Pseudocódigo y Pruebas de Escritorio

Yuliza Estupiñan
Universidad Distrital Francisco José de Caldas

Septiembre 2025

Índice

1. Paso 1 — Definición de pseudocódigo

1.1. ¿Qué es?

El pseudocódigo es la representación escrita de un algoritmo: un texto que describe, paso a paso y de forma legible, cómo resolver un problema antes de traducirlo a un lenguaje de programación. Está pensado para ser humano-lectura y facilita el diseño y la verificación del algoritmo antes de codificarlo.

1.2. Características esenciales

- Tiene un alcance delimitado por `INICIO` y `FIN`.
- Palabras reservadas (como `LEER`, `ESCRIBIR`, `SI`, `MIENTRAS`) se escriben en mayúsculas.
- Usa sangrías para mejorar la legibilidad y marcar bloques.
- Las variables se declaran con la sintaxis `<nombre>:<TIPO>` (por ejemplo `contador: ENTERO`, `nombre: CADENA`).
- Soporta tipos simples (`ENTERO, REAL, BOOLEANO, CARACTER, CADENA`), arreglos y registros (con `REG ... FIN REG`).
- Representa estructuras:
  - Secuencial
  - Condicional (`SI ... ENTONCES ... DE LO CONTRARIO ... FIN DEL SI`)
  - Repetitiva (`MIENTRAS ... FIN MIENTRAS`, `HACER ... MIENTRAS`)

1.3. Ejemplo ilustrativo

```
INICIO
x: REAL
ESCRIBIR "Ingrese un número (no 0):"
LEER x
SI x > 0 ENTONCES
    ESCRIBIR "El número es positivo"
DE LO CONTRARIO
    ESCRIBIR "El número es negativo"
FIN DEL SI
FIN
```

2. Paso 2 — Pruebas de escritorio

2.1. ¿Qué es una prueba de escritorio?

Una prueba de escritorio es una tabla donde se simula la ejecución del algoritmo, anotando valor por valor cómo cambian las variables en cada paso o iteración. Sirve para verificar que el algoritmo produce las salidas esperadas para casos representativos y para detectar errores lógicos antes de programar.

2.2. Cómo construirla

1. Seleccionar casos de prueba: típicos, límite y especiales.
2. Listar variables relevantes: columnas para cada variable y la salida.
3. Simular paso a paso: ejecutar mentalmente y registrar valores.
4. Verificar resultado final: comparar salida obtenida vs. salida esperada.

2.3. Importancia

- Detecta errores lógicos antes de programar (ahorro de tiempo).
- Permite ver claramente el flujo de datos y comportamiento de estructuras.
- Sirve como justificación documentada de la corrección del algoritmo.

3. Paso 3 — Ejemplos concretos

3.1. Ejemplo A — Determinar si un número es positivo o negativo

Entrada: num (real), con restricción: distinto de 0.
Salida: mensaje indicando "positivo" o "negativo".

```
INICIO
num: REAL
ESCRIBIR "Ingrese un número distinto de 0:"
LEER num
SI num = 0 ENTONCES
   ESCRIBIR "Número inválido, reingrese"
   LEER num
FIN DEL SI
SI num > 0 ENTONCES
   ESCRIBIR "El número es positivo"
DE LO CONTRARIO
   ESCRIBIR "El número es negativo"
FIN DEL SI
FIN
```

Prueba de escritorio:

Iteración	num	Observación / Salida
1	5	El número es positivo
1	-29	El número es negativo
1	0	Número inválido, reingrese
2	100	El número es positivo

3.2. Ejemplo B — Obtener el mayor de dos números diferentes

Entrada: num1, num2 (reales), con restricción: num1 $\neq$ num2.
Salida: el número mayor.

```
INICIO
num1, num2: REAL
ESCRIBIR "Ingrese num1:"
LEER num1
REPETIR
   ESCRIBIR "Ingrese num2 (diferente de num1):"
   LEER num2
HASTA num2 <> num1
SI num1 > num2 ENTONCES
   ESCRIBIR "El primer número es el mayor"
DE LO CONTRARIO
   ESCRIBIR "El segundo número es el mayor"
FIN DEL SI
FIN
```

Prueba de escritorio:

Iteración	num1	num2	Salida
1	5	6	El segundo número es el mayor
1	-99	-222.2	El primer número es el mayor
1	15	15	Rechazado — pedir num2 otra vez
4	15	10	El primer número es el mayor

3.3. Ejemplo C — Calcular factorial de un entero

Entrada: n (entero, $n \geq 0$).
Salida: factorial (n!).

```
INICIO
n: ENTERO
factorial: ENTERO
contador: ENTERO
ESCRIBIR "Ingrese un entero >= 0:"
LEER n
SI n < 0 ENTONCES
   ESCRIBIR "Valor inválido, reingrese"
   LEER n
FIN DEL SI
factorial := 1
contador := 2
MIENTRAS contador <= n ENTONCES
   factorial := factorial * contador
   contador := contador + 1
FIN MIENTRAS
```

```
ESCRIBIR "El factorial es ", factorial
FIN
```

