Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga

TEMA 2: Resolución de problemas y algoritmos

Fundamentos de Informática
(Ingeniería Técnica Industrial)
E.U. Politécnica

E.U. Politécnica

Autores: M.C. Aranda, A. Fernández, J. Galindo, M. Trella

Índice de contenidos

1. Introducción. Conceptos básicos.

- Programación
  - Establecer una secuencia de acciones que:
    - puedan ser ejecutadas por el procesador
    - realicen una determinada tarea
  - Fases:
    1. Resolución del problema propuesto => determinación de un **algoritmo**.
    2. Adaptación del algoritmo a la computadora => codificar el algoritmo en un lenguaje que el ordenador pueda comprender

3

1. Introducción. Conceptos básicos.

- Acción
  - Etapa en la realización de un trabajo
- Acción primitiva
  - Acción que el procesador puede ejecutar sin necesidad de información suplementaria
- **Algoritmo**
  - Secuencia ordenada de acciones primitivas que realizan un trabajo

Ejemplos de algoritmos:

Ir al trabajo
1. Levantarse
2. Darse una ducha
3. Vestirse
4. Desayunar
5. Coger el coche

Cálculo de la media aritmética de dos números con una calculadora
1. Pulsar la tecla AC
2. Teclear el primer número
3. Pulsar la tecla +
4. Teclear el segundo número
5. Pulsar la tecla +
6. Pulsar la tecla /
7. Teclear el número 2
8. Pulsar la tecla =

4

1. Introducción. Conceptos básicos.

- Aspectos que se deben considerar a la hora de escribir un algoritmo
  - Determinación de las primitivas de las que partimos
  - Lenguaje simbólico a utilizar para desarrollar el algoritmo
  - Representación de los datos
  - Establecer datos de entrada
  - Establecer datos de salida
  - Establecer las relaciones entre los datos de entrada y los de salida
- Condiciones que debe cumplir un algoritmo
  - Ser **finito**
    - El algoritmo debe acabar tras un número finito de pasos
  - Estar **bien definido**
    - Todas las ejecuciones del algoritmo con los mismos datos de entrada deben devolver los mismos datos de salida
- Diferencias entre un algoritmo y un programa
  - Los algoritmos no son directamente interpretables por la computadora => deben ser traducidos a un lenguaje de programación concreto

5

1. Introducción. Conceptos básicos.

- Ejemplo de realización de un algoritmo:
  - **Problema**: *calcular la longitud de una circunferencia y el área del círculo que limita dada la longitud de su radio*
    - Determinación de las primitivas de las que partimos
      - Operaciones aritméticas simples
    - Lenguaje simbólico a utilizar para desarrollar el algoritmo
      - Lenguaje de representación de expresiones aritméticas
    - Representación de los datos
      - Cadenas de caracteres para las incógnitas
      - Números reales
    - Establecer datos de entrada
      - Radio de la circunferencia (*radio*)
    - Establecer datos de salida
      - Longitud de la circunferencia (*longitud*)
      - Área del círculo (*area*)
    - Establecer las relaciones entre los datos de entrada y los de salida
      1. *longitud = 2 * 3.1416 * radio*
      2. *area = 3.1416 * radio * radio*

