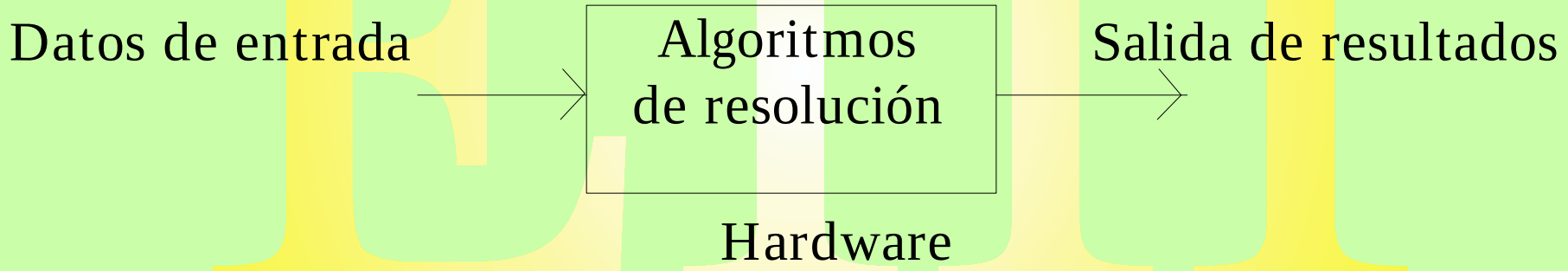
Concepto de Programa

Conjunto de ordenes que transforman los datos de entrada en una salida de resultados comprensibles.
¿Cómo lo consigue?. Usando ***Algoritmos*** que detallan los pasos a seguir para alcanzar esos resultados.



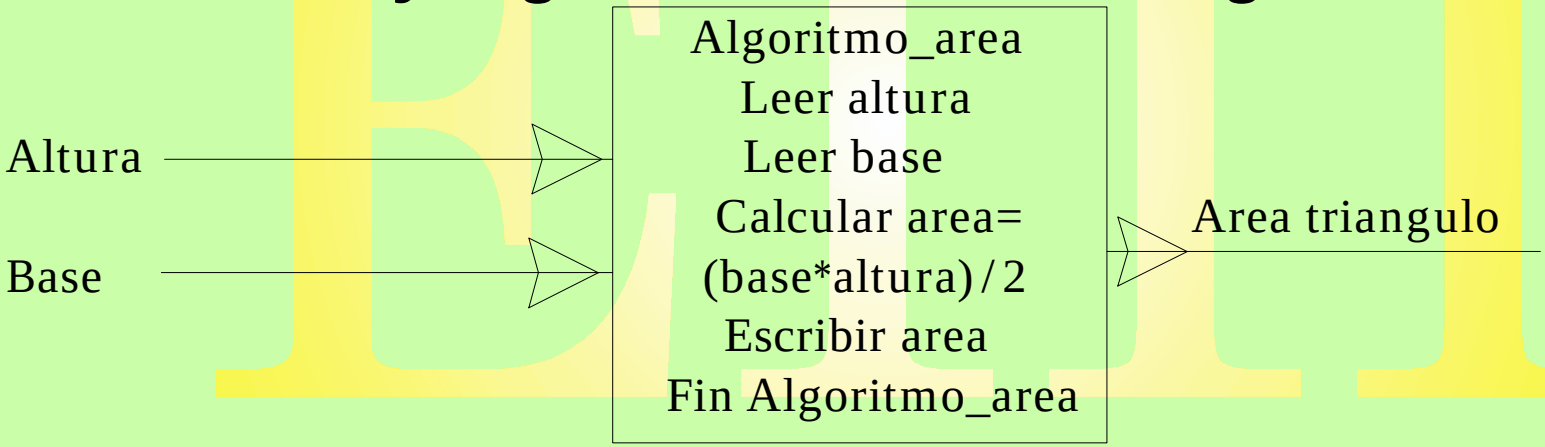
En la práctica los datos tienen una estructura más o menos compleja, dando lugar a estructuras de datos.

Algoritmos + estructuras de datos = Programas. (Wirth)

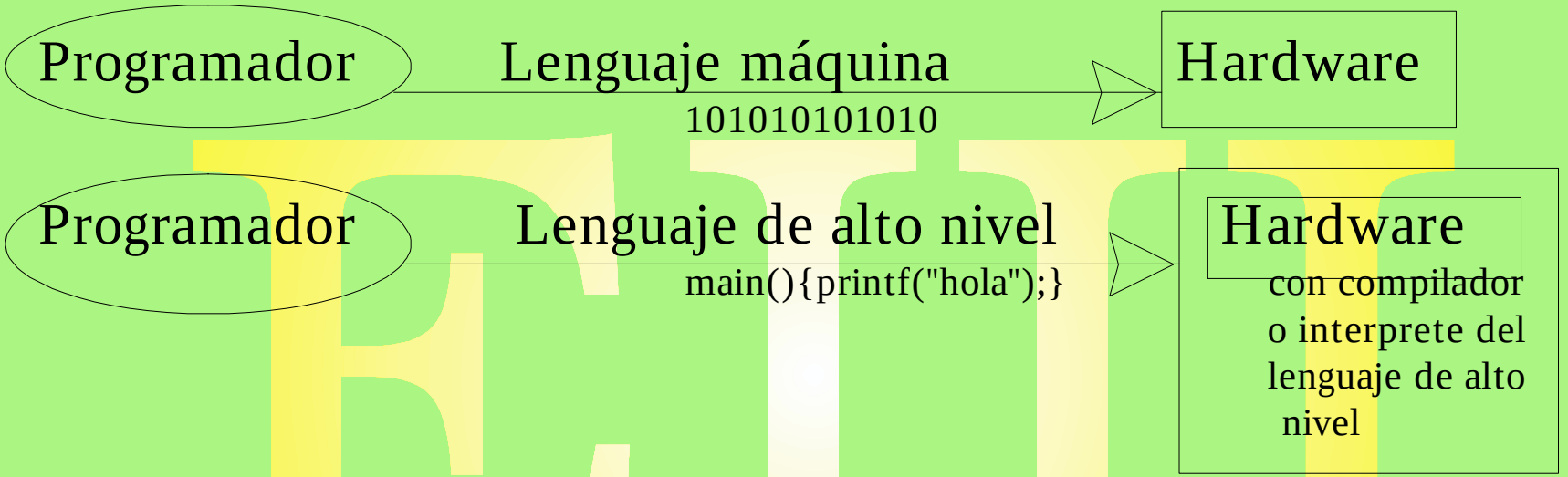
Características de los algoritmos.

- Número finito de pasos.
- Cada paso debe de estar perfectamente definido.
- Cada paso debe de poder ser ejecutado en un tiempo finito.
- Debe de existir un conjunto de datos iniciales.
- Debe de existir un conjunto de datos de salida.

Ej. Algoritmo area de un triangulo



Máquinas y programas.



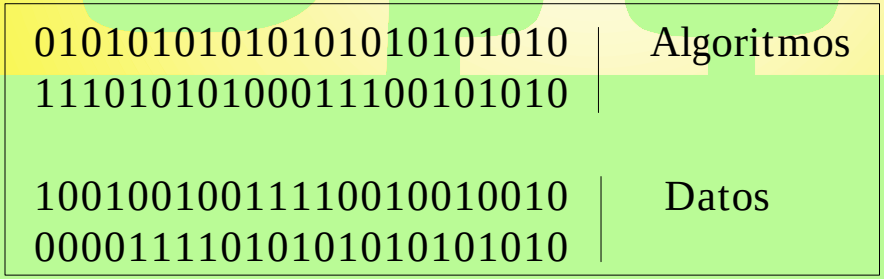
Ventajas:

1) Comodidad.
2) Rapidez.
3) Seguridad.

Tipos de lenguajes de programación.

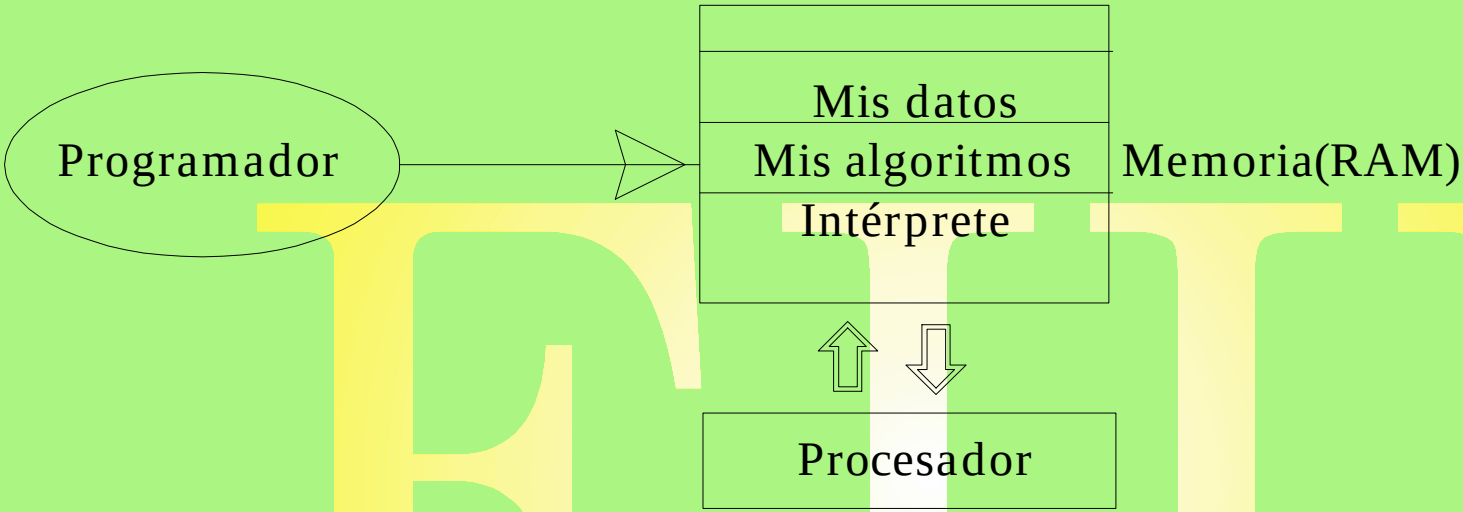
¿Cómo introducir el programa en la máquina?

a) En lenguaje máquina, es decir en ceros y unos.
Inconvenientes: laborioso,lento,poco fiable, antieconómico.
b) En lenguaje assembler (ensamblador o de bajo nivel) que consiste en microinstrucciones simbólicas que equivalen a un grupo de ceros y unos. (sum = 10001100).
Inconvenientes: cada microprocesador tiene su propio lenguaje assembler, complejo, sigue siendo lento.
c) En lenguaje de alto nivel (Fortran, C) donde una instrucción con una sintaxis simple, equivale a varias microinstrucciones. Por tanto es potente, rápido, independiente del micro, **especializado**, cómodo y fiable. Ej: A = B + C*Log(D).
Se puede utilizar: Intérpretes o Compiladores.



