



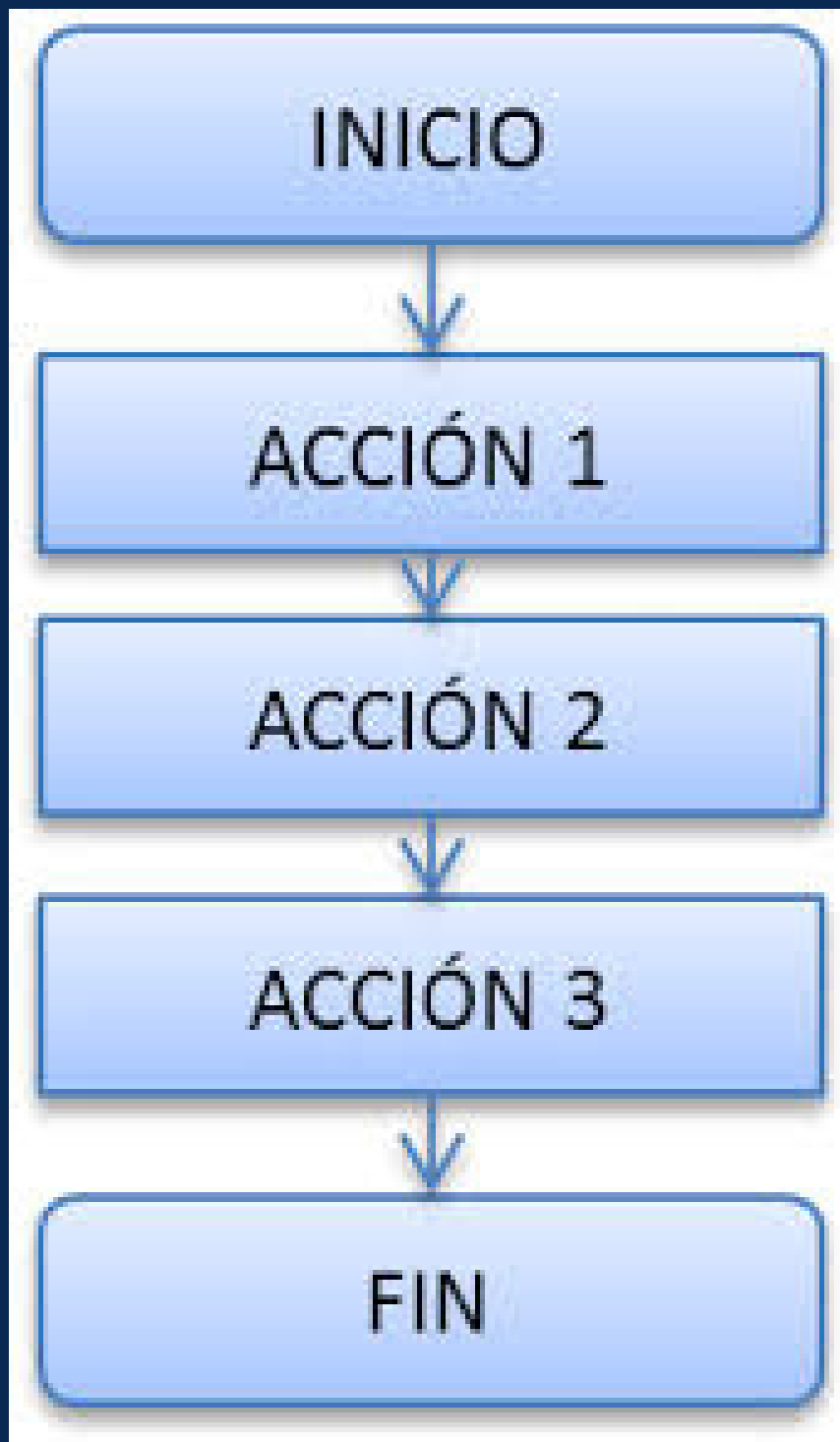
PROGRAMACION AVANZADA

ING. INDUSTRIAL

Esp. Ing.Prof.: Cristina Delia Cruz



ESTRUCTURA DE UN PROGRAMA EN C++



Una **Estructura Secuencial** es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el final del proceso.

La estructura secuencial tiene una entrada y una salida



ESTRUCTURA DE UN PROGRAMA EN C++

Ejemplo simple de estructura secuencial en C++:

Realizar un programa que tiene que realizar una serie de pasos básicos de cálculo, en los que solicita datos al usuario, realiza una operación y luego muestra el resultado. Todo ocurre de manera secuencial.

Descripción del programa:

1. Solicitar al usuario dos números enteros.
2. Sumar los dos números.
3. Mostrar el resultado de la suma.

```
#include <iostream>
using namespace std;

int main() {
    int numero1, numero2, suma;

    cout << "Ingrese el primer número: ";
    cin >> numero1;
    cout << "Ingrese el segundo número: ";
    cin >> numero2;

    suma = numero1 + numero2;

    cout << "La suma de " << numero1 << " y " << numero2 << " es: " << suma << endl;

    return 0;
}
```

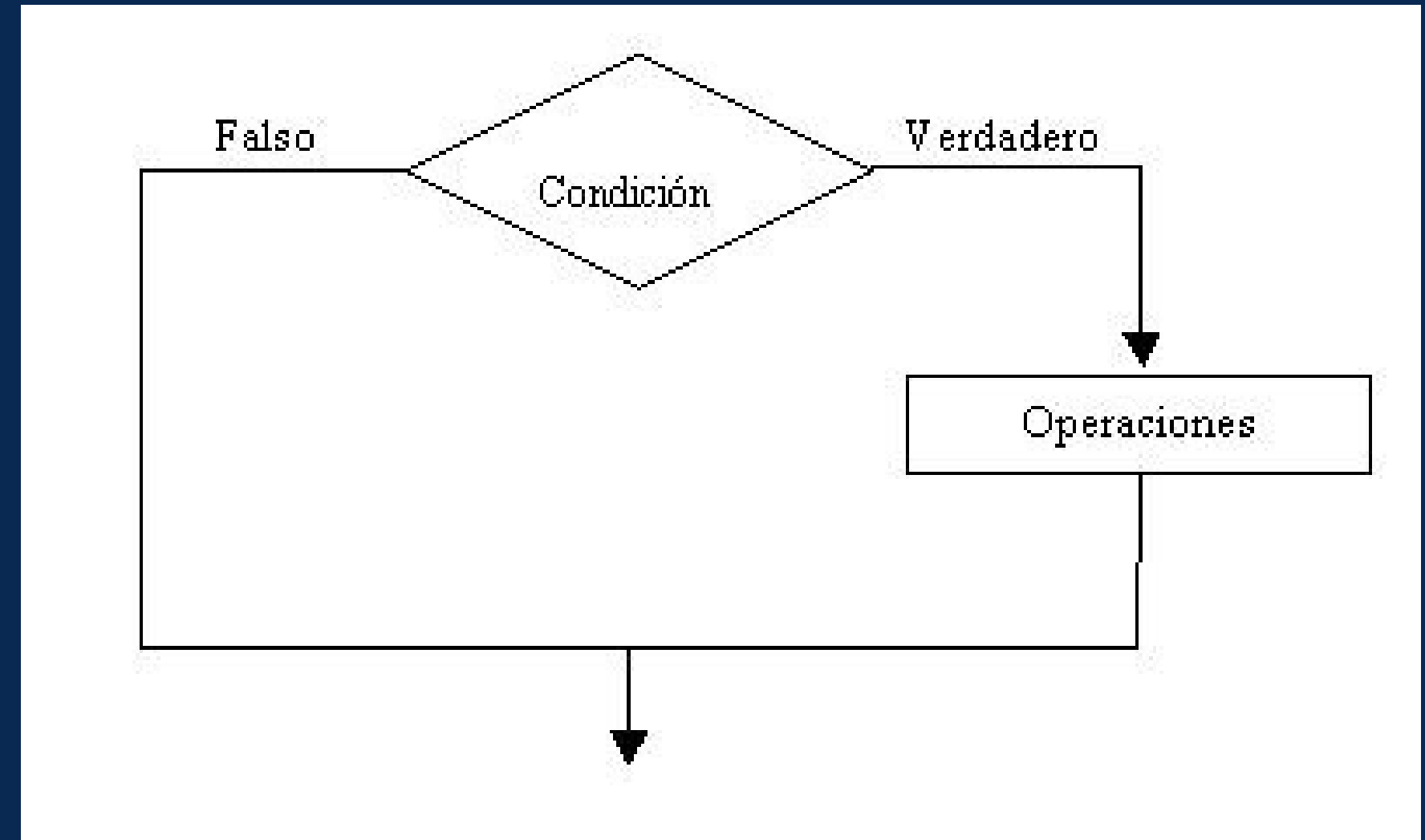


ESTRUCTURA DE DECISION SIMPLE EN C++

Las decisiones simples en C++ se basan en evaluar condiciones lógicas y ejecutar una acción en función del resultado de dicha evaluación. Se usan para tomar decisiones que determinen el flujo de un programa.

