# Fundamentos de Programación I

# 1. Resolución de problemas con computadoras

Luis Rodríguez Baena (luis.rodriguez@upsam.es)

Universidad Pontificia de Salamanca
Escuela Superior de Ingeniería y Arquitectura

# Introducción al procesamiento de la información

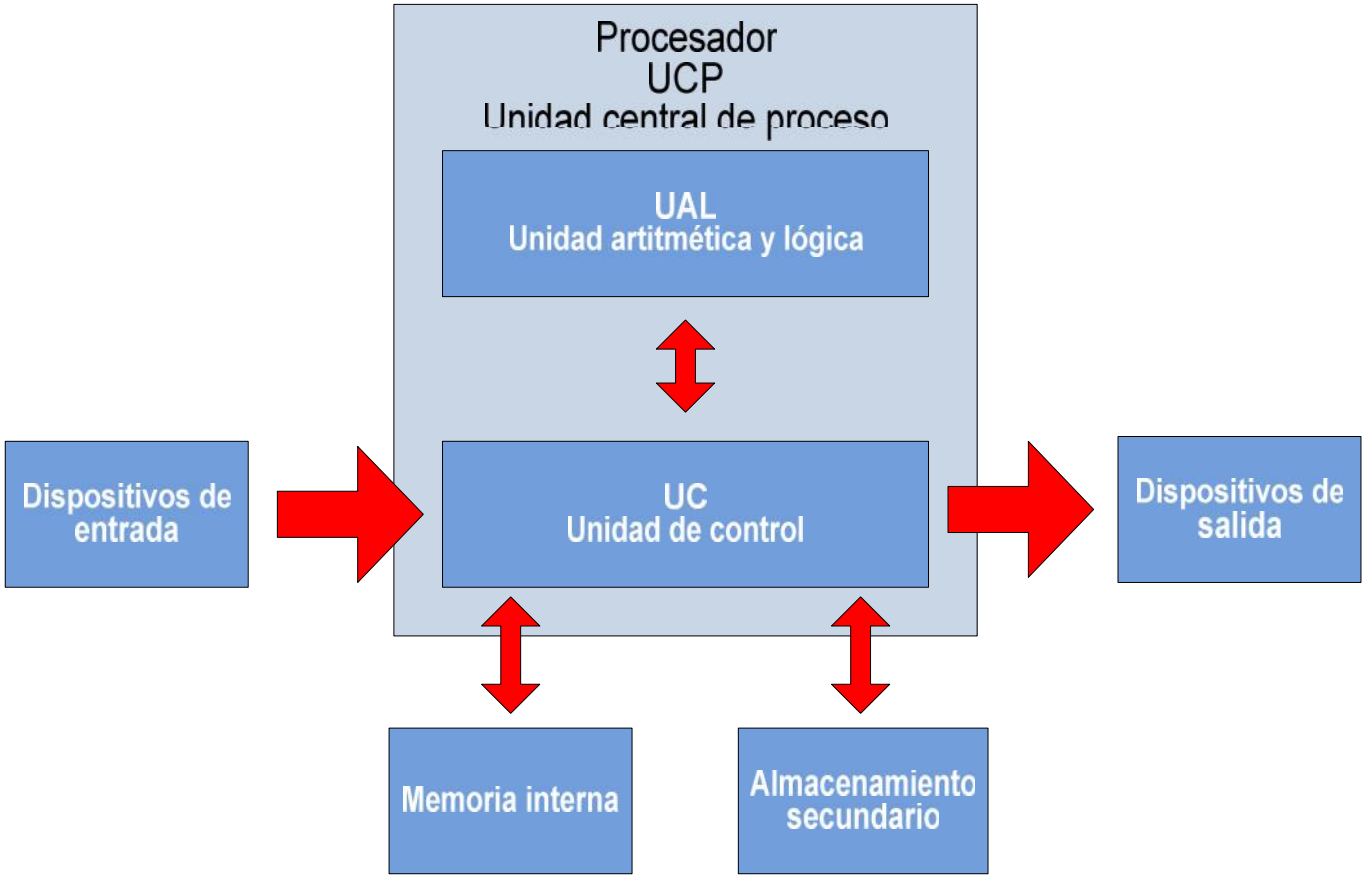
- Computadora = Sistema de procesamiento de la información.
  - Procesar datos de entrada para obtener una salida.


- Un sistema informático estará formado por:
  - Hardware
    - Parte física del sistema
      - "Conjunto de los componentes que integran la parte material de un ordenador"
  - Software:
    - Suministra a la parte física las operaciones que debe realizar el sistema.
      - "Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora"

# Hardware

- Para poder realizar las instrucciones suministradas por el software, una computadora necesita cumplir las siguientes funciones:
  - Proporcionar datos de entrada.
  - Devolver datos de salida.
  - Almacenar los datos suministrados o generados por el sistema.
  - Ejecutar operaciones aritméticas y lógicas sobre los datos.
  - Controlar y dirigir las operaciones que se realizan en él.

# Hardware (II)



# Hardware (III)

- El procesador (CPU).
  - Ejecuta y controla las operaciones que se realizan en el ordenador.
    - ✓ Solo puede realizar acciones muy simples (sumar, comparar, mover datos), pero a gran velocidad.
  - Gestiona la entrada de información al sistema desde los dispositivos de entrada, o el almacenamiento secundario, almacena la información en la memoria principal, la procesa y la envía al almacenamiento secundario o a los dispositivos de salida.
  - Dos partes principales:
    - ✓ Unidad de control.
      - ○ Coordina las operaciones.
    - ✓ Unidad aritmética y lógica.
      - ○ Realiza las operaciones de procesamiento básicas.

# Hardware (IV)

- Dispositivos de entrada y salida.
  - ¿De donde viene la información con la que trabaja un programa?
    - ✓ Puede partir de otros programas u otros sistemas informáticos.
    - ✓ Puede ser proporcionada por el usuario del sistema mediante los dispositivos de entrada.
  - Independientemente de la forma de proporcionar los datos (por teclado, mediante un ratón, por un escáner, por voz, por gestos…) el mecanismo es similar.
    - ✓ Cuando la CPU solicita información externa (lectura) se transforman los estímulos de los dispositivos en un formato binario que pueda ser interpretado por la CPU y almacenado en la memoria central.
  - Los datos procesados puede que necesiten ser interpretados por un humano.
    - ✓ El proceso de salida se realiza de forma similar, pero inversa, al de entrada:
      - ○ La información, en formato binario, se transforma a estímulos sensoriales (imagen impresa o en la pantalla, sonido, movimiento, etc.).

# Hardware (V)

- Memoria central.
  - Existen distintos lugares donde almacenar información: memoria ROM, memoria caché del procesador, registros del procesador, almacenamiento temporal de dispositivos externos (unidades de disco, CD o DVD, tarjetas gráficas).
  - Memoria central: memoria RAM.
    - ✓ Memoria de gran capacidad que permite el acceso directo a los datos a una velocidad razonablemente rápida.
    - ✓ Almacena tanto los programas que se están ejecutando, como los que éstos utilizan.
    - ✓ Memoria volátil donde se guarda información temporal.
    - ✓ Compuesta de celdas o palabras de compuestas por un número de bits que depende del procesador.
      - Cada vez que la CPU manda una orden de lectura o escritura en memoria de lee o escribe una palabra.

