software:**
  - Ciclo de vida de un programa (5 etapas).



  - Planificación de tareas (fechas de inicio y fin).
  - Control de tareas (seguimiento continuo del proyecto).

3.1 INTRODUCCIÓN (4/4)

- **Calidad del software:**
  - **Características operativas:**
    - **Corrección**, ¿hace lo que se espera de él?
    - **Eficiencia**, ¿se utilizan, óptimamente, los recursos de la computadora?
    - **Facilidad de uso**, ¿la interfaz es adecuada al usuario?
    - **Integridad**, ¿es seguro con respecto a los datos?
  - **Capacidad para sufrir cambios:**
    - **Facilidad de mantenimiento**, ¿es susceptible de ser corregido?
    - **Flexibilidad**, ¿es susceptible de ser cambiado?
    - **Facilidad de prueba**, ¿resulta fácil hacer pruebas?
  - **Adaptabilidad a nuevos entornos:**
    - **Reusabilidad**, ¿se puede usar parte del software en otro proyecto?
    - **Facilidad de interoperación**, ¿puede interactúar con otros SI?
    - **Portabilidad**, ¿se puede usar en otra máquina (procesador)?

3.2 ANÁLISIS (1/2)

- **Especificación de Requisitos Software (ERS):**
  - ¿**Qué** tiene que hacer el programa?
    - **Comportamiento interno** (gestión de los datos).
    - **Comportamiento externo** (interacción con el usuario y con otras aplicaciones).
  - **Contrato** (empresa desarrolladora y empresa cliente)
  - **PREGUNTAS:**
    - ¿Qué debe hacer el programa?
    - ¿Qué datos de entrada y de salida intervienen en el proceso?
    - ¿En qué máquina y sistema operativo se va a ejecutar?
    - ¿Quién será el usuario de la aplicación?
    - Tratamiento de errores, seguridad de los datos, etc.

3.2 ANÁLISIS (2/2)

- **PROBLEMA EJEMPLO: Suma de dos números enteros**

  1) En primer lugar, el programa debe pedir por teclado dos *números* (*datos enteros*).

  2) A continuación, calculará la *suma* de los dos *números* introducidos por el usuario.

  3) Finalmente, tiene que mostrar por pantalla el resultado obtenido (*dato entero*).

  En pantalla se mostrará:

```
Introduzca el primer número (entero): 3
Introduzca el segundo número (entero): 5
La suma es: 8
```

3.3 DISEÑO (1/6)

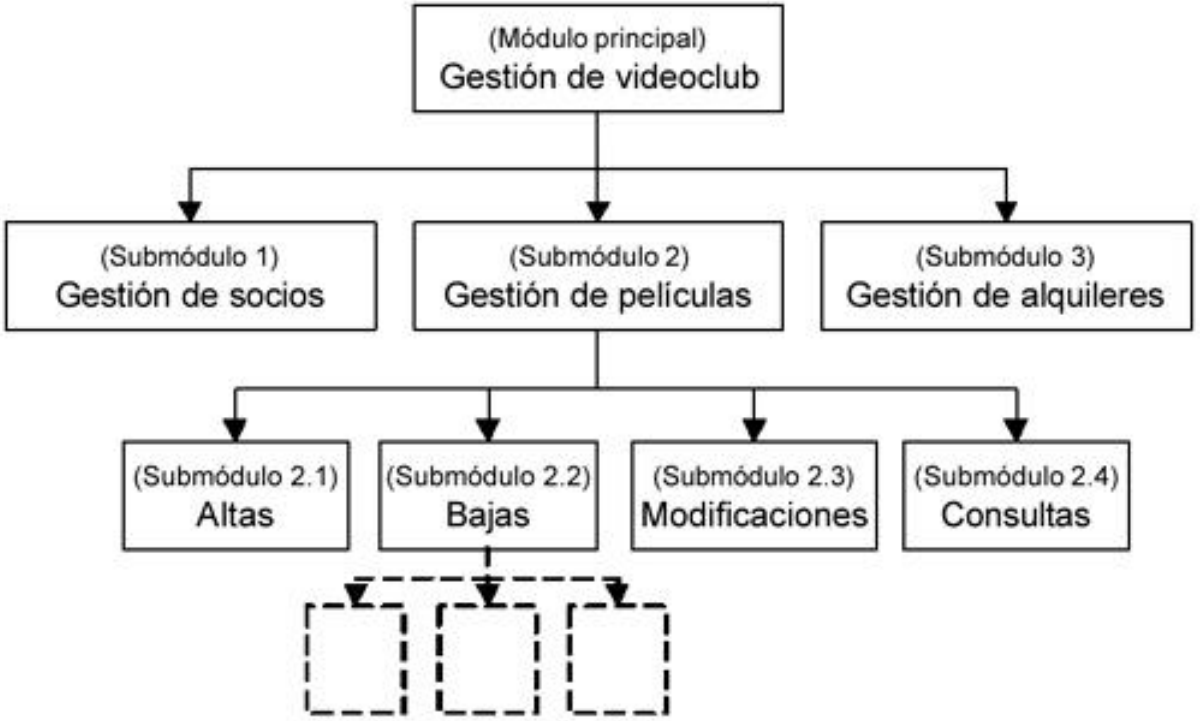
- Se tiene que encontrar una **solución informática** al problema planteado.