En C++, se utiliza el condicional 'if' para hacer decisiones simples. La sintaxis básica es la siguiente:

```
if (condición) {  
    // Bloque de código que se ejecuta si la condición es verdadera  
}
```





ESTRUCTURA DE DECISION SIMPLE EN C++

```
if (condición) {  
    // Bloque de código que se ejecuta si la condición es verdadera  
}
```

Elementos clave de las decisiones simples:

- **Condición:** Es una expresión que devuelve un valor de tipo booleano (true o false). Puede ser una comparación (por ejemplo, $a > b$) o cualquier expresión lógica que se evalúe a verdadero o falso.
- **Bloque de código:** Si la condición es verdadera (true), se ejecuta el bloque de código dentro de las llaves `{}`. Si es falsa, el bloque de código es ignorado y el programa continúa ejecutándose después del 'if'.





ESTRUCTURA DE DECISION SIMPLE EN C++

En este ejemplo, si el usuario ingresa un número mayor que 0, el programa mostrará el mensaje "El número es positivo". Si el número es menor o igual a 0, no ocurrirá nada ya que no se cumple la condición.

```
#include <iostream>
using namespace std;

int main() {
    int numero;

    cout << "Ingrese un número: ";
    cin >> numero;

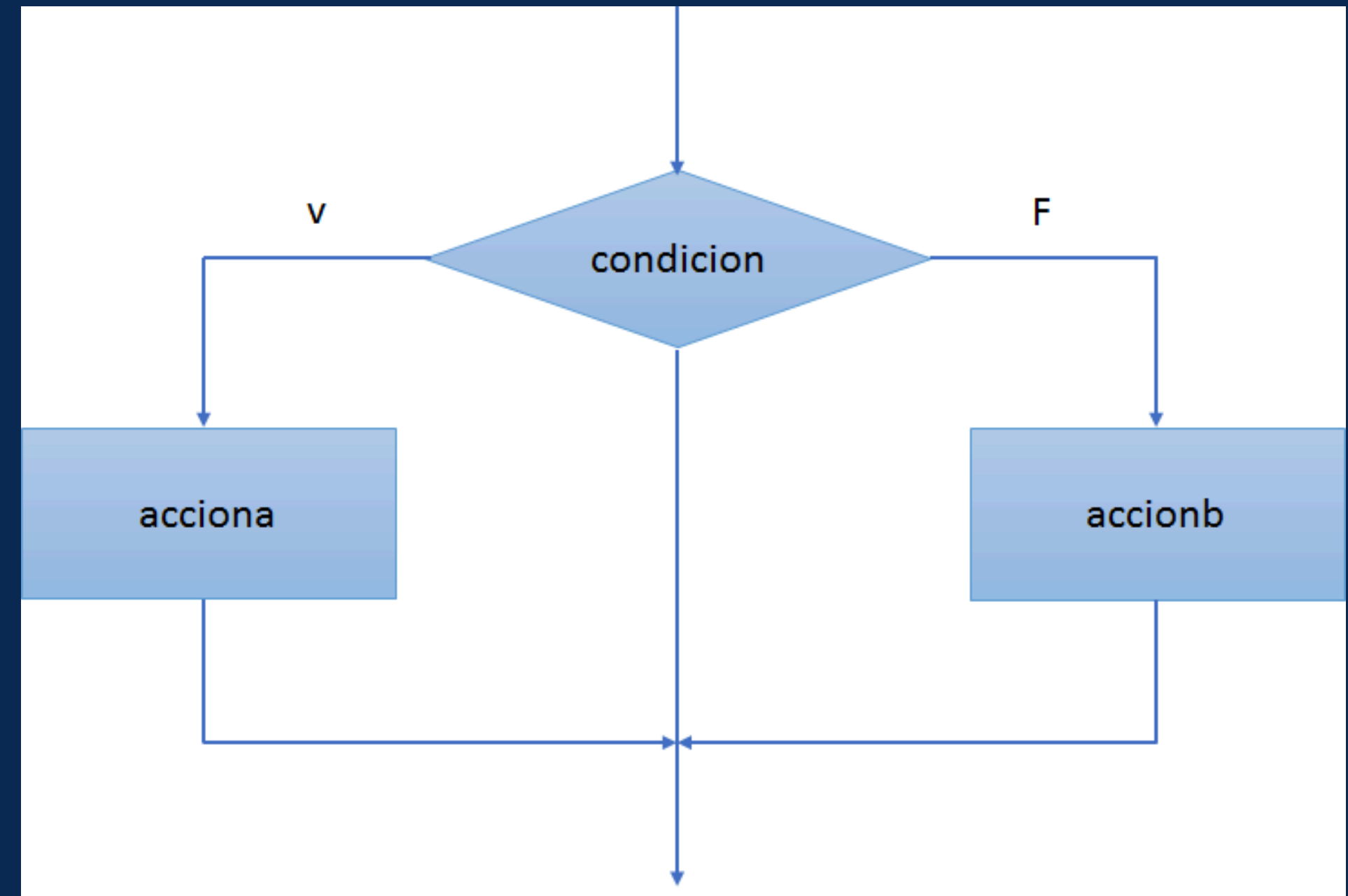
    if (numero > 0) {
        cout << "El número es positivo." << endl;
    }

    return 0;
}
```




ESTRUCTURA DE DECISION DOBLE EN C++

La estructura **if doble** en C++ se utiliza para ejecutar uno de dos bloques de código **dependiendo del resultado de una condición**. Esta estructura se conoce como **'if-else'** y permite que, si la condición no se cumple (es decir, es falsa), se ejecute otro bloque alternativo de código.





ESTRUCTURA DE DECISION DOBLE EN C++

```
if (condición) {  
    // Bloque de código que se ejecuta si la condición es verdadera  
} else {  
    // Bloque de código que se ejecuta si la condición es falsa  
}
```

Elementos clave:

- **Condición:** Es una expresión que evalúa a un valor booleano (true o false).
- Bloque **if**: **Si la condición es verdadera** (true), se ejecuta el bloque de código dentro del if.
- Bloque **else**: **Si la condición es falsa** (false), se ejecuta el bloque de código dentro del else. Este bloque es opcional.



ESTRUCTURA DE DECISION DOBLE EN C++



```
#include <iostream>
using namespace std;

int main() {
    int numero;

    cout << "Ingrese un número: ";
    cin >> numero;

    if (numero > 0) {
        cout << "El número es positivo." << endl;
    } else {
        cout << "El número es negativo o cero." << endl;
    }

    return 0;
}
```

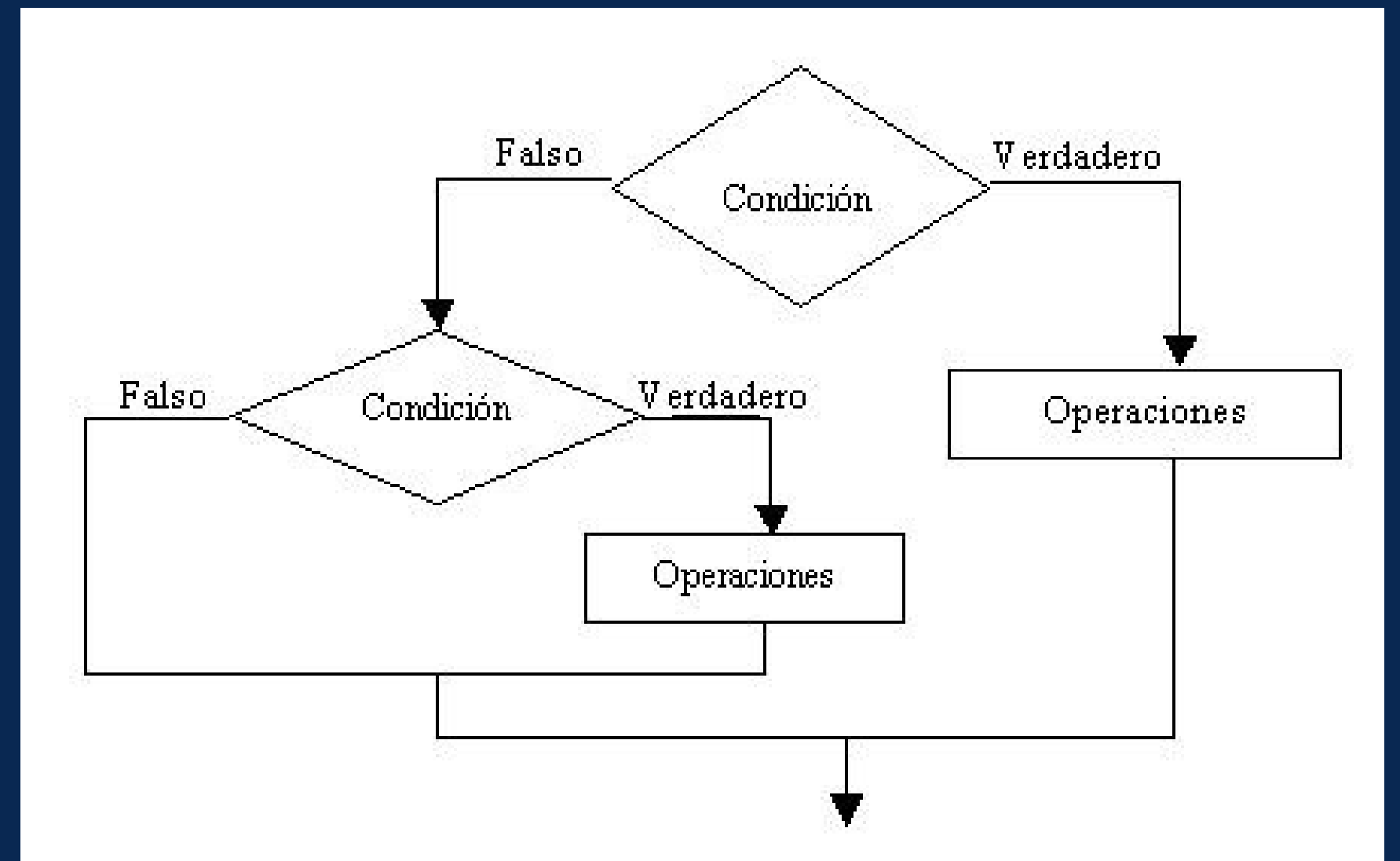
Explicación:

