Universidad LOYOLA

Departamento de Métodos Cuantitativos

Tema 3. Estructuras de Control

Alfonso Carlos Martínez Estudillo (acme@uloyola.es)

Fundamentos de Informática I

1º de Grado en Ingeniería Informática y Tecnologías Virtuales

Curso 2021-2022

Índice de contenidos

Índice de contenidos

Flujo de ejecución

- En principio, las sentencias de un programa en C++ se ejecutan secuencialmente, es decir, una a continuación de la anterior, empezando por la primera y acabando por la última.

```
1 int main(){
2   int salario;
3   cout << "Introduzca su salario: ";
4   cin >> salario;
5   cout << "Salario introducido: " << salario << endl;
6 }
```

- Con las estructuras de control podemos cambiar el orden de ejecución.

Flujo de ejecución

- **Estructuras de control**: determinan que sentencia ejecutar en cada momento alterando el flujo secuencial.
- **Tipos**:
  - Estructuras condicionales.
  - Estructuras iterativas.
  - Llamadas a funciones.
- Se pueden utilizar diagramas de flujo para mostrar la forma de ejecución de un algoritmo o programa.

Estructura secuencial

- En ella, las instrucciones se ejecutan sucesivamente, es decir, una detrás de otra sin saltos y según el orden de aparición de estas.
- Ejemplo: Cálculo de las raíces de una ecuación de segundo grado.

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

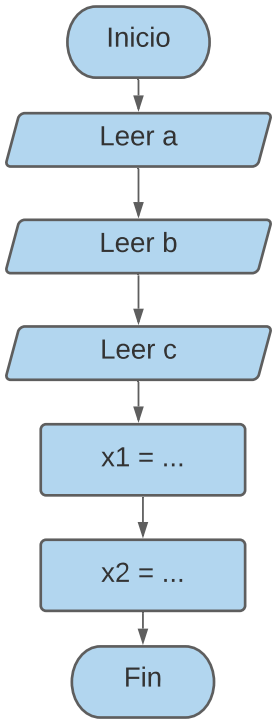
Estructura secuencial

Pseudocódigo

```
Inicio
    Leer (a)
    Leer (b)
    Leer (c)
    x1 <- (-b + raiz(b*b - 4*a*c)) / 2*a
    x2 <- (-b -  raiz(b*b - 4*a*c)) / 2*a
    Escribir("Las soluciones son: x1 y x2)
Fin
```

Estructura secuencial

Diagrama de flujo



Estructura secuencial

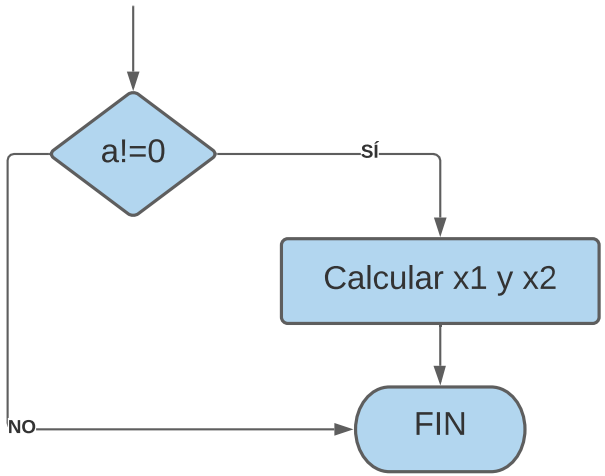
Código

```
1  #include<iostream>
2  #include<cmath>
3  using namespace std;
4
5  int main(){
6     float a, b, c, x1, x2;
7    cout << "Introduzca coeficiente a: ";
8    cin >> a;
9    cout << "Introduzca coeficiente b: ";
10   cin >> b;
11   cout << "Introduzca coeficiente c: ";
12   cin >> c;
13
14   x1 = (-b + sqrt(b*b-4*a*c))/(2*a);
15   x2 = (-b - sqrt(b*b-4*a*c))/(2*a);
16   cout << "Las soluciones son: " << x1 << " y " << x2 << endl;
17
18    return 0;
19 }
```

Índice de contenidos

Estructura condicional

- En el ejemplo anterior, si *a* vale 0, la ejecución de la división produce un error de división por 0.
- De este modo, debemos establecer la condición de que sólo se debe calcular el valor de x1 y x2 si a!=0.
- **Ejecución condicional:** ejecución de una o más sentencias dependiendo de la evaluación de una condición.



