



Metodología y Tecnología de la Programación

Curso 2008/09

Tema 1. Introducción a la Programación

Temario

Tema 1. Introducción a la Programación

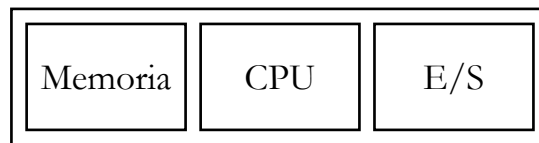
- 1.1. Resolución de Problemas y Algoritmos
- 1.2. Concepto de Programa
- 1.3. Paradigmas de Programación
- 1.4. Lenguaje de Especificación en Pseudocódigo (EAP)



Conceptos Iniciales (1/2)

- Concepto de **ordenador**: máquina digital electrónica capaz de procesar información a una velocidad muy grande
 - Digital: la información almacenada y procesada se representa mediante un conjunto finito de valores (**bits**).

Modelo de Vonn Neuman



Conceptos Iniciales (2/2)

- **Problema**: asunto o un conjunto de cuestiones que se plantean para ser resueltas
 - Problemas de distinta naturaleza
 - Objetivo de la asignatura: problemas cuya solución se puede calcular utilizando una serie de **reglas introducidas en la computadora**
- **Algoritmo**: conjunto finito, y no ambiguo de reglas expresadas en un cierto orden que, para unas condiciones iniciales, permiten resolver el problema en un tiempo finito



Cuestiones Importantes (1/2)

- No existe un método universal que permita resolver cualquier problema
- En general, la resolución de problemas es un proceso creativo donde el conocimiento, la habilidad y la experiencia juegan un papel importante
- El proceder de manera sistemática (sobre todo si se trata de problemas complejos) puede ayudar en la resolución



Cuestiones Importantes (2/2)

- El **problema** tiene que estar **bien definido**
 - Hay que saber qué es lo que hay que resolver
 - Para ello:
 - Eliminar ambigüedades
 - Eliminar la información irrelevante que aparezca en el enunciado de un problema
 - Determinar exactamente qué elementos constituyen una solución válida
- **Un mismo problema puede (y suele) tener varias soluciones**



Criterios o Estrategias Generales

- Usar toda la información útil disponible en el enunciado del problema
- Hacer explícitas las reglas y datos que parezcan implícitos
- Profundizar en el problema considerado
- Dividir un problema complejo en *subproblemas* más simples, que se puedan resolver independientemente y después combinar sus soluciones
- Otra forma de abordar un problema consiste en trabajar "hacia atrás", es decir, partir de la solución e intentar llegar al estado inicial



Afinidades entre Problemas

- El disponer de la solución de un problema afín nos puede facilitar la resolución del que nos ocupa
- Grados de afinidad:
 - Dos problemas son **isomorfos** cuando se puede establecer una aplicación biyectiva entre los estados y acciones de uno de ellos con los del otro
 - La **similaridad** es una relación entre problemas más débil, sin embargo, el proceso seguido para encontrar la solución a un problema puede ayudar a resolver otro similar
 - Otro tipo de afinidad entre dos problemas se da cuando uno de ellos es un **caso especial** del otro; resolver el caso especial puede ayudar a encontrar la solución del problema más general, o viceversa.



Propiedades de los Algoritmos (1/4)

- Propiedades que debe poseer todo buen algoritmo:
 - **Propiedad de finitud**: un algoritmo siempre debe terminar
 - **Propiedad de definitud**: toda regla de un algoritmo debe definir perfectamente la acción a desarrollar para poder aplicarla sin que pueda haber lugar a ambigüedad
 - **Propiedad de generalidad**: un algoritmo no debe contentarse con resolver un problema particular aislado sino, por el contrario, toda una clase de problemas para los que los datos de entrada y los resultados finales pertenecen respectivamente a conjuntos específicos
 - **Propiedad de eficacia**: un algoritmo debe ser eficaz, es decir, todas las operaciones a realizar deben ser lo bastante básicas para poder ser efectuadas de modo exacto y en un intervalo de tiempo finito.
 - Un algoritmo siempre es potencialmente mejorable y se ha de intentar optimizarlo, ya sea por razones de economía, rapidez o claridad



Propiedades de los Algoritmos (2/4)