- El programa solicita al usuario que ingrese un número.
- Si el número es mayor que 0, el bloque dentro del if se ejecuta y muestra el mensaje "El número es positivo."
- Si el número es menor o igual a 0, la condición no se cumple y se ejecuta el bloque dentro del else, mostrando el mensaje "El número es negativo o cero."



ESTRUCTURA DE DECISION MULTIPLE EN C++

El **if anidado** en C++ es una estructura en la que un bloque **if contiene dentro de sí mismo otro bloque if**. Es decir, **puedes tener un if dentro de otro if**. Esto permite evaluar múltiples condiciones de forma jerárquica. Un if anidado te permite tomar decisiones más complejas. Básicamente, si se cumple la primera condición, puedes volver a evaluar otra condición dentro del bloque de ese if.





ESTRUCTURA DE DECISION MULTIPLE EN C++

Sintaxis:

```
if (condición1) {  
    // Bloque de código si la condición1 es verdadera  
    if (condición2) {  
        // Bloque de código si la condición2 es verdadera  
    } else {  
        // Bloque de código si la condición2 es falsa  
    }  
} else {  
    // Bloque de código si la condición1 es falsa  
}
```

En este caso, si la condición1 es verdadera, el programa sigue evaluando la condición2 dentro del primer if. Si la condición1 es falsa, se omite el segundo if y se ejecuta el bloque del else correspondiente a la condición1.



ESTRUCTURA DE DECISION MULTIPLE EN C++

```
#include <iostream>
using namespace std;

int main() {
    int edad;
    cout << "Ingresa su edad: ";
    cin >> edad;

    if (edad >= 18) {
        // Primer 'if'
        cout << "Eres mayor de edad." << endl;

        if (edad >= 65) {
            // Segundo 'if' anidado dentro del primero
            cout << "También eres considerado una persona mayor." << endl;
        } else {
            cout << "Aún no eres considerado una persona mayor." << endl;
        }
    } else {
        cout << "Eres menor de edad." << endl;
    }

    return 0;
}
```

El programa solicita al usuario que ingrese su edad. Primero, evalúa si el usuario es mayor de edad (si su edad es mayor o igual a 18).

Si es mayor de edad, entonces se evalúa otra condición anidada: si su edad es mayor o igual a 65.

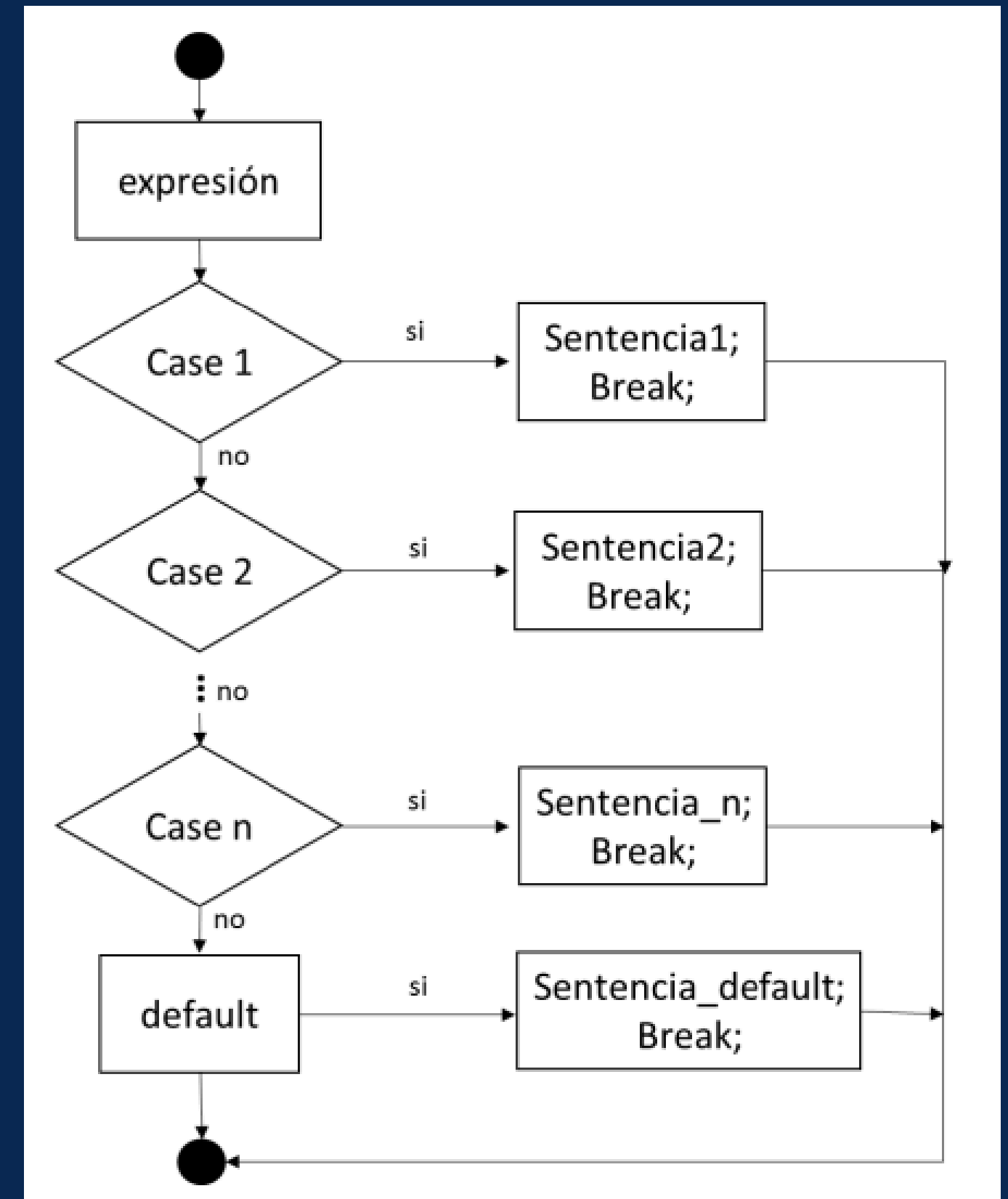
- Si también cumple esta segunda condición, el programa imprime que es considerado una "persona mayor".
- Si no cumple con esta segunda condición, imprime que aún no es una "persona mayor".

Si no es mayor de edad (edad menor a 18), simplemente imprime que es menor de edad.



ESTRUCTURA DE DECISION MULTIPLE EN C++

switch es una estructura de control de flujo en C++ que te permite seleccionar una de varias opciones **basadas en el valor de una expresión**. Es especialmente útil cuando necesitas comparar una variable contra varios valores posibles, y puede ser más limpio y eficiente que usar múltiples sentencias if o if-else.





ESTRUCTURA DE DECISION MULTIPLE EN C++

¿Cómo funciona el switch?

El **switch evalúa una expresión** (generalmente una variable) **y la compara con los diferentes casos que defines en el bloque switch.**

Si una coincidencia es encontrada, el bloque de código correspondiente a ese case se ejecuta. La ejecución continúa hasta que se encuentra una sentencia break, lo cual indica que se debe salir del bloque switch.

```
switch (expresión) {  
    case valor1:  
        // Código a ejecutar si expresión es igual a valor1  
        break;  
    case valor2:  
        // Código a ejecutar si expresión es igual a valor2  
        break;  
    // Más casos según se necesite  
    default:  
        // Código a ejecutar si no se cumple ningún 'case'  
}  

```



ESTRUCTURA DE DECISION MULTIPLE EN C++

Expresión: Es la variable o expresión cuyo valor quieres comparar (generalmente es de tipo int, char, o enum).