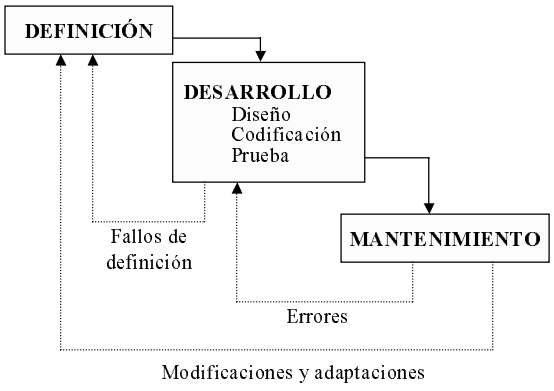
6

1. Introducción. Conceptos básicos.

- Fases de creación de un programa
  1. Definición
     - Descripción detallada de:
       - Programa que se va a realizar
       - Recursos necesarios
       - Costes
       - Plan de trabajo
  2. Desarrollo
     - Creación de programas y documentación asociada.
       - 2.1. Diseño
         - » Solución al problema (**algoritmo**)
       - 2.2. Codificación
         - » Traducción del algoritmo a un lenguaje de programación
       - 2.3. Prueba
         - » Comprobar si el programa desarrollado se corresponde con el que queríamos realizar
  3. Mantenimiento
     - Realización de mejoras y correcciones en el programa desarrollado

1. Introducción. Conceptos básicos.

- Fases de creación de un programa



2. Elementos básicos de un algoritmo.

- Datos
  - Información con la que trabaja una computadora
- Tipos de datos
  - Se clasifican atendiendo a:
    - Propiedades que poseen
    - Operaciones que se pueden realizar con ellos
  - Datos **simples**
    - Numérico
      - Real, entero
    - Carácter
      - Letras (‘a’, ‘A’,...), símbolos (‘*’, ‘%’,...), números (‘1’, ‘2’,...)
    - Lógico
      - *Verdadero* o *Falso*
    - Puntero
      - Dirección de memoria de otro dato
  - Datos **compuestos**
    - Formados por agrupaciones de otros datos (simples o compuestos)

2. Elementos básicos de un algoritmo.

- Operaciones primitivas
  - Se pueden realizar directamente en un lenguaje de programación sin necesidad de indicar cómo hay que llevarla a cabo
- Conjuntos de operaciones primitivas para los distintos tipos de datos

Numérico	
+	Suma
-	Resta
/	División
MOD	Resto de una división entera

Carácter y Numérico	
<	Menor que
>	Mayor que
= =	Igual que
<=	Menor o igual que
>=	Mayor o igual que
! =	Distinto

Lógico				
A	B	A Y B	A O B	No B
V	V	V	V	F
V	F	F	V	V
F	V	F	V	F
F	F	F	F	V

2. Elementos básicos de un algoritmo.

- Variable
  - Entidad que posee un **valor** y es conocida en un programa o algoritmo por un nombre (**identificador**)
  - El **valor** de una variable **puede cambiar** a lo largo del algoritmo
  - En un instante concreto sólo puede tener un valor
  - Todas las variables son de un determinado **tipo** y sólo pueden tomar valores de ese tipo

Identificador	Valor	Tipo
a	285	Entero
Dia_del_mes	'L'	Carácter
x4	5.3	Real
...	...	...

Ejemplos de variables

2. Elementos básicos de un algoritmo.

- Constante
  - Entidad que posee un **valor** y es conocida en un programa o algoritmo por un nombre (**identificador**)
  - El valor de una constante **NO** puede **cambiar** a lo largo del algoritmo
  - Inicialización de la constante
    - Acción por la que el identificador toma su primer y único valor
  - Todas las constantes son de un determinado **tipo** y sólo pueden ser inicializadas con valores de ese tipo
- Valores constantes
  - Valores que aparecen explícitamente en un algoritmo y no tienen identificador asociado

2. Elementos básicos de un algoritmo.

- Reglas de construcción de los identificadores
  - Un identificador debe comenzar por una letra
  - La primera letra puede ir seguida de:
    - Letras
    - Dígitos numéricos
    - Carácter '_'
  - No se permiten espacios en blanco

Identificadores	Válido
9kj	No
Dia_del_mes	Sí
Dia del mes	No
a+y	No
k9j	Sí

Ejemplos de identificadores de variables y constantes

2. Elementos básicos de un algoritmo.

- Expresiones
  - Combinación de:
    - Variables
    - Constantes
    - Valores constantes
    - Operadores
    - Paréntesis
    - Nombres de funciones especiales (*raíz cuadrada, valor absoluto, etc.*)
  - Toda expresión tiene en todo momento un **valor** constante que es el resultado de **evaluarla** de **izquierda a derecha**