- Ejemplo: algoritmo de Euclides
 - Paso 1. Tomar el número mayor como dividendo y el menor como divisor.*
 - Paso 2. Hallar el resto de la división entera.*
 - Paso 3. Si el resto es igual a 0 entonces ir al Paso 4. En caso contrario, tomar el divisor como nuevo dividendo y el resto como divisor y volver al Paso 2.*
 - Paso 4. El m.c.d. es el divisor de la última división*
- **Ejercicio de clase: analicemos sus propiedades: finitud, definitud, generalidad y eficacia**



Propiedades de los Algoritmos (3/4)

- **Ejercicio de clase: realizar el algoritmo que determina si un número es par o no**
- Primera solución:



Propiedades de los Algoritmos (4/4)

- Segunda solución:



Ejercicios de Clase (1/5)

- Desarrollar un algoritmo que solicite tres números al usuario y determine cuál de ellos es el mayor, mostrando el resultado final por pantalla.



Ejercicios de Clase (2/5)

- Desarrollar un algoritmo que solicite un número natural y determine cuántas cifras tiene.



Ejercicios de Clase (3/5)

- Desarrollar un algoritmo que determine si un número natural es primo o no



Ejercicios de Clase (4/5)

- Desarrollar un algoritmo que solicite un cadena de caracteres al usuario y determine el número de caracteres que tiene.



Ejercicios de Clase (5/5)

- Desarrollar un algoritmo que solicite un cadena de caracteres al usuario y determine el número de vocales que hay en dicha cadena de caracteres.



Temario

Tema 1. Introducción a la Programación

1.1. Resolución de Problemas y Algoritmos

1.2. Concepto de Programa

1.3. Paradigmas de Programación

1.4. Lenguaje de Especificación en Pseudocódigo (EAP)



Concepto de Programa y Lenguaje de Programación

- Para poder expresar nuestros algoritmos de forma que puedan ser reconocidos y ejecutados en un ordenador necesitamos los **lenguajes de programación**
- **Programa**: expresión de un algoritmo en un lenguaje artificial formalizado



Características de los Lenguajes de Programación

- **Sintaxis**: especifica cómo se pueden construir los programas en un lenguaje de programación
 - Permite sólo el uso de determinadas combinaciones de símbolos seleccionados y palabras clave
 - Ejemplo (en Modula-2):

```
WHILE <expresión_booleana> DO <grupo_de_sentencias>  
END;
```
- **Semántica**: permite asignar, de alguna forma, un significado a cada tipo de construcción permitida en el lenguaje, para poder escribir programas en él e interpretar su significado



Temario

Tema 1. Introducción a la Programación

1.1. Resolución de Problemas y Algoritmos

1.2. Concepto de Programa

1.3. Paradigmas de Programación

1.4. Lenguaje de Especificación en Pseudocódigo (EAP)



Definición

- Colección de patrones conceptuales que modelan la forma de razonar sobre problemas, de formular algoritmos y, a la larga, de estructurar programas
- **Idea fundamental:** modelo básico de diseño y desarrollo de programas



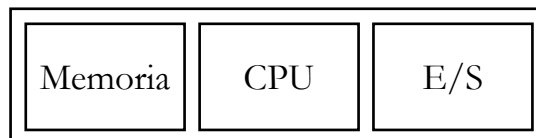
Paradigmas de Programación

Paradigma	Características
Imperativo	<i>variables, secuencia de instrucciones, asignación</i>
Funcional	<i>concepto de función, composición de funciones</i>
Orientado a Objeto	<i>objetos, clases, encapsulación, herencia, polimorfismo</i>
Lógico	<i>predicados, reglas, simbólico, deductivo</i>
Ensamblador	<i>registros, direccionamiento, interrupciones, subrutinas</i>
Heurístico	<i>estrategias de búsqueda, métodos de resolución</i>



Paradigma Imperativo: Introducción

- Basado en el modelo de Von Neumann



- **Programa:** lista de **instrucciones** u órdenes elementales que han de **ejecutarse una tras otra**, en el orden en que aparecen
- El ordenador tiene un juego de operaciones limitado, pero lo suficientemente versátil como para realizar cualquier cálculo automatizable.



