Principios de Programación

Condiciones y Selección en C

Uso de `if`, `else`, `else if`, operadores lógicos y `switch` para seleccionar acciones según una condición.

Recordando: ¿Qué se trabajó la semana pasada?

Base previa

- Ya declaramos variables y elegimos tipos
- Ya usamos operadores aritméticos
- Ya escribimos programas que leen, procesan y muestran resultados
- Ya vimos que C exige una sintaxis precisa

- Cómo hacer que el programa tome decisiones
- Cómo ejecutar caminos distintos según una condición
- Cuándo usar `if`, `if-else`, `else if` y `switch`
- Qué operadores permiten comparar y combinar valores

PRIMERA DECISIÓN EN C

```c
if (edad >= 18) {
    printf("Es mayor de edad\n");
} else {
    printf("Es menor de edad\n");
}
```

Definición

Las condiciones permiten que el programa ejecute distintos bloques según el resultado de una expresión lógica.

Definición

La selección permite que el programa **evalúe una condición** y elija entre dos o más caminos posibles.

1 · Evaluar

Se analiza una expresión que resulta `1` o `0`. (`true` o `false`)

2 · Elegir

Si la condición es verdadera, se entra por un bloque. Si es falsa, por otro.

3 · Continuar

Después de ejecutar el camino elegido, el programa sigue.

Analogía

Semáforo

- Si está en verde -> avanzar
- Si está en rojo -> detenerse
- La acción depende del estado observado/evaluado

Pseudocódigo

```
si condición entonces
  // hacer algo
sino
  // hacer otra cosa
fin si
```

Ejemplo de Selección

Situación

```
Si tengo suficiente dinero:
  Comprar el producto
Si no:
  No comprar
```

VERSIÓN EN C

```c
if (dinero >= precio) {
    printf("Comprar\n");
} else {
    printf("No comprar\n");
}
```

Uso

Las estructuras de selección permiten validar datos y ejecutar acciones distintas según el caso.

Operadores de Comparación

Qué hacen

- Comparan dos valores
- Producen `1` si la comparación es verdadera
- Producen `0` si la comparación es falsa
- Son la base de casi todas las condiciones

REFERENCIA RÁPIDA

Operador	Significado	Ejemplo	Resultado
`==`	Igual a	`5 == 3`	`0` (false)
`!=`	Diferente de	`5 != 3`	`1` (true)
`<`	Menor que	`3 < 5`	`1` (true)
`>`	Mayor que	`3 > 5`	`0` (false)
`<=`	Menor o igual	`3 <= 5`	`1` (true)
`>=`	Mayor o igual	`3 >= 5`	`0` (false)

Ejemplos de Operadores de Comparación

LECTURA DE UNA MISMA VARIABLE

```
int edad = 20;

edad == 18;  // Falso (0)
edad != 18;  // Verdadero (1)
edad < 18;   // Falso (0)
edad > 18;   // Verdadero (1)
edad <= 20;  // Verdadero (1)
edad >= 18;  // Verdadero (1)
```

Error clásico

CÓDIGO CON ERROR

```
if (edad = 18) {
    // ...
}
```

MENSAJE / PROBLEMA

= asigna. No compara. La condición termina modificando la variable en lugar de verificarla.

CORRECCIÓN

Usar == cuando la pregunta es “¿vale exactamente lo mismo?”:

```
if (edad == 18) { ... }
```

ENTEROS

```
int a = 10;
int b = 5;

a == b;   // Falso (0)
a != b;   // Verdadero (1)
a < b;    // Falso (0)
a > b;    // Verdadero (1)
a <= 10;  // Verdadero (1)
a >= 5;   // Verdadero (1)
```

DECIMALES

```
float precio = 99.99;
float descuento = 50.0;

precio > descuento;  // Verdadero (1)
precio == 100.0;     // Falso (0)
precio <= 100.0;     // Verdadero (1)
```

Nota

La comparación se escribe igual. Lo importante es identificar con precisión qué pregunta está haciendo la condición.