# Hardware (VI)

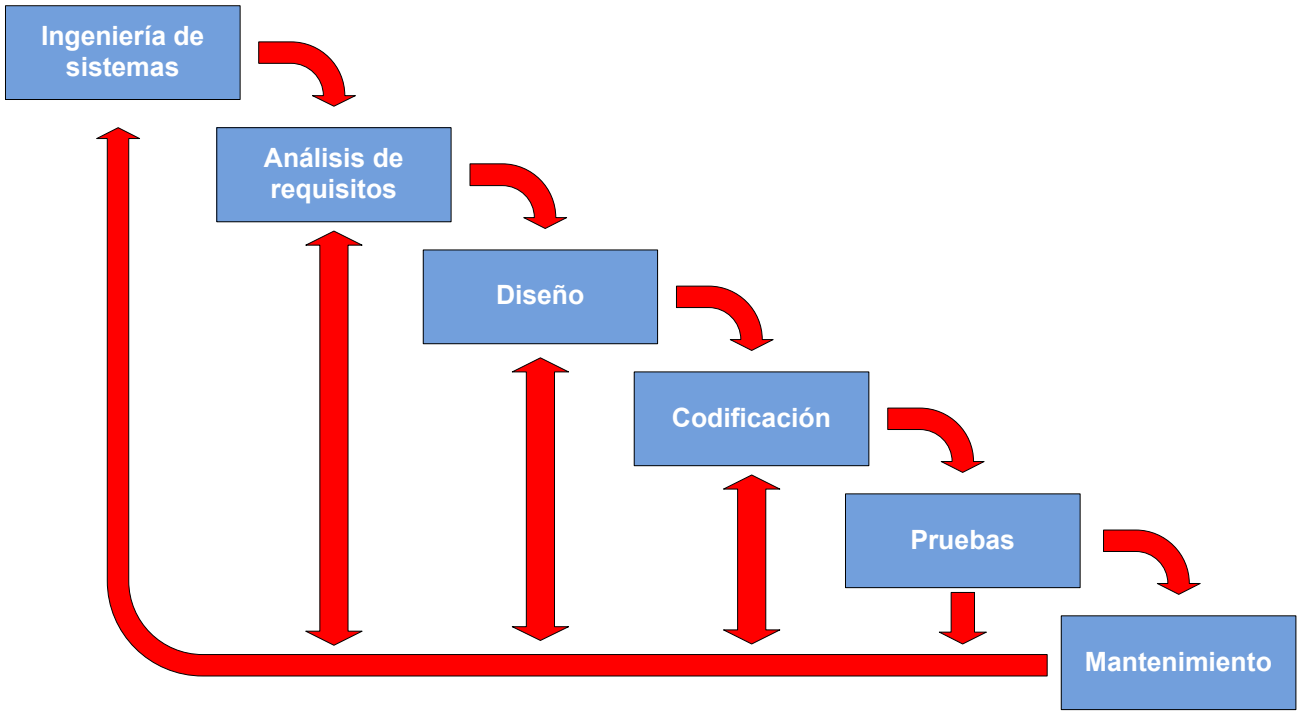
- Almacenamiento secundario.
  - Inconvenientes de la memoria central:
    - ✓ Capacidad limitada.
    - ✓ Carácter volátil de la información.
  - Necesidad de mecanismos que permitan almacenar grandes volúmenes de información de forma permanente.
  - Normalmente se trata de soportes magnéticos u ópticos.
  - La información se organiza en forma de archivos con información relacionada.
    - ✓ Archivos de texto, bases de datos, gráficos, vídeos, programas de ordenador.
    - ✓ Esos archivos se transferirán total o parcialmente hacia o desde la memoria central.

# Software

- Cualquier componente lógico de un sistema informático (programas, manuales, sistemas de ayuda, etc.).
- En general hace referencia a los programas informáticos.
- El ordenador es una herramienta multipropósito.
  - Es lo suficientemente flexible para poder hacer cualquier cosa.
  - El software (los programas) determinarán las tareas que deben hacer los componentes del sistema.
- Un programa es el conjunto de sentencias o instrucciones que se suministran a la computadora (el hardware) para que realice una función.
  - Esas sentencias se describen en el programa mediante un **lenguaje de programación**.

# Fases en la resolución de problemas mediante ordenadores

- La ingeniería de software propone una serie de fases que habrá que seguir para el desarrollo de un programa informático.



# Fases en la resolución de problemas mediante ordenadores (II)

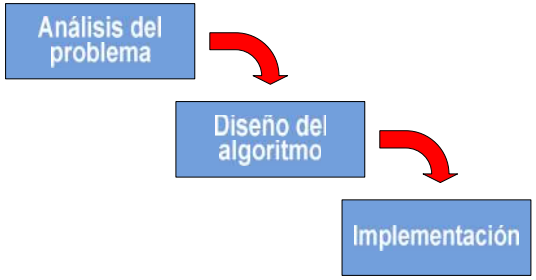
- Ingeniería y análisis del sistema.
  - El software siempre es parte de un sistema mayor.
    - ✓ Se considera el sistema en su totalidad.
  - El desarrollo de software comenzará estudiando todos los requisitos de todos los elementos del sistema (sean o no informáticos).
  - A partir de éstos se asignará un subconjunto al software.
- Análisis de requisitos del software.
  - Estudia los requisitos necesarios para cumplir la funcionalidad del problema.
  - Se realiza junto con el cliente y determina las características básicas que debe cumplir.

# Fases en la resolución de problemas mediante ordenadores (III)

- Diseño.
  - Una vez se conocen los requisitos se estudiarán las partes constitutivas del programa.
    - ✓ Estructuras de datos necesarias.
      - Diseñar la información necesaria para resolver el problema.
    - ✓ Arquitectura del software.
      - Diseñar los distintos componentes software necesarios (módulos) para resolver el problema y las relaciones entre ellos.
    - ✓ Detalle procedimiental.
      - Diseñar los algoritmos de cada uno de esos componentes.
- Codificación.
  - Transcripción de los resultados del diseño a un lenguaje de programación.
- Pruebas.
  - Diseñar un conjunto de datos y tareas a realizar para probar la aplicación antes de entrar en la fase de producción.
- Mantenimiento.
  - Modificaciones que debe sufrir la aplicación…
    - ✓ Bien por errores detectados en la fase de producción.
    - ✓ Bien por nuevas necesidades de los clientes o cambios en los procesos productivos.

# Fases en la resolución de problemas: Modelo de desarrollo resumido

- Modelo de desarrollo resumido.
  - Análisis del problema.
    - ✓ Comprender la idea general de lo que debe hacer el problema.
    - ✓ Definir el problema indicando los resultados que se quieren obtener (datos de salida) y los datos que disponemos (datos de entrada).
  - Diseño del algoritmo
    - ✓ Proceso detallado de los pasos que hay que dar para convertir los datos de entrada en resultados.
  - Implementación y pruebas.
    - ✓ Traducción del algoritmo a un lenguaje de programación.
    - ✓ Verificación del buen funcionamiento del sistema.



# Análisis del problema

- Definición de las tareas que debemos resolver.
- Un análisis simplificado del problema debería responder a estas preguntas.
  - ¿Qué queremos obtener?
    - ✓ Especificación de los datos de salida.
    - ✓ Es interesante también delimitar el formato o tipo de esos datos.
      - ¿datos numéricos? ¿una fecha? ¿una secuencia de caracteres?
  - ¿De qué datos disponemos?
    - ✓ Especificación de los datos de salida.
    - ✓ También se debería delimitar su formato.
  - ¿Cómo se convierten los datos de entrada en datos de salida?
    - ✓ Una idea general de cómo procesar los datos de entrada para convertirlos en salida.
    - ✓ Ayudará a definir el futuro algoritmo del problema.
    - ✓ Permitirá detectar carencias iniciales, como la ausencia de información relevante para el proceso.