- Dicha solución determinará **cómo** se va a resolver el problema.

- Uso del **diseño modular** o **descendente** (*top-down*).
  - **Divide y vencerás** (subproblemas)
  - **Abstracción**
  - **Encapsulación**

3.3 DISEÑO (2/6)

- **EJEMPLO DISEÑO MODULAR: Gestión de un videoclub**



3.3 DISEÑO (3/6)

- **Algoritmo:**
  - Establece de manera genérica e informal, la secuencia de ***pasos*** o ***acciones*** que resuelve un determinado problema.
  - **Notación:**
    - Pseudocódigo
    - Diagramas de flujo (ordinogramas)
- **Pseudocódigo:**
  - Es un lenguaje de programación **algorítmico**.
  - Es un lenguaje **intermedio** entre el lenguaje natural y cualquier lenguaje de programación específico, como son: C, FORTRAN, Pascal, etc.
- **Ordinograma:**
  - Representa, de manera gráfica, el **orden** de los pasos o acciones de un algoritmo.

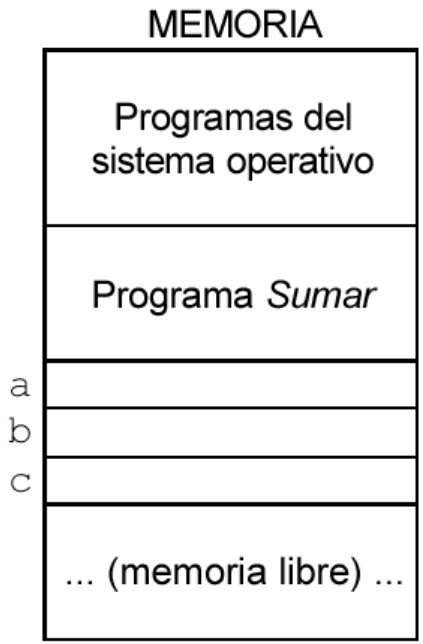
3.3 DISEÑO (4/6)

- **ALGORITMO EJEMPLO: Suma de dos números enteros**

  (Pseudocódigo)

Algoritmo Sumar

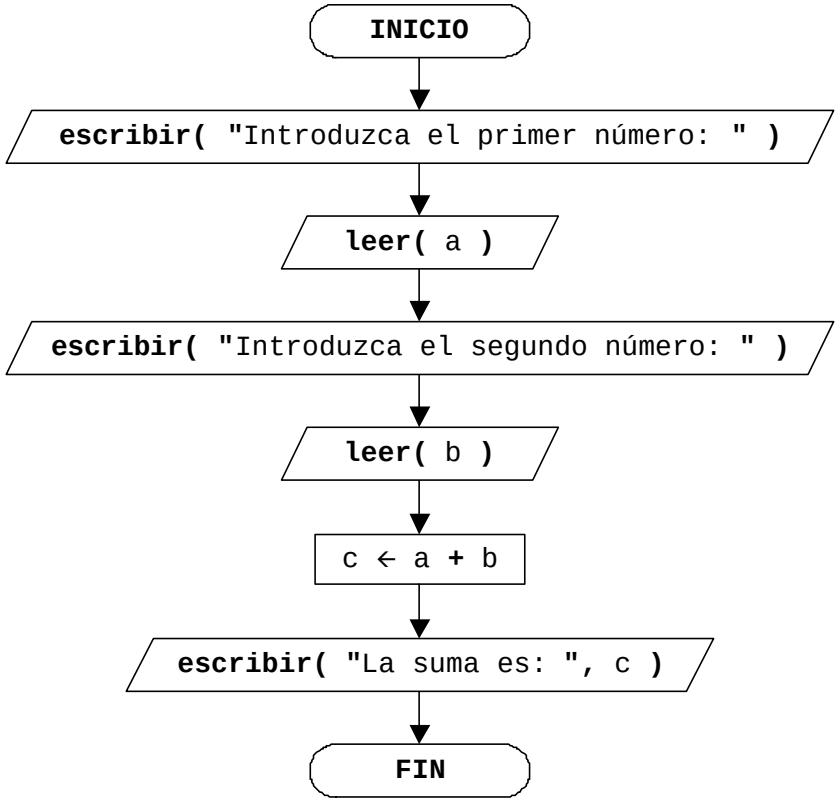
```
1   algoritmo Sumar
2
3   variables
4     entero a, b, c
5
6   inicio
7     escribir( "Introduzca el primer número (entero): " )
8     leer( a )
9     escribir( "Introduzca el segundo número (entero): " )
10    leer( b )
11    c ← a + b
12    escribir( "La suma es: ", c )
13  fin
```