Prueba de escritorio (para n=5):

Iteración	n	factorial	contador	Salida
init	5	1	2	-
1	5	2	3	-
2	5	6	4	-
3	5	24	5	-
4	5	120	6	El factorial es 120

3.4. Ejemplo D — Estructura HACER-MIENTRAS

```
INICIO
valorInicial, valorFinal: ENTERO
valorInicial := 0
valorFinal := 3
HACER
   ESCRIBIR valorInicial
   valorInicial := valorInicial + 1
MIENTRAS valorInicial < valorFinal
FIN
```

Prueba de escritorio:

Instrucción ejecutada	valorInicial	valorFinal	Salida
valorInicial := 0	0	3	-
ESCRIBIR valorInicial (1ra vez)	0	3	0
valorInicial := valorInicial + 1	1	3	-
ESCRIBIR valorInicial (2da vez)	1	3	1
valorInicial := valorInicial + 1	2	3	-
ESCRIBIR valorInicial (3ra vez)	2	3	2
valorInicial := valorInicial + 1	3	3	-
Condición valorInicial ¡valorFinal → FALSA	3	3	-

4. Ejemplo 1: Determinar si un número es positivo o negativo

4.1. Análisis

Este algoritmo empieza con la lectura de un número real y verifica si es mayor o menor que cero. Si el valor ingresado es cero, el algoritmo pide que se vuelva a ingresar el número, ya que la restricción es que no puede ser cero. Después, con una estructura condicional `SI... ENTONCES... DE LO CONTRARIO`, se decide la salida: un mensaje que indica si el número es positivo o negativo.

Estructura: condicional simple con una validación adicional para el caso inválido.
Eficiencia: alta, ya que solo requiere una o dos comparaciones como máximo.

Posibles mejoras: se podría generalizar para que acepte cero y lo clasifique como "nulo", en lugar de pedir reingreso.

4.2. Comentario

Con este ejemplo entendí la importancia de validar los datos de entrada. El algoritmo no solo resuelve el problema, sino que también enseña a controlar errores de usuario. Esto se relaciona directamente con lo visto en clase sobre la necesidad de establecer restricciones y condiciones de los problemas antes de programar.

5. Ejemplo 2: Obtener el mayor de dos números diferentes

5.1. Análisis

Este algoritmo recibe dos números reales como entrada, pero con la condición de que no pueden ser iguales. Si se ingresan iguales, se pide que uno de ellos se vuelva a introducir hasta que la restricción se cumpla. Luego, mediante una condición `SI... DE LO CONTRARIO`, se compara cuál es mayor y se muestra como salida.

Estructura: validación con ciclo de repetición y condicional para comparar.
Eficiencia: constante, ya que siempre se realiza como máximo una comparación, más la validación de la restricción.
Posibles mejoras: se podría modificar para que acepte números iguales y muestre un mensaje indicando que son iguales, en lugar de forzar siempre un reingreso.

5.2. Comentario

Este ejemplo me ayudó a comprender mejor cómo funcionan las estructuras de repetición combinadas con condicionales. Vi cómo se puede controlar la validez de los datos con un bucle hasta que cumplan las condiciones. Esto refuerza lo que hemos hablado en clase sobre la importancia de validar las entradas antes de procesar un algoritmo.

6. Ejemplo 3: Calcular el factorial de un número

6.1. Análisis

El algoritmo solicita un número entero no negativo y calcula su factorial mediante un ciclo `MIENTRAS`. Se inicializa la variable acumuladora factorial en 1 y un contador en 2, y en cada iteración el valor de factorial se multiplica por el contador hasta alcanzar el número ingresado. Finalmente, se imprime el resultado.

Estructura: inicialización de variables, ciclo iterativo con multiplicación acumulativa y salida del resultado.
Eficiencia: depende del valor de entrada n. El tiempo de ejecución es $O(n)$, ya que necesita tantas iteraciones como el número ingresado.
Posibles mejoras: para números muy grandes se podrían usar estructuras recursivas o técnicas más avanzadas de cálculo, aunque a nivel básico este enfoque es suficiente y claro.

6.2. Comentario

Este ejemplo me pareció muy interesante porque me permitió ver cómo usar variables acumuladoras dentro de un bucle. Aprendí cómo la lógica iterativa puede resolver problemas matemáticos de forma sencilla. Además, se conecta con lo discutido en clase sobre cómo los ciclos son fundamentales para reducir tareas repetitivas y sistemáticas.