CARACTERES

```c
char letra = 'A';

letra == 'A';  // Verdadero (1)
letra == 'B';  // Falso (0)
letra < 'Z';   // Verdadero (1)
```

USO DIRECTO EN if

```c
int edad = 20;

if (edad >= 18) {
    printf("Es mayor de edad\n");
}

if (edad < 18) {
    printf("Es menor de edad\n");
}
```

Nota

Los operadores de comparación siempre devuelven `1` o `0`. Eso es exactamente lo que una estructura condicional necesita para decidir.

Estructura if - Decisión Simple

SINTAXIS

```c
if (condición) {
    // instrucciones
}
```

Cómo se lee

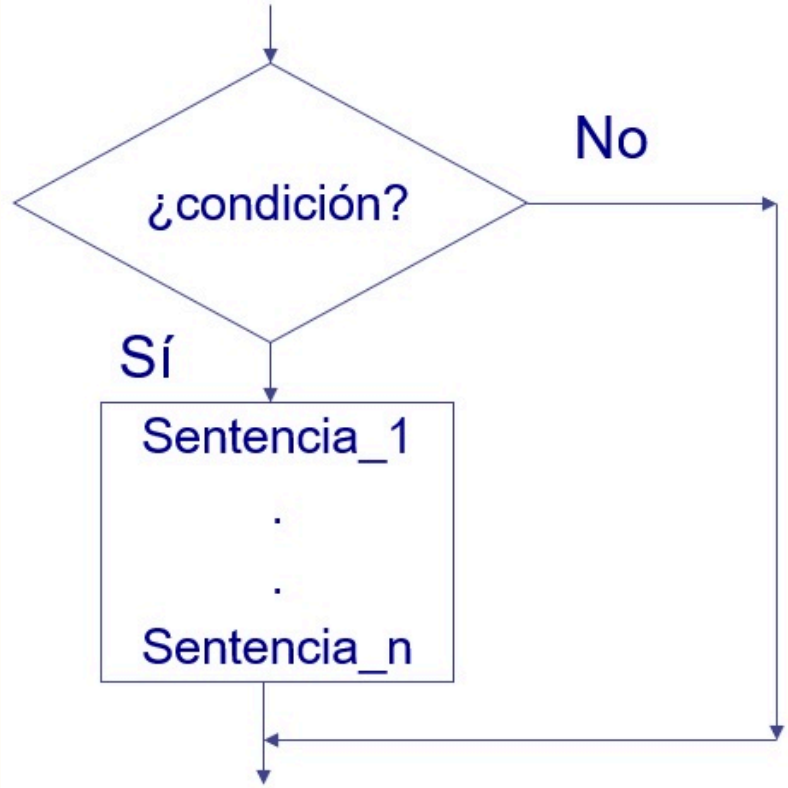
1. Se evalúa la condición
2. Si da `1`, se ejecuta el bloque
3. Si da `0`, el bloque se salta

EJEMPLO MÍNIMO

```c
int edad = 20;

if (edad >= 18) {
    printf("Es mayor de edad\n");
}
```

Cómo se evalúa un `if`



Lectura del flujo

- Primero se evalúa la condición
- Si da verdadero, se ejecuta el bloque
- Si da falso, el bloque se omite
- Después el programa sigue

Ejemplo Completo: Estructura if

PROGRAMA COMPLETO

```c
#include <stdio.h>

int main() {
    int numero;

    printf("Ingrese un número: ");
    scanf("%d", &numero);

    if (numero > 0) {
        printf("El número es positivo\n");
    }

    return 0;
}
```

SI INGRESA 5

```
Ingrese un número: 5
El número es positivo
```

SI INGRESA –3

```
Ingrese un número: -3
(No muestra nada)
```

SINTAXIS

```c
if (condición) {
    // camino verdadero
} else {
    // camino falso
}
```

Funcionamiento

- Si la condición es verdadera, entra al `if`
- Si es falsa, entra al `else`
- Siempre se ejecuta un solo camino