3.3 DISEÑO (5/6)

- **ALGORITMO EJEMPLO: Suma de dos números enteros**

  (Ordinograma)



3.3 DISEÑO (6/6)

- **Cualidades de un algoritmo:** (solución más óptima)
  - **Finitud**
  - **Precisión**
  - **Claridad**
  - **Generalidad**
  - **Eficiencia**
  - **Sencillez**
  - **Modularidad**

3.4 CODIFICACIÓN (1/9)

- **EJEMPLO (lenguaje C): Suma de dos números enteros**
  - Las acciones definidas en el algoritmo hay que convertirlas a instrucciones (**sentencias**).

Programa Sumar

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int a, b, c;
6
7    printf( "\n   Introduzca el primer numero (entero): " );
8    scanf( "%d", &a );
9    printf( "\n   Introduzca el segundo numero (entero): " );
10   scanf( "%d", &b );
11   c = a + b;
12   printf( "\n   La suma es: %d", c );
13
14   return 0;
15 }
```

3.4 CODIFICACIÓN (2/9)

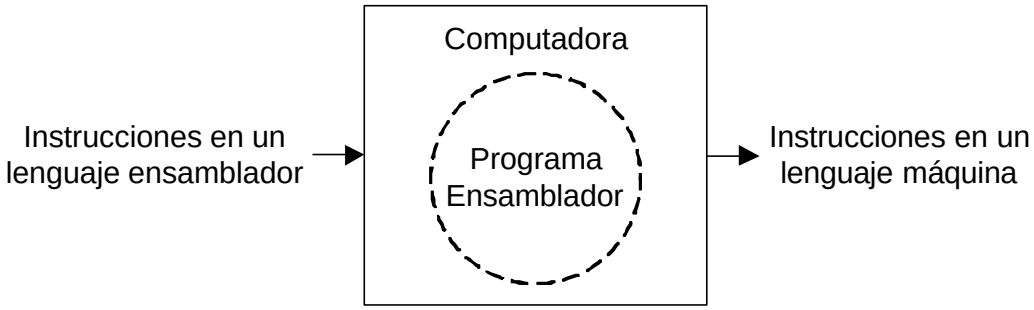
- **Lenguaje de programación:**
  - Es un lenguaje artificial que permite escribir las instrucciones de un programa informático.
  - Permite al programador comunicarse con la computadora para decirle qué es lo que tiene que hacer.
  - **Tipos de lenguajes:**
    - máquina
    - de bajo nivel
    - de alto nivel

3.4 CODIFICACIÓN (3/9)

- **Lenguaje máquina o <u>binario</u>:**
  - Es el **<u>único</u>** que entiende la computadora digital.
  - Dos símbolos: el *cero* (**0**) y el *uno* (**1**).
  - **Inconveniente: No es portable.**