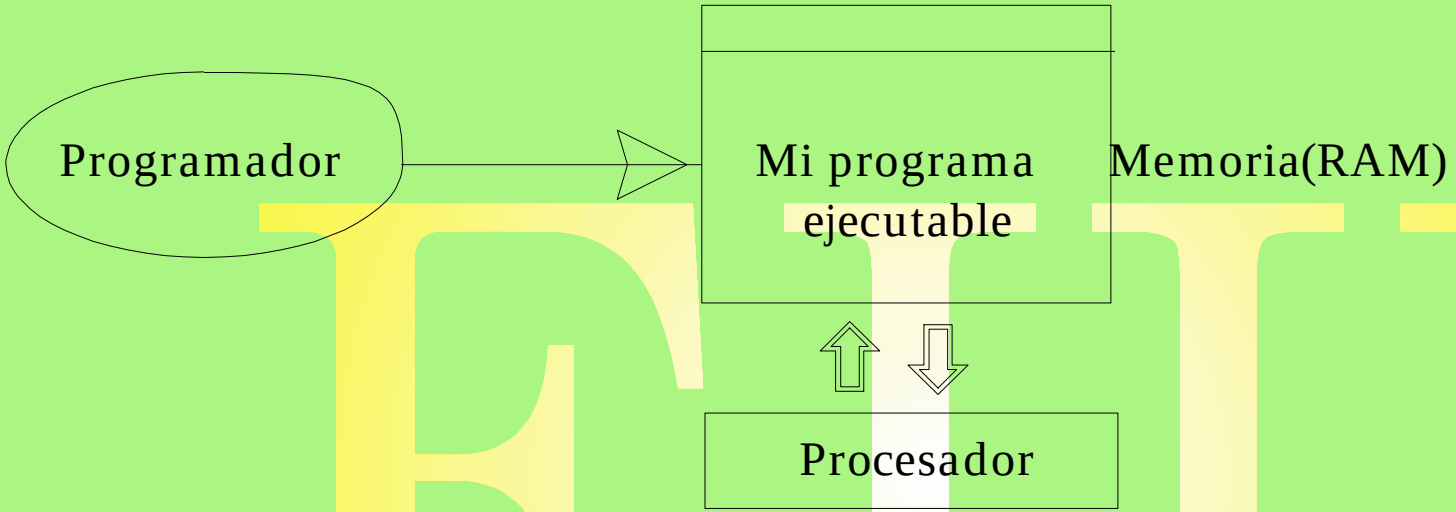
Hardware (máquina)

Intérprete.



Cada vez que necesite utilizar (ejecutar) el programa, el intérprete "traducirá simultaneamente" instrucción a instrucción a lenguaje máquina, sin generar el conjunto completo del programa traducido (**programa ejecutable**), por tanto el proceso será lento pero cómodo.

Compilador..



Traduce todo el conjunto de instrucciones de una sola vez generando un programa en lenguaje máquina denominado **programa ejecutable.**
El proceso tiene dos pasos. En primer lugar y a partir del programa original escrito en un lenguaje de alto nivel (**programa fuente**), se genera un programa en lenguaje assembler llamado **programa objeto**.Posteriormente se enlaza este con un conjunto de información externa denominado **Librerias** que resuelven determinados tipos de operaciones predeterminadas, generando el **programa ejecutable** en lenguaje máquina.

Comparación Intérprete-Compilador.

Intérprete.

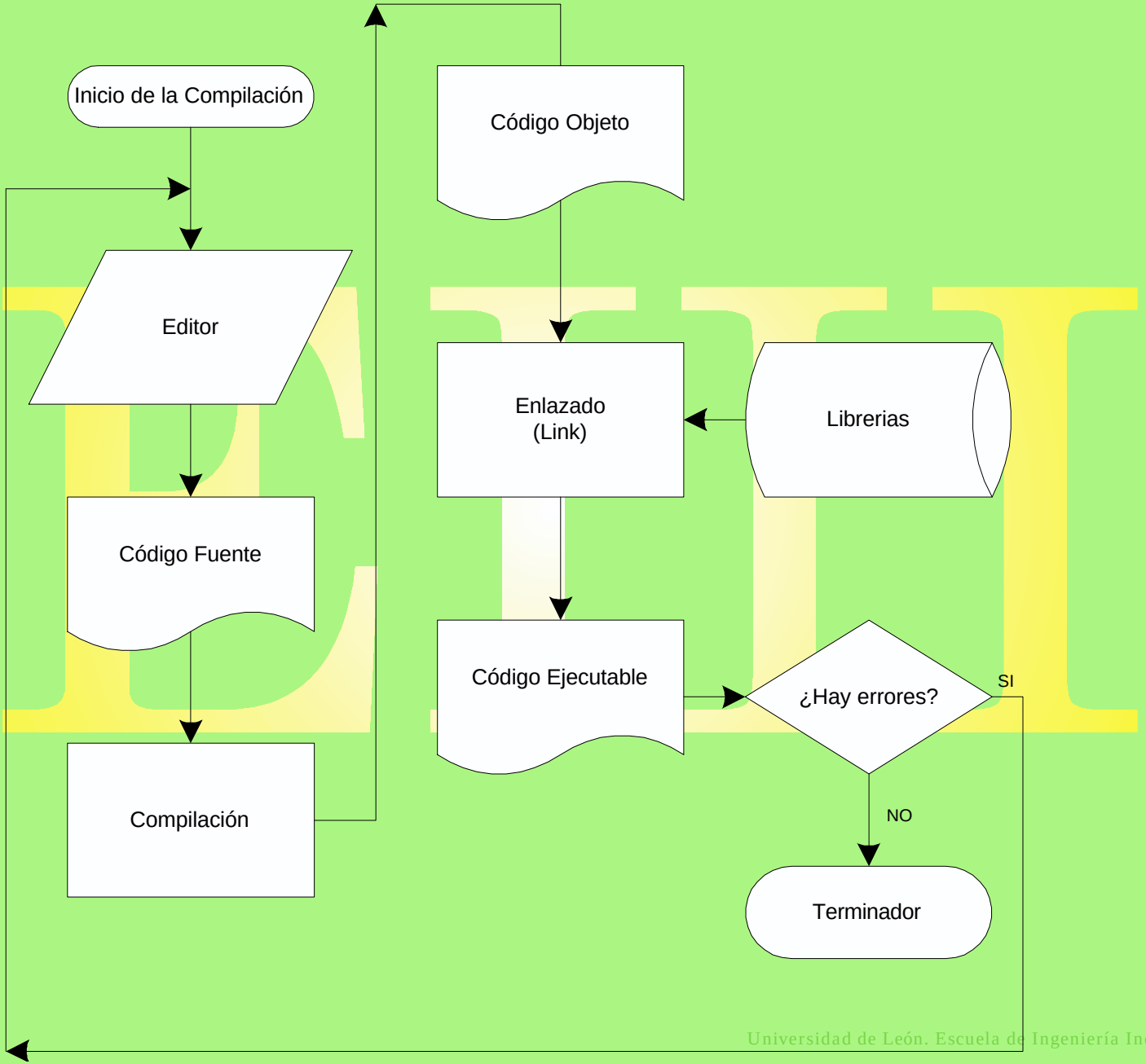
Ventajas: comodidad en el desarrollo.

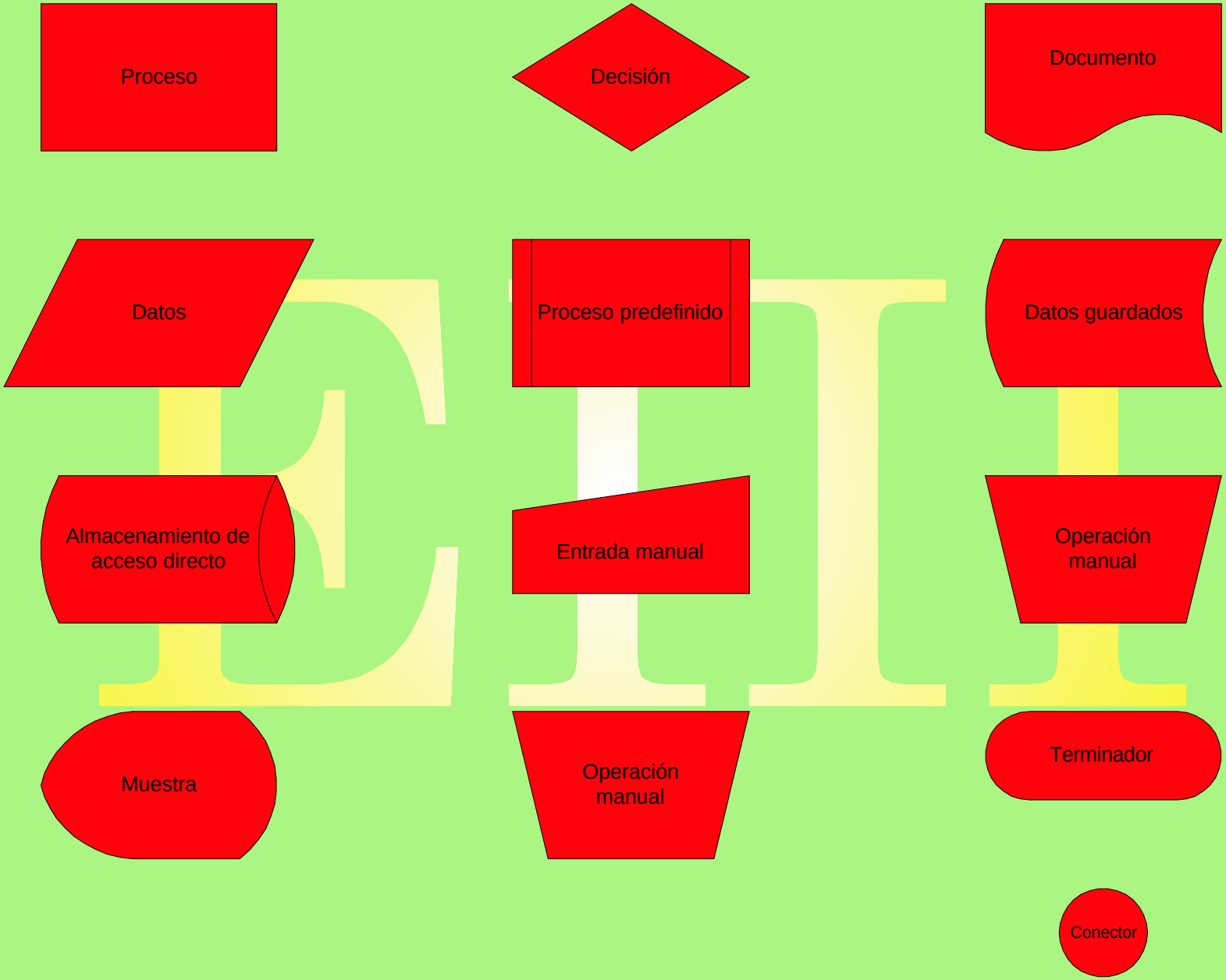
Inconvenientes: no optimiza el tiempo de ejecución ni genera un programa ejecutable..