Case: Cada case define un valor que se compara con la expresión. Si la expresión coincide con un case, se ejecuta el código correspondiente.

Break: Después de ejecutar el código de un case, la sentencia break evita que el programa continúe ejecutando el resto de los casos. Si se omite break, el programa continuará ejecutando las siguientes instrucciones (fall through).

Default: Es opcional. Se ejecuta si ninguno de los case coincide con el valor de la expresión.

```
switch (expresión) {  
    case valor1:  
        // Código a ejecutar si expresión es igual a valor1  
        break;  
    case valor2:  
        // Código a ejecutar si expresión es igual a valor2  
        break;  
    // Más casos según se necesite  
    default:  
        // Código a ejecutar si no se cumple ningún 'case'  
}  

```



ESTRUCTURA DE DECISION MÚLTIPLE EN C++

Ejemplo Switch

```
#include <iostream>
using namespace std;

int main() {
    int opcion;

    cout << "Menu de opciones:" << endl;
    cout << "1. Sumar" << endl;
    cout << "2. Restar" << endl;
    cout << "3. Multiplicar" << endl;
    cout << "4. Dividir" << endl;
    cout << "Ingresa una opcion (1-4): ";
    cin >> opcion;
```

```
    switch(opcion) {
        case 1:
            cout << "Elegiste Sumar." << endl;
            break;
        case 2:
            cout << "Elegiste Restar." << endl;
            break;
        case 3:
            cout << "Elegiste Multiplicar." << endl;
            break;
        case 4:
            cout << "Elegiste Dividir." << endl;
            break;
        default:
            cout << "Opcion no valida." << endl;
            break;
    }

    return 0;
}
```



ESTRUCTURA DE DECISION MULTIPLE EN C++

Explicación del Ejemplo Switch



Entrada de usuario: El programa muestra un menú y solicita al usuario que ingrese una opción entre 1 y 4.

Evaluación del switch: El valor de opcion se compara con los diferentes casos (case 1, case 2, etc.).

break: Una vez que se ejecuta el bloque de código para el case correspondiente, el programa salta fuera del switch para no continuar ejecutando los demás casos.

default: Si el valor ingresado no es 1, 2, 3 o 4, el bloque default se ejecuta mostrando "Opción no válida".





ESTRUCTURA DE DECISION MÚLTIPLE EN C++

Ventajas del switch sobre if-else:

Es más claro y legible cuando tienes muchas condiciones que dependen del valor de una sola variable.

El **switch** puede ser más eficiente en ciertos casos que una cadena de **if-else**, ya que puede ser optimizado internamente por el compilador.

Cosas a tener en cuenta:

El **switch** solo admite variables o expresiones que resulten en un valor entero o de tipo carácter (int, char, enum). No puedes usar cadenas de texto (string), ni tipos flotantes (float o double).

Si omites el break, el switch continuará ejecutando el siguiente caso, lo que puede llevar a comportamientos inesperados (a menos que estés buscando intencionalmente un "fall-through").



ESTRUCTURA DE DECISION MÚLTIPLE EN C++

```
#include <iostream>
using namespace std;

int main() {
    int nivel;

    cout << "Ingrese su nivel de acceso (1-3): ";
    cin >> nivel;

    switch(nivel) {
        case 1:
            cout << "Acceso de nivel 1: Usuario basico." << endl;
        case 2:
            cout << "Acceso de nivel 2: Usuario avanzado." << endl;
        case 3:
            cout << "Acceso de nivel 3: Administrador." << endl;
            break;
        default:
            cout << "Nivel no valido." << endl;
    }

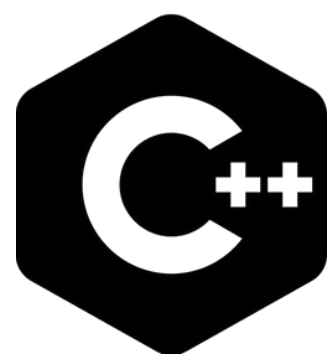
    return 0;
}
```

Ejemplo de "fall-through"

Si el usuario ingresa 1, la salida será:

```
Acceso de nivel 1: Usuario básico.
Acceso de nivel 2: Usuario avanzado.
Acceso de nivel 3: Administrador.
```

Porque no hay break después del case 1, el programa continuará ejecutando los casos 2 y 3. En resumen, el switch es una estructura de control útil para manejar múltiples valores discretos de una variable, y es más limpio que usar varias sentencias if-else cuando se tiene que comparar un solo valor con muchos casos.



EXPRESIONES CONDICIONALES EN C++

Las expresiones condicionales en C++ permiten evaluar una condición y tomar una decisión en función de si esa condición es verdadera o falsa. La estructura condicional más básica en C++ es la del operador ternario





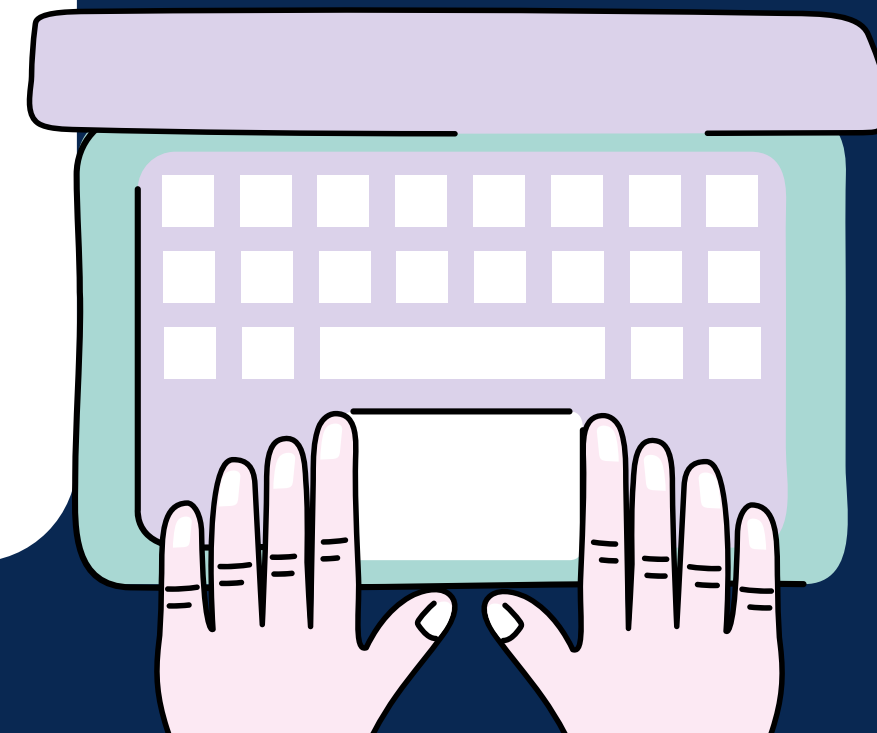
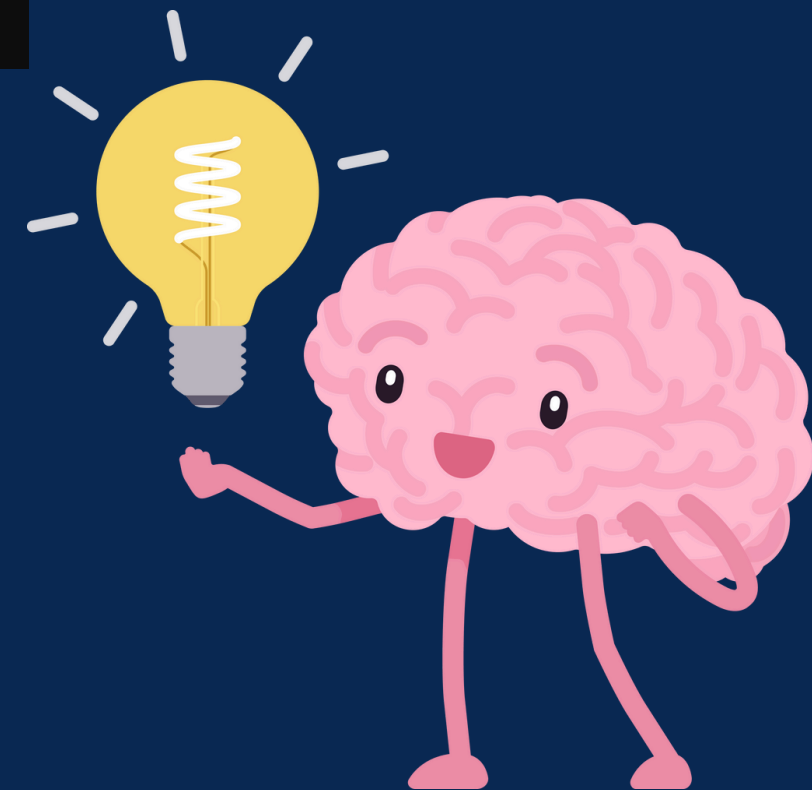
EXPRESIONES CONDICIONALES EN C++

```
condición ? expresión_si_verdadera : expresión_si_falsa;
```

condición: Es una expresión que puede evaluarse como verdadera o falsa (usualmente una comparación).

expresión_si_verdadera: Esta es la expresión que se ejecuta si la condición es verdadera.

expresión_si_falsa: Esta es la expresión que se ejecuta si la condición es falsa.