Estructura condicional

- Una estructura condicional es aquella que controla si una sentencia o conjunto de sentencias se ejecutan o no en función del cumplimiento de una condición.
- **expresion1** *operador* **expresion2**
  - **expresion**: cualquier expresión compatible.
  - *operador*: relacionales o lógicos.
- La evaluación de las condiciones se llevan a cabo haciendo uso de las tablas de verdad vistas en el tema anterior.

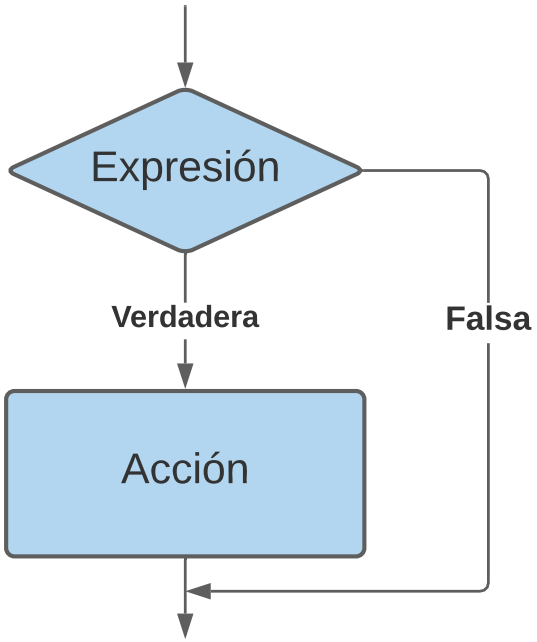
Tipos de estructuras condicionales

Existen tres tipos de estructuras condicionales:

- Esquema simple.
- Esquema doble.
- Esquema múltiple.

Esquema condicional simple

- **Expresión:** expresión lógica que determina si una acción ha de ejecutarse o no.
- **Acción:** sentencias que se ejecutarán cuando la expresión lógica sea verdadera.



Esquema condicional simple C++

- **condicion**: es una expresión lógica.
- **bloque if**: sentencias que se ejecutan cuando *condicion* es verdadero.

```
1 if (<condicion>){
2   <bloque if>
3 }
```