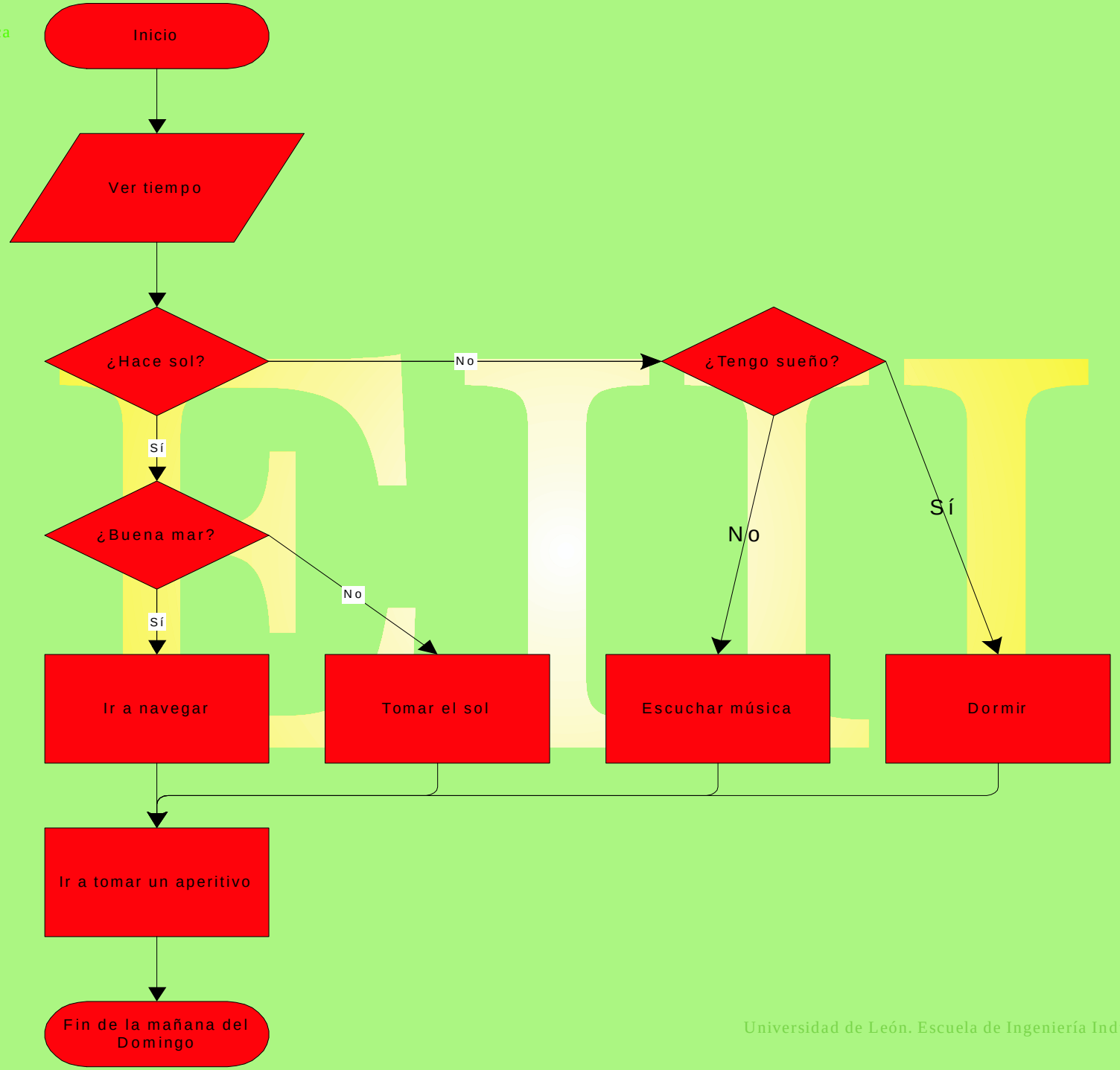
Compilador.

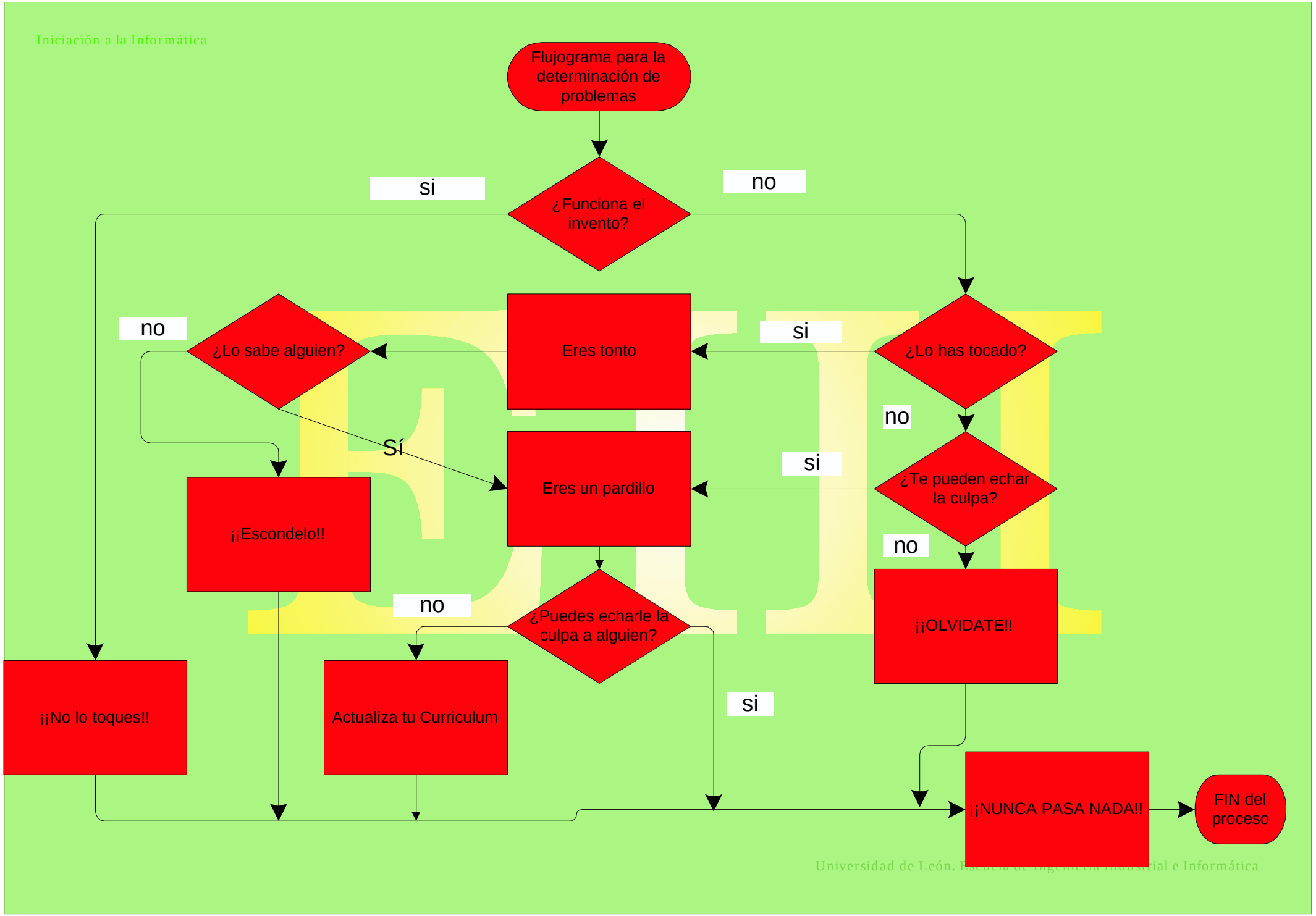
Ventajas: optimización en la ejecución. Potencia. Portabilidad.

Inconvenientes: desarrollo engorroso salvo en los nuevos entornos gráficos integrados.