# Análisis del problema (II)

## ❑ Ejemplo 1.1

- Se desea realizar un programa que permita multiplicar dos números enteros positivos sin utilizar la multiplicación.
  - ✓ **Datos de salida**: El producto de los dos números que será también un número entero positivo.
  - ✓ **Datos de entrada**: Dos números. Se supone que se trata de dos enteros positivos, aunque no se va a comprobar. En caso contrario el programa no funcionaría.
  - ✓ **Proceso:** Se deberá acumular el primero de los números de entrada tantas veces como indique el segundo.

# Análisis del problema (III)

- **Ejemplo 1.2.**
  - Se desea realizar el programa de un terminal de punto de venta en el que se irán introduciendo precios de artículos y la cantidad de unidades para obtener el total de la factura.
    - ✓ **Datos de salida**: El total de la factura, un número real.
    - ✓ **Datos de entrada**: El precio del artículo y la cantidad de unidades vendidas de cada referencia. El primer dato será real, mientras que las unidades será un número entero positivo.
    - ✓ **Proceso:** Por cada referencia artículo se deberá acumular el subtotal del producto.
- **Ejemplo 1.2.*bis***
    - ✓ **Datos de salida:** El total de la factura, un número real.
    - ✓ **Datos de entrada:** El precio del artículo y la cantidad de unidades vendidas de cada referencia. El precio será un número real (estará en euros) y la cantidad un número entero positivo.
    - ✓ **Datos auxiliares:** El subtotal, un número real.
    - ✓ **Proceso:** Por cada referencia artículo se deberá acumular el subtotal del producto. El subtotal se calculará multiplicando el precio del artículo por las unidades vendidas.

# Diseño del algoritmo
## Concepto de algoritmo

- Es necesario definir claramente los pasos que se deben dar para convertir los datos de entrada en la salida requerida.
- Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.
- Método y notación en las distintas formas del cálculo.
- Su ejecución no puede llevar ninguna decisión subjetiva.
- En el ámbito de la programación:
  - Secuencia detallada de instrucciones que debe ejecutar un programa para resolver un problema.
  - Debe ser independiente:
    - ✓ Del lenguaje de programación.
    - ✓ De la plataforma donde se ejecute.

# Diseño del algoritmo
## Características de los algoritmos

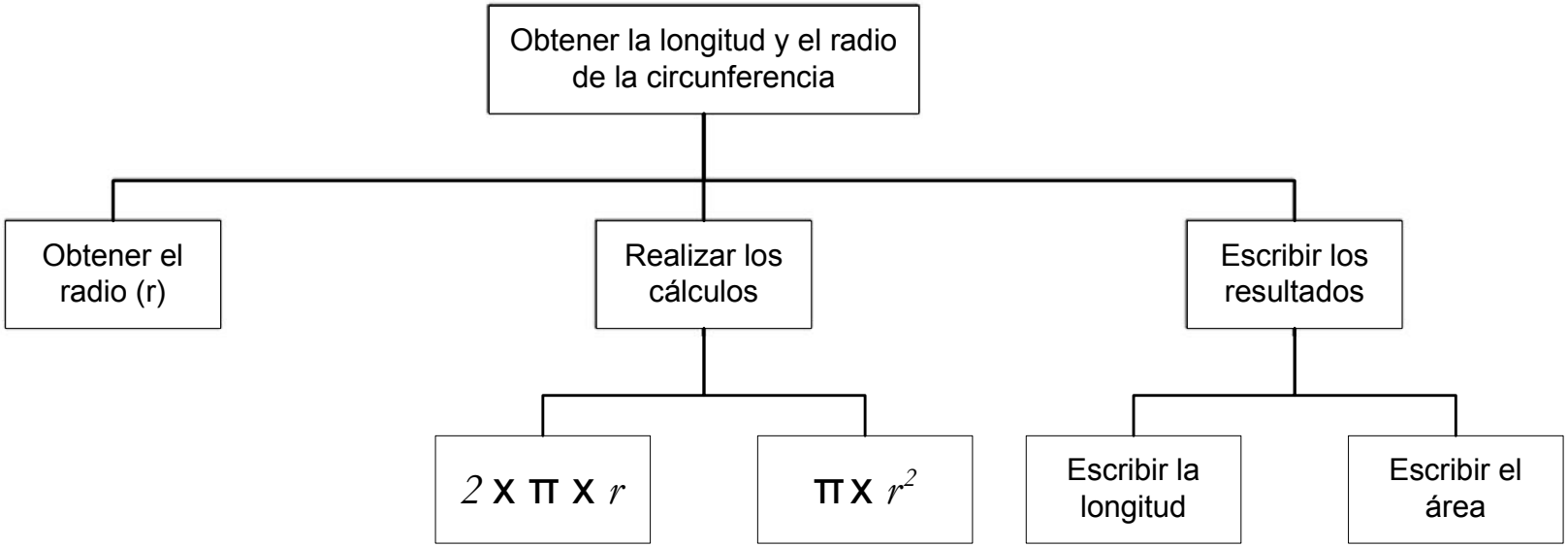
- Un algoritmo debe ser:
  - **Preciso**: debe indicar el orden de realización de cada paso.
  - **Definido**: si se sigue dos veces con los mismos datos de entrada, se deben obtener los mismos resultados.
  - **Finito**: si se sigue un algoritmo se debe terminar en algún momento, es decir, debe tener un número finito de pasos.
- En general, describirá tres partes:
  - **Entrada**. Se suministran al algoritmo los datos a partir de los cuales se desean obtener unos resultados.
  - **Proceso**. Esos datos son modificados mediante las instrucciones que contiene.
  - **Salida**. Debe suministrar esa información procesada al usuario o a otro programa que haga uso de ella.

# Diseño del algoritmo
## Refinamientos sucesivos

- La aproximación inicial al algoritmo se realiza mediante *refinamientos sucesivos*.
  - Descomposición del problema hasta llegar a las instrucciones elementales que puedan expresarse en un lenguaje de programación.
- Se expresan mediante un lenguaje de especificación de algoritmos.
- **Ejemplo 1.3**
  - Calcular la longitud y el área de una circunferencia a partir del radio de la misma
    - ✓ **Datos de salida**: La longitud y el área, dos números reales.
    - ✓ **Datos de entrada**: El radio (r) de la circunferencia.
    - ✓ **Datos auxiliares**: La constante π.
    - ✓ **Proceso**: Una vez obtenido el radio de la circunferencia habrá que aplicar las fórmulas de la longitud $2 \cdot \pi \cdot r$ y el área $\pi \cdot r2$.

# Diseño del algoritmo
## Refinamientos sucesivos (II)

- A partir del problema inicial, se descompone en acciones más elementales (subproblemas).
  - Esos subproblemas se podrán dividir a su vez hasta llegar a definir instrucciones que puedan ser representadas en un lenguaje de alto nivel.
    - ✓ Obtener, calcular, escribir son instrucciones elementales.