Uso

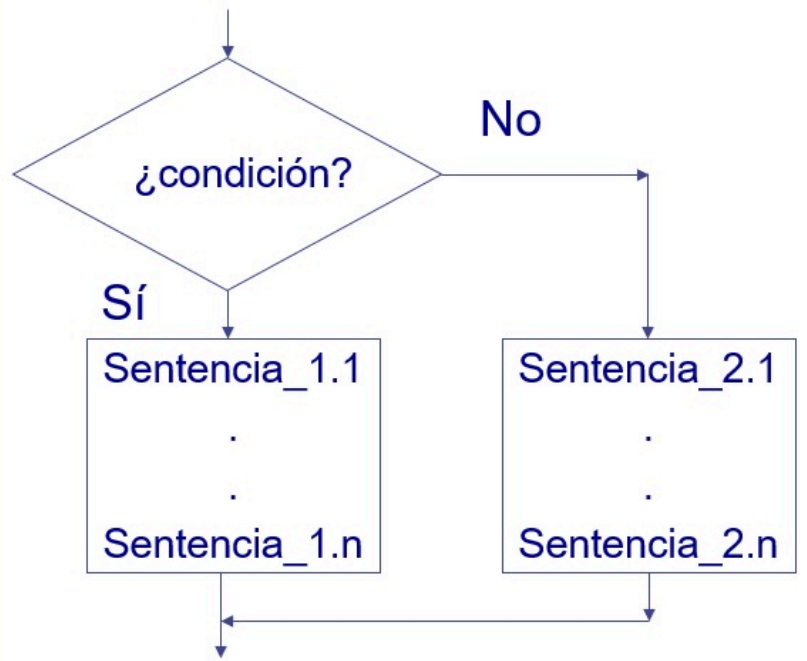
Se usa cuando hace falta definir qué ocurre tanto en el caso verdadero como en el falso.

EJEMPLO

```c
int edad = 20;

if (edad >= 18) {
    printf("Es mayor de edad\n");
} else {
    printf("Es menor de edad\n");
}
```

Cómo se evalúa un `if-else`



Lectura del flujo

- Se evalúa una sola condición
- Si da verdadero, entra al bloque `if`
- Si da falso, entra al bloque `else`
- Nunca se ejecutan los dos caminos

Ejemplo Completo: if-else

PROGRAMA COMPLETO

```c
#include <stdio.h>

int main() {
    int numero;

    printf("Ingrese un número: ");
    scanf("%d", &numero);

    if (numero > 0) {
        printf("El número es positivo\n");
    } else {
        printf("El número no es positivo\n");
    }

    return 0;
}
```

ENTRADA 5

```
Ingrese un número: 5
El número es positivo
```

ENTRADA –3

```
Ingrese un número: -3
El número no es positivo
```

Traza

La traza muestra paso a paso qué valor tiene la variable, qué condición se evalúa y qué salida aparece.

TRAZA · ENTRADA 5

Paso	numero	Salida
Inicio	—	—
`scanf`	5	Ingrese un número:
`numero > 0`	5	La condición da verdadero
`printf del if`	5	El número es positivo

Con entrada -3

Si `numero = -3`, la condición resulta falsa, el programa entra al `else` y la salida pasa a ser: `El número no es positivo`.

Correspondencias

- `Numero` -> `int` en este ejemplo
- `Leer numero` -> `scanf("%d", &numero);`
- `Escribir` -> `printf`
- `Si / SiNo / FinSi` -> `if else`
- La condición sigue siendo la misma idea: preguntar si el resto es `0`

PSEWEB

```
Algoritmo  DeterminarPar
   Definir  numero  Numero
   Leer  numero
   Si  numero  %  2  ==  0
      Escribir  "El numero es par"
   SiNo
      Escribir  "El numero es impar"
   FinSi
FinAlgoritmo
```

Código C Equivalente: Par o Impar

VERSIÓN EN C

```c
#include <stdio.h>

int main() {
    int numero;

    printf("Ingrese un número: ");
    scanf("%d", &numero);

    if (numero % 2 == 0) {
        printf("El número es par\n");
    } else {
        printf("El número es impar\n");
    }

    return 0;
}
```

Correspondencias

- `Si numero % 2 == 0` -> `if (numero % 2 == 0) {`
- `SiNo` -> `} else {`
- `FinSi` -> `}`