EXPRESIONES CONDICIONALES EN C++

```
#include <iostream>
using namespace std;

int main() {
    int edad;

    cout << "Ingrese su edad: ";
    cin >> edad;

    // Expresión condicional utilizando el operador ternario
    string resultado = (edad >= 18) ? "Mayor de edad" : "Menor de edad";

    cout << resultado << endl;

    return 0;
}
```

Condición: (edad >= 18) — Esta es la expresión condicional. Si la variable edad es mayor o igual a 18, la condición será verdadera; de lo contrario, será falsa.

Si verdadera: "Mayor de edad" — Si la condición es verdadera, se asigna esta cadena a la variable resultado.

Si falsa: "Menor de edad" — Si la condición es falsa, se asigna esta cadena a la variable resultado.

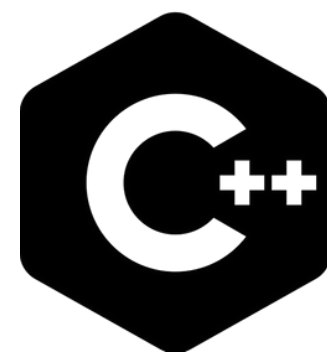
¿Por qué usar expresiones condicionales?

Concisión: Las expresiones condicionales permiten simplificar el código, evitando el uso de múltiples líneas con estructuras if-else.

Simplicidad: Son útiles cuando tienes una decisión simple con solo dos resultados posibles (verdadero o falso).

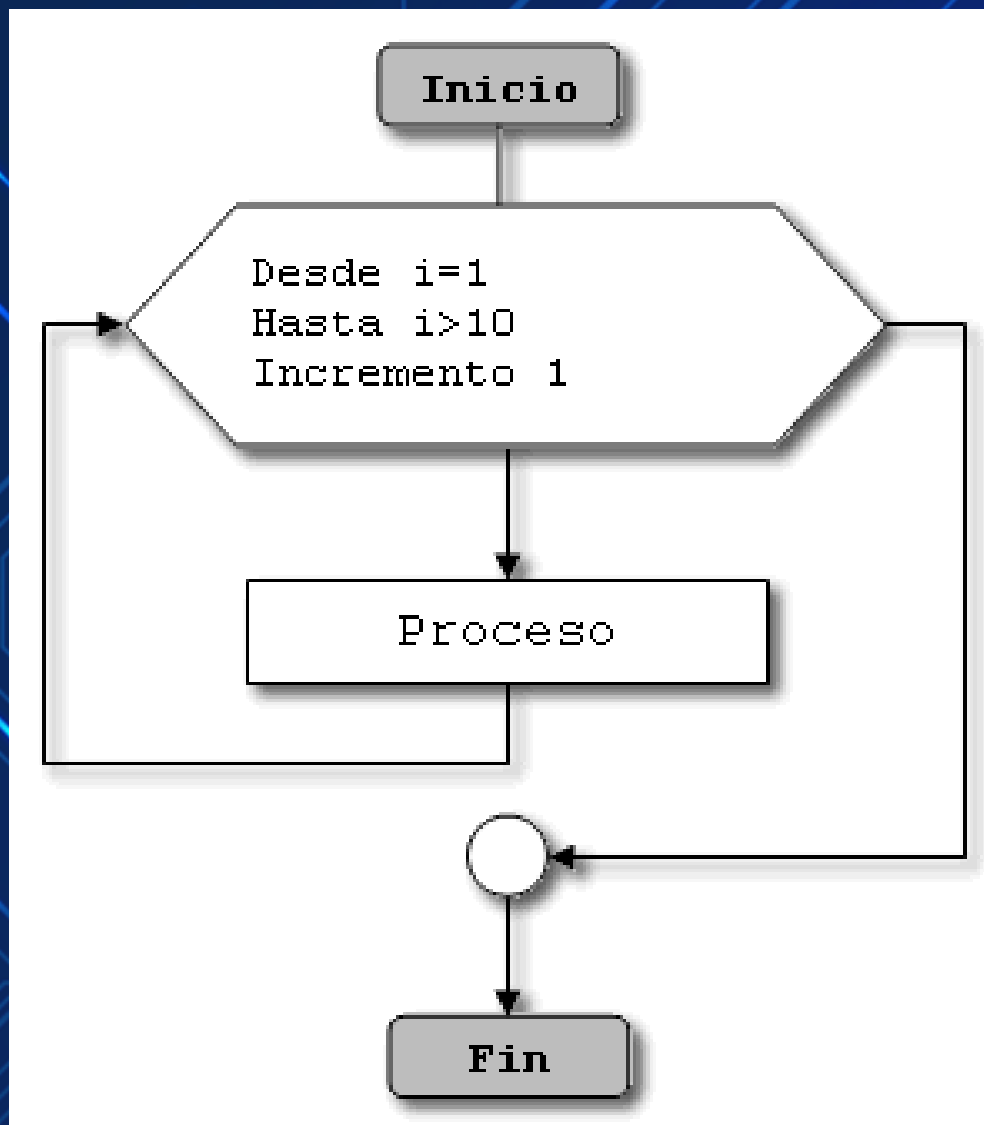
Nota:

Aunque las expresiones condicionales son útiles para decisiones simples, si tienes que realizar tareas más complejas, o si las decisiones implican más de dos caminos posibles, las estructuras if-else o switch son más apropiadas.