Inicio de la Compilación
Editor
Código Fuente
Compilación
Código Objeto
Enlazado
(Link)
Librerias
Código Ejecutable
¿Hay errores?
SI
NO
Terminador


Proceso
Decisión
Documento
Datos
Proceso predefinido
Datos guardados
Almacenamiento de acceso directo
Entrada manual
Operación manual
Muestra
Operación manual
Terminador
Conector


Inicio
Ver tiempo
¿Hace sol?
No
¿Tengo sueño?
Sí
¿Buena mar?
No
Sí
No
Sí
Ir a navegar
Tomar el sol
Escuchar música
Dormir
Ir a tomar un aperitivo
Fin de la mañana del Domingo


Flujograma para la determinación de problemas
¿Funciona el invento?
si
no
¿Lo sabe alguien?
no
Sí
Eres tonto
¿Lo has tocado?
si
no
¡¡Escondelo!!
Eres un pardillo
¿Te pueden echar la culpa?
si
no
¿Puedes echarle la culpa a alguien?
no
si
¡¡OLVIDATE!!
¡¡No lo toques!!
Actualiza tu Curriculum
¡¡NUNCA PASA NADA!!
FIN del proceso

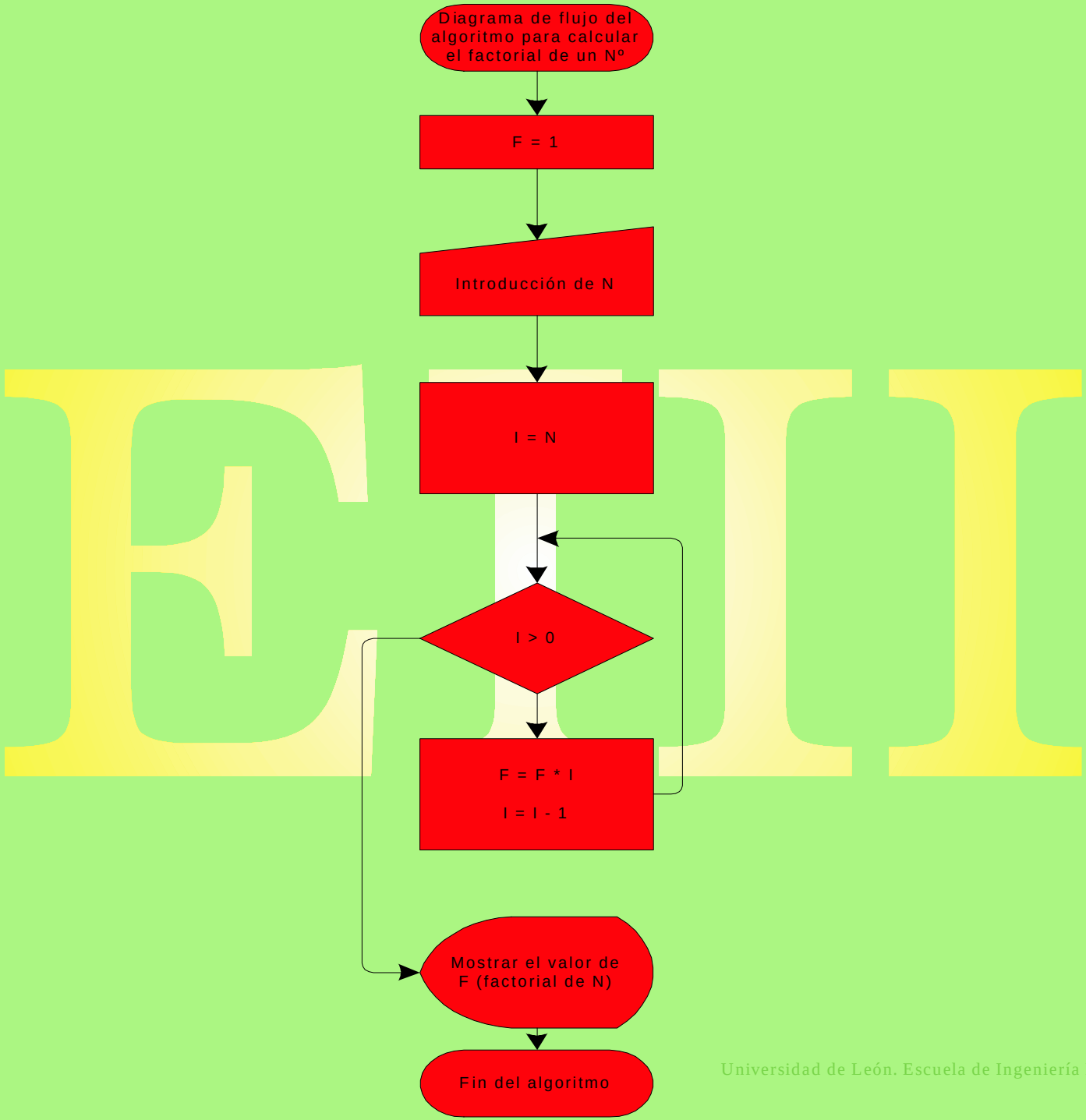


Diagrama de flujo del algoritmo para calcular el factorial de un Nº
F = 1
Introducción de N
I = N
I > 0
F = F * I
I = I - 1
Mostrar el valor de F (factorial de N)
Fin del algoritmo

Elementos de un programa.

A cada una de las ordenes de un programa que engloban constantes, variables, operadores y expresiones se las denominan **sentencias o instrucciones.**

Sentencias de entrada/salida: permiten establecer la comunicación entre los perifericos y la memoria principal.

Sentencias de asignación: permiten asignar valores a las variables.

Sentencias de control: permiten romper la secuencia de órdenes de un programa. Utilizan bifurcaciones condicionales basadas en una pregunta que sólo admite dos respuestas.

Sentencias de declaración de tipos de variables: permiten reservar zonas de memoria para alojar datos.

Procedimientos: de esta forma se define por el programador un conjunto de sentencias agrupadas en una unidad.

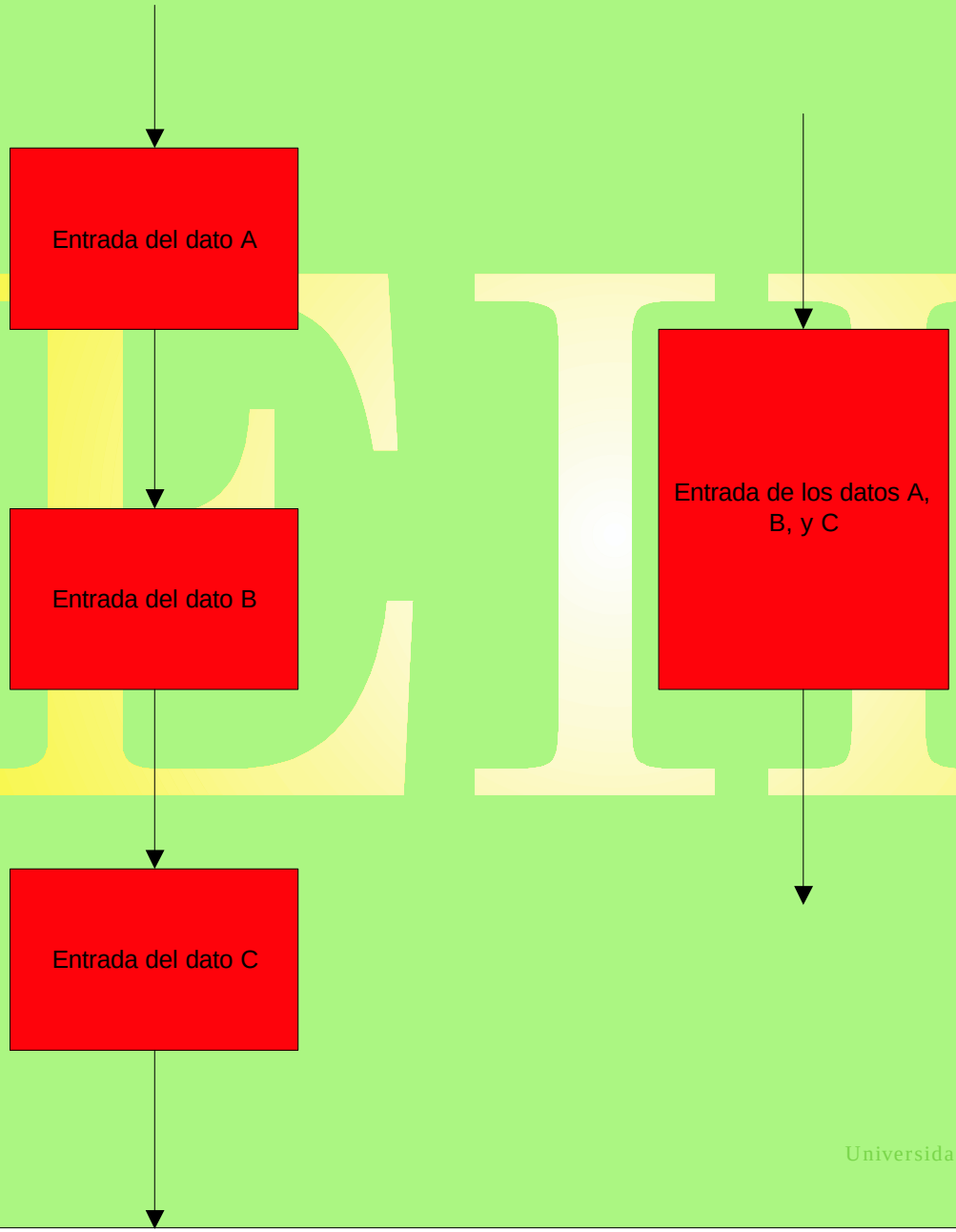
Tipos de sentencias de control.

Secuencia: consiste en disponer dos o más sentencias una a continuación de otra.