Ejemplo anterior

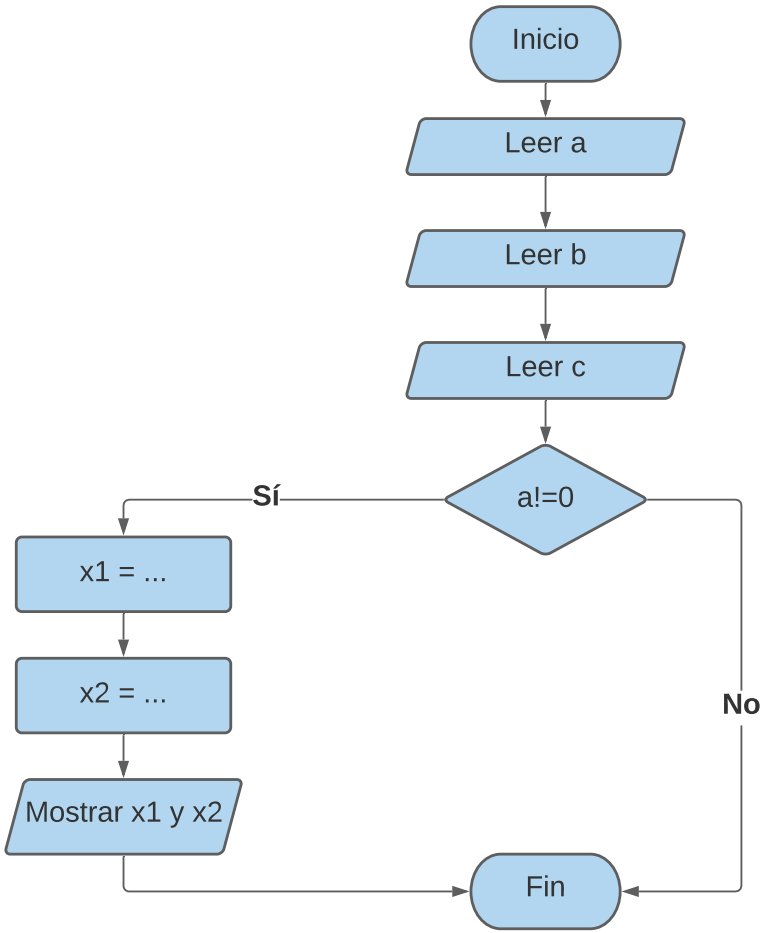
Código

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

int main(){
   float a, b, c, x1, x2;
   cout << "Introduzca coeficiente a: ";
   cin >> a;
   cout << "Introduzca coeficiente b: ";
   cin >> b;
   cout << "Introduzca coeficiente c: ";
   cin >> c;
   if (a!=0){
      x1 = (-b + sqrt(b*b-4*a*c))/(2*a);
      x2 = (-b - sqrt(b*b-4*a*c))/(2*a);
      cout << "Las soluciones son: " << x1 << " y " << x2 << endl;
   }
   return 0;
}
```

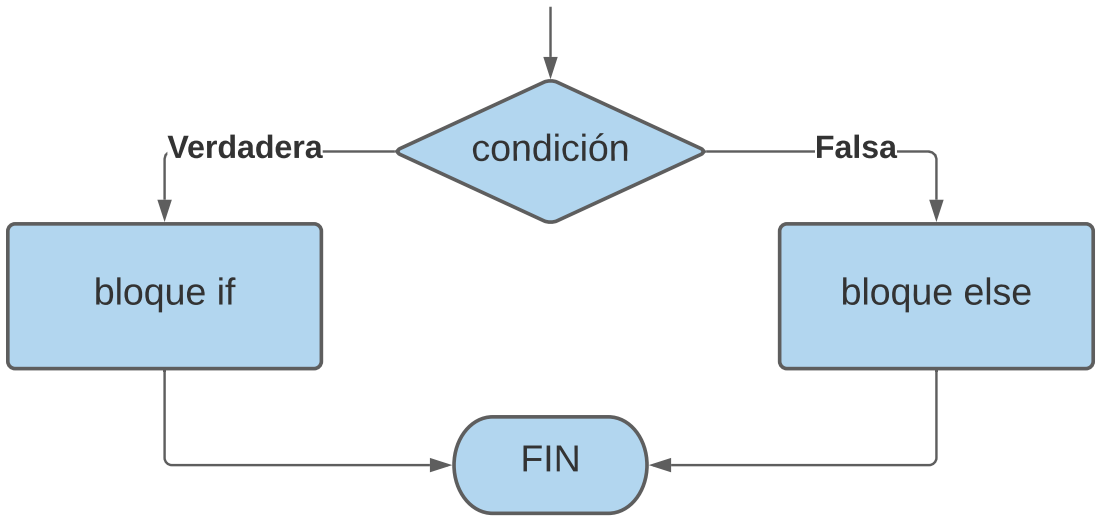
Ejemplo anterior

Diagrama de Flujo



Esquema condicional doble

- **Condición:** expresión lógica que determina si una acción ha de ejecutarse o no.
- **Bloque if:** sentencias que se ejecutarán cuando la condición lógica sea verdadera.
- **Bloque else:** sentencias que se ejecutarán cuando la condición sea falsa.



Esquema condicional doble C++

- **Condición:** expresión lógica que determina si una acción ha de ejecutarse o no.
- **Bloque if:** sentencias que se ejecutarán cuando la condición lógica sea verdadera.
- **Bloque else:** sentencias que se ejecutarán cuan la condición sea falsa.