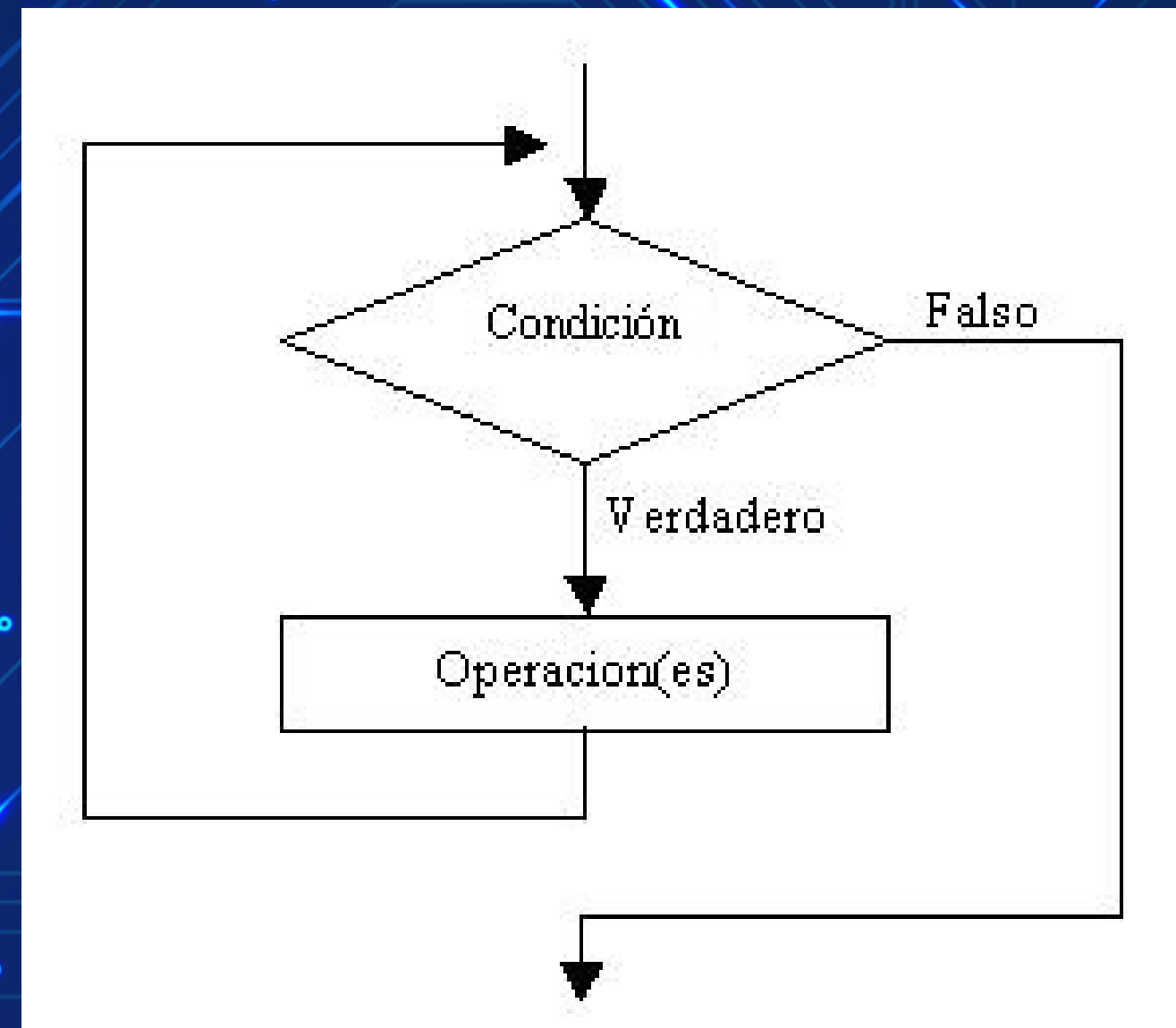


ESTRUCTURAS REPETITIVAS EN C++

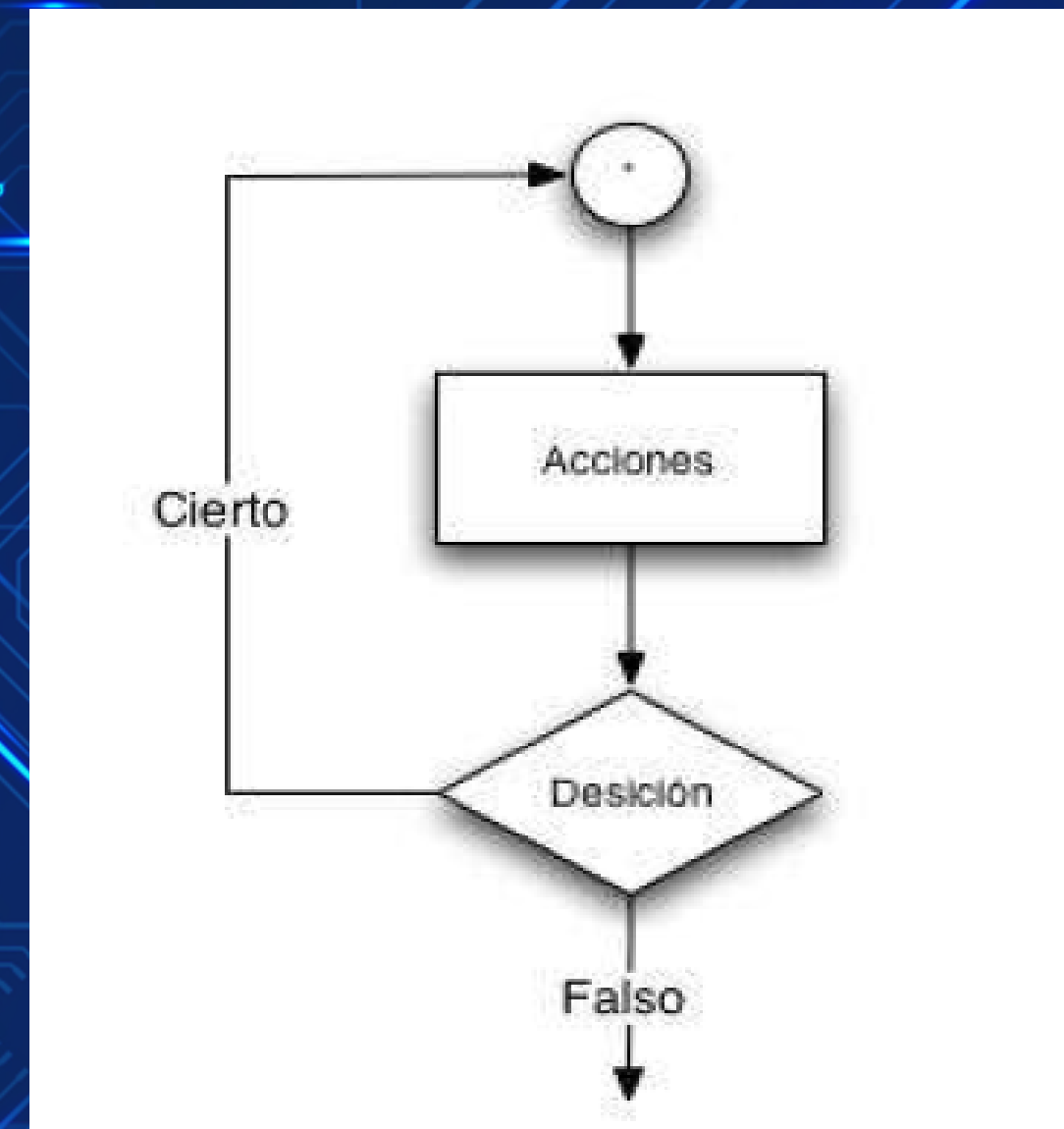
For



While



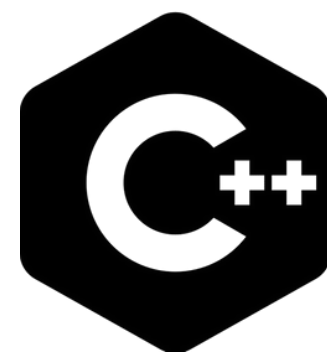
Do-While





ESTRUCTURAS REPETITIVAS EN C++

Las estructuras repetitivas en C++ (también conocidas como bucles o ciclos) son herramientas que permiten ejecutar un bloque de código varias veces de forma controlada. Estas estructuras son fundamentales para automatizar tareas que requieren repetición, sin necesidad de escribir múltiples veces el mismo código.



ESTRUCTURAS REPETITIVAS EN C++

Propósito de las estructuras repetitivas:

Automatización de tareas repetitivas:

Permiten ejecutar un conjunto de instrucciones de manera repetida hasta que se cumpla una condición específica. Esto evita la redundancia y la necesidad de escribir código de manera manual para cada repetición.

Eficiencia en el código:

Los bucles reducen la cantidad de código que se necesita escribir y, al mismo tiempo, permiten realizar operaciones masivas en un número indefinido de ocasiones (controladas por una condición de parada), lo que mejora la eficiencia en la programación.

Manipulación de conjuntos de datos:

Son esenciales para recorrer y procesar colecciones de datos, como arreglos, listas, o cualquier estructura de datos que requiera iterar sobre varios elementos.

Control sobre el flujo de ejecución:

Al usar bucles, el programador puede controlar cuántas veces se ejecutará un fragmento de código, dependiendo de una condición que se puede actualizar en cada iteración.

Aplicaciones comunes:

Recorrer listas o arreglos: Para acceder a cada elemento de una colección de datos.

Contadores o acumuladores: Para realizar operaciones como sumas, multiplicaciones, o contabilidad de eventos.

Búsqueda y filtrado: Para buscar elementos específicos dentro de un conjunto de datos.

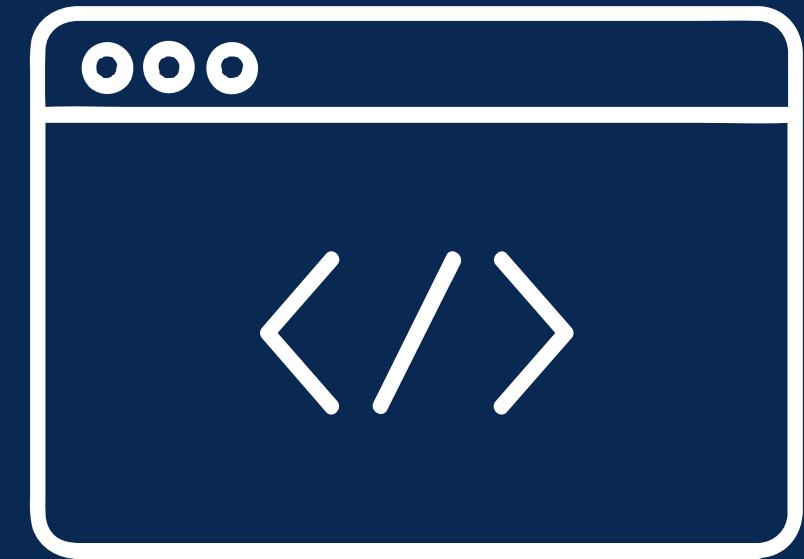
Simulaciones y cálculos iterativos: Muchas simulaciones numéricas o científicas requieren ejecutar el mismo cálculo repetidamente hasta que se alcance un cierto nivel de precisión.