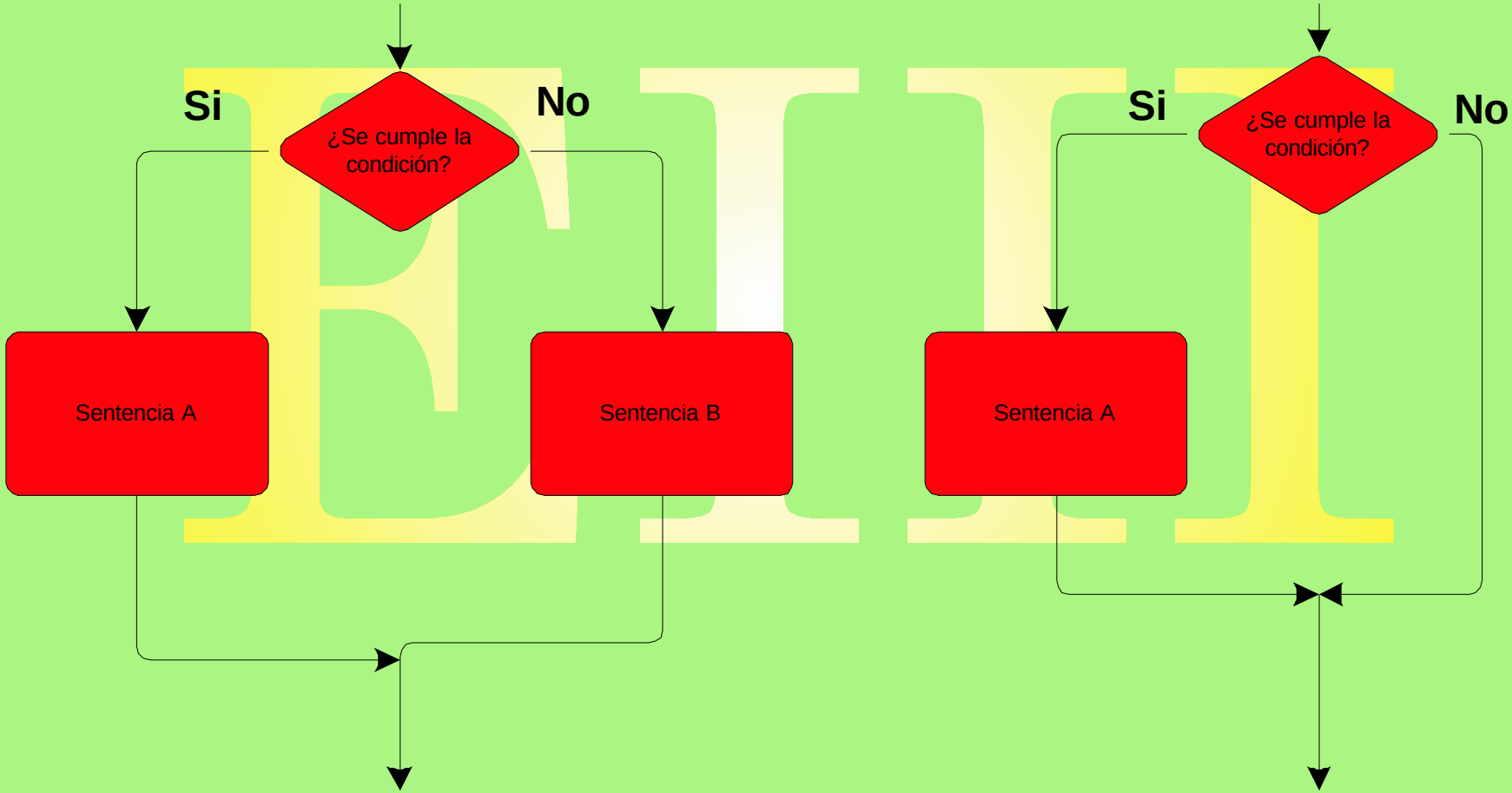
Selección o alternativa: puede ejecutar una u otra sentencia según al valor que tome una condición.

Iteración: consiste en la repetición de una o varias sentencias un determinado número de veces.

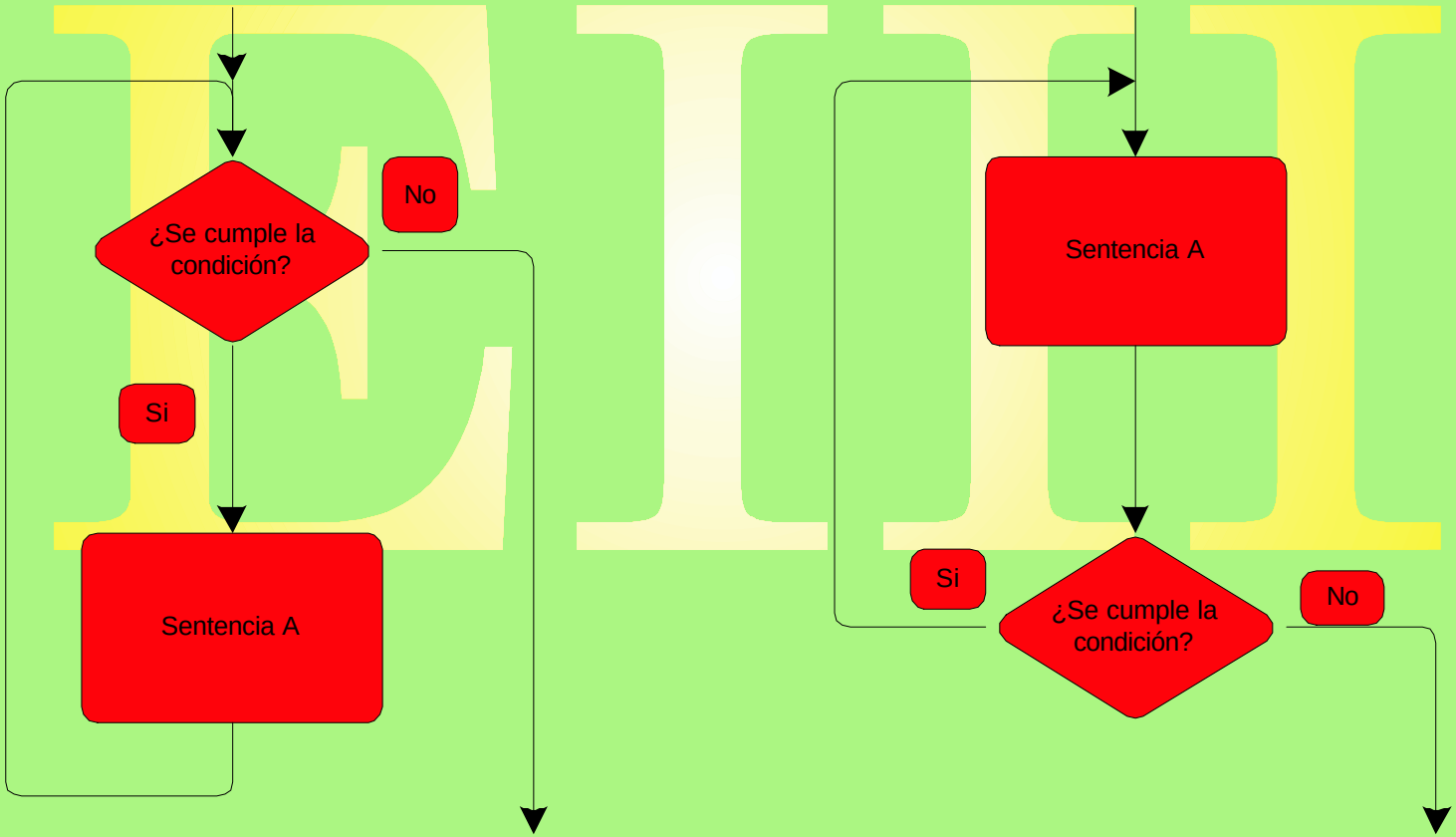
Sentencias en secuencia.



Sentencias selectivas o alternativas.



Sentencias iterativas.



Evolución histórica de los métodos de programación.

Con ayuda de los ordinogramas podemos describir los algoritmos y procedimientos de complejidad media y baja, pero en el caso de que sea alta el **aumento del tamaño físico y la maraña de relaciones** en el diagrama de flujo hará que sea difícil de manipular. Para evitar este inconveniente se utilizan dos métodos de programación complementarios:

- Programación modular que es un método de diseño que tiende a dividir el problema total en partes perfectamente diferenciadas que pueden ser analizadas, programadas y probadas por separado.

- Programación estructurada que se basa en limitar el conjunto de estructuras disponibles (privilegiadas) para solucionar un problema, tendiendo a estructurar el programa evitando el uso de saltos incondicionales (GOTO).

Definición de módulo.

Características:

a) Está formado por una o varias instrucciones (sentencias) que están físicamente juntas.

b) Se puede hacer referencia a él mediante un nombre.

c) Se le puede llamar desde diferentes puntos de un programa.

Puede ser: un programa, una función o una subrutina.

Fases del diseño modular:

1) Diseño descendente (Top-Down) de la red de módulos.
2) Análisis de cada relación (parámetros de entrada y salida).
3) Diseño de cada módulo (puede hacerse de forma independiente.)
4) Montaje ascendente (Bottom- Up).

Definición de módulo (II).

Criterios:

a) Un módulo debe corresponder a una función lógica perfectamente diferenciada.

b) Es necesario que el módulo sea pequeño para que sea claro (no más de una página de extensión).

c) No conviene utilizar demasiados niveles de módulos (hay que buscar un equilibrio entre el punto anterior y este).

d) La salida debe ser exclusivamente función de la entrada (caja negra).