```
1 if (<condicion>){
2   <bloque if>
3 }
4 else{
5   <bloque else>
6 }
```

Ejemplo anterior

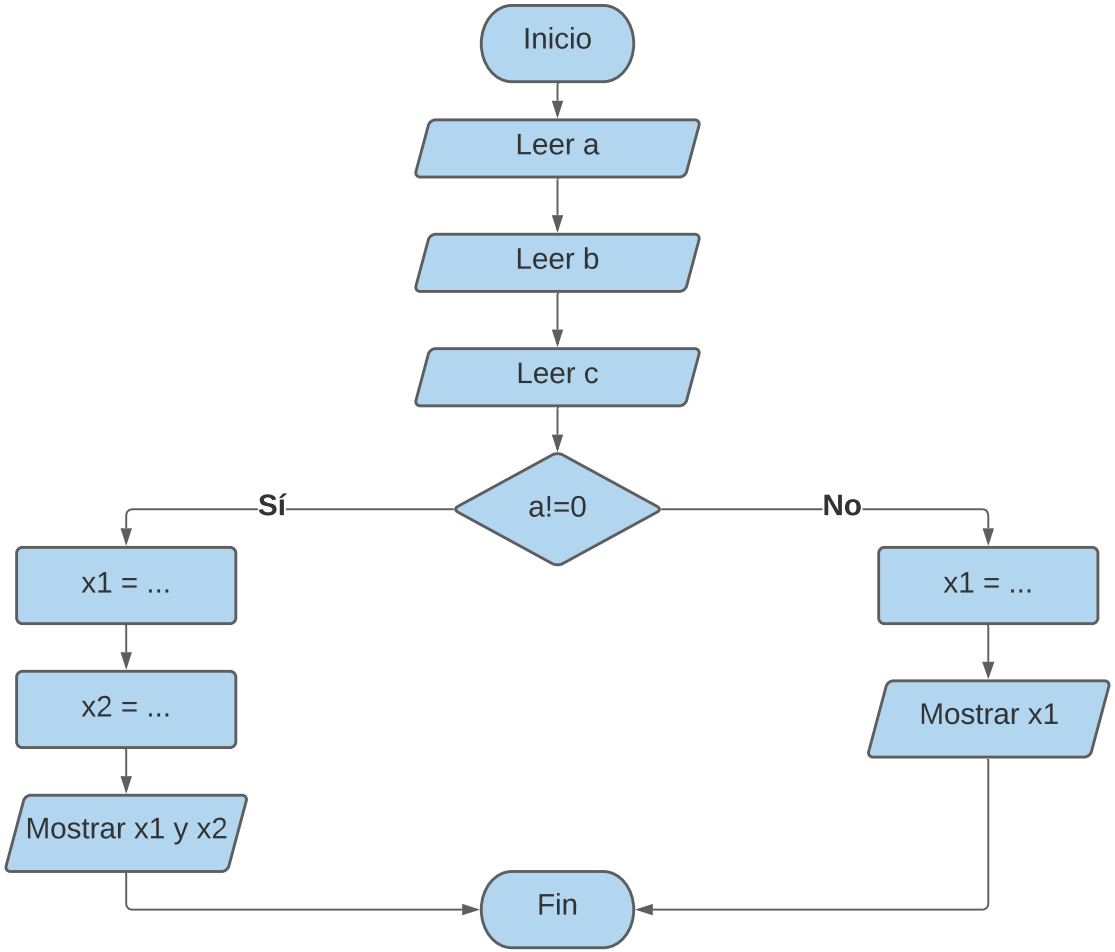
Código

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

int main(){
   float a, b, c, x1, x2;
  cout << "Introduzca coeficiente a: ";
   cin >> a;
  cout << "Introduzca coeficiente b: ";
   cin >> b;
  cout << "Introduzca coeficiente c: ";
   cin >> c;
   if (a!=0){
     x1 = (-b + sqrt(b*b-4*a*c))/(2*a);
     x2 = (-b - sqrt(b*b-4*a*c))/(2*a);
     cout << "Las soluciones son: " << x1 << " y " << x2 << endl;
  }else{
     x1 = -c/b;
     cout << "Unica solucion: " << x1 << endl;
  }
   return 0;
}
```

Ejemplo anterior

Diagrama de Flujo



Anidamiento

- Dentro de un bloque de sentencias **if** o **else** pueden incluirse otras estructuras condicionales.
- Se pueden anidar estructuras.
- Tantas como permita el compilador.

En el siguiente programa, ¿cuándo se llegará a cada una de las instrucciones?

```
1  if(cond1){
2    inst1;
3    if(cond2){
4      inst2;
5    }
6    else{
7      inst3;
8    }
9    inst4;
10 }
```

Ejemplo

Determinar si un número es positivo, negativo o cero

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5     float valor;
6    cout << "Introduzca valor: ";
7     cin >> valor;
8
9     if (valor>0)
10    {
11       cout << "El numero es mayor que 0" << endl;
12    }
13    else{
14       if (valor < 0)
15       {
16          cout << "El numero es menor que 0" << endl;
17       }
18       else{
19          cout << "El numero es igual a 0" << endl;
20       }
21    }
22    return 0;
23 }
```

Ejemplo

Determinar el mayor de tres números

```
1  if (a>=b)
2  {
3     if (a>=c){
4        max=a;
5     }
6     else {
7        max=c;
8     }
9  }
10 else {
11    if (b>=c)
12    {
13       max=b;
14    }
15    else {
16       max=c;
17    }
18 }
19 cout << "El mayor numero es: " << max << endl;
```

Evaluación de expresiones lógicas