# Diseño del algoritmo
## Descripción de algoritmos

- La descripción de un algoritmo se puede hacer mediante lenguaje natural.
  - Algoritmo del ejemplo 1.3.
    1. Obtener el radio
    2. Calcular la longitud mediante la fórmula $2 \times \pi \times r$
    3. Calcular el área mediante la fórmula $\pi \times r^2$
    4. Escribir la longitud
    5. Escribir el área

# Diseño del algoritmo
## Descripción de algoritmos (II)

- La descripción de un algoritmo se puede hacer mediante lenguaje natural.
  - **Ejemplo 1.4.**
    - ✓ Describir el algoritmo del ejemplo 1.1 (multiplicar 2 números).
      1. Obtener el primer factor de la multiplicación
      2. Obtener el segundo factor de la multiplicación
      3. Acumular el primer factor al resultado de la multiplicación
      4. Si todavía no se ha acumulado el primer factor tantas veces como indica el segundo, ir al paso 3. En caso contrario continuar en el paso 5
      5. Escribir el resultado de la multiplicación

# Diseño del algoritmo
## Descripción de algoritmos (III)

### ❑ Ejemplo 1.5.

- Describir el algoritmo del ejemplo 1.2 (calcular el precio total de una compra).
  1. Obtener el precio del producto
  2. Obtener el número de unidades vendidas de ese producto
  3. Multiplicar el precio del producto por las unidades para calcular el subtotal
  4. Acumular el subtotal al total
  5. Si hay más productos ir al paso 1. En caso contrario continuar en el paso 6
  6. Escribir el total de la venta

# Diseño del algoritmo
## Verificación y tabla de seguimiento

- Aunque la prueba final del algoritmo se hará después de la codificación, en la fase de diseño también es posible realizar pruebas informales.
- La tabla de seguimiento es una herramienta que permite probar el algoritmo sobre el papel.
- Consiste en una tabla con todos los datos que intervienen en el programa.
  - Se proporcionarán datos de entrada ficticios.
    - ✓ Puede ser conveniente coger situaciones límite.
      - Datos extremos, situaciones en las que nos figuremos que el programa puede fallar.
  - Se irá siguiendo el algoritmo paso a paso y anotando sus resultados.

# Diseño del algoritmo
## Verificación y tabla de seguimiento (II)

| Instrucciones | Variables del algoritmo | | | Observaciones |
|---|---|---|---|---|
| | factor 1 | factor 2 | producto | |
| 1. Obtener el primer factor | 2 | | | Se quiere hacer la prueba con 2 x 3 |
| 2. Obtener el segundo factor | | 3 | | |
| 3. Acumular el primer factor al producto | | | ? | ¿Cuánto vale producto? No está determinado, dependiendo del lenguaje el resultado puede ser imprevisible |
| … | | | | |

- Los algoritmos 1.4 y 1.5 presentan una indefinición.
  - Es un error presuponer que el valor inicial del producto o del total sea 0.
    - ✓ Esto no ocurre en todos los lenguajes.
  - Con los mismos datos de entrada no siempre saldrán los mismos resultados: **no están definidos**

# Diseño del algoritmo
## Verificación y tabla de seguimiento (III)

### ❑ Ejemplo 1.4 (vesión modificada).

- Describir el algoritmo del ejemplo 1.1 (multiplicar 2 números).
  1. Poner el producto de la multiplicación a 0
  2. Obtener el primer factor de la multiplicación
  3. Obtener el segundo factor de la multiplicación
  4. Acumular el primer factor al producto de la multiplicación
  5. Si todavía no se ha acumulado el primer factor tantas veces como indica el segundo, ir al paso 4. En caso contrario continuar en el paso 6
  6. Escribir el resultado de la multiplicación

# Diseño del algoritmo
## Verificación y tabla de seguimiento (IV)

| Instrucciones | Variables del algoritmo | | | Observaciones |
|---|---|---|---|---|
| | **factor 1** | **factor 2** | **producto** | |
| 1. Poner el producto de la multiplicación a 0 | | | 0 | |
| 2. Obtener el primer factor | 2 | | | Se quiere hacer la prueba con 2 x 3 |
| 3. Obtener el segundo factor | | 3 | | |
| 4. Acumular el primer factor al producto | | | 2 | Se ha acumulado 1 vez |
| 4. Acumular el primer factor al producto | | | 4 | Se ha acumulado 2 veces |
| 4. Acumular el segundo factor al producto | | | 6 | Se ha acumulado 2 veces |
| 6. Escribir el resultado del producto | | | | Se escribe 6 |

# Diseño del algoritmo
## Verificación y tabla de seguimiento (V)

### ❑ Ejemplo 1.5.

- Describir el algoritmo del ejemplo 1.2 (calcular el precio total de una compra).
  1. Poner el total del producto a 0
  2. Obtener el precio del producto
  3. Obtener el número de unidades vendidas de ese producto
  4. Multiplicar el precio del producto por las unidades para calcular el subtotal
  5. Acumular el subtotal al total
  6. Si hay más productos ir al paso 2. En caso contrario continuar en el paso 7
  7. Escribir el total de la venta

# Diseño del algoritmo
## Verificación y tabla de seguimiento (IV)

| Instrucciones | Variables del algoritmo | | | | | Observaciones |
|---|---|---|---|---|---|---|
| | precio | unidades | subtotal | total | ¿Más? | |
| 1. El total del producto a 0 | | | | 0 | | |
| 2. Obtener el precio del producto | 15 | | | | | |
| 3. Obtener el número de unidades | | 2 | | | | Se compran dos unidades del primer producto a 15 euros cada una |
| 4. Multiplicar el precio del productos por las unidades para calcular el subtotal | | | 30 | | | |
| 5. Acumular el subtotal al total | | | | 30 | | |
| 6. ¿Hay más productos? | | | | | si | |
| 2. Obtener el precio del producto | 10,5 | | | | | |
| 3. Obtener el número de unidades | | 3 | | | | Se compran tres unidades del segundo producto a 10.5 euros cada una |
| 4. Multiplicar el precio del productos por las unidades para calcular el subtotal | | | 31,5 | | | |
| 5. Acumular el subtotal al total | | | | 61,5 | | |
| 6. ¿Hay más productos? | | | | | no | |
| 7. Escribir el total de la venta | | | | | | Se rescribe 61,5 |

# Implementación y pruebas

- Constituye la tercera fase del desarrollo de un programa.
- La fase de diseño del algoritmo da como resultado los pasos que debe seguir el ordenador.
- Es necesario **codificar** ese algoritmo.
  - La **codificación** implica traducir esos pasos a un **lenguaje de programación**.
- El programa resultante deberá ser verificado mediante juegos de ensayo o de prueba.

# Implementación y pruebas
## Lenguajes de programación

- Objetivo: permitir que las personas (los programadores) indiquen al ordenador los pasos que se deben seguir.
- El lenguaje de programación proporciona la sintáxis y reglas semánticas que definen el programa.
- Se componen de:
  - Conjunto de símbolos sobre el que se define el léxico (vocabulario) del lenguaje.
  - Conjunto de reglas (sintaxis) que sirven para la construcción de instrucciones correctas del lenguaje, también llamadas sentencias o proposiciones.
  - El compilador asocia a cada sentencia un significado (semántica) que constituirán las instrucciones que realiza el ordenador..

# Lenguajes de programación
## Clasificación según su nivel de abstracción

- El nivel de abstracción de un lenguaje indica su cercanía con la máquina.
  - Cuando más lejano a la máquina más abstracto.
- Según su nivel de abstracción los lenguajes se puede clasificar en:
  - Lenguajes de bajo nivel
    - ✓ Lenguaje máquina
      - Para algunos autores no estaría dentro de esta categoría y tendría una categoría propia. un lenguaje de bajo nivel y su .
  - Lenguajes de alto nivel.
    - ✓ Constituyen la mayoría de los lenguajes utilizados en la actualidad (C, C++, Java, etc.).