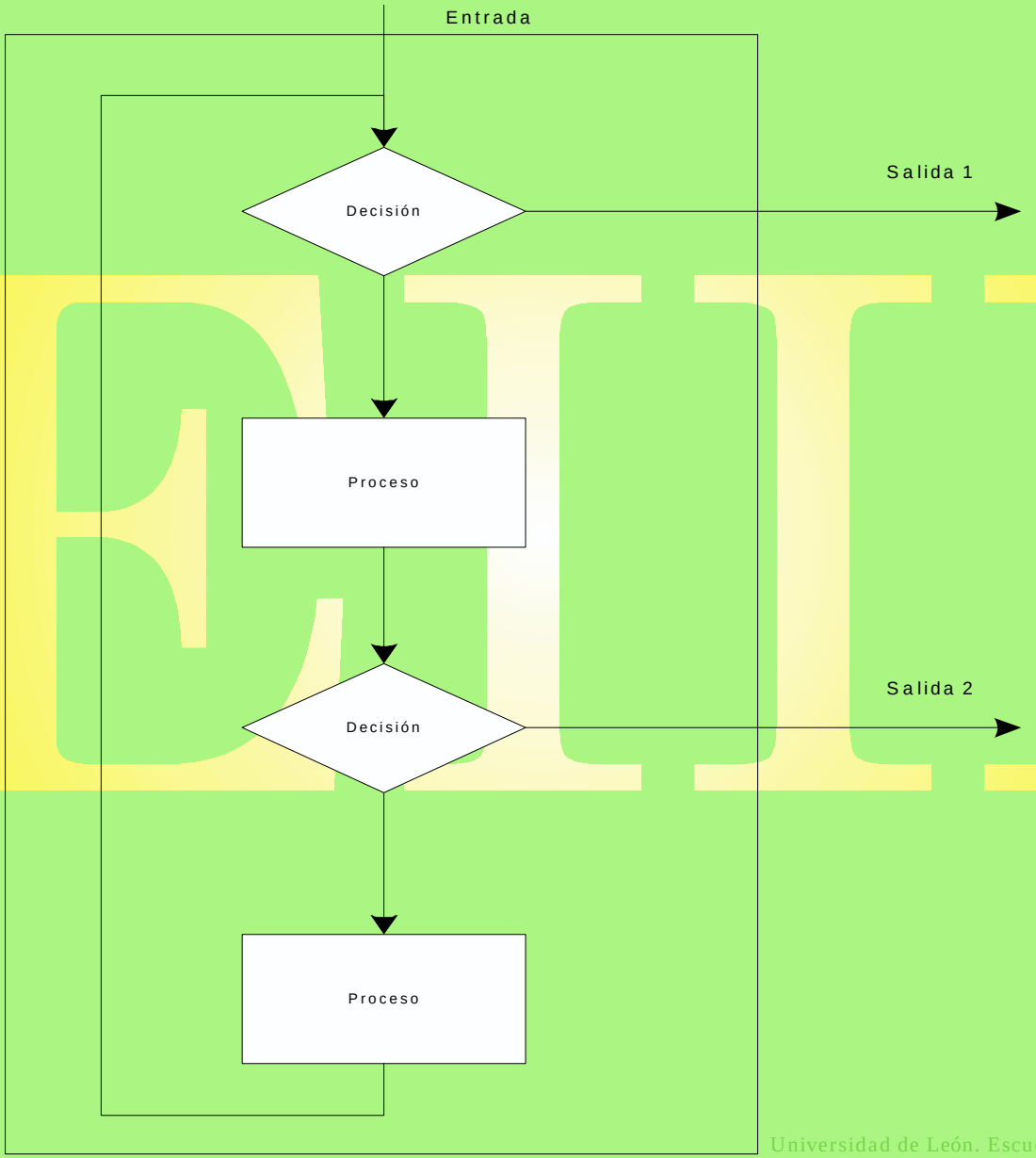
e) Punto de entrada y punto de salida únicos.

Programación estructurada.

"**Cualquier programa o algoritmo, si es resoluble, se puede siempre resolver utilizando las tres estructuras básicas o privilegiadas: secuencial, selectiva e iterativa**" o bien
"**Todo algoritmo debe resolverse utilizando las estructuras privilegiadas**".

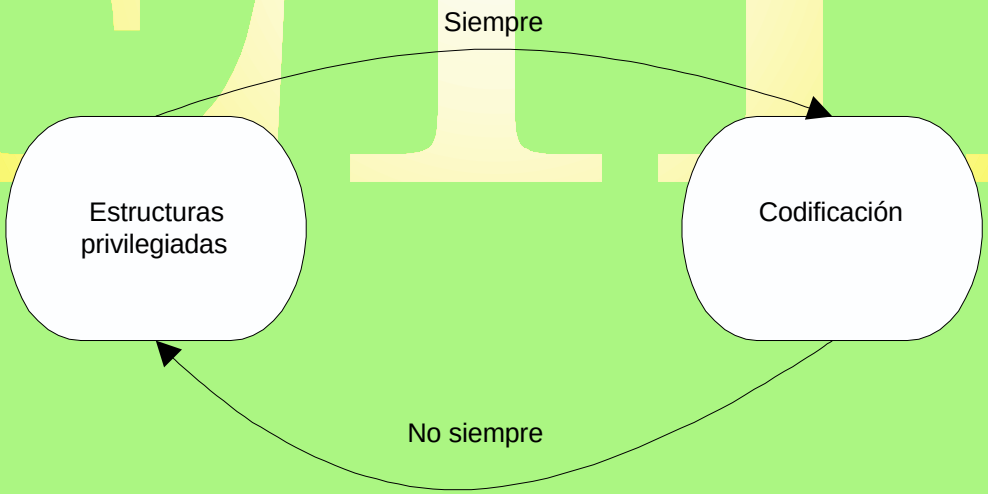
La importancia de la programación estructurada no está en lo que recomienda si no en lo que prohibe: los saltos incondicionales (Goto), que imposibilitan un seguimiento cómodo del flujo del programa.

Ejemplo de estructura básica de control no admitida



Fases de una programación estructurada.

1º) Establecer una arquitectura modular (nº de módulos y sus interconexiones) en diseño de arriba - abajo (Top - Down).
2º) Especificar la transferencia de datos entre los módulos de manera que sean mínimos.
3º) Especificar lo que hace cada módulo mediante diagrama de flujo, utilizando sólo estructuras privilegiadas.
4º) Si el grado de complejidad de un módulo es demasiado alto, fraccionarlo o descomponerlo en varios y volver al punto 2º.



Pseudocódigo.

Estructura secuencial:

Inicio_secuencia
    {S1};{S2};...{Sn}; siendo {Si} una subacción.
Fin_secuencia

Estructura alternativa simple:

SI {condición}
    ENTONCES {SQ}
Fin_SI

Estructura alternativa doble:

SI {condición}
    ENTONCES {SQ1} SI_NO {SQ2}
Fin_SI

Estructuras iterativas:

a) MIENTRAS {condición}
        HACER {SQ}
   Fin_MIENTRAS

b) REPETIR {SQ}
   HASTA_QUE {condición}
   Fin_REPETIR

```
Inicio_secuencia_principal
      SI {P1}          ENTONCES  ACCIÓN_A
      SI_NO   ACCIÓN_B
      Fin_SI
Fin_secuencia_principal
```

Desglose acción_A

```
Inicio_secuencia_A
      {EJECUTAR ACCIÓN_C};
      {EJECUTAR ACCIÓN_D};
Fin_secuencia_A
```

Desglose acción_B

```
Inicio_secuencia_B
      {EJECUTAR ACCIÓN_G};
      {EJECUTAR ACCIÓN_H};
Fin_secuencia_B
```

Desglose acción_C

```
      MIENTRAS {P2}   HACER {SQ1}
      Fin_MIENTRAS
```

Desglose acción_D

```
      SI {P3}          ENTONCES  {SQ2}
      SI_NO  {SQ3}
      Fin_SI
```