```
1 if ((a < 3) && (b < 0))
```

- **Evaluación completa (ciclo largo):** el compilador evalúa (innecesariamente) todas las expresiones lógicas.
- **Evaluación cortocircuito (ciclo corto):** si no es necesario, el compilador no evalúa todas las expresiones lógicas. Por ejemplo, si a es igual a 10, sólo se evaluaría la primera condición y al ser **false**, pararía de evaluar.

Evaluación de expresiones lógicas C++

C++ hace uso de evaluaciones cortocircuito con los operadores:

- &&: recomendable poner al principio aquellas condiciones que sean más probables de evaluarse como **false**. De esta forma, con que una condición sea **false**, la expresión se evaluará como falsa.
- ||: recomendable poner al principio aquellas condiciones que sean más probables de evaluarse como **true**. De esta forma, con que una condición sea **true**, la expresión se evaluará como verdadera.

Esquema condicional múltiple

- Esquema que permite seleccionar una, de entre múltiples alternativas.
- Se suele usar para:
  - Construcción de menús.
  - Realizar las mismas operaciones para un número determinado de constantes.

Esquema condicional múltiple C++

- **expresión**: expresión entera o carácter.
- **constante**: valor único, de tipo entero o carácter.
- Switch comprueba solo la igualdad y permite anidamiento.

```
switch (<expresion>){
  case <constante1>:
    <sentencias1>
    break;
  case <constante2>:
    <sentencias2>
    break;
  ...
  default:
    <sentencias>
}
```

Funcionamiento

- Si el valor de **expresión** es igual a alguna de las etiquetas **case**, la ejecución comienza con la primera sentencia de ese **case** y continua hasta que:
  - Se encuentra el final de la sentencia **switch**.
  - Se encuentra la sentencia **break**.
- Es habitual terminar la ejecución después de ejecutarse un único **case**. Para ello, situar la sentencia **break** como última sentencia del bloque.
- Si el valor de **expresión** no está en la lista, se ejecutarán las sentencias asociadas a la etiqueta **default**.

Ejemplo

```
1  int main(){
2    char nota;
3    cout << "Introduzca calificacion A-F: ";
4    cin >> nota;
5    switch (nota){
6      case 'A': cout << "Excelente. Examen superado";
7      break;
8      case 'B': cout << "Notable. Suficiencia";
9      break;
10     case 'C': couut << "Aprobado";
11     break;
12     case 'D': case 'E': case 'F': cout << "Suspendido";
13     break;
14     default: cout << "No es posible esta nota";
15   }
16   return 0;
17 }
```

Operador ternario condicional

- Como vimos en el tema anterior, tiene el siguiente formato:
  - condicion ? expresion1 : expresion2
- **condición**: expresión lógica.
- **expresion1**: sentencia que se ejecutará si la condición es verdadera.
- **expresion2**: expresión que se ejecutará si la condición es falsa.

```
1 cout << "Introduzca la edad: ";
2 cin >> edad;
3 (edad>=18) ? cout << "Mayor de edad" : cout << "Menor de edad";
```

Errores frecuentes

- Utilizar el operador de asignación (=) en lugar del operador de igualdad (==).
- En una sentencia **if** anidada, cada **else** se corresponde con la sentencia **if** precedente más cercana. Para evitar errores, se recomienda tabular el código.
- El selector del **switch** debe ser un entero o carácter.

Índice de contenidos

Introducción

- Realizar una tarea muchas veces (de forma repetitiva).
  - Los humanos las encontramos difíciles y tediosas.
  - Los ordenadores tienen capacidad para ejecutarlas con gran velocidad, precisión y fiabilidad.
- Estructuras repetitivas o iterativas:
  - Comúnmente conocidas como bucles.
  - Permiten la ejecución de una secuencia de sentencias un número de veces.
- Conceptos:
  - Cuerpo del bucle: conjunto de sentencias que se repiten.
  - Iteración: cada repetición del cuerpo del bucle.

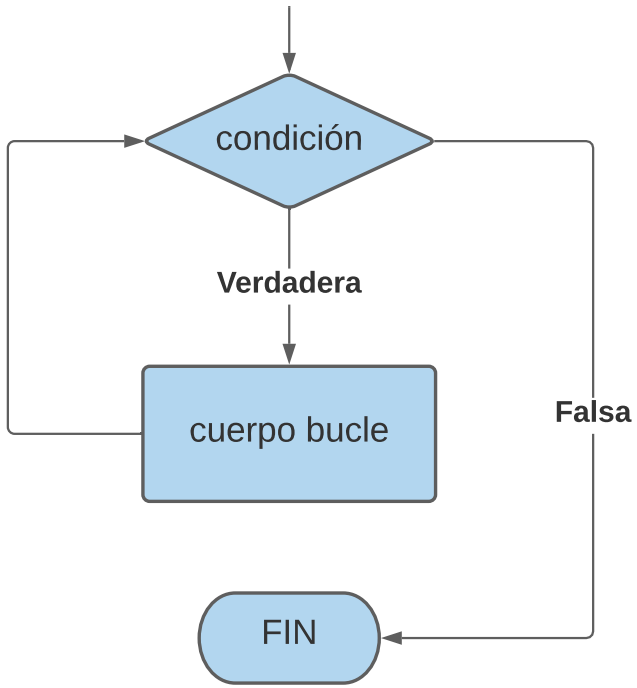
Introducción

El número de veces que se ejecuta un bucle puede estar controlado por:

- Una **condición**: el cuerpo del bucle se ejecuta hasta que se satisface una determinada condición. Tenemos bucles de test previo y test posterior.
- Por un **contador**: el cuerpo del bucle se ejecuta un número determinado de veces.

Test previo: bucle while