# Lenguajes de programación
## Clasificación según su nivel de abstracción (II)

- Lenguaje máquina
  - Escritos en instrucciones directamente inteligibles por el ordenador.
    - ✓ Las instrucciones están formadas por secuencias binarias (0 y 1).
      - Por ejemplo, sumar dos números almacenados en memoria y guardarlo en otra posición de memoria podría expresarse como: `000000 00001 00010 00110 100000`
        - Dónde los 6 primeros bits expresarían el orden de la instrucción, los cinco siguientes la primera posición de memoria, los otros 5 la segunda posición de memoria de entrada, los cinco siguientes la posición de salida y los 6 último el código de la operación
    - ✓ Ventajas:
      - No precisan ningún tipo de traducción
      - En ocasiones, pueden ser más eficientes que otros lenguajes (mayor velocidad y menor ocupación).
    - ✓ Inconvenientes:
      - Totalmente dependientes de la máquina donde se ejecutan.
      - El número de instrucciones es muy limitado.
      - Gran dificultad de escritura y depuración.
  - En la actualidad los inconvenientes superan a las ventajas.

# Lenguajes de programación
## Clasificación según su nivel de abstracción (III)

- Lenguaje ensamblador (*assembly language*).
  - Sustituye las instrucciones máquina por instrucciones nemotécnicas.
    - ✓ Por ejemplo, para sumar dos números la instrucción podría ser algo similar a:
      - `ADD AX,BX`, para sumar los números contenidos en `AX` y BX.
      - `MOV al, 61h`, para mover a al el valor 61 en hexadecimal.
      - `JUMP seccion`, para saltar a la instrucción situada en sección.
      - …
  - Ventajas:
    - ✓ Rapidez de proceso por su cercanía con la máquina.
    - ✓ Poca ocupación de memoria.
  - Desventajas.
    - ✓ Requiere traducción a lenguaje máquina.
      - Un programa llamado *assembler* se encarga de traducirlo a lenguaje máquina.
    - ✓ Dependiente de la máquina donde se ejecutan.
    - ✓ Dificultad de desarrollo y depuración.
  - Su uso ha quedado limitado a aplicaciones que necesitan un control muy fuerte del hardware o en las que el espacio de almacenamiento es crucial.

# Lenguajes de programación
## Clasificación según su nivel de abstracción (IV)

- Lenguajes de alto nivel.
  - Traducen varias instrucciones máquina a sentencias similares al lenguaje humano.
  - Ventajas:
    - ✓ Independientes de la máquina.
    - ✓ Facilidad de aprendizaje y depuración.
    - ✓ Al no depender de la máquina los programas escritos en un lenguaje de alto nivel se pueden ejecutar en distintos procesadores: *portabilidad*.
    - ✓ Reducción de costes en la programación.
  - Inconvenientes:
    - ✓ Necesidad de traducción.
    - ✓ En principio, menos eficientes tanto en velocidad como en ocupación en memoria.
  - Los inconvenientes actualmente no tienen mucho peso por el incremento de la velocidad de los ordenadores y de la capacidad de almacenamiento.

# Lenguajes de programación
## Clasificación según su generación

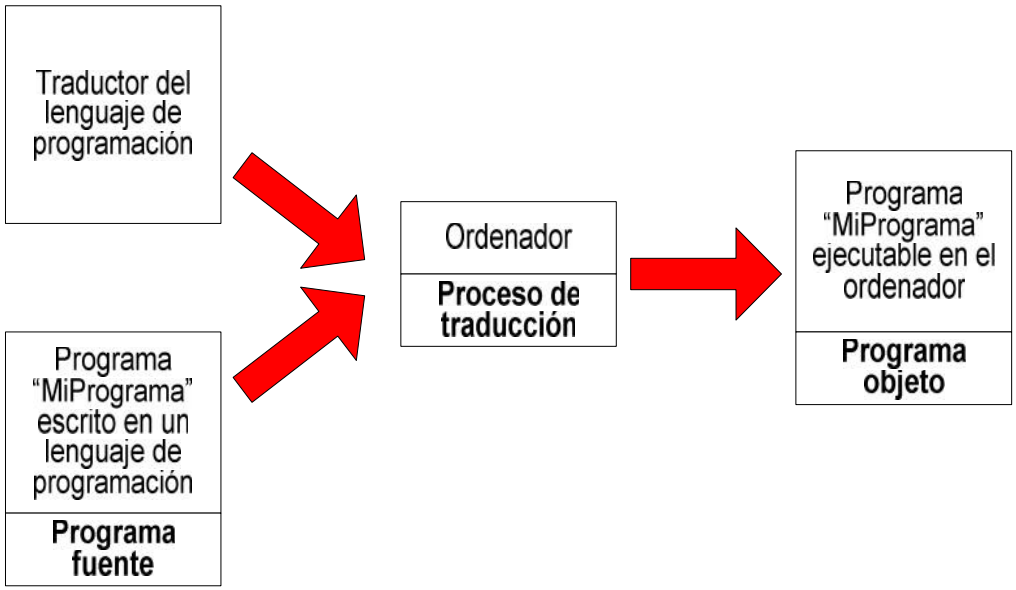
- Primera generación de lenguajes.
  - Lenguajes poco desarrollados, cercanos a la máquina.
    - ✓ Lenguaje máquina.
- Segunda generación de lenguajes.
  - Lenguajes simbólicos que sustituyen las instrucciones en código máquina por símbolos nemotécnicos: lenguaje ensamblador
- Tercera generación de lenguajes.
  - Primeros lenguajes de alto nivel.
  - Utilizan estructuras de datos abstractas (vectores) y un conjunto de instrucciones para gestionar el flujo de ejecución del programa.
    - ✓ Cobol, Fortran, Basic, Algol.
  - A partir de los años 70.
    - ✓ Utilizan tipos de datos de forma consistente y permiten la definición de tipos de datos propios.
    - ✓ Permiten asignación dinámica de memoria.
    - ✓ Primeros lenguajes orientados a objetos.
      - ○ Pascal, C, C++, SmallTalk, Java.
- Cuarta generación de lenguajes.
  - Enfocados a usos específicos: manejo de bases de datos (SQL, Natural), generadores de informes (Crystal Reports), lenguajes de animación (ActionScript), etc.

# Lenguajes de programación
## Clasificación según el paradigma utilizado

- Según el paradigma utilizado.
  - Lenguajes imperativos.
    - ✓ El programa es un conjunto de instrucciones que indican al ordenador *cómo* realizar una tarea.
      - Pascal, Cobol, C, Fortran, etc.
  - Lenguajes declarativos.
    - ✓ En lugar de utilizar órdenes, se "declaran" condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución.
    - ✓ Sentencias que indican *qué* es lo que se quiere hacer.
    - ✓ Lenguajes funcionales o lógicos.
      - Lisp, Haskell, Prolog.
  - Lenguajes orientados a objetos.
    - ✓ El objeto englobaría los datos y las instrucciones para manejarlos.
    - ✓ El programa consistiría en el envío de mensajes a objetos que reaccionarían ejecutando las instrucciones.
      - C++, Java, SmallTalk, etc.
- C sería un lenguaje de alto nivel, de la tercera generación de lenguajes y que sigue el paradigma imperativo.