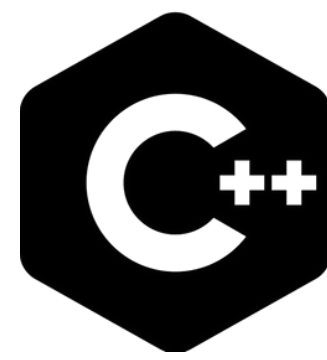


ESTRUCTURAS REPETITIVAS EN C++

Tipos de estructuras repetitivas en C++:

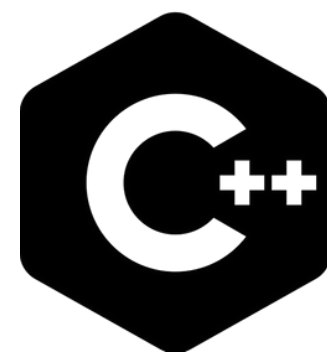
1. **while:** Ejecuta el bloque de código mientras la condición evaluada sea verdadera. Se revisa la condición antes de cada iteración.
2. **do-while:** Similar a while, pero la condición se evalúa después de cada iteración, garantizando al menos una ejecución del bloque.
3. **for:** Ideal para iteraciones controladas, donde se conoce la cantidad de repeticiones de antemano. Involucra un contador inicial, una condición de continuación, y una actualización en cada ciclo.





ESTRUCTURAS REPETITIVAS WHILE

El bucle while en C++ es una estructura repetitiva que ejecuta un bloque de código de manera repetida mientras **una condición sea verdadera**. La condición se evalúa antes de ejecutar el bloque, **por lo que si la condición es falsa al principio, el código dentro del bucle no se ejecutará** en absoluto.



ESTRUCTURAS REPETITIVAS WHILE

Funcionamiento del while:

Evaluación de la condición:

- Antes de cada iteración del bucle, se evalúa la condición especificada en el while.
- Si la condición es verdadera (true), el bloque de código dentro del while se ejecuta.
- Si la condición es falsa (false), el bucle se detiene y el control del programa continúa con el código que sigue después del while.

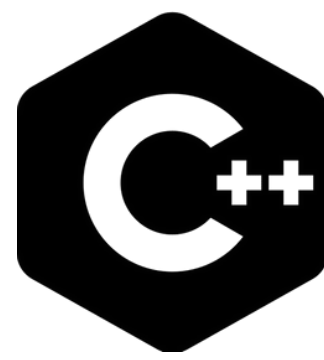
Repetición:

- Después de ejecutar el bloque de código, la condición se vuelve a verificar.
- Si la condición sigue siendo verdadera, el bucle se repite.

Condición de parada:

- Para evitar bucles infinitos, generalmente dentro del bloque de código hay alguna instrucción que hace que la condición eventualmente se vuelva falsa, como una actualización de una variable.

```
while (condición) {  
    // Bloque de código que se repite mientras la condición sea verdadera  
}
```



ESTRUCTURAS REPETITIVAS WHILE

```
#include <iostream>
using namespace std;

int main() {
    int contador = 1;

    // El bucle while se ejecutará mientras el contador sea menor o igual a 5
    while (contador <= 5) {
        cout << "Número: " << contador << endl;
        contador++; // Incrementamos el valor de la variable contador
    }

    return 0;
}
```

Condición: contador $\leq$ 5 es la condición del bucle. Mientras el valor de contador sea menor o igual a 5, el bucle se ejecutará.

Bloque de código: El bucle imprime el valor actual de contador y luego incrementa su valor con contador++.

Actualización: Cada vez que el bucle se ejecuta, contador aumenta en 1. Cuando contador llega a 6, la condición contador $\leq$ 5 se vuelve falsa, y el bucle termina.



ESTRUCTURAS REPETITIVAS DO-WHILE

El bucle **do-while** en C++ es una estructura repetitiva que, a diferencia de los bucles while y for, **garantiza que el bloque de código se ejecute al menos una vez**, independientemente de la condición. Esto **se debe a que la condición se evalúa después de que el bloque de código se ha ejecutado**, no antes.



ESTRUCTURAS REPETITIVAS DO-WHILE

Características principales del do-while:

1. **Ejecuta primero, evalúa después:** El bloque de código dentro del do-while se ejecuta al menos una vez, incluso si la condición es falsa desde el principio.
2. **Condición evaluada al final:** Después de ejecutar el bloque de código, se evalúa la condición. Si es verdadera (true), el bucle se repite; si es falsa (false), el bucle se detiene.

```
do {  
    // Bloque de código a ejecutar  
} while (condición);
```



ESTRUCTURAS REPETITIVAS DO-WHILE

```
#include <iostream>
using namespace std;

int main() {
    int i = 1;

    do {
        cout << "Número: " << i << endl;
        i++;
    } while (i <= 5);

    return 0;
}
```

Primera ejecución: El bloque dentro del do se ejecuta primero, donde i es 1, por lo tanto, se imprime Número: 1. Luego, i se incrementa a 2.

Evaluación de la condición: Después de ejecutar el bloque, se evalúa la condición $i \leq 5$. Como i es 2, la condición es verdadera, por lo que el bucle se repite.

Siguientes iteraciones: El proceso se repite para los valores de i de 2 a 5. Cuando i es 6, la condición $i \leq 5$ es falsa, y el bucle se detiene.

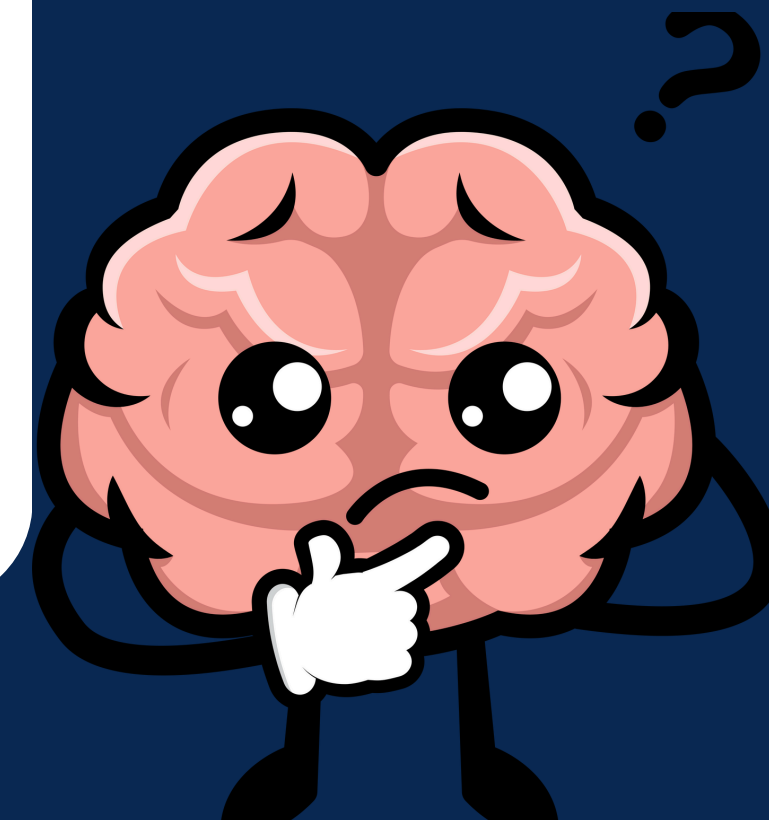
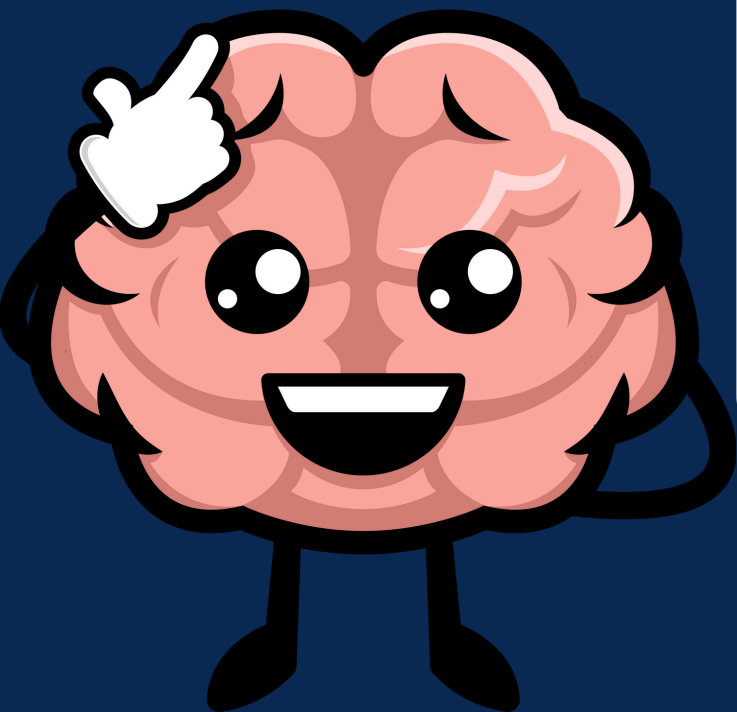


ESTRUCTURAS REPETITIVAS DO-WHILE

Diferencia con while:

En un bucle while, si la condición es falsa desde el principio, el bloque de código no se ejecuta nunca.