```
while(<condicion>){
  <cuerpo bucle>
}
```

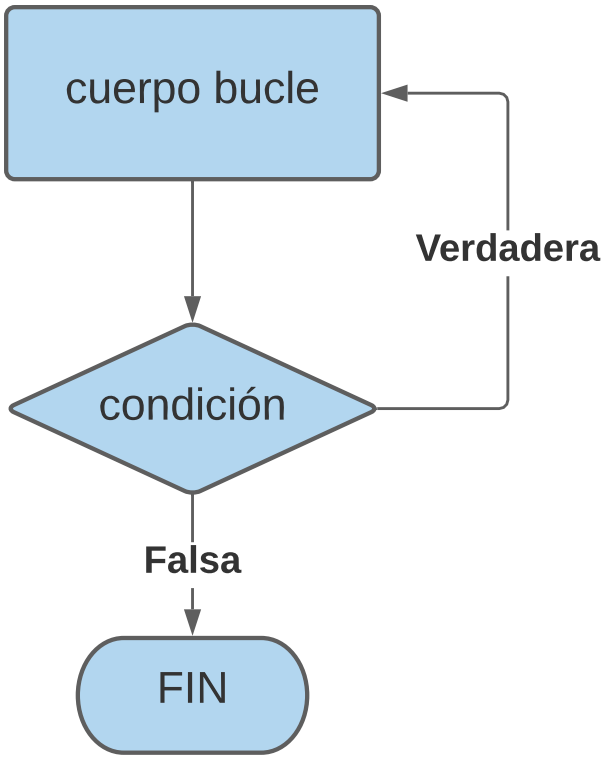


Test previo: bucle while

- Las sentencias se ejecutan **mientras** la condición del bucle sea verdadera.
- La condición del bucle se evalúa **antes** de que se ejecute el cuerpo del bucle.
- De este modo, el cuerpo del bucle se ejecutará 0 o más veces.
- No se conoce el número de iteraciones.
- En alguna iteración la condición debe ser falsa para evitar bucles infinitos.

Test posterior: bucle do-while

```
1 do{
2   <cuerpo bucle>
3 }while(<condicion>);
```



Test posterior: bucle do-while

- Las sentencias se ejecutan **mientras** la condición del bucle sea verdadera.
- La condición del bucle se evalúa **después** de que se ejecute el cuerpo del bucle.
- De este modo, el cuerpo del bucle se ejecutará al menos 1 vez.
- No se conoce el número de iteraciones.
- En alguna iteración la condición debe ser falsa para evitar bucles infinitos.

Ejemplos

Dado un número "maximo", mostrar los números menores que él.

```
int i = 0;
do{
  cout << i << endl;
  i++;
}while(i<maximo);
```

```
int i = 0;
while(i<maximo){
  cout << i << endl;
  i++;
}
```

¿Qué ocurre si maximo es igual a 0?

Ejemplos

Leer un entero y escribir los pares menores que él.

```
1 int i=0, limite;
2 cout << "Introduzca su limite: ";
3 cin >> limite;
4
5 while(i<limite)
6 {
7   cout << i << endl;
8   i=i+2;
9 }
```

Ejemplos

Ir sumando los valores leídos por teclado hasta que se introduzca el -1.