Expresión	Valor
1.5 * 3/85	0.052
1.5 * x/85	Depende del valor de *x* en el momento de su evaluación
c = = (a + b)	Depende de los valores de *a*, *b* y *c* en el momento de su evaluación

Ejemplos de expresiones

2. Elementos básicos de un algoritmo.

- Asignación
  - Operación que permite darle a una variable un determinado valor
  - A una variable se le puede asignar
    - Un valor constante
    - El valor de otra variable
    - El valor de una constante
    - El resultado de evaluar una expresión
  - Los valores asignados deben ser del mismo tipo que la variable
  - Es una operación destructiva, cualquier valor que tuviese la variable se pierde y se reemplaza por el nuevo

Operación
Asignar a **A** el valor **5**
Asignar a **A** el valor **7/3**
Asignar a **A** el valor **x/3**

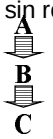
Ejemplos de asignaciones

2. Elementos básicos de un algoritmo.

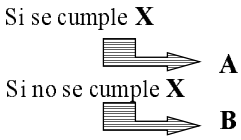
- Operaciones de entrada y salida
  - Se utilizan para intercambiar información con un medio externo
  - Entrada
    - Se le asigna a una variable un valor dado desde el exterior (p.ej. un teclado)
  - Salida
    - Se transfiere el valor de una variable a un dispositivo de salida (p.ej. la pantalla)

2. Elementos básicos de un algoritmo.

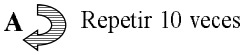
- Estructuras de control
  - Secuencial
    - Una acción sigue a otra sin romper la secuencia



  - Condicional
    - Se realiza una acción u otra dependiendo del resultado de la evaluación de una expresión lógica



  - Repetitiva, iterativa o en bucle
    - Se repiten un conjunto de acciones 0 o más veces



17

3. Representación de algoritmos

- Existen varios métodos para representar un algoritmo
  - Pseudolenguaje
  - Diagramas de flujo
- Pseudolenguaje
  - Es un lenguaje específico de descripción de algoritmos
  - La traducción de un algoritmo escrito en pseudolenguaje a un programa en un lenguaje de programación determinado es relativamente simple
- Herramientas de un pseudolenguaje para representar los elementos básicos de un algoritmo
  - Conjunto de palabras clave que proporcionan:
    - las estructuras de control
    - declaraciones de variables
    - características de modularidad
  - Sintaxis libre de un lenguaje natural que describe las características del proceso
  - Elementos para la definición y llamada a subprogramas

18

3. Representación de algoritmos

- Pseudolenguaje que usaremos en clase

```
ALGORITMO nombre
 declaraciones
INICIO
  acciones
FIN
```

Algoritmo

```
NOMBRE_CTE valor_cte
```
Declaración de una **constante**

```
tipo_vble nombre_vble
```
Declaración de una **variable**

```
LEER nombre_variable
```
Operación de **entrada**

```
ESCRIBIR nombre_variable
```
Operación de **salida**

3. Representación de algoritmos

- Pseudolenguaje que usaremos en clase

```
Acción 1
Acción 2
Acción 3
...
Acción n
```

Secuencia

```
SI
   condición
ENTONCES
   conjunto de
   acciones
   verdad
EN OTRO CASO
   conjunto de
   acciones falso
FINSI
```

```
SI
 condición
ENTONCES
   conjunto de
   acciones
   verdad
FINSI
```

```
CASO expresión SEA
 valores1: acciones
 valores2: acciones
 ...          ...
 valoresn: acciones
FINCASO
```

Estructuras de **selección** o condicionales

- Condición
  - Es una expresión (combinación válida de operadores, constantes y variables) en la que los operadores son:
    - relacionales: >, <, >=, <=, ==, !=
    - lógicas: Y, O, NO
  - El resultado de evaluar una condición puede ser:
    - Verdadero
    - Falso

