Principios de Programación

Bucles `while` y `do-while`

Segunda parte del bloque de iteración: control por condición, diferencia entre `while` y `do-while`, y problemas donde la cantidad de iteraciones se conoce durante la ejecución.

Del recorrido fijo al corte por condición

Base previa

- Con `for` la inicialización, la condición y la actualización quedan visibles desde el encabezado
- Se usó para contar, sumar y recorrer valores con cantidad prevista
- Funcionó bien cuando el recorrido estaba definido desde el comienzo

Definición

`while` repite un bloque mientras una condición sea verdadera. La condición se evalúa **antes** de cada iteración.

Casos habituales

- Leer datos hasta encontrar un valor de corte
- Repetir una validación hasta que la entrada sea correcta
- Procesar mientras una condición del problema siga siendo válida

Consecuencia

El cuerpo puede ejecutarse muchas veces o ninguna. Todo depende del valor inicial de la condición.

Bucle `while`: estructura general

SINTAXIS

```
while (condicion) {
    // Instrucciones a repetir
}
```

Lectura

- Se evalúa `condicion`
- Si es verdadero, se ejecuta el bloque
- Al terminar el bloque, se vuelve a evaluar
- Si es falso, el bucle finaliza

1 · Inicializar

Preparar el estado inicial (condicion) **antes** del bucle.

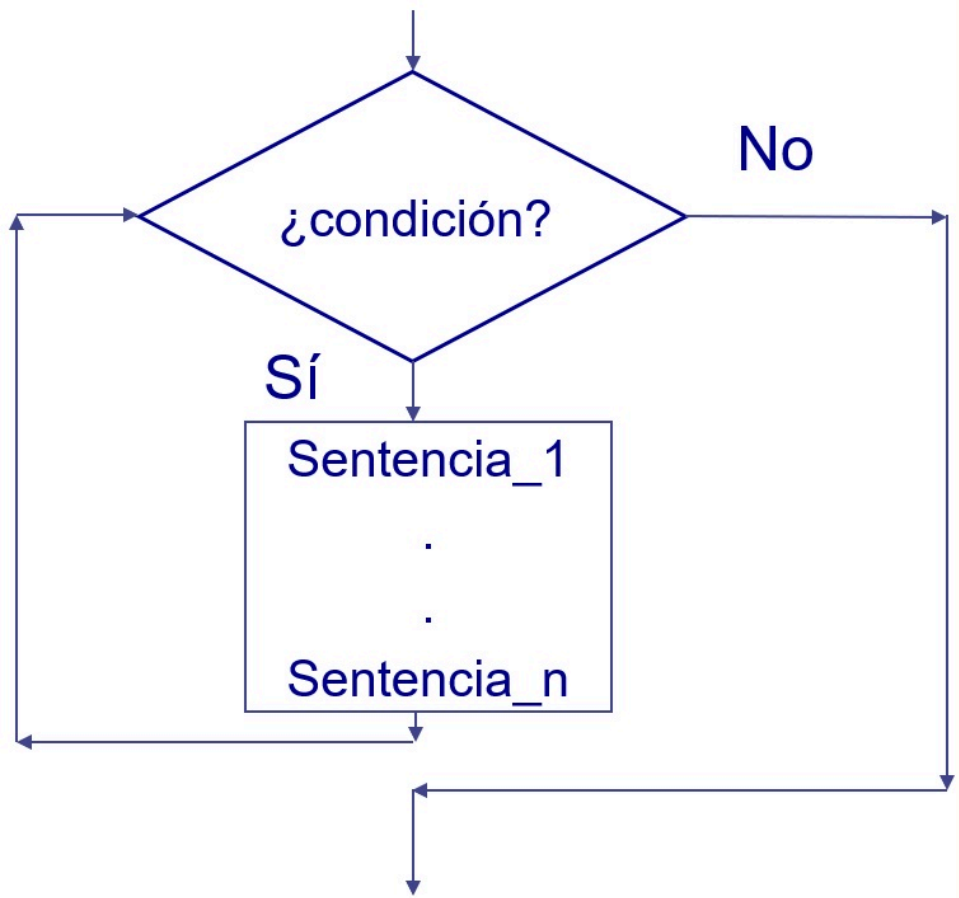
2 · Evaluar

Verificar si todavía corresponde repetir.