```
1  int suma = 0, n;
2  cout << "Introduzca un numero: ";
3  cin >> n;
4
5  while(n!=-1)
6  {
7    suma = suma + n;
8    cout << "Introduzca un numero: ";
9    cin >> n;
10 }
11 cout << "La suma es: " << suma << endl;
```

Ejemplos

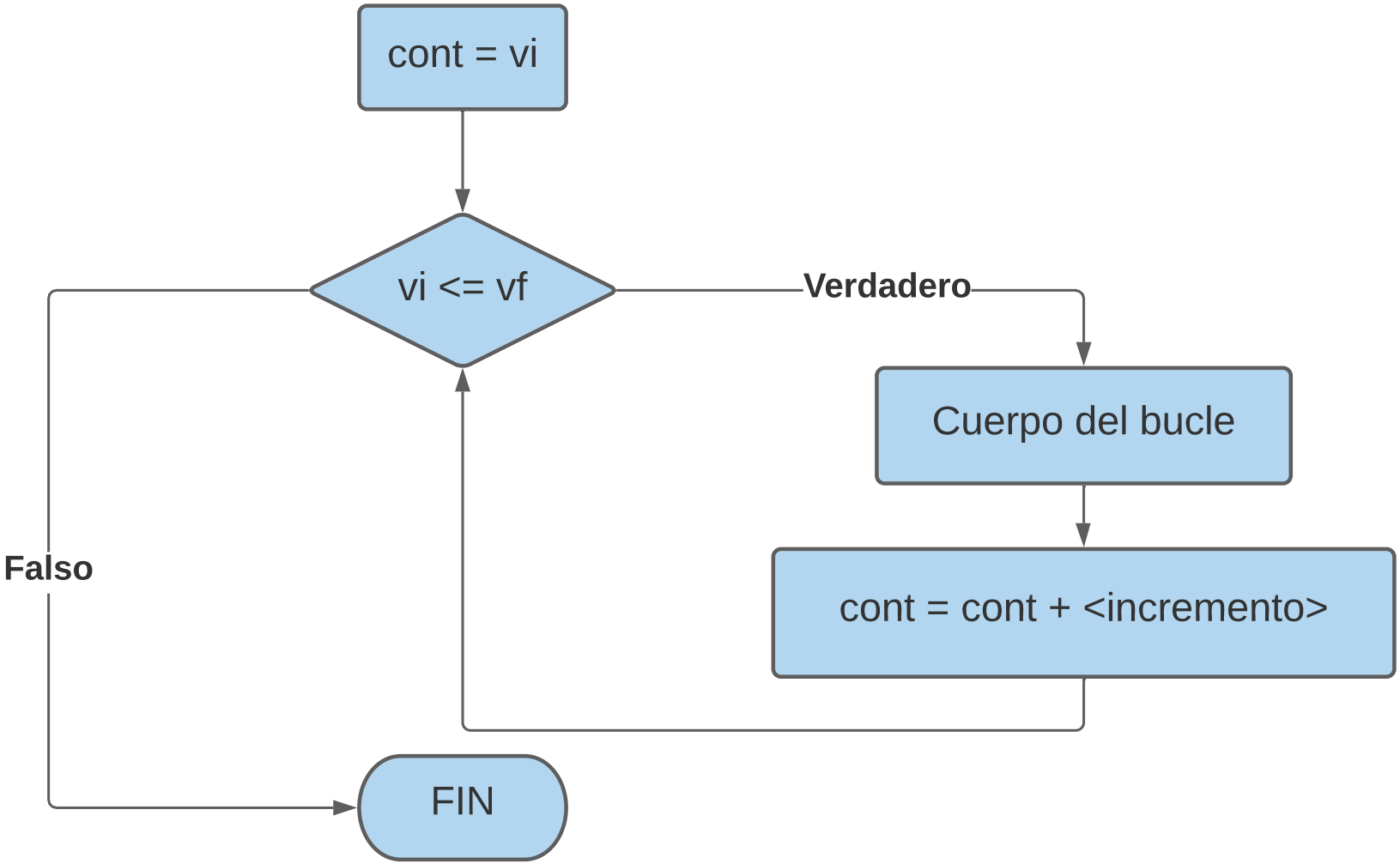
Introducir un valor y no permitir que se introduzca fuera de un rango.