EJECUCIÓN

```
Ingrese un número: 8
El número es par

Ingrese un número: 7
El número es impar
```

Estructura if-else if - Múltiples Alternativas

SINTAXIS

```
if (condición1) {
    // bloque 1
} else if (condición2) {
    // bloque 2
} else {
    // bloque final
}
```

Cómo funciona

- Las condiciones se comprueban en orden
- Se ejecuta el primer bloque cuya condición es verdadera
- Si ninguna coincide, entra al `else`
- Solo se ejecuta un bloque

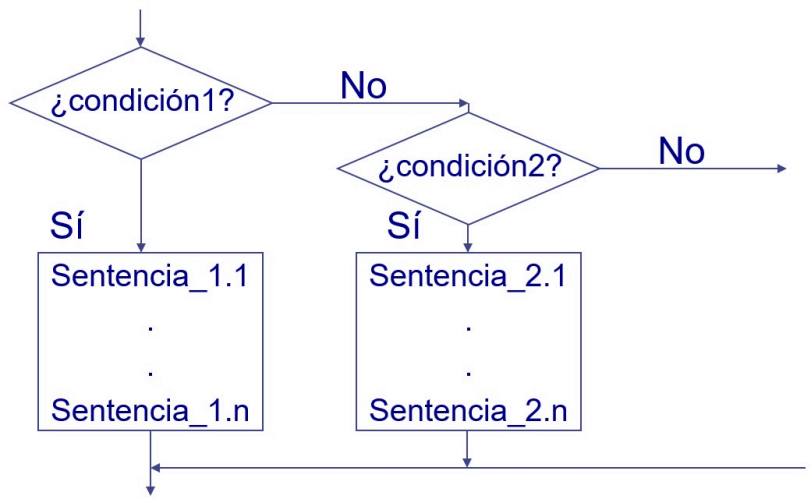
Ejemplo corto

`nota >= 90` -> Excelente

`nota >= 70` -> Aprobado

En otro caso -> Reprobado

Cómo se evalúa un `if-else if`



Lectura del flujo

- Las condiciones se prueban en orden
- Se entra al primer bloque cuya condición da verdadera
- Si ninguna coincide, se usa el `else` final
- El resto de los casos ya no se evalúa

Idea clave

El orden importa: la primera condición verdadera decide todo el camino.

Ejemplo Completo: if-else if

CLASIFICAR UN NÚMERO

```c
#include <stdio.h>

int main() {
    int numero;

    printf("Ingrese un número: ");
    scanf("%d", &numero);

    if (numero > 0) {
        printf("El número es positivo\n");
    } else if (numero < 0) {
        printf("El número es negativo\n");
    } else {
        printf("El número es cero\n");
    }

    return 0;
}
```

5

```
El número es positivo
```

-3

```
El número es negativo
```

0

```
El número es cero
```

Operadores Lógicos

Qué resuelven

- Permiten combinar varias condiciones
- Hacen posible pedir que se cumplan varios requisitos
- También permiten negar una condición existente

OPERADORES DISPONIBLES

Operador	Significado	Ejemplo
&&	Y (AND)	`a > 0 && a < 10`
||	O (OR)	`a < 0 || a > 100`
!	NO (NOT)	`!(a == 0)`

Tablas de Verdad: AND y OR

AND · &&

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

OR · ||

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

Resumen

&& exige que **ambas** condiciones sean verdaderas.
|| alcanza con que **al menos una** lo sea.

Tabla de Verdad: NOT y Ejemplos

NOT · !

A	!A
0	1
1	0

EJEMPLOS TÍPICOS

```c
int edad = 20;
int tiene_licencia = 1;

if (edad >= 18 && tiene_licencia) {
    printf("Puede conducir\n");
}

if (edad < 18 || edad > 65) {
    printf("Tarifa especial\n");
}

if (!(edad < 18)) {
    printf("Es mayor de edad\n");
}
```

Evaluación corta (short-circuit)

En C, las expresiones lógicas se evalúan de izquierda a derecha. A veces, la segunda parte ni siquiera se necesita y por eso no se evalúa.