3 · Actualizar

Modificar las variables necesarias para que la condición pueda cambiar.

Cómo se evalúa un `while`



Lectura del ciclo

- Primero se evalúa la condición
- Si es falso, el cuerpo no se ejecuta
- Si es verdadero, se ejecuta el bloque
- Al terminar, se vuelve a comprobar la condición

Validación con reintento usando `while`

CÓDIGO EN C

```c
int nota;

printf("Ingrese nota (0 a 100): ");
scanf("%d", &nota);

while (nota < 0 || nota > 100) {
    printf("Valor fuera de rango\n");
    printf("Ingrese nota (0 a 100): ");
    scanf("%d", &nota);
}
```

Primero se lee un valor. Si no cumple la condición válida, el programa vuelve a pedirlo.

Cuándo conviene

Este patrón sirve cuando el reintento depende de detectar un valor inválido antes de continuar con el resto del programa.

Conteo simple con `while`

PSEWEB

```
Algoritmo  ImprimirDel1Al5
   Definir  contador  Numero
   contador  =  1
   Mientras  contador  <=  5
      Escribir  contador
      contador  =  contador  +  1
   FinMientras
FinAlgoritmo
```

IMPLEMENTACIÓN EN C

```c
#include <stdio.h>

int main() {
    int contador = 1;

    while (contador <= 5) {
        printf("%d\n", contador);
        contador = contador + 1;
    }

    return 0;
}
```

Observación

La variable `contador` cumple dos funciones: permite decidir si el bucle continúa y registra en qué iteración se encuentra el programa.

Traza: contador de `1` a `5`

EJECUCIÓN

Paso	contador	Salida
Inicio	1	–
Condición 1 <= 5	1	–
Imprimir	1	1
Actualizar	2	–
Condición 2 <= 5	2	–
Imprimir	2	2
..	..	..
Actualizar	6	–
Condición 6 <= 5	6	Fin

Lectura de la traza

- La condición se consulta antes de cada vuelta
- La salida aparece solo cuando el bloque se ejecuta
- El bucle termina cuando `contador` pasa a ser `6`

Resultado

El cuerpo se ejecuta exactamente cinco veces.

Acumulación con `while`: suma de `N` números

CÓDIGO

```c
#include <stdio.h>

int main() {
    int cantidad, contador = 1, numero, suma = 0;

    printf("Cantidad de valores: ");
    scanf("%d", &cantidad);

    while (contador <= cantidad) {
        printf("Ingrese valor %d: ", contador);
        scanf("%d", &numero);
        suma = suma + numero;
        contador = contador + 1;
    }

    printf("Suma total: %d\n", suma);
    return 0;
}
```

Acumulación con `while`: suma de `N` números

Variables

- `contador`: controla cuántos valores faltan procesar
- `numero`: guarda el valor leído en cada iteración
- `suma`: acumula el total parcial

Observación

En este problema la cantidad de vueltas queda determinada después de leer `cantidad`. Por eso `while` se apoya en un dato obtenido durante la ejecución.

Bucle infinito

CÓDIGO CON ERROR

```c
int contador = 1;

while (contador <= 5) {
    printf("%d\n", contador);
}
```

MENSAJE / PROBLEMA

La condición depende de `contador`, pero `contador` nunca cambia.

Resultado: la condición sigue siendo verdadera y el bucle no termina.