Paradigma Imperativo: Variables (1/2)

- Las instrucciones de un programa imperativo utilizan datos almacenados en la memoria del ordenador → **variables**
 - Distinto significado que en matemáticas → en programación representa un dato almacenado bajo un nombre dado
 - Contienen un valor que puede ser usado o modificado tantas veces como se desee



Paradigma Imperativo: Variables (2/2)

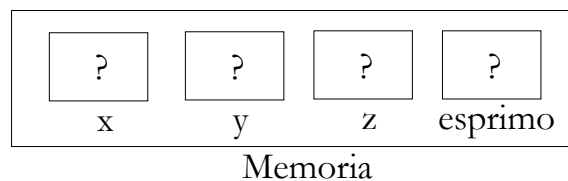
- **Declaración de variables**

$x \in \text{Natural}$

$y \in \text{Entero}$

$z \in \text{Real}$

$\text{esprimo} \in \text{Lógico}$



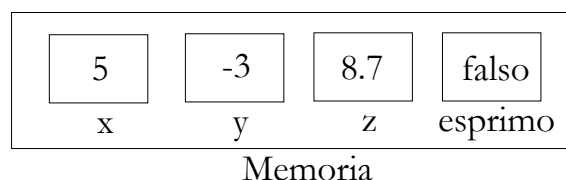
Paradigma Imperativo: Asignación (1/3)

- **Instrucciones:** operaciones realizables sobre los datos del programa
- Principal instrucción: **asignación**
 - Sintaxis: $\langle \text{variable} \rangle \leftarrow \langle \text{expresión} \rangle$
 - La *expresión* situada a la derecha del símbolo de asignación ' \leftarrow ' es una expresión funcional, donde pueden aparecer variables
 - Semántica: primero se evalúa la expresión funcional y después se almacena su valor en la variable situada a la izquierda del símbolo de asignación



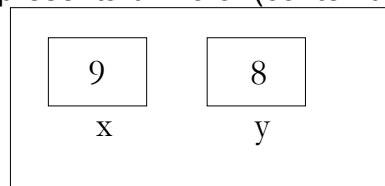
Paradigma Imperativo: Asignación (2/3)

- Ejemplo de asignación: **inicialización de variables**
 - $x \leftarrow 5$
 - $y \leftarrow -3$
 - $z \leftarrow 8.7$
 - $\text{esprimo} \leftarrow \text{falso}$



Paradigma Imperativo: Asignación (3/3)

- **Aspecto importante:** la aparición de una variable en una instrucción de asignación tiene distinto sentido, dependiendo de que aparezca a la izquierda o a la derecha del símbolo de asignación
 - Si aparece a la izquierda, representa una celda de memoria
 - Si aparece a la derecha, representa un valor (contenido en una celda de memoria)
 - Ejemplos: $x \leftarrow 4$
 $x \leftarrow y + 1$



Paradigma Imperativo: Secuencia (1/2)

- Colección de instrucciones situadas de forma consecutiva
- Forma de agrupar varias instrucciones para hacer cálculos más complicados
- Ejemplo:
 $x \leftarrow 2$
 $y \leftarrow x * x$



Paradigma Imperativo: Secuencia (2/2)

- **Ejercicio: realizar una secuencia de código que permita intercambiar el valor de dos variables**



Paradigma Imperativo: Orden de Ejecución (1/3)

- El orden de ejecución puede alterarse en caso necesario mediante el uso de instrucciones de control que permiten:
 - Ejecutar, o no, determinadas partes del programa: **estructura de control condicional (condicionales)**
 - Repetir determinadas partes del programa: **estructuras de control iterativas (bucles)**
- ➔ Ambas dependen de ciertas condiciones en los datos



Paradigma Imperativo: Orden de Ejecución (2/3)

- Estructura condicional **SI-ENTONCES-SINO**

- Sintaxis

*SI <condición> ENTONCES
<acciones-1>*

SINO

Ø | <acciones-2>

FIN SI

- Ejemplo

*SI (N mod 2 = 0) ENTONCES
Mostrar ('Es Par')*

SINO

Mostrar ('Es Impar')

FIN SI



Paradigma Imperativo: Orden de Ejecución (3/3)

- Estructura de control iterativa (o bucle) **MIENTRAS**

- Sintaxis

*MIENTRAS <condición> HACER
<acciones>*

FIN MIENTRAS

- Ejemplo

MIENTRAS (N > 0) HACER

Mostrar (N)

N ← N - 1

FIN MIENTRAS



Paradigma Imperativo: Características Fundamentales

- Concepto de **celda de memoria (variable)** para almacenar valores
 - El componente principal de la arquitectura es la memoria, compuesto por un gran número de celdas donde se almacenan los datos y que son referenciadas por medio de su nombre (**variable**)
- Operaciones de **asignación**
 - Cada valor calculado debe ser "almacenado", es decir asignado a una celda
- **Repetición**
 - Un programa imperativo normalmente realiza su tarea ejecutando repetidamente una secuencia de pasos elementales
- Aunque **existen otros componentes importantes** (tipos de datos, declaraciones, instrucciones de flujo de control, procedimientos y funciones, tipos abstractos de datos, etc.) que veremos en el tema 3