EVITAR DIVISIÓN POR CERO

```c
int numerador;
int divisor;

printf("Ingrese numerador y divisor: ");
scanf("%d %d", &numerador, &divisor);

if (divisor != 0 && numerador / divisor > 10) {
    printf("La división es mayor que 10\n");
} else {
    printf("No se puede dividir o el resultado no es mayor que 10\n");
}
```

Por qué es seguro

Si `divisor != 0` da falso, la segunda parte no se evalúa. Gracias a eso se evita intentar `numerador / divisor` cuando `divisor` vale `0`.

VALIDAR RANGO

```c
int numero;

printf("Ingrese un número entre 1 y 100:
");
scanf("%d", &numero);

if (numero >= 1 && numero <= 100) {
    printf("Número válido\n");
} else {
    printf("Número fuera de rango\n");
}
```

ACEPTAR DOS OPCIONES

```c
char respuesta;

printf("¿Está de acuerdo? (S/N): ");
scanf(" %c", &respuesta);

if (respuesta == 'S' || respuesta == 's') {
    printf("Aceptado\n");
} else {
    printf("Rechazado\n");
}
```

CONDICIÓN NEGADA

```c
int edad;

printf("Ingrese su edad: ");
scanf("%d", &edad);

if (!(edad >= 18)) {
    printf("Es menor de edad\n");
}
// Equivale a: if (edad < 18)
```

CONDICIÓN COMPUESTA

```c
int nota;
int asistencia;

if (nota >= 60 && asistencia >= 75) {
    printf("Aprobado\n");
} else {
    printf("Reprobado\n");
}
```

Precedencia

`!` se evalúa primero, luego `&&` y finalmente `||`. Cuando la condición se vuelve difícil de leer, usar paréntesis es una buena práctica.

Anidamiento de Condicionales

SINTAXIS

```
if (condición1) {
    if (condición2) {
        // instrucciones
    } else {
        // otras instrucciones
    }
} else {
    // más instrucciones
}
```

Qué significa anidar

- Colocar un `if` dentro de otro
- Resolver decisiones en etapas
- Modelar casos en los que primero hay que pasar una condición general y después una más específica

Cuidado

Si el anidamiento crece demasiado, la lectura se vuelve difícil. En muchos casos conviene reemplazarlo por `else if`.

MAYOR DE EDAD Y LICENCIA

```c
int edad = 20;
int tiene_licencia = 1;

if (edad >= 18) {
    if (tiene_licencia) {
        printf("Puede conducir\n");
    } else {
        printf("Necesita licencia\n");
    }
} else {
    printf("Es menor de edad\n");
}
```

Secuencia

Primero se pregunta si la persona puede entrar al grupo “mayores de edad”. Recién dentro de ese grupo se evalúa si tiene licencia.

DOS NIVELES DE DECISIÓN

```c
int nota;

printf("Ingrese la nota: ");
scanf("%d", &nota);

if (nota >= 70) {
    if (nota >= 90) {
        printf("Excelente\n");
    } else {
        printf("Aprobado\n");
    }
} else {
    if (nota >= 50) {
        printf("Reprobado pero cerca\n");
    } else {
        printf("Reprobado\n");
    }
}
```

Recomendación

No conviene anidar por costumbre. Si la clasificación puede escribirse con una cadena clara de `else if`, esa versión suele ser más fácil de mantener.

Estructura switch-case

Cuándo usarlo

- Cuando se compara un mismo valor contra varias opciones fijas
- Cuando una cadena larga de `else if` se vuelve repetitiva
- Cuando el caso depende de enteros o caracteres

SINTAXIS

```
switch (variable) {
    case valor1:
        // instrucciones
        break;
    case valor2:
        // instrucciones
        break;
    default:
        // instrucciones por defecto
}
```

Reglas importantes

`switch` funciona con enteros y caracteres.
`break;` corta la ejecución del caso actual.
`default` es opcional, pero casi siempre conviene incluirlo.