CORRECCIÓN

Agregar una actualización que modifique la variable de control.

```c
contador = contador + 1;
```

Error frecuente: condición mal formulada

CONDICIÓN INCORRECTA

```c
int contador = 1;

while (contador >= 1) {
    printf("%d\n", contador);
    contador = contador + 1;
}
```

CORRECCIÓN

```c
int contador = 1;

while (contador <= 5) {
    printf("%d\n", contador);
    contador = contador + 1;
}
```

Observación

La actualización por sí sola no garantiza que el bucle termine. También importa que la condición en algun momento se vuelva falsa.

Bucle `do-while`

Bucle `do-while`: estructura general

SINTAXIS

```
do {
    // Instrucciones a repetir
} while (condicion);
```

Diferencia esencial

- El bloque se ejecuta primero
- La condición se evalúa al final
- Por eso el cuerpo se ejecuta **al menos una vez**

1 · Ejecutar

Realizar el bloque una primera vez.

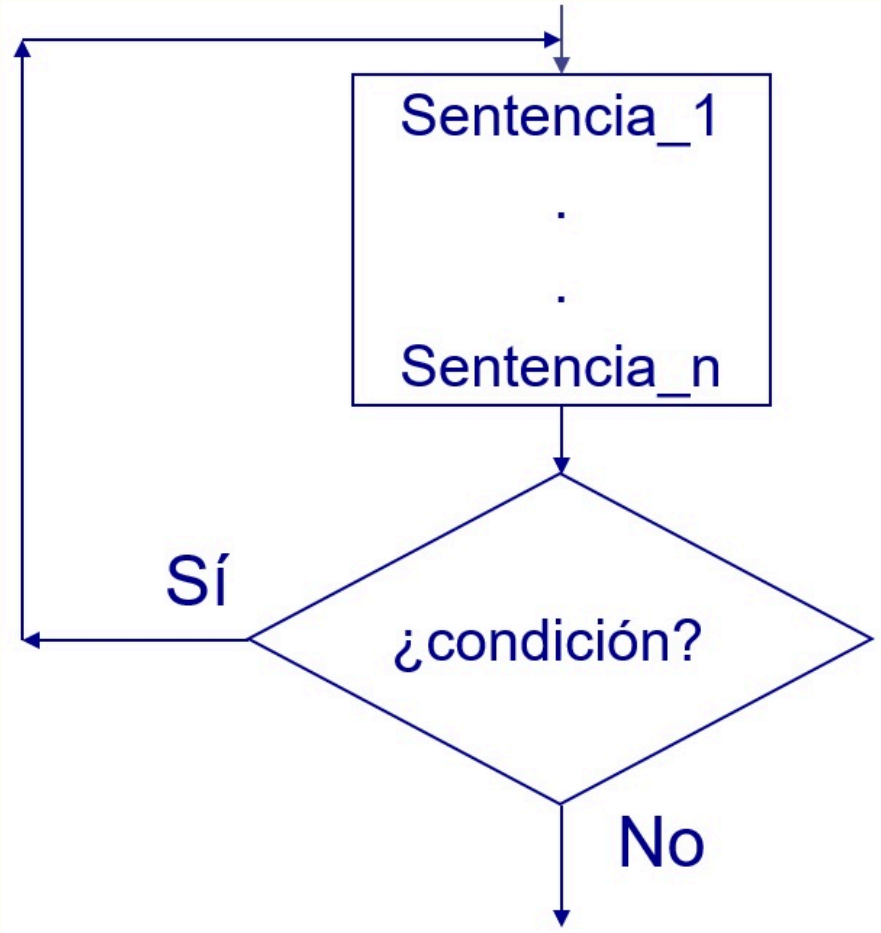
2 · Evaluar

Consultar si corresponde repetir.

3 · Repetir o salir

Volver al comienzo si la condición es verdadero.

Cómo se evalúa un `do-while`



Lectura del ciclo

- El cuerpo se ejecuta antes de evaluar la condición
- Recién al final se decide si corresponde repetir
- Si la condición es verdadero, vuelve al comienzo
- Si resulta falsa, el ciclo termina

Idea clave

`do-while` garantiza una primera ejecución, por eso sirve cuando la acción inicial no puede omitirse.

