UNIVERSIDAD DE MURCIA

Introducción a la Programación

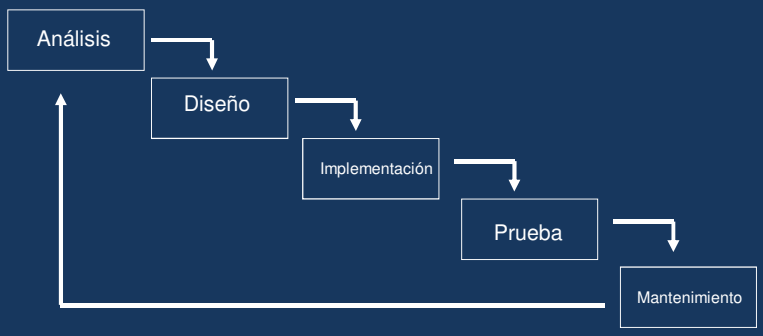
Grado en Ingeniería Informática

1. Construcción de programas

Bloque 1. Léxico y organización de un algoritmo
1. Construcción de programas

2

Etapas en el desarrollo de programas



3

Análisis

¿Qué quiero hacer?

¿Qué entradas se necesitan?

¿Cuáles son las salidas deseadas?

¿Qué debe hacer el programa para obtener la salida a partir de la entrada?

4

Análisis

¿Qué entradas se necesitan?
- En detalle:
- Exactamente cuántos datos
-¿De qué tipo cada uno de ellos?
-¿Cuáles con los rangos permitidos?
-¿Hay excepciones en la entrada?

¿Usarán el sistema expertos o usuarios sin formación?
¿Qué interface será necesario?
¿Qué casos son posibles?
¿Es necesaria documentación? ¿Cuál?
¿Qué mejoras se introducirán probablemente en el futuro?
¿Cuánto de rápido debe ser el sistema?
¿Deberá modificarse mucho en el futuro?
¿Tengo toda la información para realizar el diseño exacto?
¿Necesito hacer documentación técnica?

5

Análisis

Problema

Saber si un año cualquiera es bisiesto

Ejs: año 2000, 2100, 2022, 2024…

¿Cómo lo atacamos?

6

Análisis

Entrada: un año
un numero entero positivo

Salida: “SI” o “No”

Proceso: ¿Cómo se sabe si un año es bisiesto?

7

Análisis

Un año de cada 4 es bisiesto,
menos el último de cada siglo,
pero cada 400 años sí lo es.

¿2000? ¿2022?

¿2024?

¿2100?

8

Análisis

~~Un año de cada 4 es bisiesto, menos el último de cada siglo, pero cada 400 años sí lo es.~~

- Un año es bisiesto si es divisible por 4,
- excepto el último de cada siglo (aquel divisible por 100),
- salvo que este último de cada siglo sea divisible por 400

Año = 2100 →

2100 | 4
0 525

2100 | 100
0 21

2100 | 400
100 5

→  → No

9

Análisis

~~Un año de cada 4 es bisiesto, menos el último de cada siglo, pero cada 400 años sí lo es.~~

- Un año es bisiesto si es divisible por 4,
- excepto el último de cada siglo (aquel divisible por 100),
- salvo que este último de cada siglo sea divisible por 400

Año = 2022 →

2022 | 4
2 505

2022 | 100
22 20

2022 | 400
22 5

→  → No

10

Análisis

~~Un año de cada 4 es bisiesto, menos el último de cada siglo, pero cada 400 años sí lo es.~~

- Un año es bisiesto si es divisible por 4,
- excepto el último de cada siglo (aquel divisible por 100),
- salvo que este último de cada siglo sea divisible por 400

Año = 2000 →

$$2000 \mid 4 \quad 0 \;\; 500$$
$$2000 \mid 100 \quad 0 \;\; 20$$
$$2000 \mid 400 \quad 0 \;\; 5$$

→  → Sí

11

Análisis

~~Un año de cada 4 es bisiesto, menos el último de cada siglo, pero cada 400 años sí lo es.~~

- Un año es bisiesto si es divisible por 4,
- excepto el último de cada siglo (aquel divisible por 100),
- salvo que este último de cada siglo sea divisible por 400

Año = 2024 →

$$2024 \mid 4 \quad 0 \;\; 506$$
$$2024 \mid 100 \quad 24 \;\; 20$$
$$2024 \mid 400 \quad 24 \;\; 5$$

→  → Sí

12

Diseño

El “Análisis” establece ¿qué? quiere hacerse
El “Diseño” establece ¿Cómo? se hace
Con exactitud, paso a paso, en detalle.