Ejemplo Simple: switch-case

OPCIÓN FIJA

```c
int opcion = 2;

switch (opcion) {
    case 1:
        printf("Opción 1\n");
        break;
    case 2:
        printf("Opción 2\n");
        break;
    case 3:
        printf("Opción 3\n");
        break;
    default:
        printf("Opción inválida\n");
}
```

Ejemplo Completo: switch-case

MENÚ DE OPCIONES

```c
#include <stdio.h>

int main() {
    int opcion;

    printf("1. Sumar\n");
    printf("2. Restar\n");
    printf("3. Multiplicar\n");
    printf("Ingrese opción: ");
    scanf("%d", &opcion);

    switch (opcion) {
        case 1:
            printf("Elegiste sumar\n");
            break;
        case 2:
            printf("Elegiste restar\n");
            break;
        case 3:
            printf("Elegiste multiplicar\n");
            break;
        default:
            printf("Opción inválida\n");
    }

    return 0;
}
```

RESPUESTA CON `char`

```c
char respuesta;

printf("¿Continuar? (S/N): ");
scanf(" %c", &respuesta);

switch (respuesta) {
    case 'S':
    case 's':
        printf("Continuando...\n");
        break;
    case 'N':
    case 'n':
        printf("Cancelando...\n");
        break;
    default:
        printf("Respuesta inválida\n");
}
```

Detalle importante

Dos `case` seguidos sin `break` permiten que varios valores compartan el mismo bloque de instrucciones.

Comparación: switch vs if-else

switch

```c
switch (opcion) {
    case 1:
      printf("Uno\n");
      break;
    case 2:
      printf("Dos\n");
      break;
    case 3:
      printf("Tres\n");
      break;
}
```

if-else

```c
if (opcion == 1) {
    printf("Uno\n");
} else if (opcion == 2) {
    printf("Dos\n");
} else if (opcion == 3) {
    printf("Tres\n");
}
```

Problema

Verificar si una persona es mayor o menor de edad a partir del valor leído por teclado.

PSEWEB

```
Algoritmo  DeterminarMayorDeEdad
   Definir  edad  Numero
   Escribir  "Ingrese su edad:"
   Leer  edad
   Si  edad  >=  18
      Escribir  "Es mayor de edad"
   SiNo
      Escribir  "Es menor de edad"
   FinSi
FinAlgoritmo
```

Problema

Crear una calculadora sencilla que permita elegir una operación y, en el caso de división, evitar dividir entre cero.

Ejemplo 2: Calculadora con Menú

PSEWEB

```
Algoritmo  CalculadoraConMenu
   Definir  opcion ,  num1 ,  num2 ,  resultado  Numero
   Escribir  "1. Sumar"
   Escribir  "2. Restar"
   Escribir  "3. Multiplicar"
   Escribir  "4. Dividir"
   Escribir  "Ingrese opcion:"
   Leer  opcion
   Escribir  "Ingrese dos numeros:"
   Leer  num1
   Leer  num2
   Segun  opcion
      1 :
         resultado  =  num1  +  num2
         Escribir  "Suma: " ,  resultado
      2 :
         resultado  =  num1  -  num2
         Escribir  "Resta: " ,  resultado
      3 :
         resultado  =  num1  *  num2
         Escribir  "Producto: " ,  resultado
      4 :
         Si  num2  !=  0
            resultado  =  num1  /  num2
            Escribir  "Division: " ,  resultado
         SiNo
            Escribir  "Error: division por cero"
         FinSi
      DeOtroModo :
         Escribir  "Opcion invalida"
   FinSegun
FinAlgoritmo
```

Problema

Clasificar una nota según rangos válidos y reportar cuando el dato no pertenece al rango `0-100`.

PSEWEB

```
Algoritmo ClasificacionDeNotas
   Definir nota Numero
   Escribir "Ingrese la nota (0-100):"
   Leer nota
   Si nota >= 90 Y nota <= 100
      Escribir "Excelente"
   SiNo Si nota >= 70 Y nota < 90
      Escribir "Bueno"
   SiNo Si nota >= 60 Y nota < 70
      Escribir "Aprobado"
   SiNo Si nota >= 0 Y nota < 60
      Escribir "Reprobado"
   SiNo
      Escribir "Nota invalida"
   FinSi
FinAlgoritmo
```

Error 1 · Confundir == con =

CÓDIGO CON ERROR

```c
int edad = 20;

if (edad = 18) {
    printf("Tiene 18 años\n");
}
```

MENSAJE / PROBLEMA

= asigna 18 a edad. No pregunta si edad vale 18.