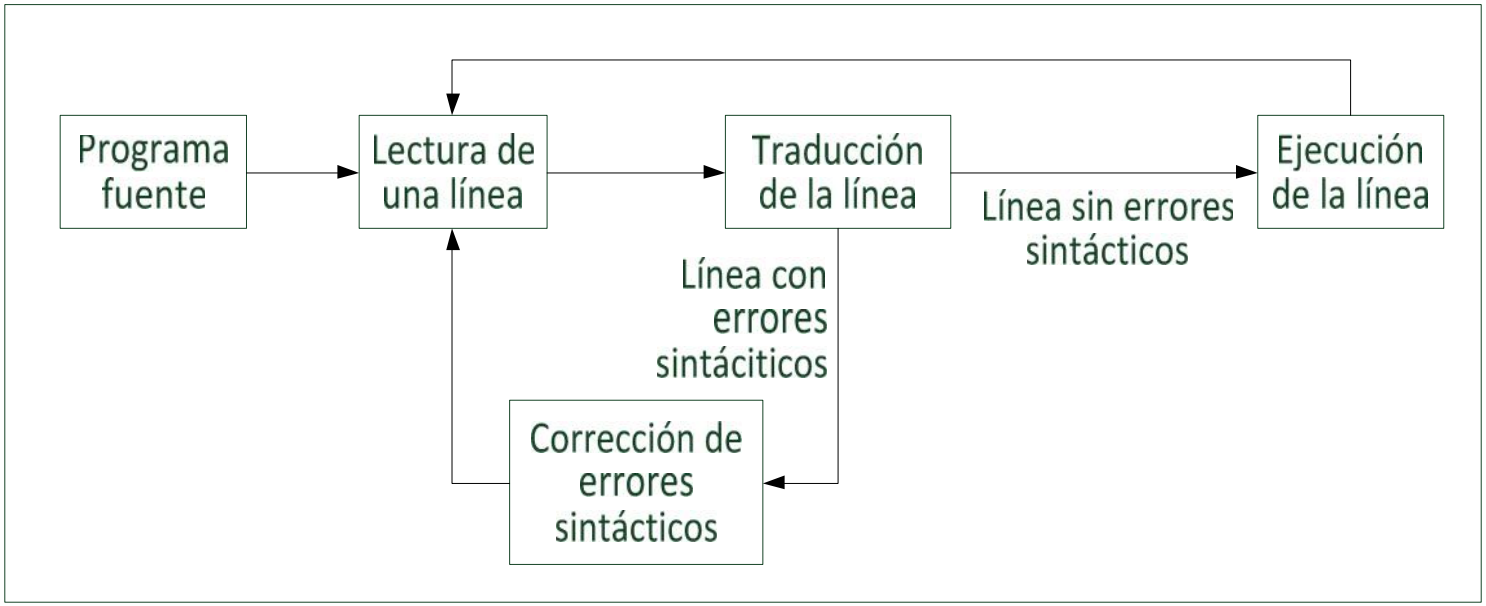
# Lenguajes de programación
## Traductores del lenguaje

- El ordenador sólo entiende códigos binarios.
- Cualquier lenguaje distinto del lenguaje máquina precisa de un proceso de traducción para que pueda ser interpretado por la CPU del ordenador.



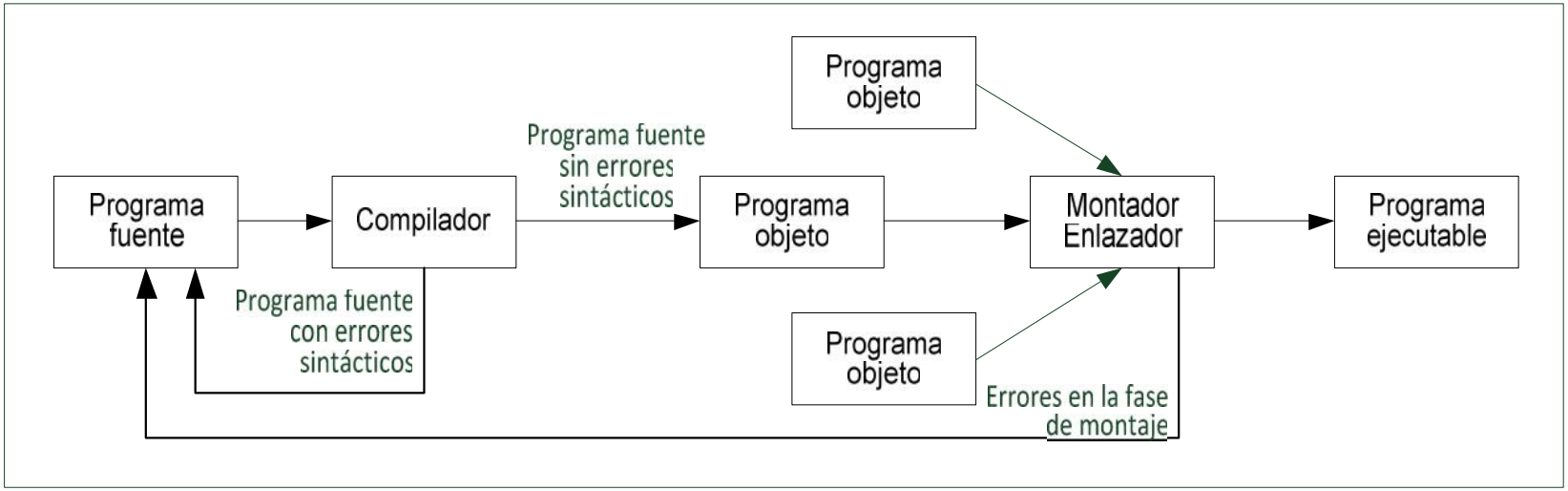
# Lenguajes de programación
## Traductores del lenguaje: interpretes

- Toma el programa fuente, lo traduce y lo ejecuta.
  - Cada vez que se ejecuta el programa es necesario realizar el proceso de traducción.
- El proceso traducción-ejecución se realiza línea a línea.
- Los errores del programa aparecen de forma interactiva mientras se está ejecutando el programa.



# Implementación y pruebas
## Traductores del lenguaje: compiladores

- Traducen el programa fuente y obtienen el programa objeto.
  - No ejecutan, sólo traducen el programa fuente.
  - Una vez traducido pueden ejecutarse sin necesidad de volver a traducir.

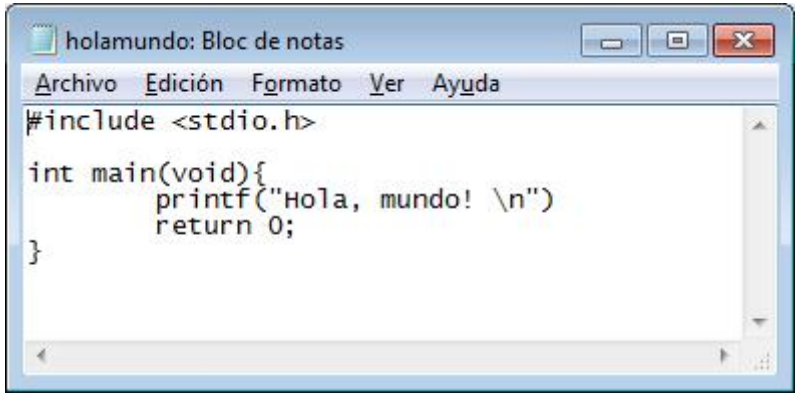


# Lenguajes de programación
## Interpretes vs. compiladores

- Un programa compilado se ejecuta más rápido que uno interpretado.
- En un programa interpretado la depuración (corrección de errores) es más simple e interactiva.
- La aplicaciones profesionales y comerciales utilizan lenguajes compilados.
- Los lenguajes interpretados se han utilizado en la fase de desarrollo de un proyecto, en la enseñanza de la programación o en aplicaciones científicas o académicas.
- El uso de un compilador precisa de tres programas distintos:
  - Un editor de textos.
  - Un compilador
  - Un enlazador.
  - Los entornos de programación integrados (IDE, *integrated development environment*) solucionan este problema.
    - ✓ Los IDE también solucionan las dificultades de depuraciónde los lenguajes compilados
- Resurgir de los lenguajes interpretados (lenguajes de *script*).
- Otros modelos de compilación.
  - Máquina Virtual Java (JVM)
  - Compilación *just in time* de la plataforma .NET.