En un bucle do-while, el bloque de código siempre se ejecuta al menos una vez, ya que la condición se evalúa después.





ESTRUCTURAS REPETITIVAS DO-WHILE

```
#include <iostream>
using namespace std;

int main() {
    int i = 6;

    // Bucle while
    while (i <= 5) {
        cout << "Bucle while: Número " << i << endl;
    }

    // Bucle do-while
    do {
        cout << "Bucle do-while: Número " << i << endl;
    } while (i <= 5);

    return 0;
}
```

El bucle while no imprimirá nada porque la condición $i \leq 5$ es falsa desde el principio. El bucle do-while imprimirá una vez Bucle do-while: Número 6 porque ejecuta el código antes de evaluar la condición.



ESTRUCTURAS REPETITIVAS DO-WHILE

Ventajas del do-while:

Es útil cuando necesitas asegurarte de que el bloque de código se ejecute al menos una vez, independientemente de la condición.

Se utiliza cuando quieres verificar la condición después de ejecutar el código.

Es ideal en situaciones donde el proceso debe ocurrir al menos una vez, como un menú interactivo que necesita ser mostrado al menos una vez antes de verificar si el usuario desea continuar.



ESTRUCTURAS REPETITIVAS FOR

El bucle **for** en C++ es una estructura repetitiva que **se utiliza cuando conocemos de antemano el número de iteraciones que queremos realizar**. Se compone de tres partes principales: **la inicialización, la condición y el incremento (o decremento)**.

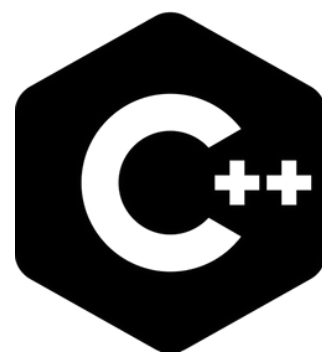


ESTRUCTURAS REPETITIVAS FOR

Funcionamiento del for:

1. **Inicialización:** Se ejecuta una sola vez al comienzo del bucle. Generalmente, se utiliza para declarar e inicializar una variable de control.
2. **Condición:** Se evalúa antes de cada iteración. Si la condición es verdadera (true), el cuerpo del bucle se ejecuta. Si la condición es falsa (false), el bucle se detiene.
3. **Incremento/Decremento:** Después de cada iteración del bucle, la variable de control se incrementa o decrementa.

```
for (inicialización; condición; incremento/decremento) {  
    // Bloque de código que se ejecuta mientras la condición sea verdadera  
}
```

ESTRUCTURAS REPETITIVAS FOR

```
#include <iostream>
using namespace std;

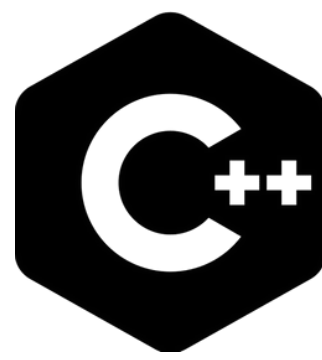
int main() {
    // Bucle for que imprime los números del 1 al 5
    for (int i = 1; i <= 5; i++) {
        cout << "Número: " << i << endl;
    }

    return 0;
}
```

Inicialización: `int i = 1` declara e inicializa la variable `i` con un valor de

Condición: `i <= 5` es la condición que se evalúa antes de cada iteración. Mientras `i` sea menor o igual a 5, el bucle continuará ejecutándose.

Incremento: `i++` incrementa el valor de `i` en 1 después de cada iteración.



ESTRUCTURAS REPETITIVAS FOR

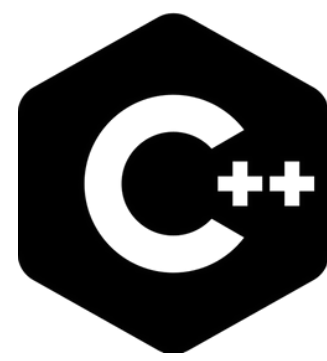
```
#include <iostream>
using namespace std;

int main() {
    // Bucle for que imprime los números del 1 al 5
    for (int i = 1; i <= 5; i++) {
        cout << "Número: " << i << endl;
    }

    return 0;
}
```

- **Primera iteración:** $i = 1$, se imprime Número: 1, luego i se incrementa a 2.
- **Segunda iteración:** $i = 2$, se imprime Número: 2, luego i se incrementa a 3.
- **Tercera iteración:** $i = 3$, se imprime Número: 3, luego i se incrementa a 4.
- **Cuarta iteración:** $i = 4$, se imprime Número: 4, luego i se incrementa a 5.
- **Quinta iteración:** $i = 5$, se imprime Número: 5, luego i se incrementa a 6.

Cuando i llega a 6, la condición $i \leq 5$ es falsa y el bucle se detiene.



ESTRUCTURAS REPETITIVAS FOR

Características clave del for:

Es útil cuando sabes de antemano cuántas veces quieres que se repita el bucle.

Puedes declarar la variable de control dentro del for (como en el ejemplo `int i`), lo que la convierte en una variable local al bucle.

La inicialización, condición y actualización están juntas, lo que hace el for más conciso que el while para muchas situaciones.



CICLOS ANIDADOS EN C++

Un ciclo anidado (también conocido como bucle anidado) **ocurre cuando colocamos un bucle dentro de otro**. Es decir, un ciclo se ejecuta dentro del bloque de código de otro ciclo. **Este tipo de estructura es muy útil para trabajar con datos bidimensionales, como arreglos o matrices, o para realizar tareas repetitivas dentro de otras repeticiones.**



CICLOS ANIDADOS EN C++

Funcionamiento

En un ciclo anidado, el bucle externo controla la cantidad de veces que se ejecuta el bucle interno. Cada vez que el ciclo externo realiza una iteración, el ciclo interno se ejecuta en su totalidad. Esto significa que el ciclo interno se repetirá completamente cada vez que el ciclo externo avance una iteración.



CICLOS ANIDADOS EN C++

Piensa en una tabla de datos de 3 filas y 3 columnas. Para recorrer cada una de las celdas de la tabla, necesitarías:

1. Un ciclo para las filas.
2. Otro ciclo para las columnas dentro de cada fila.

```
for (int i = 0; i < numeroDeFilas; i++) {  
    for (int j = 0; j < numeroDeColumnas; j++) {  
        // Código que se ejecuta en cada iteración de ambos ciclos  
    }  
}
```



CICLOS ANIDADOS EN C++

```
#include <iostream>
using namespace std;

int main() {
    int filas = 5;
    int columnas = 5;

    // Bucle externo controla las filas
    for (int i = 1; i <= filas; i++) {
        // Bucle interno controla las columnas
        for (int j = 1; j <= columnas; j++) {
            cout << i * j << "\t"; // Imprime la multiplicación de los índices i y j
        }
        cout << endl; // Salto de línea después de cada fila
    }

    return 0;
}
```



CICLOS ANIDADOS EN C++

Explicación del Ejemplo:

Bucle externo (**for (int i = 1; i <= filas; i++)**): Este bucle controla las filas de la tabla. Cada vez que realiza una iteración, controla que se pase a la siguiente fila.

Bucle interno (**for (int j = 1; j <= columnas; j++)**): Este bucle controla las columnas. Dentro de cada fila, el bucle interno recorre todas las columnas imprimiendo el resultado de la multiplicación $i * j$.



CICLOS ANIDADOS EN C++

Funcionamiento de Ejemplo:

El **ciclo externo** empieza con $i = 1$ (la primera fila). El ciclo interno se ejecuta completamente mientras $i = 1$, imprimiendo $1 * j$ (donde j va de 1 a 5). Después de completar todas las iteraciones del **ciclo interno**, se imprime un salto de línea (endl) y el **ciclo externo** avanza a la siguiente fila con $i = 2$. **Se repite el proceso hasta que ambos ciclos terminan.**



CICLOS ANIDADOS EN C++

Aplicaciones comunes de los ciclos anidados:

- Recorrer y manipular matrices bidimensionales.
- Generar tablas (como en el ejemplo anterior).
- Procesar gráficos o tablas de datos.
- Resolver problemas donde se necesitan múltiples niveles de repetición, como en la búsqueda de combinaciones o permutaciones.

Los **ciclos anidados** son una herramienta poderosa cuando se necesitan múltiples niveles de repetición en tus programas.

¿PREGUNTAS?

```
    render() {  
      return (  
        <React.Fragment>  
          <div className="py-5">  
            <div className="container">  
              <Title name="our" title="product">  
                <div className="row">  
                  <ProductConsumer>  
                    {(value) => {  
                      console.log(value)  
                    }}  
                  </ProductConsumer>  
                </div>  
              </div>  
            </div>  
          </React.Fragment>  
        )  
      )  
    }  
  }  
}
```




MUCHAS
GRACIAS!!!

Nos Vemos en la Siguiente Clase