Desglose acción_G

```
      {SQ4}
```

Desglose acción_H

```
      REPETIR  {SQ5}   HASTA_QUE        {P4}
```

Inicio_secuencia_principal

SI{P1} ENTONCES

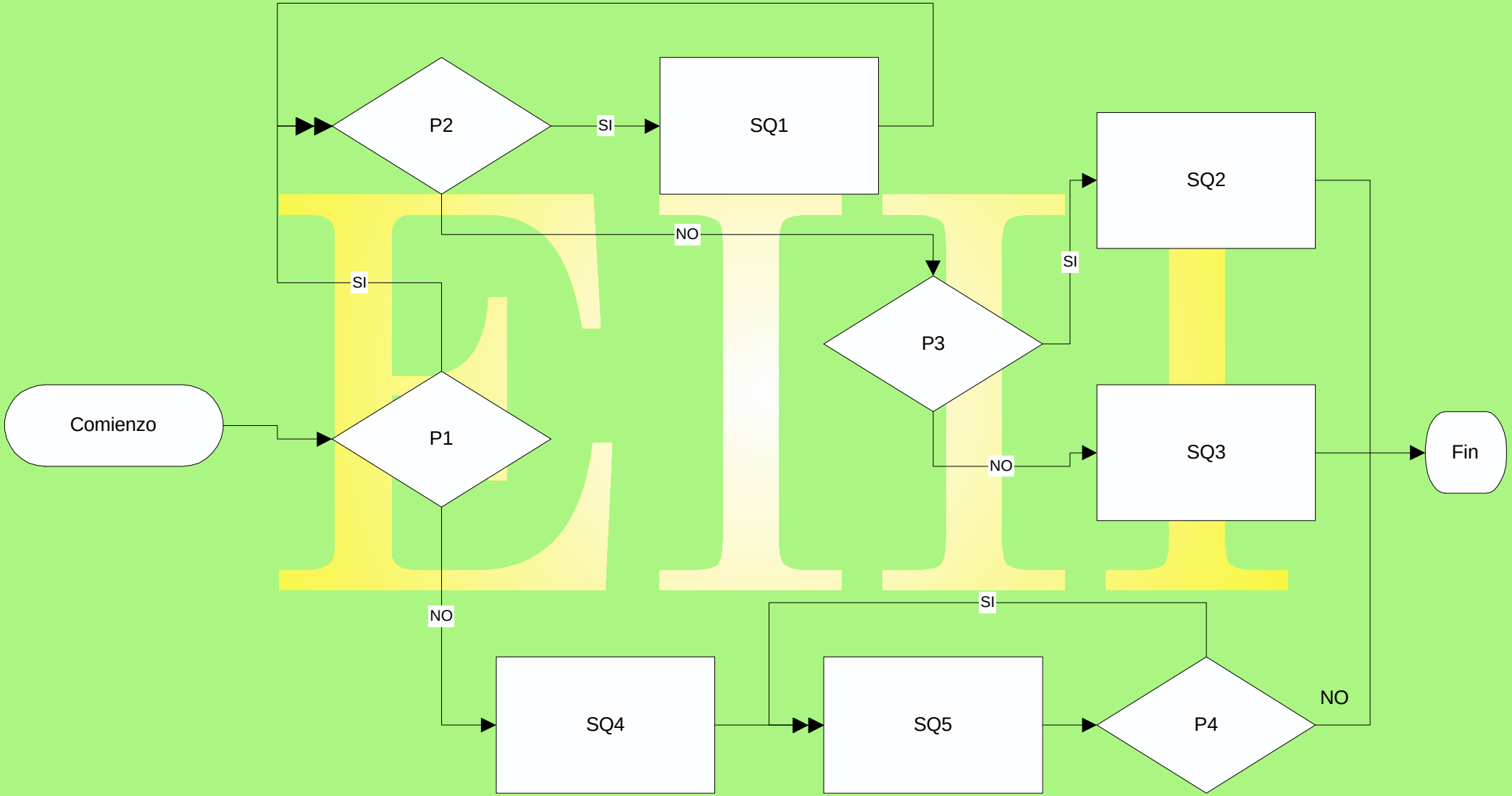
MIENTRAS {P2} HACER {SQ1}
Fin_MIENTRAS

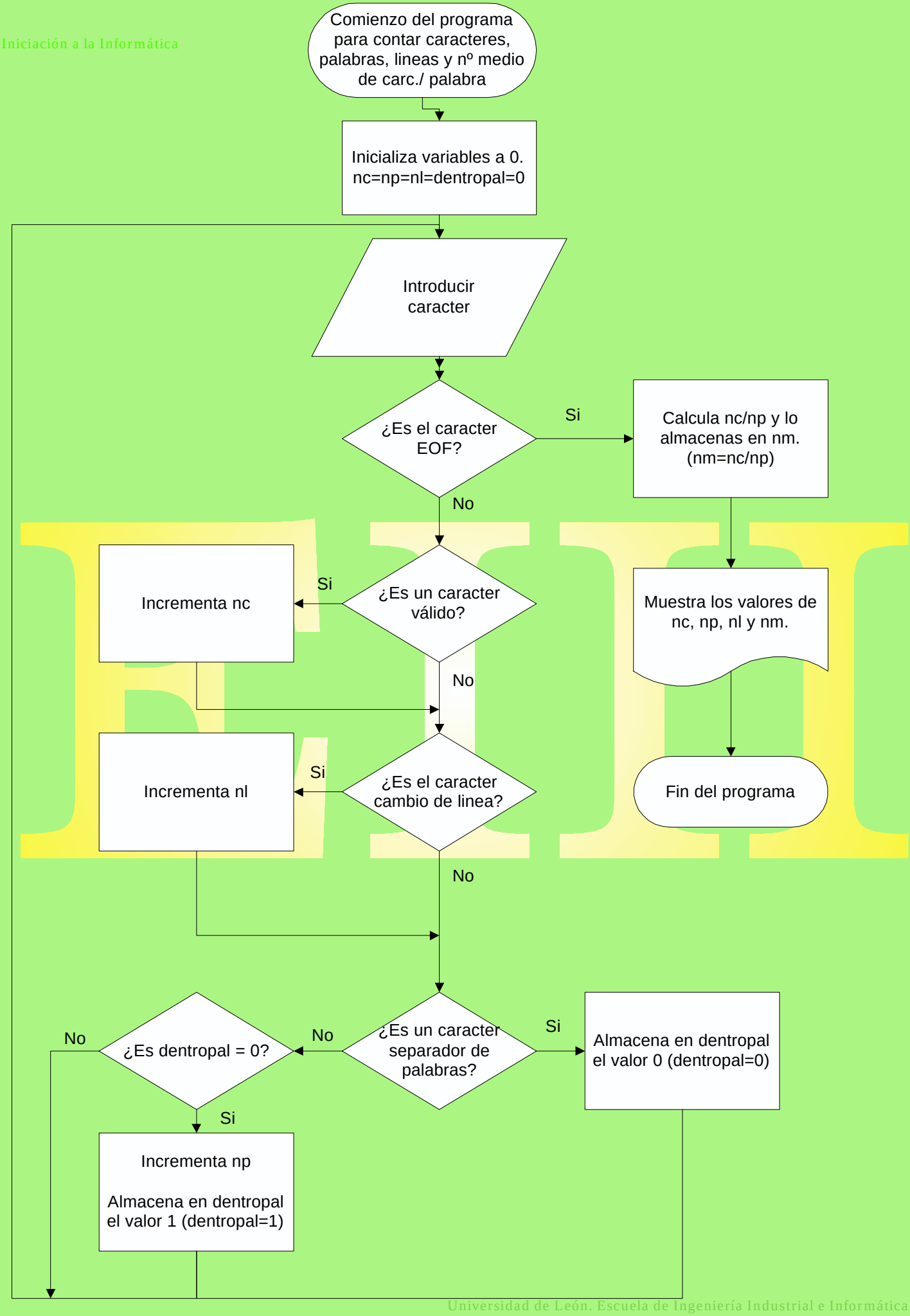
SI {P3} ENTONCES {SQ2}
SI_NO {SQ3}
Fin_SI

SI_NO {SQ4}
REPETIR {SQ5} HASTA_QUE {P4}

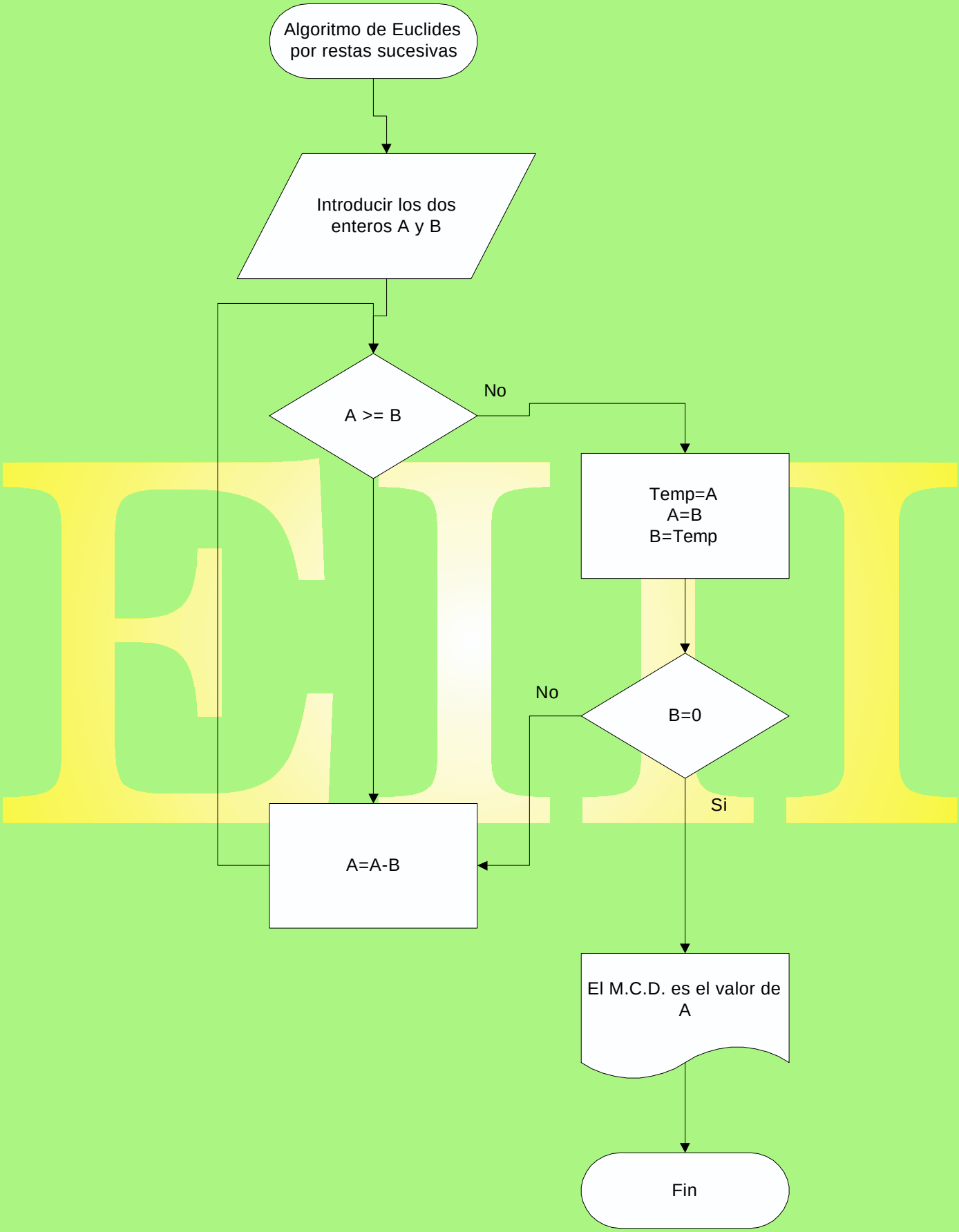
Fin_secuencia_principal

Ejemplo nº 1.