# Lenguajes de programación
## Ejemplo de compilación en C

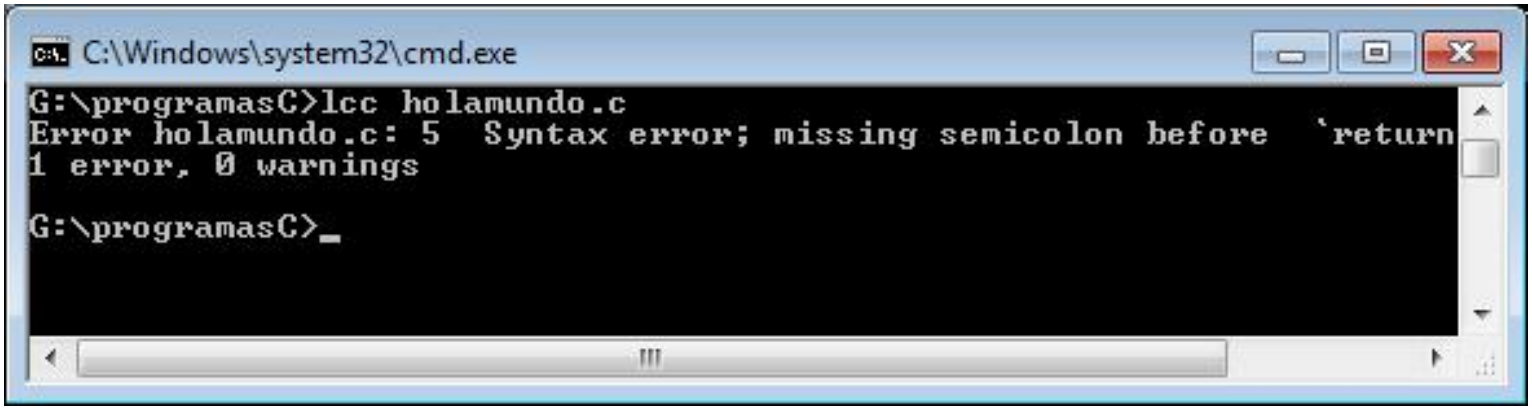
- Mediante un editor de texto (por ejemplo el bloc de notas de Windows) se edita el programa fuente (`holamundo.c`).



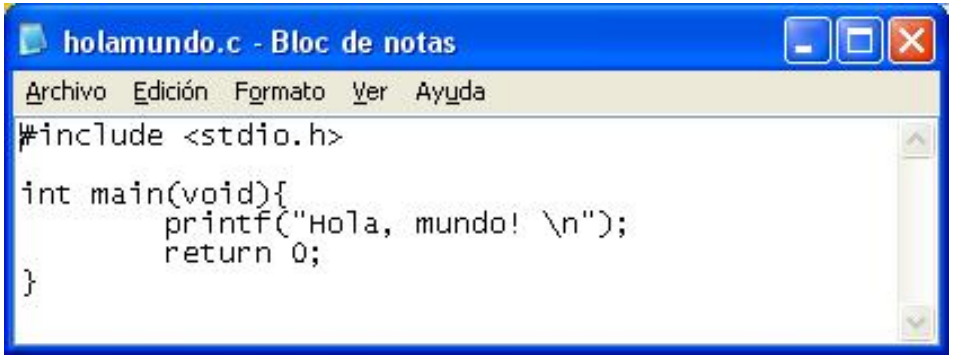

- Se ejecuta el compilador de C(en este caso `lcc`).

# Lenguajes de programación
## Ejemplo de compilación en C (II)

- Si existen errores, el compilador los indica (en este caso falta un punto y coma en la línea 5).




- Es necesario volver abrir el editor de texto para corregir el error.



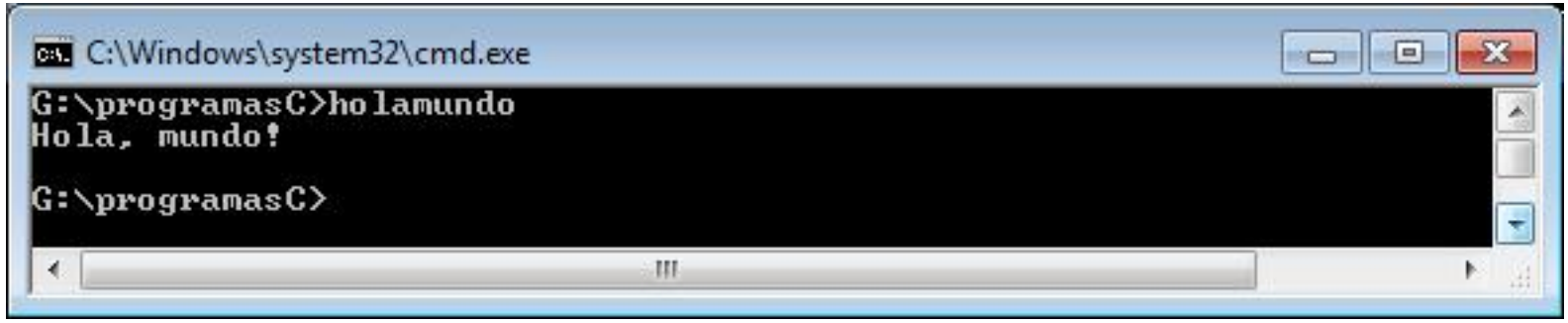
# Lenguajes de programación
## Ejemplo de compilación en C (III)

- Se vuelve a compilar, si no hay errores, genera el programa objeto (`holamundo.obj`).


- Se enlaza el programa objeto con el *linker* (en este caso `lcclnk`).

# Lenguajes de programación
## Ejemplo de compilación en C (IV)

- Por último, se ejecuta el programa y se obtienen los resultados.



# Herramientas de programación
## Diagramas de flujo

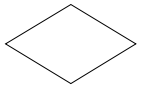
- Ya se han descrito algoritmos utilizando lenguaje natural, pero…
- El lenguaje natural es ambiguo.
  - Las **herramientas de programación** permiten describir los algoritmos de forma más precisa, normalizada y estándar.
    - ✓ Diagramas de flujo.
    - ✓ Pseudocódigo.
    - ✓ …
- Diagramas de flujo.
  - Herramienta gráfica que representa el flujo de información a lo largo de la ejecución del programa.
  - Utiliza cajas con formas estandarizadas para representar las instrucciones.
  - Las cajas están conectadas por flejas que muestran el flujo de la información a lo largo del algoritmo.

# Herramientas de programación
## Símbolos utilizados en los diagramas de flujo

**Terminal**. Indica el comienzo y el final del algoritmo. Las palabras “inicio” y “fin” indicarán de que se trata

**Proceso**. Cualquier tipo de operación que pueda originar un cambio de valor, formato o posición de la información almacenada en memoria.