Se necesita el “Algoritmo”

Algoritmo

Método para resolver un problema mediante una serie de pasos precisos, definidos y finitos.

13

Diseño

¿Y cómo organizamos eso en un algoritmo?

Un algoritmo tiene :
- Informaciones (datos)
- Acciones

Un algoritmo necesita un léxico para :
- Describir las acciones
- Describir las informaciones
- Organizar las acciones en el tiempo

Las acciones se organizan mediante :
- Secuenciación (instrucciones)
- Análisis de casos (condiciones)
- Iteraciones (repeticiones)

14

Diseño

- Un año es bisiesto si es divisible por 4,
- excepto el último de cada siglo (aquel divisible por 100),
- salvo que este último de cada siglo sea divisible por 400

```
Léxico
  año : entero
  resul : booleano
Algoritmo
  Leer (año);

  Si  ( (año MOD 4 = 0)
      y (año MOD 100 ≠ 0) )
      o (año MOD 400 = 0)
     Entonces resul ⟵ true
     Si_no resul ⟵ false;

  Escribir(resul);
Fin
```

15

Codificación

En un lenguaje concreto (C/C++)

```
#include <stdio.h>

main()
{
  int resul , anyo;

  printf ( " Introduce un año : \n " );
  scanf ( "%d", &anyo); fflush(stdin);

  if  ( ( (anyo % 4 == 0)  && (anyo % 100 != 0) ) || (anyo % 400 == 0) )
     resul = 1; else resul = 0;

  printf ("Año %d bisiesto : ",anyo);
  if (resul) printf ("SI \n"); else printf ("NO \n");

  system("pause");
}
```

16

Codificación

En un lenguaje concreto (Pascal)

```
Program bisiesto;
Var
  anyo : integer;
  resul : boolean;

Begin
  Write('Introduce un año : ');
  Readln(anyo);

  If  ( (anyo MOD 4 = 0)  and (anyo MOD 100 ≠ 0) ) or (anyo MOD 400 = 0)
     Then resul := true
     Else resul := false;

  write ('Año ', anyo, 'bisiesto :' );
  if resul then writeln ('SI')
           else writeln ('NO');
End.
```

17

Prueba

Fase de pruebas: verificación y depuración del programa

Se utilizan un conjunto amplio y variado de datos de entrada, para buscar posibles errores:

- Valores normales
- Valores extremos : en los límites de los dominios
- Valores especiales : que puedan producir problemas (Ej: a/b si b=0)

Errores de compilación o interpretación: suelen ser errores de sintaxis
Errores de ejecución: ej. Divisiones por cero, desbordamiento de rangos, etc.
Errores lógicos: el programa no hace lo que debiera, está mal programado.

18

Prueba

Ej: bisiesto

Normales: 2020, 400,
extremos: 50.000 ¿?
Especiales: 0, -400 ¿?

19

Mantenimiento

Documentación:
Escritura de las diferentes fases del ciclo de vida, esencialmente el análisis, diseño, y codificación, unidos a manuales de usuario y de referencia, así como normas para el mantenimiento.

Mantenimiento:
El programa se actualiza y modifica cada vez que sea necesario, de modo que se cumplan las necesidades cambiantes del usuario

A medida que se utiliza el programa, se anotan pequeños errores y posibles mejoras para introducirlos en la siguiente versión.

20

Compiladores, intérpretes y más

21

Lenguajes compilados

El código fuente es el programa escrito por el programador en un lenguaje de alto nivel.
El código máquina es el programa en un formato directamente entendible por la CPU.
El compilador convierte un programa de alto nivel en un programa en código máquina.
Se ejecuta el código máquina.
Se detectan muchos errores antes de ejecutar
El código es rápido y más fácil de depurar

22

Lenguajes interpretados

1. El intérprete lee la primera instrucción del programa fuente
2. Comprueba si es correcta
3. Convierte la instrucción a código máquina
4. Ejecuta el código máquina
5. Lee la siguiente instrucción
6. Vuelve al paso 2

- Se puede ir ejecutando según llega el código (ej: html)
- Es más lento
- El código máquina es menos óptimo.
- Todos los errores son en tiempo de ejecución

23

Lenguajes Semi-Compilados

1. El código fuente se compila a un lenguaje cercano al código máquina, un código intermedio (ej: bytecode de java).
2. Cada plataforma (win, lin, ios, etc.) tiene una “máquina virtual” que entiende ese código.
3. La máquina virtual ejecuta ese código (semi-interpretando el código intermedio)

- El código intermedio es válido para todas las plataformas
- Más lento que compilado pero más rápido que interpretado (generalmente)
- El código es menos óptimo.
- Java, Python, JavaScript…

24

25