CORRECCIÓN

```c
if (edad == 18) {
    printf("Tiene 18 años\n");
}
```

Error 2 · Olvidar llaves

CÓDIGO CON ERROR

```c
if (edad >= 18)
printf("Mayor\n");
printf("Siempre se ejecuta\n");
```

MENSAJE / PROBLEMA

Solo la primera línea queda dentro del `if`. La segunda se ejecuta siempre.

CORRECCIÓN

```c
if (edad >= 18) {
    printf("Mayor\n");
    printf("Siempre se ejecuta\n");
}
```

Error 3 · Punto y coma después del `if`

CÓDIGO CON ERROR

```c
if (edad >= 18); {
    printf("Siempre se ejecuta\n");
}
```

MENSAJE / PROBLEMA

El `;` termina el `if` antes de tiempo y el bloque queda suelto.

CORRECCIÓN

```c
if (edad >= 18) {
    printf("Solo si es mayor\n");
}
```

Error 4 · Olvidar `break` en `switch`

CÓDIGO CON ERROR

```
switch (opcion) {
    case 1:
        printf("Uno\n");
    case 2:
        printf("Dos\n");
        break;
}
```

MENSAJE / PROBLEMA

Si falta `break`, la ejecución continúa en el siguiente `case`.

CORRECCIÓN

```
switch (opcion) {
    case 1:
        printf("Uno\n");
        break;
    case 2:
        printf("Dos\n");
        break;
}
```

Error 5 · Comparar `float` con ==

CÓDIGO CON ERROR

```
float a = 0.1 + 0.2;
if (a == 0.3) {
    // puede no entrar
}
```

MENSAJE / PROBLEMA

En números decimales puede aparecer una pequeña diferencia de precisión.

CORRECCIÓN

```
if (a >= 0.29 && a <= 0.31) {
    // comparación por rango
}
```

Resumen de Conceptos Clave

Estructuras de selección

- `if`: ejecuta un bloque si la condición es verdadera
- `if-else`: elige entre dos caminos
- `else if`: resuelve varias alternativas
- `switch-case`: compara un valor con opciones fijas

Operadores y hábitos

- Comparación: `==`, `!=`, `<`, `>`, `<=`, `>=`
- Lógicos: `&&`, `||`, `!`
- Usar llaves siempre
- No olvidar `break` en `switch`

Conceptos Importantes y Próxima Clase

Lo que tiene que quedar claro

- Las condiciones producen `1` o `0`
- Se pueden combinar o negar
- El orden de evaluación importa
- Elegir la estructura correcta mejora la legibilidad
- Todavía falta ver cómo repetir un mismo proceso sin copiar instrucciones

Próxima clase

- Repetición con contador
- Bucle `for`
- `for` frente a `while`
- Bucles anidados
- `break` y `continue`

Ejercicio para Practicar

Consigna

Escribir un programa que:

1. Lea un número entero
2. Determine si es positivo, negativo o cero
3. Si es positivo, determine además si es par o impar

ESQUELETO

```c
#include <stdio.h>

int main() {
    int numero;

    // Escribir el código aquí

    return 0;
}
```

Ejercicio: Ejecución y Variantes

EJEMPLOS DE EJECUCIÓN

```
Ingrese un número: 8
El número es positivo y par

Ingrese un número: -5
El número es negativo

Ingrese un número: 0
El número es cero
```

Variantes

- Agregar validación de rango
- Mostrar el valor absoluto si es negativo
- Clasificar el número en más categorías

Bibliografía y Próxima Clase

Bibliografía

- **El lenguaje de programación C** — Kernighan & Ritchie
- **Cómo programar en C/C++** — Deitel & Deitel
- Capítulos sobre control de flujo y estructuras de selección

Próxima clase

- Repetición con `for`
- `for` frente a `while`
- Bucles anidados
- `break` y `continue`