Condición
a != 18
(a = = 5) O (b = = (a + 2))
NO ((a > b) Y (b > c))

Ejemplos de condiciones

3. Representación de algoritmos

- Pseudolenguaje que usaremos en clase

```
PARA (inicialización; condición; incremento_o_decremento)
      conjunto de acciones
FINPARA
```

```
MIENTRAS condición HACER
   conjunto de acciones
FINMIENTRAS
```

```
HACER
   conjunto de acciones
MIENTRAS condición
```

Estructuras de **iteración** o repetitivas

- Inicialización
  - Asignación de un valor a una variable que llamaremos variable de control
    - Ejemplos:
      x = 100;
      a = 1;
- Incremento o decremento
  - Expresión que indica en qué cantidad se incrementa o decrementa la variable de control en cada iteración
    - Ejemplos:
      x = x -2;
      a = a + 1;

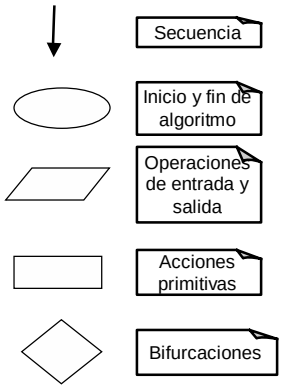
3. Representación de algoritmos

- Ejemplo de un algoritmo escrito con nuestro pseudolenguaje
  - Sumar los *n* (*n>0*) primeros números naturales

```
ALGORITMO SumaNaturales
VARIABLES:
      Naturales num, contador, suma
INICIO
  LEER num
  suma = 0
  contador = 1
  HACER
      suma = suma + contador
      contador = contador + 1
  MIENTRAS (contador != num + 1)
  ESCRIBIR "La suma resultante es"
  ESCRIBIR suma
FIN
```

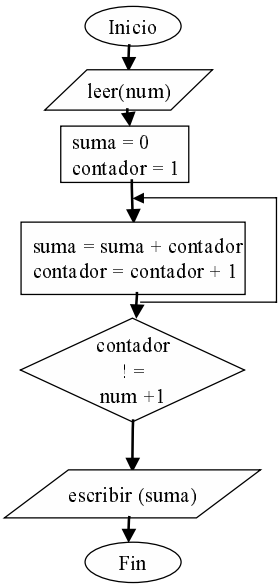
3. Representación de algoritmos

- Diagramas de flujo
  - Herramienta gráfica para la descripción de algoritmos
- Herramientas de los diagramas de flujo para para representar los elementos básicos de un algoritmo

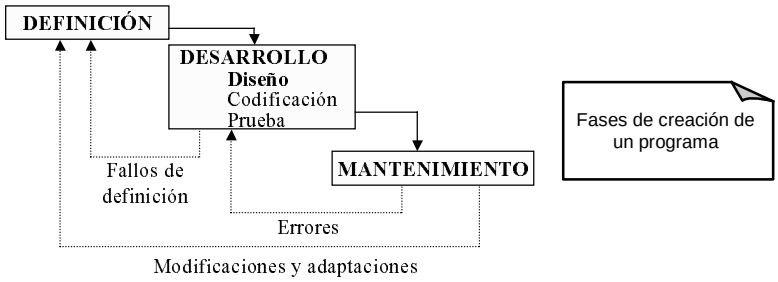


3. Representación de algoritmos

- Ejemplo de un algoritmo representado con un diagrama de flujo
  - Sumar los *n* (*n>0*) primeros números naturales



4. Metodología de diseño



- La fase de diseño es la fase de ***resolución del problema***, es aquí donde el programador debe ***diseñar un algoritmo***
- Es necesario emplear una ***metodología***, prescindir de ella puede acarrear problemas como:
  - Rigidez e inflexibilidad de los programas
  - Pérdida excesiva de tiempo en la corrección de errores
  - Documentación deficiente
  - Imposibilidad de reutilización de código