Menú simple con `do-while`

CÓDIGO

```c
#include <stdio.h>

int main() {
    int opcion;

    do {
        printf("Menu\n");
        printf("1. Opcion A\n");
        printf("2. Opcion B\n");
        printf("3. Salir\n");
        printf("Ingrese opcion: ");
        scanf("%d", &opcion);

        if (opcion == 1) {
            printf("Eligio la opcion A\n");
        } else if (opcion == 2) {
            printf("Eligio la opcion B\n");
        } else if (opcion != 3) {
            printf("Opcion invalida\n");
        }
    } while (opcion != 3);

    printf("Fin del programa\n");
    return 0;
}
```

Menú simple con `do-while`

Razón de uso

El menú debe mostrarse por lo menos una vez, porque el usuario recién puede elegir una opción después de verlo.

CONDICIÓN DE CORTE

El ciclo continúa mientras `opcion != 3`.

`while` y `do-while`

while

```
while (condicion) {
    // Puede ejecutarse 0, 1 o mas veces
}
```

do-while

```
do {
    // Se ejecuta al menos 1 vez
} while (condicion);
```

Usar `while`

Cuando la repetición depende de una condición que puede ser falsa desde el principio.

Usar `do-while`

Cuando hace falta ejecutar una primera acción antes de decidir si se repite.

Mismo problema: `while` o `do-while`

VALIDACIÓN CON `while`

```c
scanf("%d", &opcion);
while (opcion < 1 || opcion > 3) {
    printf("Opcion invalida\n");
    scanf("%d", &opcion);
}
```

VALIDACIÓN CON `do-while`

```c
do {
    scanf("%d", &opcion);
    if (opcion < 1 || opcion > 3) {
        printf("Opcion invalida\n");
    }
} while (opcion < 1 || opcion > 3);
```

Usar `while`

Cuando ya existe una primera lectura antes del ciclo y solo hace falta repetir si hubo error.

Usar `do-while`

Cuando conviene reunir lectura y validación dentro del mismo bloque porque la primera ejecución no puede omitirse.

Control adicional: `break` y `continue`

`break`

- Sale inmediatamente del bucle más interno
- Se usa cuando ya no tiene sentido seguir iterando

`continue`

- Omite el resto del cuerpo actual
- Pasa directamente a la siguiente iteración

Alcance

`break` y `continue` pueden usarse con `for`, `while` o `do-while`. Conviene usarlos solo cuando el caso especial queda más claro, no más confuso.

Ejemplo con `break`

FRAGMENTO RELEVANTE

```c
for (int i = 1; i <= cantidad; i++) {
    scanf("%d", &numero);

    if (numero < 0) {
        printf("Negativo: %d\n", numero);
        break;
    }
}
```

Contexto

El programa lee una secuencia de enteros y debe detenerse en el primer valor negativo.

Lectura

Cuando aparece el primer valor negativo, el programa abandona el bucle porque el objetivo de búsqueda ya quedó resuelto.

Ejemplo con `continue`

CÓDIGO EN C

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        if (i % 2 == 0) {
            continue;
        }

        printf("%d\n", i);
    }

    return 0;
}
```

SALIDA

```
1
3
5
7
9
```

Lectura

Cuando `i` es par, el `printf` no se ejecuta y el bucle avanza al siguiente valor.

Ejercicio 1

Leer enteros hasta ingresar `0`. Al finalizar, mostrar el promedio de los valores leídos sin contar el `0`.

Ejercicio 2

Leer enteros hasta ingresar `0`. Contar cuántos son positivos y cuántos son negativos.

Ejercicio 3

Implementar un menú con opciones de suma, resta, multiplicación y salida. El menú debe repetirse hasta elegir salir.