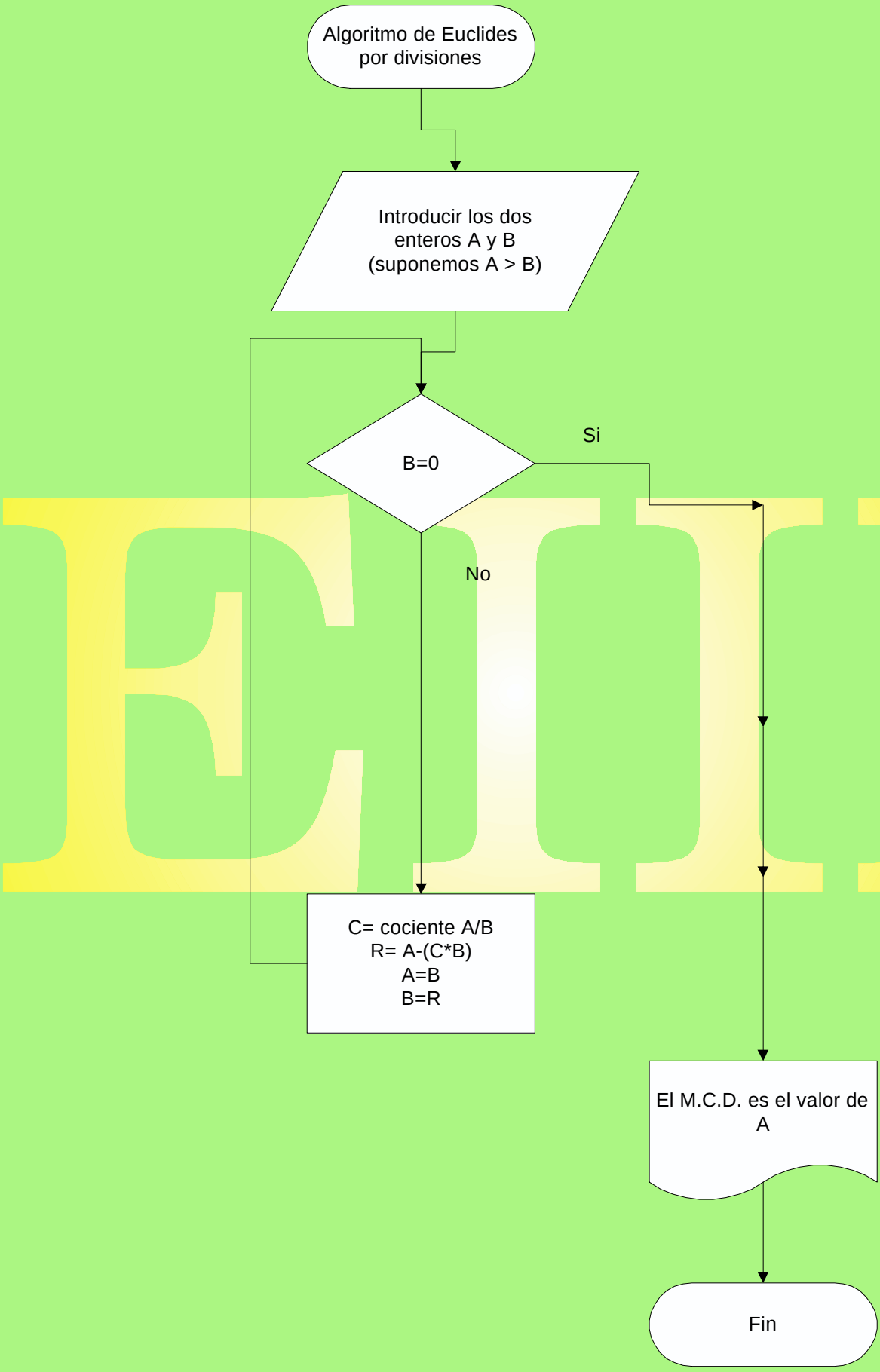

Comienzo
P1
P2
SI
SQ1
NO
SI
P3
SI
SQ2
NO
SQ3
Fin
NO
SQ4
SI
SQ5
P4
NO


Comienzo del programa para contar caracteres, palabras, lineas y nº medio de carc./ palabra
Inicializa variables a 0. nc=np=nl=dentropal=0
Introducir caracter
¿Es el caracter EOF?
Si
Calcula nc/np y lo almacenas en nm. (nm=nc/np)
No
¿Es un caracter válido?
Si
Incrementa nc
No
¿Es el caracter cambio de linea?
Si
Incrementa nl
No
Muestra los valores de nc, np, nl y nm.
Fin del programa
¿Es un caracter separador de palabras?
No
Si
Almacena en dentropal el valor 0 (dentropal=0)
¿Es dentropal = 0?
No
Si
Incrementa np
Almacena en dentropal el valor 1 (dentropal=1)

Primera solución al algoritmo de Euclides.



Segunda solución al algoritmo de Euclides.

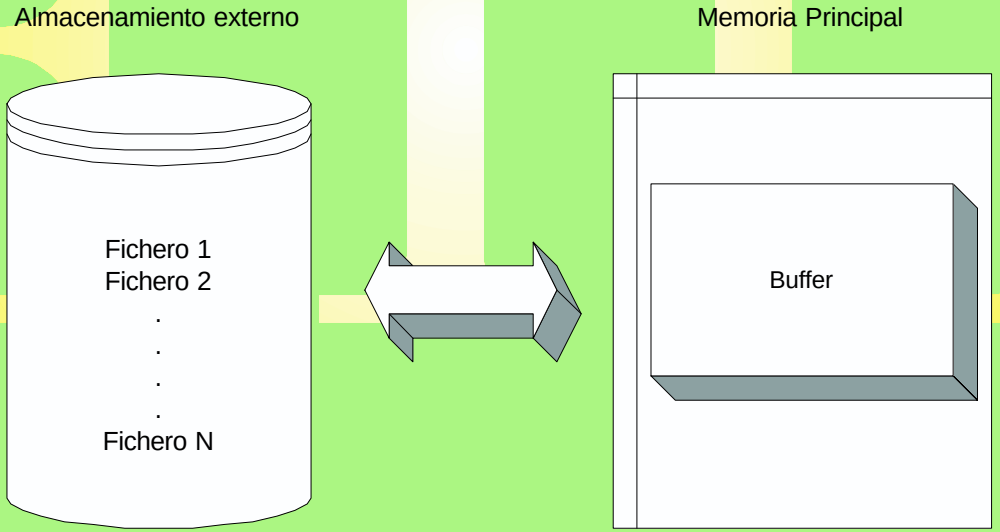


Estructuras externas: Ficheros.

Para poder trabajar con información depositada en las memorias externas del ordenador necesitamos la presencia en la memoria principal de un espacio denominado **BUFFER** (Memoria intermedia) que realice la adaptación.

Las operaciones básicas con ficheros son:

a) Apertura.
b) Lectura.
c) Escritura.
d) Cierre.



Apertura de un fichero.

Necesitamos conocer la posición y el nombre del fichero para poder abrirlo y también una dirección de memoria que nos indique a partir de donde se va a ubicar esa información.

De esta forma se establece una relación entre los datos almacenados en el exterior y una dirección interna de la memoria principal.

Además hay que indicar el **modo** de apertura:

a) Sólo lectura

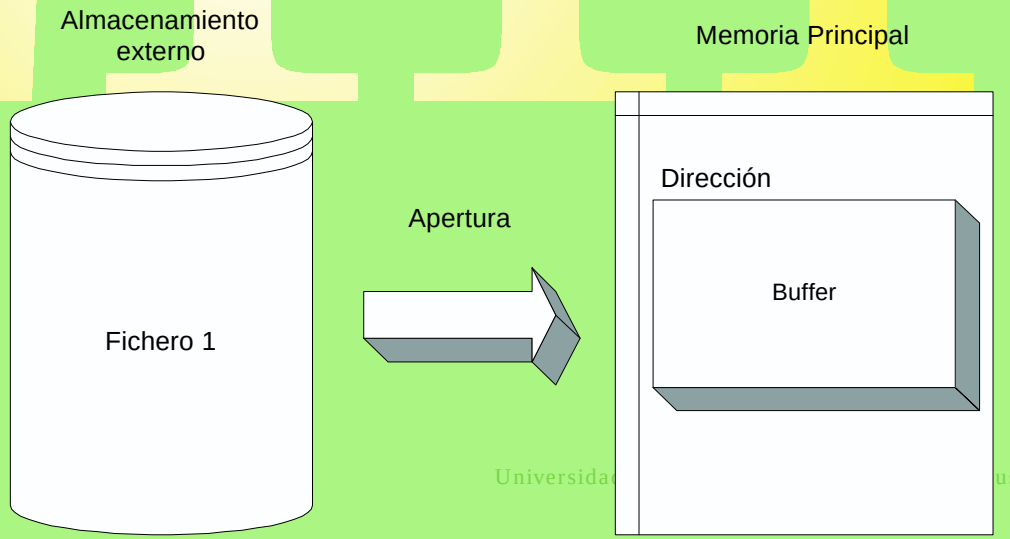
b) Escritura (que crea o sobreescribe)

c) Añadir (que crea o añade)

Se pueden mezclar los modos.

Puede ocurrir que al intentar abrir un fichero se produzca un error, entonces la dirección de memoria será NULL.

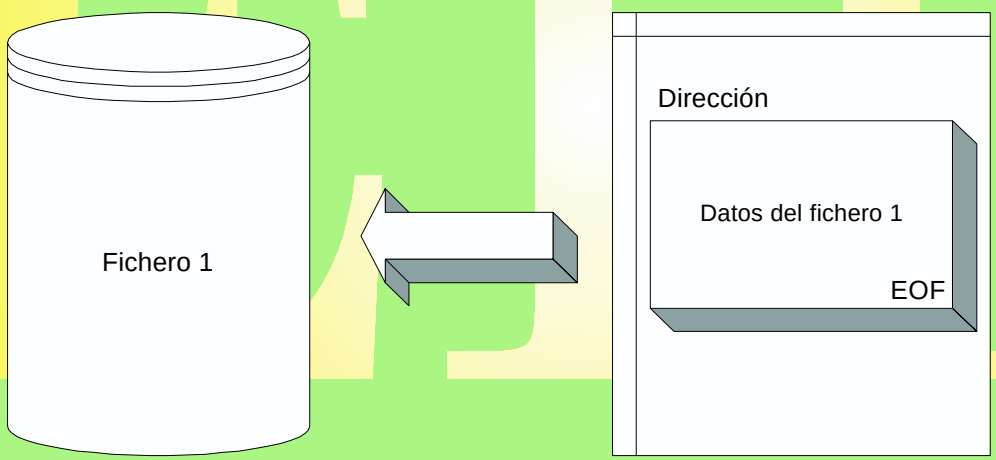
La información es depositada en parte o totalmente en la Memoria Principal para su posterior manipulación.




EIII

Escritura en un fichero.

La información se deposita en parte o totalmente en la Memoria Principal , añadiendo la marca EOF al final.



Cierre de un fichero.

Equivale a sobreescribir, en la memoria externa, toda la información que está en el buffer y liberar la dirección de memoria.
Siempre que se abra un fichero se está obligado a cerrarlo si no queremos perder datos. Un error en esta operación puede suponer pérdida de información.