25

4. Metodología de diseño

- Un problema => muchos algoritmos para resolverlo
- ¿Cómo elegir el más adecuado? Basándonos en las siguientes características:
  - Legibilidad
  - Portabilidad
  - Modificabilidad
  - Eficiencia
  - Modularidad
  - Estructuración
- Programación estructurada
  - Conjunto de técnicas que aumentan la productividad de un programa, reduciendo el tiempo para:
    - Escribir
    - Verificar
    - Depurar
    - Mantener el programa
  - Utiliza un número limitado de estructuras de control que minimizan la complejidad de los problemas
  - Teorema de BOHM-JACOPINI
    - Cualquier programa, por complejo que sea, puede escribirse utilizando sólo tres estructuras de control:
      - Secuencial
      - Selectiva
      - Repetitiva

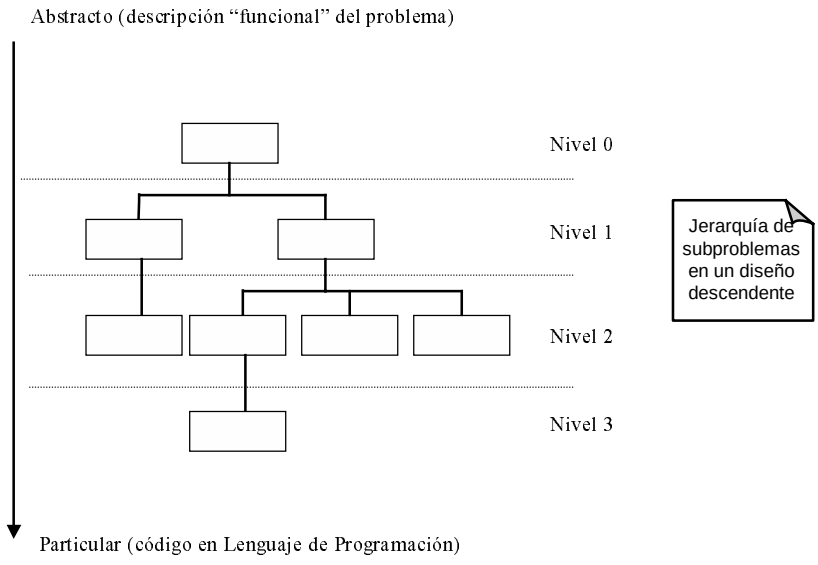
26

4. Metodología de diseño

- Diseño descendente (refinamientos sucesivos o programación modular)
  - Metodología de diseño que consiste en:
    - Dividir un problema en subproblemas manejables.
    - Cada subproblema se divide a su vez en otros subproblemas.
    - El proceso se repite hasta que ya no se pueda dividir más
    - La solución de todos los subproblemas constituye la solución global.
  - Se crea una estructura jerárquica de problemas con distintos niveles de refinamiento:
    - Nivel 0: descripción del problema
    - Nivel i: refinamientos sucesivos

27

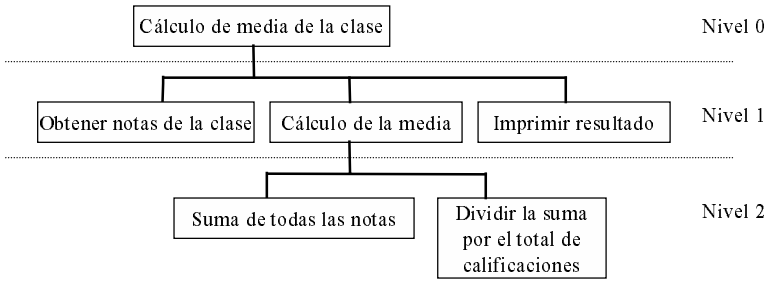
4. Metodología de diseño



28

4. Metodología de diseño

- Ejemplo de diseño descendente
  - Algoritmo que calcule la media de notas de una clase