```
int valor;
do{
  cout << "Introduzca valor: ";
  cin >> valor;
}while(valor < limite_inferior || valor > limite_superior);
```

Bucle for

- Se utilizan para repetir un conjunto de sentencias un número fijo de veces.
- Se necesita:
  - Una variable contador (*cont*);
  - Un valor inicial (*vi*);
  - Un valor final (*vf*);
  - Un incremento o decremento.

Bucle for


cont = vi
vi <= vf
Verdadero
Cuerpo del bucle
cont = cont + <incremento>
Falso
FIN

Bucle for

```
for(<asig inicial> ; <condición> ; <incremento>)
{
  <bloque for>
}
```

- **asig inicial**: asignación del valor inicial a la variable controladora del ciclo;
- **incremento**: determina el modo en que la variable contadora cambia su valor para la siguiente iteración.
- **condición**: expresión lógica que comprueba si la variable contador se encuentra dentro del rango indicado.

Bucle for

- Cuando termina un bucle **for**, la variable contadora se queda con el primer valor que hace que la condición sea falsa.
- Se conoce el número de iteraciones.
- **La variable contador no debe modificarse en el cuerpo del bucle.**
- Hay que pensar bien las condiciones para evitar bucles infinitos o bucles vacíos.

Comparativa entre while y for

Calcula la media de cinco valores introducidos por teclado.

```
int i=0, suma = 0, valor;
while (i<5){
  cout << "Introduzca valor: ";
  cin >> valor;
  suma = suma + valor;
  i++;
}
media = suma / 5.0;
```

```
int suma = 0, valor;
for(int i=0; i<5; i++)
  cout << "Introduzca valor: ";
  cin >> valor;
  suma = suma + valor;
}
media = suma / 5.0;
```

Bucle for: ejemplo

Escribir 9 veces el mensaje "Hola mundo".

Usando incrementos

```
1 for (int i=0; i < 9 ; i++){
2   cout << "Hola mundo" << endl;
3 }
4 //Otra posibilidad
5 for (int i=1; i <= 9 ; i++){
6   cout << "Hola mundo" << endl;
7 }
```

Usando decrementos

```
1 for (int i=9; i > 0 ; i--){
2   cout << "Hola mundo" << endl;
3 }
4 //Otra posibilidad
5 for (int i=9; i >= 1 ; i--){
6   cout << "Hola mundo" << endl;}
```

Bucle for: acumulador

- Estos bucles van acumulando en una variable el resultado de una expresión que se calcula en cada iteración.
- Ejemplo: Sumar los valores menores o iguales a un límite.

```
1 int suma = 0, limite;
2 cout << "Introduzca el limite: ";
3 cin >> limite;
4 for(int i=0; i<=limite; i++)
5 {
6   suma=suma+i;
7 }
8 cout << "La suma es: " << suma << endl;
```

Bucle for: contador

- Estos bucles cuentan el número de veces que ocurre una condición dentro de un bucle.
- Ejemplo: contar el número de pares en el intervalo [-20,20].

```
int cont=0;
for(int i=-20; i<=20; i++)
{
  if(i%2==0){
    cont++;
  }
}
cout << "El numero de elementos pares es: " << cont << endl;
```

Anidamiento de bucles

Dos bucles se encuentran anidados cuando uno de ellos está en el bloque de sentencias del otro. Ejemplo:

```
1 iteraciones=0;
2 for (i=0; i<2; i++){
3   for (j=0; j<2; j++) {
4     iteraciones++;
5   }
6 }
```

¿Cuál es el valor de iteraciones?

Errores frecuentes

- Bucles infinitos.
- Bucles con cero iteraciones.
- Bucles cuyo cuerpo no se ejecuta.
- Modificar la variable contadora del bucle.

Referencias

Recursos electrónicos:

- cppreference. (2020). Referencia C++: https://en.cppreference.com/w/
- Goalkicker.com (2018). C++: Note for Professionals.
- Grimes, R. (2017). Beginning C++ Programming.
- Joyanes, L. (2000). Programación en C++: Algoritmos, Estructuras de datos y Objetos.
- Juneja, B.L., Seth, A. (2009). Programming with C++.

Referencias

Libros:

- Prieto, A., Lloris, A., Torres, J.C. (2008). Introducción a la Informática (4 ed).
- Joyanes, L. (2008). Fundamentos de programación : algoritmos, estructura de datos y objetos.
- Martí, N., Ortega, Y., Verdejo, J.A. (2003). Estructuras de datos y métodos algorítmicos: Ejercicios resueltos.
- Tapia, S., García-Beltrán, A., Martínez, R., Jaen, J.A., del Álamo, J. (2012). Fundamentos de programación en C.

¿Preguntas?