DIIC
Departamento de
Sistemas de Información
y Computación

Metodología y Tecnología de la Programación
Tema 1. Introducción a la Programación

35

Paradigma Funcional: Introducción (1/2)

- **Basado en el concepto matemático de función**
- **Función:** regla de correspondencia que asocia a cada elemento de un conjunto origen (**dominio**) un elemento de un conjunto destino (**rango**)
- La expresión $f(x)$ se denomina **aplicación de función**
- Una función se especifica dando su nombre, dominio, rango y regla de correspondencia
- **Existen otros componentes importantes:** tipos de datos, expresión condicional, recursión, etc.



DIIC
Departamento de
Sistemas de Información
y Computación

Metodología y Tecnología de la Programación
Tema 1. Introducción a la Programación

36

Paradigma Funcional: Introducción (2/2)

- Formas de declarar las **reglas de correspondencia**
 - **Enumeración**: colección de parejas formadas por un elemento del dominio y su elemento correspondiente en el rango
 - Ejemplo: función que determine si un día de la semana es festivo
 - **Dominio**: {Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo}
 - **Rango**: {cierto, falso}
 - **Regla de Correspondencia**:
 - » {Lunes, falso}
 - » {Martes, falso}
 - » {Miércoles, falso}
 - » {Jueves, falso}
 - » {Viernes, falso}
 - » {Sábado, cierto}
 - » {Domingo, cierto}
 - **Comprensión**: define una regla abstracta que describe cómo calcular, para un elemento genérico x del dominio, su correspondiente elemento y del rango
 - Ejemplo: función de incrementar un número entero en la unidad
 - **Dominio**: Enteros
 - **Rango**: Enteros
 - **Regla de Correspondencia**: $\text{Incrementar}(x) = x + 1$



Ejercicios de Clase

- Desarrollar un algoritmo **según el paradigma funcional** que dados tres números enteros halle el mayor, haciendo uso para ello de la función *Max* que devuelve el mayor de dos enteros



Paradigma Orientado a Objetos: Introducción (1/2)

- Paradigma Imperativo y Funcional: centrados en la acción → asignar, repetir, comparar, operar, ...
- Otra aproximación: **Paradigma Orientado a Objetos**
 - Centrado en el **concepto de objeto**
 - Abstracción todavía mayor que el imperativo de la máquina de Von Neumann
 - El programador ya no ve las variables como celdas de memoria, sino como objetos cuya estructura interna desconoce



Paradigma Orientado a Objetos: Introducción (2/2)

- **Objetos**
 - Pertenecientes a **clases**, que definen el conjunto de operaciones utilizables sobre los objetos
 - Gestionados mediante **mensajes**, que implican la ejecución de un subprograma que incluye en su declaración y que se denomina **método**
 - Énfasis: *qué* se obtiene más bien que en *cómo* se obtiene
- **Existen otros muchos componentes importantes:** herencia, encapsulamiento, polimorfismo, enlace dinámico, etc.



Ejemplo en Java (1/2)

```
public class Racional {  
  
    private int num, den;  
    public Racional(int n, int d) {  
        num = n;  
        den = d;  
    }  
  
    public Racional suma(Racional b) {  
        int n = (num*b.den)+(b.num*den);  
        int d = den*b.den;  
        return new Racional(n,d);  
    }  
  
}
```



Ejemplo en Java (2/2)

```
public class Programa {  
  
    static public void main (String args[]) {  
        Racional x = new Racional(4,5);  
        Racional y = new Racional(2,3);  
        Racional z= x.suma(y);  
        System.out.println("El resultado es:" + z);  
    }  
  
}
```



Temario

Tema 1. Introducción a la Programación

1.1. Resolución de Problemas y Algoritmos

1.2. Concepto de Programa

1.3. Paradigmas de Programación

1.4. Lenguaje de Especificación en Pseudocódigo (EAP)



EAP: Especificación de Algoritmos en Pseudocódigo

- **Aporta independencia de los conceptos formulados de la implementación concreta de un lenguaje de programación**
 - Lenguaje no restringido a un computador concreto ni a un paradigma de programación específico
 - EAP dispone de una sintaxis, una semántica y unos objetos primitivos que nos permitirán realizar acciones en una máquina genérica y con la que podremos construir algoritmos para la misma
- Los **componentes de EAP** los iremos introduciendo, sobre todo, a lo largo del **Tema 3** (dedicado a la especificación de algoritmos)