29

5. Lenguajes de programación

- Lenguaje de programación
  - Conjunto de símbolos y reglas utilizados para construir un programa
- Clasificación de los lenguajes de programación
  - Atendiendo al nivel de proximidad al sistema utilizado por el procesador, de más bajo nivel a más alto:
    - Lenguaje máquina
    - Lenguaje ensamblador
    - Lenguajes de alto nivel

30

5. Lenguajes de programación

- Lenguaje máquina
  - Utiliza código binario
  - Cada procesador posee su propio lenguaje máquina
  - Una instrucción se compone de:
    - Código de operación
    - Operandos
  - Ventajas
    - Directamente “entendible” por el ordenador
    - Es muy eficiente
  - Inconvenientes
    - Es complicado trabajar con código binario
    - El programador debe conocer la arquitectura física del ordenador
    - Lenguaje dependiente de la máquina
    - No se pueden introducir comentarios
    - Conjunto de instrucciones muy reducido

5. Lenguajes de programación

- Lenguaje ensamblador
  - Cada instrucción en ensamblador se corresponde con una instrucción en lenguaje máquina a la que luego será traducida
  - Cada procesador posee su propio lenguaje ensamblador
  - NO es necesario que el programador conozca la arquitectura física del ordenador
  - Características:
    - Código de operación es una palabra con pocas letras
      - » ADD, MOV,...
    - Direcciones de memoria y operandos se pueden escribir de forma simbólica mediante identificadores
    - Se pueden incluir comentarios

5. Lenguajes de programación

- Lenguajes de alto nivel
  - Se aproxima al lenguaje natural
  - Cada instrucción de un lenguaje de alto nivel se corresponde con varias instrucciones máquina
  - No dependen de la arquitectura de la máquina
  - Son menos eficientes
  - FORTRAN, COBOL, C, C++, Pascal, etc.

5. Lenguajes de programación

- Traductores de lenguajes
  - Programas que traducen un programa escrito en un lenguaje de alto nivel a su correspondiente en lenguaje máquina

Programa fuente ⇨ *Código binario*

- Clasificación
  - Compiladores
    - Traducen un programa completo (*fuente*) a código binario (*objeto*)
    - El programa objeto se almacena en memoria y puede ser ejecutado sin necesidad de realizar otra vez la traducción
    - En el proceso de traducción se detectan errores de escritura en el programa fuente
  - Intérpretes
    - Traducen un programa (*fuente*) instrucción a instrucción
    - La ejecución del programa se realiza a la vez que la traducción => se lee una instrucción, se traduce y se ejecuta
    - Cada vez que se desea ejecutar el programa hay que traducirlo
    - La ejecución de un programa interpretado es más lenta que la de un programa compilado

Bibliografía

- Bibliografía principal
  - A. Prieto, A. Lloris y J.C. Torres. "Introducción a la Informática". (1ª ó 2ª Edición). McGraw-Hill, 1995.
  - Galindo, Sánchez, Yáñez, Escolano, Del Jesus, Aguilera, Rodríguez, Sánchez y Argudo. "Fundamentos Informáticos". Servicio de Publicaciones de la Universidad de Cádiz, 1996.
  - Tremblay, Bunt. "Introducción a la Ciencia de las Computadoras. Enfoque algorítmico". McGraw-Hill, 1988.
  - Peter Norton. "Introducción a la Computación". McGraw-Hill, 1995.
- Bibliografía adicional
  - Roger S. Walker. "Informática Básica". Anaya Multimedia.
  - Peter Bishop. "Conceptos de Informática". Anaya Multimedia.
  - Ll. Guilera Agüera. "Introducción a la Informática". Edunsa.
  - R. Peña Marí. "Diseño de Programas: Formalismo y Abstracción". Prentice Hall, 1998.

35