```
10100010
11110011
00100010
00010010
```

- **Lenguajes de bajo nivel o <u>ensambladores</u>:**
  - Más entendibles para el programador.
  - Abreviaturas del inglés (nemotécnicos): ADD, DIV, SUB, etc.
  - **Inconveniente: No es portable.**



3.4 CODIFICACIÓN (4/9)

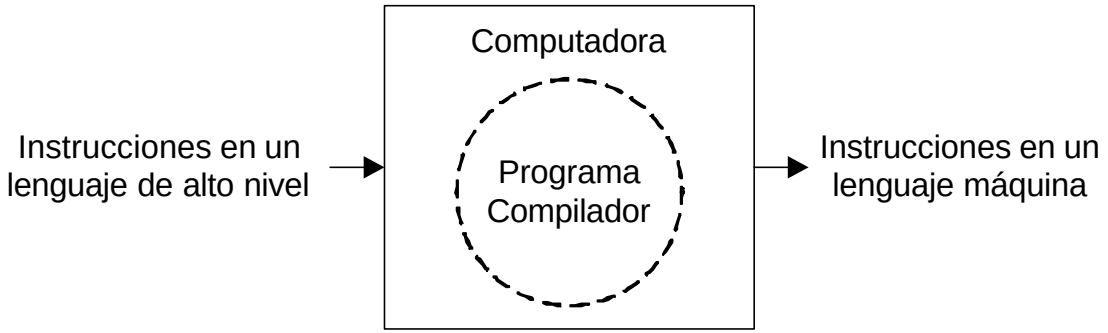
- **Lenguaje de alto nivel:** Se utilizan palabras o expresiones sintácticas muy similares al inglés (**case**, **if**, **for**, **while**,...).

  – **EJEMPLO:**