**Entrada/Salida**. Representa cualquier carga de datos desde el exterior a la memoria (desde teclado, disco, o cualquier otro dispositivo de entrada) o la salida de información a cualquier dispositivo de salida (impresora, pantalla, disco)



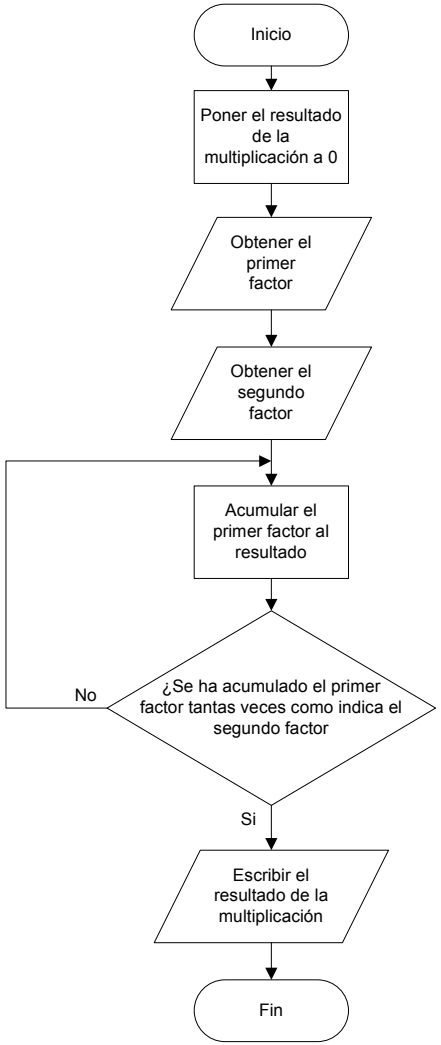
**Decisión**. Implica una ruptura en el flujo lineal de la información a lo largo del diagrama. Uno de los vértices del rombo indicará la entrada de información. El texto contenido en el rombo contendrá algún tipo de pregunta sobre la que se tomará la decisión. El resto de vértices representan las distintas salidas. Se tomará una u otra salida en virtud de la respuesta a la expresión contenida en el círculo. Cada salida estará etiquetada con alguna de las respuestas

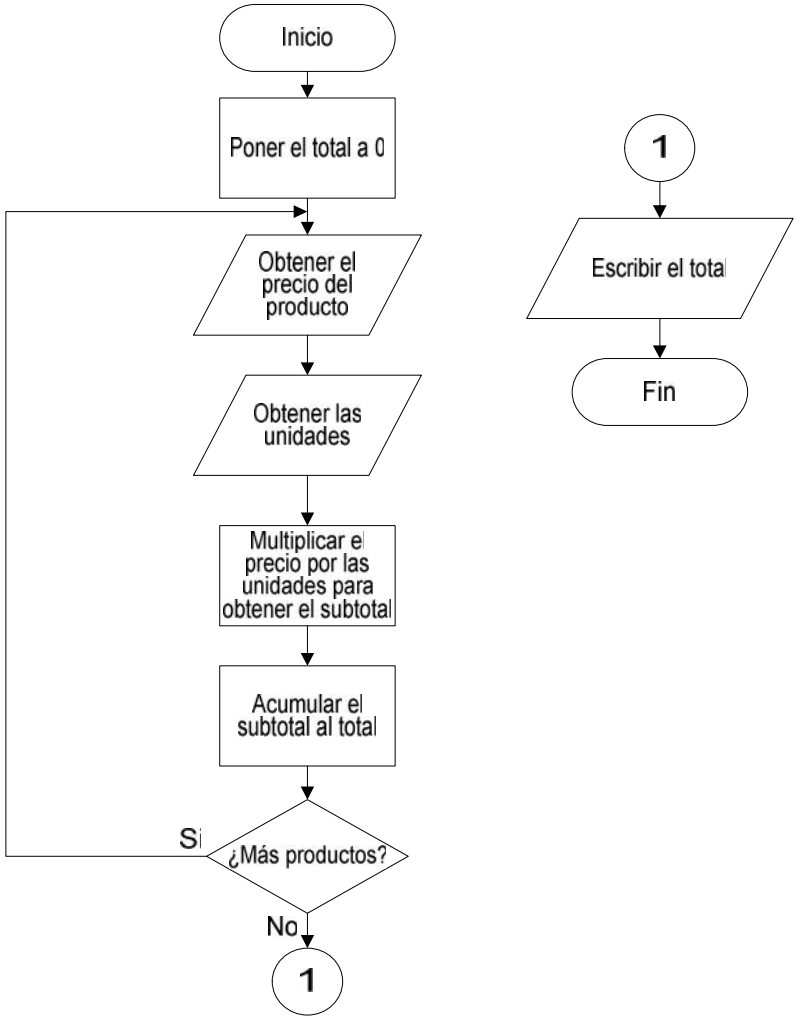
**Llamada a un módulo**. Llama al módulo identificado por la etiqueta del símbolo (*véase Tema 4)*

**Conector en página**. Une distintas partes del diagrama dentro de la misma página física donde se encuentra el diagrama. Una etiqueta indicará las distintas partes que conecta.

**Conector fuera de página**. Une distintas partes del diagrama dentro de las distintas páginas sobre las que se desarrolla el diagrama. Una etiqueta indicará las distintas partes que conecta.

# Herramientas de programación
## Ejemplos de diagramas de flujo



# Herramientas de programación
## Pseudocódigo

- Método de especificación de algoritmos similar a un lenguaje de programación de alto nivel.
- un *pseudolenguaje* de programación, nacido, no para su proceso de compilación y ejecución en un ordenador, sino para la descripción de algoritmos.
- Ventajas:
  - Al tratarse de una herramienta no gráfica es más fácil de realizar y, sobre todo, de modificar los posibles errores que tuviera.
  - Su similitud con un lenguaje de programación hacen que el paso de la fase de diseño a la de codificación sea extremadamente fácil.
- Un algoritmo descrito mediante un pseudocódigo estaría formado por una serie de sentencias formadas por palabras reservadas, variables, constantes y expresiones escritas con una sintaxis propia.
  - En la Universidad Pontifica hemos desarrollado el **Lenguaje de especificación de algoritmos UPSAM 2.1** que será el que utilizaremos.

# Herramientas de programación
## Pseudocódigo

```
inicio
   total ← 0
   repetir
      leer(precioProducto)
      leer(unidadesProducto)
      subtotal ← precioProducto * unidadesProducto
      total ← total + subtotal
   hasta_que no haya más productos
   escribir(total)
fin
```

```
inicio
   producto ← 0
   conta ← 0
   leer(primerFactor)
   leer(segundoFactor)
   repetir
      producto ← producto + primerFactor
      conta ← conta + 1
   hasta_que conta = segundoFactor
   escribir(producto)
fin
```

# Ejercicios

- Hacer el análisis y el diseño del algoritmo para los siguientes problemas:
  1. Convertir una cantidad de pesetas a euros.
  2. Convertir una cantidad en cualquier divisa a euros.
  3. Convertir una serie de cantidades en cualquier divisa a euros.
     - ✓ El algoritmo deberá pedir distintas divisas con su nombre y tipo de cambio y ver su correspondencia en euros, hasta que el nombre de la divisa sea "fin".
  4. Buscar un número en la guía telefónica (en papel) a partir del nombre del abonado.
  5. Calcular la velocidad de una serie de corredores en una prueba de atletismo de 1500 metros. Por cada corredor se introducirán los minutos y los segundos del tiempo que ha realizado cada uno. Por cada corredor se escribirán los minutos, los segundos y la velocidad conseguida en kilómetros hora.