```
if ( numero > 0 ) printf( "El número es positivo" );
```

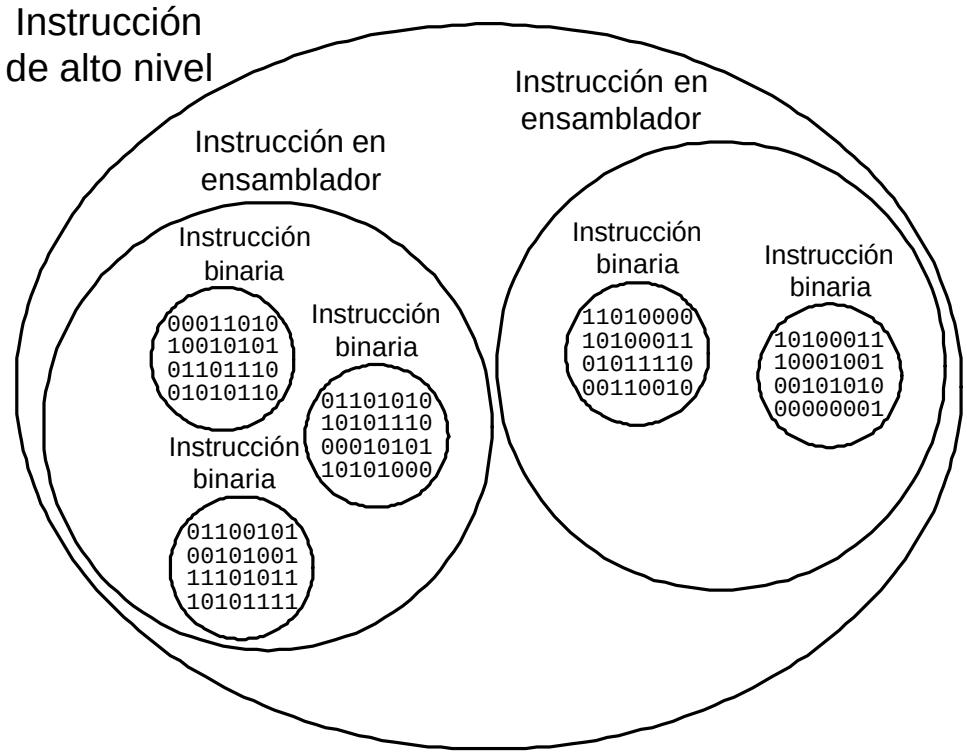
  Si numero es mayor que cero, entonces, escribir por pantalla el mensaje: "El número es positivo".

  – **Ventaja: Sí es portable.**



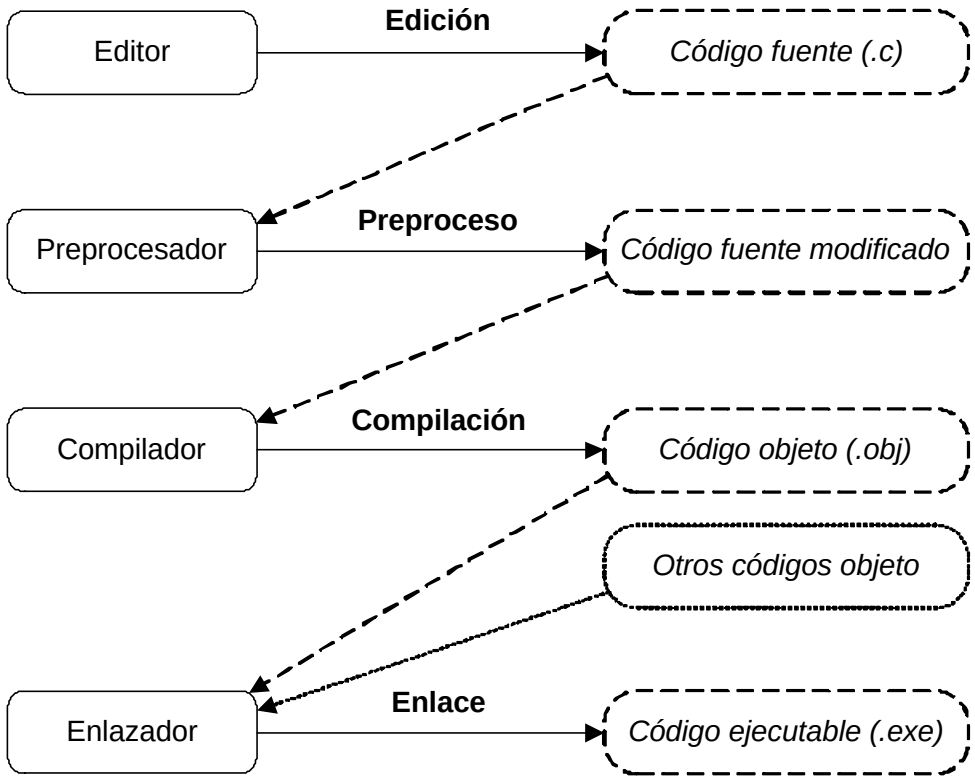
3.4 CODIFICACIÓN (5/9)

- **Relación entre las instrucciones de alto nivel, ensamblador y máquina:**



3.4 CODIFICACIÓN (6/9)

- **Fases de la puesta a punto de un programa escrito en C:**



3.4 CODIFICACIÓN (7/9)

- **Intérprete:** Se caracteriza por traducir y ejecutar, **de una en una**, las instrucciones del código fuente de un programa, pero, sin generar como salida código objeto.
  - **Proceso:**
    - Lee la primera instrucción del código fuente.
    - La traduce a código objeto y la ejecuta.
    - A continuación, hace lo mismo con la segunda instrucción.
    - Y así sucesivamente, hasta llegar a la última instrucción del programa, siempre y cuando, no se produzca ningún ***error*** que detenga el proceso.

- **Tipos de errores:**
  - de sintaxis
  - de ejecución
  - de lógica

3.4 CODIFICACIÓN (8/9)

- **Errores de sintaxis:** Se detectan en el proceso de traducción del código fuente a código binario. Impide, tanto al compilador como al intérprete, hacer la traducción.

  EJEMPLO: `prrintf( "Introduzca el primer numero (entero): " );`

- **Error de ejecución:** El ordenador no puede ejecutar alguna instrucción de forma correcta.

  EJEMPLO: `c = 5 / 0;` (**no se puede dividir entre cero**)

- **Error de lógica:** Los resultados obtenidos no son los esperados.

  EJEMPLO: `c = a * b;` **en vez de** `c = a + b;`

3.4 CODIFICACIÓN (9/9)

- **Entornos Integrados de Desarrollo (EID):**
  - Son aplicaciones informáticas que incluyen a todos los programas necesarios para realizar todas las fases de puesta a punto de un programa.
  - En el caso de C se necesita:
    - **Editor**
    - **Preprocesador**
    - **Compilador**
    - **Enlazador**
  - Suelen incluir otras herramientas software:
    - **Depurador de código**
    - **Ayuda en línea de uso del lenguaje**

3.5 PRUEBAS

- **Pruebas:**
  - Tratamiento de los datos.
  - Adaptación al resto sistema informático.
  - Interacción con otras aplicaciones.

- Hay que comprobar, exhaustivamente, la **funcionalidad**.

- Es imposible probarlo todo.

- Pueden darse situaciones inesperadas.

3.6 MANTENIMIENTO

- **Mantenimiento:**
  - **Reparación** de errores no detectados en las fases anteriores.
  - **Modificaciones** para ampliar o cambiar alguna funcionalidad.

- **Documentación:**
  - **Externa:** ERS, algoritmos, códigos fuentes, manuales de usuario,...
  - **Interna:** Comentarios en el código fuente.

EJERCICIOS RECOMENDADOS

- **Resueltos:** 1 y 2.
- **Propuestos:** 1, 2, 3 y 4.

GRACIAS POR SU ATENCIÓN

Para más información, puede visitar la web del autor:

http://www.carlospes.com